**Министерство науки и высшего образования Российской Федерации**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**ОТЧЕТ**

**по Лабораторной работе № 5**

**«ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В POSTGRESQL»**

**по дисциплине «Проектирование и реализация баз данных»**

**Обучающийся** Проскуряков Роман Владимирович

**Факультет** прикладной информатики

**Группа** K3239

**Направление подготовки** 09.03.03 Прикладная информатика

**Образовательная программа** Мобильные и сетевые технологии 2023

**Преподаватель** Говорова Марина Михайловна

Санкт-Петербург

2024/2025

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

**Практическое задание:**

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)
2. Создать триггеры для индивидуальной БД согласно варианту:

7 оригинальных триггеров - 7 баллов (max).

**Часть 4 ЛР 2**.Создать хранимые процедуры:

* Вывести сведения обо всех покупках одного из клиентов за заданную дату (данные клиента, дата, объем топлива, уплаченная сумма).
* Посчитать количество видов топлива, поставляемых каждой фирмой-поставщиком.
* Добавить новую АЗГС фирмы-производителя.

**Выполнение:**

1. Создать хранимые процедуры

* Вывести сведения обо всех покупках одного из клиентов за заданную дату (данные клиента, дата, объем топлива, уплаченная сумма)

CREATE OR REPLACE function clientPurchasesPerDay (IN idClient INT, IN dayDate DATE)

RETURNS TABLE(surname CHARACTER VARYING, name CHARACTER VARYING, patronymic CHARACTER VARYING,

phone\_number bigint, address CHARACTER VARYING,

sale\_date timestamp without time zone, sold\_liters\_volume int, id\_card int, money\_spent int)

LANGUAGE plpgsql

AS $$

BEGIN

RAISE NOTICE 'Данные о покупках клиента с ID: % за: %', idClient, dayDate;

RETURN QUERY

SELECT clients.surname, clients.name, clients.patronymic,

clients.phone\_number, clients.address,

sales.sale\_date, sales.sold\_liters\_volume, sales.id\_card,

((sales.sold\_liters\_volume \* fuel\_prices.per\_liter) \* (100 - discount\_percent) / 100 - discount\_rub) AS money

FROM

clients

JOIN client\_cards

ON clients.id\_client=client\_cards.id\_client

JOIN sales

ON client\_cards.id\_card=sales.id\_card

JOIN fuel\_prices

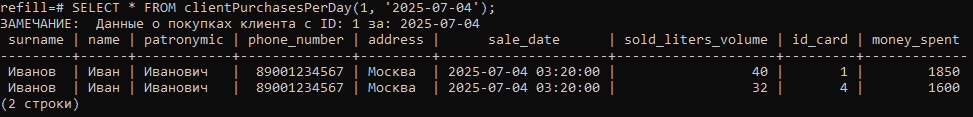
ON sales.id\_fuel\_price=fuel\_prices.id\_fuel\_price

WHERE clients.id\_client=idClient

AND sales.sale\_date::timestamp::date = dayDate;

END;

$$;



* Посчитать количество видов топлива, поставляемых каждой фирмой-поставщиком.

CREATE OR REPLACE procedure fuelTypesCountBySupplier(INOUT \_\_resultTable refcursor)

LANGUAGE plpgsql

AS $$

BEGIN

OPEN \_\_resultTable FOR

SELECT companies.id\_company, count(id\_kind\_fuel) as amount

FROM companies

LEFT JOIN produced\_fuel

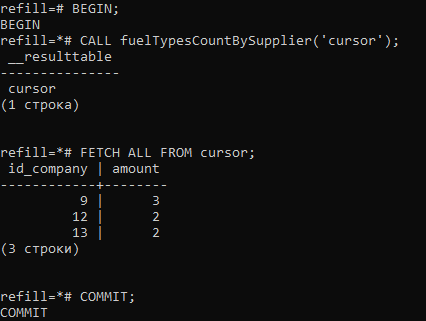
ON companies.id\_company=produced\_fuel.id\_company\_factory

WHERE type\_company=1

GROUP BY id\_company;

END;

$$;



* Добавить новую АЗГС фирмы-производителя.

CREATE OR REPLACE procedure addFillingStations(idCompanyOwner int, address CHARACTER VARYING)

LANGUAGE plpgsql

AS $$

BEGIN

IF 0 not in (SELECT type\_company FROM companies WHERE id\_company=idCompanyOwner)

THEN

RAISE NOTICE 'Компания с ID % не является компанией владельцем', idCompanyOwner;

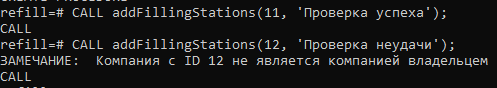
RETURN;

END IF;

INSERT INTO filling\_stations(id\_company\_owner, station\_address) VALUES(idCompanyOwner, address);

END;

$$;



1. Создать 7 оригинальных триггеров для индивидуальной БД
2. Триггер 1 – проверка, что цена соответствует времени продажи

create or replace function fn\_check\_fuel\_price\_is\_actual() returns

trigger as $$

begin

if (EXISTS

(

SELECT id\_fuel\_price FROM fuel\_prices

WHERE new.id\_fuel\_price=fuel\_prices.id\_fuel\_price AND

(start\_time<=new.sale\_date and (end\_time IS NULL OR new.sale\_date<end\_time))

)

) then

return new;

end if;

return null;

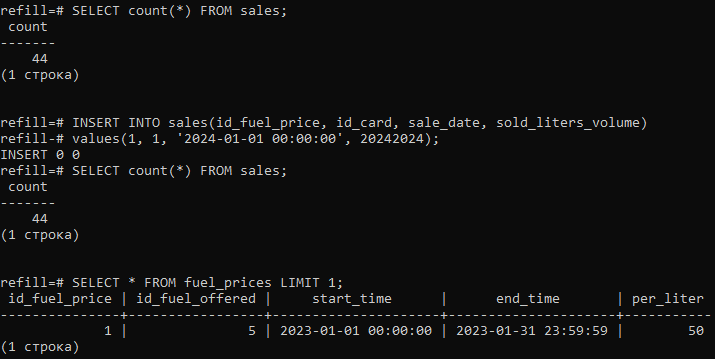
end;

$$ language plpgsql;

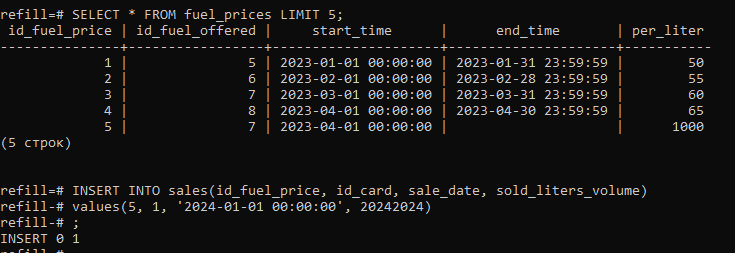
create or replace trigger check\_fuel\_price\_is\_actual before insert or update on sales

for each row execute procedure fn\_check\_fuel\_price\_is\_actual();

Не вставилось



Вставилось



1. Триггер 2 – нельзя совершить покупку по недействительной карте

create or replace function fn\_check\_validity\_client\_card() returns

trigger as $$

begin

if (Exists(

select 1 from client\_cards

where client\_cards.id\_card=new.id\_card

and new.sale\_date >= start\_date

and (end\_date is NULL or new.sale\_date < end\_date))

) then

return new;

end if;

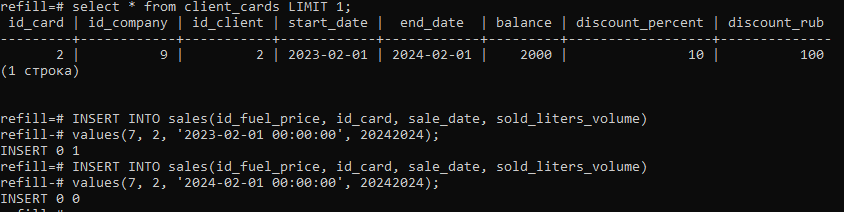
return null;

end;

$$ language plpgsql;

create or replace trigger check\_validity\_client\_card before insert or update on sales

for each row execute procedure fn\_check\_validity\_client\_card();



1. Триггер 3 – изменять в карте можно только владельца (человека и даже компанию), баланс и двигать конец действия в рамках будущего

create or replace function fn\_check\_client\_card\_changes() returns

trigger as $$

begin

if (new.start\_date!=old.start\_date

or new.discount\_rub!=old.discount\_rub

or new.discount\_percent!=old.discount\_percent

or (new.end\_date is not null and new.end\_date <= current\_date)

) then

return null;

end if;

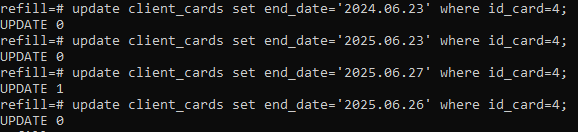
return new;

end;

$$ language plpgsql;

create or replace trigger check\_client\_card\_changes before update on client\_cards

for each row execute procedure fn\_check\_client\_card\_changes();



1. Триггер 4 – новая цена на топливо не пересекается по времени с другими на этот же товар (не должно быть двух одновременно действующих цен на одно продаваемое топливо)

create or replace function fn\_check\_uniqueness\_of\_price() returns

trigger as $$

begin

if(exists(

select 1 from fuel\_prices

where fuel\_prices.id\_fuel\_offered=new.id\_fuel\_offered

and start\_time>=new.start\_time

)) then

return null;

end if;

CREATE TEMP TABLE old\_actings\_prices(id\_fuel\_price int);

INSERT INTO old\_actings\_prices select id\_fuel\_price from fuel\_prices

where fuel\_prices.id\_fuel\_offered=new.id\_fuel\_offered

and ((end\_time is null) or

(end\_time>new.start\_time));

if(exists(SELECT \* FROM old\_actings\_prices))

then

update fuel\_prices set end\_time=new.start\_time

where fuel\_prices.id\_fuel\_price in (SELECT \* FROM old\_actings\_prices);

end if;

DROP TABLE old\_actings\_prices;

return new;

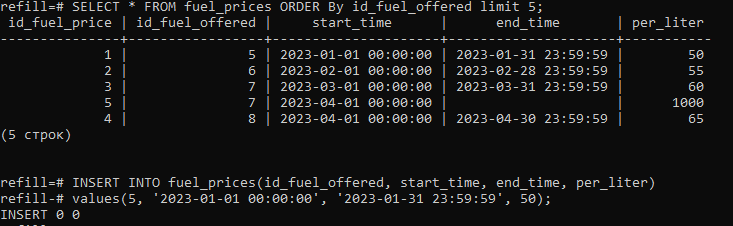
end;

$$ language plpgsql;

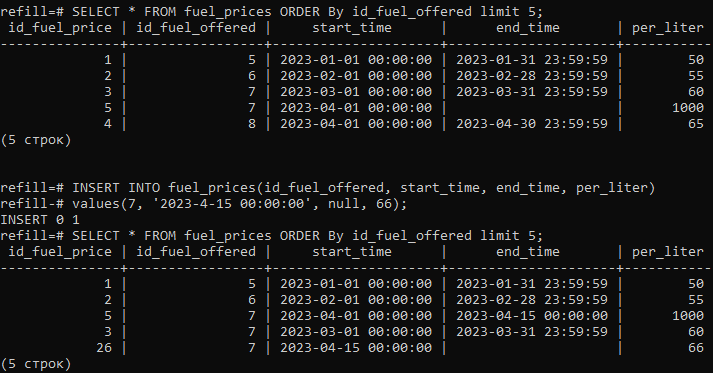
create or replace trigger check\_uniqueness\_of\_price before insert on fuel\_prices

for each row execute procedure fn\_check\_uniqueness\_of\_price();

Ничего не добавилось



Предыдущая цена автоматически завершилась на начале новой добавленной



1. Триггер 5 – у цены можно менять только дату окончания и только на будущее значение

create or replace function fn\_check\_price\_changes() returns

trigger as $$

begin

if (new.start\_time!=old.start\_time or

new.per\_liter!=old.per\_liter or

(new.end\_time is not null and new.end\_time < NOW()))

then

return null;

end if;

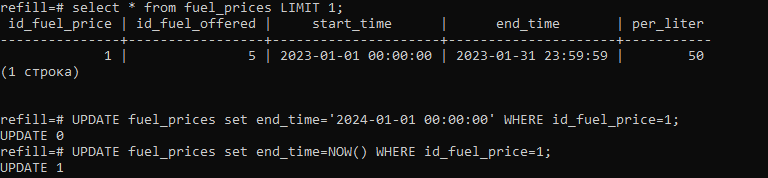
return new;

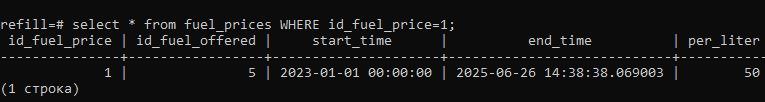
end;

$$ language plpgsql;

create or replace trigger check\_price\_changes before update on fuel\_prices

for each row execute procedure fn\_check\_price\_changes();





1. Триггер 6 – у производимого топлива компания должна быть компанией-поставщиком

create or replace function fn\_check\_produced\_fuel\_company() returns

trigger as $$

begin

if (EXISTS(

SELECT 1 FROM companies WHERE companies.id\_company=new.id\_company\_factory AND type\_company=1

)) then

return new;

end if;

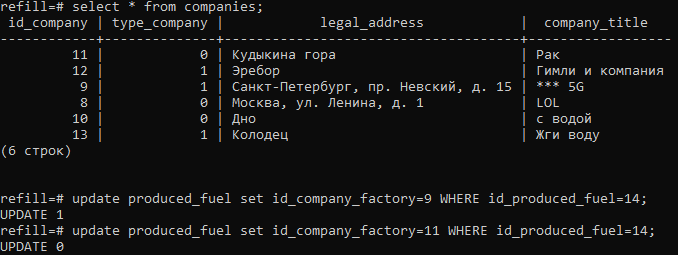
return null;

end;

$$ language plpgsql;

create or replace trigger check\_produced\_fuel\_company before insert or update on produced\_fuel

for each row execute procedure fn\_check\_produced\_fuel\_company();



1. Триггер 7 – у заправок копания должна быть компанией-владельцем

create or replace function fn\_check\_filling\_stations\_company() returns

trigger as $$

begin

if (EXISTS(

SELECT 1 FROM companies WHERE companies.id\_company=new.id\_company\_owner AND type\_company=0

)) then

return new;

end if;

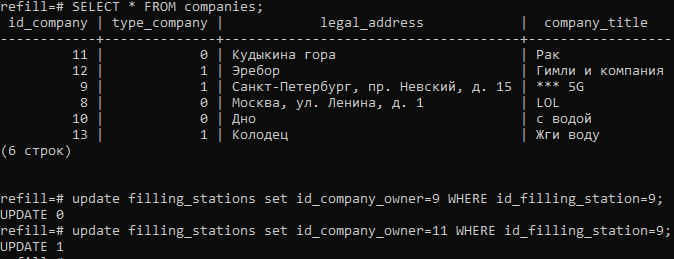
return null;

end;

$$ language plpgsql;

create or replace trigger check\_filling\_stations\_company before insert or update on filling\_stations

for each row execute procedure fn\_check\_filling\_stations\_company();



1. Триггер 8 – у карточек клиентов копания должна быть владельцем компанией-владельцем

create or replace function fn\_check\_client\_cards\_company() returns

trigger as $$

begin

if (EXISTS(

SELECT 1 FROM companies WHERE companies.id\_company=new.id\_company AND type\_company=0

)) then

return new;

end if;

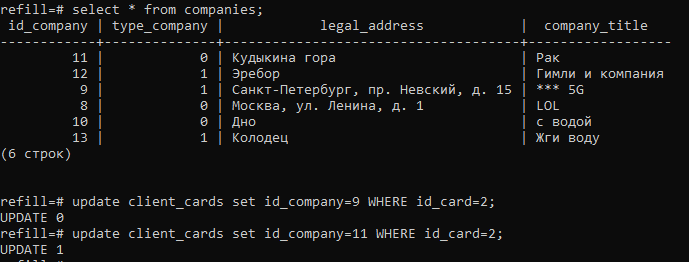
return null;

end;

$$ language plpgsql;

create or replace trigger check\_client\_cards\_company before insert or update on client\_cards

for each row execute procedure fn\_check\_client\_cards\_company();

****

**Выводы**

Триггеры – это круто. Они позволяют автоматизировать то, без чего в базе данных была бы куча ошибок, например, сложные проверки, автоматическое обновление всех зависящих столбцов.

Все внешние ключи называть так же, как и первичный на который они ссылаются. Так не придётся реализовывать отдельный триггер для каждого такого нового названия ключа как в триггерах 6, 7, 8.