CS3200 Project: Final Report
Group Name: BarronJLangYKannalathA

Food Security Database Application

Link to our video: https://youtu.be/7g1-yxVVzJY

Read[1]Me
Below include the specifications required as well as the detailed instructions on how to run our database application.

Specifications for Project

1. Our project was made through PyCharm using Anaconda Navigator with a working console as our script language for our user interaction.
2. The required package to install is *pymysql* by clicking on the red underline directly in PyCharm and clicking "import".
3. Our application is directory-independent. You can access the application program on the desktop, downloads, or anywhere so long as you have downloaded the SQL dump (refer below for specific file names) and the database exists in MySQL workbench (it has been imported correctly).

To commence the program: *Run main()* in the file ***food_security_front_end.py*** and follow the instructions as they appear. For more information on how to proceed with the program and an overall layout, please refer to the detailed flow chart below.

---

[1] **NOTE: the area table records information for a country, therefore the terms area and country are used interchangeably below.**

Documents in the folder

Description	File Name
1. Database dump with different tables and the back-end SQL procedures, triggers, functions	***food_security_back_end.sql***
2. UML Diagram	***food_security_uml.pdf***
3. Flow chart	***food_security_flowchart.pdf***
4. Reverse engineer diagram	***food_security_reverse_engineer.pdf***
5. Python file with front-end code for user interaction	***food_security_front_end.py***

Requisites for the program to run successfully:

1. MySQL Workbench connection is open (user and password entered successfully)
2. Database exists (SQL dump for *Food_Insecurity* has been run, including all procedures and data)

Technical Specifications

Problem:

1. Database for food security from the World Bank is not in third normal form, thus it removes the redundancy effectively so the data becomes consistent as well as maintains the data integrity.
2. Users are not able to perform CRUD operations on the original database to query or change the data as they wish

Our Solution:

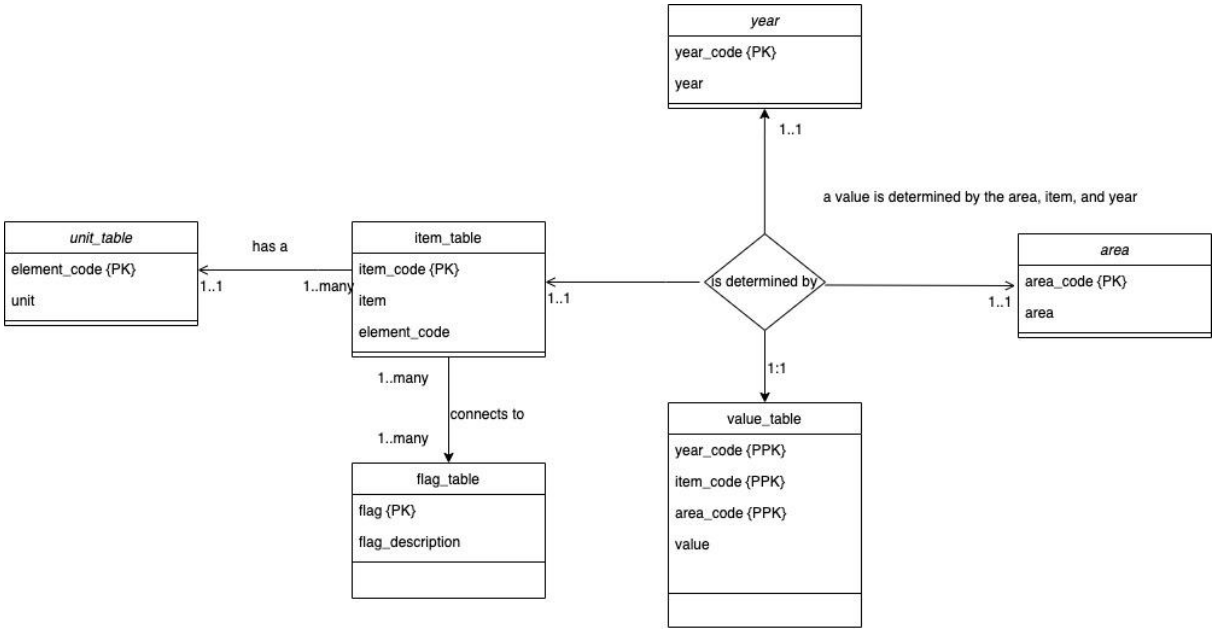
1. Deleting duplicate values and optimizing our dataset such that we have primary keys and foreign keys built for all of our tables (i.e: a year has a primary key which is its year code)
2. In our application program we created an easy-to-use user interface in which all of the CRUD operations are available for the user to utilize with proper feedback when invalid input is given:

a. the list of CRUD operations the user can perform:
    i. Creating a value tuple (which creates tuples in the value table, area table, unit table, and year table)
    ii. Creating values/counts in the value_per_country column in the area table that counts the total number of values a country has, which updates as a value tuple is added to the table
    iii. Deleting a value
    iv. Updating a value
    v. Reading
        1. the country/area code for a given country (Function)
        2. the value given a specific year, item and area codes (Function)
        3. the overall minimum value, its country and the year given an item code
        4. the overall maximum value, its country and the year given an item code
        5. the maximum value and the country given an item code and a year code
        6. the minimum value and the country given an item code and a year code
        7. the maximum value and the year given an item code and an area code
        8. the minimum value and the year given an item code and an area code

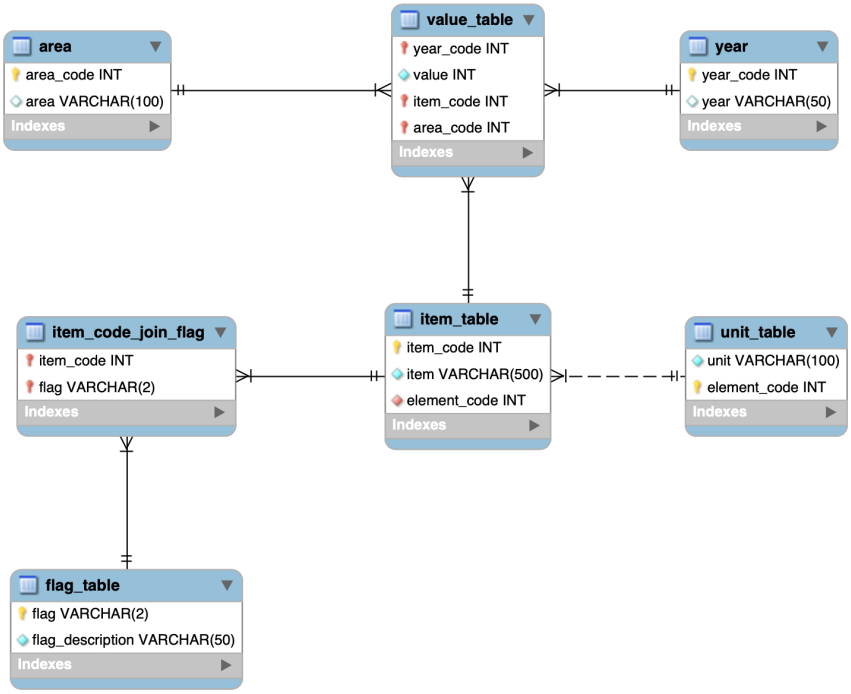
b. Our read operations can also be complex such as querying (relative and universal) maximum and minimum values given different data (year or area) in the database.

3. Error-handling for every single user input with try-catch statements, including the SQL username and password

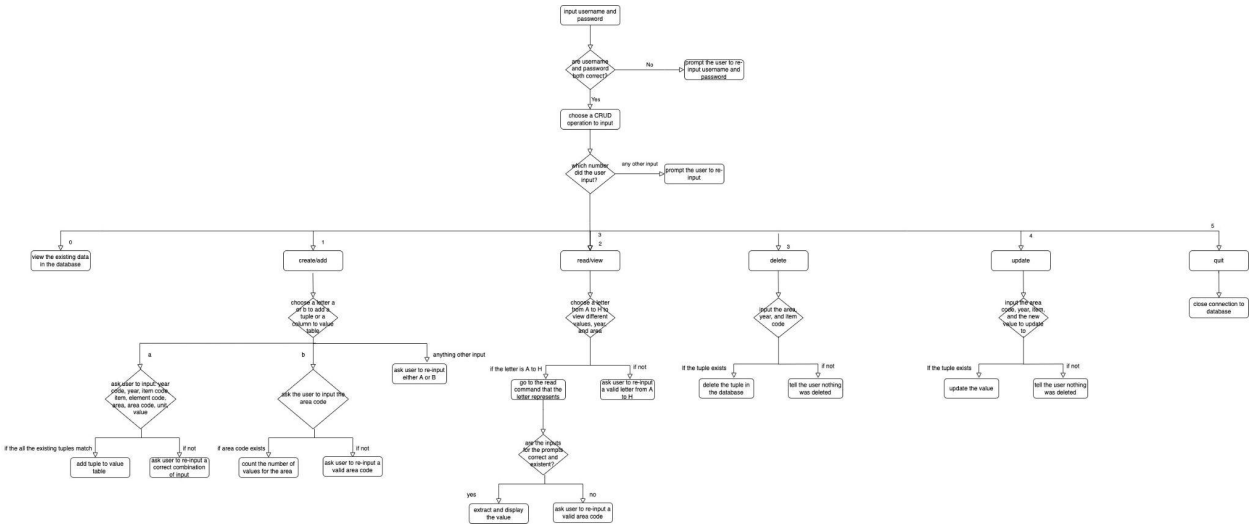
UML Conceptual Design
(see *food_security_uml.pdf* for a higher quality viewing)



Database Schema
(see *food_security_reverse_engineer.pdf* for a higher quality viewing)



Flow Chart of the system
(see *food_security_flowchart.pdf* for a higher quality viewing)



Lessons Learned

I. <u>Technical expertise gained</u>

With this project we furthered our technical knowledge through many facets, the three major ways are delineated below:

1) Considering ease of the use of a GUI to facilitate the user
    a) In this project we needed to really put ourselves in the point of view of the user of the program and to organize our GUI in a way to make the user understand what they were trying to do. This included the following measures:
        i) Error-catching when the user types in an invalid input (e.g: they type in a string for an area code) and prompting them for re-input
        ii) Notifying them when they have connected to the database, closed the connection, or simply the operation they are going to make (i.e: they are performing a CRUD operation)
2) Connecting SQL and Python
    a) Through the use of *PyMySql* we were able to run select statements, procedures, functions and more.

b) We had to figure out how to catch SQL errors in the Python application and display such errors to the user.
    i) This was done through a variety of try/catch mechanisms.

3) Optimizing datasets through normalization and importing data into tables
    a) We started with a first-normal form table, and learned to make third-normal form data in different tables. Our first attempt at a UML diagram and creating tables as described in the project proposal was unsuccessful, since our tables were not able to connect to each other and had a lot of repetitive data.
    b) We re-structured the tables completely so that every table connects to each other (as shown in the database schema through reverse engineering) with no repetitive values by:
        i) Making each table on Excel files
            (1) We learned to delete all the extra spaces on the bottom so they are not accidentally read as null data
        ii) Converting the Excel files to CSV files
            (1) We initially exported them to a wrong format of CSV and our tables were not able to be imported into the SQL tables. We were finally able to realize with the professor's help to export them to Comma Separated Values and our tables imported successfully
        iii) ORDER MATTERS: We learned to import the tables in the order they were created, otherwise the datasets cannot be successfully imported as the foreign keys are in the wrong order
        iv) We learned how to design a logical schema given our data tables, so that we are able to seamlessly integrate our data into the designed schema
    c) Result: we cleaned up the data on the World Bank, which originates from the Food and Agriculture Organization of the United Nations (FAO) (https://www.fao.org/faostat/en/#data/FS), all into third-normal form which are now easy to access

4) Finding ways around conflicting user operations and procedures
    a) We ran into an issue with one of our CRUD operations, CREATE. The procedure initialize_num_value_per_country needed to be run/stored in the SQL database for the user to query it through Python, however,

since we did not create a column value_per_country for the area table, said procedure could not run (always ran into errors) because it tried to update value_per_country, a column that did not yet exist in the area table.

i) Hence, we decided to remediate this by initializing a column when the database is created with fully null values so that the user can simply add these values into the database without running into issues with our procedures.

II. Insights

1. Designing a UML schema and normalizing tables based on data that might not be organized, UML is useful to understand relations among different tables
2. Methods to clean data:
    a. Data normalization
    b. Deletion of null tuples
    c. Use Excel to clean data and find primary, foreign keys
    d. Use CSV files for table upload
3. Naming of variables and tables is important
    a. Make sure the table names do not collide with the variable names, otherwise it can be confusing to perform operations on the data

III. Potential Alternative Designs and Approaches

Alternative approaches to our database application could have included using different programming languages for connecting SQL, like Java and R. However, we have had some experiences creating a user interface through Python so it ended up being our final choice. We also decided to not utilize a NoSQL database due to our vast knowledge in the SQL language itself: we wanted to focus on creating an effective application useful in the real world rather than becoming newly familiar with other databases.

Note:
For the purposes of a food security database, all of the data values are critical to maintain and there is no need to delete tuples in a timely fashion. Hence, events did

not make sense to utilize in our project.

IV. Future Extensions

I. Planned uses of database

Our application includes very useful ways for either the general public or directly the World Bank to query their data, a few are described below:

1. Maximum/minimum yearly values for a specific value like that of malnutrition can be queried by any user, which is useful if the World Bank needs to assess if their measures of intervention to solve food insecurity in certain nations are working.
2. As new data is collected or becomes available for analysis, our application supports the feature to add this data with ease.
    a. And maintaining data integrity while doing such (adding a tuple will never delete any other information with the way the application works)

II. Potential areas for added functionality

A. Querying flags

1. Flags represent specific notes about the data– whether this be it coming from an unreliable source or if the data is an FAO estimate.
    a) We did not use flags in our SQL commands because it is not directly related to the most important and unique variable in our database – the value column. However, we could consider adding the functionality to read/update flags.

B. Multiple ways of filtering our data.

1. The website from which we acquired our data from (The World Bank) included many different ways of filtering our data: by year, location, item names, etc.
2. Currently in our database application we have the ability for the user to query the data that they want, but a future functionality

that could be added is allowing the user to view a larger collection of tuples rather than querying individual ones.

Extra credit attempts:

1. Complicated schema: Multi-join is used for function findValue, overall_max, overall_min,
2. Difficult process for data extraction: many choices for different CRUD operations
3. Interesting queries that can be used for analysis or visualization of the data: there is a column with null values in it called value_per_country in the area table. The user is able to add values (that represent the count of the numbers of values for a specific country) to that column with the CREATE operation and input the country they want to count values for. The back-end SQL command is initialize_value_per_country, and 2 triggers to update the values when a value tuple is added or deleted).