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

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Граф ссылок в Википедии

Студенты гр. 6303 Ваганов Н.А.

Доброхвалов М.О.

Шевченко Д.В.

Преподаватель Заславский М.М.

Санкт-Петербург

2019

ЗАДАНИЕ

Студенты Ваганов Н.А., Доброхвалов М.О., Шевченко Д.В.

Группа 6303

Тема проекта: Разработка приложения для изучения графа ссылок Википедии

Исходные данные:
Необходимо реализовать приложение для работы со статьями Википедии для СУБД Neo4j

Содержание пояснительной записки:
«Содержание», «Введение», «Качественные требования к решению», «Сценарии использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студенты гр. 6303		Ваганов Н.А.
		Доброхвалов М.О.
		Шевченко Д.В.
Преподаватель		Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения для построение графа ссылок Википедии для СУБД Neo4j. Исходный код приложения и вся дополнительная информация доступна в репозитории проекта [1].

SUMMARY

In the course was planned to develop an application in the team for one of the themes. Topic was chosen building Wikipedia links graph for DMS Neo4j. The source code of the application and all additional information is available in the project repository [1].

Содержание.

Введение.

Цель работы – создать удобное решение для хранения и анализа информации графе ссылок Википедии.

Для реализации поставленной задачи было разработано веб-приложение, которое позволит хранить в электронном виде всю информацию о ссылках, позволяя при этом удобно с ними взаимодействовать.

Качественные требования к решению.

Требуется разработать приложение с использованием СУБД Neo4j.

Сценарии использования.

Макет UI.

Ссылка на макет доступна в разделе «Список использованных источников» [2].

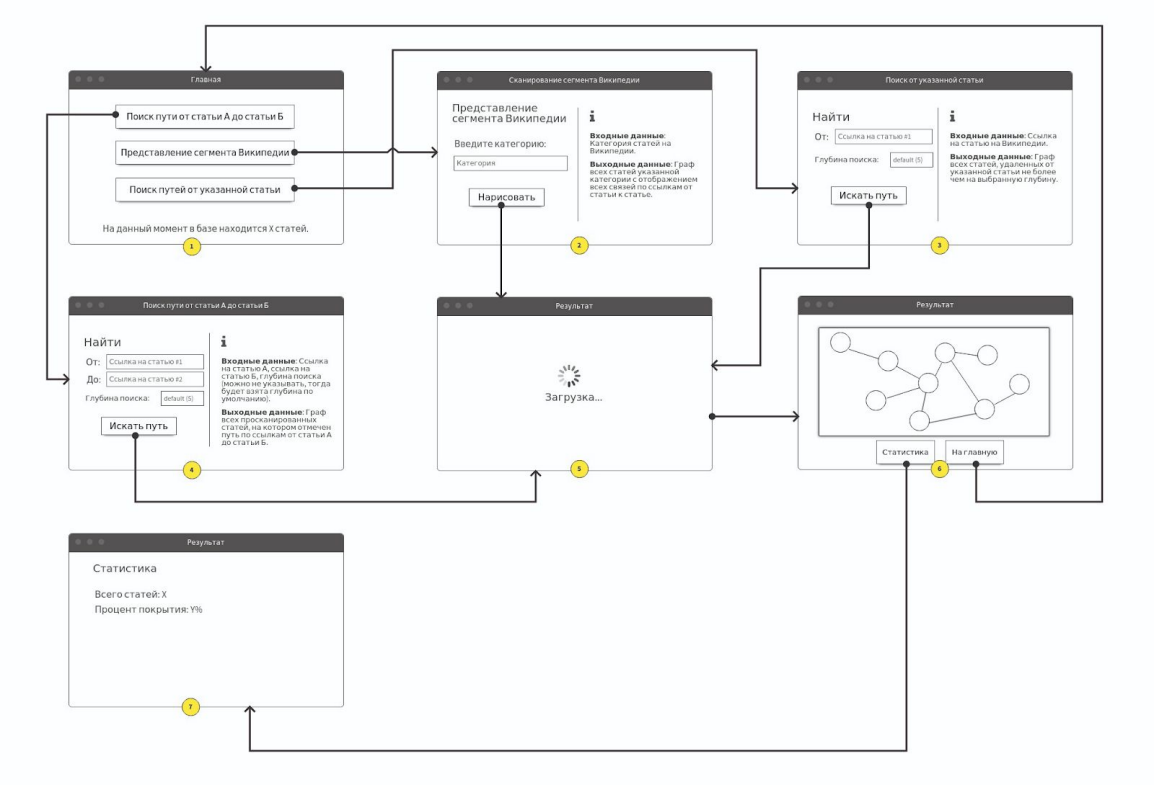

Рисунок 1 – Общий вид.

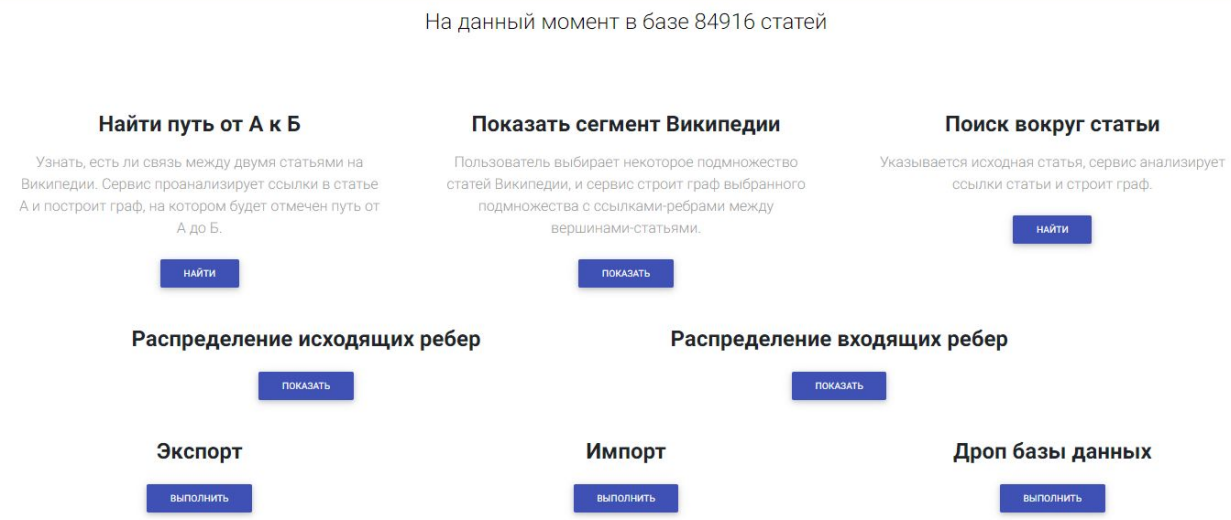

Рисунок 2 – Главная страница

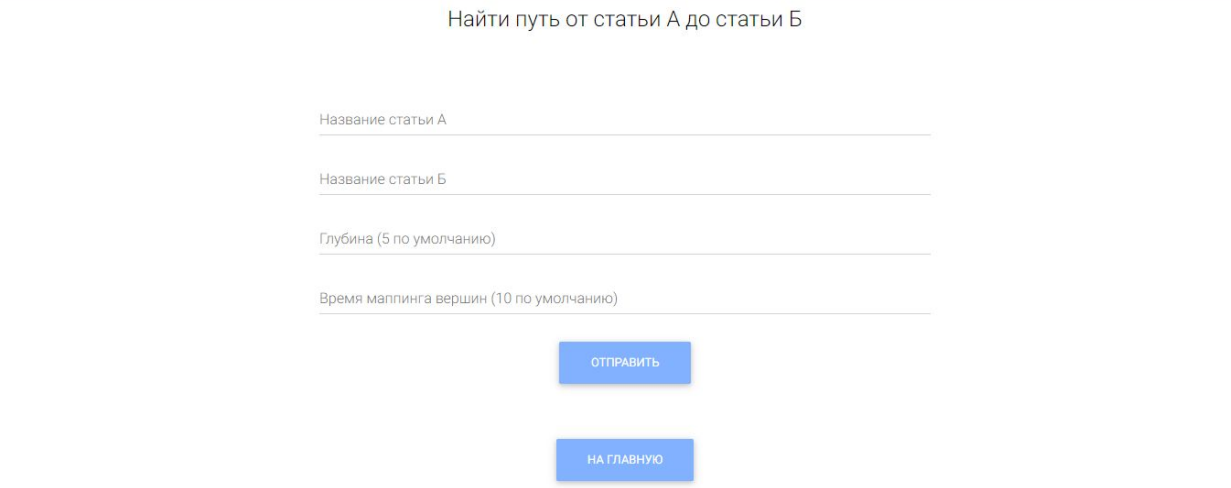

Рисунок 3 – Страница ввода данных для поиска пути

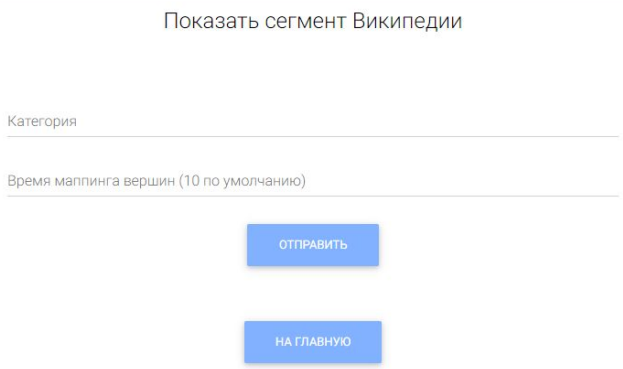

Рисунок 4 – Страница ввода данных для сегмента(категории)

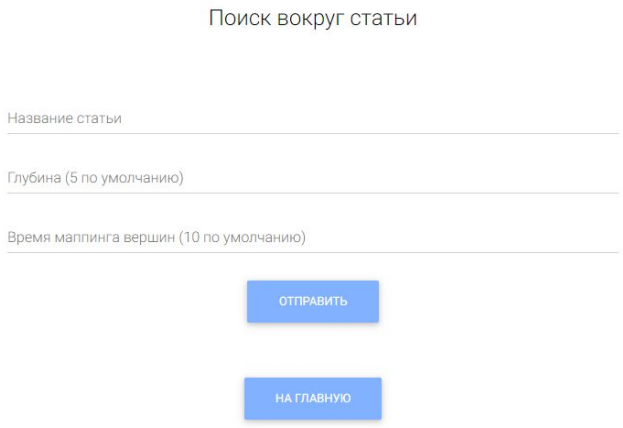

Рисунок 5 – Страница ввода данных для сканирования в глубину

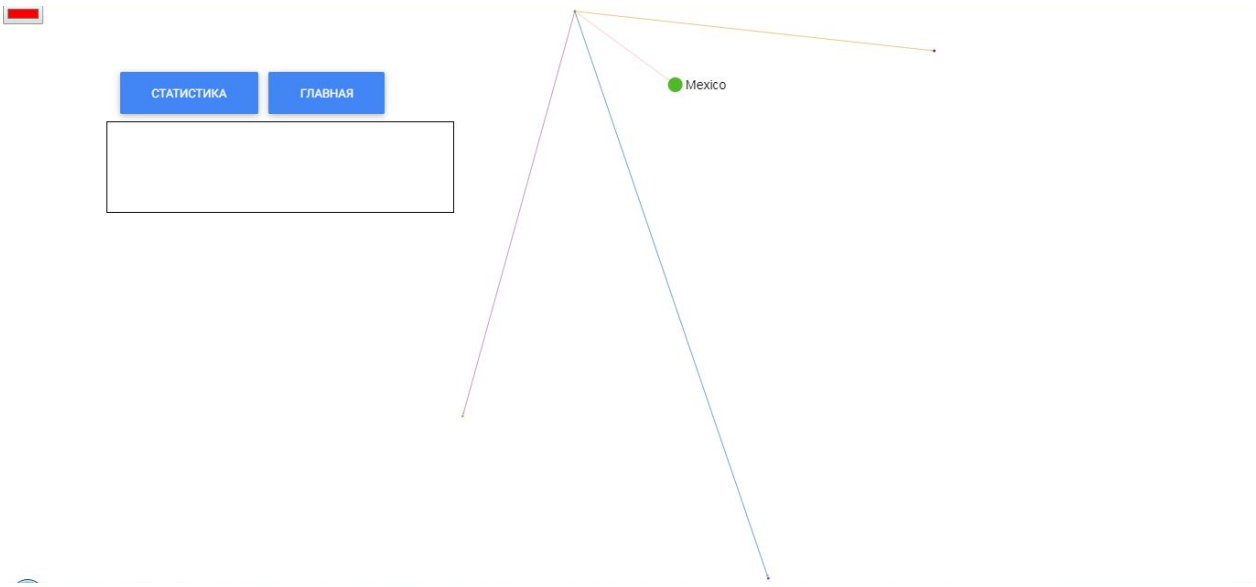

Рисунок 6 – Страница отрисовки графа

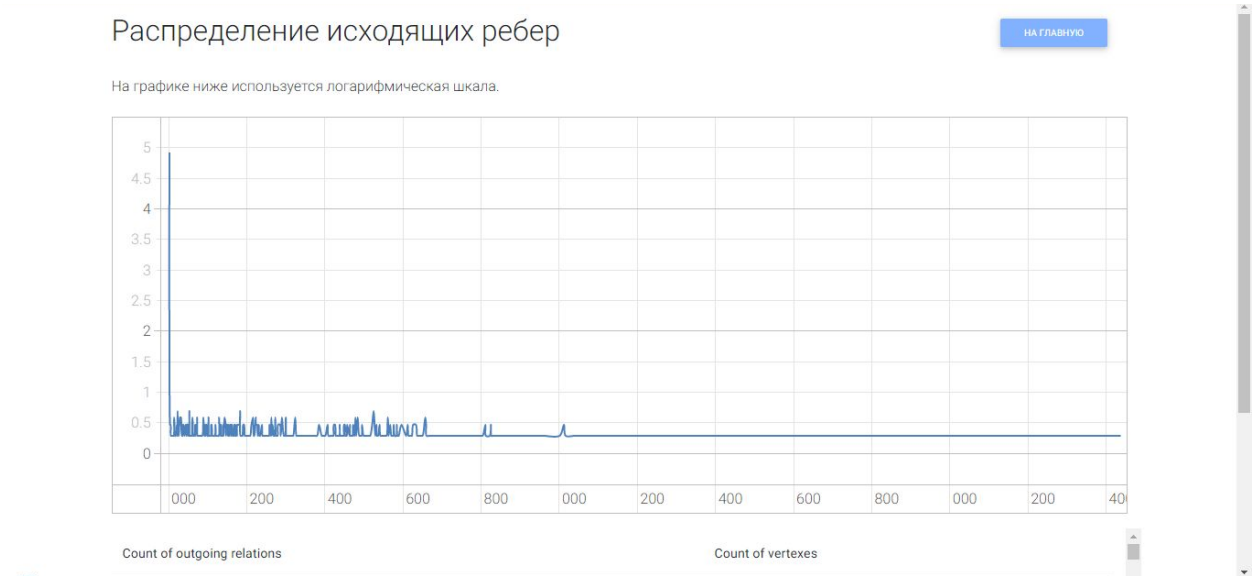

Рисунок 7 – Страница с информацией о распределении количества ребер

Описание сценариев использования.

Сценарий использования - «Поиск пути от статьи А до статьи Б».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь переходит на страницу ввода данных для поиска пути.
3. Пользователь вводит данные в поля. Например, в поле «Название статьи А» - «Roman Catholic Diocese of Tlaxcala», в поле «Название статьи Б» - «Mexicans", »в поле «Глубина» - «3», в поле «Время маппинга вершин» - «3».
4. Пользователь нажимает на кнопку «Отправить».
5. Пользователь ожидает загрузку информации.
6. Пользователь изучает предоставленную информацию.

Сценарий использования - «Показать сегмент Википедии».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь переходит на страницу ввода данных для получения данных о сегменте Википедии.
3. Пользователь вводит данные в поля. Например, в поле «Категория» - «Articles with hAudio microformats», в поле «Время маппинга вершин» - «15».
4. Пользователь ожидает загрузку информации.
5. Пользователь изучает предоставленную информацию.

Сценарий использования - «Поиск вокруг статьи».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.

2. Пользователь переходит на страницу ввода данных для получения данных о статьях, на которые ведут ссылки с указанной статьи.
3. Пользователь вводит данные в поля. Например, в поле «Название статьи» - «Luis Munive Escobar», в поле «Глубина» - «1», в поле «Время маппинга вершин» - «5».
4. Пользователь ожидает загрузку информации.
5. Пользователь изучает предоставленную информацию.

Сценарий использования - «Распределение исходящих ребер».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь переходит на страницу с информацией о распределении исходящих ребер.
3. Пользователь ожидает загрузку информации
4. Пользователь изучает график и таблицу распределения.

Сценарий использования - «Распределение входящих ребер».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь переходит на страницу с информацией о распределении входящих ребер.
3. Пользователь ожидает загрузку информации
4. Пользователь изучает график и таблицу распределения.

Модель данных.

NoSQL модель данных (Neo4j).

Описание модели данных.

Необходимо хранить два типа узлов – Статьи (лейбл Article) и Категории (лейбл Category). Отношения разделяются на два типа - принадлежность к категории (метка «CategoryLink») и ссылки между статьями (метка «ArticleLink»).

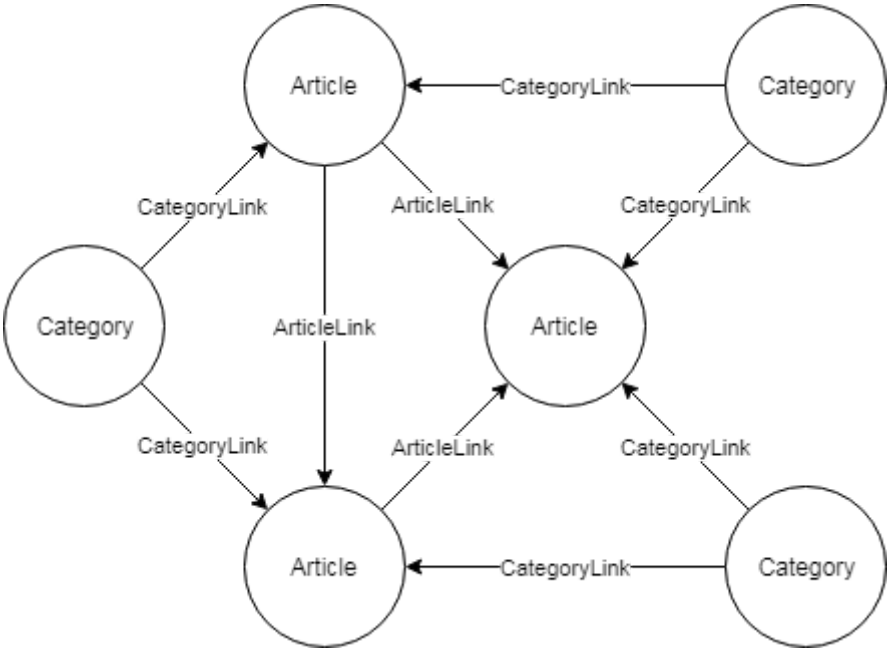

Рисунок 8 – Модель данных в Neo4j

Вычисление примерного объема данных.

У узлов с меткой *Article* будет следующий набор свойств:

- **_id** - $V_{id} = 4b$
- **articleTitle** - string => Vt = 2B * Nt, где Nt = 50 => Vt = 100B
- **size** - integer => Va_id = 8B

Суммарный объем памяти, занимаемый одной статьей (сущностью) Varticle = Vid + Vt + Va_id = 112B

У узлов с меткой Category будет следующий набор свойств:

- **_id** - $V_{id} = 4b$
- **categoryTitle** - string => Vt = 2B * Nt, где Nt = 50 => Vt = 100B

Суммарный объем памяти, занимаемый одним человеком

Vcategory = Vid + Vt = 104B

Также необходимо хранить _id для каждого типа отношений (ArticleLink и CategoryLink)

=> $V_{al} = V_{cl} = V_{rel} = 4B$

В среднем с одной статьи ведет 200 ссылок, каждая статья принадлежит 30 категориям

Следовательно, суммарный объем для хранения одной статьи будет равен:

$V_a = V_{article} + N_c * (V_{category} + V_{cl}) + N_a * V_{al}$, где

- Na - среднее количество ссылок со статьи,
- Nc - среднее количество категорий статьи.

$V_a = V_{article} + 50 * (V_{category} + V_{cl}) + 200 * V_{al} = 4272B$

Объем данных для хранения **N** статей будет равен:

$V(N) = N * (V_{article} + N_c * (V_{category} + V_{cl}) + N_a * V_{al}) = 4272B * N$

Датасет содержит ~5.8 миллионов статей, следовательно хранение всего англоязычного сегмента Википедии в Neo4j потребует около 23GB.

Примеры запросов.

- Запрос на создание узла, хранящего в себе информацию о статье

```
MERGE ( article:Article { articleTitle: "Joseph Stalin", size: 236261})
```

Пример исходного кода запроса к Neo4j на создание узла, хранящего в себе статью

- Запрос на создание узла, хранящего в себе информацию о категории

```
MERGE (category:Category{categoryTitle:'Joseph Stalin'});
```

Пример исходного кода запроса к Neo4j на создание узла, хранящего в себе категорию

- Запрос на создание отношения между статьей и категорией

```
  MATCH  (category:Category{categoryTitle:'Joseph  Stalin'}),  (  article:Article  {
   articleTitle: "Joseph Stalin"})
 MERGE (category)-[:CategoryLink]->(article)
```

Пример исходного кода запроса к Neo4j на создание отношения между статьей и категорией

- Запрос на добавление одной статьи

```
 MERGE (article:Article { articleTitle: "Joseph Stalin", size: 236261}),
   (:article{articleTitle: “Tbilisi State University”}),
...
(category:Category{categoryTitle:'Joseph Stalin'});
...
 MATCH ( articleJS:Article { articleTitle: "Joseph Stalin"}),
   (articleTSU:article{articleTitle: “Tbilisi State University”}), ...
   (categoryJS:Category{categoryTitle:'Joseph Stalin'}), ...
 MERGE (categoryJS)-[:CategoryLink]->(articleJS)
...
(articleJS)-(:ArticleLink)->(articleTSU)
```

Пример исходного кода запроса к Neo4j на добавление одной статьи

Таким образом формула количества запросов для одной статьи:

$QueryNum = 3$

- Запрос для поиска пути от статьи article_first до статьи article_second, глубиной не более depth

```
MATCH (a1:article {title: 'article_first'}) -
   [:link*1..depth]->(middle:article)-[:TO|:CC|:BCC]->(a2:article {title:
   'article_second'}) RETURN distinct a1, middle, a2
```

Пример исходного кода запроса к Neo4j для поиска пути между статьями acticle_first и article_second, глубиной не более depth

- Запрос для сканирования сегмента Википедии

```
MATCH (n:Category{categoryTitle:"categ"})-->(a),
(n:Category{categoryTitle:"categ"})-[*0..1]->(b)
OPTIONAL MATCH (a)-[r]->(b)
RETURN a,b,r;
```

Пример исходного кода запроса к Neo4j для сканирования сегмента Википедии

Результат для категории «Identifiers»:

a	b	r

{
"size": 32659,
"articleTitle":”Digital object identifier"}
	{
	"size": 22986,
	"articleTitle": "Handle System"
	}
		{}

{
"size": 13592,
"articleTitle": ”Virtual International Authority File"
}
	{
	"size": 22986,
	"articleTitle": "Handle System"
	}
		null

{
"size": 32659,
"articleTitle": "Digital object identifier"
}
	{
	"size": 13592,
	"articleTitle": "Virtual International Authority File"
	}
		{}

{
"size": 22986,
"articleTitle": "Handle System"
}
	{
	"size": 13592,
	"articleTitle": "Virtual International Authority File"
	}
		null

{
"size": 13592,
"articleTitle": "Virtual International Authority File"
}
	{
	"size": 32659,
	"articleTitle": "Digital object identifier"
	}
		{}

{
"size": 22986,
"articleTitle": "Handle System"
}
	{
	"size": 32659,
	"articleTitle": "Digital object identifier"
	}
		{}

- Запрос для получения всех статей до определенной глубины depth вокруг выбранной пользователем статьи article_title.

```
MATCH (n1:article{title:"article_title"})-[:link *0..depth]->(n2:article)
RETURN n2
```

Пример исходного кода запроса к Neo4j для получения всех статей вокруг article_title, глубиной не более depth

SQL модель данных.

Описание модели данных.

В реляционной модели можно было бы хранить данные следующим самым оптимальным способом:

1. Таблица "Articles"
    i. **title** - VARCHAR(255) -> 255B
    ii. **id** - INT -> 4B
    iii. **size** - INT -> 4B
    iv. **update_time** -> 3B
2. Таблица "Categories"
    i. **id** - INT -> 4B
    ii. **title** - VARCHAR(255) -> 255B
3. Таблица "Links"
    i. **id_link_from** - INT -> 4B
    ii. **id_link_to** - INT -> 4B

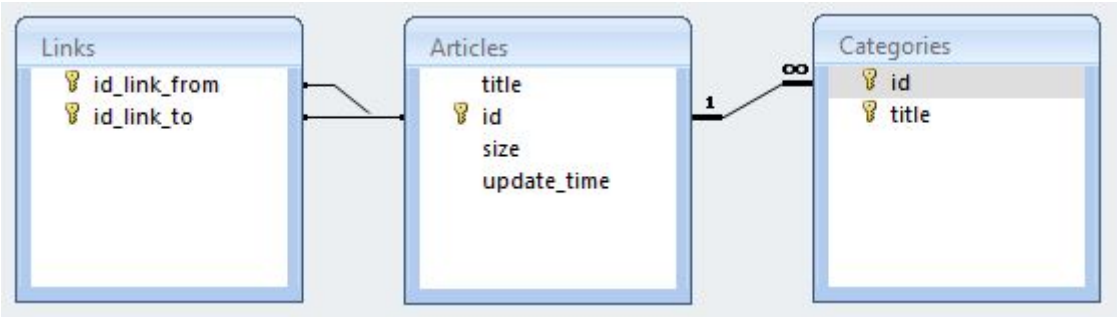

Рисунок 9 – Модель данных SQL

Вычисление примерного объема данных.

Возьмем те же средние показатели, что и при оценке объема данных в нереляционной модели (в среднем со статьи ведет 200 ссылок, каждая статья принадлежит к 40 категориям).

V(N) = (Vart + 40*Vc + 200 * Vl) * N = ((255B + 4B + 4B + 3B) + 40 * (4B + 255B) + 200*(4B + 4B)) * N = (12226 * N)B

Хранение данных будет занимать около 66GB

Запросы.

- Запрос на добавление статьи в базу данных

```
INSERT INTO Articles VALUES ("Joseph Stalin", 15641, 236261, DATE_ADD(CURDATE(), INTERVAL 7 DAY)) INSERT INTO Categories VALUES (15641, "Joseph Stalin"), (15641, "1878 births"), (15641, "1953 deaths"), (15641, "People from Gori"), (15641, "Georgia"), (15641, "People from Tiflis Governorate"), (15641, "Russian Social Democratic Labour Party members"))
```

Пример исходного кода SQL-запроса для добавления статьи

- Запрос на создание ребра (ссылки)

```
 SELECT Id FROM Articles WHERE Title IN("Joseph Stalin”, “Soviet Union")
python/c/c++
...
getting id_1, id_2 and create query:
INSERT INTO Links Values (id_1, id_2)
```

Пример исходного кода SQL-запроса для создания ребра (ссылки)

- Запрос на поиск пути от статьи article_first до статьи article_second, глубиной не более depth

```
input: start_el_title , finish_el_title, depth
sql: select id from articles where title in (start_el_title, finish_el_title);
checked_ids = set(start_el_id)
start_ids = [start_el_id]
depth

while(depth and !finded)
depth -= 1
sql: select * from links where start_id in start_ids;
check finish_el_id in to_
if True:
```

```
__ finded
else:
__ delete from to_ ids, which exists in checked_ids
__ checked_ids.update(from_)
__ start_ids = to_
endif
endwhile
if (finded) graph_seach_path using python/other progr lang
```

Пример исходного кода SQL-запроса для поиска пути между статьями article_first и article_second, глубиной не более depth

Количество запросов: depth + 1

- Запрос на сканирование определенного сегмента Википедии category

```
input: category_title
sql: SELECT Articles.* FROM Articles LEFT JOIN Categories
__ ON Articles.id = Categories.id
__ WHERE Categories.title = ;
ids = [i.id for i in info]
sql: SELECT * FROM Links
__ WHERE id_link_from in ids AND id_link_to in ids;
```

Пример исходного кода SQL-запроса для сканирования определенного сегмента Википедии category

Количество запросов: 2

- Запрос на поиск всех статей до определенной глубины depth вокруг выбранной пользователем статьи article_title.

```
sql: SELECT id FROM articles WHERE title = start_article;
start_ids = [start_el_id]
while(depth)
depth -= 1
// get ids from_ and to_
sql: SELECT * FROM links WHERE start_id IN start_ids
delete from to_ ids, which exists in checked_ids
checked_ids.update(from_)
start_ids = to_
checked_ids.update(to_)
endwhile
sql: SELECT * FROM Articles WHERE id in checked_ids;
```

```
sql: SELECT * FROM Links
__ WHERE id_link_from in checked_ids AND id_link_to in checked_ids;
```

Пример исходного кода SQL-запроса для поиска всех статей до определенной глубины depth вокруг выбранной пользователем статьи article_title

Количество запросов: *depth* + 3

Сравнение SQL и NoSQL.

- В SQL реализации модели данных пришлось бы создавать дополнительные таблицы для связей, что увеличивает суммарное количество создаваемых таблиц.
- В SQL версии данные в среднем занимают больше места.
- Количество запросов, необходимых для выполнения юзкейсов в SQL модели больше.

Разработанное приложение.

Краткое описание.

Для упрощения процесса разработки, а также для улучшения производительности приложения на конечном клиенте, было принято решение разделить приложение на frontend и backend.

Frontend реализован с использованием bootstrap[4], а также JavaScript библиотек jQuery[5] и Sigma[6] .

Backend реализован с использованием Kotlin. Сборка проекта происходит при помощи Gradle[7], работу с базой данных Neo4j[3] обеспечивает драйвер на Kotlin.

Схема экранов приложения.

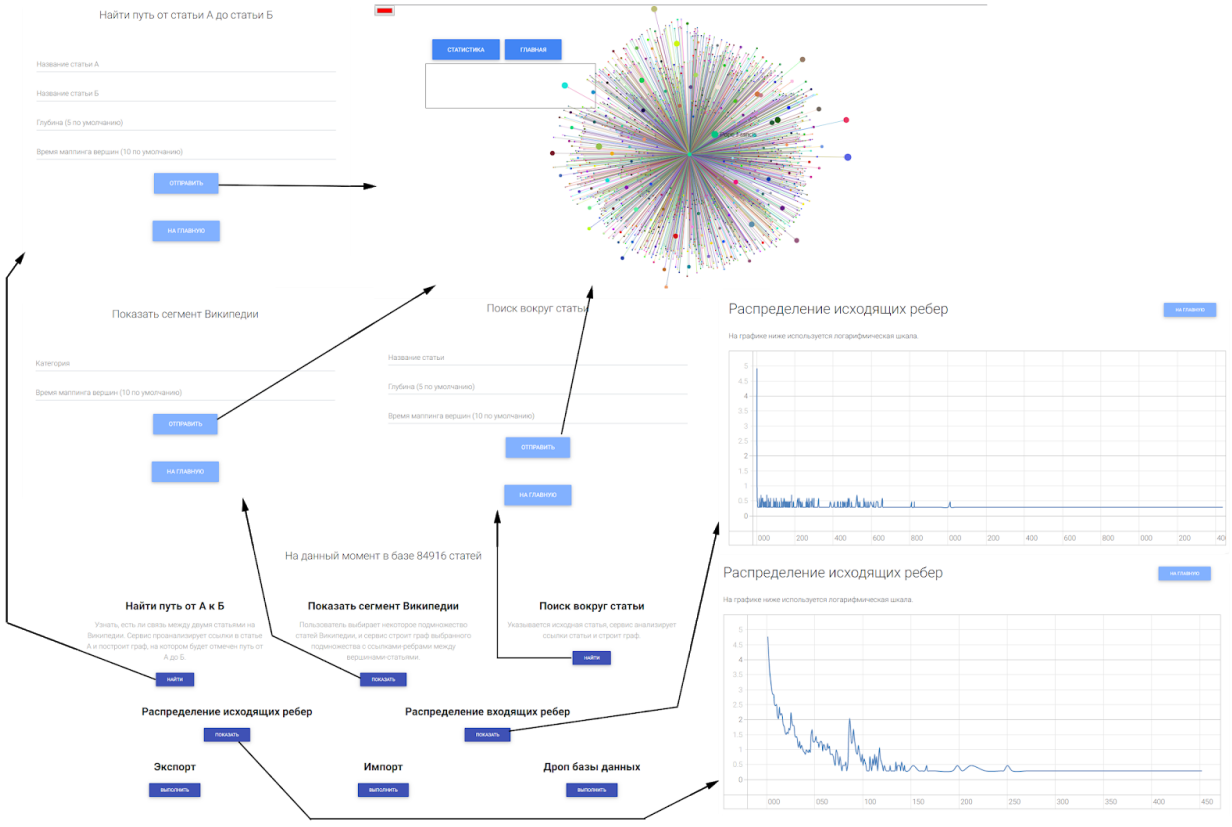

Рисунок 10 – Схема экранов приложения

Использованные технологии.

- Kotlin
- Neo4j
- Gradle
- Frontend: bootstrap, jQuery, Sigma.js

Ссылка на приложение.

Ссылка на приложение доступна в разделе «Список использованных источников» [1].

Заключение.

В ходе работы было разработано приложение для предоставления информации о графе ссылок Википедии. Помимо основного функционала, приложение предоставляет возможность импорта и экспорта данных. Для отрисовки графов используется алгоритм маппинга вершин, который базируется на сжатии ребер (сила упругости) и отталкивании вершин (закон Кулона). Для качественного результата на больших графах необходимо большое количество итераций. Возможным способом улучшения работы приложения является использование усовершенствованных алгоритмов, позволяющих снизить вычислительную нагрузку, либо использование параллельных вычислений.

Список использованных источников.

1. Репозиторий проекта «Граф ссылок в Википедии»: https://github.com/moevm/nosql1h19-wikig-raph
2. Макет UI проекта «Граф ссылок в Википедии»: https://ibb.co/Cz2C2w3
3. Документация Neo4j: https://neo4j.com/docs/graph-algorithms/current/
4. Документация по bootstrap: https://getbootstrap.com/docs/4.3/getting-started/introduction/
5. Документация по jQuery: https://api.jquery.com/
6. Документация по Sigma.js: https://github.com/jacomyal/sigma.js/wiki/Public-API
7. Документация по Gradle: https://docs.gradle.org/current/userguide/userguide.html