ADVANCED TOPICS OF DATABASE TECHNOLOGY & APPLICATIONS

OVERALL PROJECT REPORT:

Implementation of a data visualization application

TEAM

Antoniou Christodoulos AM: 2641

TABLE OF CONTENTS

SUMMARY

The final goal of the project was to implement a data visualization application which utilizes data integrated into a database. The application should be able to be used in order to visually draw conclusions regarding the depicted data. The data which populates the database was extracted from The World Bank (https://data.worldbank.org/country) and it contains various measurements per year for different indicators for the countries of the European Union. A large portion of the project revolved around designing the database schema and setting up and configuring the database. On top of that strong emphasis was given on the ETL (Extract-Transform-Load) procedure so that the extracted data can integrate in proper structure into the database. In general, the project can be abstracted into 3 sub-projects:

- The database
- REST API Backend
- Front-End Visualizer

THE DATABASE

The first phase of the project revolved around setting up a relational database. For this purpose, **MySQL Server 8.0** was used as a Database Management System along with **MySQL Workbench 8.0 CE** as a visual tool. During the installation of the tools, a general configuration type of 'Development Computer' was specified, meaning that MySQL Server will allocate the least amount of memory while running. For the purposes of the project, the default configurations were mostly sufficient for the database to function properly. However, as part of the project some of the configurations and options were studied. Further details regarding configurations can be found in the Physical Level section below. As for the data that populates the database. It was extracted from The World Bank (https://data.worldbank.org/country) and it contains various measurements per year for different indicators for the 27 countries of the European Union. More details about the transformation and loading of the data can be found in the ETL process section below.

Relational Schema – Logical Level

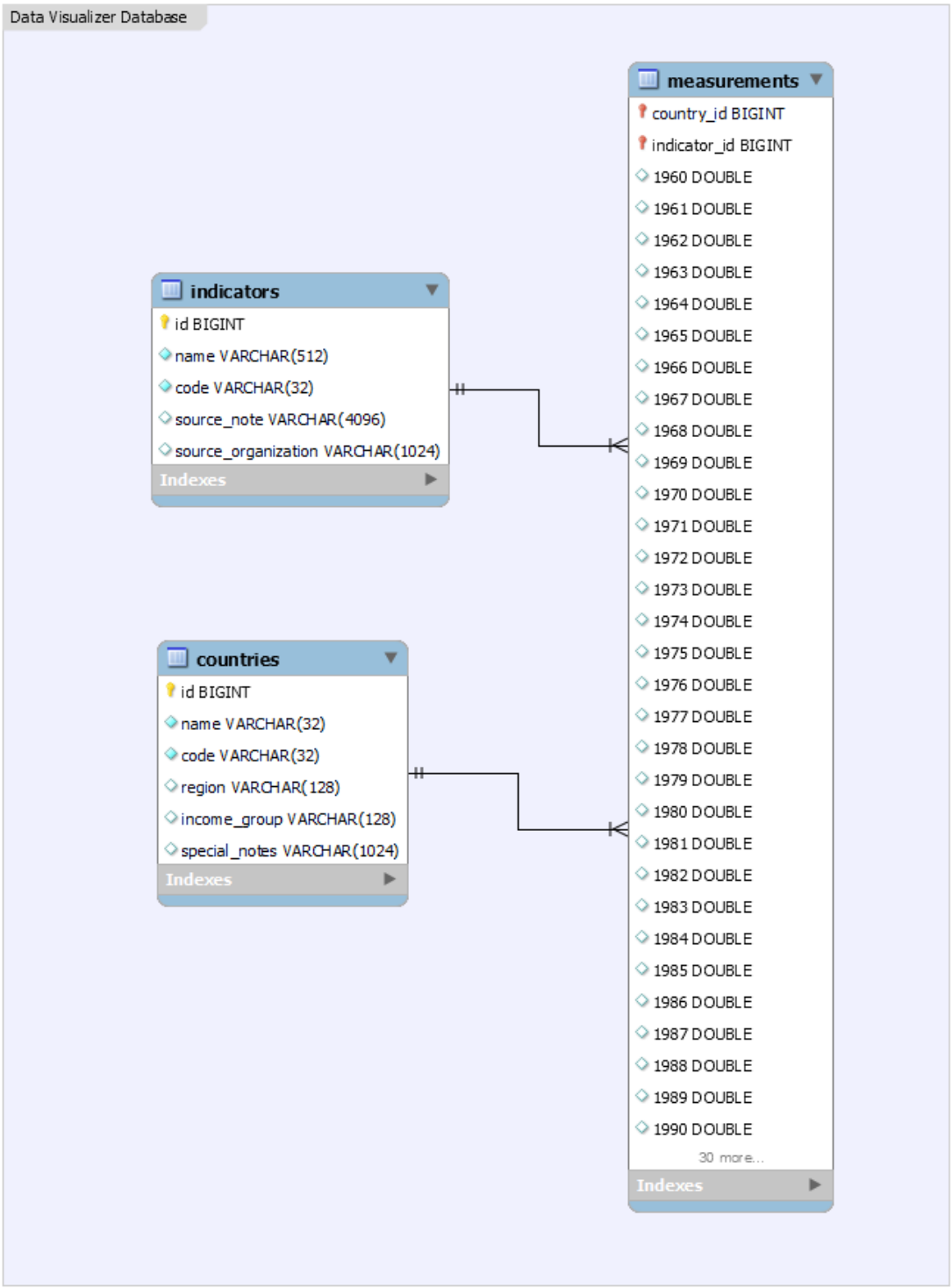
The first step to setting up the database was designing the relational schema at the logical level. Looking into the extracted data files we could determine the following:

- There are multiple countries (each with a name, code, and other information).
- There are multiple indicators (each with a name, code, and other information).
- The measurements are done on a yearly basis, starting from 1960 until 2020.

With that in mind certain interventions were applied so that the data within the database is in 3NF. Countries are stored in a respective table, with a numeric primary key and different column for name, code, and any other information. Similarly, indicators are also stored in a respective table, with a numeric primary key and a different column for name, code, etc. Measurements are also stored in a respective table with a different column for each year, starting from 1960 until 2020. In essence, the measurements are organized per country and per indicator. For that purpose, the measurements table has a primary key that consists of two foreign keys which reference the primary keys of the countries and the indicators tables respectively. The measurements table implements a Many-to-One relation with the countries and the indicators tables.

Looking into the abovementioned relational schema, we can determine that each column expects a single value of a specific type (consistent for each row entry) as well as that each column for each table has a unique name. Therefore, the schema is in 1NF. Moreover, we can determine that there are no partial dependencies since no field depends on only a part of a

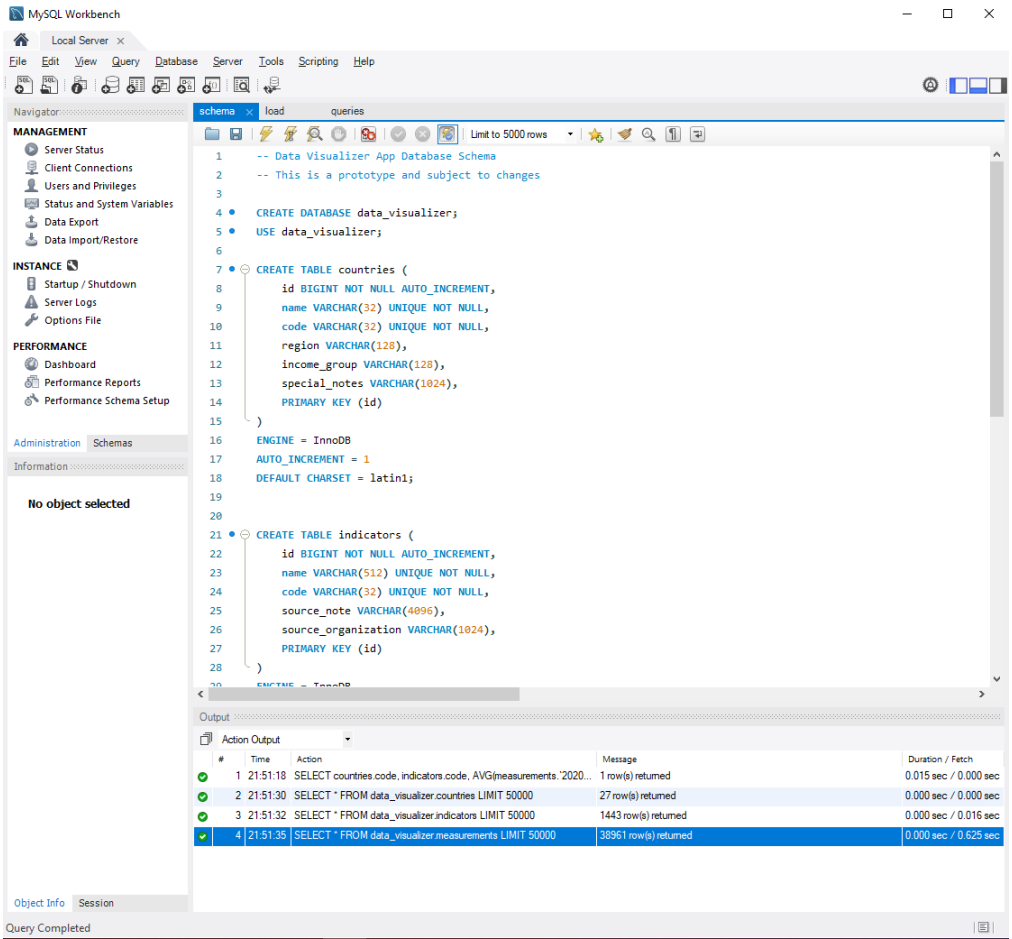
primary key, rather than the whole primary key. Therefore, the schema is in 2NF. Lastly, we can also determine that there are no transitive dependencies since no field depends on another field. Therefore, the schema is in 3NF.



Database Schema Diagram.

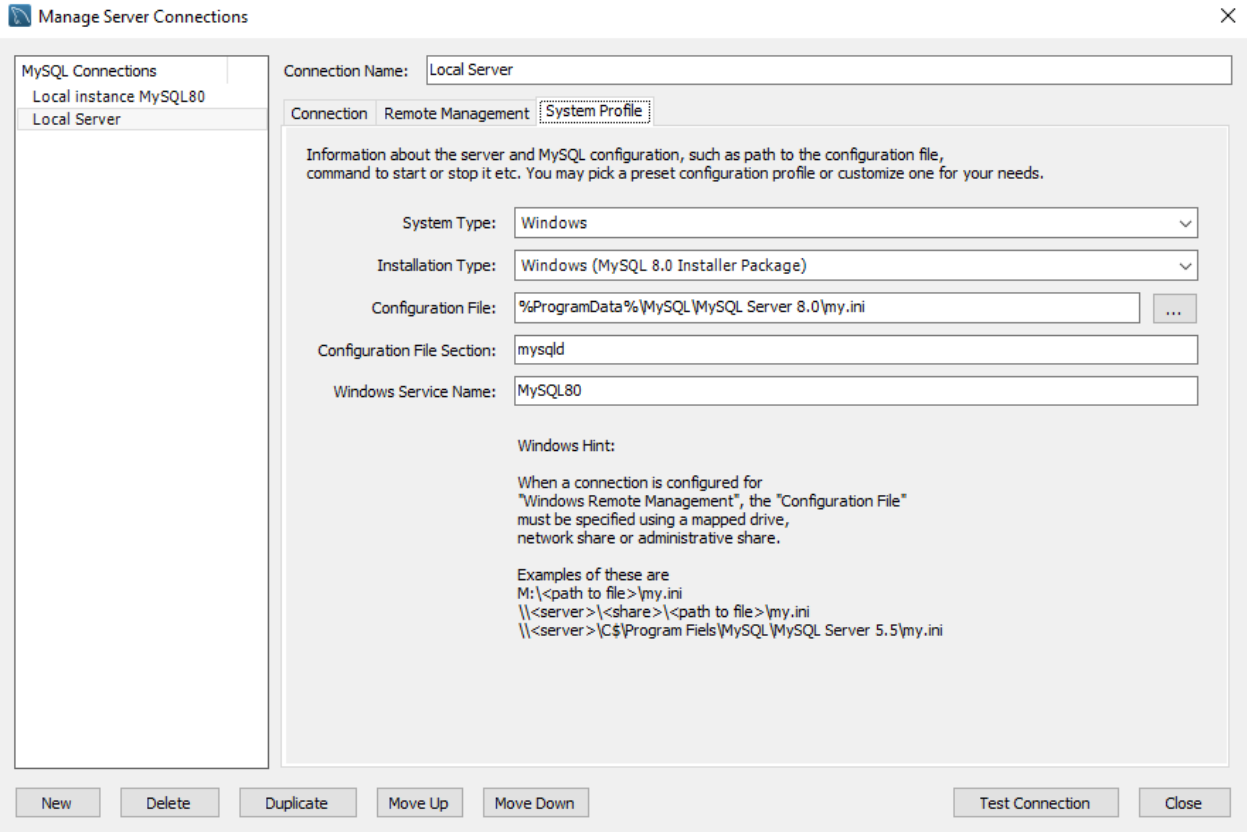
Relational Schema – Physical Level

The next step, after designing the relational database schema, is to actually implement it on the physical level. After the installation of MySQL Server and MySQL workbench a new MySQL connection had to be setup. For the purposes of the project a local server (running on localhost) was created using standard TCP/IP connection method. For the initial database setup the connection was as root user with full administrative roles and privileges. A standard user was also created with the privilege to execute SELECT queries. The standard user is used by the REST API backend to connect to the database. The schema.sql script that creates the database and the relational schema tables can be found in the project files under the **db/schema** directory. It is worth noting that the storage engine is set to InnoDB for each table. Likewise, the default charset is set to latin1 for each table. Additionally, for the countries and indicators tables the auto_increment value is set to 1. Also, the name and code columns for the countries and indicators tables are set to be unique keys, since each country name corresponds to one country code and each indicator name corresponds to one indicator code.



Screenshot of MySQL Workbench at work. Connected as root to the Local Server. The schema.sql script is open.

Further configurations can be specified in the MySQL server configurations file. The configurations file location can be found by going to the menu bar (in MySQL Workbench) and selecting: Database -> Manage Connections -> Select Local Server -> Select System Profile Tab. Note that in order for MySQL Workbench to properly locate the file the system type and installation type might have to be specified. In Windows installations the configuration file ends in .ini while in Linux installations the file ends in .cnf. The installation for the project was in Windows.

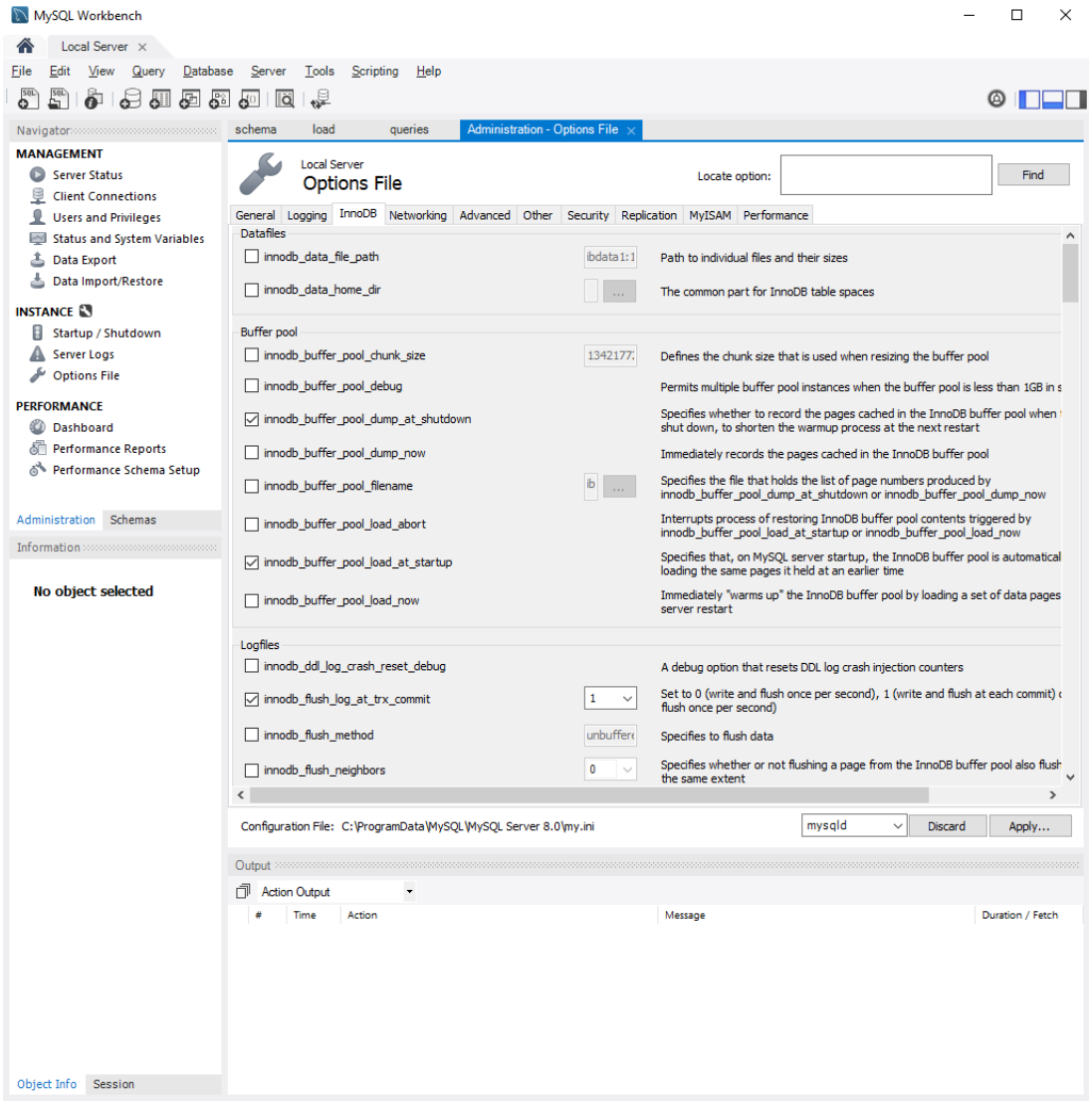


Screenshot of MySQL Workbench Manage Connections window.

The file can be edited from within MySQL workbench by selecting the "Options File" option under the Administration tab on the Navigator menu. Alternatively, the file can also be opened in a text editor and modified from there (recommended). The various configurations available go well beyond the scopes of this project. However, some important configurations can be found below:

- **port**=3306 - The TCP/IP Post the MySQL Server will listen on
- **default-storage-engine**=INNODB – The default storage engine that will be used when new tables are created (this is also specified above on the creation of each table).

- **secure-file-priv**=" C:/ProgramData/MySQL/MySQL Server 8.0/Uploads" – The secure file path from where data can be loaded.
- **innodb_buffer_pool_size**=8M – Amount of memory allocated to the buffer pool used by InnoDB to cache indexes and row data. On a dedicated database server this value may be set up to 80% of the machine's physical memory size. For the puproses of this project, the default value of 8MB seems to be sufficient but can further increased if necessary.
- **innodb_buffer_pool_instances**=8M – The number of regions that the InnoDB buffer pool is divided into.
- **innodb_buffer_pool_chunk_size**=8M – The chunk size for each InnoDB buffer pool instance. It is worth noting that chunk size multiplied by instances should equal the total InnoDB buffer pool size. For the purposes of the project one instance of 8M seemed to be sufficient.



Screenshot of MySQL Workbench. The configurations file (options file) is opened for editing.

The ETL Process (Extract – Transform – Load)

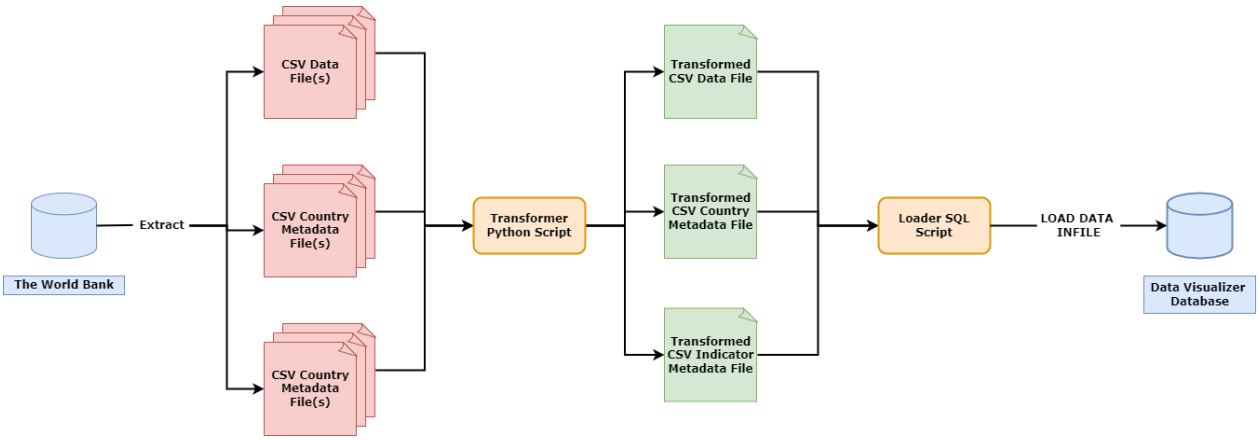
Once the MySQL Server was set up and the schema created on the physical level, it was time to populate the tables with the actual data. However, the extracted data from The World Bank (https://data.worldbank.org/country) are not compliant with the expected structure in the database tables. For this reason, an ETL (short for Extract – Transform – Load) process had to be designed. The data was extracted in Comma Separated Values (.csv) format with three files per country: a file containing the yearly measurements data, a file with country metadata and a file with indicator metadata.

The first step was to develop a python transformer script. The script takes the extracted measurements data, country metadata and indicator metadata files as input and transforms them into three .csv files: data, country metadata and indicator metadata respectively. The script can accept input files for multiple countries, for instance it can accept data and metadata files for all 27 countries of the European Union at once. The transformed data (and metadata) will be outputted at the three .csv files mentioned above. Input and output directories as well as names for the transformed data files can be specified within the script file. The input data files must be placed in the input directory that was specified. It is important for the correct execution of the script that the input data files names retain the naming conventions as extracted from The World Bank. More specifically, indicator metadata files should begin with "Metadata_Indicator", country metadata files should begin with "Metadata_Country" and data files should begin with "API".

Internally, the script works as follows. It reads the comma separated values from the input files and temporarily stores them to country, indicator or measurement objects resembling the tables of the database. Additionally, for country metadata and measurement data files null values are transformed to "\N" which is what MySQL identifies as null during loading. The three types of objects are stored in countries, indicators and measurements lists respectively. Once all the input files have been read all the data exists within these lists. Finally, the transformed data is written to newly created output files. The output files contain the data (or metadata) in a structure that is ready to be loaded into the database. The python transformer script can be found in the project files under the **db/etl** directory. The script was tested on both Linux and Windows operating systems and Linux seemed to be more appropriate. Execution on Windows might trigger encoding related warning on MySQL Server during the loading process.

The final step of the ETL process was to load the transformed data into the database. For this purpose a loader SQL script was developed. The script takes the transformed data, country metadata and indicator metadata files as input and loads them into the database tables using the LOAD DATA INFILE command. The script edits the global local_infile value to true. The full path of each data (or metadata) file should be specified in the first line of LOAD DATA INFILE command. Initially, the country and indicator metadata are loaded directly into the schema

tables. For the measurements data a temporary table (temp_measurements) is initially used to load the data. Once the measurements data is loaded to the temporary table, the data is inserted into the schema measurements table using an INSERT INTO command. This operation is performed due to the fact that the measurements data .csv file contains the country code and indicator code for each measurement while the schema table uses the country id and indicator id as foreign keys to implement the relationship between the tables. However, the numeric id's are auto generated and auto incremented by MySQL and therefore, it is not possible to determine their values before the data is loaded. A SELECT command is used within the INSERT INTO command to relate each country code and indicator code with the proper country id and indicator id respectively. This is how the schema measurements table is populated and the tables' relationships are implemented. Once the above operations is complete, the temporary table is dropped. The SQL loader script can be found in the project files under the **db/etl** directory. It can be opened and executed from within MySQL Workbench.



ETL process diagram.

Database Backup

Once the database schema was successfully populated, the final step was to implement some form of backup to recover from a potential system failure. Considering that for the purposes of the project, data is not dynamically inserted into the database, a static backup would be sufficient. The backup was generated using MySQL Workbench as follows: From the Navigator menu under the Administration tab the Data Export option was selected. The Data Export option allows for selection of tables (along with the data they contain) from the database server to be exported to data dumps in the form of SQL scripts. The scripts are ready to be executed to recover the databases state as it was during the time of the export. Each schema table has been exported to its own separate file. The data dumps can be found in the project files under the **db/backup** directory.

USER STORIES

The data visualizer application supports the following user stories:

Create Bar Charts

- As a: User
- I want: To create bar charts for multiple indicators of multiple countries
- So that: To visually draw conclusions about the depicted data.

Create Line Charts

- As a: User
- I want: To create line charts for multiple indicators of multiple countries
- So that: To visually draw conclusions about the depicted data.

Create Scatter Plots

- As a: User
- I want: To create scatter plots for two indicators of two countries
- So that: To visually draw conclusions about the depicted data.

Search Among Indicators

- As a: User
- I want: To search among the available indicators based on keywords
- So that: To ease the process of selecting an indicator for visualization.

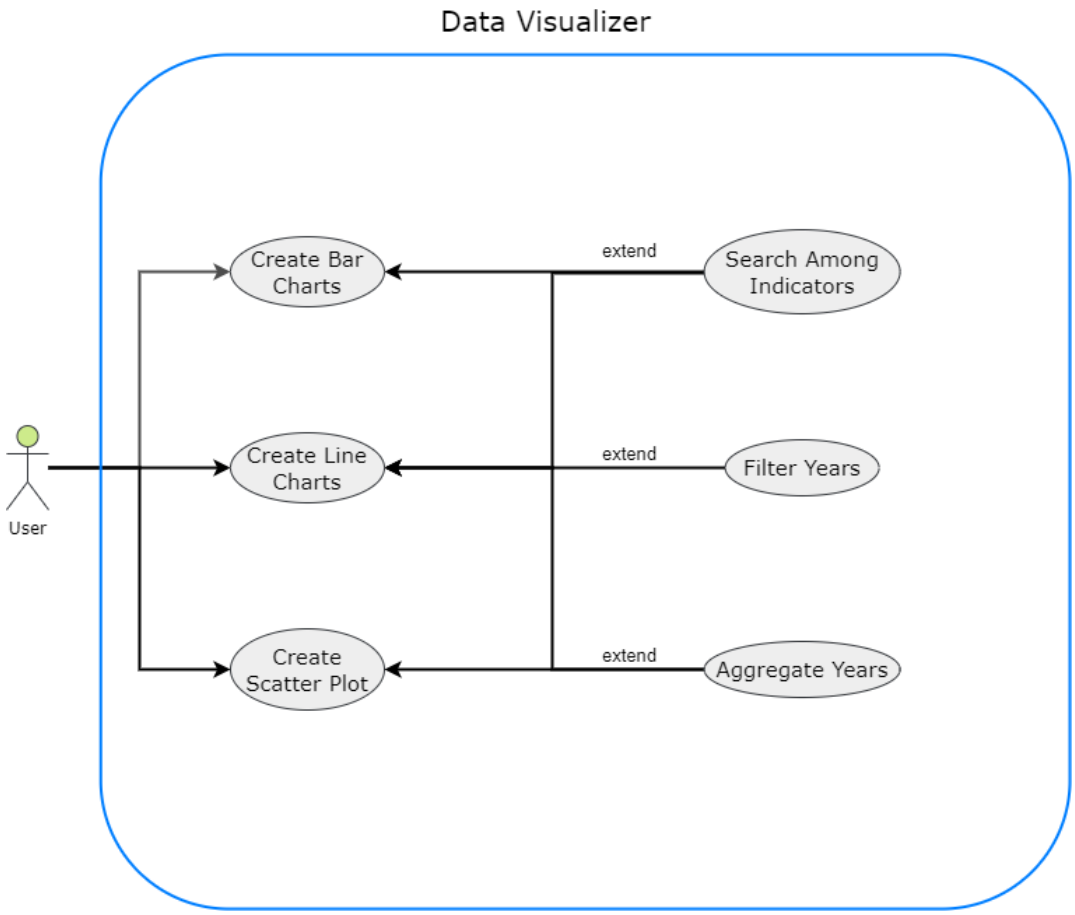
Filter Years

- As a: User
- I want: To filter the years of the selected indicators for visualization
- So that: To visualize only the desired year periods.

Aggregate Years

- As a: User
- I want: To aggregate years into 5-year or 10-year periods
- So that: To compress the visualized data.

The general application usage is described in the UML Use Case diagram below:

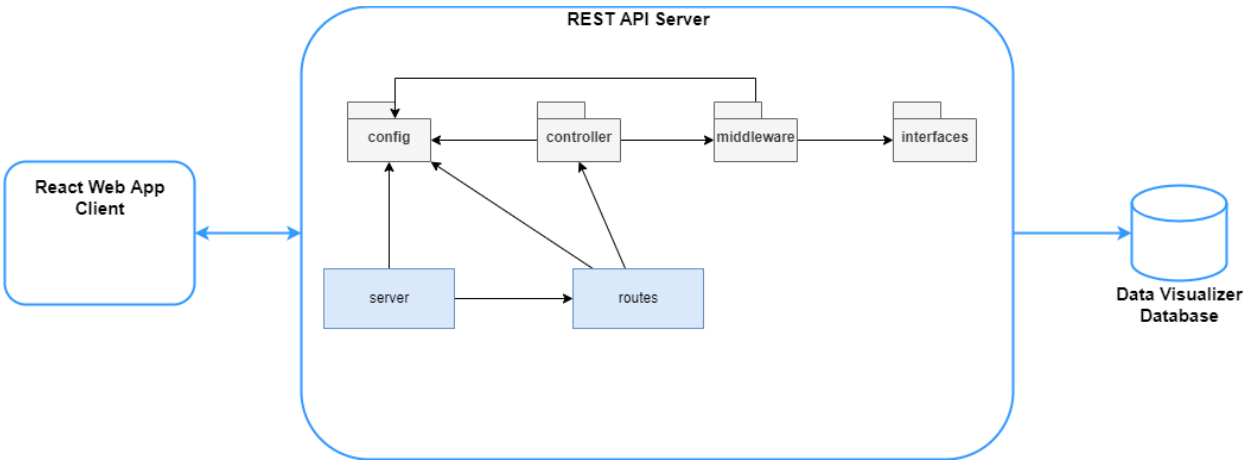


REST API Backend

The REST API server consists of the following subsystems:

- **config**: contains configurations such as the port that the server will be listening on, the implementation of the connection with the MySQL database and a custom logging system to print messages while the server is running.
- **controller**: contains the implementation of queries to the database. The server currently supports 2 static queries which fetch data regarding countries and indicators and a dynamically constructed query which fetches measurements for a specified indicator and country.
- **interfaces**: as the name suggests, interfaces are defined here. The server currently has only one interface which maps the query params required for the measurements query.
- **middleware**: contains functionality regarding the proper construction of the measurements query. Specifically, it contains modules which build the measurements query taking into account whether time aggregation or filtering was selected. It also contains a module responsible for formatting the response.
- **routes.ts**: maps the queries defined in the controller to respective REST API end points. An additional server health check end point is also defined.
- **server.ts**: creates the server.

The overall server architecture is described in the diagram below:



Run command

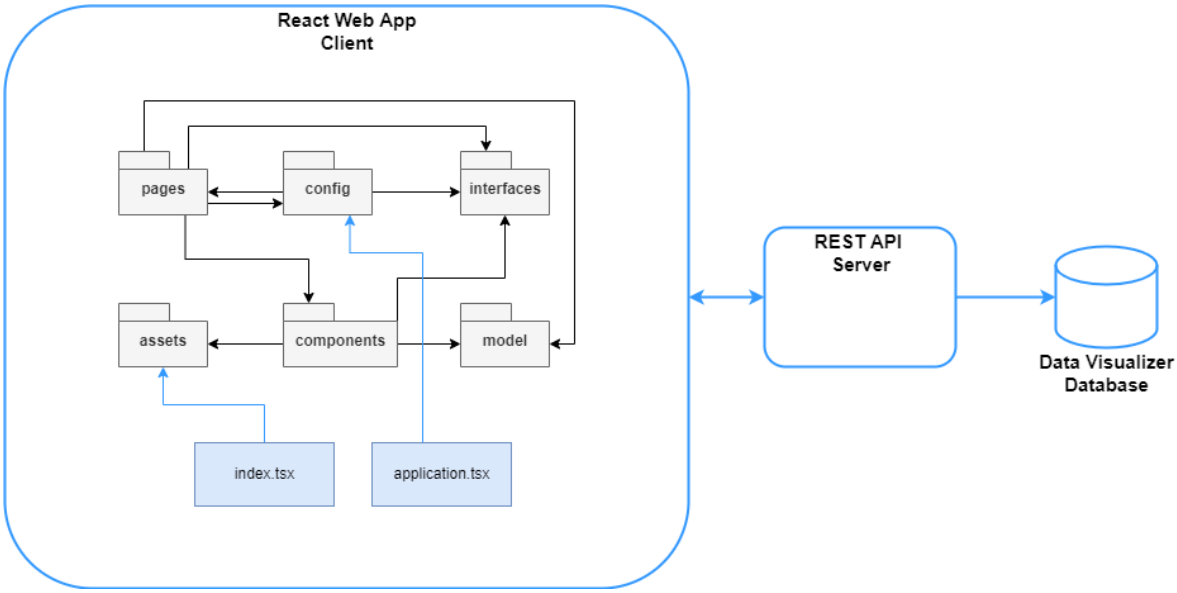
- cd data-visualizer/server
- nodemon src/server.ts

React Web App Front End

The front end client is a web application built with Node.js, Typescript and the React library. The source code consists of the following subsystems:

- **assets**: contains resources such as the background video, CSS dot animations and color definitions.
- **components**: contains the various UI functional components that make up the application's pages. There are components such as dropdown menus, header and navigation bars, the various chart types of the application and input fields were the user can specify the desired query parameters. The components are structured so that they can be reused in numerous pages.
- **config**: contains configurations such as the port were the client will be running, definitions for the routes of the application's pages and a custom logging system to print messages while the client is running.
- **interfaces**: contains UI related interface definitions.
- **model**: contains interface definitions for country, indicator and measurement objects.
- **modules**: contains a module definition so that video playback functions properly. (neglectable)
- **pages**: contains definitions for the pages of the application. Currently, the application has 3 pages. The user is greeted with a welcome message at the home page. The desired visualization is specified at the select page. API calls that fetch countries and indicators are also defined and executed here. Finally, the data visualization is presented at the visualization page. API call that fetches measurements based on the given parameters is defined and executed here.
- **application.tsx**: defines the top level application component, which uses the routes defined in the config subsystem.
- **index.tsx**: defines the top level of the web app.

The overall client architecture is described in the diagram below:



Run command

- cd data-visualizer/client
- npm start