# Создание базы данных

## Порядок выполнения

Так как в PostgreSQL не имеется оператора USE DATABASE, существующего в SQL Server, процедура создания базы данных отличается своим подходом.

Саму процедуру создания БД можно реализовать двумя способами:

1. Выполнение команд и запросов к базе через терминал;
2. Создание базы данных через GUI.

## Реализация способов создания

### Запросы в терминале

Для начала необходимо войти в терминал к основной базе сервера (по умолчанию - postgres). Это можно осуществить, введя в терминале команду psql -U <имя пользователя> и введя данные для авторизации (Рисунок 1).

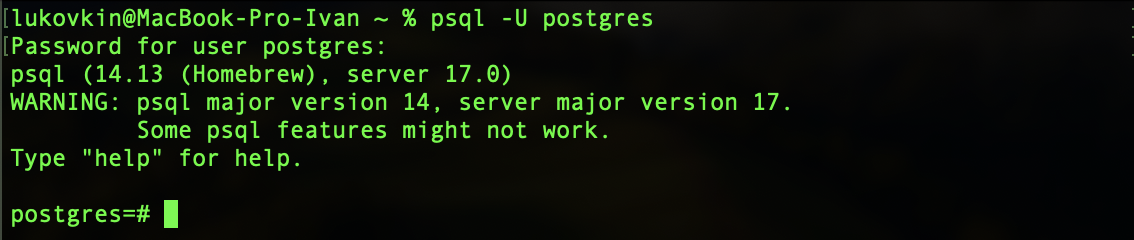


Рисунок 1 – Авторизация

После того, как подключение было установлено, можно осуществить создание базы данных и подключение к ней. Для создания используется SQL-оператор CREATE DATABASE <имя базы данных>.

При успешном выполнении SQL-запросов в терминале отображается ответ с названием осуществленной операции. Затем для проверки, что база данных была создана, необходимо переключиться на нее, используя команду:

*\c <имя базы данных>*

После успешного подключения к БД можно осуществить вывод всех баз, введя команду: *\l*.

Пример выполнения указанных команд представлен на рисунке 2.

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Рисунок 2 - Создание базы данных

Написание SQL-запросов в терминале заключается в построчном вводе запросов. Каждый запрос к базе данных отделяется точкой с запятой. Пример выполнения запроса создания и вывода новой таблицы Student представлен на рисунке 3.

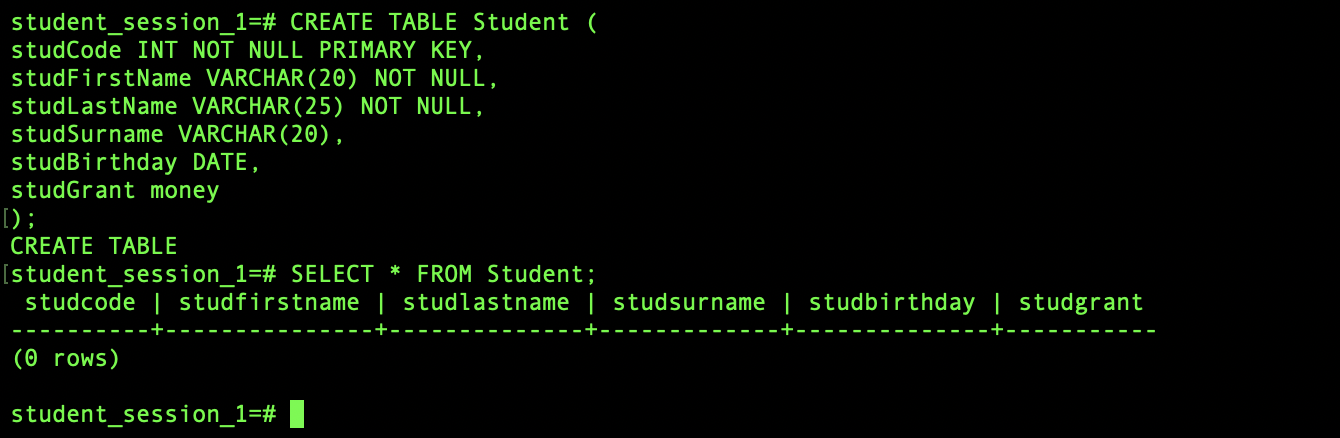


Рисунок 3 - Создание новой таблицы

### Работа в графическом интерфейсе

Для осуществления запросов в графическом интерфейсе используется программа, предоставляемая при установке СУБД – pgAdmin4. Для создания запросов необходимо открыть окно «Запросник», находящееся в меню «Инструменты» (Рисунок 4).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 4 - Меню запросов

В открывшемся окне можно ввести запрос на создание базы данных. При нажатии на кнопку Execute выполнится запрос, и в окне вывода появится сообщение о его осуществлении (Рисунок 5).

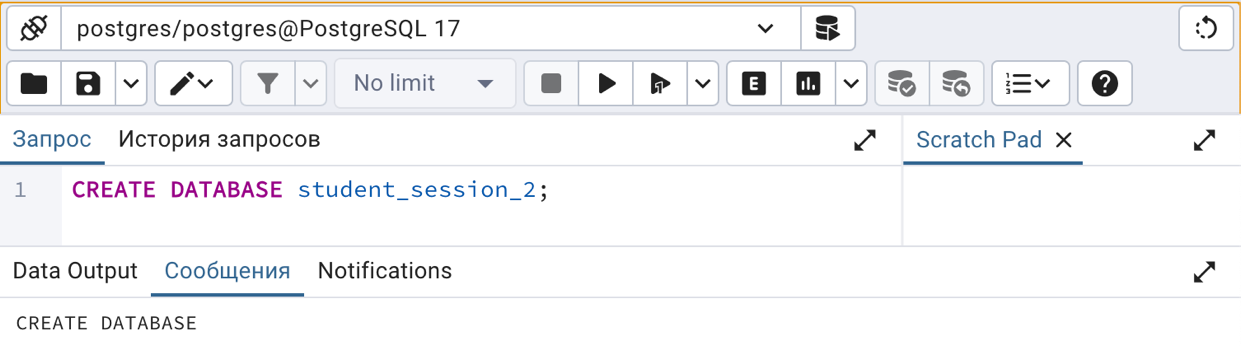


Рисунок 5 - Выполнение запроса

После создания базы данных можно создать новый запрос, выбрав в обозревателе объектов (Object explorer) необходимую базу данных и осуществив для нее необходимые запросы (Рисунок 6).

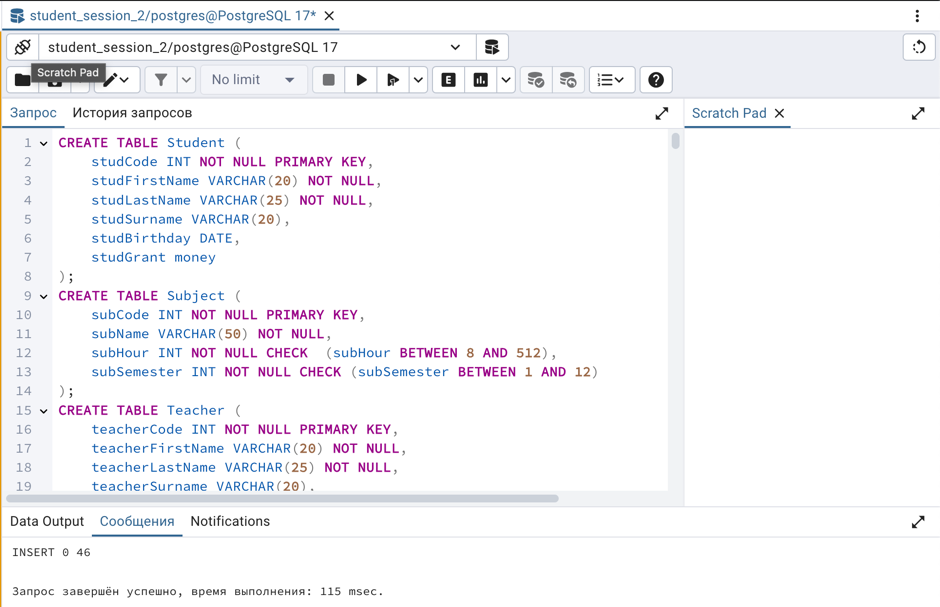


Рисунок 6 - Создание таблиц

При успешном осуществлении запроса у базы данных появятся новые таблицы (Рисунок 7).

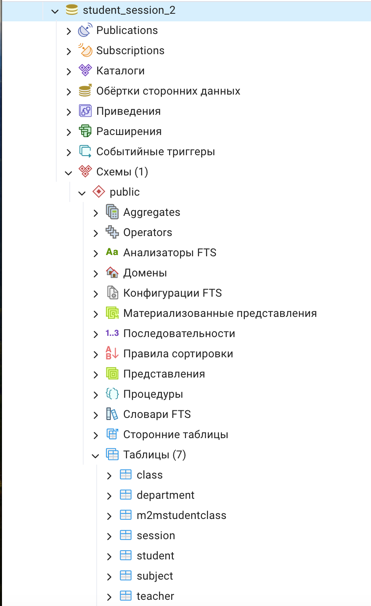


Рисунок 7 - Результат выполнения запроса создания таблиц

# Создание правил, умолчаний, представлений

# Создание хранимых процедур

# Создание триггеров

## Основные компоненты триггеров

В PostgreSQL триггеры также создаются с помощью команды CREATE TRIGGER, но необходимо сначала создать функцию для триггера, а затем привязать её к триггеру (Рисунок).

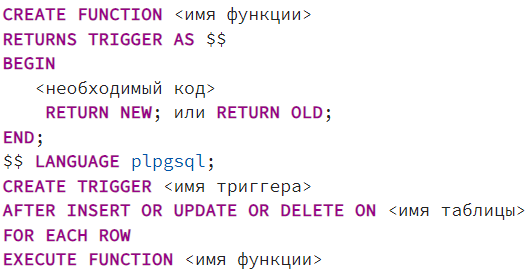


Рисунок – Синтаксис создания триггера

## Виды триггеров

### BEFORE триггеры

Срабатывают перед выполнением операции (INSERT, UPDATE или DELETE). Позволяют изменять данные перед их записью (Рисунок)

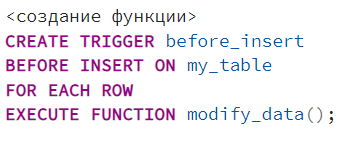


Рисунок 8 - Пример создания BEFORE триггера

Если триггер помечен опцией **FOR EACH ROW**, тогда функция вызывается для каждой строки, которая изменяется в результате события. Например, если сделать UPDATE для 100 строк, триггерная функция UPDATE будет вызываться 100 раз, по одному разу для каждой обновлённой строки.

Опция FOR EACH STATEMENT вызовет функцию только один раз для каждого оператора, независимо от количества изменяемых строк.

### AFTER триггеры

Срабатывают после завершения операции. Используются для выполнения действий, которые не требуют изменения данных (Рисунок).

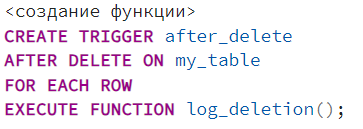


Рисунок 9 - Пример создания AFTER триггера

### INSTEAD OF триггеры

Используются в основном для представлений (views). Позволяют заменить стандартное поведение операции (Рисунок ).

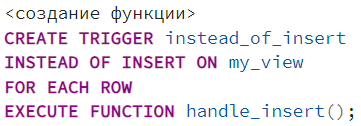


Рисунок 10 - Пример создания INSTEAD OF триггера

## Специальные переменные

В триггерных функциях доступны специальные переменные:

1. NEW. Содержит новые значения строки для операций INSERT и UPDATE.
2. OLD. Содержит старые значения строки для операций UPDATE и DELETE.

## Ограничения

1. Максимальное количество триггеров: PostgreSQL позволяет создавать до 32 триггеров на одну таблицу для одного события.
2. Рекурсивные триггеры: Триггеры могут вызывать друг друга, что может привести к рекурсии. Для управления рекурсией можно использовать параметры конфигурации.

## Основные отличия создания триггеров в SQL Server и PostgreSQL

1. Запретить пересдачу экзамена более трех раз
2. При получении студентом оценки 3, стипендию установить NULL.

Наглядные отличия в синтаксисе представлены в таблице

|  |  |  |
| --- | --- | --- |
| Таблица 1 – Примеры создания триггеров | | |
| **№** | **SQL Server** | **PostgreSQL** |
| 1 | CREATE TRIGGER trg\_check\_exam\_attempts  ON Session  INSTEAD OF INSERT  AS  BEGIN  DECLARE @student INT, @subject INT;  SELECT @student = student, @subject = subject FROM inserted;  IF (SELECT COUNT(\*) FROM Session WHERE student = @student AND subject = @subject) >= 3  BEGIN  RAISERROR('Нельзя сдавать экзамен более трех раз', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END  INSERT INTO Session (student, subject, teacher, examDate, mark)  SELECT student, subject, teacher, examDate, mark FROM inserted;  END; | CREATE OR REPLACE FUNCTION limit\_exam\_retakes()  RETURNS TRIGGER AS $$  DECLARE  exam\_count INT;  BEGIN  SELECT COUNT(\*)  INTO exam\_count  FROM Session  WHERE student = NEW.student  AND subject = NEW.subject;  IF exam\_count >= 3 THEN  RAISE EXCEPTION 'Студент не может пересдать экзамен более трех раз';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_limit\_exam\_retakes  BEFORE INSERT ON Session  FOR EACH ROW EXECUTE FUNCTION limit\_exam\_retakes(); |
| 2 | CREATE TRIGGER trg\_set\_grant\_null  ON Session  AFTER INSERT, UPDATE  AS  BEGIN  UPDATE Student  SET studGrant = NULL  WHERE studCode IN (SELECT student FROM inserted WHERE mark = 3);  END; | CREATE OR REPLACE FUNCTION set\_scholarship\_null\_on\_low\_mark()  RETURNS TRIGGER AS $$  BEGIN  IF NEW.mark = 3 THEN  UPDATE Student  SET studGrant = NULL  WHERE studCode = NEW.student  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_set\_scholarship\_null\_on\_low\_mark  AFTER INSERT OR UPDATE ON Session  FOR EACH ROW EXECUTE FUNCTION set\_scholarship\_null\_on\_low\_mark |

Основные отличия между PostgreSQL и SQL Server:

1. Синтаксис триггеров:

* В PostgreSQL триггеры создаются с помощью CREATE OR REPLACE FUNCTION и затем связываются с триггером.
* В SQL Server триггеры создаются непосредственно с помощью команды CREATE TRIGGER.

1. Обработка данных:

* В PostgreSQL используются специальные конструкции NEW и OLD для доступа к новым и старым значениям.
* В SQL Server используется специальная таблица inserted для доступа к новым значениям.

1. Обработка ошибок:

* В PostgreSQL ошибки вызываются с помощью RAISE EXCEPTION.
* В SQL Server используется RAISERROR.