Group 24 - Haskell Project

Koustubh Saxena - 230842410
Soham Trivikram Naik Khawte - 230965856
Sai Vivek Rambha - 230585403
Mohammed Rayaan Hussain - 221070112

Summary:

The goal of this project is to implement a Haskell program for retrieving Population and GDP of countries. After retrieval, the data is parsed and saved to a sqlite database.

How does the app work?

Go into the project directory and run **stack run**

The project will be compiled and the program starts off with a prompt asking the user to select one of the given below options -

1. Fetch population of a country
2. Fetch GDP of a country
3. Fetch both Population and GDP of a country
4. Display Population data of all the countries
5. Display GDP data of all countries
6. Update Population data of a country
7. Update GDP data of a country
8. Exit

Let's say the user selects option 1, the program then prompts the user to enter the name of the country. We have implemented fuzzy search so the casing or partial names should not matter. If the user entered "United", the program displays the following -

1. "United Arab Emirates"
2. "United Kingdom"
3. "United Rep. of Tanzania"
4. "United States of America"
5. "United States Virgin Islands"

The user can select the required country and then proceed to enter the year. Even if the user types the country correctly, we prompt the user to confirm his choice with an option number.

We are storing and showing Population and GDP of 2010, 2015 and 2021 as we could only find the data for both Population and GDP for only those years. Making sure of this allows us to perform joins between the 2 tables.

The user can select an option to update data as well through option 6 and 7. The prompts follow in a similar way as before.

The program runs in a loop so the user is not required to run the program every single time. The user can use **“6”** to exit the program.

Haddock documentation has been hosted on Github Pages and can be found at https://cray-mx.github.io/haskell-project/

Data Source:

We Use the UN’s public data portal (https://data.un.org/) to get information about the Population of all countries and their Gross Domestic Product (GDP), in a given year, which are reliable and updated regularly.

Fetching Data:

In the **Fetch.hs** module, we are making a HTTP request to the URLs where UN is hosting its data. The data is being downloaded in CSV format. We write this data to 2 files **“gdp.csv”** and **“pop.csv”** for GDP and Population respectively.

Parsing Data:

The **Parse.hs** module is responsible for parsing and filtering the CSV files which have a lot of unrequired fields. For example, **“pop.csv”** has fields such as “population density”, and “number of people above the age of 14” etc. We had to only select fields such as “total population”. The CSV data also had information regarding continents which we had to discard since we were only focusing on countries. We have a **Types.hs** file which contains the types for all the data we use for parsing, filtering, inserting into the database etc,.

Database Structure:

The database consists of two tables. The first table stores population data and the second table stores GDP data. This simplifies data management and query processing. The **POPULATION** table includes the columns **CountryID**, **Country name** and **Population 2010, 2015** and **2021**. Similarly the **GDP** table includes **Country name, GDP in 2010, 2015 and 2021**. In the **POPULATION** table, **CountryID** is the Primary key and Country Name acts as the Foreign key which links the **POPULATION** table with the **GDP** table.

Complex Problems Solved:

Fuzzy Search -

One of the biggest problems we faced is sanitising the input such as country name when querying from the database. We tried making sure all countries followed a casing where the first letter was capitalised in each word. For example,in the data we found United States **of** America instead of United States **Of** America.

We had many cases of inconsistency in the casing. This motivated us to try and implement fuzzy search. This allowed the user to enter just **“united”** and have the many countries containing that word which the user could use to select the appropriate country. The user can also enter **“unit*”** which would list all the countries with that prefix.

Text Encoding -

After downloading the CSV data from our data source, there was one character which was encoded incorrectly. The parser we implemented was breaking because of this one instance of bad encoding in the downloaded CSV. We had to encode the data into **“char8”** instead of **UTF-8**. We used the **openFile** function to open the file and **hSetEncoding** to handle the encoding into **“char8”.**

Issues Faced

Fetching Data from Web -

We encountered issues with UN’s data as their website was down at times. This motivates us to look at alternative data sources. We found World Bank’s data as another source. The issue with this was the CSV data was zipped and we had to unzip it.

First we decided to use the operating system's own extracting system. But quickly found this approach to be troublesome when we couldn’t run the program on Windows machines. So, we shifted to using libraries. Comparing various zip libraries, we settled on zip.

After working on a POC. We found that the library is dependent upon external C libraries that need to be installed, if we wanted to build the project on other systems. This was not ideal as we needed the project to be buildable on other machines. After some brainstorming, we decided on using docker to build our application. That way, we could control what libraries are installed inside our development environment.