Содержание

[1 СОЗДАНИЕ БАЗЫ ДАННЫХ 4](#_Toc182526390)

[1.1 Порядок выполнения 4](#_Toc182526391)

[1.2 Реализация способов создания 4](#_Toc182526392)

[1.2.1 Запросы в терминале 4](#_Toc182526393)

[1.2.2 Работа в графическом интерфейсе 5](#_Toc182526394)

[2 СОЗДАНИЕ ПРАВИЛ, УМОЛЧАНИЙ, ПРЕДСТАВЛЕНИЙ 8](#_Toc182526395)

[3 ХРАНИМЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ 9](#_Toc182526396)

[3.1 Языки программирования хранимых процедур и функций 9](#_Toc182526397)

[3.1.1 Языки программирования процедур 9](#_Toc182526398)

[3.1.2 Создание процедуры 9](#_Toc182526399)

[3.1.3 Пример выполнения процедуры 10](#_Toc182526400)

[3.1.4 Работа с переменными 10](#_Toc182526401)

[3.1.5 Вывод переменных в процедуре 10](#_Toc182526402)

[3.1.6 Вывод таблиц в процедурах 11](#_Toc182526403)

[3.2 Хранимые процедуры 11](#_Toc182526404)

[3.2.1 Создание хранимой процедуры 11](#_Toc182526405)

[3.2.2 Выполнение хранимых процедур 12](#_Toc182526406)

[3.2.3 Пример №1 12](#_Toc182526407)

[3.2.4 Пример №2 13](#_Toc182526408)

[4 СОЗДАНИЕ ТРИГГЕРОВ 15](#_Toc182526409)

[4.1 Основные компоненты триггеров 15](#_Toc182526410)

[4.2 Виды триггеров 15](#_Toc182526411)

[4.2.1 BEFORE триггеры 15](#_Toc182526412)

[4.2.2 AFTER триггеры 16](#_Toc182526413)

[4.2.3 INSTEAD OF триггеры 16](#_Toc182526414)

[4.3 Специальные переменные 16](#_Toc182526415)

[4.4 Ограничения 16](#_Toc182526416)

[4.5 Основные отличия создания триггеров в SQL Server и PostgreSQL 17](#_Toc182526417)

[Приложение А. Текст запроса на создание таблицы stud\_session 19](#_Toc182526418)

[Приложение Б. Текст запроса на создание таблицы short\_db\_session 24](#_Toc182526419)

[Приложение В. Диаграмма связей таблиц базы данных stud\_session 25](#_Toc182526420)

[Приложение Г. Диаграмма связей таблиц базы данных short\_db\_session 26](#_Toc182526421)

# 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).

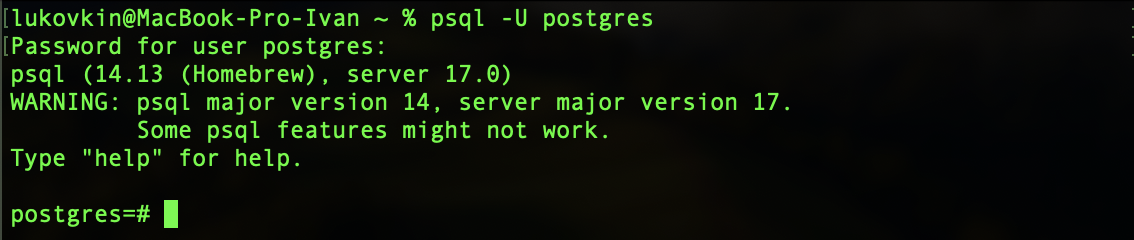


Рисунок 1 – Авторизация

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.

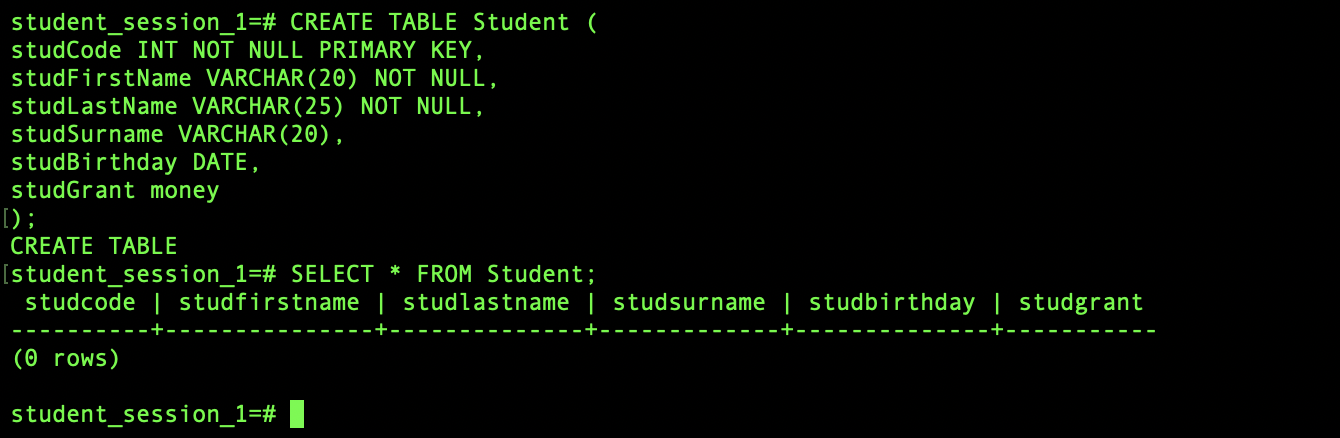


Рисунок 3 - Создание новой таблицы

### 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).

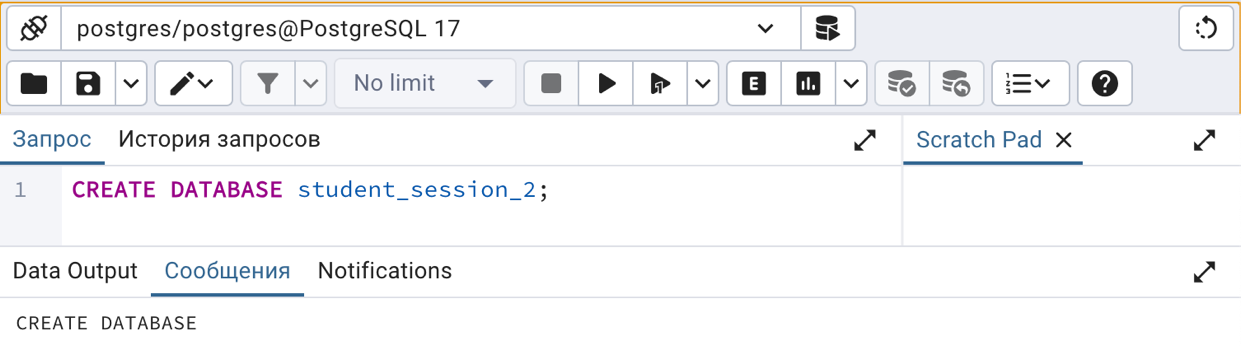


Рисунок 5 - Выполнение запроса

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).

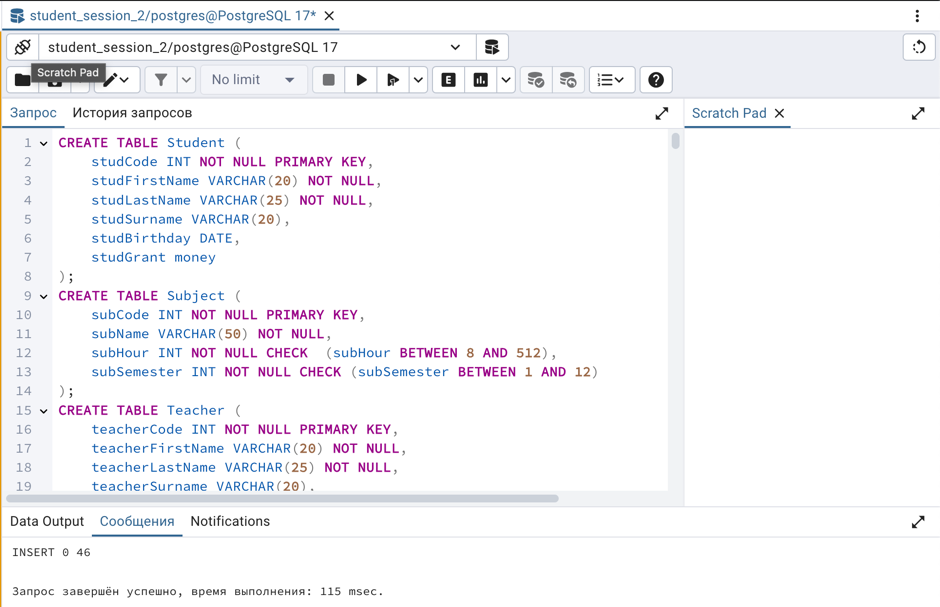


Рисунок 6 - Создание таблиц

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

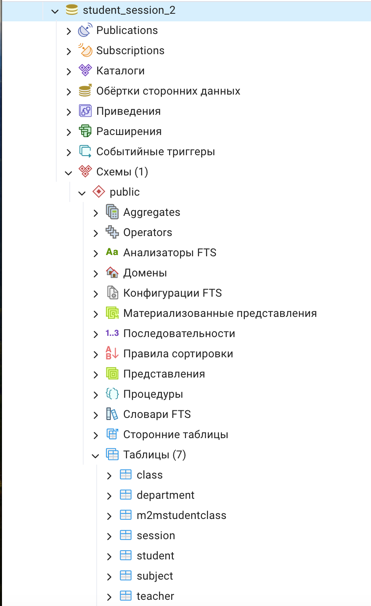


Рисунок 7 - Результат выполнения запроса создания таблиц

# СОЗДАНИЕ ПРАВИЛ, УМОЛЧАНИЙ, ПРЕДСТАВЛЕНИЙ

# ХРАНИМЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

## Языки программирования хранимых процедур и функций

Для программирования пользовательских алгоритмов, необходимых для создания хранимых процедур и функций в PostgreSQL, используются отдельные языки программирования процедур (сокр. pl – procedure language).

### Языки программирования процедур

Одни из самых популярных языков, используемых в программировании:

* pl/pgSQL – Один из самых популярных языков, созданный разработчиками СУБД для вызова системных процедур и функций.
* pl/Tcl - это загружаемый процедурный язык для СУБД PostgreSQL, позволяющий использовать язык Tcl для написания функций и триггерных процедур.
* PL/Perl — это загружаемый процедурный язык, позволяющий реализовывать функции PostgreSQL на языке программирования Perl.
* PL/Python. Позволяет писать функции PostgreSQL на языке Python.

### Создание процедуры

Синтаксис создания процедуры, выполняемой один раз:

|  |
| --- |
| DO LANGUAGE <язык программирования> $<название\_процедуры>$ DECLARE  *<объявление переменных>* BEGIN  *<текст процедуры>* END; $<название\_процедуры>$; |

Объявление блока кода с процедурой используется ключевое слово DO. Текст процедуры помещается в двойные знаки доллара ($$). Внутри этой конструкции можно объявить название процедуры. Название должно быть указано как в начале текста процедуры, так и в его конце.

Вслед за этим указывается ключевое слово LANGUAGE, после которого указывается используемый язык программирования процедуры.

Если процедура подразумевает использования переменных, их необходимо указать в области объявления (DECLARE).

Текст процедуры помещается в конструкцию BEGIN-END.

### Пример выполнения процедуры

|  |
| --- |
| DO LANGUAGE plpgsql $print\_department\_name$ DECLARE  department\_name varchar := 'Кафедра ЭВМ'; BEGIN  RAISE NOTICE 'Department name: %', department\_name; END; $print\_department\_name$; |

В данной процедуре объявлено ее название: print\_department\_name. Также используется одна переменная department\_name, которой предварительно задается значение.

### Работа с переменными

В языке PLpgSQL можно задать переменную двумя способами:

* С предварительно заданным значением;
* Без объявления начального значения – в таком случае вне зависимости от типа данных переменная будет равна null.

Синтаксис работы с переменными представлен ниже:

|  |
| --- |
| DO LANGUAGE plpgsql $print\_department\_name$ DECLARE  department\_name varchar := 'Кафедра ЭВМ'; *-- Объявление переменной с заданным значением* department\_code int; *-- Объявление переменной без заданного значения* BEGIN  department\_code := 1; *-- Присвоение значения переменной* RAISE NOTICE '%', department\_code;  RAISE NOTICE 'Department name: %', department\_name; END; $print\_department\_name$; |

### Вывод переменных в процедуре

Для вывода данных в процедуре в PostgreSQL используется конструкция RAISE NOTICE. Данная команда позволяет вывести форматированную строку.

Пример вывода:

|  |
| --- |
| DO LANGUAGE plpgsql $print\_greeting$ BEGIN  RAISE NOTICE 'Hello, PostgreSQL!'; *-- Вывод без параметров* RAISE NOTICE 'Hello, %!', 'PostgreSQL'; *-- Вывод с параметром* END; $print\_greeting$; |

### Вывод таблиц в процедурах

В отличие от других SQL-диалектов в PostgreSQL нельзя внутри процедуры воспользоваться оператором SELECT, позволяющим предоставить пользователю представление данных в виде таблицы.

Для представления таблиц в процедурах все выводимые строки в виде кортежей выводятся последовательно в цикле в терминал.

Пример вывода всех студентов из таблицы student:

|  |
| --- |
| DO LANGUAGE plpgsql $print\_student\_table$ BEGIN  DECLARE  data record;  BEGIN  for data in (SELECT *\** FROM student) loop  RAISE NOTICE '%', data;  end loop;  end; END; $print\_student\_table$; |

В данном примере для обхода каждой строки таблицы student задается переменная data с типом данных record – запись таблицы.

В цикле for идет обход всех кортежей таблицы student, и информация из каждой строки выводится последовательно через конструкцию RAISE NOTICE, позволяющую сформировать форматированную строку.

## Хранимые процедуры

Хранимые процедуры (ХП) представляют собой последовательность операторов языка SQL, которые хранятся в базе данных.

При первом вызове пользовательской хранимой процедуры СУБД генерирует и оптимизирует план выполнения хранимой процедуры. ХП сохраняются в базе данных в откомпилированном виде, поэтому скорость их выполнения выше, чем скорость выполнения обычных SQL-запросов.

### Создание хранимой процедуры

Синтаксис создания хранимой процедуры представлен ниже:

|  |
| --- |
| CREATE [OR REPLACE] PROCEDURE <имя\_процедуры>(  имя\_параметра тип\_параметра,  ...,  OUT выходной\_параметр тип\_параметра  ) AS $procedure\_text$  <Текст процедуры>  $procedure\_text$ LANGUAGE <язык процедуры>; |

Создание процедуры начинается с команды CREATE PROCEDURE. Перед ключевым словом CREATE можно добавить команду OR REPLACE, позволяющую при запуске скрипта перезаписывать существующую процедуру.

После объявления имени процедуры идет описание входных и выходных параметров в формате *имя\_параметра тип\_параметра*. Перед выходными параметрами процедуры используется ключевое слово OUT, позволяющее взаимодействовать с внешним параметром.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *get\_student\_full\_name\_by\_number*(  student\_number int,  OUT student\_full\_name varchar(30) ) AS $get\_student\_full\_name$  BEGIN  student\_full\_name := (  SELECT full\_name  FROM student  WHERE number = student\_number  );  END; $get\_student\_full\_name$ LANGUAGE plpgsql; |

### Выполнение хранимых процедур

Выполняются хранимые процедуры с помощью оператора CALL, где указывается имя хранимой процедуры и список параметров. Для того чтобы принять значение возвращаемого параметра, необходимо объявить переменную для его хранения в скрипте или другой хранимой процедуре, которая вызывает данную хранимую процедуру.

|  |
| --- |
| DO $display\_student\_full\_name$  DECLARE  student\_full\_name varchar(30);  student\_number int := 1;  BEGIN  CALL *get\_student\_full\_name\_by\_number*(student\_number);  RAISE NOTICE 'Student full name: %', student\_full\_name;  END; $ display\_student\_full\_name $; |

### Пример №1

Разработать хранимую процедуру, которая по значениям «номер студента», «номер предмета» и «дата экзамена» выдает оценку студента по данному предмету в виде слова «отлично», «хорошо», «удовлетворительно» или «неудовлетворительно».

|  |
| --- |
| CREATE OR REPLACE PROCEDURE *display\_student\_marks\_by\_text*(  student\_number int,  subject\_number int,  input\_date\_of\_exam date,  OUT word\_mark varchar(20) ) AS $get\_mark\_to\_string$  DECLARE  find\_mark int;  BEGIN  find\_mark := (  SELECT mark  FROM exam  WHERE  student = student\_number  AND subject = subject\_number  AND date\_of\_exam = input\_date\_of\_exam  );  word\_mark := (CASE find\_mark  WHEN 5 THEN 'отлично'  WHEN 4 THEN 'хорошо'  WHEN 3 THEN 'удовлетворительно'  WHEN 2 THEN 'неудовлетворительно'  ELSE 'непр. значение'  END);  END $get\_mark\_to\_string$ LANGUAGE plpgsql;  DO $display\_string\_mark$  DECLARE  student int := 1;  subject int := 1;  test\_date\_of\_exam date = '2021-06-10';  output\_mark varchar(20);  BEGIN  CALL *display\_student\_marks\_by\_text*(student, subject, test\_date\_of\_exam, output\_mark);  RAISE NOTICE 'Оценка студента №% за предмет №%, сданный %, - %', student, subject, test\_date\_of\_exam, output\_mark;  END $display\_string\_mark$ LANGUAGE plpgsql; |

### Пример №2

Разработать хранимую процедуру для добавления новых студентов в таблицу STUD. Процедура должна работать в соответствии со следующими бизнес-правилами: студенты с одинаковым ФИО не могут учиться в одной группе.

|  |
| --- |
| CREATE PROCEDURE *insert\_new\_student*(  student\_number int,  student\_full\_name varchar(30),  student\_group varchar(5),  year\_of\_birth int) AS $execute\_an\_insert$  BEGIN  IF NOT *EXISTS* (  SELECT *\** FROM student  WHERE full\_name = student\_full\_name  AND student\_group = group\_number  ) THEN  INSERT INTO student VALUES (  student\_number,  student\_full\_name,  student\_group,  year\_of\_birth  );  ELSE  RAISE exception 'Такой студент уже есть';  END IF;  END; $execute\_an\_insert$ LANGUAGE plpgsql; |

При разработке информационной системы требуется обеспечить контроль целостности данных на трех уровнях:

1. Уровень таблиц (CREATE TABLE, ALTER TABLE);

2. Уровень хранимых процедур и триггеров;

3. Уровень клиентских приложений.

Наличие трех уровней объясняется следующими причинами:

1. Обычно на каждом уровне работают разные разработчики или группы разработчиков;

2. Не все ограничения можно реализовать на первом уровне;

3. Хороший стиль программирования БД подразумевает что пользователю не дается возможность работать напрямую с таблицами баз данных. Все это делается через хранимые процедуры

# 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).

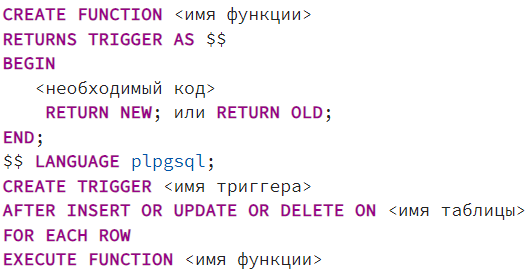


Рисунок – Синтаксис создания триггера

## Виды триггеров

### BEFORE триггеры

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

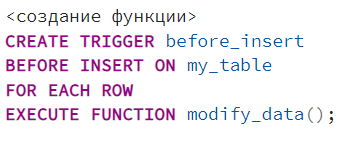


Рисунок 8 - Пример создания BEFORE триггера

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).

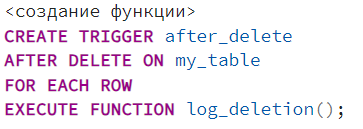


Рисунок 9 - Пример создания AFTER триггера

### INSTEAD OF триггеры

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

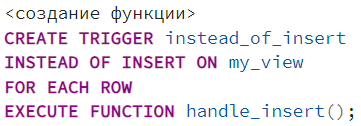


Рисунок 10 - Пример создания INSTEAD OF триггера

## 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.

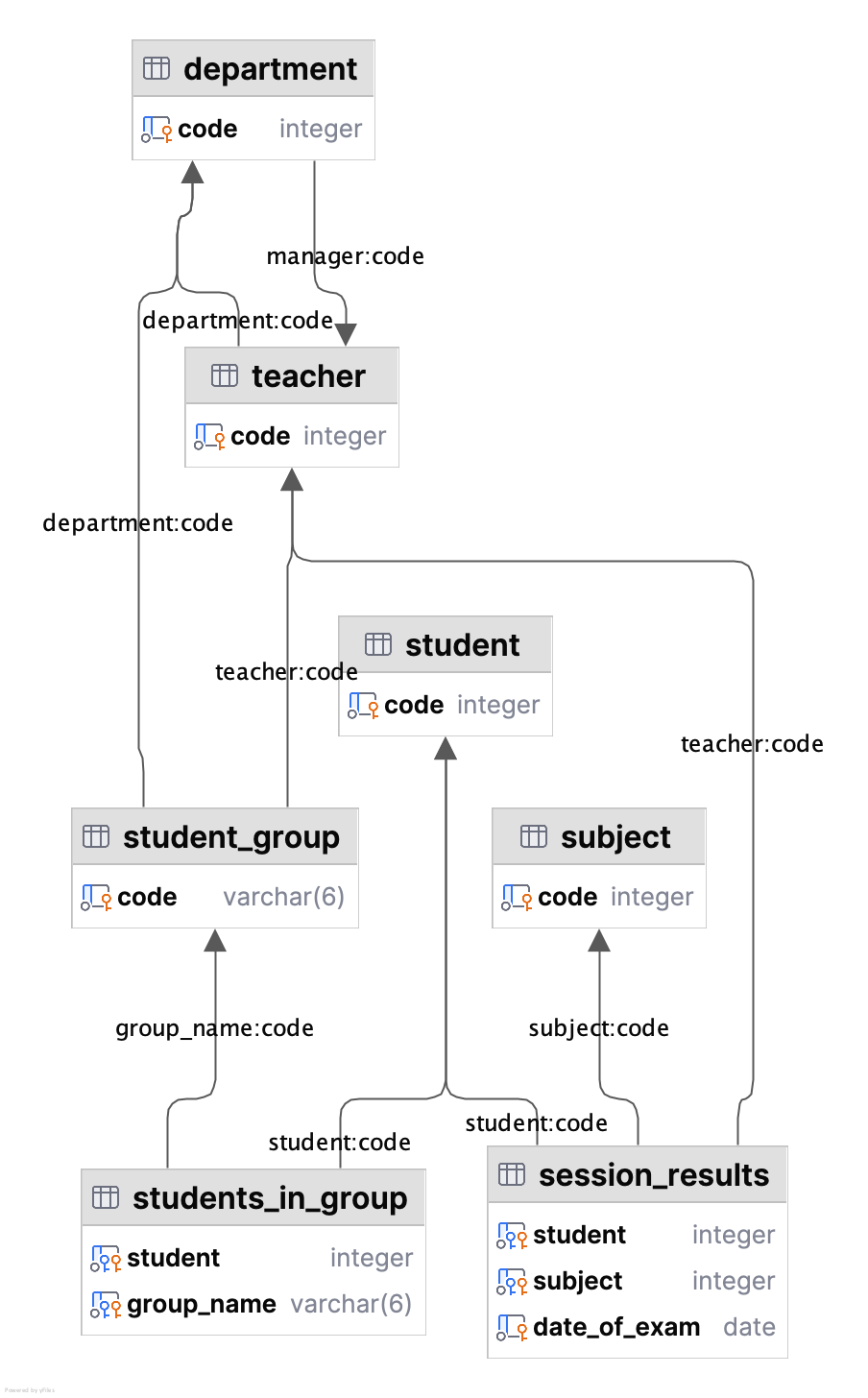
Приложение А. Текст запроса на создание таблицы stud\_session

|  |
| --- |
| DROP DATABASE IF EXISTS stud\_session (FORCE); *-- Используется для пересоздания БД* CREATE DATABASE stud\_session; *-- Создание БД*  CREATE TABLE student (  code INT NOT NULL PRIMARY KEY,  firstname VARCHAR(20) NOT NULL,  lastname VARCHAR(25) NOT NULL,  surname VARCHAR(20),  birthday DATE,  scholarship money ); CREATE TABLE subject (  code INT NOT NULL PRIMARY KEY,  full\_name VARCHAR(50) NOT NULL,  hours\_count INT NOT NULL CHECK (hours\_count BETWEEN 8 AND 512),  semesters\_count INT NOT NULL CHECK (semesters\_count BETWEEN 1 AND 12) ); CREATE TABLE teacher (  code INT NOT NULL PRIMARY KEY,  firstname VARCHAR(20) NOT NULL,  lastname VARCHAR(25) NOT NULL,  surname VARCHAR(20),  department INT,  rank VARCHAR(20),  post VARCHAR(25) ); CREATE TABLE department (  code INT NOT NULL PRIMARY KEY,  full\_name VARCHAR(50) NOT NULL UNIQUE,  manager INT ); ALTER TABLE teacher  ADD CONSTRAINT fk\_teacher\_to\_department  FOREIGN KEY (department)  REFERENCES department(code) ; ALTER TABLE department  ADD CONSTRAINT fk\_department\_manager\_to\_teacher  FOREIGN KEY (manager)  REFERENCES teacher(code) ; CREATE TABLE student\_group (  code VARCHAR(6) NOT NULL PRIMARY KEY,  teacher INT,  department INT ); ALTER TABLE student\_group  ADD CONSTRAINT fk\_department\_to\_student\_group  FOREIGN KEY (department)  REFERENCES department(code) ; ALTER TABLE student\_group  ADD CONSTRAINT fk\_teacher\_to\_student\_group  FOREIGN KEY (teacher)  REFERENCES teacher(code) ; CREATE TABLE students\_in\_group (  student INT,  group\_name VARCHAR(6),  date\_start DATE,  date\_end DATE,  CONSTRAINT pk\_students\_in\_groups  PRIMARY KEY (student, group\_name) ); ALTER TABLE students\_in\_group  ADD CONSTRAINT fk\_student\_in\_groups\_to\_student\_group  FOREIGN KEY (group\_name)  REFERENCES student\_group(code) ; ALTER TABLE students\_in\_group  ADD CONSTRAINT FK\_m2mStudentClass\_Student  FOREIGN KEY (student)  REFERENCES student (code) ; CREATE TABLE session\_results (  student INT,  subject INT,  teacher INT,  date\_of\_exam DATE,  mark INT NOT NULL CHECK (mark BETWEEN 2 AND 5),  CONSTRAINT pk\_session  PRIMARY KEY (student, subject, date\_of\_exam) ); ALTER TABLE session\_results  ADD CONSTRAINT fk\_session\_to\_student  FOREIGN KEY (student)  REFERENCES student (code) ; ALTER TABLE session\_results  ADD CONSTRAINT fk\_session\_to\_subject  FOREIGN KEY (subject)  REFERENCES subject (code) ; ALTER TABLE session\_results  ADD CONSTRAINT fk\_session\_\_to\_teacher  FOREIGN KEY (teacher)  REFERENCES teacher(code) ; INSERT INTO student (  code, firstname, surname, lastname, birthday, scholarship) VALUES  (1, 'Тимур', 'Тютючкин', 'Иванович', '2006-05-09', 1600),  (2, 'Максим', 'Булатов', 'Игоревич', '2003-10-18', 3400),  (3, 'Артем', 'Васильев', 'Денисович', '1999-03-03', 700),  (4, 'Иван', 'Луковкин', 'Иванович', '2006-04-04', 5600),  (5, 'Владислав', 'Бекренев', 'Петрович', '2005-05-05', 1900),  (6, 'Александра', 'Маликова', 'Александровна', '2006-06-06', 1500),  (7, 'Екатерина', 'Першина', 'Станиславовна', '2006-07-07', 1100),  (8, 'Евгений', 'Вольский', 'Михайлович', '2005-08-08', 1200),  (9, 'Михаил', 'Паршиков', 'Денисович', '2005-09-09', 1300),  (10, 'Максим', 'Картаков', 'Викторович', '2004-10-10', 1400),  (11, 'Николай', 'Задонский', 'Иванович', '2005-12-09', 1300),  (12, 'Анна', 'Потапова', 'Владимировна', '2005-02-25', 1300),  (13, 'Дмитрий', 'Андреев', 'Андреевич', '2006-10-11', 1300),  (14, 'Кирилл', 'Сурыгин', 'Александрович', '2006-06-21', 1300),  (15, 'Дмитрий', 'Аккуратов', 'Андреевич', '2006-06-20', 1300),  (16, 'Татьяна', 'Бурцева', 'Николаевна', '2006-12-30', 1300),  (17, 'Елизавета', 'Петрова', 'Валерьевна', '2005-01-15', 1300) ; INSERT INTO subject (code, full\_name, hours\_count, semesters\_count) VALUES  (1, 'Информатика', 108, 2),  (2, 'Физика', 512, 2),  (3, 'Базы данных', 108, 3),  (4, 'Программирование на SQL', 216, 5),  (5, 'ЭВМ и ПУ', 264, 3),  (6, 'Высшая математика', 512, 2),  (7, 'Математическая логика', 108, 2),  (8, 'Иностранный язык', 512, 3),  (9, 'Философия', 72, 2),  (10, 'Экономика', 72, 3) ; INSERT INTO department (code, full\_name) VALUES  (1, 'Кафедра ЭВМ'),  (2, 'Кафедра ОиЭФ'),  (3, 'Кафедра САПР ВС'),  (4, 'Кафедра ВМ'),  (5, 'Кафедра ВПМ'),  (6, 'Кафедра ИТГД'),  (7, 'Кафедра ИБ'),  (8, 'Кафедра иностранных языков'),  (9, 'Кафедра ИФП'),  (11, 'Кафедра КТ'),  (10, 'Кафедра ЭМОП') ; INSERT INTO teacher(  code, firstname, surname, lastname, department, rank, post) VALUES  (1, 'Борис', 'Костров', 'Васильевич', 1, 'Профессор', 'Зав. кафедрой'),  (2, 'Ангелина', 'Вьюгина', 'Алексеевна', 1, 'Доцент', 'Старший преподаватель'),  (3, 'Наталья', 'Гринченко', 'Николаевна', 1, 'Кандидат наук', 'Доцент'),  (4, 'Геннадий', 'Овечкин', 'Владимирович', 5, 'Доктор наук', 'Профессор'),  (5, 'Светлана', 'Баранова', 'Николаевна', 1, 'Без звания', 'Ассистент'),  (6, 'Марина', 'Бакулева', 'Алексеевна', 3, 'Кандидат наук', 'Доцент'),  (7, 'Вячеслав', 'Корячко', 'Петрович', 3, 'Доктор наук', 'Зав. кафедрой'),  (8, 'Дмитрий', 'Перепелкин', 'Александрович', 3, 'Доктор наук', 'Профессор'),  (9, 'Алексей', 'Бубнов', 'Алексеевич', 5, 'Кандидат наук', 'Доцент'),  (11, 'Михаил', 'Дубков', 'Викторович', 2, 'Доктор наук', 'Зав. кафедрой'),  (12, 'Кирилл', 'Бухенский', 'Валентинович', 4, 'Кандидат наук', 'Зав. кафедрой'),  (13, 'Дмитрий', 'Наумов', 'Анатольевич', 6, 'Кандидат наук', 'Зав. кафедрой'),  (14, 'Виктор', 'Пржегорлинский', 'Николаевич', 7, 'Кандидат наук', 'Зав. кафедрой'),  (15, 'Наталья', 'Есенина', 'Евгеньевна', 8, 'Кандидат наук', 'Зав. кафедрой'),  (16, 'Александр', 'Соколов', 'Станиславович', 9, 'Доктор наук', 'Зав. кафедрой'),  (17, 'Елена', 'Евдокимова', 'Николаевна', 10, 'Доктор наук', 'Зав. кафедрой'),  (18, 'Дмитрий', 'Устюков', 'Игоревич', 1, 'Кандидат наук', 'Доцент'),  (19, 'Сергей', 'Гусев', 'Игоревич', 11, 'Доктор наук', 'Зав. кафедрой'),  (10, 'Ирина', 'Панина', 'Сергеевна', 1, 'Без звания', 'Старший преподаватель') ;  UPDATE department  SET manager = CASE code  WHEN 1 THEN 1  WHEN 2 THEN 11  WHEN 3 THEN 7  WHEN 4 THEN 12  WHEN 5 THEN 4  WHEN 6 THEN 13  WHEN 7 THEN 14  WHEN 8 THEN 15  WHEN 9 THEN 16  WHEN 10 THEN 17  WHEN 11 THEN 19  END WHERE code BETWEEN 1 AND 11;   INSERT INTO student\_group (code, teacher, department) VALUES  ('245', 18, 1),  ('240', 10, 1),  ('246', 6, 3),  ('2415', 19, 11),  ('345', 5, 1),  ('270', 17, 10),  ('3714', 3, 1),  ('3724', 3, 1),  ('2414', 9, 5),  ('142', 14, 7) ; INSERT INTO students\_in\_group (student, group\_name, date\_start, date\_end) VALUES  (1, '345', '2023-09-01', NULL),  (2, '142', '2021-09-01', NULL),  (3, '142', '2021-09-01', NULL),  (4, '245', '2022-09-01', NULL),  (5, '245', '2022-09-01', NULL),  (6, '245', '2022-09-01', NULL),  (7, '3714', '2023-09-01', NULL),  (8, '3714', '2023-09-01', NULL),  (9, '3714', '2023-09-01', NULL),  (11, '345', '2023-09-01', NULL),  (12, '345', '2023-09-01', NULL),  (13, '2415', '2022-09-01', NULL),  (14, '240', '2022-09-01', NULL),  (15, '246', '2022-09-01', NULL),  (16, '246', '2022-09-01', NULL),  (17, '245', '2022-09-01', '2023-07-01'),  (17, '246', '2023-07-02', NULL),  (10, '345', '2023-09-01', NULL) ;  INSERT INTO session\_results (student, subject, teacher, date\_of\_exam, mark) VALUES  (4, 3, 3, '2023-12-20', 5),  (5, 3, 3, '2023-12-20', 4),  (6, 3, 3, '2023-12-20', 5),  (13, 3, 3, '2023-12-20', 5),  (15, 3, 3, '2023-12-20', 4),  (16, 3, 3, '2023-12-20', 4),  (17, 3, 3, '2023-12-20', 3),  (4, 8, 15, '2023-12-22', 5),  (5, 8, 15, '2023-12-22', 5),  (6, 8, 15, '2023-12-22', 5),  (13, 8, 15, '2023-12-22', 5),  (15, 8, 15, '2023-12-22', 5),  (16, 8, 15, '2023-12-22', 5),  (17, 8, 15, '2023-12-22', 5),  (4, 5, 18, '2024-01-15', 5),  (5, 5, 18, '2024-01-15', 3),  (6, 5, 18, '2024-01-15', 4),  (13, 5, 18, '2024-01-15', 5),  (15, 5, 18, '2024-01-15', 3),  (16, 5, 18, '2024-01-15', 4),  (17, 5, 18, '2024-01-15', 2),  (17, 5, 18, '2024-02-07', 4),  (1, 8, 15, '2023-12-18', 3),  (11, 8, 15, '2023-12-18', 4),  (10, 8, 15, '2023-12-18', 4),  (1, 1, 18, '2023-12-18', 3),  (11, 1, 18, '2023-12-18', 4),  (10, 1, 18, '2023-12-18', 4),  (1, 2, 11, '2024-01-18', 2),  (11, 2, 11, '2024-01-18', 2),  (10, 2, 11, '2024-01-18', 2),  (1, 2, 11, '2024-02-18', 3),  (11, 2, 11, '2024-02-10', 3),  (10, 2, 11, '2024-03-03', 4),  (1, 2, 11, '2024-06-10', 4),  (11, 2, 11, '2024-06-10', 4),  (10, 2, 11, '2024-06-10', 4),  (1, 5, 18, '2024-06-16', 3),  (11, 5, 18, '2024-06-16', 3),  (10, 5, 18, '2024-06-16', 4),  (7, 1, 2, '2024-06-10', 5),  (8, 1, 2, '2024-06-10', 4),  (9, 1, 2, '2024-06-10', 4),  (7, 9, 16, '2024-06-22', 5),  (8, 9, 16, '2024-06-22', 5),  (9, 9, 16, '2024-06-22', 5) ; |

Приложение Б. Текст запроса на создание таблицы short\_db\_session

|  |
| --- |
| DROP DATABASE IF EXISTS short\_db\_session(FORCE);  CREATE DATABASE short\_db\_session;  CREATE TABLE student (  number int not null primary key,  full\_name varchar(30),  group\_number int,  birth\_year decimal(4) ); CREATE TABLE subject (  number int not null primary key,  full\_name varchar(50),  hours\_volume int ); CREATE TABLE exam (  student int,  subject int,  date\_of\_exam date,  mark decimal(1),  primary key (student, subject, date\_of\_exam),  foreign key (student)  references student(number)  on delete cascade,  foreign key (subject)  references subject(number)  on delete cascade ); INSERT INTO student VALUES  (1,'Иванов И.И.',140,2003),  (2,'Петров П.П.',140,2003),  (3,'Сидоров С.С.',145,2003),  (4,'Кузнецов К.К.',145,2004); INSERT INTO subject VALUES  (1,'Математика',100),  (2,'Физика',200),  (3,'Программирование',200) ; INSERT INTO exam VALUES  (1,1,'2021-06-10',5),  (1,2,'2021-06-20',4),  (1,3,'2021-06-30',3),  (2,1,'2021-06-10',5),  (2,2,'2021-06-20',5),  (3,1,'2021-06-10',2) ; |

Приложение В. Диаграмма связей таблиц базы данных stud\_session



Приложение Г. Диаграмма связей таблиц базы данных short\_db\_session

