

**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»

Обучающийся: Бородин Максим Андреевич
Факультет прикладной информатики
Группа К3241
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

Цель работы:

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

Программное обеспечение:

СУБД PostgreSQL 1X, SQL Shell (psql)

Практическое задание

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры.
2. Создать триггеры для индивидуальной БД согласно варианту:
7 оригинальных триггеров - 7 баллов (max). Изучить графическое представление запросов и просмотреть историю запросов.

Схема базы данных

Модель описывает систему «Расписание занятий и распределение аудиторного фонда» образовательной организации с учётом аудиторий, учебных планов, образовательных программ, направлений подготовки, дисциплин, преподавателей, студенческих групп и назначенных занятий. Ознакомиться с моделью можно на рисунке 1 — для отображения использован генератор ERD-схемы в pgAdmin с удобным оформлением связей между таблицами.

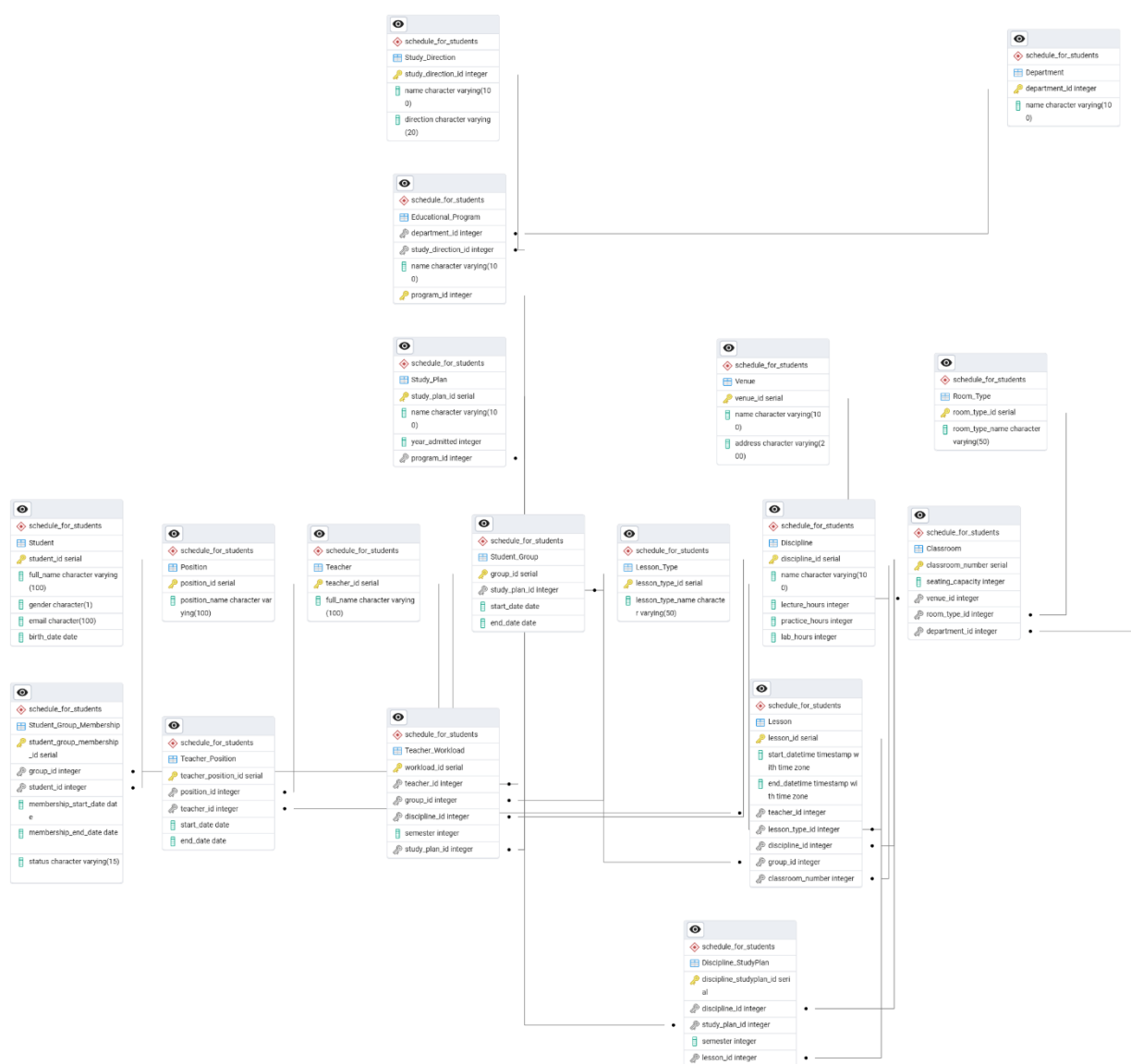


Рисунок 1 – Схема базы данных

Ход работы

Задание 1. Создать 3 процедуры для индивидуальной БД согласно варианту.

Было решено добавить 2 новых таблицы в БД для записи логов:

```
CREATE TABLE order_log (  
    log_id SERIAL PRIMARY KEY,  
    order_id INT,  
    action VARCHAR(50),  
    log_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    details TEXT  
);
```

```
CREATE TABLE complaint_status_log (  
    log_id SERIAL PRIMARY KEY,  
    complaint_id INT,  
    old_status VARCHAR(20),  
    new_status VARCHAR(20),  
    log_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

taxi_service=# \dt

Список отношений

Схема	Имя	Тип	Владелец
public	brands	таблица	postgres
public	car_tariffs	таблица	postgres
public	cars	таблица	postgres
public	complaint_status_log	таблица	postgres
public	complaints	таблица	postgres
public	current_positions	таблица	postgres
public	employees	таблица	postgres
public	models	таблица	postgres
public	order_log	таблица	postgres
public	orders	таблица	postgres
public	passengers	таблица	postgres
public	payments	таблица	postgres
public	positions	таблица	postgres
public	tariffs	таблица	postgres
public	work_schedules	таблица	postgres

(15 строк)

Были разработаны три хранимые процедуры на языке PL/pgSQL, соответствующие заданию. Процедуры используют параметры IN и OUT, включают обработку ошибок и возвращают результаты через курсоры (где применимо).

Процедура 1: Вывести список свободных аудиторий для проведения практических занятий заданной группы в заданное время.

```
CREATE OR REPLACE PROCEDURE schedule_for_students.get_free_classrooms_nocursor(  
    IN p_group_id INT,  
    IN p_start    TIMESTAMPTZ,  
    IN p_end      TIMESTAMPTZ  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN
```

```
-- Удаляем предыдущую временную таблицу (если была)
```

```
DROP TABLE IF EXISTS tmp_free_classrooms;
```

```
-- Создаем новую временную таблицу и сразу заполняем её нужными аудиториями
```

```
CREATE TEMP TABLE tmp_free_classrooms AS
```

```
SELECT c.*
```

```
FROM schedule_for_students."Classroom" c
```

```
WHERE NOT EXISTS (
```

```
    SELECT 1
```

```
    FROM schedule_for_students."Lesson" l
```

```
    WHERE l.classroom_number = c.classroom_number
```

```
    AND tstzrange(l.start_datetime, l.end_datetime)
```

```
    && tstzrange(p_start, p_end)
```

```
)
```

```
AND c.seating_capacity >= (
```

```
    SELECT COUNT(*)
```

```
    FROM schedule_for_students."Student_Group_Membership" m
```

```
    WHERE m.group_id = p_group_id
```

```
    AND (m.membership_end_date IS NULL OR m.membership_end_date >=
```

```
p_start)
```

```
);
```

```
END;
```

\$\$; Возвращает список аудиторий, свободных в указанном интервале времени и способных вместить заданную группу: проверяет отсутствие пересечений по расписанию и достаточную вместимость.

Результат:

```
CALL schedule_for_students.get_free_classrooms_nocursor(
```

```
1,
'2024-09-02 10:00+02',
'2024-09-02 12:00+02'
);
```

```
schedule=# SELECT * FROM tmp_free_classrooms;
```

classroom_number	seating_capacity	venue_id	room_type_id	department_id
3	25	1	3	1
6	30	1	2	1
7	30	1	2	1
8	200	1	4	1
9	167	1	1	1
10	137	1	1	1
11	152	1	1	1
101	30	1	3	1
102	30	1	3	1
103	30	1	3	1
201	60	1	1	1
301	30	1	2	1

(12 строк)

Процедура 2: Изменении расписания занятий для заданного преподавателя.

```
CREATE OR REPLACE PROCEDURE
schedule_for_students.update_teacher_schedule(
    IN p_teacher_id INT,
    IN p_lesson_id INT,
    IN p_new_start TIMESTAMPTZ,
    IN p_new_end TIMESTAMPTZ
)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE schedule_for_students."Lesson"
    SET start_datetime = p_new_start,
        end_datetime = p_new_end
    WHERE lesson_id = p_lesson_id
    AND teacher_id = p_teacher_id;

    IF NOT FOUND THEN
```

```

        RAISE EXCEPTION 'Нет занятия % у преподавателя %', p_lesson_id,
p_teacher_id;
    END IF;
END;
$$;

```

Изменяет время начала и конца конкретного занятия у указанного преподавателя; при отсутствии такого занятия выдаёт ошибку.

Результат:

10	11	2024-09-03 14:00:00+03	2024-09-03 15:30:00+03	1	2	1	3	2
11	12	2024-09-03 12:00:00+03	2024-09-03 13:30:00+03	6	2	4	2	4
12	13	2024-09-03 14:00:00+03	2024-09-03 15:30:00+03	6	2	4	2	4
13	14	2024-09-04 09:00:00+03	2024-09-04 10:30:00+03	4	1	2	2	1

```

CALL schedule_for_students.update_teacher_schedule(
6,
12,
'2024-09-03 09:00+03',
'2024-09-03 10:30+03'
);

```

28	30	2024-09-06 14:00:00+03	2024-09-06 15:30:00+03	3	3	1	3	3
29	12	2024-09-03 09:00:00+03	2024-09-03 10:30:00+03	6	2	4	2	4

Мы видим, что время проведения занятия поменялось

Процедура 3: Добавления записи с информацией о проведении лекционного занятия заданным преподавателем в заданную дату и время по заданной дисциплине.

```

CREATE OR REPLACE PROCEDURE schedule_for_students.add_lecture(
    IN p_teacher_id    INT,
    IN p_discipline_id INT,
    IN p_group_id      INT,
    IN p_start          TIMESTAMPTZ,
    IN p_end            TIMESTAMPTZ,
    IN p_classroom_number INT
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_lesson_type_id INT;
BEGIN
    SELECT lesson_type_id
    INTO v_lesson_type_id
    FROM schedule_for_students."Lesson_Type"
    WHERE lesson_type_name = 'Lecture'
    LIMIT 1;

    IF v_lesson_type_id IS NULL THEN

```

```

        RAISE EXCEPTION 'Тип занятия "Lecture" не найден';
    END IF;

```

```

INSERT INTO schedule_for_students."Lesson" (
    start_datetime, end_datetime,
    teacher_id, lesson_type_id,
    discipline_id, group_id, classroom_number
) VALUES (
    p_start, p_end,
    p_teacher_id, v_lesson_type_id,
    p_discipline_id, p_group_id, p_classroom_number
);
END;
$$;

```

Вставляет в расписание новое лекционное занятие для заданного преподавателя, дисциплины и группы в указанное время и аудиторию, автоматически подставляя тип «Lecture».

```

schedule=# select * from schedule_for_students."Lesson";

```

lesson_id	start_datetime	end_datetime	teacher_id	lesson_type_id	discipline_id	group_id	classroom_number
3	2024-09-02 10:00:00+03	2024-09-02 11:30:00+03	2	2	1	2	2
2	2024-09-02 10:00:00+03	2024-09-02 11:30:00+03	1	1	1	1	1
4	2024-09-02 10:00:00+03	2024-09-02 11:30:00+03	4	1	2	3	3
5	2024-09-02 12:00:00+03	2024-09-02 13:30:00+03	2	2	1	1	1
27	2024-09-05 12:00:00+03	2024-09-05 13:30:00+03	4	2	4	1	1
28	2024-09-06 10:00:00+03	2024-09-06 11:30:00+03	3	3	1	1	1
29	2024-09-06 12:00:00+03	2024-09-06 13:30:00+03	3	3	1	2	2
30	2024-09-06 14:00:00+03	2024-09-06 15:30:00+03	3	3	1	3	3
12	2024-09-03 09:00:00+03	2024-09-03 10:30:00+03	6	2	4	2	2

(29 строк)

```

CALL schedule_for_students.update_teacher_schedule(
    1, -- teacher_id
    31, -- lesson_id
    '2025-06-03 09:00+02',
    '2025-06-03 10:30+02'
);

```

28	2024-09-06 10:00:00+03	2024-09-06 11:30:00+03	3	3	1	1	3
29	2024-09-06 12:00:00+03	2024-09-06 13:30:00+03	3	3	1	2	3
30	2024-09-06 14:00:00+03	2024-09-06 15:30:00+03	3	3	1	3	3
12	2024-09-03 09:00:00+03	2024-09-03 10:30:00+03	6	2	4	2	4
31	2025-06-03 11:00:00+03	2025-06-03 12:30:00+03	1	1	2	2	5

(30 строк)

Информация обновилась, появилось занятие по заданному времени на заданную дату.

Задание 2. Создание триггеров.

Необходимо придумать и создать 7 триггеров для базы данных.

Триггер 1: `trg_teacher_conflict` (BEFORE INSERT OR UPDATE ON Lesson)

— вызывает функцию `fn_check_teacher_conflict()`.

Назначение: не допускает пересечения по времени (конфликта) разных занятий у одного преподавателя.

Что делает: вызывает функцию `fn_check_teacher_conflict()`, которая проверяет, не пересекается ли время нового или изменяемого занятия с уже существующими занятиями того же преподавателя. Если пересечение есть — выбрасывает `EXCEPTION` и не позволяет сохранить запись.

```
schedule=# -- 3.1 Проверка конфликтов у преподавателя (пересечение времени)
schedule=# CREATE OR REPLACE FUNCTION schedule_for_students.fn_check_teacher_conflict()
schedule=# RETURNS TRIGGER AS $$
schedule$$ BEGIN
schedule$$     IF EXISTS (
schedule$$         SELECT 1
schedule$$             FROM schedule_for_students."Lesson" l
schedule$$             WHERE l.teacher_id = NEW.teacher_id
schedule$$                 AND l.lesson_id <> COALESCE(NEW.lesson_id, -1)
schedule$$                 AND tstzrange(l.start_datetime, l.end_datetime)
schedule$$                     && tstzrange(NEW.start_datetime, NEW.end_datetime)
schedule$$     ) THEN
schedule$$         RAISE EXCEPTION 'Преподаватель % уже занят в это время', NEW.teacher_id;
schedule$$     END IF;
schedule$$     RETURN NEW;
schedule$$ END;
schedule$$ $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Результат:

```
schedule=# INSERT INTO schedule_for_students."Lesson" (lesson_id, start_datetime, end_datetime, teacher_id,
+02', '2025-06-10 12:00+02', 1, 1, 1, 1, 201);
INSERT 0 1
schedule=# INSERT INTO schedule_for_students."Lesson"
schedule=# (start_datetime, end_datetime, teacher_id,
schedule=# lesson_type_id, discipline_id, group_id, classroom_number)
schedule=# VALUES
schedule=# ('2025-06-10 11:00+02', '2025-06-10 13:00+02',
schedule=# 1, 1, 1, 1, 202);
ОШИБКА: Преподаватель 1 уже занят в это время
КОНТЕКСТ: функция PL/pgSQL schedule_for_students.fn_check_teacher_conflict(), строка 11, оператор RAISE
schedule=#
```

Триггер 2: `trg_classroom_conflict` (BEFORE INSERT OR UPDATE ON Lesson)

— вызывает функцию `fn_check_classroom_conflict()`.

Назначение: контролирует, чтобы одна аудитория не была задействована в двух занятиях одновременно.

Что делает: через `fn_check_classroom_conflict()` контролирует, чтобы аудитория (`classroom_number`) в новом или обновляемом уроке не была задействована в другом занятии в тот же промежуток времени. При конфликте — ошибка.

```
schedule=# -- 3.2 Проверка занятости аудитории
schedule=# CREATE OR REPLACE FUNCTION schedule_for_students.fn_check_classroom_conflict()
schedule=# RETURNS TRIGGER AS $$
schedule$$ BEGIN
schedule$$     IF EXISTS (
schedule$$         SELECT 1
schedule$$         FROM schedule_for_students."Lesson" l
schedule$$         WHERE l.classroom_number = NEW.classroom_number
schedule$$         AND l.lesson_id <> COALESCE(NEW.lesson_id, -1)
schedule$$         AND tstzrange(l.start_datetime, l.end_datetime)
schedule$$         && tstzrange(NEW.start_datetime, NEW.end_datetime)
schedule$$     ) THEN
schedule$$         RAISE EXCEPTION 'Аудитория % занята в указанный промежуток', NEW.classroom_number;
schedule$$     END IF;
schedule$$     RETURN NEW;
schedule$$ END;
schedule$$ $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Результат:

```
schedule=# INSERT INTO schedule_for_students."Lesson" (lesson_id, start_datetime, end_datetime, teacher_id,
+02', '2025-06-11 11:00+02', 2, 1, 1, 1, 301);
INSERT 0 1
schedule=# INSERT INTO schedule_for_students."Lesson"
schedule=# (start_datetime, end_datetime, teacher_id,
schedule=# lesson_type_id, discipline_id, group_id, classroom_number)
schedule=# VALUES
schedule=# ('2025-06-11 10:30+02', '2025-06-11 12:00+02',
schedule=# 3, 1, 1, 1, 301);
ОШИБКА: Аудитория 301 занята в указанный промежуток
КОНТЕКСТ: функция PL/pgSQL schedule_for_students.fn_check_classroom_conflict(), строка 11, оператор RAISE
schedule=#
```

Триггер 3: `trg_lesson_past` (BEFORE INSERT OR UPDATE ON Lesson)

— вызывает функцию `fn_check_lesson_date()`.

Назначение: запрещает планировать занятия на прошлые даты/время.

Что делает: функция `fn_check_lesson_date()` запрещает ставить начало занятия раньше текущего момента (`now()`). Если `NEW.start_datetime < now()` — выбрасывает EXCEPTION.

```

schedule=# -- 3.3 Запрет расписания в прошлом
schedule=# CREATE OR REPLACE FUNCTION schedule_for_students.fn_check_lesson_date()
schedule=# RETURNS TRIGGER AS $$
schedule$$ BEGIN
schedule$$     IF NEW.start_datetime < now() THEN
schedule$$         RAISE EXCEPTION 'Нельзя планировать занятие в прошлом: %', NEW.start_datetime;
schedule$$     END IF;
schedule$$     RETURN NEW;
schedule$$ END;
schedule$$ $$ LANGUAGE plpgsql;
CREATE FUNCTION
schedule=#

```

Результат:

```

schedule=# INSERT INTO schedule_for_students."Lesson"
schedule=#     (start_datetime, end_datetime, teacher_id,
schedule(#     lesson_type_id, discipline_id, group_id, classroom_number)
schedule=# VALUES
schedule=#     ('2020-01-01 09:00+02', '2020-01-01 10:30+02',
schedule(#     4, 1, 1, 1, 302);
ОШИБКА:  Нельзя планировать занятие в прошлом: 2020-01-01 10:00:00+03
КОНТЕКСТ:  функция PL/pgSQL schedule_for_students.fn_check_lesson_date(), строка 4, оператор RAISE
schedule=#

```

Триггер 4: trg_lesson_insert_log (AFTER INSERT ON Lesson)

— вызывает функцию fn_log_lesson_insert().

Назначение: после добавления новой записи в Lesson сохраняет в lesson_log информацию о времени начала и конца.

Что делает: функция fn_log_lesson_insert() добавляет в таблицу lesson_log запись с полями lesson_id, action='INSERT', new_start, new_end и меткой времени log_ts. Фактически фиксирует факт и время создания каждого нового занятия.

```

schedule=# -- 3.4 Логирование вставки занятия
schedule=# CREATE OR REPLACE FUNCTION schedule_for_students.fn_log_lesson_insert()
schedule=# RETURNS TRIGGER AS $$
schedule$# BEGIN
schedule$#     INSERT INTO schedule_for_students.lesson_log
schedule$#         (lesson_id, action, old_start, new_start, old_end, new_end)
schedule$#         VALUES
schedule$#             (NEW.lesson_id, 'INSERT', NULL, NEW.start_datetime, NULL, NEW.end_datetime);
schedule$#     RETURN NEW;
schedule$# END;
schedule$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
schedule=#
schedule=# CREATE TRIGGER trg_lesson_insert_log
schedule=# AFTER INSERT ON schedule_for_students."Lesson"
schedule=# FOR EACH ROW
schedule=# EXECUTE FUNCTION schedule_for_students.fn_log_lesson_insert();
CREATE TRIGGER

```

Результат:

```

schedule=# INSERT INTO schedule_for_students."Lesson" (start_datetime, end_datetime, teacher_id, lesson_type_id, discipline_id,
00+02', 3, 1, 2, 2, 301);
INSERT 0 1
schedule=# SELECT *
schedule=# FROM schedule_for_students.lesson_log
schedule=# WHERE action = 'INSERT'
schedule=# AND lesson_id = currval('schedule_for_students."Lesson_lesson_id_seq"');
 log_id | lesson_id | action | old_start | new_start | old_end | new_end | log_ts
-----+-----+-----+-----+-----+-----+-----+-----
      3 |        37 | INSERT |          | 2025-06-15 15:00:00+03 |          | 2025-06-15 17:00:00+03 | 2025-05-29 09:57:15.556602+03
(1 строка)

schedule=#

```

Триггер 5: trg_lesson_update_log (AFTER UPDATE ON Lesson)

— вызывает функцию fn_log_lesson_update().

Назначение: после изменения записи Lesson фиксирует в lesson_log старые и новые значения времени.

Что делает: функция fn_log_lesson_update() сохраняет в lesson_log старые (OLD.start_datetime, OLD.end_datetime) и новые (NEW.start_datetime, NEW.end_datetime) значения времени занятия, помечая action='UPDATE'.

Позволяет отслеживать историю всех переносов/изменений.

Результат:

```
schedule=# UPDATE schedule_for_students."Lesson"
schedule=#     SET start_datetime = '2025-06-15 15:00+02',
schedule=#         end_datetime   = '2025-06-15 17:00+02'
schedule=# WHERE lesson_id = 1001;
UPDATE 1
schedule=# SELECT *
schedule=#     FROM schedule_for_students.lesson_log
schedule=# WHERE action = 'UPDATE'
schedule=#     AND lesson_id = 1003
schedule=# ORDER BY log_ts DESC
schedule=# LIMIT 1;
```

1001

```
schedule=# SELECT * FROM schedule_for_students.lesson_log WHERE action = 'UPDATE' AND lesson_id = 1001 ORDER BY log_ts DESC LIMIT 1;
 log_id | lesson_id | action | old_start | new_start | old_end | new_end | log_ts
-----+-----+-----+-----+-----+-----+-----+-----
      4 |      1001 | UPDATE | 2025-06-10 11:00:00+03 | 2025-06-15 16:00:00+03 | 2025-06-10 13:00:00+03 | 2025-06-15 18:00:00+03 | 2025-05-29 09:58:51.833191+03
(1 строка)
```

Триггер 6: `trg_membership_no_overlap` (BEFORE INSERT OR UPDATE ON `Student_Group_Membership`) — вызывает функцию `fn_membership_no_overlap()`.

Назначение: препятствует тому, чтобы один и тот же студент (`student_id`) имел в одной и той же группе (`group_id`) две записи членства с пересекающимися датами.

Что делает: функция `fn_membership_no_overlap()` проверяет, не существует ли для того же `student_id` и `group_id` уже действующего (или изменяемого) членства с диапазоном дат, пересекающим диапазон `NEW.membership_start_date–NEW.membership_end_date`. Если пересечение есть — выбрасывается `EXCEPTION`, и новая/изменённая запись не сохраняется.

```

schedule-# AS $$
schedule$$ BEGIN
schedule$$   IF EXISTS (
schedule$$     SELECT 1
schedule$$       FROM schedule_for_students."Student_Group_Membership" m
schedule$$       WHERE m.student_id = NEW.student_id
schedule$$         AND m.group_id = NEW.group_id
schedule$$         AND m.membership_id <> COALESCE(NEW.membership_id, -1)
schedule$$         AND daterange(
schedule$$           m.membership_start_date,
schedule$$           COALESCE(m.membership_end_date, 'infinity'),
schedule$$           '[]'
schedule$$         )
schedule$$       && daterange(
schedule$$         NEW.membership_start_date,
schedule$$         COALESCE(NEW.membership_end_date, 'infinity'),
schedule$$         '[]'
schedule$$       )
schedule$$   ) THEN
schedule$$     RAISE EXCEPTION
schedule$$       'У студента % в группе % уже есть запись в пересекающийся период',
schedule$$       NEW.student_id,
schedule$$       NEW.group_id;
schedule$$   END IF;
schedule$$   RETURN NEW;
schedule$$ END;
schedule$$ $$;
CREATE FUNCTION

```

Результат:

-- Вставляем «базовое» членство без конфликтов

```

INSERT INTO schedule_for_students."Student_Group_Membership"
(student_group_membership_id, student_id, group_id, membership_start_date,
membership_end_date)
VALUES
(2001, 10, 3, '2025-01-01', '2025-03-01');

```

```

25-03-01');
INSERT 0 1

```

-- Вставляем новое членство того же студента в той же группе, не пересекающееся

```

INSERT INTO schedule_for_students."Student_Group_Membership"
(student_group_membership_id, student_id, group_id, membership_start_date,
membership_end_date)
VALUES

```

```
(2002, 10, 3, '2025-03-02', '2025-06-01');
```

```
25-06-01');  
INSERT 0 1
```

-- Пробуем вставить пересекающееся членство

```
INSERT INTO schedule_for_students."Student_Group_Membership"  
(student_group_membership_id, student_id, group_id, membership_start_date,  
membership_end_date)
```

```
VALUES
```

```
(2003, 10, 3, '2025-02-15', '2025-04-01');
```

```
25-04-01');  
ОШИБКА: У студента 10 в группе 3 уже есть запись в пересекающийся период  
КОНТЕКСТ: функция PL/pgSQL schedule_for_students.fn_membership_no_overlap/
```

-- Пробуем обновление существующей записи на конфликтный период

```
UPDATE schedule_for_students."Student_Group_Membership"
```

```
SET membership_start_date = '2025-02-01',
```

```
membership_end_date = '2025-05-01'
```

```
WHERE student_group_membership_id = 2002;
```

```
=# UPDATE schedule_for_students."Student_Group_Membership" SET me  
У студента 10 в группе 3 уже есть запись в пересекающийся период  
функция PL/pgSQL schedule_for_students.fn_membership_no_overlap/
```

--смотрим какие записи остались в таблице

```
КОНТЕКСТ: функция PL/pgSQL schedule_for_students.fn_membership_no_overlap(), строка 21, оператор RAISE  
schedule=# SELECT *  
schedule=# FROM schedule_for_students."Student_Group_Membership"  
schedule=# ORDER BY student_group_membership_id;  
 student_group_membership_id | group_id | student_id | membership_start_date | membership_end_date | status  
-----+-----+-----+-----+-----+-----  
1 | 1 | 12 | 2023-09-01 | |  
2 | 1 | 11 | 2023-09-01 | |
```

```
35 | 3 | 26 | 2023-09-01 | |  
36 | 3 | 25 | 2023-09-01 | |  
2001 | 3 | 10 | 2025-01-01 | 2025-03-01 |  
2002 | 3 | 10 | 2025-03-02 | 2025-06-01 |  
(38 строк)
```

Триггер 7: `trg_position_validate_overlap` (BEFORE INSERT OR UPDATE ON Teacher_Position) — вызывает функцию `fn_validate_teacher_position_overlap()`.

Назначение: не позволяет создавать пересекающиеся по датам записи о должностях одного и того же преподавателя.

Что делает: функция `fn_validate_teacher_position_overlap()` проверяет, чтобы у одного и того же преподавателя (`teacher_id`) периоды разных должностей (`start_date`–`end_date`) не перекрывались. При пересечении диапазонов — **EXCEPTION**.

```
schedule=# -- 3.7 Валидация перекрытия должностей преподавателя
schedule=# CREATE OR REPLACE FUNCTION schedule_for_students.fn_validate_teacher_position_overlap()
schedule=# RETURNS TRIGGER AS $$
schedule$# BEGIN
schedule$#     IF EXISTS (
schedule$#         SELECT 1
schedule$#         FROM schedule_for_students."Teacher_Position" tp
schedule$#         WHERE tp.teacher_id = NEW.teacher_id
schedule$#         AND tp.teacher_position_id <> COALESCE(NEW.teacher_position_id, -1)
schedule$#         AND daterange(tp.start_date, COALESCE(tp.end_date, 'infinity'), '[]')
schedule$#         && daterange(NEW.start_date, COALESCE(NEW.end_date, 'infinity'), '[]')
schedule$#     ) THEN
schedule$#         RAISE EXCEPTION 'У преподавателя % перекрываются периоды должностей', NEW.teacher_id;
schedule$#     END IF;
schedule$#     RETURN NEW;
schedule$# END;
schedule$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
```

Результат:

```
INSERT INTO schedule_for_students."Teacher_Position"
(teacher_position_id, teacher_id, start_date, end_date, position_id)
VALUES
(3001, 5, '2025-01-01', '2025-06-30', 5);
```

```
schedule=# INSERT INTO schedule_for_students."Teacher_Position"
ОШИБКА: У преподавателя 5 перекрываются периоды должностей
```

Задание 3 Доп задание.

Требовалось исправить триггер, который был дан во время выполнения практической работы

```
CREATE OR REPLACE FUNCTION fn_check_time_punch()
RETURNS TRIGGER AS $$
DECLARE
    last_punch RECORD;
BEGIN
    -- Поиск последней записи для сотрудника
    SELECT is_out_punch, punch_time
```



```

    INTO last_punch
  FROM time_punch
 WHERE employee_id = NEW.employee_id
 ORDER BY id DESC
 LIMIT 1;

  -- Если это первая запись для сотрудника, разрешаем вставку
  IF last_punch IS NULL THEN
    RETURN NEW;
  END IF;

  -- Проверка, что новое время больше предыдущего
  IF NEW.punch_time <= last_punch.punch_time THEN
    RETURN NULL;
  END IF;

  -- Проверка, что действия вход/выход не повторяются
  IF NEW.is_out_punch = last_punch.is_out_punch THEN
    RETURN NULL;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

1. Проверка времени:

Добавлена проверка `NEW.punch_time <= last_punch.punch_time`. Если новое время меньше или равно предыдущему, триггер возвращает `NULL`, и вставка отменяется.

2. Проверка повторяющихся действий:

Сохранена исходная логика: если `NEW.is_out_punch` совпадает с последним `is_out_punch` для того же сотрудника, вставка отменяется (`RETURN NULL`).

3. Обработка первой записи:

Если для сотрудника нет предыдущих записей (`last_punch IS NULL`), вставка разрешается (`RETURN NEW`).

Вывод

В процессе выполнения лабораторной работы были освоены навыки разработки и использования функций и триггеров в СУБД PostgreSQL. Созданы пользовательские функции для обработки данных и триггеры, обеспечивающие контроль целостности информации. Также был доработан триггер учёта входа/выхода сотрудников, в котором добавлены проверки на последовательность действий и корректность времени, что улучшило точность фиксации рабочего времени.