ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе 6**

**по дисциплине: «Базы данных**

**Тема работы «Создание функции на языке PL/pgSQL. Создание триггеров»**

**“Обслуживание работы конференции”**

студента очного отделения

2 курса 12001801 группы

Капустина Виктора Сергеевича

Проверил(а):

Петров Денис Васильевич

Белгород 2020

**Цель работы:**

1. Научиться создавать функции при помощи языка PL/pgSQLпод управлением СУБД PostgreSQL в pgAdmin;  
2. Научится создавать триггеры при помощи языка SQL под управлением СУБД PostgreSQLв pgAdmin;

**Порядок выполнения работы:**

1. Написать функции (хранимые процедуры) на языке PL/pgSQL(не менее 6).

2. Функции (хранимые процедуры) на языке PL/pgSQL должны содержать операторы управления программой (*см. теоретический материал ниже*)

3. Среди созданных функций (хранимых процедур) должны быть процедуры с входными и выходными параметрами и процедуры, возвращающие набор записей.

4. В функциях использовать обработку исключений (не менее 2).

5. Написать триггеры DML предварительно обрабатывающие (проверяющие) данные перед выполнением предложения ввода, обновления и удаления информации (INSERT, UPDATE, DELETE) (не менее 3).

6. Написать триггеры для аудита изменения данных в таблице (не менее 3).

**7.** Составить электронный отчет о проделанной работе.

**Теоретическая часть**

Процедурный язык PL/SQL позволяет использовать такие стандартные средства программирования, как циклы и анализ переменных.

Функции имеют следующий синтаксис:

**CREATE FUNCTION имя(параметры) RETURNS тип AS $$**

**[<<метка\_основного\_блока>>]**

**DECLARE /\* объявления \* /**

**…**

**BEGIN /\* основной блок \* /**

**… ; [<<метка\_вложенного\_блока>>]**

**DECLARE /\* объявления \* /**

**… ;**

**BEGIN /\* вложенный блок \* /**

**… ;**

**END [метка\_вложенного\_блока];**

**… ;**

**END [метка\_основного\_блока] ;**

**$$ LANGUAGE plpgsql;**

**Операторы управления.**

В PL/SQL используется условный оператор IF. Его синтаксис:  
IF <условие> THEN <тело условного оператора> END IF.  
Также используются следующие циклические операторы:  
Loop - безусловный цикл, While - цикл с предусловием, For - цикл со счётчиком, Exit - выход из цикла, Continue - переход на следующую итерацию.  
Для вывода сообщения используется ключевое слово RAISE.

**Обработка ошибок в PL/pgSQL функциях**

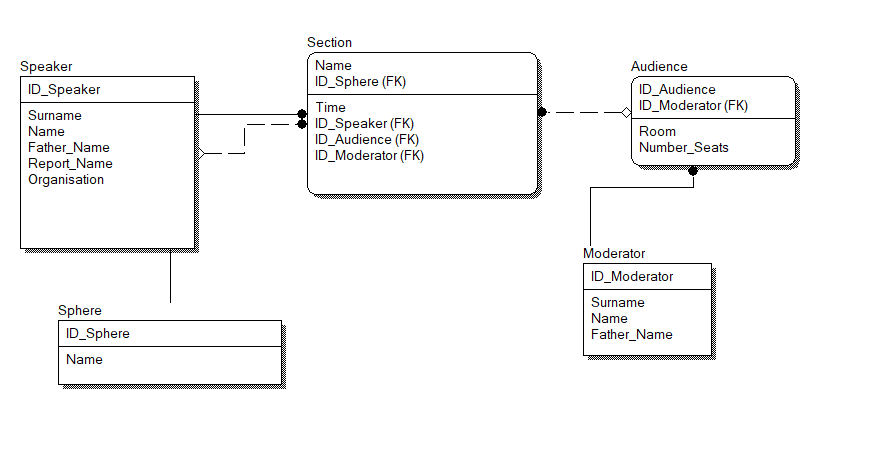
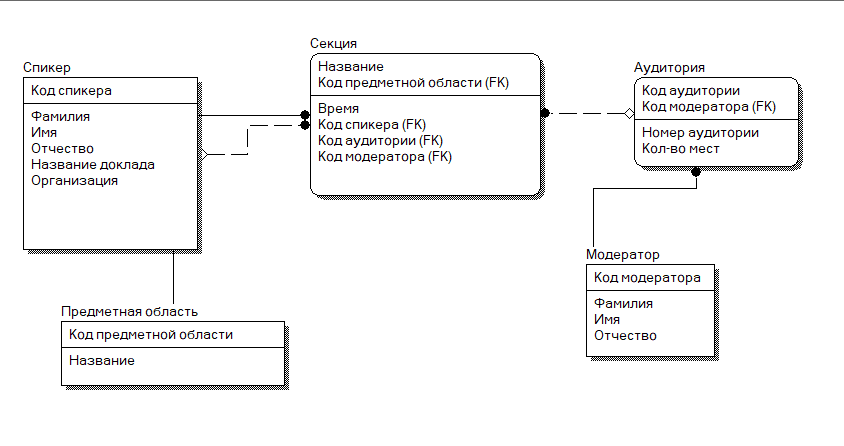
Ошибки, возникающие при выполнении команд в блоке, отслеживаются помощью предложения EXCEPTION Если возникает ошибка, то выполнение команд в блоке прерывается и управление передается в раздел EXCEPTION, где производится поиск первого из условий, удовлетворяющих возникшей ошибке, и запускается соответствующий обработчик. Далее программа выполняет первую операцию после END. Если ни одно из условий не удовлетворяет ошибке, обработка исключения передается первому из объемлющих блоков, в котором есть раздел EXCEPTION. Если такого блока нет, выполнение функции останавливается и выводится сообщение об ошибке.

**Триггеры.**

Триггеры - это тип функций, который вызывается автоматически при выполнении какой-либо инструкции манипулирования данными.

Триггеры бывают *for each row*(Для каждой строки) или *for each statement*(Для каждого вызова) Также триггерах используется две буферные переменные: new и old. New содержит новую строку, а Old содержит старую строку.

**Ход работы**

**Физическая ER модель**  
 **Логическая ER модель**

**Создание функций.**

**Функция 1**

Create function quantity\_of\_seats()

returns integer

AS

$$

Declare s integer := 0;

Begin

Select count(\*) into s from Audience where Number\_Seats > 70;

if s = 0 then return 0;

end if;

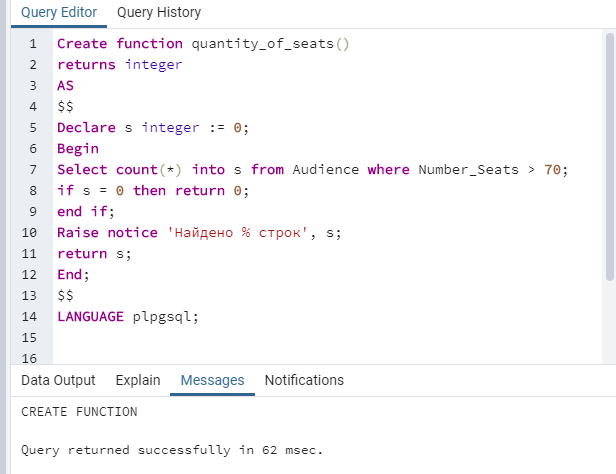
Raise notice 'Найдено % строк', s;

return s;

End;

$$

LANGUAGE plpgsql;



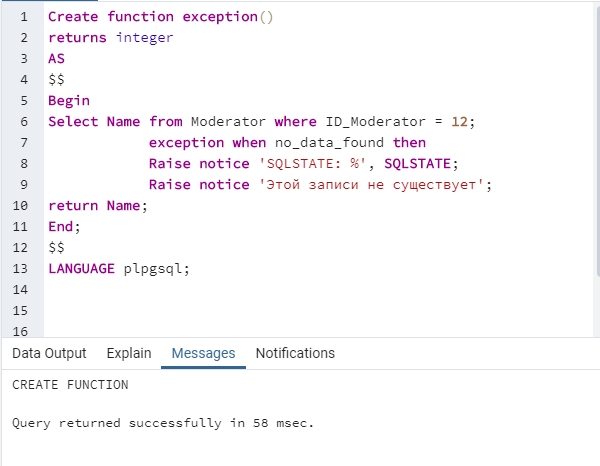
Функция выдает кол-во аудиторий, с кол-вом мест больше 70.  
**Функция 2**

Create function exception()  
returns integer  
AS  
$$  
Begin  
Select Name from Moderator where ID\_Moderator = 12;

exception when no\_data\_found then

Raise notice 'SQLSTATE: %', SQLSTATE;

Raise notice 'Этой записи не существует';  
End;  
$$  
LANGUAGE plpgsql;



Функция запрашивает на вывод имя Модератора с номеров 12, а при его отсутствии отловит ошибку о не нахождении данных. При этом выведутся данные об ошибке.

**Функция 3**

Create function exception(InName character(20))

returns integer

AS

$$

Declare s integer :=0;

Begin

Select count(\*) into s from Speaker where Name = InName;

if s= 0 then return 0;

end if;

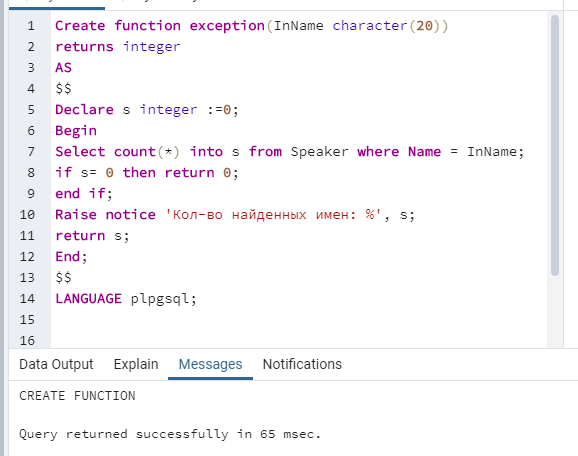
Raise notice 'Кол-во найденных имен: %', s;

return s;

End;

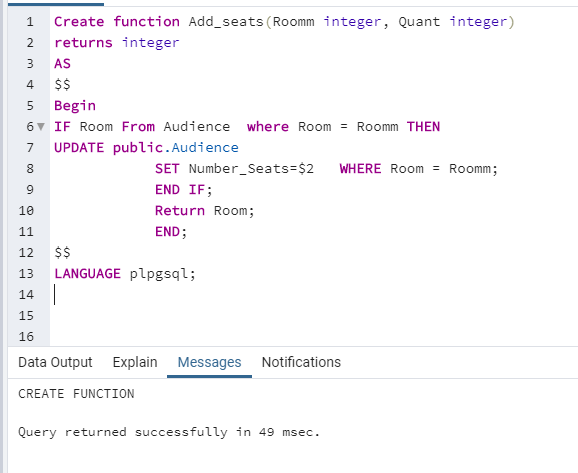
$$

LANGUAGE plpgsql;



**Функция 4**

Create function Add\_seats(Roomm integer, Quant integer)  
returns integer  
AS  
$$  
Begin  
IF Audience.room = Roomm THEN  
UPDATE public.Audience  
 SET Number\_Seats=$2 WHERE ID\_Audience = Roomm;  
 END IF;  
 Return Room;  
 END;  
$$  
LANGUAGE plpgsql;



**Функция 5**

Create function while(integer)

returns integer

AS

$$

Declare s integer := $1;

Begin

if s = 0 then return 0;

end if;

while s != 0 loop

Perform Surname, Name from Speaker where Organisation = NULL and ID\_Speaker = s;

s := s - 1;

end loop;

return s;

End;

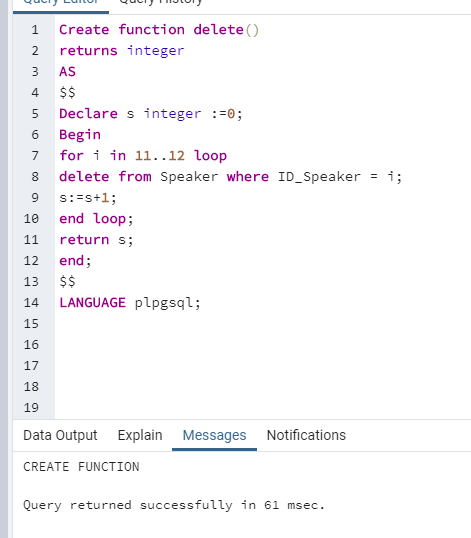
$$

LANGUAGE plpgsql;



В данной функции мы находим Спикеров, которые не указали организацию. Для этого создается переменная s, которая циклом while проходит по таблице и сравнивает условие.

**Функция 6**

Create function delete()  
returns integer  
AS  
$$  
Declare s integer :=0;  
Begin  
for i in 11..12 loop  
delete from Speaker where ID\_Speaker = i;  
s:=s+1;  
end loop;  
return s;  
end;  
$$  
LANGUAGE plpgsql;  
  
Функция проходит по таблице и удаляет записи ID\_Speaker которых равно i. i в данном примере переменная используемого цикла for

**Триггеры.**

**Триггеры, проверяющие данные перед операциями с информацией**

**Триггер 1**

CREATE FUNCTION Check\_Room()

RETURNS trigger

AS

$$

BEGIN

IF NEW.Room IS NULL THEN

RAISE EXCEPTION '% не может быть NULL', NEW.Room;

END IF;

IF NEW.Number\_Seats IS NULL THEN

RAISE EXCEPTION '% не может быть NULL', NEW.Number\_Seats;

END IF;

RETURN NEW;

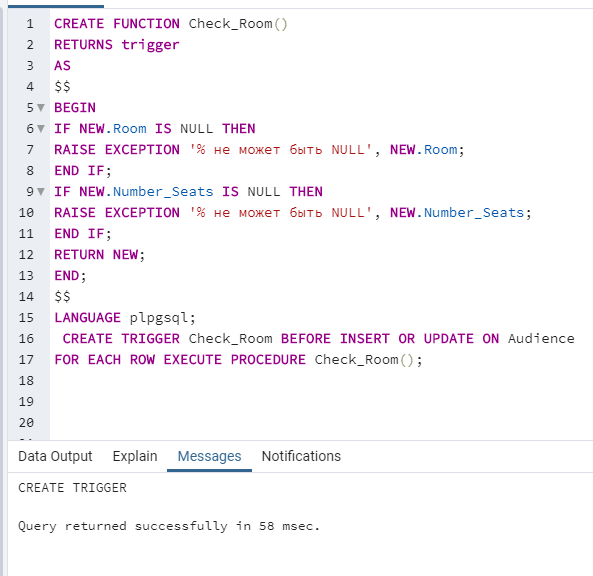
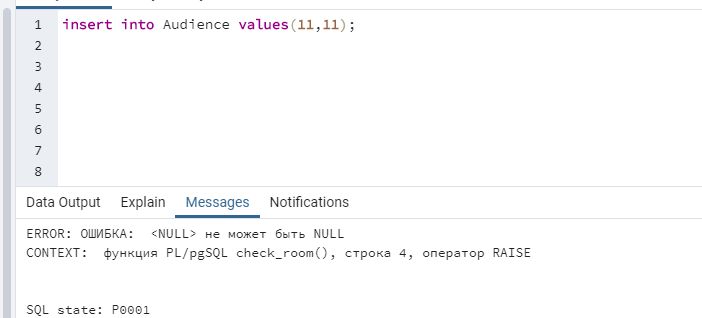
END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER Check\_Room BEFORE INSERT OR UPDATE ON Audience

FOR EACH ROW EXECUTE PROCEDURE Check\_Room();

  
  
В данной триггерной функции проверяется на NULL входные значение в поля Room, Number\_Seats. Если проверка словила такие данные, то выведется исключение.

**Триггер 2**

CREATE FUNCTION check\_seats()

RETURNS trigger

AS

$$

BEGIN

IF NEW.Number\_seats < 0 THEN

RAISE EXCEPTION '% не может быть меньше нуля', NEW.Number\_Seats;

END IF;

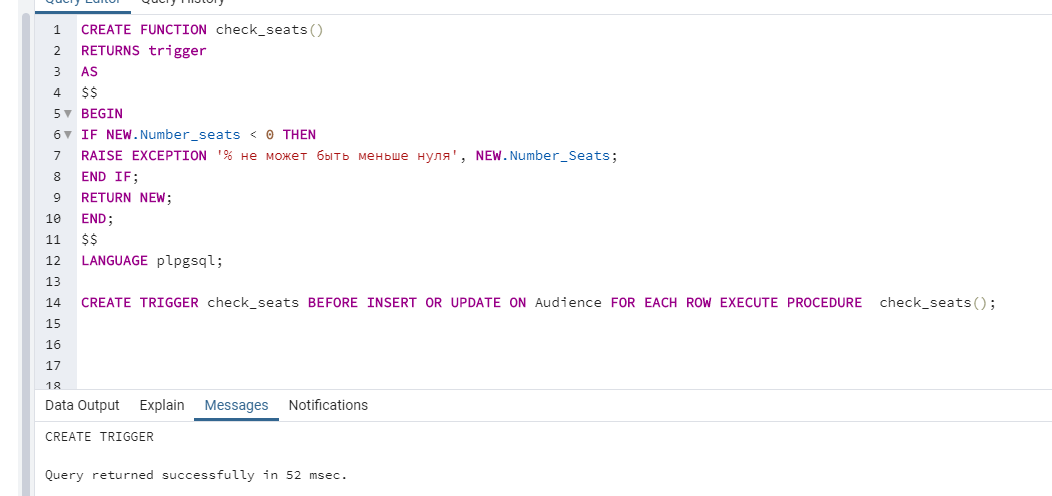
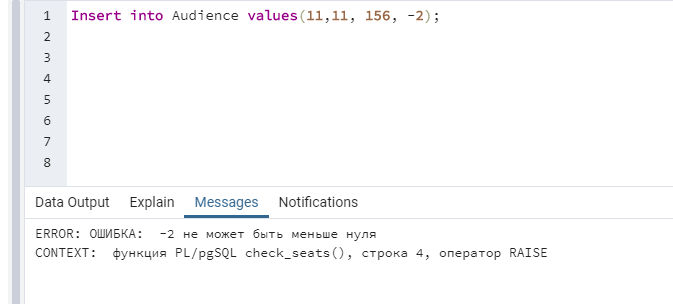
RETURN NEW;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER check\_seats BEFORE INSERT OR UPDATE ON Audience FOR EACH ROW EXECUTE PROCEDURE check\_seats();

  
  
В данной триггерной функции проверяются входные значение в поле Number\_Seats. Если они отрицательны, то выведется исключение.

**Триггер 4**

CREATE FUNCTION delete\_seats()

RETURNS trigger

AS

$$

BEGIN

IF OLD.Number\_Seats < 20 THEN

DELETE FROM Audience WHERE OLD.Number\_Seats <20;

END IF;

RETURN OLD;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER delete\_seats BEFORE DELETE ON Audience FOR EACH ROW EXECUTE PROCEDURE delete\_seats();

  
В данной триггерной функции проверяются входные значение в поле Number\_Seats. Если мест меньше 20, то произведет удаление.

**Триггеры для аудита изменения данных в таблице.**

**Логи DELETE**

CREATE TABLE logs\_audience(ID\_edit serial NOT NULL PRIMARY KEY, ID\_user integer NOT NULL, Operation varchar(12), Name\_of\_table text NOT NULL, DateTime\_edit TimeStamp NOT NULL);

CREATE FUNCTION delete\_log()

RETURNS trigger

AS

$$

BEGIN

IF (TG\_OP = 'DELETE') THEN

INSERT INTO logs\_audience (ID\_edit, ID\_user, Operation, Name\_of\_table, DateTime\_edit) SELECT user, 'Удалил', TG\_TABLE\_NAME, now();

RETURN OLD;

END IF;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER delete\_log AFTER DELETE ON logs\_audience

FOR EACH ROW EXECUTE PROCEDURE delete\_log();



В данной триггерной функции создается таблица для хранения логов изменения таблицы Audience. При операции DELETE занесутся данные об этой операции (ID изменения, ID пользователя, название операции, имя изменяемой таблицы и дата и время)

**Логи UPDATE**

CREATE FUNCTION update\_Audience()

RETURNS trigger

AS

$$

BEGIN

IF (TG\_OP = 'UPDATE') THEN

INSERT INTO logs\_Audience (ID\_edit, ID\_user, Operation, Name\_of\_table, DateTime\_edit) SELECT user, 'Обновил', TG\_TABLE\_NAME, now();

RETURN OLD;

END IF;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER update\_Audience AFTER UPDATE ON logs\_Audience

FOR EACH ROW EXECUTE PROCEDURE update\_Audience();

  
При операции UPDATE занесутся данные об этой операции (ID изменения, ID пользователя, название операции, имя изменяемой таблицы и дата и время)

CREATE FUNCTION insert\_Audience()

RETURNS trigger

AS

$$

BEGIN

IF (TG\_OP = 'INSERT') THEN

INSERT INTO logs\_Audience (ID\_edit, ID\_user, Operation, Name\_of\_table, DateTime\_edit) SELECT user, 'Добавил', TG\_TABLE\_NAME, now();

RETURN OLD;

END IF;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER insert\_Audience AFTER INSERT ON logs\_Audience

FOR EACH ROW EXECUTE PROCEDURE insert\_Audience();

  
При операции INSERT занесутся данные об этой операции (ID изменения, ID пользователя, название операции, имя изменяемой таблицы и дата и время)