

ЛАБОРАТОРНАЯ РАБОТА №3

ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В PostgreSQL

Выполнил: РЫБАЛКО ОЛЕГ ДМИТРИЕВИЧ K32392

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

Оборудование: компьютерный класс.

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

Практическое задание:

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.

Задание 1:

1. Создание хранимой процедуры для получения расписания занятий для групп на определенный день недели.

Я создаю процедуру, в которой делаю запрос на получение занятий, которые проходят в переданный пользователем день недели у группы с указанным идентификатором. В качестве ответа возвращается таблица с номером пары и датой проведения

```
CREATE OR REPLACE FUNCTION GetScheduleByDayOfWeek(groupId INT, dayOfWeek INT)
    RETURNS TABLE (НомерПары INTEGER, ДатаПроведения TIMESTAMPTZ)
AS $$
BEGIN
    RETURN QUERY
    SELECT НомерПары, ДатаПроведения
    FROM Занятие as a
    INNER JOIN Группа as b
    ON a.ИдентификаторГруппы = b.Идентификатор
    WHERE extract(isodow from a.ДатаПроведения) = dow
    AND b.Идентификатор = groupId;
END
$$ LANGUAGE plpgsql;
```

getschedulebydayofweek	
(1,"2023-03-28 10:00:00")	

2. Создание хранимой процедуры для записи слушателя на курс.

В таблицу «Обучается» данная процедура вставляет новую строку, где код набора равен коду набора на курсе, который передан пользователем. Дату начала и дату окончания обучения данная процедура выбирает из таблицы набора на программу. Слушателя данная процедура получает по идентификатору, который передал пользователь

```
CREATE OR REPLACE PROCEDURE EnrollListenerToCourse(
    IN listenerId INTEGER,
    IN courseId INTEGER
)
AS $$
BEGIN
    INSERT INTO Обучается (ИдентификаторГруппы, ДатаНачалаОбучения,
ДатаОкончанияОбучения, Статус, ИдентификаторСлушателя)
    VALUES (
        (SELECT Идентификатор FROM Группа WHERE КодНабора = courseId),
        (SELECT ДатаНачала FROM НаборНаПрограмму WHERE Код = courseId),
        (SELECT ДатаОкончания FROM НаборНаПрограмму WHERE Код = courseId),
        'Учится',
        listenerId
    );
END
$$ LANGUAGE plpgsql;
```



CALL

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

В данной процедуре я делаю запрос на получение номеров аудитории, в которых не проводятся занятия в переданный день недели

```
CREATE OR REPLACE PROCEDURE GetFreeLectureRoomsByDayOfWeek(IN dayOfWeek INT)
AS $$
DECLARE
    freeRooms TEXT[];
BEGIN
    SELECT ARRAY_AGG(Аудитория.Номер)
    INTO freeRooms
    FROM Аудитория
    WHERE Аудитория.Номер NOT IN (
        SELECT DISTINCT Занятие.НомерАудитории
        FROM Занятие
        WHERE EXTRACT(DOW FROM Занятие.ДатаПроведения) = dayOfWeek
        AND Занятие.Тип = «Лекционное»
    );

    IF freeRooms IS NULL OR array_length(freeRooms, 1) = 0 THEN
        RAISE NOTICE 'Свободных аудиторий не найдено';
    ELSE
        RAISE NOTICE 'Свободные аудитории: %', freeRooms;
    END IF;
END
$$ LANGUAGE plpgsql;
```

```
NOTICE: Свободные аудитории: {102,101,103,104,106}
CALL
```

Задание 2.

Для начала я создал процедуру, которая в зависимости от типа операции (INSERT, UPDATE, DELETE) будет записывать нужные строки в таблицу «Лог». Далее я создал процедуру, которая проходит по всем таблицам в базе и добавляет триггер на INSERT, UPDATE, DELETE, который вызывает процедуру LogEvent. Затем я вызываю процедуру создания триггеров и удаляю триггер из таблицы логов.

```
CREATE OR REPLACE FUNCTION LogEvent()
  RETURNS TRIGGER AS $$
DECLARE
  tableName TEXT;
BEGIN
  tableName := TG_TABLE_NAME;

  IF TG_OP = 'INSERT' THEN
    INSERT INTO Лог (Таблица, Событие, Время)
      VALUES (tableName, 'Вставка', CURRENT_TIMESTAMP);
  ELSIF TG_OP = 'DELETE' THEN
    INSERT INTO Лог (Таблица, Событие, Время)
      VALUES (tableName, 'Удаление', CURRENT_TIMESTAMP);
  ELSIF TG_OP = 'UPDATE' THEN
    INSERT INTO Лог (Таблица, Событие, Время)
      VALUES (tableName, 'Редактирование', CURRENT_TIMESTAMP);
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION CreateLogTrigger()
  RETURNS VOID AS $$
DECLARE
  tableName TEXT;
  triggerName TEXT;
BEGIN
  FOR tableName IN SELECT table_name FROM information_schema.tables WHERE
table_schema = 'public' AND table_type = 'BASE TABLE' LOOP
    triggerName := 'LogEventTrigger_' || tableName;
    EXECUTE 'CREATE TRIGGER ' || triggerName || '
      AFTER INSERT OR DELETE OR UPDATE
      ON ' || tableName || '
      FOR EACH ROW
      EXECUTE FUNCTION LogEvent()';
  END LOOP;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT CreateLogTrigger();  
DROP TRIGGER LogEventTrigger_Лог ON Лог;
```

Таблица	Событие	Время
Обучается	Вставка	2023-05-24 21:47:27.044492
Занятие	Вставка	2023-05-24 21:51:26.373817
Занятие	Вставка	2023-05-24 21:51:34.704473
Занятие	Удаление	2023-05-24 21:52:06.503109
Занятие	Редактирование	2023-05-24 21:52:15.806375

Вывод:

После выполнения данной лабораторной работы я стал лучше понимать, как работают процедуры и триггеры в базе данных PostgreSQL. Также, я на практике смог применить свои знания, создав несколько процедур и триггеров для своей базы данных