Designing a Volunteer Matching System (VMS) with the Finnish Red Cross (FRC)

# Introduction

In this work we designed and implement relational database that models a real-world scenarios of volunteer matching system for Finnish Red Cross and defined the usage to support the matching. This project was conducted in a team of two students.

# Summary part 1

We drew a UML diagram for the Volunteer Matching System VMS database based on the information defined and using the notations taught in the course. Next, we converted the UML diagram to the relational data model, presented the schemas of the relations, and underlined the attributes which form the key for each relation, all of which is presented in The Project1 deliverable file. Here we show only the UML diagram.

# UML diagram

## Relational data model, present the schemas of the relations, and underline the attributes which form the key for each relation.

**Relations of UML classes:**

Volunteer(voID, name, birthday, email, address, travelReadinessTime)

Application(appID, reqID, voID, lastModifiedTS, valid)

Request(reqID, benID, interestArea, volNumNeeded, priority, startDate, endDate, registerbyDate)

Skill(skillName, description)

InterestArea(interestArea)

Beneficiary (benID, name, address)

Location(cityID, name, geolocation)

**Relations for associations:**

canDo(skillName,volID)

operatingRange(voID, cityID)

whereNeeded(cityID, reqID)

canChoose(voID, interestArea)

skillsNeeded(skillName, reqID, minimumNeedofppl, ImportanceValue)

## Provide answers to the following questions:

2. Are there any forms of redundancy or other anomalies in the database structure?

- after merging some of association relations, we managed to get rid of redundancies (otherwise implied by the specification).

3. Is the database in the Boyce-Codd Normal Form? If it is not, use the decomposition algorithm (submit both original and decomposed version)

Yes, it is in BCNF form.

Volunteer(voID, name, birthday, email, address, travelReadinessTime)

* In addition to the volID, which is a PK, email could also be a candidate. Since, email addresses are unique. If we assume that volunteer can sign up in the system with the same address only once, email -> voID. And since voID is the PK, this FD contains all attributes of the class.

Name is not neccessarily a unique attribute, nor is birthday, address or travelReadinessTime

* What about the pairs of attributes? No obvious candidates either.

Name, birthday – could happen so that two people with the same names were born on the same day, so no candidates.

Name and email address is possible. The closure would again contain all the attributes.

Name, address – could be so that mother and daughter live at the same address and have same names, so they dont define unique tuples.

Name, birthday, address – it is extremely unlikely that two different people with the same values for these attributes would exists, but it is possible – partners who happen to have the same name and birthday, and they live together. So to be on the save side, not considerable.

Application(appID, reqID, voID, lastModifiedTS, valid)

* No single attribute (except for appID) makes a functional dependency with any other attribute in this relation. At least it’s safer to consider it this way.
* What about the pairs of attributes? No obvious candidates either.

e.g., – meaning it’s guaranteed that a volunteer cannot apply for the same Request more than once! Is it true? Probably not, as the same volunteer could probably apply to the same request more than once.

* What about the triplets of attributes? One option seems likely. And since appID is a primary key, the closure of this FD contains all attributes of the class.

Request(reqID, benID, interestArea, volNumNeeded, priority, startDate, endDate, registerbyDate)

* No single attribute (except for reqID) makes a functional dependency with any other attribute in this relation.

e.g. one benID can not determine any other specific attributes of this class because one beneficiary can make multiple requests or have multiple interest areas, etc.

* Pairs of attributes:

One combination of two attributes may seem to be possible:

benID, interestArea→reqID

This FD would mean that beneficiaries would create a request in an area for only one time period (a specific combo of [startDate, endDate]). In real world, they might actually need same type of requests for different dates.

Other pairs don’t seem likely. E.g. one combination of interestArea and volumeNeeded can be inquired by different beneficiaries, or by the same beneficiary for multiple dates.

* Triples of attributes:

Similar to pairs; One combination of triplet may seem to be possible:

benID, interestArea, startDate/endDate →reqID

but that would mean that beneficiaries could create a request for specific are only for one time In real world, they might actually need similar requests for different dates.

* Combination of four attributes:

One combination seems possible:

benID, interestArea, startDate/endDate →reqID

This combination of attributes can uniquely define a request. We can assume that there wouldn’t be the same request done by one beneficiary in a specific area and at a unique time-frame for different volNumNeeded, priority and registerDate. That doesn’t sound realistic.

Since reqID is a primary key, closure of this combination then contains all the attributes of the class. Thus, it is in BCNF form.

SkillsNeeded(skillName, reqID, minimumNeedofppl, ImportanceValue)

* No single attribute would be enough to make a functional dependency with other attributes in this relation.
* Pairs of attributes:

Besides the combination of PKs, no other combination are likely to be possible.

SkillName with minimumNeedofppl or with ImportanceValue can not reliably determine requestID, since the same number of people with the same skills can be inquired by different requests.

ReqID with minimumNeedofppl or with ImportanceValue can not reliably determine skillName, because the same request with the same number of people and of same importance might need a different skill.

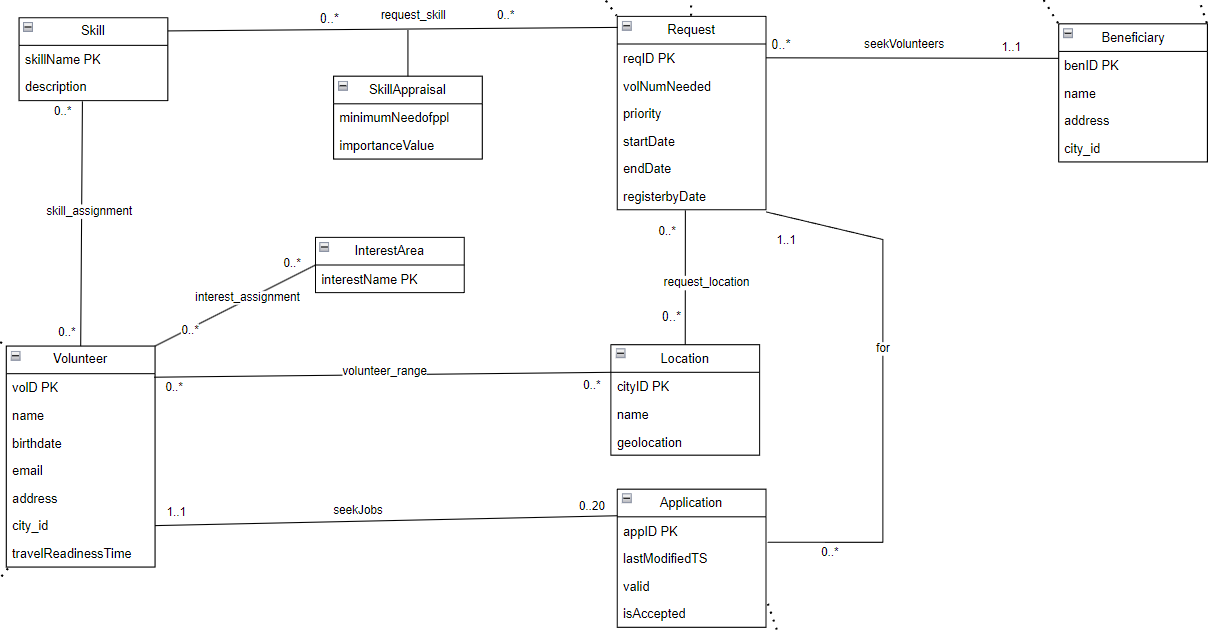
* Triplets of attributes:

Same as for pairs, unless the triplet would contain both PKs, the combination would not determine a unique tuple of the class.

# Part 2

In this part we updated the UML diagram based on the synthetic data and created a database, which we populated wit the data. Furthermore, we created the demanded queries, views, transactions, triggers, and visualized some trends.

### Our modified (updated) UML diagram:



## Part 2 assignment summary:

Part 2 deliverables are submitted in separate files. All individual task solutions are accompanied by comments explaining what we attempted to do:

Python file used to create the database – submitted as **create\_DB.ipynb**.

Tasks a) BASIC queries [8 + 4 free choices] - submitted as an sql file **all\_queries.sql**.

Tasks b) ADVANCED views, triggers and function – submitted as an sql file **views\_triggers\_funcs.sql**.

Tasks c) Transactions – submitted as sql file **transactions.sql**.

Tasks d) Analysis [2 + 1 free choice] - submitted as **data\_analysis\_and\_visualization.ipynb**.

* Scoring system script is submitted in a separate file **scoring\_system.ipynb**.

# Reflections

Working on this project in a team of two presented some challenges in terms of workload. This occasionally made it difficult to cross-check each other's work as thoroughly as we would have liked. However, it also made it significantly easier to reach agreements during brainstorming sessions and when planning the timeline and meetings. Throughout the project, we divided tasks equally, communicated openly, and both of us remained diligent, doing our best to perform well.

Overall, we understood how relational databases are crucial for effective data management.

# Conclusions

In this project we made a UML diagram from the project description, converted it into a relational data model, and created and populated the database with tables containing real-world data.

Furthermore, this project provided extensive hands-on practice with Structured Query Language (SQL). We created queries for data viewing, triggers, functions, and transactions, which we leveraged in the analysis part of the project. This practical experience was invaluable in reinforcing our understanding of database management and SQL.