МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра кибернетики (№ 22)

Направление подготовки 09.03.04 Программная инженерия

Курсовая работа по предмету

”Основы автоматизированных информационных технологий”

Тема: «Расписание и сдача ЕГЭ»

Преподаватель:	Тихомирова Дарья Валерьевна
Студент:	Колесников Михаил Леонидович
Группа:	Б22-534

Москва 2024

Содержание

1 Описание предметной области

1.1 Формулировка задания

Спроектировать базу данных для проведения Единого Государственного экзамена, проводящегося ежегодно в школах разных городов Российской Федерации. База данных должна содержать информацию о студентах, школах и учителях, а также отражать ежегодные данные по сдаваемым предметам, составленное расписание и полученные учениками результаты.

1.2 Конкретизация предметной области

Необходимо создать систему, отражающую информацию о проведении и результатах экзаменов по всей стране. По каждому предмету есть ежегодная информация, так как Министерство образования ежегодно вносит коррективы в тот или иной экзамен. База данных должна отражать точное расписание экзаменов по всем городам каждый год, а также результаты конкретного ученика по всем выбранным им предметам.

1.3 Пользователи системы

Основными пользователями системы являются:

- **Администраторы школ** - управляют данными учебных заведений, аудиториями, расписанием экзаменов
- **Преподаватели** - ведут учет предметов, заданий, результатов экзаменов
- **Учащиеся** - просматривают расписание экзаменов, свои результаты
- **Экзаменационная комиссия** - вносит результаты экзаменов, проверяет работы
- **Технические специалисты** - обслуживают систему, обеспечивают целостность данных

1.4 Сроки хранения информации

- **Персональные данные учащихся** - хранятся до 5 лет после окончания обучения
- **Результаты экзаменов (ExamResult)** - хранятся постоянно в архиве
- **Расписание экзаменов (Exam)** - хранится 3 года после даты проведения
- **Учебные задания (Task)** - хранятся 5 лет для возможного апелллирования
- **Справочная информация** (школы, адреса, предметы) - хранится постоянно

1.5 События, изменяющие состояние базы данных

Критические события системы:

- Добавление/изменение расписания экзаменов (сущность Exam)
- Регистрация учащегося на экзамен (отношение go_to)
- Внесение результатов экзамена (сущность ExamResult)

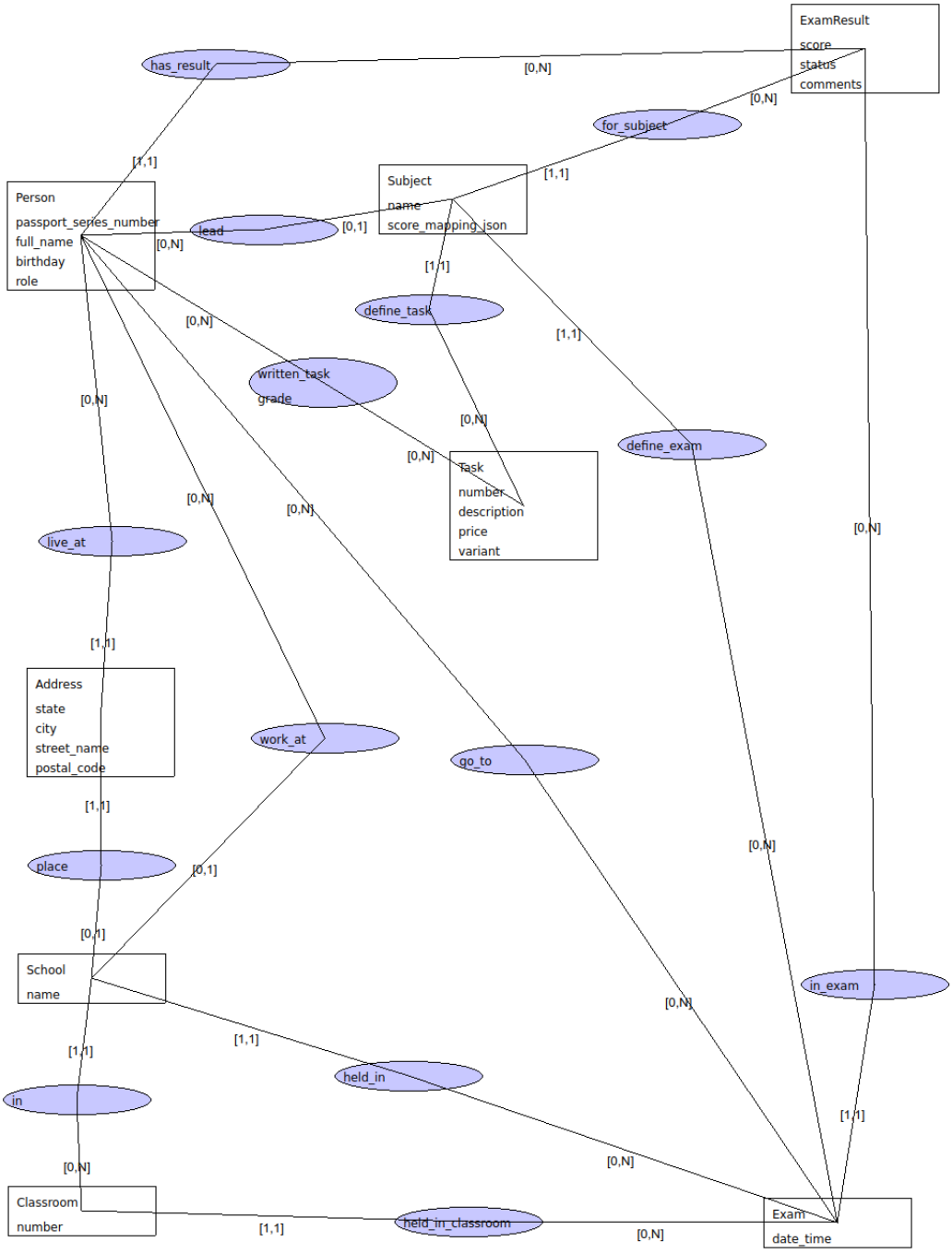
- Назначение преподавателя на предмет (отношение lead)
- Изменение состава экзаменационной комиссии (отношение work_at)
- Подача апелляции (изменение статуса в ExamResult)
- Обновление учебных заданий (сущность Task)

1.6 Основные запросы к базе данных (на естественном языке)

- Получить расписание всех экзаменов для указанной школы на заданную дату
- Найти всех учащихся, сдающих конкретный предмет в указанный период
- Показать средний балл по предмету для выбранной школы
- Получить список всех аудиторий, занятых в определенный день
- Найти всех преподавателей, ведущих указанный предмет
- Получить историю экзаменов конкретного учащегося с результатами
- Показать распределение оценок по конкретному экзамену
- Найти всех учащихся, проживающих в указанном городе/районе
- Получить список заданий для конкретного варианта экзамена
- Показать статистику сдачи экзаменов по школам за указанный период

2 Концептуально-информационная модель предметной области

2.1 Er-диаграмма модели



2.2 Оценка мощностных характеристик сущностей и связей

2.2.1 Оценка мощностных характеристик сущностей

- **Сущность Address**:
  - Ожидаемое количество: 251,000
    Формула: $1000_{\text{школ}} + 500000_{\text{учеников}} + 150000_{\text{учителей}}$
- **Сущность School**:
  - Ожидаемое количество: 1,000
- **Сущность Man**:
  - Ожидаемое количество: 650,000
    Формула: $100000_{\text{учеников}} \times 5_{\text{лет}} + 150000_{\text{учителей}}$
- **Сущность Subject**:
  - Ожидаемое количество: 16
- **Сущность Exam**:
  - Ожидаемое количество: 8,000
    Формула: $16_{\text{предметов}} \times 100_{\text{школ/год}} \times 5_{\text{лет}}$
- **Сущность ExamResult**:
  - Ожидаемое количество: 1,500,000
    Формула: $100000_{\text{учеников/год}} \times 3_{\text{предмета}} \times 5_{\text{лет}}$
- **Сущность Task**:
  - Ожидаемое количество: 48,000
    Формула: $16_{\text{предметов}} \times 30_{\text{заданий}} \times 5_{\text{лет}} \times 20_{\text{вариантов}}$
- **Сущность Classroom**:
  - Ожидаемое количество: 20,000
    Формула: $20_{\text{аудиторий/школу}} \times 1000_{\text{школ}}$

2.2.2 Оценка мощностных характеристик связей

- **written_task**:
  - Ожидаемое количество: 3,000,000
    Формула: $1.5\text{M}_{\text{результатов}} \times 30_{\text{заданий}}$
- **go_to**:
  - Ожидаемое количество: 1,500,000
    Формула: ExamResult
- **has_result**:
  - Ожидаемое количество: 1,500,000
    Формула: ExamResult

- **live_at**:
  - Ожидаемое количество: 650,000
    Формула: Man
- **work_at**:
  - Ожидаемое количество: 150,000
    Формула: (количество учителей)

3 Концептуальное проектирование

3.1 Принятые проектные соглашения

В проекте базы данных для системы ”Расписание и сдача ЕГЭ”приняты следующие соглашения:

- **Именование объектов**:
  - Таблицы - в единственном числе (School, Man, Subject)
  - Поля - в snake_case (passport_series_number, street_name)
- **Типы данных**:
  - Для персональных данных - VARCHAR с ограничением длины
  - Даты - тип DATE или TIMESTAMP
  - JSON-данные (score_mapping_json) - тип JSONB
- **Нормализация**:
  - База данных приведена к 3НФ
- **Архитектура**:
  - Клиент-серверная архитектура
  - Использование хранимых процедур для сложных операций

3.2 Обоснование выбора модели базы данных

Для системы ”Расписание и сдача ЕГЭ”выбрана реляционная модель базы данных по следующим причинам:

- **Структурированность данных**:
  - Четко определенные связи между сущностями
  - Жесткая схема данных обеспечивает целостность
- **Транзакционность**:
  - Требуется ACID-совместимость для операций с результатами экзаменов
  - Важна согласованность данных при параллельном доступе
- **Масштабируемость**:

  - Предсказуемый рост данных
  - Возможность репликации для отчетных серверов

- **Безопасность**:
  - Разграничение доступа на уровне строк (RLS)
  - Поддержка шифрования данных

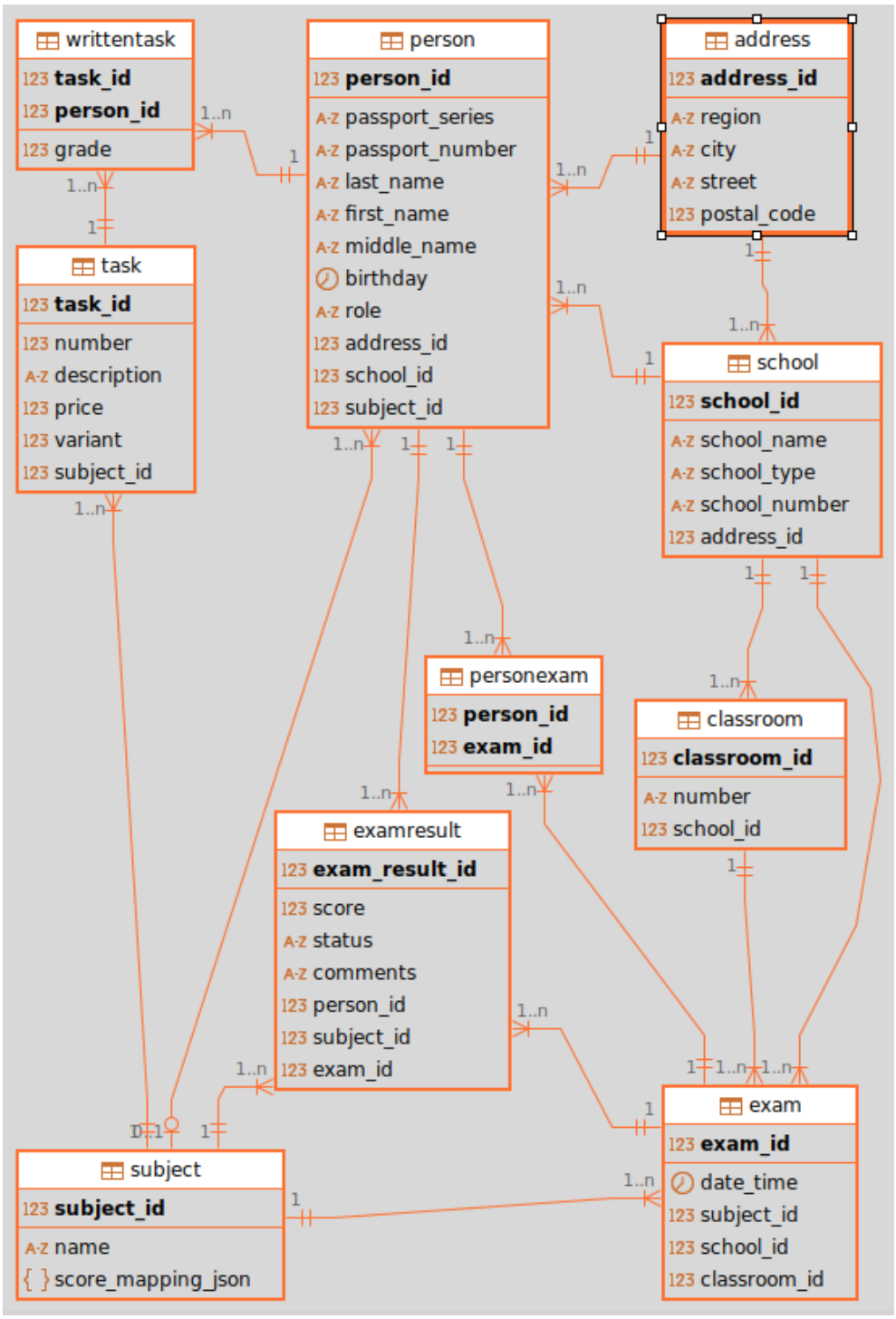
3.3 Используемые в системе кодификаторы

- **Кодификатор предметов (Subject)**:
  - Цифровой код (например, 01 - Математика, 02 - Физика)
  - Соответствует официальным кодам ЕГЭ

- **Кодификатор территорий (Address)**:
  - Код региона

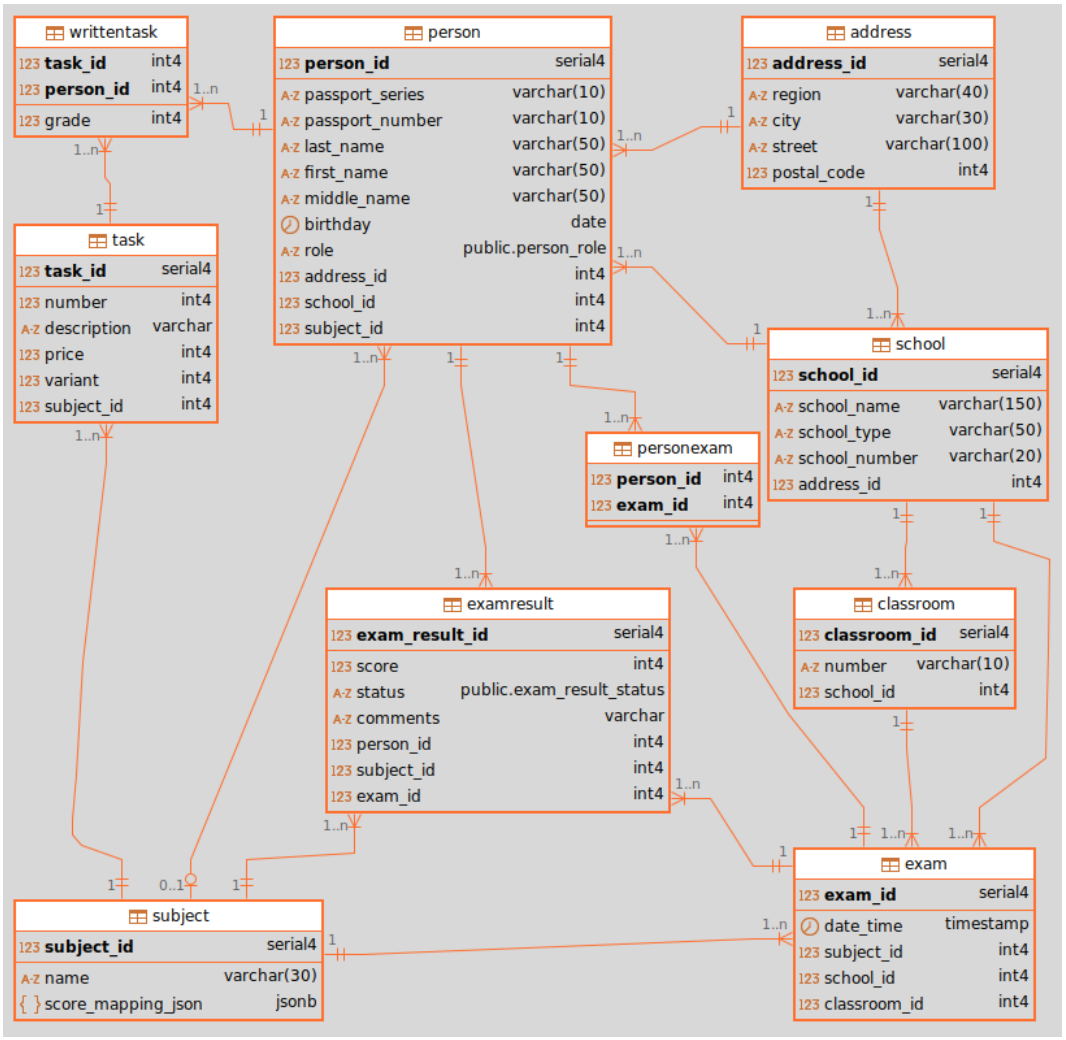
- **Кодификатор статусов экзамена (ExamResult)**:
  - 0 - Зарегистрирован
  - 1 - Сдан
  - 2 - Не сдан
  - 3 - На апелляции
  - 4 - Аннулирован

3.4 Концептуальная модель базы данных

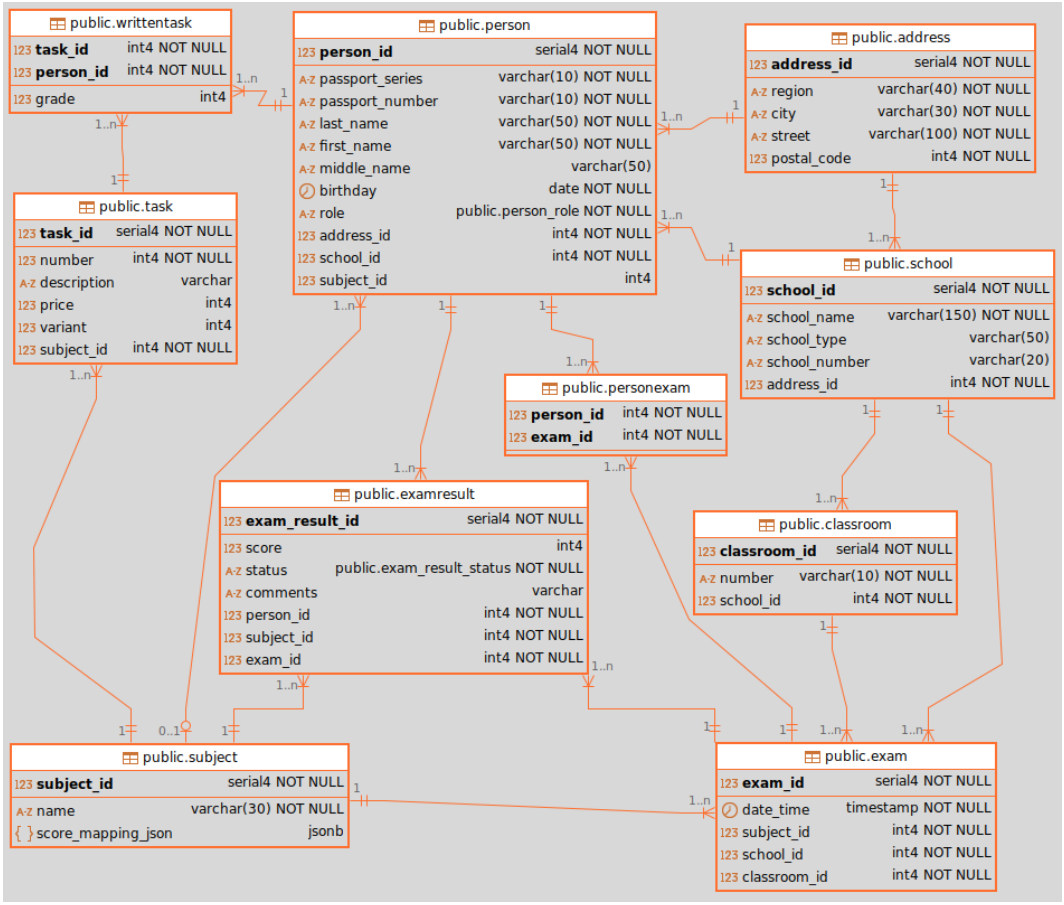


4 Логическое проектирование

4.1 Er-диаграмма базы данных (logical)



4.2 Схемы отношений базы данных (physical)



4.3 Схема реляционной базы данных

Address(address_id, region, city, street, postal_code)	R1
Subject(subject_id, name, score_mapping_json)	R2
School(school_id, school_name, school_type, school_number, address_id)	R3
Person(person_id, passport_series, passport_number, last_name, first_name, middle_name, birthday, role, address_id, school_id, subject_id)	R4
Classroom(classroom_id, number, school_id)	R5
Exam(exam_id, date_time, subject_id, school_id, classroom_id)	R6
Task(task_id, number, description, price, variant, subject_id)	R7
ExamResult(exam_result_id, score, status, comments, person_id, subject_id, exam_id)	R8
PersonExam(person_id, exam_id)	R9
WrittenTask(task_id, person_id, grade)	R10

4.4 Схемы основных запросов на реляционной алгебре

- Получить всех учителей, зарегистрированных на экзамен по предмету ”Name”

```
TeachersMathExam(id) = ((Person[Person.role = 'Teacher' &
    Person.subject_id = Subject.subject_id] Subject)[Subject.name =
    'Name'][Person.person_id = PersonExam.person_id]
    PersonExam)[Person.person_id]
```

- Найти всех школьников, не зарегистрированных ни на один экзамен

```
UnregisteredStudents(id) = (Person[Person.role =
    'SchoolChild'])[Person.person_id] \
    (PersonExam)[PersonExam.person_id]
```

- Получить список предметов, по которым проводились экзамены в школе Name

```
SubjectsSchool42(id) = ((Exam[Exam.subject_id = Subject.subject_id]
    Subject)[Exam.school_id = School.school_id & School.school_name =
    'Name'] School)[Subject.subject_id]
```

- Найти всех учеников, получивших хотя бы одну неудовлетворительную оценку (<40)

```
FailedStudents40(id) = (Person[Person.person_id = ExamResult.person_id
    & ExamResult.score < 40 & Person.role = 'SchoolChild']
    ExamResult)[Person.person_id]
```

- Получить список школ, в которых нет зарегистрированных экзаменов

```
SchoolsNoExams(id) = (School)[School.school_id] \ (Exam)[Exam.school_id]
```

- Найти всех школьников, у которых по всем предметам, которые он сдавал, 100 баллов

```
SchoolChildren(id, score) = (Person[Person.role='SchoolChild' &
    Person.person_id = ExamResult.person_id]ExamResult)[person_id, score]
CoolSchoolChildren(id) = (SchoolChildren \ SchoolChildren[score !=
    100])[id]
```

- Получить всех учителей, которые преподают в школе, не находящейся в городе, в котором они живут

```
T(Person.person_id, Person.school_id, Address.city) =
    (Person[Person.role = 'Teacher' & Person.address_id =
    Address.address_id]Address)[Person.person_id, Person.school_id,
    Address.city]
S(School.school_id, Address.city) = (School[School.address_id =
    Address.address_id]Address)[School.school_id, Address.city]
PoorTeachers(person_id) = (T[T.school_id = S.school_id & T.city !=
    S.city]S)[person_id]
```

5 Физическое проектирование

5.1 Обоснование выбора конкретной СУБД

В качестве системы управления базами данных (СУБД) для реализации проекта была выбрана PostgreSQL. Этот выбор обоснован рядом факторов:

- PostgreSQL — это мощная объектно-реляционная СУБД с открытым исходным кодом, которая поддерживает расширенные возможности SQL и соответствует стандартам.
- Система обладает высокой производительностью и эффективной планировкой запросов, что важно при работе с большим объемом данных и сложными связями между таблицами.
- PostgreSQL широко используется в индустрии и хорошо поддерживается сообществом, что обеспечивает доступность документации, инструментов и решений типовых задач.
- Также немаловажным преимуществом является наличие встроенной поддержки пользовательских типов, таких как перечисления (ENUM), которые активно применяются в данной базе данных.
- Наконец, PostgreSQL хорошо интегрируется с Python через библиотеки psycopg и SQLAlchemy, что важно на этапе генерации и заполнения данных.

Таким образом, PostgreSQL является оптимальным выбором как с технической, так и с практической точек зрения.

5.2 Создание базы данных

```
services:
  postgres:
    container_name: postgres-db2
    image: postgres
    environment:
```

```
      POSTGRES_USER: root
      POSTGRES_PASSWORD: root
      POSTGRES_DB: database
      PGDATA: /var/lib/postgresql/data
    volumes:
      - postgres:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped
volumes:
  postgres:
```

5.3 Создание таблиц

```
CREATE TABLE Address (
    address_id SERIAL PRIMARY KEY,
    region VARCHAR(40) NOT NULL,
    city VARCHAR(30) NOT NULL,
    street VARCHAR(100) NOT NULL,
    postal_code INTEGER NOT NULL
);

CREATE TABLE School (
    school_id SERIAL PRIMARY KEY,
    school_name VARCHAR(150) NOT NULL,
    address_id INTEGER NOT NULL,
    FOREIGN KEY (address_id) REFERENCES Address(address_id)
);

CREATE TABLE Subject (
    subject_id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL,
    score_mapping_json JSONB
);

CREATE TYPE person_role AS ENUM ('SchoolChild', 'Teacher');

CREATE TABLE Person (
    person_id SERIAL PRIMARY KEY,
    passport_series VARCHAR(10) NOT NULL,
    passport_number VARCHAR(10) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    middle_name VARCHAR(50),
    birthday DATE NOT NULL,
    role person_role NOT NULL,
    address_id INTEGER NOT NULL,
    school_id INTEGER NOT NULL,
    subject_id INTEGER,
    FOREIGN KEY (address_id) REFERENCES Address(address_id),
    FOREIGN KEY (school_id) REFERENCES School(school_id),
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id)
);
```

```sql
CREATE TABLE Classroom (
    classroom_id SERIAL PRIMARY KEY,
    number VARCHAR(10) NOT NULL,
    school_id INTEGER NOT NULL,
    FOREIGN KEY (school_id) REFERENCES School(school_id)
);

CREATE TABLE Exam (
    exam_id SERIAL PRIMARY KEY,
    date_time TIMESTAMP NOT NULL,
    subject_id INTEGER NOT NULL,
    school_id INTEGER NOT NULL,
    classroom_id INTEGER NOT NULL,
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id),
    FOREIGN KEY (school_id) REFERENCES School(school_id),
    FOREIGN KEY (classroom_id) REFERENCES Classroom(classroom_id)
);

CREATE TABLE Task (
    task_id SERIAL PRIMARY KEY,
    number INTEGER NOT NULL,
    description VARCHAR,
    price INTEGER,
    variant INTEGER,
    subject_id INTEGER NOT NULL,
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id)
);

CREATE TYPE exam_result_status AS ENUM (
    'Registered',
    'Passed',
    'Failed',
    'Absent',
    'Appealed',
    'Canceled',
    'Processing',
    'Waiting'
);

CREATE TABLE ExamResult (
    exam_result_id SERIAL PRIMARY KEY,
    score INTEGER,
    status exam_result_status NOT NULL,
    comments VARCHAR,
    person_id INTEGER NOT NULL,
    subject_id INTEGER NOT NULL,
    exam_id INTEGER NOT NULL,
    FOREIGN KEY (person_id) REFERENCES Person(person_id),
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id),
    FOREIGN KEY (exam_id) REFERENCES Exam(exam_id)
);
```

```sql
CREATE TABLE WrittenTask (
    task_id INTEGER NOT NULL,
    person_id INTEGER NOT NULL,
    grade INTEGER,
    PRIMARY KEY (task_id, person_id),
    FOREIGN KEY (task_id) REFERENCES Task(task_id),
    FOREIGN KEY (person_id) REFERENCES Person(person_id)
);

CREATE TABLE PersonExam (
    person_id INTEGER NOT NULL,
    exam_id INTEGER NOT NULL,
    PRIMARY KEY (person_id, exam_id),
    FOREIGN KEY (person_id) REFERENCES Person(person_id),
    FOREIGN KEY (exam_id) REFERENCES Exam(exam_id)
);
```

5.4 Заполнение таблиц. ETL-процессы загрузки базы данных

5.5 Запросы в терминах SQL

- Получить всех учителей, зарегистрированных на экзамен по предмету ”Name”

```sql
SELECT DISTINCT p.person_id
FROM Person p
JOIN Subject s ON p.subject_id = s.subject_id
JOIN PersonExam pe ON p.person_id = pe.person_id
WHERE p.role = 'Teacher' AND s.name = 'Name';
```

- Найти всех школьников, не зарегистрированных ни на один экзамен

```sql
SELECT p.person_id
FROM Person p
WHERE p.role = 'SchoolChild'
AND p.person_id NOT IN (SELECT person_id FROM PersonExam);
```

- Получить список предметов, по которым проводились экзамены в школе Name

```sql
SELECT DISTINCT s.subject_id
FROM Exam e
JOIN Subject s ON e.subject_id = s.subject_id
JOIN School sc ON e.school_id = sc.school_id
WHERE sc.school_name = 'Name';
```

- Найти всех учеников, получивших хотя бы одну неудовлетворительную оценку (<40)

```sql
SELECT DISTINCT p.person_id
FROM Person p
JOIN ExamResult er ON p.person_id = er.person_id
WHERE p.role = 'SchoolChild' AND er.score < 40;
```

- Получить список школ, в которых нет зарегистрированных экзаменов

```sql
SELECT s.school_id
FROM School s
WHERE s.school_id NOT IN (SELECT school_id FROM Exam);
```

- Найти всех школьников, у которых по всем предметам, которые он сдавал, 100 баллов

```
WITH SchoolChildren AS (
    SELECT p.person_id, er.score
    FROM Person p
    JOIN ExamResult er ON p.person_id = er.person_id
    WHERE p.role = 'SchoolChild'
)
SELECT DISTINCT person_id AS id
FROM SchoolChildren sc1
WHERE NOT EXISTS (
    SELECT 1
    FROM SchoolChildren sc2
    WHERE sc1.person_id = sc2.person_id AND sc2.score != 100
);
```

- Получить всех учителей, которые преподают в школе, не находящейся в городе, в котором они живут

```
SELECT DISTINCT p.person_id
FROM Person p
JOIN Address pa ON p.address_id = pa.address_id
JOIN School s ON p.school_id = s.school_id
JOIN Address sa ON s.address_id = sa.address_id
WHERE p.role = 'Teacher' AND pa.city != sa.city;
```

5.6 Оценка размеров базы данных и каждого из файлов

Relation	Attributes	Size (bytes)	Avg Records	Total Size (bytes)	Total Size (MB)
Address	address_id (SERIAL) region (VARCHAR(40)) city (VARCHAR(30)) street (VARCHAR(100)) postal_code (INTEGER)	4 40 30 100 4	251,000	44,678,000	42.6
School	school_id (SERIAL) school_name (VARCHAR(150)) address_id (INTEGER)	4 150 4	1,000	158,000	0.15
Person	person_id (SERIAL) passport_-series (VARCHAR(10))	4 10	650,000	123,500,000	117.8

Relation	Attributes	Size (bytes)	Avg Records	Total Size (bytes)	Total Size (MB)
	passport_-number (VARCHAR(10))	10			
	last_name (VARCHAR(50))	50			
	first_name (VARCHAR(50))	50			
	middle_name (VARCHAR(50))	50			
	birthday (DATE)	4			
	role (person_-role)	10			
	address_id (INTEGER)	4			
	school_id (INTEGER)	4			
	subject_id (INTEGER)	4			
Subject	subject_id (SERIAL)	4	16	2,144	0.002
	name (VARCHAR(30))	30			
	score_-mapping_json (JSONB)	100			
Exam	exam_id (SERIAL)	4	8,000	192,000	0.18
	date_time (TIMESTAMP)	8			
	subject_id (INTEGER)	4			
	school_id (INTEGER)	4			
	classroom_id (INTEGER)	4			
ExamResult	exam_-result_id (SERIAL)	4	1,500,000	195,000,000	186.0
	score (INTEGER)	4			
	status (exam_-result_status)	10			
	comments (VARCHAR)	100			
	person_id (INTEGER)	4			

Relation	Attributes	Size (bytes)	Avg Records	Total Size (bytes)	Total Size (MB)
	subject_id (INTEGER)	4			
	exam_id (INTEGER)	4			
Task	task_id (SERIAL)	4	48,000	5,760,000	5.49
	number (INTEGER)	4			
	description (VARCHAR)	100			
	price (INTEGER)	4			
	variant (INTEGER)	4			
	subject_id (INTEGER)	4			
Classroom	classroom_id (SERIAL)	4	20,000	360,000	0.34
	number (VARCHAR(10))	10			
	school_id (INTEGER)	4			
WrittenTask	task_id (INTEGER)	4	3,000,000	36,000,000	34.33
	person_id (INTEGER)	4			
	grade (INTEGER)	4			
PersonExam	person_id (INTEGER)	4	1,500,000	12,000,000	11.44
	exam_id (INTEGER)	4			