Project 3: Data Conversion

Overview

In all projects so far, you were provided with a dataset in a format consistent with the relational model, so it was straightforward to load them into a relational database. Unfortunately, life is often more complicated; more likely than not, your data will be based on a non-relational model, and you have to perform a "conversion" process first to make them suitable for the relational model. In this project, we provide JSON-formatted data on Nobel prize laureates and ask you to (1) convert the JSON data into a format that can be bulk-loaded into MySQL, (2) write SQL queries on the loaded data to answer a few questions, and (3) create a PHP page that returns information on a Nobel laureate given their ID. Through this exercise, you will get a taste of the real-world conundrum, where we spend more time "getting ready" than doing the "real work."

Development Environment

Project 3 will continue to use the "mysql-apache" container that you have been using in Projects 1 and 2.

Part A: Download and examine the JSON data file

To start this project, download the JSON dataset on Nobel prize laureates nobel-laureates.json into the `/home/cs143/data` directory using commands similar to the following:

```
$ mkdir -p /home/cs143/data
$ cd /home/cs143/data
$ wget http://oak.cs.ucla.edu/classes/cs143/project3/nobel-laureates.json
```

Learn JSON

Together with CSV and XML, JSON is one of the most popular data exchange formats on the Internet. JSON stands for JavaScript Object Notation and started as the syntax for representing objects in JavaScript. As JavaScript gained popularity, JSON's popularity as a data exchange format has increased as well. The JSON syntax is straightforward to learn and understand. JSON supports the primitive data types such as numbers and strings, as well as arrays and "objects," which are a collection of (attribute name, value) pairs. Strings are encloded in double quotes like `"John"`. An "array" is represented using square brackets, like `[1, 2, 3]`. An object and its "properties" are

represented using curly braces. For example, `{ "sid": 201, "name": "John" }` represents an object whose `sid` is 201 and `name` is "John". This notational convention can be applied recursively, like

```
[{
    "sid": 201,
    "name": {
        "first": "Megan",
        "last": "Fox"
    }
 },{
    "sid": 301,
    "name": {
        "first": "John",
        "last": "Cho"
    }
}]
```

To learn more detail, read one of many tutorials on JSON available on the Internet. We find the JSON tutorial on TutorialPoints is reasonably succinct and informative.

Explore our data

Now that you have a basic understanding of JSON, your task is to examine the provided JSON file to understand the data. The data was downloaded from the official Nobel Prize Web site through their public API. You will be translating this data into relations and loading it into MySQL server in our container. Open the JSON file and investigate. Since every "value" in the JSON file is labeled with their "name," data is mostly self-explanatory. For example, the first object in the `laureate` array has information on Dr. A. Michael Spence, who won the 2001 Nobel Economics Prize:

```
{
    "id":"745",
    "knownName":{
        "en":"A. Michael Spence",
        "se":"A. Michael Spence"
    },
    "givenName":{
        "en":"A. Michael",
        "se":"A. Michael"
    },
    "familyName":{
        "en":"Spence",
        "se":"Spence"
    },
    "fullName":{
        "en":"A. Michael Spence",
        "se":"A. Michael Spence"
    },
    "gender":"male",
    "birth":{
        "date":"1943-00-00",
        "place":{
        "city":{
            "en":"Montclair, NJ",
            "no":"Montclair, NJ",
            "se":"Montclair, NJ"
        },
        "country":{
            "en":"USA",
            "no":"USA",
            "se":"USA"
        },
        "cityNow":{
            "en":"Montclair, NJ",
            "no":"Montclair, NJ",
            "se":"Montclair, NJ"
        },
        "countryNow":{
            "en":"USA",
            "no":"USA",
            "se":"USA"
        },
        "continent":{
            "en":"North America"
        },
        "locationString":{
            "en":"Montclair, NJ, USA",
            "no":"Montclair, NJ, USA",
            "se":"Montclair, NJ, USA"
        }
        }
```

```
    },
    "links":{
        "rel":"laureate",
        "href":"http:\/\/masterdataapi.nobelprize.org\/2\/laureate\/745",
        "action":"Get",
        "types":"application\/json"
    },
    "nobelPrizes":[
        {
        "awardYear":"2001",
        "category":{
            "en":"Economic Sciences",
            "no":"\u00d8konomi",
            "se":"Ekonomi"
        },
        "categoryFullName":{
            "en":"The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfre
            "no":"Sveriges Riksbanks pris i \u00f8konomisk vitenskap til minne om Alf
            "se":"Sveriges Riksbanks pris i ekonomisk vetenskap till Alfred Nobels mi
        },
        "sortOrder":"2",
        "portion":"1\/3",
        "dateAwarded":"2001-10-10",
        "prizeStatus":"received",
        "motivation":{
            "en":"for their analyses of markets with asymmetric information",
            "se":"f\u00f6r deras analys av marknader med assymetrisk informations"
        },
        "prizeAmount":10000000,
        "prizeAmountAdjusted":12295082,
        "affiliations":[
            {
                "name":{
                    "en":"Stanford University",
                    "no":"Stanford University",
                    "se":"Stanford University"
                },
                "nameNow":{
                    "en":"Stanford University"
                },
                "city":{
                    "en":"Stanford, CA",
                    "no":"Stanford, CA",
                    "se":"Stanford, CA"
                },
                "country":{
                    "en":"USA",
                    "no":"USA",
                    "se":"USA"
                },
                "cityNow":{
```

```
                    "en":"Stanford, CA",
                    "no":"Stanford, CA",
                    "se":"Stanford, CA"
                },
                "countryNow":{
                    "en":"USA",
                    "no":"USA",
                    "se":"USA"
                },
                "locationString":{
                    "en":"Stanford, CA, USA",
                    "no":"Stanford, CA, USA",
                    "se":"Stanford, CA, USA"
                }
            }
        ],
        "links":{
            "rel":"nobelPrize",
            "href":"https:\/\/masterdataapi.nobelprize.org\/2\/nobelPrize\/eco\/2001"
            "action":"Get",
            "types":"application\/json"
        }
        }
    ]
}
```


Some more information on our data:

1. Some property values are given in multiple languages. For example, Dr. Spence's family name is given in both English (en) and Swedish (se), which happen to be the same. Ignore any language other than English in this project.


```
"familyName":{
    "en":"Spence",
    "se":"Spence"
},
```


2. Many names, like city names, are given in multiple versions. For example, Dr. Spence's birth city is given as `city` and `cityNow`, so that if the city name has changed over time, the original name and its current name can be shown. Use only the original name in this project, ignoring everything else.

```
"city":{
    "en":"Montclair, NJ",
    "no":"Montclair, NJ",
    "se":"Montclair, NJ"
},
...
"cityNow":{
    "en":"Montclair, NJ",
    "no":"Montclair, NJ",
    "se":"Montclair, NJ"
},
```

3. The vast majority of the Nobel laureates are people, but a number of prizes have been awarded to organizations. These "organization laureates" have the `orgName` property and omit other irrelevant properties such as `familyName`. For example, the following is the JSON object corresponding to the laureate "Red Cross."

```
{
    "id":"482",
    "orgName":{
        "en":"International Committee of the Red Cross",
        "no":"Den Internasjonale R\u00f8de Kors Komit\u00e9",
        "se":"Internationella R\u00f6dakorskommitt\u00e9n"
    },
    "nativeName":"Comit\u00e9 international de la Croix Rouge",
    "founded":{
        "date":"1863-00-00",
        ...
    }
}
```

4. The `id` property is unique among all `laureate` objects, whether they are people or organizations.
5. Some descriptions on a few specific properties whose meanings may not be obvious from their names:
    - `links` property has information on how the JSON object can be retrieved from the Nobel Prize official API. You can ignore them.
    - `sortOrder` property of `nobelPrizes` is 1 if there was a single recipient in the given year and category. Its value ranges 1 through n if the prize was shared by n laureates. The `portion` property shows what fraction of the prize money was given to the recipient. For example, the 2001 Nobel Economic prize was shared by three people. Dr. Spence was the second recipient and received 1/3 of the $1M prize money.

```
{
    "id":"745",
    "knownName":{
        "en":"A. Michael Spence",
    },
    ...
    "nobelPrizes":[
        {
        "awardYear":"2001",
        "category":{
            "en":"Economic Sciences",
        },
        ...
        "sortOrder":"2",
        "portion":"1\/3",
        "dateAwarded":"2001-10-10",
        "prizeStatus":"received",
        "prizeAmount":10000000,
        ...
    ]
}
```

- Most laureates gladly accepted their prize, but a few declined (e.g., Jean-Paul Sartre), and a few were forced not to accept by their country (e.g., Boris Pasternak). This information is captured in the property `prizeStatus`.
- The `residences` and `affiliations` properties show the laureate's residence(s) and affiliation(s) at the time of the award.

Part B: Design your relational schema

Now that you understand the data you'll be working with, design a good relational schema for it. In your design, make sure that the following property values are stored if they appear in our dataset.

```
"laureates": {
    "id", "givenName", "familyName", "gender",
    "birth: {
        "date",
        "place": {
            "city", "country"
        }
    },
    "orgName",
    "founded": {
        "date",
        "place": {
            "city", "country"
        }
    },
    "nobelPrizes": [{
        "awardYear", "category", "sortOrder",
        "affiliations": [{
            "name", "city", "country"
        }]
    }]
}
```

Discard other properties in our dataset in your schema. If a property value is given in multiple languages, just store the English value (“en”).

In the schema design process, iterating through the following four steps should help you:

1. List your relations. Please specify all keys that hold on each relation. You need not specify attribute types at this stage.
2. Identify (nontrivial) functional dependencies that hold on each relation, excluding those that effectively specify keys. It is safe to assume that if a particular functional dependency holds on the current instance of the data, it holds in *any* instance.
3. Are all of your relations in Boyce-Codd Normal Form (BCNF)? If not, either redesign them and start over, or justify to yourself why you feel it is advantageous to use non-BCNF relations.
4. Are there other redundancies or inefficiencies that you notice? Can we redesign the relations to avoid them?

Clearly, there is no single “right” answer to the schema design, but some designs are likely to be more performant or less redundant (or both). So try to come up with a design with those properties.

Part C: Write a data transformation program

Next, you will write a program that transforms the JSON data into MySQL load files that are consistent with the relational schema you settled on in Part B. Remember that our MySQL tutorial linked in Project 1 had a brief explanation on the MySQL LOAD command and the load file format. If necessary, read MySQL LOAD command reference to learn more details on the MySQL load file.

You can use any tools/languages preinstalled in our container to write the transformation program. In particular, our container has node (v14.17), python3 (v3.8), php cli (v7.4) preinstalled, so you can use either JavaScript, Python, or PHP to implement your converter. Your choice is not limited to these three; any tool or language can be used as long as they are available in our container.

To help you get started, we provide "skeleton codes", `convert.js`, `convert.py`, `convert.php`, and a Bash execution script `convert.sh` in convert.zip. Download and unzip the file. Open the unzipped `convert.sh` file and uncomment a line corresponding to your chosen language. Then run the script like the following,

```
$ ./convert.sh
745     A. Michael      Spence
```

and make sure that you see an output similar to the above.

Now modify the code in `convert.js/py/php` so that it will (1) take the input JSON data located in `/home/cs143/data/nobel-laureates.json` and (2) produce a (set of) SQL load file(s) in the ***current directory***. The load files should be named according to the corresponding relation of your schema with `.del` as its extension. For example, if your schema has two relations, `Laureates` and `NobelPrizes`, then your program must produce two load files `Laureates.del` and `NobelPrizes.del` in the current directory. Make sure that the entire conversion process can be executed simply by executing `convert.sh`. In most cases, this can be done just by updating the file `convert.js/py/php`, but if you decide to create multiple source files for your converter, make sure to modify the `convert.sh` as well, so that we can execute your code simply by running `convert.sh`.

JSON Parsing Functions

Depending on your choice of programming language, you may find it helpful to use existing JSON decoding/encoding functions:

PHP:

- `json_decode($json_string)`: decode a JSON string to a PHP object
- `json_encode($value)`: encode a PHP object to a JSON string

Python:

- `json.loads(json_string)`: decode a JSON string to a Python object
- `json.dump(object)`: encode a Python object to a JSON string

JavaScript:

- `JSON.parse(json_string)`: decode a JSON string to a JavaScript object
- `JSON.stringify(object)`: encode a JavaScript object to a JSON string

Look up the appropriate library APIs online if you decide to use the above library functions.

Duplicate elimination

When transforming the JSON data to relational tuples, you may discover that certain information appears multiple times in the original data but it must be stored as a single tuple in your table. You may ***NOT*** rely on the MySQL bulk loader to achieve this. MySQL will generate an error but continue loading when a tuple with a duplicate value in a key attribute is encountered. However, using this "feature" to eliminate duplicates is an unreliable hack and should be avoided.

For duplicate detection and elimination in your program, you may find "dictionary data type" (or "associative array") helpful. For example, whenever you encounter a potential "tuple" to be stored, you can first check whether the tuple already exists in a dictionary and write it to the load file only if it does not (and then insert the tuple into the dictionary, so that it won't be written again in the future).

Numbers for basic sanity check: The provided dataset contains 955 nobel laureates (either person or organization), and 962 nobel prizes (counting duplicates where a nobel prize is shared among multiple laureates) or 603 unique nobel prizes.

Notes on CR/LF issue

If your host OS is Windows, you need to pay particular attention to how each line of a text file ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux, Mac OS X, and tools in cygwin/WSL) use only an LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a Unix tool (such as `mysql`). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected behavior from these tools. If you encounter this problem, you may want to run the `dos2unix` command in VM on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Part D: Load your data into MySQL

Now create and populate your table(s) in MySQL inside the `class_db` database. We suggest that you first debug the schema creation and loading process interactively using the MySQL command-line interface. When you are satisfied that everything is working, follow the instructions to set up for batch loading (see **Section D.2** below), which allows tables to be conveniently recreated and populated from scratch with one command.

D.1 Create and load your databases interactively

Using the command-line interface to MySQL, issue a set of `CREATE TABLE` commands for all of the relations in your schema. Once the tables are created, load your data in MySQL using `LOAD DATA LOCAL INFILE` command from the load files produced by your conversion program. Please make sure that no warning or error is encountered during the load. A common source of warning is when a column length (say, `VARCHAR(n)`) is smaller than the loaded data. In that case, make sure to increase your column length to accommodate the loaded data.

D.2 Create SQL load script

Now create a SQL script named `load.sql` that performs the following

1. Drop all existing tables for Project 3 if they exist.
2. Create all tables according to your schema.
3. Populate the tables using the data in the SQL load files ***in the current directory*** produced by your conversion program.

Part E: Write SQL queries

Now that your database is fully populated, explore the data by writing a few interesting SQL queries. In particular, write SQL queries corresponding to the following five questions:

1. What is the id of "Marie Curie"? (Answer: 6)
2. What country is the affiliation "CERN" located in? (Answer: Switzerland)
3. Find the family names associated with five or more Nobel prizes (Answer: Smith, Wilson)
4. How many different locations does the affiliation "University of California" have? Assume that a location is uniquely identified by its city and country. (Answer: 6)
5. In how many years a Nobel prize was awarded to an organization (as opposed to a person) in at least one category? (Answer: 26)

Note: There are 25 unique organization laureates in our dataset, but some of them received the prize more than once.

Write the SQL SELECT queries, execute them, and make sure that you get the correct answers. If not, debug your schema/conversion program/load script/queries, so that they produce the correct answers.

Once you finish writing and debugging the queries, write five SQL script files, named as `q1.sql`, ..., `q5.sql`, that contains each of the above SQL SELECT queries.

Part F: Implement Web-service JSON API

Your final task is to implement a "Web service" that takes a Nobel laureate's ID and returns the JSON data of the laureate. The Web service should be accessible at `http://localhost:8888/laureate.php?id=XXX` and return the corresponding JSON data. The

returned JSON data must include all properties that we asked you to store in Part B, and its "schema" (or "structure") should be as close as possible to the original data. For example, for `id=745` your service should return a result similar to the following:

```
{
    "id":"745",
    "givenName":{ "en":"A. Michael" },
    "familyName":{ "en":"Spence" },
    "gender":"male",
    "birth":{
        "date":"1943-00-00",
        "place":{
            "city":{ "en":"Montclair, NJ" },
            "country":{ "en":"USA" }
        }
    },
    "nobelPrizes":[{
        "awardYear":"2001",
        "category":{ "en":"Economic Sciences" },
        "sortOrder":"2",
        "affiliations":[{
            "name":{ "en":"Stanford University" },
            "city":{ "en":"Stanford, CA" },
            "country":{ "en":"USA" }
        }]
    }]
}
```

To help you get started, we provide a "skeleton PHP code" laureate.php that takes an id parameter and returns fake JSON data for the given ID. Download the file, rename it to `laureate.php` (the linked file has the extra extension `.txt` to make the code downloadable). Your job is to modify the provided skeleton code to return the correct JSON data for the input ID.

Notes

1. The "page" returned by `laureate.php` should be pure JSON data. **DO NOT** enclose the JSON data inside HTML tags.
2. **Your JSON output must include the property name `en` whenever appropriate** to preserve the original JSON structure.
3. You can construct the JSON output with your custom code, but we find it the easiest to use PHP's `json_encode()` function. This tutorial on PHP JSON Parsing may be helpful if you decide to use this function.
4. To check if the JSON data returned from your PHP is in a valid format, you can use one of many online JSON validators, such as JSONLint.

Submit Your Project

What to Submit

For this project, you will have to submit one zip file:

- `project3.zip`: You need to create this zip file using the packaging script provided below. This file will contain all source codes that you wrote for Project 3

Creating `project3.zip`

Roughly, the zip file that you submit should have the following structure:

```
project3.zip
 +- convert.sh
 +- convert.js/py/php or any other files for conversion
 +- load.sql
 +- q1.sql
 +- q2.sql
 +- q3.sql
 +- q4.sql
 +- q5.sql
 +- laureate.php
 +- README.txt (Optional)
```

Make sure that your code meets the following requirements:

1. Your data conversion program should take the JSON data located at `/home/cs143/data/nobel-laureates.json`.
2. Your data conversion program should produce the SQL load files in the ***current directory*** with the extension `.del`.
3. We should be able to execute your conversion program simply by running `convert.sh`. Nothing else.
4. Your SQL load script must drop all existing tables for Project 3 ***if they exist***, create the tables, and load data from the files in the ***current directory*** created by your conversion program.
5. All you SQL scripts (both load and query scripts) must be executable by `mysql class_db < your-script.sql`
6. Your `laureate.php` file must work if we copy it to the `/home/cs143/www/` directory.
7. The README.txt file may be optionally included if you want to include any information regarding your submission.

To help you package your submission zip file, you can download and use our packaging script p3_package. After downloading the script in your Project 3 directory, remove all files that should not be submitted in the directory and execute the packaging script.

When executed, our packaging script will collect all files located in the current directory and create the `project3.zip` file according to our specification like the following:

```
$ ./p3_package
  adding: convert.sh (deflated 27%)
  adding: convert.js (deflated 62%)
  adding: load.sql (deflated 66%)
  adding: q1.sql (deflated 8%)
  adding: q2.sql (stored 0%)
  adding: q3.sql (deflated 12%)
  adding: q4.sql (stored 0%)
  adding: q5.sql (deflated 9%)
  adding: laureate.php (deflated 32%)
[SUCCESS] Created '/home/cs143/project3/project3.zip'
```

Testing of Your Submission

In order to minimize any surprises during grading, we are providing a simplified version of the "grading script" p3_test that we will use to test the functionality and correctness of your submission. In essence, the grading script unzips your submission to a temporary directory and executes your files to test whether they run OK on the grading environment. Download the grading script and execute it in the container like:

```
$ ./p3_test project3.zip
```

(if your project3.zip file is not located in the current directory, you need to add the path to the zip file before project3.zip. You may need to use `chmod +x p3_test` if there is a permission error.)

When everything runs properly, you will see an output similar to the following from the grading script:

```
$ ./p3_test project3.zip
Running the conversion program via convert.sh...

Showing the load files created by your conversion program:
laureates.del  (and other .del files)

Running your load.sql script...

Running your query script q1.sql...
id
6

Running your query script q2.sql...
country
Switzerland

Running your query script q3.sql...
familyName
Smith
Wilson

Running your query script q4.sql...
(select attribute)
6

Running your query script q5.sql...
(select attribute)
26

Copying your laureate.php file to ~/www/
All done!

Please ensure that you have the Web service available at http://localhost:8888/laurea
(1) Please make sure that the returned JSON format is valid
(2) Please make sure that the returned JSON structure is the same as the original, ex
(3) Please make sure that both person and organization laureates have the correct str
```

You **MUST** run your submission using the script before your final submission and ensure that it runs fine and produces expected results.

Submitting Your Zip File

Visit GradeScope to submit your zip file electronically by the deadline. In order to accommodate the last minute snafu during submission, you will have 1-hour window after the deadline to finish your submission process. That is, as long as you start your submission before the deadline and complete within 1 hour after the deadline, you are OK.