МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | М.В. Величко |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| КОНТРОЛЬНАЯ РАБОТА |
| ТРИГГЕРЫ. ОБЕСПЕЧЕНИЕ АКТИВНОЙ ЦЕЛОСТНОСТИ ДАННЫХ БАЗЫ ДАННЫХ |
| по дисциплине: Проектирование баз данных |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 |  |  |  | М.Д.Быстров |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2021/3572 | |  |  |  |

|  |  |
| --- | --- |
| Шифр ИНДО |  |

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 .

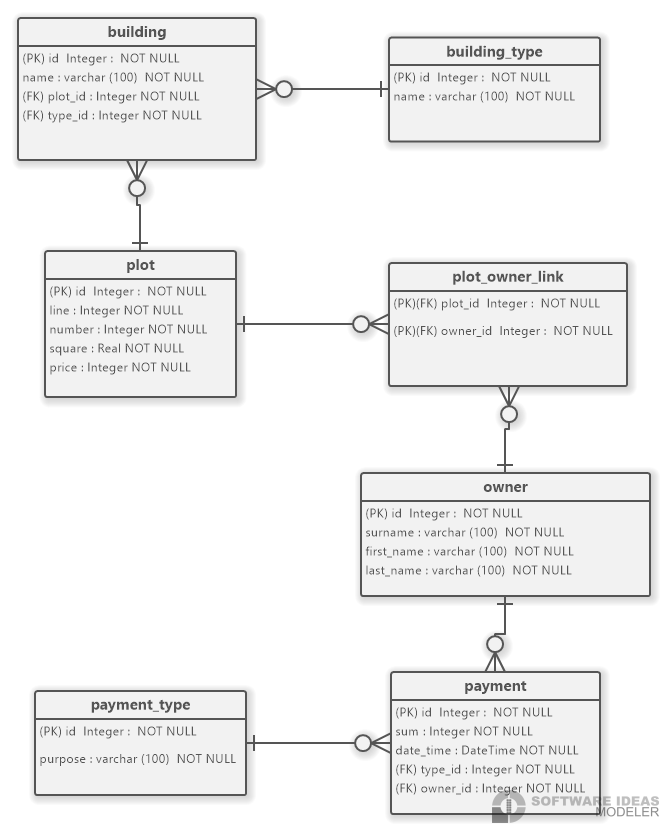
- Вычисление/поддержание в актуальном состоянии вычислимых (производных) атрибутов

- логирование (запись) изменений;

- проверка корректности проводимых действий.).

Вычислимые 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

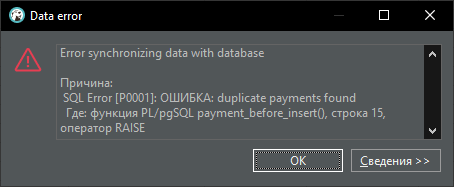
--

**CREATE** **TRIGGER** before\_update **BEFORE** **UPDATE** **ON** public.plot **FOR** **EACH** **ROW** **EXECUTE** **FUNCTION** public.plot\_before\_update();

Скриншоты

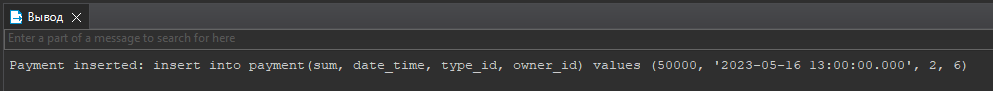
1. Проверка перед вставкой платежа на дубликаты

**insert** **into** payment(sum, date\_time, type\_id, owner\_id) **values** (50000, '2023-05-16 13:00:00.000', 2, 6)

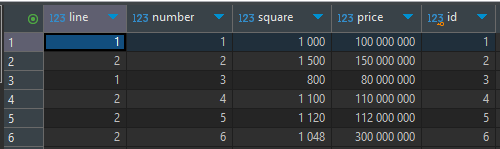


2.Логирование после вставки платежа

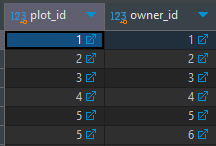
**insert** **into** payment(sum, date\_time, type\_id, owner\_id) **values** (50000, '2023-05-16 13:00:00.000', 2, 6);



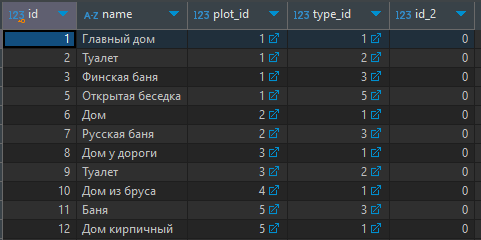
3. Перед обновлением участка: обновление данных в связанных таблицах (контроль ссылочной целостности)  
 Таблица участков до выполнения обновления (участок 1)



Связи участков с владельцами



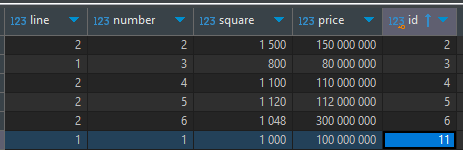
Постройки на участках



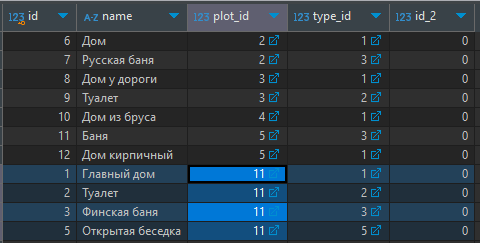
Запрос обновления идентификатора участка

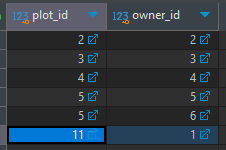
**update** plot **set** id = 11 **where** id = 1;

Обновленный участок:

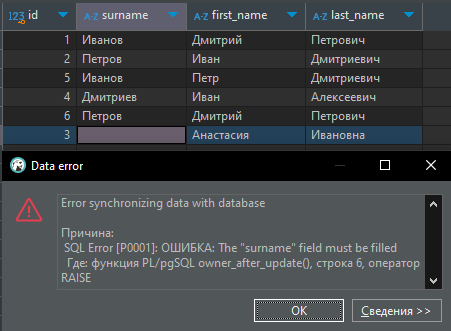


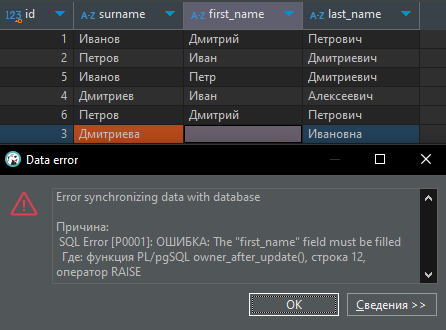
Обновленные идентификаторы в таблице построек



Обновленные связи участков с владельцами  


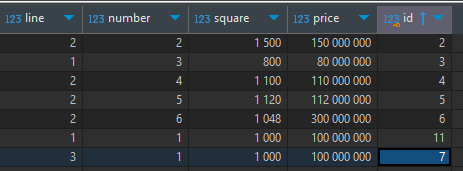
4. Проверка на заполнение данных имени и фамилии в таблице собственников (после обновления)

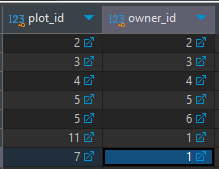


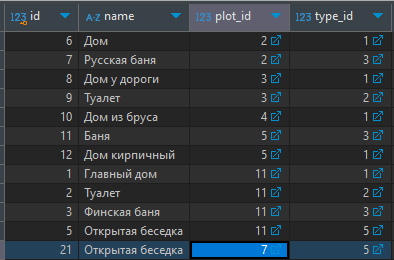


5. Очистка данных перед удалением участка: очистка данных в таблице связи собственников с участками и удаление построек (выполнение до удаления)

Состояние данных до удаления



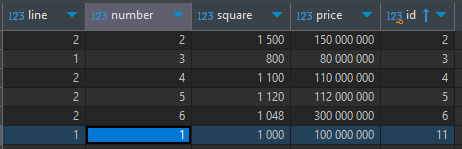


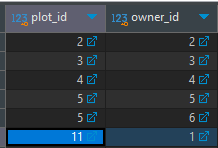


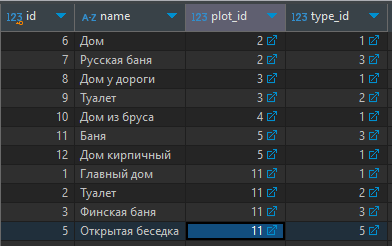
Запрос удаления

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

Состояние данных после удаления

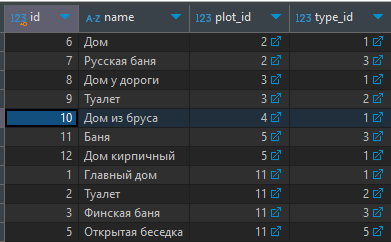






6. После удаления: логирование о возникновении пустых участков при удалении постройки

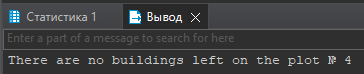
Единственная на участке постройка (участок 4)



Запрос на удаление

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

Вывод в лог сообщения о пустом участке:



**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.