Modeling the Distribution of Vaccines

Table of Contents

1. Introduction 3

2. Design and Use-cases 3

2.1 Purpose of the database and use-cases

2.2 UML diagram and relational schema

2.3 Assumptions and design choices

3. Performance Analysis and Improvement 6

4. Working as a Team 7

5. Conclusion 9

1. Introduction

This Project involved the creation of a Database to manage the Distribution of Vaccines. Initially, we created a UML diagram based on the information that was provided to us and the specifications involving the same. Upon review and updation of the UML diagram, we built our database based on the data provided. The queries and data visualization were constructed as per needs.

2. Design and Use-cases

2.1. Purpose of the Database and Use-cases

The primary purpose of the database is to manage the distribution of vaccine and tracking of a vaccine from the manufacturer to the transportation to the storage in hospitals to vaccinations it was used to the patients it was used on to the symptoms it might have caused. The database acts as an information tracking dashboard for vaccines and helps streamline the distribution process of vaccines.

Vaccines are an important element in healthcare systems and a database to keep track of them will help to acquire vital information regarding them such as in case a vaccine exhibits a critical symptom, it is easier to track the source, patients, vaccination stations and other info regarding the vaccine with the help of a database. Our project therefore provides an important service in the collection and storing of vaccine distribution information in a structured manner between stakeholders.

2.2. UML diagram and relational schema

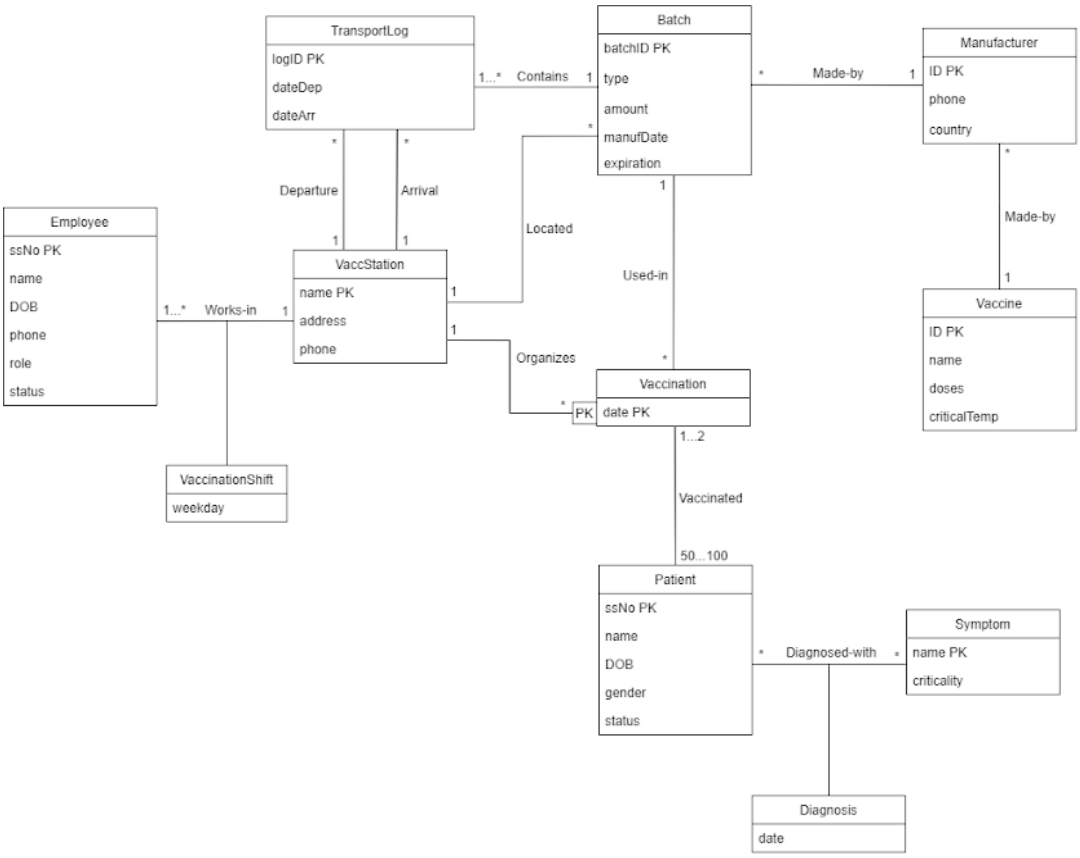

Figure 1. Revised UML model of the database

Manufacturer(ID, phone, vaccine, country); ID → phone, vaccine, country; phone → ID, vaccine, country
Vaccine(ID, name, doses, criticalTemp); ID → name, doses, criticalTemp
Batch(batchID, type, manufacturer, amount, manufDate, expiration, station); batchID → type, manufacturer, amount, manufDate, expiration, station
VaccStation(name, address, phone); name → address, phone; phone → name, address; address → name, phone
TransportLog(logID, dateDep, dateArr, departure, arrival, batchID); logID → dateDep, dateArr, departure, arrival, batchID

Employee(ssNo, name, DOB, phone, vaccStation, role, status); ssNo → name, DOB, phone, vaccStation, role, status; phone → ssNo, name, DOB, vaccStation, role, status
VaccinationShift(worker, weekday, station); worker, weekday → station
Vaccination(station, date, batchID); station, date → batchID
Patient(ssNo, name, DOB, gender); ssNo → name, DOB, gender
Vaccinated(date, ssNo, station); date, ssNo → station
Symptom(name, criticality); name → criticality
Diagnosis(ssNo, date, symptom)

2.3. Assumptions and design choices

The UML model and relations were created based on the description in Part 1 and some additional assumptions. Each manufacturer, for example, was assumed to have only one country of origin and one vaccine that they manufacture. It was also assumed that each employee works only in one vaccination station but can work several days a week so that worker and weekday determine the vaccination station in VaccinationShift. The phone numbers in Manufacturer, VaccStation, and Employee were also assumed to be unique and thus could also be used as a key. The address in VaccStation was also taken to be unique. In Vaccinated, each patient was assumed to be vaccinated at most once a day so that the date and social security number together determine the vaccination station. For Symptom, it was also assumed that each symptom is either critical or noncritical so that the name of the symptom determines criticality.

Some more general design choices were also made when planning the database. CriticalTemp was expressed as a range as the data contained minimum and maximum temperatures for each vaccine. A log ID was added to TransportLog to differentiate between transports of the same batch between the same vaccination stations on the same day, although this seems somewhat unlikely. When creating the tables, appropriate constraints such as not null constraint and foreign key constraint whenever the attribute referenced another attribute from another table were also added. For Employee and Symptom, the status and criticality were constrained to be either 0 or 1 and for

Patient, the gender was checked to be M or F. Weekday in VaccinationShift was also checked to be appropriate. When populating the tables, erroneous data points were identified and removed because deleting data in this case is better than modifying it to fit the tables considering the use of the database.

3. Performance Analysis and Improvements

Based on the feedback for Part 1 of the project, the UML model as well as the schemas of the relations were modified slightly for Parts 2 and 3. These changes removed some redundancies as, for example, the relations DepartureHospital and ArrivalHospital were removed because the information was already contained in the TransportLog relation, and the original relation Symptom was broken into two smaller relations, Symptom and Diagnosis, using the assumption that a symptom is either critical or noncritical. Some other relations containing information about countries of origin and work shifts were also removed based on the data. Overall, these changes certainly improved usability and likely also increased the efficiency of the database by facilitating the execution of queries. In addition, some relation and attribute names were changed to better correspond to the column names in the data file so that the code for creating the database would be cleaner.

The database could still be improved and adjusted to different kind of data. It could for example be extended to include multiple countries of origin for each manufacturer and several different vaccines associated with one manufacturer instead of strictly one as manufacturers often manufacture more than one type of vaccine in real life. Each employee would also likely work in more than one vaccination station. Describing the data about the current location of the batches could perhaps also have been done little differently. Rather than adding an attribute station to relation Batch as the information was contained in the

provided data, an updatable view containing the batch ID and location could for example have been created.

4. Working as a Team

For Part 1, we had a group discussion regarding the choosing of the topic and decided to pick Vaccine Distribution because we felt this solved a much relevant and important issue in hand. We split the designing of the UML in 4 parts based respectively on the different slides in the Part 1 deliverable requirements. Upon which we had a discussion and modified necessary portions of the UML.

For Part 2, upon feedback regarding our UML diagram and the requirements of Part 2, we split ourselves into 2 groups. The first group worked on rectifying the UML diagram and implementing the database based on the data provided along with cleaning the data. Upon completion of this, the second group implemented the 7 queries. We did a cross verification of each other's work before submission.

For Part 3, since it involved 10 queries, we split it equally and worked on it individually for a few days. Upon which we cross verified each other's work and worked together on the final two queries before submission.

As for our project scheduling regarding our tasks, our group was efficient and quick in our work. Hence the days taken for each individual and group tasks were low. We also always completed the project in advance to the deadline. The scheduling is as follows:

Part 1:
25 April: Group Discussion and splitting of tasks
26-30 April: Working on parts of UML individually
2 April: Group Discussion to modify and finalize the UML Diagram

Part 2:
9 May: Group Discussion and splitting of tasks
10-13 May: Group A worked on rectifying UML Diagram and Implementing the Database
13-16 May: Group B worked on the queries
19 May: Group Discussion and cross-verification of each other's work.

Part 3:
27 May: Group Discussion and splitting of tasks
27 May - 2 June: Individual work on the queries
4 June: Group Discussion to cross-verify the queries and solve the final 2 queries together.

Project Deliverable Package:
8 June: Group Discussion and splitting of tasks
9 - 12 June: Individual reporting based on the given headings for the document
13 June: Group Discussion to finalize the document

We always broke down the tasks and split them equally amongst our group members. We set a time period for us to work individually on them and during that time period, if any of us were stuck with a problem or had queries regarding it, we would post it in our group channel upon which always one of us helps the member out. At the end of our individual work, we always had long group discussions, where we verified each other's work and modified parts if needed. This way, each individual member in our team had an equal share of work in every part of the project as well as our group discussion made the project a whole with our teamwork and verifications of each other's work.

5. Conclusions

Our database has an easy to understand and implementable structure regarding the vaccine distribution. It handles and stores the given data cleanly with proper and logical relationships between tables. It primarily needs improvement in data cleaning as the current code cannot handle invalid inputs such as of the Date type.

The current database exists in PostgreSQL which is not widely used by companies and does not provide easy access across networks and interconnectivity with varied nodes and coverage across networks. The future plans would be to utilize a secure and efficient architecture on Azure or Amazon Web Services (AWS) and build the database there. We can fetch the database from Data Lakes and the cloud functionality that would be built makes it easier to scale our database and utilize it for various applications. This cloud-based approach broadens the horizons on the possible implementation of our database in various services and sectors.

Overall, our first build of a database handles the necessary requirements and stores the essential data properly and only lacks in handling improper data. We feel happy with the work we have accomplished and are even considering experimenting this further in the cloud and possibly create a small-scale application out of it.