Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по теме:

*«Процедуры, функции, триггеры в PostgreSQL»* по дисциплине: Проектирование и реализация баз данных

Специальность:
09.02.03 Мобильные и сетевые технологии

Проверила:
Говорова М.М.
Дата: «..» … 2023 г.
Оценка _____

Выполнил: студент
группы К3239 Прокопец
Семен

Санкт-Петербург 2023

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

**Практическое задание:**

**Вариант 1**

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).

2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

**Вариант 2**

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2. 
   2.1. 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.
   2.2. Создать авторский триггер по варианту индивидуального задания.

**Предметная область** – Вариант 16. БД "Спортивный клуб"

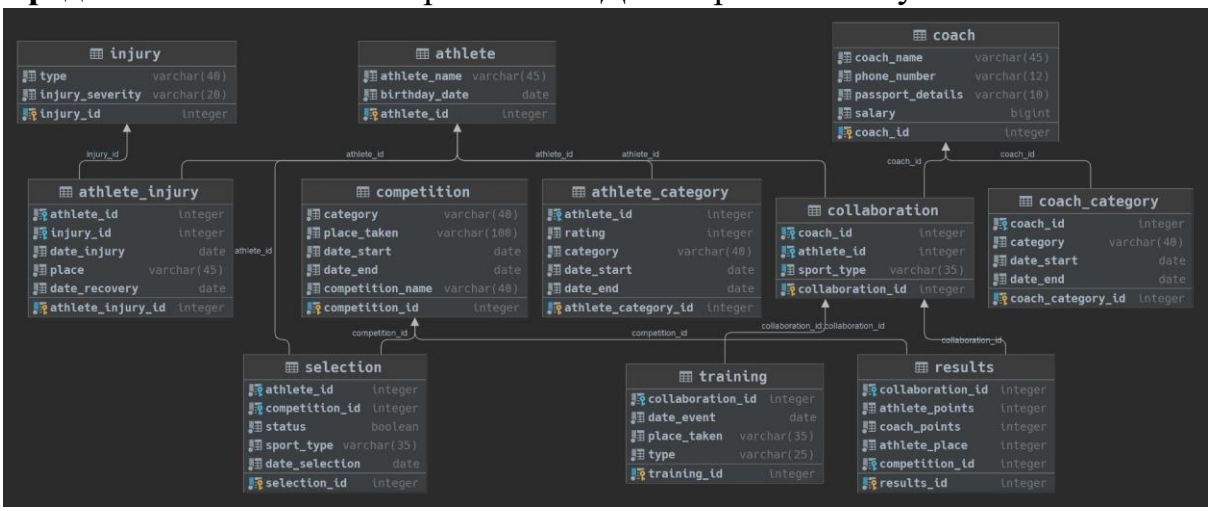


Рисунок 1 – ERD базы данных

**Выполнение работы:**

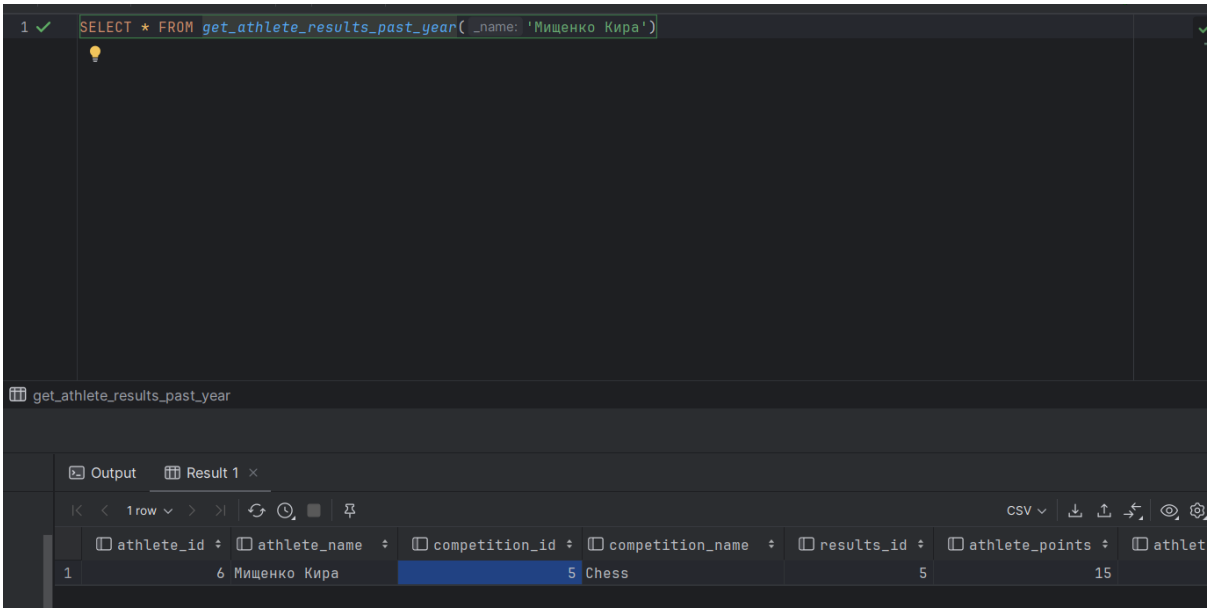
1. **Для вывода данных о результатах заданного спортсмена за прошедший год.**

Создание процедуры:

```
CREATE OR REPLACE FUNCTION get_athlete_results_past_year(_name TEXT)
RETURNS TABLE( athlete_id INT, athlete_name VARCHAR, competition_id INT,
competition_name VARCHAR, results_id INT, athlete_points INT, athlete_place
INT
)
LANGUAGE plpgsql
AS $$
BEGIN
RETURN QUERY
SELECT
    a.athlete_id,
    a.athlete_name,
    com.competition_id,
    com.competition_name,
    r.results_id,
    r.athlete_points,
    r.athlete_place
FROM athlete a
    JOIN collaboration col USING(athlete_id)
    JOIN results r USING(collaboration_id)
    JOIN competition com USING(competition_id)
WHERE a.athlete_name = _name AND com.date_end >= NOW() - INTERVAL '1 year';
END;
$$;
```

Алгоритм: Функция принимает в качестве аргумента имя спортсмена и возвращает таблицу сформированную из данных возвращаемых запросом SELECT в теле функции. Отбираются только те данные которые принадлежат соответствующему спортсмену и где дата окончания соревнования в пределах прошедшего года.

Проверка функции:




**2. Для вывода данных о соревнованиях, проводимых в первом квартале текущего года.**

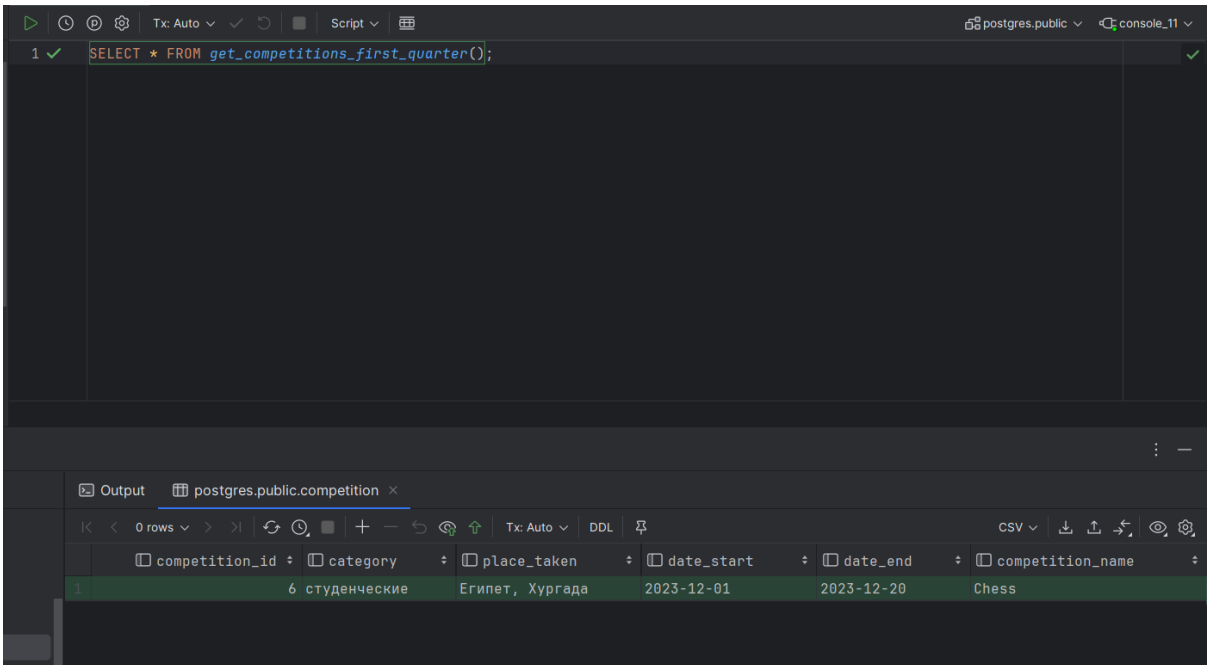
Создание процедуры:

```
CREATE OR REPLACE FUNCTION get_competitions_first_quarter()
RETURNS SETOF competition
LANGUAGE plpgsql
AS $$
BEGIN
RETURN QUERY
    SELECT *
    FROM competition
    WHERE date_start >= DATE_TRUNC('year', CURRENT_DATE)::DATE
      AND date_start < DATE_TRUNC('year', CURRENT_DATE)::DATE + INTERVAL '3
month';
END;
$$;
```

```
postgres=# CREATE OR REPLACE FUNCTION get_competitions_first_quarter()
postgres-# RETURNS SETOF competition
postgres-# LANGUAGE plpgsql
postgres-# AS $$
postgres$# BEGIN
postgres$# RETURN QUERY
postgres$# SELECT *
postgres$# FROM competition
postgres$# WHERE date_start >= DATE_TRUNC('year', CURRENT_DATE)::DATE
postgres$# AND date_start < DATE_TRUNC('year', CURRENT_DATE)::DATE + INTERVAL '3 month';
postgres$# END;
postgres$# $$;
CREATE FUNCTION
postgres=#
```

Алгоритм: Функция возвращает все соревнования в виде “среза” таблицы “*competition*”, где дата начала находится в диапазоне квартала нынешнего года.

Проверка функции:




**3. Для повышения рейтинга и оклада тренера после участия в соревновании.**

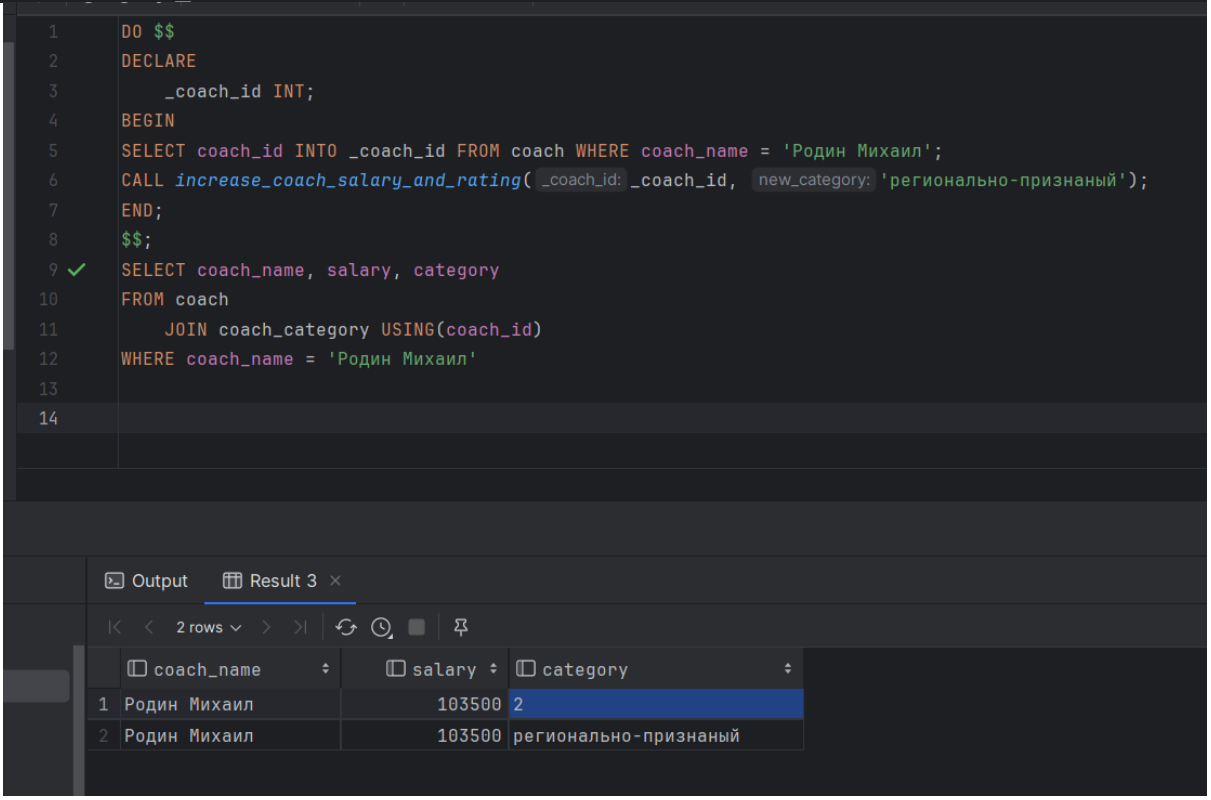
```
CREATE OR REPLACE PROCEDURE increase_coach_salary_and_rating(
_coach_id INT, new_category VARCHAR(40))
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO coach_category ( coach_id, category, date_start, date_end
)
VALUES ( _coach_id, new_category, CURRENT_DATE,
        CURRENT_DATE + INTERVAL '6 months');
UPDATE coach
SET salary = salary * 1.15
WHERE coach_id = _coach_id;
END;
$$;
```

```
postgres=# CREATE OR REPLACE PROCEDURE increase_coach_salary_and_rating(
postgres(# _coach_id INT, new_category VARCHAR(40)
postgres(# )
postgres-# LANGUAGE plpgsql
postgres-# AS $$
postgres$# BEGIN
postgres$# INSERT INTO coach_category ( coach_id, category, date_start, date_end
postgres$# )
postgres$# VALUES ( _coach_id, new_category, CURRENT_DATE,
postgres$# CURRENT_DATE + INTERVAL '6 months'
postgres$# );
postgres$# UPDATE coach
postgres$# SET salary = salary * 1.15
postgres$# WHERE coach_id = _coach_id;
postgres$# END;
postgres$# $$;
CREATE PROCEDURE
postgres=#
```

Алгоритм: Процедура принимает ID тренера и категорию на которую необходимо изменить существующую в качестве аргументов. Процедура создает новую категорию для соответствующего тренера и назначает срок действия в пол года от текущего момента. Кроме того также повышается зарплата тренера на 15 процентов.

Проверка процедуры:

```
DO $$
DECLARE
_coach_id INT;
BEGIN
SELECT coach_id INTO _coach_id FROM coach WHERE coach_name = 'Родин
Михаил';
CALL increase_coach_salary_and_rating(_coach_id, 'регионально-признаный');
END;
$$;
SELECT coach_name, salary, category
FROM coach
    JOIN coach_category USING(coach_id)
WHERE coach_name = 'Родин Михаил'
```




**4. Создать триггер для логирования событий вставки, удаления, редактирования данных.**

Суть триггера: Спортзалу необходимо четко контролировать все тренировки спортсменов, поэтому необходимо создать таблицу логов всех изменений при работе с таблицей training.

Создание таблицы логов:

```
CREATE TABLE training_audit_log ( log_id SERIAL PRIMARY KEY,
                                  training_id INT NOT NULL,
                                  collaboration_id INT NOT NULL,
                                  date_event DATE NOT NULL,
                                  place_taken VARCHAR(35) NOT NULL,
                                  type VARCHAR(25) NOT NULL,
                                  operation CHAR NOT NULL,
                                  change_timestamp TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
                                  changed_by VARCHAR(50) NOT NULL
)
```

```
postgres-# CREATE TABLE training_audit_log ( log_id SERIAL PRIMARY KEY, training_id INT NOT NULL, collaboration_id INT
OT NULL, date_event DATE NOT NULL, place_taken VARCHAR(35) NOT NULL, type VARCHAR(25) NOT NULL, operation CHAR NOT NUL
 change_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, changed_by VARCHAR(50) NOT NULL
postgres(# )
```

Укажем в ней все атрибуты таблицы training и дополнительные для остлеживания различных параметров таких как:

1. operation – тип операции
2. change_timestamp – временная метка изменения
3. changed_by – автор изменения

Создание триггерной функции:

```
create function log_training_changes() returns trigger language plpgsql as
$$

BEGIN
CASE TG_OP
WHEN 'INSERT' THEN
INSERT INTO training_audit_log( training_id, collaboration_id, date_event,
place_taken, type, operation, changed_by
)

VALUES (
        NEW.training_id,
        NEW.collaboration_id,
        NEW.date_event,
        NEW.place_taken,
        NEW.type,
        'I',CURRENT_USER);
RETURN NEW;
WHEN 'UPDATE' THEN
INSERT INTO training_audit_log( training_id, collaboration_id, date_event,
place_taken, type, operation, changed_by
)
VALUES (
        NEW.training_id,
```

```
        NEW.collaboration_id,
        NEW.date_event,
        NEW.place_taken,
        NEW.type,
        'U',CURRENT_USER);
RETURN NEW;
WHEN 'DELETE' THEN
INSERT INTO training_audit_log( training_id,
                                collaboration_id,
                                date_event,
                                place_taken,
                                type,
                                operation,
                                changed_by)
VALUES (
        OLD.training_id,
        OLD.collaboration_id,
        OLD.date_event,
        OLD.place_taken,
        OLD.type,
        'D',
        CURRENT_USER
        );
RETURN OLD;
END CASE;
END;
$$;
```

```
postgres=# create function log_training_changes() returns trigger language plpgsql as $$
postgres$# BEGIN
postgres$# CASE TG_OP
postgres$# WHEN 'INSERT' THEN
postgres$# INSERT INTO training_audit_log( training_id, collaboration_id, date_event, place_taken, type, operation, chan
ged_by
postgres$# )
postgres$# VALUES (
postgres$# NEW.training_id,
postgres$# NEW.collaboration_id,
postgres$# NEW.date_event,
postgres$# NEW.place_taken,
postgres$# NEW.type,
postgres$# 'I',
postgres$# CURRENT_USER
postgres$# );
postgres$# RETURN NEW;
postgres$# WHEN 'UPDATE' THEN
postgres$# INSERT INTO training_audit_log( training_id, collaboration_id, date_event, place_taken, type, operation, chan
ged_by
postgres$# )
postgres$# VALUES (
postgres$# NEW.training_id,
postgres$# NEW.collaboration_id,
postgres$# NEW.date_event,
postgres$# NEW.place_taken,
postgres$# NEW.type,
postgres$# 'U',
```

```
postgres$# CURRENT_USER
postgres$# );
postgres$# RETURN NEW;
postgres$# WHEN 'DELETE' THEN
postgres$# INSERT INTO training_audit_log( training_id, collaboration_id, date_event, place_taken, type, operation,
postgres$# changed_by
postgres$# )
postgres$# VALUES (
postgres$# OLD.training_id,
postgres$# OLD.collaboration_id,
postgres$# OLD.date_event,
postgres$# OLD.place_taken,
postgres$# OLD.type,
postgres$# 'D',
postgres$# CURRENT_USER
postgres$# );
postgres$# RETURN OLD;
postgres$# END CASE;
postgres$# END;
postgres$# $$;
CREATE FUNCTION
postgres=#
```

Создание самого триггера и привязка к нему уже существующей функции:

```
CREATE TRIGGER log_training_after_modify_operations
AFTER INSERT OR UPDATE OR DELETE ON training
FOR EACH ROW
EXECUTE FUNCTION log_training_changes();
```

```
postgres=# CREATE TRIGGER log_training_after_modify_operations
postgres-# AFTER INSERT OR UPDATE OR DELETE ON training
postgres-# FOR EACH ROW
postgres-# EXECUTE FUNCTION log_training_changes();
CREATE TRIGGER
postgres=#
```

Триггер будет запускать функцию *“log_training_changes()”* для каждой строки которую одна из операция (INSERT, UPDATE, DELETE) будет изменять.

Проверим работу данного триггера:

Как можно заметить, теперь каждая операция модификации данных логируется в отдельную таблицу *“training_audit_log”*

**Заключение**:

В ходе выполнения данной работы, были изучены методы использования процедур и функций в рамках СУБД PostgreSQL. Были реализованы функции и процедуры для штатных ситуация для предметной области спортивного класса.

Помимо того, были рассмотрены и созданы триггеры и триггерные функции для логирования событий над таблицей.

Данная лабораторная работа дает практические навыки использования и применения функций, процедур и триггеров над базой данных.