



Министерство образования и науки 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 «Теоретическая информатика и компьютерные технологии»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ
РАБОТЕ
ПО КУРСУ: «База данных»
НА ТЕМУ:**

«Веб-приложение системы рекомендаций фильмов»

Студент ИУ9-52Б

Нгуен Т.Д.

(Ф.И.О)

(Подпись, дата)

Руководитель курсового проекта

Синявин А. В.

(Ф.И.О)

(Подпись, дата)

2021 г.

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)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-9
(Индекс)

_____ И.П. Иванов (И.О.Фамилия)

«___» _____ 2021 г.

З А Д А Н И Е

на выполнение курсовой работы

по дисциплине : «База данных»

Тема курсовой работы: Веб-приложение системы рекомендаций фильмов

Студент : Нгуен Тхань Дат ИУ9-626

График выполнения проекта: 25% к 4 нед., 50% к 8 нед., 75% к 11 нед., 100% к 14 нед.

1. Техническое задание

Этап 1: Определитесь с технологиями веб-разработки, выбрав систему управления базами данных и основной алгоритм, используемый в рекомендательных системах..

Этап 2: Разработка базы данных, создание запросов к базе данных для будущих работ.

Этап 3: Разработка пользовательских интерфейсов для веб-приложений с помощью React framework.

Этап 4: Реализация алгоритма рекомендации с подготовленной базой данных на сервере веб-приложений..

2. Оформление курсовой работы

2.1. Расчетно-пояснительная записка на 25 листах формата А4.

2.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) _____

Дата выдачи задания «___» _____ 2021 г.

Оглавление

I.	Введение	4
1.	Система рекомендаций фильмов	4
2.	Цель проекта	4
3.	Обзорный подход	4
4.	Использование технологий	5
5.	Примечание	5
II.	Методология	5
1.	Описание алгоритма Matrix Factorization	5
2.	Реализация алгоритма	8
III.	Обзор пользовательских интерфейсов	9
1.	Описание всех функций пользовательского интерфейса	9
2.	Особенности пользовательского интерфейса	10
IV.	Дизайн и характеристики сервера системы	14
1.	Описать зачем нужен сервер и как он работает	14
2.	Технологии, используемые для разработки сервера	15
V.	Описание базы данных и запросы к ней.	17
1.	Реляционная база данных	17
2.	Обработка и преобразование базы данных	17
3.	Некоторые запросы к базе данных для каждого отдельного действия пользователя	26
3.1.	Запрос на отдельный фильм	26
3.2.	Поиск по списку фильмов	27
3.3.	Поиск знаменитостей, связанных с отдельным фильмом	28
3.4.	Обработка некоторых основных действий пользователя	29
VI.	Заключение и дальнейшая работа	34
VII.	Список литературы	35

I. Введение

1. Система рекомендаций фильмов

Этот проект представляет собой систему рекомендаций фильмов, которая с помощью математических уравнений дает список фильмов, которые пользователю может понравиться смотреть. Система не только рекомендует, но также помогает пользователям смотреть трейлер любого фильма, который они ищут, а также предоставляет много информации, такой как рейтинг IMDB, жанры, обзоры, актеры и т. Д. смотреть фильм, в конечном итоге спрашивать друзей и семью о предложениях, цель этого приложения - предоставить пользователю выбор фильмов, которые они могут захотеть посмотреть. Рекомендации по фильмам составляются с помощью сложных математических уравнений, чтобы попытаться определить, какие фильмы, скорее всего, понравятся пользователю, и дать им высокую оценку. Проект представляет собой приложение с простой навигацией, которое позволит пользователю искать фильмы, оценивать фильмы и получать рекомендации по фильмам на основе определенных критериев.

2. Цель проекта

Цель этого проекта - разработать систему рекомендаций, которая может узнать о своих пользователях и предоставить им набор фильмов, которые им, скорее всего, понравятся. Кроме того, помогите пользователям легче находить новые фильмы и сохранять список своих (не) любимых фильмов. Цель состоит в том, чтобы предоставить пользователю как пользовательские рекомендации, так и рекомендации на основе фильмов. Цель состоит в том, чтобы иметь приложение с простой навигацией, позволяющее пользователям легко перемещаться.

3. Обзорный подход

Этот проект будет основан на Javascript, поэтому большая часть вычислений будет производиться на Javascript. Он будет использовать SQL для хранения и извлечения данных. The front-end (или пользовательские интерфейсы) разработан с использованием React[1], а the back-end (или сервер) - с помощью Express[2]. Обе они являются популярными библиотеками для веб-разработки на Node.JS. Используемая система управления базами данных (СУБД) - SQL Server 2019. Алгоритм, используемый для рекомендательной системы под названием Matrix Factorization, который представляет собой класс алгоритмов Collaborative Filtering(CF), используемых в рекомендательных системах.

4. Использование технологий

Чтобы облегчить пользователю поиск фильмов, система позволяет смотреть трейлеры любых фильмов напрямую, без перехода на новую страницу. С помощью Youtube API[3] наша система получает видео высокого качества (почти с разрешением 1440p или 1080p) из базы данных Youtube и воспроизводит их без перенаправления на страницу Youtube. Не только трейлеры к фильмам, афиши и описания фильмов имеют решающее значение. Поблагодарить themoviedb[4], которые предоставили большое количество качественных постеров с фильмами через свой API. И последнее, но не менее важное: нам нужна вся информация о фильмах, такая как название, актеры, жанры, рейтинги ... Особая благодарность IMDb Corp.[5] для предоставленной всей этой информации точной и достоверной.

5. Примечание

В условиях ограничений по времени и объему памяти оборудования, сохранение информации всех созданных фильмов невозможно. Таким образом, в этом проекте сохраняется информация только о фильмах, созданных за последние 10 лет (2010-2020).

II. Методология

1. Описание алгоритма Matrix Factorization

- Collaborative filtering[6] - это метод, используемый рекомендательными системами. Этот метод автоматического прогнозирования (filtering) интересов пользователя путем сбора информации о предпочтениях или вкусовых предпочтениях многих пользователей (Collaborating).
- Matrix Factorization - это класс алгоритмов Collaborative filtering. Этот метод стал широко известен благодаря своей эффективности, как сообщил Simon Funk[7].
- Matrix Factorization - это способ создания скрытых функций при умножении двух разных типов сущностей. Collaborative filtering - это применение матричной факторизации для определения взаимосвязи между «элементами» и «пользователями». В нашей задаче «элементы» будут известны как «фильмы». С вводом оценок пользователей по элементам мы хотели бы предсказать, как пользователи будут оценивать элементы, чтобы пользователи могли получить рекомендацию на основе прогноза.

- Посмотрим на пример, у нас есть таблица рейтинга клиентов, состоящая из 5 пользователей и 5 фильмов, а рейтинги являются целыми числами от 1 до 5, матрица представлена в таблице ниже.

	Movie1	Movie2	Movie3	Movie4	Movie5
U1		5	4	2	1
U2	1			5	3
U3	1	4	4	1	
U4			2		2
U5	3	1	1		

Table1-Users' ratings table on movie

- Поскольку не каждый пользователь ставит оценки всем фильмам, в матрице много пропущенных значений, что приводит к разреженной матрице. Следовательно, нулевые значения, не предоставленные пользователями, будут заполнены 0, так что заполненные значения будут предоставлены для умножения. С помощью матричной факторизации мы можем обнаружить эти скрытые особенности, чтобы сделать прогноз на основе рейтинга с учетом сходства предпочтений и взаимодействий пользователей.
- С момента первоначальной работы Simon Funk[7] было предложено множество подходов к матричной факторизации. Такой как Funk MF[8], SVD++[9], Asymmetric SVD++[10], Hybrid MF[11], Deep learning MF[12], ... В этом проекте я выбрал очень простой метод в поле машинного обучения с именем gradient descent[13], цель которого - найти локальный минимум разницы.
- Сначала определен набор пользователей (U), элементов (D), $R = |U|$ и $|D|$. Матрица $|U| * |D|$ включает в себя все оценки, выставленные пользователями. Цель состоит в том, чтобы обнаружить K скрытых функций. Учитывая ввод двух матриц-матриц P ($|U| * k$) и Q ($|D| * k$), он сгенерирует результат продукта R. Матрица P представляет связь между пользователем и функциями, а матрица Q представляет связь между элементом и функциями. Мы можем получить прогноз рейтинга элемента путем вычисления скалярного произведения двух векторов, соответствующих P_i и P_j .

$$R \approx P * Q^T = \tilde{R}$$

$$\tilde{R}_{ij} = P_i^T Q_j = \sum_{k=1} P_{ik} Q_{kj}$$

- Мы определили "error" функцию, которая представляет собой разницу между реальным значением и прогнозируемым значением.

$$E_{ij}^2 = (R_{ij} - \tilde{R}_{ij})^2 = (R_{ij} - \sum_{k=1} P_{ik} Q_{kj})^2$$

- Наша задача - обновить матрицы P, Q, чтобы минимизировать значение “error” функции. Градиент может минимизировать это значение, и поэтому мы дифференцируем приведенное выше уравнение по отношению к этим двум переменным отдельно.

$$\frac{\partial}{\partial P_{ik}} E_{ij}^2 = -2(R_{ij} - \tilde{R}_{ij})(Q_{kj}) = -2E_{ij}Q_{kj}$$

$$\frac{\partial}{\partial Q_{kj}} E_{ij}^2 = -2(R_{ij} - \tilde{R}_{ij})(P_{ik}) = -2E_{ij}P_{ik}$$

- Из градиента математическая формула может быть обновлена как для P_{ik} , так и для Q_{kj} . α - это шаг для достижения минимума при вычислении градиента, а α обычно устанавливается с небольшим значением.

$$P'_{ik} = P_{ik} + \alpha \frac{\partial}{\partial P_{ik}} E_{ij}^2 = P_{ik} + 2\alpha E_{ij}Q_{kj}$$

$$Q'_{kj} = Q_{kj} + \alpha \frac{\partial}{\partial Q_{kj}} E_{ij}^2 = Q_{kj} + 2\alpha E_{ij}P_{ik}$$

- Из приведенного выше уравнения, P'_{ik} и Q'_{kj} могут обновляться с помощью итераций до тех пор, значение “error” достигнет своего минимума.

$$E = \sum E_{ij}^2 = \sum (R_{ij} - \sum_{k=1} P_{ik}Q_{kj})^2$$

2. Реализация алгоритма

- Код из проекта:

```
const matrix_factorization = (R,K,N,M) => {
  var P = createRandomMatrix(N,K);
  var Q = createRandomMatrix(M,K);

  var alpha = 0.02;
  var beta = 0.002;
  var epsilon = 0.0001;

  Q = T(Q,M,K);
  while (true) {
    for (let i=0;i<N;i++)
      for (let j=0;j<M;j++)
        if (R[i][j] > 0) {
          var Eij = R[i][j];
          for (let t=0;t<K;t++) Eij -= P[i][t]*Q[t][j];
          for (let t=0;t<K;t++) {
            P[i][t] += alpha * (2 * Eij * Q[t][j] - beta * P[i][t]);
            Q[t][j] += alpha * (2 * Eij * P[i][t] - beta * Q[t][j]);
          }
        }
    var e = 0;
    for (let i=0;i<N;i++)
      for (let j=0;j<M;j++)
        if (R[i][j]>0) {
          var curr = R[i][j];
          for (let t=0;t<K;t++) curr -= P[i][t]*Q[t][j]
          e = e + curr*curr;
        }
    if (e<epsilon) break;
  }

  return matrix_mul(P,Q,K,N,M);
}
```

- Посмотрим небольшой пример проверки исправления алгоритма.
Приведены данные рейтинговых оценок 3 пользователей 6 фильмов
(оценка от 0 до 10, оценка 0 означает, что пользователи еще не знали этот фильм)

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	8	7	7	0	0	6
User 2	0	6	7	3	7	0
User 3	6	0	4	2	9	4

Table 1: Initial rating point

- Прогнозируемая матрица генерируется ниже. Матрица прогнозов имеет аналогичный результат с истинными значениями, а оценки 0 заменяются прогнозом, основанным на аналогичных предпочтениях пользователей в отношении фильмов.

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie 6
User 1	8.001952	7.000869	6.99874	3.91094	8.87168	5.99512
User 2	7.13439	5.99828	6.99650	3.00044	7.00008	3.92563
User 3	5.99604	5.21907	4.00086	1.99841	8.99678	4.00256

Table 2: Result after executed algorithm

- Так, в примере, User1 не знал(а) Movie5, на основе предсказания результата нашего алгоритма, ему(ей) может понравиться этот фильм с довольно высокой оценкой ($\approx 8,87/10$).
- **Примечание:** Поскольку в алгоритме используется метод машинного обучения, это означает использование более количества данных, получение более точного результата.

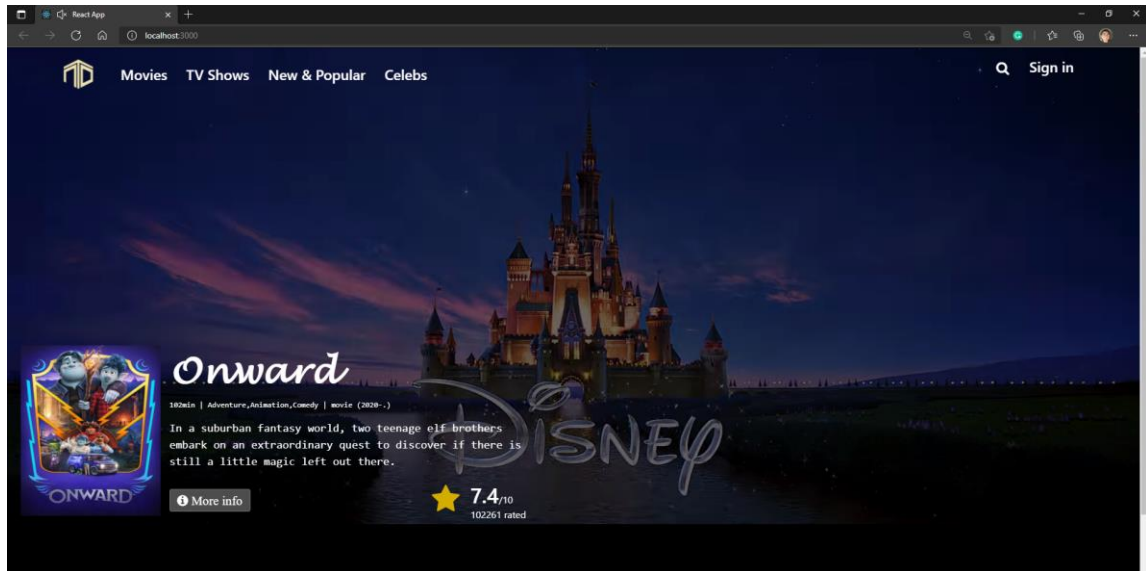
III. Обзор пользовательских интерфейсов

1. Описание всех функций пользовательского интерфейса

- Пользовательские интерфейсы (UI) нашей системы, разработанные с использованием React. По сути, UI содержит несколько компонентов, отображающих информацию. По сути, компоненты - это многоразовые пользовательские интерфейсы, которые позволяют разделить приложение на отдельные блоки, которые действуют независимо друг от друга. Компоненты принимают произвольный ввод с данными (props) и возвращают элемент React, чтобы объявить, что должно появиться на экране. Они могут взаимодействовать с другими компонентами через props для создания сложного пользовательского интерфейса.
- Пользователи могут войти в систему, если у них была зарегистрированная учетная запись, в противном случае они могут зарегистрироваться для новой.
- Пользователи могут искать любой фильм по ключевому слову. После того, как пользователи выбрали фильм, система покажет им всю информацию о нем. Пользователь может поставить оценку этому фильму или добавить его в свой сохраненный список. Система использует эту оценку для рекомендации новых фильмов, которые отображаются на главной странице.

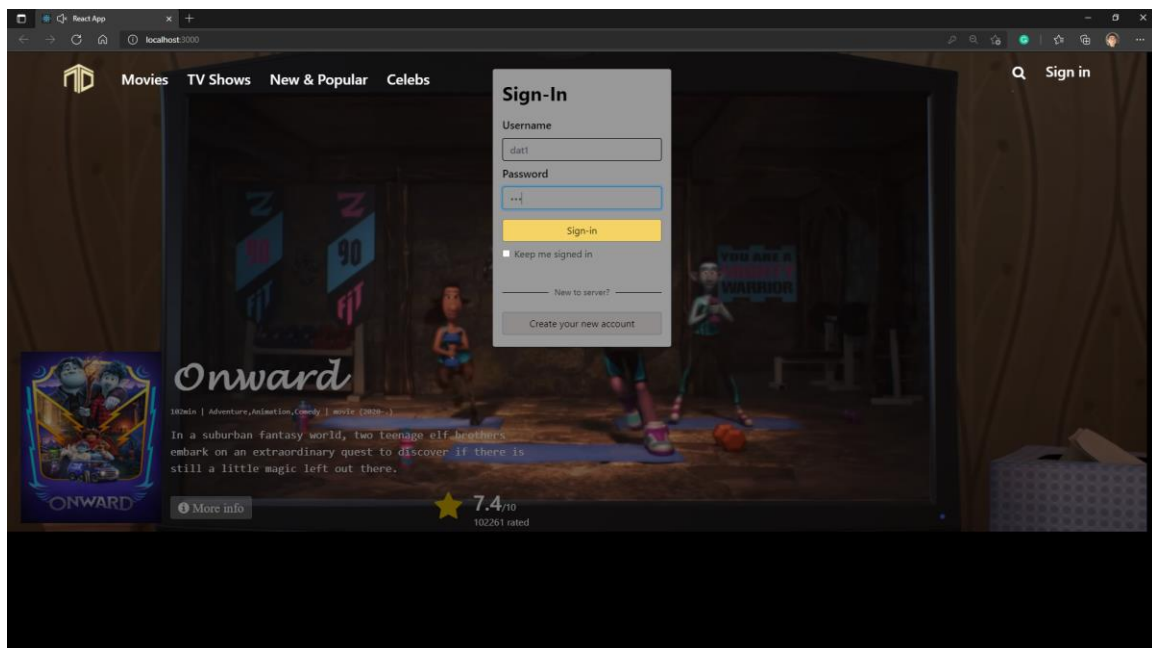
2. Особенности пользовательского интерфейса

- Главная страница, когда пользователь не авторизовался. Один из самых популярных фильмов показал свою информацию и трейлер. Пользователь может войти в свою учетную запись, нажав кнопку «Sign in» в правом верхнем углу.

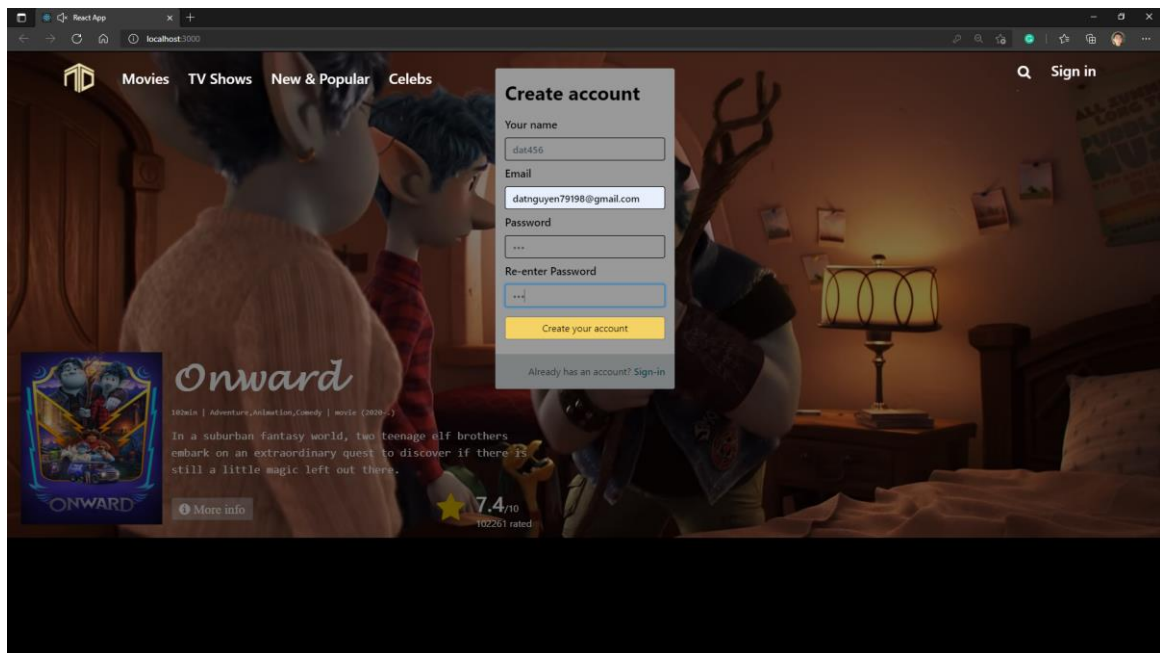


Picture 1: Main page

- Интерфейс входа и регистрации. Если у пользователей еще нет учетной записи, они могут зарегистрировать новую.

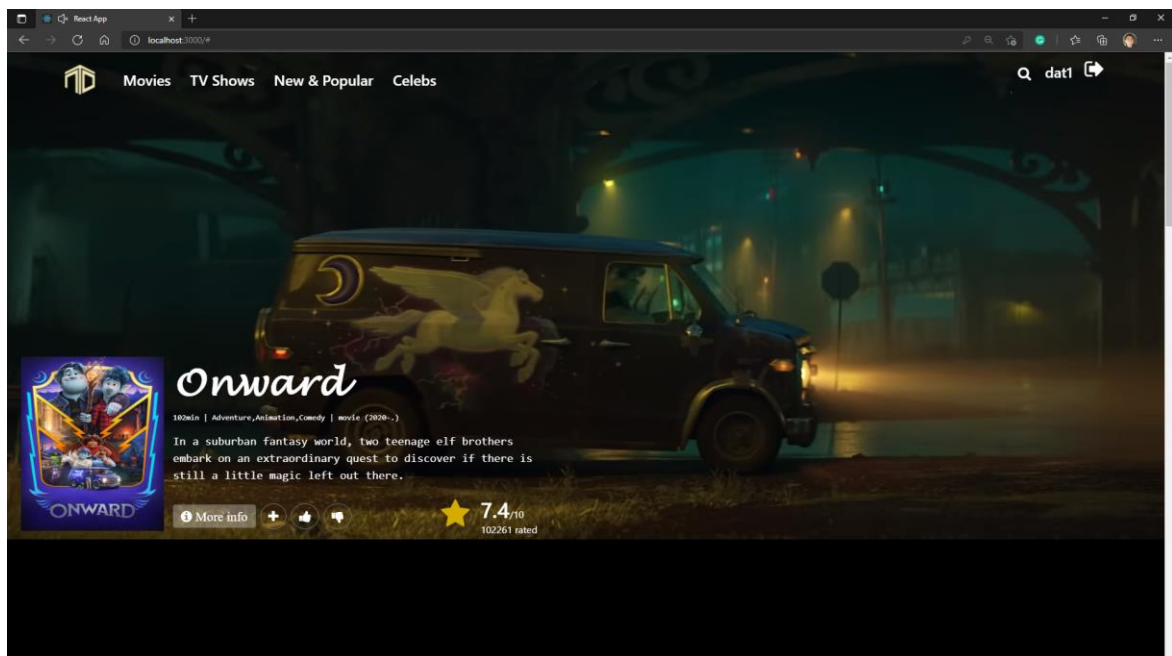


Picture 2: Sign in function

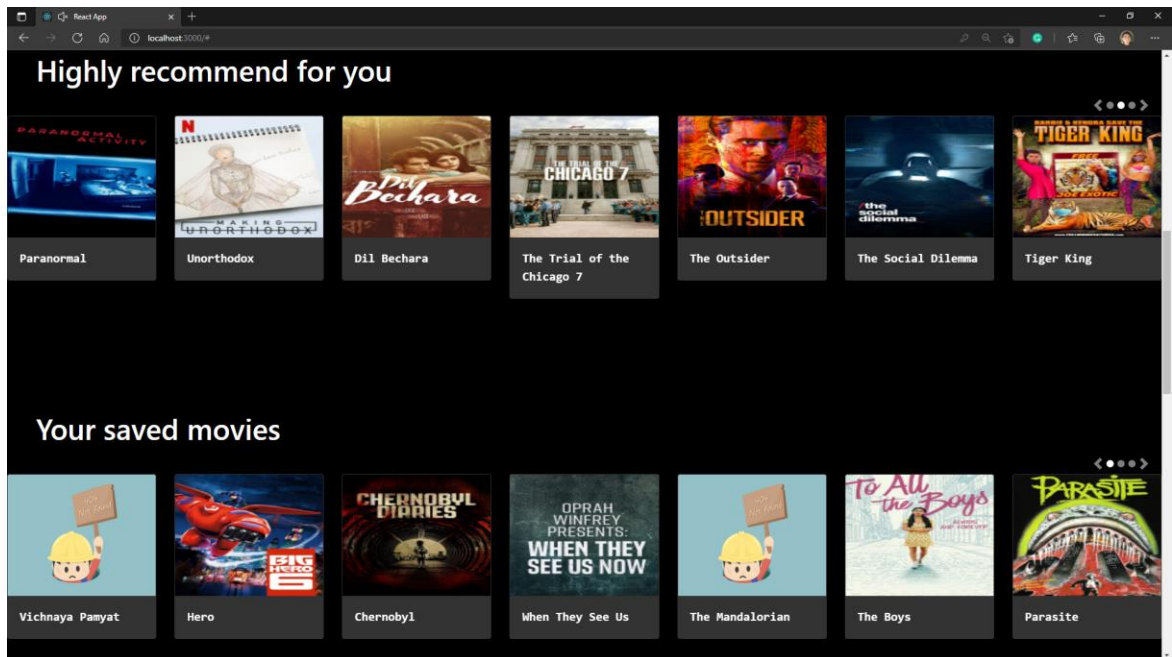


Picture 3: Sign up function

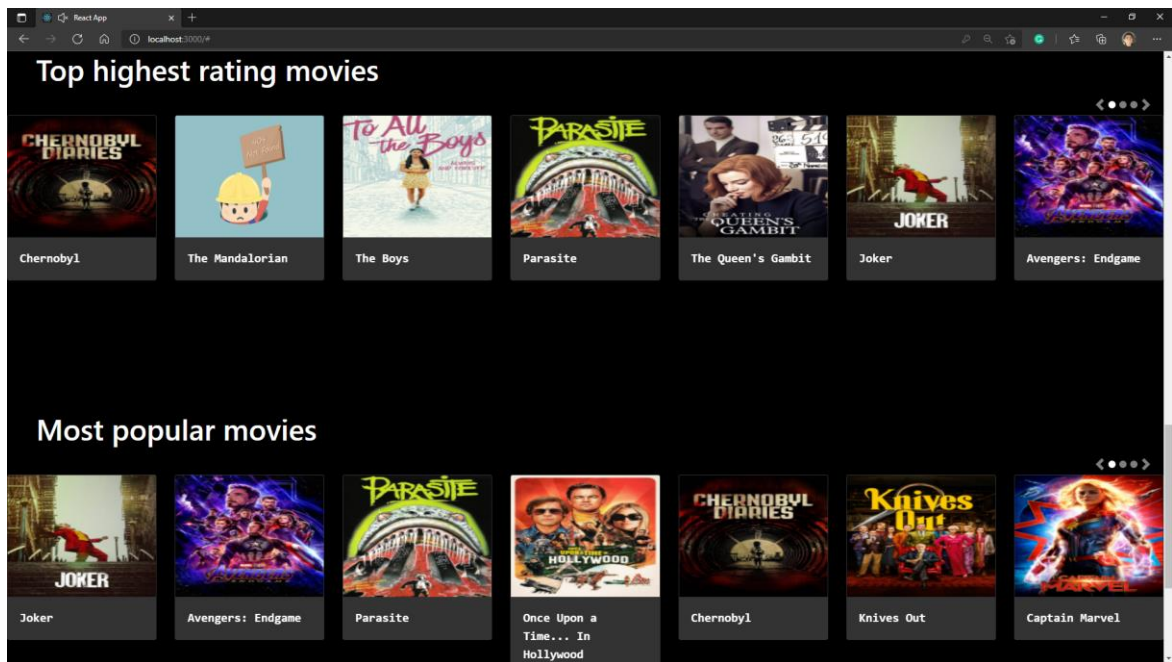
- Главная страница после того, пользователи войдут в свою учетную запись. Они также могут выйти, нажав кнопку в правом верхнем углу. Под фильмом на верхней главной странице находятся списки фильмов, рекомендованных сервером для отдельного пользователя (3 списка, в каждом списке 21 рекомендованный фильм) и их список сохраненных фильмов.



Picture 4: Main page after user signed in successfully

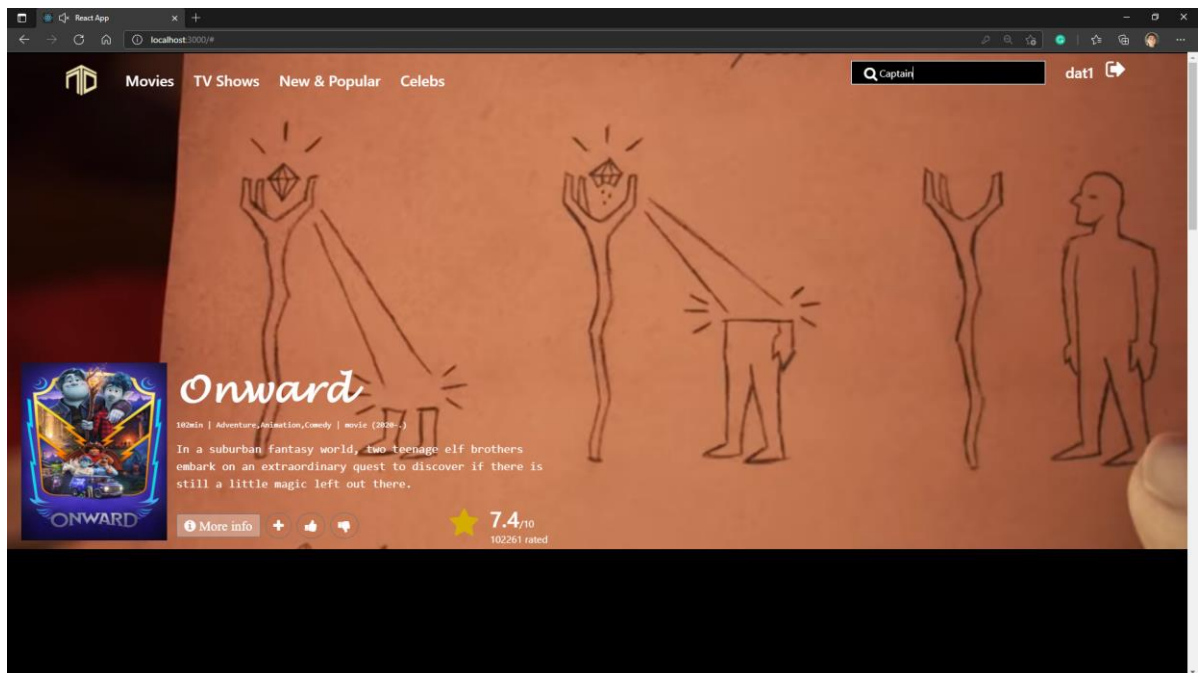


Picture 5: Recommended movies for user and their saved movies

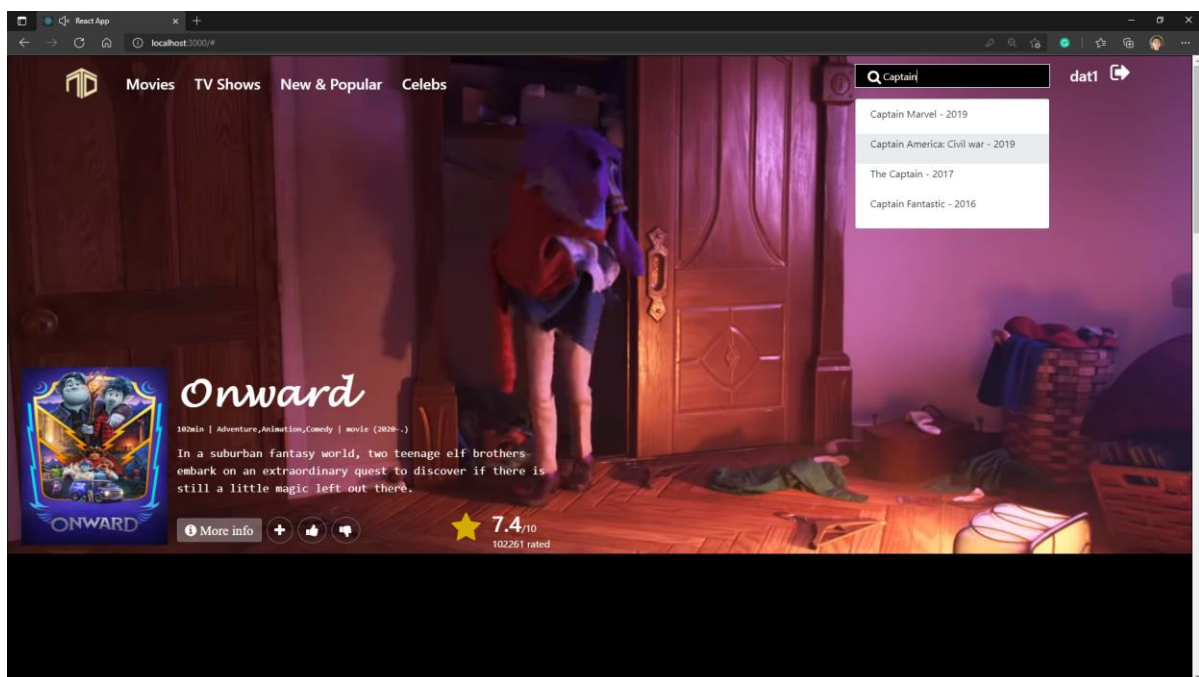


Picture 6: Recommended movies based on rating and popularity

- Система не только рекомендует, но и помогает пользователю найти фильм, который он ищет. Если щелкнуть символ “Лупа” вверху и ввести полное имя или ключевое слово, система выдаст результаты, как показано ниже.

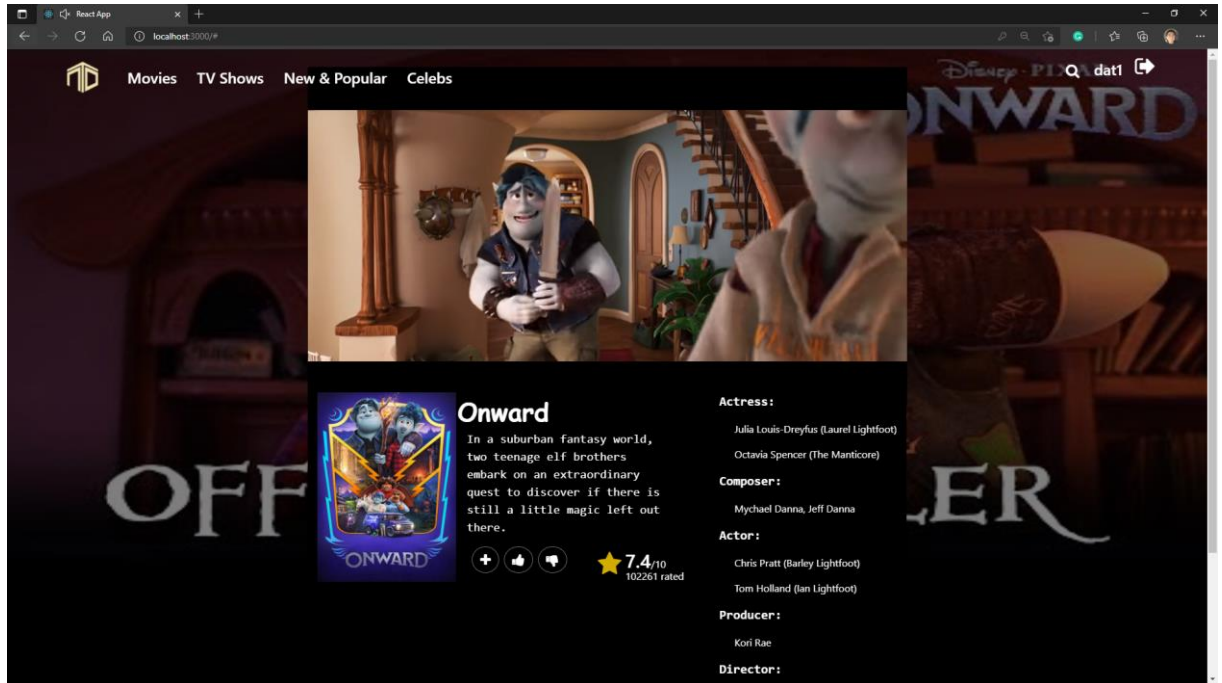


Picture 7: Searching movie function



Picture 8: Searching result for user's keyword

- Когда пользователь хочет увидеть дополнительную информацию о каком-либо фильме, щелкнув по нему, появится интерфейс, показанный ниже. Там пользователь может найти его информацию, трейлер. Кроме того, пользователи могут добавить этот фильм в свой сохраненный список и указать рейтинг для этого фильма.



Picture 9: Information of chosen movie

IV. Дизайн и характеристики сервера системы

1. Описать зачем нужен сервер и как он работает

- Всякий раз, когда пользователь выполняет действие, такое как поиск фильма, внешний интерфейс или пользовательский интерфейс представляет собой инструмент, который делает это действие визуализируемым и более простым для пользователя. Это действие можно назвать «запросом» - пользователь запрашивает часть информации. Сервер работает как «средний человек», что означает получение запросов от пользователя, обработку этих запросов и «возврат» информации, которую пользователь хочет найти.

2. Технологии, используемые для разработки сервера

- В этом проекте я использовал библиотеку Express для разработки сервера. Express помогает контролировать процесс «запрос» - «возврат» на стороне сервера, что упрощает управление и сопровождение. Library Express использует методы HTTP-запроса для приема запросов от пользователей. HTTP-запрос - это действие, которое должно быть выполнено над ресурсом, идентифицированным заданным URL-адресом запроса. Например, когда пользователь хочет выполнить поиск в фильме, пользовательский интерфейс отправляет метод HTTP-запроса GET на URL-адрес <domain>/SEARCH MOVIE?Name=<movie name want to find>. Сервер будет обрабатывать каждый HTTP-запрос и ответ на интерфейсную часть информации в теле HTML-страницы.

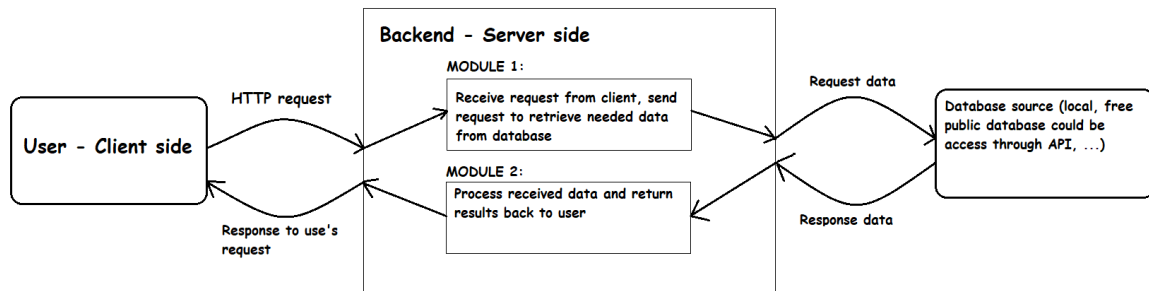


Figure 1: High-architecture system design

- В этом проекте я управлял 2 типами запросов. Запросить данные из базы данных и данные из API. Обработка данных запроса из внутренней базы данных Я объясню более подробно в части V.
- О запросе данных из API, в этом проекте я использовал 2 онлайн-источника данных. Они предоставили бесплатный API для доступа к своим данным. Во-первых, это видеоданные, предоставленные Youtube, во-вторых, данные афиши фильма, предоставленные компанией themoviedb.
- Несмотря на то, что есть 2 разных API, но один и тот же метод их использования. Использование HTTP-запроса с некоторыми настройками в URL, заголовке, ... зависит от того, какой API. После этого мы можем получить доступ к их общедоступной базе данных и получить необходимую информацию.

- Передав название фильма, который хочет найти, API Youtube вернет индекс видео в своей базе данных. Затем поместите этот индекс в URL-адрес, чтобы получить видео.

```
export default axios.create({
  baseURL : 'https://www.googleapis.com/youtube/v3/',
  params : {
    part : 'snippet',
    maxResults : 1,
    key : KEY
  }
})

/*
const res = await youtube.get('/search', {
  params : {
    q : requestApi
  }
})

videoLink = "https://www.youtube.com/embed/"+youtubeId+
"?cc_load_policy=3
&autoplay=1
&mute=0
&controls=0
&origin=http%3A%2F%2Flocalhost%3A3000
&playsinline=1&showinfo=0&rel=0
&iv_load_policy=3
&modestbranding=0
&playlist="+youtubeId+"
&color=white&loop=1&enablejsapi=1&widgetid=1";
*/
```

Picture 10: Calling method to Youtube's API

- Тот же метод, что и API YouTube, передав имя фильма, API-интерфейс themoviedb вернет индекс этого фильма в своей базе данных. С помощью этого индекса мы можем свободно извлекать эти данные.

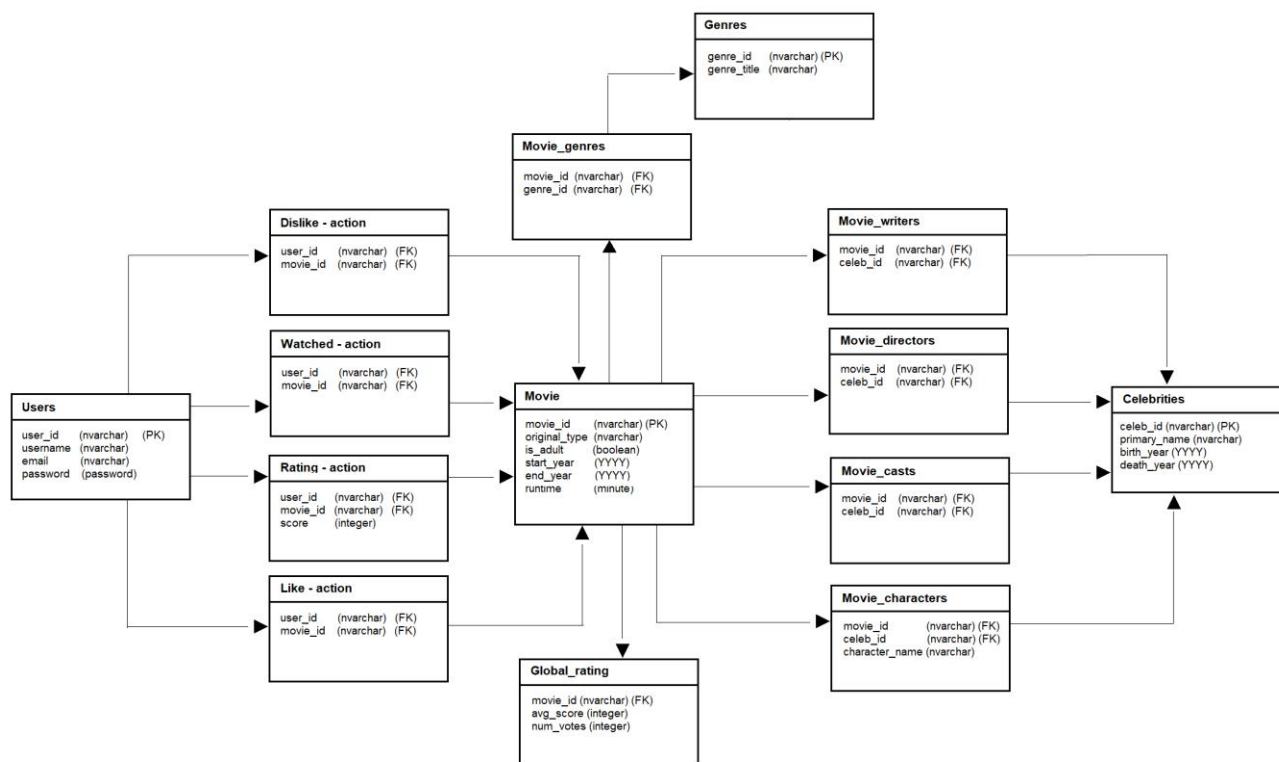
```
router.get('/search', (req,res,next) => {
  var baseUri = "https://api.themoviedb.org/3/search/movie?"
  for (const key in req.query) {
    baseUri = baseUri + key +"="+req.query[key]+"&";
  }
  baseUri = baseUri.slice(0,baseUri.length-1)
  httprequest (
    {url : baseUri},
    (error, response, body) => {
      console.log(baseUri);
      res.send(JSON.parse(body));
    }
  )
})
```

Picture 11: Calling method to TheMovieDB's API

V. Описание базы данных и запросы к ней.

1. Реляционная база данных

- Система, использующая реляционную базу данных и использующая SQL для запросов и обслуживания базы данных.
- Реляционная база данных - это цифровая база данных, основанная на реляционной модели данных. Эта модель организует данные в одну или несколько таблиц (или «отношений») столбцов и строк с уникальным ключом, идентифицирующим каждую строку. Чтобы система управления базами данных (СУБД) работала эффективно и точно, она должна использовать транзакции ACID.
- Сущности и их отношения могут быть построены в схеме, как показано ниже.



Picture 12: Database schema

2. Обработка и преобразование базы данных

- Схема базы данных сформирована, теперь нам нужно обработать базу данных по этой схеме.
- Поскольку необработанные данные находятся в формате CSV. После импорта в формат SQL все данные, очевидно, не имеют схемы, которую мы уже создали. Поэтому нам нужно предварительно обработать эти данные перед их использованием.

2.1. Таблица данных фильма

- Таблица данных SQL с первичным ключом

```
create table Movie (  
    [movie_id] varchar(50) primary key not null,  
    [original_type] varchar(MAX) not null,  
    [is_adult] varchar(20) null,  
    [start_year] varchar(20) null,  
    [end_year] varchar(20) null,  
    [runtime] varchar(20) null  
);  
go
```

- Пример сырых данных фильмов без предварительной обработки

```
select top 20 * from [dbo].[title.basics]  
go
```

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
1	tt0011216	movie	Spanish Fiesta	La fête espagnole	0	2019	\N	67	Drama
2	tt0011801	movie	Totet nicht mehr	Totet nicht mehr	0	2019	\N	\N	Action,Crime
3	tt0060366	short	A Embalagem de Vidro	A Embalagem de Vidro	0	2020	\N	11	Documentary,Short
4	tt0062336	movie	El Tango del Viudo y Su Espejo Defomante	El Tango del Viudo y Su Espejo Defomante	0	2020	\N	70	Drama
5	tt0065392	movie	Bucharest Memories	Anintiri bucurestene	0	2020	\N	\N	Documentary
6	tt0089435	short	Kokoa	Kokoa	0	2019	\N	13	Animation,Short
7	tt0091490	short	Martina's Playhouse	Martina's Playhouse	0	2019	\N	20	Drama,Short
8	tt0116991	movie	Mariette in Ecstasy	Mariette in Ecstasy	0	2019	\N	\N	Drama
9	tt0120589	movie	A Dangerous Practice	A Dangerous Practice	0	2021	\N	\N	Drama
10	tt0123063	short	Everyday	Everyday	0	2019	\N	17	Short
11	tt0129960	tvMovie	Eine geschlossene Gesellschaft	Eine geschlossene Gesellschaft	0	2019	\N	\N	Biography,Documentary,Drama
12	tt0139453	movie	Mauri	Mauri	0	2019	\N	101	Drama,Mystery,Romance
13	tt0172112	short	Ambulans	Ambulans	0	2019	\N	11	Short
14	tt0188299	movie	Heroic Sons and Daughters	Ying xiong er nü	0	2020	\N	\N	War
15	tt0195562	short	Bliss	Bliss	0	2019	\N	6	Short
16	tt0195933	movie	Mysteries	Mysteries	0	2019	\N	\N	\N
17	tt0276132	movie	The Fetishist	The Fetishist	0	2019	\N	\N	Animation
18	tt0279481	movie	Travel Daze	Travel Daze	0	2019	\N	\N	\N
19	tt0280237	tvSeries	Big Brother	Big Brother	0	2020	\N	\N	Game-Show,Reality-TV
20	tt0293429	movie	Mortal Kombat	Mortal Kombat	0	2021	\N	\N	Action,Adventure,Fantasy

- Добавить необработанные данные в таблицу данных SQL на основе схемы таблицы

```

insert into Movie(movie_id, original_type, is_adult, start_year, end_year, runtime)
select tconst, primaryTitle, isAdult, startYear, endYear, runtimeMinutes
from [dbo].[title.basics]
go

select top 20 * from Movie
go

```

	movie_id	original_type	is_adult	start_year	end_year	runtime
1	tt0011216	Spanish Fiesta	0	2019	\N	67
2	tt0011801	Tötet nicht mehr	0	2019	\N	\N
3	tt0060366	A Embalagem de Vidro	0	2020	\N	11
4	tt0062336	El Tango del Viudo y Su Espejo Deformante	0	2020	\N	70
5	tt0065392	Bucharest Memories	0	2020	\N	\N
6	tt0089435	Kokoa	0	2019	\N	13
7	tt0091490	Martina's Playhouse	0	2019	\N	20
8	tt0116991	Mariette in Ecstasy	0	2019	\N	\N
9	tt0120589	A Dangerous Practice	0	2021	\N	\N
10	tt0123063	Everyday	0	2019	\N	17
11	tt0129960	Eine geschlossene Gesellschaft	0	2019	\N	\N
12	tt0139453	Mauri	0	2019	\N	101
13	tt0172112	Ambulans	0	2019	\N	11
14	tt0188299	Heroic Sons and Daughters	0	2020	\N	\N
15	tt0195562	Bliss	0	2019	\N	6
16	tt0195933	Mysteries	0	2019	\N	\N
17	tt0276132	The Fetishist	0	2019	\N	\N
18	tt0279481	Travel Daze	0	2019	\N	\N
19	tt0280237	Big Brother	0	2020	\N	\N
20	tt0293429	Mortal Kombat	0	2021	\N	\N

2.2. Таблица базы данных знаменитостей

- Таблица данных SQL с первичным ключом celeb_id. Она содержит имя, год рождения и год смерти.

```

create table Celebrities (
    [celeb_id] varchar(50) primary key not null,
    [primary_name] varchar(100) not null,
    [birth_year] varchar(20) null,
    [death_year] varchar(20) null
);
go

```

- Необработанные данные NoSQL будут иметь вид:

```
select top 10 * from [dbo].[name.basics]
go
```

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm11098073	Jacqui Templeton	\N	\N	NULL	\N
nm11098074	Lynette Beaton	\N	\N	NULL	\N
nm11098075	Kevin Murphy	\N	\N	actor	\N
nm11098079	Scott Beaton	\N	\N	NULL	tt11211198
nm11098080	David Laing	\N	\N	composer	\N
nm11098082	Meredith Harding	\N	\N	NULL	\N
nm11098083	Waclaw Domanski	\N	\N	NULL	\N
nm11098084	Katarzyna Konieczna	\N	\N	actress	tt11954606,tt13201518,tt0439389,tt9494016
nm11098092	Ziyue Du	\N	\N	director	\N
nm11098093	Ellie	\N	\N	NULL	\N

- Добавить необработанные данные в таблицу данных SQL на основе схемы таблицы

```
insert into Celebrities([celeb_id], [primary_name], [birth_year], [death_year])
select nconst, primaryName, birthYear, deathYear
from [dbo].[name.basics]
go

select top 20 * from [dbo].[Celebrities]
```

	celeb_id	primary_name	birth_year	death_year
1	nm0000001	Fred Astaire	1899	1987
2	nm0000002	Lauren Bacall	1924	2014
3	nm0000003	Brigitte Bardot	1934	\N
4	nm0000004	John Belushi	1949	1982
5	nm0000005	Ingmar Bergman	1918	2007
6	nm0000006	Ingrid Bergman	1915	1982
7	nm0000007	Humphrey Bogart	1899	1957
8	nm0000008	Marlon Brando	1924	2004
9	nm0000009	Richard Burton	1925	1984
10	nm0000010	James Cagney	1899	1986
11	nm0000013	Doris Day	1922	2019
12	nm0000014	Olivia de Havilland	1916	2020
13	nm0000015	James Dean	1931	1955
14	nm0000017	Marlene Dietrich	1901	1992
15	nm0000018	Kirk Douglas	1916	2020
16	nm0000022	Clark Gable	1901	1960
17	nm0000023	Judy Garland	1922	1969
18	nm0000025	Jerry Goldsmith	1929	2004
19	nm0000026	Cary Grant	1904	1986
20	nm0000027	Alec Guinness	1914	2000

2.3. Таблица данных, содержащая фильмы и их жанры

- Таблица содержит [movie_id] в качестве внешнего ключа, который относится к таблице Movie.

```
create table Movie_genres (
    [movie_id] varchar(50) not null,
    constraint FK_movie foreign key ([movie_id]) references Movie(movie_id),
    [genres_type] varchar(200) null
);
go
```

- Необработанные данные содержат movie_id, movie_name, и строка представляет жанры фильма, имеющие [movie_id]

```
select top 20 tconst,primaryTitle,genres from [title.basics]
go
```

	tconst	primaryTitle	genres
1	tt0011216	Spanish Fiesta	Drama
2	tt0011801	Tötet nicht mehr	Action,Crime
3	tt0060366	A Embalagem de Vidro	Documentary,Short
4	tt0062336	El Tango del Viudo y Su Espejo Defomante	Drama
5	tt0065392	Bucharest Memories	Documentary
6	tt0089435	Kokoa	Animation,Short
7	tt0091490	Martina's Playhouse	Drama,Short
8	tt0116991	Mariette in Ecstasy	Drama
9	tt0120589	A Dangerous Practice	Drama
10	tt0123063	Everyday	Short
11	tt0129960	Eine geschlossene Gesellschaft	Biography,Documentary,Drama
12	tt0139453	Mauri	Drama,Mystery,Romance
13	tt0172112	Ambulans	Short
14	tt0188299	Heroic Sons and Daughters	War
15	tt0195562	Bliss	Short
16	tt0195933	Mysteries	\N
17	tt0276132	The Fetishist	Animation
18	tt0279481	Travel Daze	\N
19	tt0280237	Big Brother	Game-Show,Reality-TV
20	tt0293429	Mortal Kombat	Action,Adventure,Fantasy

- И мы вставляем в нашу определенную таблицу

```
insert into Movie_genres(movie_id,[genres type])
select tconst, genres
from [dbo].[title.basics]
go

select * from Movie_genres
go
```

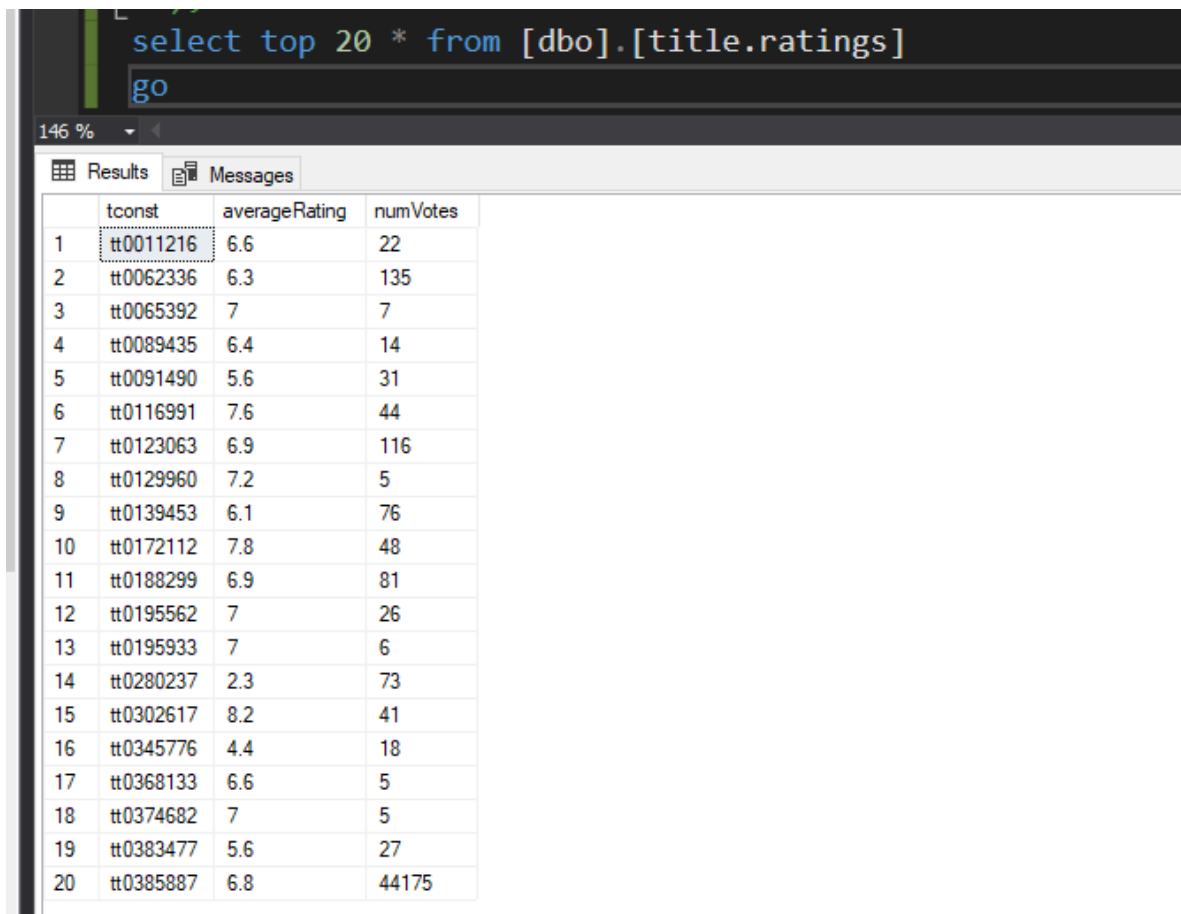
	movie_id	genres_type
1	tt0011216	Drama
2	tt0011801	Action,Crime
3	tt0060366	Documentary,Short
4	tt0062336	Drama
5	tt0065392	Documentary
6	tt0089435	Animation,Short
7	tt0091490	Drama,Short
8	tt0116991	Drama
9	tt0120589	Drama
10	tt0123063	Short
11	tt0129960	Biography,Documentary,Drama
12	tt0139453	Drama,Mystery,Romance
13	tt0172112	Short
14	tt0188299	War
15	tt0195562	Short
16	tt0195933	\N
17	tt0276132	Animation
18	tt0279481	\N
19	tt0280237	Game-Show,Reality-TV
20	tt0293429	Action,Adventure,Fantasy
21	tt0293513	\N
22	tt0302617	Documentary

2.4 Таблица данных глобальных рейтингов и общего количества голосов за отдельный фильм, записанный IMDB

- Наша таблица SQL содержит [movie_id] в качестве внешнего ключа, который относится к отдельному фильму и свойствам.

```
create table Movie_rating (  
    [movie_id] varchar(50) not null,  
    constraint FK_movie_rate foreign key ([movie_id]) references Movie(movie_id),  
    [average_rating] float null,  
    [num_votes] int null  
);
```

- Необработанные данные выглядят так:



```
select top 20 * from [dbo].[title.ratings]  
go
```

	tconst	averageRating	numVotes
1	tt0011216	6.6	22
2	tt0062336	6.3	135
3	tt0065392	7	7
4	tt0089435	6.4	14
5	tt0091490	5.6	31
6	tt0116991	7.6	44
7	tt0123063	6.9	116
8	tt0129960	7.2	5
9	tt0139453	6.1	76
10	tt0172112	7.8	48
11	tt0188299	6.9	81
12	tt0195562	7	26
13	tt0195933	7	6
14	tt0280237	2.3	73
15	tt0302617	8.2	41
16	tt0345776	4.4	18
17	tt0368133	6.6	5
18	tt0374682	7	5
19	tt0383477	5.6	27
20	tt0385887	6.8	44175

- И запросы на вставку данных

```
insert into Movie_rating(movie_id,average_rating, num_votes)
select tconst, averageRating, numVotes
from [dbo].[title.ratings]
go

select top 20 * from Movie_rating
```

146 %

Results Messages

	movie_id	average_rating	num_votes
1	tt0011216	6.6	22
2	tt0062336	6.3	135
3	tt0065392	7	7
4	tt0089435	6.4	14
5	tt0091490	5.6	31
6	tt0116991	7.6	44
7	tt0123063	6.9	116
8	tt0129960	7.2	5
9	tt0139453	6.1	76
10	tt0172112	7.8	48
11	tt0188299	6.9	81
12	tt0195562	7	26
13	tt0195933	7	6
14	tt0280237	2.3	73
15	tt0302617	8.2	41
16	tt0345776	4.4	18
17	tt0368133	6.6	5
18	tt0374682	7	5
19	tt0383477	5.6	27
20	tt0385887	6.8	44175

2.5. Отношения между знаменитостью и отдельным фильмом

- Необработанные данные содержат взаимосвязь между фильмом [movie_id] и знаменитостью [celebrity], а также их роль в этом фильме в столбце [category].

```
select category from [dbo].[title.principals]
group by category
go

select top 20 * from [dbo].[title.principals]
where category = N'director'
go
```

146 %

Results Messages

	category
1	producer
2	editor
3	actress
4	writer
5	self
6	archive_sound
7	cinematographer
8	actor
9	director

	tconst	ordering	nconst	category	job	characters
1	tt10623252	4	nm10830824	director	\N	\N
2	tt10623252	3	nm10830825	director	\N	\N
3	tt10623264	1	nm10830847	director	\N	\N
4	tt10623266	1	nm10830848	director	\N	\N
5	tt10623272	1	nm10830853	director	\N	\N
6	tt10839436	5	nm10830871	director	\N	\N
7	tt10623308	1	nm10830882	director	\N	\N
8	tt10623386	1	nm10830929	director	\N	\N
9	tt10623392	1	nm10830932	director	\N	\N
10	tt10623394	1	nm10830932	director	\N	\N
11	tt10623408	1	nm10830987	director	\N	\N
12	tt10623410	2	nm10830988	director	\N	\N

- Процесс извлечения данных для каждой роли очень похож друг на друга, поэтому я показываю примеры для роли директора. Остальные такие же.
- Итак, таблица всех режиссерских ролей содержит 2 внешних ключа, обозначающих знаменитость как режиссера в соответствующем фильме.

```
create table Movie_directors (
    movie_id varchar(50) not null,
    celeb_id varchar(50) not null,
    constraint FK_movie_dir foreign key ([movie_id]) references Movie(movie_id),
    constraint FK_celeb foreign key (celeb_id) references Celebrities(celeb_id),
)
```


- И запросы на вставку данных

```
insert into Movie_directors(movie_id,celeb_id)
select tconst, nconst
from [dbo].[title.principals]
where category = N'director'
go

select top 20 * from Movie_directors
go
```

146 %

Results Messages

	movie_id	celeb_id
1	tt0011216	nm0241273
2	tt0060366	nm0206937
3	tt0062336	nm0749914
4	tt0062336	nm0765384
5	tt0065392	nm12030368
6	tt0065392	nm12030369
7	tt0065392	nm0300168
8	tt0089435	nm0016087
9	tt0091490	nm0014411
10	tt0116991	nm0007037
11	tt0120589	nm0004000
12	tt0123063	nm0725257
13	tt0129960	nm0107048
14	tt0139453	nm0593039
15	tt0172112	nm0605155
16	tt0188299	nm0943210
17	tt0195562	nm0548592
18	tt0195933	nm0548592
19	tt0276132	nm0870719
20	tt0279481	nm0692785

3. Некоторые запросы к базе данных для каждого отдельного действия пользователя

3.1. Запрос на отдельный фильм

- Во-первых, нам нужно создать VIEW для будущей работы. В базе данных VIEW - это набор результатов сохраненного запроса к данным, который пользователи базы данных могут запрашивать так же, как в постоянном объекте коллекции базы данных.
- Эта предустановленная команда запроса хранится в словаре базы данных. VIEWS могут использоваться как повторно используемые разделы SELECT / CODE, которые могут быть включены в другие выборки / запросы, к которым нужно присоединиться, и использовать различные различные фильтры, без необходимости заново создавать весь SELECT каждый раз.
- View содержит всю основную информацию, относящуюся к фильму :

```
if OBJECT_ID(N'dbo.movie_title_view',N'V') is not null
[
    drop view [dbo].[movie_title_view]
go

create view dbo.movie_title_view (
    [movie_id],[original_type],[is_adult],
    [start_year],[end_year],[runtime],
    [num_votes], [avg_scores]
)
with schemabinding
as
select
    movie.movie_id as [movie_id], movie.original_type as [original_type],
    movie.is_adult as [is_adult], movie.start_year as [start_year], movie.end_year as [end_year],
    movie.runtime as [runtime],
    rating.average_rating as [avg_scores], rating.num_votes as [num_votes]
from [dbo].[Movie] as movie
inner join [dbo].[Movie_rating] rating
on movie.[movie_id] = rating.[movie_id]
go
```

Picture 13: VIEW consist of movie information

- При извлечении данных из таблицы или представления в базе данных система будет последовательно искать каждую строку и проверять, соответствует ли она сформированному условию. Это вызовет своевременное выполнение при поиске по огромной таблице. По этой причине мы придумали другой метод поиска - INDEX.

- INDEX - это структура на диске, связанная с таблицей или представлением, которая ускоряет извлечение строк из таблицы или представления. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в структуре (B-дереве), которая позволяет SQL Server быстро и эффективно находить строку или строки, связанные со значениями ключей.

```

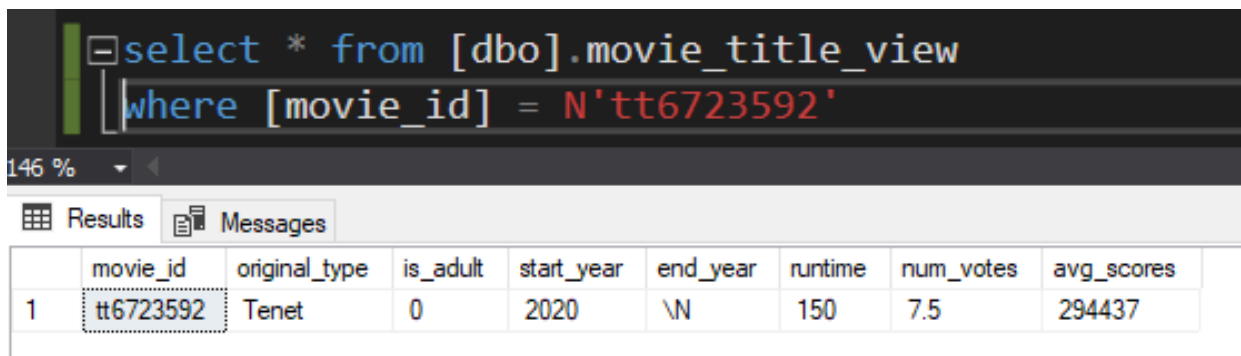
create unique clustered index idx_movie_id
on [dbo].movie_title_view([movie_id])
go

create nonclustered index idx_movie_view
on [dbo].movie_title_view([movie_id],[start_year])
include([original_type],[is_adult],[num_votes],[runtime],[avg_scores])

```

Picture 14: UNIQUE INDEX and NONCLUSTERED INDEX on VIEW

- Пример, когда пользователь хочет увидеть дополнительную информацию, щелкнув по нему, мы узнаем идентификатор фильма, который пользователь хочет увидеть, поэтому мы ищем его в нашем VIEW
- Запрос и результат:



```

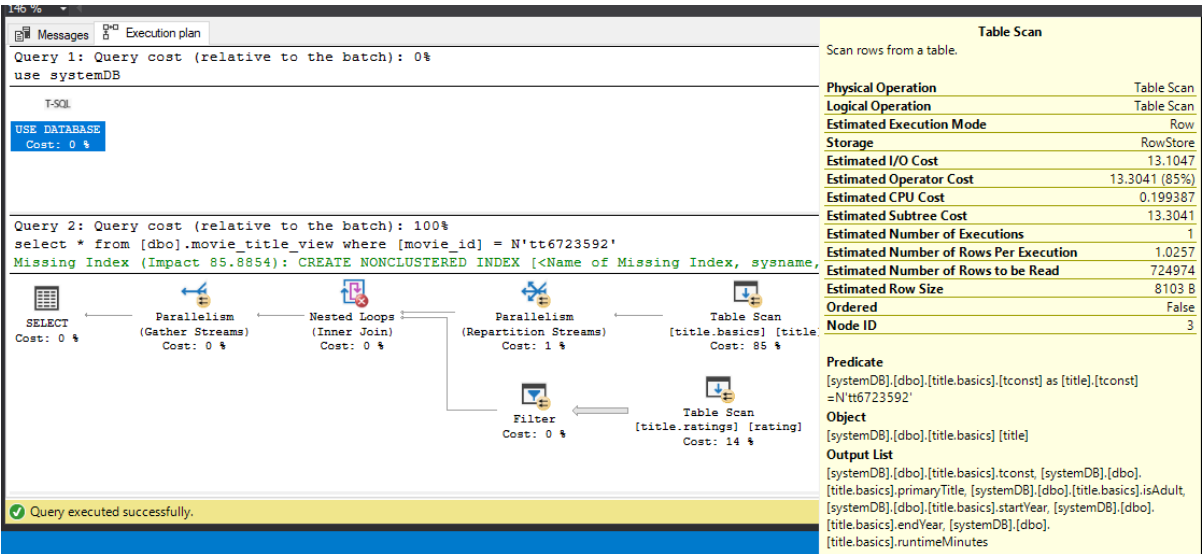
select * from [dbo].movie_title_view
where [movie_id] = N'tt6723592'

```

	movie_id	original_type	is_adult	start_year	end_year	runtime	num_votes	avg_scores
1	tt6723592	Tenet	0	2020	\N	150	7.5	294437

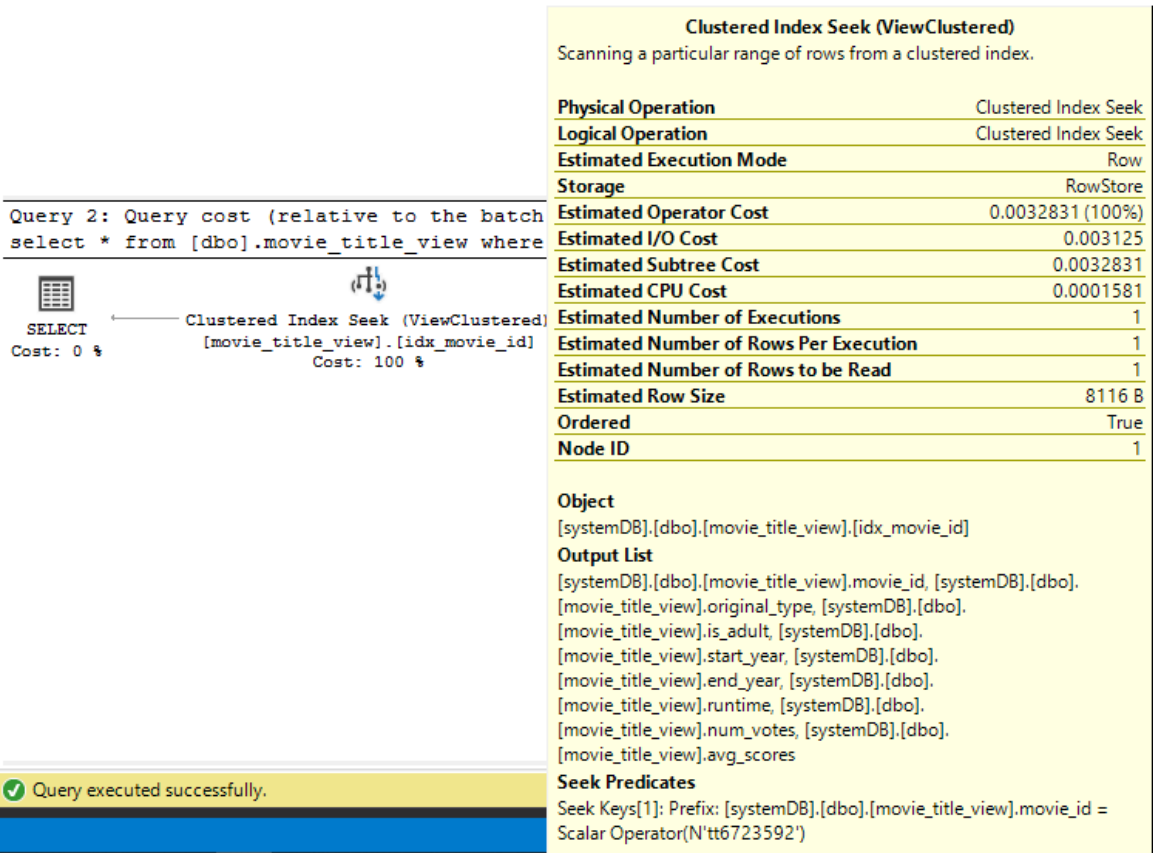
Picture 15: Query and it's result

- Стоимость оборудования при использовании обычного поиска:



Picture 16: Estimated cost of table scan operation

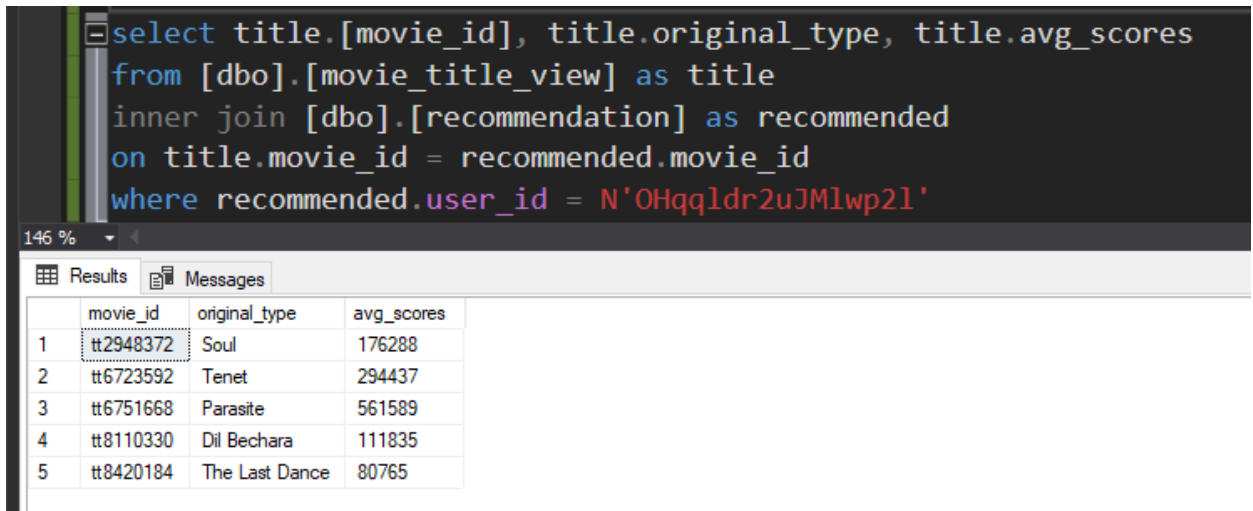
- И стоимость оборудования при использовании поиска по INDEXES. Что намного лучше обычного поиска



Picture 17: Estimated cost of index search

3.2. Поиск по списку фильмов

- Используя тот же вид, что и выше, и поиск по индексу, мы также можем получить список данных с низкой задержкой. Пример, когда система находит список рекомендательных фильмов для конкретного пользователя:



The screenshot shows a SQL query in a script editor and its results in a table. The query selects movie details for a specific user. The results table has five rows of data.

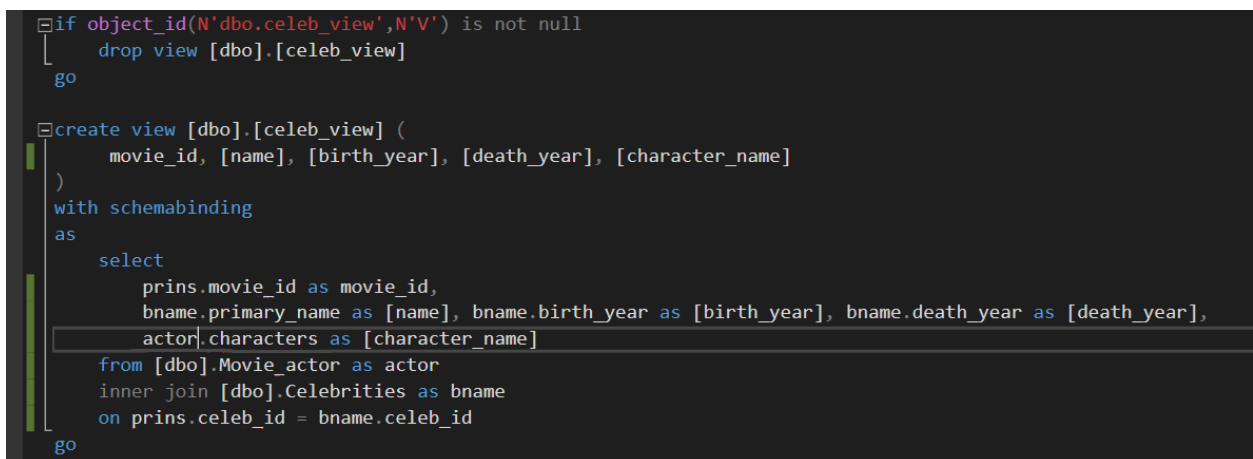
```
select title.[movie_id], title.original_type, title.avg_scores
from [dbo].[movie_title_view] as title
inner join [dbo].[recommendation] as recommended
on title.movie_id = recommended.movie_id
where recommended.user_id = N'OHqqlDr2uJM1wp21'
```

	movie_id	original_type	avg_scores
1	tt2948372	Soul	176288
2	tt6723592	Tenet	294437
3	tt6751668	Parasite	561589
4	tt8110330	Dil Bechara	111835
5	tt8420184	The Last Dance	80765

Picture 18: Query for recommended movie for individual user

3.3. Поиск знаменитостей, связанных с отдельным фильмом

- Когда пользователь ищет конкретный фильм, отображается не только основная информация, но и отображаются все знаменитости.
- Поскольку у нас была связь между movie_id и Celeb_id, мы можем создать VIEW как обертку информации и запросить ее с помощью поиска по INDEX.
- VIEW содержит фильм и информацию о знаменитостях, связанных с фильмом



The screenshot shows a SQL script to create a view named 'celeb_view'. The script includes a drop statement if the view exists, followed by a create statement with a select query that joins movie and celebrity data.

```
if object_id(N'dbo.celeb_view',N'V') is not null
[
    drop view [dbo].[celeb_view]
go

create view [dbo].[celeb_view] (
    movie_id, [name], [birth_year], [death_year], [character_name]
)
with schemabinding
as
select
    prins.movie_id as movie_id,
    bname.primary_name as [name], bname.birth_year as [birth_year], bname.death_year as [death_year],
    actor.characters as [character_name]
from [dbo].Movie_actor as actor
inner join [dbo].Celebrities as bname
on prins.celeb_id = bname.celeb_id
go
```

Picture 19: VIEW consist of celebrity information and their role in specific movie

- Вставить UNIQUE INDEX в VIEW на основе значения [movie_id]

```
create unique clustered index idx_celeb_view
on [dbo].[celeb_view]([movie_id])
go
```

Picture 20: UNIQUE INDEX on VIEW

- Запрос, сравнение результатов и производительности между сканированием таблицы и поиском по INDEX

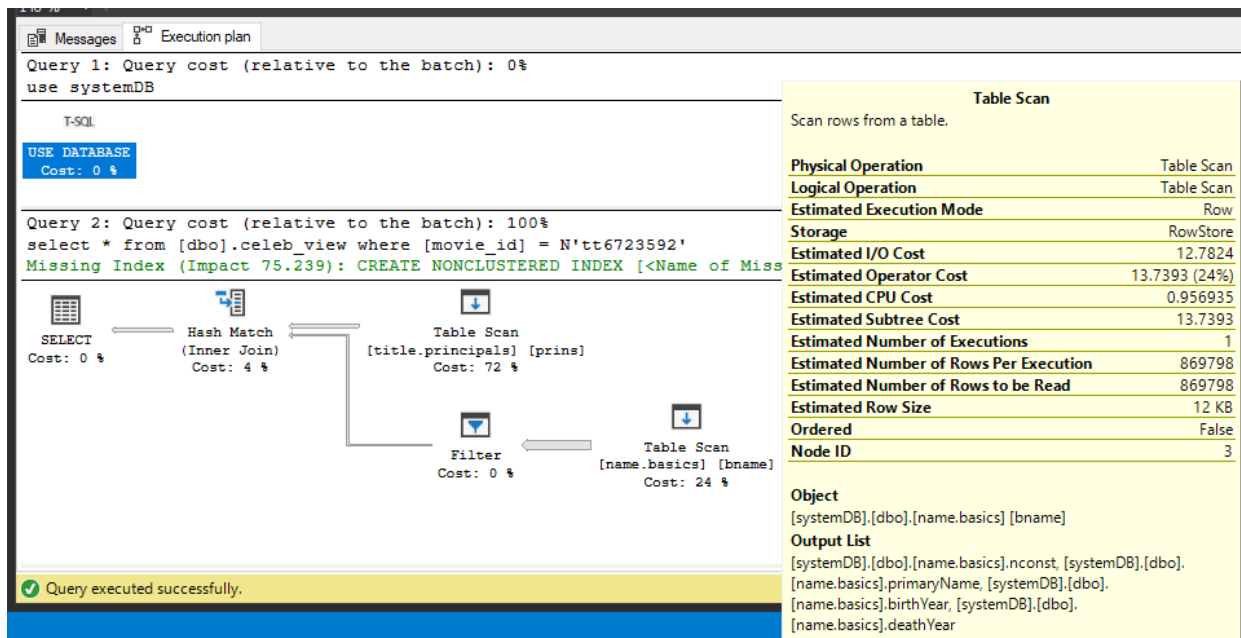
146 %

Results Messages

	movie_id	name	birth_year	death_year	role	character_name
1	tt6723592	Nathan Crowley	1966	\N	production_designer	\N
2	tt6723592	Hoyte Van Hoytema	1971	\N	cinematographer	\N
3	tt6723592	John David Washington	1984	\N	actor	["Protagonist"]
4	tt6723592	Robert Pattinson	1986	\N	actor	["Neil"]
5	tt6723592	Juhan Ulfak	1973	\N	actor	["Passenger"]
6	tt6723592	Ludwig Göransson	1984	\N	composer	\N
7	tt6723592	Christopher Nolan	1970	\N	director	\N
8	tt6723592	Emma Thomas	\N	\N	producer	\N
9	tt6723592	Jennifer Lame	\N	\N	editor	\N
10	tt6723592	Elizabeth Debicki	1990	\N	actress	["Kat"]

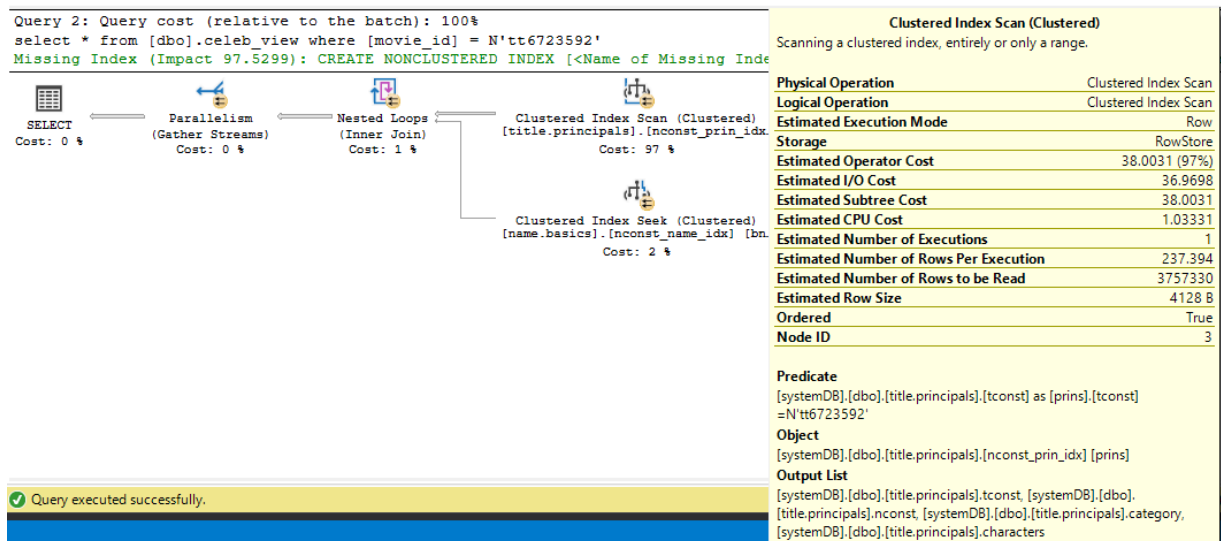
Picture 21: Query for all role in specific movie

- Сканирования таблицы



Picture 22: Estimated cost for scan table

- Поиска по INDEXES



Picture 23: Estimated cost for index search

3.4. Обработка некоторых основных действий пользователя

Поддержка SQL-сервера: мы модифицируем запрос событий с помощью моей собственной процедуры с помощью TRIGGER. Триггер - это особый тип хранимой процедуры, которая автоматически запускается при возникновении события на сервере базы данных.

3.4.1. Когда пользователь регистрирует новую учетную запись

- Когда пользователь хочет зарегистрировать свою учетную запись, сервер обрабатывает запрос и получает необходимую информацию для новой учетной записи, включая имя_пользователя, хешированный пароль (ключ для хранилища хеш-функций на сервере), ... Если [user_name] не существует в база данных с уникальным значением [user_id] - добавьте его в нашу базу данных, иначе сервер вернет ошибку, чтобы заметить пользователя.
- Процедура триггера вместо обычного запроса вставки

```
create trigger insert_userdata_trigger
on [dbo].[users.data]
instead of insert
as
begin
    if exists(select top 1 i.[user_name]
              from inserted as i
              where i.[user_name]
                    in (select [user_name] from [dbo].[users.data]))
    begin
        exec sp_addmessage 50001, 15, N'User_name exist', @lang = 'us_english', @replace='REPLACE';
        RAISERROR(50001,15,-1)
    end
    else
    begin
        insert into [dbo].[users.data] ([user_id], [user_name], [password])
        select CONVERT(nvarchar(100), NEWID()), i.[user_name], i.[password]
        from inserted as i
    end
end
go
```

Picture 24: TRIGGER for INSERT query into users database

- Если требование выполнено

```
insert [dbo].[users.data]([user_id],[user_name], [password]) values (N'',N'dat4', N'dsakgfs31df2132jdfghsd243')
go
```

	user_id	user_name	password
1	0C1ACC5A-DF76-4BCD-87E3-9FB31CCAECE1	dat4	dsakgfs31df2132jdfghsd243
2	9N3n8k48FaafmX	dat2	654ce6bde3f0baaa4d9a96248cd7e651
3	OHqqlr2uJMIwp2	dat1	654ce6bde3f0baaa4d9a96248cd7e651
4	QHmMy5IOoe8yDX3C	dat3	654ce6bde3f0baaa4d9a96248cd7e651

Picture 25: Result if insert successfully

- В противном случае верните ошибку

```

insert [dbo].[users.data]([user_id],[user_name], [password]) values (N'',N'dat1', N'dsakgfs31df2132jdfghsd243')
go

```

Messages

Msg 50001, Level 15, State 1, Procedure insert_userdata_trigger, Line 13 [Batch Start Line 36]
 User_name exist

(1 row affected)

Completion time: 2021-09-14T14:41:19.9736338+03:00

Picture 26: Result if insert fail

3.4.2. Когда пользователи хотят добавить новый фильм в свои личные категории (понравился, не понравился, сохранен для будущего просмотра, ...)

- Поскольку процедуры очень похожи друг на друга, просто рассмотрим пример, когда пользователь хочет добавить новый фильм в свою категорию для просмотра в будущем.
- Когда пользователь хочет добавить фильм в свой список, нажав кнопку, сервер обрабатывает этот запрос и отправляет в базу данных сигнал - [user_name] хочет добавить [movie_id] в базу данных [saved_movies]
- Если [user_name] еще не добавил [movie_id] в свой список, управление базой данных добавит новые данные и вернется к уведомлению пользователя, которое является его / ее фильмом, успешно добавленным в базу данных. В противном случае, при очень обычном UX-дизайне, руководство удалит [movie_id] из списка фильмов [user_name] и отправит обратно пользователю уведомление, которое [movie_id] было удалено из его / ее списка желаний.
- Процедура триггера вместо обычного запроса вставки

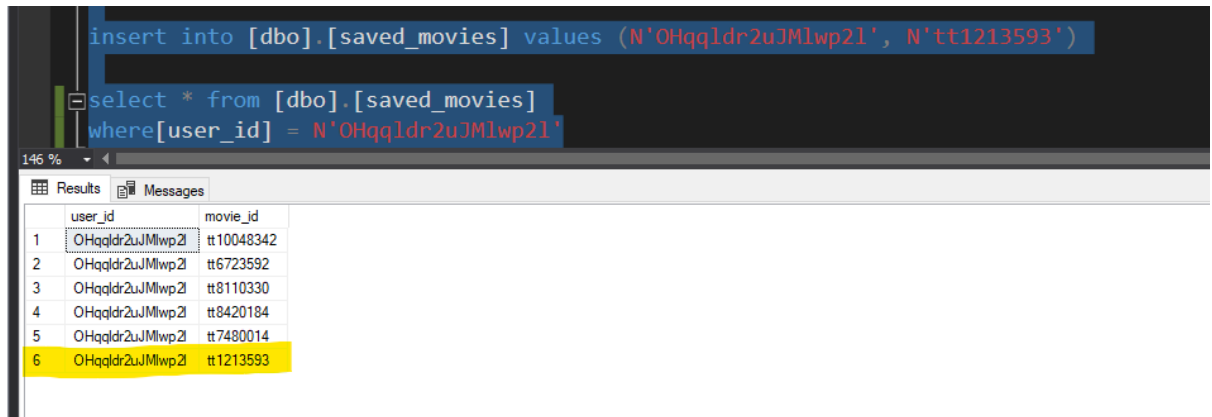
```

create trigger insert_saved_movie_trigger
on [dbo].[saved_movies]
instead of insert
as
begin
    if exists (select top 1 *
              from inserted as i
              inner join [dbo].[saved_movies] as mov
              on i.[user_id] = mov.[user_id] and i.[movie_id] = mov.[movie_id])
    begin
        exec sp_addmessage 50001, 15, N'Movie had been saved', @lang = 'us_english', @replace='REPLACE';
        delete from [dbo].[saved_movies]
        where exists (
            select *
            from inserted as i
            where i.[user_id] = [saved_movies].[user_id]
            and i.[movie_id] = [saved_movies].[movie_id]
        )
    end
else
    begin
        exec sp_addmessage 50001, 15, N'Add movie to saved list', @lang = 'us_english', @replace='REPLACE';
        insert into [dbo].[saved_movies] ([user_id], [movie_id])
        select top 1 i.[user_id], i.[movie_id]
        from inserted as i
    end
end

```

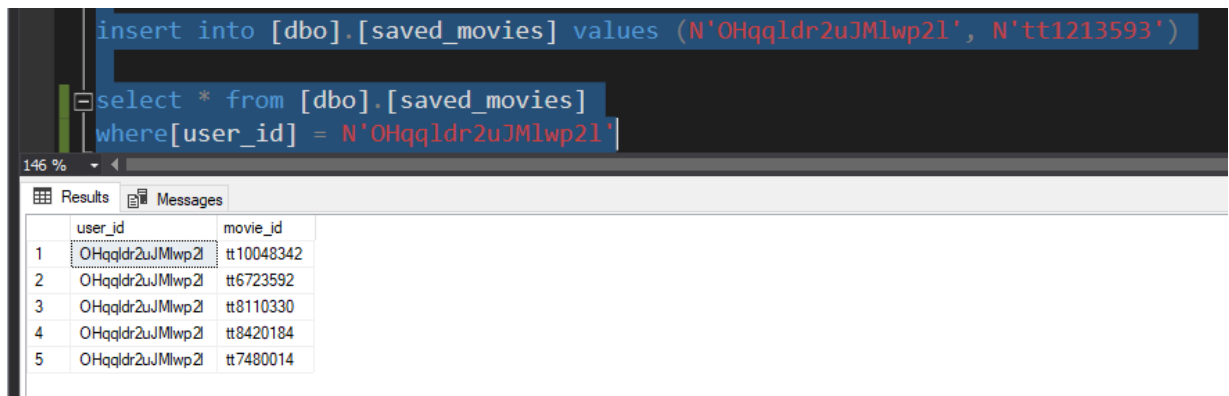
Picture 27: TRIGGER for INSERT query into personal list

- Если фильм добавлен впервые, база данных успешно добавлена в базу данных



Picture 28: Result if insert new one to database

- Если встретились старые данные, удалить их из базы данных



Picture 29: Result if insert data which already in database

VI. Заключение и дальнейшая работа

- Система рекомендаций фильмов - не новая концепция в современной индустрии высоких технологий. Таким образом, многие крупные компании ведут свой бизнес на основе такой системы, например Netflix, IMDb, Из-за ограничений технологии и времени мой проект создавался только в образовательных целях, а не как продукт, способный справиться с огромными рабочими нагрузками. Если мы поместим это в реальные обстоятельства, возникнет много проблем. Такие как:

- Данные могут быть очень большими, поэтому одна база данных может не соответствовать всем данным. А при использовании нескольких баз данных возникает множество проблем, таких как связь между базами данных, связь между базами данных и сервером, обслуживание базы данных и т. Д.
- При большом количестве клиентов, использующих продукт, одного сервера может быть недостаточно для обработки всех запросов.
- Сложность алгоритма по времени слишком велика и не очень надежна, требуется улучшение, чтобы улучшить взаимодействие с пользователем.
- Текущая система не обладает отказоустойчивостью, поэтому, если произойдет что-то плохое, что приведет к отключению системы, единственный способ - перезагрузить всю систему, что в реальной жизни является очень плохой идеей.
- В течение последних нескольких десятилетий, с появлением множества веб-сервисов, рекомендательные системы занимают все больше и больше места в нашей жизни. От электронной коммерции (предлагать покупателям статьи, которые могут их заинтересовать) до онлайн-рекламы (предлагать пользователям правильный контент, соответствующий их предпочтениям), рекомендательные системы сегодня неизбежны в нашей повседневной работе в Интернете. Они действительно важны в некоторых отраслях, поскольку они могут приносить огромный доход, когда они эффективны, или также быть способом значительно отстать от конкурентов.

VII. Список литературы

- [1] [React – A JavaScript library for building user interfaces \(reactjs.org\)](https://reactjs.org/)
- [2] [Express - Node.js web application framework \(expressjs.com\)](https://expressjs.com/)
- [3] [YouTube Data API | Google Developers](https://developers.google.com/youtube/)
- [4] [API Overview — The Movie Database \(TMDB\) \(themoviedb.org\)](https://themoviedb.org/)
- [5] [IMDb](https://www.imdb.com/)
- [6] [Collaborative filtering - Wikipedia](https://en.wikipedia.org/wiki/Collaborative_filtering)
- [7] [pres1-matrixfactorization.pdf \(hpi.de\)](https://www.hpi.de/~informatik/lehre/ss16/ai/ps1/pres1-matrixfactorization.pdf)
- [8] [Matrix factorization \(recommender systems\) - Wikipedia](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems))
- [9] [Singular Value Decomposition \(SVD\) In Recommender System \(analyticsindiamag.com\)](https://analyticsindiamag.com/singular-value-decomposition-svd-in-recommender-system/)
- [10] [\[PDF\] Understanding and improving relational matrix factorization in recommender systems \(researchgate.net\)](https://www.researchgate.net/publication/312111111_Understanding_and_improving_relational_matrix_factorization_in_recommender_systems)
- [11] [Mixed Recommender System- MF\(Matrix Factorization\) with item similarity based CF\(Collaborative Filtering\) | by Sourish Dey | Towards Data Science](#)
- [12] [An Efficient Deep Learning Approach for Collaborative Filtering Recommender System - ScienceDirect](#)