VACCINE DISTRIBUTION

CS – A1155 – Databases for Data Science

Group 20

Table of Contents

I. Introduction

1. Overview

The main objective of the project is to build a database to keep track of the different vaccine types, transportation of vaccine batches, treatment plans, staff schedules of vaccinations events, and patient data for Corona vaccine distribution and treatment in Finland.

2. Purpose of the database

The purpose of the database is to model and manage the distribution of vaccines. The database is designed to track information about vaccine manufacturers, vaccine batches, vaccination events, patients, and symptoms.

3. Use cases

Here are some potential use cases for the database:

i. Vaccine Distribution Tracking: The primary use case of this database is to track the distribution of vaccines. It can provide information about where vaccines are being distributed, which hospitals or clinics are receiving them, and the quantity of vaccines being distributed.

ii. Vaccination Event Management: The database can be used to manage vaccination events. It can provide information about the staff members working at a vaccination event, the location of the event, and the patients who have been vaccinated.

iii. Patient Tracking: The database can be used to track patients who have been vaccinated. It can provide information about which vaccines a patient has received, when they received them, and any symptoms they have experienced after vaccination.

iv. Vaccine Batch Tracking: The database can be used to track vaccine batches. It can provide information about the current location of a vaccine batch, the last location in the transportation log, and any inconsistencies in location data.

v. Symptom Analysis: The database can be used to analyze symptoms experienced by patients after vaccination. It can provide information about the frequency of different symptoms for each vaccine type, and whether a symptom was diagnosed before or after a patient received a vaccine.

vi. Vaccination Status Reporting: The database can be used to report on the vaccination status of patients. It can provide information about whether a patient has attended enough vaccinations.

These use cases can support a wide range of activities related to vaccine distribution and public health management. They can help health authorities to manage vaccine distribution more effectively, monitor patient health after vaccination, and make informed decisions based on the data.

4. Handling of the database

Setting up the database

First, we need to run the files ‘creating_the_database.py’ to create the database, connect to it and fill it with the provided data.

Queries for creating tables

The queries needed for creating the tables and filling them with the provided data are contained in the the file ‘Script-2.sql’, and we could run it with the same file mentioned above.

Queries for the data visualization

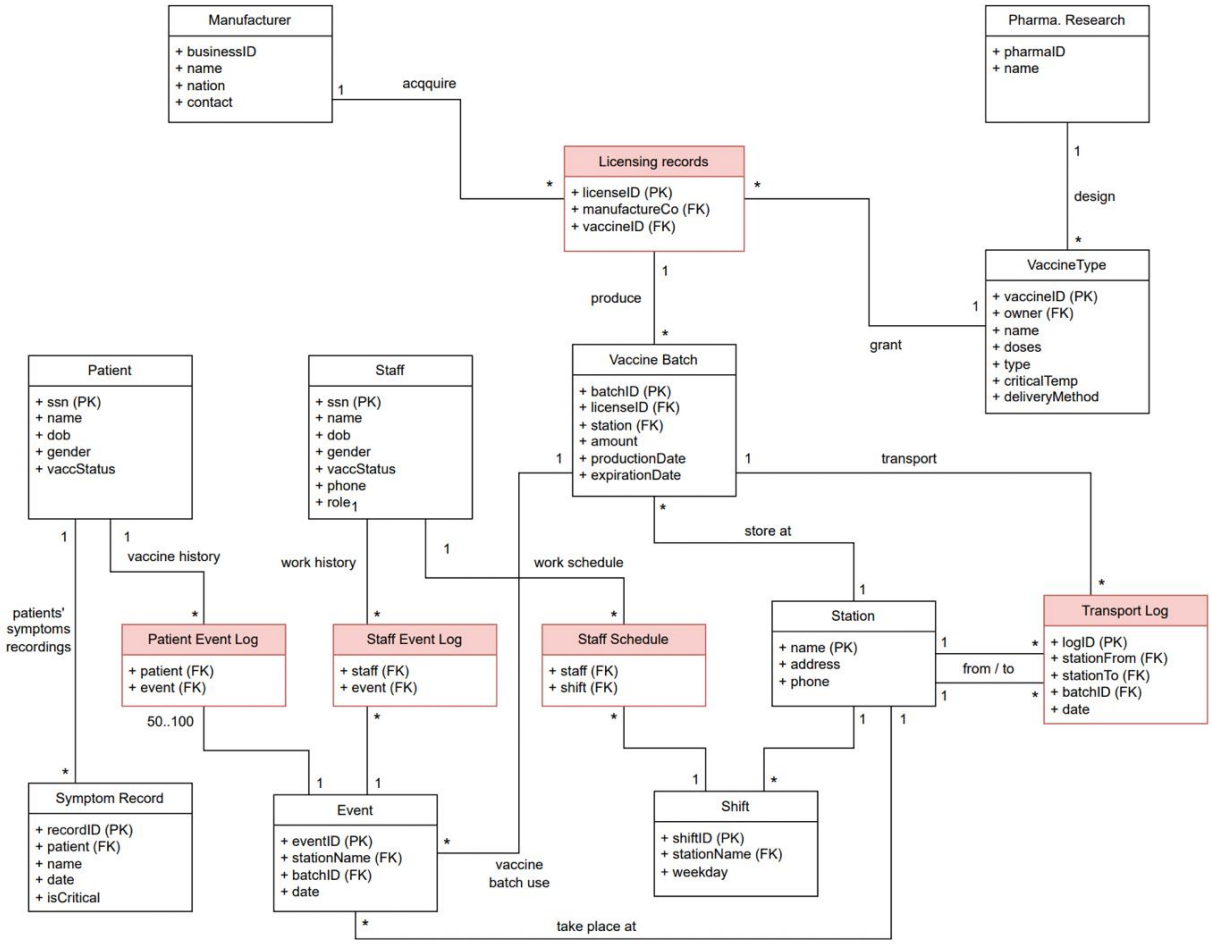
The queries used to visualize the data is located in the file ‘visualization.py’. After running the file, we should have the plots and figures as outputs.

II. UML and relational schema

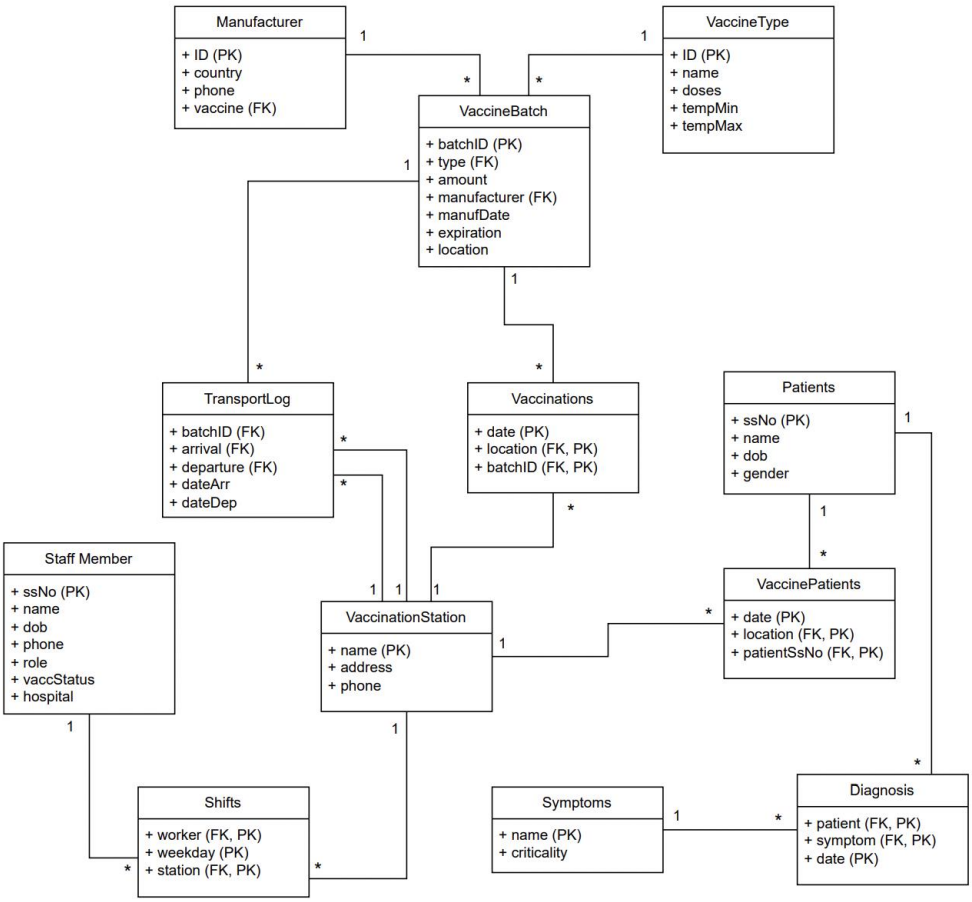
5. UML modeling and relational schema

a) UML modeling

- Old UML


Manufacturer
+ businessID
+ name
+ nation
+ contact
acqquire
Pharma. Research
+ pharmaID
+ name
design
Licensing records
+ licenseID (PK)
+ manufactureCo (FK)
+ vaccineID (FK)
produce
VaccineType
+ vaccineID (PK)
+ owner (FK)
+ name
+ doses
+ type
+ criticalTemp
+ deliveryMethod
grant
Patient
+ ssn (PK)
+ name
+ dob
+ gender
+ vaccStatus
Staff
+ ssn (PK)
+ name
+ dob
+ gender
+ vaccStatus
+ phone
+ role
Vaccine Batch
+ batchID (PK)
+ licenseID (FK)
+ station (FK)
+ amount
+ productionDate
+ expirationDate
transport
store at
vaccine history
work history
work schedule
patients' symptoms recordings
Patient Event Log
+ patient (FK)
+ event (FK)
Staff Event Log
+ staff (FK)
+ event (FK)
Staff Schedule
+ staff (FK)
+ shift (FK)
Station
+ name (PK)
+ address
+ phone
from / to
Transport Log
+ logID (PK)
+ stationFrom (FK)
+ stationTo (FK)
+ batchID (FK)
+ date
50..100
Symptom Record
+ recordID (PK)
+ patient (FK)
+ name
+ date
+ isCritical
Event
+ eventID (PK)
+ stationName (FK)
+ batchID (FK)
+ date
vaccine batch use
Shift
+ shiftID (PK)
+ stationName (FK)
+ weekday
take place at

- Updated UML



Modification notes:

After receiving feedback from part 1 and examining the sample data, we decided to make some changes to our original database design. We removed class Pharmaceutical Company since the provided data does not contain information regarding this class. Consequently, class LicensingRecords is also removed, since it is now redundant, and the information could be extracted from class Manufacturer and VaccineType. Association classes such as Patient Event Log, Staff Event Log, and Staff Schedule were also modified accordingly to fit the provided data.

b) Relational schema

Old relational schemas:

- **Manufacturers**(ManuID, name, country, phone)
- **PharmaceuticalComps**(ID, name)
- **Vaccines**(VaccineID, PharmaCom, name, type, criticalTemp, doses, deliveryMethod)
- **LicensingRecords**(LicenseID, Manufacturer, VaccineID)
- **VaccineBatch**(BatchID, LicenseID, station, amount, productionDate, expirationDate)
- **Station**(name, address, phone)
- **TransportLog**(logID, stationFrom, stationTo, batchID, date)
- **Patient**(ssNo, name, DOB, gender, vaccStatus)
- **Staffs**(ssNo, name, DOB, gender, vaccStatus, phone, role)
- **SymptomRecords**(recordID, patient, name, date, isCritical)
- **Shifts**(shiftID, stationName, weekday)
- **StaffEventLog**(staff, event)
- **PatientEventLog**(patient, event)
- **StaffSchedule**(staff, shift)
- **Event**(EventID, stationName, batchID, date)

Updated relational schemas:

- **Manufacturers**(ManuID, name, country, phone)
- **VaccineType**(VaccineID, name, doses, tempMin, tempMax)
- **VaccineBatch**(BatchID, type, manufacturer, amount, manufDate, expirationDate, location)
- **VaccineStation**(name, address, phone)
- **TransportLog**(batchID, arrival, departure, datearr, datedep)
- **Patients**(ssNo, name, DOB, gender)
- **Staffmembers**(ssNo, name, DOB, vaccStatus, phone, role, hospital)

- **Symptoms**(name, criticality)
- **Shifts**(worker, station, weekday)
- **Vaccinations**(date, location, batchID)
- **Vaccinepatients**(date, location, patientssno)
- **Diagnosis**(patient, symptom, date)

6. Design choices

a) Design justification

In the UML model, each rectangle represents a relation or an association relation. Here are some of the design choices and reasoning behind them:

- Class Selection:

- The UML model includes 10 primary classes and 5 association classes. Each class represents a unique entity in the database, allowing for each to exist independently and store specific information pertaining to that entity. For example, the Manufacturers and Pharmaceutical Companies classes allow for independent storage of information about manufacturers and pharmaceutical companies, even if they are not currently producing any vaccines.
- The distinction between the Vaccines and Vaccine Batch classes is crucial not only for avoiding redundancy, but also for tracking individual batches of a vaccine. This is important in cases such as recalls, or to track vaccine effectiveness and side effects.
- The Symptom Records class is essential to track each symptom individually. This can help in identifying trends, such as an increase in specific side effects.
- Although Patients and Staffs share some attributes, the roles and activities they participate in are fundamentally different, necessitating separate classes.

- Relationship Selection

- The database uses association lines to represent relationships between classes. A good example is the relationship between Pharmaceutical Companies and Vaccines, which is a one-to-many relationship, as one company can produce multiple vaccines, but a vaccine is typically produced by only one company.

- Association Selection:

- Association classes such as Licensing Records and Transport Log contribute significantly to the traceability and accountability of the system. Licensing Records capture the legal permissions given to manufacturers to produce specific vaccines, enabling the traceability of a vaccine batch back to its manufacturer. The Transport Log monitors the movement of a vaccine batch, which is crucial in scenarios such as a recall.
- The StaffEventLog and PatientEventLog association classes create a historical record of events associated with specific staff members or patients. This association is important for tracking activities over time and can provide insights into trends or patterns, for instance, identifying if a particular batch of vaccines has resulted in reported symptoms.
- The StaffSchedule class ties staff members to specific shifts, helping manage staff allocation and workload. This could be vital in understanding patterns in vaccine administration and managing resources effectively.
- The Event class and its associations enable the tracking of specific events at various stations, such as the arrival of a vaccine batch. This can support inventory management, resource allocation, and trend identification.

- Cardinality:

- The database uses multiplicity notation to represent the cardinality of relationships between classes. This specifies the quantity relationship between entities. For instance, one Vaccine Batch can be associated with multiple Transport Logs, but each Transport Log entry is associated with only one Vaccine Batch.

- Normalization:

- The design adheres to the principles of normalization, with each attribute in each class dependent on the class's primary key. This eliminates redundancy and dependency in the database, indicating that the design is likely in at least 3rd Normal Form (3NF).

- BCNF Compliance:

- The database satisfies Boyce-Codd Normal Form (BCNF) as every determinant is a candidate key. For instance, in the Vaccines class, VaccineID is the only determinant and it's a candidate key, so it meets the BCNF condition.

b) Assumptions

The design of the database is based on several key assumptions:

- Every vaccine batch is linked to a specific manufacturer and vaccine type.
- Each staff member and patient have unique identification numbers to avoid ambiguity.
- Every vaccination event is tied to a specific batch of vaccines.
- There is a reliable and effective system in place to update the Transport Log and Event records regularly to ensure data accuracy.
- Patient symptoms are recorded and updated promptly for effective monitoring of vaccine side effects.
- Every vaccine requires storage within a specific temperature range, which is accounted for during transport and storage.

7. Functional dependencies, anomalies and BCNF

c) Functional dependencies

Manufacturers(ManuID, name, country, phone)

ManuID → name, country, phone, vaccine

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

VaccineID → name, doses, tempMin, tempMax

VaccineBatch(BatchID, type, manufacturer, amount, manufDate, expirationDate, location)

BatchID → type, manufacturer, amount, manufDate, expirationDate, location

VaccineStation(name, address, phone)

Name → address, phone

Phone → Name, Address

TransportLog(batchID, arrival, departure, datearr, datedep)
batchID, datearr, datedep → arrival, departure

Patients(ssNo, name, DOB, gender)

ssNo → name, DoB, gender

Staffmembers(ssNo, name, DOB, vaccStatus, phone, role, hospital)

ssNo → name, DoB, gender, phone, role, hospital

Symptoms(name, criticality)

name → criticality

Shifts(worker, station, weekday)

Worker, station → weekday

Worker , weekday → station

Vaccinations(date, location, batchID)
Date, location → batchID

Vaccinepatients(date, location, patientssno)

date, location → patientssno

Diagnosis(patient, symptom, date)

Patient, date → symptom

d) Anomalies and BCNF

There are no redundancies or anomalies in the database, and the database is in BCNF.

III. Performance analysis and improvements

1. Analysis

Looking at our relational schema, it's clear that certain tables like Patient, Staffs, and VaccineBatch may become significant in size over time. The performance of queries against these tables, especially under high loads or for complex query patterns, will be key to our performance analysis.

2. Data Visualization

Data visualization is a crucial aspect for the practical usage of our database. We could present the vaccine distribution progress (such as number of vaccines delivered and administered) in graphical form. Furthermore, graphs showing staff workload distribution and patient demographics could provide valuable insights for administrators. The database schema design facilitates these visualizations by providing clear relationships and a comprehensive picture of the information.

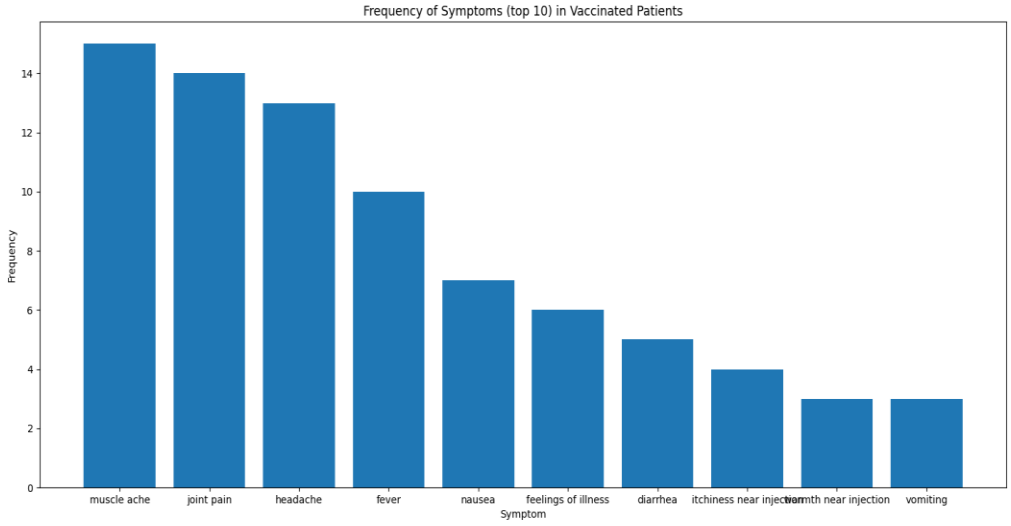


Figure 1: Frequency of Symptoms (top 10) in Vaccinated Patients

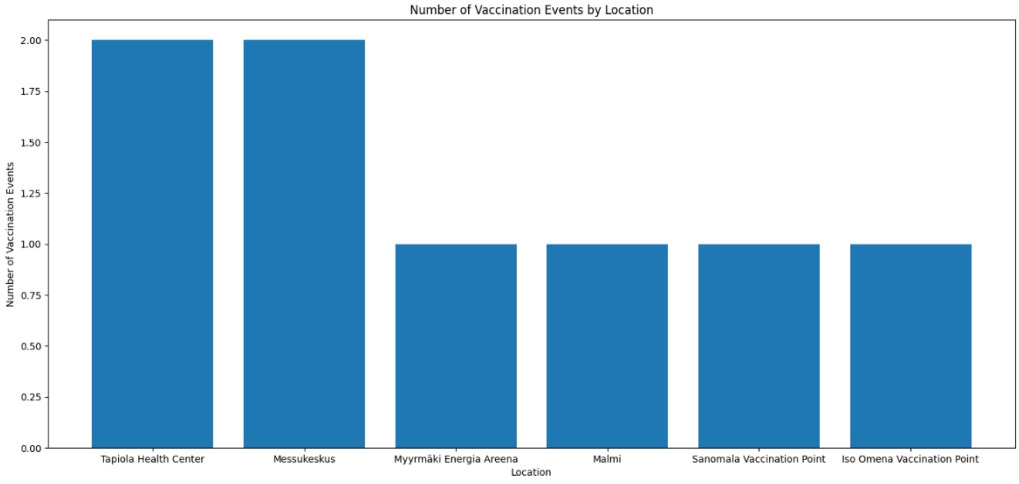


Figure 2: Number of Vaccination events by Location

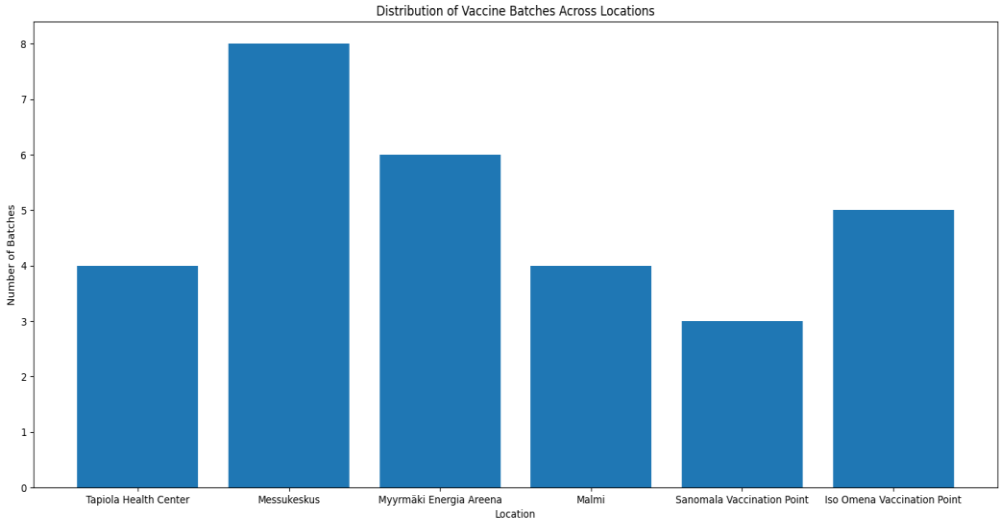


Figure 3: Distribution of Vaccine batches across Locations

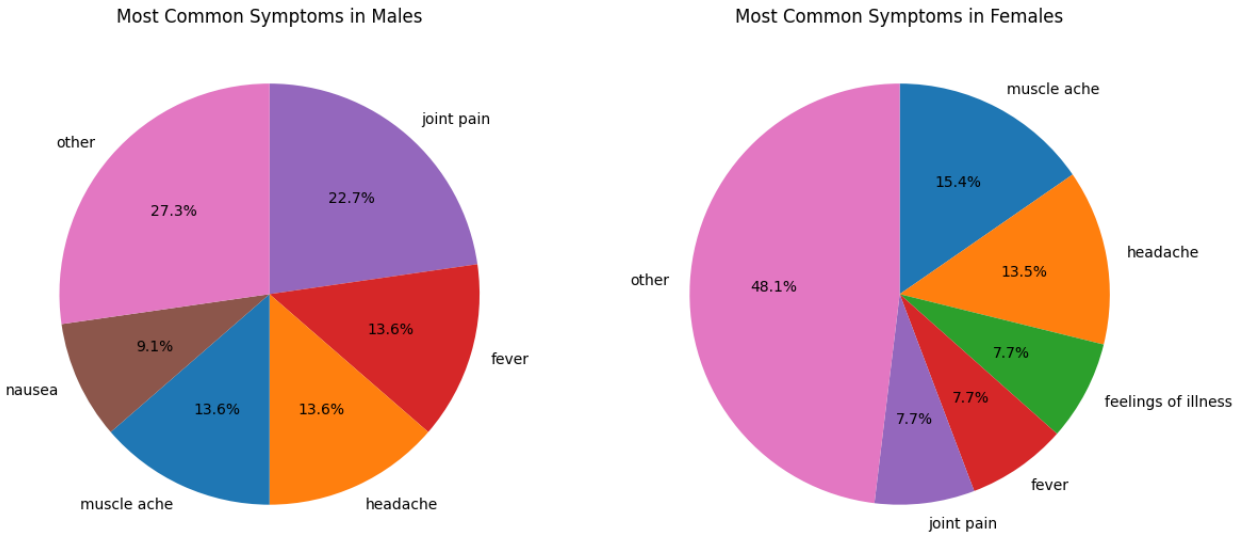


Figure 4: Most common Symptoms in Males and Females respectively

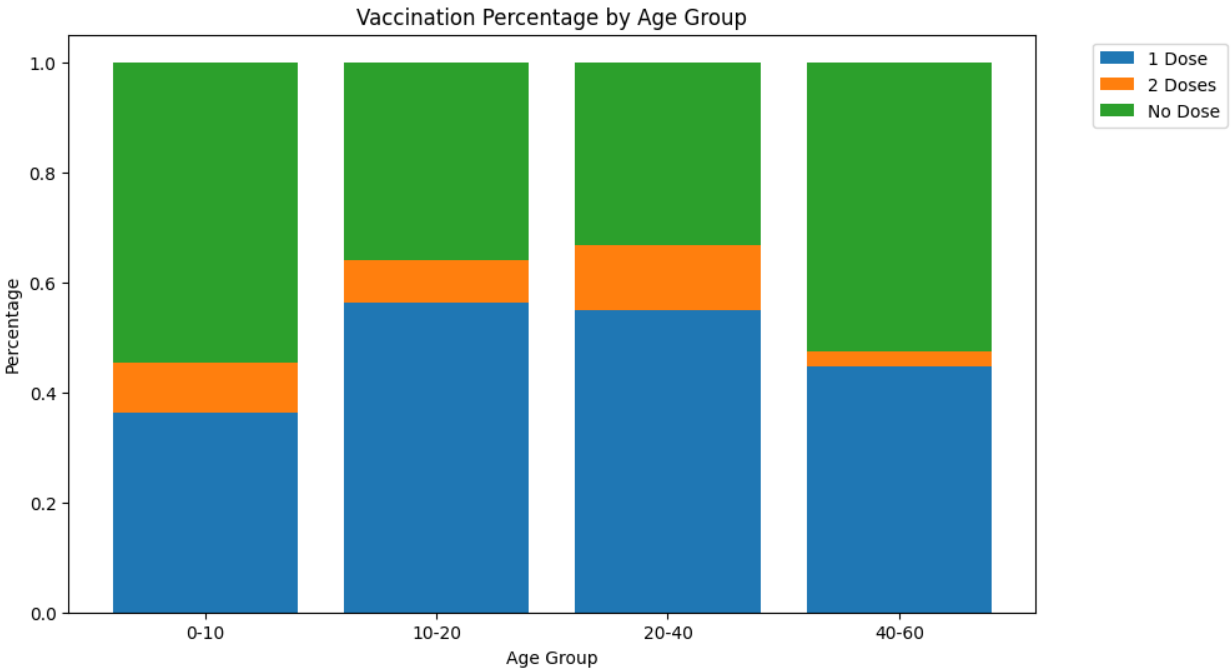


Figure 5: Vaccination Percentage by Age group

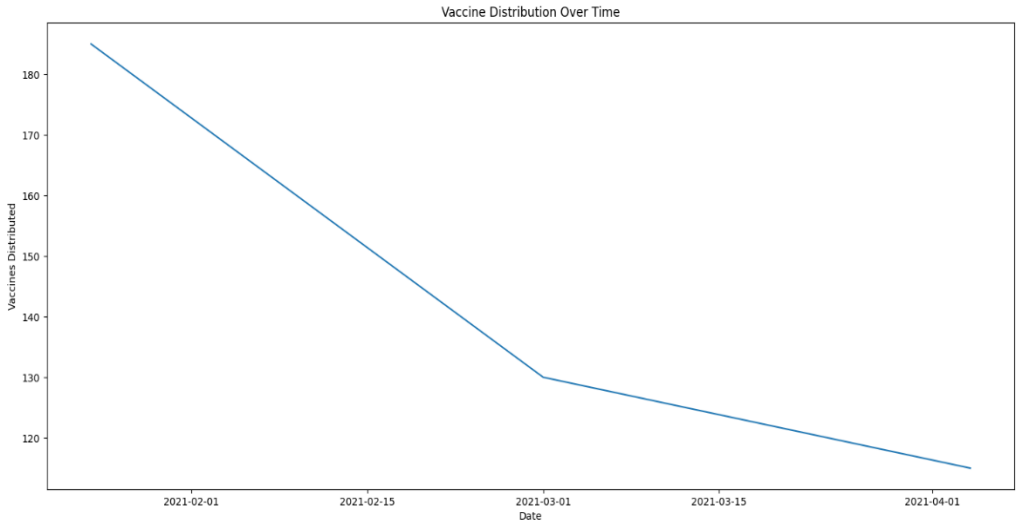


Figure 6: Vaccine Distribution Over Time

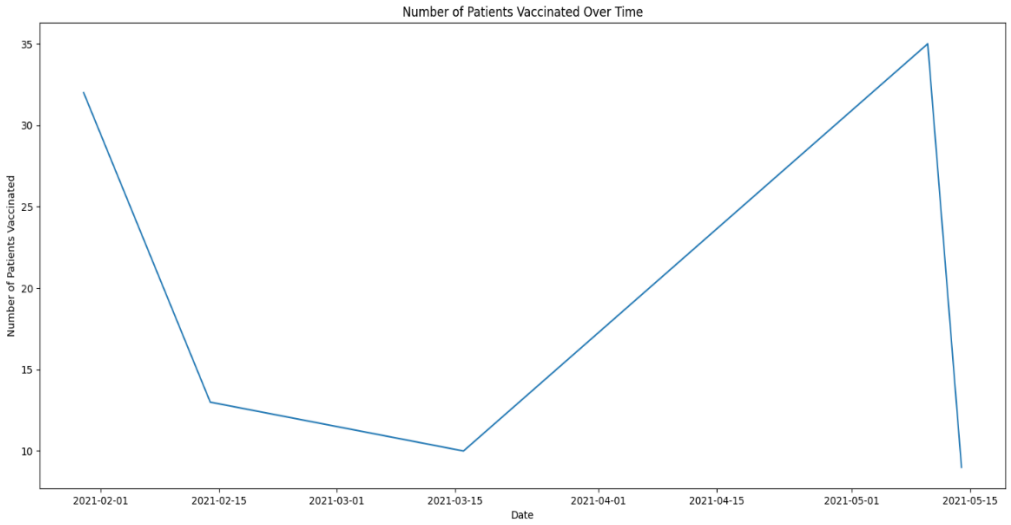


Figure 7: Number of Patients Vaccinated Over Time

3. Possible improvements

Based on the performance analysis, several areas of improvement have been identified:

- Indexing: Implementing indexes on frequently queried attributes such as ssNo in Patient and Staffs tables, and BatchID in the VaccineBatch table could dramatically improve query performance.
- Partitioning: Given the likely large size of tables like Patient, Staffs, and VaccineBatch, table partitioning could be beneficial. For instance, partitioning the Patient table by DOB or the VaccineBatch table by manufDate could increase efficiency.
- Denormalization: While our database is designed to be in BCNF for preventing data anomalies, there could be some performance benefits to selectively denormalizing certain parts of the database. For instance, including the manufacturer name directly in the VaccineBatch table could reduce the need for joining with the Manufacturers table in some queries.
- Caching: Implementing a caching strategy for frequently accessed data would also help in reducing retrieval time, improving overall performance.

IV. Group work

4. Division of tasks and roles

When we started the project, we had come to the agreement that there should be no fixed work allocation throughout the project. The motivation for this policy is to ensure that each of the member gets the opportunity to be hands-on with all facets of the project to maximize their learning experience.

As the result, the division of work in our group varies greatly by each phase of the project.

For part 1 of the project, Phuong and Quan were mainly responsible for the design of the database as well as the UML, while Trang was in charge of the presentation of the UML itself, and she also worked on the documentation with Hien.

For part 2 of the project, all four of us worked on the SQL queries, and Hien and Quan were mainly responsible for the Python script, while Trang was in charge of the documentation. For the presentation, Trang also designed the slides, and Phuong wrote the content.

For part 3 of the project, we all worked together on the queries, the Python script, and the documentation.

All in all, we would argue that our group division in this project were fair, and while it allowed each member to play to their strength, it also allowed us to venture and push our comfort zone.

5. Estimated schedule

Part	**Date**	**Task**
Part 1	11/5 – 14/5	Designing database and UML.
	15/5	Completing documentation.
Part 2	25/5 – 26/5	Creating and populating database.
	27/5 – 28/5	Writing SQL queries and testing.

	29/5	Completing documentation.
Part 3	9/6 – 11/6	Answering task questions, i.e. carrying out data analysis, and testing.
	12/6	Completing documentation.

6. Challenges and difficulties

As the project was quite a challenge, we did encounter many problems working as a group. As we are a group with a variety of different expertise, initially for the first phase of the project, we agreed to listen to the person with the most experience and knowledge on the problem at hand. For example, because Phuong had more experience working with UMLs, we went with his recommendation anytime we had an issue with the design of our UML. However, as we progressed with the project, we came to realize the importance of letting all voices be heard, so anytime a problem came up, we would gather as a group to voice our opinions and made our cases. It was really helpful for us to learn how to stand our grounds and made a case for what we think is right.

V. Conclusion

In conclusion, the development of the Corona vaccine distribution and treatment database for Finland has been a challenging but rewarding project. The database serves the purpose of tracking and managing the distribution of vaccines, vaccination events, patient data, and symptoms. It offers several advantages in terms of data organization, tracking capabilities, and potential use cases. However, there are also some limitations and areas for improvement.

Advantages of the database include its comprehensive coverage of relevant entities and relationships, adherence to normalization principles, and compliance with BCNF. The UML modeling and relational schema provide a clear representation of the system, enabling efficient data management and analysis. The use cases demonstrate the practical

value of the database, ranging from vaccine distribution tracking to symptom analysis and vaccination status reporting.

One notable disadvantage of the database design is the potential for scalability issues with certain tables, such as Patient, Staffs, and VaccineBatch, as they may accumulate large amounts of data over time. Performance analysis has identified the need for indexing, partitioning, denormalization, and caching strategies to optimize query execution and retrieval time. These improvements could be implemented in the future to enhance the database's efficiency and scalability.

In terms of the project as a whole, it has provided valuable insights into database design, UML modeling, and the challenges of collaborative group work. The division of tasks and roles within the group allowed for individual growth and learning experiences. The project required us to navigate through complex decision-making processes, accommodate different perspectives, and adapt our approach based on feedback and data analysis.

In the future, the database could be further enhanced by incorporating additional features, such as real-time data updates from vaccination centers and integration with external systems for seamless information exchange. Continuous monitoring of performance and user feedback would help identify areas that require refinement and ensure the database remains effective in supporting vaccine distribution and treatment in Finland.

Overall, the project has successfully achieved its objective of developing a database to model and manage the distribution of Corona vaccines in Finland. It has demonstrated the importance of effective database design and utilization in public health management. Despite the challenges encountered along the way, the project has provided valuable learning experiences and insights into the practical application of database systems in real-world scenarios.

VI. References

https://version.aalto.fi/gitlab/databases_projects/summer-2023/project-vaccine-distribution