

Final Report

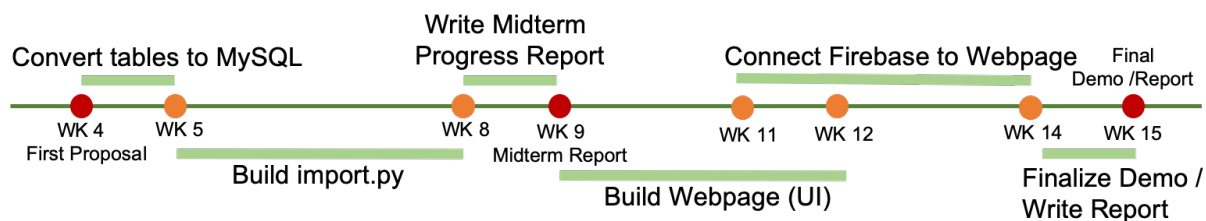
Keyword-Driven Exploration of Relational Data using Firebase

Project Idea

Initially, we intended on using History of Baseball and F1 Races databases found on Kaggle in addition to World database for this project. However, after exploring and working with the two sport databases both in MySQL and Firebase, we reached the conclusion that it is best for us to replace F1 with Sakila database for the sake of simplicity and efficiency, explained later in Components-a.

With this project, we hoped to build a website interface that allows users to search for a specific key word, case insensitive. Once the keyword is input, the user will be shown with relevant results that either have the keyword as a whole or, in the case of a two-word input, contain part of the keyword.

Milestones



Components

a. Databases

As mentioned above, the databases used right now are not our first picks. The main reason we decided to use Sakila database instead of F1 is because F1 contains too much statistic rather than a balanced mix of words and numbers. While we could have kept F1 database and replace Baseball database instead due to the large amount of data it possesses, our decision to switch out F1 database also stems from the lack of relational schema presented. This means that we had to create the schema ourselves by exploring the database, which ended up taking up too much of our time just by trying to make sense of the data.

Since there is a large amount of dataset to be processed, we decided on picking three to four tables per database to build the website on instead of using all 23 tables in Baseball database or all categories from Sakila. The reason for this is to ensure that we can test the website and all its components quickly and can make adjustments and test them when needed.

Tables 'teams', 'teamfranchises', 'divisions' and 'leagues' are tables picked from Baseball database. Sakila database consists of tables about films and relevant details, including film sales, store records and film descriptions. Our team decided to use 'film', 'actor' and 'film-actor' tables. This pick specifically will showcase the many-to-many relationships that other databases might not have.

Below are Figure(Fig.) 1, 2 and 3 representing relational schema in World, Baseball and Sakila databases.

Figure 1 World Schema

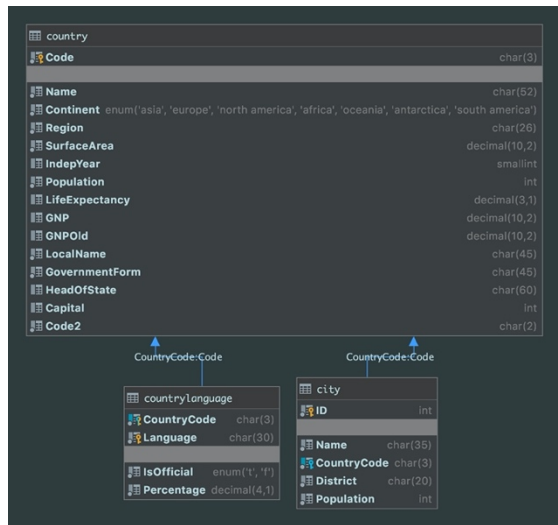


Figure 2 Baseball Schema

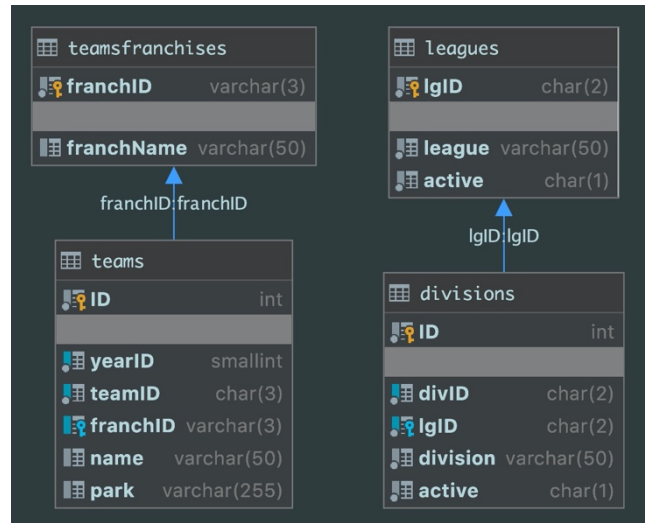
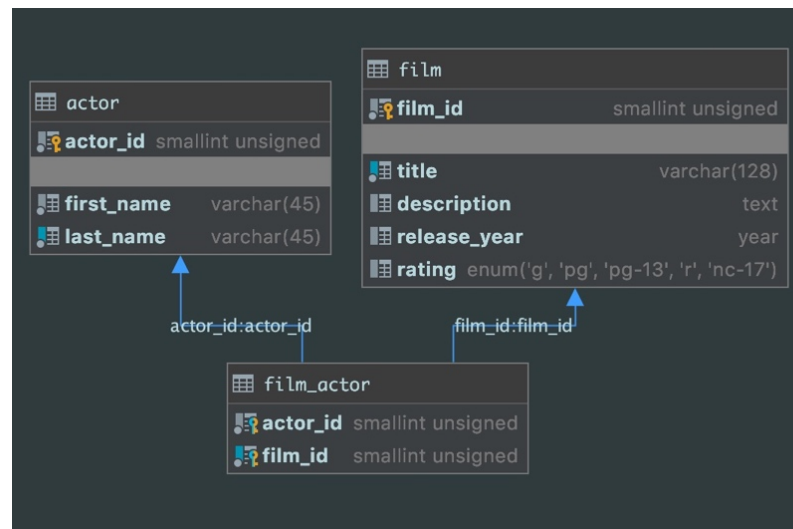


Figure 3 Sakila(Film/Actor) Schema



b. Import.py and Firebase

In this next step, we wrote a Python file that connects Firebase to our MySQL server (using username "inf551" and password "inf551") to export all tables and upload them to

Firebase. The file is also written so that an index will be created after the tables from a database are uploaded. The scripts for these processes are displayed below in Fig.4 and Fig.5

This import will generally work for any SQL files given that the username and password of the server are set to be the same as those mentioned above.

```
idx = index
for pk, dic in dictionary.items():
    primary = pk
    # if str(pk).isnumeric():
    #     pass
    # else:
    #     idx[pk] = [{"table": table, "column": 'primary', "primary": primary}]
    for k, v in dic.items():
        if v is None:
            pass
        else:
            if type(v) == str:
                v = clean(v)
                words = v.lower().split()
                for w in words:
                    w = str(clean(w))
                    if w.split() == '' or is_number(w):
                        pass
                    else:
                        if w in idx.keys():
                            idx[w].append({"table": table, "column": k, "primary": primary})
                        else:
                            idx[w] = [{"table": table, "column": k, "primary": primary}]
            else:
                pass
```

Figure 4 Create Index

```
# get table names
cursor.execute('show tables')
tables = []
for x in cursor:
    tables.append(str(x)[2: -3])

# tables to firebase
index = {}
for table in tables:
    cursor.execute("DESC " + table)
    k = cursor.fetchall()

    keys = []
    for i in k:
        keys.append(i[0])

    table_ref = ref.child(table)
    dictionary = {}
    cursor.execute('SELECT * FROM ' + table)

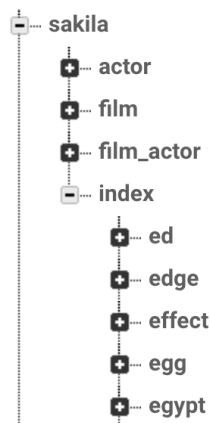
    for value in cursor:
        dic = {}
        for i in range(0, len(value)):
            dic[keys[i]] = clean(value[i])
            dictionary[clean(value[0])] = dic

    table_ref.update(dictionary)

index = add_index(dictionary, table, index)
index_ref = ref.child('index')
index_ref.update(index)
```

Figure 5 MySQL to Firebase

We decided to exclude numbers from the indexes because there were a larger amount of numbers and the data size was too big to be uploaded.



c. User Interface & Word Search

We intended for our website interface to be straight-forward and user-friendly. When accessing our site, from left to right, the user will see a drop-down button, search bar and a search icon.

To look up a word, the user first has to select a database, then type in any keyword. If the keyword is not in the database, there will be a pop-up window alerting the user of the error, prompting him to try a different word.

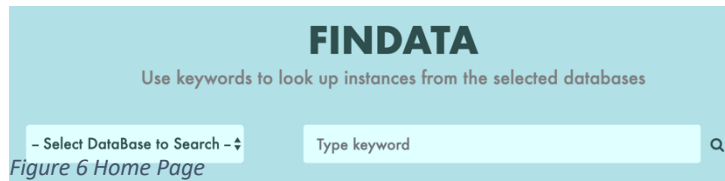


Figure 6 Home Page

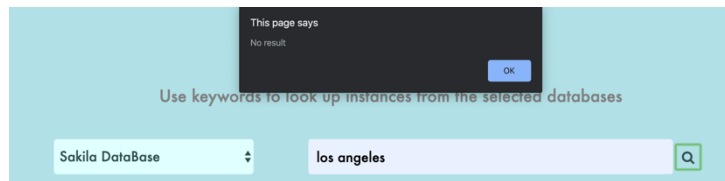


Figure 7 Example No Result

When the website recognizes a keyword or sequence of keywords presented in a selected database, it will give the user a series of outputs in result tables form, along with the

Figure 8 Example

Figure 9 Example

time taken to perform that search. Results with more matching values will be displayed first. Fig. 8 and 9 show examples of such process.

The website will also let the user know if the keyword appears in other tables. An example is shown in Fig. 10 below. The search word is 'ana', which is in both 'teams' and 'teamsfranchises' tables.

Figure 11 Example

d. Data Exploration

User can click on any given word in a column or row to see results for that specific word. For example, in Fig.11, user clicks on column 'name' on the row with words 'Los Angeles Angels'. The website will show him results for words in this specific cell that he clicked, shown in Fig.13. The same is done for Sakila database when user click PG in rating (Fig.14)

teams Showing 121 rows for: los,angeles,angels

ID	franchiseID	name	park	teamID	yearID
1320	LAD	Los Angeles Dodgers	Los Angeles Memorial Coliseum	LAA	1958
1336	LAD	Los Angeles Dodgers	Los Angeles Memorial Coliseum	LAA	1959
1332	LAD	Los Angeles Dodgers	Los Angeles Memorial Coliseum	LAA	1960
1369	LAD	Los Angeles Dodgers	Los Angeles Memorial Coliseum	LAA	1961
2446	ANA	Anaheim Angels	Angels Stadium of Anaheim	ANA	2004
1368	ANA	Los Angeles Angels	Wrigley Field LA	LAA	1961
1367	ANA	Los Angeles Angels	Dodger Stadium	LAA	1962
1407	ANA	Los Angeles Angels	Dodger Stadium	LAA	1963
1427	ANA	Los Angeles Angels	Dodger Stadium	LAA	1964
2489	ANA	Los Angeles Angels of Anaheim	Angel Stadium	LAA	2005

Showing 1 to 10 of 121 entries

Previous 1 2 3 4 5 ... 13 Next

teamsfranchises Showing 2 rows for: los,angeles,angels

franchiseID	franchiseName
ANA	Los Angeles Angels of Anaheim
LAD	Los Angeles Dodgers

Showing 1 to 2 of 2 entries

search time: 650.932000016764 milliseconds

Figure 13 Data Exploration-Baseball

Sakila DataBase cat

film Showing 417 rows for: pg

description	film_id	rating	release_year	title
A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	1	PG	2006	
A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	6	PG	2006	
A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	7	PG 13	2006	
A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	9	PG 13	2006	
A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia	12	PG	2006	
A Action Packed Drama of a Dentist And a Crocodile who must Battle a Feminist in The Canadian Rockies	13	PG	2006	
A Thoughtful Drama of a Composer And a Feminist who must Meet a Secret Agent in The Canadian Rockies	18	PG 13	2006	
A Emotional Display of a Pioneer And a Technical Writer who must Battle a Man in A Saloon	19	PG	2006	
A Touching Panorama of a Waitress And a Woman who must Outrace a Dog in An Abandoned Amusement Park	28	PG 13	2006	
A Action Packed Reflection of a Crocodile And a Explorer who must Find a Sumo Wrestler in An Abandoned Mine Shaft	33	PG 13	2006	

Showing 1 to 10 of 417 entries

Previous 1 2 3 4 5 ... 42 Next

search time: 681.4549999956542 milliseconds

Figure 14 Data Exploration-Sakila

e. Connecting front and back end

This is the most challenging part for us as a team so far, as Javascript is not something we are familiar with. During our demo presentation, we still had yet to figure out how to implement foreign key exploration by letting the user click on any foreign key available that can lead him to a different result page (Fig.15)

Since the demo, not only have we figured out how to structure and present our result better, we also figured out how to make data exploration possible.

FINDATA Use keywords to look up instances from the selected databases

World Database los angeles

117	San Nicolás de los Arroyos	ARG	Buenos Aires	119302	city
568	Los Angeles	CHL	Biobío	158215	city
569	Puerto Montt	CHL	Los Lagos	152194	city
571	Osorno	CHL	Los Lagos	141468	city
574	Valdivia	CHL	Los Lagos	133106	city
588	Santiago de los Caballeros	DOM	Santiago	365463	city
597	Santo Domingo de los Colorados	ECU	Pichincha	202111	city
603	Quevedo	ECU	Los Rios	129631	city
2542	San Nicolás de los Garza	MEX	Nuevo León	495540	city
2593	Chilpancingo de los Bravo	MEX	Guerrero	192509	city
2660	Los Cabos	MEX	Baja California Sur	105199	city
3558	Los Teques	VEN	Miranda	178784	city
3794	Los Angeles	USA	California	3694820	city
3963	East Los Angeles	USA	California	126379	city
568	Los Angeles	CHL	Biobío	158215	city
793	Angeles	PHL	Central Luzon	263971	city
117	San Nicolás de los Arroyos	ARG	Buenos Aires	119302	city

Figure 15 Example-1st Ver.

Performance Analysis

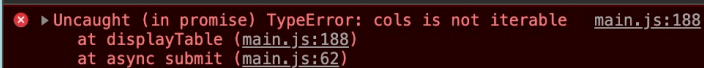
a. Query Process

When user inputs keywords, the keywords, which are strings, will be cleaned, getting rid of numbers, punctuation and changing any upper to lower case. For each substring of input, our query will then search the database index to find the table and the primary key of its occurrence. This means that a dictionary is created (tableName: [pk1, pk2,...]) for each substring.

The site then retrieves data from `firebase(db.ref(database).child(table).child(pk))` and sorts data in a way that rows containing more substring of input goes first. It then waits for all the data to be loaded, and then creates tables when done. These tables will then be loaded on to the website.

b. Performance

In order to process data in javascript, we need to build asynchronous functions to fetch the entire data onto a variable, and after confirming with the Sakila database, we came to a conclusion that it takes roughly 300ms to retrieve 1,000 rows



```
✖ ▶ Uncaught (in promise) TypeError: cols is not iterable    main.js:188
    at displayTable (main.js:188)
    at async submit (main.js:62)
```

c. Exploration

Same process as above, but keywords are taken from selected columns (as mentioned above), not the input string.

Responsibilities

Originally, we agreed on working on `import.py` together and having the more experienced coder work on the interface. However, the roles changed as we went ahead with the project.

a. Lena

While `import.py` was done together, Lena took up the responsibility of improving the script, making it run smoothly when dealing with large data sets. She also took over Javascript after the demo and made the website a more feasible product as Phuong took too much time figuring out Javascript herself.

b. Phuong

Phuong originally was in charge of putting the deliverables together in addition to contributing to `import.py`. However, she later worked on building the interface with HTML and CSS, and also worked on the initial version of the Javascript connecting front

and back end together. She later handed it over to Lena to work on the rest and improving the site's functions.

Conclusion

As we wrapped up this project, we realized that we have much more to learn about data structure and management. If we get the chance to, we would like to improve this project to enable it working and handling multiple large data sets. This is one of the setbacks that we encountered during the process of doing this project.