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)

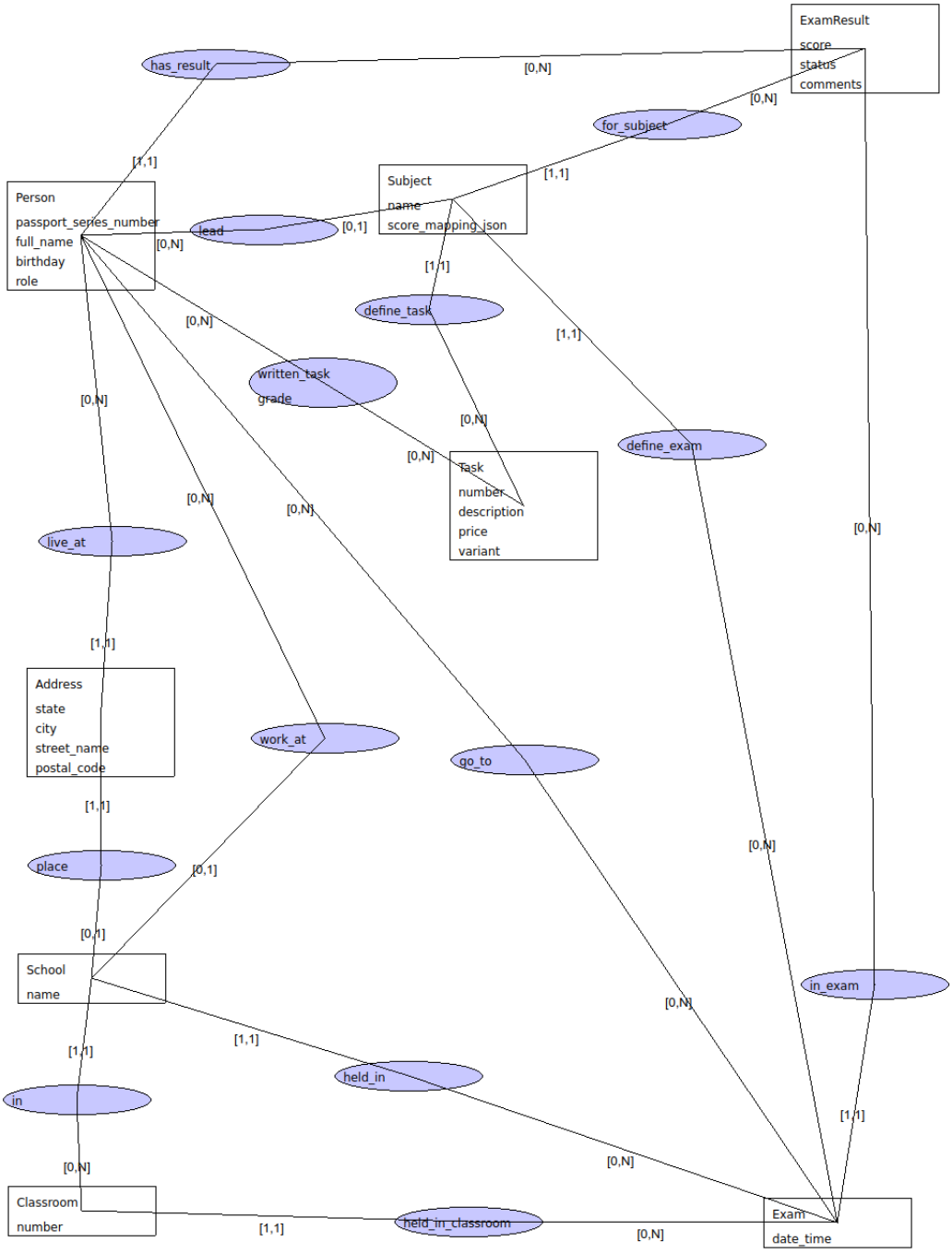
- Назначение преподавателя на предмет (отношение lead)
- Изменение состава экзаменационной комиссии (отношение work_at)
- Подача апелляции (изменение статуса в ExamResult)
- Обновление учебных заданий (сущность Task)

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

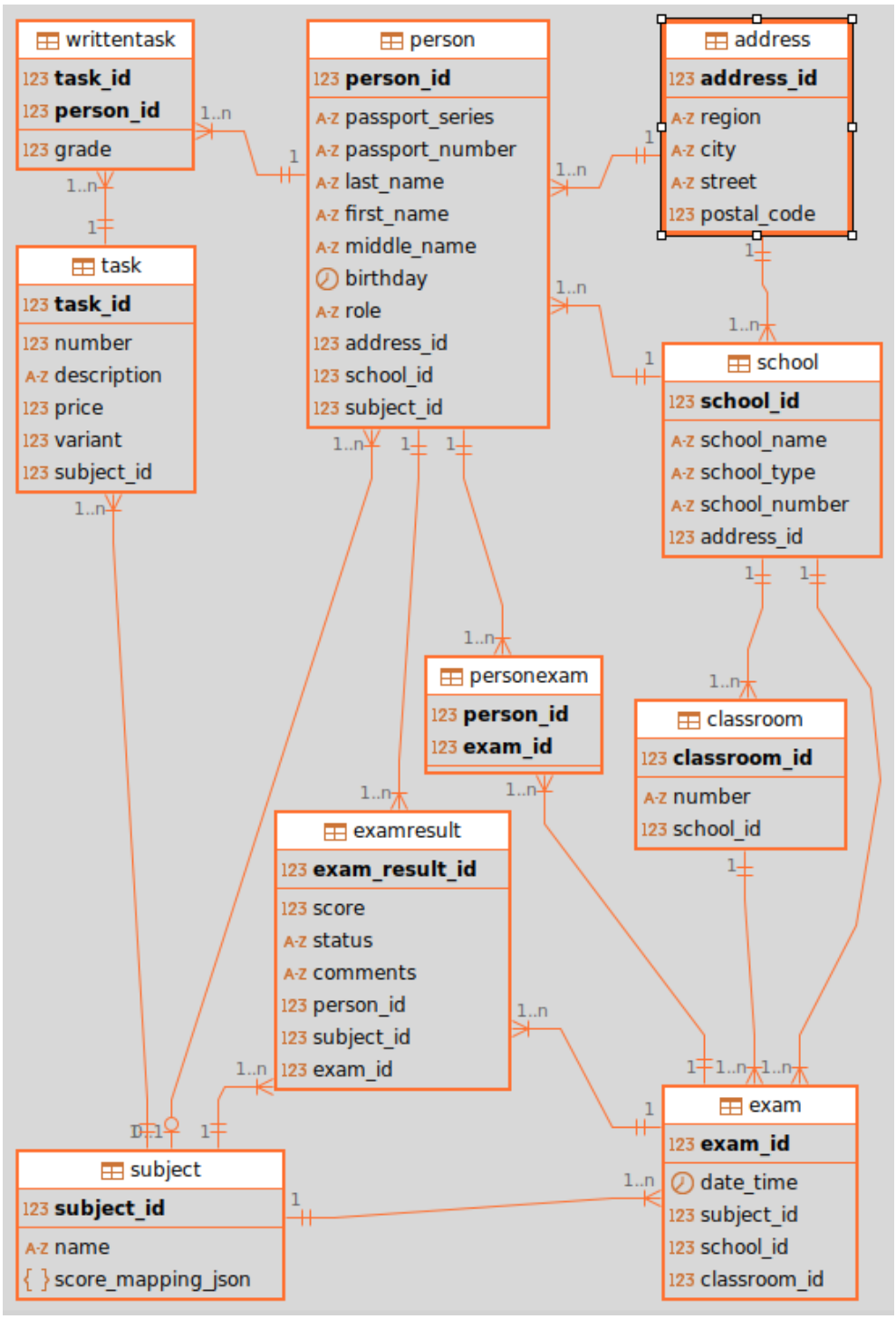
3.3 Используемые в системе кодификаторы

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

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

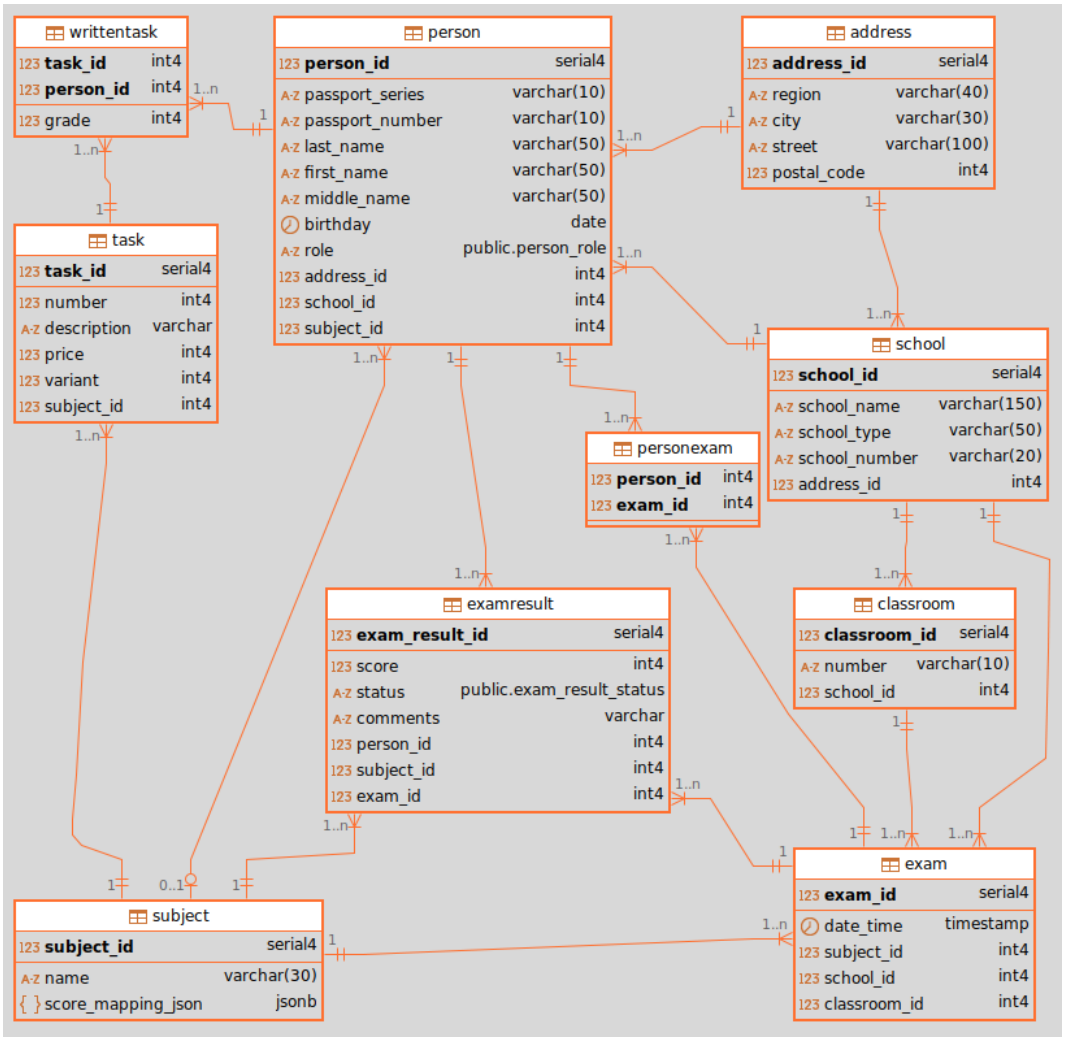
- **Кодификатор статусов экзамена (ExamResult)**:
  - 0 - Зарегистрирован
  - 1 - Сдан
  - 2 - Не сдан
  - 3 - На апелляции
  - 4 - Аннулирован

3.4 Концептуальная модель базы данных

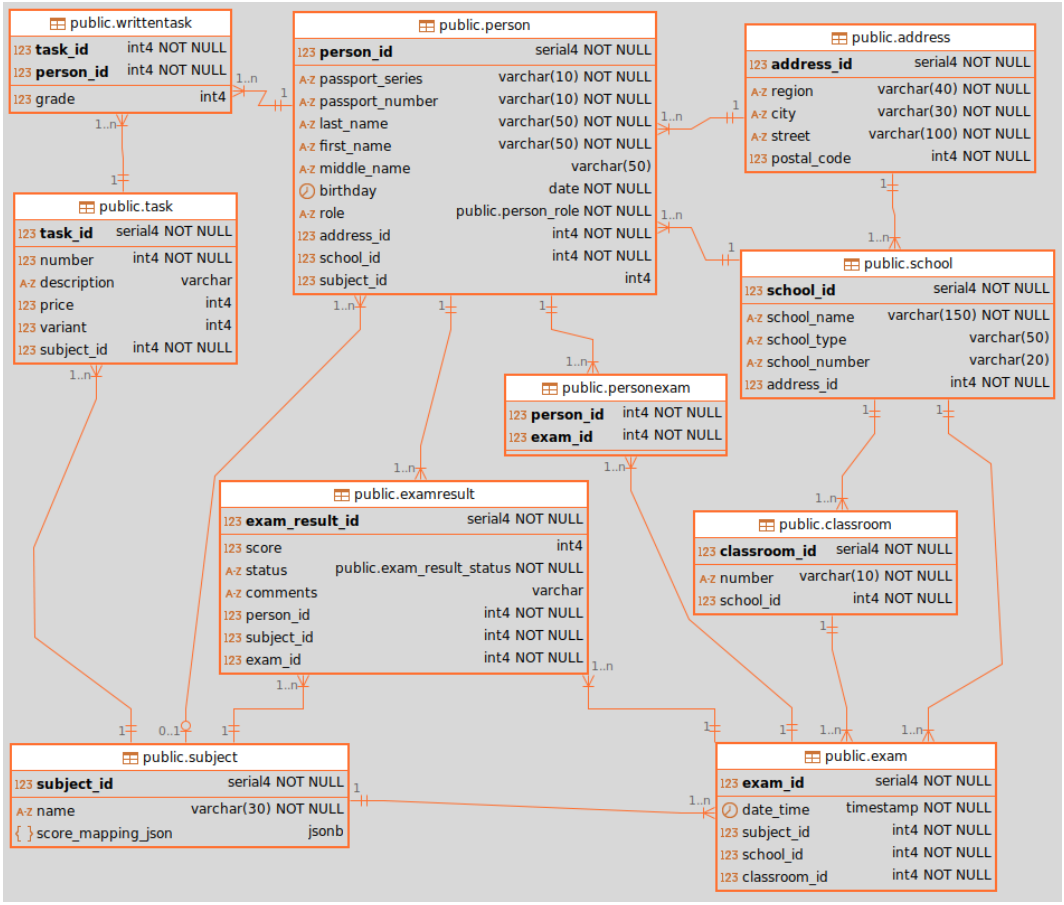


4 Логическое проектирование

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]
```

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

Конфигурация Docker Compose для создания базы данных: 7.1

5.3 Создание таблиц

Код для создания таблиц в базе данных: 7.2

5.4 Заполнение таблиц. ETL-процессы загрузки базы данных

- Файл создания заданий: 7.3.1
- Файл создания информации о школах: 7.3.2
- Файл создания информации об экзаменах: 7.3.3
- Файл создания создания информации об учителях: 7.3.4
- Файл создания создания информации о школах: 7.3.5

5.5 Запросы в терминах SQL

- Получить всех учителей, зарегистрированных на экзамен по предмету ”Name-7.4.1
- Найти всех школьников, не зарегистрированных ни на один экзамен 7.4.2
- Получить список предметов, по которым проводились экзамены в школе Name 7.4.3
- Найти всех учеников, получивших хотя бы одну неудовлетворительную оценку (<40) 7.4.4
- Получить список школ, в которых нет зарегистрированных экзаменов 7.4.5
- Найти всех школьников, у которых по всем предметам, которые он сдавал, 100 баллов 7.4.6
- Вывести средние баллы по предметам по школам 7.4.7
- Вывести учеников с максимальным средним баллом 7.4.8

5.6 Оценка размеров базы данных и каждого из файлов

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

Relation	Attributes	Type	Size (bytes)	Avg Records	Total Size (bytes)	Total Size (MB)
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			
	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			

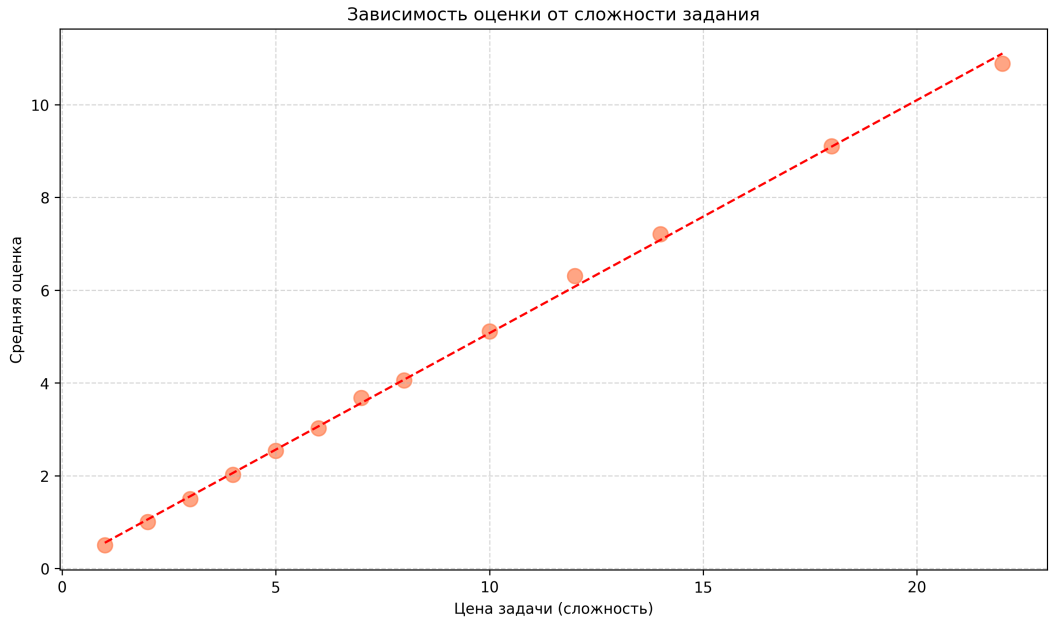
6 Отчеты

6.1 Отчет №1. Отчет о зависимостии оценки от сложности задания

Выполнил: Колесников Михаил Леонидович. Инструментальные средства: matplotlib, python. Дата: 26.05.2025

```
SELECT
      t.price AS task_price,
      AVG(wt.grade) AS average_grade
FROM WrittenTask wt
JOIN Task t ON wt.task_id = t.task_id
GROUP BY t.price
ORDER BY t.price;
```

Листинг 1: 1. Точечная диаграмма: зависимость оценки от сложности задания

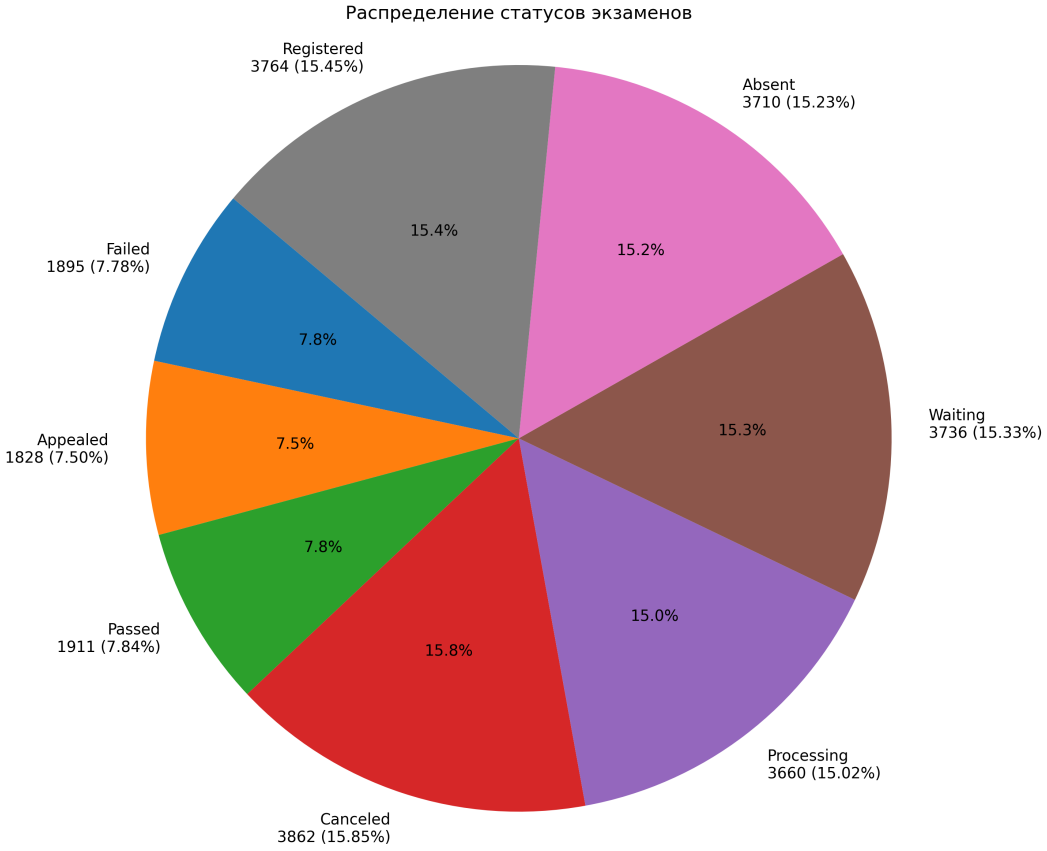


6.2 Отчет №2. Отчет о статусох экзаменов

Выполнил: Колесников Михаил Леонидович. Инструментальные средства: matplotlib, python. Дата: 26.05.2025

```
SELECT
    status,
    COUNT(*) AS count,
    ROUND(COUNT(*) * 100.0 / total.total, 2) AS percentage
FROM ExamResult
CROSS JOIN (SELECT COUNT(*) AS total FROM ExamResult) AS total
GROUP BY status, total.total;
```

Листинг 2: 2. Круговая диаграмма статусов экзаменов

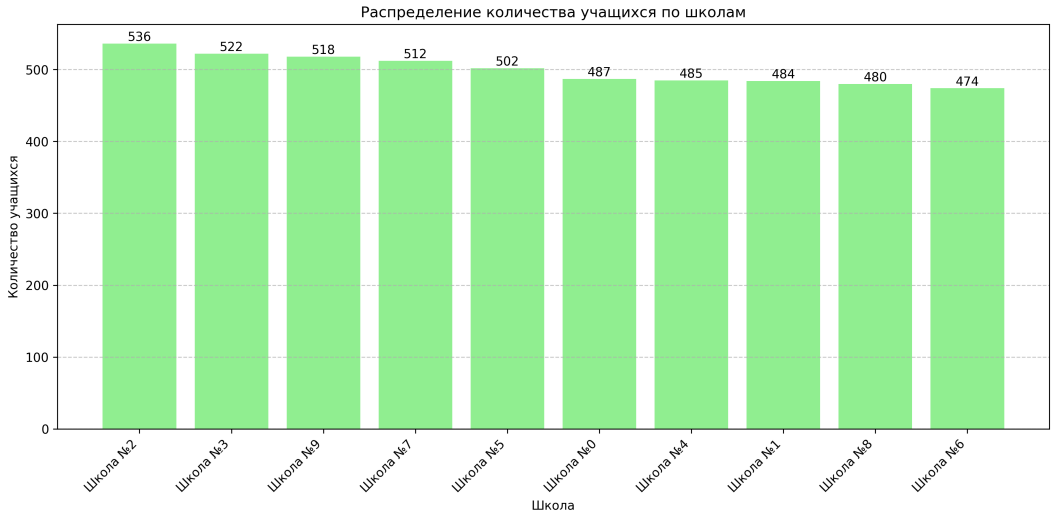


6.3 Отчет №3. Отчет о распределении учащихся по школам

Выполнил: Колесников Михаил Леонидович. Инструментальные средства: matplotlib, python. Дата: 26.05.2025

```
SELECT s.school_name, COUNT(p.person_id) AS student_count
FROM School s
JOIN Person p ON s.school_id = p.school_id
WHERE p.role = 'SchoolChild'
GROUP BY s.school_name
ORDER BY student_count DESC;
```

Листинг 3: 3. Столбчатая диаграмма распределения учащихся по школам



7 Приложение

Приложение 1

7.1 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:
```

Листинг 4: Конфигурация Docker Compose

7.2 Создание таблиц

```
CREATE TABLE Address (
    address_id SERIAL PRIMARY KEY,
    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,
    school_number INTEGER,
    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,
    school_id INTEGER NOT NULL,
    subject_id INTEGER,
    FOREIGN KEY (address_id) REFERENCES Address(address_id) ON DELETE SET
        NULL,
    FOREIGN KEY (school_id) REFERENCES School(school_id),
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id) ON DELETE SET
        NULL
);

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) ON DELETE CASCADE
);

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

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) ON DELETE CASCADE
);

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

```sql
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) ON DELETE CASCADE,
    FOREIGN KEY (subject_id) REFERENCES Subject(subject_id) ON DELETE
        CASCADE,
    FOREIGN KEY (exam_id) REFERENCES Exam(exam_id) ON DELETE CASCADE
);


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) ON DELETE CASCADE,
    FOREIGN KEY (person_id) REFERENCES Person(person_id) ON DELETE CASCADE
);

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) ON DELETE CASCADE,
    FOREIGN KEY (exam_id) REFERENCES Exam(exam_id) ON DELETE CASCADE
);
```

Листинг 5: SQL-запросы для создания таблиц

Приложение 2

7.3 Заполнение таблиц. ETL-процессы загрузки базы данных

7.3.1

```python
import pickle
import psycopg2
from config.config import db_config, number_of_variants_per_subject
from initial_data_preparing.create_subject import
    create_and_get_subjects_data


def generate_tasks():
    def parse_subject_mapping(file_path):
        points = {}

        with open(file_path, 'r', encoding='utf-8') as file:
            current_subject = None
```

```
        for line in file.read().split('\n'):
            if not line:
                continue
            if line[0].isalpha():
                current_subject = line
                points[current_subject] = {}
                continue

            tasks, price = line.split()
            if '-' in tasks:
                task1, task2 = map(int, tasks.split('-'))
            else:
                task1 = task2 = int(tasks)
            for i in range(task1, task2 + 1):
                points[current_subject][i] = price

    return points

def create_tasks(cur, subject_id_to_name, subjects_points,
    number_of_variants: int):
    tasks_data = []
    task_id = 0
    task_id_to_task_data = []
    task_number_subject_to_ids = {}
    for subject_id in subject_id_to_name:
        subject_name = subject_id_to_name[subject_id]
        for task_number in subjects_points[subject_name]:
            for variant_number in range(number_of_variants):
                price = subjects_points[subject_name][task_number]
                description = f"{subject_name}-task"
                tasks_data.append(
                    (task_id, task_number, description, price,
                        variant_number, subject_id)
                )
                task_id_to_task_data.append([task_number, subject_name])

                if (task_number, subject_id) not in
                    task_number_subject_to_ids:
                    task_number_subject_to_ids[task_number, subject_id] =
                        []
                task_number_subject_to_ids[task_number,
                    subject_id].append(task_id)
                task_id += 1
    cur.executemany(
        "insert into task (task_id, number, description, price, variant,
            subject_id) values (%s, %s, %s, %s, %s, %s);",
        tasks_data,
    )
    assert task_id == len(task_id_to_task_data)
    return task_id_to_task_data, task_number_subject_to_ids

def main():
    subjects_points =
```

```
            parse_subject_mapping("initial_data_preparing/subject_data.txt")

        with psycopg2.connect(**db_config) as conn:
            with conn.cursor() as cur:
                subject_id_to_name, subject_name_to_id, subject_id_to_tasks =
                     create_and_get_subjects_data()

                task_id_to_task_data, task_number_subject_to_ids =
                    create_tasks(
                    cur,
                    subject_id_to_name,
                    subjects_points,
                    number_of_variants_per_subject
                )

                with open('subject_id_to_tasks.pkl', 'wb') as f:
                    pickle.dump(subject_id_to_tasks, f)
                with open('task_id_to_task_data.pkl', 'wb') as f:
                    pickle.dump(task_id_to_task_data, f)
                with open('task_number_subject_to_ids.pkl', 'wb') as f:
                    pickle.dump(task_number_subject_to_ids, f)

    main()
```

Листинг 6: python-код для генерации заданий

7.3.2

```
from faker import Faker
import psycopg2

def generate_school():
    def create_addresses_for_schools(cur, faker: Faker, count):
        final_data = []
        for address_id in range(count):
            address = (
                address_id,
                faker.street_name(),
                faker.postcode(),
            )
            final_data.append(address)

        cur.executemany(
            "INSERT INTO Address (address_id, street, postal_code) VALUES
                (%s, %s, %s);",
            final_data,
        )

    def create_schools(cur, faker, school_count):
        create_addresses_for_schools(cur, faker, school_count)
        school_data = []
        for i in range(school_count):
            school_name = f"    №{i}"
            school_data.append(
```

```
                (i, school_name, i, i)
            )
        cur.executemany(
            "INSERT INTO School (school_id, school_name, address_id,
                school_number) VALUES (%s, %s, %s, %s);",
            school_data,
        )

    def create_classrooms(cur, school_count):
        classroom_data = []
        classroom_id = 0
        for school_number in range(school_count):
            for classroom_number in range(500):
                classroom_data.append(
                    (classroom_id, classroom_number, school_number)
                )
                classroom_id += 1
        cur.executemany(
            "INSERT INTO Classroom (classroom_id, number, school_id) VALUES
                (%s, %s, %s);",
            classroom_data
        )

    def main():
        from config.config import db_config, school_count
        faker = Faker("ru_RU")

        with psycopg2.connect(**db_config) as conn:
            with conn.cursor() as cur:
                create_schools(cur, faker, school_count)
                create_classrooms(cur, school_count)


    main()
```

Листинг 7: python-код для генерации школ

7.3.3

```
from faker import Faker
import psycopg2


def generate_exams():
    def create_exams(db_config, faker: Faker, subject_count,
        classroom_count):
        with psycopg2.connect(**db_config) as conn:
            with conn.cursor() as cur:
                exam_data = []
                exam_id = 0
                for classroom_number in range(classroom_count):
                    for subject_id in range(subject_count):
                        exam_data.append(
                            (
```

```
                                exam_id,
                                faker.date_time_between(start_date="-1y",
                                    end_date="now"),
                                subject_id,
                                classroom_number
                            )
                        )
                        exam_id += 1

                cur.executemany(
                    "insert into exam (exam_id, date_time, subject_id,
                        classroom_id) values (%s, %s, %s, %s);",
                    exam_data,
                )
        return exam_data

    def main():
        from config.config import db_config, SUBJECT_COUNT

        faker = Faker("ru_RU")
        exam_data = create_exams(db_config, faker, SUBJECT_COUNT, 500)

    main()
```

Листинг 8: python-код для генерации экзаменов

7.3.4

```
from faker import Faker
import psycopg2
import random


def generate_teachers():
    def create_teachers(
            db_config,
            faker: Faker,
            teacher_count,
            subject_count,
            school_count):
        with psycopg2.connect(**db_config) as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT exam_id, subject_id from exam;")
                rows = cur.fetchall()
                exam_ids = [(row[0], row[1]) for row in rows]

        teachers_data = []
        addresses_data = (
            (
                address_id,
                faker.street_name(),
                faker.postcode(),
            ) for address_id in range(school_count, school_count +
                teacher_count)
```

```
)
teacher_with_exam_data = []

address_id = school_count
teacher_id = 0
for teacher_number in range(teacher_count):
    school_id = random.randint(0, school_count - 1)
    subject_id = random.randint(0, subject_count - 1)
    teachers_data.append(
        [
            teacher_id,
            str(random.randint(1000, 9999)),
            str(random.randint(100000, 999999)),
            faker.last_name(),
            faker.first_name(),
            faker.middle_name(),
            faker.date_of_birth().strftime('%Y-%m-%d'),
            'Teacher',
            address_id,
            school_id,
            subject_id,
        ]
    )

    for ex_id, sub_id in random.sample(exam_ids, 10):
        if sub_id == subject_id:
            continue
        teacher_with_exam_data.append(
            (
                ex_id,
                teacher_id
            )
        )

    address_id += 1
    teacher_id += 1

with psycopg2.connect(**db_config) as conn:
    with conn.cursor() as cur:
        cur.executemany(
            "INSERT INTO Address (address_id, street, postal_code)
                VALUES (%s, %s, %s);",
            addresses_data,
        )
        cur.executemany(
            """INSERT INTO Person
            (person_id, passport_series, passport_number, last_name,
                first_name, middle_name, birthday, role, address_id,
                school_id, subject_id)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);""",
            teachers_data,
        )
        cur.executemany(
```

```
                    "insert into PersonExam (exam_id, person_id) values (%s,
                        %s);",
                    teacher_with_exam_data,
                )

    def main():
        from config.config import db_config, teacher_count, SUBJECT_COUNT,
            school_count
        faker = Faker("ru_RU")
        create_teachers(db_config, faker, teacher_count, SUBJECT_COUNT,
            school_count)

    main()
```

Листинг 9: python-код для генерации учителей

7.3.5

```
from faker import Faker
import random
import psycopg2
import pickle

from config.config import teacher_count
from initial_data_preparing.create_subject import parse_subject_mapping


def create_school_children(
        db_config,
        faker: Faker,
        school_count,
        school_children_count,
        task_number_subject_to_ids,
        subject_id_to_tasks,
        address_id_first
):
    subject_points =
        parse_subject_mapping("initial_data_preparing/subject_data.txt")

    exam_statuses = [
        'Registered',
        'Passed',
        'Failed',
        'Absent',
        'Appealed',
        'Canceled',
        'Processing',
        'Waiting'
    ]
    exam_result_data = []
    written_task_data = []

    addresses_data = tuple(
        (
```

```
        school_children_id,
        faker.street_name(),
        faker.postcode()
    ) for school_children_id in range(address_id_first, address_id_first
       + school_children_count)
)
with psycopg2.connect(**db_config) as conn:
    with conn.cursor() as cur:
        cur.executemany(
            "INSERT INTO Address (address_id, street, postal_code)
                VALUES (%s, %s, %s);",
            addresses_data,
        )
del addresses_data

school_children_data = tuple(
    (
        school_children_id,
        str(random.randint(1000, 9999)),
        str(random.randint(100000, 999999)),
        faker.last_name(),
        faker.first_name(),
        faker.middle_name(),
        faker.date_of_birth().strftime('%Y-%m-%d'),
        'SchoolChild',
        school_children_id,
        random.randint(0, school_count - 1),
        None,
    ) for school_children_id in range(address_id_first, address_id_first
       + school_children_count)
)
with psycopg2.connect(**db_config) as conn:
    with conn.cursor() as cur:
        cur.executemany(
            """INSERT INTO Person
            (person_id, passport_series, passport_number, last_name,
                first_name, middle_name, birthday, role, address_id,
                school_id, subject_id)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);""",
            school_children_data,
        )
del school_children_data

with psycopg2.connect(**db_config) as conn:
    with conn.cursor() as cur:
        cur.execute("SELECT subject_id, name FROM Subject")
        rows = cur.fetchall()
        subject_id_to_name = {row[0]: row[1] for row in rows}
        cur.execute("SELECT exam_id from exam where subject_id=2;")
        rows = cur.fetchall()
        rus_exam_ids = [row[0] for row in rows]
        cur.execute("SELECT exam_id from exam where subject_id=0;")
        rows = cur.fetchall()
```

```
        math_base_exam_ids = [row[0] for row in rows]
        cur.execute("SELECT exam_id from exam where subject_id=1;")
        rows = cur.fetchall()
        math_prof_exam_ids = [row[0] for row in rows]
        cur.execute("SELECT exam_id, subject_id from exam where
            subject_id > 2;")
        rows = cur.fetchall()
        third_exam_ids = [(row[0], row[1]) for row in rows]

schoolchild_with_exam_data = tuple(
    (
        school_children_id,
        ex_id,
    )
    for school_children_id in range(address_id_first, address_id_first +
        school_children_count)
    for ex_id in (
        random.choice(rus_exam_ids),
        random.choice(math_prof_exam_ids) if random.getrandbits(1) else
            random.choice(math_base_exam_ids),
        random.choice(third_exam_ids)[0]
    )
)

for school_children_id, ex_id in schoolchild_with_exam_data:
    sub_id = (ex_id - 1) % 16
    exam_status = random.choice(exam_statuses)
    if exam_status not in ("Passed", "Failed", "Appealed"):
        exam_result_data.append((
            0,
            exam_status,
            "some comments",
            school_children_id,
            sub_id,
            ex_id
        ))

    cur_task_data = [
        (
            random.choice(task_number_subject_to_ids[task_number,
                sub_id]),
            school_children_id,
            random.randint(0,
                int(subject_points[subject_id_to_name[sub_id]][task_number]))
        ) for task_number in range(1, len(subject_id_to_tasks[sub_id]) +
            1)
    ]
    written_task_data += cur_task_data

    exam_result_data.append((
        sum(i[0] for i in cur_task_data),
        exam_status,
        "some comments",
```

```
            school_children_id,
            sub_id,
            ex_id
        ))

    with psycopg2.connect(**db_config) as conn:
        with conn.cursor() as cur:
            cur.executemany(
                "insert into ExamResult (score, status, comments, person_id,
                    subject_id, exam_id) values (%s, %s, %s, %s, %s, %s);",
                exam_result_data,
            )
            cur.executemany(
                "insert into PersonExam (person_id, exam_id) values (%s,
                    %s);",
                schoolchild_with_exam_data,
            )

            cur.executemany(
                "insert into WrittenTask (task_id, person_id, grade) values
                    (%s, %s, %s);",
                written_task_data,
            )
            conn.commit()



def generate_schoolchildren():
    from config.config import db_config, school_children_count,
        school_count, teacher_count
    faker = Faker("ru_RU")

    with open('subject_id_to_tasks.pkl', 'rb') as f:
        subject_id_to_tasks = pickle.load(f)
    with open('task_number_subject_to_ids.pkl', 'rb') as f:
        task_number_subject_to_ids = pickle.load(f)

    create_school_children(
        db_config,
        faker,
        school_count,
        school_children_count,
        task_number_subject_to_ids,
        subject_id_to_tasks,
        school_count + teacher_count
    )
```

Листинг 10: python-код для генерации школьников

Приложение 3

7.4 Запросы в терминах SQL

7.4.1

```
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';
```

Листинг 11: SQL-запрос 1

7.4.2

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

Листинг 12: SQL-запрос 2

7.4.3

```
SELECT DISTINCT s.subject_id
FROM Exam e
JOIN Classroom c ON e.classroom_id = c.classroom_id
JOIN Subject s ON e.subject_id = s.subject_id
JOIN School sc ON c.school_id = sc.school_id
WHERE sc.school_name = '    №0';
```

Листинг 13: SQL-запрос 3

7.4.4

```
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;
```

Листинг 14: SQL-запрос 4

7.4.5

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

Листинг 15: SQL-запрос 5

7.4.6

```
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
);
```

Листинг 16: SQL-запрос 6

7.4.7

```
SELECT
    s.school_name,
    sub.name AS subject_name,
    COUNT(er.exam_result_id) AS exams_taken,
    AVG(er.score) AS avg_score,
    MAX(er.score) AS max_score
FROM
    ExamResult er
    JOIN Subject sub ON er.subject_id = sub.subject_id
    JOIN Person p ON er.person_id = p.person_id
    JOIN School s ON p.school_id = s.school_id
GROUP BY
    s.school_name, sub.name
HAVING
    AVG(er.score) > 70 AND
    COUNT(er.exam_result_id) > 5
ORDER BY
    avg_score DESC;
```

Листинг 17: SQL-запрос 7

7.4.8

```
WITH student_rank AS (
    SELECT
        p.person_id,
        AVG(er.score) AS average_score,
        DENSE_RANK() OVER (ORDER BY AVG(er.score) DESC) AS rank
    FROM
        Person p
        JOIN ExamResult er ON p.person_id = er.person_id
    WHERE
```

```
        p.role = 'SchoolChild'
        AND er.status = 'Passed'
    GROUP BY
        p.person_id
)
SELECT
    person_id,
    ROUND(average_score, 2)
FROM
    student_rank
WHERE
    rank = 1;
```

Листинг 18: SQL-запрос 8