**CS541 – project 2**

Wen-Ling Chi

Email: chi64@purdue.edu

**Part 1 – Understanding the data (20 points)**

* 1. **Is each column present in every data entry? List which columns do not appear in every data entry.**

|  |  |
| --- | --- |
| **Filename** | **Columns** |
| airport.json | • city • airport\_province  • island  • elevation |
| city.json | • population • elevation • country\_capital |
|  |
| country.json | • capital • province |
| country-other localname.json | • othername |
| countrypopulations.json | • capital |
| economy.json | • gdp • agriculture • service • industry • inflation • unemployment |
| ethnicgroup.json | • country\_capital  • ethnic\_group\_percentage |
| language.json | • country\_capital  • percentage |
| organization.json | • city  • country  • province  • established |
| politics.json | • independence  • wasdependent • dependent  • government |
| population.json | • country\_province  • population growth • infant mortality |
| province.json | • population • capital • capprov |

**1-2 Given the file descriptions and the knowledge graph, are there any redundant columns in the files?  
List which files have redundant columns. Name the columns and provide reasons why, in your opinion, those columns are redundant?**

|  |  |
| --- | --- |
| **Filename** | **Redundant columns / Reason** |
| airport.json | • country\_name: we have country\_code as a foreign key to the country table. |
| city.json | • country\_name: we have country as a foreign key to the country table. • country\_capital: if we know the country code, then we should also know the capital. |
| cityothername.json | • country\_area: since we know its city name, we can also know its country. Through country table, we can find the area of the country. • country\_capital: since we know its city name, we can also know its country. Through country table, we can find the capital of the country. |
| country.json | • province: we already have “capital” as the foreign key referencing the city table, and we can find out the province the capital belongs to from “province” in the city |
| countrypopulations.json | • country\_name: we have country\_code as a foreign key to the country table. • capital: if we know the country name, then we should also know the capital. |
| ethnicgroup.json | • country: we have country\_code as a foreign key to the country table. • country\_capital: we have country\_code as a foreign key to the country table. |
| language.json | • country\_capital: we have country as a foreign key to the country table. • country\_area: we have country as a foreign key to the country table. |
| located-on.json | • country: this table doesn’t connect to country table, so this column is unnecessary. Also, if we know the city, then we should also know the country.  • province\_area: this is unnecessary attribute for this data. |
| population.json | • country\_name: we have country\_code as a foreign key to the country table. • country\_province: if we know the country\_code, we can also know its province. Only “Gaza Strip” has no province, but in the country table, it also has no province. |
| religion.json | • country\_name: we have country\_code as a foreign key to the country table. • country\_population: if we know the country, then we should also know the population of the country. |

* 1. **What logical constraints would you set for which columns in the files? For example, latitude should always be in the range of [-90, 90], and the longitude should be in [-180, 180]. Here you don't have to set constraints for every column of every file.**

|  |  |
| --- | --- |
| **Filename** | **Constraints** |
| airport.json | • latitude: [-90, 90] • longitude: [-180, 180] |
| city.json | • latitude: [-90, 90] • longitude: [-180, 180] |
| country.json | • encompass\_percentage: [0, 100] |
| economy.json | • agriculture: [0, 100] • service: [0, 100]  • industry: [0, 100] • unemployment: [0, 100] |
| ethnicgroup.json | • ethnic\_group\_percentage: [0, 100] |
| language.json | • percentage: [0, 100] |
| religion.json | • percentage: [0, 100] |

**Part 2 – Relational database schema**

Run steps.

1. Clean all your tables in your database

>> Use Oracle SQL Developer to run sql file

**>> sql\_schema/project2\_dropTable.sql**

1. Create tables / schema

>> Use Oracle SQL Developer to run sql file

>> **sql\_schema/project2\_schema.sql**

**Diagram

Description automatically generated**

Fig. 1 - The entity-relationship diagram of my database

**Part 3 – Inserting the data to Oracle DBMS**

Run steps.

1. Create insert command through python

>> Use cmd / terminal to run python file

>> **python sql\_schema/createInsertValue\_sql.py**

>>output: It will generate **project2\_data.sql** in the **sql\_schema** folder

1. Insert values into tables

>> Use Oracle SQL Developer to run sql file

>> **project2\_data.sql**

>> This might take a long time to insert values into tables

**Part 4 – Using MongoDB**

Run steps.

1. Install 3.6 version of MongoDB(windows) / Set environment Variables

>> put (~/MongoDB/Server/3.6/bin) into environment variables

1. Create MongoDB schema

>> Use cmd / terminal to run python file. This python file will be in the mongodb\_schema folder.

>> **python mongodb\_schema.py**

>> output: It will generate 23 collections in the project2 DB

1. Create json files for the data for mongodb

>> Use cmd / terminal to run .py. This python file will be in the mongodb\_schema folder.

>> **python insertValue\_mongodb.py**

>>output: It will generate **23 json files** in mongodb\_data folder

1. Insert data into mongoDB

>> Click **import\_data\_mongodb.bat.** This python file will be in the mongodb\_schema folder.

(It will automatically insert values from 23 json files to mongoDB, but because it needs to use **mongoimport**. That’s why I chose 3.6 version)

**Part 5 – Queries**

**5-2 Create a table to summarize the execution time in the two DBMS for each of the queries. For example, you could use the following table template**

**>>**

**>> python mongodb\_query.py 10**

|  |  |  |
| --- | --- | --- |
| Query Number | Execution time | |
| Oracle (ms) | MongoDB (ms) |
| 5-1-a | 259 | 97 |
| 5-1-b | 650 | 416 |
| 5-1-c | 99 | 60 |
| 5-1-d | 316 | 81 |
| 5-1-e | 109 | 359 |
| 5-1-f | 95 | 42 |
| 5-1-g | 54 | 35 |
| 5-1-h | 71 | 509 |
| 5-1-i | 114 | 106 |
| 5-1-j | 98 | 188 |

**5-3 Comment on the execution times you have obtained. Give educated reasons why one DBMS has  
performed better than the other?**

In the above form, most of MongoDB’s execution times are better than Oracle. MongoDB has less constraints. Thus, compare to Oracle SQL, we need less checks when we execute the query in MongoDB.

For the few cases, MongoDB’s execution times are worse than Oracle. From my observation, MongoDB performs worse than Oracle SQL if we need to use multiple collections/tables. It is possible that the Oracle SQL Database will cache and create automatically indexing internally.

**5-4 Suggest one performance improvement technique that you can use on each DBMS to improve the performance. Implement your technique on each DBMS and report your results in a table similar to questions 5-2.**

Ref : <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/CREATE-INMEMORY-JOIN-GROUP.html#GUID-87CA7034-4F80-4D46-8EE1-5CC865C2D676>

**5-5 Comment on the execution times you have obtained after implementing your performance improvement technique. How much improvement do you observe compared to your results in 5-2?**