ECE 356 Project Report
Demographics
Group 35

Siddharth Bhattacharjee
Yathartha Panigrahy
Tanmay Khurana

Fall 2021

Project Github: https://github.com/sid42/ECE356_Project_F21

Introduction

This is the report accompanying Group 35's ECE 356 Final Project Submission for the Fall 2021 term. As part of the final project, we designed an entity-relationship model and converted it to a relational schema. We also implemented SQL server-side code to create and load data into the schemas and built a client in Python to interface with the server-side data through SQL.

The data domain for this project was demographic information of several countries. The 2 datasets that were provided by the teaching staff are listed below:

1. https://www.kaggle.com/beatnick/some-country-data
2. https://www.kaggle.com/census/international-data

Additionally, we formatted these datasets to fit our design requirements for this project. This formatted dataset can be found at the link below:

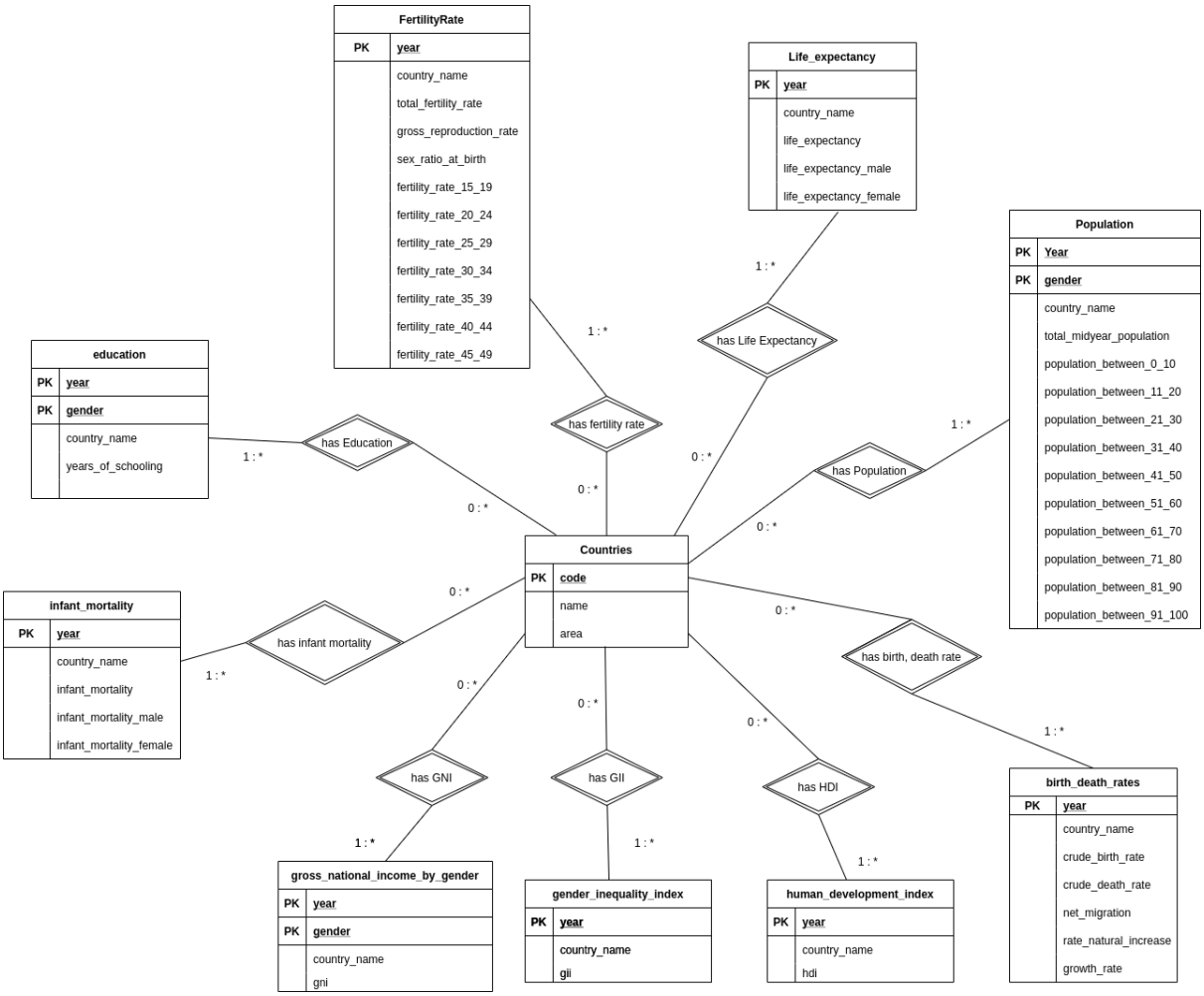
- https://www.kaggle.com/siddharthuw/ece-356-demographics-group-35-additional-csvs

The Github repository for the project code can be found below:

- https://github.com/sid42/ECE356_Project_F21

In this report, we will attempt to elaborate on the design decisions we made while implementing the entity-relationship model, server SQL code, and the Python client.

Entity-Relationship Model



Link to high-resolution image of the ER diagram:
https://github.com/sid42/ECE356_Project_F21/blob/master/server/Demographics-ERDiagram.png

As the first step in completing this project, we designed an entity-relationship model. As seen in the image, the **Countries** entity is central to the design and has weak entity relations with all the other entities in the design. A list of all the entities and relations is given below:

1. **Countries**
2. **Gross_national_inome_by_gender** has a weak entity relation with **Countries** called *has GNI* (0:*, 1:*)
3. **Gender_Inequality_Index** has a weak entity relation with **Countries** called *has GII* (0:*, 1:*)
4. **Human_development_index** has a weak entity relation with **Countries** called *has HDI* (0:*, 1:*)

5. **Birth_death_rates** has a weak entity relation with **Countries** called *has birth, death rates* (0:*, 1:*)
6. **Population** has a weak entity relation with **Countries** called *has population* (0:*, 1:*)
7. **Life_Expectancy** has a weak entity relation with **Countries** called *has life expectancy* (0:*, 1:*)
8. **Fertility_rates** has a weak entity relation with **Countries** called *has fertility rates* (0:*, 1:*)
9. **Education** has a weak entity relation with **Countries** called *has Education* (0:*, 1:*)
10. **infant_mortality** has a weak entity relation with **Countries** called *has Infant Mortality* (0:*, 1:*)

Design Choices for the ER Model

The entity-relationship design choices made in this project were driven by simplicity and practicality:

- Simplicity: We wanted our ER diagram to be reasonably comprehensible yet rich enough to support considerable functionality in the client and server. Thus, for example, by adding weak entity relations between the central **Countries** entity and all the other entities, we managed to clearly define several different entities representing different domains of demographic information in a consistent and comprehensible manner. The relational schema conversion due to weak entity relationships also became straightforward while maintaining richness in the data.
- Practicality: Although we would have liked to have an ER model which perfectly encapsulated all the demographic information of the world, we were constrained by the fact that we only had 2 highly different demographics datasets and had to create the ER model such that would allow extensive usage of the information across both datasets. This is why, for example, there is the extensive usage of country code, gender, and year as primary keys in these relationships as almost all the data in both datasets is found through these attributes.

Cardinality for the ER Model

The cardinality shown from **Countries** to all other entity relations is **0:***. This means that if a country exists in the **Countries,** it does not necessarily have to be present in any of the weak entity relations. For example, there may be no Gross National Income data for Canada, and therefore although Canada will exist in the **Countries** entity, it is not present in *has GNI* relation. However, if Gross National Income data for Canada does exist, then it can be present in the relationship any number of times.

The cardinality shown from all the other entities to their relations is **1:***. This means that if a country or row exists in the entity, then it must exist in the relationship at least once. There is no upper bound on how many times it can exist in the relation

Relational Schema

After designing the entity-relationship model, we translated the model into a relational schema. This transformation was relatively straightforward as all relationships in the ER model are weak entity relationships primarily dependent on the **Countries** entity. This meant that all entities could essentially be represented simply by their definition in the ER model with the addition of the primary key of the **Countries** entity in the primary keys of all other tables except the **Countries** table. The list of primary keys for each of the tables is given below (bold indicates that this attribute was added due to the weak entity relation between **Countries** and the listed entity):

1. countries(**code**)
2. gross_national_income (**code**, year, gender)
3. gender_inequality_index **(code**, year)
4. Human_development_index (**code**, year)
5. Birth_death_rates (**code**, year)
6. Population (**code,** year, gender)
7. Life_Expectancy (**code,** year)
8. fertility_rates (**code,** year)
9. Education (**code,** year, gender)
10. infant_mortality(**code,** year, gender)

Although the weak-entity relationship makes it straightforward to translate the ER model into a relational schema, the richness of data is still maintained by virtue of the several tables containing different aspects of demographic information.

The primary keys were selected to be as efficient as possible. For example, all tables contain *country_code* and *country_name* attributes and *country_name* could have been selected for the primary key instead of *country_code.* However, since *code* is of type *CHAR(2)* and *name* is of type *VARCHAR(255),* we decided to use country_code as it is more efficient. The same applies for *year*(a 4-digit *INT*) and *gender*(Either ‘M’ or ‘F’, *CHAR(1))*

Design Choices for the Relational Schema

As seen in the design choices for the ER model, the primary reason for structuring the relational schema the way we did was to accommodate the data present in the datasets. We noticed that across both datasets, the demographic information was mostly compartmentalized by country code, year, and gender. By making these attributes the primary keys, we were able

to consistently and comprehensively consolidate the data present across both datasets.

Additionally, a few more design choices were made with respect to the relational schema:

Foreign Key Constraint

All tables except the **Countries** table contain a foreign key on their *code* attribute referencing the *code* attribute in the **countries** table. This prevents the user from adding information in secondary tables if the country does not exist in the **countries** table and also prevents the deletion of a country from the **countries** table if its demographics data exists in secondary tables.

This was done to ensure that only countries in the **countries** table can have relevant information in secondary tables (secondary tables refers to all tables except the **countries** table). This maintains a hierarchy of information in the server and prevents unexpected values in the secondary tables.

Indexes

Indexes were added to the most commonly used attributes while querying and updating data for all the tables to maintain a balance between query performance and size taken up by the indexes. The indexes used in this project are listed below:

1. Countries
   - index(code) → PK index
   - index(name)
2. gross_national_income_by_gender
   - index(country_code, year, gender) -> PK index
   - index(country_name)
   - index(year)
   - index(gni)
3. gender_inequality_index
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(gii)

4. Human_development_index
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(hdi)
5. Birth_death_rates
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(crude_birth_rate)
   - index(crude_death_rate)
6. Population
   - index(country_code, year, gender) -> PK index
   - index(country_name)
   - index(year)
   - index(total_midyear_population)
7. Life_Expectancy
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(life_expectancy)
8. fertility_rates (**code,** year)
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(total_fertility_rate)
   - index(gross_reproduction_rate)
   - index(sex_ratio_at_birth)
9. Education (**code,** year, gender)
   - index(country_code, year, gender) -> PK index
   - index(country_name)
   - index(year)
   - index(years_of_schooling)
10. infant_mortality(**code,** year, gender)
   - index(country_code, year) -> PK index
   - index(country_name)
   - index(year)
   - index(infant_mortality)

UNIQUE Constraint in Countries

A UNIQUE constraint was added to the *country_name* attribute of the **countries** table. This was done to make sure that no 2 countries with different country codes have the same country name.

Data Errors and Data Formatting

One of the datasets contained projected demographic information about countries until the year 2050. These values were removed in the data .sql loading script.

```
-- ================================================================

-- Delete all values where year > 2021

-- ================================================================

DELETE FROM population WHERE year > 2021;
DELETE FROM infant_mortality WHERE year > 2021;
DELETE FROM life_expectancy WHERE year > 2021;
DELETE FROM fertility_rates WHERE year > 2021;
DELETE FROM human_development_index WHERE year > 2021;
DELETE FROM gender_inequality_index WHERE year > 2021;
DELETE FROM gross_national_income_by_gender WHERE year > 2021;
DELETE FROM education WHERE year > 2021;
DELETE FROM birth_death_rates WHERE year > 2021;
```

Link to data loading script:
https://github.com/sid42/ECE356_Project_F21/blob/master/server/load.up.sql

Additional python scripts were used to format the CSVs from the original datasets into CSVs that were usable by the loading scripts.

Testing

The project code contains a *server.test.sql* that runs queries to test the design choices listed above. The file can be found here: https://github.com/sid42/ECE356_Project_F21/blob/master/server/server.test.sql

Client

The client to interface with the server is written in python and can the code can be found here:
https://github.com/sid42/ECE356_Project_F21/tree/master/client
The commands and their options can be by typing *help* once the client is running.

Ideal Client Requirements

The ideal client would have the following features:

1. Allow query, update and delete operations as the data domain permits
2. Contain transformative commands / operations that serve additional insight into the data
3. Contain consistent and user-friendly commands that reduce command complexity as much as possible
4. Comprehensive commands whose inputs are not tightly coupled to the relational schema design
5. Comprehensive automated test coverage for all client operations including edge cases

Actual Client Proposed

We would have liked to implement the following features in our client application:

1. Allow query, update and delete operations as the data domain permits
2. Contain consistent and user-friendly commands that reduce command complexity as much as possible
3. Comprehensive commands whose inputs are not tightly coupled to the relational schema design
4. Comprehensive automated test coverage for all client operations

Actual Client Implemented

The following features were implemented for the client:

1. Allow query, update and delete operations as the data domain permits
2. Comprehensive commands whose inputs are not tightly coupled to the relational schema design
3. Rudimentary automated test coverage for all client operations

Tests for the Client

Testing for the client was done through custom automated tests which can be seen here:
https://github.com/sid42/ECE356_Project_F21/blob/master/client/test.py

Justification for the Client Design

Justification for dropping consistent commands: Two of the tables in the database, namely *population* and *fertility_rates,* contain the most number of attributes of all the tables. To accommodate the larger number of attributes while inserting and updating data into the table, the commands to add information for these 2 tables is inconsistent with the commands to add information for the remaining tables

Justification for reduced coverage of test cases: Due to the complexity of implementing coverage for edge cases for all operations, tests only cover the basic create, read, update and delete operations for most tables.