IMDB Data Management

Richard Joerger
Rochester Institute of Technology
raj2348@g.rit.edu

Yifei Sun
Rochester Institute of Technology
ys8800@g.rit.edu

Ajeeta Khatri
Rochester Institute of Technology
ak6038@g.rit.edu

Zhuo Liu
Rochester Institute of Technology
zl9901@g.rit.edu

1. OVERVIEW

The purpose of this paper is to serve as a final discussion of what we've done on our database project. It encompasses our ER model, the conversion to relationships, normalization of those relationships, constructing our database, cleaning the data for the database, and the implementation and usage of our program. The main focus of this paper will be on the work which has been done between the last paper and this paper, as well as any last minute design and implementation changes we have made due to time constraints.

2. DATASET DESCRIPTION

The chosen IMDB dataset describes various aspects related to movies. Some of the examples of the data in the dataset is the data related to titles, actors, crews, episodes and ratings. All the data in the dataset is interrelated to different entities.

Below is a sample data representation. The data for actors can be represented by the following fields (nconst, primaryName, birthYear, deathYear, primaryProfession, knownForTitles) and a sample data is (nm0000001, Fred Astaire, 1899 ,1987, soundtrack, actor, miscellaneous, tt0043044, tt0050419, tt0053137, tt0072308). So, given this information Fred Astaire is the primary name of the person whose primaryProfession is soundtrack, actor, and miscellaneous. The fields for sample data for movie title can be represented as (titleId, ordering, title, region, language, types, attributes, isOriginalTitle). An example of this is (tt0000001, 1, Carmencita - spanyol tánc,HU, ", imdbDisplay, ", 0). In this case, we know that this movie, Carmencita - spanyol tánc, was released in Hungarian and is the original title. The attributes for the data related to episodes is as follows (tconst, parentTconst, seasonNumber, episodeNumber) and some sample data can be given by (tt0041951, tt0041038, 1, 9). This table is effectively a go between which helps maps an episode and its season to its entry in the title database. The dataset also contains the data regarding the movie crew, specifically the director and writer information for all the titles in IMDb where these fields are described (tconst, directors, writers). Sample data related to this can be represented as (tt0000001, nm0005690, "). This data is directly related to the principles as the tconst maps to values in the principles data. The data related to principles is given by (tconst, ordering, nconst, category, job, characters) and a sample data looks like (tt0000001, 1, nm1588970, self, ", ["Herself"]).The ratings for each movies is stored with the following attributes (tconst, averageRating, numVotes), where a single rating looks like (tt0000001, 5.6, 1533).

Some of the user scenarios for our systems could be querying for the movies titles based on the laguange of the movie. For example, if the user wants to find all the movies in English language, the query would look something like:

(Select * from title where language = 'en').
This would return a list of all the movies in our database with English language. In another scenario if the user wants to get a list of all famous movies casting his favourite actor, the user can also get the list of movies based on the actor. The query would looks something like.
(Select knownForTitles from actors where name = 'Brad Pitt').
These are some of the high level scenarios, we can further narrow them based on the filters selected by the user.

3. DATABASE AND RELATIONAL MODEL

3.1 ER-Model and database structure

All the tables and the constraints are described as follows:
users (uid,passwd,age,language) PK(uid).
history (uid,tconst,date) PK(uid,tconst) FK_1 (uid) FK_2 (tconst).
rating (uid,tconst,rating,isVote)PK(uid,tconst)FK_1 (uid) FK_2 (tconst).
general_movie (tconst,title,isAdult,type)PK(tconst).
genres (tconst,genre)PK(tconst,genre) FK(tconst).
localize (tconst,local-name,language,isOriginal) PK(tconst,local-name,language) FK(tconst).
tvSeries(series-tconst,isOver) PK(series-tconst) FK(series-tconst).
tvEpisode(episode-tconst,episode-number) PK(episode-tconst) FK(episode-tconst).
has(series-tconst,episode-tconst,season-number,broadcast-year) PK(series-tconst,episode-tconst). FK(series-tconst, episode-tconst)
movie(movie-tconst,release-year,runtime) PK(movie-tconst) FK(movie-tconst).
videoGame(game-tconst,sells-year) PK(game-tconst) FK(game-tconst).
news(news-tconst) PK(news-tconst) FK(news-tconst).
persons(primary-name,birth-year,death-year) PK (primary-name, birth-year).
professions(primary-name,birth-year,profession) PK (primary-name, birth-year, profession) FK(primary-name, birth-year).
acts(act-name, birth-year, movie-tconst, character) PK (act-name, birth-year, movie-tconst, character) FK_1 (act-name,

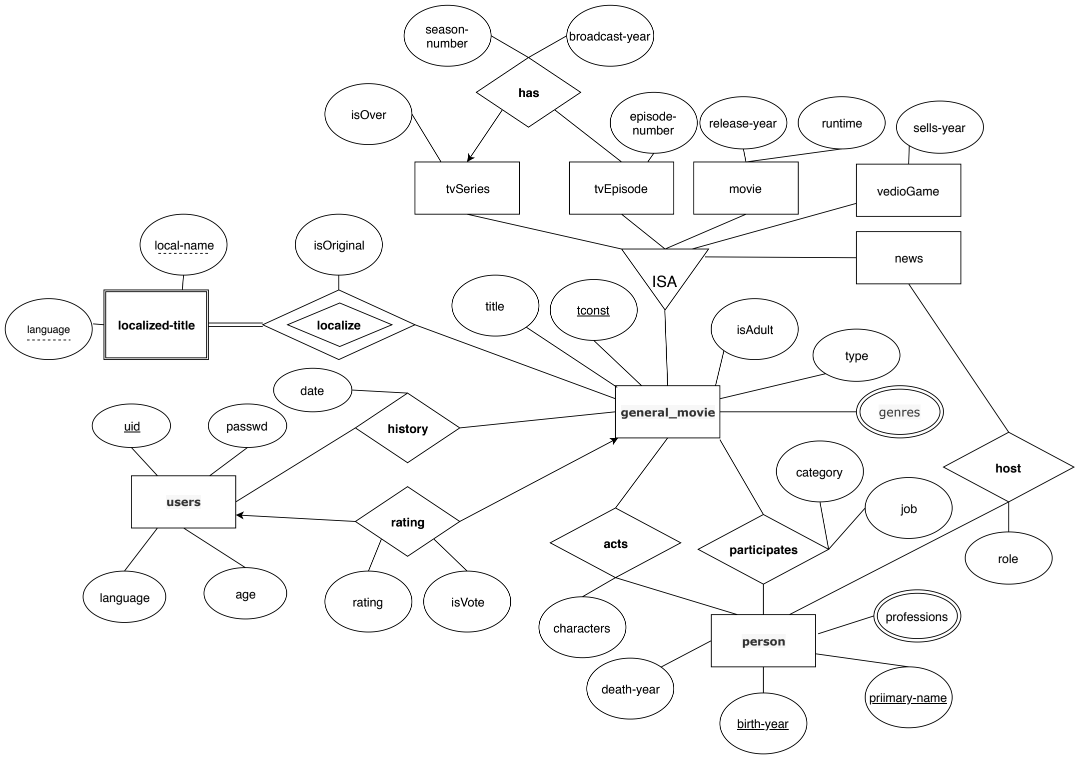

Figure 1: ER Model for our Application

birth-year) FK_2 (movie-tconst).
participates(act-name, birth-year, movie-tconst, category, job) PK (act-name, birth-year, movie-tconst) FK_1 (act-name, birth-year) FK_2(movie-tconst).
host(news-tconst, host-name, birth-year, role) PK(news-tconst, host-name, birth-year) FK_1 (news-tconst) FK_2 (host-name, birth-year)

3.2 Raw data importing

In order to manipulate the IMDB data, we need to import the raw data from a .tsv file into our database. For this part, I choose to create tables in the same way that IMDB did. We create and maintain 'title_akas', 'title_basics', 'title_principals', 'title_crews', 'title_episodes', 'title_ratings' and 'name_basics' tables. We'll keep all the data along with every attribute even though there some of those attributes may have null values. Because we've kept all the data, we can manipulate these table and preform each query the way we want via SQL.

3.3 Raw data cleaning

Our data cleaning has two parts. The first step is to remove noisy data which has null values in a significant attribute column like 'birth-year' in 'persons'. The second step is to delete duplicated data.

What we got from IMDB is all raw data therefore we can not rely on its integrity and thus we must clean it. For instance, in the 'persons' table, our primary key is primary name and birth year. However, in the raw data, we may find some entries which do not have a birth year value. We will definitely remove this kind of data. At the same time, we assume that primary name and birth year can uniquely identify one person. So, when we have several tuples with same primary-title and birth-year, our solution is to record one of them in our database.

The way I do it is pretty straightforward. After we import all the raw data, we select the real data and generate a bright table with valid data. The data cleaning SQL script contains the query which will do the cleaning and where we store the data that is valid. Below is an example of a basic query which would remove all null values from the data.

SELECT primary-name, birth-year, death-year
FROM raw_names WHERE birth-year IS NOT NULL;

Since we have some composite keys as primary keys in our databases, we have to clean all the duplicated data which has the same primary key attributes. For example, in the 'persons' table, we want birthYear and primary-name to be the primary key. We have to remove the data for which these two attributes have the same value. The following query shows how I removed the duplicate data.

CREATE TABLE new_data
SELECT * FROM 'name_basics' N1
GROUP BY (N1.primary-name, N1.birthYear);

What's more, we clean the data in participates and acts tables that have person who is not in our persons data. Here show the way we clean those bad data and get the data of acts table.

select distinct *
from 'title_principals' old
where old.category = 'actor'
and old.characters is not null
or old.category = 'actress' AND old.characters is not null
and exists (select *

```
from surrogate_person s
where s.nconst = old.nconst)
and exists ( select *
from general_movies g
where g.tconst = old.tconst);
```

Now, all we need to do is drop the old raw data and use the cleaned data to generate our database and implement our application's functions. We can also find that after data cleaning, the size of our database decreased significantly.

3.4 Table Construction

We have solved some problems when we construct tables.

First, the performance of our system is not ideal when it comes to huge size data set like participates table. We decided to generate more sub-tables based on the type of participation like producer, writer and director. These tables keep the person id, movie id and his/her job in it.

Second, the original data has attributes with arrays, all we need to split this attribute and generate a new data table. When it comes to the professions attribute of person. The original data looks like 'actor,producer,soundtrack'. I use the 'help_topic' in mysql and here is the script I write to split these attribute and construct new table.

```
create table professions as select a.`primary_name`,
a.`birth_year`,
substring_index
(substring_index(a.profession,',',b.`help_topic_id`+1),',',-1)
from `old_professions` a
join mysql.`help_topic` b
on b.`help_topic_id` <
(length(a.profession) - length(replace(a.profession,',',''))+1)
order by a.`primary_name`,a.`birth_year`;
```

Now we are finally able to use the database to implement system functions.

4. DESIGN CHOICES

4.1 Application Design

Our original plan for application design was a simple Java server which would handle HTTP requests which would come from a web browser and using HTML and JavaScript to display the results returned. We've maintained this approach but have decided to move away from using Java as our web server to using Python with Flask as it simplifies the process. We no longer have to deal with handling individual requests, instead we can focus on making the queries and presenting the data back to the user. So really all this switch does is change the language we are working in while also providing us a framework which simplifies the process of managing a web server. We will still have to write individual queries, we will still have to write user interfaces to get data from the user, and we will still have to present the data to the user in an easy to read format. An additional reason we have switched away from Java while writing the back end is because there exists integration's between Python and R that will help make the later data mining component easier.

4.2 Query Interface Design

Our first step in designing out query interface was figuring out what queries we would support. We came up with a list of queries that seemed interesting and were complicated enough to warrant a queryable interface for them. We each generated a list of 5 queries we felt fulfilled these requirements. Then, as a team, we picked the most interesting among those and decided to work on them. We realized quite quickly that these queries would need to be on separate pages as it would simplify the input for the user, but it would also make our lives easier to as well. Rather than having to deal with multiple HTML forms on the same page, we would instead be working with a single form on a single page. We also decided it would make the most sense to take advantage of Flask's built in templating system. This allowed us to write one template which would use to display results for all of the queries.

The usage is actually rather straight forward. Our homepage has a list of English descriptions of queries and then a button that links to the page which requires you to enter the information you wish to query with. The user clicks "submit" and then they are automatically routed to a page which displays the results.

4.3 Query statement

As we have given 5 query plus our 5 query supports, our system have 10 query support provided. In order to simplify the query statement and make it perform faster. Instead of join tables in select statement, we create view in advance in some of the scenarios. At the beginning, we create a view which join the persons table with its surrogate key.

```
create view personx as
select sur.nconst, p.primary_name, p.birth_year,
p.death_year
from persons p, surrogate_person sur
where p.birth_year = sur.birth_year
and p.primary_name = sur.primary_name;
```

1. List the names of alive actors whose name starts with a given keyword (such as "Phi") and did not participate in any movie in a given year (such as 2014).

I create the view which join our participates table with movie table.

```
create view movie_participate as
select mov.movie_tconst, par.nconst, mov.release_year
from movie mov, participates par
where mov.`movie_tconst` = par.tconst;
```

Now, I have to select the alive person with given name and he/she has to be a actor/actress but do not participates in any movie in 2014.

```
select per.`primary_name`
from personx per
where per.`death_year` IS NULL
and per.`primary_name` LIKE 'Phi%'
and not exists(
select *
from movie_participate par
where par.nconst = per.nconst
and par.`release_year` = 2014)
and exists( select *
from acts a
where a.nconst = per.nconst);
```

2.List the names of alive producers who have produced more than a given number (such as 50) of talk shows in a given year (such as 2017) and whose name contains a given keyword (such as "Gill").

```
SELECT per.`primary_name` as name
FROM personx per
WHERE per.`primary_name` LIKE '%e%'
AND per.`death_year` IS NULL
```

```
AND per.nconst IN (
SELECT pro.nconst
FROM producers pro, talkshow tak
WHERE pro.tconst = tak.tconst
AND tak.'show_year' = 2014
group by pro.nconst
having count(*) > 1);
```

3.List the average runtime for movies whose original title contain a given keyword such as ("star") and were written by somebody who is still alive.

```
SELECT AVG(gen.runtime)
FROM 'general_movies' gen
WHERE title LIKE '%star%'
AND gen.type = 'movie'
AND gen.tconst IN (
SELECT tconst
FROM writers w
WHERE exists (
SELECT nconst
FROM personx per
WHERE per.'death_year' IS NULL
AND per.nconst = w.nconst));
```

4.List the names of alive producers with the greatest number of long-run movies produced (runtime greater than 120 min).

```
create view producer_movie_person as
select pro.nconst, pro.tconst, gen.title,
gen.type, gen.runtime, per.primary_name,
per.birth_year, per.death_year
from producers pro, general_movies gen, personx per
where pro.tconst = gen.tconst and pro.nconst = per.nconst;
```

Now I can select producer's name

```
select primary_name , max(runtime)
from producer_movie_person
where runtime > 120 and death_year is null
group by nconst, primary_name;
```

5.List the unique name pairs of actors who have acted together in more than a given number (such as 2) movies and sort them by average movie rating (of those they acted together).

```
create view tmp_act as
select a.nconst, a.tconst, p.primary_name
from acts a, personx p
where a.nconst = p.nconst
create view act_act as
select act1.primary_name as a1,
act2.primary_name as a2, act1.tconst
from tmp_act act1, tmp_act act2
where act1.tconst = act2.tconst
and cast(substring(act1.nconst,3) as signed)
> cast(substring(act2.nconst,3) as signed);
```

Now I select the pairs with descending rating.

```
select distinct a.a1, a.a2, p.rating
from act_act a, previous_rating p
where p.tconst = a.tconst
group by a.a1, a.a2, p.rating
having count(*) > 2
order by p.rating desc;
```

6.List the tv series with x number of episodes and which has a rating above 4 for the last 5 years.

```
select h.series_tconst, gen.title
from has h, tvepisode epi, general_movies gen
where h.episode_tconst = epi.episode_tconst
and gen.tconst = h.series_tconst
and gen.runtime = 90
and exists (
select *
from previous_rating previous
where h.series_tconst = previous.tconst and rating > 4)
group by h.series_tconst, gen.title
having count(*) > 5;
```

7.List all the movies with actor (actor name) and which are in language (English, Spanish etc).

```
Select distinct lo.local_name, per.primary_name, lo.lang
from general_movies m, personx per, localize lo, acts act
where act.nconst = per.nconst
and lo.tconst = m.tconst
and m.tconst = act.tconst
and m.type = 'movie' and lo.lang = 'en'
and per.primary_name like '%a%';
```

8.List all the producers who died before their movie was released.

```
select distinct per.primary_name
from producers pro, personx per
where pro.nconst = per.nconst
and exists (select *
from movie mov
where pro.tconst = mov.movie_tconst
and mov.release_year > per.death_year);
```

9.List all the movies where the actor and director both had the same birth year contributed to the same movie.
I will create 2 views act_name_birth and director_name_birth to keep all the birth year of actors and directors. Then I join these 2 tables to get the pairs of actor and director who contribute to the same movie. Now I get view dir_act_mov.

```
create view act_name_birth as
select p.primary_name, p.nconst, p.birth_year, act.tconst
from acts act, personx p
where p.nconst = act.nconst;
create view director_name_birth as
select p.primary_name, p.nconst, p.birth_year, dir.tconst
from directors dir, personx p
where p.nconst = dir.nconst;
create view dir_act_mov as
select dir.birth_year as dir_birth, dir.nconst as dir_nconst,
act.birth_year as act_birth,
act.nconst as act_nconst, dir.tconst
from director_name_birth dir, act_name_birth act
where dir.tconst = act.tconst;
```

Now I select the titles which are under given requirements.

```
select distinct gen.title
from dir_act_mov a, general_movies gen
where a.dir_birth = a.act_birth
and gen.tconst = a.tconst
and not a.dir_nconst = a.act_nconst;
```

10. List all the shows which have a total run time less than a particular value.

```
select mov.title
from general_movies mov , generes gen
where mov.tconst = gen.tconst
and gen.genre like '%show%'
and mov.runtime < 10
```

5. APPLICATION DEMO

The first thing which you get dropped into is the homepage. As you can see in the figure 2 you are given the list supported queries which you can access by clicking the button.

When clicking the first query button, you're presented with two text fields, as shown in figure 3. These are the parameters of the query, so in this case, the results returned will contain the list of actors whose names start with the keyword "Phi" and didn't participate in a movie for the year 2014.

Figure 4 shows the results of the query shown in figure 3. You can quickly navigate home by clicking the home button.

From here on out, we will just show the search parameter page and then the results from the query.

5.1 Remaining Screenshots

6. WORK LOAD DISTRIBUTION

The workload was not entirely uniform but everyone did their best to contribute at each step of the process. The initial design of the database was handled primarily by Yifei and Zhou. They worked together to create our ER models. From there, Yifei and Richard worked together to handle normalization, essentially they analyzed the ER model, translated that model into a relationship model, and then verified that we were in BCNF. All while this was going on Ajeeta and Zhou worked on the first paper. Ajeeta worked to make the template which was used to display all of our results. We all worked together to write the queries since as we all went back and forth and help each other when one of us got stuck.

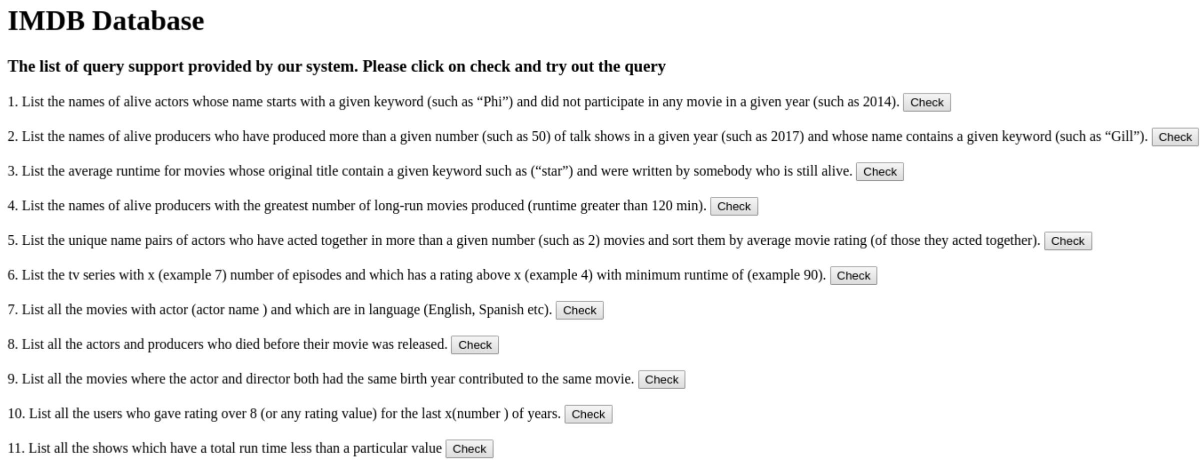

Figure 2: Home page of the website

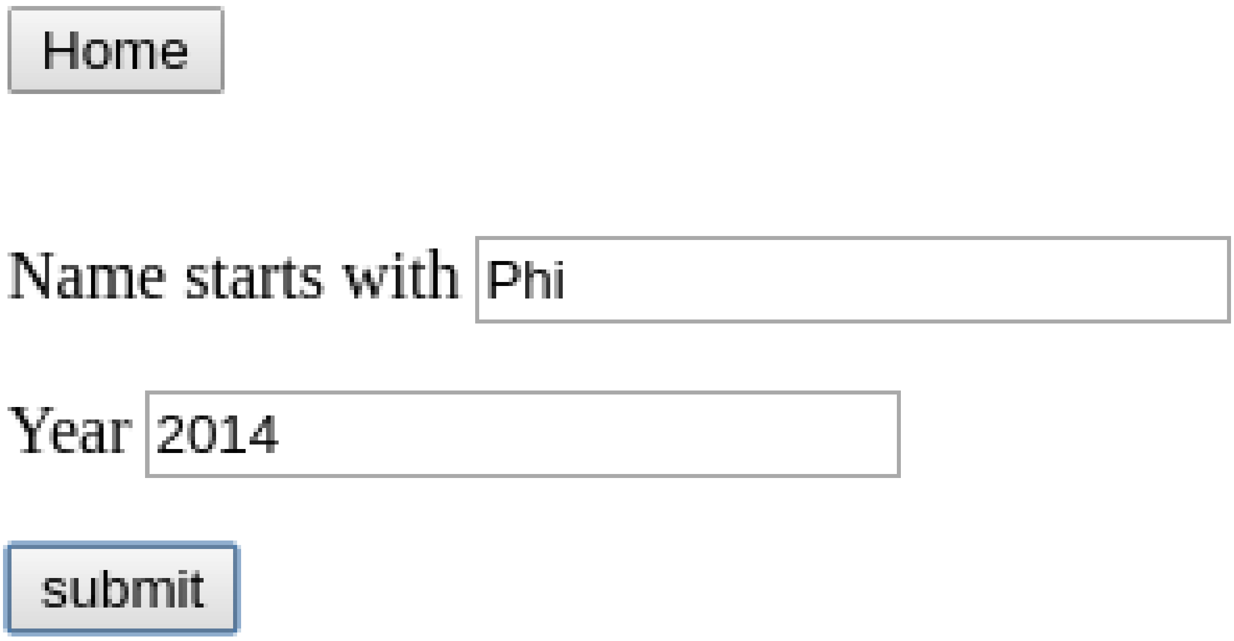

Figure 3: Query 1 Search parameter input

Actor Name
Phi Nguyen
Phi Nhung
Phil Abrams
Phil Adams
Phil Addis
Phil Agland
Phil Allora
Phil Amato
Phil Angelides
Phil Armijo
Phil Arthur
Phil Baker
Phil Balsley
Phil Barney
Phil Baron
Phil Baroni
Phil Blevins
Phil Bloom
Phil Bonifield
Phil Boot
Phil Boroff
Phil Bourassa
Phil Bradley
Phil Bredesen
Phil Brock
Phil Bronstein
Phil Brookes
Phil Brough

Figure 4: Query 1 Results

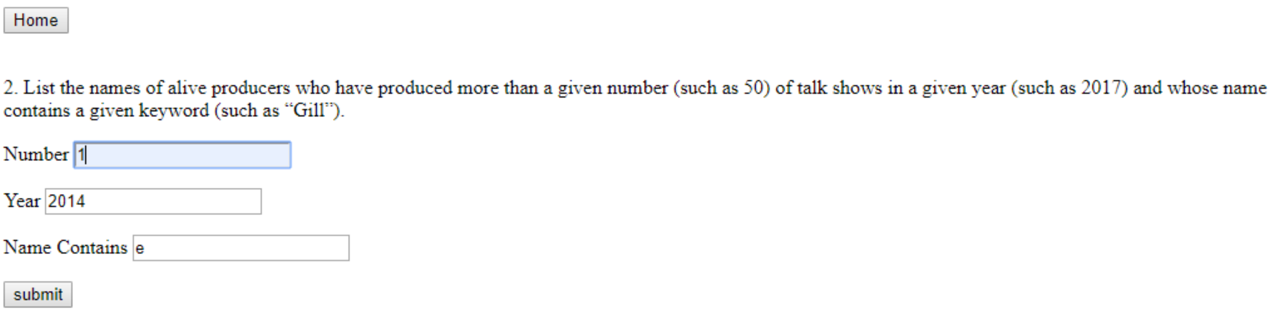

Figure 5: Query 2 search parameter input

Figure 6: Query 2 results

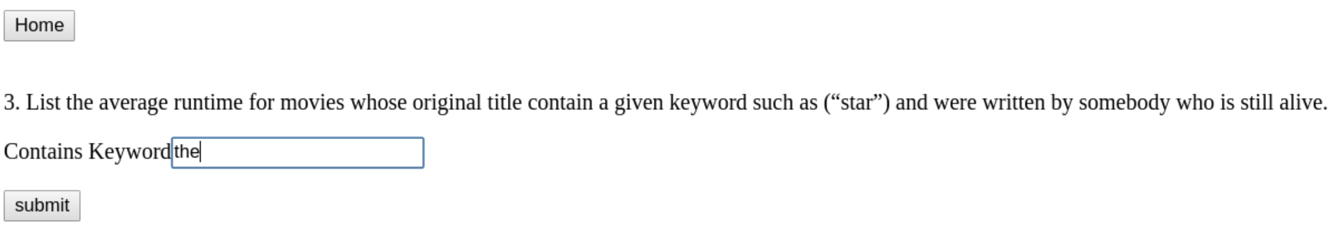

Figure 7: Query 3 search parameter input

Home

Runtime
50.0000

Figure 8: Query 3 results

Home

4. List the names of alive producers with the greatest number of long-run movies produced (runtime greater than 120 min).

submit

Figure 9: Query 4 search parameter input

Home

Runtime	
Kevin Brownlow	150
Harry Grossman	310

Figure 10: Query 4 results

Home

5. List the unique name pairs of actors who have acted together in more than a given number (such as 2) movies and sort them by average movie rating (of those they acted together).

Number of movies 2

submit

Figure 11: Query 5 search parameter input

Home

Actor Name 1	Actor Name 2	Rating
Jerry Seinfeld	Julia Louis-Dreyfus	9.6
David Duchovny	Gillian Anderson	9.2
Woody Harrelson	Matthew McConaughey	9.2
Terry Farrell	Michael Dorn	9.1
Jerry Seinfeld	Julia Louis-Dreyfus	9.1
David Duchovny	Gillian Anderson	9.1
Jonathan Frakes	Michael Dorn	9.0
Lacey Chabert	Neve Campbell	9.0
Jerry Seinfeld	Julia Louis-Dreyfus	9.0
Edward Asner	Hank Azaria	9.0
David Duchovny	Gillian Anderson	9.0
Kirsten Dunst	Ewan McGregor	9.0
Billy Bob Thornton	Ewan McGregor	9.0
Rose McGowan	Alyssa Milano	8.9
Jerry Seinfeld	Julia Louis-Dreyfus	8.9
Terry Farrell	Michael Dorn	8.9
Jonathan Frakes	Michael Dorn	8.9
David Duchovny	Gillian Anderson	8.9
Lili Taylor	Timothy Hutton	8.9
Jerry Seinfeld	Julia Louis-Dreyfus	8.8
Ron Perlman	Linda Hamilton	8.8
Rose McGowan	Alyssa Milano	8.8
Lacey Chabert	Neve Campbell	8.8
Robert Duncan McNeill	Robert Beltran	8.8
Kate Mulgrew	Robert Beltran	8.8
Kate Mulgrew	Robert Duncan McNeill	8.8
David Duchovny	Gillian Anderson	8.8
Michael Madsen	Harvey Keitel	8.8
Tim Roth	Harvey Keitel	8.8
Tim Roth	Michael Madsen	8.8
Robin Wright	Kevin Spacey	8.8
Diane Lane	Tommy Lee Jones	8.7
Robert Duvall	Tommy Lee Jones	8.7
Robert Duvall	Diane Lane	8.7
Danny Glover	Tommy Lee Jones	8.7
Danny Glover	Diane Lane	8.7
Danny Glover	Robert Duvall	8.7
Rose McGowan	Alyssa Milano	8.7
Lacey Chabert	Neve Campbell	8.7
Jerry Seinfeld	Julia Louis-Dreyfus	8.7
Jonathan Frakes	Michael Dorn	8.7
David Duchovny	Gillian Anderson	8.7
James Arness	Bruce Boxleitner	8.7
Stephen Fry	Rowan Atkinson	8.6
Jonathan Frakes	Michael Dorn	8.6
Gates McFadden	Michael Dorn	8.6
Gates McFadden	Jonathan Frakes	8.6
Marina Sirtis	Michael Dorn	8.6
Marina Sirtis	Jonathan Frakes	8.6
Brent Spiner	Michael Dorn	8.6
Brent Spiner	Jonathan Frakes	8.6
Brent Spiner	Gates McFadden	8.6
Brent Spiner	Marina Sirtis	8.6
Wil Wheaton	Michael Dorn	8.6
Wil Wheaton	Jonathan Frakes	8.6
Wil Wheaton	Gates McFadden	8.6
Wil Wheaton	Marina Sirtis	8.6
Wil Wheaton	Brent Spiner	8.6
Majel Barrett	Michael Dorn	8.6
Majel Barrett	Jonathan Frakes	8.6
Majel Barrett	Gates McFadden	8.6
Majel Barrett	Marina Sirtis	8.6
Majel Barrett	Brent Spiner	8.6
Majel Barrett	Wil Wheaton	8.6
David Duchovny	Gillian Anderson	8.6
Robert Duncan McNeill	Robert Beltran	8.6
Kate Mulgrew	Robert Beltran	8.6
Kate Mulgrew	Robert Duncan McNeill	8.6
Ron Perlman	Linda Hamilton	8.6
Clancy Brown	Adrienne Barbeau	8.6
Rose McGowan	Alyssa Milano	8.6
Ron Perlman	Mark Hamill	8.6
Powers Boothe	Clancy Brown	8.6
Bess Armstrong	Claire Danes	8.6

Figure 12: Query 5 results

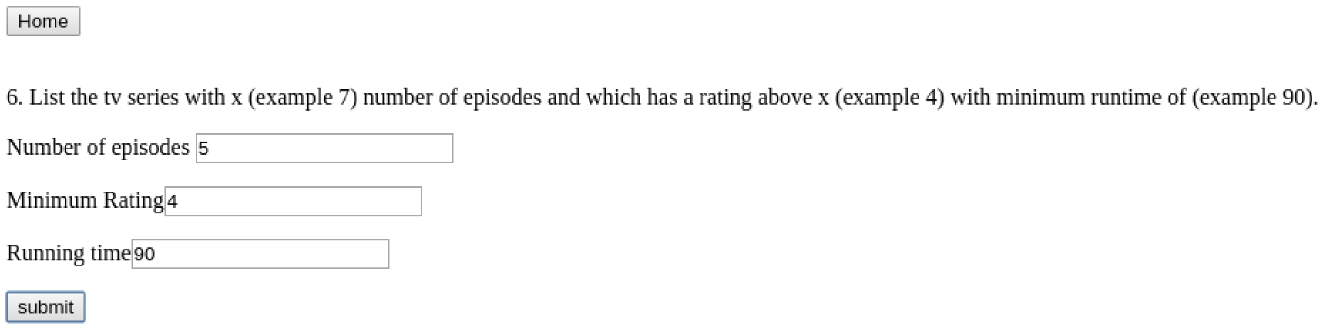

Figure 13: Query 6 search parameter input

Title
Your Show of Shows
Producers' Showcase
Ford Star Jubilee
Playhouse 90
Bandstand
The DuPont Show of the Month
Arrest and Trial
Another World
Cimarron Strip
The Joey Bishop Show
The Name of the Game
The David Frost Show
The Dick Cavett Show
The Pink Panther Show
Longstreet
Banacek
Hec Ramsey
Madigan
Hawkins
Movin' On
Saturday Night Live
Switch
Bergerac
SCTV Network 90
Matt Houston
Die Schwarzwaldklinik
Navarro
Rivalen der Rennbahn
One Foot in the Grave

Figure 14: Query 6 results

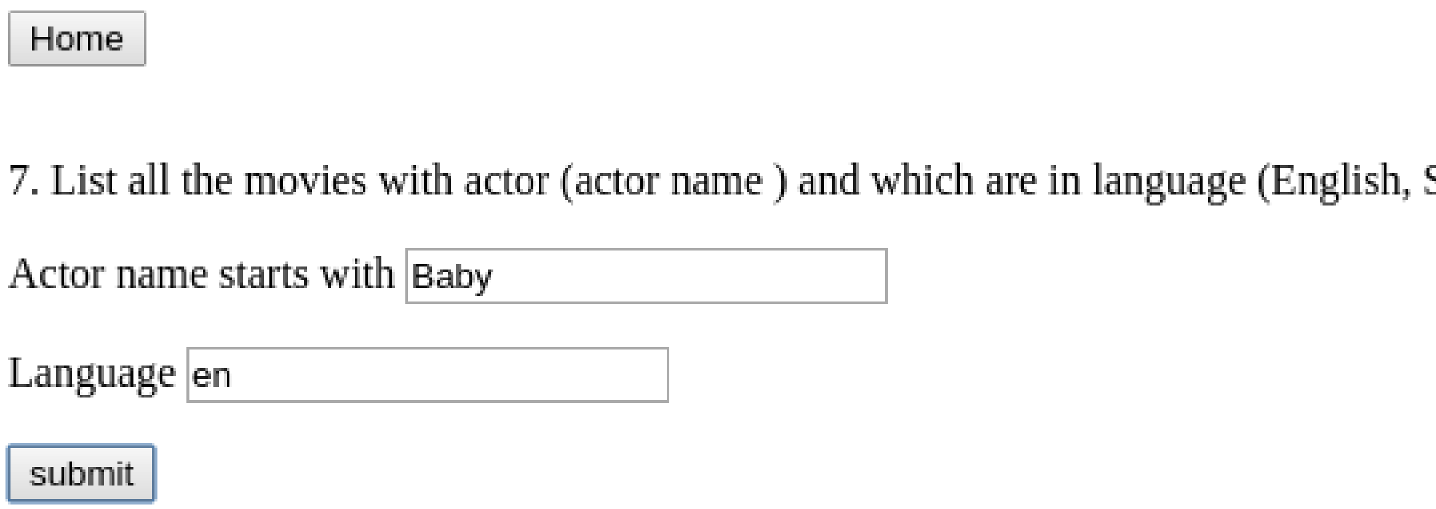

Figure 15: Query 7 search parameter input

Home

Title	Name	Language
The Viking	Bob Bartlett	en
The Carson City Kid	Bob Steele	en
Road to Morocco	Bob Hope	en
Nazty Nuisance	Bobby Watson	en
Nazty Nuisance	Bobby Watson	en
They Got Me Covered	Bob Hope	en
Belle of the Yukon	Bob Burns	en
The Princess and the Pirate	Bob Hope	en
Bud Abbott and Lou Costello in Hollywood	Bob Haymes	en
Live Wires	Bobby Jordan	en
My Favorite Brunette	Bob Hope	en
Where There's Life	Bob Hope	en
Der Apfel ist ab	Bobby Todd	en
The Fallen Idol	Bobby Henrey	en
Hallo, FrÃ¤ulein!	Bobby Todd	en
The Window	Bobby Driscoll	en
The Lemon Drop Kid	Bob Hope	en
Road to Bali	Bob Hope	en
Casanova's Big Night	Bob Hope	en
That Certain Feeling	Bob Hope	en
Beau James	Bob Hope	en
Gun Duel in Durango	Bobby Clark	en
China Doll	Bob Mathias	en
Alias Jesse James	Bob Hope	en
The Beatniks	Bob Wells	en
Bachelor in Paradise	Bob Hope	en
Hell Is for Heroes	Bobby Darin	en
Pressure Point	Bobby Darin	en
The Road to Hong Kong	Bob Hope	en
McHale's Navy	Bob Hastings	en

Figure 16: Query 7 results

Home

See the results

Figure 17: Query 8 search parameter "input"

Home

Producer Name
David Carradine
Sydney Pollack
Tony Scott
Ed Vassallo
Claude Baks
Socrates Ballis
Humbert Balsan
Marc Beaudet
Christian Blackwood
Roger Blanc
David Bombyk
Jim Booth
Larry Brezner
Rafi Bukai
Frank Capra Jr.
Béatrice Caufman
Mario Cecchi Gori
René Cleitman
Larry Darmour
Francisco del Villar
Nancy Dickerson
Paolo Ferrara
Gil Friesen
Fernando de Fuentes hijo
Fernando de Fuentes
Alejandro Galindo
Georges Glass
Frank Glicksman
Leslie Grantham
Harold Greenberg
Jesús Grovas
Samuel Hadida
Kirk Edward Hansen
Olle Hellbom
S. Hillkowitz
Gregg Hoffman
Hans Hömberg
Thomas H. Ince
Peter Jamison
Otto Kahn
Mikheil Kalatozishvili
Ferenc Kardos
István Kardos
Kirin Kiki
Julie Kirkham
Werner Koenig
Óscar Kramer
Gulshan Kumar
Franklin R. Levy
Amilcar Lyra
A.V. Meiyappan
Robert de Nesle
Robert F. Newmyer
Milan Nikolic
Grigori Nikulin
Nutan
James Paris
J.G. Patterson Jr.
René Pignières
John E. Quill
Raoul N. Rizik
Mark Rosenberg
Micha Shagrir
Sándor Simó
Mary Craig Sinclair
Upton Sinclair
Ivan Solovyov
John Spotton
Dawn Steel
Joe Strummer
Andreas Thiel
Viktor Tregubovich
Paul Vatelli

Figure 18: Query 8 results

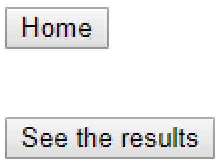

Figure 19: Query 9 search parameter ”input”

Home

Title
Buffalo Bill and the Indians, or Sitting Bull's History Lesson
Quintet
Inglourious Basterds
Fargo
The Big Lebowski
Heat
Awakenings
Blow
Where the Truth Lies
Devil's Knot
Panic Room
The Frighteners
Red Corner
Three Christs
A Real Woman
Hannibal
Heaven & Earth
The Package
Under Siege
The Fugitive
Come Early Morning
Mindhunters
Bobby
Chloe
Scarface
Carlito's Way
Batman Returns
Dark Shadows
Ghosts of Mississippi
Present Tense, Past Perfect
Havana
Le grand bleu
Beetlejuice
Alter Ego
One Small Step
The Big Chill
Ain't We Got Fun
I Wanna Be a Sailor
Little Red Walking Hood
Picador Porky
Porky's Duck Hunt
Porky's Garden
Cinderella Meets Fella
Daffy Duck & Egghead
Daffy Duck in Hollywood
A Feud There Was
The Mice Will Play
Detouring America
Screwball Football
Ceiling Hero
Cross Country Detours
The Early Worm Gets the Bird
A Gander at Mother Goose
Woody Woodpecker and His Friends
Pilot
Penguin One, Us Zero
Michael McKean/Chaka Khan/The Folksmen
Ed Asner/The Kinks
Household Saints
The Comedian
Ruthless People
The Musketeer
Zodiac
Lincoln
The Color Purple
Superman
Superman II
Aria
The Beautiful Deception
Stray Dogs
Siblings
The Banquet of Chestnuts
The House of the Spirits
Night Train to Lisbon

Figure 20: Query 9 results

Home

10. List all the shows which have a total run time less than a particular value

Runtime in hours 10

submit

Figure 21: Query 10 search parameter input

Home

Title	Runtime
Americana	30
America's Town Meeting	60
Author Meets the Critics	30
Break the Bank	30
Movieland Quiz	30
On the Corner	30
We, the People	30
Blind Date	30
The Eyes Have It	30
Easy Aces	15
Hold It Please	30
Kay Kyser's Kollege of Musical Knowledge	60
Leave It to the Girls	30
Stump the Authors	30
Sunday at Home	15
Think Fast	30
This Is Show Business	30
Abe Burrows' Almanac	30
American Forum of the Air	30
Answer Yes or No	30
Beat the Clock	30
By Popular Demand	30
Can You Top This	30
Chance of a Lifetime	30
The Faye Emerson Show	15
Pantomime Quiz	30
Penthouse Party	30
Star of the Family	30
Take a Chance	30

Figure 22: Query 10 results