VACCINE DISTRIBUTION DATABASE PROJECT REPORT

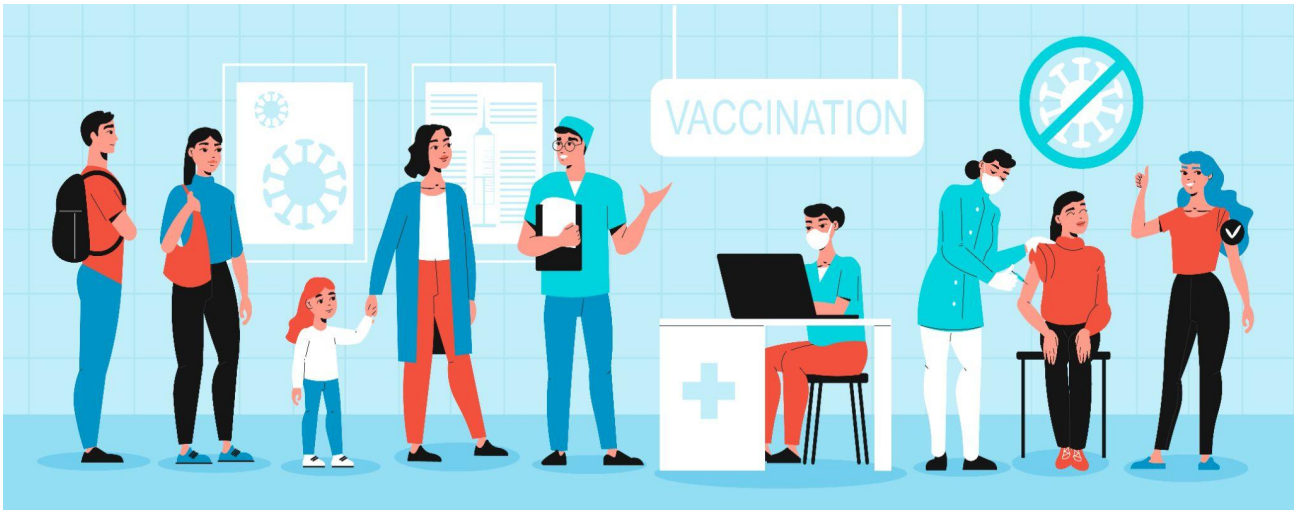


Image by macrovector on Freepik

Ivan Feng

Priscilla Ong

Tien Chu

Hoang Mai

DATABASES FOR DATA SCIENCE 2023

Contents

1. Project Overview

The successful development and distribution of COVID-19 vaccines around the world have become a pivotal moment in the fight against the pandemic. With millions of doses administered worldwide, it is essential for public health services to keep track of the vaccination progress to monitor immunisation efforts. By consolidating vaccination data and being able to organise it in a structural manner, it is easier to assess the impact of immunisation campaigns and identify areas for improvement. If done right, we can leverage data-driven insights to combat the future pandemic and protect communities through widespread vaccination.

With that spirit in mind, throughout this project, our team attempted to build a working and robust database for the task of vaccine distribution from the beginning. We feel that this topic is especially relevant in today's context, given how countries across the globe have been distributing vaccinations to its people in response to the deadly COVID-19 pandemic.

For this project, we:

1) Designed the database with the help of Unified Modelling Language (UML) diagrams,
2) Implemented the database,
3) Performed data cleansing,
4) Conducted preliminary data analysis.

While we acknowledge that real world data and systems are going to be much more complex and messy, this project aims to model real world systems as closely as possible.

2. Design and Considerations

2.1 Introduction

The data we receive is essentially a mock database given as an excel file, and the requirement is creating a good relational database from the given data and implementing the requested queries. This process, in other words, is designing a database according to existing data as well as accessing data correctly for a given purpose (e.g. data analysing).

This database, in practical terms, is a test double. As the given database is much smaller than a real one where there can be millions of rows, its queries can be manually checked and mistakes could be discovered before running on the real database.

2.2 Assumptions

We made the following assumptions to construct the model:

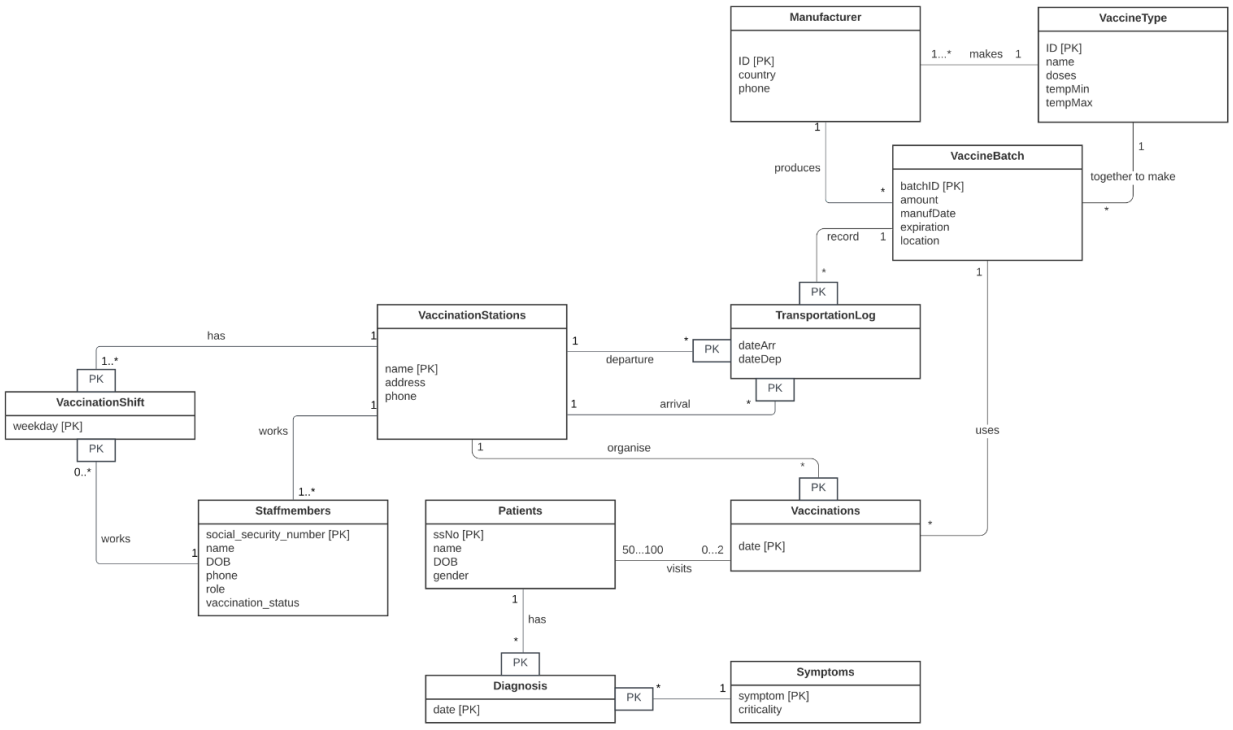
1. Each Manufacturer can only make one vaccine type (and hence one vaccine id).
2. Each patient can attend at most one vaccination event per day.
3. For diagnosis of symptoms, each patient can visit multiple doctors on the same day, but not at the same time. This also means that only one doctor is seen per visit.
4. Each Staff can only work in one hospital.
5. Each Staff can be allocated multiple weekday shifts for vaccination events. However, no staff has a vaccination shift over the weekends.
6. Most Staff would be assigned Vaccination duties.
7. Each manufacturer, hospital, and patient has only one contact number.
8. We make no assumption that phone numbers have to be unique to an entity.
9. We assume each entry of the transportation log contains exactly one vaccine batch.

While the aforementioned assumptions impose certain restrictions on the database design, they are plausible when accounting for what happens in the real world.

2.3 Database modelling

2.3.1 UML Model

The UML diagram of the final model, along with its associated relational schema, can be found below:



2.3.2 Relational Schemas

The design is heavily influenced by the excel file's format as it is convenient for the implementation, and we do not see any redundancies problem following it. We designed the database relational schemas as below:

VaccineType (ID, name, doses, tempMin, tempMax)

Manufacturer (ID, country, phone, vaccine)

VaccineBatch (batchID, amount, type, manufacturer, manufDate, expiration, location)

VaccinationStations (name, address, phone)

Transportationlog (batchID, arrival, departure, dateArr, dateDep)

Staffmembers (social_security_number, name, DOB, phone, role, vaccination_status, hospital)

Shifts (station, worker, weekday)

Vaccinations (date, location, batchID)

Patients (ssNo, name, DOB, gender)

VaccinePatients (date, location, patientSsNo)

Symptoms (name, criticality)

Diagnosis (id, patient, symptom, date)

3. Implementation

3.1 Ensuring Boyce - Codd Normal Form

After deciding on the relational schemas, the first thing to do is to ensure Boyce-Codd Normal Form (BCNF) for the relations to avoid redundancies that can cause anomalies in the system. Since this database is simple, it is relatively easy to check for BCNF as most of the relations have either one or all of the attributes forming the primary key. The one remaining Vaccinations relation that does not exhibit this property is easy to be checked.

Assuming that the relations are more complex, the workflow to check for BCNF follows:

- Find a tool designated for checking BCNF (keyword: BCNF solver, BCNF normalisation tool, etc…). Either you can find them by Googling, and they might come in convenient forms like interactive websites or handy python scripts. Professional database schema tools are likely to provide such a feature, but our group has not tried any of them.
- Input your relation schema (one relation at a time), as well as all the functional dependencies. The tool should state whether the current relation is in BCNF form.
- If the current relation is not in BCNF form, break it down into smaller relations and try the previous step again.

3.2 Scripting

We used python to parse the given excel, create, then populate the corresponding relations. In the end we have:

- One python script to create the database tables, as well as populate them.
- One SQL file to create the tables in the database.
- One python file to answer the questions in part 2 of the project.
- One jupyter notebook to answer the questions in part 3 of the projects.

Here is a summary of our workflow:

The first step is to make experiments in a jupyter notebook. Jupyter notebook offers an advantage over .py files by allowing "partial file execution.". This means you can select specific code cells to execute, making the process of trial and error much less troublesome. This feature, coupled with the ability to document using markdown, is why data-related work often favours notebooks over plain Python scripts. In cases where the cell is expected to modify the database, we utilise DBeaver to manually verify the outcome.

Once we are satisfied with the notebook, we proceed to convert it into a Python script. The Jupyter notebook extension provides a tool for this conversion. The resulting script is then executed multiple times on group members' machines. Between each testing iteration, the database is deleted. To validate the data process, we compare the tables in the database to the corresponding rows in the Excel sheets, as the database's relations closely resemble those of the Excel sheets.

3.3 Brief User Manual

For database setup, the project deliverable package specifically contains these following files:

1. The SQL file used to create the tables in our database
   `create_base_tables.sql`
2. The python file used to connect to the database, import, clean and populate data into the tables, and execute sql queries

`postgresql_database_creation_and_population.py`

3. The .txt file containing necessary dependencies
   `requirements.txt`
4. The .json file storing our credentials separately for login purposes
   `course_credentials.json`

Instructions to run the Python script are as following:

1. Install dependencies in - `requirements.txt`. We added `openpyxl` as a dependency to read the Excel file with sample data and `tabulate` for display.
2. Update Database credentials - `course_credentials.json`
3. Once the user has the virtual environment and the right credentials, the following command can be executed in the terminal to run the python script:

```
python postgresql_database_creation_and_population.py
```

4. Analysis and Improvements

In this section, we analyse the database we have come up with, as well as provide a list of possible areas of improvements that could be pursued.

4.1 Analysis

The data that was presented to us was rather clean – the relations to be modelled were explicitly laid out in the excel file, and there were minimal preprocessing steps that had to be taken to implement the database. For example, the excel containing the relations already had minimal redundancies (with most being in Boyce-Codd Normal Form), which meant we did not have to spend too much time on revising and/or designing the database after receiving the data. In reality, medical data is messy and it may not be so straightforward to design a database system that contains minimal redundancies, while still being accurate.

As an add-on, we generate and present some visualisations based on the data we have received. These visualisations can be found as part of the submission package in a separate document.

4.2 Areas of Improvement

At present, many of the design choices we have made were influenced by the data we received for the database. For example, after performing some preliminary analysis, we discovered that the team of medical personnel is mutually exclusive from the set of patients we had received. Consequently, we simplified the database design and split these two groups of people into different classes: StaffMembers and Patients. Admittedly, this is not a realistic scenario as medical personnel could very well be patients. Moving forward, one area of improvement that could be made would be to take this possibility into account, which would make the database more robust and general.

In addition, when designing the database, we had made certain assumptions which were plausible under the set of data we had received. For instance, nurses were only assigned vaccination shifts at the same location, which allowed for us to represent their shift data in a rather straightforward manner. However, in the real world, it is widely known that the medical facilities are experiencing resource shortages in light of the COVID-19 pandemic – having nurses work at only one location may not accurately reflect what happens in the real world. Furthermore, it is possible for shifts to change from week-to-week. Hence, another area of improvement would be to revise the representation of the relation VaccinationShift in the database to handle the aforementioned possibilities.

During our project, we encountered an issue with SQLAlchemy's handling of transactions, resulting in a persistent lock on the database. This problem becomes more pronounced when multiple group members execute their scripts simultaneously against the same database. As a result, we were unable to modify the table as the “to_sql()” function stay endlessly during the first two questions of part three of the project. To resolve this, we need to invoke “.commit()” at the end of each code cell to end the ongoing transaction, and ensure that we close the connection at the conclusion of the notebook. Failing to release the locks can negatively impact the database's functionality and may even lead to system deadlocks.

While the current database is relatively small and optimising it for performance would not result in any significant performance boost, it is important to note that in much larger

databases with numerous tables and hundreds of thousands of rows, setting up appropriate indices can greatly reduce the query time. There are numerous demonstrations on YouTube showcasing adding the correct index can result in query time reductions of up to 100 times. Although we understand the advantages of using indices, we are still unfamiliar with efficient techniques for crafting them. Optimization is a delicate art that takes time to master, and as novices, we have yet to attain that level of expertise.

Another improvement aspect can be considered is to implement triggers in the database. An SQL trigger allows us to specify SQL actions that should be executed automatically when a specific event occurs in the database. Triggers are beneficial and useful with the three most common case uses: enforcing business rules, automating tasks and maintaining database integrity. In this vaccination distribution database specifically, there are a few possible scenarios where triggers can be applied, for example:

- A trigger to ensure that the number of each patient's vaccination entries should not exceed the number of doses required. In other words, a patient should not take more vaccine doses than needed.
- A trigger to ensure that a patient cannot attend more than one vaccination event per day.
- A trigger to ensure that a vaccination station cannot organise multiple vaccination events on one weekday. A second invalid entry should be removed by the trigger.

5. Working as a Team

5.1 Collaboration Tools

For the project, we have used different communication platforms and collaboration tools, including:

- **Telegram** for daily communication, scheduling meetings or ad-hoc discussions related to the project.
- **Zoom** for online meetings. We also had face to face group meetings at school.

- **Git** for project file sharing.
- **Google docs** for project's documentations and presentation.
- **Liveshare** for real-time coding collaboration.

5.2 Work Distribution

We intentionally assigned overlapping tasks to every member. Our work distribution throughout the whole project is specified below:

Project part 1:

Each team member made a personal attempt to design their own UML model in order to get an overall understanding for the requirements of the project. Subsequently, we presented our ideas and compared our UML models to determine the best one for submission. After that, each team member took responsibility for converting three classes into relational schemas, resulting in a total of twelve schemas. Additionally, we identified the functional dependencies within each relational schema, which will be used to check for BCNF.

Project part 2:

Two team members were assigned the task of developing the Python script used to create and populate the database. The remaining two members took responsibility for crafting the SQL queries required to create the twelve tables. The distribution of the seven SQL queries was overlapped, with each member attempting at least four queries. This approach was facilitated so that members could cross check their answers with one another.

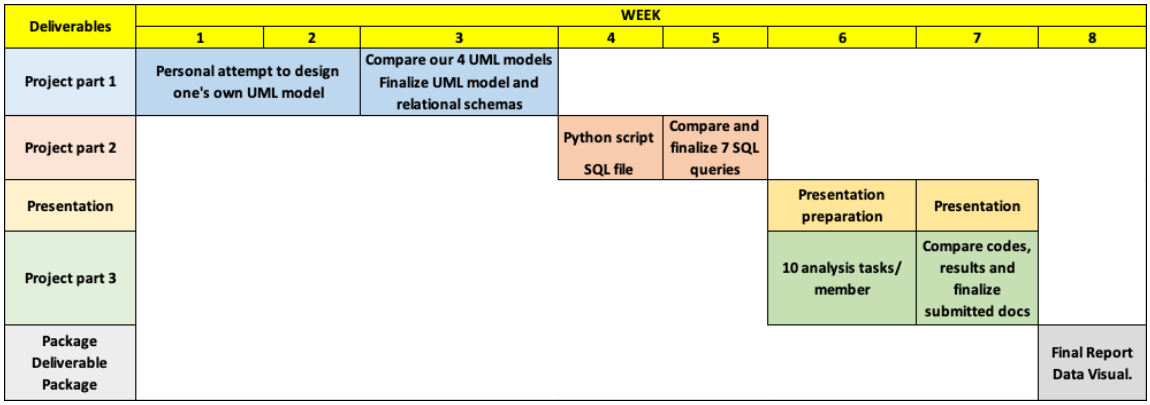
Project part 3:

We encouraged each member to attempt all 10 analysis tasks on Jupyter notebook. Afterward, we shared our results and cross-checked our answers. Once we reached a consensus, we selected the solutions that were the most straightforward and/or elegant and included them in our final submission.

We firmly believe that our approach in distributing the workload would allow each member to maximise their learning opportunities by gaining exposure to a variety of tasks, and not be restricted to a particular aspect of the project.

5.2 Project Schedule

Our estimated schedule for the project during the 8-week period is illustrated as below:



6. Conclusion

Overall, this project has provided us with the opportunity to gain practical experience with simulated real tasks involving working with databases. Furthermore, it allowed us to apply the theoretical knowledge we have recently acquired throughout the course in a hands-on manner. We have derived several valuable lessons from this project:

1. Be comfortable with ambiguity: Database projects often involve dealing with complex and sometimes ambiguous data. It is important for us to be comfortable navigating through uncertainties and ambiguities that may arise during the project.
2. Scrutinise the data: This involves examining the quality, accuracy, completeness, and consistency of the data. It is crucial to identify anomalies, errors, or inconsistencies and take appropriate steps to rectify them. Data cleansing and validation techniques should be applied to ensure the integrity and reliability of the database.
3. Iterate and Improve: It is important to continuously iterate, review, and improve the database design and implementation based on feedback, user requirements, and evolving business needs. This iterative process allows for refinement and enhancement of the database system over time.

4. Communication is key for group work: Effective communication is essential for the success of a group working on a database project. Team members must communicate clearly, openly, and frequently with each other to ensure everyone is on the same page. Regular meetings, discussions, and status updates can facilitate effective collaboration, problem-solving, and decision-making. Clear communication also helps in managing expectations, resolving conflicts, and keeping all team members aligned towards the project goals.

Through this hands-on exercise, we have gained a greater appreciation of the art of designing a database. We believe that the experience gained from this project will put us in good stead for contributing to our future database projects.