МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБО6РОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | Н.В. Путилова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «Триггеры. Обеспечение активной целостности данных базы данных» |
| по курсу: Проектирование баз данных |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Самарин Д. В. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**Цель работы:**

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 (полей) таких полей нет;

- логирование (запись) изменений. сделал;

-обеспечения безопасности данных. Полей, для которых нужно обеспечивать безопасность тоже нет;

- логическое (мягкое) удаление данных. Для удаления я уже сделал триггеры

- проверка корректности проводимых действий. сделал).

Вычислимые поля можно добавить при необходимости.

По аналогии с примерами, приведенными в п. 1 реализовать для своей базы данных триггеры для всех событий (insert, delete, update) до и после. Часть из которых будет обеспечивать ссылочную целостность, остальные могут иметь другое назначение из 3 предложенных. Вычислимые поля можно добавить при необходимости

**Вариант задания (16 вариант)**

Школьные экскурсии: тип (развлекательная/образовательная), дисциплины к которым имеет отношение образовательная экскурсия, стоимость с человека, список участников, ответственный за проведение учитель

а. экскурсии на верфи (слово «верфь» в любом месте названия)

б. экскурсии, относящиеся к биологии и географии

в. учителя, не отвечающие за экскурсии

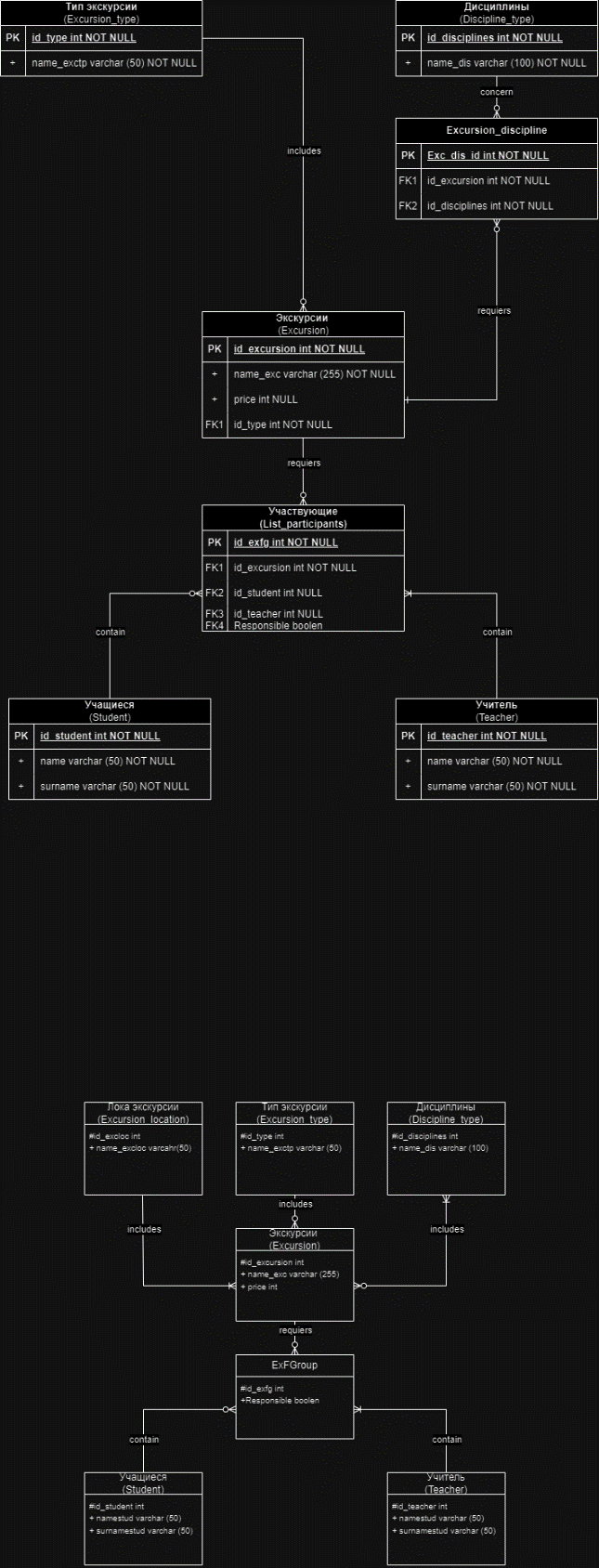
г. учащиеся, которые не ездили в музей истории религии, но ездили в Эрмитаж

д. учитель, отвечавший за число экскурсий больше среднего

е. учителя, отвечающие за самые дорогие экскурсии

ж. Экскурсии, на которые ездили все учащиеся

**3. Физическая модель БД**



**4. Триггеры:**

1)Если цена меньше или равна нулю,

то триггер вызовет исключение, и вставка будет отменена.

--BEFORE INSERT

CREATE OR REPLACE FUNCTION check\_positive\_price()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.price <= 0 THEN

RAISE EXCEPTION 'Невозможно добавить экскурсию с отрицательной или нулевой ценой';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_positive\_price\_trigger

BEFORE INSERT ON excursion

FOR EACH ROW

EXECUTE FUNCTION check\_positive\_price();

Пример:Insert into excursion (excursion\_id, excursion\_name, price , excursion\_type\_id) VALUES(7,'Музей растений',0,1);

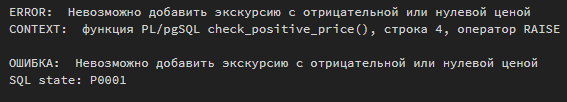
Было:

Таблица excursion:



После работы триггеры

Таблица excursion:





**2) После вставки новой записи в excursion\_discipline, триггер обновляет счетчик в соответствующей строке в таблице excursion.**

--AFTER INSERT

CREATE OR REPLACE FUNCTION update\_discipline\_count()

RETURNS TRIGGER AS $$

BEGIN

UPDATE excursion

SET discipline\_count = (SELECT COUNT(\*) FROM excursion\_discipline WHERE excursion\_id = NEW.excursion\_id)

WHERE excursion\_id = NEW.excursion\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_discipline\_count\_trigger

AFTER INSERT ON excursion\_discipline

FOR EACH ROW

EXECUTE FUNCTION update\_discipline\_count();

**Пример:**

Insert into excursion\_discipline(excursion\_discipline\_id, excursion\_id, discipline\_id) VALUES(11,5,7);

Insert into excursion\_discipline(excursion\_discipline\_id, excursion\_id, discipline\_id) VALUES(13,20,4);

Insert into excursion\_discipline(excursion\_discipline\_id, excursion\_id, discipline\_id) VALUES(14,20,9);

Insert into excursion\_discipline(excursion\_discipline\_id, excursion\_id, discipline\_id) VALUES(15,15,9);

До:

Таблица excursion\_discipline:

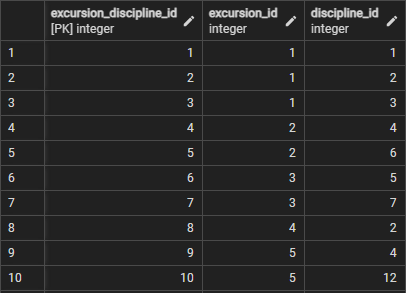
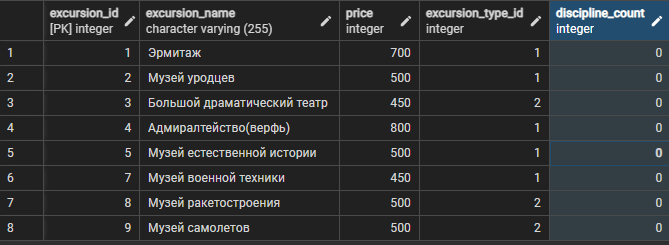


Таблица excursion:



После:

Таблица excursion:

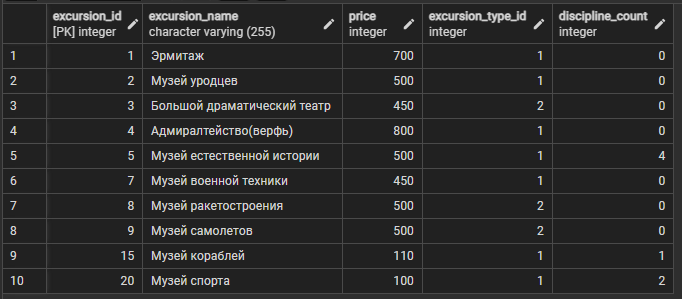
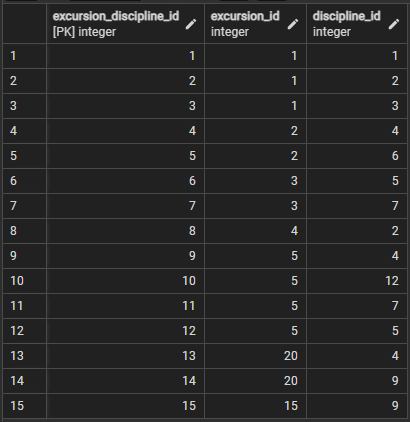


Таблица excursion\_discipline:



**3) При удалении экскурсии, на которую зарегистрированы участники будет выдаваться запрет на удаление**

-- BEFORE DELETE

CREATE OR REPLACE FUNCTION prevent\_delete\_excursion()

RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT 1 FROM List\_of\_participants WHERE excursion\_id = OLD.excursion\_id) THEN

RAISE EXCEPTION 'Нельзя удалить экскурсию, на которую есть участники.';

END IF;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_delete\_excursion\_trigger

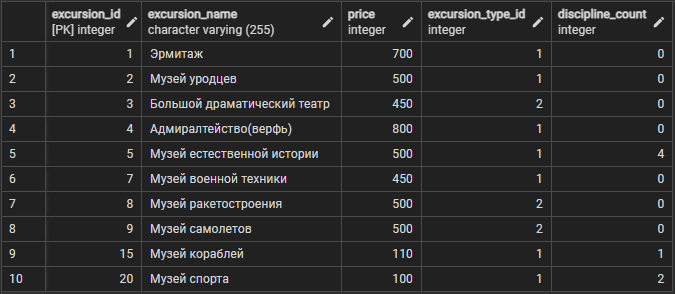
BEFORE DELETE ON excursion

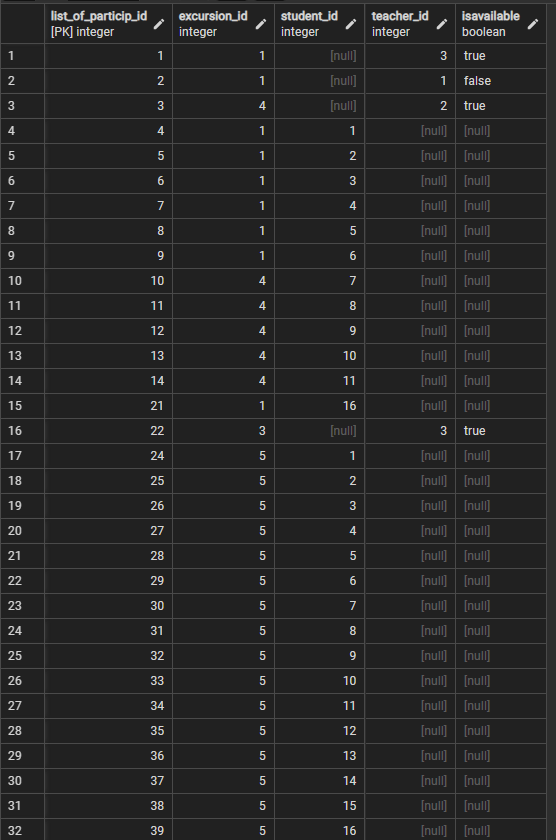
FOR EACH ROW

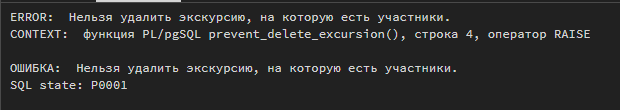
EXECUTE FUNCTION prevent\_delete\_excursion();

**Пример:**  
DELETE FROM excursion WHERE excursion\_id = 1;

Было:





После использования триггера:  


**4) Отслеживание удаленных записей из excursion**

CREATE SEQUENCE IF NOT EXISTS excursion\_delete\_log\_log\_id\_seq;

CREATE TABLE IF NOT EXISTS public.excursion\_delete\_log

(

log\_id integer NOT NULL DEFAULT nextval('excursion\_delete\_log\_log\_id\_seq'::regclass),

deleted\_at timestamp with time zone DEFAULT CURRENT\_TIMESTAMP,

excursion\_id integer,

excursion\_name character varying(255) COLLATE pg\_catalog."default",

CONSTRAINT excursion\_delete\_log\_pkey PRIMARY KEY (log\_id)

)

--AFTER UPDATE

CREATE OR REPLACE FUNCTION after\_update\_excursion\_function()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO excursion\_changes\_log (excursion\_id, old\_name, new\_name, old\_price, new\_price, change\_date)

VALUES (OLD.excursion\_id, OLD.excursion\_name, NEW.excursion\_name, OLD.price, NEW.price, CURRENT\_TIMESTAMP);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Создание триггера

CREATE TRIGGER trg\_after\_update\_excursion

AFTER UPDATE ON excursion

FOR EACH ROW

EXECUTE FUNCTION after\_update\_excursion\_function();

**Пример:**

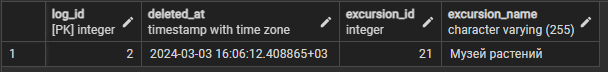
DELETE FROM excursion WHERE excursion\_id = 21;

Было:



После работы триггера:

Таблица log\_deleted\_excursion:



**5) Отслеживание измененных записей из excursion**

CREATE SEQUENCE IF NOT EXISTS excursion\_changes\_log\_log\_id\_seq;

CREATE TABLE IF NOT EXISTS public.excursion\_changes\_log

(

log\_id integer NOT NULL DEFAULT nextval('excursion\_changes\_log\_log\_id\_seq'::regclass),

excursion\_id integer,

old\_name character varying(255) COLLATE pg\_catalog."default",

new\_name character varying(255) COLLATE pg\_catalog."default",

old\_price integer,

new\_price integer,

change\_date timestamp without time zone DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT excursion\_changes\_log\_pkey PRIMARY KEY (log\_id)

)

--AFTER UPDATE

CREATE OR REPLACE FUNCTION after\_update\_excursion\_function()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO excursion\_changes\_log (excursion\_id, old\_name, new\_name, old\_price, new\_price, change\_date)

VALUES (OLD.excursion\_id, OLD.excursion\_name, NEW.excursion\_name, OLD.price, NEW.price, CURRENT\_TIMESTAMP);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Создание триггера

CREATE TRIGGER trg\_after\_update\_excursion

AFTER UPDATE ON excursion

FOR EACH ROW

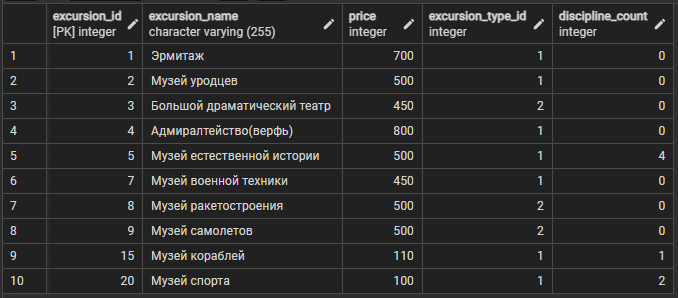
EXECUTE FUNCTION after\_update\_excursion\_function();

**Пример:**

Update excursion Set excursion\_name = 'Музей смартфонов' where excursion\_id = 15;

Было:

Таблица excursion:

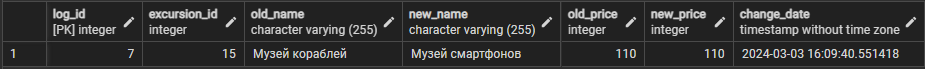


После работы триггера:

Таблица excursion:



Таблица: excursion\_changes\_log



**6) При обновлении первичного ключа экскурсии обновляются ключи и в таблице с дисциплинами экскурсий**

-- BEFORE UPDATE

CREATE OR REPLACE FUNCTION update\_excursion\_id\_references()

RETURNS TRIGGER AS $$

BEGIN

SET CONSTRAINTS fk\_excusrion\_discipline DEFERRED;

UPDATE excursion\_discipline

SET excursion\_id = NEW.excursion\_id

WHERE excursion\_id = OLD.excursion\_id;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER update\_excursion\_id\_trigger

BEFORE UPDATE OF excursion\_id ON excursion

FOR EACH ROW

EXECUTE FUNCTION update\_excursion\_id\_references();

**Пример:**

Update excursion Set excursion\_id = 25 where excursion\_id = 20;

Было:

Таблица excursion:

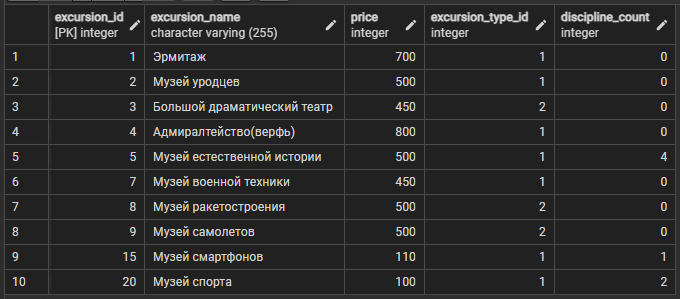
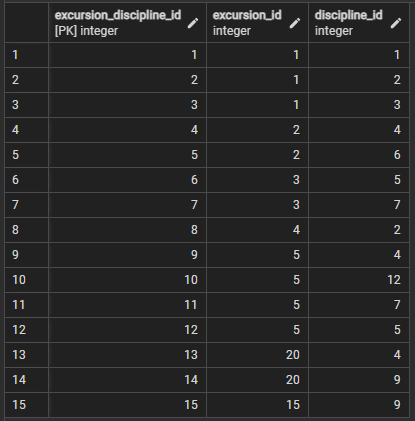


Таблица excursion\_discipline:



Стало:

Таблица excursion:

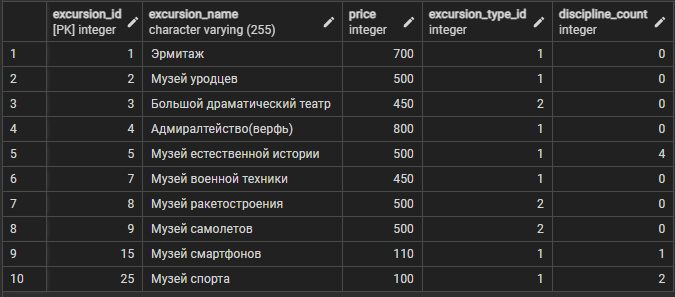
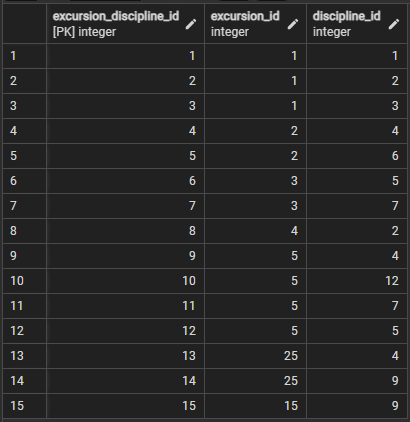


Таблица excursion\_discipline:



**Вывод:** в ходе лабораторной работы, я научился применять триггеры для всех событий.