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: Гайдук Алина Сергеевна
Факультет прикладной информатики
Группа K3241
Направление подготовки 09.03.03 Прикладная информатика
Образовательная программа Мобильные и сетевые технологии 2023
Преподаватель Говорова Марина Михайловна

Санкт-Петербург
2025

Оглавление

Введение 3

1 Схема модели базы данных 4

2 Создание хранимых процедур 5

2 Создание триггеров 10

3 Дополнительное задание 19

Вывод 20

Введение

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

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

Практическое задание (min - 6 баллов, max - 10 баллов, доп. баллы - 3):

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)
2. Создать триггеры для индивидуальной БД согласно варианту: 7 оригинальных триггеров.

1 Схема модели базы данных

Модель представляет организацию сессии внутри университета, включает в себя информацию о преподавателях и их должностях, студентах, стипендии, дисциплинах и их типах, расписании, аттестационной комиссии и прочем. Подробнее ознакомиться с моделью можно на рисунке 1.

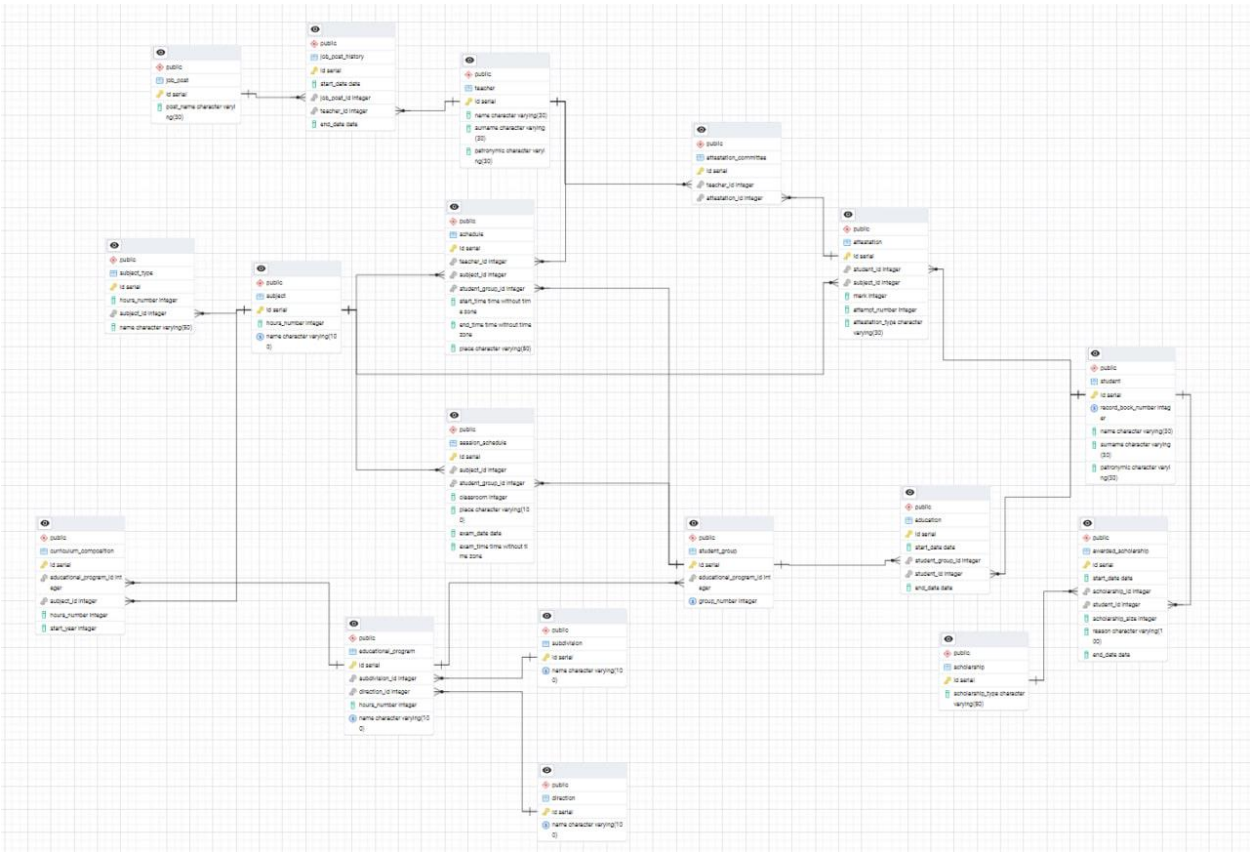

Рисунок 1 – Модель для создания базы данных

2 Создание хранимых процедур

Необходимо создать три хранимые процедуры, указанные в четвертой части второй лабораторной работы. Допустимо использование IN/OUT параметров.

1. Создать хранимую процедуру для повышения стипендии отличникам на 10%.

Я создала хранимую процедуру, которая поднимает стипендию, домножая ее саму на 1.10 (см. рис. 2). Основным условием здесь является то, что минимальным значением всех оценок у студента должно быть 5.

```
session_db=# CREATE OR REPLACE PROCEDURE increase_scholarship_for_honors()
session_db-# LANGUAGE plpgsql
session_db-# AS $$
session_db$# BEGIN
session_db$#   UPDATE awarded_scholarship
session_db$#   SET scholarship_size = scholarship_size * 1.10
session_db$#   WHERE student_id IN (
session_db$#     SELECT student_id
session_db$#     FROM attestation
session_db$#     GROUP BY student_id
session_db$#     HAVING MIN(mark) = 5
session_db$#   );
session_db$# END;
session_db$# $$;
CREATE PROCEDURE
session_db=#
```

Рисунок 2 – Создание хранимой процедуры

Для того, чтобы проверить работу процедуры, я вывела всех отличников (см. рис. 3). Далее я вызвала процедуру с помощью CALL (см. рис. 4). После этого я снова вывела всех отличников с их стипендиями (см. рис. 5).

```
session_db=# SELECT s.id, st.name, st.surname, s.scholarship_size
session_db-# FROM awarded_scholarship s
session_db-# JOIN student st ON st.id = s.student_id
session_db-# WHERE s.student_id IN (
session_db(#   SELECT student_id
session_db(#   FROM attestation
session_db(#   GROUP BY student_id
session_db(#   HAVING MIN(mark) = 5
session_db(# );
 id |   name   | surname | scholarship_size
----+----------+---------+------------------
  1 | Алексей  | Сидоров |             5000
  3 | Олег     | Новиков |             4500
  4 | Андрей   | Андреев |             5500
  8 | Владимир | Лебедев |             2500
 12 | Максим   | Павлов  |             4600
 15 | Илья     | Зайцев  |             4200
(6 строк)
```

Рисунок 3 – Все отличники в БД

```
session_db=# CALL increase_scholarship_for_honors();
CALL
```

Рисунок 4 – Вызов процедуры

```
session_db=# SELECT s.id, st.name, st.surname, s.scholarship_size
session_db-# FROM awarded_scholarship s
session_db-# JOIN student st ON st.id = s.student_id
session_db-# WHERE s.student_id IN (
session_db(#   SELECT student_id
session_db(#   FROM attestation
session_db(#   GROUP BY student_id
session_db(#   HAVING MIN(mark) = 5
session_db(# );
 id |   name   | surname | scholarship_size
----+----------+---------+------------------
  1 | Алексей  | Сидоров |             5500
  3 | Олег     | Новиков |             4950
  4 | Андрей   | Андреев |             6050
  8 | Владимир | Лебедев |             2750
 12 | Максим   | Павлов  |             5060
 15 | Илья     | Зайцев  |             4620
(6 строк)
```

Рисунок 6 – Обновленные стипендии отличников

2. Создать хранимую процедуру для перевода студентов на следующий курс.

Перед созданием процедуры мне пришлось добавить новый столбец в таблицу – current_year (см. рис. 7). Созданная мной процедура просто добавляет к этому столбцу единицу, если студент еще не окончил обучение в вузе (см. рис. 8).

```
session_db=# ALTER TABLE education ADD COLUMN current_year INTEGER DEFAULT 1;
ALTER TABLE
```

Рисунок 7 – Добавление нового столбца в таблицу

```
session_db=# CREATE OR REPLACE PROCEDURE promote_students()
session_db-# LANGUAGE plpgsql
session_db-# AS $$
session_db$# BEGIN
session_db$#    UPDATE education
session_db$#    SET current_year = current_year + 1
session_db$#    WHERE end_date > CURRENT_DATE;
session_db$# END;
session_db$# $$;
CREATE PROCEDURE
session_db=#
```

Рисунок 8 – Создание процедуры

Для проверки вывожу всех студентов из таблицы education (см. рис. 9), вызываю процедуру (см. рис. 10) и снова вывожу всех студентов. Их курс обучения обновился (см. рис. 11).

```
session_db=# SELECT s.id, s.name, s.surname, e.current_year
session_db-# FROM education e
session_db-# JOIN student s ON s.id = e.student_id;
 id |   name    |  surname  | current_year
----+-----------+-----------+--------------
  1 | Алексей   | Сидоров   |            1
  2 | Мария     | Кузнецова |            1
  3 | Олег      | Новиков   |            1
  4 | Андрей    | Андреев   |            1
  5 | Дмитрий   | Смирнов   |            1
  6 | Сергей    | Кузнецов  |            1
  7 | Николай   | Попов     |            1
  8 | Владимир  | Лебедев   |            1
  9 | Михаил    | Новиков   |            1
 10 | Евгений   | Морозов   |            1
 11 | Александр | Соболев   |            1
 12 | Максим    | Павлов    |            1
 13 | Артем     | Козлов    |            1
 14 | Роман     | Васильев  |            1
 15 | Илья      | Зайцев    |            1
(15 строк)
```

Рисунок 9 – Просмотр всех студентов

```
session_db=# CALL promote_students();
CALL
```

Рисунок 10 – Вызов процедуры

```
session_db=# SELECT s.id, s.name, s.surname, e.current_year
session_db-# FROM education e
session_db-# JOIN student s ON s.id = e.student_id;
 id |    name    |  surname   | current_year
----+------------+------------+--------------
  1 | Алексей    | Сидоров    |            2
  2 | Мария      | Кузнецова  |            2
  3 | Олег       | Новиков    |            2
  4 | Андрей     | Андреев    |            2
  5 | Дмитрий    | Смирнов    |            2
  6 | Сергей     | Кузнецов   |            2
  7 | Николай    | Попов      |            2
  8 | Владимир   | Лебедев    |            2
  9 | Михаил     | Новиков    |            2
 10 | Евгений    | Морозов    |            2
 11 | Александр  | Соболев    |            2
 12 | Максим     | Павлов     |            2
 13 | Артем      | Козлов     |            2
 14 | Роман      | Васильев   |            2
 15 | Илья       | Зайцев     |            2
(15 строк)
```

Рисунок 11 – Обновленный год обучения студентов

3. Создать хранимую процедуру для изменения оценки при успешной пересдаче экзамена.

Созданная мной процедура изображена на рисунке 12. В ней я обновляю оценку, а также добавляю единицу к текущему количеству попыток.

```
session_db=# CREATE OR REPLACE PROCEDURE update_mark_on_retake(
session_db(#   IN p_student_id INT,
session_db(#   IN p_subject_id INT,
session_db(#   IN p_new_mark INT
session_db(# )
session_db-# LANGUAGE plpgsql
session_db-# AS $$
session_db$# BEGIN
session_db$#   UPDATE attestation
session_db$#   SET mark = p_new_mark,
session_db$#       attempt_number = attempt_number + 1
session_db$#   WHERE student_id = p_student_id
session_db$#     AND subject_id = p_subject_id
session_db$#     AND mark < p_new_mark;
session_db$# END;
session_db$# $$;
CREATE PROCEDURE
```

Рисунок 12 – Создание процедуры

Для проверки вывожу одну неудачную попытку аттестации, вызываю процедуру и снова вывожу аттестацию того же студента по тому же предмету (см. рис. 13–15).

```
session_db=# SELECT * FROM attestation  WHERE student_id = 7 AND subject_id = 6;
 id | student_id | subject_id | mark | attempt_number | attestation_type
----+------------+------------+------+----------------+------------------
  7 |          7 |          6 |    2 |              1 | Зачёт
(1 строка)
```

Рисунок 13 – Просмотр неудачной аттестации

```
session_db=# CALL update_mark_on_retake(7, 6, 5);
CALL
```

Рисунок 14 – Вызов процедуры

```
session_db=# SELECT * FROM attestation  WHERE student_id = 7 AND subject_id = 6;
 id | student_id | subject_id | mark | attempt_number | attestation_type
----+------------+------------+------+----------------+------------------
  7 |          7 |          6 |    5 |              2 | Зачёт
(1 строка)
```

Рисунок 15 – Обновленная оценка и номер попытки в аттестации

2 Создание триггеров

Необходимо придумать и создать 7 триггеров для базы данных. Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.

1. Если студент получает оценку 2, то активная стипендия автоматически обнуляется.

Я создала триггер, который обнуляет стипендию, если в базе данных у студента появляется оценка 2 (см. рис. 16).

```
session_db=# CREATE OR REPLACE FUNCTION suspend_scholarship_on_fail()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF NEW.mark = 2 THEN
session_db$#     UPDATE awarded_scholarship
session_db$#     SET scholarship_size = 0
session_db$#     WHERE student_id = NEW.student_id
session_db$#       AND CURRENT_DATE BETWEEN start_date AND end_date;
session_db$#   END IF;
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_suspend_scholarship_on_fail
session_db-# AFTER INSERT OR UPDATE ON attestation
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION suspend_scholarship_on_fail();
CREATE TRIGGER
session_db=#
```

Рисунок 16 – Создание триггера

Далее работу триггера нужно проверить, я специально добавила некоторым студентам стипендию, которую он получает прямо сейчас (не в прошлом). После этого я вывела студентов с актуальной стипендией. Одному из студентов я проставила оценку 2 за экзамен. Далее я вывела все стипендии, которые получал/получает студент в вузе. Сумма стипендии обновилась, и теперь студент получает 0 (см. рис. 17–19).

```
session_db=# SELECT student_id, scholarship_size, start_date, end_date
session_db-# FROM awarded_scholarship
session_db-# WHERE CURRENT_DATE BETWEEN start_date AND end_date;
 student_id | scholarship_size | start_date |  end_date
------------+------------------+------------+------------
          1 |             5000 | 2025-04-14 | 2025-06-14
          2 |             4500 | 2025-05-04 | 2025-05-24
          3 |             4000 | 2025-04-29 | 2025-05-19
(3 строки)
```

Рисунок 17 – Просмотр студентов с актуальной стипендией

```
session_db=# INSERT INTO attestation (student_id, subject_id, mark, attempt_number, attestation_type)
session_db-# VALUES (1, 1, 2, 1, 'Экзамен');
INSERT 0 1
```

Рисунок 18 – Добавление студенту аттестации с оценкой 2

```
session_db=# SELECT * FROM awarded_scholarship WHERE student_id = 1;
 id | start_date | scholarship_id | student_id | scholarship_size |       reason        |  end_date
----+------------+----------------+------------+------------------+---------------------+------------
  1 | 2024-01-01 |              1 |          1 |             5500 | Высокая успеваемость | 2024-06-30
 21 | 2025-04-14 |              1 |          1 |                0 | Тестовая стипендия   | 2025-06-14
(2 строки)
```

Результат 19 – Обновленная стипендия студента

2. Триггер, устраняющий конфликт в расписании преподавателя. Один преподаватель не может быть в двух местах в одно и то же время.

Я создала триггер, который представлен на рисунке 20. В нем сверяется время начала и окончания, а также группа студентов, в которой преподаватель ведет.

```
session_db=# CREATE OR REPLACE FUNCTION prevent_teacher_overlap()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF EXISTS (
session_db$#     SELECT 1 FROM schedule
session_db$#     WHERE teacher_id = NEW.teacher_id
session_db$#       AND start_time < NEW.end_time
session_db$#       AND end_time > NEW.start_time
session_db$#       AND student_group_id <> NEW.student_group_id
session_db$#   ) THEN
session_db$#     RAISE EXCEPTION 'Преподаватель уже занят в это время';
session_db$#   END IF;
session_db$#
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_prevent_teacher_overlap
session_db-# BEFORE INSERT OR UPDATE ON schedule
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION prevent_teacher_overlap();
CREATE TRIGGER
```

Рисунок 20 – Создание триггера

Для проверки работы этого триггера я нарочно создам пересечение в расписании преподавателя. Для начала я просмотрела расписание преподавателя с id=1, а затем создала ему пару с другой группой в это же время. Триггер сработал, и появилась ошибка (см. рис. 21).

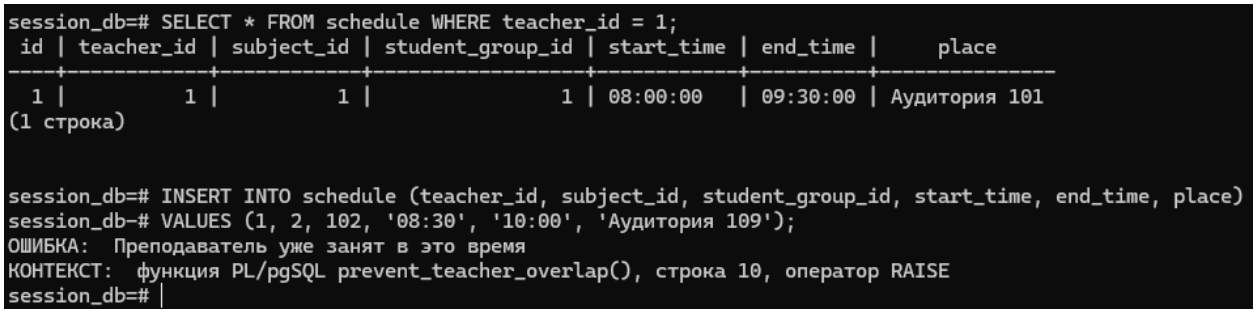

Рисунок 21 – Проверка работы триггера

3. Триггер для автоматической выдачи новой стипендии при отличной успеваемости. Если после новой оценки у студента все оценки ≥ 4.5, ему нужно выдать академическую стипендию.

Этот триггер вызывает функцию, которая вычисляет средний балл, и если балл оказывается больше 4.5, то идет проверка: есть ли у студента активная стипендия? Если есть, то стипендия завершается в этот же день, а новая академическая стипендия начинает свое действие с завтрашнего дня. Процесс создания триггера показан на рисунке 22.

Чтобы проверить работу триггера, я проверяю активные стипендии студента, добавляю ему новые оценки. После этого срабатывает триггер, и студенту назначается академическая стипендия за высокий средний балл (см. рис 23–24).

```
session_db=# CREATE OR REPLACE FUNCTION grant_new_scholarship_if_eligible()
session_db-# RETURNS TRIGGER AS $$
session_db$# DECLARE
session_db$#   avg_mark NUMERIC;
session_db$#   has_active_scholarship BOOLEAN;
session_db$# BEGIN
session_db$#   -- 1. Вычисляем средний балл (включая новую оценку)
session_db$#   SELECT AVG(mark)
session_db$#   INTO avg_mark
session_db$#   FROM (
session_db$#     SELECT mark FROM attestation WHERE student_id = NEW.student_id
session_db$#     UNION ALL
session_db$#     SELECT NEW.mark
session_db$#   ) AS all_marks;
session_db$#
session_db$#   -- 2. Если балл ≥ 4.5 — продолжаем
session_db$#   IF avg_mark >= 4.5 THEN
session_db$#
session_db$#     -- 3. Есть ли активная академическая стипендия?
session_db$#     SELECT EXISTS (
session_db$#       SELECT 1 FROM awarded_scholarship
session_db$#       WHERE student_id = NEW.student_id
session_db$#         AND scholarship_id = 1
session_db$#         AND CURRENT_DATE BETWEEN start_date AND end_date
session_db$#     ) INTO has_active_scholarship;
session_db$#
session_db$#     -- 4. Завершаем текущую академическую стипендию (на сегодня)
session_db$#     IF has_active_scholarship THEN
session_db$#       UPDATE awarded_scholarship
session_db$#       SET end_date = CURRENT_DATE
session_db$#       WHERE student_id = NEW.student_id
session_db$#         AND scholarship_id = 1
session_db$#         AND CURRENT_DATE BETWEEN start_date AND end_date;
session_db$#     END IF;
session_db$#
session_db$#     -- 5. Вставляем новую стипендию с завтрашнего дня
session_db$#     INSERT INTO awarded_scholarship (
session_db$#       student_id, scholarship_id, start_date, end_date, scholarship_size, reason
session_db$#     )
session_db$#     VALUES (
session_db$#       NEW.student_id, 1,
session_db$#       CURRENT_DATE + INTERVAL '1 day',
session_db$#       CURRENT_DATE + INTERVAL '6 months' + INTERVAL '1 day',
session_db$#       5000,
session_db$#       'Переназначение академической стипендии'
session_db$#     );
session_db$#
session_db$#   END IF;
session_db$#
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_grant_new_scholarship
session_db-# AFTER INSERT OR UPDATE ON attestation
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION grant_new_scholarship_if_eligible();
```

Рисунок 22 – Создание триггера

```
session_db=# SELECT * FROM awarded_scholarship WHERE student_id = 4;
 id | start_date | scholarship_id | student_id | scholarship_size |        reason          |  end_date
----+------------+----------------+------------+------------------+------------------------+------------
  4 | 2024-09-01 |              4 |          4 |             6050 | Президентская стипендия | 2025-02-28
(1 строка)
```

Рисунок 23 – Проверка активных стипендий студента

```
session_db=# INSERT INTO attestation (student_id, subject_id, mark, attempt_number, attestation_type)
session_db-# VALUES
session_db-# (4, 1, 5, 1, 'Экзамен'),  (4, 3, 4, 1, 'Экзамен'), (4, 4, 5, 1, 'Экзамен'), (4, 2, 5, 1, 'Экзамен'), (4, 10, 5, 1, 'Экзамен');
INSERT 0 5
session_db=# SELECT * FROM awarded_scholarship WHERE student_id = 4;
 id | start_date | scholarship_id | student_id | scholarship_size |                reason                |  end_date
----+------------+----------------+------------+------------------+--------------------------------------+------------
  4 | 2024-09-01 |              4 |          4 |             6050 | Президентская стипендия              | 2025-02-28
 29 | 2025-05-15 |              1 |          4 |             5000 | Переназначение академической стипендии | 2025-11-15
```

Рисунок 24 – Обновленная стипендия студента

4. Триггер для автоотчисления при ≥ 3 двоек

Я создала триггер, который считает количество двоек после добавления оценки, и обновляет end_date в таблице education, если студент получил три двойки за экзамены (см. рис. 25).

```
session_db=# CREATE OR REPLACE FUNCTION auto_expel()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF (
session_db$#     SELECT COUNT(*) FROM attestation
session_db$#     WHERE student_id = NEW.student_id AND mark = 2
session_db$#   ) >= 3 THEN
session_db$#     UPDATE education
session_db$#     SET end_date = CURRENT_DATE
session_db$#     WHERE student_id = NEW.student_id AND end_date > CURRENT_DATE;
session_db$#   END IF;
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_auto_expel_failing_student
session_db-# AFTER INSERT OR UPDATE ON attestation
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION auto_expel();
CREATE TRIGGER
session_db=#
```

Рисунок 25 – Создание триггера

Для проверки данного триггера я вывела всех студентов, и одному из них добавила необходимое количество двоек (три). После чего просмотрела столбцы таблицы education для этого студента. Дата окончания обучения обновилась, студент отчислен (см. рис. 26).

```
session_db=# SELECT * FROM education
session_db-# WHERE end_date > CURRENT_DATE;
 id | start_date | student_group_id | student_id |  end_date  | current_year
----+------------+------------------+------------+------------+--------------
  1 | 2021-09-01 |                1 |          1 | 2025-06-30 |            2
  2 | 2021-09-01 |                2 |          2 | 2025-06-30 |            2
  3 | 2021-09-01 |                3 |          3 | 2025-06-30 |            2
  4 | 2021-09-01 |                4 |          4 | 2025-06-30 |            2
  5 | 2021-09-01 |                5 |          5 | 2025-06-30 |            2
  6 | 2021-09-01 |                6 |          6 | 2025-06-30 |            2
  7 | 2021-09-01 |                7 |          7 | 2025-06-30 |            2
  8 | 2021-09-01 |                8 |          8 | 2025-06-30 |            2
  9 | 2021-09-01 |                9 |          9 | 2025-06-30 |            2
 10 | 2021-09-01 |               10 |         10 | 2025-06-30 |            2
 11 | 2021-09-01 |               11 |         11 | 2025-06-30 |            2
 12 | 2021-09-01 |               12 |         12 | 2025-06-30 |            2
 13 | 2021-09-01 |               13 |         13 | 2025-06-30 |            2
 14 | 2021-09-01 |               14 |         14 | 2025-06-30 |            2
 15 | 2021-09-01 |               15 |         15 | 2025-06-30 |            2
(15 строк)


session_db=# SELECT COUNT(*) FROM attestation
session_db-# WHERE student_id = 5 AND mark = 2;
 count
-------
     0
(1 строка)


session_db=# INSERT INTO attestation (student_id, subject_id, mark, attempt_number, attestation_type)
session_db-# VALUES
session_db-#   (5, 1, 2, 1, 'Экзамен'),
session_db-#   (5, 10, 2, 1, 'Экзамен'),
session_db-#   (5, 12, 2, 1, 'Экзамен');
INSERT 0 3
session_db=# SELECT * FROM education
session_db-# WHERE student_id = 5;
 id | start_date | student_group_id | student_id |  end_date  | current_year
----+------------+------------------+------------+------------+--------------
  5 | 2021-09-01 |                5 |          5 | 2025-05-14 |            2
(1 строка)


session_db=#
```

Рисунок 26 – Отчисление студента

После выполнения этого задания я заметила, что курс обучения не сходится с годом начала учебы, поэтому я обновила таблицу education так, чтобы current_year учитывал start_date (см. рис. 27).

```
session_db=# UPDATE education
session_db-# SET current_year = EXTRACT(YEAR FROM AGE(CURRENT_DATE, start_date)) + 1
session_db-# WHERE end_date > CURRENT_DATE;
UPDATE 14
```

Рисунок 27 – Исправление current_year

5. Триггер на запрет пересечения экзаменов по времени в одной группе.

Я создала триггер, чтобы в расписании сессии не было двух экзаменов в одно и то же время у одной группы (см. рис. 28).

```
session_db=# CREATE OR REPLACE FUNCTION prevent_exam_overlap()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF EXISTS (
session_db$#     SELECT 1 FROM session_schedule
session_db$#     WHERE student_group_id = NEW.student_group_id
session_db$#       AND exam_date = NEW.exam_date
session_db$#       AND exam_time = NEW.exam_time
session_db$#   ) THEN
session_db$#     RAISE EXCEPTION 'У этой группы уже есть экзамен в это время';
session_db$#   END IF;
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_prevent_exam_overlap
session_db-# BEFORE INSERT ON session_schedule
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION prevent_exam_overlap();
CREATE TRIGGER
session_db=#
```

Рисунок 28 – Создание триггера

Чтобы проверить работу триггера, я просмотрела таблицу расписания сессий и попыталась добавить запись об аттестации, пересекающейся по номеру группы и времени с существующей. Триггер сработал, появилась ошибка (см. рис. 29).

```
session_db=# SELECT * FROM session_schedule;
 id | subject_id | student_group_id | classroom |       place        | exam_date  | exam_time
----+------------+------------------+-----------+--------------------+------------+-----------
  1 |          1 |                1 |       101 | Корпус A, ауд. 101 | 2025-01-15 | 10:00:00
  2 |          2 |                2 |       202 | Корпус B, ауд. 202 | 2025-01-16 | 11:00:00
  3 |          3 |                3 |       103 | Корпус A, ауд. 103 | 2025-01-17 | 12:00:00
  4 |          4 |                4 |       204 | Корпус B, ауд. 204 | 2025-01-18 | 13:00:00
  5 |          5 |                5 |       105 | Корпус A, ауд. 105 | 2025-01-19 | 14:00:00
  6 |          6 |                6 |       206 | Корпус B, ауд. 206 | 2025-01-20 | 15:00:00
  7 |          7 |                7 |       107 | Корпус A, ауд. 107 | 2025-01-21 | 10:00:00
  8 |          8 |                8 |       208 | Корпус B, ауд. 208 | 2025-01-22 | 11:00:00
  9 |          9 |                9 |       109 | Корпус A, ауд. 109 | 2025-01-23 | 12:00:00
 10 |         10 |               10 |       210 | Корпус B, ауд. 210 | 2025-01-24 | 13:00:00
 11 |         11 |               11 |       111 | Корпус A, ауд. 111 | 2025-01-25 | 14:00:00
 12 |         12 |               12 |       212 | Корпус B, ауд. 212 | 2025-01-26 | 15:00:00
 13 |         13 |               13 |       113 | Корпус A, ауд. 113 | 2025-01-27 | 10:00:00
 14 |         14 |               14 |       214 | Корпус B, ауд. 214 | 2025-01-28 | 11:00:00
 15 |         15 |               15 |       115 | Корпус A, ауд. 115 | 2025-01-29 | 12:00:00
(15 строк)


session_db=# INSERT INTO session_schedule (subject_id, student_group_id, classroom, place, exam_date, exam_time)
session_db-# VALUES (999, 1, 121, 'Корпус A, ауд. 121', '2025-01-15', '10:00');
ОШИБКА:  У этой группы уже есть экзамен в это время
КОНТЕКСТ:  функция PL/pgSQL prevent_exam_overlap(), строка 9, оператор RAISE
session_db=#
```

Рисунок 29 – Успешная работа триггера

6. Триггер для блокировки второго зачёта по одному предмету: если студент уже получил зачёт по предмету, второй раз сдавать нельзя.

Я создала триггер, который запрещает вставку в attestation, если студент уже сдал зачет по тому же предмету с оценкой 3 и выше (см. рис. 30).

```
session_db=# CREATE OR REPLACE FUNCTION prevent_duplicate_pass()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF EXISTS (
session_db$#     SELECT 1 FROM attestation
session_db$#     WHERE student_id = NEW.student_id
session_db$#       AND subject_id = NEW.subject_id
session_db$#       AND attestation_type = 'Зачёт'
session_db$#       AND mark >= 3
session_db$#   ) THEN
session_db$#     RAISE EXCEPTION 'Студент уже имеет зачёт по этому предмету';
session_db$#   END IF;
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_prevent_duplicate_pass
session_db-# BEFORE INSERT ON attestation
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION prevent_duplicate_pass();
CREATE TRIGGER
session_db=#
```

Рисунок 30 – Создание триггера

Для проверки работы триггера я вывела таблицу студентов, уже сдавших зачеты, и попробовала добавить одному из них запись об еще одной удачной попытки аттестации по зачету, вышла ошибка (см. рис. 31).

```
session_db=# SELECT * FROM attestation
session_db-# WHERE attestation_type = 'Зачёт'
session_db-#   AND mark >= 3;
 id | student_id | subject_id | mark | attempt_number | attestation_type
----+------------+------------+------+----------------+-----------------
  5 |          5 |          4 |    4 |              1 | Зачёт
  9 |          9 |          8 |    4 |              2 | Зачёт
 13 |         13 |         12 |    4 |              1 | Зачёт
 15 |         15 |         14 |    5 |              1 | Зачёт
  3 |          3 |          2 |    5 |              3 | Зачёт
  7 |          7 |          6 |    5 |              2 | Зачёт
(6 строк)


session_db=# INSERT INTO attestation (student_id, subject_id, mark, attempt_number, attestation_type)
session_db-# VALUES (9, 8, 4, 2, 'Зачёт');
ОШИБКА:  Студент уже имеет зачёт по этому предмету
КОНТЕКСТ:  функция PL/pgSQL prevent_duplicate_pass(), строка 10, оператор RAISE
session_db=#
```

Рисунок 31 – Успешная работа триггера

7. Триггер для запрета оценки, если студент не зачислен (нет активной записи в education)

Я создала триггер, который запрещает вставку записи в attestation, если студент отчислен/не зачислен (см. рис. 32).

```
session_db=# CREATE OR REPLACE FUNCTION prevent_attestation_without_education()
session_db-# RETURNS TRIGGER AS $$
session_db$# BEGIN
session_db$#   IF NOT EXISTS (
session_db$#     SELECT 1 FROM education
session_db$#     WHERE student_id = NEW.student_id
session_db$#       AND CURRENT_DATE BETWEEN start_date AND end_date
session_db$#   ) THEN
session_db$#     RAISE EXCEPTION 'Студент не зачислен – нельзя выставить оценку';
session_db$#   END IF;
session_db$#   RETURN NEW;
session_db$# END;
session_db$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
session_db=#
session_db=# CREATE TRIGGER trg_prevent_attestation_without_education
session_db-# BEFORE INSERT ON attestation
session_db-# FOR EACH ROW
session_db-# EXECUTE FUNCTION prevent_attestation_without_education();
CREATE TRIGGER
session_db=#
```

Рисунок 31 – Создание триггера

Для проверки я вывела список студентов, которые либо были отчислены, либо завершили свое обучение. Попался студент, которого ранее я отчислила в ходе проверки триггера на автоотчисление за три оценки 2. Я попробовала добавить ему запись о новой аттестации, появилась ошибка (см. рис. 33).

```
session_db=# SELECT student_id FROM education
session_db-# WHERE end_date < CURRENT_DATE
session_db-# LIMIT 1;
 student_id
------------
          5
(1 строка)


session_db=# INSERT INTO attestation (student_id, subject_id, mark, attempt_number, attestation_type)
session_db-# VALUES (5, 1, 5, 1, 'Экзамен');
ОШИБКА:  Студент не зачислен – нельзя выставить оценку
КОНТЕКСТ:  функция PL/pgSQL prevent_attestation_without_education(), строка 8, оператор RAISE
session_db=#
```

Рисунок 33 – Удачная работа триггера

3 Дополнительное задание

Я исправила триггер, который был дан во время выполнения практической работы (см. рисунок 34).

```
emp_time=# create or replace function fn_check_time_punch() returns trigger as $$
emp_time$# declare
emp_time$#   last_out boolean;
emp_time$#   last_time timestamp;
emp_time$# begin
emp_time$#   select is_out_punch, punch_time
emp_time$#   into last_out, last_time
emp_time$#   from time_punch
emp_time$#   where employee_id = new.employee_id
emp_time$#   order by punch_time desc
emp_time$#   limit 1;
emp_time$#
emp_time$#   if last_time is not null then
emp_time$#     if last_out = new.is_out_punch then
emp_time$#       raise exception 'Действие не может повторяться подряд';
emp_time$#     end if;
emp_time$#     if new.punch_time <= last_time then
emp_time$#       raise exception 'Время не может идти назад';
emp_time$#     end if;
emp_time$#   end if;
emp_time$#
emp_time$#   return new;
emp_time$# end;
emp_time$# $$ language plpgsql;
CREATE FUNCTION
emp_time=# create or replace trigger check_time_punch before insert on time_punch
emp_time-# for each row execute procedure fn_check_time_punch();
CREATE TRIGGER
emp_time=#
```

Рисунок 34 – Исправленный триггер

В триггере добавлена проверка, чтобы действия вход/выход не повторялись подряд, и чтобы новое время punch не было меньше или равно предыдущему. Также добавлен LIMIT 1 для безопасного выбора последней записи сотрудника, добавлены сообщения, которые будут выводиться в качестве ошибки.

Вывод

В ходе выполнения лабораторной работы были получены практические навыки создания и применения процедур, функций и триггеров в СУБД PostgreSQL. Разработаны пользовательские процедуры и триггеры, обеспечивающие логическую целостность данных при вводе информации.

Была проведена модификация триггера контроля входа/выхода сотрудника. Устранены логические ошибки в работе триггера, что позволило повысить достоверность учёта рабочего времени.