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](#_bookmark0) 4
   1. [Формулировка задания](#_bookmark1) 4
   2. [Конкретизация предметной области](#_bookmark2) 4
   3. [Пользователи системы](#_bookmark3) 4
   4. [Сроки хранения информации](#_bookmark4) 4
   5. [События, изменяющие состояние базы данных](#_bookmark5) 4
   6. [Основные запросы к базе данных (на естественном языке)](#_bookmark6) 5
2. [Концептуально-информационная модель предметной области](#_bookmark7) 6
   1. [Er-диаграмма модели](#_bookmark8) 6
   2. [Оценка мощностных характеристик сущностей и связей](#_bookmark9) 7
      1. [Оценка мощностных характеристик сущностей](#_bookmark10) 7
      2. [Оценка мощностных характеристик связей](#_bookmark11) 7
3. [Концептуальное проектирование](#_bookmark12) 8
   1. [Принятые проектные соглашения](#_bookmark13) 8
   2. [Обоснование выбора модели базы данных](#_bookmark14) 8
   3. [Используемые в системе кодификаторы](#_bookmark15) 9
   4. [Концептуальная модель базы данных](#_bookmark16) 10
4. [Логическое проектирование](#_bookmark17) 11
   1. [Er-диаграмма базы данных (logical)](#_bookmark18) 11
   2. [Схемы отношений базы данных (physical)](#_bookmark19) 12
   3. [Схема реляционной базы данных](#_bookmark20) 13
   4. [Схемы основных запросов на реляционной алгебре](#_bookmark21) 13
5. [Физическое проектирование](#_bookmark22) 14
   1. [Обоснование выбора конкретной СУБД](#_bookmark23) 14
   2. [Создание базы данных](#_bookmark24) 14
   3. [Создание таблиц](#_bookmark25) 14
   4. [Заполнение таблиц. ETL-процессы загрузки базы данных](#_bookmark26) 14
   5. [Запросы в терминах SQL](#_bookmark27) 15
   6. [Оценка размеров базы данных и каждого из файлов](#_bookmark28) 15
6. [Отчеты](#_bookmark29) 17
   1. [Отчет №1. Отчет о зависимостии оценки от сложности задания](#_bookmark30) 17
   2. [Отчет №2. Отчет о статусох экзаменов](#_bookmark31) 18
   3. [Отчет №3. Отчет о распределении учащихся по школах](#_bookmark32) 19
7. [Приложение](#_bookmark33) 20
   1. [Создание базы данных](#_bookmark34) 20
   2. [Создание таблиц](#_bookmark35) 20
   3. [Заполнение таблиц. ETL-процессы загрузки базы данных](#_bookmark36) 22
      1. 22
      2. [24](#_TOC_250003)
      3. [25](#_TOC_250002)
      4. [26](#_TOC_250001)
      5. [28](#_TOC_250000)
   4. [Запросы в терминах SQL](#_bookmark42) 32
      1. 32
      2. 32
      3. 32
      4. 32
      5. 32
      6. 33
      7. 33
      8. 33

# Описание предметной области

## 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.

## 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 образо- вания 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.

## Пользователи системы

#### Основными пользователями системы являются:

* **Администраторы школ** - управляют данными учебных заведений, аудиториями, расписа- нием 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

## 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

## 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)

## 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

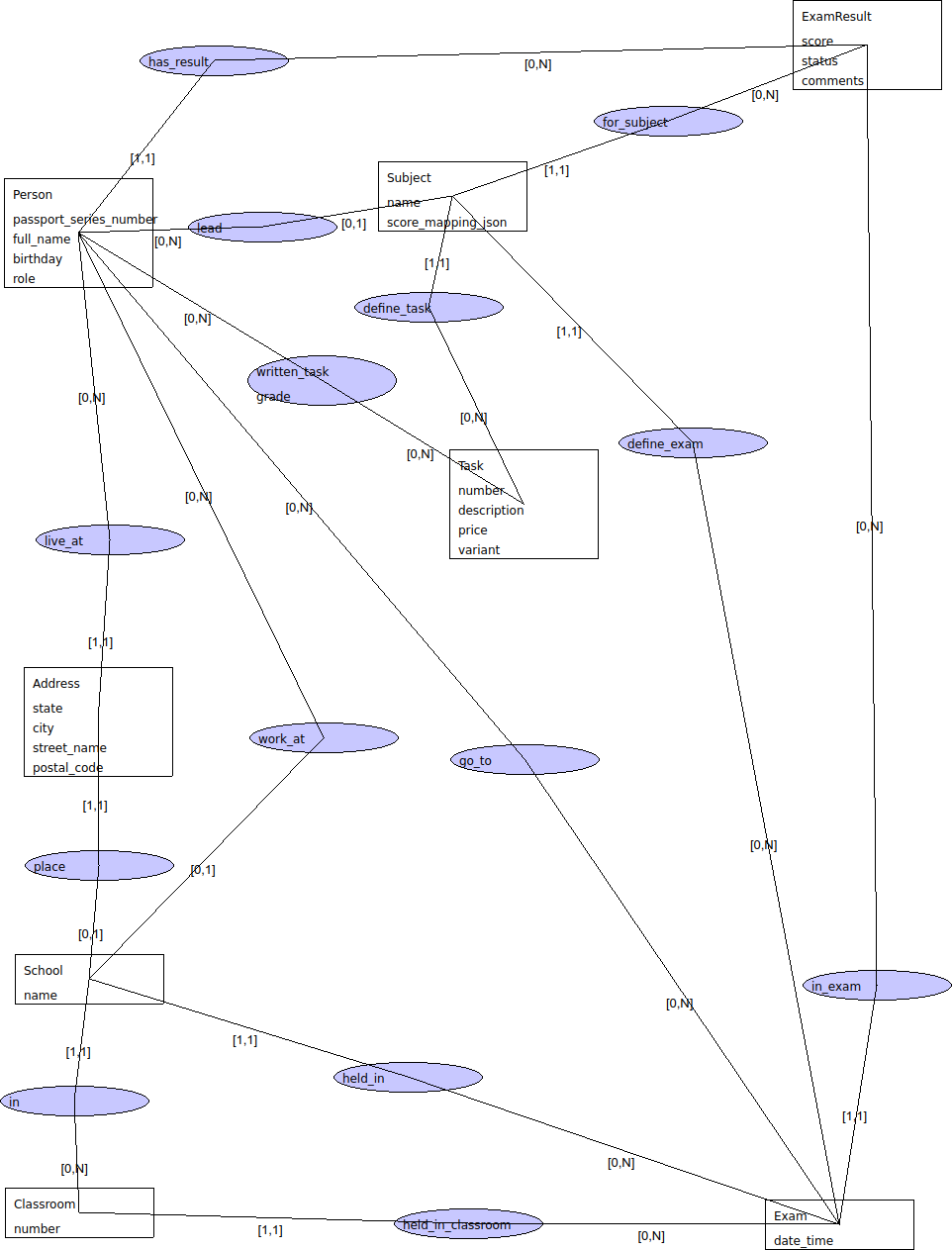
#### Найти всех учащихся, проживающих в указанном городе/районе

#### Получить список заданий для конкретного варианта экзамена

#### Показать статистику сдачи экзаменов по школам за указанный период

# Концептуально-информационная модель предметной области

## Er-диаграмма модели

****

## Оценка мощностных характеристик сущностей и связей

### Оценка мощностных характеристик сущностей

* + - * **Сущность Address**:

#### Ожидаемое количество: 251,000

#### Формула: 1000школ + 500000учеников + 150000учителей

* + - * **Сущность School**:

#### Ожидаемое количество: 1,000

* + - * **Сущность Man**:

#### Ожидаемое количество: 650,000

#### Формула: 100000учеников *×* 5лет + 150000учителей

* + - * **Сущность Subject**:

#### Ожидаемое количество: 16

* + - * **Сущность Exam**:

#### Ожидаемое количество: 8,000

Формула: 16предметов *×* 100школ/год *×* 5лет

* + - * **Сущность ExamResult**:

#### Ожидаемое количество: 1,500,000

Формула: 100000учеников/год *×* 3предмета *×* 5лет

* + - * **Сущность Task**:

#### Ожидаемое количество: 48,000

Формула: 16предметов *×* 30заданий *×* 5лет *×* 20вариантов

* + - * **Сущность Classroom**:
        + Ожидаемое количество: 20,000 Формула: 20аудиторий/школу *×* 1000школ

### Оценка мощностных характеристик связей

* + - * **written\_task**:
        + Ожидаемое количество: 3,000,000 Формула: 1*.*5Mрезультатов *×* 30заданий
      * **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)

# Концептуальное проектирование

## 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

## 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

## Используемые в системе кодификаторы

* **Кодификатор предметов (Subject)**:

#### Цифровой код (например, 01 - Математика, 02 - Физика)

#### Соответствует официальным кодам ЕГЭ

* **Кодификатор статусов экзамена (ExamResult)**:

#### Registered - Зарегистрирован на экзамен

#### Passed - Экзамен сдан успешно

#### Failed - Экзамен не сдан

#### Absent - Отсутствовал на экзамене

#### Appealed - Подана апелляция

#### Canceled - Экзамен отменен

#### Processing - В процессе проверки

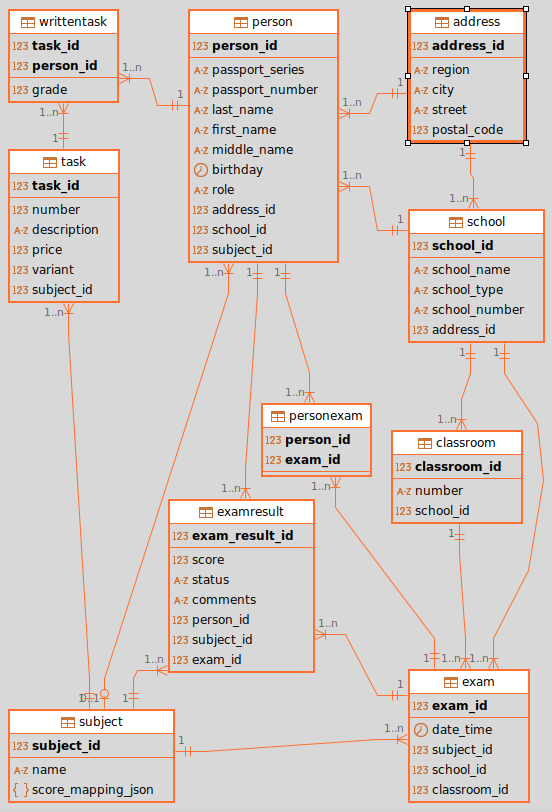
#### Waiting - Ожидает проверки

* **Типы ролей персон (person\_role):**

#### SchoolChild - Ученик школы

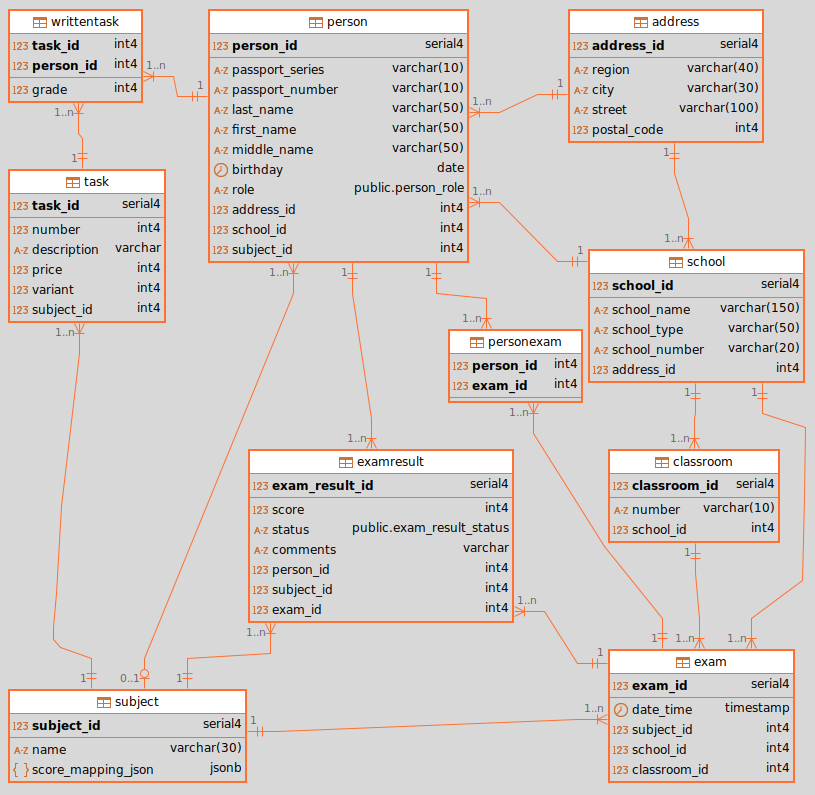
#### Teacher - Учитель

## Концептуальная модель базы данных

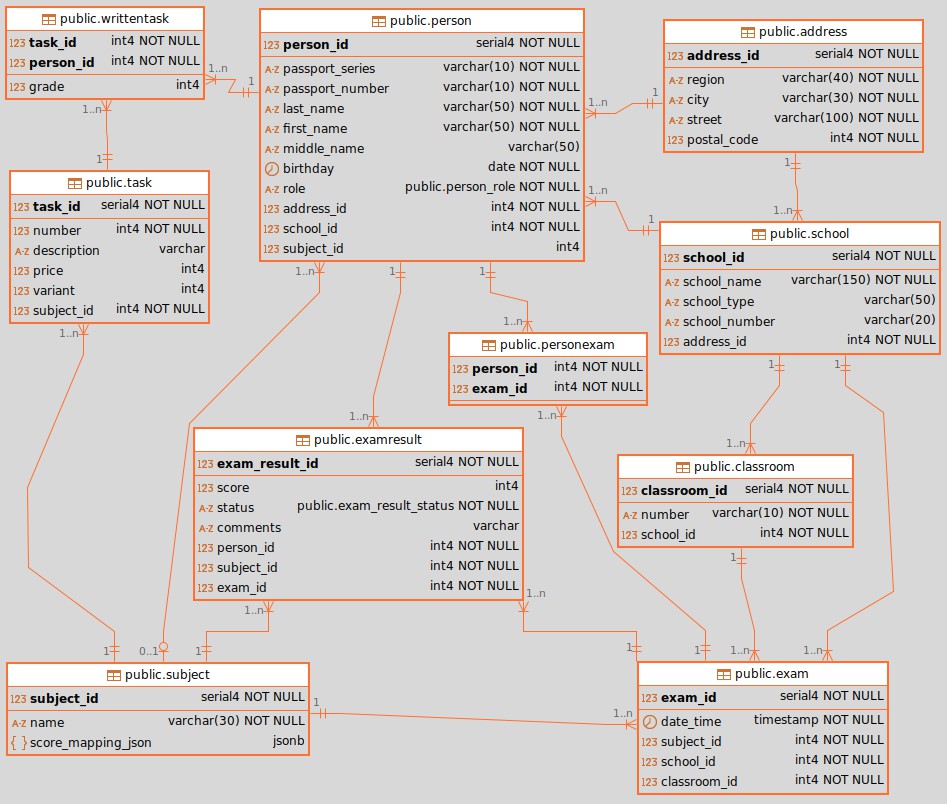
****

# Логическое проектирование

## Er-диаграмма базы данных (logical)

****

## Схемы отношений базы данных (physical)

****

## 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 |

## 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]

# Физическое проектирование

## 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, кото- рая 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 пользова- тельских 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 практи- ческой точек зрения.

## Создание базы данных

#### Конфигурация Docker Compose для создания базы данных: [7.1](#_bookmark34)

## Создание таблиц

#### Код для создания таблиц в базе данных: [7.2](#_bookmark35)

## Заполнение таблиц. ETL-процессы загрузки базы данных

#### Файл создания заданий: [7.3.1](#_bookmark37)

#### Файл создания информации о школах: [7.3.2](#_bookmark38)

#### Файл создания информации об экзаменах: [7.3.3](#_bookmark39)

#### Файл создания создания информации об учителях: [7.3.4](#_bookmark40)

#### Файл создания создания информации о школах: [7.3.5](#_bookmark41)

## Запросы в терминах SQL

#### Получить всех учителей, зарегистрированных на экзамен по предмету ”Name-[7.4.1](#_bookmark43)

#### Найти всех школьников, не зарегистрированных ни на один экзамен [7.4.2](#_bookmark44)

#### Получить список предметов, по которым проводились экзамены в школе Name [7.4.3](#_bookmark45)

#### Найти всех учеников, получивших хотя бы одну неудовлетворительную оценку (<40) [7.4.4](#_bookmark46)

#### Получить список школ, в которых нет зарегистрированных экзаменов [7.4.5](#_bookmark47)

#### Найти всех школьников, у которых по всем предметам, которые он сдавал, 100 баллов [7.4.6](#_bookmark48)

#### Вывести средние баллы по предметам по школам [7.4.7](#_bookmark49)

#### Вывести учеников с максимальным средним баллом [7.4.8](#_bookmark50)

## Оценка размеров базы данных и каждого из файлов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Relation** | **Attributes** | **Type** | **Size (bytes)** | **Avg Records** | **Total Size (bytes)** | **Total Size (MB)** |
|  | address\_id | SERIAL | 4 |  |  |  |
|  | region | VARCHAR(40) | 40 |  |  |  |
| Address | city | VARCHAR(30) | 30 | 251,000 | 44,678,000 | 42.6 |
|  | street | VARCHAR(100) | 100 |  |  |  |
|  | postal\_code | INTEGER | 4 |  |  |  |
|  | school\_id | SERIAL | 4 |  |  |  |
| School | school\_name | VARCHAR(150) | 150 | 1,000 | 158,000 | 0.15 |
|  | school\_number | INTEGER | 4 |  |  |  |
|  | address\_id | INTEGER | 4 |  |  |  |
|  | person\_id | SERIAL | 4 |  |  |  |
|  | passport\_series | VARCHAR(10) | 10 |  |  |  |
|  | passport\_- | VARCHAR(10) | 10 |  |  |  |
|  | number |  |  |  |  |  |
|  | last\_name | VARCHAR(50) | 50 |  |  |  |
| Person | first\_name | VARCHAR(50) | 50 | 650,000 | 123,500,000 | 117.8 |
|  | 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\_id | SERIAL | 4 |  |  |  |
| Subject | name | VARCHAR(30) | 30 | 16 | 2,144 | 0.002 |
|  | score\_- | JSONB | 100 |  |  |  |
|  | mapping\_json |  |  |  |  |  |
|  | exam\_id | SERIAL | 4 |  |  |  |
|  | date\_time | TIMESTAMP | 8 |  |  |  |
| Exam | subject\_id | INTEGER | 4 | 8,000 | 192,000 | 0.18 |
|  | school\_id | INTEGER | 4 |  |  |  |
|  | classroom\_id | INTEGER | 4 |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Relation** | **Attributes** | | **Type** | **Size (bytes)** | | **Avg Records** | | **Total Size (bytes)** | | **Total Size (MB)** | |
|  | exam\_result\_id | | SERIAL | 4 | |  | |  | |  | |
|  | score | | INTEGER | 4 | |  | |  | |  | |
|  | status | | exam\_result\_- | 10 | |  | |  | |  | |
| ExamResult |  | | status |  | | 1,500,000 | | 195,000,000 | | 186.0 | |
|  | comments | | VARCHAR | 100 | |  | |  | |  | |
|  | person\_id | | INTEGER | 4 | |  | |  | |  | |
|  | subject\_id | | INTEGER | 4 | |  | |  | |  | |
|  | exam\_id | | INTEGER | 4 | |  | |  | |  | |
|  | task\_id | | SERIAL | 4 | |  | |  | |  | |
|  | number | | INTEGER | 4 | |  | |  | |  | |
| Task | description  price | VARCHAR  INTEGER | | | 100  4 | | 48,000 | | 5,760,000 | | 5.49 |
|  | variant | INTEGER | | | 4 | |  | |  | |  |
|  | subject\_id | INTEGER | | | 4 | |  | |  | |  |
|  | classroom\_id | SERIAL | | | 4 | |  | |  | |  |
| Classroom | number | VARCHAR(10) | | | 10 | | 20,000 | | 360,000 | | 0.34 |
|  | school\_id | INTEGER | | | 4 | |  | |  | |  |
|  | task\_id | INTEGER | | | 4 | |  | |  | |  |
| WrittenTask | person\_id | INTEGER | | | 4 | | 3,000,000 | | 36,000,000 | | 34.33 |
|  | grade | INTEGER | | | 4 | |  | |  | |  |
| PersonExam person\_id | | INTEGER | | | 4 1,500,000 12,000,000 11.44 | | | | | | |
| exam\_id | | INTEGER | | | 4 | | | | | | |

# Отчеты

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

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

SELECT

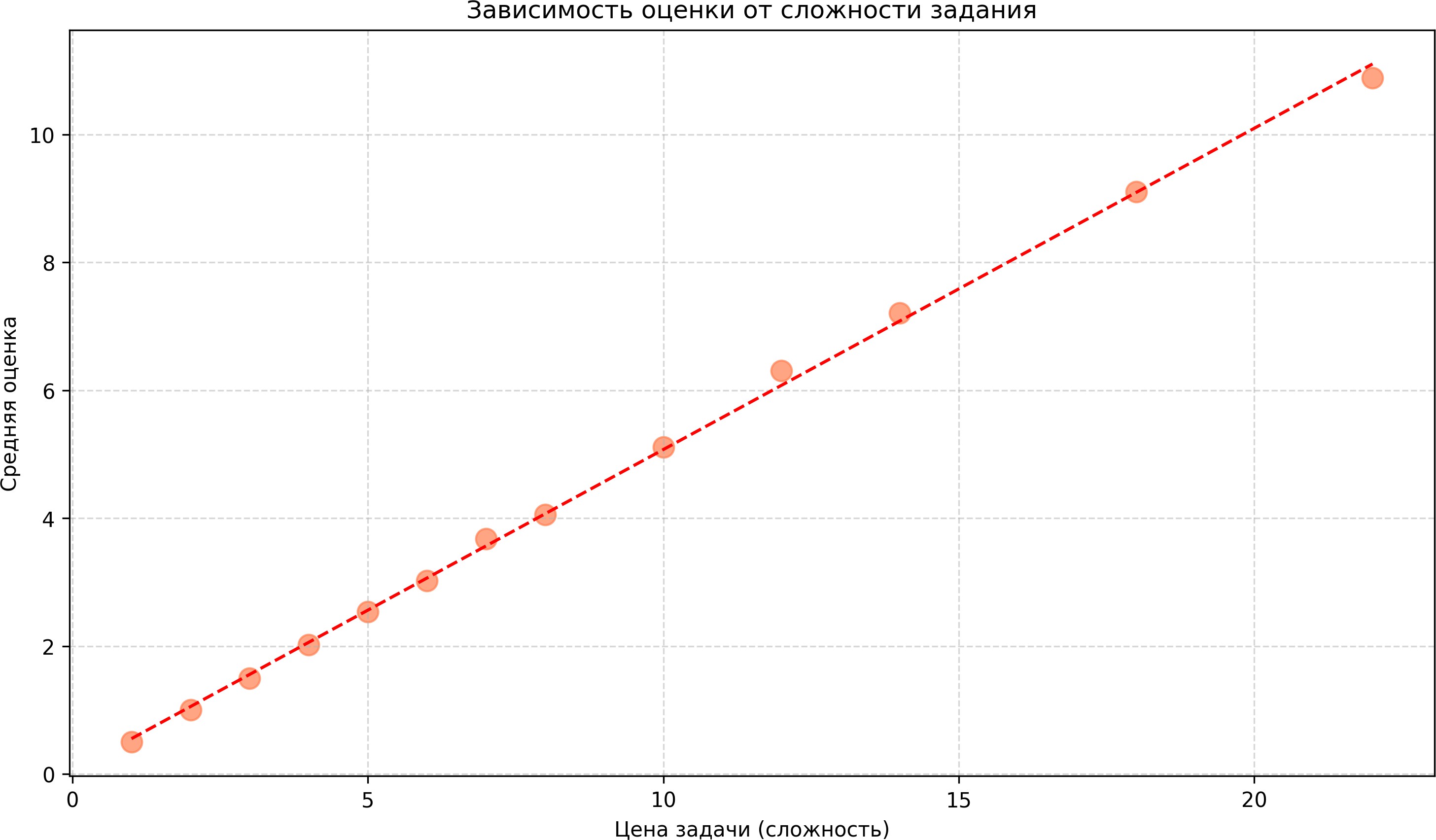
t.price AS task\_price , AVG(wt.grade) AS average\_grade

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



## Отчет №2. Отчет о статусох экзаменов

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

SELECT

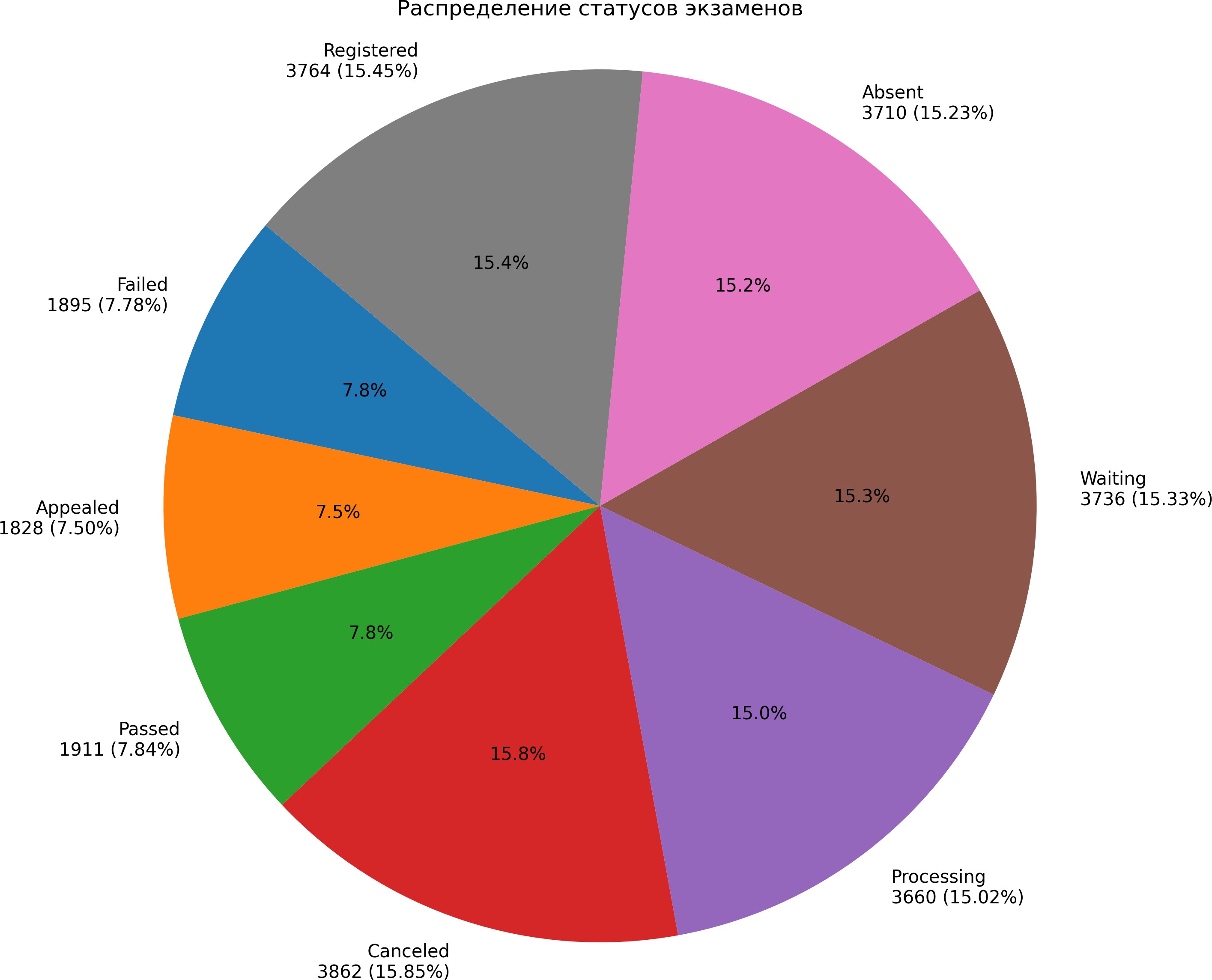
status ,

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



## Отчет №3. Отчет о распределении учащихся по школах

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

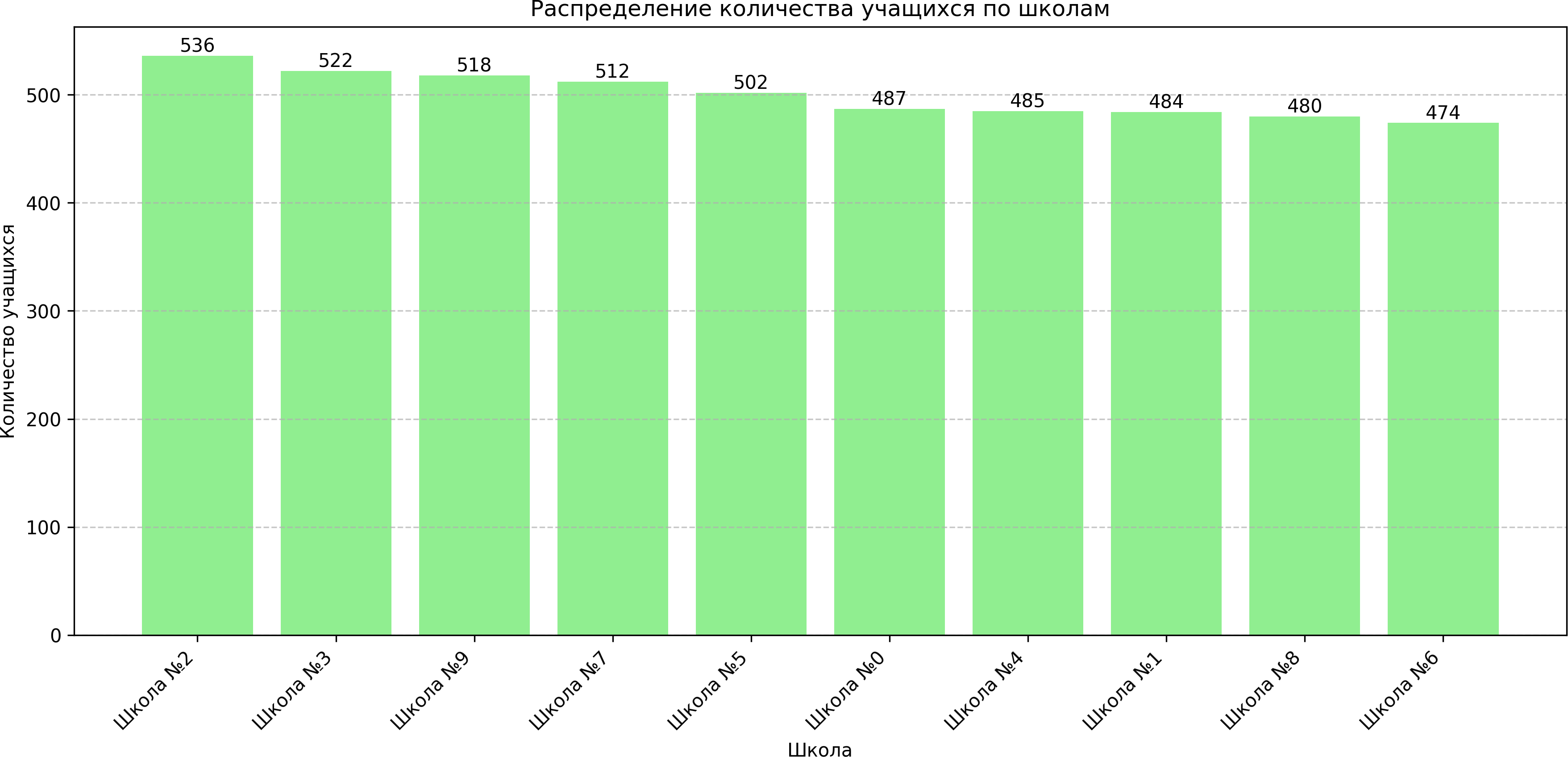
SELECT s.school\_name , COUNT(p. person\_id) AS student\_count FROM School s

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



# Приложение

## 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:

**Приложение 1**

#### Листинг 4: Конфигурация Docker Compose

## 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'

);

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**

## Заполнение таблиц. ETL-процессы загрузки базы данных



**import** pickle

**import** psycopg 2

**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:

task 1, task 2 = map(int, tasks.split('-'))

##### else :

task 1 = task 2 = int(tasks)

**for** i **in** range(task 1, task 2 + 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** psycopg 2. 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

### 7.3.2

main()

)

**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)

#### Листинг 6: python-код для генерации заданий

**from** faker **import** Faker

**import** psycopg 2

**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** psycopg 2. connect(\*\* db\_config) **as** conn:

**with** conn.cursor() **as** cur: create\_schools(cur, faker, school\_count) create\_classrooms(cur, school\_count)

### 7.3.3

main()

#### Листинг 7: python-код для генерации школ

**from** faker **import** Faker

**import** psycopg 2

**def** generate\_exams():

**def** create\_exams(db\_config , faker: Faker, subject\_count , classroom\_count):

**with** psycopg 2. 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** psycopg 2

**import** random

**def** generate\_teachers():

**def** create\_teachers(

db\_config , faker: Faker, teacher\_count , subject\_count , school\_count):

**with** psycopg 2. 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** psycopg 2. 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(

)

**def** main():

"insert into PersonExam (exam\_id , person\_id) values (%s,

%s);",

teacher\_with\_exam\_data ,

**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)

### 7.3.5

main()

#### Листинг 9: python-код для генерации учителей

**from** faker **import** Faker

**import** random **import** psycopg 2 **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** psycopg 2. 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** psycopg 2. 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** psycopg 2. 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** psycopg 2. 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**

## Запросы в терминах 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';

#### Листинг 11: SQL-запрос 1



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



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



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



SELECT s. school\_id FROM School s

WHERE s. school\_id NOT IN (SELECT school\_id FROM Exam);

#### Листинг 15: SQL-запрос 5



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



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



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