Министерство науки и высшего образования Российской Федерации ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО» (Университет ИТМО)

Факультет прикладной информатики

Образовательная программа Мобильные и сетевые технологии

Направление подготовки 09.03.03 Мобильные и сетевые технологии

О Т Ч Е Т

ЛАБОРАТОРНАЯ РАБОТА №5

“Процедуры, функции, триггеры в PostgreSQL”

Обучающийся: Цветкова Татьяна К3241

Преподаватель: Говорова Марина Михайловна

Санкт-
Петербург,
2025

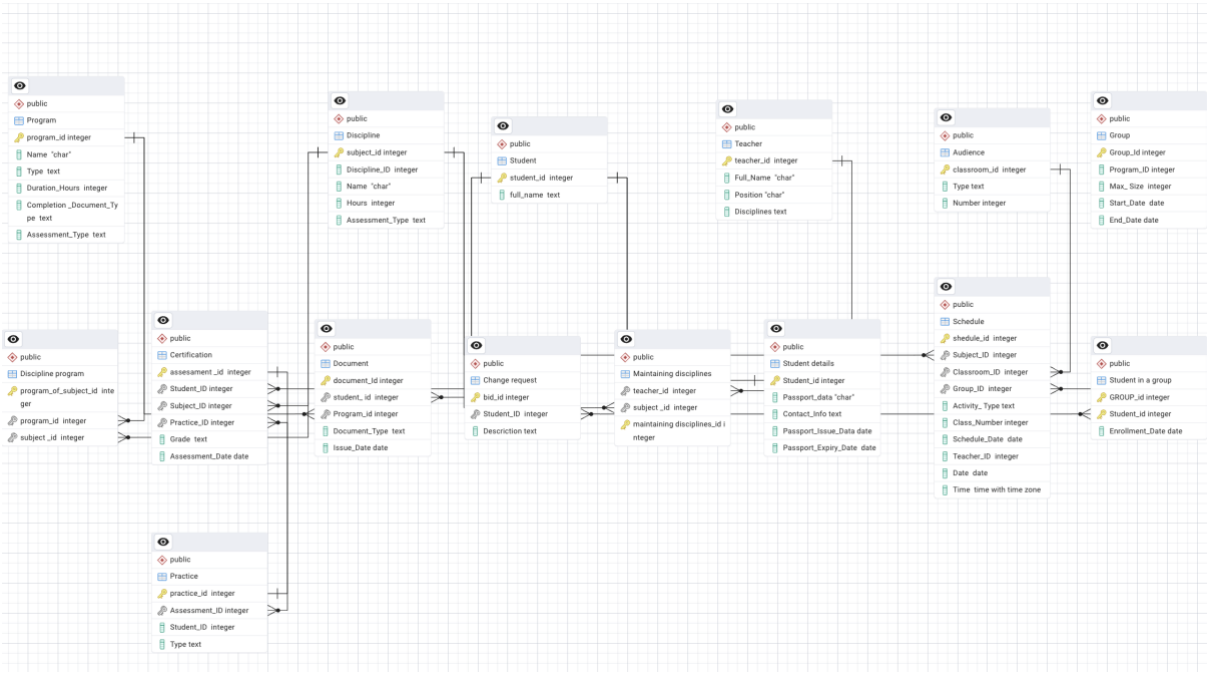
Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание:

По варианту индивидуальной БД (ЛР 2, часть IV) необходимо:

1. Создать три хранимые процедуры (CALL).
2. Разработать семь оригинальных триггеров (AFTER/BEFORE, ROW).
3. Проверить работу объектов в консоли psql, сделать скриншоты.

Схема базы данных:



Выполнение:

1. Скрипты кода разработанных объектов (процедур/функций) и подтверждающие скриншоты работы и результатов в psql согласно индивидуальному заданию (часть 4 и 5):

Я подготовила три процедуры, которые выполняют важные операции в базе данных: добавление нового студента вместе с его паспортными данными, запись результатов аттестации только для тех студентов, которые числятся в расписании по указанному предмету, и перевод студента из одной группы в другую с проверкой вместимости.

Процедура 1. Добавление нового студента с его паспортными данными

В БД у нас две таблицы: `Student (student_id, full_name)` и `Student_Data (student_id, passport_data, contact_info, passport_issue_date, passport_expire_date)`. Задача процедуры — одним вызовом создать и запись в `Student`, и запись в `Student_Data`.

```
CREATE OR REPLACE FUNCTION fn_add_student_with_data(
    p_full_name       VARCHAR,
    p_passport_data   CHAR(10),
    p_contact_info    VARCHAR,
    p_passport_issue  DATE,
    p_passport_expire DATE
) RETURNS INTEGER AS $$
DECLARE
    new_student_id INTEGER;
BEGIN
    -- Вставляем запись в таблицу Student и получаем новый student_id
    INSERT INTO Student(full_name)
    VALUES (p_full_name)
    RETURNING student_id INTO new_student_id;

    -- Вставляем запись в таблицу Student_Data
    INSERT INTO Student_Data(
      student_id,
      passport_data,
      contact_info,
      passport_issue_date,
      passport_expire_date
    ) VALUES (
      new_student_id,
      p_passport_data,
      p_contact_info,
      p_passport_issue,
      p_passport_expire
    );

    -- Возвращаем идентификатор созданного студента
    RETURN new_student_id;
END;
$$ LANGUAGE plpgsql;
```

Что делает:

1. Вставляет в `Student` новое ФИО студента.
2. Сохраняет детали паспорта + контакты в `Student_Data`.
3. Возвращает созданный `student_id` (INTEGER).

Процедура 2. Запись результатов аттестации (с проверкой, что у студента есть дисциплина в расписании)

Таблица `Attestation (assessment_id, student_id, subject_id, practice_hours, grade, assessment_date)` хранит результаты аттестации (экзамен/зачёт). Перед тем как записать оценку, проверим, что у этого студента действительно есть пара по этой дисциплине в расписании (иначе — бросим исключение).

```
CREATE OR REPLACE FUNCTION fn_record_attestation(
    p_student_id      INTEGER,
    p_subject_id      INTEGER,
    p_practice_hours  INTEGER,
    p_grade           INTEGER,
    p_assessment_date DATE
) RETURNS VOID AS $$
DECLARE
    tmp_cnt INTEGER;
BEGIN
    -- Проверяем, что в расписании есть пара для этой группы/студента по дисциплине p_subject_id
    SELECT COUNT(*)
      INTO tmp_cnt
    FROM Schedule AS s
    JOIN Student_in_Group AS sg ON sg.group_id = s.group_id
    WHERE sg.student_id = p_student_id
      AND s.subject_id = p_subject_id;

    IF tmp_cnt = 0 THEN
      RAISE EXCEPTION
        'Студент % не числится на дисциплине % согласно расписанию',
        p_student_id, p_subject_id;
    END IF;
```

```
    -- Вставляем запись в Attestation
    INSERT INTO Attestation(
      student_id,
      subject_id,
      practice_hours,
      grade,
      assessment_date
    ) VALUES (
      p_student_id,
      p_subject_id,
      p_practice_hours,
      p_grade,
      p_assessment_date
    );
END;
$$ LANGUAGE plpgsql;
```

- **Что делает:**

    1. Считает, есть ли в `Schedule` хотя бы одна пара для группы, где учится заданный `student_id`, по дисциплине `subject_id`.
    2. Если нет — выбрасывает `EXCEPTION`, блокирует вставку.
    3. Если да — делает `INSERT` в `Attestation`.

Процедура 3. Перевод (релокация) студента из одной группы в другую

Если студенту нужно изменить группу (например, перекомпоновка небольших групп). Мы должны:

1. Проверить, что новая группа ещё в диапазоне своих `max_size`.
2. Удалить старую запись из `Student_in_Group`.
3. Вставить новую запись с текущей датой в `Student_in_Group`.
4. Обновить поле `group_id` у записей в таблице `Schedule` и/или в таблицах, где хранится история (если необходимо).

```
CREATE OR REPLACE FUNCTION fn_transfer_student_between_groups(
    p_student_id   INTEGER,
    p_new_group_id INTEGER
```

```
) RETURNS VOID AS $$
DECLARE
    cur_count     INTEGER;
    cur_max       INTEGER;
    old_group_id  INTEGER;
BEGIN
    -- Находим текущую (последнюю) запись студента в Student_in_Group
    SELECT group_id
      INTO old_group_id
    FROM Student_in_Group
    WHERE student_id = p_student_id
    ORDER BY enrollment_date DESC
    LIMIT 1;

    IF old_group_id IS NULL THEN
      RAISE EXCEPTION
        'Студент % не состоит ни в одной группе',
        p_student_id;
    END IF;

    -- Проверяем текущую численность новой группы
    SELECT COUNT(*)
      INTO cur_count
    FROM Student_in_Group
    WHERE group_id = p_new_group_id;

    -- Получаем максимальный размер новой группы
    SELECT max_size
      INTO cur_max
    FROM "Group"
    WHERE group_id = p_new_group_id;

    IF cur_count >= cur_max THEN
      RAISE EXCEPTION
        'Группа % уже заполнена (max_size = %)',
        p_new_group_id, cur_max;
    END IF;

    -- Удаляем старую запись о принадлежности студента к группе
    DELETE FROM Student_in_Group
```

```
  WHERE student_id = p_student_id
    AND group_id = old_group_id;

  -- Вставляем новую запись с текущей датой
  INSERT INTO Student_in_Group(
    student_id,
    group_id,
    enrollment_date
  ) VALUES (
    p_student_id,
    p_new_group_id,
    CURRENT_DATE
  );
END;
$$ LANGUAGE plpgsql;
```

Что делает:

1. Находит, в какой группе сейчас студент (самая свежая запись по `enrollment_date`).
2. Проверяет, что в `p_new_group_id` ещё есть место (`COUNT < max_size`).
3. Если всё ок — удаляет старую строку из `Student_in_Group` и создаёт новую.

1. **Триггер: аудит изменений в паспортных/контактных данных студента**

Таблица: "Student details"

Аудит-таблица: student_data_audit

```
CREATE OR REPLACE FUNCTION fn_audit_student_data()
 RETURNS TRIGGER AS $$
BEGIN
 -- Я вставляю старые данные студента в таблицу аудита
 INSERT INTO student_data_audit (
```

```
    student_id,       -- id студента
    old_passport,     -- прежние паспортные данные
    old_contact,      -- прежние контактные данные
    old_issue_date,   -- прежняя дата выдачи паспорта
    old_expire_date,  -- прежняя дата истечения паспорта
    changed_at,       -- момент изменения
    changed_by        -- кто изменил (пользователь БД)
  ) VALUES (
    OLD.student_id,
    OLD.passport_data,
    OLD.contact_info,
    OLD.passport_issue_date,
    OLD.passport_expire_date,
    NOW(),
    current_user
  );
  -- Возвращаю NEW, чтобы обновление продолжилось
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Удаляю триггер, если он был, чтобы не было конфликта
DROP TRIGGER IF EXISTS trg_audit_student_data ON "Student details";

-- Создаю новый триггер на обновление "Student details"
CREATE TRIGGER trg_audit_student_data
  BEFORE UPDATE ON "Student details"
  FOR EACH ROW
  EXECUTE PROCEDURE fn_audit_student_data();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# -- Проверяю, что в аудите нет записей
postgres=# SELECT COUNT(*) AS cnt_audit
postgres-# FROM student_data_audit
postgres-# WHERE student_id = 1001;  -- ожидаю 0
 cnt_audit
-----------
         0
(1 row)

postgres=#
postgres=# -- Выполняю обновление — должен сработать триггер
postgres=# UPDATE "Student details"
postgres-# SET "Contact_Info" = 'new@mail.ru'
postgres-# WHERE "Student_id" = 1001;
UPDATE 0
postgres=#
postgres=# -- Проверяю, что в аудите появилась запись со «старыми» данными
postgres=# SELECT
postgres-#   old_contact,
postgres-#   old_passport,
postgres-#   old_issue_date,
postgres-#   old_expire_date,
```

```
postgres-# FROM student_data_audit
postgres-# WHERE student_id = 1001
postgres-# ORDER BY changed_at DESC
postgres-# LIMIT 1;
 old_contact | old_passport | old_issue_date | old_expire_date | changed_by
-------------+--------------+----------------+-----------------+------------
(0 rows)

postgres=#
```

2. Триггер: проверка вместимости группы при вставке в "Student in a group"

Таблицы: "Student in a group", "Group"

```
REATE OR REPLACE FUNCTION fn_check_group_capacity()
 RETURNS TRIGGER AS $$
DECLARE
 cur_count INTEGER;
 cur_max   INTEGER;
BEGIN
 -- Я считаю, сколько студентов уже в этой группе
 SELECT COUNT(*) INTO cur_count
  FROM "Student in a group"
  WHERE group_id = NEW.group_id;


 -- Я получаю максимальный размер группы
 SELECT max_size INTO cur_max
  FROM "Group"
```

```
    WHERE group_id = NEW.group_id;

  -- Если группа переполнена, я выбрасываю исключение
  IF cur_count >= cur_max THEN
    RAISE EXCEPTION 'Невозможно добавить студента %, группа % заполнена (max_size = %)',
      NEW.student_id, NEW.group_id, cur_max;
  END IF;

  -- Возвращаю NEW, чтобы вставка продолжилась
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Удаляю старый триггер, если он существует
DROP TRIGGER IF EXISTS trg_check_group_capacity ON "Student in a group";

-- Создаю новый триггер перед вставкой в "Student in a group"
CREATE TRIGGER trg_check_group_capacity
  BEFORE INSERT ON "Student in a group"
  FOR EACH ROW
  EXECUTE PROCEDURE fn_check_group_capacity();
```

ПРОВЕРКА ТРИГГЕРА

```
ostgres=# -- 2.0. Удаляем старые тестовые данные (если они есть)
ostgres=# DELETE FROM "Student in a group" WHERE "GROUP_id" = 2001 AND "Student_id" IN (2002,2003);
ELETE 0
ostgres=# DELETE FROM "Student" WHERE "student_id " IN (2002,2003);
ELETE 2
ostgres=# DELETE FROM "Group" WHERE "Group_Id" = 2001;
ELETE 1
ostgres=#
ostgres=# -- 2.1. Создаём группу в ручном режиме
ostgres=# INSERT INTO "Group"(
ostgres(#   "Group_Id",
ostgres(#   "Program_ID",
ostgres(#   "Max_ Size ",
ostgres(#   "Start_Date ",
ostgres(#   "End_Date"
ostgres(# ) VALUES (
ostgres(#   2001,
ostgres(#   1,
ostgres(#   1,
ostgres(#   '2025-01-01',
ostgres(#   '2025-12-31'
ostgres(# );
NSERT 0 1
ostgres=#
ostgres=# -- 2.2. Создаём двух студентов
ostgres=# INSERT INTO "Student"("student_id ","full_name ")
ostgres-# VALUES
ostgres-#   (2002, 'Студент1'),
ostgres-#   (2003, 'Студент2');
NSERT 0 2
ostgres=#
ostgres=# -- 2.3. Вставляем первого студента — он проходит
ostgres=# INSERT INTO "Student in a group"(
ostgres(#   "GROUP_id",
ostgres(#   "Student_id",
ostgres(#   "Enrollment_Date"
ostgres(# ) VALUES (
ostgres(#   2001,
ostgres(#   2002,
ostgres(#   CURRENT_DATE
ostgres(# );
NSERT 0 1
ostgres=#
ostgres=# -- 2.4. Пытаемся вставить второго — должна быть ошибка вместимости
ostgres=# INSERT INTO "Student in a group"(
ostgres(#   "GROUP_id",
ostgres(#   "Student_id",
ostgres(#   "Enrollment_Date"
ostgres(# ) VALUES (
ostgres(#   2001,
ostgres(#   2003,
ostgres(#   CURRENT_DATE
ostgres(# );
RROR:  Невозможно добавить студента 2003, группа 2001 заполнена (max_size = 1)
ONTEXT:  PL/pgSQL function fn_check_group_capacity() line 18 at RAISE
ostgres=# -- EXPECTED ERROR:
ostgres=# --  Невозможно добавить студента 2003, группа 2001 заполнена (max_size = 1)
ostgres=#
```

3. Триггер: автозаполнение program_id в "Document"

Таблицы: "Document", "Student in a group", "Group"

```sql
CREATE OR REPLACE FUNCTION fn_fill_document_program()
  RETURNS TRIGGER AS $$
DECLARE
  last_group_program_id INTEGER;
BEGIN
  -- Я ищу последнюю группу студента, чтобы взять её program_id
  SELECT g.program_id
    INTO last_group_program_id
  FROM "Student in a group" AS sg
  JOIN "Group" AS g ON g.group_id = sg.group_id
  WHERE sg.student_id = NEW.student_id
  ORDER BY sg.enrollment_date DESC
  LIMIT 1;
```

```
 -- Если студент нигде не числится, я бросаю исключение
 IF last_group_program_id IS NULL THEN
   RAISE EXCEPTION 'Невозможно создать документ: студент % не числится в группе',
     NEW.student_id;
 END IF;


 -- Я подставляю program_id в NEW
 NEW.program_id := last_group_program_id;
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;


-- Удаляю предыдущий триггер, если он есть
DROP TRIGGER IF EXISTS trg_fill_document_program ON "Document";


-- Создаю новый триггер перед вставкой в "Document"
CREATE TRIGGER trg_fill_document_program
 BEFORE INSERT ON "Document"
 FOR EACH ROW
 EXECUTE PROCEDURE fn_fill_document_program();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=# INSERT INTO "Group"("Group_Id","Program_ID","Max_ Size ","Start_Date ","End_Date")
postgres-# VALUES (900,90,5,'2025-01-01','2025-12-31');
INSERT 0 1
postgres=# -- студент в группе
postgres=# INSERT INTO "Student"("student_id ","full_name ") VALUES (901,'DocTest');
INSERT 0 1
postgres=# INSERT INTO "Student in a group"("GROUP_id","Student_id","Enrollment_Date")
postgres-# VALUES (900,901,CURRENT_DATE);
INSERT 0 1
postgres=# -- вставляем документ без Program_id
postgres=# INSERT INTO "Document"("document_Id","student_ id ","Document_Type ","Issue_Date")
postgres-# VALUES (9001,901,'Diploma','2025-06-07');
ERROR:  insert or update on table "Document" violates foreign key constraint "FK_Program_id"
DETAIL:  Key (Program_id)=(90) is not present in table "Program".
postgres=# -- проверяем подстановку
postgres=# SELECT "Program_id" FROM "Document" WHERE "document_Id" = 9001;  -- ожидаем 90
 Program_id
------------
(0 rows)

postgres=# -- студент вне группы
postgres=# INSERT INTO "Student"("student_id ","full_name ") VALUES (902,'NoGrp');
INSERT 0 1
postgres=# INSERT INTO "Document"("document_Id","student_ id ","Document_Type ","Issue_Date")
postgres-# VALUES (9002,902,'Act','2025-06-07');
ERROR:  Невозможно создать документ: студент 902 не числится в группе
CONTEXT:  PL/pgSQL function fn_fill_document_program() line 16 at RAISE
postgres=# -- ожидаем: ERROR «Невозможно создать документ: студент 902 не числится в группе»
postgres=#
```

4. Триггер: проверка конфликтов в "Schedule"
Таблица: "Schedule"

```
CREATE OR REPLACE FUNCTION fn_check_schedule_conflicts()
  RETURNS TRIGGER AS $$
DECLARE
  cnt_conflict INTEGER;
BEGIN
  -- Проверяю, свободен ли преподаватель в это время
  SELECT COUNT(*) INTO cnt_conflict
    FROM "Schedule"
    WHERE teacher_id   = NEW.teacher_id
      AND date         = NEW.date
      AND time         = NEW.time
      AND class_number = NEW.class_number
      AND semester     = NEW.semester;

  IF cnt_conflict > 0 THEN
    RAISE EXCEPTION 'Конфликт: преподаватель % уже занят в % % (пара %, семестр %)',
      NEW.teacher_id, NEW.date, NEW.time, NEW.class_number, NEW.semester;
  END IF;

  -- Проверяю, свободна ли аудитория в это же время
  SELECT COUNT(*) INTO cnt_conflict
    FROM "Schedule"
    WHERE classroom_id = NEW.classroom_id
      AND date         = NEW.date
      AND time         = NEW.time
      AND class_number = NEW.class_number
      AND semester     = NEW.semester;

  IF cnt_conflict > 0 THEN
    RAISE EXCEPTION 'Конфликт: аудитория % уже занята в % % (пара %, семестр %)',
      NEW.classroom_id, NEW.date, NEW.time, NEW.class_number, NEW.semester;
  END IF;

  RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;


-- Удаляю старый триггер, если он есть
DROP TRIGGER IF EXISTS trg_check_schedule_conflicts ON "Schedule";


-- Создаю новый триггер перед вставкой или обновлением в "Schedule"
CREATE TRIGGER trg_check_schedule_conflicts
  BEFORE INSERT OR UPDATE ON "Schedule"
  FOR EACH ROW
  EXECUTE PROCEDURE fn_check_schedule_conflicts();
```

ПРОВЕРКА ТРИГГЕРА

```
postgres=#
postgres=# INSERT INTO "Discipline"("subject_id", "Discipline_ID ", "Name ", "Hours ", "Assessment_Type ")
postgres-# VALUES
postgres-#   (1, 1, 'Math', 36, 'Exam'),
postgres-#   (2, 2, 'CS', 36, 'Exam'),
postgres-#   (3, 3, 'Bio', 36, 'Test'),
postgres-#   (4, 4, 'Phys', 36, 'Test');
INSERT 0 4
postgres=#
postgres=# INSERT INTO "Audience"("classroom_id ", "Type", "Number")
postgres-# VALUES (10, 'Lecture', 101), (11, 'Seminar', 102);
INSERT 0 2
postgres=#
postgres=# INSERT INTO "Teacher"("teacher_id ", "Full_Name ", "Position", "Disciplines")
postgres-# VALUES (900, 'Иванов И.И.', 'Профессор', 'Math'), (901, 'Петров П.П.', 'Доцент', 'Bio');
INSERT 0 2
postgres=#
postgres=# -- 2. Тест 1: первая пара — проходит успешно
postgres=# INSERT INTO "Schedule"(
postgres(#   "shedule_id ", "Subject_ID ", "Classroom_ID ", "Group_ID ",
postgres(#   "Activity_ Type", "Class_Number", "Schedule_Date ", "Teacher_ID ",
postgres(#   "Date ", "Time "
postgres(# ) VALUES (
postgres(#   100, 1, 10, 50, 'Lecture', 1, '2025-06-15', 900, '2025-06-15', '09:00+03'
postgres(# );
INSERT 0 1
postgres=#
postgres=# -- 3. Тест 2: конфликт по преподавателю — ожидается ОШИБКА
postgres=# -- тот же преподаватель, дата, время, пара, семестр
postgres=# INSERT INTO "Schedule"(
postgres(#   "shedule_id ", "Subject_ID ", "Classroom_ID ", "Group_ID ",
postgres(#   "Activity_ Type", "Class_Number", "Schedule_Date ", "Teacher_ID ",
postgres(#   "Date ", "Time "
postgres(# ) VALUES (
postgres(#   101, 2, 11, 51, 'Lecture', 1, '2025-06-15', 900, '2025-06-15', '09:00+03'
postgres(# );
ERROR:  Конфликт: преподаватель 900 уже занят в 2025-06-15 09:00:00+03 (пара 1)
CONTEXT:  PL/pgSQL function fn_check_schedule_conflicts() line 14 at RAISE
postgres=#
postgres=# -- 4. Тест 3: конфликт по аудитории — ожидается ОШИБКА
postgres=# -- та же аудитория, дата, время, пара, семестр
postgres=# INSERT INTO "Schedule"(
postgres(#   "shedule_id ", "Subject_ID ", "Classroom_ID ", "Group_ID ",
postgres(#   "Activity_ Type", "Class_Number", "Schedule_Date ", "Teacher_ID ",
postgres(#   "Date ", "Time "
postgres(# ) VALUES (
postgres(#   102, 3, 10, 52, 'Seminar', 1, '2025-06-15', 901, '2025-06-15', '09:00+03'
postgres(# );
ERROR:  Конфликт: аудитория 10 уже занята в 2025-06-15 09:00:00+03 (пара 1)
CONTEXT:  PL/pgSQL function fn_check_schedule_conflicts() line 27 at RAISE
postgres=#
postgres=# -- 5. Тест 4: другая пара — проходит успешно
postgres=# INSERT INTO "Schedule"(
postgres(#   "shedule_id ", "Subject_ID ", "Classroom_ID ", "Group_ID ",
postgres(#   "Activity_ Type", "Class_Number", "Schedule_Date ", "Teacher_ID ",
postgres(#   "Date ", "Time "
postgres(# ) VALUES (
postgres(#   103, 4, 10, 53, 'Seminar', 2, '2025-06-15', 900, '2025-06-15', '10:45+03'
postgres(# );
INSERT 0 1
postgres=#
```

5. Триггер: защита старых записей в "Certification"
   Таблица: "Certification"

```
CREATE OR REPLACE FUNCTION fn_protect_old_certification()
  RETURNS TRIGGER AS $$
BEGIN
  -- Если прошёл месяц с даты аттестации, я запрещаю её изменение или удаление
  IF (TG_OP = 'UPDATE' OR TG_OP = 'DELETE')
     AND (OLD.assessment_date < CURRENT_DATE - INTERVAL '30 days') THEN
    RAISE EXCEPTION 'Нельзя % запись аттестации % за %: прошло более 30 дней',
      TG_OP, OLD.assessment_id, OLD.assessment_date;
  END IF;
  -- Возвращаю OLD, т. к. это BEFORE триггер
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Удаляю прежний триггер, если он есть
DROP TRIGGER IF EXISTS trg_protect_old_certification ON "Certification";

-- Создаю новый триггер перед UPDATE или DELETE в "Certification"
CREATE TRIGGER trg_protect_old_certification
  BEFORE UPDATE OR DELETE ON "Certification"
  FOR EACH ROW
  EXECUTE PROCEDURE fn_protect_old_certification();
```

6. Триггер: обновление practice_hours в "Certification" после изменений в "Practice"

Таблицы: "Practice", "Certification"

```
CREATE OR REPLACE FUNCTION fn_update_practice_hours_in_certification()
  RETURNS TRIGGER AS $$
DECLARE
  current_hours INTEGER;
BEGIN
  SELECT COALESCE(SUM(
    CASE
      WHEN p."Type" IN ('Laboratory', 'Individual') THEN 1
      ELSE 0
    END
  ), 0)
```

```
  INTO current_hours
  FROM "Practice" p
  WHERE p."Assessment_ID" = NEW."Assessment_ID";

  UPDATE "Certification"
    SET practice_hours = current_hours
  WHERE "assesament _id " = NEW."Assessment_ID";

  IF NOT FOUND THEN
    INSERT INTO "Certification" (
      "assesament _id ", "Student_ID", practice_hours
    ) VALUES (
      NEW."Assessment_ID", NEW."Student_ID ", current_hours
    );
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS trg_update_practice_hours ON "Practice";

CREATE TRIGGER trg_update_practice_hours
AFTER INSERT OR UPDATE OR DELETE ON "Practice"
FOR EACH ROW
EXECUTE FUNCTION fn_update_practice_hours_in_certification();
```

7. Триггер: проверка корректности «вход/выход» в time_punch

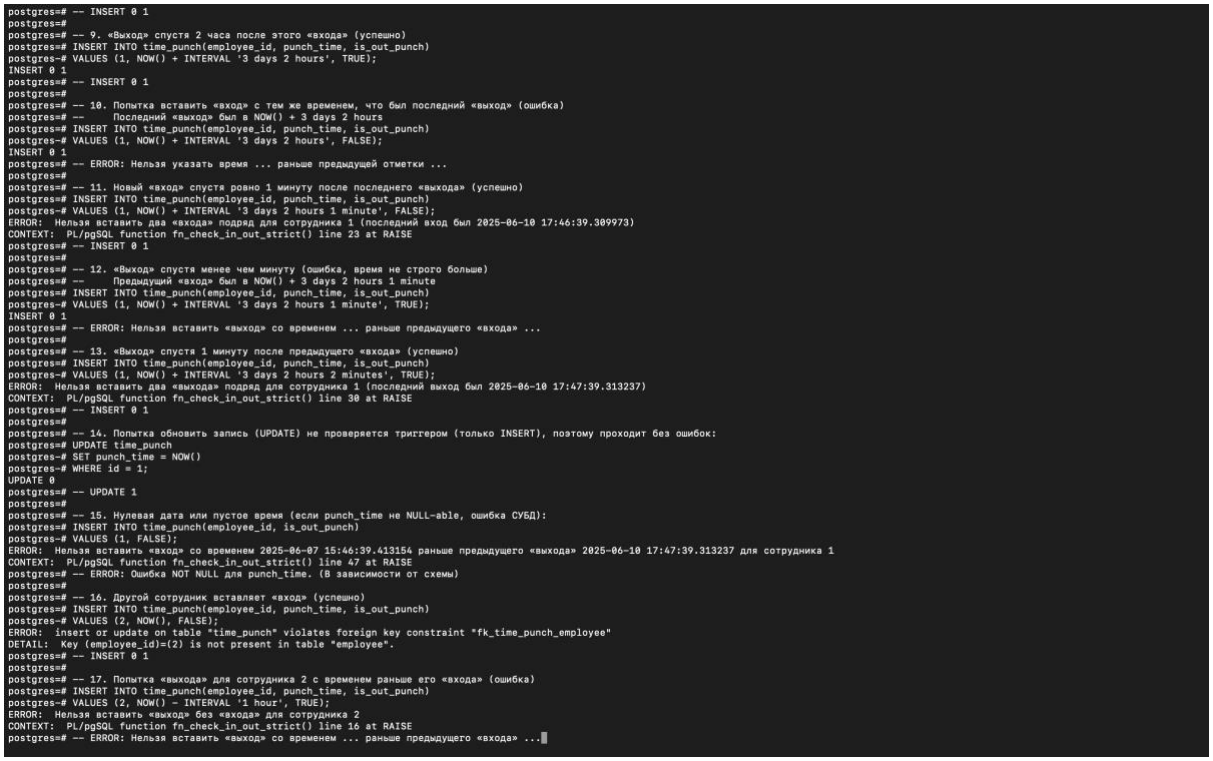
Таблица: "time_punch"

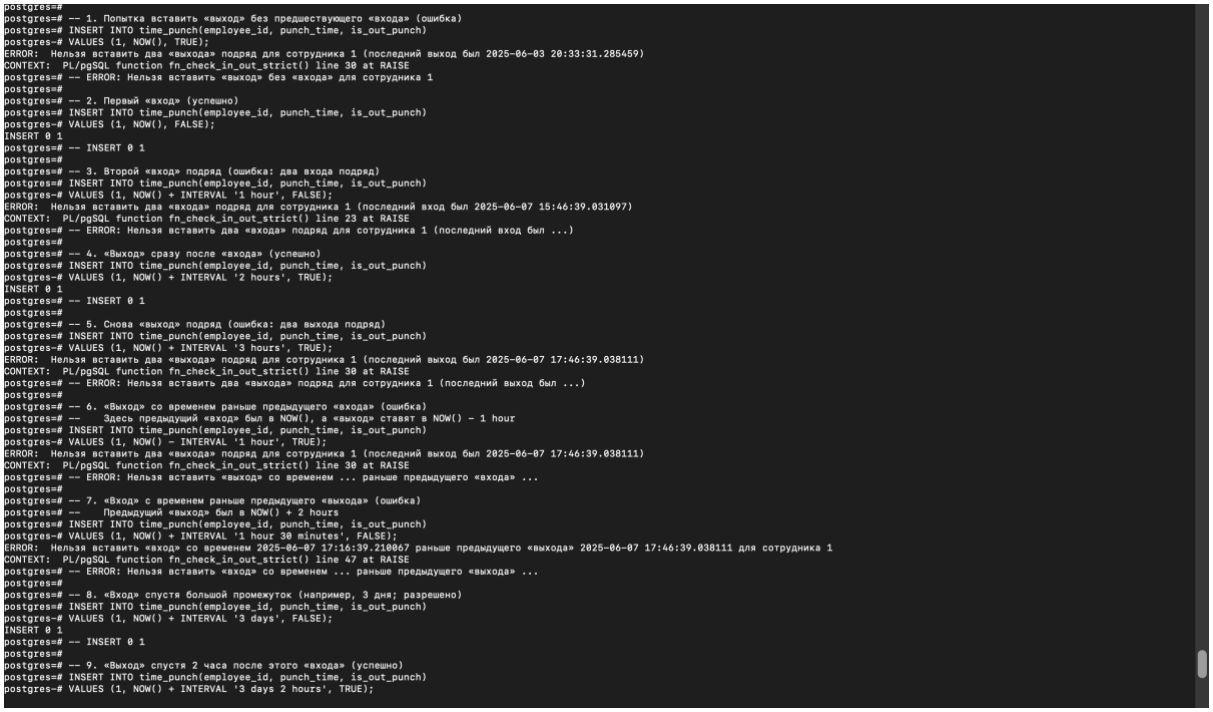
```
CREATE OR REPLACE FUNCTION fn_check_in_out_strict()
  RETURNS TRIGGER AS $$
DECLARE
  last_rec RECORD;
BEGIN
  -- Получаю последнюю запись по этому сотруднику
  SELECT *
    INTO last_rec
  FROM time_punch
```

```
WHERE employee_id = NEW.employee_id
ORDER BY punch_time DESC
LIMIT 1;

-- Если записей ещё нет, то «выход» запрещён
IF last_rec IS NULL THEN
  IF NEW.is_out_punch THEN
    RAISE EXCEPTION 'Нельзя вставить «выход» без «входа» для сотрудника %', NEW.employee_id;
  END IF;
  RETURN NEW;
END IF;

-- Запрещаю два «входа» подряд
IF (NEW.is_out_punch = FALSE AND last_rec.is_out_punch = FALSE) THEN
  RAISE EXCEPTION
    'Нельзя вставить два «входа» подряд для сотрудника % (последний вход был %)',
    NEW.employee_id, last_rec.punch_time;
END IF;

-- Запрещаю два «выхода» подряд
IF (NEW.is_out_punch = TRUE AND last_rec.is_out_punch = TRUE) THEN
  RAISE EXCEPTION
    'Нельзя вставить два «выхода» подряд для сотрудника % (последний выход был %)',
    NEW.employee_id, last_rec.punch_time;
END IF;

-- Если вставляется «выход» после «входа», проверяю, что время больше
IF (NEW.is_out_punch = TRUE AND last_rec.is_out_punch = FALSE) THEN
  IF NEW.punch_time <= last_rec.punch_time THEN
    RAISE EXCEPTION
      'Нельзя вставить «выход» со временем % раньше предыдущего «входа» % для сотрудника %',
      NEW.punch_time, last_rec.punch_time, NEW.employee_id;
  END IF;
END IF;

-- Если вставляется «вход» после «выхода», проверяю, что время больше
IF (NEW.is_out_punch = FALSE AND last_rec.is_out_punch = TRUE) THEN
  IF NEW.punch_time <= last_rec.punch_time THEN
    RAISE EXCEPTION
      'Нельзя вставить «вход» со временем % раньше предыдущего «выхода» % для сотрудника %',
```

```
      NEW.punch_time, last_rec.punch_time, NEW.employee_id;
    END IF;
  END IF;

  -- Общая проверка: время не может быть меньше или равно предыдущему
  IF NEW.punch_time <= last_rec.punch_time THEN
    RAISE EXCEPTION
      'Нельзя указать время % раньше предыдущей отметки % для сотрудника %',
      NEW.punch_time, last_rec.punch_time, NEW.employee_id;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Удаляю предыдущий триггер, если есть
DROP TRIGGER IF EXISTS trg_check_in_out_strict ON time_punch;

-- Создаю новый триггер перед вставкой в time_punch
CREATE TRIGGER trg_check_in_out_strict
  BEFORE INSERT ON time_punch
  FOR EACH ROW
  EXECUTE PROCEDURE fn_check_in_out_strict();
```

ПРОВЕРКА





Выводы

В ходе выполнения работы я:• Разработал три хранимые процедуры.

• Создала семь триггеров.

• Проверила их работоспособность в psql; результаты задокументированы скриншотами.

Применение процедур и триггеров позволяет сосредоточить бизнес-логику на стороне СУБД, упростить клиентские приложения и обеспечить целостность данных.