Aalto-yliopisto
Aalto-universitetet
Aalto University

# Final deliverables: Finnish Red Cross

**Group 10**

---

**Huy To**

**Hoang Tran**

**Tinh Ta**

**Linh La**

**Databases for Data Science**
**Spring 2024 V**

**Table of contents**

# 1 Introduction

## 1.1 Project introduction

Our group project for the course “Databases for Data Science” focuses on the design and implementation of a relational database system to address real-world needs, following the principles of database normalisation. This project aims to integrate the use of Unified Modelling Language (UML) [1] for designing relational schemas and SQL for querying, enabling efficient data capture and extraction essential for modelling and analysing critical aspects of a given problem domain.

In spring 2024, we have an opportunity to collaborate with the Finnish Red Cross (FRC) [2] on a Volunteer Matching System (VMS). This system is intended to match the volunteer [3] capacity of the Red Cross with local multidimensional vulnerabilities and crises, which we called Request [4] in our project. The goal is to create a robust database that effectively supports this matching process, aligning with the FRC’s humanitarian mission and operational principles.

## 1.2 Project objectives

Relational Database Design: We will design a relational database that models the supply side of the volunteer matching system, reflecting the capacity and availability of Red Cross volunteers.

SQL Implementation: Using synthetic data provided, we will implement SQL queries to extract and manipulate data, addressing 22 specific questions. Our responses will be evaluated based on correctness, complexity, and relevance.

By the end of this project, we aim to demonstrate a comprehensive understanding of relational database design and SQL implementation, preparing us for future projects in data science. The collaboration with the Finnish Red Cross not only enriches our learning experience but also allows us to contribute to a meaningful cause.

## 1.3 Project parts

### 1.3.1 Part 1: Supply side modelling

We will create a relational data model representing the supply of volunteers. This model will include attributes and relationships relevant to the volunteer capacity, ensuring alignment with the FRC’s operational needs.

### 1.3.2 Part 2: SQL Implementation

Using the provided synthetic data, we will develop and execute SQL queries to retrieve and analyse information. This part tests our ability to apply SQL in practical scenarios, enhancing our skills in data manipulation and analysis.

### 1.3.3 Additional part: Dynamic matching

We developed an algorithm to match the volunteer to the request, assuming a request can be classified as Oversupply request and Undersupply request

# 2 Summary of part 1 and part 2

## 2.1 Part 1

### 2.1.1 UML

### 2.1.1.1 Original UML

#### 2.1.1.1.1 UML model

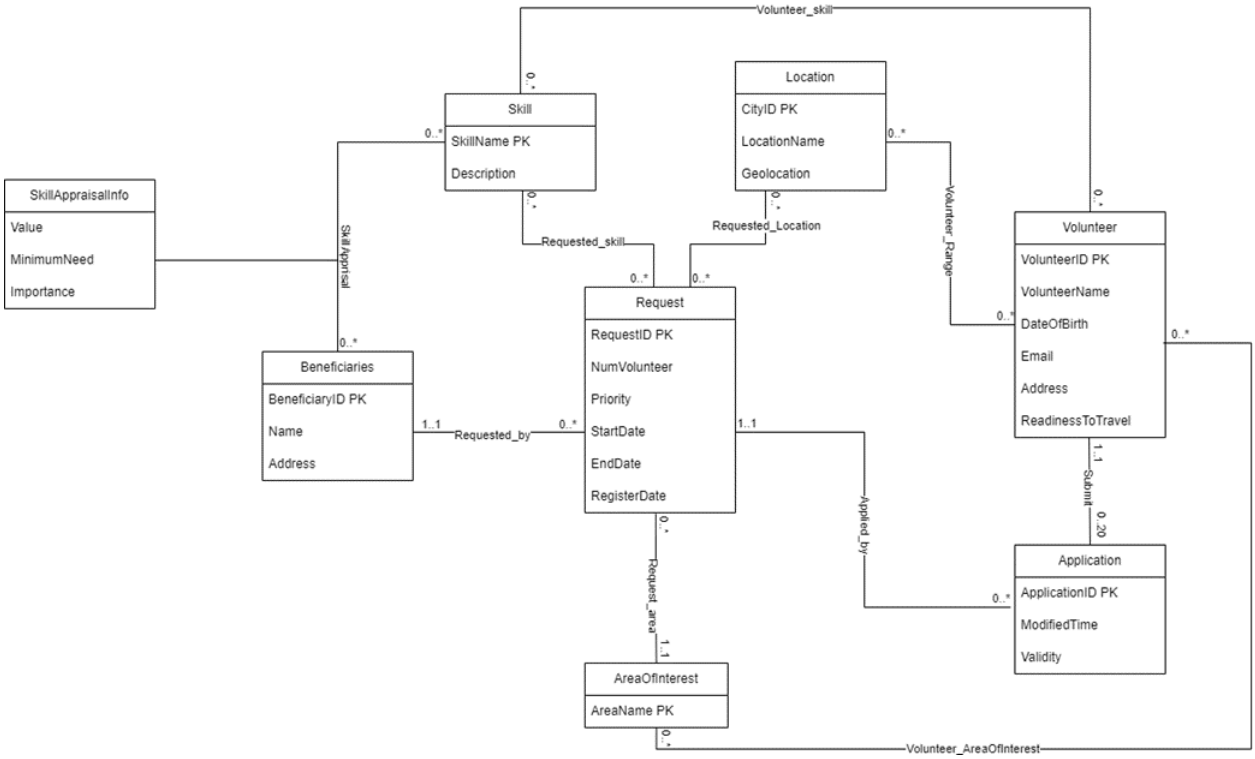



Figure 1 Original UML

In this section, we will present how each class and association fulfil the requirements of the project.

| Requirement | UML class representation |
|---|---|
| Beneficiaries have unique IDs, names, and addresses | Main class: Beneficiaries<br>Attributes: BenficiaryID (PK), Name, Address |
| Any Beneficiary can make as many volunteering requests as they need | Association: Requested_by between Beneficiaries and Request<br>Type of association: one-to-many |
| Requests should include a unique ID, the ID of the beneficiary it was sent by, the number of volunteers needed, a priority value to indicate how urgent the request is, a start date, and end date, and a register by date | Main class: Request<br>Attributes: RequestID (PK), NumVolunteer, Priority, StartDate, EndDate, RegisterDate |
| Requests also have the area of interest where the request lies | Association: Request_area between Request and AreaOfInterest<br>Type of association: one-to-many |

| | |
|---|---|
| Each of these skills have a unique name and a description | Main class: Skill<br>Attributes: SkillName (PK), Description |
| Each request skill has a request ID and a skill name | Asosciation: Requested_skill<br>Connect: between Request and Skill<br>Type of association: many-to-many |
| Each request location has a request ID and a city ID | Association: Request_Location<br>Connect: between Request and Location<br>Type of association: many-to-many |
| Volunteers have unique ID, name, birthday, email, address, readiness to travel (minutes) | Main class: Volunteer<br>Attribute: VolunteerID (PK), VolunteerName, DateOfBirth, Email, Address, ReadinessTo-Travel) |
| A volunteer can choose any combination of areas of interests | Association: Volunteer_AreaOfInterst<br>Connect: between Volunteer and Ar-eaOfInterst<br>Type of association: many-to-many |
| A volunteer can choose any combination of skills | Association: Volunteer_Skill<br>Connect: between Volunteer and Skill<br>Type of association: many-to-many |
| Beneficiaries can appraise skills by assigning skills a value, a minimum need, and a value to indicate importance | Association class: SkillAppraisalInfo<br>Attribute: Value, MinimumNeed, Importance<br>Connect between Beneficiaries and Skill. |
| Applications should include a unique ID, the ID of the request it was made to, the time it was modified, and they should indicate whether they are valid or not | Main class: Application<br>Attribute: ApplicationID, ModifiedTime, Va-lidity |

| | |
|---|---|
| Applications should include the ID of the request it was made to | Association class: Applied_by<br>Connect between Application and Request<br>Type of association: one-to-many |
| Applications the ID of the request it was made to, the ID of the volunteer it was sent by | Association class: Summit<br>Connect between Application and Volunteer<br>Type of association: one-to-many |
| Volunteers can sign up to the system, browse through the volunteering requests, and send up to 20 applications where they apply to the requests | Assocation: Submit<br>Connect: between Volunteer and Application<br>Type of association: one-to-many |
| Each city has an ID, a name, and a geolocation | Main class: Location<br>Attribute: CityID (PK), Name, Geolocation) |
| Volunteers operate in ranges. Each volunteer range has a volunteer ID and a city ID | Assocation: Volunteer Range<br>Connect: between Volunteer and Location<br>Type of association: many-to-many |

Table 1 Explanation on how UML fulfils the requirements

#### 2.1.1.1.2 Relational tables and BCNF

Following the UML, we devised the corresponding relational tables:

- Request (RequestID, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate)
  - ➔ {RequestID -> BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate}
- Requested_location (RequestID, CityID)
  - ➔ {RequestID, CityID -> RequestID, CityID}
- Requested_Skill (RequestID, SkillName)
  - ➔ {RequestID, SkillName -> RequestID, SkillName}
- Skill (SkillName, Description)
  - ➔ {SkillName -> Description}
- Location (CityID, LocationName, Geolocation)
  - ➔ {CityID -> LocationName, Geolocation}

- AreaOfInterest (AreaName)
    - ➔ {AreaName -> AreaName}
- Beneficiaries (BeneficiaryID, Name, Address)
    - ➔ {BeneficiaryID -> Name, Address}
- SkillAppraisal (SkillName, BeneficiaryID, Value, MinimumNeed, Importance)
    - ➔ {SkillName, BeneficiaryID -> Value, MinimumNeed, Importance}
- Volunteer (VolunteerID, VolunteerName, DateOfBirth, Email, Address, ReadinessTo-Travel)
    - ➔ {VolunteerID -> VolunteerName, DateOfBirth, Email, Address, ReadinessTo-Travel}
- Volunteer_Range (VolunteerID, CityID)
    - ➔ {VolunteerID, CityID -> VolunteerID, CityID}
- Volunteer_AreaOfInterest (VolunteerID, AreaName)
    - ➔ {VolunteerID, AreaName -> VolunteerID, AreaName}
- Volunteer_Skill (VolunteerID, SkillName)
    - ➔ {VolunteerID, SkillName -> VolunteerID, SkillName}
- Application (ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity)
    - ➔ {ApplicationID -> ApplicationID, RequestID, VolunteerID, ModifiedTime, Validity}

Based on the relational model, we have the following relations that are in BCNF:

- Request (<u>RequestID</u>, BeneficiaryID, NumVolunteer, Priority, AreaName, StartDate, EndDate, RegisterDate)
- Skill (<u>SkillName</u>, Description)
- Location (<u>CityID</u>, LocationName, Geolocation)
- AreaOfInterest (<u>AreaName</u>)
- Beneficiaries (<u>BeneficiaryID</u>, Name, Address)
- SkillAppraisal (<u>SkillName</u>, <u>BeneficiaryID</u>, Value, MinimumNeed, Importance)
- Volunteer (<u>VolunteerID</u>, VolunteerName, DateOfBirth, Email, Address, ReadinessTo-Travel)
- Application (<u>ApplicationID</u>, RequestID, VolunteerID, ModifiedTime, Validity)

#### 2.1.1.2 Revised UML

##### 2.1.1.2.1 UML model

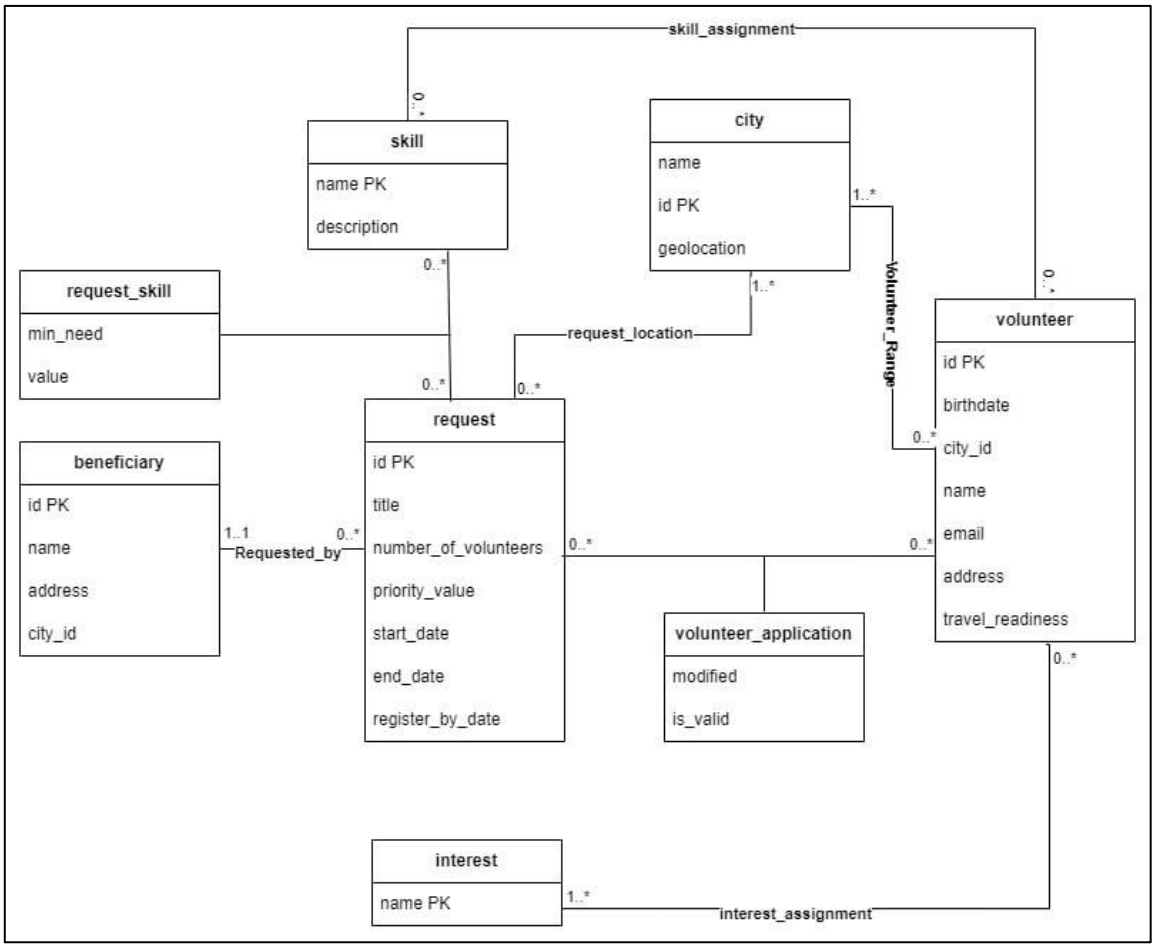



Figure 2 Revised UML

Based on the data provided, we made some modifications to our UML created in Part 1 of the project:

- The association class “request_skill” is connected to the two associations “skill” and “request.” Initially, we connected this association class to “skill” and “beneficiary”. It is realized the needed skills change for each request, so connecting the beneficiary and skill table does not make sense since a combination of beneficiary and a paricular request will be repeated, which is not possible given the primary key constraint.
- We changed attribute names in each association to align with the column names in the provided Excel data file.

Other than these two changes, no difference between our UML and the data.

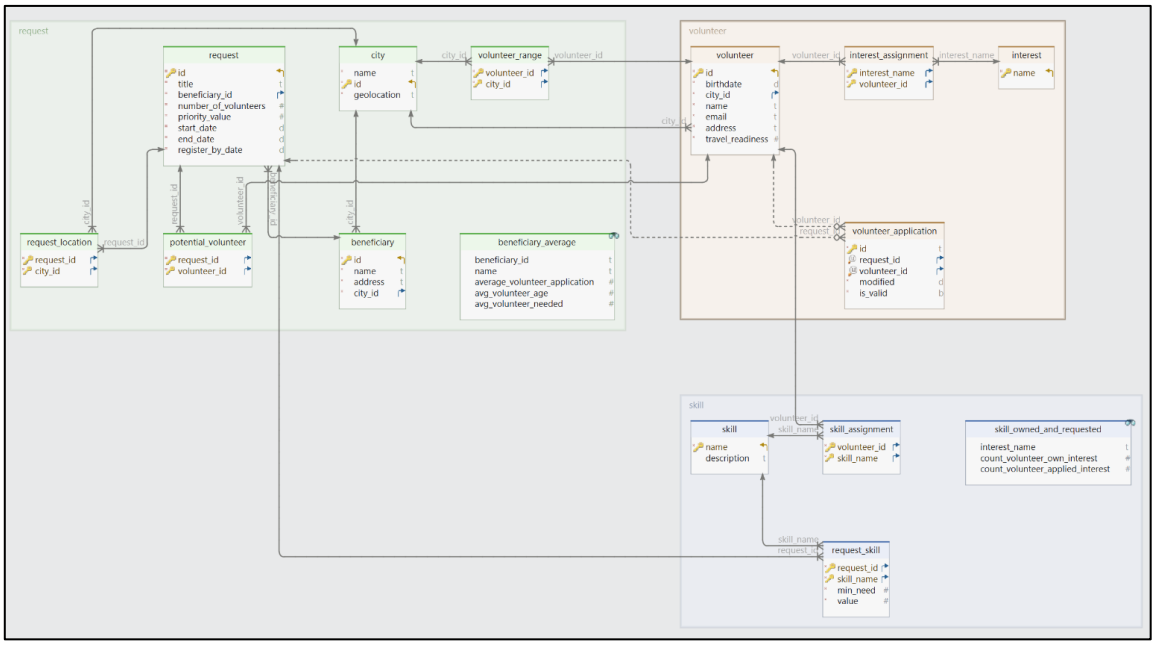



Figure 3 Database schema

#### 2.1.1.2.2 Relational tables and BCNF

Following the UML, we devised the corresponding relational tables:

- request (id, title, beneficiary_id, number_of_volunteers, priority_value, start_date, end_date, register_by_date)
  - ➔ {id -> title, beneficiary_id, number_of_volunteers, priority_value, start_date, end_date, register_by_date}
- request_location (request_id, city_id)
  - ➔ {request_id, city_id -> request_id, city_id}
- request_skill (request_id, skill_name, min_need, value)
  - ➔ {request_id, skill_name -> min_need, value}
- skill (skill_name, description)
  - ➔ {skill_name -> description}
- city (id, name, geolocation)
  - ➔ {id -> name, geolocation}
- interest_assignment (interest_name, volunteer_id)
  - ➔ {interest_name, volunteer_id -> interest_name, volunteer_id}
- beneficiary (id, name, address, city_id)
  - ➔ {id -> name, address, city_id}
- volunteer (id, birthdate, name, city_id, email, address, travel_readiness)
  - ➔ {id -> birthdate, name, city_id, email, address, travel_readiness}

- volunteer_range (volunteer_ID, city_id)
    - ➔ {volunteerI_id, city_id -> volunteerI_id, city_id}
- interest (name)
    - ➔ {name -> name}
- skill_assignment (volunteer_id, skill_name)
    - ➔ {volunteer_id, skill_name -> volunteer_id, skill_name}
- volunteer_application (id, request_id, volunteer_id, modified, is_valid)
    - ➔ {id -> request_id, volunteer_id, modified, is_valid}

Based on the relational model, we have the following relations that are in BCNF:

- request (id, title, beneficiary_id, number_of_volunteers, priority_value, start_date, end_date, register_by_date)
- skill (skill_name, description)
- city (id, name, geolocation)
- beneficiary (id, name, address, city_id)
- volunteer (id, birthdate, name, city_id, email, address, travel_readiness)
- volunteer_application (id, request_id, volunteer_id, modified, is_valid)

## 2.2 Part 2

### 2.2.1 Basic SQL queries

We approached all the queries from the supply and demand perspectives, meaning that we divided 8 queries given as well as 4 queries of our own into two types of queries, including request queries and volunteer queries.

Request queries are:

- Query 1: What are the start date and end date of each request?
- Query 2: Who are the volunteers whose skills match the requesting skills?
- Query 3: How many volunteers are missing per skill for each request?
- Query 4: What are the most important requests made by beneficiaries?
- Query 8: What are the most prioritised skills from all requests?
- Query 9: Who is the most suitable volunteer for each request?
- Query 10: Which skill is requested the most by each beneficiary?
- Query 11: How many skills based on the number of requests are there?
- Query 12: Who are the volunteers who match each request's location and have at least two matching skills but did not apply?

Volunteer queries are:

- Query 5: Which requests are within each volunteer's range and match at least two skills?
- Query 6: Which are the requests whose titles match each volunteer's area of interest?
- Query 7: Which requests were applied by the volunteers who are out of location range of the request?

We will explain query 2 and query 5 as examples for concision purposes.

In query 2, we want to find for each request the volunteers whose skill assignments match the requesting skills and to list these volunteers from those with the most matching skills to those the least. The first assumption is that 0 matching skills are considered the least matching skills. The second assumption is that only volunteers who applied to the request and have a valid application are considered.

The approach is to create a CTE volunteer_matching to find the valid volunteer_application and map out the request_skill and volunteer_skill and count cases to find the number of skills matched. The output of query 2 is presented in figure 4.

| | request_id | volunteer_id | matched_skills |
|---|---|---|---|
| 1 | 1 | 230283-963X | 3 |
| 2 | 1 | 211099-910H | 1 |
| 3 | 1 | 011074-9149 | 1 |
| 4 | 1 | 250681-919H | 1 |
| 5 | 1 | 211074-9401 | 1 |
| 6 | 1 | 160903A941P | 1 |
| 7 | 1 | 210753-990T | 0 |
| 8 | 2 | 190697-999B | 6 |
| 9 | 2 | 220782-910B | 5 |
| 10 | 2 | 270794-9576 | 5 |
| 11 | 2 | 200569-926L | 4 |
| 12 | 2 | 101003A9918 | 3 |
| 13 | 3 | 190697-999B | 7 |
| 14 | 3 | 030693-935X | 6 |
| 15 | 3 | 100494-989U | 5 |
| 16 | 3 | 220782-910B | 3 |
| 17 | 3 | 200958-9326 | 3 |
| 18 | 4 | 231269-913S | 6 |
| 19 | 4 | 250684-9410 | 6 |
| 20 | 4 | 010573-901K | 6 |
| 21 | 4 | 311267-9044 | 2 |
| 22 | 4 | 100104A910M | 0 |

Figure 4 Output of query 2

In query 5, we want to find for each volunteer the requests that are within their volunteer range and match at least two of their skills. An assumption is that even the requests that do not require any skills are included.

The approach is to create three CTEs: 1) city_request_matched: finds the volunteer ID that has at least one operation range that is required in the request_location of the request_id, 2) request_skill_matched: finds the count of matched skills for a request ID and the volunteer ID with the condition that at least two skills are matched, and 3) request_no_skill: find the volunteer ID and request ID mapping that do not require a skill. After that we join 1) city_requested_matched and request_skill_matched, and 2) city_request_matched and request_no_skill. Finally, we union these two joins to find all the combinations.

### 2.2.2 Advanced implementation

For the advanced implementation, we were interested in the following questions:

- For each beneficiary:
    - What is the average number of volunteers who applied to a request?

To find the average, we counted the number of volunteers applied to requests from a beneficiary since one volunteer with a unique id can apply to many requests for the same beneficiary and divide the value by the total number of unique requests issued by that beneficiary.

    - What is the average age of these volunteers?

We first found the age of each unique volunteer using their birthdate and find the average of them.

    - What is the average number of volunteers needed per request?

We took the average of the total number of volunteers needed by a request.

- How to identify a valid ID?

Following the standards provided by Vero, we created a trigger to check the volunteer id validity before inserting that into the volunteer table.

- How to update the total number of volunteers needed when changes are made to the request skill requirements table?

For this question, we created a trigger to read the changes of min_need in the request_skill table and then find the difference the old value and the new value. After that, update the difference in the request table for the same request_id.

- How to filter potential candidates for a request before the register date ended?

We created a potential_volunteer table. Create a procedure to check if the volunteer range is within the requested location and has at least one matching skill. If conditions are satisfied, then insert into potential_volunteer table, otherwise rollback.

**Analysis:**

***The number of volunteers available in each city compared to the number of volunteers that applied for a request in that city is as follows:***

- The city with the highest number of available volunteers is Rusko (63 volunteers). The city with the second highest number of available volunteers is Hailuoto (61 volunteers).
- The city with the lowest number of available volunteers is Säkylä (49 volunteers). The city with the second lowest number of available volunteers is Liperi (55 volunteers).

- The city with the highest number of applied volunteers is Rusko (182 volunteers). The city with the second highest number of applied volunteers is Rautavaara (181 volunteers).
- The city with the lowest number of applied volunteers is Liperi (169 volunteers). The city with the second lowest number of applied volunteers is Säkylä (172 volunteers).

The correlation between **volunteer_needed** and **applied_volunteer** is: **0.58**

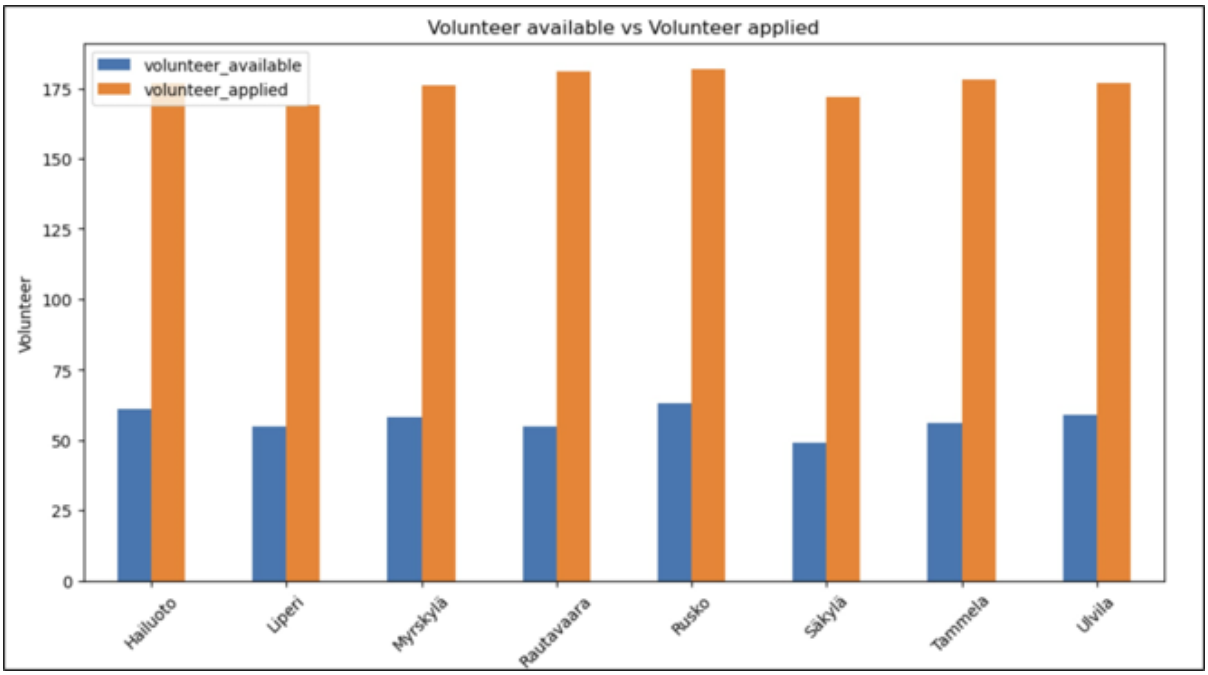



Figure 5 The number of volunteers needed vs. the number of volunteers who applied

***Scoring system to calculate the matching percentage from all the attributes of a volunteer:***

**Step 1:** Filter out potential volunteers applying for requests based on the following conditions:

1. The volunteer's interest matches the title of the request
2. The volunteer's range includes the same locations as the location of the request. The output is the number of cities matched.
3. The volunteer's travel readiness is less than 700 minutes. The average travel readiness is chosen as the threshold.
4. The volunteer has at least one matching skill required by the request.

**Step 2:** Calculate the score for the following criteria:

- **Skill match**: the number skills matched
- **Location match**: the number of cities matched
- **Travel readiness**: a score on a 1-4 scale based on four ranges of travel readiness: 0-200: 1 point, 200-400: 0.6 points, 400-600: 0.4 points, 600-700: 0.2 points.

***Note:*** *The **interest match** condition has the same score for all potential volunteers and is therefore omitted from this step onwards.*

**Step 3**: Normalise the score of skill match and location match to the range 0 to 1.

**Step 4**: Define the weights for the criteria: **skill match 50%**, **location match 30%**, and **travel readiness 20%**.

**Step 5**: Calculate weighted average score for each volunteer in a request based on the three criteria.

Outputs are displayed in figure 6 and figure 7. Figure 6 presents outputs with unnnormalised scores for each request while figure 7 shows normalised and final average score for each volunteer in a request.

| | id | volunteer_id | count_matched_skill | count_matched_city | travel_readiness_score |
|---|---|---|---|---|---|
| 1 | 1 | 011074-9149 | 2 | 2 | 0.4 |
| 2 | 1 | 211074-9401 | 2 | 2 | 0.4 |
| 3 | 1 | 250681-919H | 5 | 1 | 1 |
| 4 | 2 | 270794-9576 | 7 | 2 | 1 |
| 5 | 2 | 220782-910B | 7 | 2 | 0.4 |
| 6 | 2 | 200569-926L | 6 | 1 | 1 |
| 7 | 3 | 220782-910B | 7 | 1 | 0.4 |
| 8 | 4 | 100104A910M | 1 | 1 | 0.6 |
| 9 | 4 | 311267-9044 | 4 | 2 | 0.4 |
| 10 | 4 | 250684-9410 | 11 | 1 | 1 |
| 11 | 5 | 200958-9326 | 8 | 2 | 0.2 |
| 12 | 5 | 290464-9503 | 3 | 2 | 0.4 |
| 13 | 5 | 031161-910W | 7 | 2 | 0.4 |
| 14 | 6 | 100199-9382 | 4 | 1 | 1 |
| 15 | 6 | 311267-9044 | 4 | 1 | 0.4 |
| 16 | 7 | 250681-919H | 5 | 1 | 1 |
| 17 | 8 | 150960-943U | 7 | 1 | 1 |
| 18 | 9 | 120388-925P | 7 | 2 | 0.2 |
| 19 | 9 | 160995-949P | 7 | 1 | 0.4 |
| 20 | 9 | 220682-954Y | 3 | 1 | 0.2 |
| 21 | 10 | 270794-9576 | 7 | 1 | 1 |
| 22 | 10 | 190860-981U | 4 | 1 | 0.4 |
| 23 | 10 | 221153-9789 | 7 | 1 | 1 |
| 24 | 11 | 091105A9022 | 2 | 2 | 0.4 |
| 25 | 11 | 161058-932D | 4 | 1 | 0.6 |
| 26 | 11 | 031161-910W | 7 | 2 | 0.4 |
| 27 | 11 | 010469-981A | 9 | 1 | 0.6 |

Figure 6 Potential candidates with unnormalised scores for each request

| | request_id | volunteer_id | skill_score | location_score | travel_readiness_score | overall_score |
|---|---|---|---|---|---|---|
| 1 | 1 | 250681-919H | 0.56 | 0.2 | 1 | 0.54 |
| 2 | 1 | 011074-9149 | 0.22 | 0.4 | 0.4 | 0.31 |
| 3 | 1 | 211074-9401 | 0.22 | 0.4 | 0.4 | 0.31 |
| 4 | 2 | 270794-9576 | 0.35 | 0.4 | 1 | 0.5 |
| 5 | 2 | 200569-926L | 0.3 | 0.2 | 1 | 0.41 |
| 6 | 2 | 220782-9108 | 0.35 | 0.4 | 0.4 | 0.38 |
| 7 | 3 | 220782-9108 | 1 | 1 | 0.4 | 0.88 |
| 8 | 4 | 250684-9410 | 0.69 | 0.25 | 1 | 0.62 |
| 9 | 4 | 311267-9044 | 0.25 | 0.5 | 0.4 | 0.36 |
| 10 | 4 | 100104A910M | 0.06 | 0.25 | 0.6 | 0.23 |
| 11 | 5 | 031161-910W | 0.39 | 0.33 | 0.4 | 0.37 |
| 12 | 5 | 200958-9326 | 0.44 | 0.33 | 0.2 | 0.36 |
| 13 | 5 | 290464-9503 | 0.17 | 0.33 | 0.4 | 0.26 |
| 14 | 6 | 100199-9382 | 0.5 | 0.5 | 1 | 0.6 |
| 15 | 6 | 311267-9044 | 0.5 | 0.5 | 0.4 | 0.48 |
| 16 | 7 | 250681-919H | 1 | 1 | 1 | 1 |
| 17 | 8 | 150960-943U | 1 | 1 | 1 | 1 |
| 18 | 9 | 120388-925P | 0.41 | 0.5 | 0.2 | 0.4 |
| 19 | 9 | 160995-949P | 0.41 | 0.25 | 0.4 | 0.36 |
| 20 | 9 | 220682-954Y | 0.18 | 0.25 | 0.2 | 0.2 |
| 21 | 10 | 270794-9576 | 0.39 | 0.33 | 1 | 0.49 |
| 22 | 10 | 221153-9789 | 0.39 | 0.33 | 1 | 0.49 |
| 23 | 10 | 190860-981U | 0.22 | 0.33 | 0.4 | 0.29 |
| 24 | 11 | 010469-981A | 0.41 | 0.17 | 0.6 | 0.37 |
| 25 | 11 | 031161-910W | 0.32 | 0.33 | 0.4 | 0.34 |
| 26 | 11 | 161058-932D | 0.18 | 0.17 | 0.6 | 0.26 |
| 27 | 11 | 091105A9022 | 0.09 | 0.33 | 0.4 | 0.23 |

Figure 7 Normalised and final average score for each volunteer in a request, sorted from highest to lowest

Then, we compared top 5 candidates from the table above with the results of Query 9 to evaluate the accuracy of the framework. The output of query 9 are illustrated in figure 8.

| | request_id | volunteer_id | num_skill |
|---|---|---|---|
| 1 | 1 | 230283-963X | 3 |
| 2 | 2 | 190697-999B | 6 |
| 3 | 3 | 190697-999B | 7 |
| 4 | 4 | 231269-913S | 6 |
| 5 | 4 | 010573-901K | 6 |
| 6 | 4 | 250684-9410 | 6 |
| 7 | 5 | 130597-961E | 7 |
| 8 | 6 | 080787-976R | 6 |
| 9 | 7 | 271194-957D | 7 |
| 10 | 8 | 141194-948R | 1 |
| 11 | 8 | 230280-986W | 1 |
| 12 | 8 | 150960-943U | 1 |
| 13 | 8 | 201079-9821 | 1 |
| 14 | 9 | 180955-948M | 9 |
| 15 | 10 | 170263-963W | 6 |
| 16 | 11 | 100494-989U | 5 |
| 17 | 11 | 230283-963X | 5 |
| 18 | 12 | 220857-9810 | 7 |
| 19 | 13 | 190703A954E | 7 |
| 20 | 15 | 221153-9789 | 3 |
| 21 | 15 | 050888-938M | 3 |
| 22 | 16 | 220101A912N | 2 |
| 23 | 17 | 301198-9549 | 6 |
| 24 | 18 | 141194-948R | 4 |
| 25 | 20 | 090685-9817 | 7 |
| 26 | 21 | 090664-910H | 9 |
| 27 | 22 | 231269-913S | 3 |
| 28 | 22 | 190792-9111 | 3 |
| 29 | 23 | 150886-940M | 2 |
| 30 | 23 | 211099-910H | 2 |

Figure 8 Query 9's output

The results do not match query 9's output because only the number of skills is considered in query 9 while with this weighting system many other factors with their weights are taken into consideration simultaneously.

***The number of valid volunteer applications compared to the number of valid requests:***

The month with the highest number of *valid applications* is **July with 307 applications,** while the month with the lowest number of valid applications is **March with 127 applications**.

The month with the highest number of *valid requests is* **August with 43 requests**, and the month with the lowest number of valid requests is **February with 19 requests**.

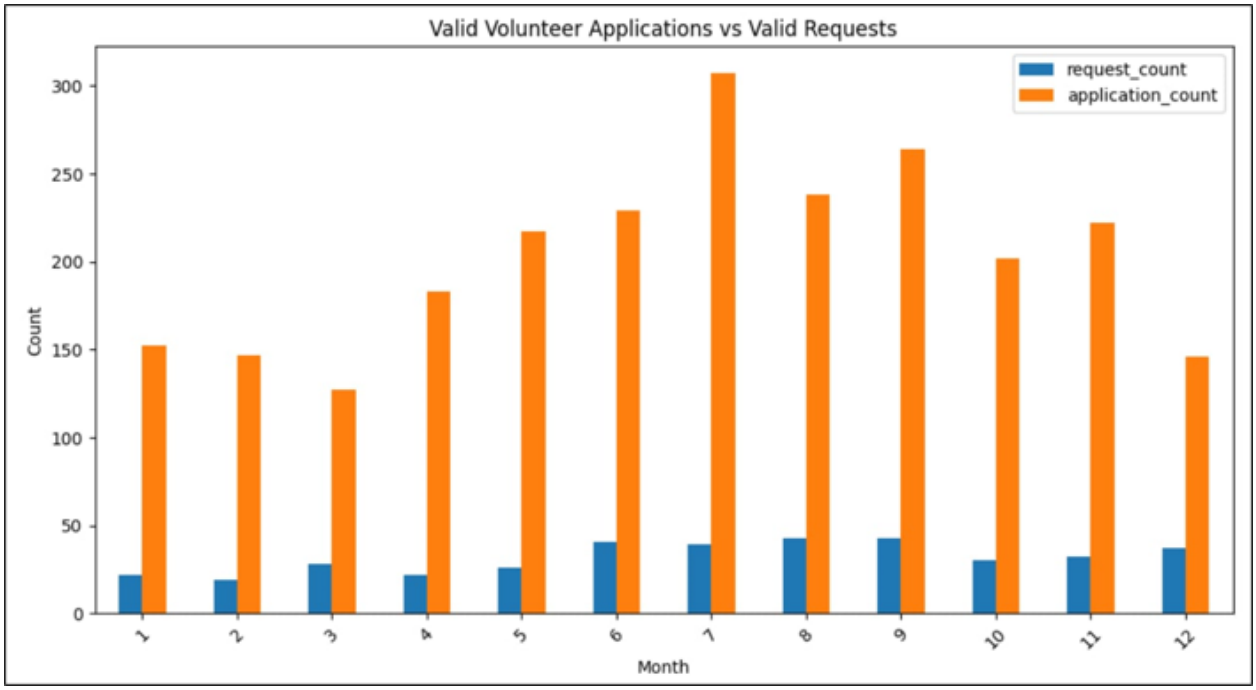



Figure 9 The number of valid volunteer applications vs. the number of valid requests in each month

It is noticeable that the number of requests remains relatively consistent throughout the year, while a seasonal pattern is evident as the number of applications tends to be higher during the summer compared to the winter.

To see the insight clearer, we deep dive into the difference between the number of requests and the number of valid applications. The most significant discrepancy can be observed during certain months in the summer, specifically from June to September.

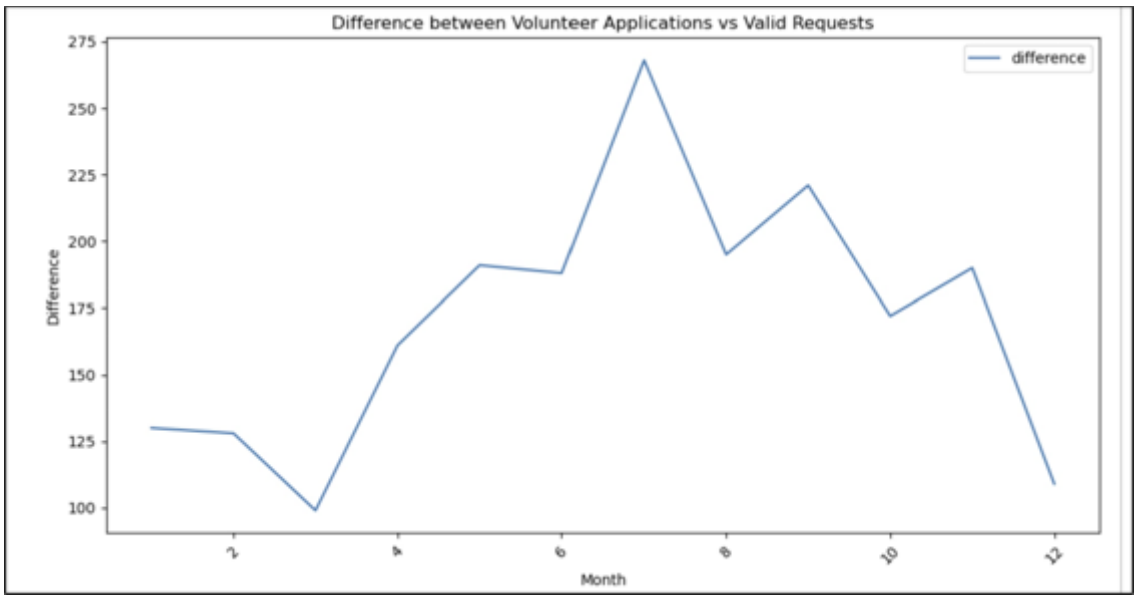



Figure 10 The difference between volunteer applications and valid requests

There is a positive correlation between **volunteer_needed** and **applied_volunteer** based on month, which is 0.66. The FRC could consider having program-specific marketing campaigns to motivate volunteers to apply more during the winter.

***Relationship between the number of requests and the number of applications based on skills:***

The correlation between **volunteer_needed** and **applied_volunteer** based on skills is only 0.2.

The positive value indicates that as the number of requests increases, the number of volunteer applications tends to increase as well. However, the correlation is weak, meaning that this trend is not very strong or consistent. The weak correlation suggests that further analysis is needed to understand the relationship fully. This could include looking at other potential variables or exploring non-linear relationships.

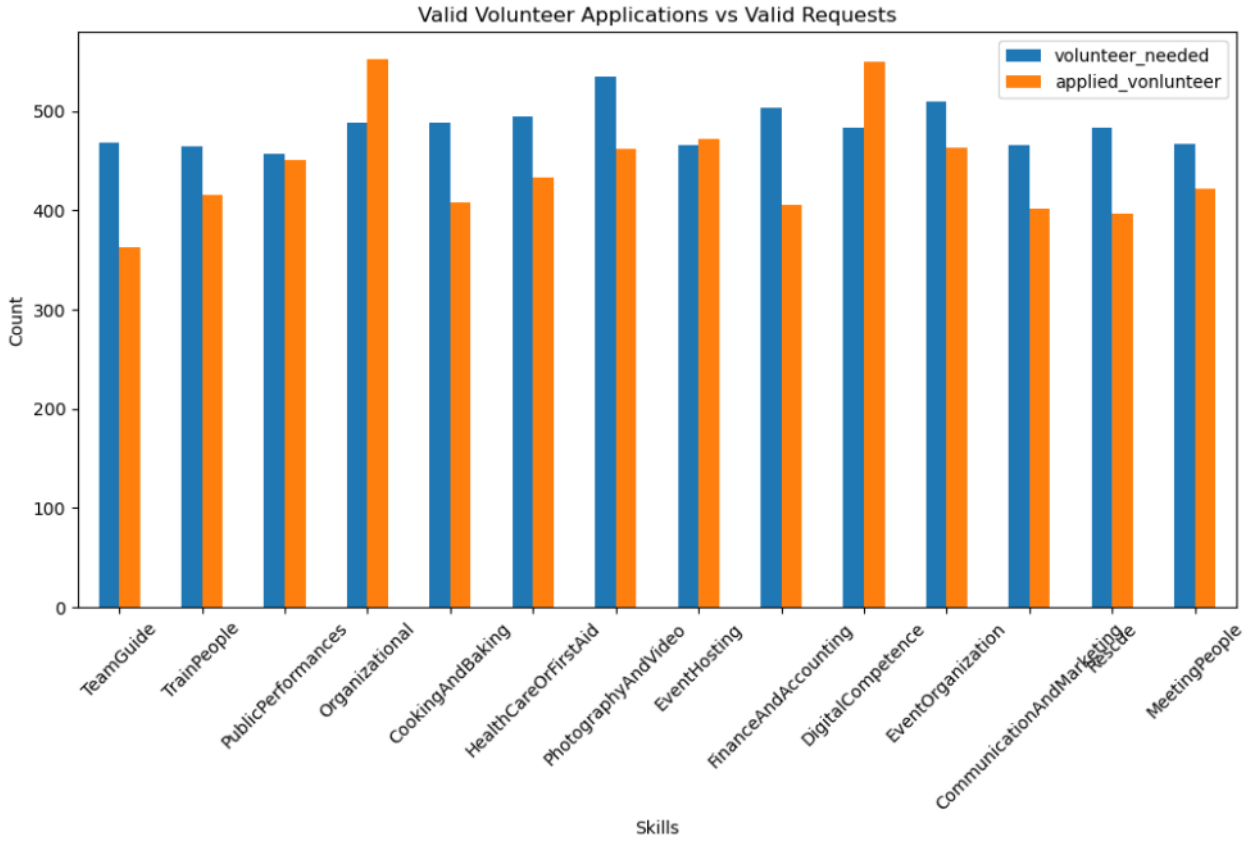



Figure 11 Valid volunteer applications vs. valid requests, sorted by skills

# 3 Reflection

## 3.1 Task division

- Hoang: leading and instructing the team, assigning the tasks, formulating part 1's UML and relational tables, finalising part 1's document, creating part 2's basic and advanced queries, doing optional additions
- Huy: doing part 1's BCNF, helping with part 2's basic and advanced queries, making presentation structures and slides, writing and finalising the final deliverables
- Linh: formulating part 1's UML and relational tables, creating part 2's basic and advanced queries, revising UML done in part 1, writing the final deliverables

- Tinh: formulating part 1's UML and relational tables, finalising part 1's document, creating part 2's basic and advanced queries, revising schema done in part 1, doing optional additions

### 3.2 Learnings

The project demonstrated the importance of thorough database design and normalisation, effective use of SQL for data manipulation, and the ability to derive meaningful insights from data analysis. Collaborating with the FRC provided a real-world context, enhancing the practical relevance of the project and contributing to a meaningful cause. Some of our key learnings include:

- UML and relational database design: We successfully designed a UML model and corresponding relational tables in BCNF, ensuring accurate representation of entities and associations for the Volunteer Matching System.
- SQL implementation: We developed and executed a variety of SQL queries to extract and manipulate data, enhancing our skills in practical data handling and complex query formulation.
- Advanced data analysis: Our analysis revealed seasonal trends in volunteer applications and identified key factors affecting volunteer-request matching, highlighting the importance of multi-criteria evaluation in volunteer management.

## 4 Optional additions – Matching volunteers: Demand

### 4.1 Proposals

In project part 2, we have made three proposals for matching volunteers on the demand side, including dynamic matching, predictive analytics, and web scraping.

**Proposal 1**: Dynamic matching

With dynamic matching, we can identify **oversupply** (*num_needed* < *num_applied* i.e. the number of volunteers needed is smaller than the number of volunteers who applied) or **undersupply** (*num_needed* > *num_applied* i.e. the number of volunteers needed is larger than the number of volunteers who applied). For each case, we can assign volunteers in different ways based on different criteria as follows:

- Oversupply: Define various selection criteria to rank those volunteers who have applied. This helps single out the volunteers who meet most criteria in case there are too many applicants.
- Undersupply: When there are too few applicants, we can select all applicants without determining any additional requirements.

**Proposal 2**: Predictive analysis

We can predict the trend based on the current data (we already gain from the analysis part that there are more applications in the summer than in the winter). Different machine learning models can be utilised to predict the future. Potential methods to be used include linear regression, logistic regression, or time series models. Boosting algorithms can also be employed to increase the models' accuracy should the need arise.

**Proposal 3**: Web scraping

We can find the number of jobs on the Internet needed by each beneficiary and the number of people who have applied for a specific job. For instance, we can scrape the number of jobs through Google search or on LinkedIn using keywords “hospital jobs in Finland” and then compare the search results among the beneficiaries.

After carefully considering the feasibility of each alternative relative to time constraints, we have decided to stick to the first proposal, dynamic matching.

### 4.2 Design

In this section, we analyse how to assign the volunteers into request in an effective way.

We classify the requests into 2 types:

- Oversupply requests is requests whose number of applicants is higher than number of volunteers needed
- Undersupply requests is requests whose number of applicants is lower than number of volunteers needed

With each type of request, we have different strategy to assign the volunteers into request

- Undersupply requests: we assign all the applied volunteers to the request
- Oversupply requests: we use our developed matching system mentioned in question 2 of section Analysis to single out and order the potential volunteer based on some criteria

### 4.3 Implementation

Please find in the attached Python Jupyter Notebooks [5] the implementation and result of this part

To storing data about assigned volunteers, we create a table assigned_volunteer with 2 columns (request_id, volunteer_id)

```
In [17]: create_table_query = """
         CREATE TABLE IF NOT EXISTS assigned_volunteer (
             request_id VARCHAR,
             volunteer_id VARCHAR
         )
         """


         with connection.cursor() as cur:
             cur.execute(create_table_query)
             cur.close()
```

Figure 12: Query to create table assigned_volunteer

Then, we find the request which has the number of applicants lower than the number of volunteers needed. We call it undersupply request. After that, we assign all the applied volunteer to the request.

```
# find the request which has the number of applicants lower than the number of volunteers needed. We called it undersupply reques
# after that, we assign all the applied volunteer to the request
under_supply_query = """
    with cte as (select va.request_id, count(*) from volunteer_application va
    join request r  on va.request_id = r.id
    group by va.request_id, r.number_of_volunteers
    having count(*) <= r.number_of_volunteers
    order by cast(va.request_id as integer) asc)
    select cte.request_id,volunteer_application.volunteer_id  from cte join volunteer_application on cte.request_id = volunteer_a

"""

with connection.cursor() as cur:
    cur.execute(under_supply_query)
    results = cur.fetchall()
    update_sql = "INSERT INTO assigned_volunteer (request_id, volunteer_id) VALUES (%s, %s)"
    df = pd.DataFrame(results, columns=['request_id', 'volunteer_id'])
    for _, row in df.iterrows():
        cur.execute(update_sql, (row['request_id'], row['volunteer_id']))
    cur.close()
    print(df)
```

```
     request_id volunteer_id
0             1  211074-9401
1             1  230283-963X
2             1  250681-919H
3             1  210753-990T
4             1  211099-910H
...         ...          ...
2055        382  020759-9272
2056        382  210694-902R
2057        382  220490-940Y
2058        382  200472-937X
2059        382  010573-901K

[2060 rows x 2 columns]
```

Figure 13: Finding the undersupply requests and assign volunteers

With the oversupply request, we follow the following steps:

- First, we find the request which has the number of applicants higher than the number of volunteers needed. We call it oversupply request.
- After that, we calculate the score of each volunteer based on 3 criteria: the number of skills matched, the number of cities matched, and the travel readiness.
- Each criterion is multiplied by a single weight, i.e. the importance of the criterion.
- Finally, we rank the volunteer based on the overall score and assigned them to the respective request.

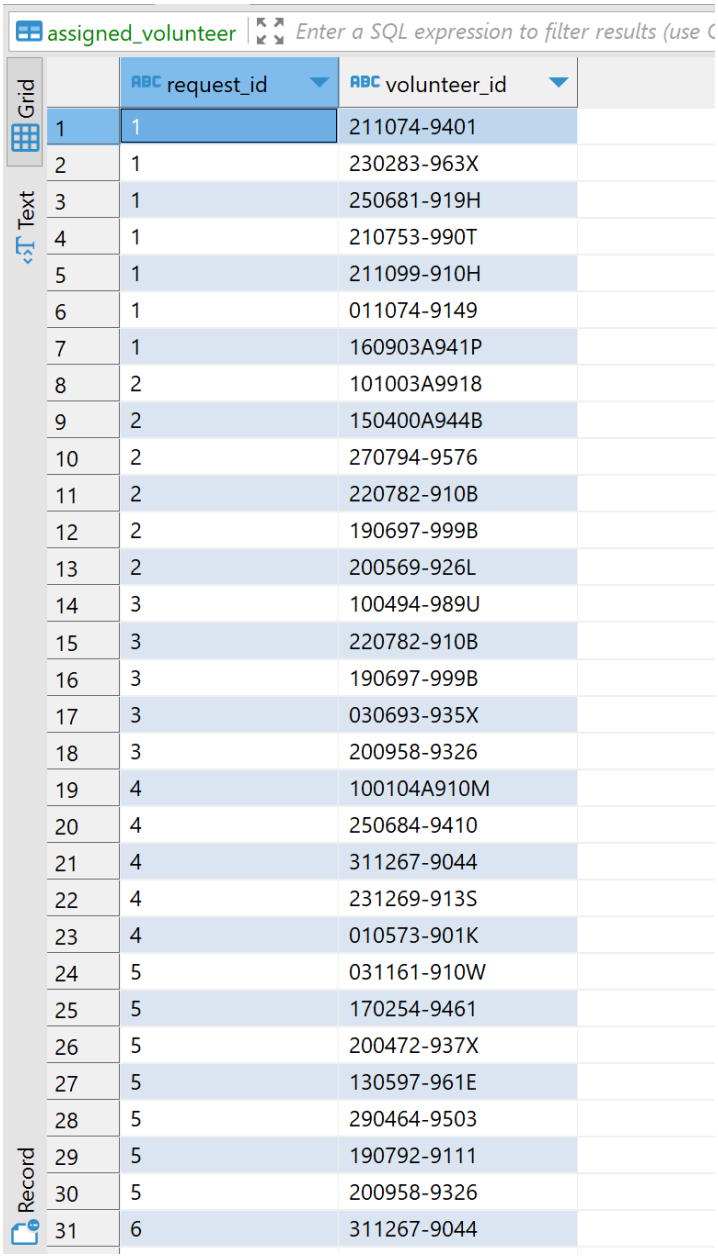
```
: # find the request which has the number of applicants higher than the number of volunteers needed. we called it oversupply reque
  # after that, we calculate the score of each volunteer based on 3 criteria: the number of skills matched, the number of cities m
  # each criteria will be multiplied with a single weight which represents the importance of the criteria
  # after that, we rank the volunteer based on the overall score and assigned them to the respective request
  oversupply_query = """
      with oversupply_request as (
      select va.request_id, count(*) from volunteer_application va
      join request r  on va.request_id = r.id
      group by va.request_id, r.number_of_volunteers
      having count(*) > r.number_of_volunteers
      order by cast(va.request_id as integer) asc
   ),oversupply_request_and_volunteer as (
      select oversupply_request.request_id,volunteer_application.volunteer_id  from oversupply_request join volunteer_application
  )
      ,cte as(
      select
          r.id, vr.volunteer_id,
          count(distinct sa.skill_name) as count_matched_skill,
          count(distinct vr.city_id) as count_matched_city,
          CASE
              WHEN v.travel_readiness >= 0 AND v.travel_readiness < 200 THEN 1
              WHEN v.travel_readiness >= 200 AND v.travel_readiness < 400 THEN 0.6
              WHEN v.travel_readiness >= 400 AND v.travel_readiness < 600 THEN 0.4
              WHEN v.travel_readiness >= 600 AND v.travel_readiness < 700 THEN 0.2
              ELSE 0 -- This handles any travel_readiness values that do not fit into the specified ranges
          END AS travel_readiness_score
      from request r
      join volunteer_application va on va.request_id = r.id
      join volunteer_range vr on vr.volunteer_id = va.volunteer_id
      join request_location rl on rl.city_id = vr.city_id and rl.request_id = r.id
      join volunteer v on v.id = vr.volunteer_id
      join skill_assignment sa on sa.volunteer_id =va.volunteer_id
      join request_skill rs on rs.skill_name = sa.skill_name
      where v.travel_readiness < 700 and va.is_valid = true
      group by r.id, vr.volunteer_id, v.travel_readiness
      order by cast(r.id as integer) asc
      ),
      cte2 as (
      select
      id,
      sum(count_matched_skill) as sum_count_matched_skill,
      sum(count_matched_city) as sum_count_matched_city
      from cte
      group by id
      order by cast(id as integer) asc
      )
      select
      cte.id as request_id,
      cte.volunteer_id,
      cte.count_matched_skill/cte2.sum_count_matched_skill as skill_score,
      cte.count_matched_city/cte2.sum_count_matched_city as location_score,
      cte.travel_readiness_score,
      ((cte.count_matched_skill/cte2.sum_count_matched_skill)*0.5 +
      (cte.count_matched_city/cte2.sum_count_matched_city)*0.3 +
      (cte.travel_readiness_score)*0.2) as overall_score

      from cte
      join cte2 on cte.id= cte2.id
      join oversupply_request_and_volunteer on cte.id=oversupply_request_and_volunteer.request_id and cte.volunteer_id = oversuppl
      order by cte.id::numeric, (cte.count_matched_skill/cte2.sum_count_matched_skill)*0.5 +
      (cte.count_matched_city/cte2.sum_count_matched_city)*0.3 +
      (cte.travel_readiness_score)*0.2 desc

  """
  with connection.cursor() as cur:
      cur.execute(oversupply_query)
      results = cur.fetchall()
      update_sql = "INSERT INTO assigned_volunteer (request_id, volunteer_id) VALUES (%s, %s)"
      df = pd.DataFrame(results, columns=['request_id', 'volunteer_id','skill_score','location_score','travel_readiness_score','ov
      for _, row in df.iterrows():
          cur.execute(update_sql, (row['request_id'], row['volunteer_id']))
      cur.close()
      print(df)
```

Figure 14: Finding the oversupply requests and assign volunteers

The result can be seen in figure 15.




| | request_id | volunteer_id |
|---|---|---|
| 1 | 1 | 211074-9401 |
| 2 | 1 | 230283-963X |
| 3 | 1 | 250681-919H |
| 4 | 1 | 210753-990T |
| 5 | 1 | 211099-910H |
| 6 | 1 | 011074-9149 |
| 7 | 1 | 160903A941P |
| 8 | 2 | 101003A9918 |
| 9 | 2 | 150400A944B |
| 10 | 2 | 270794-9576 |
| 11 | 2 | 220782-910B |
| 12 | 2 | 190697-999B |
| 13 | 2 | 200569-926L |
| 14 | 3 | 100494-989U |
| 15 | 3 | 220782-910B |
| 16 | 3 | 190697-999B |
| 17 | 3 | 030693-935X |
| 18 | 3 | 200958-9326 |
| 19 | 4 | 100104A910M |
| 20 | 4 | 250684-9410 |
| 21 | 4 | 311267-9044 |
| 22 | 4 | 231269-913S |
| 23 | 4 | 010573-901K |
| 24 | 5 | 031161-910W |
| 25 | 5 | 170254-9461 |
| 26 | 5 | 200472-937X |
| 27 | 5 | 130597-961E |
| 28 | 5 | 290464-9503 |
| 29 | 5 | 190792-9111 |
| 30 | 5 | 200958-9326 |
| 31 | 6 | 311267-9044 |

Figure 15 Output of the assigned_volunteer table

## 5 Conclusion

In this project, our group successfully designed and implemented a relational database system for the Finnish Red Cross (FRC) Volunteer Matching System (VMS), including (1) the creation of a UML model, (2) the normalization of relational tables into BCNF, (3) importing data into the database, and (4) the development of SQL queries to facilitate the matching of volunteers with requests.

**Key Achievements:**

***Database design and normalization:*** We built a detailed UML model that accurately represented the data structure and relationships between attributes. The UML was transformed into relational tables, ensuring that all were in BCNF to eliminate redundancy and improve data integrity.

***SQL implementation:*** We formulated and executed basic and advanced SQL queries that addressed the requirements of the VMS, including complex queries for volunteer-request matching and advanced procedures for data validation and updates. Additionally, we proposed further problems that are considered important in helping the FRC find the best volunteers for each request.

***Data analysis and insights:*** Our analysis revealed important patterns, such as seasonal trends in volunteer applications and discrepancies between volunteer supply and demand in different cities. We also developed a scoring system to evaluate volunteer suitability for requests based on multiple criteria decision analysis, providing a robust framework for volunteer-request matching.

***Optional additions and proposals****:* We proposed dynamic matching, predictive analysis, and web scraping to enhance the VMS. Our chosen approach, dynamic matching, effectively addressed both oversupply and undersupply scenarios.

Overall, we appreciate that the project has not only exposed us to designing and implementing a sophisticated database system for a real organization but also highlighted the potential for further innovations in the domain of volunteer matching systems.

**REFERENCE**

[1] https://www.uml.org/

[2] https://www.redcross.fi/

[3] https://www.redcross.fi/become-a-volunteer/

[4] https://www.redcross.fi/get-help-and-support/

[5] The attached Python Jupyter Notebook