

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

ОТЧЕТ

по Лабораторной работе № 5

**«Запросы на выборку и модификацию данных. Представления. Работа с
индексами»**

по дисциплине «Проектирование и реализация баз данных»

Обучающиеся Дедкова Анастасия Викторовна

Факультет прикладной информатики

Группа К3240

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2023

Преподаватель Говорова Марина Михайловна

**Санкт-Петербург
2024/2025**

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 Выполнение.....	4
1.1 Разработанные объекты по индивидуальному заданию.....	4
1.2 Триггеры.....	9
1.3 Дополнительное задание.....	18
ЗАКЛЮЧЕНИЕ.....	27

ВВЕДЕНИЕ

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

Практическое задание:

– создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры,

– 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).

Дополнительные баллы - 3:

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).

1 Выполнение

1.1 Разработанные объекты по индивидуальному заданию

Индивидуальное задание 4, согласно варианту 7 лабораторной работы 2.

Создать хранимые процедуры:

– для получения расписания занятий для групп на определенный день недели,

```
CREATE OR REPLACE FUNCTION
courses_scheme.get_schedule_for_group_day_date(
    input_group_number VARCHAR,
    input_week_day courses_scheme.week_day_type,
    input_class_date DATE
)
RETURNS TABLE (
    group_number VARCHAR,
    class_id INTEGER,
    teacher_name VARCHAR,
    discipline_name VARCHAR,
    room_number VARCHAR,
    class_type VARCHAR,
    class_date DATE,
    week_day courses_scheme.week_day_type,
    class_number VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        g.group_number,
```

```

c.class_id,
(t.last_name || ' ' || t.name_middlename)::VARCHAR AS teacher_name,
d.discipline_name,
r.room_number,
c.class_type::VARCHAR,
c.class_date,
c.week_day,
c.class_number::VARCHAR
FROM courses_scheme.classes c
JOIN courses_scheme.groups g ON c.group_id = g.group_id
JOIN courses_scheme.teachers t ON c.teacher_id = t.teacher_id
JOIN courses_scheme.disciplines d ON c.discipline_id = d.discipline_id
JOIN courses_scheme.rooms r ON c.room_id = r.room_id
WHERE g.group_number = input_group_number
      AND c.week_day = input_week_day
      AND c.class_date = input_class_date
ORDER BY c.class_number;
END;
$$;

```

Код для вызова функции:

```

SELECT * FROM
courses_scheme.get_schedule_for_group_day_date('WD_1',
'понедельник'::courses_scheme.week_day_type, '2025-05-05');

```

На рисунке 1 результат вызова функции 1.

group_number	class_id	teacher_name	discipline_name	room_number	class_type	class_date	week_day	class_number
WD_1	81	Зайцева Екатерина Петровна	Веб-разработка	111	лекционное	2025-05-05	понедельник	1
WD_1	82	Туманов Роман Викторович	UX/UI-дизайн	112	практическое	2025-05-05	понедельник	2
WD_1	83	Михайлов Кирилл Игоревич	Мобильная разработка	113	практическое	2025-05-05	понедельник	3

(3 строки)

Рисунок 1 – Результат выполнения функции 1.

– записи на курс слушателя,

```

CREATE OR REPLACE PROCEDURE
courses_scheme.enroll_student_to_program(
    IN p_student_id INTEGER,
    IN p_program_id INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
    target_group_id INTEGER;
    start_edu_date DATE;
BEGIN
    -- Находим подходящую группу
    SELECT g.group_id, g.start_date
    INTO target_group_id, start_edu_date
    FROM courses_scheme.groups g
        LEFT JOIN courses_scheme.current_students cs ON g.group_id =
cs.group_id
        WHERE g.program_id = p_program_id
            AND (g.max_students > (SELECT COUNT(*) FROM
courses_scheme.current_students WHERE group_id = g.group_id))
            AND g.start_date <= CURRENT_DATE
            AND g.end_date >= CURRENT_DATE
    ORDER BY g.start_date
    LIMIT 1;

    IF target_group_id IS NULL THEN
        RAISE EXCEPTION 'Нет доступных групп для программы с id %',
p_program_id;
    END IF;

```

```

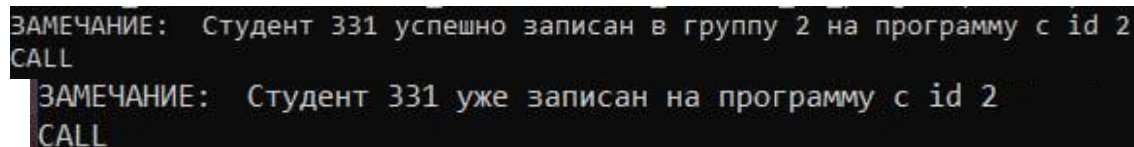
IF EXISTS (
    SELECT 1 FROM courses_scheme.current_students cs
    JOIN courses_scheme.groups g2 ON cs.group_id = g2.group_id
    WHERE cs.student_id = p_student_id
    AND g2.program_id = p_program_id
) THEN
    RAISE NOTICE 'Студент % уже записан на программу с id %',
p_student_id, p_program_id;
    RETURN;
END IF;
INSERT INTO courses_scheme.current_students (group_id, student_id,
status_edu, date_start_edu)
VALUES (target_group_id, p_student_id, 'студент', start_edu_date);
RAISE NOTICE 'Студент % успешно записан в группу % на
программу с id %', p_student_id, target_group_id, p_program_id;
END;
$$;

```

Код для вызова процедуры:

```
CALL courses_scheme.enroll_student_to_program(331, 2);
```

На рисунке 2 результат вызова процедуры 2.



```

ЗАМЕЧАНИЕ: Студент 331 успешно записан в группу 2 на программу с id 2
CALL
ЗАМЕЧАНИЕ: Студент 331 уже записан на программу с id 2
CALL

```

Рисунок 2 – Результат выполнения процедуры 2.

– получения перечня свободных лекционных аудиторий на любой день недели. Если свободных аудиторий не имеется, то выдать соответствующее сообщение.

```

CREATE OR REPLACE FUNCTION
courses_scheme.get_free_lecture_rooms(input_week_day VARCHAR, input_date
DATE)
    RETURNS TABLE (
        room_number VARCHAR,
        room_id INTEGER
    ) AS $$
BEGIN
    RETURN QUERY
    SELECT r.room_number, r.room_id
    FROM courses_scheme.rooms r
    LEFT JOIN courses_scheme.classes c
    ON r.room_id = c.room_id
    AND c.week_day::VARCHAR = input_week_day
    AND c.class_date = input_date
    WHERE r.room_type = 'лекционная'
    AND c.class_id IS NULL;
    IF NOT FOUND THEN
        RAISE NOTICE 'Нет свободных лекционных аудиторий на
указанный день и дату.';
    END IF;
END;
$$ LANGUAGE plpgsql;
Код для вызова функции:
SELECT * FROM courses_scheme.get_free_lecture_rooms('понедельник',
'2025-04-06');

```

На рисунке 3 результат вызова функции 3.

room_number	room_id
111	1
118	8
215	15
311	16
315	20
122	22
128	28
220	30
224	34
321	38
(10 строк)	

Рисунок 3 – Результат выполнения функции 3.

1.2 Триггеры

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).

Выбираем вариант 2.2

1. Автозаполнение дня недели по дате при создании или обновлении записи в расписании - триггер 1

```
CREATE OR REPLACE FUNCTION courses_scheme.fn_set_class_week_day()
```

```
RETURNS trigger AS
```

```
$$
```

```
BEGIN
```

```
NEW.week_day := CASE EXTRACT(ISODOW FROM NEW.class_date)
```

```
WHEN 1 THEN 'понедельник'::courses_scheme.week_day_type
```

```
WHEN 2 THEN 'вторник'::courses_scheme.week_day_type
```

```
WHEN 3 THEN 'среда'::courses_scheme.week_day_type
```

```
WHEN 4 THEN 'четверг'::courses_scheme.week_day_type
```

```
WHEN 5 THEN 'пятница'::courses_scheme.week_day_type
```

```
WHEN 6 THEN 'суббота'::courses_scheme.week_day_type
```

```

    WHEN 7 THEN 'воскресенье'::courses_scheme.week_day_type
END;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_set_class_week_day
    BEFORE INSERT OR UPDATE ON courses_scheme.classes
    FOR EACH ROW
    EXECUTE FUNCTION courses_scheme.fn_set_class_week_day();

```

Запрос для проверки работы триггера:

```

INSERT INTO courses_scheme.classes
    (teacher_id, discipline_id, group_id, room_id, class_type, class_date, class_number,
    class_status)
VALUES
    (5, 9, 2, 2, 'практическое', '2025-05-19', '2', 'Yes')
RETURNING class_id, class_date, week_day;

```

На рисунке 4 результат проверки триггера 1.

class_id	class_date	week_day
178	2025-05-19	понедельник

(1 строка)

Рисунок 4 – Результат проверки триггера 1.

2. Запрет на удаление группы, если в ней есть студенты - триггер 2

```

CREATE OR REPLACE FUNCTION courses_scheme.fn_prevent_group_delete()
    RETURNS trigger AS
$$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM courses_scheme.current_students

```

```

WHERE group_id = OLD.group_id
AND status_edu = 'студент'
) THEN
RAISE EXCEPTION
'Удаление невозможно: в группе % есть студенты со статусом "студент"',
OLD.group_id;
END IF;
RETURN OLD;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_prevent_group_delete
BEFORE DELETE ON courses_scheme.groups
FOR EACH ROW EXECUTE FUNCTION
courses_scheme.fn_prevent_group_delete();
Запрос для проверки работы триггера:
DELETE FROM courses_scheme.groups
WHERE group_id = 11;
На рисунке 5 результат проверки триггера 2.

```

```

ОШИБКА: Удаление невозможно: в группе 11 есть студенты со статусом "студент"
КОНТЕКСТ: функция PL/pgSQL fn_prevent_group_delete(), строка 9, оператор RAISE

```

Рисунок 5 – Результат проверки триггера 2.

3. Проверка, что дата зачисления студента попадает в период существования группы - триггер 3

```

CREATE OR REPLACE FUNCTION
courses_scheme.fn_validate_curr_student_date()
RETURNS trigger AS
$$
DECLARE
grp_start date;

```

```

grp_end date;
BEGIN
SELECT start_date, end_date
  INTO grp_start, grp_end
FROM courses_scheme.groups
WHERE group_id = NEW.group_id;
IF NEW.date_start_edu < grp_start
  OR (grp_end IS NOT NULL AND NEW.date_start_edu > grp_end) THEN
  RAISE EXCEPTION
    'Дата начала обучения (%) выходит за пределы периода группы % [% – %]',
    NEW.date_start_edu, NEW.group_id, grp_start, grp_end;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER trg_validate_curr_student_date
  BEFORE INSERT OR UPDATE ON courses_scheme.current_students
  FOR EACH ROW EXECUTE FUNCTION
courses_scheme.fn_validate_curr_student_date();
Запрос для проверки работы триггера (создание записи):
INSERT INTO courses_scheme.current_students
  (group_id, student_id, status_edu, date_start_edu)
VALUES
  (2, 331, 'студент', '2024-01-01');

```

На рисунке 6 результат проверки триггера 3 при создании записи.

```

ОШИБКА: Дата начала обучения (2024-01-01) выходит за пределы периода группы 2 [2025-02-01 - 2025-07-31]
КОНТЕКСТ: функция PL/pgSQL fn_validate_curr_student_date(), строка 13, оператор RAISE

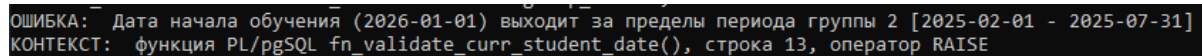
```

Рисунок 6 – Результат проверки триггера 3 при создании записи.

Запрос для проверки работы триггера (обновление записи):

```
UPDATE courses_scheme.current_students  
    SET date_start_edu = '2026-01-01'  
WHERE student_id = 157  
    AND group_id = 2;
```

На рисунке 7 результат проверки триггера 3 при обновлении записи.



ОШИБКА: Дата начала обучения (2026-01-01) выходит за пределы периода группы 2 [2025-02-01 - 2025-07-31]
КОНТЕКСТ: функция PL/pgSQL fn_validate_curr_student_date(), строка 13, оператор RAISE

Рисунок 7 – Результат проверки триггера 3 при обновлении записи.

4. Проверка дублирования записи об участии студента в практике - триггер 4

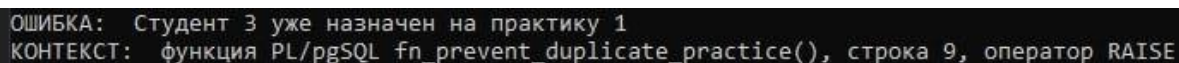
```
CREATE OR REPLACE FUNCTION  
courses_scheme.fn_prevent_duplicate_practice()  
    RETURNS trigger AS  
$$  
BEGIN  
    IF EXISTS (  
        SELECT 1  
        FROM courses_scheme.student_practice  
        WHERE curr_stud_id = NEW.curr_stud_id  
            AND practice_id = NEW.practice_id  
    ) THEN  
        RAISE EXCEPTION  
        'Студент % уже назначен на практику %',  
        NEW.curr_stud_id, NEW.practice_id;  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_prevent_duplicate_practice
  BEFORE INSERT ON courses_scheme.student_practice
  FOR EACH ROW EXECUTE FUNCTION
  courses_scheme.fn_prevent_duplicate_practice();
```

Запрос для проверки работы триггера:

```
INSERT INTO courses_scheme.student_practice (practice_id, curr_stud_id)
VALUES (1, 3);
```

На рисунке 7 результат проверки триггера 4.



```
ОШИБКА: Студент 3 уже назначен на практику 1
КОНТЕКСТ: функция PL/pgSQL fn_prevent_duplicate_practice(), строка 9, оператор RAISE
```

Рисунок 7 – Результат проверки триггера 4.

5. Проверка, что преподаватель действительно «привязан» к дисциплине, при создании или обновлении записи в расписании - триггер 5

```
CREATE OR REPLACE FUNCTION
```

```
courses_scheme.fn_validate_teacher_qualification()
```

```
  RETURNS trigger AS
```

```
  $$
```

```
  BEGIN
```

```
    IF NOT EXISTS (
```

```
      SELECT 1
```

```
        FROM courses_scheme.teacher_discipline
```

```
        WHERE teacher_id = NEW.teacher_id
```

```
          AND discipline_id = NEW.discipline_id
```

```
    ) THEN
```

```
      RAISE EXCEPTION
```

```
        'Преподаватель % не может быть назначен для выбранной дисциплины %',
```

```
        NEW.teacher_id, NEW.discipline_id;
```

```
    END IF;
```

```

RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_validate_teacher_qualification
BEFORE INSERT OR UPDATE ON courses_scheme.classes
FOR EACH ROW EXECUTE FUNCTION
courses_scheme.fn_validate_teacher_qualification();

```

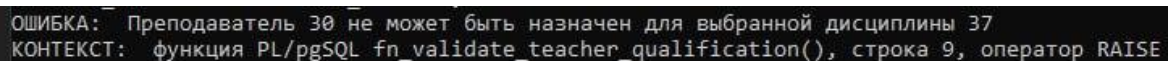
Запрос для проверки работы триггера:

```

SELECT class_id, teacher_id, discipline_id
FROM courses_scheme.classes
WHERE class_id = 97;

```

На рисунке 8 результат проверки триггера 5.



```

ОШИБКА: Преподаватель 30 не может быть назначен для выбранной дисциплины 37
КОНТЕКСТ: функция PL/pgSQL fn_validate_teacher_qualification(), строка 9, оператор RAISE

```

Рисунок 8 – Результат проверки триггера 5.

6. Проверка, что преподаватель не назначен одновременно на две пары в одно и то же время и дату - триггер 6

```

CREATE OR REPLACE FUNCTION
courses_scheme.fn_check_teacher_schedule_conflict()
RETURNS trigger AS
$$
DECLARE
    conflict_count int;
BEGIN
    SELECT COUNT(*)
    INTO conflict_count

```

```

FROM courses_scheme.classes
WHERE teacher_id = NEW.teacher_id
      AND class_date = NEW.class_date
      AND class_number = NEW.class_number
      AND class_id <> COALESCE(NEW.class_id, -1);

IF conflict_count > 0 THEN
    RAISE EXCEPTION
        'Преподаватель % уже назначен на пару % в этот день %',
        NEW.teacher_id, NEW.class_number, NEW.class_date;
END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_teacher_schedule_conflict
    BEFORE INSERT OR UPDATE ON courses_scheme.classes
    FOR EACH ROW
    EXECUTE FUNCTION courses_scheme.fn_check_teacher_schedule_conflict();
Запрос для проверки работы триггера:
INSERT INTO courses_scheme.classes
    (teacher_id, discipline_id, group_id, room_id, class_type, class_date, class_number,
class_status)
VALUES

```



```
(5, 9, 4, 5, 'практическое', '2025-05-12', '2', 'Yes' );
```

На рисунке 9 результат проверки триггера 6.

```
ОШИБКА: Преподаватель 5 уже назначен на пару 2 в этот день 2025-05-12
КОНТЕКСТ: функция PL/pgSQL fn_check_teacher_schedule_conflict(), строка 14, оператор RAISE
```

Рисунок 9 – Результат проверки триггера 6.

7. Проверка вхождения дисциплины в программу группы при добавлении или изменении записи в расписании - триггер 7

CREATE OR REPLACE FUNCTION

courses_scheme.fn_check_discipline_in_group_program()

RETURNS trigger AS

\$\$

DECLARE

cnt integer;

BEGIN

SELECT COUNT(*)

INTO cnt

FROM courses_scheme.program_discipline pd

JOIN courses_scheme.groups g ON g.program_id = pd.program_id

WHERE g.group_id = NEW.group_id

AND pd.discipline_id = NEW.discipline_id;

IF cnt = 0 THEN

RAISE EXCEPTION

'Дисциплина % не входит в программу группы %',

NEW.discipline_id, NEW.group_id;

END IF;

```

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_discipline_in_group_program
    BEFORE INSERT OR UPDATE ON courses_scheme.classes
    FOR EACH ROW EXECUTE FUNCTION
courses_scheme.fn_check_discipline_in_group_program();

```

Запрос для проверки работы триггера:

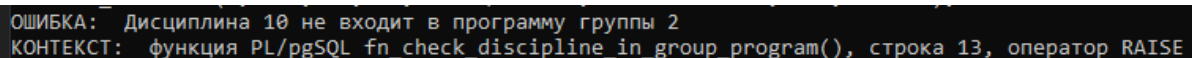
```

INSERT INTO courses_scheme.classes
    (teacher_id, discipline_id, group_id, room_id, class_type, class_date, class_number,
class_status)

VALUES
    (5, 10, 2, 3, 'лекционное', '2025-05-21', '1', 'Yes');

```

На рисунке 10 результат проверки триггера 7.



```

ОШИБКА: Дисциплина 10 не входит в программу группы 2
КОНТЕКСТ: функция PL/pgSQL fn_check_discipline_in_group_program(), строка 13, оператор RAISE

```

Рисунок 10 – Результат проверки триггера 7.

1.3 Дополнительное задание

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).

```

CREATE OR REPLACE FUNCTION fn_check_time_punch()
RETURNS trigger
LANGUAGE plpgsql
AS
$$
DECLARE
    prev_rec time_punch;
    next_rec time_punch;
    min_gap interval := interval '1 minute'; -- минимальный
интервал между входом и выходом
BEGIN
    -- Предыдущая запись того же сотрудника с punch_time <
NEW.punch_time
    SELECT *
    INTO prev_rec
    FROM time_punch
    WHERE employee_id = NEW.employee_id
    AND punch_time < NEW.punch_time
    ORDER BY punch_time DESC
    LIMIT 1;

    -- Следующая запись того же сотрудника с punch_time >
NEW.punch_time
    SELECT *
    INTO next_rec
    FROM time_punch
    WHERE employee_id = NEW.employee_id
    AND punch_time > NEW.punch_time
    ORDER BY punch_time ASC
    LIMIT 1;

    -- 1. Если это первая запись сотрудника — должен быть вход
    IF prev_rec IS NULL THEN
        IF NEW.is_out_punch THEN
            RAISE EXCEPTION
            'Сначала требуется отметка входа (employee_id=%).',
            NEW.employee_id;

```

```

        END IF;
        RETURN NEW;
    END IF;

    -- 2. Запрет двух одинаковых записей подряд (назад)
    IF prev_rec IS NOT NULL AND NEW.is_out_punch =
prev_rec.is_out_punch THEN
        RAISE EXCEPTION
        'Два одинаковых события подряд (employee_id=%,
is_out_punch=%).',
        NEW.employee_id, NEW.is_out_punch;
    END IF;

    -- 3. Запрет двух одинаковых записей подряд (вперёд)
    IF next_rec IS NOT NULL AND NEW.is_out_punch =
next_rec.is_out_punch THEN
        RAISE EXCEPTION
        'Два одинаковых события подряд (employee_id=%,
is_out_punch=%).',
        NEW.employee_id, NEW.is_out_punch;
    END IF;

    -- 4. Время новой записи должно быть строго больше
предыдущей
    IF prev_rec IS NOT NULL AND NEW.punch_time <=
prev_rec.punch_time THEN
        RAISE EXCEPTION
        'Время должно быть позже предыдущего (new=%, prev=%).',
        NEW.punch_time, prev_rec.punch_time;
    END IF;

    -- 5. Если предыдущая запись — вход и она открыта на другой
день — запретить вставку
    IF date_trunc('day', NEW.punch_time) <> date_trunc('day',
prev_rec.punch_time)
        AND NOT prev_rec.is_out_punch
    THEN

```

```

        RAISE EXCEPTION
        'Предыдущий вход % не закрыт (employee_id=%). Сначала
отметьте выход.',
        prev_rec.punch_time, NEW.employee_id;
    END IF;

    -- 6. Если предыдущая запись — выход, но день не совпадает —
запретить вставку
    IF date_trunc('day', NEW.punch_time) <> date_trunc('day',
prev_rec.punch_time)
        AND prev_rec.is_out_punch
    THEN
        RAISE EXCEPTION
        'Вход и выход должны относиться к одному дню
(employee_id=%).',
        NEW.employee_id;
    END IF;

    -- 7. Минимальный интервал между входом и выходом
    IF NEW.is_out_punch
        AND NEW.punch_time - prev_rec.punch_time < min_gap THEN
        RAISE EXCEPTION
        'Интервал вход-выход меньше % (employee_id=%).',
        min_gap, NEW.employee_id;
    END IF;

    RETURN NEW;
END;
$$;

DROP TRIGGER IF EXISTS trg_check_time_punch ON time_punch;

CREATE TRIGGER trg_check_time_punch
BEFORE INSERT OR UPDATE ON time_punch
FOR EACH ROW
EXECUTE FUNCTION fn_check_time_punch();

```

Объяснение работы триггера:

Сначала находим у данного сотрудника (`employee_id = NEW.employee_id`) последнюю запись с временем строго раньше вставляемой (`punch_time < NEW.punch_time`). Это непосредственно предыдущая отметка по времени.

Затем находим у того же сотрудника первую запись со временем строго позже новой — следующую отметку по времени.

1. Если у сотрудника еще нет ни одной записи, значит, это первая. Первая отметка не может быть выходом — только входом. Если первая запись — выход, функция выдает ошибку.

2. Нельзя вставлять два входа подряд или два выхода подряд. Если предыдущая запись того же типа (IN или OUT), то ошибка.

3. То же самое правило, но проверяется следующая запись после вставляемой. Это нужно, чтобы предотвратить ситуацию, когда новая запись вставляется в середину и нарушает порядок (например, два выхода подряд).

4. Проверка хронологического порядка. Время новой отметки должно быть строго больше времени предыдущей.

5. Если предыдущая запись — вход, а новая относится уже к следующему календарному дню, значит, смена не закрыта выходом в прошлом дне. Проверка срабатывает на любом новом событии (IN или OUT), если предыдущий вход не закрыт выходом и при этом находится в другом календарном дне.

6. Интервал между входом и выходом должен быть не меньше минимально допустимого.

Тестирование работы триггера

1. Вход и выход в нормальном режиме:

```
\echo === ТЕСТ-1: валидная смена (IN → OUT)
```

```
BEGIN;
```

```
SAVEPOINT t1;
```

```
INSERT INTO time_punch
```

```
VALUES (DEFAULT, 1, false, '2025-01-01 09:00'); -- IN
```

```
INSERT INTO time_punch
```

```
VALUES (DEFAULT, 1, true, '2025-01-01 17:00'); -- OUT
```

RELEASE SAVEPOINT t1;

COMMIT;

```
=== ТЕСТ-1: валидная смена (IN > OUT)
emp=#
emp=# BEGIN;
BEGIN
emp=*#
emp=*# SAVEPOINT t1;
SAVEPOINT
emp=*#
emp=*# INSERT INTO time_punch
emp-*# VALUES (DEFAULT, 1, false, '2025-01-01 09:00'); -- IN
INSERT 0 1
emp=*# INSERT INTO time_punch
emp-*# VALUES (DEFAULT, 1, true, '2025-01-01 17:00'); -- OUT
INSERT 0 1
emp=*#
```

2. Первое событие - выход (должен сработать триггер):

\echo === ТЕСТ-2: первый punch = OUT

BEGIN;

SAVEPOINT t2;

INSERT INTO time_punch VALUES (DEFAULT, 2, true, '2025-01-01
10:00'); -- ожидаем ERROR

ROLLBACK TO t2;

COMMIT;

```
=== ТЕСТ-2: первый punch = OUT
emp=# BEGIN;
BEGIN
emp=*# SAVEPOINT t2;
SAVEPOINT
emp=*# INSERT INTO time_punch VALUES (DEFAULT, 2, true, '2025-01-01 10:00'); -- ожидаем ERROR
ОШИБКА: Сначала требуется отметка входа (employee_id=2).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 18, оператор RAISE
emp=!# ROLLBACK TO t2;
ROLLBACK
emp=*# COMMIT;
COMMIT
```

3. Два входа подряд (должен сработать триггер):

\echo === ТЕСТ-3: два IN подряд

BEGIN;

SAVEPOINT t3;

INSERT INTO time_punch VALUES (DEFAULT, 3, false, '2025-01-02
09:00'); -- IN

```

INSERT INTO time_punch VALUES (DEFAULT, 3, false, '2025-01-02
09:05'); -- ожидаем ERROR
ROLLBACK TO t3;
COMMIT;

```

```

emp=# \echo === ТЕСТ-3: два IN подряд
=== ТЕСТ-3: два IN подряд
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t3;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 3, false, '2025-01-02 09:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 3, false, '2025-01-02 09:05'); -- ожидаем ERROR
ОШИБКА: Два одинаковых события подряд (employee_id=3, is_out_punch=f).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 27, оператор RAISE

```

4. Два выхода подряд (должен сработать триггер):

```

\echo === ТЕСТ-4: два OUT подряд
BEGIN;
SAVEPOINT t4;
INSERT INTO time_punch VALUES (DEFAULT, 4, false, '2025-01-02
09:10'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 4, true, '2025-01-02
10:00'); -- OUT
INSERT INTO time_punch VALUES (DEFAULT, 4, true, '2025-01-02
10:05'); -- ожидаем ERROR
ROLLBACK TO t4;
COMMIT;

```

```

=== ТЕСТ-4: два OUT подряд
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t4;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 4, false, '2025-01-02 09:10'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 4, true, '2025-01-02 10:00'); -- OUT
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 4, true, '2025-01-02 10:05'); -- ожидаем ERROR
ОШИБКА: Два одинаковых события подряд (employee_id=4, is_out_punch=t).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 27, оператор RAISE

```

5. Выход раньше входа - нарушена хронология:

```

\echo === ТЕСТ-5: OUT раньше IN
BEGIN;
SAVEPOINT t5;
INSERT INTO time_punch VALUES (DEFAULT, 5, false, '2025-01-03
11:00'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 5, true, '2025-01-03
10:59'); -- ожидаем ERROR

```


ROLLBACK TO t5;
COMMIT;

```
=== ТЕСТ-5: OUT раньше IN
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t5;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 5, false, '2025-01-03 11:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 5, true, '2025-01-03 10:59'); -- ожидаем ERROR
ОШИБКА: Сначала требуется отметка входа (employee_id=5).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 18, оператор RAISE
```

6. Незакрытая смена (должен сработать триггер):

\echo === ТЕСТ-6: открытая смена через полночь

```
BEGIN;
SAVEPOINT t6;
INSERT INTO time_punch VALUES (DEFAULT, 6, false, '2025-01-03
23:30'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 6, false, '2025-01-04
08:00'); -- ожидаем ERROR
ROLLBACK TO t6;
COMMIT;
```

```
=== ТЕСТ-6: открытая смена через полночь
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t6;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 6, false, '2025-01-03 23:30'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 6, false, '2025-01-04 08:00'); -- ожидаем ERROR
ОШИБКА: Два одинаковых события подряд (employee_id=6, is_out_punch=f).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 27, оператор RAISE
```

7. Выход на следующий день (должен сработать триггер):

\echo === ТЕСТ-7: OUT на другой день

```
BEGIN;
SAVEPOINT t7;
INSERT INTO time_punch VALUES (DEFAULT, 7, false, '2025-01-04
22:00'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 7, true, '2025-01-05
00:30'); -- ожидаем ERROR
ROLLBACK TO t7;
COMMIT;
```

```

emp=# \echo === ТЕСТ-7: OUT на другой день
=== ТЕСТ-7: OUT на другой день
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t7;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 7, false, '2025-01-04 22:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 7, true, '2025-01-05 00:30'); -- ожидаем ERROR
ОШИБКА: Предыдущий вход 2025-01-04 22:00:00 не закрыт (employee_id=7). Сначала отметьте выход.
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 43, оператор RAISE

```

8. Проверка соблюдения минимального интервала:

\echo === ТЕСТ-8: слишком короткая смена

```

BEGIN;
SAVEPOINT t8;
INSERT INTO time_punch VALUES (DEFAULT, 8, false, '2025-01-05
14:00:00'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 8, true, '2025-01-05
14:00:30'); -- ожидаем ERROR
ROLLBACK TO t8;
COMMIT;

```

```

emp=# \echo === ТЕСТ-8: слишком короткая смена
=== ТЕСТ-8: слишком короткая смена
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t8;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 8, false, '2025-01-05 14:00:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 8, true, '2025-01-05 14:00:30'); -- ожидаем ERROR
ОШИБКА: Интервал вход-выход < 00:01:00 (employee_id=8).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 60, оператор RAISE

```

9. Вставка отметки выхода в середину закрытого интервала:

\echo === ТЕСТ-9: OUT посередине

```

BEGIN;
SAVEPOINT t9;
INSERT INTO time_punch VALUES (DEFAULT, 9, false, '2025-01-06
08:00'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-06
12:00'); -- OUT
INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-06
10:00'); -- ожидаем ERROR
ROLLBACK TO t9;
COMMIT;

```

```

=== ТЕСТ-9: OUT посередине
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t9;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 9, false, '2025-01-06 08:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-06 12:00'); -- OUT
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-06 10:00'); -- ожидаем ERROR
ОШИБКА: Два одинаковых события подряд (employee_id=9, is_out_punch=t).

```

10. Выход раньше входа (update):

```

\echo === ТЕСТ-10: UPDATE нарушает порядок
BEGIN;
SAVEPOINT t10;
INSERT INTO time_punch VALUES (DEFAULT, 9, false, '2025-01-07
09:00'); -- IN
INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-07
17:00'); -- OUT
UPDATE time_punch
SET punch_time = '2025-01-07 08:30' -- ожидаем ERROR
WHERE employee_id = 9 AND is_out_punch
AND punch_time = '2025-01-07 17:00';
ROLLBACK TO t10;
COMMIT;

```

```

emp=# \echo === ТЕСТ-10: UPDATE нарушает порядок
=== ТЕСТ-10: UPDATE нарушает порядок
emp=# BEGIN;
BEGIN
emp=# SAVEPOINT t10;
SAVEPOINT
emp=# INSERT INTO time_punch VALUES (DEFAULT, 9, false, '2025-01-07 09:00'); -- IN
INSERT 0 1
emp=# INSERT INTO time_punch VALUES (DEFAULT, 9, true, '2025-01-07 17:00'); -- OUT
INSERT 0 1
emp=# UPDATE time_punch
emp=# SET punch_time = '2025-01-07 08:30' -- ожидаем ERROR
emp=# WHERE employee_id = 9 AND is_out_punch
emp=# AND punch_time = '2025-01-07 17:00';
ОШИБКА: Сначала требуется отметка входа (employee_id=9).
КОНТЕКСТ: функция PL/pgSQL fn_check_time_punch(), строка 18, оператор RAISE

```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была успешно достигнута поставленная цель — овладение практическими навыками создания и

использования процедур, функций и триггеров в базе данных PostgreSQL. Работа включала реализацию процедур с использованием параметров IN и OUT, создание логики на стороне сервера с применением триггеров, а также выполнение дополнительных проверок бизнес-логики с учетом возможных узких мест.

Были разработаны следующие ключевые компоненты:

три хранимые процедуры:

- для получения расписания занятий на определенный день недели для заданной группы;
- для зачисления студента в подходящую учебную группу по программе;
- для получения списка свободных лекционных аудиторий на указанный день;

семь оригинальных триггеров, реализующих:

- автоматическое определение дня недели по дате занятия;
- запрет на удаление группы при наличии студентов;
- проверку корректности даты зачисления студента в группу;
- предотвращение дублирования записей о прохождении практики;
- проверку соответствия преподавателя преподаваемой дисциплине;
- недопущение назначения преподавателя на несколько занятий в одно и то же время;
- проверку соответствия дисциплины учебной программе группы.

Дополнительно был модифицирован триггер на проверку корректности входа и выхода сотрудника, реализующий комплексную валидацию данных с учётом распространённых ошибок: отсутствие выхода после входа, повторный вход без выхода, выход без входа, а также контроль временных интервалов между действиями.

Результат лабораторной работы подтверждает освоение широкого спектра механизмов, необходимых для обеспечения целостности и

корректности данных в PostgreSQL. Использование триггеров позволило перенести контроль бизнес-логики на уровень базы данных, минимизируя вероятность ошибок при работе с данными. Хранимые процедуры обеспечили удобство повторного использования логики в прикладных задачах.

Выводы:

В результате лабораторной работы были приобретены важные практические навыки по разработке и использованию серверной логики в PostgreSQL: создание процедур, написание триггерных функций, реализация проверок данных и автоматизация процессов в базе. Эти навыки критически важны для построения надежных, масштабируемых и управляемых информационных систем.