Vaccine distribution project

Final deliverable package

Group 2

Adilet Beketov (parts I-III)
Thang Phan (parts I-III and the final deliverable)
Daria Liutina (parts I-III and the final deliverable)

Table of contents

Introduction

The goal

The goal of this project was to create a database which contains data related to corona vaccinations in Finland in 2021, and then analyse it. Some of the data we had to store and look into included:

- who got vaccinated, when and where;
- what kinds of vaccines were used and what side effects related to them were reported;
- how the vaccine batches were used and transported;
- who performed the vaccinations.

Summary of the project parts

The project was divided into three parts. In this section we summarise what was implemented in each of these parts.

1. In the first part we designed a UML diagram and a relational schema based on the given requirements for the data which needs to be stored. Some assumptions had to be made. We also analysed the functional dependencies in our schema and made sure to prevent possible anomalies, even though some minor redundancies were left. It wasn't the final version of the schema yet as we updated it later based on the received feedback.

2. In the second part we created the database on the remote server and populated it with the data provided to us in an Excel file. In this part we wrote a Python script to connect to the database and read the data from the xlsx-file into a dataframe. We then cleaned and prepared the data (changed some of the column names etc.) in Python before writing it to the database. The tables were created via postgreSQL commands in a separate file which we read in the Python script.

   Then we wrote SQL queries (in a separate file) to answer the questions we were asked about the data. Again, the Python script reads the file and prints the results in the console.

3. In the third part we did a more extensive analysis of the data using both SQL queries and tools from the pandas-library. This part included some visualisation, however, we did extra visualisation for the final stage of the project.

The final deliverable package compiles the results of all the three parts of the project, as well as some extra visualisation+GUI and further development ideas. The package includes this documentation text, as well as all the code and instructions for running our project on any computer (as long as there's a connection to the database on the remote server).

Running the code

In order to run our code, one needs:

- access to the server with the database,
- any IDE which can run Python scripts, and
- packages listed in the requirements.txt.

There are two Python scripts: Python_script.py from part II, which populates the database and runs some of the queries, and assignment3.py from part III, which analyses the data in the populated database. There are also two SQL files: tablecreation.sql creates the tables in the database, and SQL_queries contains the queries from part II. The Python_script.py reads both of these files, so no action from the user is needed, as long as all of these files are in the same folder.

Running the visualisation part happens via terminal. The exact instructions are in the README.md file. The visualisation happens locally and opens in a browser. There's a GUI with some extra features, e.g. medical workers can find the shifts assigned to them.

Design and Use Cases

Design assumptions

In order to design the relational schema we had to make some assumptions. The full list of the assumptions is the following:

- a social security number (ssNo) is unique and determines the rest of the person's data
- one manufacturer can only receive one licence for producing one type of vaccine so there can be only one vaccine ID associated with the manufacturer's ID;
- the same vaccine batch may be transported several times (also between the same hospitals);
- one person can be only one clinic's employee;
- vaccine batches can't be split between hospitals;
- a patient might report a symptom several times;
- a patient only attends one vaccination event per day;
- a person is fully vaccinated (status = 1) if they have received at least two doses of vaccine (regardless of the vaccine type, i.e two doses of different vaccines will mean full vaccination).

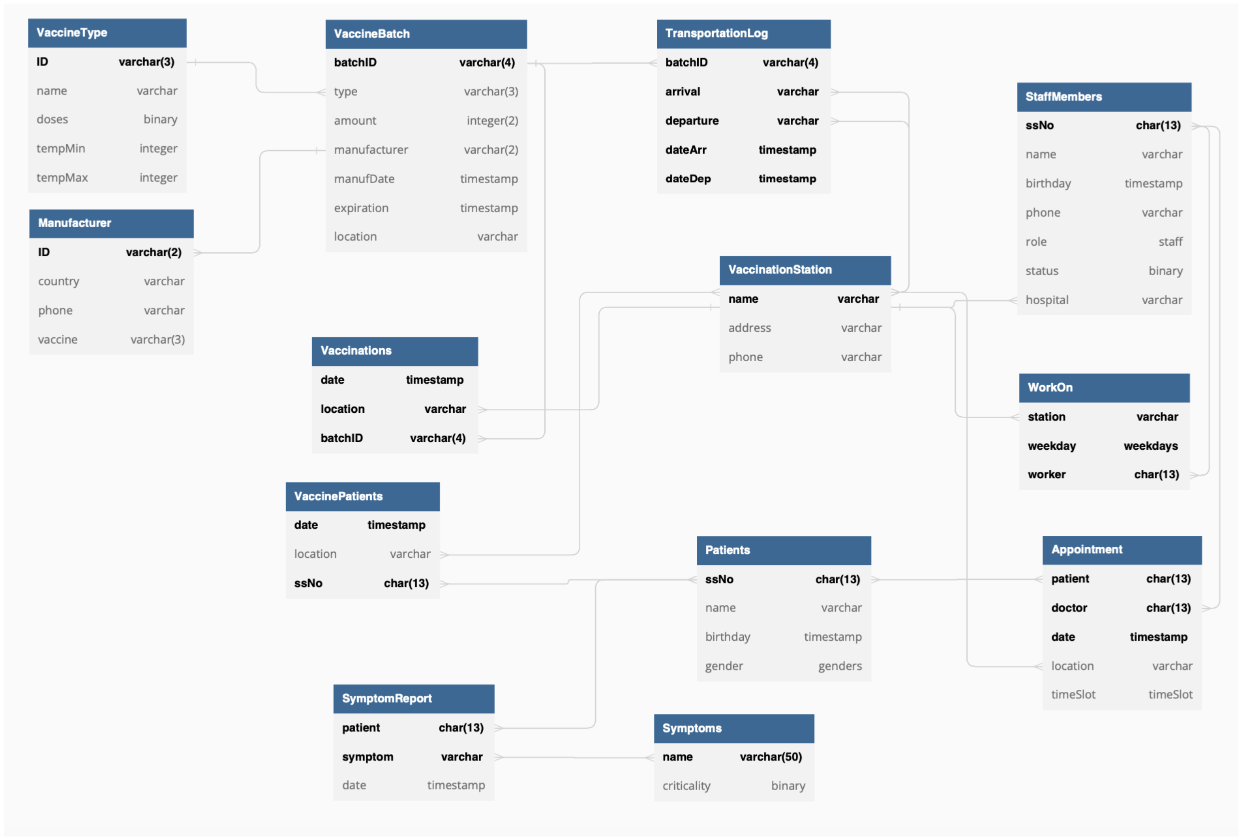
There were also assumptions that we made that would be necessary in case of scaling the database, as the provided data already satisfied these conditions. They include:

- all the dates are character strings in the format 'year-month-day';
- manufacturer ID is an 2-character long string and it's unique;
- vaccine ID is a 3-character long string and it's unique. It can also be referenced to as vaccine type;
- batch ID is a 3-character long string and it's unique;
- every vaccine type requires 2 doses;
- critical storage temperature varies between vaccines and is between -90 to +8 degrees Celsius;
- a batch contains at least 10 and at most 20 doses of vaccine.
- each vaccination station can only have one phone number and one address, stored as strings;
- weekday is a character string and it has one of the following values: 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'. There's no vaccination during the weekend;
- names are stored as max 30 character strings;

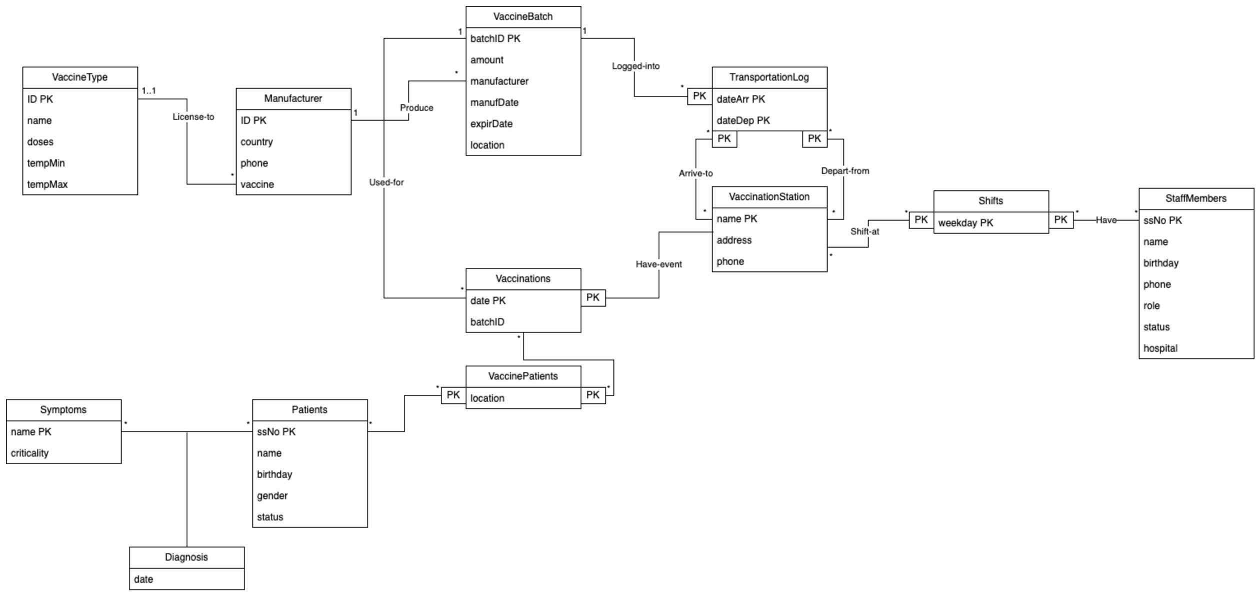
We had a problem with the following assumption that we made about the ssNos: "...it has a fixed form of 'YYMMDD-NNNC', where the first 6 digits come from the birthdate, NNN is a number between 002 and 899 and C is the check character which can be a digit or a letter". As we noticed later, the staff's ssNos had all the four digits of the birth year in them. However, it didn't affect how we processed them so it wasn't a problem. Scalability of the database would require a strict check of the form of the ssNos, more on that in the Further development suggestions section.

The UML diagram and the relational schema

The original version of the UML diagram (made in dbdiagram.io):



The final version of the UML diagram (made in draw.io):



The final relational schema and functional dependencies:

- Manufacturer(<u>ID</u>, country, phone, vaccine)
    - ID -> country, phone, vaccine
- VaccineBatch(<u>batchID</u>, amount, manufacturer, manufDate, expiration, location)
    - batchID -> amount, manufacturer, type, manufDate, expiration, location
    - manufacturer -> type
    - manufDate -> expiration
    - expiration -> manufDate
- VaccineType(<u>ID</u>, name, doses, tempMin, tempMax)
    - ID -> name, doses, tempMin, tempMax
- VaccinationStation(<u>name</u>, address, phone)
    - name -> address, phone
- TransportationLog(<u>batchID</u>, <u>arrival</u>, <u>departure</u>, <u>dateArr</u>, <u>dateDep</u>)
    - No FDs
- Shifts(<u>station</u>, <u>weekday</u>, <u>worker</u>)
    - No FDs
- StaffMembers(<u>ssNo</u>, name, birthday, phone, role, status, hospital)
    - ssNo -> name, birthday, phone, role, status, hospital
- Vaccinations(<u>date</u>, <u>location</u>, batchID)
    - No FDs
- Patients(<u>ssNo</u>, name, birthday, gender, status)
    - ssNo -> name, birthday, gender, status
- Symptoms (<u>name</u>, criticality)
    - name -> criticality
- VaccinePatients(<u>date</u>, location, <u>ssNo</u>)
    - date, ssNo -> location
- Diagnosis(<u>patient</u>, <u>symptom</u>, <u>date</u>)
    - No FDs

Working as a Team

There were three members in our group (the fourth member had to leave the course). All three are with different prior knowledge and school background. The division of work was based on our strengths, but leaving a possibility for each of us to develop and learn new skills.

Prior to the course Adilet didn't have experience with databases, but he was comfortable with Python (and pandas) as well as with development in general. He did some parts of the Python scripts and queries for the third part of the project.

Thang had experience with both development in Python and with databases. He has also done visualisation before. He wrote some of the queries and helped others with the databases-related questions. He also did the visualisation part.

Daria had no prior experience with databases and only the basic knowledge of Python. This made the project challenging but fruitful as she got to learn a bunch of new skills during working on the project (learning git and Gitlab was a big step in the process, not without making mistakes). She did most of the documentation and project coordination.

The practicalities of the development were pretty straightforward. We used Gitlab to store our code and have version control. Communication happened in Telegram and we had group calls (grinding sessions) on Zoom (we tried Discord but there were problems with it when using Aalto's VPN). We used DBeaver for testing our database and queries in the development phase. We also decided to use the remote server during the development phase to make sure one person's changes worked correctly for other team members too. This required just a little bit of caution and communication to avoid damaging the database.

The biggest difficulty was challenging schedules of team members. Sooner or later all the tasks were completed, but often there wasn't much time left. Clear communication was the key to successful completion of the project, even though it alone can't solve all the problems. Supporting each other and trying to solve tasks as a group helped a lot, too. Each of us got to learn a lot and so the main goal of this project - to learn and develop - was achieved.

Analysis and changes

We made a significant improvement by implementing the Unified Modeling Language (UML). Initially, we chose DBDiagram to create the UML diagram, but upon completion, we realized that several essential elements were missing. Consequently, we opted to switch to draw.io for a fresh start, allowing us to accurately represent the UML diagram. The diagram drawn in draw.io is much clearer and includes associations' names, association classes, and clear multiplication symbols.

Given the dataset, we discovered several misconceptions within our database. These included inadequate or redundant tables and attributes, incorrect data types, and missing values in attributes having the "not null" constraint, among others. Recognizing these issues, we fostered open communication and exchanged our ideas on how to address these various problems effectively. At the end of the second part, we decided to remove the "Appointment" table as it wasn't useful and we don't have the data for it and from the "VaccineBatch" table, we also remove the "type" attribute as we assume that one vaccine type only produced by one manufacturer. By doing that we made our database more concise and avoided redundancy.

In the process of writing queries and conducting analyses for parts 2 and 3 of the project, we actively shared our ideas and provided explanations for the code we developed and how it functioned. This collaborative approach ensured a thorough understanding of the rationale behind our code and promoted knowledge sharing among team members. It also helped us to point out our mistakes that we have made or misunderstanding of the task descriptions and make the necessary changes before submitting the result.

About the query performance in our database, since the database is relatively small and we don't have a lot of tables and relations, it usually took less than 1 second (specifically from 30 ms to 200 ms) to execute a query and see the result.

GUI and visualization

The web application built with Streamlit and Plotly offers a seamless and user-friendly experience for interacting with the database. Streamlit’s simplicity and intuitive design make it an excellent choice for developers who may not have extensive front-end development experience. It allows you to quickly create a website by writing Python code, eliminating the need to learn complex web development technologies.

Streamlit provides a wide range of GUI elements that can be easily incorporated into your application. Buttons enable users to trigger specific actions or queries, select boxes allow them to choose from a predefined set of options, input boxes enable data entry, and sliders facilitate interactive exploration of numerical values. These elements empower users to specify tasks such as data visualization or information retrieval from the database.

Plotly, on the other hand, is a powerful graphing library that seamlessly integrates with Streamlit. It enables you to create stunning visualizations, including interactive charts, graphs, and maps, to present the data from your database in a compelling and informative way. With Plotly's extensive customization options, you can tailor the visualizations to suit your specific requirements, allowing users to gain insights.

The combination of Streamlit and Plotly offers a productive workflow for building data-driven web applications. By leveraging the simplicity of Streamlit and the visualization capabilities of Plotly, we’ve created a data-driven applications that enable users to visualize data, explore insights, and interact with the underlying database effortlessly.

Further Development and Business Applications

If we want to turn the database into a product it would require further development. First of all the product design should be implemented in cooperation with the future clients, such as public and private clinics. The clients might express a need for extra features we haven't thought of. As it's going to be a healthcare-related service the industry and the state will set certain requirements for it. They include: data protection against data losses and leaks (including protection against attacks and malicious software), possibility to modify the service to a client's needs, reliability (24/7 technical support and uptime guarantees), compliance with local regulations.

The future development of the database would require scaling of it. In that case more efficient database design is necessary, which involves:

- using indexing for faster queries;
- adding constraints on attributes' sizes to use memory efficiently;
- strict data consistency checks before adding data to the database.

Making the database more user-friendly will be necessary, too. Currently the web app is running locally so building a fully functional web/desktop app is the next step. This would also include designing various levels of access to the database (e.g. medical workers, patients, admins, guest users etc.). No IT skills should be needed for any users to use the app. There should be a way for users to construct and run new queries, preferably in a GUI.

Extending the range of the usage for our product is also necessary to achieve competitive advantage in the market of healthcare IT-products. One way to do that would use ML methods to find hidden patterns and trends in data and use them to make predictions about the future vaccinations. For example, ML can help to predict how many doses of vaccine will be used within a certain time period, so that the supply of them is optimised. Another important factor to consider is adaptability of our service. Corona vaccinations are not permanent and thus it's important to find other usages for our product. One possibility is expanding it to other vaccinations, too, for example to keep track of mandatory and voluntary vaccinations of kindergarten children and school students.

Conclusions

Our database is designed in Boyce-Codd Normal Form (BCNF), which offers numerous advantages. By adhering to BCNF, our database ensures enhanced data integrity, elimination of redundancies, prevention of update, insert, and delete anomalies, and preservation of functional dependencies. We have implemented several rules and constraints to maintain the consistency of the data, promoting reliable and accurate information.

However, it is important to note that our current database design may have some limitations. Since the dataset is relatively small, our focus was not primarily on optimizing query performance. While the database functions effectively for its current size, it may not be as efficient when dealing with significantly larger datasets. Additionally, as the data is static, we have not incorporated specific in-and-out data handling functions into the database design.

In the future, considerations may involve incorporating dynamic data handling mechanisms to cater to evolving requirements. We have outlined several ideas for future database enhancements:

- Scalability: If the dataset grows significantly, we will introduce additional constraints to maintain data integrity. Each attribute may be assigned at least one constraint, ensuring the prevention of anomalies within the database.
- Performance Optimization: Should the size of the database storage increase, we will incorporate appropriate indexes to optimize query performance. These indexes will enable faster data retrieval and improved overall system efficiency.
- Enhanced Security: we will implement additional security measures. This may involve adding a security layer to safeguard against unauthorized access and protect the confidentiality of the data.
- Data Protection: Recognizing the importance of the data, we will incorporate backup and recovery strategies to mitigate the risks of data loss or system failures. Regular backups and a well-defined recovery process will be implemented to ensure data can be restored to a consistent state.

By considering these factors and proactively modifying our database, we aim to ensure its continued effectiveness, scalability, security, and resilience in the face of future changes.

In conclusion, our team's project has proven to be a highly valuable learning experience in the field of database management. We have successfully designed and implemented a fully functional database system that adheres to industry best practices while effectively addressing the project's requirements. Moreover, this project has provided us with the opportunity to interact with the database through tasks such as performing queries, analysis, and visualization, allowing us to gain practical experience in utilizing the database for real-world scenarios.

Throughout the duration of the project, we have acquired a wealth of practical knowledge and hands-on experience in various aspects of database management. We have honed our skills in database design, applied normalization techniques to ensure data integrity, written complex queries, performed in-depth data analysis, and even delved into additional skills not solely related to database management, such as version control using Git and GitLab, coding in Python, and effective teamwork and collaboration.