МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

Группа М8О-406Б-21 **Направление подготовки** 01.03.02 «Прикладная математика и информатика»

Профиль Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: Проектирование системы переноса и генерации взаимосвязанных данных из производственной среды при тестировании образовательной платформы

Автор ВКРБ:	Мезенин Олег Александрович	(_____)
Руководитель:	Миронов Евгений Сергеевич	(_____)
Консультант:	Ляпина Светлана Юрьевна	(_____)
Консультант:	—	(_____)
Рецензент:	—	(_____)

К защите допустить

Заведующий кафедрой № 806 Крылов Сергей Сергеевич (_____)

____ мая 2025 года

Москва 2025

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 36 страниц, 9 рисунков, 16 использованных источников, 1 приложения.

TBD: расписать keywords

TBD: расписать анотацию

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

Перенос данных — это процесс копирования и перемещения данных из одного источника или системы в другую без удаления данных из оригинальной базы данных

CI/CD — сокращение от Continuous Integration и Continuous Deployment, представляет собой совокупность практик и инструментов, применяемых в области разработки программного обеспечения с целью автоматизации и оптимизации процессов интеграции, тестирования и развёртывания приложений

DevOps-инженер — это специалист, работающий на стыке разработки программного обеспечения (development) и его эксплуатации (operations), задачей которого является создание надёжного и эффективного процесса разработки, тестирования и развёртывания приложений

CLI — сокращение от Command Line Interface, представляет собой интерфейс взаимодействия пользователя с операционной системой или программным обеспечением, который осуществляется посредством ввода текстовых команд в терминал

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

БД — база данных

СУБД — система управления базами данных

ВВЕДЕНИЕ

В настоящее время рынок онлайн-образования демонстрирует значительный рост. Всё большее число людей различных возрастных категорий предпочитают дистанционные форматы, которые обеспечивают гибкость и доступность обучения независимо от географического положения. Сегмент онлайн-образования, направленный на школьников, составляет значительную часть рынка. [1] В связи с этим внимание следует уделить платформам, разработанным для школьников, их родителей и учителей.

Подобные системы позволяют объединить учебные материалы, задания, оценки и обратную связь в одном пространстве. Для школьников это обеспечивает обучение в интерактивной и увлекательной форме, для родителей — возможность контролировать достижения своих детей и участвовать в их образовательном процессе, а для учителей — эффективное управление учебным процессом и возможность персонализированного подхода в обучении.

Эти платформы требуют надёжной и эффективной системы управления данными для обеспечения стабильной работы и поддержки большого количества пользователей. Одним из распространённых решений в области управления базами данных является PostgreSQL [2] — реляционная система управления базами данных с открытым исходным кодом.

В процессе разработки платформы важно не только работать над техническими аспектами, но и уделять особое внимание качеству и надежности. Это делает тестирование незаменимой стадией разработки. Благодаря тестированию можно убедиться в стабильности и бесперебойной работе систем.

Использование реальных данных и проведение тестов в производственной среде позволяет максимально точно оценить работоспособность системы. Но такой подход сопряжён с рисками, такими как нарушения конфиденциальности данных, потенциальные сбои в работе системы и возможное негативное влияние на реальных пользователей.

Чтобы тестирование образовательных платформ было безопасным и в то же время эффективным, необходимо создать тестовую среду, максимально приближенную к производственным условиям. Создание тестовой среды требует тщательной проработки и учета всех аспектов, связанных с

функционированием платформы. Это включает в себя настройку инфраструктуры, идентичной производственной, и воспроизведение всех взаимосвязей данных, которые присутствуют в реальных условиях эксплуатации.

Особую значимость в этом контексте приобретает работа с данными. Для достижения максимальной схожести с производственной средой необходимо использовать тестовые данные, которые точно отражают объёмы, структуры и взаимосвязи данных, присутствующие в реальной системе.

Создание тестового набора данных может быть реализовано с использованием различных методик. Один из подходов включает полный перенос данных из производственной базы данных с последующей анонимизацией конфиденциальной информации. Утилита pg_dump [3] может быть использована для экспорта данных из PostgreSQL, обеспечивая перенос структуры и содержимого базы данных.

Для защиты пользовательской информации и анонимизации данных можно применить утилиту pg_anonymizer [4], которая помогает скрыть чувствительные данные, заменяя их на фиктивные значения, что сохраняет конфиденциальность пользователей.

Несмотря на эффективное использование таких средств, основной недостаток данного метода заключается в невысокой скорости переноса всех данных, что может быть критичным при работе с крупными наборами данных и требовать значительных временных затрат для выполнения всей операции.

Частичный перенос данных может быть более эффективным для тестирования, так как зачастую не требуется полный объём данных для покрытия большинства сценариев. Однако этот подход сопряжён с рядом вопросов и потенциальных сложностей.

При полном переносе данных тестировщик получает доступ ко всем данным, что позволяет ему проводить тестовые сценарии. В случае частичного переноса данных требуется определить и описать необходимые для тестирования данные, избегая глубокого изучения структуры базы данных и её связей. Возникает вопрос: каким образом тестировщику эффективно описать данные, нужные для проведения тестов?

Следующим вопросом является сохранение целостности данных. В случае полного переноса реляционная целостность поддерживается автоматически, поскольку все взаимосвязанные данные переносятся

полностью, обеспечивая корректность и согласованность системы. При частичном переносе необходимо уделить особое внимание поддержанию целостности данных. Недопустимо, чтобы после переноса в тестовой среде данных было недостаточно для выполнения тестовых сценариев. Следовательно, второй вопрос заключается в следующем: как обеспечить целостность данных, минимизируя при этом объём переносимых данных?

Перенос данных из производственной среды с использованием анонимизации представляет собой лишь один из способов получения тестовых данных. Альтернативный подход состоит в генерации данных на основе заданных характеристик. Как может быть реализован этот процесс? Пользователь задаёт характеристики, такие как возраст, пол или средний балл по математике; каждый из параметров может сопровождаться набором или диапазоном значений. Программа затем генерирует тестовые данные, основываясь на фактических данных, либо на схеме данных производственной базы, а также на указанных пользователем характеристиках. Таким образом, полученные данные будут иметь структуру, аналогичную реальным данным, но не будут связаны с конкретными записями в производственной базе. Но для этого метода также возникают два ключевых вопроса: как пользователю описать необходимые данные и как их корректно сгенерировать.

Дипломная работа будет посвящена проектированию системы, обеспечивающей перенос взаимосвязанных данных из одной базы данных в другую на основе пользовательских запросов с применением методов анонимизации, а также способной генерировать тестовые данные. В рамках исследования будут рассмотрены алгоритмы переноса и генерации данных, а также разработка языка для описания данных, необходимых пользователю.

1 Постановка задачи и теоретические предпосылки

1.1 Тестирование

Чтобы определить требования к нашей системе, для начала нужно разобраться с основными аспектами тестирования программного обеспечения.

1.1.1 Процесс тестирования

Тестирование — это процесс оценки системы или её компонентов с целью проверки их соответствия заданным требованиям. Основная цель тестирования заключается в выявлении дефектов, обеспечении качества и подтверждении работоспособности программного обеспечения.

Процесс тестирования можно разбить на несколько ключевых этапов [5]:

а) Планирование:
- разрабатывается стратегия тестирования, включая выбор типов тестирования;
- создается тест-план, где описываются ресурсы, сроки, и критерии начала и завершения тестирования;
- проводится анализ рисков, связанных с тестированием и определяются действия для минимизации их воздействия.

б) Подготовка:
- разработка тестовых случаев и сценариев на основе требований и спецификаций;
- подготовка тестовой среды, включая оборудование и программные средства, необходимые для проведения тестов;
- подготовка данных для тестирования и настройка автоматизированных тестов, если они предусмотрены.

в) Проведение:
- непосредственное выполнение тестов согласно плану тестирования;
- фиксация и документирование результатов тестирования, выявление дефектов и несоответствий.

г) Совершенствование:

- анализ результатов тестирования и составление отчёта по итогам выполненной работы;
- обмен опытом и полученными знаниями внутри команды и с другими заинтересованными сторонами;
- обновление и улучшение тестовых документов и процессов на основе полученных данных.

Наша система будет применяться на этапе подготовки: с её помощью можно будет создавать тестовые данные.

1.2 Ручное и автоматизированное тестирование

Тестирование можно классифицировать на ручное и автоматизированное.

Ручное тестирование предполагает процесс, при котором тестировщик выполняет тесты без помощи автоматизированных инструментов. Оно позволяет глубже вникнуть в пользовательский опыт, и его легко применять к новым или часто изменяющимся функциональностям, однако оно может быть времяемким, подвержено человеческим ошибкам и не всегда позволяет повторить результаты для их сравнения.

Автоматизированное тестирование предполагает использование программных инструментов для автоматизации выполнения тестовых сценариев. Это предполагает разработку скриптов, которые автоматически выполняют тесты и сверяют результаты с ожидаемыми. Автоматизированное тестирование обеспечивает высокую скорость и стабильность результатов, что делает его эффективным для регрессионного тестирования и стабильных функциональностей, однако требует значительных ресурсов для разработки скриптов и не всегда подходит для тестирования пользовательских интерфейсов и динамически изменяющихся требований.

Оба подхода часто используются совместно для достижения наилучшего результата. Обычно ручное тестирование применяют на начальных стадиях или для исследовательского тестирования, тогда как автоматизированное тестирование использует для повторяющихся задач или сложных сценариев, которые требуют стабильного выполнения.

1.3 Сценарии использования

Перед определением требований к нашей системы рассмотрим возможные сценарии использования.

Определим роли:

– тестировщик — специалист, занимающийся проверкой качественности, надёжности и работоспособности программного обеспечения;
– разработчик — специалист, занимающийся разработкой программного обеспечения;
– DevOps-инженер — специалист, занимающийся поддержкой процессов CI/CD.

Будем называть метаданными входные данные, включающие в себя правила переноса, генерации или анонимизации данных.

1.3.1 Перенос данных для проведения тестовых сценариев

Роли:

– тестировщик;
– разработчик.

Цель: создать тестовые данные на основе существующих реальных данных для их дальнейшего применения в тестовых сценариях.

Предпосылки:

– разработчиком написаны метаданные, включающие в себя правила переноса и анонимизации данных;
– тестировщик осведомлен о метаданных, необходимых для проведения тестовых сценариев;
– тестировщик имеет уникальный идентификатор исходной базы данных;
– имеется пустая тестовая база данных, доступ к которой имеется у тестировщика.

Основной сценарий:

а) Тестировщик осуществляет изменения в метаданных при необходимости.
б) Тестировщик инициирует запуск CLI, указывая следующие параметры:

- уникальный идентификатор исходной базы;
- уникальный идентификатор целевой базы или параметры подключения к ней;
- метаданные или уникальный идентификатор метаданных.

в) Тестировщик ожидает завершения процесса переноса данных.

Постусловия:

- тестировщик применяет данные, сгенерированные на основе реальных данных, в тестовых сценариях.

Исключения:

- тестировщик получает ошибку некорректных метаданных, если он менял их, после пункта б.

1.3.2 Диагностика ошибки

Роли:

- тестировщик.

Цель: провести диагностику ошибки, возникшей с определенными данными в продуктивной среде.

Предпосылки:

- в продуктивной среде обнаружены проблемы с определенными данными, такими как данные пользователя;
- тестировщик владеет уникальным идентификатором данных, в которых выявлена ошибка в продуктивной среде;
- тестировщик имеет уникальный идентификатор исходной базы данных;
- имеется пустая тестовая база данных, доступ к которой имеется у тестировщика.

Основной сценарий:

а) Тестировщик описывает метаданные, включая данные, в которых произошла ошибка.

б) Тестировщик инициирует запуск CLI, указывая следующие параметры:

- уникальный идентификатор исходной базы;
- уникальный идентификатор целевой базы или параметры подключения к ней;
- метаданные.

в) Тестировщик ожидает завершения процесса переноса данных.

Альтернативный сценарий:

– тестировщик может взять готовые метаданные, если они ему подходит, а не описывать свои.

Постусловия:

– тестировщик осуществляет взаимодействие с перенесенными данными и проводит диагностику ошибки.

Исключения:

– тестировщик получает ошибку некорректных метаданных после пункта б.

1.3.3 Тестирование новой миграции

Роли:

– тестировщик;
– разработчик.

Цель: провести тестирование новой миграции.

Предпосылки:

– тестировщик имеет уникальный идентификатор исходной базы данных;
– имеется пустая тестовая база данных, доступ к которой имеется у тестировщика;
– разработчик создал новую миграцию, нуждающуюся в тестировании;
– разработчик предоставил метаданные к миграции, содержащие правила анонимизации и переноса данных.

Основной сценарий:

а) Тестировщик инициирует запуск CLI, указывая следующие параметры:
    – уникальный идентификатор исходной базы;
    – уникальный идентификатор целевой базы или параметры подключения к ней;
    – предоставленные метаданные.
б) Тестировщик ожидает завершения процесса переноса данных.

Постусловия:

– тестировщик взаимодействует с перенесенными данными и

осуществляет тестирование миграций.

1.3.4 Перенос данных для автоматизированного тестирования

Роли:

– тестировщик;
– разработчик;
– DevOps-инженер.

Цель: обеспечеть возможность проведения автоматизированного тестирования изменений через сценарий CI/CD.

Предпосылки:

– разработчиком написаны метаданные, включающие в себя правила переноса и анонимизации данных;
– тестировщиком реализованы автоматизированные тесты;
– DevOps-инженер разработал сценарий CI/CD, который, в случае появления нового запроса на слияния в системе контроля версий, инициирует развёртывание тестового окружения с пустой базой данных и осуществляет перенос необходимых данных с помощью нашей системы.

Основной сценарий:

а) Разработчик вносит изменения в исходный код и создает новый запрос на слияние в системе контроля версий.
б) Запускается процесс CI/CD, который выполняет следующие действия:
    – развёртывание тестового окружения с пустой базой данных;
    – перенос необходимых данных в тестовую базу с помощью нашей системы;
    – автоматический запуск тестов для проверки корректности внесённых изменений.

1.3.5 Генерация данных для проведения тестовых сценариев

Роли:

– тестировщик;
– разработчик.

Цель: создать тестовые данные на основе существующих реальных данных для их дальнейшего применения в тестовых сценариях.

Предпосылки:

- разработчиком написаны метаданные, включающие в себя правила генерации данных;
- тестировщик осведомлен о метаданных, необходимых для проведения тестовых сценариев;
- тестировщик имеет уникальный идентификатор исходной базы данных;
- имеется пустая тестовая база данных, доступ к которой имеется у тестировщика.

Основной сценарий:

а) Тестировщик осуществляет изменения в метаданных при необходимости.

б) Тестировщик инициирует запуск CLI, указывая следующие параметры:
   - уникальный идентификатор исходной базы;
   - уникальный идентификатор целевой базы или параметры подключения к ней;
   - метаданные или уникальный идентификатор метаданных.

в) Тестировщик ожидает завершения процесса генерации данных.

Постусловия:

- тестировщик применяет сгенерированные данные в тестовых сценариях.

Исключения:

- тестировщик получает ошибку некорректных метаданных, если он менял их, после пункта б.

1.4 Определение требований к системе

Требования к системе можно классифицировать на две основные категории: функциональные и архитектурные. Функциональные требования описывают конкретный набор функций и возможностей, которые система должна обеспечивать для удовлетворения потребностей пользователей. Требования к свойствам архитектуры системы фокусируются на качественных характеристиках системы, которые обеспечивают её

устойчивую и эффективную работу.

1.4.1 Функциональные требования

Система должна поддерживать следующий функционал:

- Перенос и анонимизация данных. Система должна обеспечивать возможность переноса взаимосвязанных данных между базами данных, а также их анонимизацию. Правила выбора взаимосвязанных данных и правила анонимизации задаются метаданными.
- Генерация данных. Система должна предоставлять функционал для генерации данных. Правила генерации задаются метаданными.
- Работа с метаданными. Система должна осуществлять проверку корректности метаданных, а также обеспечивать функциональность для загрузки предварительно подготовленных метаданных или их интеграции на этапе ввода данных.
- Гибкость в выборе базы данных. Необходимо внедрить функциональность, позволяющую пользователю выбрать базу данных, указав как уникальный идентификатор базы, так и строку подключения.

1.4.2 Требования к свойствам архитектуры

Архитектура системы должна соответствовать следующим критериям:

- Безопасность. Архитектура должна обеспечивать защиту от несанкционированного доступа к базе данных, а также обеспечивать защиту конфиденциальной информации.
- Надёжность. Система должна функционировать без сбоев в течение продолжительных периодов времени. Это предполагает наличие мер по обеспечению отказоустойчивости и возможности восстановления после сбоев.
- Высокая производительность. Архитектура должна обеспечивать эффективную обработку больших объёмов данных.
- Асинхронная обработка запросов. Поскольку перенос или генерация данных может занимать значительное время, система должна поддерживать выполнение запросов в асинхронном режиме. Это

означает, что клиентские приложения не должны блокироваться в ожидании завершения операции, но должны иметь возможность проверять статус выполнения запроса по мере необходимости.

2 Проектирование системы, разработка языка и алгоритма

2.1 Архитектура системы

Рассмотрим архитектуру системы по модели C4 [6].

2.1.1 Взаимодействие систем

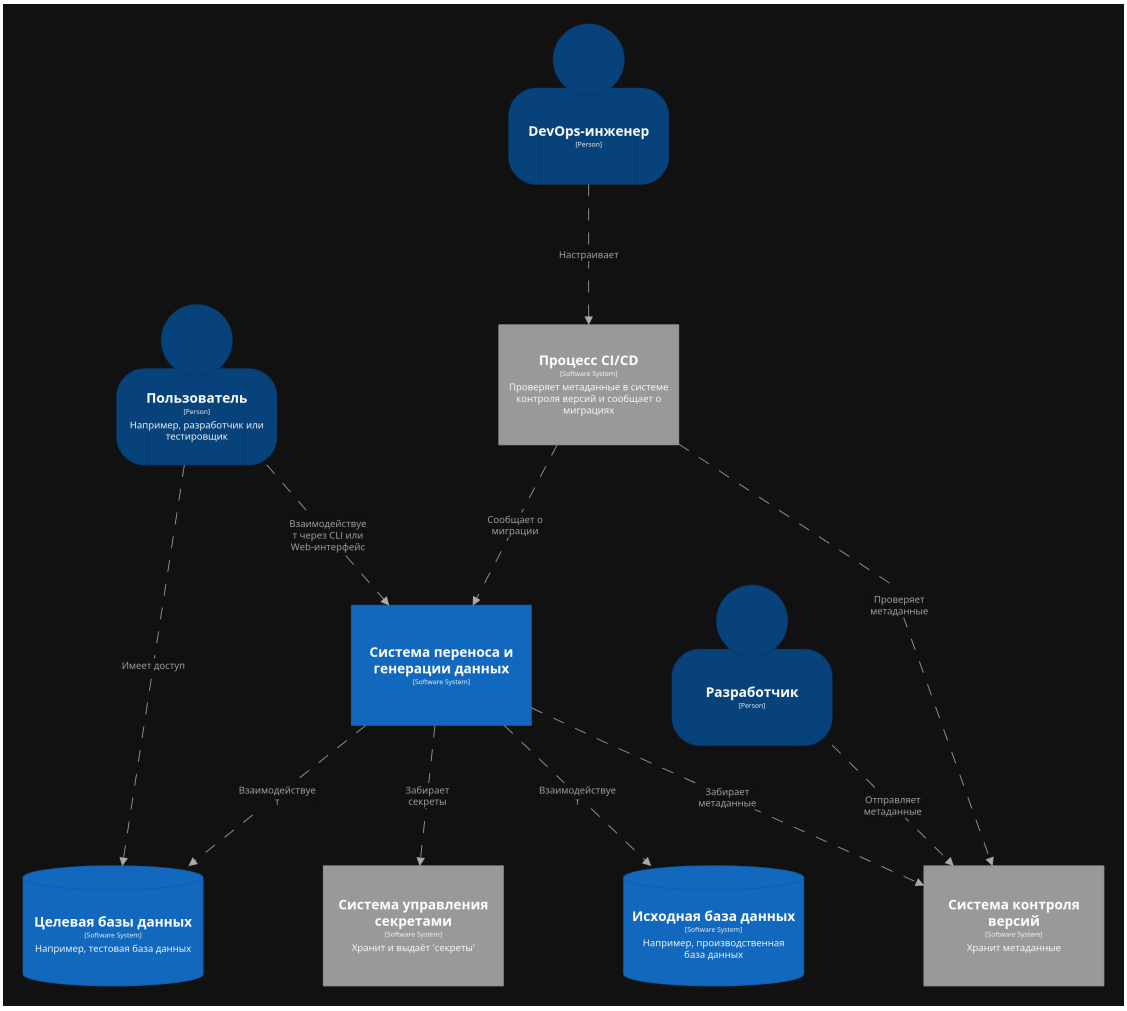

Рисунок 1 – Взаимодействие систем

На уровне контекста можно наблюдать взаимодействие различных систем, а также взаимодействие этих систем с пользователями и специалистами.

Внешние системы выделены серым цветом:

– система контроля версий выступает в качестве хранилища метаданных, представленных разработчиком;
– система управления секретами предоставляет данные для

подключения к базе данных;

- процесс CI/CD, который настраивает DevOps-инженер, выполняет несколько функций:
  - генерация тестовой среды и запуск автоматизированных тестов при получении запроса на слияние в систему контроля версий;
  - проверка корректности метаданных в момент их фиксации в системе контроля версий;
  - при выполнении миграций осуществляется уведомление системы переноса и генерации данных о процессах миграции – это необходимо для предотвращения неопределенного поведения, которое может возникнуть, если система в текущий момент работает с мигрируемой базой данных.

Синим цветом выделены собственные системы: исходная и целевая базы данных, с которыми осуществляется взаимодействие пользователей, а также система переноса и генерации данных, которую мы рассмотрим более детально далее.

2.1.2 Система переноса и генерации данных

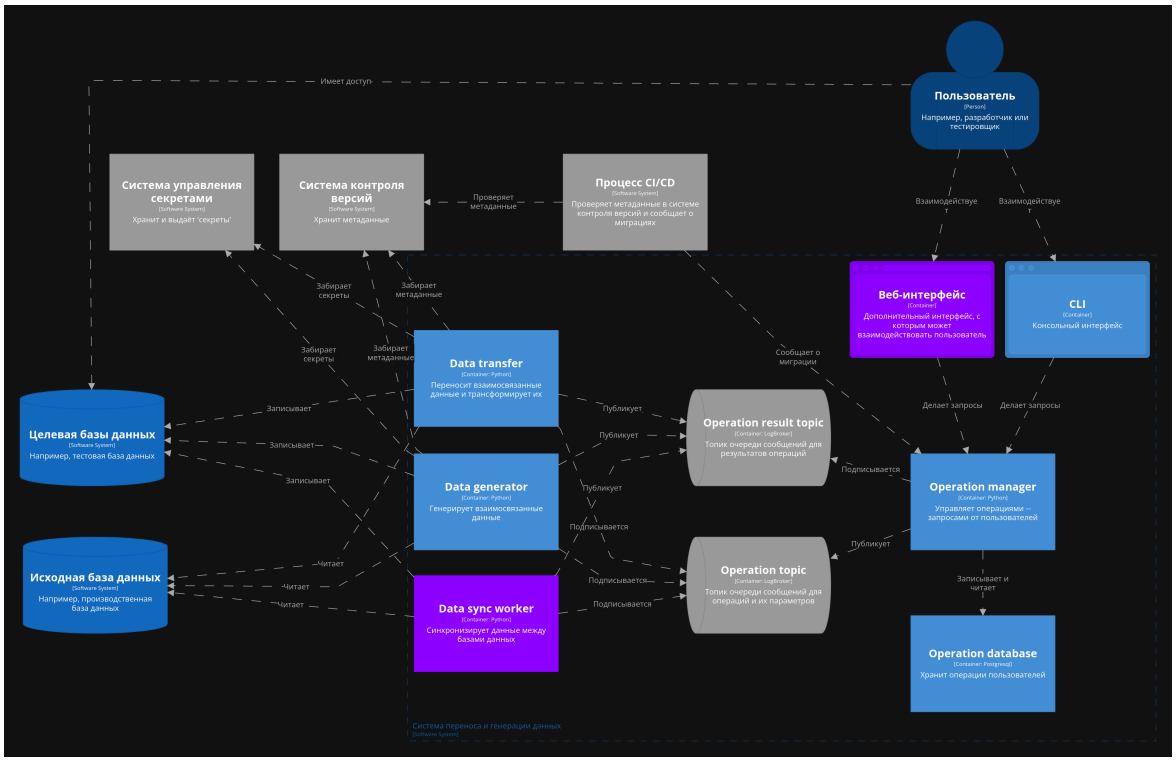

Рисунок 2 – Система переноса и генерации данных

Рассмотрим систему переноса и генерации данных на уровне контейнеров.

Взаимодействие пользователя с системой осуществляется через командную строку (CLI), хотя может быть реализован и альтернативный интерфейс взаимодействия, например, веб-интерфейс (выделен фиолетовым как контейнер, который может быть добавлен в перспективе).

Пользователь посылает запросы в контейнер Operation Manager. Возможны два типа запросов: запуск операции по переносу или генерации данных, а также проверка статуса выполняемой операции. В случае получения запроса на запуск операции, информация об операции сохраняется в базе данных, а уникальный идентификатор операции возвращается пользователю, что позволяет ему отслеживать статус выполнения.

Далее Operation Manager отправляет запросы на выполнение операции в очередь сообщений Logbroker [7]. Контейнер, обрабатывающий такой запрос, определяется типом операции: если речь идет о переносе данных, операция обрабатывается контейнером Data Transfer; если о генерации данных — контейнером Data Generator.

На схеме также представлен контейнер Data Sync Worker, являющийся гипотетическим контейнером, предназначенным для обработки операций по синхронизации данных в базах данных.

Контейнеры Data Transfer и Data Generator осуществляют перенос и генерацию данных, взаимодействуя с базами данных, системой контроля версий и системой управления секретами. В процессе выполнения операции, а также после её выполнения, информация о статусе возвращается в очередь сообщений, из которой Operation Manager извлекает данные и обновляет статус операции в базе данных.

Более детальное описание структуры компонентов Data Transfer и Data Generator будет представлено в последующих разделах.

Рассмотрим диаграмму последовательности взаимодействия пользователя с системой на примере генерации данных.

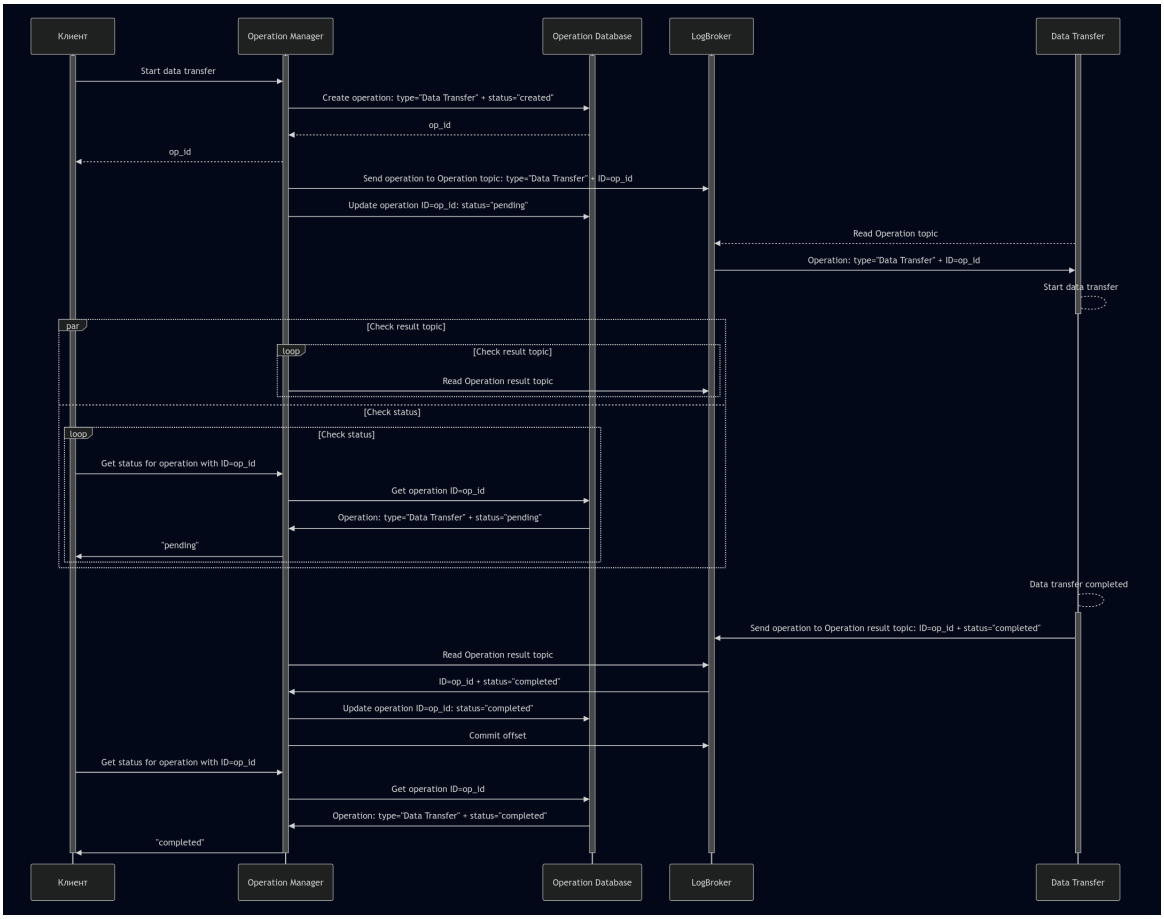

Рисунок 3 – Диаграмма последовательности взаимодействия пользователя с системой на примере генерации данных

На диаграмме можно заметить, как клиент и система общаются асинхронно.

2.1.3 Компоненты переноса данных

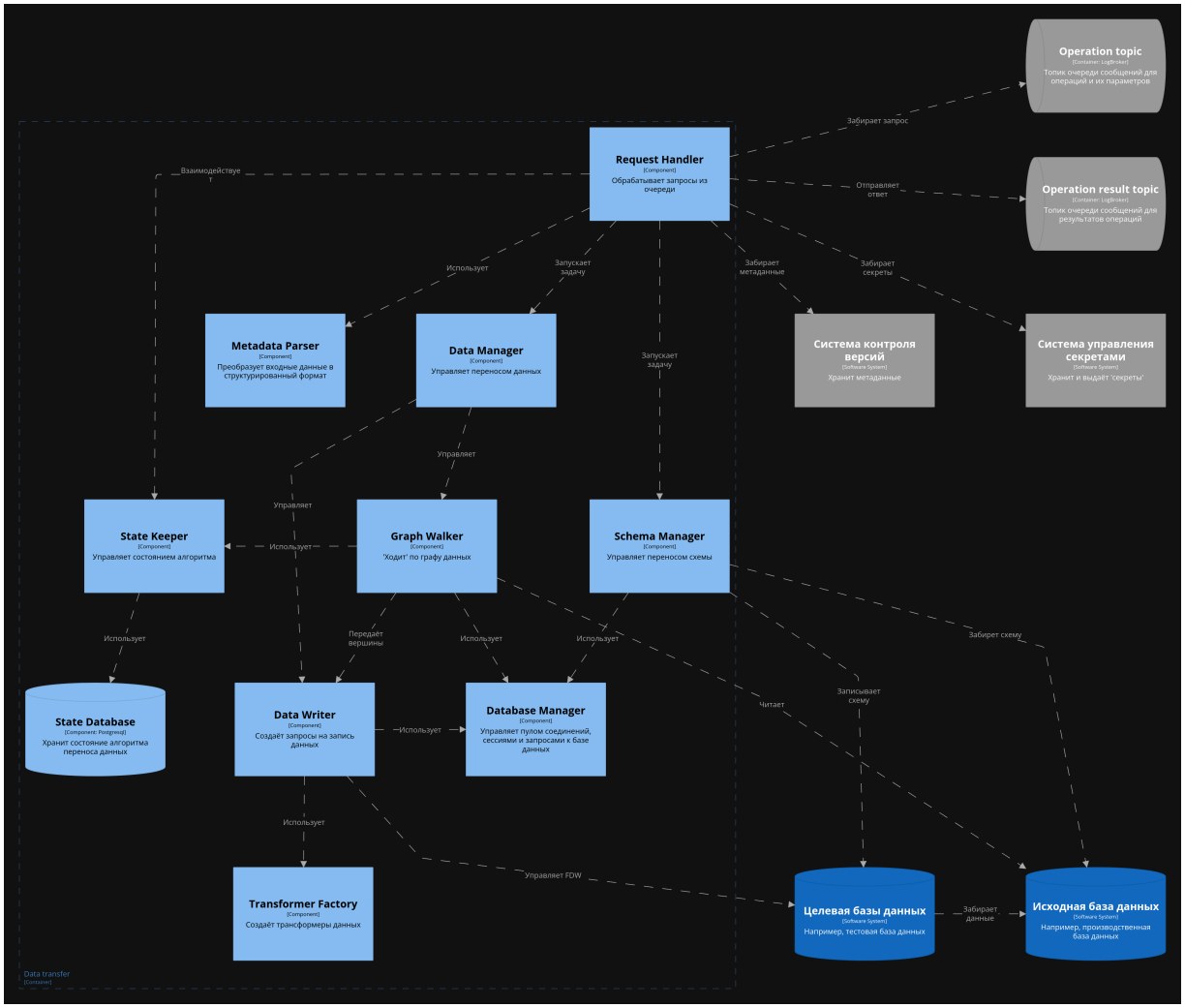

Рисунок 4 – Компоненты Data Transfer

Рассмотрим процесс обработки запросов на перенос задач. Компонент Request Handler принимает такие запросы. Они должны включать следующие элементы:

– идентификатор метаданных;
– идентификатор исходной базы данных;
– идентификатор целевой базы данных.

Идентификатор метаданных может принимать форму одной из двух сущностей: уникального идентификатора, связанного с метаданными в Системе контроля версий, или самих метаданных. В случае предоставления уникального идентификатора, Request Handler обращается к Системе контроля версий для извлечения соответствующих метаданных.

Идентификатор базы данных может быть представлен либо в форме

уникального идентификатора кластера базы данных, либо в виде параметров подключения к базе данных. При предоставлении уникального идентификатора кластера базы данных, Request Handler обращается к Системе управления секретами для получения необходимых параметров подключения, включая пароль.

Полученные метаданные направляются в компонент Metadata Parser для их валидации и преобразования во внутренний формат представления. Далее инициируется Data Manager, который получает информацию о подключениях к базам данных, а также внутреннее представление метаданных, и запускает компоненты Graph Walker и Data Writer.

Компонент Graph Walker выполняет алгоритм обхода данных исходной базы, который будет рассмотрен позже, следуя инструкциям, заданным в метаданных. Он использует State Keeper для сохранения состояния алгоритма и Database Manager для выполнения операций с базой данных. В процессе работы, Graph Walker извлекает метаинформацию о посещенных вершинах и асинхронно передает её компоненту Data Writer.

Компонент Data Writer управляет преобразованием данных и их записью в целевую базу данных. Он принимает на вход метаинформацию о данных, включая их идентификаторы, и с помощью Database Manager создаёт объект Foreign Data Writer [8] в целевой СУБД, обеспечивая прямое подключение к исходной СУБД. Data Writer формирует запросы, включающие идентификаторы данных, направляя их в целевую СУБД для извлечения конкретных данных из исходной базы. Запросы дополняются трансформерами, сформированными на основе метаданных через компонент Transformer Factory. Трансформер представляет собой набор правил преобразования данных, например, для анонимизации данных.

Рассмотрим взаимодействие основных компонент на диаграмме последовательности.

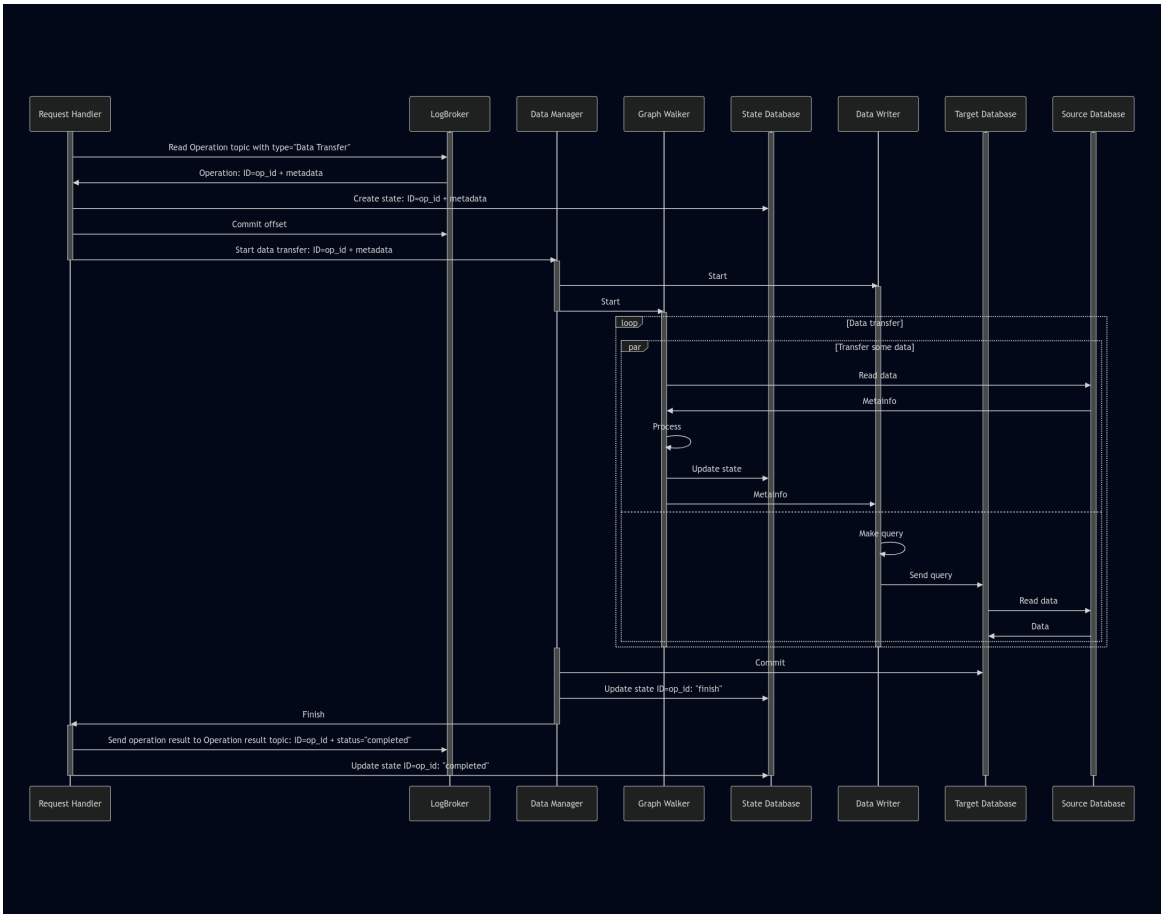

Рисунок 5 – Диаграмма последовательности взаимодействия пользователя с системой на примере генерации данных

Данный подход предусматривает непосредственный перенос данных из исходной базы в целевую, с возможными изменениями в соответствии с правилами, определёнными в метаданных, что способствует повышению производительности системы.

Также на рисунке 4 представлен компонент Schema Manager, который предоставляет функциональность по управлению схемами баз данных. В частности, данный компонент позволяет осуществлять перенос схемы между различными базами данных, а также выполнять удаление схемы.

2.1.4 Компоненты генерации данных

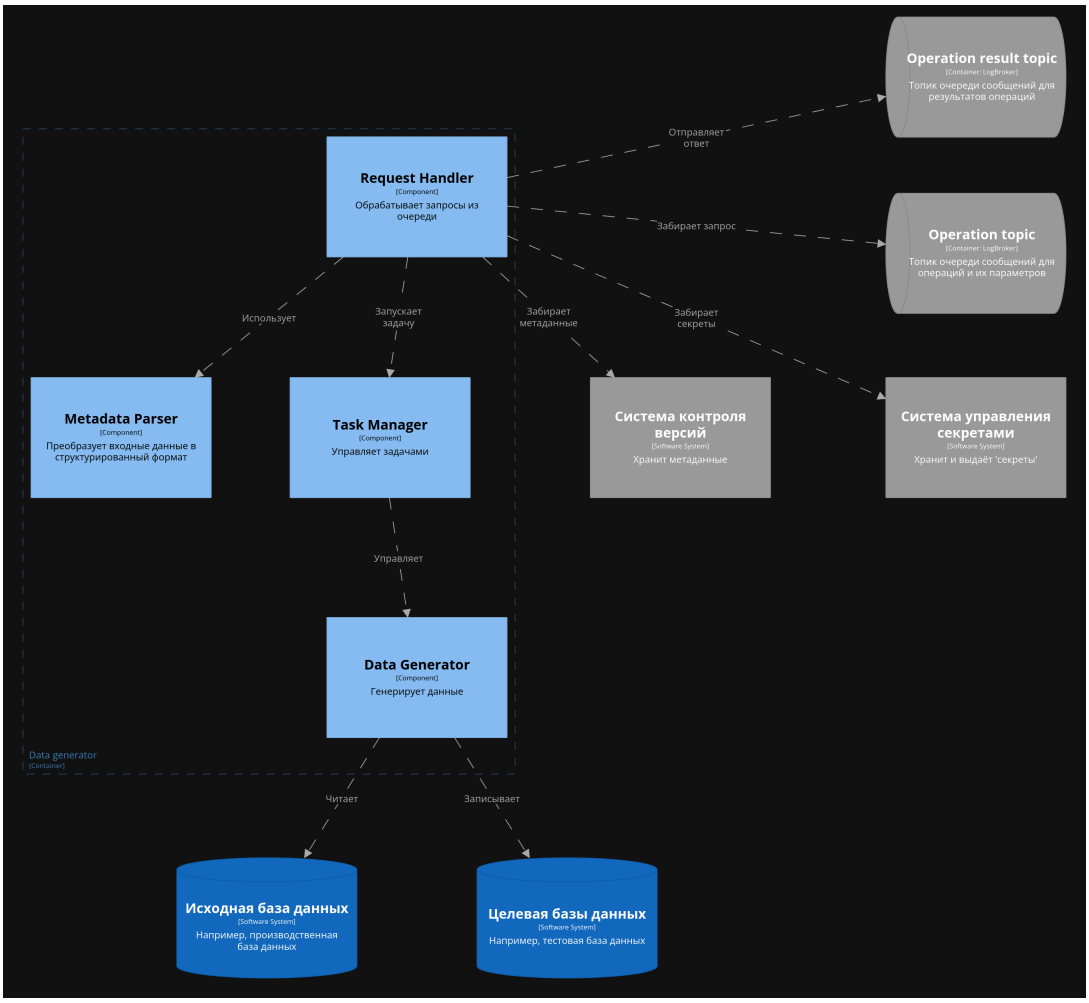

Рисунок 6 – Компоненты Data Generator

TBD: расписать взаимодействие компонент Data generator

2.1.5 Персистентность состояния и восстановление после сбоя

Процессы переноса данных могут быть времязатратными, независимо от объёма данных. В случае неожиданной остановки сервиса переноса и генерации данных, вызванной сбоем, утрата прогресса по переносу данных становится крайне нежелательным исходом.

Рассмотрим, как выполняется персистентность состояния системы, то есть её способность сохранять и восстанавливать своё состояние после остановки.

Весь процесс выполнения операции от её инициации до её завершения можно разделить на два этапа:

– отправка и чтение из очереди сообщений;
– непосредственный процесс переноса данных.

Сбои могут возникать на любом из этих этапов. Для этапа отправки и чтения сообщений из очереди предусмотрена семантика доставки exactly-once [9], которая поддерживается использованием архитектурных паттернов Inbox Pattern и Outbox Pattern [10]. Эти паттерны, часть реализации которых можно наблюдать на диаграммах последовательностей 3 и 5, обеспечивают гарантированную доставку сообщений и идемпотентную обработку входящих данных.

При сбое на этапе переноса данных механизмы восстановления включают проверку состояния незавершённых операций в базе данных состояния (State Database). В ней хранится прогресс каждой операции переноса данных и включает уникальные идентификаторы операции, состояние алгоритма и метаданные. При восстановлении система переноса данных проверяет наличие незавершённых операций и продолжает их с того места, где процесс был остановлен. Запись в целевую базу данных успешно завершается только при успешном выполнении алгоритма, что требует повторной записи данных в случае восстановления после сбоя, основываясь на метаинформации из сохранённого состояния.

Таким образом, архитектура системы включает механизмы обеспечения персистентности состояния и восстановления после сбоев.

2.1.6 Безопасность

TBD: расписать, как выполняется свойство безопасности системы

2.2 Алгоритм обхода данных

Всю основную функциональность переноса данных можно разделить на две части: запись данных в целевую базу данных, которую мы рассматривали ранее, и обход исходных данных, алгоритм которого мы рассмотрим подробнее в этом разделе.

2.2.1 Метаграфы

На сегодняшний день в научном сообществе не существует общепринятого и устойчивого определения метаграфа. Работа [11] была

первым источником, в котором появился термин ”метаграф”. Эта концепция всё ещё находится в стадии развития и интерпретируется различными исследователями по-разному в зависимости от их специфических задач и целей [12] [13] [14] [15]. В общем смысле, метаграф можно рассматривать как структуру, которая обобщает и расширяет идеи классических графов, чтобы решить более сложные задачи анализа данных и моделирования.

В данной дипломной работе предлагается собственное определение метаграфа, адаптированное под требования и задачи рассматриваемого алгоритма.

$MG =< V, MV, E, ME >$ – метаграф, где V – множество вершин, MV – множество метавершин, E – множество рёбер, ME – множество метарёбер.

$v_i = \{atr_k\}, v_i \in V$, где v_i – вершина, atr_k – атрибут.

$mv_i =< \{v_j\}, \{atr_k\} >, mv_i \in MV, v_j \in V$, где mv_i – метавершина, v_j – вершина, atr_k – атрибут.

$e_i =< v_s, v_e >, e_i \in E, v_s, v_e \in V$, где e_i – ребро, v_s – исходная вершина, v_e – конечная вершина.
$me_i =< mv_s, mv_e, \{atr_k\} >, me_i \in ME, mv_s, mv_e \in MV$, где me_i – метаребро, mv_s – исходная метавершина, mv_e – конечная метавершина, atr_k – атрибут.

Также введём ограничение на множество метавершин: $\forall mv_i, mv_j \in V, i \neq j => \{v_x\}_i \cap \{v_y\}_j = \emptyset$, т.е. каждая вершина не может содержаться в нескольких метавершинах.

2.2.2 Представление базы данных в виде метаграфа

Реляционные базы данных традиционно состоят из множества связанных таблиц, каждая из которых содержит структурированные данные. В метаграфе эти элементы могут быть представлены следующим образом.

- Метавершины. В контексте метаграфа метавершинам соответствуют конкретные таблицы реляционной базы данных. Набор атрибутов метавершины содержат информацию о структуре таблицы, включая её название, идентификатор и набор столбцов.
- Вершины. Вершины, представляющие собой элементы более низкого уровня в сравнении с метавершинами, соответствуют записям в таблицах. Каждый набор атрибутов вершины напрямую соответствует набору значений в конкретной записи таблицы.
- Метарёбра. Метарёбра описывают связи между различными

таблицами в базе данных. Эти связи формируются посредством внешних ключей [16], которые указывают на зависимость одной таблицы от другой. Атрибуты метарёбер содержат информацию о полях, которые связаны между таблицами, что создаёт основу для передачи данных и взаимозависимости структур.

– Рёбра. Рёбра представляют связи между конкретными записями в таблицах и формируются на основании соответствующих метарёбер. Эти рёбра конкретизируют связь на уровне данных, отражая указанные в метарёбрах отношения через конкретные внутренние связи записей, например, через одинаковые или согласованные идентификаторы.

Для иллюстрации представления базы данных в виде метаграфа рассмотрим небольшой пример, содержащий пять таблиц по несколько записей в каждой.

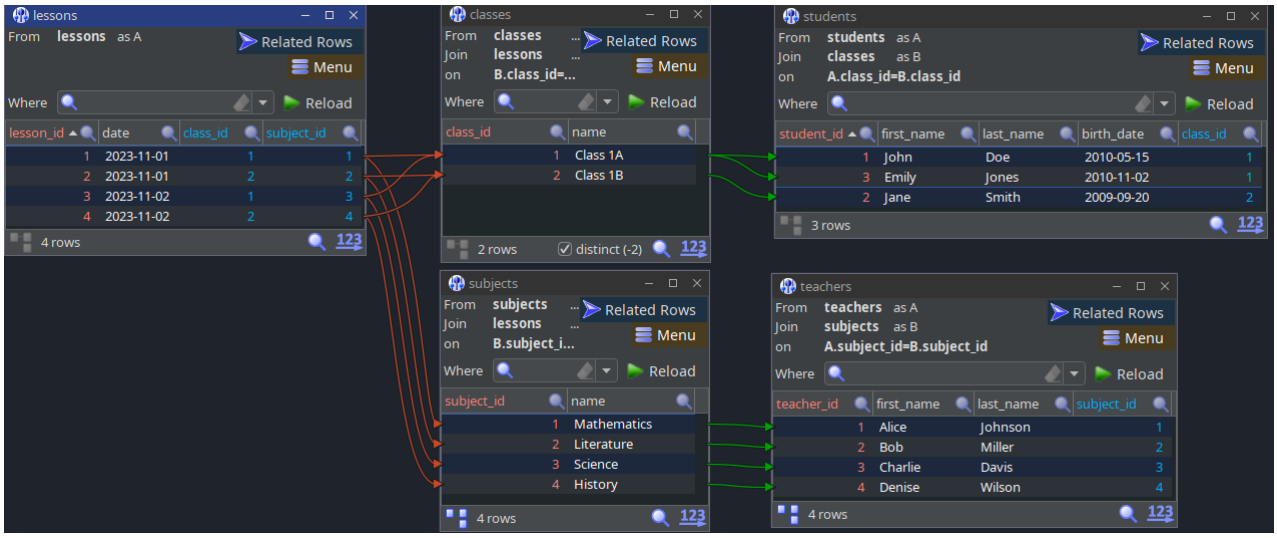

Рисунок 7 – Пример реляционной базы данных

На рисунке 7 представлена схема взаимосвязей между конкретными записями в базе данных. Красные стрелки указывают на то, что записи, из которых исходят стрелки, имеют ссылки на другие записи. Это свидетельствует о наличии во внешней таблице внешнего ключа, ссылающегося на другую таблицу. В свою очередь, зелёные стрелки обозначают противоположную ситуацию: данные, на которые они указывают, являются объектом ссылок со стороны других записей.

Рассмотрим графическое представление метаграфа для такой базы данных.

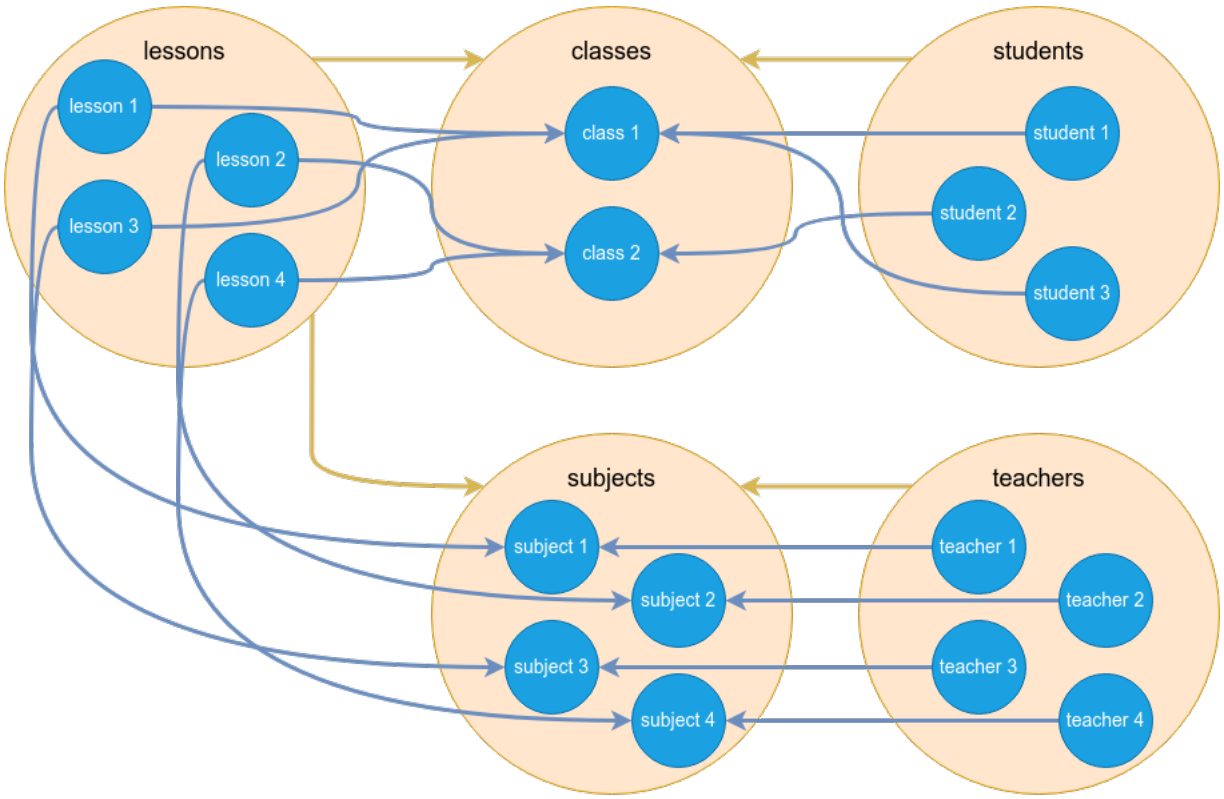

Рисунок 8 – Пример метаграфа

TBD: Расписать про метаграф.

2.2.3 Описание алгоритма

TBD: здесь хочу описать общий алгоритм, не привязываясь к конкретным СУБД

```
1 // TBD: написать алгоритм
2 BFSforMetagraph(graph, startVertex)
3     // Инициализация
4     queue ← {}
5     visited ← {}
6
```

Рисунок 9 – Алгоритм

2.2.4 Устройство Postgresql

TBD: здесь хочу описать некоторые моменты из устройства Postgresql: ctid, tableoid – они применяются в имплементации алгоритма.

2.2.5 Имплементация

TBD: здесь хочу наложить алгоритм на Postgresql и написать имплементацию алгоритма на Python (которая уже готова, но её просто так нельзя выкладывать :D).

2.3 Генерация данных

TBD: генерация данных далеко не основная функциональность системы, поэтому времени ей будет уделено немного

2.4 Разработка языка

TBD: есть синтаксис языка, но нужно реализовать парсер. Структура этого раздела пока под вопросом

2.4.1 Синтаксис и грамматика языка

2.4.2 Корректность языка

2.4.3 Язык задания графа

2.4.4 Язык переноса данных

2.4.5 Язык генерации данных

3 Демонстрация программного продукта

TBD: по плану протестировать прототип на реальной базе + тесты производительности + анализ результатов + выводы жизнеспособности системы

ЗАКЛЮЧЕНИЕ

TBD: написать заключение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исследование рынка онлайн-образования. — URL: https://main.talenttech.ru/research/issledovanie-rynka-onlajn-obrazovaniya/ (дата обращения 01.03.2025).

2. PostgreSQL: The world’s most advanced open source database. — URL: https://www.postgresql.org/ (дата обращения 02.01.2025).

3. PostgreSQL: Documentation: 17: pg_dump. — URL: https://www.postgresql.org/docs/current/app-pgdump.html (дата обращения 02.01.2025).

4. PostgreSQL: pg_anonymize, a new extension for simple and transparent data anonymization. — URL: https://www.postgresql.org/about/news/pg_anonymize-a-new-extension-for-simple-and-transparent-data-anonymization-2606/ (дата обращения 02.01.2025).

5. Блэк Р. Ключевые процессы тестирования. — Лори, 2025. — ISBN 978-5-85582-392-9.

6. Home | C4 model. — URL: https://c4model.com/ (дата обращения 29.03.2025).

7. Logbroker: сбор и поставка больших объемов данных в Яндексе / Хабр. — URL: https://habr.com/ru/companies/yandex/articles/239823/ (дата обращения 05.04.2025).

8. PostgreSQL: Documentation: 17: F.36. postgres_fdw — access data stored in external PostgreSQL servers. — URL: https://www.postgresql.org/docs/current/postgres-fdw.html (дата обращения 30.03.2025).

9. Механизм доставки сообщений в Kafka | ADS Arenadata Docs Guide. — URL: https://docs.arenadata.io/ru/ADStreaming/current/concept/architecture/kafka/delivery_guarantees.html (дата обращения 05.04.2025).

10. Microservices 101: Transactional Outbox and Inbox. — URL: https://softwaremill.com/microservices-101/ (дата обращения 05.04.2025).

11. Basu A. R. W. B. Metagraphs and their applications. — New York: Springer, 2007.

12. Астанин С. В. Драгныш Н. В. Ж. Н. К. Вложенные метаграфы как модели сложных объектов // Инженерный вестник Дона. — 2012. — URL: http: //www.ivdon.ru/magazine/archive/n4p2y2012/1434.

13. С. Глоба М. Ю. Терновой Е. С. Ш. Метаграфы как основа для представления и использования баз нечетких знаний // Открытые семантические технологии проектирования интеллектуальных систем = Open Semantic Technologies for Intelligent Systems (OSTIS-2015) : материалы V междунар. науч.-техн. конф. (Минск, 19-21 февраля 2015 года). — 2015. — С. 237—240.

14. Самохвалов Э. Н. Ревунков Г. И. Г. Ю. Е. Использование метаграфов для описания семантики и прагматики информационных систем // Вестник МГТУ им. Н. Э. Баумана. Сер. «Приборостроение». — 2015. — С. 83—99. — URL: https://vestnikprib.bmstu.ru/articles/673/673.pdf.

15. Черненький В. М. Терехов В. И. Г. Ю. Е. Представление сложных сетей на основе метаграфов // Нейроинформатика-2016. XVIII Всероссийская научно-техническая конференция. Сборник научных трудов. Ч. 1. М.: НИЯУ МИФИ, 2016. — 2016. — С. 173—178.

16. PostgreSQL – Foreign Key | GeeksforGeeks. — URL: https : / / www . geeksforgeeks . org / postgresql - foreign - key/ (дата обращения 12.04.2025).

ПРИЛОЖЕНИЕ А
Исходный код

TBD: ссылка на исходный код