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

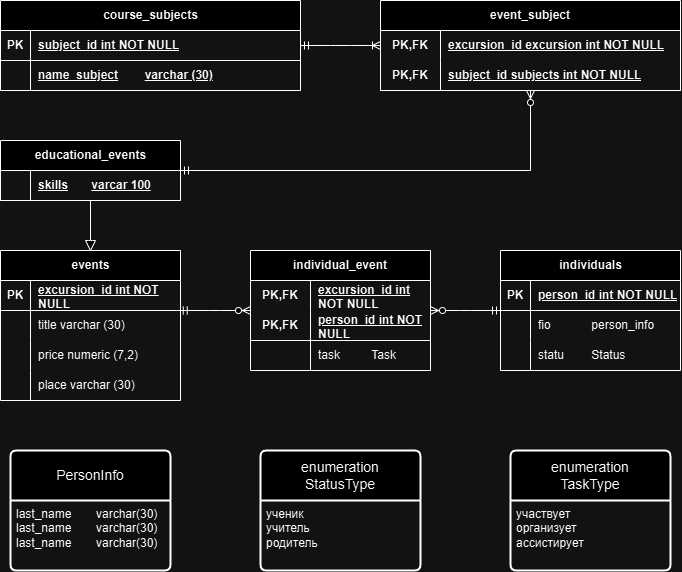
2024

**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 TaskType AS ENUM ('участвует', 'организует', 'ассистирует');  -- Определение пользовательского типа Status для перечисления  CREATE TYPE StatusType AS ENUM ('учитель', 'ученик', 'родитель');  -- Создание составного типа PersonInfo для хранения информации о человеке  CREATE TYPE PersonInfo AS (  last\_name VARCHAR(30),  name VARCHAR(20),  patronymic VARCHAR(30)  );  -- Создание таблицы course\_subjects для хранения предметов  CREATE TABLE course\_subjects (  course\_subject\_id SERIAL PRIMARY KEY,  subject\_name VARCHAR(30) NOT NULL  );  -- Создание таблицы events для хранения экскурсионных мероприятий  CREATE TABLE events (  event\_id SERIAL PRIMARY KEY,  event\_title VARCHAR(30) NOT NULL,  event\_price NUMERIC(7, 2) NOT NULL,  event\_place VARCHAR(30) NOT NULL  );  -- Создание таблицы educational\_events с наследованием от events  CREATE TABLE educational\_events (  PRIMARY KEY (event\_id),  event\_skills VARCHAR(100)  ) INHERITS (events);  -- Создание таблицы event\_subject для связывания образовательных экскурсий с предметами  CREATE TABLE event\_subject (  event\_id INT REFERENCES educational\_events(event\_id)  ON DELETE CASCADE  ON UPDATE CASCADE,  course\_subject\_id INT REFERENCES course\_subjects(course\_subject\_id)  ON DELETE CASCADE  ON UPDATE CASCADE,  PRIMARY KEY (event\_id, course\_subject\_id)  );  -- Создание таблицы individuals для хранения информации о людях  CREATE TABLE individuals (  individual\_id SERIAL PRIMARY KEY,  personal\_info PersonInfo NOT NULL,  individual\_status StatusType NOT NULL  );  -- Создание таблицы individual\_event для связывания людей с экскурсиями и заданиями  CREATE TABLE individual\_event (  event\_id INT,  individual\_task TaskType NOT NULL,  PRIMARY KEY (event\_id, individual\_id),  individual\_id INT REFERENCES individuals(individual\_id)  ON DELETE CASCADE  ON UPDATE CASCADE  ); |

**Триггеры:**

|  |
| --- |
| -- Создание триггера для проверки существования записи в events перед добавлением в individual\_event  CREATE OR REPLACE FUNCTION validate\_event\_exists()  RETURNS TRIGGER AS $$  BEGIN  IF NOT EXISTS (  SELECT 1  FROM events  WHERE event\_id = NEW.event\_id  ) THEN  RAISE EXCEPTION 'Мероприятие с таким ID не существует в таблице events', NEW.event\_id;  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_validate\_event\_exists  BEFORE INSERT ON individual\_event  FOR EACH ROW  EXECUTE FUNCTION validate\_event\_exists();  -- Создание триггера для запрета обновления event\_id в events, если есть записи в individual\_event  CREATE OR REPLACE FUNCTION block\_event\_id\_update()  RETURNS TRIGGER AS $$  BEGIN  IF EXISTS (  SELECT 1  FROM individual\_event  WHERE event\_id = OLD.event\_id  ) THEN  RAISE EXCEPTION 'Невозможно обновить event\_id, так как существуют записи в individual\_event с этим event\_id';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_block\_event\_id\_update  BEFORE UPDATE OF event\_id ON events  FOR EACH ROW  EXECUTE FUNCTION block\_event\_id\_update();  -- Создание триггера для запрета обновления event\_id на несуществующий id в individual\_event  CREATE OR REPLACE FUNCTION block\_invalid\_event\_id\_update()  RETURNS TRIGGER AS $$  BEGIN  IF NOT EXISTS (  SELECT 1  FROM events  WHERE event\_id = NEW.event\_id  ) THEN  RAISE EXCEPTION 'Невозможно обновить event\_id, так как указанный ID не существует в events';  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_block\_invalid\_event\_id\_update  BEFORE UPDATE OF event\_id ON individual\_event  FOR EACH ROW  EXECUTE FUNCTION block\_invalid\_event\_id\_update();  -- Создание триггера для каскадного удаления записей из individual\_event при удалении из events  CREATE OR REPLACE FUNCTION cascade\_delete\_individual\_event()  RETURNS TRIGGER AS $$  BEGIN  DELETE FROM individual\_event WHERE event\_id = OLD.event\_id;  RETURN OLD;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_cascade\_delete\_individual\_event  BEFORE DELETE ON events  FOR EACH ROW  EXECUTE FUNCTION cascade\_delete\_individual\_event(); |