

Санкт-Петербург 2024

Цель работы

Цель работы: Получение умений и навыков по проектированию и созданию триггеров баз данных, включая их использование для поддержания активной ссылочной целостности.

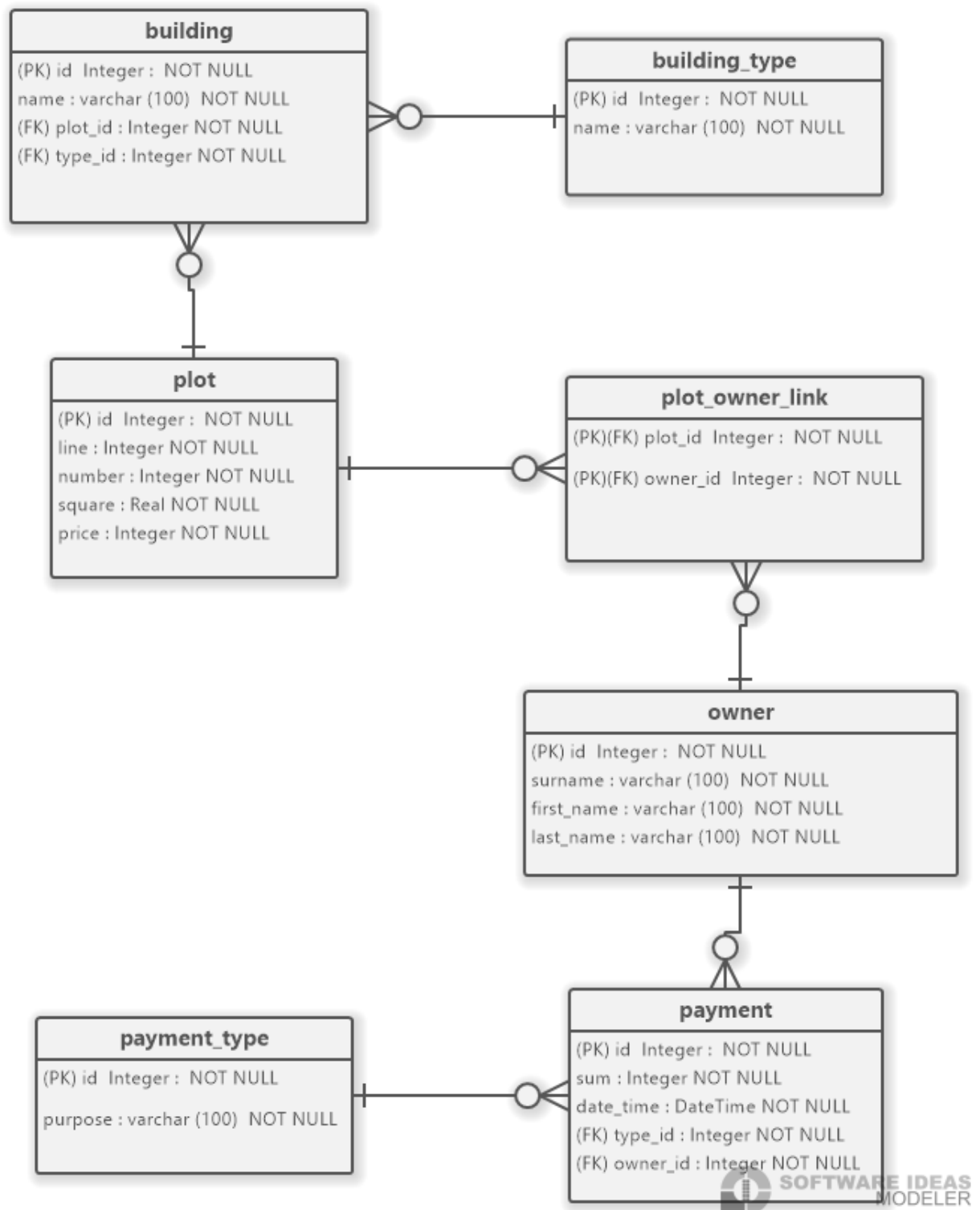
Задание

Реализовать для своей базы данных триггеры для всех событий (insert, delete, update) до и после. (6 триггеров) Часть из которых будет обеспечивать ссылочную целостность, остальные могут иметь другое назначение из других предложенных, но не менее 2 различных.

- Вычисление/поддержание в актуальном состоянии вычисляемых (производных) атрибутов
- логирование (запись) изменений;
- проверка корректности проводимых действий.).

Вычисляемые поля можно добавить при необходимости.

Физическая модель БД



Назначение разработанных триггеров текстом

1. Триггер перед вставкой платежа: проверка на наличие платежа с совпадающим владельцем, типом, суммой и временем совершения
2. Триггер после вставки платежа: логирование о совершении платежа
3. Триггер до обновления участка: обновление ссылок в таблице связи владельца с участками и в таблице построек (ссылочная целостность)
4. Триггер после обновления владельца: проверка на минимальную длину имени и фамилии
5. Триггер до удаления участка: очистка данных в таблицах построек и связи владельца с участком (ссылочная целостность)
6. Триггер после удаления постройки: логирование о возникновении пустых участков (на которых не расположено ни одной постройки)

Скрипт для создания триггеров

```
--
-- TOC entry 251 (class 1255 OID 16515)
-- Name: building_after_delete(); Type: FUNCTION; Schema: public; Owner: postgres
--

CREATE FUNCTION public.building_after_delete() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
    DECLARE
        building_count integer;
    BEGIN

        select count(*) from building b
        where b.plot_id = OLD.plot_id
        into building_count;

        if (building_count = 0) then

            RAISE INFO 'There are no buildings left on the plot № %',
OLD.plot_id;

        end if;

        return OLD;

    END;
$$;

ALTER FUNCTION public.building_after_delete() OWNER TO postgres;

--
-- TOC entry 238 (class 1255 OID 16510)
-- Name: owner_after_update(); Type: FUNCTION; Schema: public; Owner: postgres
--

CREATE FUNCTION public.owner_after_update() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
    BEGIN

        if (length(NEW.surname) < 1) then

            RAISE EXCEPTION 'The "surname" field must be filled';

        end if;

        if (length(NEW.first_name) < 1) then

            RAISE EXCEPTION 'The "first_name" field must be filled';

        end if;

        return NEW;

    END;
$$;
```

```
ALTER FUNCTION public.owner after update() OWNER TO postgres;
```

```
--  
-- TOC entry 235 (class 1255 OID 16500)  
-- Name: payment_after_insert(); Type: FUNCTION; Schema: public; Owner: postgres  
--
```

```
CREATE FUNCTION public.payment_after_insert() RETURNS trigger  
    LANGUAGE plpgsql  
    AS $$  
    BEGIN  
  
        RAISE INFO 'Payment inserted: %', current_query();  
  
        return new;  
  
    END;  
$$;
```

```
ALTER FUNCTION public.payment after insert() OWNER TO postgres;
```

```
--  
-- TOC entry 234 (class 1255 OID 16493)  
-- Name: payment_before_insert(); Type: FUNCTION; Schema: public; Owner: postgres  
--
```

```
CREATE FUNCTION public.payment_before_insert() RETURNS trigger  
    LANGUAGE plpgsql  
    AS $$  
    declare  
        payment_count integer;  
    BEGIN  
  
        select count(*)  
        into payment_count  
        from public.payment p  
            where p.sum = new.sum  
                and p.date_time = new.date_time  
                and p.type_id = new.type_id  
                and p.owner_id = new.owner_id;  
  
        if (payment_count > 0) then  
            RAISE EXCEPTION 'duplicate payments found';  
        end if;  
  
        return new;  
  
    END;  
$$;
```

```
ALTER FUNCTION public.payment before insert() OWNER TO postgres;
```

```
--  
-- TOC entry 249 (class 1255 OID 16513)  
-- Name: plot_before_delete(); Type: FUNCTION; Schema: public; Owner: postgres  
--
```

```
CREATE FUNCTION public.plot_before_delete() RETURNS trigger  
    LANGUAGE plpgsql  
    AS $$
```

```

BEGIN

    delete from building
    where plot_id = OLD.id;

    delete from plot_owner_link
    where plot_id = OLD.id;

    return OLD;
END;
$$;

ALTER FUNCTION public.plot before delete() OWNER TO postgres;

--
-- TOC entry 237 (class 1255 OID 16505)
-- Name: plot_before_update(); Type: FUNCTION; Schema: public; Owner: postgres
--

CREATE FUNCTION public.plot_before_update() RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN

    -- обновить связанные таблицы, если изменился идентификатор
    if (OLD.id != NEW.id) then

        ALTER TABLE building DISABLE TRIGGER ALL;

        update building
        set plot_id = NEW.id
        where plot_id = OLD.id;

        ALTER TABLE building ENABLE TRIGGER ALL;

        ALTER TABLE plot_owner_link DISABLE TRIGGER ALL;

        update plot_owner_link
        set plot_id = NEW.id
        where plot_id = OLD.id;

        ALTER TABLE plot_owner_link ENABLE TRIGGER ALL;

    end if;

    return NEW;
END;
$$;

ALTER FUNCTION public.plot before update() OWNER TO postgres;

--
-- TOC entry 4702 (class 2620 OID 16516)
-- Name: building after_delete; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER after_delete AFTER DELETE ON public.building FOR EACH ROW EXECUTE
FUNCTION public.building_after_delete();

```

```
--
-- TOC entry 4704 (class 2620 OID 16508)
-- Name: payment after_insert; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER after_insert AFTER INSERT ON public.payment FOR EACH ROW EXECUTE
FUNCTION public.payment_after_insert();

--
-- TOC entry 4703 (class 2620 OID 16511)
-- Name: owner after_update; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER after_update AFTER UPDATE ON public.owner FOR EACH ROW EXECUTE
FUNCTION public.owner_after_update();

--
-- TOC entry 4706 (class 2620 OID 16514)
-- Name: plot before_delete; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER before_delete BEFORE DELETE ON public.plot FOR EACH ROW EXECUTE
FUNCTION public.plot_before_delete();

--
-- TOC entry 4705 (class 2620 OID 16507)
-- Name: payment before_insert; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER before_insert BEFORE INSERT ON public.payment FOR EACH ROW EXECUTE
FUNCTION public.payment_before_insert();

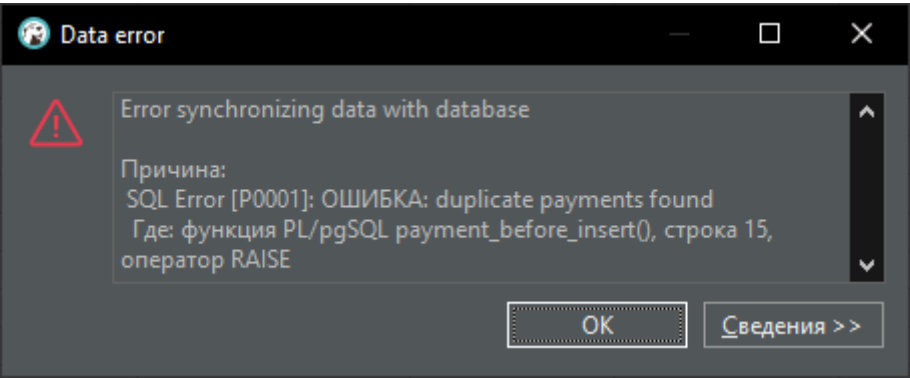
--
-- TOC entry 4707 (class 2620 OID 16506)
-- Name: plot before_update; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER before_update BEFORE UPDATE ON public.plot FOR EACH ROW EXECUTE
FUNCTION public.plot_before_update();
```


Скриншоты

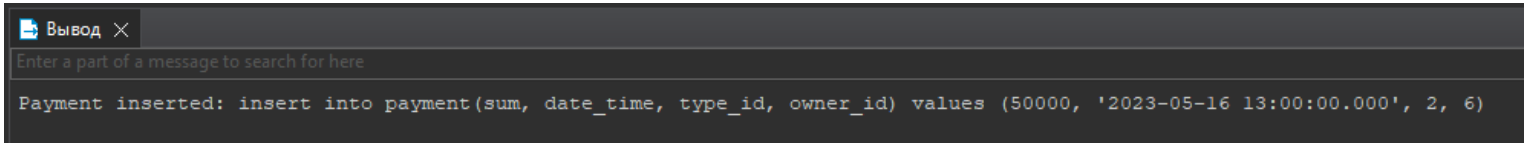
1. Проверка перед вставкой платежа на дубликаты

```
insert into payment(sum, date_time, type_id, owner_id) values (50000, '2023-05-16 13:00:00.000', 2, 6)
```



2. Логирование после вставки платежа

```
insert into payment(sum, date time, type id, owner id) values (50000, '2023-05-16 13:00:00.000', 2, 6);
```



3. Перед обновлением участка: обновление данных в связанных таблицах (контроль ссылочной целостности)

Таблица участков до выполнения обновления (участок 1)

	123 line	123 number	123 square	123 price	123 id
1	1	1	1 000	100 000 000	1
2	2	2	1 500	150 000 000	2
3	1	3	800	80 000 000	3
4	2	4	1 100	110 000 000	4
5	2	5	1 120	112 000 000	5
6	2	6	1 048	300 000 000	6

Связи участков с владельцами

123 plot_id	123 owner_id
1	1
2	2
3	3
4	4
5	5
5	6

Постройки на участках

123 id	A-Z name	123 plot_id	123 type_id	123 id_2
1	Главный дом	1	1	0
2	Туалет	1	2	0
3	Финская баня	1	3	0
5	Открытая беседка	1	5	0
6	Дом	2	1	0
7	Русская баня	2	3	0
8	Дом у дороги	3	1	0
9	Туалет	3	2	0
10	Дом из бруса	4	1	0
11	Баня	5	3	0
12	Дом кирпичный	5	1	0

Запрос обновления идентификатора участка

```
update plot set id = 11 where id = 1;
```

Обновленный участок:

123 line	123 number	123 square	123 price	123 id
2	2	1 500	150 000 000	2
1	3	800	80 000 000	3
2	4	1 100	110 000 000	4
2	5	1 120	112 000 000	5
2	6	1 048	300 000 000	6
1	1	1 000	100 000 000	11

Обновленные идентификаторы в таблице построек

123 id	A-Z name	123 plot_id	123 type_id	123 id_2
6	Дом	2	1	0
7	Русская баня	2	3	0
8	Дом у дороги	3	1	0
9	Туалет	3	2	0
10	Дом из бруса	4	1	0
11	Баня	5	3	0
12	Дом кирпичный	5	1	0
1	Главный дом	11	1	0
2	Туалет	11	2	0
3	Финская баня	11	3	0
5	Открытая беседка	11	5	0

Обновленные связи участков с владельцами

123 plot_id	123 owner_id
2	2
3	3
4	4
5	5
5	6
11	1

4. Проверка на заполнение данных имени и фамилии в таблице собственников (после обновления)

123 id	A-Z surname	A-Z first_name	A-Z last_name
1	Иванов	Дмитрий	Петрович
2	Петров	Иван	Дмитриевич
5	Иванов	Петр	Дмитриевич
4	Дмитриев	Иван	Алексеевич
6	Петров	Дмитрий	Петрович
3		Анастасия	Ивановна

Data error

Error synchronizing data with database

Причина:
SQL Error [P0001]: ОШИБКА: The "surname" field must be filled
Где: функция PL/pgSQL owner_after_update(), строка 6, оператор RAISE

OK
 Сведения >>

123 id	A-Z surname	A-Z first_name	A-Z last_name
1	Иванов	Дмитрий	Петрович
2	Петров	Иван	Дмитриевич
5	Иванов	Петр	Дмитриевич
4	Дмитриев	Иван	Алексеевич
6	Петров	Дмитрий	Петрович
3	Дмитриева		Ивановна

Data error

Error synchronizing data with database

Причина:
SQL Error [P0001]: ОШИБКА: The "first_name" field must be filled
Где: функция PL/pgSQL owner_after_update(), строка 12, оператор RAISE

OK
 Сведения >>

5. Очистка данных перед удалением участка: очистка данных в таблице связи собственников с участками и удаление построек (выполнение до удаления)

Состояние данных до удаления

123 line	123 number	123 square	123 price	123 id
2	2	1 500	150 000 000	2
1	3	800	80 000 000	3
2	4	1 100	110 000 000	4
2	5	1 120	112 000 000	5
2	6	1 048	300 000 000	6
1	1	1 000	100 000 000	11
3	1	1 000	100 000 000	7

123 plot_id	123 owner_id
2	2
3	3
4	4
5	5
5	6
11	1
7	1

123 id	A-Z name	123 plot_id	123 type_id
6	Дом	2	1
7	Русская баня	2	3
8	Дом у дороги	3	1
9	Туалет	3	2
10	Дом из бруса	4	1
11	Баня	5	3
12	Дом кирпичный	5	1
1	Главный дом	11	1
2	Туалет	11	2
3	Финская баня	11	3
5	Открытая беседка	11	5
21	Открытая беседка	7	5

Запрос удаления

```
delete from plot where id = 7;
```

Состояние данных после удаления

123 line	123 number	123 square	123 price	123 id
2	2	1 500	150 000 000	2
1	3	800	80 000 000	3
2	4	1 100	110 000 000	4
2	5	1 120	112 000 000	5
2	6	1 048	300 000 000	6
1	1	1 000	100 000 000	11

123 plot_id	123 owner_id
2	2
3	3
4	4
5	5
5	6
11	1

123 id	A-Z name	123 plot_id	123 type_id
6	Дом	2	1
7	Русская баня	2	3
8	Дом у дороги	3	1
9	Туалет	3	2
10	Дом из бруса	4	1
11	Баня	5	3
12	Дом кирпичный	5	1
1	Главный дом	11	1
2	Туалет	11	2
3	Финская баня	11	3
5	Открытая беседка	11	5

6. После удаления: логирование о возникновении пустых участков при удалении постройки

Единственная на участке постройка (участок 4)

123 id	A-Z name	123 plot_id	123 type_id
6	Дом	2	1
7	Русская баня	2	3
8	Дом у дороги	3	1
9	Туалет	3	2
10	Дом из бруса	4	1
11	Баня	5	3
12	Дом кирпичный	5	1
1	Главный дом	11	1
2	Туалет	11	2
3	Финская баня	11	3
5	Открытая беседка	11	5

Запрос на удаление

```
delete from building where id = 10;
```

Вывод в лог сообщения о пустом участке:

Статистика 1	Вывод ×
Enter a part of a message to search for here	
There are no buildings left on the plot № 4	

Выводы

В ходе выполнения контрольной работы были получены навыки по созданию триггеров для СУБД PostgreSQL. Созданы следующие типы триггеров: триггеры для контроля ссылочной целостности, триггеры для контроля корректности данных, триггеры для логирования совершения операций.

Работа триггеров проверена запросами и отражена в отчете.

Получены знания о нюансах работы с триггерами в СУБД PostgreSQL, такими как создание триггеров на основе функций, необходимости возврата из функции неявно определенных кортежей для выполнения либо отмены выполнения операции, необходимости отключения встроенных триггеров СУБД для обхода проверок внешних ограничений.