

Report of the Web Application

Project Idea

Our project builds a Web Application, which supplies users with a keyword-driven interface and supports basic search operation and exploration into data via the foreign-key relations among tables. Under the hood, the search operation sorts the tuples from different tables according to the number occurrences of keywords; the exploration into data is based on the foreign-key relations.

Overview

DATASET

1. “World”¹ Database, consists of country, city and countrylanguage tables;
2. “Films & Actors”² Database, consists of film, language, film_actor and actor tables;
3. “Customers’ Order”³ Database, consists of products, productlines, orders, orderdetails and customers tables.

DATA MANAGEMENT

Firebase REALTIME database

BACK-END FRAMEWORK

Flask

FRONT-END DESIGN

HTML CSS

Screenshots of components

1) Data Schemas

a. “World” Database

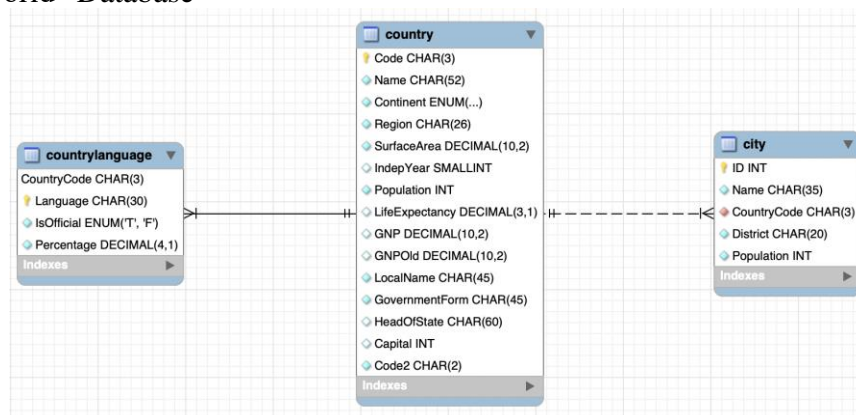


Table	Foreign Keys	References
city	countrycode	country (code)

¹ <https://dev.mysql.com/doc/world-setup/en/>

² <https://dev.mysql.com/doc/sakila/en/>

³ <https://www.mysqltutorial.org/mysql-sample-database.aspx>

country	-	-
countrylanguage	countrycode	country (code)

b. “Films & Actors” Database

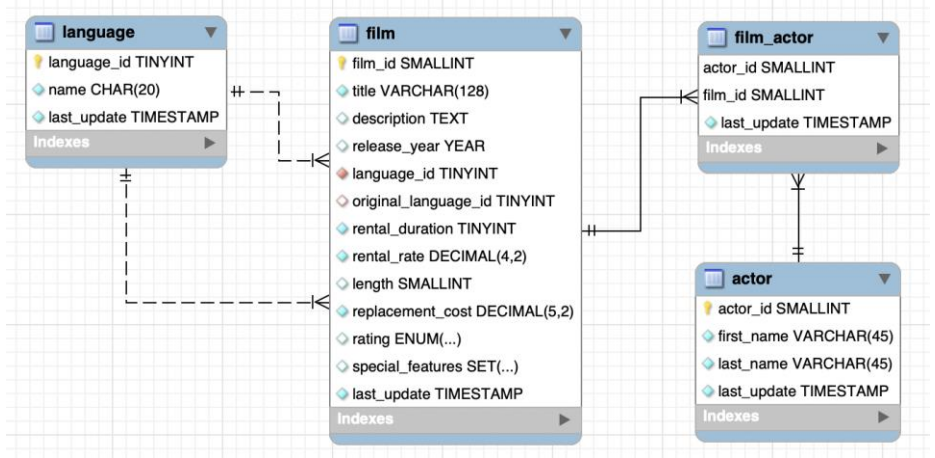


Table	Foreign Keys	References
language	-	-
film	language_id	language (language_id)
	origin_language_id	language (lanauge_id)
film_actor	actor_id	actor (actor_id)
	film_id	film (film_id)
actor	-	-

c. “Customers’ Order” Database

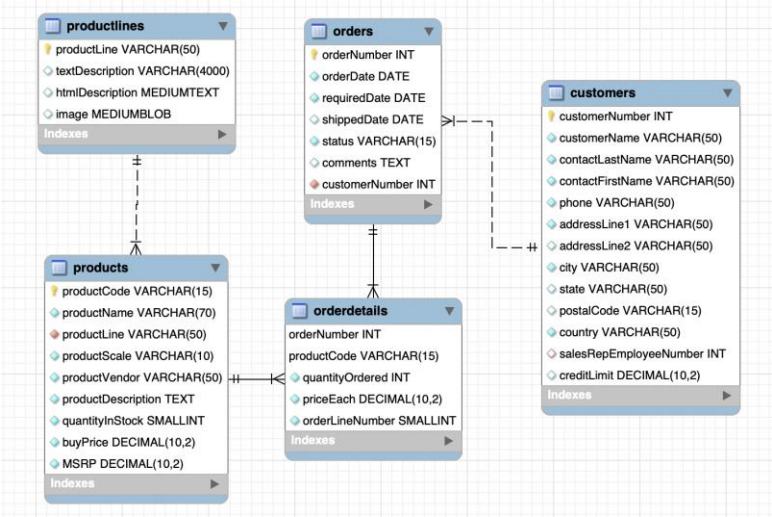
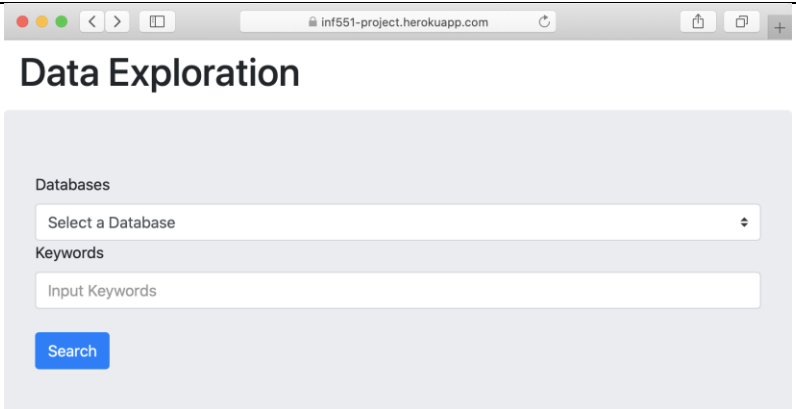


Table	Foreign Keys	References
customers	-	-
orderdetails	orderNumber	orders (orderNumber)
	productCode	products (productCode)
products	productLine	productlines (productLine)
orders	customerNumber	customers (customerNumber)
productlines	-	-

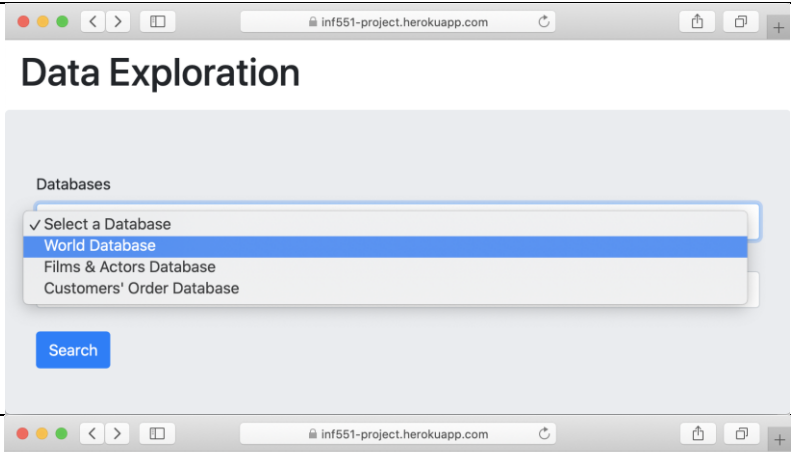
2) Firebase Data

World	Films & Actors	Customers' Order
<ul style="list-style-type: none"> World <ul style="list-style-type: none"> city country countrylanguage World_index World_schema <ul style="list-style-type: none"> city country countrylanguage 	<ul style="list-style-type: none"> FilmsActors <ul style="list-style-type: none"> actor film film_actor language FilmsActors_index FilmsActors_schema <ul style="list-style-type: none"> actor film film_actor language 	<ul style="list-style-type: none"> CustomersOrder <ul style="list-style-type: none"> customers orderdetails orders productlines products CustomersOrder_index CustomersOrder_schema <ul style="list-style-type: none"> customers orderdetails orders productlines products

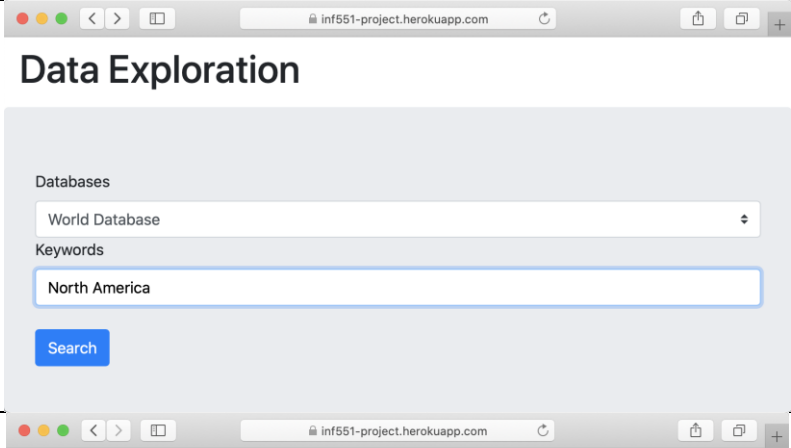
3) App Interface

Index Page	
------------	--

Select a Database



Input Keywords



Search Result

Table Name: city

ID	Name	CountryCode	District	Population
752	Rustenburg	ZAF	North West	97008
751	Potchefstroom	ZAF	North West	101817
732	Klerksdorp	ZAF	North West	261911
469	Belfast	GBR	North Ireland	287500
4077	Jabaliya	PSE	North Gaza	113901
4050	Qana	USA	North Carolina	21312

Foreign Key Page

Search

Table Name: country

Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy
ZAF	South Africa	Africa	Southern Africa	1221037.00	1910	40377000	51.1

Implementation Details

1) Importing data from MySQL to Firebase

- a) Query Data Schema
 1. Retrieve columns names of each table by querying on INFORMATION_SCHEMA;
 2. Retrieve foreign-key constraints of each table by joining INFORMATION_SCHEMA and INFORMATION.KEY_COLUMN_USAGE;
 3. Put the data to Firebase under <db>/<table_schema> node.
- b) Retrieve Data
 1. Transform data of each table into JSON, inside which the keys is a string that concatenates the primary key(s) of each tuple;
 2. Put the data to Firebase under <db>/<table>.
- c) Build Index
 1. Iterate on all the words;
 2. Regularize all the words to compatible with the requirements of Firebase key;
 3. Record occurrences of each word by triples (<table>, <column> and <primary key(s)>);
 4. Put the data to the Firebase under <db>/<db_index>.

2) Front-end Design

As we specified before, we are building a web-based application supporting search and exploration on the data with respect to database.

- a. Selection box: select a database from which users are going to explore the data;
- b. Keyword box: input the keywords;

HTML is used to design the front-end page and bootstrap is used to prettify the result.

3) Back-end Design

Flask is used to design back-end and interact with front-end interface and Firebase REALTIME Database.

4) Search

Tuples are sorted by the relevance to keywords and the number of occurrences. The tuples where the keywords cluster inside fewer attributes with more occurrences are ranked higher. That is, when “A B C” is search, the tuples where “A”, “B” and “C” appear in the same attributes are going to be ranked before those where “A”, “B” and C” do not; moreover, the tuples with same relevance to the keywords will be sorted further according the number of the occurrences with respect to “A”, “B” and “C”.

For example, when “South America” is searched:

BOL	Bolivia	South America	South America	1098581.00	1825	8329000	63.7	8571.00	7967.00	Bolivia	Republic	Hugo Bánzer Suárez	1
ARG	Argentina	South America	South America	2780400.00	1816	37032000	75.1	340238.00	323310.00	Argentina	Federal Republic	Fernando de la Rúa	6
SGS	South Georgia and the South Sandwich Islands	Antarctica	Antarctica	3903.00				0.00		South Georgia and the South Sandwich Islands	Dependent Territory of the UK	Elisabeth II	
ZAF	South Africa	Africa	Southern Africa	1221037.00	1910	40377000	51.1	116729.00	129092.00	South Africa	Republic	Thabo Mbeki	7
USA	United States	North America	North America	9363520.00	1776	278357000	77.1	8510700.00	8110900.00	United States	Federal Republic	George W. Bush	8
SPM	Saint Pierre and Miquelon	North America	North America	242.00		7000	77.6	0.00		Saint-Pierre-et-Miquelon	Territorial Collectivity of	Jacques Chirac	9

Even though both “SGS” and “ARG” have 4 occurrences of “South America”, “ARG” has “South” and “America” appear as a whole in the same column. Thus, “ARG” will be ranked higher.

Performance analysis on query processing & exploration process

1) Query data by keyword(s)

- Retrieve all the indices from to the corresponding index node on Firebase;
- Save keywords of different primary key as a key into dictionary;
- Rank the primary key according to how many times the keywords appear in total, how many times for each key, and what keyword each column contains;
- Divide the result into three part:
 - The tuples that contain all the keywords in the same column;
 - The tuples that contain all the keywords but not in the same column;
 - The tuples that only contain part of the keywords.
- Sort tuples within each part;
- Return the tuples in the expected order to the front-end interface;

Complexity of query data is $O(N)$; complexity of sorting the data takes $O(N \log N)$.

2) Explore data by foreign key(s)

The result of querying data is shown on the index page, where the columns have foreign-key constraints are available to be explored further. Given the data stored in Firebase are actually JSON, exploration process takes $O(1)$ since the column referenced is a primary or unique column of the referenced table.