

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: Таипов Т.А

Факультет: ИКТ

Группа: K3241

Преподаватель: Говорова М.М.



Санкт-Петербург 2023

Оглавление

Вариант 7. БД «Курсы»	3
Ход работы	4
Вывод	12

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, pgadmin 4.

Практическое задание:

1. Создать процедуры/функции согласно индивидуальному заданию (часть 4).

2.1. Модифицировать триггер (триггерную функцию) на проверку корректности входа и выхода сотрудника (см. Практическое задание 1 Лабораторного практикума (Приложение)) с максимальным учетом «узких» мест некорректных данных по входу и выходу).

2.2. Создать авторский триггер по варианту индивидуального задания.

Вариант 7. БД «Курсы»

Описание предметной области: Сеть учебных подразделений НОУ ДПО занимается организацией внебюджетного образования.

Имеется несколько образовательных программ краткосрочных курсов, предназначенных для определенных специальностей, связанных с программным обеспечением ИТ. Каждая программа имеет определенную длительность и свой перечень изучаемых дисциплин. Одна дисциплина может относиться к нескольким программам. На каждую программу может быть набрано несколько групп обучающихся.

По каждой дисциплине могут проводиться лекционные, лабораторные/практические занятия и практика определенном объеме часов. По каждой дисциплине и практике проводится аттестация в формате экзамен/дифзачет/зачет.

Необходимо хранить информацию по аттестации обучающихся.

Подразделение обеспечивает следующие ресурсы: учебные классы, лекционные аудитории и преподавателей. Необходимо составить расписание занятий.

БД должна содержать следующий минимальный набор сведений: Фамилия слушателя. Имя слушателя. Паспортные данные. Контакты. Код программы. Программа. Тип программы. Объем часов. Номер группы. максимальное количество человек в группе (для набора). Дата начала обучения. Дата окончания обучения. Название дисциплины. Количество часов. Дата занятий. Номер пары. Номер аудитории. Тип аудитории. Адрес площадки. Вид занятий (лекционные, практические или лабораторные). Фамилия преподавателя. Имя и отчество преподавателя. Должность преподавателя. Дисциплины, которые может вести преподаватель.

Задание 1.1 (ЛР 1 БД). Выполните инфологическое моделирование базы данных системы. (Ограничения задать самостоятельно.)

Задание 1.2. Создайте логическую модель БД, используя ИЛМ (задание 1.1). Используйте необходимые средства поддержки целостности данных в СУБД.

Задание 2. Создать запросы:

- Вывести все номера групп и программы, где количество слушателей меньше 10.
- Вывести список преподавателей с указанием количества программ, где они преподавали за истекший год.

- Вывести список преподавателей, которые не проводят занятия на третьей паре ни в один из дней недели.
- Вывести список свободных лекционных аудиторий на ближайший понедельник.
- Вычислить общее количество обучающихся по каждой программе за последний год.
- Вычислить среднюю загруженность компьютерных классов в неделю за последний месяц (в часах).
- Найти самые популярные программы за последние 3 года.

Задание 3. Создать представление:

- для потенциальных слушателей, содержащее перечень специальностей, изучаемых на них дисциплин и количество часов;
- общих доход по каждой программе за последний год.

Задание 4. Создать хранимые процедуры:

- Для получения расписания занятий для групп на определенный день недели.
- Записи на курс слушателя.
- Получения перечня свободных лекционных аудиторий на любой день недели. Если свободных аудиторий не имеется, то выдать соответствующее сообщение.

Задание 5. Создать необходимые триггеры.

Ход работы:

1. Процедура для получения расписания занятий для групп на определенный день недели:

```
CREATE OR REPLACE FUNCTION get_schedule_for_group_on_day(
    schedule_day VARCHAR(11)
)
RETURNS TABLE (
    schedule_id INT,
    group_id INT,
    group_name VARCHAR(20),
    date DATE,
    "time" TIME,
    audience_number INT,
    discipline_name VARCHAR(50),
    discipline_type VARCHAR(50),
    lecturer_name VARCHAR(50)
)
AS $$
SELECT
    s.id AS schedule_id,
    g.id AS group_id,
    g.name AS group_name,
    s.date,
    s."time",
    a."number" AS audience_number,
    d.name AS discipline_name,
    d.type AS discipline_type,
    l.full_name AS lecturer_name
FROM
    public.schedule s
JOIN
    public."group" g ON s.group_id = g.id
JOIN
    public.audience a ON s.audience_id = a.id
JOIN
    public.discipline_to_curriculum dc ON s.discipline_in_curriculum_id = dc.id
JOIN
    public.discipline d ON dc.discipline_id = d.id
JOIN
    public.lecturer l ON s.lecturer_id = l.id
WHERE
    s.date >= CURRENT_DATE
    AND EXTRACT(DOW FROM s.date) = CASE
        WHEN schedule_day = 'Понедельник' THEN 1
        WHEN schedule_day = 'Вторник' THEN 2
        WHEN schedule_day = 'Среда' THEN 3
        WHEN schedule_day = 'Четверг' THEN 4
        WHEN schedule_day = 'Пятница' THEN 5
        WHEN schedule_day = 'Суббота' THEN 6
        WHEN schedule_day = 'Воскресенье' THEN 0
    END;
$$ LANGUAGE SQL;
```

```

postgres=# CREATE OR REPLACE FUNCTION get_schedule_for_group_on_day(
postgres(#     schedule_day VARCHAR(11)
postgres(# )
postgres=# RETURNS TABLE (
postgres(#     schedule_id INT,
postgres(# group_id INT,
postgres(# group_name VARCHAR(20),
postgres(# date DATE,
postgres(# "time" TIME,
postgres(# audience_number INT,
postgres(# discipline_name VARCHAR(50),
postgres(# discipline_type VARCHAR(50),
postgres(# lecturer_name VARCHAR(50)
postgres(# )
postgres=# AS $$
postgres$$ SELECT
postgres$$     s.id AS schedule_id,
postgres$$ g.id AS group_id,
postgres$$ g.name AS group_name,
postgres$$     s.date,
postgres$$     s."time",
postgres$$     a."number" AS audience_number,
postgres$$     d.name AS discipline_name,
postgres$$ d.type AS discipline_type,
postgres$$ l.full_name AS lecturer_name
postgres$$ FROM
postgres$$     public.schedule s
postgres$$ JOIN
postgres$$     public."group" g ON s.group_id = g.id
postgres$$ JOIN
postgres$$     public.audience a ON s.audience_id = a.id
postgres$$ JOIN
postgres$$     public.discipline_to_curriculum dc ON s.discipline_in_curriculum_id = dc.id
postgres$$ JOIN
postgres$$     public.discipline d ON dc.discipline_id = d.id
postgres$$ JOIN
postgres$$     public.lecturer l ON s.lecturer_id = l.id
postgres$$ WHERE
postgres$$     s.date >= CURRENT_DATE
postgres$$     AND EXTRACT(DOW FROM s.date) = CASE
postgres$$         WHEN schedule_day = 'Понедельник' THEN 1
postgres$$         WHEN schedule_day = 'Вторник' THEN 2
postgres$$         WHEN schedule_day = 'Среда' THEN 3
postgres$$         WHEN schedule_day = 'Четверг' THEN 4
postgres$$         WHEN schedule_day = 'Пятница' THEN 5
postgres$$         WHEN schedule_day = 'Суббота' THEN 6
postgres$$         WHEN schedule_day = 'Воскресенье' THEN 0
postgres$$     END;
postgres$$ $$ LANGUAGE SQL;
CREATE FUNCTION

```

1 **SELECT** * **FROM** get_schedule_for_group_on_day('Вторник')

Data Output Messages Notifications

	schedule_id integer	group_id integer	group_name character varying	date date	time time without time zone	audience_number integer	discipline_name character varying	discipline_type character varying	lecturer_name character varying
1	5	2	group2	2024-03-12	14:00:00	5	discipline2	practice	lecturer5

2. Процедура для записи слушателя на курс:

```
CREATE OR REPLACE PROCEDURE enroll_student(id_student INT, id_program INT,
std_type studenttogrouptype)
```

```
LANGUAGE plpgsql
```

```
AS $$
```

```
BEGIN
```

```
    IF NOT EXISTS (SELECT 1 FROM student WHERE id = id_student) THEN
```

```
        RAISE EXCEPTION 'Студент с ID % не существует.', student_id;
```

```
    END IF;
```

```
    IF NOT EXISTS (SELECT 1 FROM program WHERE id = id_program) THEN
```

```
        RAISE EXCEPTION 'Курс с ID % не существует.', curriculum_id;
```

```
    END IF;
```

```
    IF EXISTS (
```

```
        SELECT 1
```

```
        FROM student_to_group stg
```

```
            JOIN "group" g ON stg.group_id = g.id
```

```
            JOIN curriculum c ON c.id = g.curriculum_id
```

```
        WHERE student_id = id_student
```

```
        AND c.program_id = id_program
```

```
    ) THEN
```

```
        RAISE EXCEPTION 'Студент уже записан на данный курс.';
```

```
    END IF;
```

```
    INSERT INTO student_to_group (id, student_id, group_id, type, start_date, status)
```

```
    VALUES (id_student, id_student, (SELECT g.id FROM curriculum c JOIN "group" g ON c.id
= g.curriculum_id JOIN program p ON p.id = c.program_id WHERE p.id = id_program ORDER
BY g.current_members DESC LIMIT 1), std_type, CURRENT_DATE, 'studying');
```

```
END;
```

```
$$;
```

```

postgres=# CREATE OR REPLACE PROCEDURE enroll_student(id_student INT, id_program INT, std_type studenttogrouptype)
postgres=# LANGUAGE plpgsql
postgres=# AS $$
postgres=# BEGIN
postgres=# IF NOT EXISTS (SELECT 1 FROM student WHERE id = id_student) THEN
postgres=# RAISE EXCEPTION 'Студент с ID % не существует.', student_id;
postgres=# END IF;
postgres=# IF NOT EXISTS (SELECT 1 FROM program WHERE id = id_program) THEN
postgres=# RAISE EXCEPTION 'Курс с ID % не существует.', curriculum_id;
postgres=# END IF;
postgres=# IF EXISTS (
postgres=# SELECT 1
postgres=# FROM student_to_group stg
postgres=# JOIN "group" g ON stg.group_id = g.id
postgres=# JOIN curriculum c ON c.id = g.curriculum_id
postgres=# WHERE student_id = id_student
postgres=# AND c.program_id = id_program
postgres=# ) THEN
postgres=# RAISE EXCEPTION 'Студент уже записан на данный курс.';
postgres=# END IF;
postgres=# INSERT INTO student_to_group (id, student_id, group_id, type, start_date, status)
postgres=# VALUES (id_student, id_student, (SELECT g.id FROM curriculum c JOIN "group" g ON c.id = g.curriculum_id JOIN program p ON p.id = c.program_id WHERE p.id = id_program ORDER BY g.current_members DESC LIMIT 1), std_type, CURRENT_DATE, 'studying');
postgres=# END;
postgres=# $$;
CREATE PROCEDURE

```

1 CALL enroll_student(10, 2, 'contract')

Data Output Messages Notifications

CALL

Query returned successfully in 91 msec.

1 SELECT * FROM student_to_group

Data Output Messages Notifications

	id [PK] integer	student_id integer	group_id integer	type studenttogrouptype	start_date date	end_date date	status studenttogroupstatus
1	1	1	1	tuition_free	2020-01-01	[null]	studying
2	2	2	1	tuition_free	2020-01-01	[null]	studying
3	3	3	1	contract	2020-01-01	[null]	studying
4	4	4	1	tuition_free	2020-01-01	[null]	studying
5	5	5	2	contract	2020-01-01	[null]	studying
6	6	6	2	contract	2020-01-01	[null]	studying
7	7	7	2	contract	2020-01-01	[null]	studying
8	8	8	2	tuition_free	2020-01-01	[null]	studying
9	9	9	4	contract	2023-01-01	[null]	studying
10	10	10	3	contract	2024-03-11	[null]	studying

- Процедура для получения перечня свободных лекционных аудиторий на любой день недели:

```

CREATE OR REPLACE FUNCTION get_free_audiences_for_day(aud_day VARCHAR(11))
RETURNS TABLE(address VARCHAR(50), aud_number INT, aud_type audienctype)
LANGUAGE plpgsql
AS $$
BEGIN

```

```

    IF NOT EXISTS (
        SELECT 1
        FROM audience a
        JOIN place p ON a.place_id = p.id
        WHERE a.id NOT IN (
            SELECT DISTINCT s.audience_id

```



```

FROM public.schedule s
WHERE s.date < CURRENT_DATE
      OR EXTRACT(DOW FROM s.date) = CASE
          WHEN aud_day = 'Понедельник' THEN 1
          WHEN aud_day = 'Вторник' THEN 2
          WHEN aud_day = 'Среда' THEN 3
          WHEN aud_day = 'Четверг' THEN 4
          WHEN aud_day = 'Пятница' THEN 5
          WHEN aud_day = 'Суббота' THEN 6
          WHEN aud_day = 'Воскресенье' THEN 0
      END
    )
) THEN
    RAISE EXCEPTION 'На %, все аудитории заняты.', aud_day;
ELSE
    RETURN QUERY
    SELECT p.address, a.number AS aud_number, a.type AS aud_type
    FROM audience a
    JOIN place p ON a.place_id = p.id
    WHERE a.id NOT IN (
        SELECT DISTINCT s.audience_id
        FROM public.schedule s
        WHERE s.date < CURRENT_DATE
              OR EXTRACT(DOW FROM s.date) = CASE
                  WHEN aud_day = 'Понедельник' THEN 1
                  WHEN aud_day = 'Вторник' THEN 2
                  WHEN aud_day = 'Среда' THEN 3
                  WHEN aud_day = 'Четверг' THEN 4
                  WHEN aud_day = 'Пятница' THEN 5
                  WHEN aud_day = 'Суббота' THEN 6
                  WHEN aud_day = 'Воскресенье' THEN 0
              END
    );
END IF;
END;
$$;

```

```

postgres=# CREATE OR REPLACE FUNCTION get_free_audiences_for_day(aud_day VARCHAR(11))
postgres=# RETURNS TABLE(address VARCHAR(50), aud_number INT, aud_type audiencetype)
postgres=# LANGUAGE plpgsql
postgres=# AS $$
postgres$# BEGIN
postgres$#     IF NOT EXISTS (
postgres$#         SELECT 1
postgres$#         FROM audience a
postgres$#         JOIN place p ON a.place_id = p.id
postgres$#         WHERE a.id NOT IN (
postgres$#             SELECT DISTINCT s.audience_id
postgres$#             FROM public.schedule s
postgres$#             WHERE s.date < CURRENT_DATE
postgres$#         OR EXTRACT(DOW FROM s.date) = CASE
postgres$#             WHEN aud_day = 'Понедельник' THEN 1
postgres$#             WHEN aud_day = 'Вторник' THEN 2
postgres$#             WHEN aud_day = 'Среда' THEN 3
postgres$#             WHEN aud_day = 'Четверг' THEN 4
postgres$#             WHEN aud_day = 'Пятница' THEN 5
postgres$#             WHEN aud_day = 'Суббота' THEN 6
postgres$#             WHEN aud_day = 'Воскресенье' THEN 0
postgres$#         END
postgres$#     )
postgres$# ) THEN
postgres$#     RAISE EXCEPTION 'На (%), все аудитории заняты.', aud_day;
postgres$# ELSE
postgres$#     RETURN QUERY
postgres$#     SELECT p.address, a.number AS aud_number, a.type AS aud_type
postgres$#     FROM audience a
postgres$#     JOIN place p ON a.place_id = p.id
postgres$#     WHERE a.id NOT IN (
postgres$#         SELECT DISTINCT s.audience_id
postgres$#         FROM public.schedule s
postgres$#         WHERE s.date < CURRENT_DATE
postgres$#     OR EXTRACT(DOW FROM s.date) = CASE
postgres$#         WHEN aud_day = 'Понедельник' THEN 1
postgres$#         WHEN aud_day = 'Вторник' THEN 2
postgres$#         WHEN aud_day = 'Среда' THEN 3
postgres$#         WHEN aud_day = 'Четверг' THEN 4
postgres$#         WHEN aud_day = 'Пятница' THEN 5
postgres$#         WHEN aud_day = 'Суббота' THEN 6
postgres$#         WHEN aud_day = 'Воскресенье' THEN 0
postgres$#     END
postgres$#     );
postgres$#     END IF;
postgres$# END;
postgres$# $$;
CREATE FUNCTION
1 SELECT * FROM get_free_audiences_for_day('Понедельник')

```

Data Output [Messages](#) Notifications

ERROR: На Понедельник, все аудитории заняты.

```
1 SELECT * FROM get_free_audiences_for_day('Вторник')
```

Data Output Messages Notifications

	address character varying	aud_number integer	aud_type audience type
1	Lenina 1	1	computer_lab
2	Lenina 2	2	computer_lab
3	Lenina 3	3	computer_lab
4	Lenina 4	4	computer_lab
5	Lenina 5	5	lecture

4. Триггер для избегания пересечений в записи студента.

```
CREATE OR REPLACE FUNCTION check_enrollment_overlap()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM student_to_group sg
        WHERE NEW.student_id = sg.student_id
              AND sg.status = 'studying'
    ) THEN
        RAISE EXCEPTION 'Студент уже состоит группе.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER enrollment_overlap_check
BEFORE INSERT ON student_to_group
FOR EACH ROW
EXECUTE FUNCTION check_enrollment_overlap();
```

```
postgres=# CREATE OR REPLACE FUNCTION check_enrollment_overlap()
postgres=# RETURNS TRIGGER AS $$
postgres$# BEGIN
postgres$#     IF EXISTS (
postgres$#         SELECT 1
postgres$#         FROM student_to_group sg
postgres$#         WHERE NEW.student_id = sg.student_id
postgres$# AND end_date <= CURRENT_DATE
postgres$#     ) THEN
postgres$#         RAISE EXCEPTION 'Студент уже состоит группе.';
postgres$#     END IF;
postgres$#     RETURN NEW;
postgres$# END;
postgres$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=#
postgres=# CREATE TRIGGER enrollment_overlap_check
postgres=# BEFORE INSERT ON student_to_group
postgres=# FOR EACH ROW
postgres=# EXECUTE FUNCTION check_enrollment_overlap();
CREATE TRIGGER
```

```
1 INSERT INTO student_to_group(id, student_id, group_id, type, start_date, status)
2 VALUES(11, 3, 1, 'contract', CURRENT_DATE, 'studying')
```

Data Output [Messages](#) Notifications

ERROR: Студент уже состоит группе.

1 **SELECT** * **FROM** student_to_group

Data Output Messages Notifications

	id [PK] integer	student_id integer	group_id integer	type studenttogrouptype	start_date date	end_date date	status studenttogroupstatus
1	1	1	1	tuition_free	2020-01-01	[null]	studying
2	2	2	1	tuition_free	2020-01-01	[null]	studying
3	3	3	1	contract	2020-01-01	[null]	studying
4	4	4	1	tuition_free	2020-01-01	[null]	studying
5	5	5	2	contract	2020-01-01	[null]	studying
6	6	6	2	contract	2020-01-01	[null]	studying
7	7	7	2	contract	2020-01-01	[null]	studying
8	8	8	2	tuition_free	2020-01-01	[null]	studying
9	9	9	4	contract	2023-01-01	[null]	studying

5. Триггер на удаление студента при уходе с программы.

```
CREATE OR REPLACE FUNCTION delete_student()
RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM public.student s
    WHERE s.id NOT IN (SELECT DISTINCT stg.student_id FROM public.student_to_group stg);

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_students_list
AFTER DELETE ON public.student_to_group
FOR EACH ROW
EXECUTE FUNCTION delete_student();
```

```
postgres=# CREATE OR REPLACE FUNCTION delete_student()
postgres=# RETURNS TRIGGER AS $$
postgres=# BEGIN
postgres=#     DELETE FROM public.student s
postgres=#     WHERE s.id NOT IN (SELECT DISTINCT stg.student_id FROM public.student_to_group stg);
postgres=#
postgres=#     RETURN NULL;
postgres=# END;
postgres=# $$ LANGUAGE plpgsql;
postgres=# CREATE FUNCTION
postgres=#
postgres=# CREATE TRIGGER update_students_list
postgres=# AFTER DELETE ON public.student_to_group
postgres=# FOR EACH ROW
postgres=# EXECUTE FUNCTION delete_student();
postgres=# CREATE TRIGGER
```

```
1 delete from student_to_group where student_id = 5
```

Data Output Messages Notifications

DELETE 1

Query returned successfully in 74 msec.

Query Query History

```
1 select * from student
```

Data Output Messages Notifications

	id [PK] integer	full_name character varying (50)	date_of_birth date	passport_number character varying (10)	phone_number character varying (15)
1	1	Ivanov Ivan Ivanovich	1999-01-01	1234567890	+77529123456
2	2	Petrov Petr Petrovich	1999-01-01	2345678901	+77529123457
3	3	Sidorov Sidor Sidorovich	1999-01-01	3456789012	+77529123457
4	4	Pupkin Vasya Vasilievich	1999-01-01	4567890123	+77529123567
5	6	Petrova Maria Petrovna	1999-01-01	6789012345	+77529123567
6	7	Sidorova Olga Sidorovna	1999-01-01	7890123456	+77529123467
7	8	Pupkina Vasilisa Vasilievna	1999-01-01	8901234567	+77529123467
8	9	Pupkin Vasilis Vasiliev	1999-01-01	8901234576	+77529123476

```
1 select * from student_to_group
```

Data Output Messages Notifications

	id [PK] integer	student_id integer	group_id integer	type studenttogrouptype	start_date date	end_date date	status studenttogroupstatus
1	1	1	1	tuition_free	2020-01-01	[null]	studying
2	2	2	1	tuition_free	2020-01-01	[null]	studying
3	3	3	1	contract	2020-01-01	[null]	studying
4	4	4	1	tuition_free	2020-01-01	[null]	studying
5	6	6	2	contract	2020-01-01	[null]	studying
6	7	7	2	contract	2020-01-01	[null]	studying
7	8	8	2	tuition_free	2020-01-01	[null]	studying
8	9	9	4	contract	2023-01-01	[null]	studying

Вывод

В ходе выполнения данной лабораторной работы научились создавать и работать с процедурами, функциями и триггерами в PostgreSQL.