

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Лабораторная работа № 3
**«ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В
POSTGRESQL»**

Выполнил: Евдокимов Владислав Борисович

Группа: К3242

Преподаватель: Говорова Марина Михайловна

Санкт-Петербург
2022

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

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

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.

Вариант 2

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.

Указание. Работа выполняется в консоли SQL Shell (psql)

ТЕХНОЛОГИЯ ВЫПОЛНЕНИЯ РАБОТЫ:

1. Название БД

Вариант 7. «Курсы»

Описание предметной области: Подразделение занимается организацией внебюджетного образования. Имеется несколько типов краткосрочных курсов, предназначенных для определенных специальностей, связанных с программным обеспечением ИТ. Каждый тип курсов имеет определенную длительность и свой перечень изучаемых дисциплин. На каждую программу может быть набрано несколько групп обучающихся. По каждой дисциплине могут проводиться лекционные и лабораторные занятия. 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.

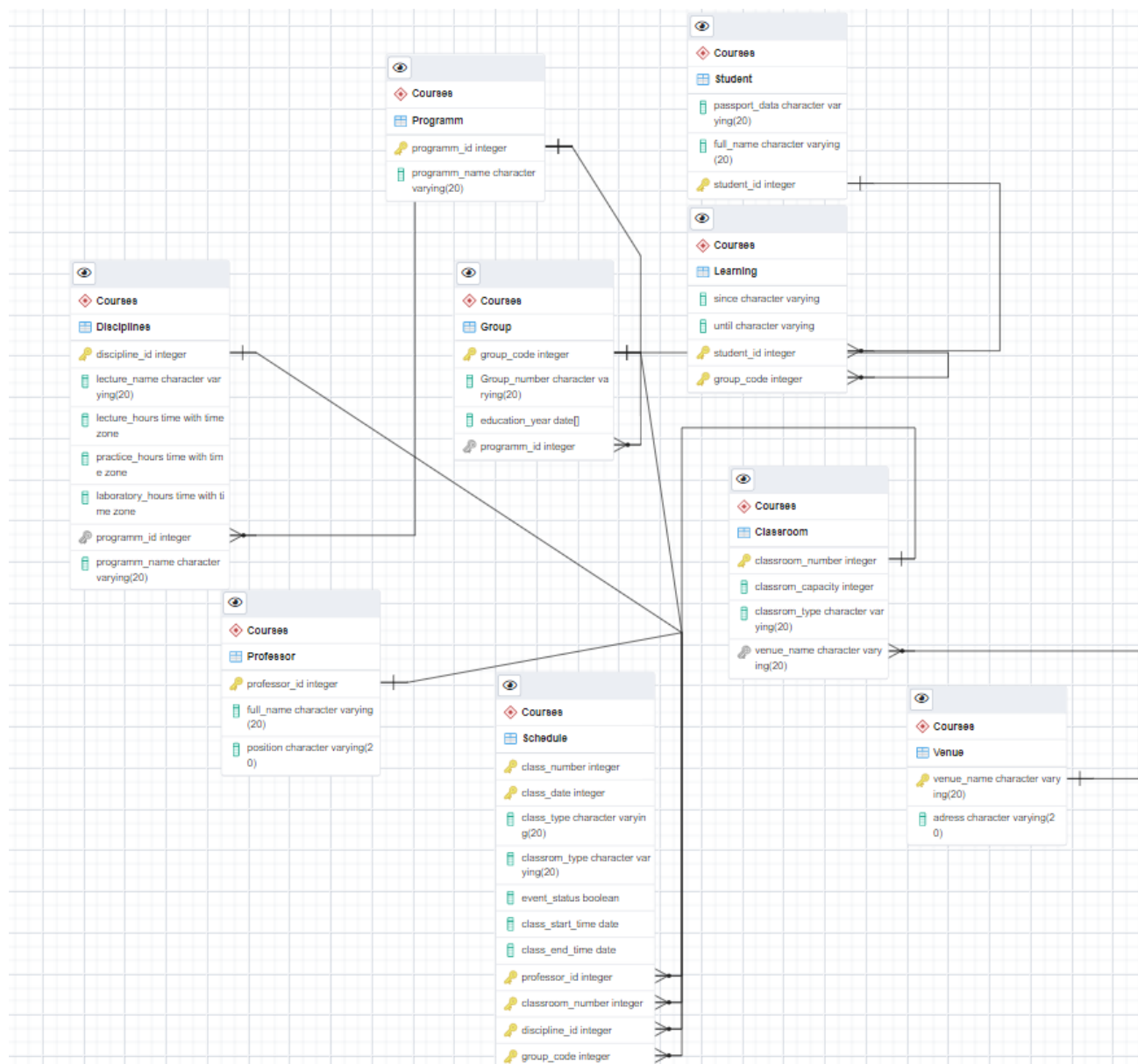
Состав реквизитов сущностей:

- a) **Направление** (Код программы, наименование)
- b) **Дисциплины** (ID дисциплины, код направления, название дисциплины, лекционные часы, лабораторные часы, практические часы)
- c) **Группа** (Код группы, номер группы, год обучения, код направления)
- d) **Слушатель** (ID слушателя, контакты, имя, фамилия, код группы, паспортные данные)
- e) **Расписание** (Код расписания, ID преподавателя, ID дисциплины, код группы, тип занятий, номер пары, кол-во часов, номер аудитории, статус проведения, даты занятий, тип аудитории, номер класса, врем конца занятий, время начала занятий)
- f) **Площадка проведения** (Название, адрес)

g) Аудитория (Номер аудитории, тип аудитории, вместимость, название площадки)

h) Преподаватель (ID преподавателя, ФИО, должность)

2. Схема логической модели базы данных:



Создание хранимых процедур

- 1) Для получения расписания занятий для групп на определенный день недели.

```
CREATE OR REPLACE FUNCTION schedule_for_today
(schedule_date date) RETURNS TABLE (class_number integer,
classroom_type character varying, event_status text, professor_id
integer, classroom_number integer, discipline_id integer, group_code
integer, class_start_time character varying, class_end_time character
varying, schedule_id integer, real_class_date date) AS $$
SELECT class_number, classroom_type, event_status, professor_id,
classroom_number, discipline_id, group_code, class_start_time,
class_end_time, schedule_id, real_class_date
FROM courses.schedule WHERE real_class_date = schedule_date ;
$$
LANGUAGE SQL;
```

```
courses=# CREATE OR REPLACE FUNCTION schedule_for_today (schedule_date date) RETURNS TABLE (class_number integer, classroom_type character varying, event_status text, professor_id integer, classroom_number integer, discipline_id integer, group_code integer, class_start_time character varying, class_end_time character varying, schedule_id integer, real_class_date date) AS $$
courses=# SELECT class_number, classroom_type, event_status, professor_id, classroom_number, discipline_id, group_code, class_start_time, class_end_time, schedule_id, real_class_date
courses=# FROM courses.schedule where real_class_date= schedule_date ;
courses=# $$
courses=# LANGUAGE SQL;
CREATE FUNCTION
```

Вызов функции: `select * from schedule_for_today('2022-05-16');`

```
courses=# select * from schedule_for_today('2022-05-16');
 class_number | classroom_type | event_status | professor_id | classroom_number | discipline_id | group_code | class_start_time | class_end_time | schedule_id | real_class_date
-----
          3 | Laboratory    | Pass        + |          99 |          214 |      345678 |      3242 | 10:00           | 11:30         |          18 | 2022-05-16
          4 | Lecture       | Pass        + |         101 |          355 |          18 |      3141 | 13:30           | 15:00         |          66 | 2022-05-16
(2 строки)

courses=#
```

2) Запись на курс слушателя:

```
CREATE OR REPLACE FUNCTION welcome_to_family (pd
character varying, fn character varying, s_id integer, gc integer, ss
character varying, l_id integer)
RETURNS VOID AS $$
INSERT INTO courses.student (passport_data, full_name, student_id)
VALUES (pd, fn, s_id);
INSERT INTO courses.learning (student_id, group_code,
student_status, learning_id) VALUES (s_id, gc, ss, l_id);
UPDATE courses.group set student_amount = student_amount + 1
WHERE group_code = gc;
$$
LANGUAGE SQL;
```




```
courses=# CREATE OR REPLACE FUNCTION welcome_to_family (pd character varying, fn character varying, s_id integer, gc integer, ss character varying, l_id integer)
courses-# RETURNS VOID AS $$
courses$# INSERT INTO courses.student (passport_data, full_name, student_id) VALUES (pd, fn, s_id);
courses$# INSERT INTO courses.learning (student_id, group_code, student_status, learning_id) VALUES (s_id, gc, ss, l_id)
;
courses$# $$
courses-# LANGUAGE SQL;
CREATE FUNCTION
courses=#
```

Вызов функции: SELECT * FROM welcome_to_family ('556624533',
'Fag Gag', '575757', '3242', 'Student', '4352');

```
courses=# SELECT * FROM welcome_to_family ('556624533', 'Fag Gag', '575757', '3242', 'Student', '4352');
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "student_pkey"
ПОДРОБНОСТИ: Ключ "(student_id)=(575757)" уже существует.
КОНТЕКСТ: SQL-функция "welcome_to_family", оператор 1
```

До:







Messages	Notifications	Query History	Data Output	Explain
	student_id integer	group_code integer	student_status character varying (15)	learning_id [PK] integer
19	1	3141	Expelled	111
20	3	3141	Expelled	333
21	4	3141	Expelled	444
22	5	3141	Expelled	555
23	2	3242	Expelled	222
24	7	3141	Student	666
25	8	3141	Student	777
26	9	3141	Student	888
27	10	3141	Student	999
28	6	3242	Expelled	1010

Messages	Notifications	Query History	Data Output	Explain			
 group_code [PK] integer	 group_number character varying (15)	 programm_id integer	 programm_name character varying (50)	 student_amount integer	 education_year_since date	 education_year_until date	
1	3242	2	450345663	Intelligent Systems in Humanities	19	2021-09-01	2022-06-01
2	3141	3	4504068	Mobile development	8	2021-09-01	2022-06-01
3	3140	4	450345663	Intelligent Systems in Humanities	0	2021-09-01	2022-06-01

Messages	Notifications	Query History	Data Output	Explain
	passport_data character varying (40)	full_name character varying (40)	student_id [PK] integer	
45	134	werg		45
46	314	werg		46
47	341	werg		47
48	341	werg		48
49	314	werg		49
50	314	werg		50
51	35 11 639720	Perov Pavel		101
52	56 36 43567	Roy Pydreel		111
53	46 35 43657	Cherov Akhmad Tea		0
54	05 05 320382	Ivanov Ivan Ivanovich		505

После:

Messages	Notifications	Query History	Data Output	Explain
	passport_data character varying (40)	full_name character varying (40)	student_id [PK] integer	
50	314	werg	50	
51	35 11 639720	Perov Pavel	101	
52	56 36 43567	Roy Pydreel	111	
53	46 35 43657	Cherov Akhmad Tea	0	
54	05 05 320382	Ivanov Ivan Ivanovich	505	
55	556624533	Fag Gag	575757	

Messages	Notifications	Query History	Data Output	Explain				
 group_code [PK] integer 	group_number character varying (15) 	programm_id integer 	programm_name character varying (50) 	student_amount integer 	education_year_since date	education_year_until date		
1	3141	3	4504068	Mobile development	8	2021-09-01	2022-06-01	
2	3140	4	450345663	Intelligent Systems in Humanities	0	2021-09-01	2022-06-01	
3	3242	2	450345663	Intelligent Systems in Humanities	20	2021-09-01	2022-06-01	

Messages	Notifications	Query History	Data Output	Explain
	student_id integer	group_code integer	student_status character varying (15)	learning_id [PK] integer
24	7	3141	Student	666
25	8	3141	Student	777
26	9	3141	Student	888
27	10	3141	Student	999
28	6	3242	Expelled	1010
29	575757	3242	Student	4352

3) Получение перечня свободных лекционных аудиторий на любой день недели. Если свободных лекционных аудиторий нет, вывести соответствующее сообщение.

```
CREATE OR REPLACE FUNCTION free_lecture_rooms(toime date)
RETURNS TABLE(classroom_number integer) AS $$
DECLARE
    pls INTEGER;
BEGIN
    SELECT COUNT(DISTINCT schedule.classroom_number) INTO pls
    FROM schedule WHERE classroom_type = 'Lecture'
    and schedule.classroom_number NOT IN
    (SELECT schedule.classroom_number FROM schedule WHERE
    schedule.real_class_date = toime and schedule.classroom_type = 'Lecture');
    IF pls > 0 THEN
        RETURN QUERY(SELECT DISTINCT schedule.classroom_number
        FROM schedule WHERE schedule.classroom_number NOT IN
        (SELECT schedule.classroom_number FROM schedule WHERE
        real_class_date = toime and schedule.classroom_type = 'Lecture')
        and schedule.classroom_type = 'Lecture');
    ELSE RAISE NOTICE 'Свободных лекционных аудиторий нет!';
    END IF;
END;
$$
LANGUAGE plpgsql;
```

Если свободные аудитории есть:

```
courses=# SELECT * FROM free_lecture_rooms('2022-05-06');
 classroom_number
-----
            103
            206
            355
           2337
           3212
(5 строк)
```

Если их нет:

```
courses=# SELECT * FROM free_lecture_rooms('2022-05-06');
ЗАМЕЧАНИЕ: Свободных лекционных аудиторий нет!
 classroom_number
-----
(0 строк)
```

Задание 2: Создание триггеров

Создадим универсальный триггер для логирования действий добавления, обновления и удаления данных. Предварительно создадим таблицу «logs» для этого.

```
courses=# CREATE OR REPLACE FUNCTION iud() RETURNS TRIGGER AS $$
courses$# DECLARE
courses$# mstr varchar(30);
courses$# astr varchar(100);
courses$# retstr varchar(254);
courses$# BEGIN
courses$# IF TG_OP = 'INSERT' THEN
courses$# astr = NEW;
courses$# mstr := 'INSERTION';
courses$# retstr := mstr||astr;
courses$# INSERT INTO logs(state_of, table_name_eeee, changes_done_to, time_of_change) values (mstr, TG_TABLE_NAME, retstr, NOW());
courses$# RETURN NEW;
courses$# ELSIF TG_OP = 'UPDATE' THEN
courses$# astr = NEW;
courses$# mstr := 'Update user ';
courses$# retstr := mstr||astr;
courses$# INSERT INTO logs(state_of, table_name_eeee, changes_done_to, time_of_change) values (mstr, TG_TABLE_NAME, retstr, NOW());
courses$# RETURN NEW;
courses$# ELSIF TG_OP = 'DELETE' THEN
courses$# astr = OLD;
courses$# mstr := 'REMOVAL';
courses$# retstr := mstr || astr;
courses$# INSERT INTO logs(state_of, table_name_eeee, changes_done_to, time_of_change) values (mstr, TG_TABLE_NAME, retstr, NOW());
courses$# RETURN OLD;
courses$# END IF;
courses$# END;
courses$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Создадим триггер и сделаем запросы:

```
courses=# CREATE TRIGGER tt AFTER INSERT OR UPDATE OR DELETE ON courses.professor FOR EACH ROW EXECUTE PROCEDURE iud();
CREATE TRIGGER
courses=# UPDATE professor set "position" = 'senior janitor' where professor_id = '34553';
UPDATE 1
courses=# INSERT INTO professor(professor_id, full_name, "position") values ('245345', 'ert wrtgt wrtbg', 'crapp');
INSERT 0 1
courses=# DELETE FROM professor WHERE professor_id ='4254';
DELETE 1
```

Результаты:

```
courses=# select * from logs;
state_of | table_name_eeee | changes_done_to | time_of_change
-----+-----+-----+-----
Update user | professor | Update user (34553," Kashpey Andrew Smith","senior janitor") | 2022-05-15 01:28:55.628
276+03
INSERTION | professor | INSERTION(245345,"ert wrtgt wrtbg",crap) | 2022-05-15 01:31:09.109
239+03
REMOVAL | professor | REMOVAL(4254,"Vervg EVWR ver",crap) | 2022-05-15 01:32:15.135
399+03
(3 строки)
```

Вывод:

В ходе выполнения лабораторной работы я узнал принцип создания хранимых процедур и триггеров.