PURDUE UNIVERSITY | Department of Computer Science

November 5th, 2021

CS 54100: Database Systems

Project 2

[Due date: December 2nd, 2021, at 23:59 EST]

Course Team		
Instructor	GTAs	
Professor Elisa Bertino bertino@purdue.edu	Charis Katsis ckatsis@purdue.edu	Danushka Menikkumbura dmenikku@purdue.edu

Assessment objectives:

This project assesses your ability to:

1. Perform database normalization.
2. Extract the database schema.
3. Formulate SQL queries.
4. Become competent with other DBMS such as MongoDB.

Background knowledge:

Thorough knowledge of the course content covered up to the release of this project is required. Additionally, experience using high-level programming languages such as C, C++, Java, Python, etc is required.

Project logistics:

- You must work on this project individually.
  - Please keep to CS54100 Academic Integrity Policy. You can find it in the course schedule document.
  - Any form of course integrity policy violation will result in a zero grade for this project.
- You need to use the departmental Oracle database for the parts pertaining to relational database.
- You will need to install the MongoDB database in your own system. We provide installation instructions subsequently in the document.
- The project total is 100 points.

Project deliverables:

- The project deliverable must be submitted through Brightspace. Follow the instructions provided in each project part.
- We strongly suggest you start the project early.
- **The project is due on December 2nd, 2021, at 23:59 EST.**
- Late submission policy: The submission of the project is accepted only within 5 days from the return deadline. 10 points will be deducted for each day (fraction of day) of delay in returning the project.

Introduction

In this assignment, you are given a set of data encoded in some file format. The objectives are:

1) Understand the data using the file descriptions and the knowledge graph in this document.
2) Design the schemas for data storing and manipulation in two database management systems (DBMS): Oracle SQL and MongoDB (NoSQL).
3) Insert the data into the DBMSs.
4) Write queries to answer specific questions.
5) Observe the query execution times in the two DBMS environments.

Knowledge Graph

Figure 1 shows the knowledge graph pertaining to the dataset for this assignment. The graph helps you understand the knowledge that can be extracted from the data. For example, an instance of the "Country" has a "capital," which is an instance of a "City." Open the files country.json and the city.json and understand how this information is encoded in the files. Use the information provided for those files in the Section "Data files." Do the same for all other graph connections. Once you have finished this exploration, you should probably find yourself familiar with the dataset.

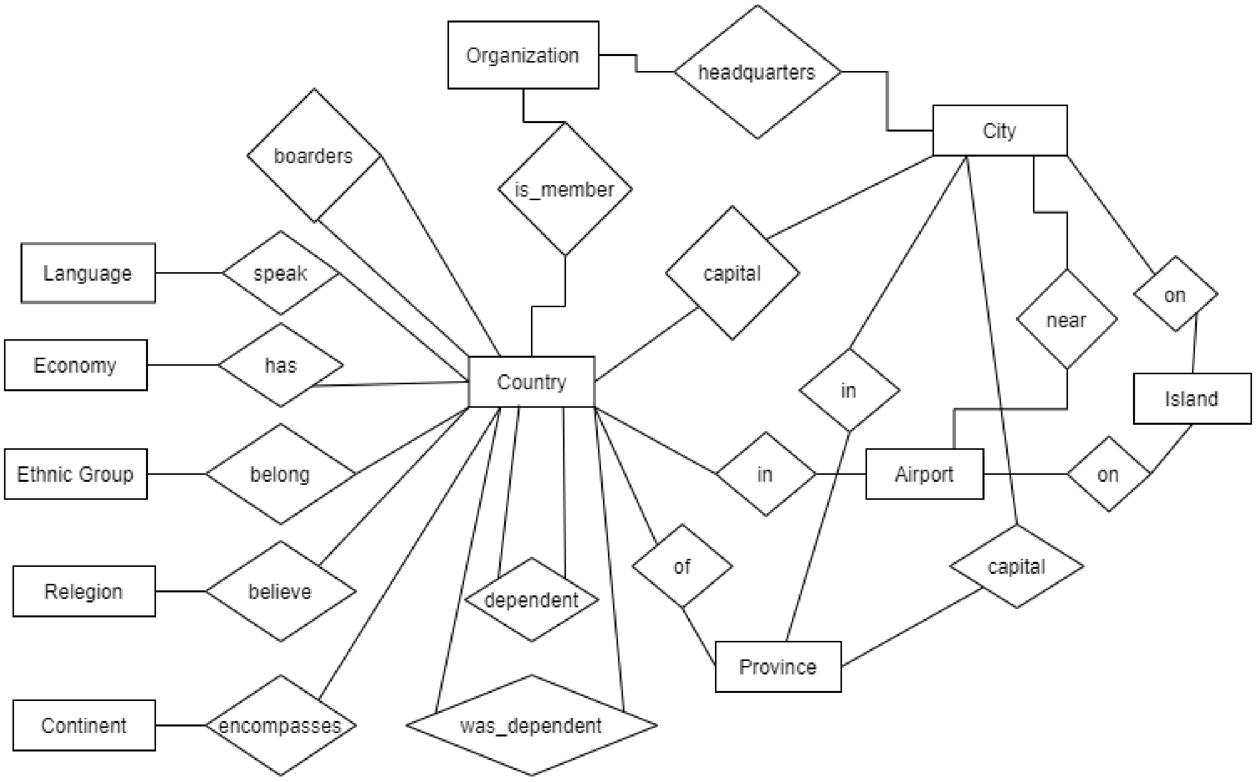

Figure 1: The knowledge graph which describes the provided data.

Data files

The dataset provided to you corresponds to realistic geographical data. All the data are provided to you encoded in JSON files, and they have two parts: the "column" part and the "items" part. The "column" part has the column names and the data types. For example, city.json has a column "province" of type "varchar2."

Table 1 provides descriptions for the individual files of the dataset.

Filename	Description
airport.json	Contains airport entries • iatacode: the IATA code of the airport. • name: the name of the airport • country_code: the country code where the airport is located • country_name: the country name where the airport is located. • city: if the airport is associated with a city, the name of the city. • province: the province where the city belongs to. • island: if it is located on an island, the name of this island • latitude: geographic latitude • longitude: geographic longitude • elevation: the elevation (above sea level) of the city • gmtOffset: the GMT offset of the local time
borders.json	Contains information about neighboring countries. For every pair of neighboring countries (A, B), *only one tuple is given* – thus, the entries are not symmetric. • country1: a country code • country2: a country code • length: length of the border between country1 and country2.
city.json	Contains information about cities. • name: the name of the city. • country: the code of the country where it belongs to. • province: the name of the province where it belongs to. • population: population of the city. • elevation: the elevation (above sea level) of the city. • latitude: geographic latitude. • longitude: geographic longitude.

	• country_name: the name of the country. • country_capital: the country's capital.
citylocalname.json	Contains information about the local name of the city. • city: the name of the city • province: the name of the province • country: the code of the country where it belongs to • localname: the local name, usually in a local alphabet (UTF-8)
cityothername.json	Contains information about other city known names. • city: the name of the city. • province: the name of the province. • country: the code of the country where it belongs to. • country_area: the ground area of the country. • country_capital: the country's capital. • Othername: the city's other known name.
citypopulations.json	Contains information about the population number of the cities in different years. • city: the name of the city. • province: the name of the province. • country: the code of the country where it belongs to. • population: number of inhabitants. • year: in which year.
country.json	Contains the countries (and similar areas) of the world with some data. • name: the country name. • code: the country code. • capital: the name of the capital. • province: the province where the capital belongs to. • area: the total area. • population: the population number. • encompasses_continent: the continent name that the country belongs to. • encompass_percentage: percentage, how much of the area of a country belongs to the continent. • continent_area: total area of the continent.

	Note: a country may encompass more than one continent.
country-other-localname.json	Contains information about the local and other name of the country. • country: the country code. • localname: the local name, usually in a local alphabet (UTF-8). • othername: other name for the country.
countrypopulations.json	Contains information about the population number of the countries in different years. • country: the country code • country_name: the country name. • capital: the country's capital. • population: number of inhabitants • year: in which year the population (in the population column) was recorded.
economy.json	Contains economical information about the countries. • country: the country code. • GDP: gross domestic product (in million $). • agriculture: percentage of agriculture of the GDP. • service: percentage of services of the GDP. • industry: percentage of industry of the GDP. • inflation: inflation rate (per annum). • unemployment: unemployment rate.
ethnicgroup.json	Contains information about the ethnic groups in a country • country_code: the country code. • country: country name. • country_capital: the country's capital. • ethnic_group_name: name of the religion. • ethnic_group_percentage: percentage of the language in this country.
ismember.json	Contains memberships in political and economic organizations. • organization: the abbreviation of the organization. • country: the code of the member country. • type: the type of membership.

language.json	Contains information about the languages spoken in a country • country: the country code. • country_capital: the country's capital. • country_area: the country's area. • language: name of the language. • percentage: percentage of the language in this country.
located-on.json	Contains information about cities located in islands. • city: the name of the city. • country: the country code where the city belongs to. • province: the province where the city belongs to. • province_area: the total area of the province. • island: the island it is (maybe only partially) located on. Note that for a given city, there can be several islands where it is located on.
organization.json	Contains information about political and economic organizations. • name: the full name of the organization. • abbreviation: its abbreviation. • city: the city where the headquarters are located. • country: the code of the country where the headquarters are located. • province: the name of the province where the headquarters are located. • established: date of establishment.
politics.json	Contains political information about the countries. • country: the country code. • independence: date of independence (if independent). • wasdependent: the political body where the area was dependent of; usually a country (but not always). • dependent: the country code where the area belongs to. • government: type of government.
population.json	Contains information about the population of the countries. • Country_code: the country code. • country_name: the country's name. • country_province: the country's province.

	• population growth: population growth rate (per annum) • infant mortality: infant mortality (per thousand)
province.json	Contains information about administrative divisions. • name: the name of the administrative division. • country: the country code where it belongs to. • area: the total area of the province. • population: the population of the province. • capital: the name of the capital. • capprov: the name of the province where the capital belongs to Note that capprov is not necessarily equal to name. E.g., the municipality of Bogota (Colombia) is a province of its own, and Bogota is also the capital of the surrounding province Cundinamarca.
provincelocalname.json	Contains information about the local name of the province. • province: the name of the province. • country: the code of the country where it belongs to. • localname: the local name, usually in a local alphabet (UTF-8).
provinceothername.json	Contains other known names for the provinces. • province: the name of the province. • country: the code of the country where it belongs to. • othername: other name for the province.
provincepopulation.json	Contains information about the population number of the provinces in different years. • province: the name of the province. • country: the code of the country where it belongs to. • population: number of inhabitants. • year: in which year.
religion.json	Contains information about the religions in a country • country: the country code. • country_name: the country's name. • country_population: the country's population.. • name: name of the religion. • percentage: percentage of the religion in this country.

Table 1: File names and descriptions.

It is clear that every data file in the set has a specific purpose and provides data for different parts of the knowledge graph.

Part 1 – Understanding the data (20 points)

Use the information provided in the introduction section and try to understand the data. For each file, check the data for each of the columns.

1-1 Is each column value present in every data entry? List which column values do not appear in every data entry.

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?

1-3 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.

Part 2 – Relational database schema (20 points)

Use the Section "Introduction" information and your analysis in Part 1 to create a relational database schema for Oracle DBMS.

2-1 Write an SQL script that contains the schema for the storage and manipulation of the data provided in the JSON files. Requirements:
   a. The database needs to be in 3 NF. This means that you need to have one-to-many relationships between any two related tables. What is more, any data redundancies need to be removed.
   b. Define the primary and foreign keys.
   c. Define the CHECK constraints you came up with for the appropriate columns (question 1-3).

2-2 Compile the entity-relationship diagram of your database. You can do that after you have executed your schema script in SQL Developer. Refer to the handout of the first TA lecture to remember how one can compile this diagram.

Part 3 – Inserting the data to Oracle DBMS (20 points)

In many real-life scenarios across different computer science domains, data may not be provided to you in the form you need them. For example, in project 1, you were provided with the data in the form of INSERT statements, so you didn't have to perform any action but execute the script. In the machine learning domain, the data usually need to be scaled and cleaned up before fitting them into the machine learning module.

Your task here is to implement a simple software parsing tool.

Input: {1} JSON file such as the ones provided, {2} the table name {3} column names

Output: an SQL script with INSERT statements for the table you specify in {2} with the column names you specify in {3} and the data you want from {1}. Specifically, you need statements of this form:

`INSERT INTO` {2} `(` {3} `) VALUES (` data you want from {1} `);`

3-1 Implement the parser using a language of your choice, preferably C, C++, Java, or Python. Document and annotate your code so that a competent programmer can understand its operation. Provide details on how to compile and execute your code. Give examples.

Part 4 – Using MongoDB (20 points)

Installation

In this project, you get a chance to learn and experiment with the popular document-oriented (aka NoSQL) database system, *MongoDB* [1][2]. You have to install the MongoDB community edition locally using the instructions available at [3]. Additionally, MongoDB comes with a powerful GUI, *MongoDB Compass* [4], for querying and analyzing your data in a graphical environment.

Using MongoDB involved two key concepts:

(1) Data modeling [5]

(2) Data aggregation [6]

The former is equivalent to writing DDL in a relational database system such as Oracle, and the latter is to writing DML. There are other concepts that you may need to master in order to improve the performance of your MongoDB aggregation queries. For example, indexing documents [7].

[1] https://docs.mongodb.com/manual/

[2] https://docs.mongodb.com/manual/introduction/

[3] https://docs.mongodb.com/manual/administration/install-community/

[4] https://docs.mongodb.com/compass/current/

[5] https://docs.mongodb.com/manual/core/data-modeling-introduction/

[6] https://docs.mongodb.com/manual/core/aggregation-pipeline/

[7] https://docs.mongodb.com/manual/indexes/

Tasks

You are required to perform *two* tasks:

4-1 Come up with a MongoDB data model to represent the information given in the JSON files. You are required to design your data model (aka schema) while taking important concepts such as data embedding, data referencing, etc., into consideration. You can find a good starting point for MongoDB schema design at [1]. Here, you can reuse the same conceptual schema you used in Part 2 above.

4-2 Once you have your MongoDB schema, write a MongoDB document generator using a language of your choice, preferably C, C++, Java, or Python. You are required to submit a document that includes clear instructions on running your parser. If you choose to use a language other than what is mentioned here, you have to provide clear instructions on how to install the tools required to use your parser.

[1] https://www.mongodb.com/developer/article/mongodb-schema-design-best-practices/

Part 5 – Queries (20 points)

5-1 Implement the following queries in both Oracle and MongoDB. For each of the following queries, capture the execution time for both DBMS environments (question 5-2).

a. Select the name of the country that shares the longest border with one or more other countries.

b. For each country, list the city that has the maximum average population.

c. List the following information of the countries where agriculture contributes the most to their economy. Name of the country, its GDP, percentage contribution from agriculture, and inflation.

d. List the following information of the countries in the descending order of their ethnic diversity. Name of the country, number of ethnic groups, and the percentage of the major ethnicity. Ethnic diversity of a country increases as the number of ethnic groups that live in the country increases.

e. Find the number of countries with the number of ethnic groups equal to the number of languages used.

f. The GDP and IMR (infant mortality rate) together define the development of a country. Find the top 10 countries with the highest GDP (in decreasing order) and their corresponding IMR value.

g. Find the following information of the country with the highest religious freedom. Name of the country, the religions that are practiced in the country. Note that the larger the number of religions practiced in a country, the higher its religious freedom is.

h. What is the proportion of commonwealth countries in the top 100 countries in terms of GDP.

i. Find the country with the largest population density in each continent. List the name of the country and its population density against the name of the continent. The population density of a country is the ratio between its total population and area.

j. Find the names of countries and their capitals where the capital has more than *one* airport.

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

Query number	**Execution time**	
	Oracle	**MongoDB**
5-1-a	…	…
5-1-b	…	…
5-1-c	…	…
…	…	…

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

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.

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?

Submission Instructions

1. Prepare a PDF document with all your answers for Parts 1 to 5 and detailed documentation of your software parsers (for Oracle and MongoDB). The documentation needs to provide thorough instructions on the compilation and execution of your parsers and a description of their operation.

2. Include the source code for each of the parsers in separate directories.

Create a zip file named <username>_project2.zip with items 1 and 2. For example, if your Purdue email is alice@purdue.edu then you should name the file alice_project2.zip. Then upload your zip file through the Project 2 submission page on Brightspace.