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

КУРСОВАЯ РАБОТА

по дисциплине «Введение в нереляционные базы данных»

ТЕМА: АНАЛИЗ СЮЖЕТОВ ФИЛЬМОВ

Студенты гр. 6304 Иванов В.С.

Григорьев И.С.

Зубов К.А.

Преподаватель Заславский М.М.

Санкт-Петербург
2019

ЗАДАНИЕ

Студенты

Иванов В.С.

Григорьев И.С.

Зубов К.А.

Группа 6304

Тема проекта: Разработка приложения для анализа сюжетов фильмов.

Исходные данные:

Необходимо реализовать приложение для работы с информацией о фильмах 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.6304 _____ Иванов В.С.

_____ Григорьев И.С.

_____ Зубов К.А.

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 анализа сюжетов фильмов для СУБД Neo4j, так как был вызван интерес в связи с возможностью нахождения интересных закономерностей в творчестве различных режиссеров. Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql1h19-plot-anaysis

SUMMARY

In the course was planned to develop an application in the team for one of the themes. Topic was chosen analysis of movie’s plots for DMS Neo4j. It was caused by the possibility of finding interesting patterns in the work of various directors. Source code and all the additional information can be found at: https://github.com/moevm/nosql1h19-plot-anaysis

Содержание.

Введение. 7
Качественные требования к решению. 7
Сценарии использования. 7
Макет UI. 7
Описание сценариев использования. 11
Сценарий использования - «Посмотреть сюжет/съёмочную группу фильма». 11
Сценарий использования - «Фильма нет в базе данных». 12
Сценарий использования - «Фильмы с конкретным актёром». 12
Сценарий использования - «Фильмы конкретного режиссёра». 13
Сценарий использования - «Актёра/режиссёра нет в базе данных». 14
Сценарий использования - «Фильмы определённого жанра и/или происхождения». 14
Сценарий использования - «Статистика по временному периоду». 15
Сценарий использования - «Статистика (актёры и режиссёры) по жанру». 15
Сценарий использования - «Статистика (сюжет) по жанру». 16
Сценарий использования - «Анализ всех сюжетов режиссёров». 17
Модель данных. 17
Описание тестируемого датасета. 17
NoSQL модель данных (Neo4j). 17
Описание модели данных. 17
Вычисление примерного объема данных. 18
Запросы. 19
SQL модель данных. 21
Описание модели данных. 21
Вычисление примерного объема данных. 21
Запросы. 23
Сравнение SQL и NoSQL. 24
Разработанное приложение. 24
Краткое описание. 24
Схема экранов приложения. 25
Использованные технологии. 25
Ссылки на Приложение. 25
Выводы. 26
Результаты. 26
Недостатки и пути для улучшения полученного решения. 26
Приложение. 26
Документация по сборке и развертыванию приложения. 26

Снимки экранов приложения. 27
Список использованных источников. 30

Введение.

Цель работы – создать удобное решение для хранения и анализа информации о фильмах.

Было решено разработать веб-приложение, которое позволит хранить в электронном виде всю информацию о фильмах, позволяя при этом удобно с ними взаимодействовать.

Качественные требования к решению.

Требуется разработать приложение с использованием СУБД Neo4j.

Сценарии использования.

Макет UI.

Макет доступен по ссылке - https://app.moqups.com/01jKHyQqhk/view/page/ad64222d5

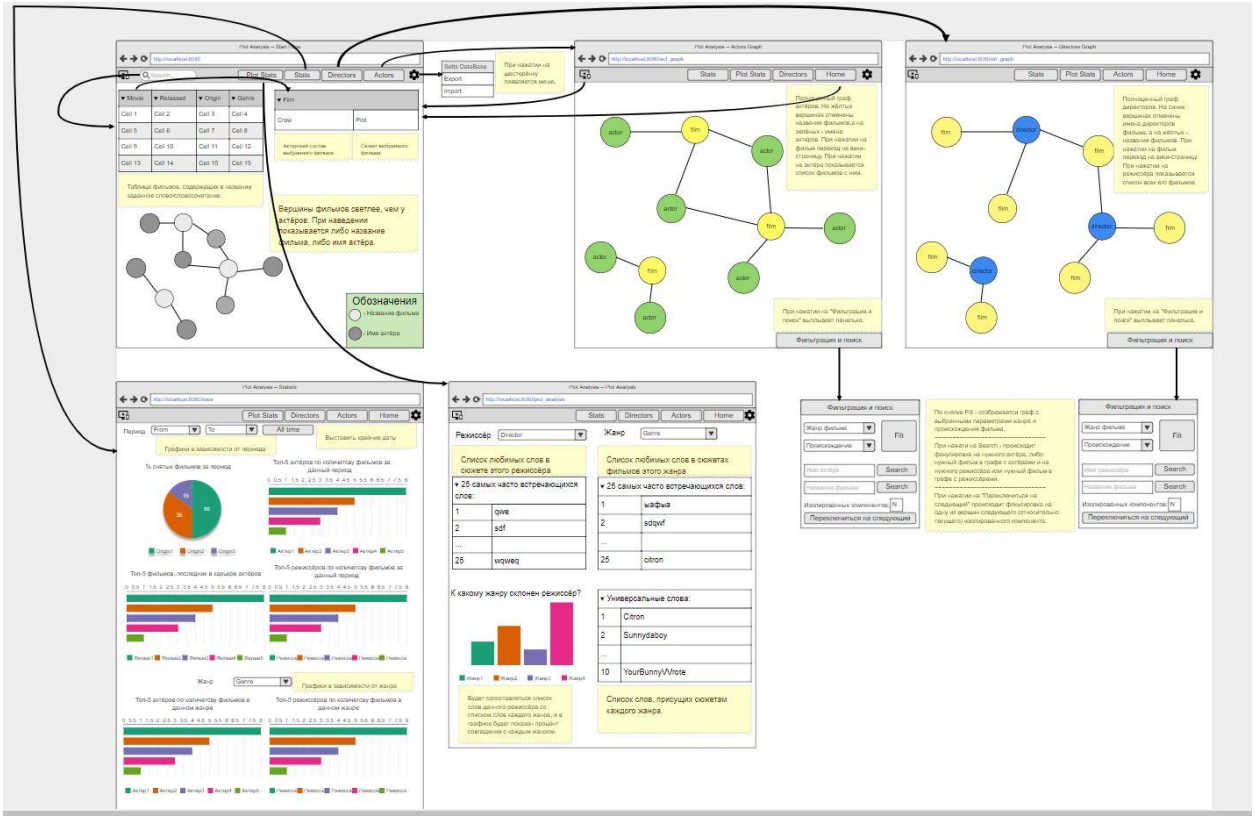

Рисунок 1 – Общий вид.

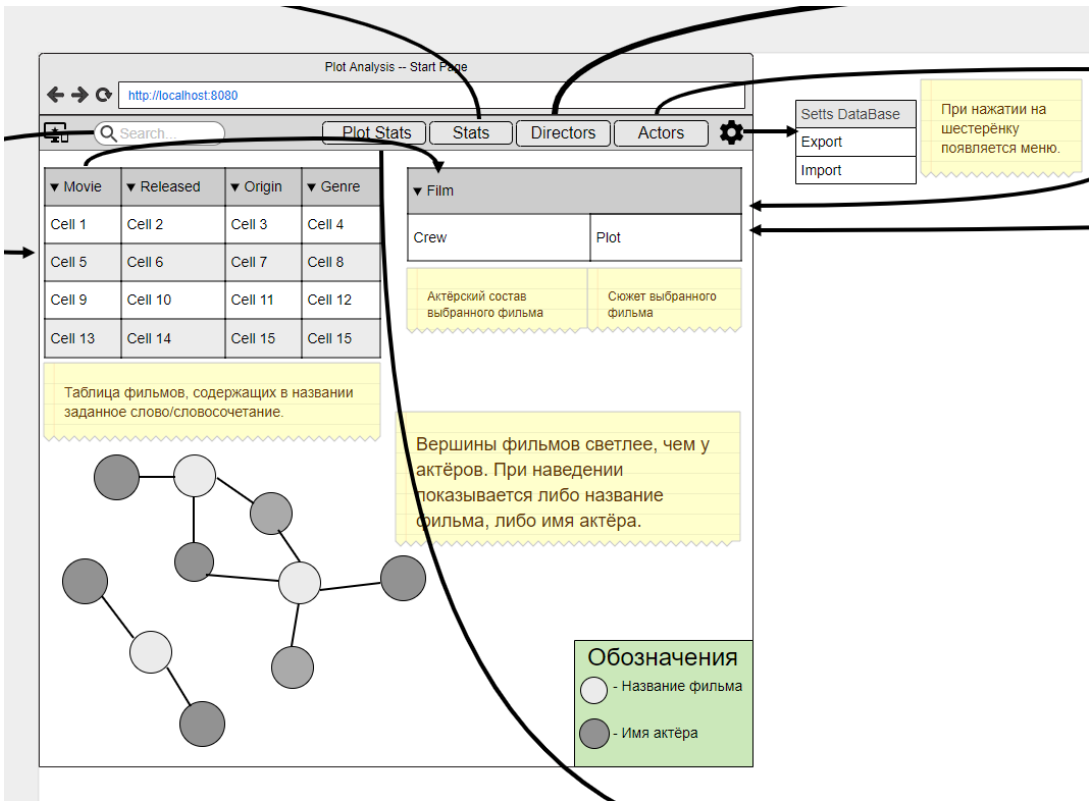

Рисунок 2 – Главная страница

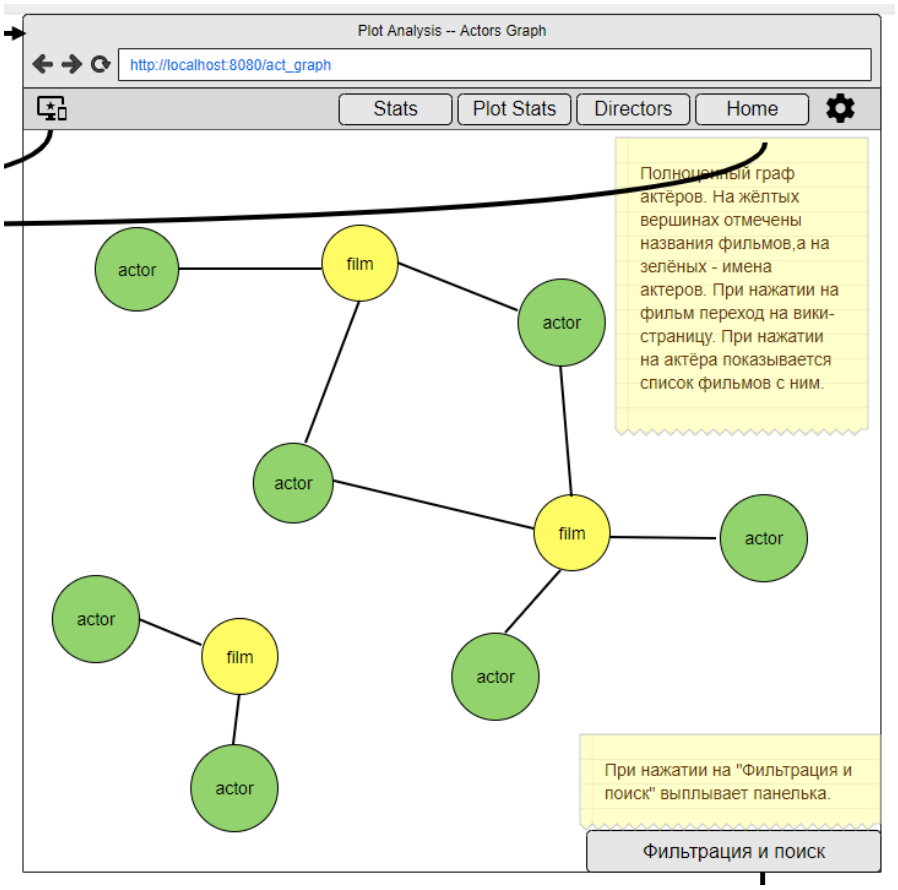

Рисунок 3 – Граф фильмы-актёры

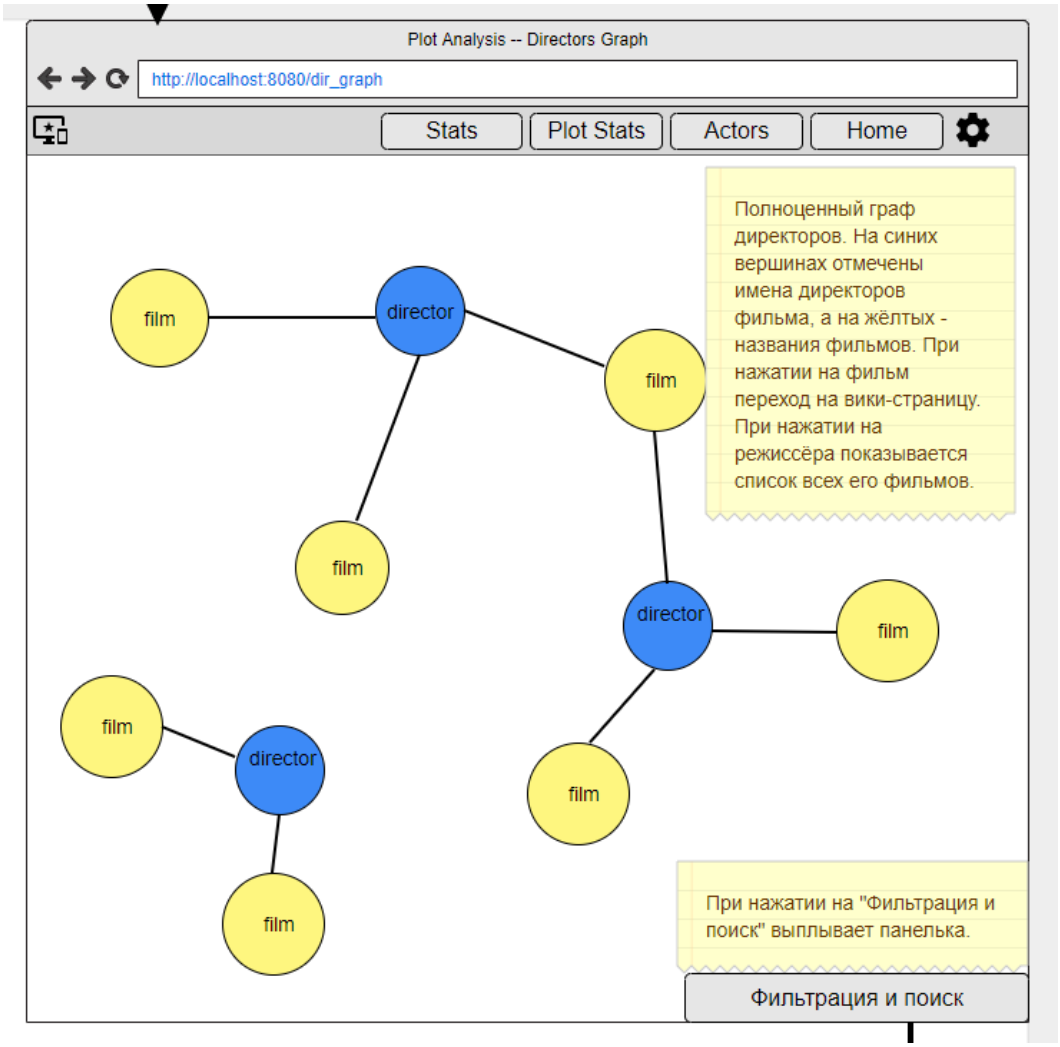

Рисунок 4 – Граф фильмы-режиссёры

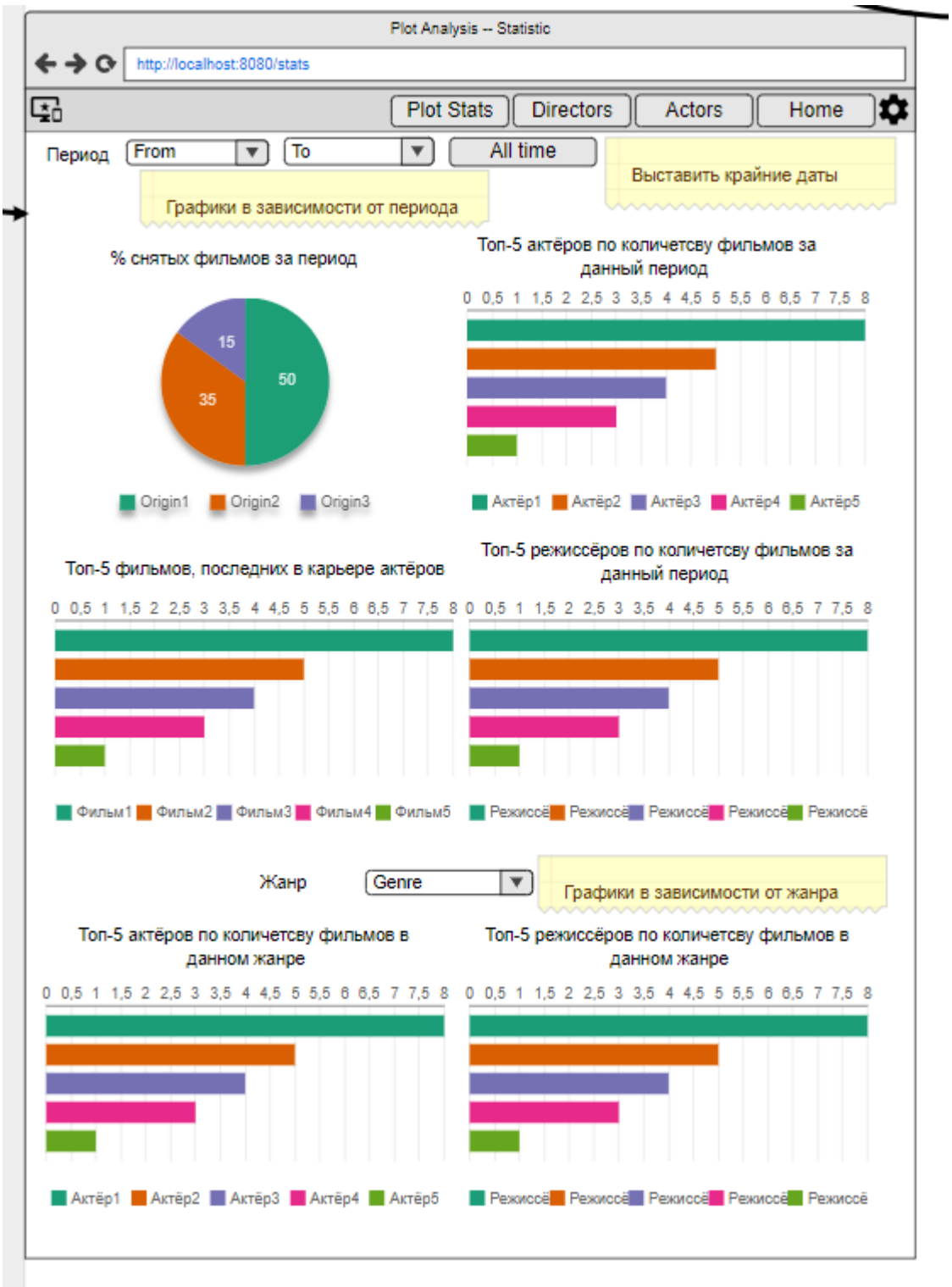

Рисунок 5 – Страница с общей статистикой

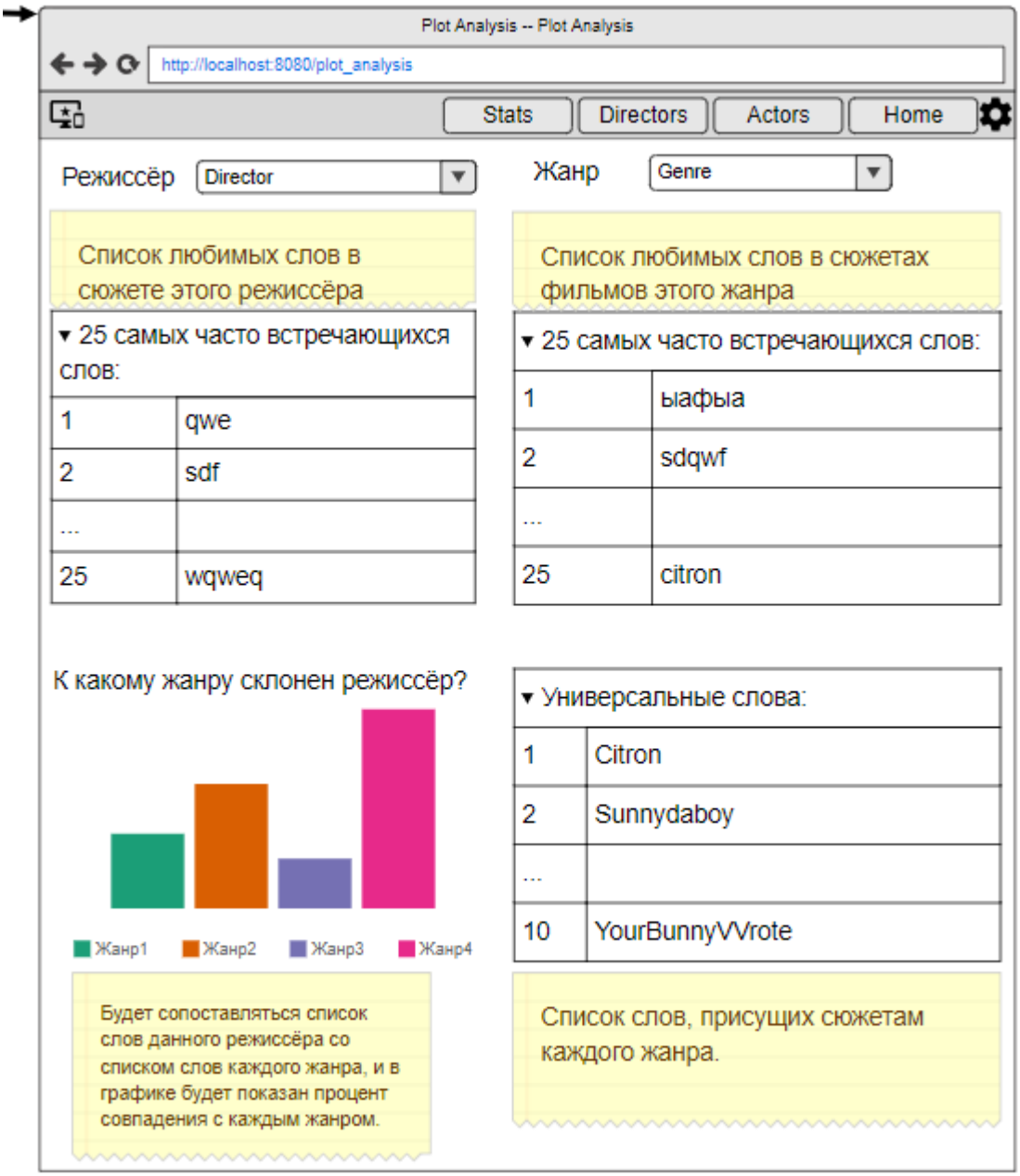

Рисунок 6 – Страница анализа сюжетов фильмов

Описание сценариев использования.

Сценарий использования - «Посмотреть сюжет/съёмочную группу фильма».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.

2. Пользователь вводит в поле «Search», находящееся в navbar’e, название (или часть названия) интересующего его фильма.
3. В таблице слева появляются все фильмы из базы данных, содержащие в своём названии строку, введённую пользователем.
4. Пользователь нажимает на нужное ему название в данном списке.
5. В таблице справа появляются съемочная группа (имена) и краткий сюжет фильма.
6. Пользователь читает предоставленную информацию.

Альтернативный сценарий:

- Фильма нет в базе данных.

Сценарий использования - «Фильма нет в базе данных».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь вводит в поле «Search», находящееся в navbar’e, название (или часть названия) интересующего его фильма.
3. Таблица становится пустой.
4. Пользователь покидает сайт либо использует другой сценарий.

Альтернативный сценарий:

- Посмотреть сюжет/съёмочную группу фильма.

Сценарий использования - «Фильмы с конкретным актёром».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Actors» в navbar’e.
3. Происходит переход на страницу с графом «Фильм-Актёры».
4. Пользователь нажимает на «Фильтрация и поиск» в правом нижнем углу.
5. Всплывает окошко.

6. Пользователь вводит имя актёра в поле «Имя актёра» и нажимает на соответствующий «Search» в этом окне.
7. На графе происходит фокусировка на данного актёра.
8. Пользователь видит наглядно в каких фильмах снимался данный актёр, но для удобства нажимает на кружок с актёром.
9. Появляется модальное окно со списком ссылок (на wiki) на все фильмы, в которых он принимал участие.

Альтернативный сценарий:

- Актёра/режиссёра нет в базе данных.

Сценарий использования - «Фильмы конкретного режиссёра».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Directors» в navbar’e.
3. Происходит переход на страницу с графом «Режиссёр-Фильмы».
4. Пользователь нажимает на «Фильтрация и поиск» в правом нижнем углу.
5. Всплывает окошко.
6. Пользователь вводит имя режиссёра в поле «Имя режиссёра» и нажимает на соответствующий «Search» в этом окне.
7. На графе происходит фокусировка на данного режиссёра.
8. Пользователь видит наглядно какие фильмы снимал данный режиссёр, но для удобства нажимает на кружок с режиссёром.
9. Появляется модальное окно со списком ссылок (на wiki) на все фильмы, которые он снимал.

Альтернативный сценарий:

- Актёра/режиссёра нет в базе данных.

Сценарий использования - «Актёра/режиссёра нет в базе данных».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Actors»/«Directors» в navbar’е.
3. Происходит переход на страницу с соответствующем графом.
4. Пользователь нажимает на «Фильтрация и поиск» в правом нижнем углу.
5. Всплывает окошко.
6. Пользователь вводит имя режиссёра/актёра в соответствующее поле и нажимает на соответствующий «Search» в этом окне.
7. С графом ничего не происходит. Появляется предупреждение что этого актёра/режиссёра нет в базе.
8. Пользователь покидает сайт либо использует другой сценарий.

Альтернативный сценарий:

- Фильмы с конкретным актёром.
- Фильмы конкретного режиссёра.

Сценарий использования - «Фильмы определённого жанра и/или происхождения».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Actors»/«Directors» в navbar’е, в зависимости от того, что его больше привлекает (актёры или режиссёры).
3. Происходит переход на страницу с соответствующем графом.
4. Пользователь нажимает на «Фильтрация и поиск» в правом нижнем углу.
5. Всплывает окошко.

6. Пользователь выбирает жанр в поле «Жанр фильма» и/или происхождение в поле «Происхождение» в этом окне.
7. Пользователь нажимает на кнопку «Filt» в этом окне.
8. Граф перестраивается с учётом данной информации.
9. Пользователь видит наглядно какие фильмы относятся к данному жанру и/или происхождению.
10. Пользователь нажимает на заинтересовавший его фильм.
11. Происходит переход на wiki-страницу этого фильма.

Альтернативный сценарий:

1. Пользователя ничего не заинтересовало.
2. Пользователь покидает сервис.

Сценарий использования - «Статистика по временному периоду».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Stats» в navbar’e.
3. Происходит переход на страницу со статистикой.
4. Пользователь выбирает временной период в полях «From» и «To» или нажимает на кнопку «All time».
5. Обновляются графики с определённой статистикой данного периода.
6. Пользователь смотрит статистику.

Альтернативный сценарий:

1. Пользователя интересует другой период.
2. Переход на шаг 4 сценария «Статистика по временному периоду».

Сценарий использования - «Статистика (актёры и режиссёры) по жанру».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Stats» в navbar’e.
3. Происходит переход на страницу со статистикой.

4. Пользователь выбирает жанр в поле «Genre».
5. Обновляются графики с определённой статистикой данного жанра.
6. Пользователь смотрит статистику.

Альтернативный сценарий:

- Статистика (сюжет) по жанру
-
    i. Пользователя интересует другой жанр.
-
    ii. Переход на шаг 4 сценария «Статистика (актёры и режиссёры) по жанру».

Сценарий использования - «Статистика (сюжет) по жанру».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Plot Stats» в navbar’e.
3. Происходит переход на страницу с анализом сюжетов.
4. Пользователь выбирает жанр в поле «Genre».
5. Обновляется таблица «25 самых часто встречающихся слов в сюжетах данного жанра».
6. Пользователь смотрит статистику в данной таблице, а также в таблице «Универсальные слова для сюжетов всех жанров».

Альтернативный сценарий:

- Статистика (актёры и режиссёры) по жанру.
-
    i. Пользователя интересует другой жанр.
-
    ii. Переход на шаг 4 сценария «Статистика (сюжет) по жанру».

Сценарий использования - «Анализ всех сюжетов режиссёров».

Действующее лицо: Пользователь.

Основной сценарий:

1. Пользователь зашёл на сайт.
2. Пользователь нажимает на кнопку «Plot Stats» в navbar’e.
3. Происходит переход на страницу с анализом сюжетов.
4. Пользователь выбирает режиссёра в поле «Director».
5. Обновляется таблица «25 самых часто встречающихся слов в сюжете режиссёра», а также график «К какому жанру склонен режиссёр».
6. Пользователь смотрит предоставленную статистику.

Альтернативный сценарий:

1. Пользователя интересует другой режиссёр.
2. Переход на шаг 4 сценария «Анализ сюжетов режиссёров».

Модель данных.

Описание тестируемого датасета.

Выбранный датасет содержит следующие поля:

- Год выпуска (Release Year)
- Название фильма (Title)
- Происхождение фильма (Origin/Ethnicity)
- Режиссеры (Director)
- Актерский состав (Cast)
- Жанр (Genre)
- Адрес страницы фильма на википедии (Wiki Page)
- Сюжет (Plot)

NoSQL модель данных (Neo4j).

Описание модели данных.

Необходимо хранить два типа узлов - фильмы (метка Movie) и люди (метка Person), которые участвовали в его создании (актеры и режиссеры).

Актеров и режиссеров будем разделять по типам связей (соответственно необходимо еще хранить и два типа связей - метки ACTED_IN и DIRECTED).

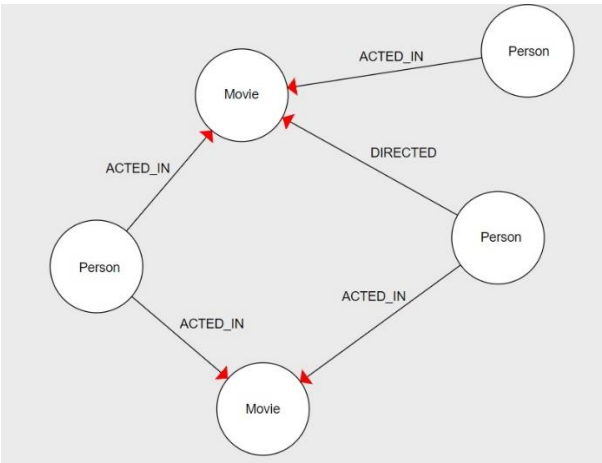

Рисунок 7 – Модель данных в Neo4j

Вычисление примерного объема данных.

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

- _id - Vid = 4b
- title - string => Vt = 2b * Nt, где Nt = 10 в среднем => Vt = 20b
- released - int => Vr = 4b
- origin - string => Vo = 2b * No, где No = 10 в среднем => Vo = 20b
- genre - string => Vg = 2b * Ng, где Ng = 10 в среднем => Vg = 20b
- plot - string => Vp = 2b * Np, где Np = 200 в среднем => Vp = 400b

Суммарный объем памяти, занимаемый одним фильмом (сущностью)

Vmovie = Vid + Vt + Vr + Vo + Vg + Vp = 468b

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

- _id - Vid = 4b
- name - string => Vn = 2b * Nn, где Nn = 20 в среднем => Vn = 40b

Суммарный объем памяти, занимаемый одним человеком

Vperson = Vid + Vn = 44b

Также необходимо хранить _id для каждого типа отношений (ACTED_IN и DIRECTED)

=> **Vact** = **Vdir** = **Vrel** = 4b

В среднем в каждом фильме снимается 5 актеров, каждый фильм создавали 2 режиссера.

Следовательно суммарный объем для хранения одного фильма будет равен: **V** = Vmovie + (Na + Nd) * (Vperson + Vrel), где

- Na - среднее количество актеров на фильм,
- Nd - среднее количество режиссеров на фильм.

V = Vmovie + (5 + 2) * (Vperson + Vrel) = 468B + 7 * 48B = 804b

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

V(N) = V * N = N * (Vmovie + (Na + Nd) * (Vperson + Vrel))
V(N) = N * (Vmovie + (5 + 2) * (Vperson + Vrel))= 804 * N

Датасет содержит почти что 35000 фильмов, следовательно хранение всех фильмов в Neo4j займет около 26Mb (датасет доступен для скачивания в размере 30Mb)

Запросы.

- Пример запроса на создание узла, хранящего в себе фильм

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, origin:'American', genre:'epic', plot:'A programmer is brought back to reason and reality when learning he was living in a program created by gigantic machines which make human birth artificial. In order to set humanity free, Neo will have to face many enemies by using  technologies and self-trust.'});
```

- Пример запроса на создание узла, хранящего в себе человека (актера, режиссера)

```
CREATE (Keanu:Person {name:'Keanu Reeves'});
```

- Пример запроса на создание отношения (ACTED_IN или DIRECTED) между человеком (актером, режиссером) и фильмом

```
(Keanu)-[:ACTED_IN]->(TheMatrix);
```

- Пример итогового запроса на добавление одного фильма в базу^

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, origin:'American', genre:'epic', plot:'A programmer is brought back to reason and reality when learning he was living in a program created by gigantic machines which make human birth artificial. In order to set humanity free, Neo will have to face many enemies by using technologies and self-trust.'})
CREATE (Keanu:Person {name:'Keanu Reeves'})
```

```
CREATE (Carrie:Person {name:'Carrie-Anne Moss'})
CREATE (Laurence:Person {name:'Laurence Fishburne'})
CREATE (Hugo:Person {name:'Hugo Weaving'})
CREATE (LillyW:Person {name:'Lilly Wachowski'})
CREATE (LanaW:Person {name:'Lana Wachowski'})
CREATE (JoelS:Person {name:'Joel Silver'})
CREATE
  (Keanu)-[:ACTED_IN]->(TheMatrix),
  (Carrie)-[:ACTED_IN]->(TheMatrix),
  (Laurence)-[:ACTED_IN]->(TheMatrix),
  (Hugo)-[:ACTED_IN]->(TheMatrix),
  (LillyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)
;
```

Таким образом формула количества запросов для одного фильма: QueryNum = 1 + 2 * (Na + Nd), где

- Na - среднее количество актеров на фильм,
- Nd - среднее количество режиссеров на фильм.

Количество запросов для **N** фильмов (пусть в среднем в каждом фильме снимается 5 актеров, каждый фильм создавали 2 режиссера): QueryNum(N) = N * (1 + 2 * (Na + Nd)) = 15 * N

Пускай в среднем в каждом фильме снимается 5 актеров, каждый фильм создавали 2 режиссера. Количество фильмов ~ 35000, тогда: QueryNum(35000) = 35000 * (1 + 2 * (5 + 2)) = 525 000 запросов на добавление в базу 35000 фильмов

- Пример запроса на выборку по свойствам

```
MATCH (m:Movie {origin: 'American', genre: 'epic'})
RETURN m
;
```

- Пример запроса на выборку по временному периоду

```
MATCH (m:Movie)
WHERE m.released >= 1999 AND m.released <= 2005
RETURN m
;
```

- Пример запроса на поиск изолированных групп

```
call algo.unionFind.stream(null,null,{})
yield nodeId, setId
return algo.getNodeById(nodeId).title as fragId, setId
order by setId, fragId
;
```

SQL модель данных.

Описание модели данных.

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

Пять таблиц. Основные - "Фильмы", "Актеры", "Режиссеры". Дополнительные, возникающие по правилам генерации отношений из-за связей сущностей Фильмы-Актеры и Фильмы-Режиссеры, представляющих собой связи многие-ко-многим - "Играет в", "Снимает".

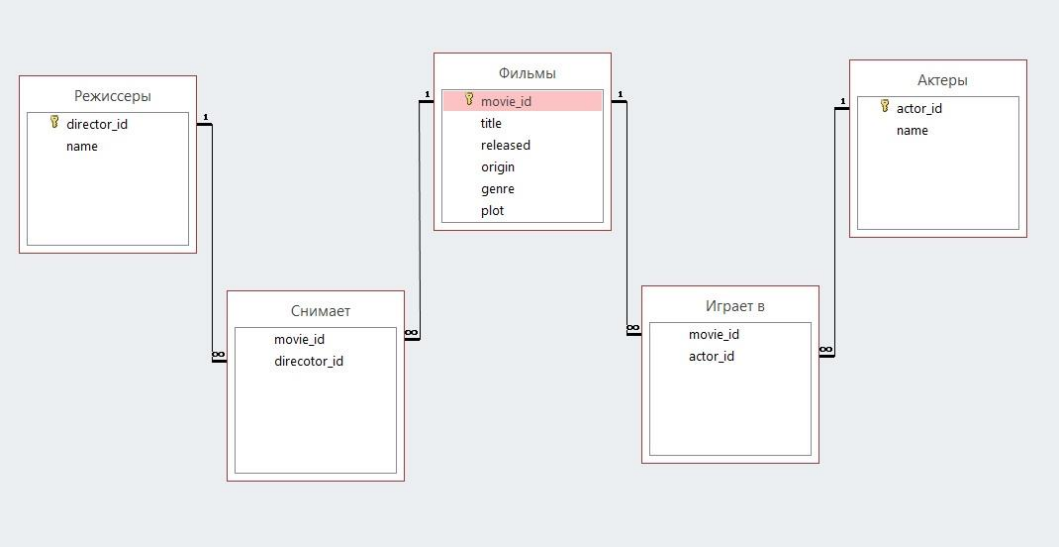

Рисунок 8 – Модель данных SQL

Вычисление примерного объема данных.

- Таблица "Фильмы"
  - movie_id - INT 4b
  - title - VARCHAR(N), по байту на CHAR N ~ 10
  - released - INT 4b
  - origin - VARCHAR(N), по байту на CHAR N ~ 10
  - genre - VARCHAR(N), по байту на CHAR N ~ 10

    - plot - VARCHAR(N), по байту на CHAR N ~ 200

      Итог: Vm = 238b на строку в таблице "Фильм"
- Таблица "Актеры"
    - actor_id - INT 4b
    - name - VARCHAR(N), по байту на CHAR N ~ 20

      Итог: Va = 24b на строку в таблице "Актеры"
- Таблица "Режиссеры"
    - director_id - INT 4b
    - name - VARCHAR(N), по байту на CHAR N ~ 20

      Итог: Vd = 24b на строку в таблице "Режиссеры"
- Таблица "Играет в"
    - movie_id - INT 4b
    - actor_id - INT 4b

      Итог: Vam = 8b на строку в таблице "Играет в"
- Таблица "Снимает"
    - movie_id - INT 4b
    - direcotr_id - INT 4b

      Итог: Vdm = 8b на строку в таблице "Снимает"

Возьмем те же средние показатели, что и при оценке объема данных в нереляционной модели (в среднем в каждом фильме снимается 5 актеров, каждый фильм создавали 2 режиссера).

Таким образом при количестве фильмов ~ 35000 получаем:

V = сумма объемов данных в каждой таблице

Va = Vd = Vad

Vam = Vdm = Vmm

V(N) = N * (Vm + (Na + Nd) * (Vad + Vmm)) , где

- Na - среднее количество актеров на фильм,
- Nd - среднее количество режиссеров на фильм,
- N - количество фильмов

V(N) = 462 * N

V(35000) = 462 * 35000 = 15.4Mb

Запросы.

- Пример запроса на добавление записи в таблицу "Фильм"

```
 INSERT INTO Фильмы (title, released, origin, genre, plot) values('The Matrix', 1999, 'American', 'epic', 'A programmer is brought back to reason and reality when learning he was living in a program created by gigantic machines which make human birth artificial. In order to set humanity free, Neo will have to face many enemies by using technologies and self-trust.');
```

- Пример запросов на добавление записей в таблицы "Актер", "Режиссер"

```
INSERT INTO Актеры (name) values('Quentin Tarantino');
INSERT INTO Режиссеры (name) values('Quentin Tarantino');
```

- Пример запросов на добавление записей в таблицы "Снимает", "Играет в"

```
 INSERT INTO Снимает (movie_id, director_id) values('Reservoir dogs', 'Quentin Tarantino');
 INSERT INTO 'Играет в' (movie_id, actor_id) values('Reservoir dogs', 'Quentin Tarantino');
```

- Пример итогового запроса на добавление одного фильма в базу

```
  INSERT INTO Фильмы (title, released, origin, genre, plot) values('The Matrix', 1999, 'American', 'epic', 'A programmer is brought back to reason and reality when learning he was living in a program created by gigantic machines which make human birth artificial. In order to set humanity free, Neo will have to face many enemies by  using technologies and self-trust.');
    INSERT INTO Актеры (name) values('Keanu Reeves');
    INSERT INTO "Играет в" (movie_id, actor_id) values('The Matrix', LAST_INSERT_ID());
    INSERT INTO Режиссеры (name) values('Lilly Wachowski');
    INSERT INTO "Снимает" (movie_id, actor_id) values('The Matrix', LAST_INSERT_ID());
```

Таким образом формула количества запросов для одного фильма точно такая же, как и в NoSQL модели:

QueryNum = 1 + 2 * (Na + Nd), где

- Na - среднее количество актеров на фильм,
- Nd - среднее количество режиссеров на фильм.

Количество запросов для **N** фильмов (пусть в среднем в каждом фильме снимается 5 актеров, каждый фильм создавали 2 режиссера):

QueryNum(N) = N * (1 + 2 * (Na + Nd)) = 15 * N

Т.е. 525 000 запросов на создание 35 000 фильмов

- Пример запроса на выборку по полям

```
SELECT *
FROM Фильмы
WHERE Фильмы.origin == 'American' AND Фильмы.genre == 'epic'
AND Фильмы.released >= 1999 AND Фильмы.released <= 2005
```

Сравнение SQL и NoSQL.

- В SQL реализации модели данных пришлось бы создавать дополнительные таблицы для связей, что увеличивает суммарное количество создаваемых таблиц.
- Данные в SQL модели будут занимать меньший объем (для 35k фильмов 15.4 Mb), по сравнению с данными в NoSQL модели (для 35k фильмов 26 Mb).
- Количество запросов для NoSQL модели данных никак не отличается от SQL
- NoSQL реализация будет выигрывать в структуре и скорости сложных графовых запросах, таких как, например, запрос на получение несвязных подграфов.

Разработанное приложение.

Краткое описание.

Веб-приложение, хранящее в себе фильмы, актёров, режиссёров и связи между ними, с возможностями анализа имеющейся информации о них и предоставления некоторой статистики.

Схема экранов приложения.

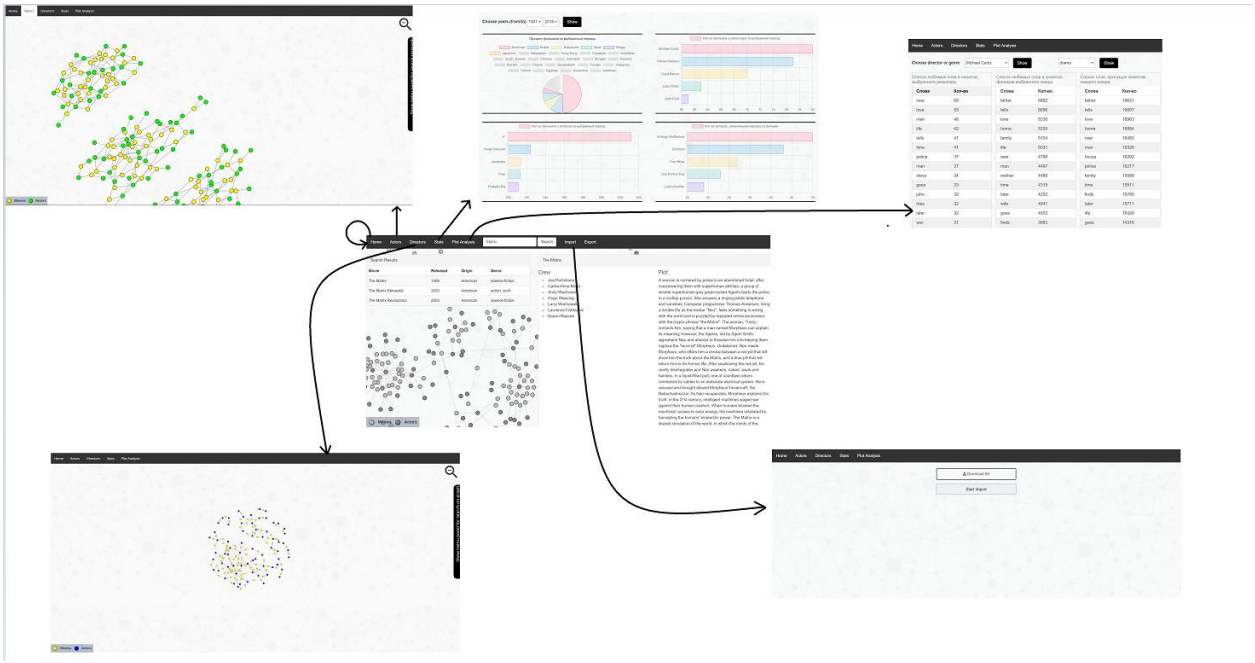

Рисунок 9 – Схема экранов приложения

Использованные технологии.

- Python 3.7
- Neo4j
- neo4j-python-driver - Neo4j Bolt Driver for Python
- Flask - Python microframework based on Werkzeug, Jinja 2 and good intentions.
- Neo4j-Server
- Frontend: jquery, bootstrap, d3.js, w3

Ссылки на Приложение.

Ссылка на github: https://github.com/moevm/nosql1h19-plot-anaysis

Выводы.

Результаты.

В ходе работы было разработано приложение для анализа информации о фильмах, актёрах и режиссёрах. В учебных целях были добавлены import и export датасетов.

Недостатки и пути для улучшения полученного решения.

На данный момент import больших датасетов может занимать длительное время. Одним из способов решения может быть использование другого алгоритма импорта, с написанием python скриптов для обработки датасета. А также анализ сюжетов всех фильмов также может занимать определённое время, т.к. сюжет каждого фильма приходится разбивать на слова, после чего уже работать с ними. Для ускорения можно перенести часть обработок на python, но это потребует больших затрат памяти.

Приложение.

Документация по сборке и развертыванию приложения.

1) Скачать проект из репозитория.
2) For Linux:

```
$ virtualenv neo4j-movies
$ source neo4j-movies/bin/activate
```

For Windows:

```
$ virtualenv neo4j-movies
$ neo4j-movies\Scripts\activate
```

3) Установить зависимости

```
(neo4j-movies)$ pip install -r requirements.txt
```

4) Запустить локальный Neo4j Server.
5) For Linux:

```
(neo4j-movies)$ export NEO4J_PASSWORD="my-password"
(neo4j-movies)$ python movies.py
```

For Windows:

```
(neo4j-movies)> SET NEO4J_PASSWORD=my-password
(neo4j-movies)> python movies.py
```

6) Перейти по http://localhost:8080.

Снимки экранов приложения.

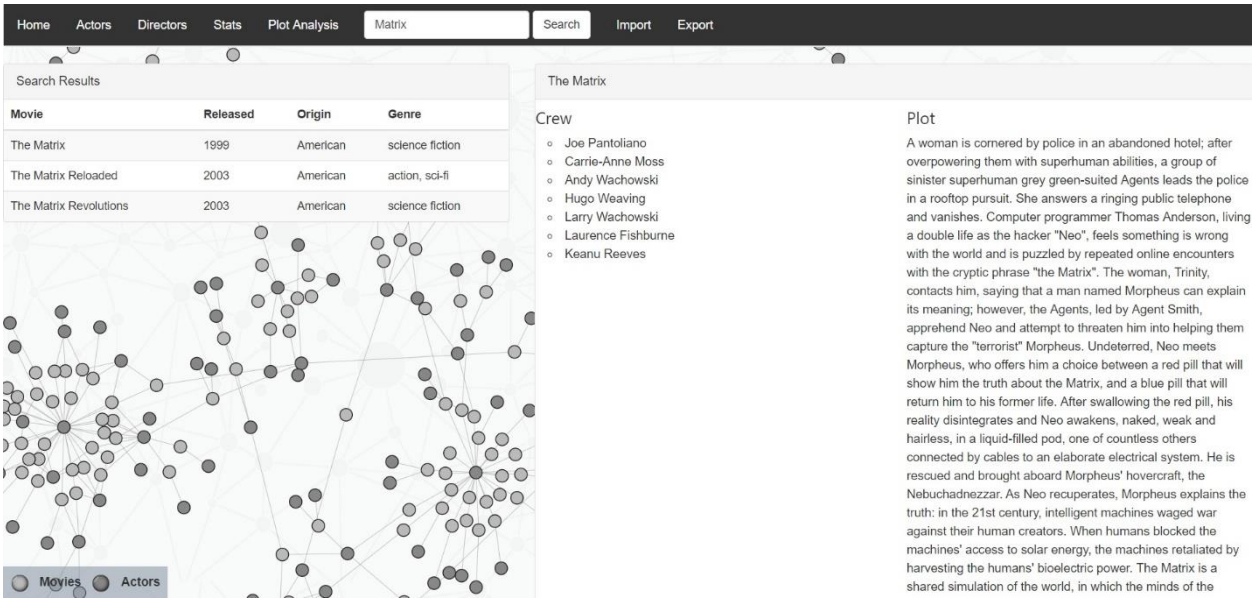

Рисунок 10 – Главная страница

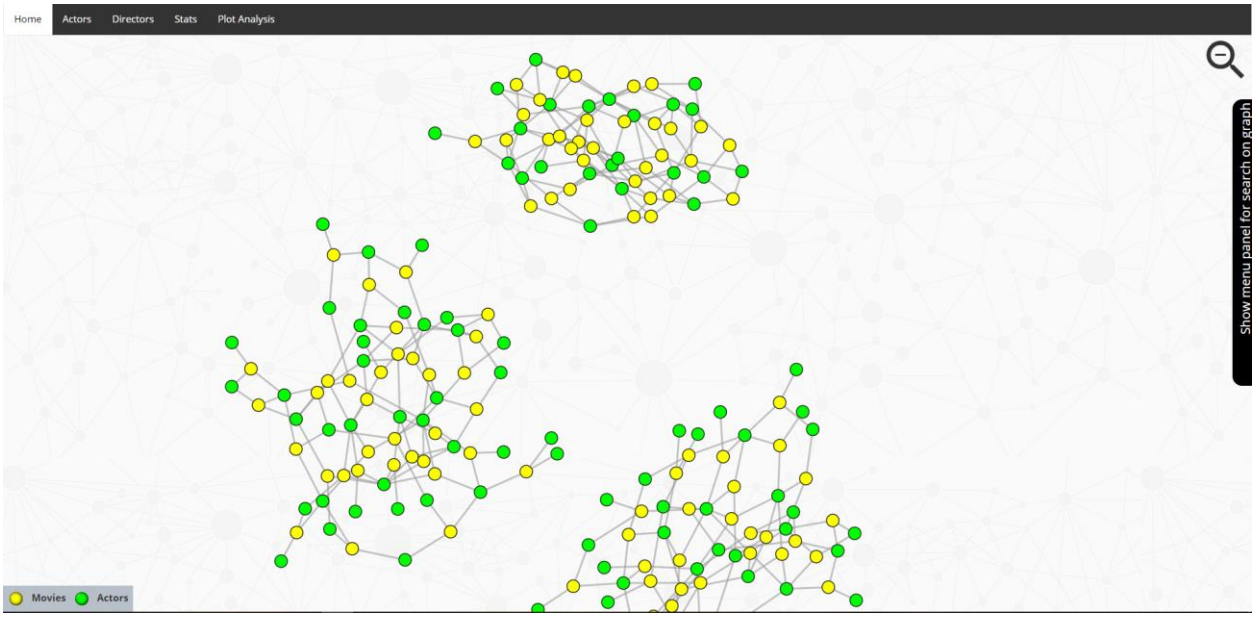

Рисунок 11 – Граф актёры-фильмы

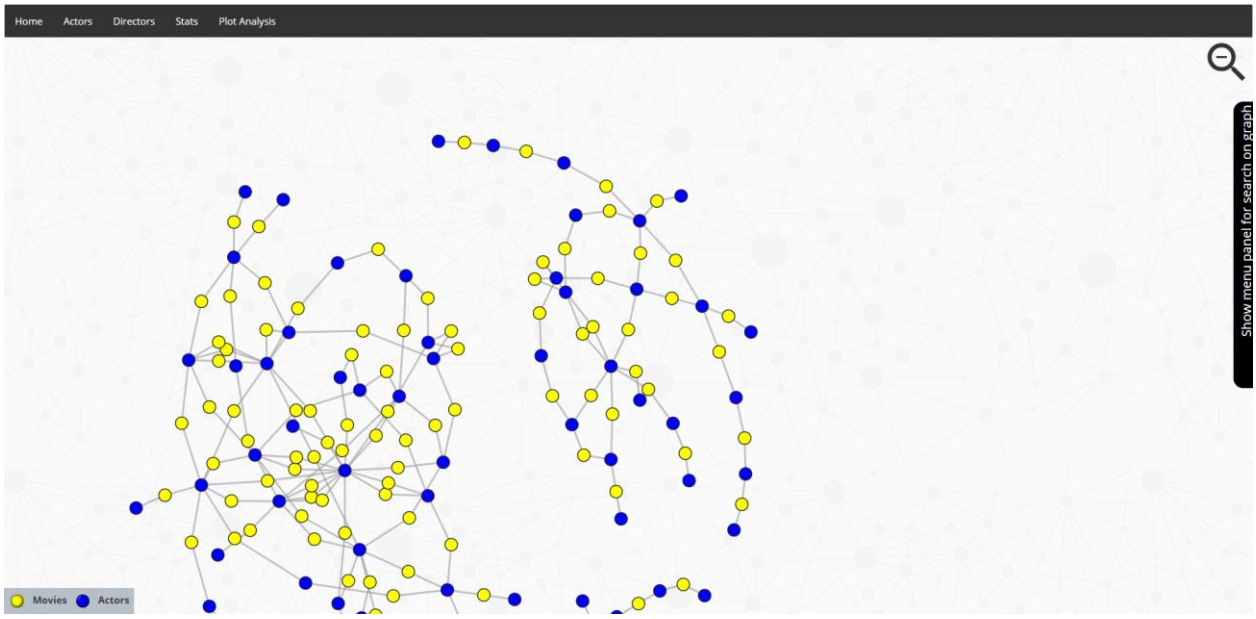

Рисунок 12 – Граф режиссёры-фильмы

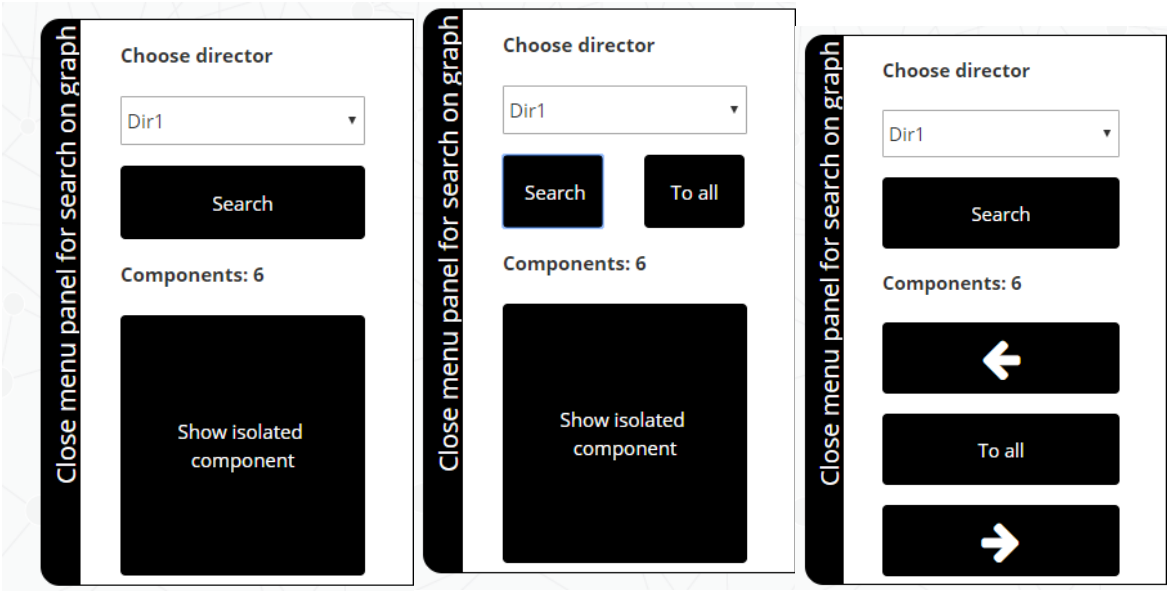

Рисунок 13 – Панель для поиска актеров/директоров и изолированных компонент

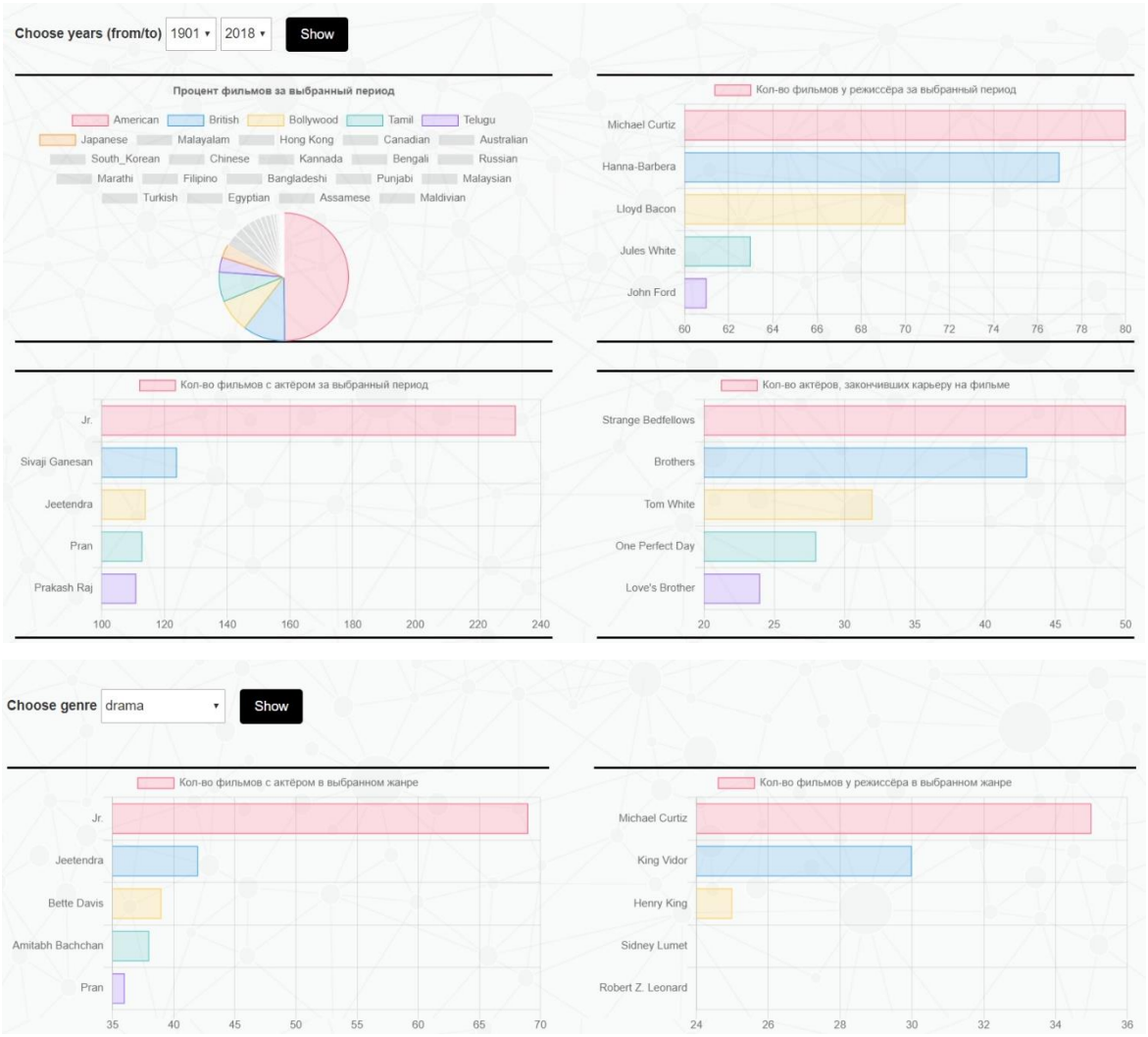

Рисунок 14 – Страница со статистикой

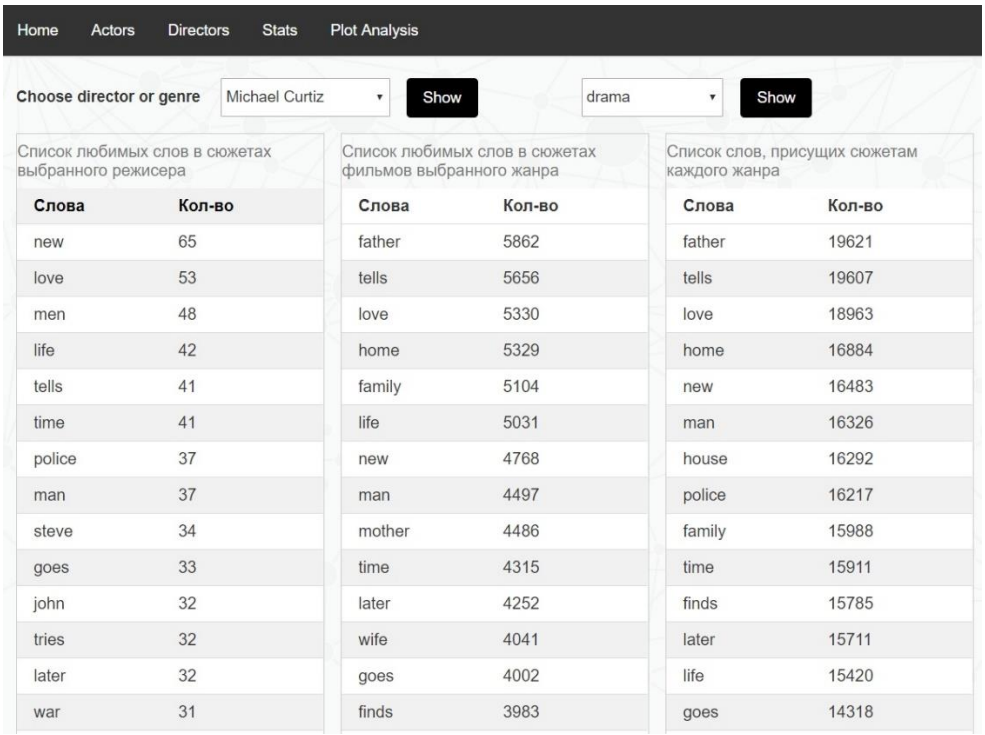

Рисунок 15 – Страница анализа сюжетов

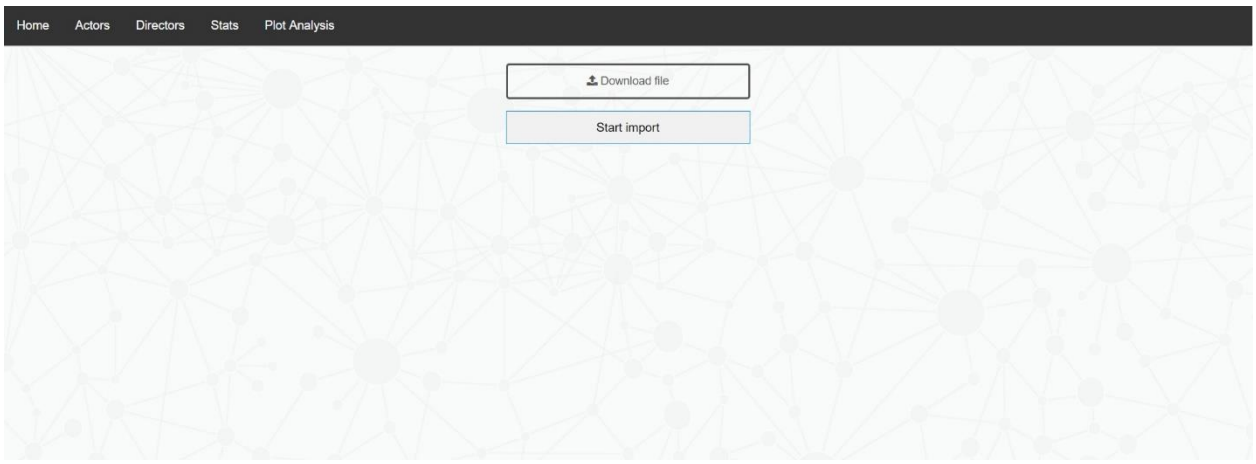

Рисунок 16 – Страница импорта

Список использованных источников.

1. Документация Neo4j: https://neo4j.com/docs/graph-algorithms/current/