



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:
«Метод миграции данных из реляционной в
документно-ориентированную базу данных с
использованием частотного и семантического
анализа»

Студент ИУ7-84Б
(Группа)

(Подпись, дата)

Шингаров И. Д.
(И. О. Фамилия)

Руководитель ВКР

(Подпись, дата)

Вишневская Т. И.
(И. О. Фамилия)

Нормоконтролер

(Подпись, дата)

Мальцева Д. Ю.
(И. О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 74 с., 30 рис., 4 табл., 34 источн., 1 прил.

В данной работе представлена разработка метода миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа. Проведен обзор существующих систем управления базами данных (СУБД). Рассмотрены подходы к миграции данных из реляционных в документо-ориентированные базы данных. Представлено сравнение методов миграции данных.

Представленный метод реализуется в виде программного обеспечения и проводится оценка его применимости и эффективности.

Ключевые слова: базы данных, миграция данных, СУБД, РСУБД, NoSQL, MongoDB, документо-ориентированные БД.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ВВЕДЕНИЕ	5
1 Аналитический раздел	6
1.1 Базы данных	6
1.2 Реляционные базы данных	7
1.2.1 Основные понятия	8
1.2.2 Язык запросов	10
1.2.3 ACID Свойства	11
1.3 Нереляционные базы данных	12
1.3.1 Нереляционные модели данных	14
1.3.2 Теорема Брюера (CAP)	16
1.3.3 Сравнение NoSQL баз данных	17
1.3.4 Сравнительный анализ реляционных и нереляционных баз данных	18
1.4 Методы миграции	20
1.4.1 Описание процесса миграции	20
1.4.2 Методы на основе агрегации	21
1.4.3 Методы на основе нормализации	25
1.4.4 Метод основанный на введении нереляционной схемы . .	28
1.4.5 Метод использующий оберточный слой для запросов . .	30
1.4.6 Методы на основе объектного преобразования	31
1.4.7 Методы на основе ETL	32
1.5 Сравнение методов миграции	33
1.6 Применение миграции данных	35
1.7 Постановка задачи миграции из реляционного в документно-ориентированное хранилище	36
2 Конструкторский раздел	38
2.1 Описание метода миграции данных	38
2.2 Описание алгоритма получения метаданных о реляционной схеме	39
2.3 Описание алгоритма частотного анализа JOIN-операций	42

2.4	Описание алгоритма преобразования схемы	45
2.5	Алгоритм преобразования данных в формат JSON	48
3	Технологический раздел	50
3.1	Выбор средств реализации	50
3.2	Структура разработанного ПО	51
3.2.1	Доступ к данным	52
3.2.2	Преобразование схемы	55
3.2.3	Миграция данных	57
3.3	Пользовательский интерфейс	59
4	Исследовательский раздел	65
4.1	Исследование зависимости объема памяти от глубины вложенности формируемого документа	65
4.2	Исследование зависимости времени конвертации схемы от глубины вложенности	67
	ЗАКЛЮЧЕНИЕ	69
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	73
	ПРИЛОЖЕНИЕ А	74

ВВЕДЕНИЕ

Крупные компании в наше время имеют высокую потребность в сборе и анализе информации. В частности, множество компаний имеют большое число клиентов и как следствие, с ростом числа клиентов растет объем транзакционных данных. Появляется потребность в том, чтобы проводить быструю аналитику для формирования, например, маркетинговых кампаний, однако зачастую они упираются в производительность СУБД.

Наиболее популярным решением для хранения данных выступают реляционные базы данных. Они заняли большую часть рынка и являются наиболее оптимальным решением при малом или среднем объеме данных [1]. Однако, при росте объемов данных, достоинства РСУБД начинают их ограничивать. И с повсеместным распространением облачных и распределенных технологий, появляется потребность в выборе альтернативного решения для хранения данных. Как следствие, возникает потребность в переносе данных в новое хранилище.

Цель работы – разработать метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа.

Для достижения поставленной цели необходимо решить следующие задачи:

- рассмотреть подходы к миграции данных из реляционных в документо-ориентированные базы данных и провести сравнение методов миграции данных;
- разработать метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа;
- спроектировать и реализовать ПО, демонстрирующее работу метода;
- провести исследование эффективности разработанного метода по времени и по памяти.

1 Аналитический раздел

В данном разделе производится анализ предметной области, дается краткая характеристика различных типов баз данных, приводятся основные понятия и излагаются области применения миграции баз данных. Приводится описание и сравнение методов миграции данных из реляционной в документо-ориентированную базу данных. Производится постановка задачи в виде IDEF0-диаграммы.

1.1 Базы данных

База данных (БД) – совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, которая поддерживает одну или более областей применения.

Система управления базами данных (СУБД) – комплекс программных средств, который предоставляет возможность обращаться к базе данных для некоторой группы пользователей.

Задачи СУБД:

- управление данными во внешней памяти;
- управление транзакциями;
- журналирование;
- поддержка языка запросов;

СУБД можно классифицировать по используемой для организации хранения модели данных. Среди разных моделей данных существуют такие как: иерархическая, сетевая, объектно-ориентированная, реляционная, ассоциативная и другие.

Реляционная модель данных, является наиболее широко используемой. На таблице 1.1 представлен рейтинг СУБД по популярности на момент марта 2023 года с указанием используемой модели данных [1]. Метод подсчёта рейтинга указан в источнике [2].

Таблица 1.1 – Рейтинг популярности СУБД

Позиция	СУБД	Модель данных	Рейтинг
1	Oracle	Реляционная	1232.64
2	MySQL	Реляционная	1172.46
3	MsSQL	Реляционная	920.09
4	PostgreSQL	Реляционная	617.90
5	MongoDB	Документо-ориентированная	436.61
6	Redis	Ключ-значение	168.13

1.2 Реляционные базы данных

Идея реляционной модели была предложена Э. Ф. Коддом в 1970 году [3]. В реляционной модели данные хранятся в виде таблиц связанных отношениями. Отношения могут описаны логически в форме «один-к-одному» «один-к-многим» «многие-к-многим». Визуально представить такую модель можно с помощью нотации Чена, предложенной для описания моделей «сущность-связь» в 1976 году [4].

Данные в реляционной модели должны удовлетворять следующим свойствам:

1. каждая строка таблицы отображает кортеж или запись;
2. все значения атрибутов атомарны;
3. все строки уникальны;
4. порядок строк не значим;
5. порядок колонок не значим.

1.2.1 Основные понятия

Рассмотрим основные понятия реляционной модели данных.

Ключ

Понятие ключа является одним из основных для РСУБД. Ключ позволяет идентифицировать запись и облегчает создание отношения между сущностями. Засчет ключей достигается важное свойство реляционной модели – уникальность записей.

Есть три основных типа ключей:

- **Потенциальный ключ** используется для того, чтобы определить сущность без обращения к другим данным таблицы. Представляет собой минимальный набор атрибутов для идентификации сущности. Потенциальных ключей может быть несколько.
- **Первичный ключ** используется для идентификации записи в таблице. Первичный ключ может быть только один и выбирается из множества потенциальных ключей. Засчет первичного ключа обеспечивается уникальность и отсутствие избыточности данных. На рисунке 1.1 поля «ID» сущностей «человек» и «собака» являются первичными ключами.
- **Внешний ключ** ссылается на соответствующий первичный ключ другой сущности для того, чтобы однозначно определить эту другую сущность. На рисунке 1.1 поле «хозяин» является внешним ключом сущности «собака» и ссылается на сущность «человек» определяя связь «один-к-многим» между человеком и собакой.

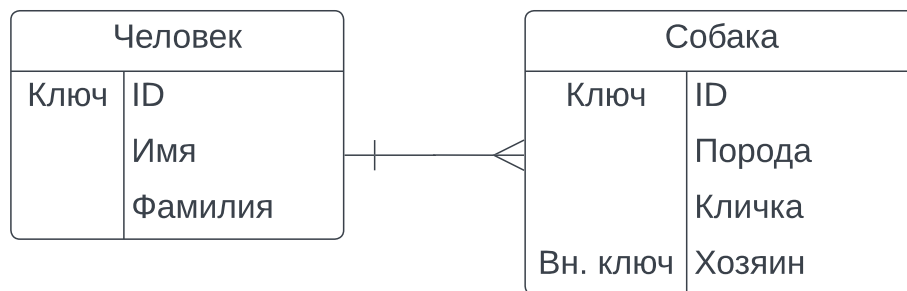


Рисунок 1.1 – Отношение с использованием первичного и внешнего ключа

Нормализация данных

Нормализация – это набор правил, согласно которым необходимо проектировать сущности или таблицы в реляционной модели для определения отношений через первичные и внешние ключи. Эти правила нужны для решения проблемы сложных доменов описанной Э. Ф. Коддом [3]. Нормализация представляет собой процесс разбиения таблиц на подтаблицы облегчая дальнейшее взаимодействие с данными в базе.

Можно выделить следующие преимущества нормализации:

- устранение аномалий при обновлении;
- оптимизация запросов;
- обеспечение целостности данных;
- гибкость и производительность при агрегации, сортировках.

В 1979 Кодд ввел три типа нормализации, которые были названы нормальными формами [5]:

- Первая нормальная форма (1NF) – согласно первой нормальной форме не должно присутствовать повторяющихся групп. Если различные колонки таблицы содержат одинаковый тип информации, тогда таблица не соответствует 1NF. Данные должны быть разбиты на минимально допустимые фрагменты. Согласно Кодду все данные в таблице должны быть неделимы.

- Вторая нормальная форма (2NF) – вторая нормальная форма наследует требования 1NF. В 2NF все неключевые атрибуты должны быть функционально зависимыми от первичного ключа.
- Третья нормальная форма (3NF) – третья нормальная форма удовлетворяет всем требованиям 1NF и 2NF. В 3NF необходимо, чтобы все неключевые атрибуты должны зависеть от первичного ключа и не зависеть от неключевых атрибутов.

1.2.2 Язык запросов

Реляционные базы данных используют сильно-структурированный язык для доступа к базе и последующего получения данных. Этот язык получил название «Structured Query Language» (SQL). Данный язык был разработан Дональдом Чемберлейном и Реймондом Бойсом в 70е для получения доступа к реляционным базам данных [6].

SQL быстро заработал популярность и в дальнейшем большинство разработчиков РСУБД внедрили его в свои продукты. Основной причиной популярности языка стало то, что он значительно облегчал разработчикам взаимодействие с базами данных, что в свою очередь сокращало время на разработку и стоимость программного продукта в целом. В следствие популярности Американский Национальный Институт Стандартизации (ANSI) разработал стандарт SQL – ANSI SQL. Несмотря на стандарт языка, множество СУБД используют собственные модификации языка SQL.

1.2.3 ACID Свойства

Важный набор понятий, гарантирующий безопасность и надежность транзакции известен как ACID свойства [7]. Логическая единица работы которая производится внутри последовательности операций называется транзакцией. Для того, чтобы обеспечить безопасную, надёжную и согласованную транзакцию, логическая единица работы должны удовлетворять четырем свойствам: неразрывность, правильность, изоляция и устойчивость. Аббревиатура ACID соответственно сформирована из первых букв слов Atomicity, Consistency, Isolation, Durability.

- Неразрывность (англ. Atomicity) – “Транзакции неразрывны (Всё или ничего)”. В транзакции либо выполняются все необходимые операции, либо изменения не производятся.
- Правильность (англ. Consistency) – “Транзакции переводят базу из одного правильного состояния в другое”. При этом правильность не обязана обеспечиваться на всех этапах транзакции.
- Изолированность (англ. Isolation) – “Транзакции изолированы друг от друга”. Промежуточное состояние транзакции не может быть получено любой другой транзакцией. Изоляция необходима для того чтобы поддерживать согласованность между транзакциями. Изменения производимые параллельным выполнением транзакций также должны быть изолированы друг от друга.
- Устойчивость (англ. Durability) – После завершения успешной транзакции изменения должны быть зафиксированы в системе даже в случае выхода системы из строя.

1.3 Нереляционные базы данных

Нереляционные базы данных принято называть NoSQL от «Not Only SQL», так как они разрабатывались для распределенного хранения данных и тем самым они выходят за рамки ограничений накладываемых реляционной моделью данных, для которой был разработан язык SQL.

NoSQL базы данных проектируются с учетом потребностей в распределенных и высоконагруженных облачных вычислениях на больших объемах данных. Также зачастую к ним предъявляется требование иметь гибкую схему для эффективного хранения неструктурированных данных [8].

Можно выделить такие ключевые особенности NoSQL баз данных:

1. Возможность горизонтального масштабирования.
2. Возможность партиционирования и распределения по множеству серверов.
3. Сравнительно ненадёжный параллелизм в сравнении с ACID.
4. Схожий с SQL синтаксис. Простой язык запросов.
5. Возможность динамически менять количество атрибутов.
6. Эффективное использование ОЗУ и распределенных индексов для хранения данных.

Горизонтальное масштабирование, репликация и распределение данных по нескольким серверам дают возможность более быстро осуществлять операции чтения и записи данных. Однако, NoSQL не удовлетворяет свойствам ACID, которые обеспечивают правильность параллельных транзакций. Веб-приложения в основном выполняются в распределенной среде, где основным требованием является возможность к масштабированию и согласно «CAP-теореме» для распределенной системы не предоставляется возможным обеспечить одновременно правильность, доступность и устойчивость к разделению. Одновременно выполняться могут только лишь два свойства из трех.

Менее строгая модель BASE (Basically Available, Soft state, Eventual consistency) заменяет ACID в рамках NoSQL [9].

- Базовая доступность (англ. Base availability) – На любой запрос будет получен ответ о успешном или неуспешном завершении.
- Неустойчивое состояние (англ. Soft state) – Состояние системы неустойчиво – может изменяться со временем. Причем изменения могут происходить без внешнего взаимодействия.
- Согласованность в конечном счёте (англ. Eventual consistency) – В конечном счете база данных будет правильной и согласованной, несмотря на то, что она может быть несогласованна в момент времени.

Структуры данных в нереляционных базах данных отличаются в сравнении с используемыми в реляционной модели. Далее будут описаны некоторые типы моделей данных используемые в NoSQL базах данных.

1.3.1 Нереляционные модели данных

В данном разделе будут рассмотрены разные типы нереляционных моделей данных. Согласно подходу к хранению данных нереляционные базы данных можно классифицировать по перечисленным моделям:

- Столбцовая (Колоночная) модель – Несмотря на то, что и колоночная и реляционная база основываются на понятии строк и колонок, для колоночного хранилища нет необходимости определять колонки. В сравнении с РСУБД, где данные хранятся в виде строк в таблице, колоночное хранилище предполагает хранение данных как набор колонок. Колонка может включать себя множество атрибутов, поэтому её могут называть «Широкой». Это позволяет оптимизировать выполнение сложных запросов. Основные причины использования колоночного хранилища:
 - в качестве распределенного хранилища;
 - для пакетной обработки больших объемов данных, предполагающих операции агрегации, сортировки либо другой обработки данных;
 - для создания аналитических хранилищ.

Множество популярных NoSQL СУБД используют колоночную модель данных, например: Apache Cassandra, HBase, Bigtable, DynamoDB.

- Документо-ориентированная модель – хранилище спроектированное для хранения данных в виде документов, которое позволяют записывать, получать и изменять слабо-структурированные данные. Для того, чтобы облегчить работу разработчика документы в документо-ориентированном хранилище не зависят от конкретной схемы. В документо-ориентированной базе данных, все данные представляются в виде документа в определенном формате. Документы могут быть в таких форматах как: Некоторые форматы документов: XML, JSON, BSON, YAML и другие. На рисунке 1.1 представлен документ в формате JSON. Также следует отметить, что документ может содержать другой документ внутри себя.

Листинг 1.1 – Пример структуры данных в документо-ориентированной БД

```
{
  "_id": {
    "$oid": "5553a998e4b..."
  },
  "st": "x+51900+003200",
  "ts": {
    "$date": {"$numberlong" : "4473540..."}
  },
  "position": {
    "type": "Point",
    "coordinates": [
      {"$numberDouble": "3.2"},
      {"$numberDouble": "52.7"}
    ]
  },
  // ...
}
```

В документ-ориентированных СУБД каждый документ обязан иметь уникальный идентификатор за его генерацию зачастую отвечает СУБД. В основном документ-ориентированная модель используется для создания web-приложений, имеющих потребность в обработке крупных объемов распределенных по сети текстовых данных. В качестве примеров популярных документо-ориентированных СУБД можно привести: MongoDB, CouchDB, Terrastore.

- Модель Ключ-Значение – в модели ключ-значение отсутствует схема и все данные хранятся в единой хэш-таблице. Идентификаторы или ключи символьные и могут быть как сгенерированными системой, так и заданными разработчиком вручную, аналогично идентификатору документа в документо-ориентированной базе данных. Хранилища типа ключ-значение в основном используются в виде ‘in-memory’ распределенного кэша, для организации быстрого доступа к данным. Среди наиболее популярных СУБД основанных на модели ключ-значение есть такие как Memcached, MemcacheDB и Redis.
- Графовая модель – Графовые базы данных хранят данные в виде графо-

вых структур, состоящих из узлов, рёбер и свойств. Узлы отображают концептуальные объекты, соединенные линиями, которые называют ребрами. Ребра используются для создания связей между узлами и свойствами. Как в реляционной модели, графовые базы данных работают с отношениями путем прохождения по ребрам. Используя графовые алгоритмы графовые базы данных могут хранить данные в таком виде, который можно масштабировать по нескольким серверам для дальнейших связей ребрами. Узлы и отношения – базовые понятия графовой модели, в которой узлы организуются по свойствам ассоциированным с отношениями, где данные хранимые в узлах также имеют собственные свойства.

Графовые базы данных обычно используются в случаях, когда отношения между данными более важны. Например, многие социальные сети используют графовые базы данных. Среди известных графовых СУБД можно выделить Neo4J, FlockDB, AllegroGraph.

1.3.2 Теорема Брюера (CAP)

В 2000г. Эриком Брюером была выдвинута CAP-теорема [10; 11]. Основная ее идея звучит как: “В любой реализации параллельных вычислений нельзя добиться одновременно не более двух свойств, таких как согласованность данных, доступность и устойчивость к разделению”. Каждой системе необходимо соответствовать следующим свойствам [12]:

- согласованность данных (англ. Consistency) – данные в каждом узле не должны противоречить друг другу и имеют общее согласованное состояние;
- доступность (англ. Availability) – любой запрос к распределенной системе завершается корректно;
- устойчивость к разделению (англ. Partition tolerance) – расщепление распределенной системы на изолированные сегменты не приводит к некорректности работы системы. Таким образом при отказе одного из узлов гарантируется работа общей системы.

Однако удовлетворить всем трем свойствам не предоставляется возможным. Согласованность данных легко достигается в реляционной базе данных,

в силу соответствия ACID. В то же время, РСУБД проблематично горизонтально масштабировать, однако NoSQL базы данных просто масштабируются, но не могут предоставить тот же уровень согласованности данных, в силу соответствия BASE.

1.3.3 Сравнение NoSQL баз данных

На таблице 1.2 указаны результаты сравнения NoSQL СУБД разных моделей данных.

Таблица 1.2 – Сравнение наиболее популярных NoSQL СУБД.

Название	Модель данных	Выполняемые свойства CAP-теоремы	Модель целостности	Область применения
Redis	Ключ-значение	Согласованность Устойчивость к разделению	BASE	Кэширование данных сессий
MongoDB	Документно-ориентированная	Согласованность Устойчивость к разделению	BASE	Хранение больших коллекций данных
Cassandra	Колоночно-ориентированная	Доступность Устойчивость к разделению	BASE	Распределенное хранение и обработка больших данных
Neo4j	Графовая	Согласованность Доступность	ACID	Хранение и анализ социальных графов

Выбор нереляционной базы данных

Для более детального рассмотрения был выбран процесс миграции данных с РСУБД на документо-ориентированную СУБД. Выбор документо-ориентированной СУБД обусловлен тем, что документо-ориентированные СУБД являются вторыми по распространенности после реляционных [1]. Также стоит учесть, что принципы документо-ориентированной модели явно противоречат реляционной, так как оптимальным подходом является денормализация схемы, в противовес нормализации. Помимо этого открытыми являются проблемы избыточности данных после миграции и переноса отношений «многие-к-многим».

1.3.4 Сравнительный анализ реляционных и нереляционных баз данных

- Надежность транзакции: РСУБД соответствует ACID и предоставляет надежность транзакции. Нереляционные базы, опираются на ослабленную спецификацию — BASE.
- Модель данных: РСУБД основаны на реляционной модели, и данные представляются в виде таблиц с набором строк связанных отношением. Нереляционные базы данных используют различные модели, например: ключ-значение, документо-ориентированные, колоночные и графовые.
- Масштабируемость: Web-приложения требуют возможности горизонтального масштабирования, т. к. они распределены по нескольким серверам. NoSQL базы данных поддерживают горизонтальное масштабирование, в свою очередь горизонтальное масштабирование РСУБД является трудоемкой задачей.
- Обработка больших данных: Из-за проблем с масштабированием и партиционированием данных в распределенной кластеризованной среде, обработка больших объемов данных на РСУБД проблематична.
- Устойчивость к сбоям: Данные в РСУБД консистентны и поддерживаются в правильном состоянии за счет транзакций и журналирования. Сохранность данных в нереляционных базах зависит от правильности их репликации.
- Безопасность: В РСУБД внедряются сложные механизмы безопасности. Нереляционные базы спроектированы таким образом, чтобы добиться производительности в обработке больших объемов данных, за счет потерь в их безопасности. Безопасность данных является крупным объектом обсуждения в рамках активно развивающихся облачных систем.

В таблице 1.3 приведено сравнение реляционных и нереляционных СУБД.

Таблица 1.3 – Сравнение РСУБД и NoSQL СУБД

Критерий \ Вид	РСУБД	Нереляционные СУБД
Надежность транзакции	Да (ACID)	Нет (BASE)
Модель данных	Реляционная	Множество нереляционных
Горизонтальное масштабирование	Нет	Да
Обработка больших данных	Не предназначались, имеют ограничения	Учитывают потребности в обработке больших данных
Надежность	Да	Зависит от репликации
Безопасность	Да	Нет

1.4 Методы миграции

Миграция – процесс переноса данных из одной или более текущих баз данных в одну или более новых баз данных с возможным изменением структуры данных.

Промышленные применения реляционной модели данных не поддерживают растущие требования по производительности в сравнении с нереляционными в рамках анализа больших объемов данных [13]. Проведя сравнительный анализ реляционных и нереляционных СУБД, можно сделать вывод, что основное различие состоит в структуре хранения данных.

Существуют различные подходы к миграции данных. Они предлагают различную методологию по переносу данных и схемы из реляционного хранилища в документо-ориентированное. Основной задачей, при переносе схемы и данных является представление реляционных отношений. Методы могут быть основаны как на анализе и преобразовании схемы данных, так и не зависеть от схемы и манипулировать только данными.

1.4.1 Описание процесса миграции

С учетом структуры и деталей хранения, нереляционные СУБД отличаются от реляционных. Реляционная модель строго структурирована и данные нормализованы в таблицах согласно их отношениям, в нереляционных базах данные хранятся в полуструктурированном или неструктурированном виде и обычно денормализованы. Таким образом процесс миграции не является тривиальной задачей.

На рис 1.2 представлена общая схема процесса миграции.

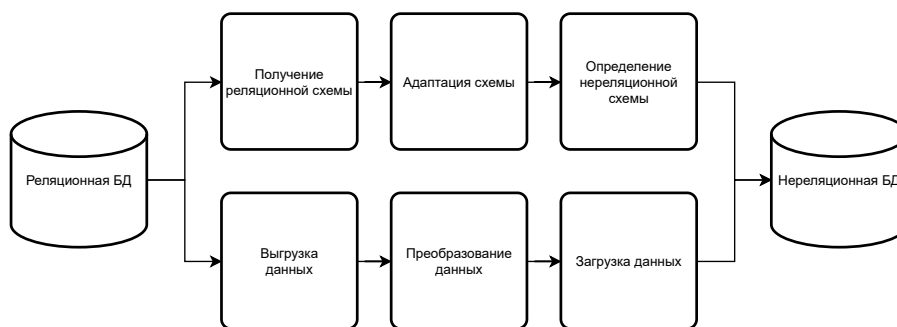


Рисунок 1.2 – Схема процесса миграции

Далее будут рассмотрены несколько методов миграции.

1.4.2 Методы на основе агрегации

Одни из наиболее распространенных подходов при миграции в документо-ориентированное хранилище из РСУБД основаны на агрегации. В основе метода лежит идея о денормализации структуры, путем агрегирования всех зависимых сущностей в единую главную сущность. Таким образом из множества таблиц и отношений между ними образуется единый документ, где все зависимые документы являются вложенными внутрь основного документа. Подобные подходы позволяют получать тесно связанные данные одним запросом к одной большой сущности, таким образом получается избежать JOIN-операций, не поддерживаемых в множестве нереляционных СУБД, однако миграция сложных схем подобным образом приводит к большой избыточности данных и их повторению. Также стоит отметить, что в случае повторения данных необходимо самостоятельно следить за их согласованностью.

В статье [14] представлен полу-автоматизированный метод преобразования реляционной базы данных в документо-ориентированную на основе табличной денормализации. Пользователю предлагается выбрать необходимые для переноса главную и зависимые сущности. Далее все зависимые сущности переносятся в виде вложенных в главный документов. Из недостатков этого метода можно выделить то, что он не позволяет сразу перенести всю реляционную структуру и требует вмешательства пользователя. Также при излишней денормализации растет объем документа и как следствие выполнение сложных запросов с несколькими коллекциями становится менее эффективным. Помимо этого такой подход неприменим к сложным реляционным структурам, так как конверсия в единый денормализованный документ не всегда возможна.

На рисунке 1.3 представлена общая схема алгоритма на основе табличной денормализации.

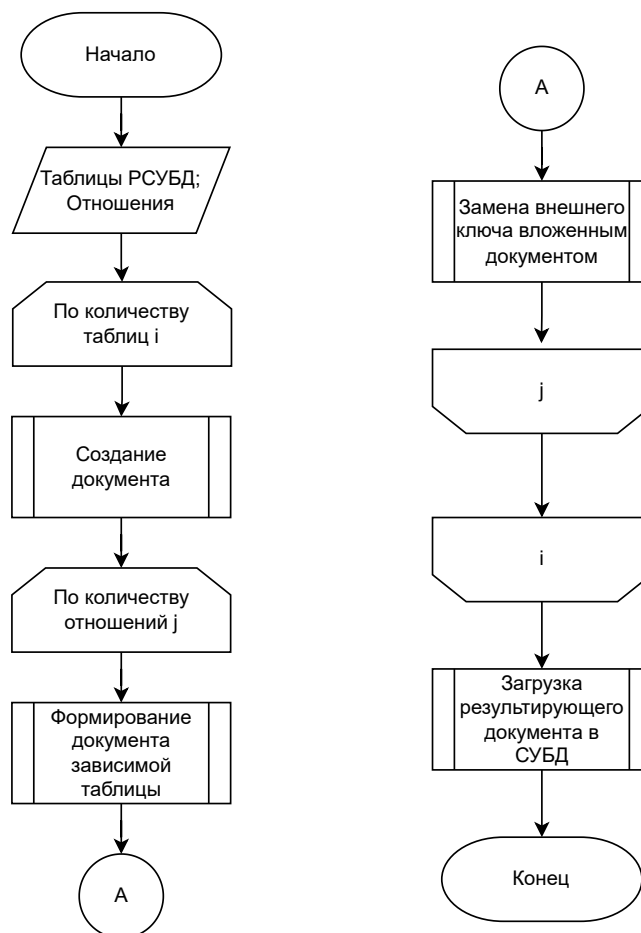


Рисунок 1.3 – Алгоритм миграции на основе табличной денормализации

Также существуют полностью автоматизированные методы денормализации. В статье [15] представлен метод миграции схемы основанный на анализе графа отношений. Предлагается рассматривать реляционную схему как направленный ациклический граф. Далее на основе понятия полноты данных вводятся операции расширения узла. Так как изначально каждый узел представляет собой сущность или таблицу в результате работы алгоритма граф преобразуется таким образом, что в каждом узле хранится вся зависящая от него информация.

На рисунке 1.4 приведена схема алгоритма для поиска цепочки всех узлов необходимых для дополнения сущности. В качестве входных данных принимаются значения:

$$\begin{aligned} O[v] &\leftarrow |b(v)|, \\ P &\leftarrow \{v | v \in V \wedge O[v] = 0\}, \\ Q &\leftarrow V - P, \\ T &\leftarrow \emptyset, \\ S &\leftarrow \{\}, \end{aligned}$$

где:

- $O[v]$ – количество входящих граней;
- Q – узлы у которых $O[v] = 0$;
- P – узлы имеющие входящие грани и не имеющие исходящих;
- T – узлы не имеющие ни входящих ни исходящих граней;
- S – путь по графу, составляющий полную сущность;
- $f(x)$ – узлы, имеющие прямой путь в x .

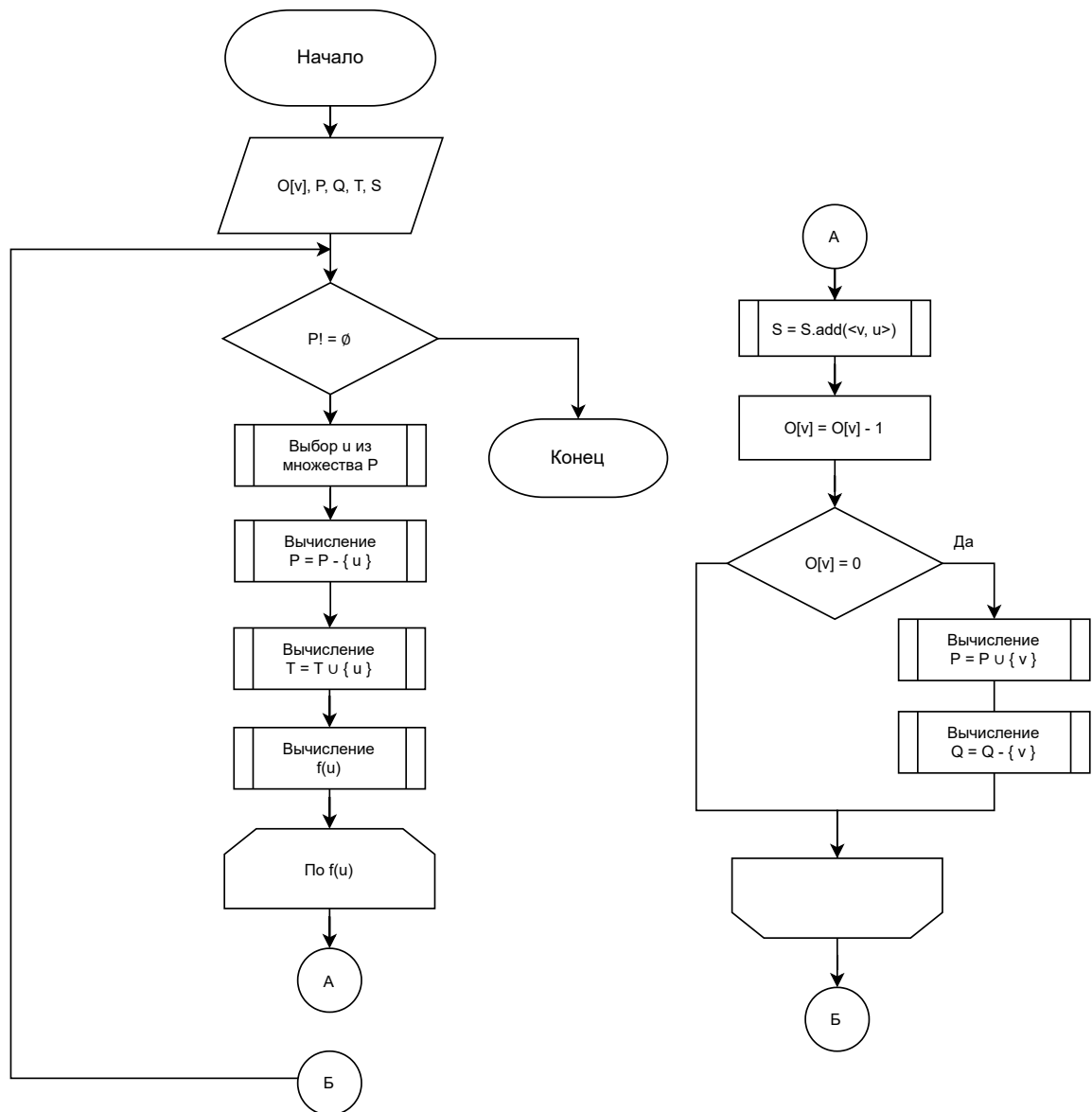


Рисунок 1.4 – Алгоритм миграции на основе расширения узлов направленного ациклического графа

В итоге получаем несколько денормализованных коллекций документов, с которыми можно работать не прибегая к JOIN-операциям. Однако такой подход ведет к высокой избыточности, так как все зависимые данные будут дублироваться. Также к недостаткам метода можно отнести то, что не все графы отношений можно выразить с помощью направленного ациклического графа. Таким образом подобный метод не подходит для переноса схем имеющих отношения «многие-к-многим».

1.4.3 Методы на основе нормализации

Несмотря на то, что документо-ориентированная схема предполагает денормализацию данных, существуют методы напрямую переносящие нормализованную реляционную схему в документо-ориентированную модель. В таком случае отношения реализуются не засчет вложенности документов, а с помощью внешних ключей. Внешние ключи определяются так же как и в РСУБД – через указание первичного ключа (идентификатора) другой сущности (документа).

В статье [16] описывается каким образом нормализованные данные из РСУБД могут быть представлены в документо-ориентированной СУБД без нарушения нормальной формы. На основе этого может быть построен алгоритм согласно которому все данные из РСУБД переносятся с сохранением всех своих внешних и первичных ключей.

Однако, в силу того, что документо-ориентированные СУБД не подразумевают нормализованное представление данных, сохранение согласованности данных и реализация JOIN-операций отводится разработчику. Это значительно затрудняет операции вставки, изменения и удаления, так как СУБД не берет на себя отслеживание изменений в зависимых сущностях.

На рисунке 1.5 представлена схема алгоритма переноса нормализованной схемы.

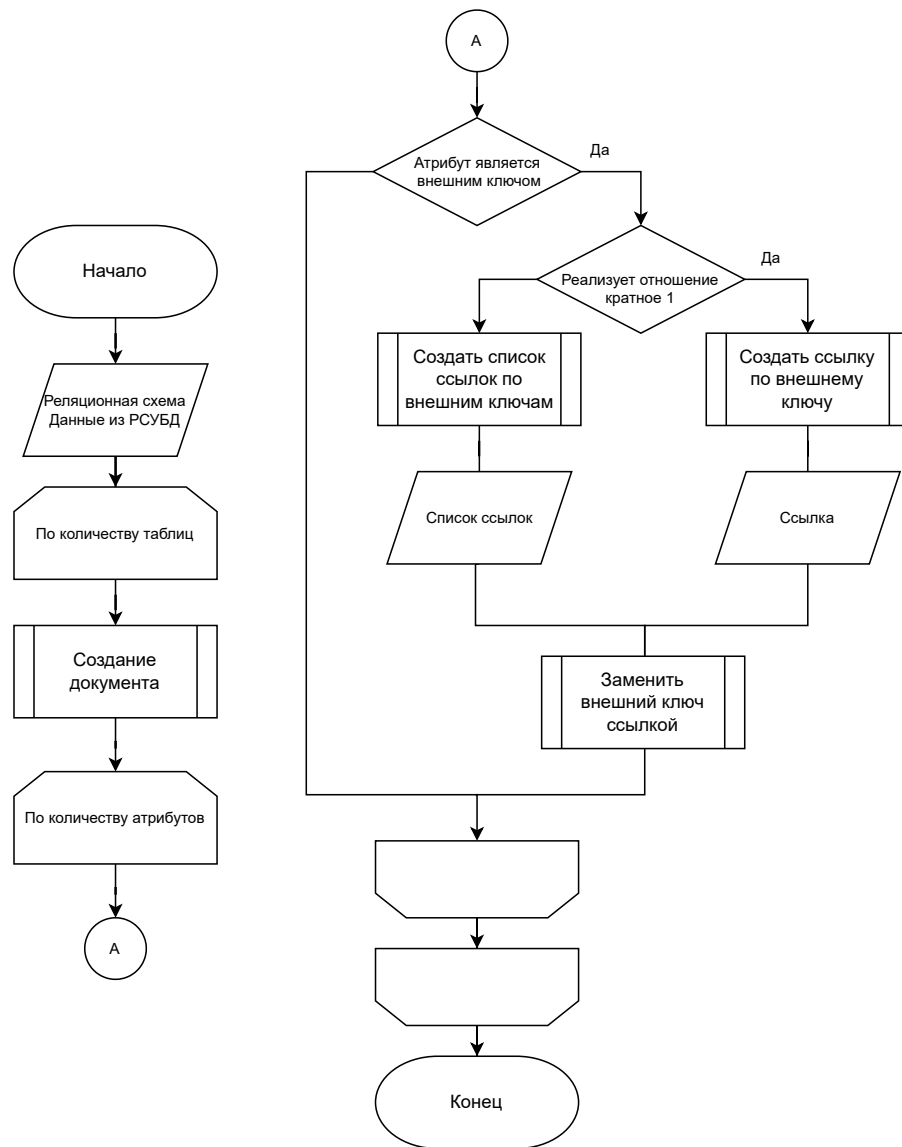


Рисунок 1.5 – Алгоритм переноса нормализованной схемы с использованием ссылок

В статье [17] предлагается подход на основе перевода таблицы из РСУБД в формат CSV (Comma Separated Value) и далее загрузке данного файла в документо-ориентированную СУБД. Данный метод очень сильно зависит от того, каким образом документо-ориентированная СУБД будет воспринимать полученный формат файла. Зачастую при использовании подобного подхода могут не учитываться отношения реляционной модели и тем самым мы теряем возможность обращения к связанным между собой данным. Поэтому данный подход предпочтителен, когда необходимо перенести конкретную таблицу из реляционной в документо-ориентированную базу данных. На рисунке 1.6 представлена общая схема метода миграции через CSV файл.

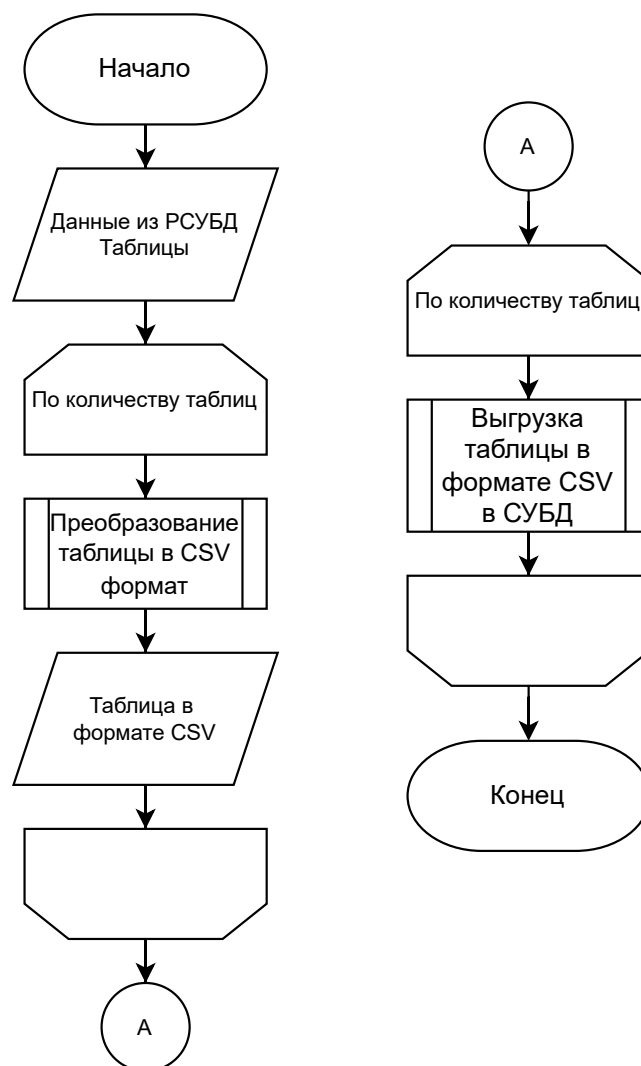


Рисунок 1.6 – Алгоритм миграции на основе перевода в CSV файл

1.4.4 Метод основанный на введении нереляционной схемы

Методы вводящие понятие собственной нереляционной схемы частично объединяют описанные выше подходы. Зачастую предлагается введение методологии или набора правил, согласно которым реляционная схема приводится к виду совместимому с документо-ориентированной моделью. В статье [18] описывается один из таких методов. Предлагается набор правил, согласно которому на основе вида сущности и отношения принимается решение о том, каким образом реализуется данное отношение в документо-ориентированной модели. Отношение может быть реализовано как с помощью ссылки, так и зачет вложенности документов.

Метод предлагает подход для миграции данных в таком формате, который является компромиссным между нормализованным и денормализованным. Также данный метод предлагает решение для переноса связей «многие-к-многим» хоть и не решает проблем с отслеживанием согласованности данных в рамках этого отношения. Засчет этого он может использоваться для переноса сложных реляционных структур, в то время как методы на основе агрегации зачастую не применимы к структурам, которые нельзя представить в виде направленного ациклического графа.

Таким образом предлагается оптимальное решение не приводящее к излишней избыточности, при этом более оптимальное с точки зрения хранения данных и выполнения запросов нежели перенос нормализованной структуры с сохранением нормальной формы.

На рисунке 1.7 представлена общая схема алгоритма преобразования реляционной схемы в документо-ориентированную согласно методологии описанной в [18];

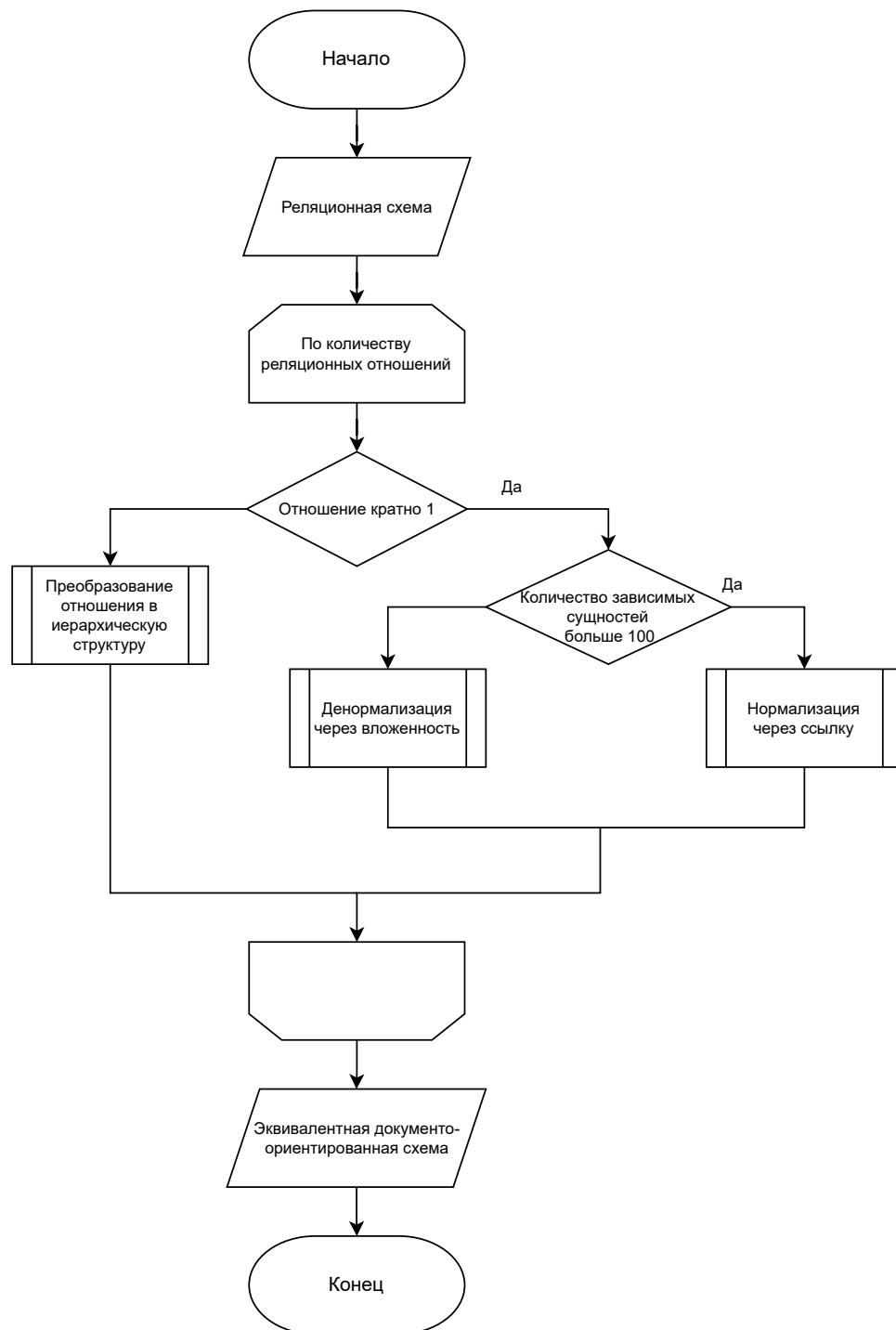


Рисунок 1.7 – Алгоритм миграции на основе нереляционной схемы DODS

1.4.5 Метод использующий оберточный слой для запросов

Методы использующие оберточный слой для запросов позволяют проводить процесс миграции без анализа схемы исходной реляционной базы данных. Сравнение конкретных систем использующих этот метод представлено в статье [19]. Подход с оберточным слоем предполагает создание слоя преобразующего запросы на языке SQL в запросы совместимые с желаемой нереляционной СУБД. Таким образом данные миграция данных осуществляется за счет средств языка SQL, который поддерживается большинством реляционных СУБД. Такой подход позволяет объединить преимущества хранения в документо-ориентированной модели и возможности языка запросов SQL, однако результирующая структура данных может быть не оптимальной в рамках нереляционной модели данных.

На рисунке 1.8 представлена общая схема метода использующего оберточный слой для запросов.



Рисунок 1.8 – Метод миграции использующий оберточный слой для запросов

1.4.6 Методы на основе объектного преобразования

Существуют методы на основе объектных преобразований, они используют объектно-реляционные (ORM) и объектно-нереляционные (ONM) преобразования. Использование таких систем позволяет ввести промежуточное представление сущности в виде объекта, что помогает создать единый интерфейс взаимодействия с базами данных независимо от модели, а также облегчает разработчикам взаимодействие с данными в рамках объектно-ориентированной парадигмы. В работе [20] рассмотрено несколько подобных систем.

Например, Hibernate широко используется для работы и с реляционными и нереляционными базами данных и предлагает собственные языки запросов HQL и JPQL, что позволяет единообразно обращаться к многим видам баз данных [21]. Такие системы получили популярность так как облегчают наиболее распространенные операции получения и изменения данных, однако они не являются оптимальными для работы в больших приложениях, где требуется множество сложных аналитических запросов к данным. Аналогичный подход с использованием системы LinQ рассмотрен в статье [22].

На рисунке 1.9 представлена общая схема метода миграции на основе объектного преобразования.

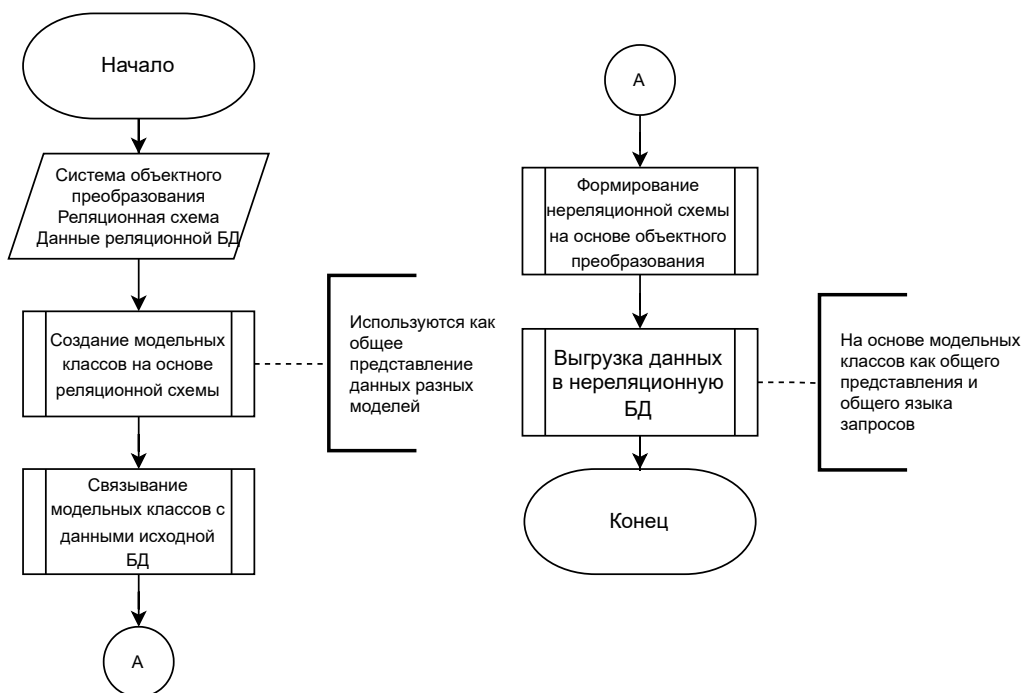


Рисунок 1.9 – Алгоритм миграции на основе объектного преобразования

1.4.7 Методы на основе ETL

ETL (Extract Transform Load) – подход к обработке данных, предполагающий последовательное получение данных, преобразование и выгрузку. На основе этого подхода основаны некоторые методы миграции данных. В статье [23] предложен метод миграции данных из реляционной БД в документо-ориентированную основанный на ETL подходе. Сначала путём подключения к реляционной БД получается структура таблиц. Далее пользователю предоставляется возможность выбрать данные необходимые для переноса в новую БД. После выбора необходимых таблиц и атрибутов, происходит выгрузка данных и преобразование их в формат совместимый с документо-ориентированной моделью, затем данные загружаются в новую базу данных. Метод нельзя назвать полностью автоматизированным, так как этап определения новой схемы требует вмешательства пользователя, однако этот метод автоматизирует этап переноса данных в выбранную пользователем схему.

Существуют решения с открытым исходным кодом использующим данный подход, например: MongoSyphon [24].

На рисунке 1.10 представлена общая схема ETL подхода.

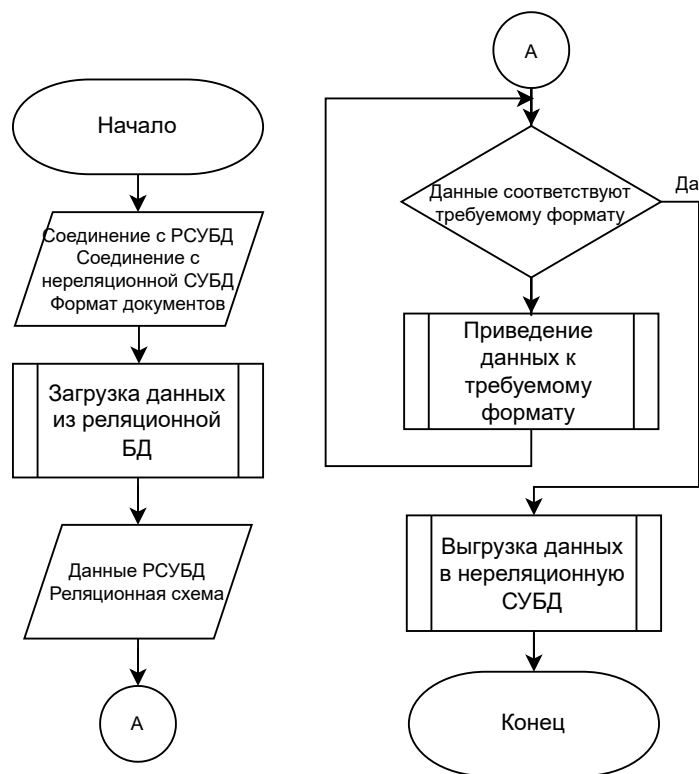


Рисунок 1.10 – Алгоритм миграции на основе ETL

1.5 Сравнение методов миграции

В ходе исследования были установлены следующие критерии сравнения методов миграции:

- применимость – применимость метода включает в себя два критерия: применимость к схемам и применимость к сущностям.

Применимость к схемам показывает, к каким схемам наиболее применим данный метод. Схема называется сложной если она включает в себя более 10 сущностей.

Применимость к сущностям показывает, к миграции сущностей какого размера применим данный метод. Сущность называется большой если она насчитывает более 10 атрибутов и имеет более двух отношений.

- поддержка отношения «многие-к-многим» – данный критерий показывает, позволяет ли метод переносить отношения типа «многие-к-многим» из РСУБД в документо-ориентированную;
- избыточность данных – данный критерий определяет количество дублирующейся информации при миграции. Избыточность определяется как высокая, если все зависимые данные дублируются для каждой сущности или как низкая, если дублирование данных не присутствует. Средняя избыточность показывает, что данные дублируются, но в количестве меньшем количества зависимых сущностей;
- автономность – данный критерий показывает потребность вмешательства разработчика в процесс миграции.

В таблице 1.4 приведено сравнение методов миграции по сформулированным критериям.

Таблица 1.4 – Сравнение методов миграции

Критерий Название	Применимость		Избыточность данных	Поддержка отношения «многие-к-многим»	Автономность
	К схеме	К сущностям			
Метод на основе табличной денормализации	Простые	Малые	Максимальная	Нет	Нет
Графовый метод на основе агрегации	Сложные	Малые	Максимальная	Нет	Да
Метод на основе нормализации	Сложные	Большие	Минимальная	Да	Да
Метод на основе нереляционной схемы	Любые	Любые	Средняя	Да	Нет
Метод на основе обертки запросов к СУБД	Любые	Любые	Средняя (Зависит от реализации)	Да	Нет
Метод на основе объектного преобразования	Простые	Малые	Средняя	Да	Нет
Метод на основе ETL процесса	Любые	Большие	Средняя (Зависит от реализации)	Да	Нет

Вывод

Сравнение показало, что метод на основе введения нереляционной схемы является оптимальным, так как он имеет возможность полной автоматизации и предполагает возможность оптимизации количества дубликатов данных. Также методология предлагаемая данным методом имеет возможность быть модифицированной на основе дополнительных входных данных.

1.6 Применение миграции данных

Задача миграции данных из реляционного в документо-ориентированное хранилище возникает по разным причинам:

1. Масштабирование приложения — при росте Web-приложений возникает потребность в увеличении объема и производительности хранилища, зачастую это достигается увеличением количества серверов и вынесением приложения в распределенную среду. Таким образом возникает потребность в переносе данных в более приспособленное для таких условий хранилище.
2. Отказ от строгой структуры данных — иногда оказывается, что спроектированная в ходе разработки реляционная схема не подходит для решаемой задачи. В таком случае возникает потребность в переносе данных в хранилище с менее строгой схемой, например документо-ориентированное.
3. Аналитика — накопление больших объемов данных затрудняет работу с РСУБД. В таких случаях прибегают к созданию нереляционных хранилищ, на которых проще и эффективнее проводить аналитику.

1.7 Постановка задачи миграции из реляционного в документо-ориентированное хранилище

Задача миграции данных состоит в том, чтобы перенести данные из одной базы данных в другую наиболее оптимальным образом. В силу того, что реляционная и документо-ориентированная модели значительно отличаются такой процесс не является тривиальным. Возникают проблемы в переносе отношений и организации доступа к данным в новой модели. В ходе исследования было замечено, что большая часть методов не учитывает особенности взаимодействия с ранее хранимыми данными. Как следствие, в качестве метода решения задачи был выбран метод на основе введения нереляционной схемы. Однако, рассмотренные в ходе работы методы никак не учитывают статистику запросов к РСУБД. Таким образом предлагается оптимизация процесса миграции на основе статистики запросов. Например, в ходе миграции стоит учесть, что данные которые чаще запрашиваются совместно, стоит агрегировать.

На рисунке 1.11 представлена постановка задачи в форме IDEF0-диаграммы.



Рисунок 1.11 – IDEF0-диаграмма метода миграции

2 Конструкторский раздел

В данном разделе представлены результаты разработки метода миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа. Представлены используемые алгоритмы, описаны входные и выходные данные. Описан процесс преобразования схемы из реляционной в документо-ориентированную. Выполнено проектирование для программной реализации метода с использованием функциональной модели IDEF0 и схем алгоритмов.

2.1 Описание метода миграции данных

Входными данными являются:

1. Реляционная схема.
2. Статистика запросов к СУБД.
3. Выбранные таблицы и кратности отношений.
4. Данные реляционной БД.

В ходе работы метода происходит сбор метаданных о таблицах РСУБД, таких как названия таблиц, первичные и внешние ключи, столбцы и типы данных. Далее эти метаданные дополняются результатами частотного анализа JOIN-операций, засчет того, что связи между сущностями помечаются как «часто объединяемые». На основе сформированных метаданных, засчет семантического анализа происходит формирование JSON-схемы, как промежуточного представления данных в документо-ориентированной БД. Согласно сформированной схеме, производится перенос данных из реляционной в документо-ориентированную БД.

Выходными данными являются:

1. Данные в документо-ориентированной БД.

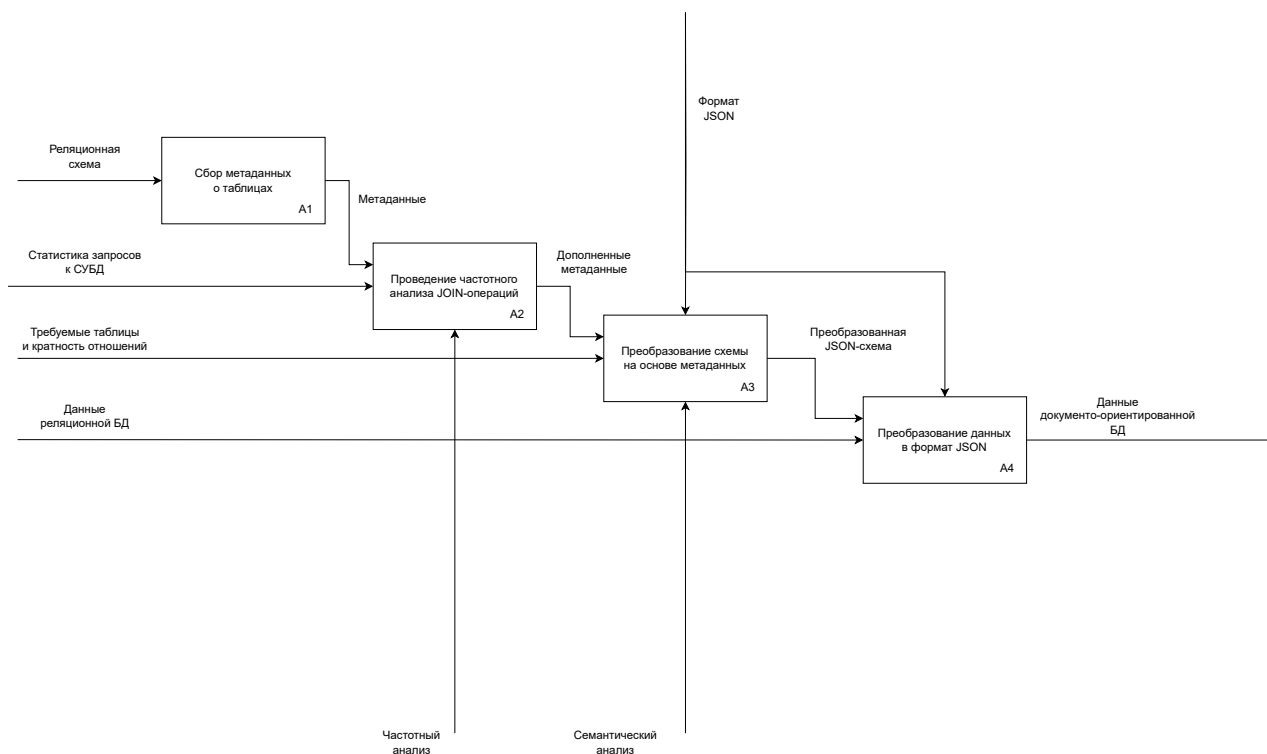


Рисунок 2.1 – IDEF0-диаграмма первого уровня

2.2 Описание алгоритма получения метаданных о реляционной схеме

Реляционные СУБД хранят информацию о созданных в них таблицах в отдельной схеме, которая называется `information_schema` и является частью стандарта ANSI [?]. Данные о таблицах, их столбцах и ключах хранятся в виде информационных таблиц и представлений, что позволяет поддержание стандарта SQL, тем самым позволяет использовать предлагаемый инструментарий в большинстве РСУБД. Многие СУБД также предоставляют собственные дополнительные инструменты для получения метаданных, например, системные каталоги, но они проприетарны для каждой системы управления базами данных и не портируемы.

В первую очередь необходимо собрать список интересующих нас таблиц. Далее, для каждой из этих таблиц нужно получить информацию о названиях столбцов, типах данных и ключах. Информация о ключах и связях между таблицами является наиболее важной, поэтому необходимо получить не только список первичных и внешних ключей в каждой таблице, но и внешние ключи, которые ссылаются на интересующие нас таблицы. Таким образом мы

получаем не только исходящие но и входящие связи.

В результате сбора метаданных мы получаем список структур, каждая из которых хранит метаданные для одной таблицы.

Структуры имеют вид:

- название таблицы;
- список столбцов;
- список первичных ключей;
- список внешних ключей;
- список исходящих связей таблицы;
- список входящих связей таблицы;

Собранные метаданные служат основой для семантического анализа схемы, который позволяет предварительно определить отношения между таблицами для дальнейшего выбора метода переноса связи из реляционной в документо-ориентированную модель.

Тем не менее, метаданные не содержат достаточно информации для организации процесса миграции, так как информация о кратности отношений и интересующих нас таблицах зависит от особенностей предметной области и не может быть определена без данных от пользователя.

На рисунке 2.2 представлен алгоритм сбора метаданных о таблицах базы данных.

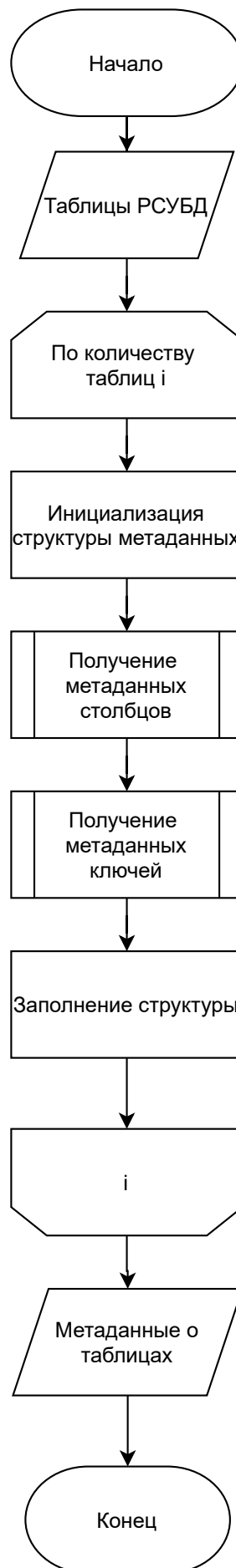


Рисунок 2.2 – Алгоритм сбора метаданных таблиц реляционной базы данных.

2.3 Описание алгоритма частотного анализа JOIN-операций

Большинство современных РСУБД предоставляют возможность логирования и сбора статистики о работе СУБД. Этот инструментарий может быть использован для анализа запросов к базе данных.

История запросов к СУБД обрабатывается, для определения частоты совершения JOIN-операций между двумя таблицами. Если таблица участвует в JOIN-операции значительно чаще чем в SELECT, можно предположить, что данные в зависимой таблице не востребованы отдельно от основной, поэтому она выдвигается как кандидат для пометки как «часто объединяемая».

На основе этих данных, «часто объединяемые» таблицы рекомендуются к предварительной агрегации. Таким образом в документо-ориентированной модели их отношение будет реализовано засчет вложенных документов.

На рисунке 2.3 представлен алгоритм дополнения метаданных результатами частотного анализа.



Рисунок 2.3 – Алгоритм дополнения метаданных результатами частотного анализа.

На рисунке 2.4 представлен алгоритм анализа истории запросов к базе данных.

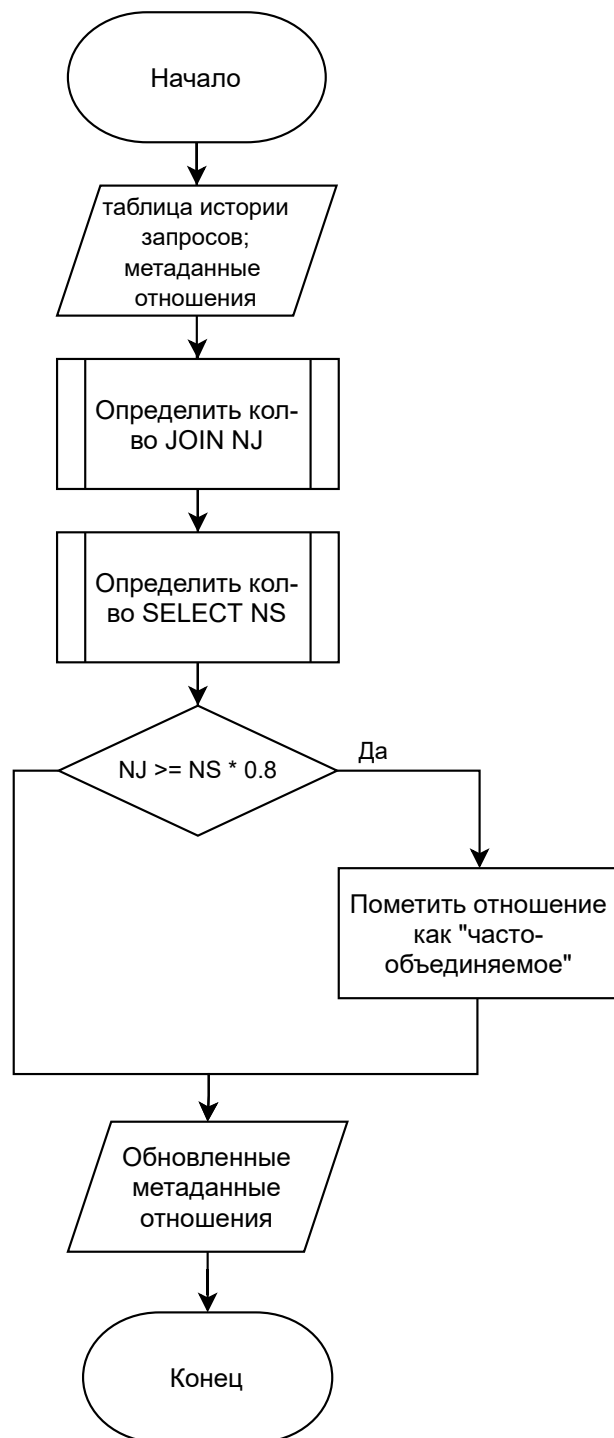


Рисунок 2.4 – Алгоритм анализа истории запросов.

2.4 Описание алгоритма преобразования схемы

После сбора и предобработки метаданных следует преобразовать реляционную схему в документо-ориентированную JSON-схему. Происходит выбор таблиц, на основе которых будут созданы коллекции документов в документо-ориентированной СУБД. Далее для каждой таблицы, на основе собранных метаданных с использованием информации о предметной области и семантического анализа формируется JSON-документ, который будет отражать вид данных в документо-ориентированной БД. Все неключевые столбцы таблицы преобразуются в JSON-атрибуты. Первичный ключ заменяется идентификатором объекта. После чего для каждого внешнего ключа предлагается набор действий в зависимости от имеющейся информации. Зависимые таблицы либо встраиваются в основной документ, либо создается ссылка на идентификатор из другой коллекции.

На рисунке 2.5 представлен алгоритм преобразования реляционной схемы в JSON-схему.



Рисунок 2.5 – Алгоритм преобразования схемы.

На рисунке 2.6 представлен алгоритм определения типа преобразования.

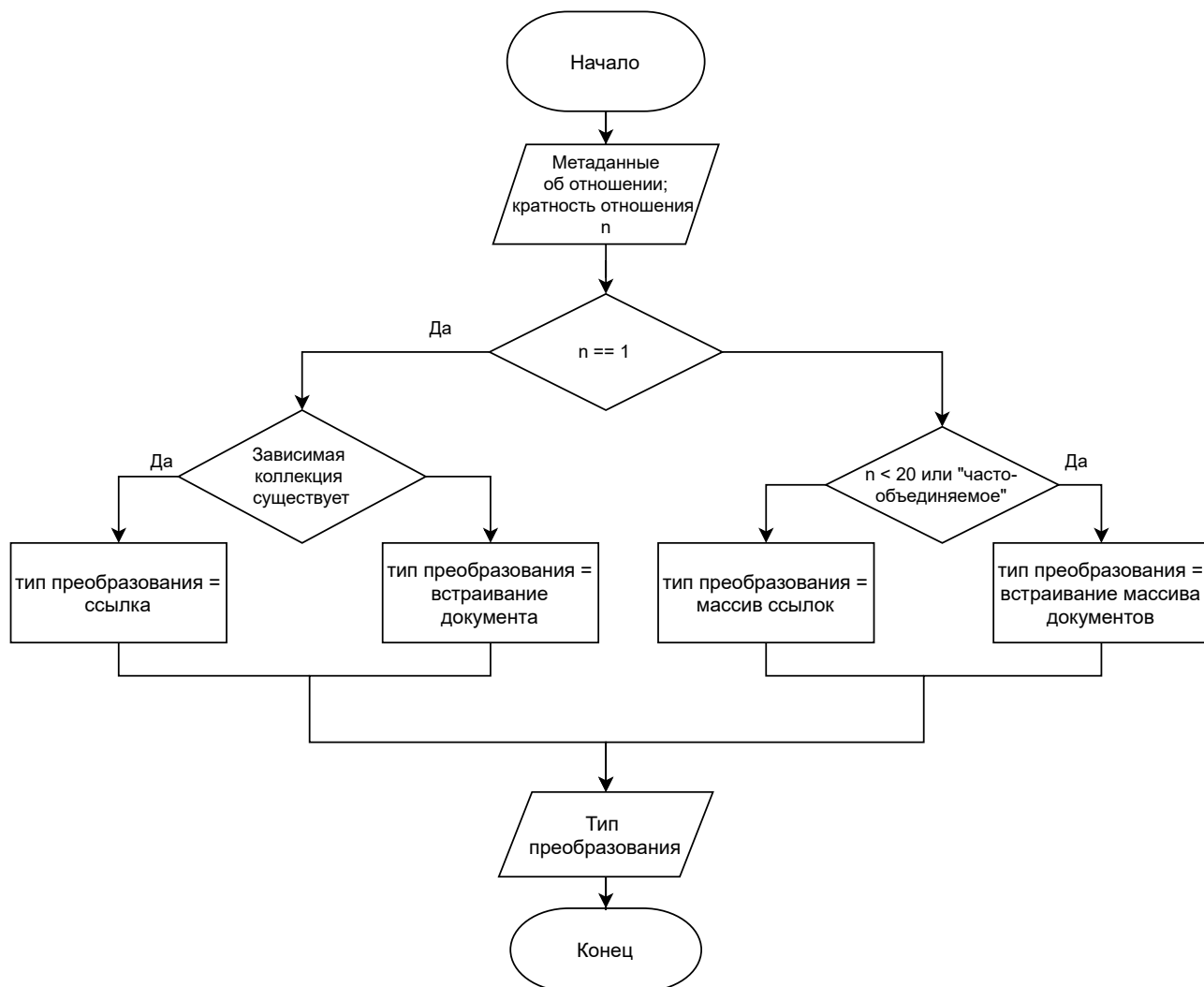


Рисунок 2.6 – Алгоритм определения типа преобразования.

2.5 Алгоритм преобразования данных в формат JSON

Сформированная схема используется для последующего преобразования данных. Согласно метаданным и сформированной схеме, данные реляционной БД выгружаются и каждая запись преобразуется в эквивалентный JSON-документ согласно схеме, после чего сформированный документ выгружается в целевую документо-ориентированную БД.

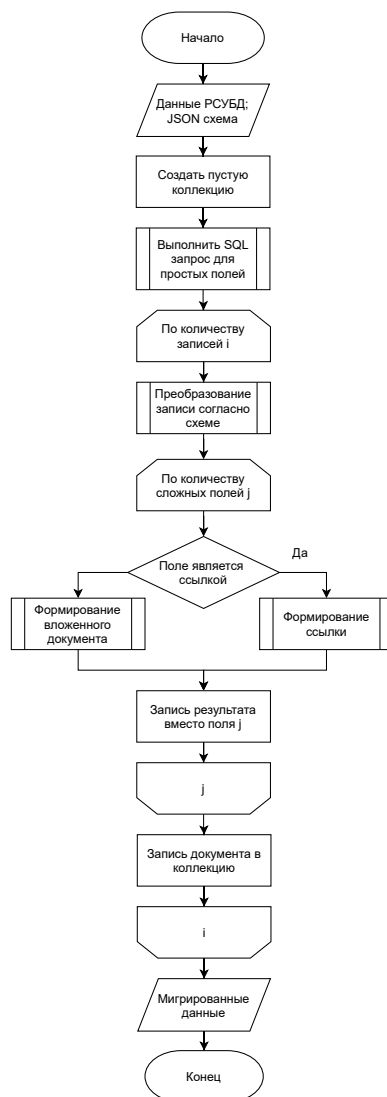


Рисунок 2.7 – Алгоритм преобразования данных.

Выводы

В данном разделе были описаны результаты разработки метода миграции данных из реляционной в документо-ориентированную базу данных с использованием семантического и частотного анализа. Были представлены используемые алгоритмы, описаны входные и выходные данные. Выполнено проектирование для дальнейшей программной реализации метода при помощи функциональной модели IDEF0 и схем алгоритмов.

3 Технологический раздел

В данном разделе производится выбор средств программной реализации метода миграции данных из реляционной в документо-ориентированную базу данных, описывается формат входных и выходных данных. Приводятся детали реализации программных компонентов, проводится проверка корректности работы разработанного метода, описывается взаимодействие пользователя с интерфейсом ПО, реализующим метод.

3.1 Выбор средств реализации

Для реализации разрабатываемого метода был выбран язык Java версии 17 [25]. Выбор языка обусловлен наличием опыта разработки и нативной поддержкой протокола JDBC, что позволяет взаимодействие с большинством РСУБД посредством JDBC-драйвера [26].

В разработанном ПО были использованы следующие библиотеки и фреймворки:

- Spring Framework (Spring Boot) [27] — фреймворк реализующий контейнер инверсии зависимостей, используется для связывания модулей приложения и конфигурации;
- Spring Data JDBC [28] — модуль интегрирующий JDBC в Spring Boot и предоставляющий верхнеуровневый интерфейс для обращения к базе данных;
- Spring Data MongoDB [29] — модуль предоставляющий интерфейс обёртку для драйвера MongoDB;
- Spring Shell [30] — модуль предоставляющий функциональность для разработки консольных интерфейсов;
- Jackson [31] — библиотека предоставляющая интерфейс для работы с JSON-объектами;
- pg_stat_statements [32] — модуль PostgreSQL логирующий запросы к СУБД.

3.2 Структура разработанного ПО

Структура разработанного ПО представлена на рисунке 3.1.

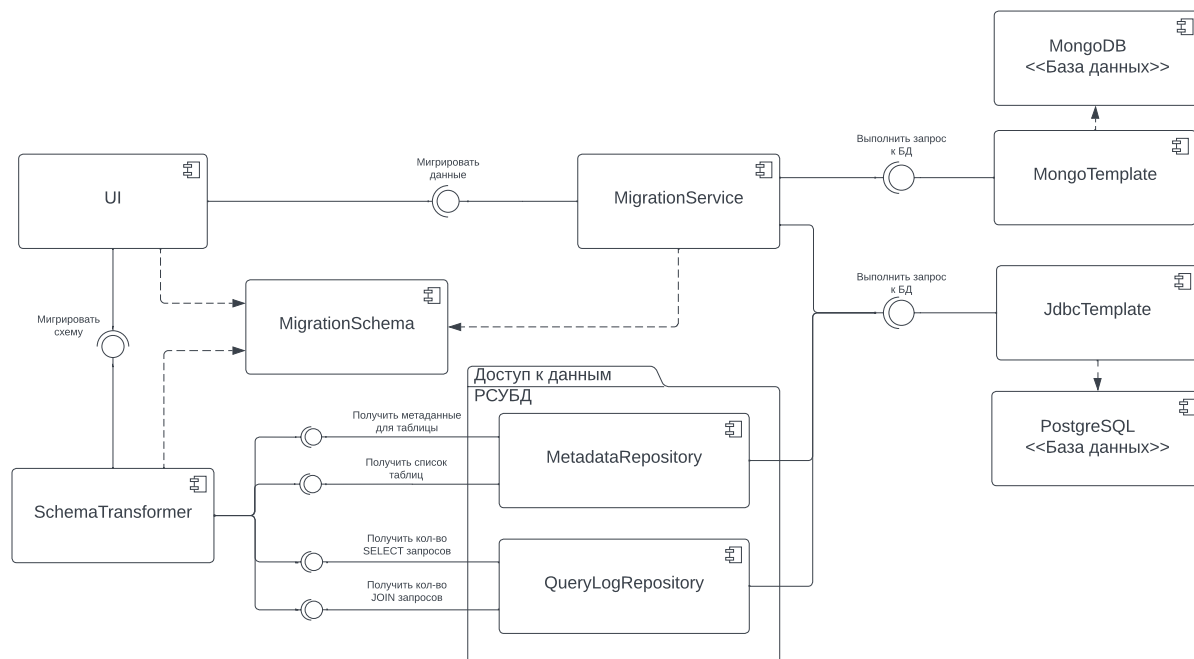


Рисунок 3.1 – Диаграмма компонентов

По разделено на следующие основные компоненты:

- Компонент доступа к данным, отвечает за получение метаданных и выполнение запросов к БД;
- Компонент преобразования схемы, отвечает за формирование промежуточной схемы;
- Компонент миграции, производит миграцию данных согласно промежуточной схеме;
- Компонент пользовательского интерфейса, отвечает за взаимодействия пользователя с системой и ввод данных.

3.2.1 Доступ к данным

ПО содержит два компонента доступа к данным.

1. Компонент доступа к PostgreSQL.
2. Компонент доступа к MongoDB.

Компонент доступа к MongoDB используется для записи данных в документо-ориентированную базу.

Компонент доступа к PostgreSQL используется для чтения, получения метаданных и истории запросов.

Получение метаданных

Получение метаданных о таблицах происходит по протоколу JDBC. Реализация протокола содержит класс `DatabaseMetadata` и метод `Connection::getMetadata`. `DatabaseMetadata` содержит информацию о доступных схемах, таблицах, столбцах, ограничениях и ключах.

Каждая СУБД содержит собственные средства для получения метаданных. PostgreSQL реализует как `information_schema`, описанную в стандарте SQL-92 [33], так и собственный набор представлений `pg_catalog` [34].

Перечисленные инструменты используются для получения следующей информации:

- список таблиц в схеме;
- метаданные столбцов (названия, типы данных);
- метаданные ключей (первичные и внешние ключи таблицы);
- метаданные отношений (внешние ключи, ссылающиеся на рассматриваемую таблицу).

Полученные метаданные позволяют сформировать граф, описывающий отношения между таблицами в схеме. На листинге 3.1 представлен метод получения метаданных о таблице.

Листинг 3.1 – Метода получения метаданных о таблице

```
public TableMetaData getTableMetadata(String schema, String
    name) {
    var table = new TableMetaData(name);
    Connection connection = null;
    try {
        connection =
            jdbcTemplate.getDataSource().getConnection();
        var metadata = connection.getMetaData();
        var pkeys =
            getPrimaryKeyMetadata(metadata.getPrimaryKeys(null,
                schema, name));
        var fkeys =
            getForeignKeyMetadata(metadata.getImportedKeys(null,
                schema, name));
        var cols = getColumnMetadata(metadata.getColumns(null,
            schema, name, null));
        var importedRelationships =
            getImportedRelationships(metadata, schema, name);
        var exportedRelationships =
            getExportedRelationships(metadata, schema, name);
        table.getPrimaryKeyMetadata().addAll(pkeys);
        table.getForeignKeyMetadata().addAll(fkeys);
        table.getColumnMetadata().addAll(cols);
        table.getImportedRelationships()
            .addAll(importedRelationships);
        table.getExportedRelationships()
            .addAll(exportedRelationships);
        table.setTableType(table.getForeignKeyMetadata().size()
            == 0 ?
                TableType.STRONG : TableType.WEAK);
    } catch (SQLException se) {
        log.error("Error while accessing metadata for table {}",
            name, se);
    } finally {
        DataSourceUtils.releaseConnection(connection,
            jdbcTemplate.getDataSource());
    }

    return table;
}
```

Получение истории запросов

Для получения истории запросов используется модуль `pg_stat_statements`. Этот модуль отвечает за логирование запросов к СУБД и подсчитывает количество выполненных запросов.

Путём агрегирования записей в этой таблице можно получить информацию о том, как часто производятся `SELECT` операции над определенной таблицей. Аналогично, возможно подсчитать количество `JOIN` операций между двумя таблицами.

Получив эти данные, можно высчитать относительное количество `JOIN`-ов и сделать выбор в пользу того или иного способа реализации отношения в документо-ориентированной схеме.

На листинге 3.2 представлены методы получения количества запросов.

Листинг 3.2 – Метода получения метаданных о таблице

```
public int getJoinData(String firstTable, String secondTable) {
    String likestmt = "%%%s%% %%join%%
        %%%s%%".formatted(firstTable, secondTable);
    String query = "SELECT sum(calls)
        FROM pg_stat_statements
        WHERE query LIKE ?";
    Integer joinCount = jdbcTemplate
        .queryForObject(query, Integer.class, likestmt);

    return joinCount != null ? joinCount : 0;
}

public int getSelectData(String table) {
    String likeSelectStmt = "select %%from %%%s%%";
    String likeJoinStmt = "%%%s%%join%%";
    String query = "SELECT sum(calls)
        FROM pg_stat_statements
        WHERE query LIKE ? AND query NOT LIKE ?";
    Integer selectCount = jdbcTemplate
        .queryForObject(query, Integer.class,
            likeSelectStmt, likeJoinStmt);

    return selectCount != null ? selectCount : 0;
}
```

3.2.2 Преобразование схемы

Компонент преобразования схемы отвечает за формирование промежуточной документо-ориентированной схемы из полученных метаданных. На основе метаданных, информации о предметной области и кратности отношений формируется JSON-документ, на основе которого будут преобразованы данные из реляционной базы данных.

Преобразование схемы происходит следующим образом:

1. Производится выбор таблиц, которые будут отображены в коллекции документов.
2. Начиная с каждой таблицы производится обход графа отношений в глубину, на каждом шаге которого производится либо перенос отношения в виде вложенного документа, либо в виде ссылки. Если отношение избыточно, его можно разорвать.
3. Запрашивается информация о кратности отношения на уровне записи в таблице, в случае, если переносится отношение «один-к-многим» формируется массив.
4. Метаданные используются для сохранения информации о отображении из РСУБД.
5. Формируется промежуточная схема, содержащая информацию о том, какие данные могут быть перенесены напрямую, а какие требуют дополнительных запросов. Для этого созданные ссылки и вложенные документы содержат информацию о источнике.

На листинге 3.3 представлены методы получения количества запросов.

Листинг 3.3 – Листинг промежуточной схемы

```
{
  "__name": "clients",
  "properties": {
    "id": { "name": "id",
            "datatype": "serial",
            "type": "property",
            "pk": true },
    "first_name": { "name": "first_name",
                    "datatype": "varchar",
                    "type": "property" },
    "client_cards": [ { "__name": "client_cards",
                        "properties": {
                          "card_id": { "source": "cards",
                                        "pkname": "id",
                                        "type": "reference" },
                          "card_num": { "name": "card_num",
                                        "datatype": "varchar",
                                        "type": "property",
                                        "pk": true },
                          "balance": { "name": "balance",
                                       "datatype": "numeric",
                                       "type": "property" } },
                        "type": "embedded" } ],
    "loans": [ { "__name": "loans",
                 "properties": {
                   "id": { "name": "id",
                           "datatype": "serial",
                           "type": "property",
                           "pk": true },
                   "amount": { "name": "amount",
                               "datatype": "numeric",
                               "type": "property" } },
                 "type": "embedded" } ]
  }
}
```


3.2.3 Миграция данных

Компонент миграции отвечает за перенос данных согласно сформированной схеме. Перед началом процесса миграции формируется очередь по наличию ссылок, таким образом в процессе миграции не произойдет попытки создать ссылку на документ, который еще не был создан.

Далее по очереди происходит формирование всех необходимых коллекций.

Сначала формируется представление документа верхнего уровня, далее параллельно для каждого документа, рекурсивно происходит формирование всех вложенных документов. Отдельно формируются запросы на формирование ссылок.

В результате миграции данных в MongoDB появляется отображение требуемых данных из PostgreSQL.

На листинге 3.4 представлен код формирования очереди миграции.

Листинг 3.4 – Формирование очереди

```
Deque<MigrationSchema> queue = new ArrayDeque<>();
migrationSchemaSet.stream()
    .filter(MigrationSchema::hasNoReferences)
    .forEach(queue::add);
migrationSchemaSet.removeAll(queue);
while (migrationSchemaSet.size() != 0) {
    Iterator<MigrationSchema> iterator =
        migrationSchemaSet.iterator();
    while (iterator.hasNext()) {
        MigrationSchema schema = iterator.next();
        var queueNameSet = queue.stream()
            .map(MigrationSchema::getName)
            .collect(Collectors.toSet());
        boolean hasQueuedReference = queueNameSet.stream()
            .anyMatch(schema::hasReference);
        boolean hasUnsafeReference = migrationSchemaSet.stream()
            .anyMatch(unsafe ->
                schema.hasReference(unsafe.getName()));
        if (hasQueuedReference && !hasUnsafeReference)
            queue.add(schema); iterator.remove();
    }
}
```

На листинге 3.5 представлен код миграции данных согласно сформированной схеме.

Листинг 3.5 – Миграция данных по схеме

```
topLevelSelectResult.parallelStream().forEach(resultRow -> {
    Document doc = new Document();
    if (schema.getPrimaryKey() != null)
        doc.put(schema.getPrimaryKey().name(),
            resultRow.get(schema.getPrimaryKey().name()));
    for (String key : schema.getProperties().keySet())
        doc.put(key, resultRow.get(key));
    for (String key : schema.getEmbeddedProperties().keySet())
        doc.put(key, resultRow.get(key));
    for (Map.Entry<String, EmbedArrProp> arrProp :
        schema.getEmbeddedArrayProperties().entrySet()) {
        Object currentId =
            resultRow.get(getPrimaryKeyColumn(schema.getName()));
        var propArray = getPropArray(schema, arrProp, currentId);
        doc.put(arrProp.getKey(), propArray);
    }
    for (Map.Entry<String, EmbeddedDoc> embeddedDoc :
        schema.getEmbeddedDocuments().entrySet()) {
        Object currentId =
            resultRow.get(getPrimaryKeyColumn(schema.getName()));
        var embeddedDocument =
            getDocument(embeddedDoc.getValue(), schema,
                currentId);
        doc.put(embeddedDoc.getKey(), embeddedDocument);
    }
    for (Map.Entry<String, Ref> referenceEntry :
        schema.getReferences().entrySet()) {
        Object currentId =
            resultRow.get(getPrimaryKeyColumn(schema.getName()));
        var reference = getReference(schema,
            referenceEntry.getValue(), currentId);
        doc.put(referenceEntry.getValue().name(), reference);
    }
    mongoTemplate.insert(doc, schema.getName());
});
```

3.3 Пользовательский интерфейс

Разработанное ПО поставляется в виде файла с расширением .jar и конфигурационного файла.

Запуск программы происходит командой `java -jar MigrationTool.jar`. В одной директории с программой должен находиться конфигурационный файл `application.properties`. На листинге 3.6 представлена структура конфигурационного файла.

Листинг 3.6 – Листинг файла конфигурации

```
# Конфигурация подключения к PostgreSQL
# ip_adress -- IP адрес подключения к базе данных
# port -- порт, по которому происходит подключение
# db_name -- название базы данных, к которой происходит
    подключение
# (pg_password, pg_username) -- пароль и имя пользователя
spring.datasource.url=
    jdbc:postgresql://{ip_adress}:{port}/{db_name}
spring.datasource.password={pg_password}
spring.datasource.username={pg_username}

# Конфигурация подключения к MongoDB
# (mdb_username, mdb_pass) -- Имя пользователя и пароль
# mdb_ip -- IP адрес сервера с MongoDB
# mdb_port -- Порт к которому происходит подключение
# mdb_name -- Название базы данных
# mdb_auth_source -- база аутентификации
# (MongoDB требует указать базу данных из которой берется
    информация для аутентификации)

spring.data.mongodb.uri=
    mongodb://{mdb_username}:{mdb_pass}@{mdb_ip}:{mdb_port}/
    {mdb_name}?authSource={mdb_auth_source}
```

В конфигурационном файле указываются параметры подключения к реляционной и документо-ориентированной СУБД: адрес, порт, название базы, параметры аутентификации.

Пользовательский интерфейс представляет собой консольное приложение, реализованное с использованием Spring Shell. При запуске приложения

пользователь попадает в интерактивную среду принимающую команды для исполнения. Список всех доступных команд может быть получен командой `help`.

На рисунке 3.2 представлен результат выполнения команды `help`.

```
shell:>help
AVAILABLE COMMANDS

Built-In Commands
  help: Display help about available commands
  stacktrace: Display the full stacktrace of the last error.
  clear: Clear the shell screen.
  quit, exit: Exit the shell.
  history: Display or save the history of previously run commands
  version: Show version info
  script: Read and execute commands from a file.

Info
  show tables all: Show all tables
  show table: Showcase Table rendering
  show tables selected: Show selected tables
  show meta: Show table metadata
  show json: Show JSON intermediate schema for a table
  show result: Show result

Migration
  migration finish: Transit data to target DB
  dump result: Dump resulting schema to text file
  migration start: Start migration algorithm
  migration init: Start migration process

shell:>
```

Рисунок 3.2 – Список доступных команд

Для того чтобы приступить к процессу миграции необходимо ввести команду **migration init**. После выполнения команды пользователь попадает в меню, в котором можно ввести схему в которой будет происходить работа и произвести выбор таблиц на основе которых будут созданы коллекции документов.

На рисунке 3.3 представлено меню выбора таблиц.

```
? Enter schema name nbo
? Choose tables to make main collections [Use arrows to move], type to filter
  ☐ acquirers
  ☐ targets
  ☐ categories
  ☒ cards
  ☒ clients
  ☐ client_cards
  ☒ transactions
  ☒ offers
  > ☐ loans
  ☒ offer_history
```

Рисунок 3.3 – Меню выбора таблиц

Во время выбора таблиц допускается фильтрация по имени таблицы путём ввода с клавиатуры. Выбор осуществляется за счет стрелок и нажатия клавиши «Пробел», при нажатии на клавишу «Ввод» выполнение команды завершится.

Список выбранных таблиц, а также введенная схема будут сохранены. После чего будет выполнен сбор метаданных и возврат в исходную интерактивную среду.

Для того чтобы продолжить процесс миграции следует ввести команду **migration start**. При вводе этой команды пользователь попадает в меню миграции таблиц, в котором необходимо по очереди провести миграцию каждой таблицы, пока не останется не мигрированных таблиц.

Перечислены таблицы, их статус, а также количество внешних ключей и внешних отношений с другими таблицами.

На рисунке 3.4 представлено меню выбора таблиц для миграции.

```
? Select table to migrate [Use arrows to move], type to filter
      offers      (FK: 1 REFERENCED BY: 1) [NOT MIGRATED]
      cards      (FK: 1 REFERENCED BY: 2) [NOT MIGRATED]
  >      clients   (FK: 0 REFERENCED BY: 4) [NOT MIGRATED]
      offer_history (FK: 2 REFERENCED BY: 0) [NOT MIGRATED]
      transactions (FK: 3 REFERENCED BY: 0) [NOT MIGRATED]

<*> RETURN
```

Рисунок 3.4 – Меню миграции таблиц

При выборе таблиц, происходит переход в режим миграции таблицы. В нем на экран выводится рассматриваемая таблица и перечисляются отношения которые необходимо мигрировать. Пользователь находится в режиме миграции таблицы, пока не будут мигрированы все отношения таблицы.

На рисунке 3.5 представлено меню выбора отношения для миграции.

```
? Select table to migrate cards
```

Name	Type	Key?
id	serial	PK
name	varchar	-
type	varchar	-
category	int4	FK
bonus	numeric	-

```
There are several external references
# cards.id <---- client_cards.card_id
# cards.id <---- offers.card_id

? Select relationship to resolve [Use arrows to move], type to filter
>
      category      FK      [ UNRESOLVED]
      client_cards  EXT      [ UNRESOLVED]
      offers        EXT      [ UNRESOLVED]
```

Рисунок 3.5 – Меню миграции таблицы

Далее при выборе отношения необходимо выбрать каким образом будет мигрировано отношение. На данном этапе выводится информация о количестве SELECT и JOIN запросов к таблицам.

На рисунке 3.6 представлено меню миграции отношения.

```
? Select table to migrate offers
```

Name	Type	Key?
id	serial	PK
name	varchar	-
description	varchar	-
card_id	int4	FK
start_dttm	date	-
end_dttm	date	-

```
There are several external references
# offers.id <---- offer_history.offer_id

? Select relationship to resolve cards
SELECT DATA

Table offers was selected 45560 times independently
JOIN DATA

Tables are joined 252 times
Joins occur 0.005531
? Select relationship to resolve [Use arrows to move], type to filter
> EMBED
  OMIT
  REFERENCE
```

Рисунок 3.6 – Меню миграции отношения

Далее происходит переход на этап определение кратности отношения, от кратности отношения будет зависеть то, какой тип будет иметь поле в JSON-документе.

На рисунке 3.7 представлено меню ввода кратности отношения.

```
? Select relationship to resolve REFERENCE
? Define a relationship type [Use arrows to move], type to filter
> ONE2ONE
  ONE2MANY
```

Рисунок 3.7 – Меню ввода кратности отношения

В случае, если отношение реализуется вложенным документом, необходимо произвести аналогичную процедуру миграции для вложенного документа. Когда все таблицы мигрированы происходит возврат в интерактивную среду.

Для того чтобы завершить процесс миграции следует ввести команду **migration finish**. При вводе этой команды на экран будет выведена сформированная очередь и будет начат процесс миграции данных в соответствии с сформированной на предыдущем этапе схемой.

На рисунке 3.8 представлен вывод команды **migration finish**.

```
shell:>migration finish
QUEUE: [cards, offers]
QUEUE: [cards, offers, clients]
QUEUE: [cards, offers, clients, offer_history]
QUEUE: [cards, offers, clients, offer_history, transactions]
Migrating cards...Migrating offers...Migrating clients...
```

Рисунок 3.8 – Вывод сформированной очереди

По завершении этой команды в MongoDB появятся мигрированные данные из PostgreSQL.

4 Исследовательский раздел

В данном разделе проводится исследование эффективности разработанного метода миграции данных. Проводится исследование зависимостей времени конвертации схемы от количества столбцов в таблицах и глубины вложенности документов. Описывается исследование зависимости объема выделяемой памяти при миграции данных от количества вложенных документов в схеме.

4.1 Исследование зависимости объема памяти от глубины вложенности формируемого документа

Процесс миграции данных существенно требователен по памяти. В связи с этим было проведено исследование зависимости выделяемой памяти в основных сегментах реализации метода.

Платформа JVM предоставляет инструментарий для профилирования разработанных приложений. Для замеров выделенной памяти использовался JDK Flight Recorder (JFR).

Для проведения исследования была сформирована база данных, состоящая из 15 таблиц связанных в древовидную структуру. Структура БД представлена на рисунке 4.1.

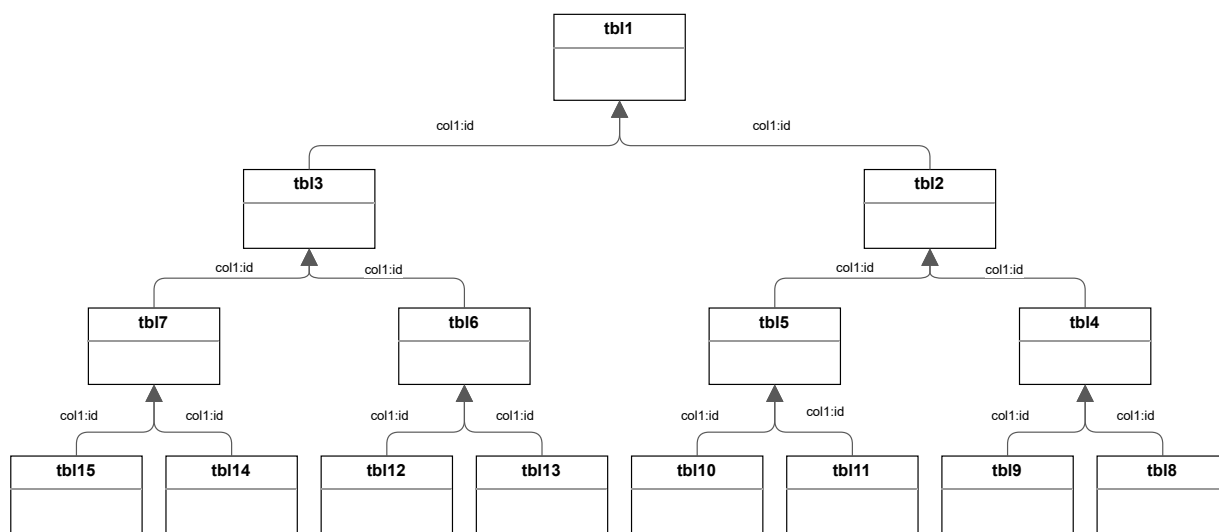


Рисунок 4.1 – Древовидная структура БД для исследования

Далее производились замеры выделенной памяти при миграции схемы и миграции данных. Также замерялось общее потребление памяти. Варьировалась глубина вложенности и как следствие количество и размер документов. На рисунке 4.2 представлены результаты исследования.

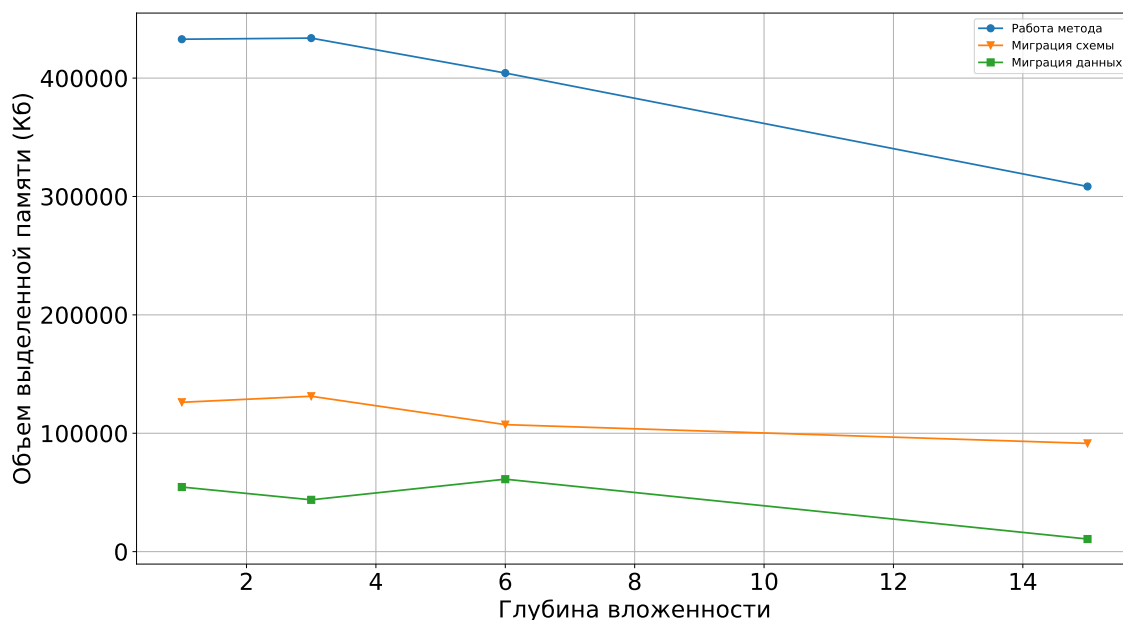


Рисунок 4.2 – Зависимость количества выделенной памяти к глубине вложенности

Выводы

В ходе исследования было выявлено, что денормализация структуры не приводит к росту потребляемой памяти, в то время как, миграция сильно денормализованной схемы требует больше памяти. Также при детальном рассмотрении количества потребляемой памяти было выявлено, что при отображении отношения в ссылку требуется столько же памяти, сколько при создании вложенного документа, при этом выполняется больше запросов к БД. Из чего можно сделать вывод, что следует отдавать предпочтение денормализации схемы при миграции схем содержащих большое количество сущностей.

Исходя из наблюдений за количеством выделяемой памяти также следует необходимость оценки доступной оперативной памяти при миграции больших объемов данных.

4.2 Исследование зависимости времени конвертации схемы от глубины вложенности

Формирование схемы является одним из основных этапов метода и определяет дальнейшие преобразования. По времени формирования схемы можно оценить время миграции без влияния сторонних факторов.

Для того, чтобы замерить время конвертации схемы был реализован дополнительный модуль, который не учитывал время ввода-вывода и подсчитывал время затраченное на преобразование конкретной схемы. Исследование проводилось на базе данных, структура которой обозначена на рисунке 4.1.

На рисунке 4.3 представлен график зависимости времени формирования схемы, от глубины формируемого дерева документов.

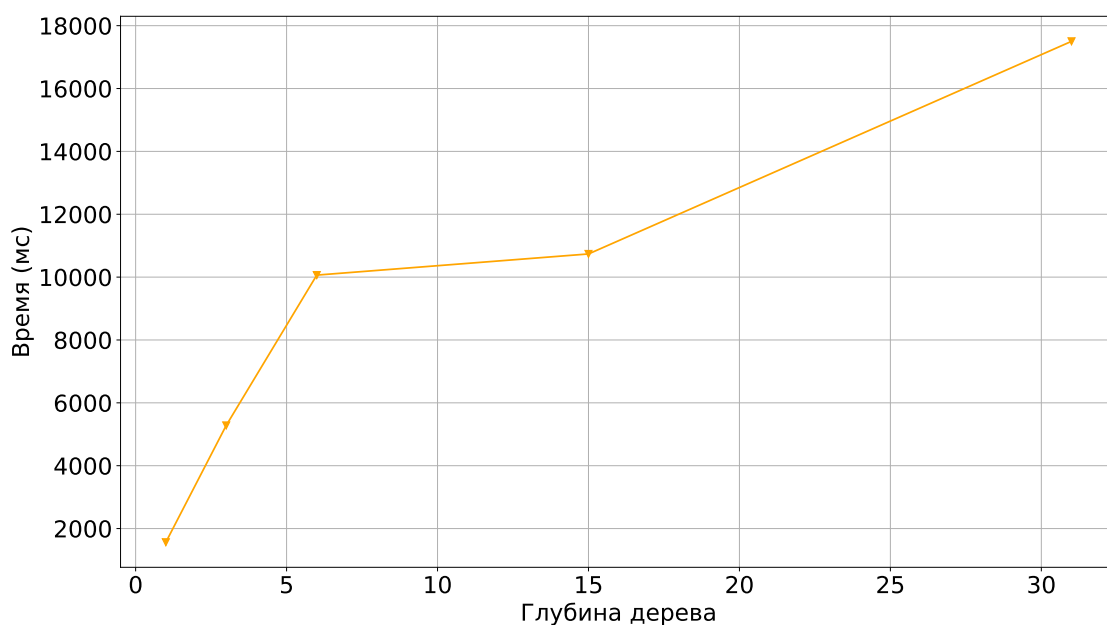


Рисунок 4.3 – Зависимость времени формирования схемы от глубины вложенности

Также были произведены замеры времени конвертации в зависимости от количества столбцов в таблице.

График зависимости приведен на рисунке 4.4.

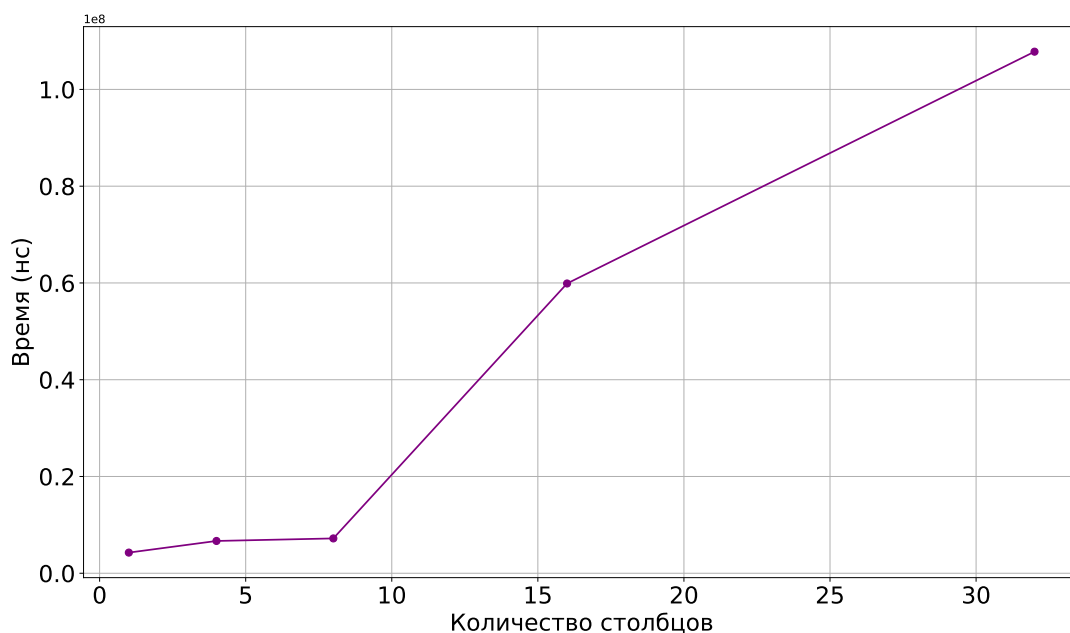


Рисунок 4.4 – Зависимость времени конвертации от количества столбцов

Выводы

Исходя из результатов исследования можно сделать вывод, что время конвертации растет пропорционально глубине вложенности формируемого документа. Количество столбцов в каждой таблице практически не влияет на время конвертации так как занимает наносекунды.

По полученным результатам можно судить о эффективности разработанного метода и о применимости к миграции относительно больших объемов данных.

ЗАКЛЮЧЕНИЕ

В результате был разработан метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа.

В ходе работы были выполнены все поставленные задачи.

- рассмотрены подходы к миграции данных из реляционных в документо-ориентированные базы данных и проведено сравнение методов миграции данных;
- разработан метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа;
- спроектировано и реализовано ПО, демонстрирующее работу метода;
- проведено исследование эффективности разработанного метода по времени и по памяти.

Можно выделить следующие направления дальнейшего развития.

- покрытие широкого набора типов данных;
- разработка графического интерфейса;
- миграция циклических зависимостей;
- миграция индексов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DB-Engines Ranking [Электронный ресурс]. — Режим доступа URL: <https://db-engines.com/en/ranking> (Дата обращения: 10.11.2022).
2. DB-Engines ranking method [Электронный ресурс]. — Режим доступа URL: https://db-engines.com/en/ranking_definition (Дата обращения: 29.05.2023).
3. *Codd E. F.* A Relational Model of Data for Large Shared Data Banks // Commun. ACM. — New York, NY, USA, 1970. — Янв. — Т. 13, № 6. — С. 377–387. — ISSN 0001-0782. — DOI: 10.1145/362384.362685. — URL: <https://doi.org/10.1145/362384.362685>.
4. *Chen, Peter, Pin-Shan.* The Entity-Relationship Model—toward a Unified View of Data // ACM Trans. Database Syst. — New York, NY, USA, 1976. — Март. — Т. 1, № 1. — С. 9–36. — ISSN 0362-5915. — DOI: 10.1145/320434.320440. — URL: <https://doi.org/10.1145/320434.320440>.
5. *Codd E. F.* Extending the Database Relational Model to Capture More Meaning // ACM Trans. Database Syst. — New York, NY, USA, 1979. — Дек. — Т. 4, № 4. — С. 397–434. — ISSN 0362-5915. — DOI: 10.1145/320107.320109. — URL: <https://doi.org/10.1145/320107.320109>.
6. *Chamberlin D. D., Boyce R. F.* SEQUEL: A Structured English Query Language // Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control. — Ann Arbor, Michigan : Association for Computing Machinery, 1974. — С. 249–264. — (SIGFIDET '74). — ISBN 9781450374156. — DOI: 10.1145/800296.811515. — URL: <https://doi.org/10.1145/800296.811515>.
7. *Date C.* An Introduction to Database Systems. — Addison-Wesley Publishing Company, 1995. — (Addison-Wesley systems programming series). — ISBN 9780201543292.
8. *Khan W., Kumar T., Cheng Z.* SQL and NoSQL Databases Software architectures performance analysis and assessments—A Systematic Literature review. — 2022.

9. *Sadalage P. J., Fowler M.* NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence //. — 1st. — Addison-Wesley Professional, 2012. — ISBN 0321826620.
10. *Brewer E. A.* Towards Robust Distributed Systems (Abstract) // Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. — Portland, Oregon, USA : Association for Computing Machinery, 2000. — C. 7. — (PODC '00). — ISBN 1581131836. — DOI: 10.1145/343477.343502. — URL: <https://doi.org/10.1145/343477.343502>.
11. *Brewer E. A.* A Certain Freedom: Thoughts on the CAP Theorem // Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. — Zurich, Switzerland : Association for Computing Machinery, 2010. — C. 335. — (PODC '10). — ISBN 9781605588889. — DOI: 10.1145/1835698.1835701. — URL: <https://doi.org/10.1145/1835698.1835701>.
12. *Lee E. A., Bateni S., Lin S.* Quantifying and Generalizing the CAP Theorem. — 2021. — arXiv: 2109.07771. — URL: <https://arxiv.org/abs/2109.07771>.
13. *Bansal N., Soni K., Sachdeva S.* Journey of Database Migration from RDBMS to NoSQL Data Stores // Big-Data-Analytics in Astronomy, Science, and Engineering / под ред. S. Sachdeva, Y. Watanobe, S. Bhalla. — Cham : Springer International Publishing, 2022. — C. 159—177. — ISBN 978-3-030-96600-3.
14. *Arora R., Aggarwal R. R.* An algorithm for transformation of data from MySQL to NoSQL (MongoDB) // International Journal of Advanced Studies in Computer Science and Engineering. — 2013. — T. 2, № 1. — C. 6—12.
15. *Zhao G., Lin Q., Li L.* Schema Conversion Model of SQL Database to NoSQL // 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. — 2014. — C. 355—362. — DOI: 10.1109/3PGCIC.2014.137.
16. *Gu Y., Shen S., Wang J.* Application of nosql database mongodb // 2015 IEEE International Conference on Consumer Electronics-Taiwan. — IEEE. 2015. — C. 158—159.

17. *Chickerur S., Goudar A., Kinnerkar A.* Comparison of relational database with document-oriented database (mongodb) for big data applications // 2015 8th International Conference on Advanced Software Engineering & Its Applications (ASEA). — IEEE. 2015. — С. 41—47.
18. *Hamouda S., Zainol Z.* Document-Oriented Data Schema for Relational Database Migration to NoSQL // 2017 International Conference on Big Data Innovations and Applications (Innovate-Data). — 2017. — С. 43—50. — DOI: 10.1109/Innovate-Data.2017.13.
19. *Schreiner G. A., Duarte D., Mello R. d. S.* When relational-based applications go to NoSQL databases: A survey // Information. — 2019. — Т. 10, № 7. — С. 241.
20. *Störl U., Hauf T., Klettke M.* Schemaless NoSQL data stores-Object-NoSQL Mappers to the rescue? // Datenbanksysteme für Business, Technologie und Web (BTW 2015). — 2015.
21. Hibernate OGM [Электронный ресурс]. — Режим доступа URL: <https://hibernate.org/ogm/> (Дата обращения: 16.11.2022).
22. *Drenckberg S. G., Politze M., Center I.* Migration of a web service back-end from a relational to a document-oriented database. — 2018.
23. *Hanine M., Bendarag A., Boutkhoul O.* Data migration methodology from relational to NoSQL databases // International Journal of Computer, Electrical, Automation, Control and Information Engineering. — 2015. — Т. 9, № 12. — С. 2566—2570.
24. MongoSyphon [Электронный ресурс]. — Режим доступа URL: <https://github.com/johnlpage/MongoSyphon> (Дата обращения: 26.11.2022).
25. OpenJDK 17 [Электронный ресурс]. — Режим доступа URL: <https://openjdk.org/projects/jdk/17/> (Дата обращения: 20.05.2023).
26. JDBC API Specification [Электронный ресурс]. — Режим доступа URL: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> (Дата обращения: 20.05.2023).
27. Spring Boot [Электронный ресурс]. — Режим доступа URL: <https://spring.io/projects/spring-boot> (Дата обращения: 20.05.2023).

28. Spring Data JDBC Reference Documentation [Электронный ресурс]. — Режим доступа URL: <https://docs.spring.io/spring-data/jdbc/docs/current/reference/html/> (Дата обращения: 20.05.2023).
29. Spring Data MongoDB Reference Documentation [Электронный ресурс]. — Режим доступа URL: <https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/> (Дата обращения: 20.05.2023).
30. Spring Data MongoDB Reference Documentation [Электронный ресурс]. — Режим доступа URL: <https://docs.spring.io/spring-shell/docs/2.1.2/site/reference/htmlsingle/> (Дата обращения: 20.05.2023).
31. Jackson-Databind Documentation [Электронный ресурс]. — Режим доступа URL: <https://github.com/FasterXML/jackson-databind/wiki> (Дата обращения: 20.05.2023).
32. PostgreSQL pg_stat_statements Documentation [Электронный ресурс]. — Режим доступа URL: <https://www.postgresql.org/docs/current/pgstatstatements.html> (Дата обращения: 20.05.2023).
33. ANSI SQL-92 [Электронный ресурс]. — Режим доступа URL: <http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt> (Дата обращения: 20.05.2023).
34. PostgreSQL System Catalogs [Электронный ресурс]. — Режим доступа URL: <https://www.postgresql.org/docs/current/catalogs.html> (Дата обращения: 20.05.2023).

ПРИЛОЖЕНИЕ А