МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Введение в нереляционные базы данных»

Тема: Информационная система оценок и аналитика проблемных студентов

Студент гр. 2381 _____ Ильясов М.Р.

Студент гр. 2381 _____ Комосский Е.А.

Студент гр. 2381 _____ Кузнецов И.И.

Студент гр. 2381 _____ Мавликаев И.С.

Студент гр. 2381 Рыжиков И.А.

Преподаватель _____ Заславский М.М.

Санкт-Петербург
2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Ильясов М.Р. 2381

Студент Комосский Е.А. 2381

Студент Кузнецов И.И. 2381

Студент Мавликаев И.С. 2381

Студент Рыжиков И.А. 2381

Тема работы: Информационная система оценок и аналитика проблемных студентов

Исходные данные:

Задача - организовать сервис, где будут агрегироваться все действия студента в ВУЗе (учебные активности, посещаемость ВУЗа, участие в социальной активности ….), и на их основе будут строится прогнозы по успеваемости, рисках для студента / группы / потока.

Содержание пояснительной записки:

«Содержание», «Введение», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Литература»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 05.02.2025

Дата сдачи реферата: 29.05.2025

Дата защиты реферата: 29.05.2025

Студент гр. 2381	_____	Ильясов М.Р.
Студент гр. 2381	_____	Комосский Е.А.
Студент гр. 2381	_____	Кузнецов И.И.
Студент гр. 2381	_____	Мавликаев И.С.
Студент гр. 2381		Рыжиков И.А.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

SUMMARY

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность решаемой проблемы

Современные образовательные учреждения сталкиваются с необходимостью оперативного анализа учебной активности студентов для своевременного выявления проблем в обучении и принятия обоснованных управленческих решений. Актуальность данной задачи обусловлена ростом объёмов образовательных данных и потребностью в их интеграции из различных источников: посещаемость занятий, оценки, участие в социальной и научной деятельности.

Постановка задачи

В рамках курсовой работы поставлена задача разработки информационной системы, позволяющей агрегировать данные о действиях студентов в процессе обучения, обеспечивать их централизованное хранение и использовать для построения прогнозов по успеваемости и оценке рисков как для отдельных студентов, так и для учебных групп и потоков.

Предлагаемое решение

Предлагаемое решение представляет собой веб-сервис, основанный на базе данных MongoDB, с интерфейсами для работы с пользователями, студентами, преподавателями и учебными дисциплинами, а также с модулем аналитики, предусматривающим расширение для прогнозирования успеваемости.

Качественные требования к решению

К качественным требованиям к системе относятся:

- масштабируемость (возможность обработки больших объёмов данных),
- расширяемость архитектуры,
- удобство пользовательского интерфейса,
- корректность хранения и отображения информации,

- возможность подключения дополнительных источников данных в будущем.

СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

1. Просмотр статистики и истории действий

Акторы:

- Студент
- Староста
- Преподаватель
- Администратор

Предусловия:

- Пользователь имеет доступ к системе.
- Данные о статистике и истории действий загружены.

Основной поток:

1. Пользователь заходит в систему.
2. Открывает раздел статистики.
3. Выбирает интересующий его период или группу (если доступно).
4. Получает визуализацию данных (графики, проценты, средний балл и т. д.).

Альтернативные потоки:

- Если данные отсутствуют, отображается сообщение о недоступности информации.
- Если у пользователя нет прав на просмотр, выводится уведомление об ограничении доступа.

Дополнительно для студента:

- У студента есть более подробный профиль, содержащий расширенную информацию о нём.
- Другие пользователи (например, преподаватели и администраторы) могут просматривать профиль студента.

Общая таблица статистики пользователей:

- Отображается таблица со статистикой всех пользователей.
- Минимальный набор фильтров: группа, год рождения (если применимо), вероятность отчисления, средний и предсказанный баллы.
- Если интерфейс позволяет, дополнительно: посещаемость, оценки, активности.
- Возможность сортировки и фильтрации данных по ключевым категориям.

Отображение сложных данных:

- Для сложных данных в таблице применяется механизм раскрывающихся блоков (аккордеонов), аналогично блокам кода в вебе.
- В общем виде отображаются ключевые метрики (например, средний балл, посещаемость, вероятность отчисления).
- При раскрытии блока показываются детализированные данные (оценки по предметам, активности, индивидуальные показатели).

2. Редактирование данных (массовые изменения)

Акторы:

- Администратор
- Преподаватель (ограниченное редактирование)

Предусловия:

- Пользователь обладает правами на редактирование.
- Данные доступны для изменения.

Основной поток:

1. Пользователь открывает раздел редактирования.

2. Выбирает группу данных для редактирования.
3. Вносит изменения (например, обновляет оценки, посещаемость).
4. Подтверждает изменения.

Альтернативные потоки:

- Если пользователь пытается редактировать данные, загруженные кем-то другим, выводится предупреждение.
- Если пользователь не имеет прав на редактирование, действие блокируется.

3. Импорт данных

Акторы:

- Администратор
- Староста (для своей группы)
- Преподаватель (свои данные)

Предусловия:

- Данные подготовлены для загрузки (формат CSV или иные поддерживаемые структуры).

Основной поток:

1. Пользователь заходит в раздел загрузки данных.
2. Выбирает файл или вводит данные вручную.
3. Подтверждает загрузку.
4. Система проверяет корректность данных.
5. Данные сохраняются в системе.

Альтернативные потоки:

- Если файл содержит ошибки, пользователь получает уведомление о некорректных данных.
- Если у пользователя нет прав на загрузку, операция блокируется.

4. Удаление данных

Акторы:

- Администратор

Предусловия:

- Удаление разрешено только для данных, загруженных самим пользователем в течение ограниченного времени.

Основной поток:

1. Пользователь открывает список загруженных данных.
2. Выбирает данные для удаления.
3. Подтверждает удаление.

Альтернативные потоки:

- Если данные старше разрешённого периода, удаление невозможно.
- Если у пользователя нет прав на удаление, выводится сообщение об ошибке.

5. Сообщение об ошибке

Акторы:

- Студент
- Староста
- Преподаватель

Предусловия:

- Ошибка выявлена в загруженных данных.

Основной поток:

1. Пользователь открывает форму сообщения об ошибке.
2. Выбирает тип ошибки (например, неверные данные, отсутствие данных).

3. Вводит комментарий.
4. Отправляет запрос.
5. Система уведомляет ответственного (администратора или старосту).

Альтернативные потоки:

- Если ошибка не связана с системой, пользователю предлагается обратиться к преподавателю напрямую.

Типы ошибок:

1. Общая ошибка сайта – оформляется в виде формы обратной связи.
2. Ошибка в одной записи – должна либо появляться при наведении рядом с данными, либо быть всегда видимой.

6. Настройка данных (категории, вычисляемые поля)

Акторы:

- Администратор

Предусловия:

- Пользователь имеет права на изменение структуры данных.

Основной поток:

1. Пользователь заходит в настройки данных.
2. Добавляет новые категории или вычисляемые поля.
3. Определяет параметры обработки (например, автоматический расчёт среднего балла).
4. Сохраняет изменения.

Альтернативные потоки:

- Если изменение может повлиять на существующие данные, система запрашивает подтверждение.
- Если формат некорректен, отображается сообщение об ошибке.

7. Экспорт данных

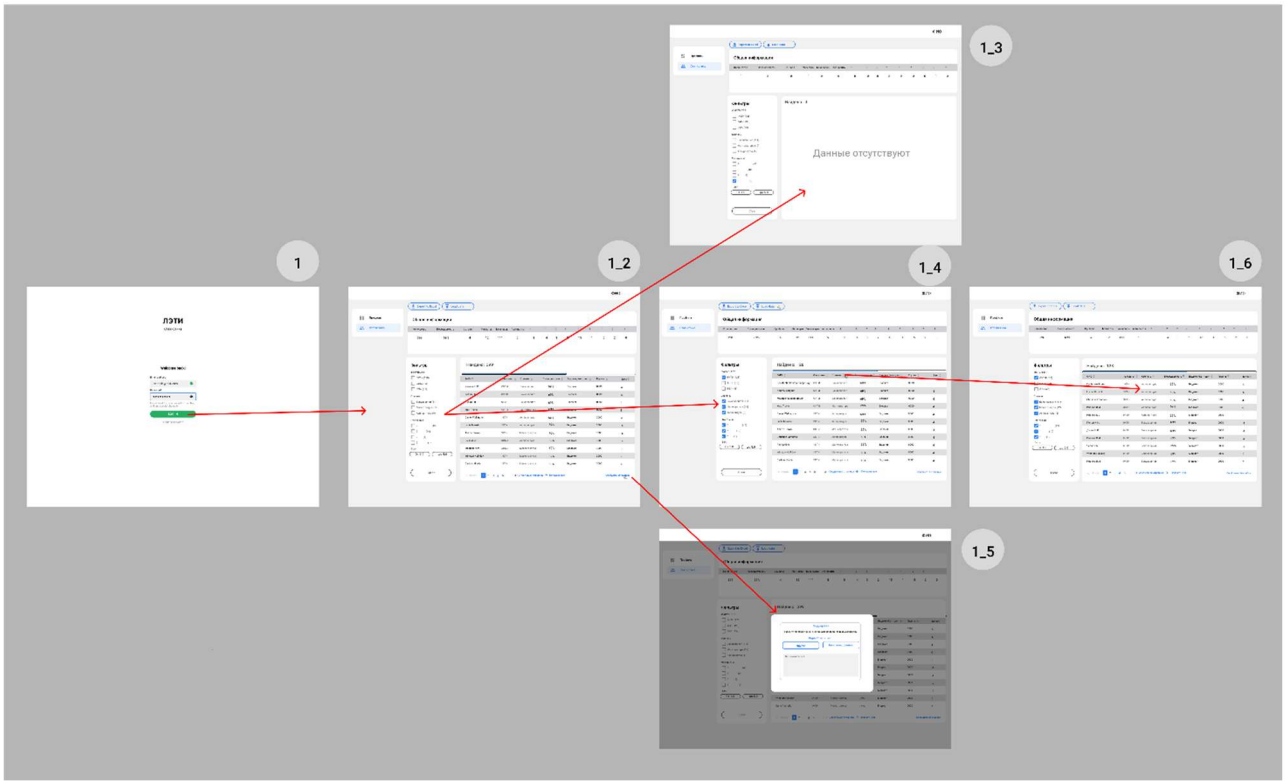
Акторы:

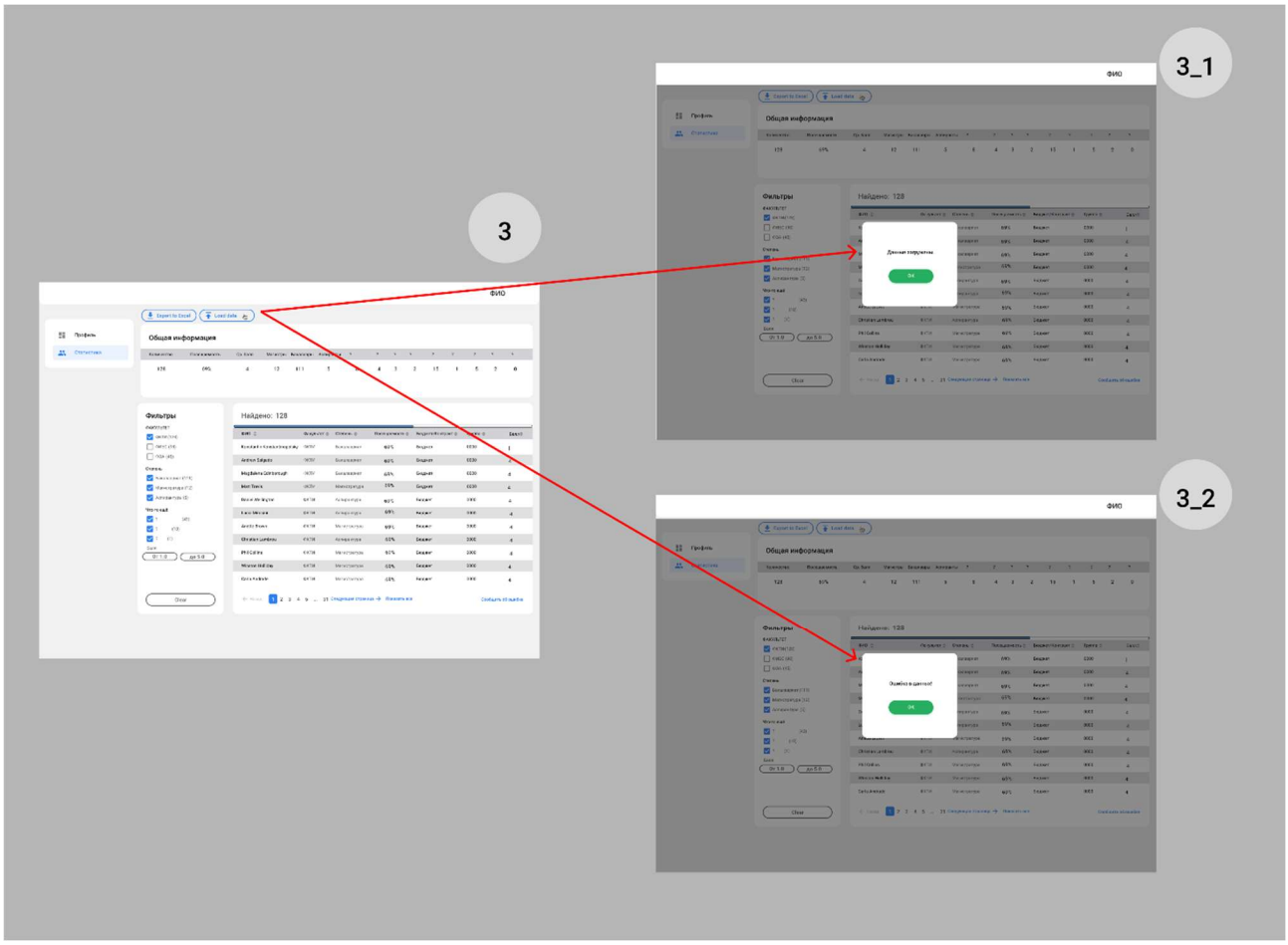
- Администратор

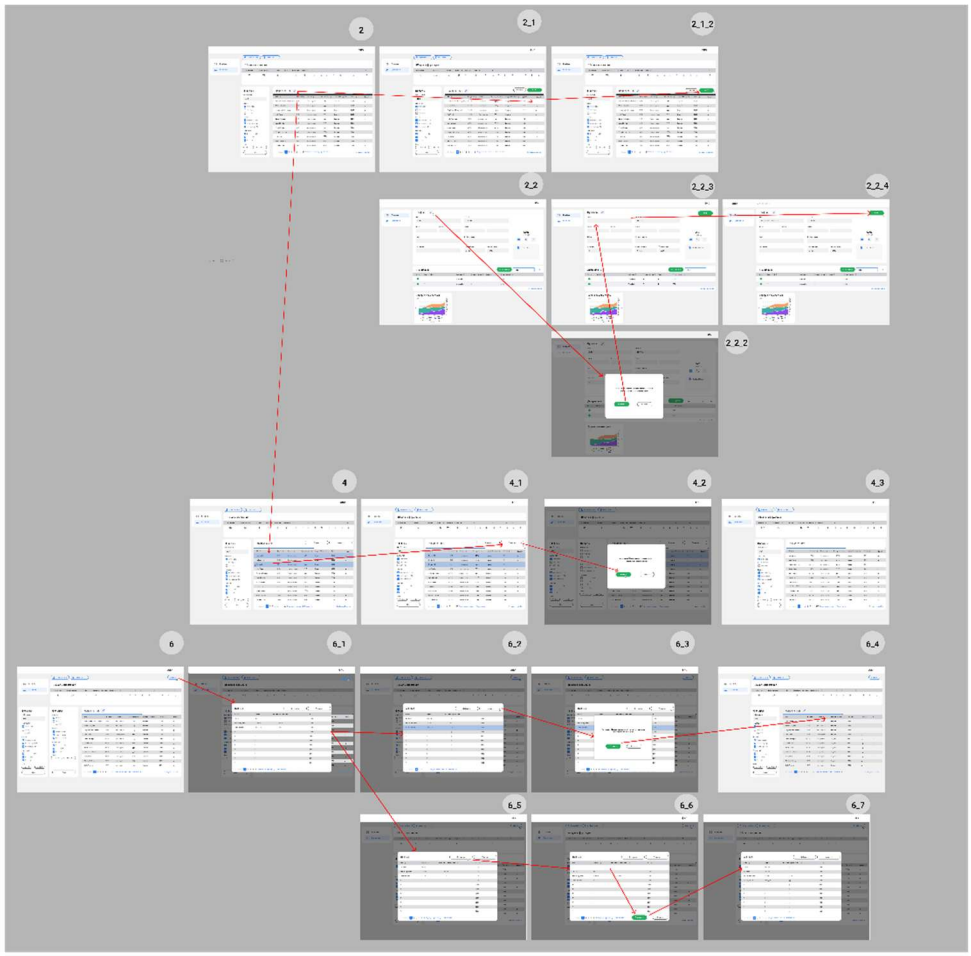
Предусловия:

- Пользователь имеет права на выгрузку данных.
- Основной поток:
- Пользователь заходит в список загруженных данных.
- Выставляет необходимые фильтры.
- Начинает экспорт.
- Данные выгружаются в формате csv/json.

МАКЕТ UI


1
1_2
1_3
1_4
1_5
1_6
Данные отсутствуют


3
3_1
3_2
ФИО
Общая информация
Фильтры
Найдено: 128


2
2_1
2_1_2
2_2
2_2_3
2_2_4
2_2_2
4
4_1
4_2
4_3
6
6_1
6_2
6_3
6_4
6_5
6_6
6_7

МОДЕЛЬ ДАННЫХ

Нереляционная модель

```
  "User": {
    "_id": "ObjectId",
    "email": "string",
    "login": "string",
    "password_hash": "string",
    "first_name": "string",
    "middle_name": "string",
    "last_name": "string",
    "active": "bool",
    "role_id": "string"
  },
  "Student": {
    "_id": "ObjectId",
    "user_id": "ObjectId",
    "birth_date": "date",
    "admission_year": "int",
    "student_type": {
      "type": "string",
      "enum": ["bachelor", "master", "aspirant", "specialist"]
    },
    "course": "int",
    "program_name": {
      "type": "string",
      "enum": ["Theoretical math", "Applied physics", "Computer
science"]
    },
    "faculty": {
      "type": "string",
      "enum": ["Math", "Physics", "IT"]
    },
    "group_name": {
      "type": "string",
```

```
    "enum": ["2323", "1421", "3501"]
  },
  "funding_type": {
    "type": "string",
    "enum": ["budget", "contract"]
  },
  "statistic": {
    "average_score": "decimal",
    "attendance_percent": "decimal",
    "calculation_date": "date",
    "count_activities": "int",
    "exlusion_probability": "decimal",
    "subjects": [
      {
        "subject_id": "ObjectId",
        "total_lessons": "int",
        "attendance_lessons": "int",
        "year": "int",
        "season": {
          "type": "string",
          "enum": ["autumn", "spring"]
        },
        "prediction_score": "decimal",
        "score": "decimal",
        "grade_value": {
          "type": "string",
          "enum": ["pass", "fail", "5", "4", "3", null]
        }
      }
    ]
  }
},
"Log": {
  "_id": "ObjectId",
  "user_id": "ObjectId",
```

```
    "action_type": "string",
    "action_date": "date",
    "ip_address": "string",
    "affected_entity": "string",
    "entity_id": "ObjectId",
    "description": "string",
    "role": "string"
  },
  "Teacher": {
    "_id": "ObjectId",
    "user_id": "ObjectId",
    "assigned_groups": ["string"],
    "assigned_subjects": ["ObjectId"]
  },
  "SubjectMeta": {
    "_id": "ObjectId",
    "subject_name": "string",
    "description": "string",
    "grade_type": {
      "type": "string",
      "enum": ["pass/fail", "exam"]
    },
    "is_activity": "bool"
  }
}
```

Описание назначений коллекций, типов данных и сущностей

Коллекция	**Назначение**
User	Учетная запись: email, логин, пароль, роль
Student	Студенческий профиль + вложенная статистика

Коллекция	Назначение
Teacher	Преподаватель: доступные группы и предметы
SubjectMeta	Метаинформация о предмете: название, тип, описание
Log	История действий пользователя

User

- _id (ObjectId, уникальный id) — **12 байт**
- email (string) — **~30 байт**
- login (string) — **~15 байт**
- password_hash (string) — **~60 байт**
- first_name (string) — **~15 байт**
- middle_name (string) — **~15 байт**
- last_name (string) — **~15 байт**
- active (bool) — **1 байт**
- role_id (string) — **~10 байт**

Итого ~150 байт на пользователя.

Student

- _id (ObjectId) — **12 байт**
- user_id (ObjectId) — **12 байт**

- birth_date (date) — **8 байт**
- admission_year (int) — **4 байта**
- student_type (string, enum) — **~10 байт**
- course (int) — **4 байта**
- program_name (string, enum) — **~20 байт**
- faculty (string, enum) — **~10 байт**
- group_name (string, enum) — **~6 байт**
- funding_type (string, enum) — **~8 байт**
- statistic — **60 байт**
    - average_score (decimal) — **16 байт**
    - attendance_percent (decimal) — **16 байт**
    - calculation_date (date) — **8 байт**
    - count_activities (int) — **4 байта**
    - exlusion_probability (decimal) — **16 байт**
- subjects[] (array of objects, ~50 предметов) — **~3400 байт**
    - Один элемент массива — **~68 байт**
        - subject_id (ObjectId) — **12 байт**
        - total_lessons (int) — **4 байта**
        - attendance_lessons (int) — **4 байта**
        - year (int) — **4 байта**
        - season (string, enum) — **~6 байт**
        - prediction_score (decimal) — **16 байт**

    - score (decimal) — **16 байт**
    - grade_value (string, enum) — **~6 байт**

Итого ~5 038 байт на студента.

Log

- _id (ObjectId) — **12 байт**
- user_id (ObjectId) — **12 байт**
- action_type (string) — **~10 байт**
- action_date (date) — **8 байт**
- ip_address (string) — **~15 байт**
- affected_entity (string) — **~15 байт**
- entity_id (ObjectId) — **12 байт**
- description (string) — **~50 байт**
- role (string) — **~10 байт**

Итого ~150 байт на лог.

Teacher

- _id (ObjectId) — **12 байт**
- user_id (ObjectId) — **12 байт**
- assigned_groups (array of strings, ~10 групп) — **~60 байт**
- assigned_subjects (array of ObjectId, ~2 предмета) — **24 байта**

Итого ~110 байт на преподавателя.

SubjectMeta

- _id (ObjectId) — **12 байт**
- subject_name (string) — **~25 байт**
- description (string) — **~100 байт**
- grade_type (string, enum) — **~10 байт**
- is_activity (bool) — **1 байт**

Итого ~160 байт на предмет.

Оценка объема информации, хранимой в модели

Общие параметры

Мы оценим размер каждой коллекции и выразим итог через N, учитывая:

- **N** — количество студентов.
- **U = N** — количество пользователей (один к одному).
- **T = N / 20** — количество преподавателей (в среднем 1 на 20 студентов).
- **L = N × 5** — количество логов (допустим, в среднем 5 записей на студента).
- **S = 150** — количество уникальных предметов (SubjectMeta), фиксировано.

Размер объектов

Объект	**Размер одного экземпляра**	**Кол-во**	**Формула размера**
User	~150 байт	= N	150 × N

Объект	Размер одного экземпляра	Кол-во	Формула размера
Student	~5038 байт	= N	5038 × N
Teacher	~110 байт	= N / 20	110 × N / 20 = 5.5 × N
Log	~150 байт	= 5 × N	150 × 5 × N = 750 × N
SubjectMeta	~160 байт	= 150	24 000 байт (константа)

Общая формула размера в байтах

Общий_объем(N) =

150 × N // User

+ 5 038 × N // Student

+ 5.5 × N // Teacher

+ 750 × N // Log

+ 24 000 // SubjectMeta (константа)

Или упрощённо:

Общий_объем(N) ≈ 5 943.5 × N + 24 000 байт

Примеры

Кол-во студентов (N)	Общий объем (байт)	Примерно в МБ
1 000	5 971 500	~5.7 МБ
10 000	59 566 00	~56.7 МБ
100 000	595 566 000	~566.8 МБ

Избыточность данных (отношение между фактическим объемом модели и «чистым» объемом данных)

Чистый объём данных

User

- login (string) — **~15 байт**
- password_hash (string) — **~60 байт**
- email (string) — **~30 байт**
- first_name (string) — **~15 байт**
- middle_name (string) — **~15 байт**
- last_name (string) — **~15 байт**

Итого: **~150 байт на пользователя**

Student

- user_id (ObjectId) — **12 байт**
- birth_date (date) — **8 байт**

- admission_year (int) — **4 байта**
- student_type (string, enum) — **~10 байт**
- course (int) — **4 байта**
- program_name (string, enum) — **~20 байт**
- faculty (string, enum) — **~10 байт**
- group_name (string, enum) — **~6 байт**
- funding_type (string, enum) — **~8 байт**
- statistic — **60 байт**
    - average_score (decimal) — **16 байт**
    - attendance_percent (decimal) — **16 байт**
    - calculation_date (date) — **8 байт**
    - count_activities (int) — **4 байта**
    - exlusion_probability (decimal) — **16 байт**
- subjects[] (array of objects, ~50 предметов) — **~2 000 байт**
    - Один элемент массива — **~40 байт**
        - subject_id (ObjectId) — **12 байт**
        - attendance_lessons (int) — **4 байта**
        - year (int) — **4 байта**
        - season (string, enum) — **~6 байт**
        - score (decimal) — **16 байт**
        - grade_value (string, enum) — **~6 байт**

Итого: ~2 152 байта на одного студента

Teacher

- user_id (ObjectId) — **12 байт**
- assigned_groups (array of strings, ~10 групп) — **~60 байт**
- assigned_subjects (array of ObjectId, ~2 предмета) — **24 байта**

Итого: **~96 байт на одного преподавателя**

→ В среднем 1 преподаватель на 20 студентов: 96 × N / 20 = 4.8 × N

Log

- user_id (ObjectId) — **12 байт**
- action_type (string) — **~10 байт**
- action_date (date) — **8 байт**
- affected_entity (string) — **~10 байт**
- entity_id (ObjectId) — **12 байт**

Итого: **~52 байта × 5 записей на студента = 260 × N**

SubjectMeta

- subject_name (string) — **~25 байт**
- description (string) — **~100 байт**
- grade_type (string) — **~10 байт**
- is_activity (bool) — **1 байт**

Итого: **~160 байт × 150 = 24 000 байт (константа)**

Размер объектов

Объект	Размер одного экземпляра	Кол-во	Формула размера
User	~150 байт	= N	150 × N
Student	~2 152 байт	= N	2 152 × N
Teacher	~96 байт	= N / 20	4.8 × N
Log	~52 байта × 5 = 260 байт	= 5 × N	260 × N
SubjectMeta	~160 байт	= 150	24 000 байт (константа)

Общая формула "чистого" объёма в байтах

Чистый_объем(N) =

150 × N // User

+ 2 152 × N // Student

+ 4.8 × N // Teacher

+ 260 × N // Log

+ 24 000 // SubjectMeta

Или:

Чистый_объем(N) = 2 566.8 × N + 24 000 байт

Избыточность модели

Избыточность = Общий_объем / Чистый_объем

Избыточность = (5 943.5 × N + 24 000) / (2 566.8 × N + 24 000)

Примеры избыточности для разных значений N

Кол-во студентов (N)	Избыточность
1 000	2.30
10 000	2.31
100 000	2.32
N→∞N→∞	2.32

Направление роста модели при увеличении количества объектов каждой сущности

Так как в модели все данные выражаются через N, то при увеличении количества объектов каждой сущности, модель будет расти линейно.

При этом, стоит учитывать, что несмотря на то, что в модели есть массивы объектов, они не влияют на линейный рост, так как количество элементов в них фиксировано и ограниченно сверху (например, 50 предметов в учебном плане).

Таким образом, при увеличении количества объектов в модели, она будет расти линейно, что позволяет легко масштабировать систему.

Примеры данных для нереляционной базы данных

User

```
{
  "_id": { "$oid": "661adfa5e13f1a1234567890" },
  "email": "ivan.ivanov@example.com",
```

```
  "login": "ivanivan",
  "password_hash": "$2b$10$....",
  "first_name": "Иван",
  "middle_name": "Иванович",
  "last_name": "Иванов",
  "active": true,
  "role_id": "student"
}
```

Student

```
{
  "_id": { "$oid": "661ae05de13f1a1234567891" },
  "user_id": { "$oid": "661adfa5e13f1a1234567890" },
  "birth_date": { "$date": "2003-06-21T00:00:00Z" },
  "admission_year": 2021,
  "student_type": "bachelor",
  "course": 3,
  "program_name": "Прикладная математика и информатика",
  "faculty": "IT",
  "group_name": "2323",
  "funding_type": "budget",
  "statistic": {
    "average_score": 4.2,
    "attendance_percent": 87.5,
    "calculation_date": { "$date": "2025-04-10T00:00:00Z" },
    "count_activities": 12,
    "exlusion_probability": 0.08,
    "subjects": [
      {
        "subject_id": { "$oid": "661ae1d9e13f1a12345678a0" },
        "attendance_lessons": 28,
        "year": 2024,
        "season": "autumn",
        "score": 5,
        "grade_value": "5"
```

```
      },
      {
        "subject_id": { "$oid": "661ae1d9e13f1a12345678a1" },
        "attendance_lessons": 22,
        "year": 2025,
        "season": "spring",
        "score": null,
        "grade_value": null
      }
    ]
  }
}
```

Teacher

```
{
  "_id": { "$oid": "661ae1fbe13f1a1234567892" },
  "user_id": { "$oid": "661ae1fbe13f1a1234567893" },
  "assigned_groups": ["2323", "2330"],
  "assigned_subjects": [
    { "$oid": "661ae1d9e13f1a12345678a0" },
    { "$oid": "661ae1d9e13f1a12345678a1" }
  ]
}
```

Log

```
{
  "_id": { "$oid": "661ae30fe13f1a1234567894" },
  "user_id": { "$oid": "661ae1fbe13f1a1234567893" },
  "action_type": "update",
  "action_date": { "$date": "2025-04-12T14:42:00Z" },
  "ip_address": "192.168.1.101",
  "affected_entity": "Student",
  "entity_id": { "$oid": "661ae05de13f1a1234567891" },
```

```
  "description": "Изменена посещаемость по предмету ID 661ae1d9e13f1a12345678a0",
  "role": "teacher"
}
```

SubjectMeta

```
{
  "_id": { "$oid": "661ae1d9e13f1a12345678a0" },
  "subject_name": "Введение в нереляционные базы данных",
  "description": "Курс по основам работы с NoSQL базами данных",
  "grade_type": "exam",
  "is_activity": false
}
```

Примеры запросов для нереляционной базы данных

1. Получить данные по одному студенту

```
async def get_student_by_user_id(db, user_id: str):
    student = await db.Student.find_one({ "user_id": ObjectId(user_id) })
    return student
```

- Коллекции: Student
- Масштаб: O(1)

2. Получить полные данные по студенту вместе с User (логин, email, ФИО)

```
async def get_full_student_info(db, user_id: str):
    pipeline = [
        {
            "$match": { "user_id": ObjectId(user_id) }
        },
        {
            "$lookup": {
                "from": "User",
                "localField": "user_id",
                "foreignField": "_id",
                "as": "user"
            }
        },
        {
            "$unwind": "$user"
        },
        {
            "$project": {
                "_id": 1,
                "birth_date": 1,
                "admission_year": 1,
                "student_type": 1,
```

```
                "course": 1,
                "program_name": 1,
                "faculty": 1,
                "group_name": 1,
                "funding_type": 1,
                "statistic": 1,
                "user.email": 1,
                "user.first_name": 1,
                "user.middle_name": 1,
                "user.last_name": 1
            }
        }
    ]
    cursor = db.Student.aggregate(pipeline)
     return await cursor.to_list(length=1)
```

Коллекции: Student, User

Использует: $lookup, $project

Масштаб: O(1)

Получить всех всех студентов постранично, отсортированных по exlusion_probability по возростанию.

```
async def get_students_sorted(db, page: int = 0, page_size: int = 20):
    now = datetime.utcnow()
    current_year = now.year

    cursor = db.Student.find({
        "admission_year": { "$lte": current_year },
        "course": { "$lt": 5 },  # фильтр: ещё учится
        "statistic.exlusion_probability": { "$ne": None }
    }).sort(
        "statistic.exlusion_probability", 1
    ).skip(
        page * page_size
    ).limit(
        page_size
    )

    return [doc async for doc in cursor]
```

- Коллекции: Student
- Использует: find, sort, skip, limit
- Масштаб: O(N), требует индекс

4. Преподаватель заходит и видит список студентов из своих групп, с только теми предметами, которые он сам ведёт.

```
async def get_students_for_teacher(db, teacher_user_id: str):
    teacher = await db.Teacher.find_one({ "user_id": ObjectId(teacher_user_id) })
```

```
    if not teacher:
        return []

    groups = teacher["assigned_groups"]
    subject_ids = teacher["assigned_subjects"]

    pipeline = [
        {
            "$match": {
                "group_name": { "$in": groups }
            }
        },
        {
            "$project": {
                "user_id": 1,
                "group_name": 1,
                "faculty": 1,
                "program_name": 1,
                "statistic.average_score": 1,
                "statistic.attendance_percent": 1,
                "statistic.subjects": {
                    "$filter": {
                        "input": "$statistic.subjects",
                        "as": "subj",
                        "cond": { "$in": ["$$subj.subject_id", subject_ids] }
                    }
                }
            }
        }
    ]

    cursor = db.Student.aggregate(pipeline)
     return await cursor.to_list(length=None)
```

- Коллекции: Teacher, Student
- Использует: find, $filter, $project
- Масштаб: O(N/20), фильтрация по group_name, subject_id

v. Фильтрация студентов по группе/факультету/типу финансирования

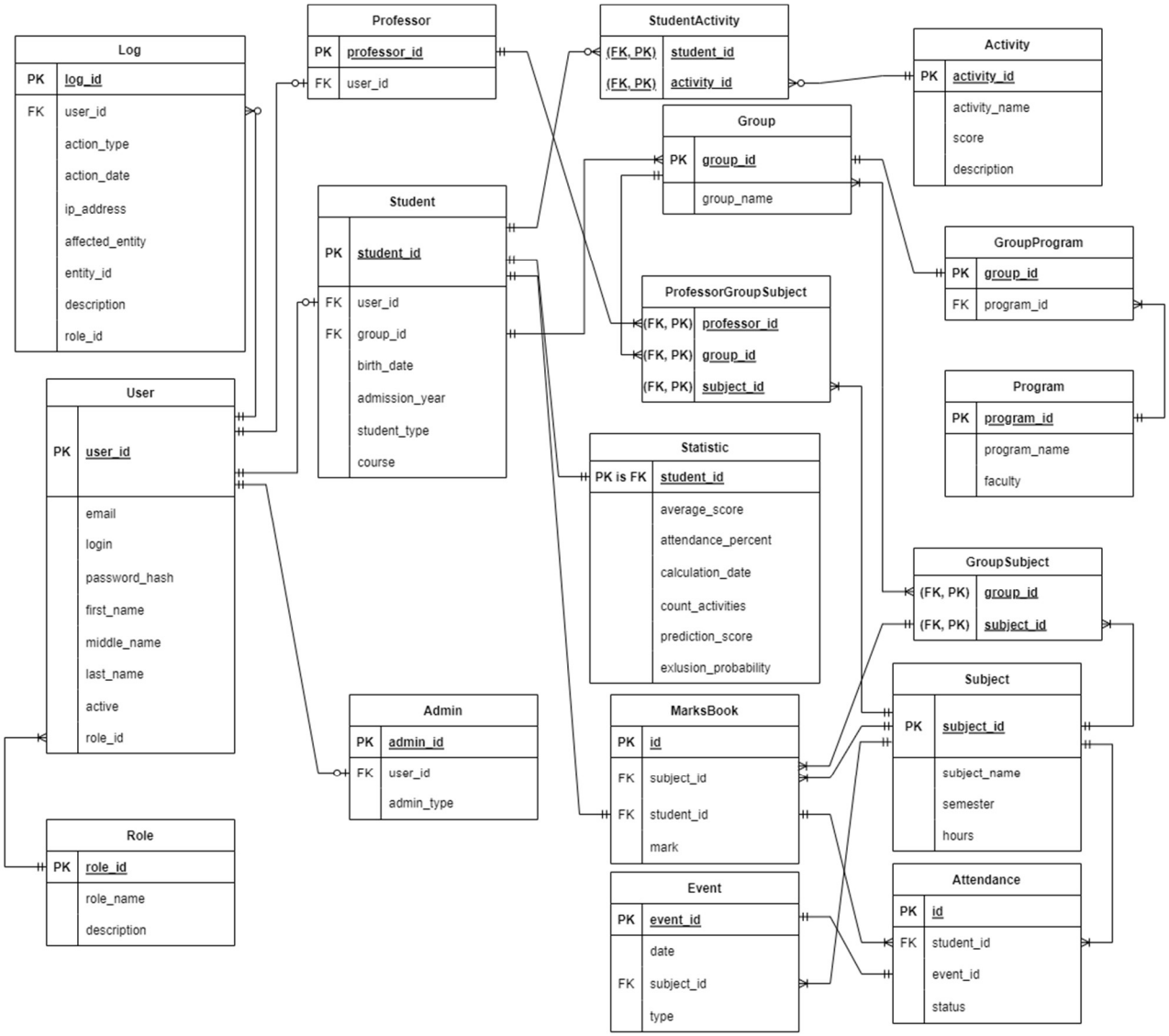
```
async def filter_students(db, group_name=None, faculty=None, funding_type=None):
    filters = {}
    if group_name:
        filters["group_name"] = group_name
    if faculty:
        filters["faculty"] = faculty
    if funding_type:
        filters["funding_type"] = funding_type

    cursor = db.Student.find(filters)
    return [doc async for doc in cursor]
```

Коллекции: Student

Использует: find, комбинированные фильтры

Масштаб: зависит от выборки

Реляционная модель



Описание назначений таблиц, типов данных и сущностей

Таблица User

Хранит информацию о пользователях.

Поле	Тип	Описание
user_id	int	Уникальный ID (auto-increment)
email	string	Название электронной почты
login	string	Логин
password_hash	string	Хеш пароля
first_name	string	Имя
middle_name	string	Отчество
last_name	string	Фамилия
active	bool	Активная иили нет учётная запись
role_id	int	Роль пользователя ("student" / "professor" / "admin")

Таблица Student

Хранит информацию о студентах.

Поле	Тип	Описание
student_id	int	Уникальный ID (auto-increment)

Поле	Тип	Описание
user_id	int	Уникальный ID пользователя
group_id	int	ID группы
birth_date	datetime	Дата рождения
admission_year	int	Год поступления
student_type	string	Тип студента - ('bachelor'/'master'/'aspirant'/'specialist')
course	int	Номер курса
program_name	string	Название программы обучения

Таблица Professor

Хранит информацию о преподавателях.

Поле	Тип	Описание
professor_id	int	ID преподавателя
user_id	int	ID пользователя

Поле	Тип	Описание
position	string	Должность

Таблица Admin

Хранит информацию об администраторах.

Поле	Тип	Описание
admin_id	int	ID администратора
user_id	int	ID пользователя
admin_type	string	Тип администратора

Таблица Log

Таблица логов.

Поле	Тип	Описание
log_id	int	ID записи (auto-increment)
user_id	int	Уникальный ID пользователя
action_type	string	Тип действия ("elit" / "add" / "delete")
action_date	datetime	Время действия

Поле	Тип	Описание
ip_address	string	IP-адрес
affected_entity	string	Сущность, которая редактировалась
entity_id	int	ID сущности
description	string	Описание
role_id	string	Роль пользователя ("student" / "professor" / "admin")

Таблица Group

Таблица с распределением студентов по группам.

Поле	Тип	Описание
group_id	int	Уникальный ID группы
group_name	string	Номер группы

Таблица Subject

Таблица с информацией о предмете.

Поле	Тип	Описание
subject_id	int	Уникальный ID (auto-increment)

Поле	Тип	Описание
subject_name	string	Название предмета
semester	int	Номер семестра
hours	int	Количество часов

Таблица Program

Таблица программ обучения.

Поле	Тип	Описание
program_id	int	ID учебной программы
program_name	string	Название программы
faculty	categorial	Название факультета

Таблица Role

Хранит список ролей пользователей (admin, student, professor и т.д.)

Поле	Тип	Описание
role_id	int	ID роли
role_name	string	Название роли
description	string	Описание

Таблица GroupProgram

Таблица соответствия между группами и программами.

Поле	Тип	Описание
group_id	int	ID учебной группы
program_id	int	ID учебной программы

Таблица GroupSubject

Учебный план, у каких групп какие предметы.

Поле	Тип	Описание
group_id	int	ID группы
subject_id	int	ID предмета

Таблица Activity

Таблица с инфориацией об активностях, проводиых в ВУЗе.

Поле	Тип	Описание
activity_id	int	ID активности
activity_name	string	Название активности
score	int	Количество баллов, в которое можно оценить активность
description	string	Описание активности

Таблица StudentActivity

Информация о том, какие студенты занимаются какими активностями.

Поле	Тип	Описание
student_id	int	ID студента
activity_id	int	ID активности

Таблица MarksBook

Зачетная книжка студента.

Поле	Тип	Описание
id	int	ID записи
subject_id	int	ID предмета
student_id	int	ID студента
mark	int	Оценка за предмет

Таблица Statistic

Хранит сжатую статистику о студенте.

Поле	Тип	Описание
student_id	int	Уникальный ID (auto-increment)
average_score	decimal	Средний балл
attendance_percent	decimal	Процент посещаемости

Поле	Тип	Описание
calculation_date	datetime	Дата вычисления статистики
count_activities	int	Количество активностей
prediction_score	decimal	Предсказанный средний балл
exlusion_probability	decimal	Вероятность отчисления

Таблица Event

Хранит информацию о событии.

Поле	Тип	Описание
event_id	int	ID записи
subject_id	int	ID предмета
date	datetime	Время проведения занятия
type	string	Тип пары ("lecture"/"practice")

Таблица Attendance

Хранит посещаемость лекций и практик студентами.

Поле	Тип	Описание
id	int	ID записи

Поле	Тип	Описание
student_id	int	ID студента
event_id	int	ID события
status	bool	1 - пара посещена, 0 - не посещена

Примеры данных для реляционной базы данных

Таблица User

user_id	email	login	password_hash	first_name	middle_name	last_name	active	role_id
0001	vasiliev101@mail.ru	hagr89	fdh647h9	Andrew	Andreevich	Vasiliev	0	student
0002	ivanovstep@mail.ru	user1	23746dy5	Stepan	Alekseevich	Ivanov	0	student

Таблица Student

student_id	user_id	group_id	birth_date	admission_year	student_type	course	program_name
0001	0001	2323	2004-04-02	2022	bachelor	3	Theoretical math
0002	0002	2323	2004-09-30	2022	bachelor	3	Theoretical math

Таблица Professor

professor_id	user_id	position
0001	0003	dean
0002	0004	professor

Таблица Admin

admin_id	user_id	admin_type
0001	0005	head_admin
0002	0006	admin

Таблица Log

log_id	user_id	action_type	action_date	ip_address	affected_entity	entity_id	description	role_id
0001	0005	add	2025-02-15 15:36:45	192.168.0.103	"Event"	0001	Добавление 2 строк	0001

Таблица Subject

subject_id	subject_name	semester	hours
0001	History	2	64
0002	Math statistic	4	64

subject_id	subject_name	semester	hours
0003	English	1	58
0004	Functional Analysis	6	64

Таблица MarksBook

id	subject_id	student_id	mark
0001	0008	0007	5
0002	0005	0007	3
0003	0019	0007	4

Таблица Attendance

id	student_id	event_id	status
0001	0001	0534	1
0002	0001	0635	0

Таблица Group

group_id	group_name
2381	2381

Таблица Activity

activity_id	activity_name	score	description
0001	math olympiad	1	Winter math olympiad

Таблица Event

event_id	subject_id	date	type
0001	0004	2025-04-21	lecture
0002	0004	2025-04-22	practice

Таблица Role

role_id	role_name	description
0001	head_admin	Главный администратор

Таблица ProfessorGroupSubject

professor_id	group_id	subject_id
0001	2387	0006

Таблица Program

program_id	program_name	faculty
0011	Program engineering	FCTI

Таблица GroupProgram

group_id	program_id
2304	0011
2305	0011
2381	0012

Таблица GroupSubject

group_id	subject_id
2300	0075
2304	0045
2305	0023

Таблица StudentActivity

student_id	activity_id
0002	0001

Примеры запросов для реляционной базы данных

1. Получить данные по одному студенту

```
SELECT
  s.*,
  stat.average_score, stat.attendance_percent, stat.calculation_date,
  stat.count_activities, stat.prediction_score, stat.exlusion_probability,
  sbj.subject_id, sbj.subject_name, sbj.semester, sbj.hours,
  mb.mark
FROM Student s
JOIN Statistic stat ON stat.student_id = s.student_id
LEFT JOIN MarksBook mb ON mb.student_id = s.student_id
LEFT JOIN Subject sbj ON sbj.subject_id = mb.subject_id
 WHERE s.user_id = $1;
```

- Таблицы: Student, Statistic, MarksBook, Subject
- Масштаб: **O(1)** по Student, **O(k)** по количеству предметов (где k — предметы конкретного студента, в среднем ≈ 50)

2. Получить полные данные по студенту вместе с User (логин, email, ФИО)

```
SELECT
  s.*,
  u.email, u.first_name, u.middle_name, u.last_name,
  stat.average_score, stat.attendance_percent, stat.calculation_date,
  stat.count_activities, stat.prediction_score, stat.exlusion_probability,
  sbj.subject_id, sbj.subject_name, sbj.semester, sbj.hours,
  mb.mark
FROM Student s
JOIN "User" u ON u.user_id = s.user_id
LEFT JOIN Statistic stat ON stat.student_id = s.student_id
LEFT JOIN MarksBook mb ON mb.student_id = s.student_id
LEFT JOIN Subject sbj ON sbj.subject_id = mb.subject_id
WHERE s.user_id = $1;
```

- Таблицы: Student, User, Statistic, MarksBook, Subject
- Масштаб: **O(1)** по Student, **O(k)** по количеству предметов (где k — предметы конкретного студента, в среднем ≈ 50)

3. Получить всех всех студентов постранично, отсортированных по exlusion_probability по возростанию.

```
SELECT
  s.student_id,
  s.user_id,
  s.group_id,
  s.course,
  s.program_name,
  stat.average_score,
  stat.exlusion_probability
FROM Student s
LEFT JOIN Statistic stat ON stat.student_id = s.student_id
ORDER BY stat.exlusion_probability ASC
LIMIT $1 OFFSET $2;
```

- Таблицы: Student, Statistic
- Масштаб: **O(N)** (где N — общее количество студентов, зависит от LIMIT + OFFSET)

4. Преподаватель заходит и видит список студентов из своих групп, с только теми предметами, которые он сам ведёт.

```
SELECT
  s.student_id,
  s.group_id,
  s.course,
  s.program_name,
  u.first_name,
  u.last_name,
  subj.subject_id,
  subj.subject_name,
  mb.mark
FROM ProfessorGroupSubject pgs
JOIN Student s ON s.group_id = pgs.group_id
JOIN "User" u ON u.user_id = s.user_id
JOIN MarksBook mb ON mb.student_id = s.student_id AND mb.subject_id = pgs.subject_id
JOIN Subject subj ON subj.subject_id = mb.subject_id
WHERE pgs.professor_id = $1
ORDER BY s.group_id, subj.subject_name;
```

- Таблицы: ProfessorGroupSubject, Student, User, MarksBook, Subject
- Масштаб: **O(k × m)**, где k — студентов в группах, m — предметов от преподавателя

5. Фильтрация студентов по группе/факультету/типу финансирования

```
SELECT
  s.student_id,
  s.group_id,
  g.group_name,
  s.program_name,
  p.faculty,
  s.course
FROM Student s
JOIN "Group" g ON g.group_id = s.group_id
JOIN GroupProgram gp ON gp.group_id = g.group_id
JOIN Program p ON p.program_id = gp.program_id
WHERE
  ($1 IS NULL OR g.group_name = $1)
```

```
  AND ($2 IS NULL OR p.faculty = $2)
  AND ($3 IS NULL OR s.student_type = $3);
```

- Таблицы: Student, Group, GroupProgram, Program
- Масштаб: **O(N)** (зависит от количества студентов)
- Поддержка фильтрации по всем 3 параметрам (каждый — необязателен)

Оценка объема информации, хранимой в модели SQL

Общие параметры для SQL

Для оценки объёма данных примем:

- **N** — количество студентов
- **U = N** — количество пользователей (1:1 со студентами)
- **T = N / 20** — количество преподавателей
- **L = 5 × N** — количество логов
- **S = 150** — количество предметов
- **M = 50** — среднее число записей в MarksBook на студента
- **A = 10** — количество активностей на студента
- **E = 200** — общее число событий (Event)
- **G = N / 25** — количество групп
- **P = 20** — количество программ
- **PGS = T × 2** — записи в ProfessorGroupSubject`

Размер объектов реляционной базы данных

Таблица	Размер одного экземпляра	Кол-во	Формула размера
User	~150 байт	N	150 × N
Student	~974 байта	N	974 × N
Professor	~14 байт	N / 20	0.7 × N
Log	~36 байт × 5	5 × N	180 × N
Subject	~33 байта	150	4 950 (константа)
Program	~40 байт	20	800 (константа)
MarksBook	~8 байт	50 × N	400 × N
Statistic	~60 байт	N	60 × N
Activity	~50 байт	10 × N	500 × N
StudentActivity	~8 байт	10 × N	80 × N
Group	~20 байт	N / 25	0.8 × N
GroupProgram	~8 байт	N / 25	0.8 × N
GroupSubject	~8 байт	150	1 200 (константа)
Event	~16 байт	200	3 200 (константа)
Attendance	~8 байт	10 × N	80 × N

Таблица	Размер одного экземпляра	Кол-во	Формула размера
ProfessorGroupSubject	~12 байт	2 × T	1.2 × N

Общая формула объема в байтах

Общий_объем(N) =

150 × N // User

+ 974 × N // Student

+ 0.7 × N // Professor

+ 180 × N // Log

+ 400 × N // MarksBook

+ 60 × N // Statistic

+ 500 × N // Activity

+ 80 × N // StudentActivity

+ 80 × N // Attendance

+ 1.2 × N // ProfessorGroupSubject

+ 0.8 × N // Group

+ 0.8 × N // GroupProgram

+ 4 950 // Subject

+ 800 // Program

+ 1 200 // GroupSubject

+ 3 200 // Event

= 2 487.5 × N + 10 150 байт

Примеры

Кол-во студентов (N)	**Общий объем (байт)**	**Примерно в МБ**
1 000	2 497 650	~2.4 МБ
10 000	24 885 150	~23.7 МБ
100 000	248 760 150	~237.2 МБ

Избыточность данных SQL

Чистый объём данных SQL

User

- login (string) — **~15 байт**
- password_hash (string) — **~60 байт**
- email (string) — **~30 байт**
- first_name (string) — **~15 байт**
- middle_name (string) — **~15 байт**
- last_name (string) — **~15 байт**

Итого: **~150 байт на пользователя**

Student

- user_id (int, FK) — **4 байта**
- birth_date (date) — **8 байт**

- admission_year (int) — **4 байта**
- student_type (string, enum) — **~10 байт**
- course (int) — **4 байта**
- program_name (string) — **~20 байт**
- group_id (int) — **4 байта**
- statistic — **60 байт**
    - average_score (decimal) — **16 байт**
    - attendance_percent (decimal) — **16 байт**
    - calculation_date (date) — **8 байт**
    - count_activities (int) — **4 байта**
    - exlusion_probability (decimal) — **16 байт**
- marksbook (50 предметов) — **50 × 4 (subject_id) + 4 (mark)** ≈ **~400 байт**

  Итого: **~514 байт + 60 + 400 = ~974 байта на студента**

Professor

- user_id (int, FK) — **4 байта**
- position (string) — **~10 байт**

Итого: **~14 байт на преподавателя**

→ В среднем 1 преподаватель на 20 студентов → 14 × N / 20 = 0.7 × N

Log

- user_id (int) — **4 байта**

- action_type (string) — **~10 байт**
- action_date (datetime) — **8 байт**
- affected_entity (string) — **~10 байт**
- entity_id (int) — **4 байта**

Итого: **~36 байт × 5 записей = 180 байт / студент**

Subject

- subject_name (string) — **~25 байт**
- semester (int) — **4 байта**
- hours (int) — **4 байта**

Итого: **~33 байта × 150 = 4 950 байт (константа)**

Program

- program_name (string) — **~25 байт**
- faculty (string) — **~15 байт**

Итого: **~40 байт × ~20 программ = ~800 байт** → можно считать **константой**

Размер объектов SQL

Объект	**Размер одного экземпляра**	**Кол-во**	**Формула размера**
User	~150 байт	= N	150 × N
Student	~974 байт	= N	974 × N

Объект	Размер одного экземпляра	Кол-во	Формула размера
Professor	~14 байт	= N / 20	0.7 × N
Log	~180 байт	= 5 × N	180 × N
Subject	~33 байта	= 150	4 950 (константа)
Program	~40 байт	= ~20	800 (константа)

Общая формула чистого объёма sql

Чистый_объем(N) =

150 × N // User

+ 974 × N // Student

+ 0.7 × N // Professor

+ 180 × N // Log

+ 4 950 // Subject

+ 800 // Program

Или:

Чистый_объем(N) = 1 304.7 × N + 5 750 байт

Избыточность реляционной модели

Избыточность = Общий_объем / Чистый_объем

Подставим формулы:

Общий_объем(N) = 2 487.5 × N + 10 150

Чистый_объем(N) = 1 304.7 × N + 5 750

Избыточность = (2 487.5 × N + 10 150) / (1 304.7 × N + 5 750)

Примеры избыточности для разных значений N реляционной БД

Кол-во студентов (N)	Избыточность
1 000	**1.91**
10 000	**1.91**
100 000	**1.91**
NtoinftyNtoinfty	**1.91**

Направление роста модели при увеличении количества объектов каждой сущности

Реляционная модель масштабируется линейно по количеству студентов N, поскольку практически все основные сущности (такие как User, Student, Statistic, MarksBook, Log, StudentActivity, Attendance) прямо или косвенно связаны со студентом.

Сравнение моделей

Удельный объём информации

Параметр	**NoSQL**	**SQL**
Формула объёма	2 566.8 × N + 24 000 байт	2 487.5 × N + 10 150 байт
Объём при N = 10 000	25 692 000 байт (~24.5 МБ)	24 885 150 байт (~23.7 МБ)
Избыточность	~2.32	~1.91

Запросы по отдельным юзкейсам

Юзкейс	NoSQL	SQL
1. Получить одного студента	1 find_one(Student) Коллекции: 1 **Сложность:** O(1)	1 SELECT с JOIN (Statistic, MarksBook, Subject) Таблицы: 4 **Сложность:** O(k)
2. Студент + User	1 aggregate с $lookup (User) Коллекции: 2 **Сложность:** O(1)	1 SELECT с JOIN (User, Statistic, MarksBook, Subject) Таблицы: 5 **Сложность:** O(k)
3. Студенты постранично по exlusion_probability	1 find().sort().skip().limit() Коллекции: 1 **Сложность:** O(log N + P) *(при индексе)*	1 SELECT + JOIN(Statistic) + ORDER BY + LIMIT/OFFSET Таблицы: 2 **Сложность:** O(log N + P)
4. Преподаватель → студенты по группам и предметам	1 find_one(Teacher) + 1 aggregate(Student) с $filter Коллекции: 2 **Сложность:** O(N/20 × m)	1 SELECT с JOIN (ProfessorGroupSubject, Student, User, MarksBook, Subject) Таблицы: 5 **Сложность:** O(k × m)
5. Фильтрация по группе, факультету,	1 find(Student) с комбинированными фильтрами Коллекции: 1 **Сложность:** O(fN	1 SELECT с JOIN (Group, GroupProgram, Program) Таблицы: 4 **Сложность:** O(fN)

Юзкейс	NoSQL	SQL
финансированию	) *(зависит от фильтров)*	

Примечания:

- k — количество предметов у одного студента (≈ 50)
- m — количество предметов, читаемых преподавателем
- P — размер страницы (limit)
- fN — фильтрованное подмножество студентов

NoSQL использует меньшее количество коллекций, особенно если данные вложены. Подходит для прямого доступа, но сложные выборки требуют aggregate.

SQL объединяет все данные в одном SELECT с JOIN, что позволяет выразительно формулировать сложные запросы, особенно при работе с аналитикой.

Вывод

Обе модели — SQL и NoSQL — демонстрируют линейный рост объёма при увеличении числа студентов и справляются с базовыми операциями за 1–2 запроса, но SQL обеспечивает большую выразительность в аналитических сценариях благодаря JOIN-запросам, тогда как NoSQL проще и эффективнее при работе с вложенными структурами. SQL-модель оказывается чуть менее избыточной (~1.91 против ~2.32), а разница в реальном объёме хранения при масштабе до 100 000 студентов не превышает 5–10%, что делает выбор модели скорее вопросом архитектурных предпочтений и типов запросов.

ВЫВОДЫ

Большая часть поставленных задач реализована: спроектирована NoSQL-модель хранения данных (MongoDB), разработаны основные интерфейсы для работы с пользователями, студентами, преподавателями и предметами. В систему заложены возможности для дальнейшего расширения и аналитической обработки данных, включая прогнозирование успеваемости. Сервис частично реализует заявленную цель, демонстрируя работоспособную архитектуру и потенциал для дальнейшего развития.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1.**Алгоритмы. Построение и анализ** [Книга] / авт. Кормен Томас Штайн Клиффорд, Ривест Рональд , Лейзерсон Чарльз. - 1990.

2.**Разрешение коллизий** [В Интернете] // Викиконспекты ИТМО. - 20 декабря 2023 г.. - https://neerc.ifmo.ru/wiki/index.php?title=Разрешение_коллизий.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: modules/DoubleHashTable.py

```
class DoubleHashTable:
    def __init__(self, capacity: int):
        self.__capacity = 2 ** (capacity - 1).bit_length()
        self.__data = [(None, None)] * self.__capacity

    def values(self) -> list:
        return [pair[1] for pair in self.__data if pair[0] is not None]

    def keys(self) -> list:
        return [pair[0] for pair in self.__data if pair[0] is not None]

    def items(self):
        return [pair for pair in self.__data if pair[0] is not None]

    def hash_func_1(self, key) -> int:
        return hash(key) % self.__capacity

    def hash_func_2(self, key) -> int:
        return (2 * hash(key) + 1) % self.__capacity

    def __setitem__(self, key, value):
        index = self.hash_func_1(key)
        offset = self.hash_func_2(key)
        for _ in range(self.__capacity):
            if self.__data[index][0] is None or self.__data[index][0] == key:
                self.__data[index] = (key, value)
                return
            index = (index + offset) % self.__capacity
        raise RuntimeError
```

```
    def __getitem__(self, key):
        index = self.hash_func_1(key)
        offset = self.hash_func_2(key)
        for i in range(self.__capacity):
            if self.__data[index][0] is None:
                raise KeyError
            if self.__data[index][0] == key:
                return self.__data[index][1]
            index = (index + offset) % self.__capacity
        raise KeyError
```

Название файла: modules/time_perf.ipynb

```
import time

import numpy as np

from DoubleHashTable import DoubleHashTable


def best_case(n):
    hash_table = DoubleHashTable(n)
    pre_last = n - 1
    for i in range(pre_last):
        hash_table[i] = "value"
    # Добавление нового ключа и измерение времени
    start_time = time.perf_counter()
    hash_table[pre_last] = pre_last
    end_time = time.perf_counter()
    return end_time - start_time


def average_case(n: int):
    hash_table = DoubleHashTable(n)
```

```
        pre_last = n * n
        arr = np.random.randint(0, pre_last, size=n, dtype=np.int64)
        for i in arr:
            hash_table[2 * i] = i

        start_time = time.perf_counter()
        hash_table[pre_last] = pre_last
        end_time = time.perf_counter()
        return end_time - start_time


    def worst_case(n: int):
        hash_table = DoubleHashTable(n)
        m = 2 ** (n - 1).bit_length()
        pre_last = m * m - m
        for i in range(0, pre_last, m):
            #            print(i,            hash_table.hash_func_1(i),
hash_table.hash_func_2(i))
            hash_table[i] = i

        start_time = time.perf_counter()
        hash_table[pre_last] = pre_last
        end_time = time.perf_counter()
        return end_time - start_time


    def measure_time(data_volumes, case):
        times = []
        for volume in data_volumes:
            # Заполнение хеш-таблицы данными
            times.append(case(volume))
        return times


    #%%
```

```
# Объемы данных
data_volumes = [10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000]
#%%

# Время, затраченное на добавление новых ключей

best_times = measure_time(data_volumes, best_case)
#%%
average_times = measure_time(data_volumes, average_case)
#%%
worst_volumes = [10, 50, 100, 500, 1000, 5000, 10000]
# worst_volumes = data_volumes
worst_times = measure_time(worst_volumes, worst_case)
#%%
import matplotlib.pyplot as plt

plt.plot(data_volumes, best_times, label="Лучшее время")
plt.plot(data_volumes, average_times, label="Среднее время")
plt.plot(worst_volumes, worst_times, label="Худшее время")
plt.xscale("log")
plt.yscale("log")
plt.title('Время добавления ключей в хеш-таблицу')
plt.xlabel('Объем данных')
plt.ylabel('Время, с')
plt.legend()
plt.grid(True)
plt.show()
```

Название файла: main.py

```
from modules.DoubleHashTable import DoubleHashTable


def main():
```

```
    a = DoubleHashTable(128)
    a[8] = 4
    a[7] = 532
    print(a.keys())
    print(a.items())


if __name__ == '__main__':
    main()
```

Название файла: tests.py

```
import pytest

from modules.DoubleHashTable import DoubleHashTable


def test_should_create_hash_table():
    assert DoubleHashTable(capacity=100) is not None


@pytest.fixture
def hash_table():
    sample_table = DoubleHashTable(capacity=128)

    sample_table[10] = 1000
    sample_table[1] = 32
    sample_table[34] = 8732
    return sample_table


def test_should_insert_key_value_pair(hash_table):
    assert (10, 1000) in hash_table.items()
    assert (1, 32) in hash_table.items()
    assert (34, 8732) in hash_table.items()
```

```
def test_should_raise_runtime_error():
    hash_table = DoubleHashTable(capacity=2)
    hash_table[10] = 1000
    hash_table[1] = 32
    with pytest.raises(RuntimeError):
        hash_table[34] = 8732


def test_should_override_value(hash_table):
    assert (10, 1000) in hash_table.items()

    hash_table[10] = 32

    assert (10, 1000) not in hash_table.items()
    assert (10, 32) in hash_table.items()


def test_should_get_value_by_key(hash_table):
    assert hash_table[10] == 1000
    assert hash_table[1] == 32
    assert hash_table[34] == 8732


def test_should_raise_key_error_because_not_found(hash_table):
    with pytest.raises(KeyError):
        hash_table[0]


def test_should_raise_key_error_because_not_found_2():
    hash_table = DoubleHashTable(capacity=2)
    hash_table[2] = 34
    hash_table[4] = 34
    with pytest.raises(KeyError):
```

```
hash_table[0]
```