Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский государственный университет аэрокосмического приборостроения» (ГУАП)

ОТЧЕТ ЗАЩИЩЕН

Ст. преподаватель Путилова Н.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Должность Ф.И.О дата, подпись

**Отчет**

**о лабораторной работе № 9**

**«Объектно-реляционные базы данных. Проектирование и создание»**

по дисциплине «Проектирование баз данных»

ОТЧЕТ ВЫПОЛНИЛ:

Студент Самарин Д.В 4134 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ф.И.О. группа дата, подпись

Санкт-Петербург

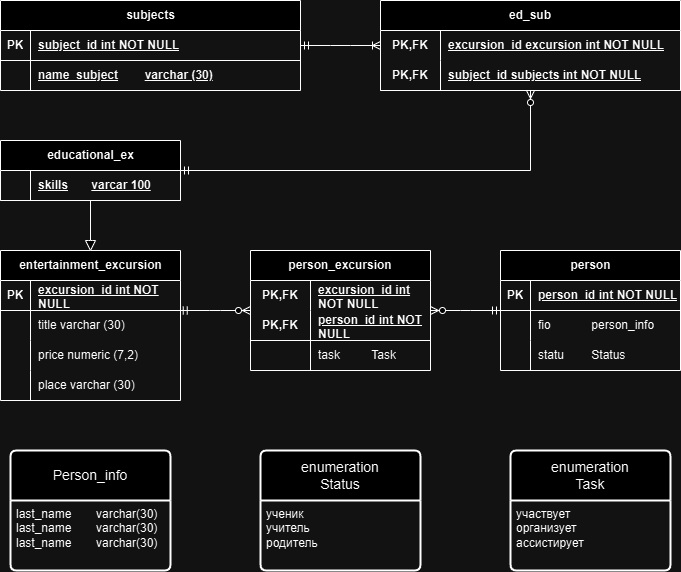
2024

**Вариант 16**

Школьные экскурсии: тип (развлекательная/образовательная), дисциплины к которым имеет отношение образовательная экскурсия, стоимость с человека, список участников, ответственный за проведение учитель, категории участников экскурсии (участвует, организует, ассистирует при проведении (старшие классы у младших или родители))

1. экскурсии в усадьбы (слово усадьба в любом месте названия)
2. экскурсии, относящиеся к 2 дисциплинам
3. учащиеся, которые не ездили в музей истории религии
4. экскурсия, собравшая наибольшее число ассистентов
5. учителя, отвечающие за самые дорогие экскурсии

**Физическая модель БД**



**Скрипт создания базы данных:**

|  |
| --- |
| *-- Определение пользовательского типа Task для перечисления*  *CREATE TYPE Task AS ENUM ('участвует', 'организует', 'ассистирует');*  *-- Определение пользовательского типа Status для перечисления*  *CREATE TYPE Status AS ENUM ('учитель', 'ученик', 'родитель');*  *-- Создание составного типа Person\_info для хранения информации о человеке*  *CREATE TYPE Person\_info AS (*  *last\_name VARCHAR(30),*  *name VARCHAR(20),*  *patronymic VARCHAR(30)*  *);*  *-- Создание таблицы subjects для хранения предметов*  *CREATE TABLE subjects (*  *subject\_id SERIAL PRIMARY KEY,*  *name\_subject VARCHAR(30) NOT NULL*  *);*  *-- Создание таблицы entertainment\_excursion для хранения экскурсионных мероприятий*  *CREATE TABLE entertainment\_excursion (*  *excursion\_id SERIAL PRIMARY KEY,*  *title VARCHAR(30) NOT NULL,*  *price NUMERIC(7, 2) NOT NULL,*  *place VARCHAR(30) NOT NULL*  *);*  *-- Создание таблицы educational\_ex с наследованием от entertainment\_excursion*  *CREATE TABLE educational\_ex (*  *PRIMARY KEY (excursion\_id),*  *skills VARCHAR(100)*  *) INHERITS (entertainment\_excursion);*  *-- Создание таблицы ed\_sub для связывания образовательных экскурсий с предметами*  *CREATE TABLE ed\_sub (*  *excursion\_id INT REFERENCES educational\_ex(excursion\_id)*  *ON DELETE CASCADE*  *ON UPDATE CASCADE,*  *subject\_id INT REFERENCES subjects(subject\_id)*  *ON DELETE CASCADE*  *ON UPDATE CASCADE,*  *PRIMARY KEY (excursion\_id, subject\_id)*  *);*  *-- Создание таблицы person для хранения информации о людях*  *CREATE TABLE person (*  *person\_id SERIAL PRIMARY KEY,*  *fio Person\_info NOT NULL,*  *status Status NOT NULL*  *);*  *-- Создание таблицы person\_excursion для связывания людей с экскурсиями и заданиями*  *CREATE TABLE person\_excursion (*  *excursion\_id INT,*  *task Task NOT NULL,*  *PRIMARY KEY (excursion\_id, person\_id),*  *person\_id INT REFERENCES person(person\_id)*  *ON DELETE CASCADE*  *ON UPDATE CASCADE*  *);* |

**Триггеры:**

|  |
| --- |
| -- Создание триггера для проверки существования записи в entertainment\_excursion перед добавлением в person\_excursion  CREATE OR REPLACE FUNCTION check\_entertainment\_excursion\_exists()  RETURNS TRIGGER AS $$  BEGIN  IF NOT EXISTS (  SELECT 1  FROM entertainment\_excursion  WHERE excursion\_id = NEW.excursion\_id  ) THEN  RAISE EXCEPTION 'Экскурсия с таким ID не существует в таблице entertainment\_excursion', NEW.excursion\_id;  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER check\_entertainment\_excursion\_exists\_trigger  BEFORE INSERT ON person\_excursion  FOR EACH ROW  EXECUTE FUNCTION check\_entertainment\_excursion\_exists();  -- Создание триггера для запрета обновления excursion\_id на несуществующий id в person\_excursion  CREATE OR REPLACE FUNCTION prevent\_update\_person\_excursion()  RETURNS TRIGGER AS $$  BEGIN  IF NOT EXISTS (  SELECT 1  FROM entertainment\_excursion  WHERE excursion\_id = NEW.excursion\_id  ) THEN  RAISE EXCEPTION 'Невозможно обновить excursion\_id, так как указанный ID не существует в entertainment\_excursion';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER prevent\_update\_person\_excursion\_trigger  BEFORE UPDATE OF excursion\_id ON person\_excursion  FOR EACH ROW  EXECUTE FUNCTION prevent\_update\_person\_excursion();  -- Создание триггера для каскадного удаления записей из person\_excursion при удалении из entertainment\_excursion  CREATE OR REPLACE FUNCTION cascade\_delete\_person\_excursion()  RETURNS TRIGGER AS $$  BEGIN  DELETE FROM person\_excursion WHERE excursion\_id = OLD.excursion\_id;  RETURN OLD;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER cascade\_delete\_person\_excursion\_trigger  BEFORE DELETE ON entertainment\_excursion  FOR EACH ROW  EXECUTE FUNCTION cascade\_delete\_person\_excursion();  -- Создание функции проверки связанных записей  CREATE OR REPLACE FUNCTION check\_related\_person\_excursion() RETURNS TRIGGER AS $$  BEGIN  -- Проверка наличия связанных записей в таблице person\_excursion  IF EXISTS (SELECT 1 FROM person\_excursion WHERE excursion\_id = OLD.excursion\_id) THEN  RAISE EXCEPTION 'нельзя обновить id entertainment\_exursion, есть связанные данные в person\_exursion';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  -- Создание триггера на таблицу entertainment\_excursion  CREATE TRIGGER before\_update\_entertainment\_excursion  BEFORE UPDATE OF excursion\_id ON entertainment\_excursion  FOR EACH ROW  EXECUTE FUNCTION check\_related\_person\_excursion(); |