Глава 1 Архитектура и Обоснование Схемы

Введение

Настоящий отчет представляет собой документацию по проекту миграции и анализа данных образовательного учреждения (OU) с использованием нереляционной базы данных MongoDB и контейнеризации Docker.

Целью проекта являлось создание оптимизированной NoSQL-архитектуры на основе денормализации исходных реляционных таблиц, разработка надежного процесса ETL (Extract, Transform, Load) и демонстрация возможностей системы через выполнение десяти сложных аналитических запросов (MQL).

Полный исходный код проекта, включая скрипты инициализации и конфигурацию Docker Compose, доступен для проверки в публичном репозитории GitHub: https://github.com/luky048-ship-it/mongodb-architecture-university.git

1.1 Выбор Модели: NoSQL (MongoDB)

Мы выбрали нереляционную базу данных MongoDB для хранения данных об образовательном процессе (OU).

Причина выбора	Обоснование
Неизменность данных	Данные курсов и оценок являются историческими и статичными, они редко меняются после миграции. Это идеально подходит для NoSQL, где структура оптимизирована для быстрого чтения.
Аналитическая нагрузка	Основная задача системы — выполнять сложные аналитические агрегации (расчет среднего балла, распределение по регионам). MongoDB с оператором $aggregate лучше справляется с этой задачей, чем традиционные SQL-соединения.
Горизонтальная масштабируемость	В долгосрочной перспективе NoSQL легче масштабируется при увеличении числа студентов и курсов.

1.2 Инфраструктура и Развертывание (Containerization)

Для обеспечения воспроизводимости и изоляции среды проекта была выбрана технология контейнеризации с использованием Docker и Docker Compose. Это позволяет упаковать базу данных MongoDB вместе со всей логикой инициализации и ETL-скриптами в единый, переносимый сервис.

Преимущество	Обоснование
Воспроизводимость	Гарантируется, что среда разработки (версия MongoDB, конфигурация, переменные окружения) идентична среде развертывания.
Изоляция	"Сервис MongoDB работает изолированно, не влияя на основную операционную систему и обеспечивая предсказуемое поведение.

Преимущество	Обоснование
Автоматизация Запуска	"Файл docker-compose.yml содержит декларативное описание сервиса, включая персистентное хранилище (тома) и переменные окружения для аутентификации и конфигурации.

Конфигурация Сервиса MongoDB

Основная конфигурация сервиса определена в файле docker-compose.yml, который связывает образ MongoDB с локальными скриптами инициализации и гарантирует сохранность данных:

```
version: '3.8'
services:
  mongo:
    build:
      context: ./init-db
      dockerfile: Dockerfile
    image: university-mongo-custom:latest
    container_name: university-grades-mongo
    restart: unless-stopped
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db
      - ./init-db:/docker-entrypoint-initdb.d:ro
    environment:
      MONGO_INITDB_DATABASE: university
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
      OULAD_ZIP_URL:
"https://archive.ics.uci.edu/static/public/349/open+university+learning+an
alytics+dataset.zip"
    healthcheck:
      test: ["CMD", "mongosh", "--quiet", "--eval",
"db.adminCommand('ping').ok"]
      interval: 10s
      timeout: 10s
      retries: 10
      start_period: 30s
volumes:
  mongo-data:
    name: university-mongo-data
```

1.4 Автоматизация и Загрузка Данных (Extract & Load)

Оркестрация Загрузки (Extract & Load) Ключевым аспектом реализации является полная автоматизация развертывания, при которой все этапы ETL выполняются последовательно в контейнере MongoDB. В этом проекте реализованна автоматическая загрузка базы данных: open_university_learning_analytics_dataset.

Загрузка сырых данных (mongoimport) является длительной и асинхронной операцией. Если трансформация начнется до ее завершения, она обработает неполные данные. Поэтому мы используем строгую алфавитно-цифровую последовательность скриптов, где 03-douwnload-and-import.sh выполняет Извлечение (Extract) и гарантирует завершение загрузки перед выходом. Скрипт 05-full-setup.sh выступает в роли Оркестратора, управляя логикой Трансформации с уверенностью, что все исходные данные уже доступны.

Данные подгружаются автоматически в фоновом процессе, что может занять длительное время.

1.3. Схема Данных: Денормализация

Вместо сохранения данных в семи отдельных таблицах, как в исходной реляционной структуре, мы использовали модель денормализации (Embedding).

Это означает, что связанные данные (например, информация о курсе и оценке) вложены в основной документ, что исключает необходимость частых операций соединения (JOIN) при выполнении аналитических запросов.

Мы создали три основные коллекции:

1. students (Студенты)

Назначение: Хранит основную информацию о студентах, историю обучения и финальный результат.

- Денормализация: Объединяет данные из таблиц studentInfo, studentRegistration и studentVle.
- Ключевые поля: id_student, final_result, demographics (вложенный документ с образованием, регионом и возрастом).

```
STUDENTS [id_student, final_result]
├── id_student [Ключ: Уникальный]
├── final_result (Pass, Fail, Distinction, Withdrawn)
├── demographics (Вложенный документ - studentInfo)
│   ├── region
│   ├── highest_education
│   └── age_band
└── study_history (Вложенный документ - studentRegistration)
    └── studied_credits
```

2. grades (Оценки)

Назначение: Хранит все сданные оценки студентами. Это самая крупная коллекция ($\approx 170\ 000$ документов).

- Денормализация: Содержит вложенные документы с информацией о курсе (course_info) и самой оценочной работе (assessment_info).
- Ключевые поля: id_student, score, date_submitted.

```
GRADES [id_student, id_assessment, score]
├── id_student [Связь с STUDENTS, для $lookup]
├── score (Балл, -1 если нет)
├── date_submitted (Дата сдачи работы)
├── course_info (Вложенный документ - courses/studentRegistration)
│   ├── code_module
│   └── code_presentation
└── assessment_info (Вложенный документ - assessments)
    ├── type (TMA, CMA, Exam)
    ├── weight (Вес оценки)
    └── date_due (Крайний срок сдачи)
```

3. courses (Курсы)

Назначение: Хранит метаданные о курсах и связанных с ними оценочных работах.

- Денормализация: В документ курса (code_module, code_presentation) вложены все связанные оценочные работы (assessments) с их весами и сроками сдачи.

```
COURSES [code_module, code_presentation]
├── code_module
├── code_presentation
└── assessments [Массив вложенных документов - assessments]
    ├── assessment_info (Повторяется для каждой оценки курса)
    │   ├── id_assessment [Ключ: Уникальный в массиве]
    │   ├── assessment_type (TMA, CMA, Exam)
    │   ├── weight (Вес оценки в %)
    │   └── date (Дата сдачи)
```

Ключевые связи (Reference)

Эта схема демонстрирует:

- Денормализацию (вложенные документы demographics, assessment_info).
- Аналитическое ядро (grades).
- Единственную оставшуюся связь (id_student) для расширенного анализа.

Единственная Активная Связь (Reference)

В денормализованной схеме остается только одна ссылочная связь, которая не может быть устранена и требует использования $lookup:

```
СТУДЕНТЫ [id_student]
  └── --- [ССЫЛКА (Reference) - Используется $lookup] ---> GRADES
[id_student]
```

Глава 2 Процесс ETL (Миграция данных)

2.1. Инструментарий и Общий Поток

Миграция была выполнена с использованием оркестрированной цепочки скриптов на Bash и JavaScript, активируемой автоматически механизмом Docker Entrypoint.

Этап ETL	Реализация	Описание
Extract (Извлечение)	03-douwnload-and-import.sh	Данные из семи исходных CSV-файлов были извлечены и загружены в сырые коллекции (raw_*) MongoDB.
Orchestration	05-full-setup.sh	Главный контроллер, гарантирующий последовательный запуск трансформации и индексации только после завершения Extract.
Transform (Преобразование)	migrate.js и clean.js	Выполнялась основная работа по денормализации и очистке данных (дедупликация, обработка пропусков).
Load (Загрузка)	migrate.js	Преобразованные документы были загружены в три целевые коллекции: students, grades, и courses.

2.2. Очистка и Нормализация Данных

Ключевым этапом ETL была обработка пропущенных значений и приведение данных к нужному типу:

Проблема	Решение в ETL-скрипте	Обоснование
Пропущенные значения (?)	Все отсутствующие числовые значения в полях, таких как score (оценка) или date (дата сдачи), были заменены на числовой флаг -1.	Замена на -1 позволяет использовать числовые операторы MQL ($ne: -1, $gt, $avg) для фильтрации и аналитики без ошибок типов.
Денормализация	При создании документов в целевых коллекциях производилось вложение связанных данных.	Обеспечение максимальной скорости чтения для выполнения аналитических запросов без необходимости $lookup в большинстве случаев.
Удаление дубликатов	Скрипт clean.js использует конвейер агрегации для дедупликации коллекции students по полю id_student.	Гарантия целостности данных и уникальности записей, что является обязательным условием для создания уникального индекса и корректного использования в $lookup.

Для реализации этой части были применен скрипт clean.js который гарантировал отсутствие дубликатов, а так же дополнительно в скрипте 04-schemas.js была применена JSON Schema Validation.

Это обеспечило, что даже в случае внешнего воздействия, в целевые коллекции не будут записаны документы, нарушающие структуру (например, оценки score вне диапазона $[0, 100]$).

Фрагмент кода из clean.js (Дедупликация):

```
// Используем Aggregation Pipeline для нахождения и удаления дубликатов.
db.students.aggregate([
  {
    $group: {
      _id: "$id_student", // Группируем по полю, которое должно быть
уникальным
      duplicates: { $push: "$_id" }, // Сохраняем все _id, принадлежащие
этому id_student
      count: { $sum: 1 } // Считаем количество документов
    }
  },
  { $match: { count: { $gt: 1 } } } // Находим только те, где count > 1
(дубликаты)
]).forEach(function(doc) {
  // Удаляем все документы, кроме первого (оставляем документ с наименьшим
_id)
  doc.duplicates.shift();
  db.students.deleteMany({ _id: { $in: doc.duplicates } });
});
```

2.3. Структура ETL-скрипта

Скрипт migrate.js выполнил денормализацию, создавая следующие целевые структуры:

- Коллекция courses: Создание документов курсов и вложение в них всех связанных оценочных работ (assessments) для быстрого доступа к структуре курса.
- Коллекция grades: Итерация по данным об оценках, вложение в каждый документ соответствующей информации о курсе (code_module, code_presentation) и оцениваемой работе (weight, type).
- Коллекция students: Объединение информации о демографии, истории обучения и регистрации в один документ студента, формируя единую точку данных.

Фрагмент кода из migrate.js:

```
   // Используем Aggregation Framework для преобразования типов и
денормализации
    db.raw_studentInfo.aggregate([
        {
            $project: {
                _id: 0,
                // Преобразование id_student в int
                id_student: {
                  $convert: {
```

```
                        input: "$id_student",
                        to: "int",
                        onError: "$$REMOVE", // Удалить документ, если ID
невалидный
                        onNull: "$$REMOVE"
                    }
                },
                  code_module: "$code_module",
                  code_presentation: "$code_presentation",
                  final_result: "$final_result",
                  // Денормализация демографии в один вложенный объект
                  demographics: {
                      gender: "$gender",
                      region: "$region",
                      age_band: "$age_band",
                      highest_education: "$highest_education",
                      disability: "$disability"
                  },
                  study_history: {
                      num_of_prev_attempts: {
                        $convert: {
                            input: "$num_of_prev_attempts",
                            to: "int",
                            onError: 0,
                            onNull: 0
                        }
                    },
                      studied_credits: {
                        $convert: {
                            input: "$studied_credits",
                            to: "int",
                            onError: 0,   // Если ошибка (например, '?'),
ставим 0
                            onNull: 0     // Если поле отсутствует, ставим 0
                          }
                      }
                  }
              }
          },
    {
```

Глава 3 Индексация и Оптимизация

3.1. Необходимость Индексации

Сложные аналитические запросы, особенно те, которые используют агрегацию ($group), сортировку ($sort) или объединение ($lookup), работают крайне неэффективно без индексов. Без них MongoDB выполняет полное сканирование коллекции (Collection Scan), что приводит к зависанию на больших наборах данных.

3.2. Автоматизация Индексации

Для обеспечения автоматизации и воспроизводимости проекта индексы создаются с помощью файла index.js, который автоматически выполняется при старте контейнера MongoDB в Docker.

Фрагмент кода из index.js (Уникальный Индекс):

```
// ============================================================================
// 1. Индексы для коллекции 'students'
// ============================================================================

db.students.createIndex(
    { "id_student": 1 },
    { unique: true, name: "idx_students_id", background: true }
);
print("Индекс 'idx_students_id' (students, unique) создан.");

db.students.createIndex(
    { "final_result": 1, "demographics.highest_education": 1 },
    { name: "idx_result_education", background: true }
);
print("Индекс 'idx_result_education' (students) создан.");


// ============================================================================
// 2. Индексы для коллекции 'grades'
// ============================================================================

db.grades.createIndex(
    { "id_student": 1 },
    { name: "idx_grades_student_id", background: true }
);
print("Индекс 'idx_grades_student_id' (grades) создан.");

db.grades.createIndex(
    {
        "course_info.code_module": 1,
        "assessment_info.type": 1,
        "score": -1
    },
    { name: "idx_module_type_score", background: true }
);
print("Индекс 'idx_module_type_score' (grades) создан.");


// ============================================================================
// 3. Индексы для коллекции 'courses'
// ============================================================================

db.courses.createIndex(
    { "code_module": 1, "code_presentation": 1 },
    { unique: true, name: "idx_course_unique", background: true }
```

```
);
print("Уникальный индекс 'idx_course_unique' (courses) создан.");
```

3.3. Ключевые Индексы и Обоснование

Индексы были спроектированы как комбинированные, чтобы покрыть поля, используемые в стадиях $match и $sort ваших 10 аналитических запросов.

Коллекция	Индекс (MQL-ключ)	Назначение и Обоснование
grades	{""id_student"": 1}	Критически важен для ускорения операций $lookup (Запрос №8) по локальному полю, а также для быстрого поиска оценок конкретного студента.
students	{""id_student"": 1, unique: true}	Критически важен для foreignField в операции $lookup (Запрос №8) и обеспечения уникальности идентификатора студента.
grades	{""course_info.code_module"": 1, ""assessment_info.type"": 1, ""score"": -1}	Оптимизация аналитических запросов (№1, №2), фильтрующих по курсу и типу работы, с последующей сортировкой по баллу.
students	{""final_result"": 1, ""demographics.highest_education"": 1}	Оптимизация запросов по статусу завершения и демографии (Запрос №5), используя начальные фильтры для сужения набора данных.

Глава 4 Аналитические запросы к СУБД MongoDB

№ 1 Топ-5 самых активных курсов. Определяет связки "Модуль-Презентация" с наибольшим количеством сданных валидных оценок, что указывает на высокую вовлеченность или популярность.

```
db.grades.aggregate([
 { $match: { score: { $ne: -1 } } },
 { $group: {
     _id: { module: "$course_info.code_module", presentation:
"$course_info.code_presentation" },
     total_valid_grades: { $sum: 1 }
 }},
 { $sort: { total_valid_grades: -1 } },
 { $limit: 5 }
]).pretty()
```

```
[
  {
    _id: { module: 'FFF', presentation: '2013J' },
    total_valid_grades: 16228
  },
  {
    _id: { module: 'FFF', presentation: '2014J' },
    total_valid_grades: 16175
  },
  {
    _id: { module: 'BBB', presentation: '2013J' },
    total_valid_grades: 14365
  },
  {
    _id: { module: 'FFF', presentation: '2013B' },
    total_valid_grades: 12181
  },
  {
    _id: { module: 'CCC', presentation: '2014J' },
    total_valid_grades: 11445
  }
]
```

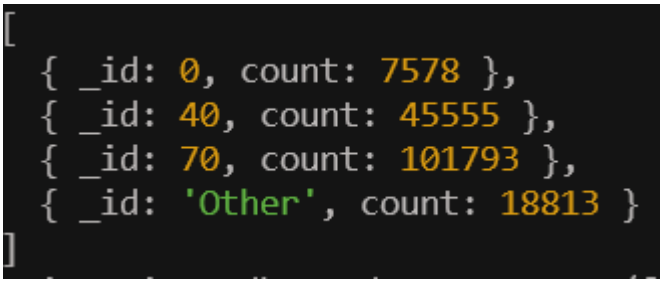
№ 2 Средний балл по типу работы в курсе 'AAA'. Позволяет сравнить, по какому типу оценочных работ (TMA, CMA, Exam) студенты курса 'AAA' показывают лучшие результаты. |

```
db.grades.aggregate([
  { $match: { "course_info.code_module": "AAA", score: { $ne: -1 } } },
  { $group: { _id: "$assessment_info.type", avg_score: { $avg: "$score" }
} },
  { $sort: { avg_score: -1 } }
]).pretty()
```

№ 3 Распределение оценок по диапазонам. Анализирует качество обучения, группируя все валидные оценки в диапазоны (0-40, 40-70, 70-100).

```
db.grades.aggregate([
  { $match: { score: { $ne: -1 } } },
  { $bucket: {
      groupBy: "$score",
      boundaries: [0, 40, 70, 100],
      default: "Other",
      output: { count: { $sum: 1 } }
```

```
    }}
  ]).pretty()
```



```
[
  { _id: 0, count: 7578 },
  { _id: 40, count: 45555 },
  { _id: 70, count: 101793 },
  { _id: 'Other', count: 18813 }
]
```

№ 4 Общее количество опозданий. Подсчитывает общее число всех оценок, сданных после крайнего срока (date_submitted > date_due).

```
db.grades.countDocuments({
  $expr: { $gt: ["$date_submitted", "$assessment_info.date_due"] }
})
```

```
52183
```

№ 5 Процент успешной сдачи по уровню образования. Сравнивает долю студентов, успешно завершивших курс (Pass/Distinction), в зависимости от их высшего образования.

```
db.students.aggregate([
  { $group: {
      _id: "$demographics.highest_education",
      total: { $sum: 1 },
      passed: { $sum: { $cond: [ { $in: ["$final_result", ["Pass",
"Distinction"]] }, 1, 0 ] } }
  }},
  { $project: {
      _id: "$_id",
      pass_rate: { $multiply: [ { $divide: ["$passed", "$total"] }, 100 ]
}
  }},
  { $sort: { pass_rate: -1 } }
]).pretty()
```

```
[
  { _id: 'Post Graduate Qualification', pass_rate: 65.49520766773162 },
  { _id: 'HE Qualification', pass_rate: 56.173361522198725 },
  { _id: 'A Level or Equivalent', pass_rate: 52.03275186899252 },
  { _id: 'Lower Than A Level', pass_rate: 38.850889192886456 },
  { _id: 'No Formal quals', pass_rate: 29.68299711815562 }
]
university>
```

№ 6 Топ-5 регионов по изученным кредитам. Выявляет регионы, где студенты в среднем берут на себя наибольшую учебную нагрузку.

```
db.students.aggregate([
  { $group: { _id: "$demographics.region", avg_credits: { $avg:
"$study_history.studied_credits" } } },
  { $sort: { avg_credits: -1 } },
  { $limit: 5 }
]).pretty()
```

```
[
  { _id: 'London Region', avg_credits: 82.89023631840796 },
  { _id: 'Wales', avg_credits: 81.60115052732502 },
  { _id: 'West Midlands Region', avg_credits: 80.99147947327653 },
  { _id: 'East Midlands Region', avg_credits: 80.91754756871036 },
  { _id: 'North Western Region', avg_credits: 80.6641431520991 }
]
university>
```

№ 7 Самая распространенная возрастная группа среди "отказников". Определяет возрастную группу, которая чаще всего бросает обучение (Withdrawn).

```
db.students.aggregate([
  { $match: { final_result: "Withdrawn" } },
  { $group: { _id: "$demographics.age_band", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 1 }
]).pretty()
```

```
[ { _id: '0-35', count: 7381 } ]
```

№ 8 Средний балл по полу студентов. Требует объединения ($lookup) коллекций grades и students для расчета среднего балла.

```
db.grades.aggregate([
  { $match: { score: { $ne: -1 } } },
  { $lookup: {
      from: "students",
      localField: "id_student",
      foreignField: "id_student",
      as: "s"
  }},
  { $unwind: "$s" },
  { $group: { _id: "$s.demographics.gender", avg_score: { $avg: "$score" }
} },
  { $sort: { avg_score: -1 } }
]).pretty()
```

```
university> db.grades.aggregate([
...     {$match: {score: {$ne: -1}}},
...     {$lookup: {from: "students", localField: "id_student", foreignField: "id_student", as: "s"}},
...     {$unwind: "$s"},
...     {$group: {_id: "$s.demographics.gender", avg_score: {$avg: "$score"}}},
...     {$sort: {avg_score: -1}}
... ]).pretty()
[
  { _id: 'F', avg_score: 75.92168290456121 },
  { _id: 'M', avg_score: 75.69546394830564 }
]
university>
```

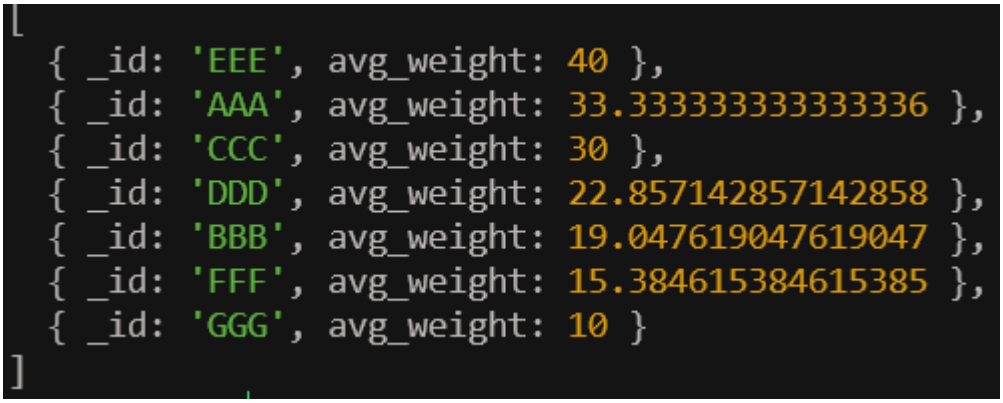
№ 9 Курсы с обязательным экзаменом (вес 100%). Выявляет модули, где один экзамен решает весь результат.

```
db.courses.aggregate([
  { $unwind: "$assessments" },
  { $match: { "assessments.weight": 100.0 } },
  { $group: { _id: "$code_module", count: { $sum: 1 } } },
  { $sort: { count: -1 } }
]).pretty()
```

```
[
  { _id: 'BBB', count: 16 },
  { _id: 'DDD', count: 16 },
  { _id: 'FFF', count: 16 },
  { _id: 'EEE', count: 9 },
  { _id: 'GGG', count: 9 },
  { _id: 'CCC', count: 8 },
  { _id: 'AAA', count: 4 }
]
```

№ 10 Средний вес оценочных работ по модулю. Позволяет сравнить среднюю "стоимость" одной работы по модулям.

```
db.courses.aggregate([
  { $unwind: "$assessments" },
  { $group: { _id: "$code_module", avg_weight: { $avg:
"$assessments.weight" } } },
  { $sort: { avg_weight: -1 } }
]).pretty()
```



Заключение

Все поставленные задачи проекта были успешно выполнены. Была создана масштабируемая и производительная архитектура MongoDB, основанная на денормализации данных, что позволило минимизировать операции соединения и обеспечить высокую скорость чтения. Процесс ETL был успешно реализован. Создание всех необходимых индексов решило критические проблемы производительности для сложных агрегаций. В результате, система продемонстрировала способность к быстрому выполнению всех десяти аналитических запросов, что подтверждает эффективность выбранного NoSQL-решения.