

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Основная часть	5
1.1 Постановка задачи	5
1.2 Входные и выходные данные	6
1.3 Используемые средства	6
1.4 Требования к разрабатываемому ПО	7
1.5 Взаимодействие с ПО	8
1.6 Структура разработанного ПО	9
1.6.1 Доступ к данным	10
1.6.2 Преобразование схемы	13
1.6.3 Миграция данных	15
1.7 Тестирование	17
1.8 Пользовательский интерфейс	19
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	25

ВВЕДЕНИЕ

Во время выполнения выпускной квалификационной работы был разработан метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа.

Цель работы — разработать программное обеспечение, реализующее разработанный метод, провести тестирование.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать инструменты и технологии, требуемые для реализации ПО;
- описать входные и выходные данные;
- сформулировать требования к разрабатываемому ПО;
- описать взаимодействие пользователя с интерфейсом ПО;
- провести тестирование.

1 Основная часть

В данном разделе формализуется постановка задачи, описываются средства программной реализации метода миграции данных из реляционной в документо-ориентированную базу данных. Проводится тестирование метода и предоставляются его результаты. Описывается взаимодействие пользователя с интерфейсом ПО, реализующим метод.

1.1 Постановка задачи

Формализуем постановку задачи в виде IDEF0-диаграммы. На рисунке 1.1 представлена IDEF0-диаграмма метода миграции данных из реляционной в документо-ориентированную базу данных нулевого уровня.



Рисунок 1.1 – IDEF0-диаграмма нулевого уровня

1.2 Входные и выходные данные

В качестве входных данных разработанное ПО принимает:

- подключение к РСУБД PostgreSQL;
- подключение к документо-ориентированной СУБД MongoDB;
- название схемы;
- информация о предметной области в виде выбранных таблиц и кратности отношений между сущностями.

Выходными данными являются мигрированные данные в MongoDB.

1.3 Используемые средства

Для реализации разрабатываемого метода был выбран язык Java версии 17 [1]. Выбор языка обусловлен наличием опыта разработки и нативной поддержкой протокола JDBC, что позволяет взаимодействие с большинством РСУБД посредством JDBC-драйвера [2].

В разработанном ПО были использованы следующие библиотеки и фреймворки:

- Spring Framework (Spring Boot) [3] — фреймворк реализующий контейнер инверсии зависимостей, используется для связывания модулей приложения и конфигурации;
- Spring Data JDBC [4] — модуль интегрирующий JDBC в Spring Boot и предоставляющий верхнеуровневый интерфейс для обращения к базе данных;
- Spring Data MongoDB [5] — модуль предоставляющий интерфейс обёртку для драйвера MongoDB;
- Spring Shell [6] — модуль предоставляющий функциональность для разработки консольных интерфейсов;
- Jackson [7] — библиотека предоставляющая интерфейс для работы с JSON-объектами;

- pg_stat_statements [8] — модуль PostgreSQL логирующий запросы к СУБД.

1.4 Требования к разрабатываемому ПО

Разработанное ПО, должно предоставлять:

- возможность подключения к реляционной и документо-ориентированной базе данных;
- возможность выбора схемы и таблиц, которые будут мигрированы;
- возможность ввода кратности отношений между сущностями;
- возможность переноса данных из PostgreSQL в MongoDB.

1.5 Взаимодействие с ПО

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 1.1 представлена структура конфигурационного файла.

Листинг 1.1 – 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.

1.6 Структура разработанного ПО

Структура разработанного ПО представлена на рисунке 1.2.

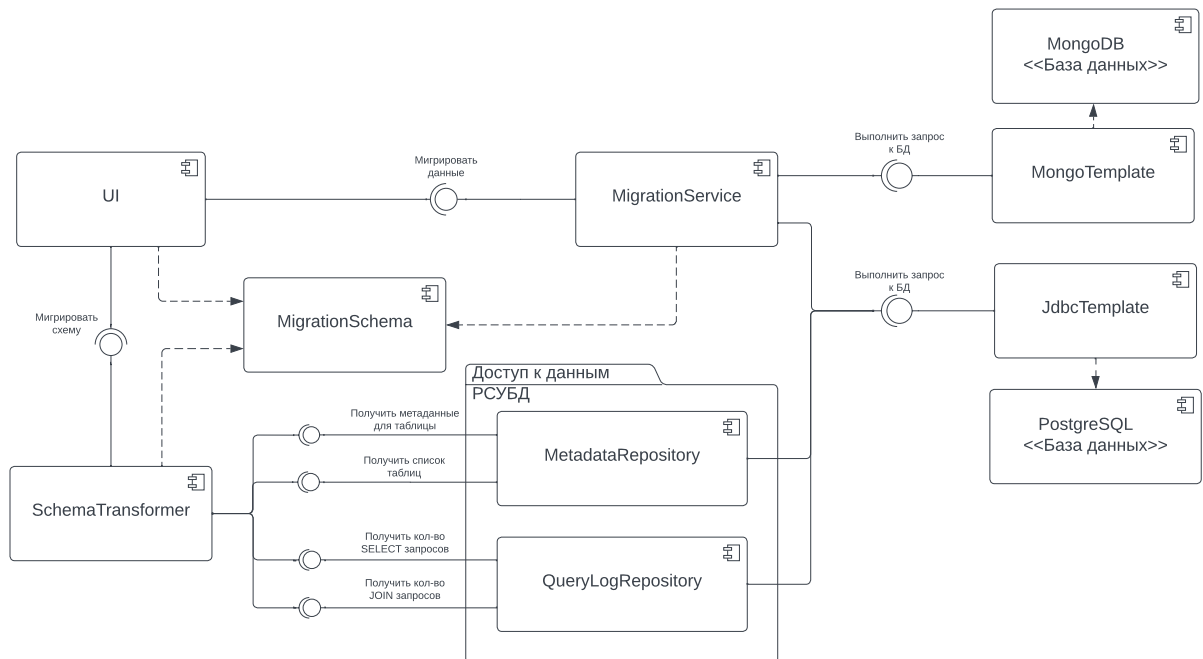


Рисунок 1.2 – Диаграмма компонентов

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.

1.6.1 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 [9], так и собственный набор представлений `pg_catalog` [10].

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 1.2 представлен метод 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 1.3 представлены методы получения количества запросов.

Листинг 1.3 – 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;
}
```

1.6.2 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 1.4 представлены методы получения количества запросов.

Листинг 1.4 – 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" } ]
  }
}
```

1.6.3 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 1.5 представлен код формирования очереди миграции.

Листинг 1.5 – 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 1.6 представлен код миграции данных согласно сформированной схеме.

Листинг 1.6 – 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());
});
```

1.7 Тестирование

Тестирование разработанного ПО проводилось несколькими путями. При разработке использовались модульные тесты, которые определяли работу модулей преобразования схемы, правильность переноса промежуточной схемы и отображение типов.

Помимо этого проводилось тестирование на нескольких базах данных.

На рисунке 1.3 представлена ER-диаграмма одной из баз на которой проводилось тестирование.

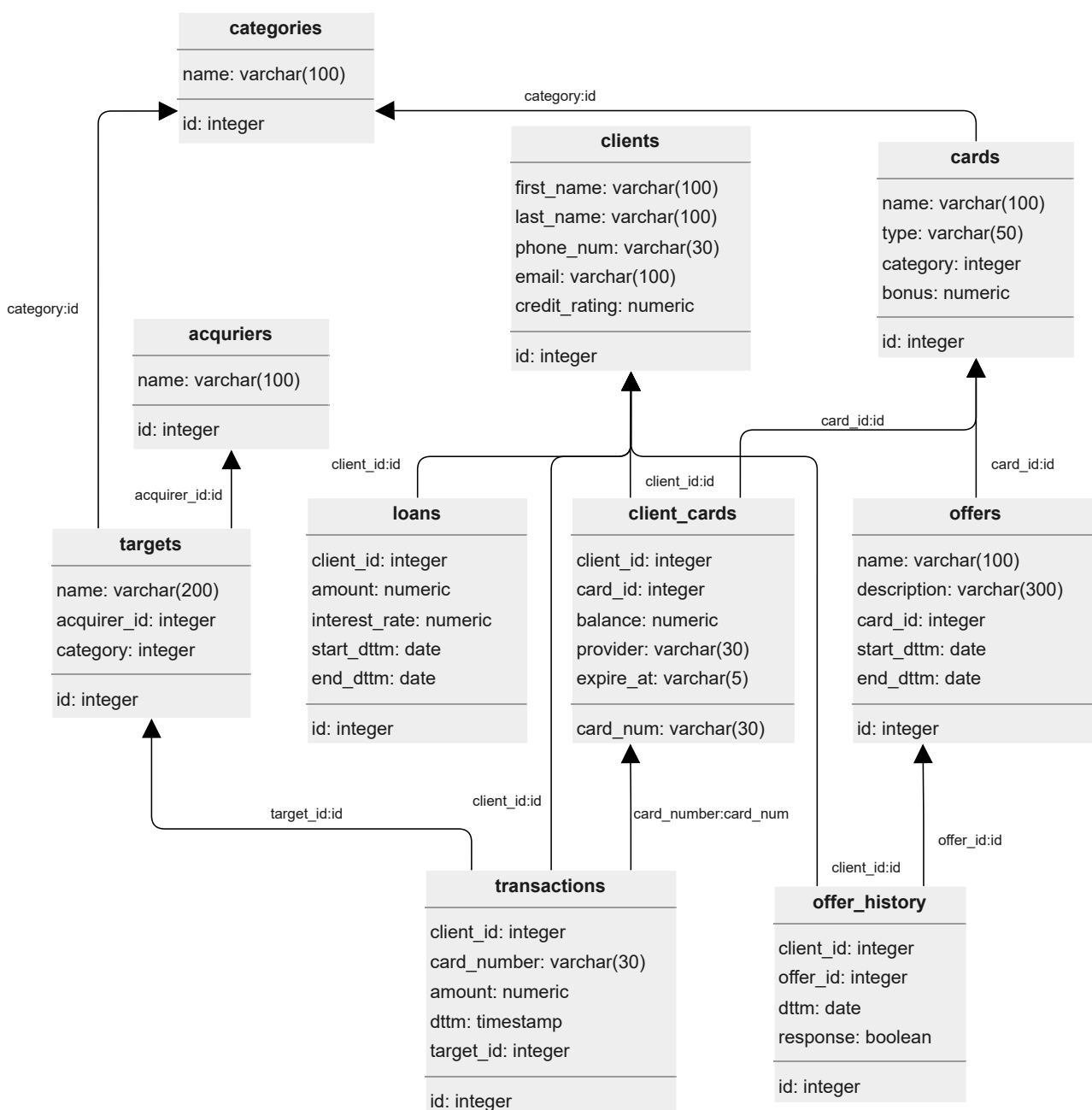


Рисунок 1.3 – ER-диаграмма тестовой базы данных

В ходе тестирования производилась миграция базы данных, после чего производилась проверка на наличие эквивалентного отображения каждой записи из реляционной базы данных в документо-ориентированной.

Каждый созданный документ верхнего уровня хранит копию первичного ключа из РСУБД, что позволяет проверить каждая ли запись была перенесена путём выполнения запросов с поиском по одному значению в обе базы данных.

Также проверялось наличие отношений, которые были отброшены, так как в результирующей базе не должно присутствовать отношений, которые были разорваны в процессе миграции.

На листинге 1.7 представлен пример результирующего документа в MongoDB.

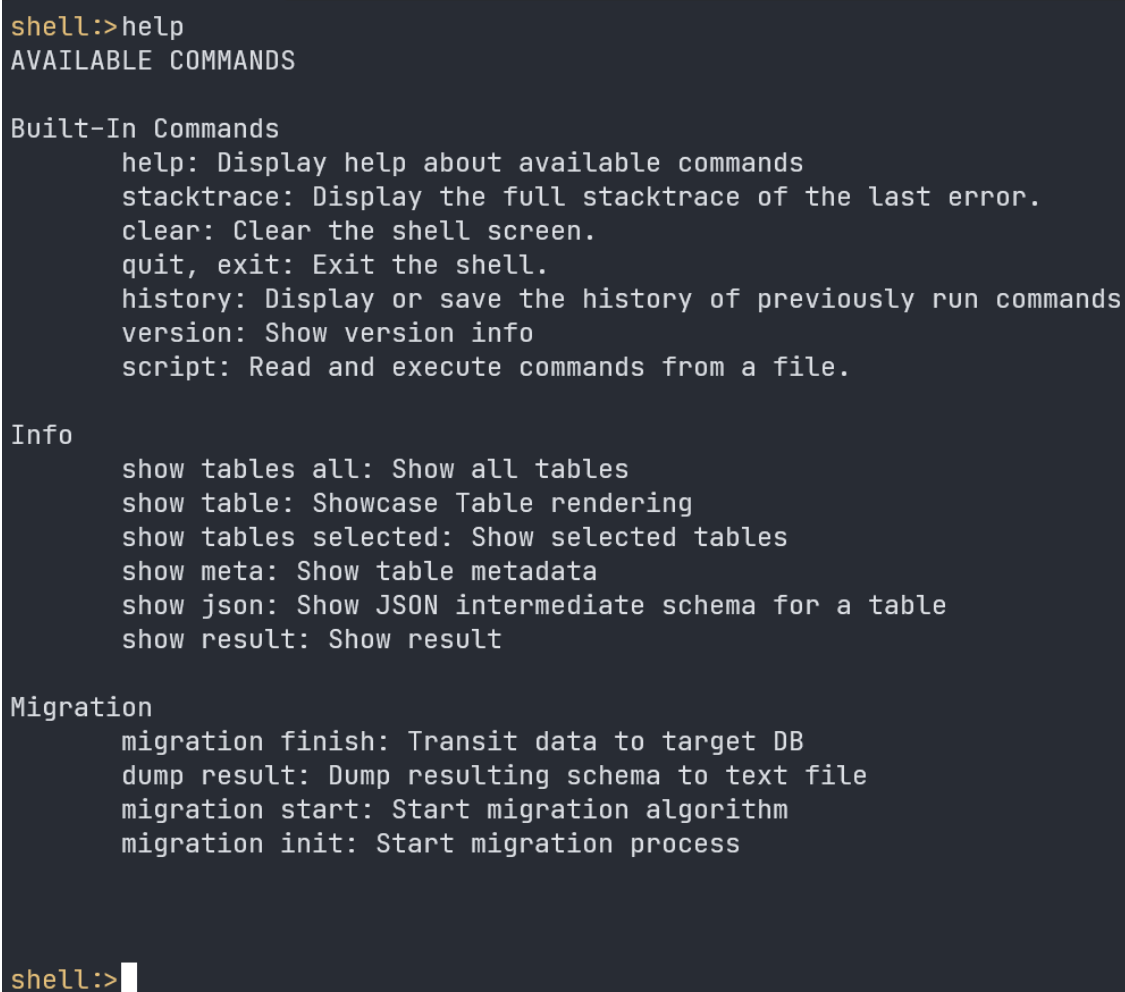
Листинг 1.7 – Документ коллекции clients

```
"_id": {"$oid": "6476733d0391932e5df198d7"},
"client_cards": [ {
  "card_num": "4504589339479680",
  "balance": {"$numberDecimal": 945015},
  "provider": "American Express",
  "expire_at": "10/26",
  "card_id": {"$oid": "6476725fd6e65f5a77e01fdc"}
}, {
  "card_num": "348536622402117",
  "balance": {"$numberDecimal": 944193},
  "provider": "American Express",
  "expire_at": "08/23",
  "card_id": {"$oid": "6476725fd6e65f5a77e01fd7"}
} ],
"credit_rating": "232",
"email": "klitkina@example.net",
"first_name": "Октябрина",
"id": 15,
"last_name": "Филипповна",
"loans": [ {
  "id": 5922,
  "amount": {"$numberDecimal": 97912860},
  "start_dttm": {"$date": "2022-08-07T21:00:00.000Z"},
  "interest_rate": {"$numberDecimal": 6},
  "end_dttm": {"$date": "2023-08-30T21:00:00.000Z"}
} ],
"phone_num": "+7 189-846-93-55"
```


1.8 Пользовательский интерфейс

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 1.4 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 1.4 – 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 1.5 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 1.5 – 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 смигрированных таблиц.

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

На рисунке 1.6 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 1.6 – Меню миграции таблиц

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

На рисунке 1.7 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 1.7 – 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 1.8 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 1.8 – 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 1.9 представлено меню ввода кратности отношения.

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

Рисунок 1.9 – 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 1.10 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 1.10 – Вывод сформированной очереди

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

ЗАКЛЮЧЕНИЕ

Было разработано программное обеспечение, реализующее разработанный в ходе выполнения выпускной квалификационной работы метод миграции данных из реляционной в документо-ориентированную базу данных с использованием частотного и семантического анализа.

В ходе работы были выполнены все поставленные задачи:

- описаны инструменты и технологии, требуемые для реализации ПО;
- описаны входные и выходные данные;
- сформулированы требования к разрабатываемому ПО;
- описано взаимодействие пользователя с интерфейсом ПО;
- проведено тестирование метода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. OpenJDK 17 [Электронный ресурс]. — Режим доступа URL: <https://openjdk.org/projects/jdk/17/> (Дата обращения: 20.05.2023).
2. JDBC API Specification [Электронный ресурс]. — Режим доступа URL: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> (Дата обращения: 20.05.2023).
3. Spring Boot [Электронный ресурс]. — Режим доступа URL: <https://spring.io/projects/spring-boot> (Дата обращения: 20.05.2023).
4. 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).
5. 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).
6. 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).
7. Jackson-Databind Documentation [Электронный ресурс]. — Режим доступа URL: <https://github.com/FasterXML/jackson-databind/wiki> (Дата обращения: 20.05.2023).
8. PostgreSQL pg_stat_statements Documentation [Электронный ресурс]. — Режим доступа URL: <https://www.postgresql.org/docs/current/pgstatstatements.html> (Дата обращения: 20.05.2023).
9. ANSI SQL-92 [Электронный ресурс]. — Режим доступа URL: <http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt> (Дата обращения: 20.05.2023).
10. PostgreSQL System Catalogs [Электронный ресурс]. — Режим доступа URL: <https://www.postgresql.org/docs/current/catalogs.html> (Дата обращения: 20.05.2023).