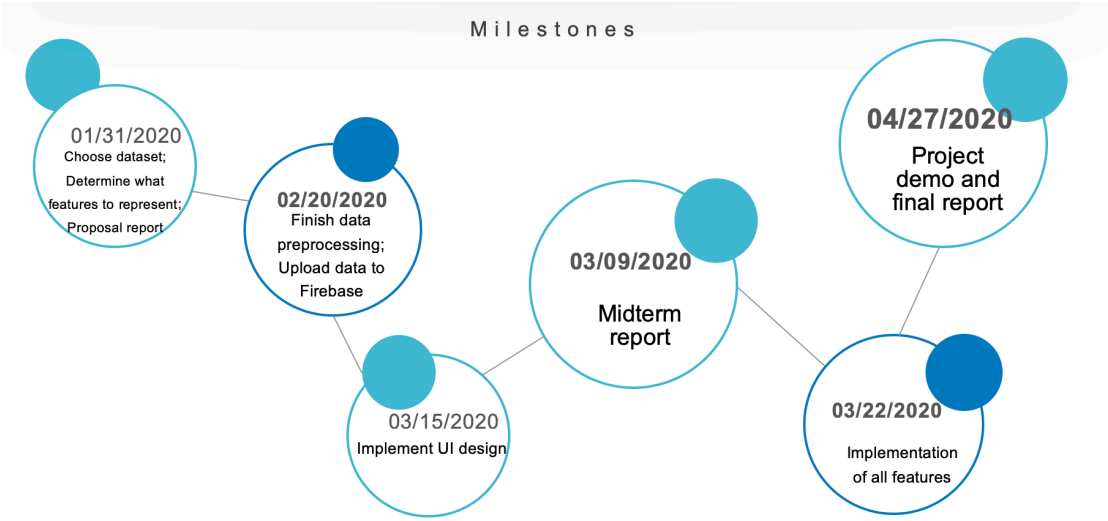
Final Report

Xuewei Huang, Yuan Jiang

1.Project Idea

The goal of this project is to import data from MySQL to Firebase and develop a keyword-driven interface to explore the data via their foreign-key relationships. As one database was provided, we chose 2 additional database based on our interests. We are aimed to build a general keyword-driven searching, sorting and navigation interface for the three database, combined with what we have learned about database in class. People can see the relationship between tables and the expansion of tables when using our interface.

2.Checklist



3.Working Component

Database

Film Database:
film_info(film_id, name, company, director, genre_level, runtime, score, votes, star, writer, year)
film_company(company_id, company, country)
film_genre_level(id, genre, rating)

foreign key: film_info(company) refers to film_company(company)
foreign key: film_info(genre_level) refers to film_genre_level(id)

Hotel Database:
hotel_info(hotel_id, hotel_name, city_name, price_from, lon, lat)
hotel_city(city_id, city, area)
hotel_score(hid, atmosphere, cleanliness, facilities, location, security, staff, score)

foreign key: hotel_info(city_name) refers to hotel_city(city)
foreign key: hotel_score(id) refers to hotel_info(hotel_id)

World Database:
world_city(ID, Name, CountryCode, District, Population)
world_country(Code, Name, Continent, Region, SurfaceArea, IndepYear, Population, LifeExpectancy, GNP, GNPOld, LocalName, Government, HeadOfState, Capital, Code2)
world_countrylanguage(CountryCode, Language, IsOfficial, Percentage)

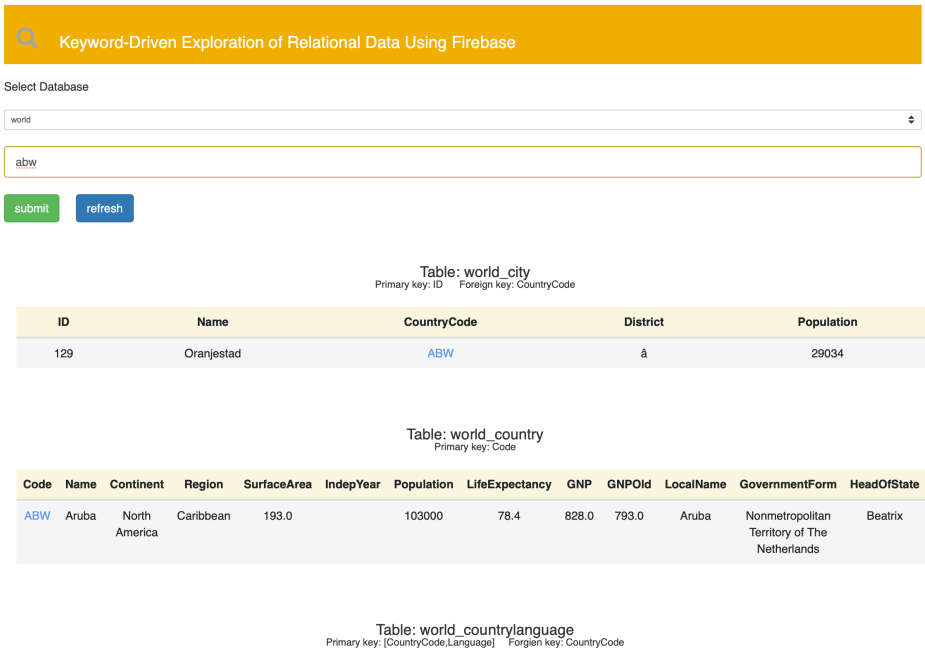
foreign key: world_countrylanguage (CountryCode) refers to world_country (Code)
foreign key: world_city (CountryCode) refers to world_country (Code)

Firebase

We have 3 databases, each containing 3 tables. First, we imported it into MySQL and did some preprocessing, like changing types of some data and imputation. Then we uploaded it to firebase. To meet the requirement of the searching and navigation, we built the inverted-index table for each original table except hotel_score since this table only contains number which cannot be used as key in firebase. Below is the example of our firebase.


inf551 project
inf551-project-fccc4
film
film_company
film_company_invert_index
film_genre_level
film_genre_level_invert_index
film_info
film_info_invert_index
hotel
hotel_city
hotel_city_invert_index
hotel_info
hotel_info_invert_index
hotel_score
world
city_invert_index
country_invert_index
countrylanguage_invert_index
world_city
world_country
world_countrylanguage

Interface



First, we have a title for this interface which has a yellow background.

Second, we have a selection box for database selection and below is a keywords input box where users can type several keywords separated by space. The keywords are case insensitive.

Third, we can see tuples containing at least keyword returned in each table. And the results are sorted as requirements in each table. Here in the screenshot, we searched ‘abw’, which is contained in all the three tables. But we need to scroll to see all the results.

Table: world_country
Primary key: Code

Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld	LocalName	GovernmentForm	HeadOfState	Capital	Code2
ABW	Aruba	North America	Caribbean	193.0		103000	78.4	828.0	793.0	Aruba	Nonmetropolitan Territory of The Netherlands	Beatrix	129.0	AW

Table: world_country
Primary key: Code

Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld	LocalName	GovernmentForm	HeadOfState	Capital	Code2
ABW	Aruba	North America	Caribbean	193.0		103000	78.4	828.0	793.0	Aruba	Nonmetropolitan Territory of The Netherlands	Beatrix	129.0	AW

Table: world_countrylanguage
Primary key: [CountryCode,Language] Forgien key: CountryCode

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	English	F	9.5
ABW	Papiamento	F	76.7
ABW	Spanish	F	7.4

Forth, we can click any of the primary key or foreign key to see the linked results. As the picture above shown, we clicked 'ABW' on table city. Then the linked table country with the foreign key 'ABW' was shown in the place of table city, just like the picture shown. The rest two tables which wasn't clicked stayed at the original place.

Finally, if we want to search other words, we need to click on the 'refresh' button.
(note: if there is no change when you click on something, please click one more time and give some time for the results to be shown on the screen. It takes time to search some specific words or some operations.)

Some example keywords: film: action r; hotel: tokyo house; world: abw afg herat.

4.Implementation

We use React.js to build front end. The image above shows what our app looks like.
First, the selection box: the box contains four parts: a list to select the database, an input box to enter the keywords, a button to submit the query and a button to refresh the page.

```
<h5>Select Database</h5><br/>
            <select className="custom-select" value={this.state.selectOption} onChange={(e)=>this.getValue(
          {
            this.state.database.map((ele,index)=>{
              return(
                <option key={index}>{ele}</option>
              )
            })
          }
        </select>
        <br/> <br/>
            <input placeholder="please enter key words" type="text" name="name" className="form-control"
                   value={ this.state.inputValue }
                   onChange={ this.handleInput.bind(this) }
            />
            <button className="btn btn-success" onClick={this.search.bind(this)}>submit</button>

            <button className="btn btn-primary" style={{margin:'20px'}} onClick={this.refreshPage} >refresh
```

Second, the result presentation: take database world and table city as an example. There are three <h*></h*> to introduce the name of table, the primary key and foreign keys, and the amount of data retrieved from firebase.
Then we use map() to dynamically generate column names and data. When we click the foreign key of one table, this table will refresh and present the primary key table and tuples containing the foreign key. However, other tables will not refresh and keep its data until user clicks the foreign key/primary key in them. This is partial refresh in React.js.

```
if(this.props.key_1 === 'world_city'){
  return (
    <div className="container" style={{marginTop:'30px'}}>
        <h4 className="text-center ">Table: {this.props.key_1}</h4>
        <h6 className="text-center ">Primary key: ID      Foreign key: 
        <h6 className="text-center ">The amount of data: <Amount lists ={this.props.lists} /></h6>
        <br/><table className="table table-striped table-hover">

          <thead>
            <tr className='warning'>
              { world_city_c.map((v,i)=>
                <th className="info" key = {i} className="text-center">{v}</th>)  }
              </tr>
          </thead>
        <tbody>
          {this.props.lists.map((list,i)=>{
          return (
              <tr key={i} className="text-center">
                <td>{list.ID}</td>
                <td>{list.Name}</td>
                <td><a style={{color:'CornflowerBlue'}} onClick={() => this.toShopDetails(list.CountryCode
                <td>{list.District}</td>
                <td>{list.Population}</td>
              </tr>
          )
        })}
        </tbody>
        </table>
ır/></div>)}
```

5.Performance analysis on query processing & exploration process

Search:

We use React.js to build our project. When the user choosing the database and entering the key words, the front end will receive these two parameters and send request to firebase.

Firebase will get the url of each key word and find its invert index. In invert index, each word has a column, a key which is the primary key of the table and the table name, these three attributes help locate the key word.

```
for (var i in tableIndex_world){
var ref = firebase.database().ref(this.state.selectOption+tableIndex_world[i]+keyWord);
ref.on("value", function(snapshot) {
    snapshot.forEach(function(childSnapshot) {
    var childData = childSnapshot.val();
```

When we get the colunm, key and table, we can search the key word in the data table. Take database world and table city as an example: there's three tables in world database and firebase will search the key words in these three tables. When searching table city, we use .orderByChild() function to find the tuples whose ID is equal to the key word's key. There may be many tuples contain the key word so we use an array to store the results.

```
//world_city
if(childData.table === "world_city"){
    var rref = firebase.database().ref("world/"+childData.table);
    rref.orderByChild('ID').equalTo(String(childData.key)).on("value", function(snapshot) {
        var childData_2 = snapshot.val();

        var start = Number(Object.keys(childData_2)[0]);var end = Number(Object.keys(childData_2)[0]) + Object.keys(childData_2).
        for (var j = start; j <= end;j++){
            if(typeof(childData_2[j])!='undefined'){
                world_city_list[j] = {'ID':childData_2[j].ID,'Name':childData_2[j].Name,'CountryCode':childData_2[j].CountryCode,
                }
    }
    });}
```

Sort:

When we get the search reault, we use index() to sort the tuples, so that the result will be ordered by the number of keywords appearing in the tuple.

```
index(table,key){
    var dict = {}
    for(var row in table){
        for(var item in table[row] ){
            for (var word in key){
                if(table[row][item].toUpperCase().match(key[word].toUpperCase())){
                    if (dict[row]){
                        dict[row] += 1
                    }else{
                        dict[row] =1
                    }
         }
    }   }}
// Create items array
var items = Object.keys(dict).map(function(key) {
  return [key, dict[key]];
});
// Sort the array based on the second element
items.sort(function(first, second) {
  return second[1] - first[1];
});
var arr= []
for (let [key, value] of Object.entries(items)) {
  arr.push(value[0])
}
// Create a new array with only the first 5 items
var dict_new = []
for(var row in table){
    for (var i in arr){
        if (arr[i] === row){
            dict_new[i] = table[row]
        }
    }
}
return(dict_new)
```

Navigation:

We use two components: fk.js and pk.js to complete navigating from returned/current tuples to other tuples through the foreign key relationships.

When user clicking the foreign key in one table, fk.js will receive the foreign key's

database, table and value. These three parameters will be send to firebase and search for the primary key tuples. The search progress is just like the key words search.

```
if (this.state.world_city === 'world_city'){
  return(
    <Fk lists={this.state.lists_world_city} key_1={this.state.world_city} fb={this.props.fb}/>
  )
  }
  if (this.state.hotel_info === 'hotel_info'){
    return(
      <Fk lists={this.state.lists_hotel_info} key_1={this.state.hotel_info} fb={this.props.fb}/>
    )
    }
    if (this.state.hotel_city === 'hotel_city'){
      return(
        <Fk lists={this.state.lists_hotel_city} key_1={this.state.hotel_city} fb={this.props.fb}/>
      )
      }
      if (this.state.film_company === 'film_company'){
        return(
          <Fk lists={this.state.lists_film_company} key_1={this.state.film_company} fb={this.props.fb
        )
        }
        if (this.state.film_info === 'film_info'){
          return(
            <Fk lists={this.state.lists_film_info} key_1={this.state.film_info} fb={this.props.fb}/>
```

Amount of data:

React.js does not have a function to calculate the amount of data, so we write another function component to calculate the amount of data retrieved from the server on every request.

```
export default function Amount(propos) {
    var bytes = 0;
    var obj = propos.lists;
    function sizeOf(obj) {
        if(obj !== null && obj !== undefined) {
            switch(typeof obj) {
            case 'number':
                bytes += 8;
                break;
            case 'string':
                bytes += obj.length * 2;
                break;
            case 'boolean':
                bytes += 4;
                break;
            case 'object':
                var objClass = Object.prototype.toString.call(obj).slice(8, -1);
                if(objClass === 'Object' || objClass === 'Array') {
                    for(var key in obj) {
                        if(!obj.hasOwnProperty(key)) continue;
                        sizeOf(obj[key]);
                    }
                } else bytes += obj.toString().length * 2;
                break;
            }
        }
        return bytes;
    };

    function formatByteSize(bytes) {
        if(bytes < 1024) return bytes + " bytes";
        else if(bytes < 1048576) return(bytes / 1024).toFixed(3) + " KiB";
        else if(bytes < 1073741824) return(bytes / 1048576).toFixed(3) + " MiB";
        else return(bytes / 1073741824).toFixed(3) + " GiB";
    };

    return formatByteSize(sizeOf(obj));
}
```

6.Group responsibility

This group was done in a team of two, Xuewei Huang and Yuan Jiang. Xuewei's responsibility was building the cloud database which includes preprocessing data into the right format, generating some inverted-index tables corresponding to the original tables to meet the requirement of backend, and finally upload them to firebase. Since Yuan is more experienced in React.js, she was mainly responsible for the web-based user interface and the implementation of the components on the web which includes the representation of sorted data and navigation results. Xuewei made some revision for the implementation. Other parts such as project proposal, mid-term report, and final report was done collectively.