Aalto University
School of Business

CS-A1155: Databases for Data Science 2024

COURSE PROJECT

DESIGNING A VOLUNTEER MATCHING SYSTEM WITH THE FINNISH RED CROSS (FRC)

Group 20 - Member
Dung Tran
Tien Huynh
Trang Le Forsell (L)
Zong-Rong Yang

14.06.2024

Table of Contents

List of Figures

1 Introduction

In an increasingly interconnected world, the spirit of volunteerism has emerged as a cornerstone of community development and social support. However, the process of matching willing volunteers with appropriate opportunities remains a significant challenge. Within the scope of Databases for Data Science course, our VMS project aims to utilize of the Unified Modelling Language (UML) to design relational schemas and translate complex real-world problems into coherent and functional database designs.
In addition to database design, the project emphasizes the use of SQL (Structured Query Language) and python scripting language for data extraction, manipulation and visualization to capture and analyse critical aspects of the problem domain efficiently.

This document includes an updated version of parts 1 and 2, along with additional implementation and reflections on the course project. To view the code referenced in this document, please refer to the relevant SQL or Python file, or read the document in Word format.

2 Database design

2.1 UML diagram

2.1.1 Round 1

For Round 1, we created the below UML diagram based on some additional assumptions, such as:

- One request can have many criterion (or in other words, request criteria). Each request criterion comprises of SkillCriterion and LocationCriterion. SkillCriterion is used for skill appraisal done by beneficiaries.
- One LocationCriterion has only one specific city.
- Volunteer can send up to 20 *active* applications: if a Volunteer sends 20 applications and got all rejected, when they try to participate for the 21st time, the system will error out and not let them apply.
- Volunteer and application are types of composition association: if there is no volunteer, there is no application. Similarly, if volunteer no longer exists, the application no longer exists.

- VolunteerOperatingRange is defined by the system using the combination of volunteer's postcode, how long the volunteer is willing to travel (travel_minutes) and volunteer's means of traveling (travel_mean e.g. car, public transport, walking, etc.)
- One volunteer has only one operating range because the range depends on a unique tuple postcode, travel_minutes, travel_mean.
- One City can have none or many VolunteerOperatingRange.
- There could be city without volunteer.
- However, volunteer always have a city where they can be operating in.

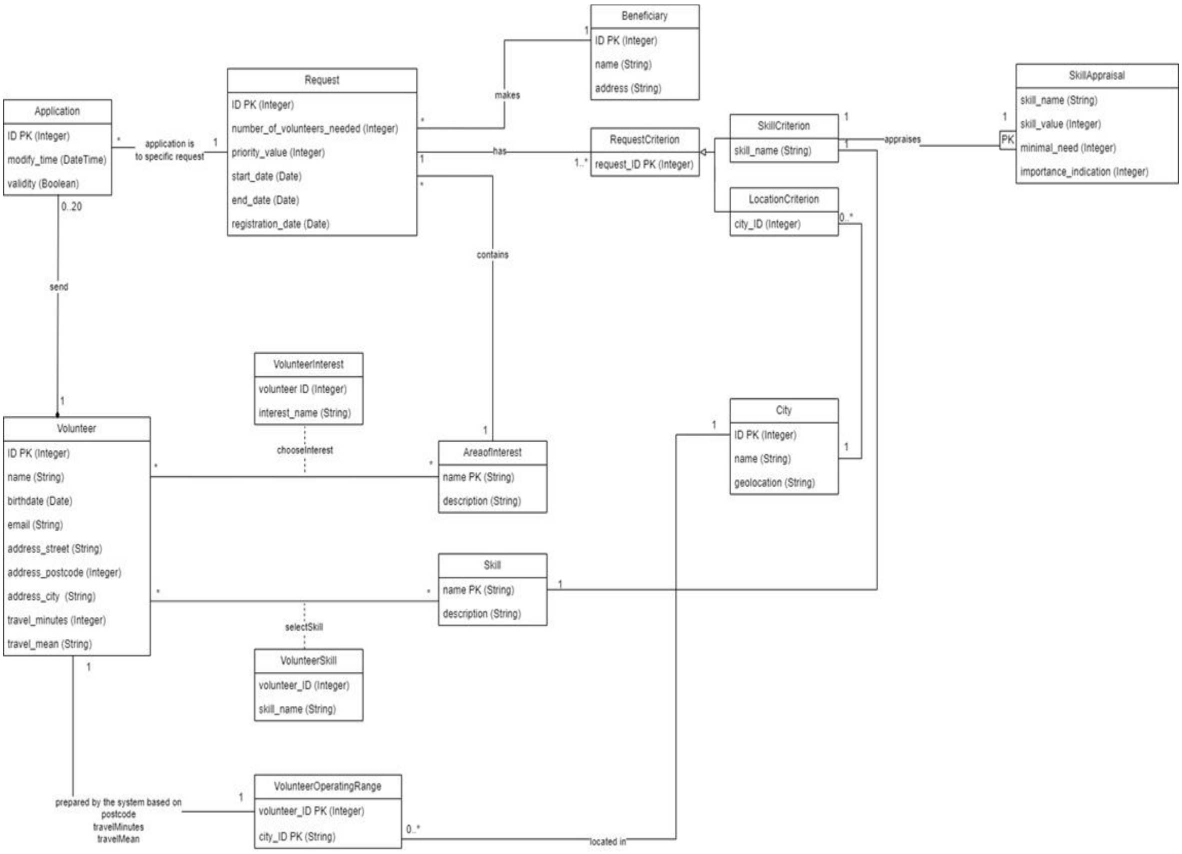


Figure 1: UML diagram - Part 1

2.1.2 Round 2

In this round, we made some modifications (as below) on our UML diagram, based on the synthetic data:

- Remove subclasses of "SkillCriterion" and "LocationCriterion"
- Remove some unnecessary attributes (i.e. 'address_street, address_postcode, address_city, travel_mean), as the synthetic data does not provide these data.

- Change ‘Request_skill’, ‘Request_location’, ‘Volunteer_range’ to associations, to avoid overlapping.
- Now ‘Request_skill’ is the association between ‘Request’ and ‘Skill’ classes. Also, the relationship between ‘Request’ and ‘Skill’ classes is determined as ‘many-many’ as based on the synthetic data, we noticed that one request can have many request skills and one skill can belongs to different requests. By converting ‘Request_skill’ to the association (with the many-many relationships between both classes), it can inherit primary keys for both classes. This practice is reasonable, as skill_name (primary key of ‘Skill’ class) or request_id (primary key of ‘Request’ class) alone is not enough to uniquely identify each tuple. Same practice for ‘Request_location’, ‘Volunteer_range’.
- There is additional info, such as minimal need and value (indicated the importance) for each request skill, hence we create an association class ‘request_skill_additional_info’ to the ‘Request_skill’ association.
- Minor changes in names of each class to match with the provide synthetic data (ex: request_skill, request_location, etc.)

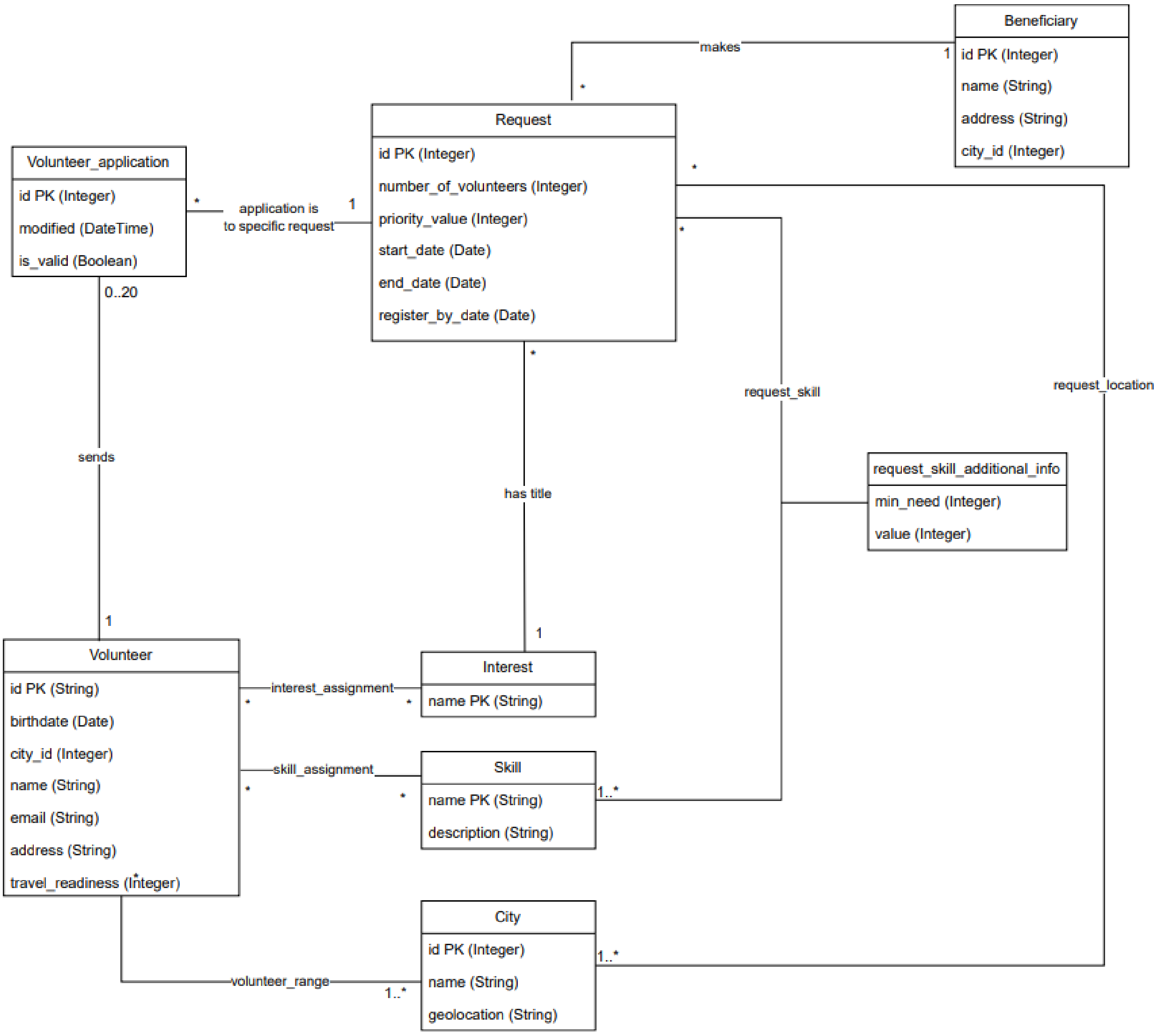


Figure 2: Modified UML diagram – Part 2

2.2 Relational Data Schema

In this section, we converted the above UML diagram to the relational data model. Primary keys of each class/association are underlined.

Note: Primary keys of each class/association are underlined. Some names of the attributes have been changed for clarity.

Step 1: For classes, relations are defined as below:

- beneficiary (id, name, address, city_id)
- volunteer_application (id, modified, is_valid)

- request (id, number_of_volunteers, priority_value, start_date, end_date, register_by_date)
- volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
- interest (name)
- skill (name, description)
- city (id, name, geolocation)

Step 2: For association, relations are defined as below:

In this step, we listed many-one/exactly one or vice versa and many-many associations.

Associations	**Relationship**	**Explanation**	**Primary_key**
makes (request_id , beneficiary_id)	many-exactly one	One beneficiary can make as many requests as they want while one request can belong to only one beneficiary.	Primary key of the many-side class – 'Request' class.
application_is_to_specific_request (request_id, application_id)	many-exactly one	One application is appliable to one request. In the meantime, one request can belong to only one beneficiary. It does not make sense when one request presents two beneficiaries at the same time.	Primary key of many-side class – 'request' class
sends (volunteer_id, application_id)	many (up to 20)-exactly one	One volunteer can send as many as 20 applications while one application belongs to only one volunteer.	Primary key of many-side class – 'volunteer' class
has_title (request_id, interest_name)	many-exactly one	One request has only 1 title (which is the name of the interest)	Primary key of many-side class – 'request' class
interest_assignment (volunteer_id, interest_name)	many - many	One volunteer is interested in many fields while a lot of volunteers have the same interest in that field as well.	Primary key of both classes – 'volunteer' class and 'interest' class

skill_assignment (volunteer_id, skill_name)	many – many	One volunteer has multiple skills at hand. At the same time there are many volunteers who possess the same skill.	Primary key of both classes – ‘volunteer’ class and ‘skill’ class
volunteer_range (volunteer_id, city_id)	many - many	One volunteer_range includes many cities. Besides, one city can belong to many volunteer_range at the same time.	Primary key of both classes – ‘volunteer’ and ‘city’ class
request_skill (request_id, skill_name, min_need,_value)	many-many	One request can request many skills. One skill can belong to many requests.	Primary key of both classes – request’ and ‘skill’ class
request_location (request_id, city_id)	many-many	One request can request many cities. One city can be chosen by many requests.	Primary key of both classes – request’ and ‘ctiy’ class

Step 3: Combined relations:

Now we considered whether or not a many-one association should have its own relation, or should the relation be replaced by simply adding the key from the one-side class to the attribute list of the many-side class.

Since each request belongs to exactly one beneficiary (indicated by the many-exactly one relationship between ‘Beneficiary’ class and ‘Request’ class), we can combine ‘makes’ association to ‘Request’ class. ‘Request’ class now includes its own attributes and the primary key of ‘Beneficiary’ class (beneficiary ID):

> request (id, beneficiary_id, number_of_volunteers, priority_value, start_date, end_date, register_by_date)

Similarly, each request has only one area of interest (indicated by the many-exactly one relationship between ‘Area of Interest’ class and ‘Request’ class), we can combine ‘has_title’ association to ‘Request’ class. ‘Request’ class now includes its own attributes and the primary key of ‘Area of Interest ’ class (the name of the area of interest), which is indicated as ‘title’ in the synthetic data:

request (id, beneficiary_id, number of volunteers, priority value,
interest_name(title), start_date, end_date, register_by_date)

Also, each application can belong to only one request (indicated by the many-exactly one relationship between 'Volunteer_application' class and 'Request' class), we can combine 'application_is_to_specific_request' association to 'Volunteer_application' class. 'Volunteer_application' class now includes its own attributes and the primary key of 'Request' class (Request ID):

volunteer_application (id, request_id, modified, is_valid)

Meanwhile, each application can be created by only one volunteer (indicated by the many-exactly one relationship between 'Volunteer_application' class and 'Volunteer' class), we can combine 'sends' association to 'Application' class. 'Application' class now includes its own attributes and the primary key of 'Volunteer' class (Request ID):

application (id, request_id, volunteer_id, modified, is_valid)

Next, 'interest_assignment', 'skill_assignment', 'request_skill', 'request_location', 'volunteer_range' are many-many associations, hence these associations stay as they are.

Final relational schema:

Class:

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

Relation:

- interest_assignment (volunteer_id, interest_name)
- skill_assignment (volunteer_id, skill_name)
- request_skill (request_id, skill_name, min_need, value)
- request_location (request_id, city_id)
- volunteer_range (volunteer_id, city_id)

2.3 BCNF and Anomalies

BCNF:

A relation needs to meet two requirements to be in BCNF:

1. Every relational determinant needs to be a candidate key.

2. No non-trivial functional dependencies should exist where the determinant is not a candidate key.

Looking at the following relations, we can see that:

1. Relation: Beneficiary (ID, name, address, city_id)
   FD: BeneficiaryID → name, address, city_id
   $\{\mathrm{BeneficiaryID}\}^{+} \rightarrow \{\mathrm{BeneficiaryID}, \mathrm{name}, \mathrm{address}, \mathrm{city\_id}\}$
   BeneficiaryID is the primary key, and name, address, and city_id are fully functionally dependent on the BeneficiaryID. Therefore, it satisfies the first condition of BCNF.
2. Relation: volunteer_application (id, request_id, volunteer_id, modified, is_valid)
   FD: id →request_id, volunteer_id, modified, is_valid
   $\{id\}^{+} \rightarrow \{id, request\_id, volunteer\_id, modified, is\_valid\}$
   id is the primary key, and request_id, volunteer_id, modified, and is_valid are fully functionally dependent on the id. Therefore, it satisfies the first condition of BCNF.
3. Relation: request (id, beneficiary_id, title, number_of_volunteers, priority_value, start_date, end_date, register_by_date)
   FD: id → beneficiary_id, title, number_of_volunteers, priority_value, start_date, end_date, register_by_date
   $\{\mathrm{id}\}^{+} \rightarrow \{\mathrm{id}, \mathrm{beneficiary\_id}, \mathrm{title}, \mathrm{number\_of\_volunteers},$
   $\mathrm{priority\_value}, \mathrm{start\_date}, \mathrm{end\_date}, \mathrm{register\_by\_date}\}$
   id is the primary key, and beneficiary_id, title, number_of_volunteers, priority_value, start_date, end_date, and register_by_date are fully functionally dependent on the id. Therefore, it satisfies the first condition of BCNF.
4. Relation: volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
   FD: id → birthdate, city_id, name, email, address, travel_readiness
   $\{\mathrm{id}\}^{+} \rightarrow \{\mathrm{id}, \mathrm{birthdate}, \mathrm{city\_id}, \mathrm{name}, \mathrm{email}, \mathrm{address}, \mathrm{travel\_readiness}\}$
   id is the primary key, and birthdate, city_id, name, email, address, and travel_readiness are fully functionally dependent on the id. Therefore, it satisfies the first condition of BCNF.
5. Relation: interest (name)
   FD: name → name

$\{\text{name}\}^+ \rightarrow \{\text{name}\}$

name is the primary key and is fully functionally dependent on itself. Therefore, it satisfies the first condition of BCNF.

6. Relation: skill (name, description)

   FD: name →description

   $\{\text{name}\}^+ \rightarrow \{\text{name}, \text{description}\}$

   name is the primary key, and description is fully functionally dependent on the name. Therefore, it satisfies the first condition of BCNF.

7. Relation: city (id, name, geolocation)

   FD: id → name, geolocation

   $\{\text{id}\}^+ \rightarrow \{\text{id}, \text{name}, \text{geolocation}\}$

   id is the primary key, and name and geolocation are fully functionally dependent on the id. Therefore, it satisfies the first condition of BCNF.

8. Relation: interest_assignment (volunteer_id, interest_name)

   FD: volunteer_id, interest_name → volunteer_id, interest_name

   $\{\text{volunteer\_id}, \text{interest\_name}\}^+ \rightarrow \{\text{volunteer\_id}, \text{interest\_name}\}$

   volunteer_id and interest_name together form the primary key and are fully functionally dependent on each other. Therefore, it satisfies the first condition of BCNF.

9. Relation: skill_assignment (volunteer_id, skill_name)

   FD: volunteer_id, skill_name → volunteer_id, skill_name

   $\{volunteer\_id, skill\_name\}\}^+ \rightarrow \{volunteer\_id, skill\_name\}$

   volunteer_id and skill_name together form the primary key and are fully functionally dependent on each other. Therefore, it satisfies the first condition of BCNF.

10. Relation: request_skill (request_id, skill_name, min_need, value)

    FD: request_id, skill_name→ min_need, value

    $\{\text{request\_id}, \text{skill\_name}\}^+ \rightarrow \{\text{request\_id}, \text{skill\_name}, \text{min\_need}, \text{value}\}$

    request_id and skill_name together form the primary key, and min_need and value are fully functionally dependent on them. Therefore, it satisfies the first condition of BCNF.

11. Relation: request_location (request_id, city_id)

    FD: request_id, city_id → request_id, city_id

    $\{\text{request\_id}, \text{city\_id}\}^+ \rightarrow \{\text{request\_id}, \text{city\_id}\}$

request_id and city_id together form the primary key and are fully functionally dependent on each other. Therefore, it satisfies the first condition of BCNF.

12. Relation: volunteer_range (volunteer_id, city_id)
    FD: volunteer_id, city_id → volunteer_id, city_id
    $\{\text{volunteer\_id}, \text{city\_id}\}^{+} \rightarrow \{\text{volunteer\_id}, \text{city\_id}\}$
    volunteer_id and city_id together form the primary key and are fully functionally dependent on each other. Therefore, it satisfies the first condition of BCNF.

There are no non-trivial functional dependencies involving determinants that are not candidate keys. As a result, the model also satisfies the second BCNF requirement too. As a result, the model is in BCNF.

Potential Anomalies

The City (id, name, geolocation) table can have problems when updating information, because if a city's name or location changes, every related record needs to be updated. To avoid this, all other tables should refer to cities by their ID. This way, changes only need to be made in the city table, keeping the data consistent and reducing repetition.

Another potential anomaly is in the Request and volunteer_application relationship. The anomaly arises from the fact that a volunteer can send up to 20 applications. With the current schema design, if a volunteer submits more than one application for the same request, each application would be treated as a separate entity. The schema doesn't enforce any constraints or rules to prevent a volunteer from sending multiple applications for the same request, potentially leading to data redundancy and loss of context.

3 Data Processing

The data cleaning process involved several key steps:

- First, as ‘request’ and ‘volunteer_application’ tables have ID as Serial, we need to resynchronize the sequence.This ensures that the sequence values align with the current maximum ‘ID’ values, guaranteeing that the next ‘ID’ generated will be unique.
    - Language used: SQL

- Second, the geolocation field in city_df was split into latitude and longitude by the "/" delimiter, converted to float data types, and the original geolocation column was removed.
    - Language used: Python
- Next, skill and interest names across multiple data frames were modified to add spaces between words using a regular expression to identify capitalized words. This transformation was applied to skill_df, skill_assignment_df, request_skill_df, interest_df, and interest_assignment_df.
    - Language used: Python
- Finally, the 'is_valid' column in volunteer_application_df was converted from boolean values (True/False) to integers (1/0) to facilitate SQL import.
    - Language used: Python

Note: Codes can be found in our accompanied SQL and Python files.

4 Basic Query

For readability, the queries section will focus on the requirements and interpretation of the final results. For detailed information on the code blocks and outcomes, please refer to the provided .py and .sql files.

4.1 Query 1

Task: For each request, include the starting date and the end date in the title.

Problem interpretation: For each request, we want to ensure that the title includes both the starting date and the end date. This means that every section or query result should have a title that specifies the time frame it covers.

4.1.1 Code Explanation

Basic - Query 1

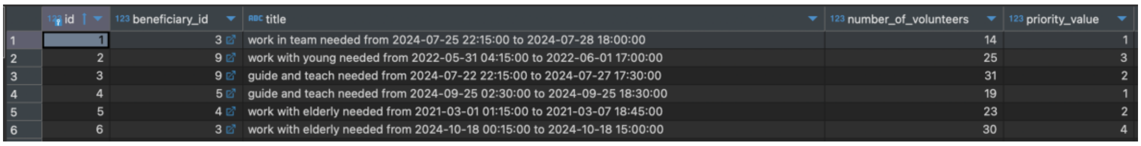
In this query:

- The || operator in SQL is used to concatenate strings. Here, it concatenates the current value of the title column with the string ' from ', ' to ' .
- The TO_CHAR function is used to convert a date or timestamp to a string in a specified format. 'YYYY-MM-DD HH24:MI:SS' is the format in which the date will be represented

4.1.2 Results

This query will produce a result set where each row includes the id, a concatenated title with the formatted start and end dates, beneficiary_ID, number_of_volunteers, priority_value, formatted start_date, formatted end_date, and register_by_date.

Due to space limit, we only showcase few results as examples:



	id	beneficiary_id	title	number_of_volunteers	priority_value
1	1	3	work in team needed from 2024-07-25 22:15:00 to 2024-07-28 18:00:00	14	1
2	2	9	work with young needed from 2022-05-31 04:15:00 to 2022-06-01 17:00:00	25	3
3	3	9	guide and teach needed from 2024-07-22 22:15:00 to 2024-07-27 17:30:00	31	2
4	4	5	guide and teach needed from 2024-09-25 02:30:00 to 2024-09-25 18:30:00	19	1
5	5	4	work with elderly needed from 2021-03-01 01:15:00 to 2021-03-07 18:45:00	23	2
6	6	3	work with elderly needed from 2024-10-18 00:15:00 to 2024-10-18 15:00:00	30	4

4.2 Query 2

Task: For each request, find volunteers whose skill assignments match the requesting skills. List these volunteers from those with the most matching skills to those with the least (even 0 matching skills). Only consider volunteers who applied to the request and have a valid application.

Problem interpretation: The goal is to create a sorted list of volunteers for each request, based on the number of matching skills, considering only those who have applied to the above-mentioned request and have valid applications. This means it is necessary to check the application's validity for each volunteer against each request. Besides, each request has a set of required skills, and each volunteer has a set of skills they possess. From the filtered list of volunteers (those who applied and have valid applications), we:

- compare their skills with the skills required by the request
- count how many skills each volunteer has that match the skills required by the request.
- then, sort the volunteers in descending order of the number of matching skills. Request with 0 matching skills should be included as well.

4.2.1 Code Explanation

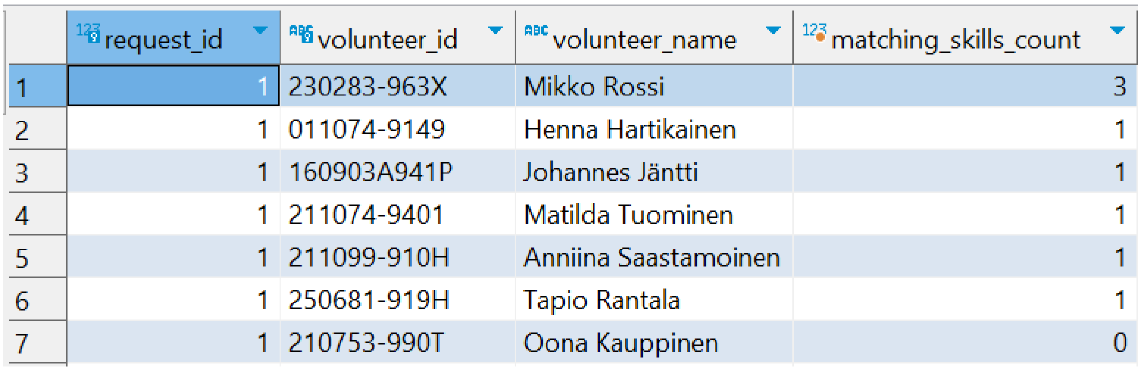
Basic - Query 2

- This query begins by selecting data from the request table, including the request ID, volunteer ID, and volunteer name. It then joins the request table with the volunteer_application table to link requests with volunteer applications. Subsequently, it connects the volunteer_application table (va) with the volunteer table to gather volunteer details.
- Additionally, a left join is performed with the skill_assignment table to include all skill assignments of volunteers. This ensures that all volunteers, regardless of their skill matches, are considered. Another left join is executed with the request_skill table to match volunteer skills with request skills, facilitating a comprehensive analysis of skill matching.
- Furthermore, the query filters the results to only include volunteers with valid applications (va.is_valid = 1). Afterward, the data is grouped by request ID, volunteer ID, and volunteer name to aggregate the count of matching skills for each volunteer per request.
- Lastly, the results are ordered by request ID, the number of matching skills in descending order, and volunteer ID. This ordering prioritizes volunteers with the most matching skills for each request, aiding in efficient volunteer assignment.

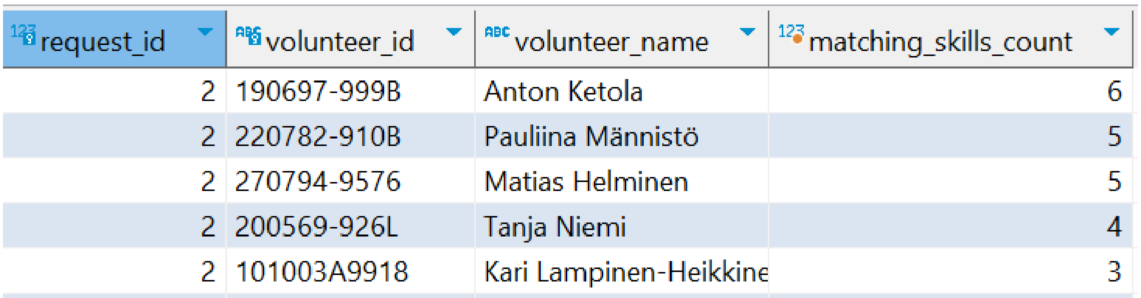
4.2.2 Results

The query generates a list of volunteers for each request, sorted by the number of matching skills (in descending order). It only includes volunteers with valid applications and considers the volunteers' skills that match the request's required skills. Volunteers with the same number of matching skills are sorted by their volunteer_id.

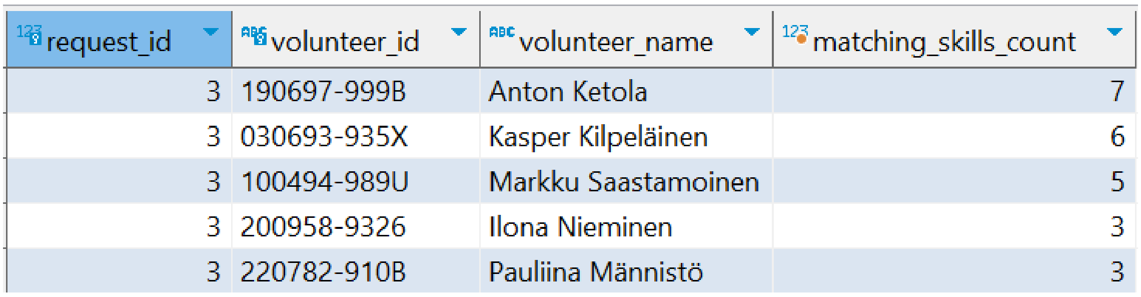
Due to space limit, we only showcase few results (Request 1, 2, 3) as examples:



	request_id	volunteer_id	volunteer_name	matching_skills_count
1	1	230283-963X	Mikko Rossi	3
2	1	011074-9149	Henna Hartikainen	1
3	1	160903A941P	Johannes Jäntti	1
4	1	211074-9401	Matilda Tuominen	1
5	1	211099-910H	Anniina Saastamoinen	1
6	1	250681-919H	Tapio Rantala	1
7	1	210753-990T	Oona Kauppinen	0



request_id	volunteer_id	volunteer_name	matching_skills_count
2	190697-999B	Anton Ketola	6
2	220782-910B	Pauliina Männistö	5
2	270794-9576	Matias Helminen	5
2	200569-926L	Tanja Niemi	4
2	101003A9918	Kari Lampinen-Heikkine	3



request_id	volunteer_id	volunteer_name	matching_skills_count
3	190697-999B	Anton Ketola	7
3	030693-935X	Kasper Kilpeläinen	6
3	100494-989U	Markku Saastamoinen	5
3	200958-9326	Ilona Nieminen	3
3	220782-910B	Pauliina Männistö	3

4.3 Query 3

Task: For each request, show the missing number of volunteers needed per skill (minimum needed of that skill). Assume a volunteer fulfills the need for all the skills they possess.

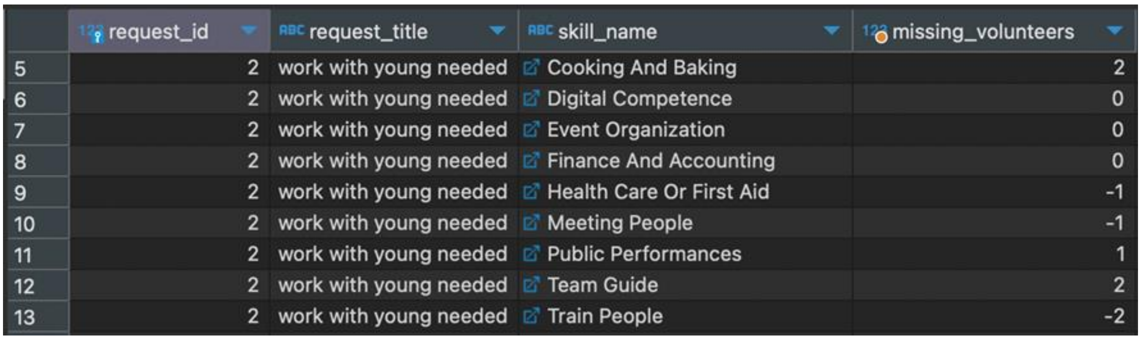
Problem interpretation: The query addresses the problem of identifying the gaps in volunteer skills for each request. Specifically, it identifies the required skills and minimum number of volunteers needed for each skill per request. Counts the number of volunteers with valid application currently applied to each request. The volunteer must possess the required skills. Then calculates the difference (missing volunteers) for each skill by subtracting the count of valid volunteers from the minimum required. Finally, the query provides a detailed list of requests along with the number of additional volunteers needed for each required skill.

4.3.1 Code Explanation

Basic - Query 3

4.3.2 Results

The result of this query will be a list of requests showing the number of additional volunteers needed for each skill per request. Positive values indicate that the skills still require more volunteers because the minimum need exceeds the current number of volunteers with those skills. Negative values indicate there is a surplus of volunteers, while a value of 0 means the number of volunteers exactly meets the minimum need. The skills for each request are ordered in ascending order.



	request_id	request_title	skill_name	missing_volunteers
5	2	work with young needed	Cooking And Baking	2
6	2	work with young needed	Digital Competence	0
7	2	work with young needed	Event Organization	0
8	2	work with young needed	Finance And Accounting	0
9	2	work with young needed	Health Care Or First Aid	-1
10	2	work with young needed	Meeting People	-1
11	2	work with young needed	Public Performances	1
12	2	work with young needed	Team Guide	2
13	2	work with young needed	Train People	-2

4.4 Query 4

Task: Sort requests and the beneficiaries who made them by the highest number of priority (request's priority value) and the closest 'register by date'.

Problem interpretation: The question asks to sort the requests and the beneficiaries who requested them based on two criteria:

- Priority Value: The requests should be sorted by the priority value in descending order, meaning that requests with a higher priority value should come first. This ensures that the most urgent or important requests are listed at the top.
- Register By Date: Within the same priority level, the requests should be sorted by the 'register by date' in ascending order, meaning that requests with the earliest 'register by date' should come first. Among requests with the same priority value, those with the nearest registration deadline are prioritized, indicating a sense of urgency for volunteer sign-ups.

- It is reasonable to consider requests whose registration deadline is not over yet to ensure that efforts are focused on requests that are still relevant and actionable.

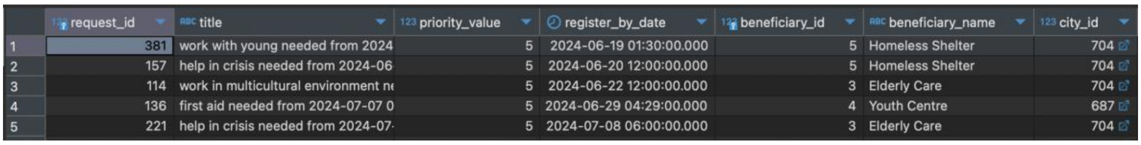
4.4.1 Code Explanation

Basic - Query 4

The SQL query selects request details such as request ID, title, beneficiary ID, beneficiary name, priority value, and register by date from the request table, joining it with the beneficiary table using the beneficiary_id. The results are ordered first by priority_value in descending order to show the highest priority requests first, and then by register_by_date in ascending order to display the closest registration dates first. This provides a sorted list of requests along with their associated beneficiaries, prioritized by urgency and closest 'register by date'.

4.4.2 Result

The result will be a list where Requests with the highest priority value are at the top. Among requests with the same priority, those with the nearest registration deadline appear first. Each request is accompanied by the details of the beneficiary who made it.



	request_id	title	priority_value	register_by_date	beneficiary_id	beneficiary_name	city_id
1	381	work with young needed from 2024	5	2024-06-19 01:30:00.000	5	Homeless Shelter	704
2	157	help in crisis needed from 2024-06	5	2024-06-20 12:00:00.000	5	Homeless Shelter	704
3	114	work in multicultural environment ne	5	2024-06-22 12:00:00.000	3	Elderly Care	704
4	136	first aid needed from 2024-07-07 0	5	2024-06-29 04:29:00.000	4	Youth Centre	687
5	221	help in crisis needed from 2024-07-	5	2024-07-08 06:00:00.000	3	Elderly Care	704

4.5 Query 5

<u>Task</u>: For each volunteer, list requests that are within their volunteer range and match at least 2 of their skills (also include requests that don't require any skills).

<u>Problem interpretation:</u> The question asks for a list of requests for each volunteer that meet two specific criteria:

- <u>Volunteer range:</u> The requests must be within the cities that the volunteer is willing to travel to.
- <u>Skill matching:</u> This filters requests to those where the volunteer has at least two of the required skills, but also includes requests that do not have any specific skill requirements, as they are open to any volunteer.

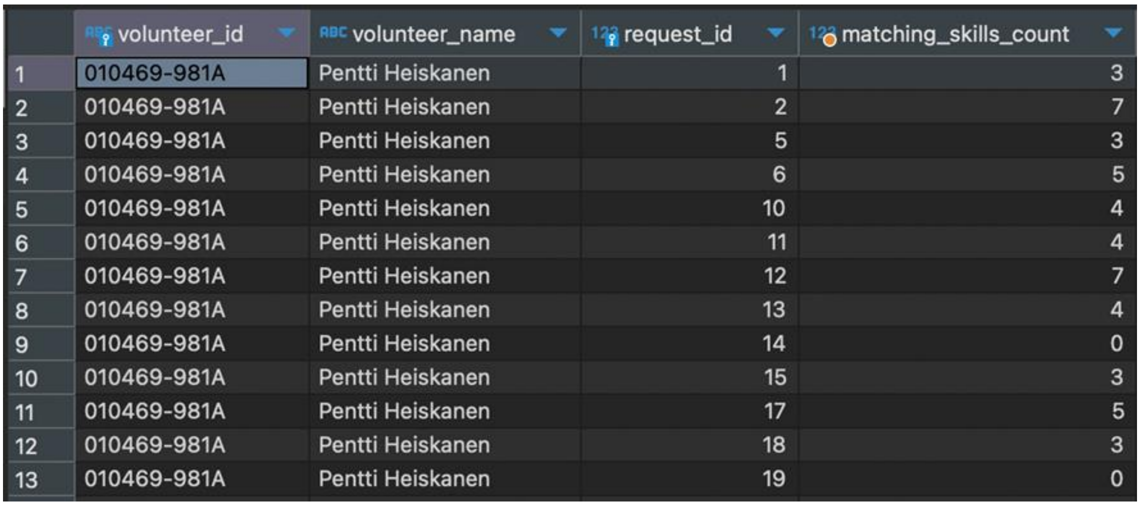
4.5.1 Code Explanation

Basic - Query 5

4.5.2 Results

The result of this query will be a list of volunteers and the requests they are eligible for, along with the count of matching skills. Specifically, each row will contain:

- Volunteer_id: The unique identifier for the volunteer.
- Volunteer_name: Name of the volunteer
- Request_id: The unique identifier for the request.
- Matching_skill_count: The number of distinct skills that match between volunteer and request.



	volunteer_id	volunteer_name	request_id	matching_skills_count
1	010469-981A	Pentti Heiskanen	1	3
2	010469-981A	Pentti Heiskanen	2	7
3	010469-981A	Pentti Heiskanen	5	3
4	010469-981A	Pentti Heiskanen	6	5
5	010469-981A	Pentti Heiskanen	10	4
6	010469-981A	Pentti Heiskanen	11	4
7	010469-981A	Pentti Heiskanen	12	7
8	010469-981A	Pentti Heiskanen	13	4
9	010469-981A	Pentti Heiskanen	14	0
10	010469-981A	Pentti Heiskanen	15	3
11	010469-981A	Pentti Heiskanen	17	5
12	010469-981A	Pentti Heiskanen	18	3
13	010469-981A	Pentti Heiskanen	19	0

4.6 Query 6

Task: For each volunteer, list all the requests where the title matches their area of interest and are still available to register.

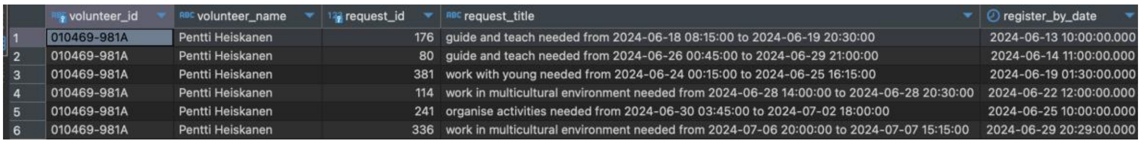
Problem interpretation: We are tasked with finding requests that align with each volunteer's specific interests and are currently open for registration. At the same time, it would make more sense to list requests that the volunteer hasn't applied for yet. This ensures that he/she can explore and potentially apply for new opportunities within areas of interest. This approach encourages volunteers to engage with a wider range of requests and helps organizations fill positions that may otherwise go unnoticed or unfulfilled.

4.6.1 Code Explanation

Basic - Query 6

4.6.2 Results

The result of the query provides a tailored list of available requests for each volunteer, matching their interests and ensuring they haven't already applied for the listed opportunities. Besides, the registration deadline is not over yet.



	volunteer_id	volunteer_name	request_id	request_title	register_by_date
1	010469-981A	Pentti Heiskanen	176	guide and teach needed from 2024-06-18 08:15:00 to 2024-06-19 20:30:00	2024-06-13 10:00:00.000
2	010469-981A	Pentti Heiskanen	80	guide and teach needed from 2024-06-26 00:45:00 to 2024-06-29 21:00:00	2024-06-14 11:00:00.000
3	010469-981A	Pentti Heiskanen	381	work with young needed from 2024-06-24 00:15:00 to 2024-06-25 16:15:00	2024-06-19 01:30:00.000
4	010469-981A	Pentti Heiskanen	114	work in multicultural environment needed from 2024-06-28 14:00:00 to 2024-06-28 20:30:00	2024-06-22 12:00:00.000
5	010469-981A	Pentti Heiskanen	241	organise activities needed from 2024-06-30 03:45:00 to 2024-07-02 18:00:00	2024-06-25 10:00:00.000
6	010469-981A	Pentti Heiskanen	336	work in multicultural environment needed from 2024-07-06 20:00:00 to 2024-07-07 15:15:00	2024-06-29 20:29:00.000

4.7 Query 7

Task: List the request ID and the volunteers who applied to them (name and email) but are not within the location range of the request. Order volunteers by readiness to travel.

Problem interpretation: We need to retrieve a list of requests along with the volunteers who have applied to them, including their names and emails. However, we're only interested in volunteers whose location range does not entirely match the location specified for the request since there may be cases where some of the volunteer's city IDs match the request's city IDs, but not all of them. In other words, we assume that a volunteer's location range does not belong to the request's location if none of the volunteer_range's city_id is present in the request's location. Furthermore, we will consider only valid volunteer applications. Additionally, we need to order these volunteers by their readiness to travel. Some requests might require volunteers to travel longer distances to reach the location where the service is needed. Ordering volunteers by their readiness to travel ensures that volunteers who can get ready to travel more quickly should be considered first. This prioritization ensures that volunteers who are more prepared to travel, potentially over longer distances, are prioritized. It also suggests a stronger commitment to volunteer work, especially if travel is a significant aspect of the opportunity.

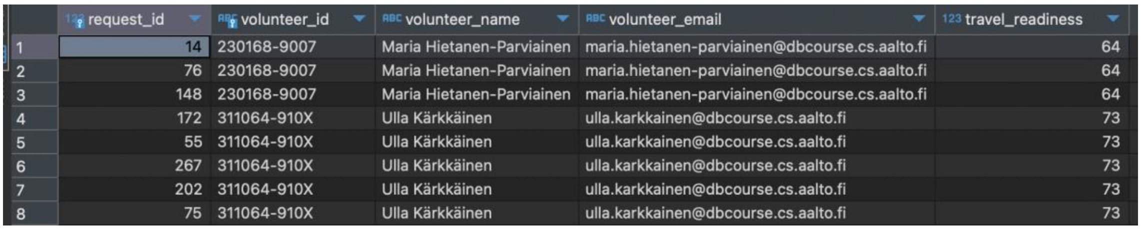
4.7.1 Code Explanation

Basic - Query 7

4.7.2 Results

Each row in the result will represent a request, displaying the request_id and the volunteers who have applied to it. This includes the volunteer's name and email.

Only volunteer applications with a status of 'valid' are considered in the result set. The query checks that none of the city_id in the volunteer's location range matches any of the ids specified for the request's location. This ensures that volunteers whose location range does not entirely align with the request's location are included in the result. The volunteers in the result set are ordered by their readiness to travel. This ordering can help prioritize volunteers who can get ready to travel more quickly should be considered first, potentially making them better suited for certain volunteer opportunities.



	request_id	volunteer_id	volunteer_name	volunteer_email	travel_readiness
1	14	230168-9007	Maria Hietanen-Parviainen	maria.hietanen-parviainen@dbcourse.cs.aalto.fi	64
2	76	230168-9007	Maria Hietanen-Parviainen	maria.hietanen-parviainen@dbcourse.cs.aalto.fi	64
3	148	230168-9007	Maria Hietanen-Parviainen	maria.hietanen-parviainen@dbcourse.cs.aalto.fi	64
4	172	311064-910X	Ulla Kärkkäinen	ulla.karkkainen@dbcourse.cs.aalto.fi	73
5	55	311064-910X	Ulla Kärkkäinen	ulla.karkkainen@dbcourse.cs.aalto.fi	73
6	267	311064-910X	Ulla Kärkkäinen	ulla.karkkainen@dbcourse.cs.aalto.fi	73
7	202	311064-910X	Ulla Kärkkäinen	ulla.karkkainen@dbcourse.cs.aalto.fi	73
8	75	311064-910X	Ulla Kärkkäinen	ulla.karkkainen@dbcourse.cs.aalto.fi	73

4.8 Query 8

Task: Order the skills overall (from all requests) in the most prioritized to least prioritized (average the importance value).

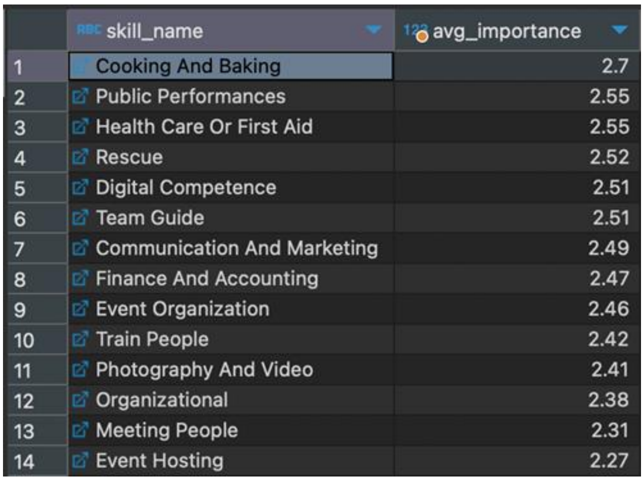
Problem interpretation: Each request may require certain skills. Each skill required for a request has an associated importance value indicating how critical that skill is for the request. In general, the question asks to determine which skills are most important on average across all requests by looking at the average importance value assigned to each skill.

4.8.1 Code Explanation

Basic - Query 8

4.8.2 Results

The average importance value of a skill indicates how critical that skill is on average across all requests. A higher average importance value means that, on average, the skill is considered more important for fulfilling the requests. With an average importance value of 2.7, 'Cooking and Baking' is the most critical skill on average. This means that requests typically rate this skill as important. Organizations should prioritize ensuring that they have enough volunteers with cooking skills to meet demand. 'Public Performances' and 'Healthcare or First Aid' shared the second and third places respectively with the same average importance value – 2.55. 'Event hosting' is the least critical, the organization might only offer this training to volunteers who specifically express an interest or to those who will be working on projects where such skills are beneficial. Organization can allocate training resources and recruitment efforts based on these priorities. At the same time, volunteers can be encouraged to acquire or improve upon higher-priority skills. In general, the average importance value helps quantify how essential each skill is across all requests, guiding the organization in making informed decisions about training, resource allocation, and volunteer placement in this project. By focusing on the most prioritized skills, the organization can improve its efficiency and effectiveness in meeting the needs of the community.



	skill_name	avg_importance
1	Cooking And Baking	2.7
2	Public Performances	2.55
3	Health Care Or First Aid	2.55
4	Rescue	2.52
5	Digital Competence	2.51
6	Team Guide	2.51
7	Communication And Marketing	2.49
8	Finance And Accounting	2.47
9	Event Organization	2.46
10	Train People	2.42
11	Photography And Video	2.41
12	Organizational	2.38
13	Meeting People	2.31
14	Event Hosting	2.27

4.9 Query 9 (Free choice)

Task: Here we run 2 queries:

- Query 9a: For each city, count the number of requests that this city received.

- Query 9b: For each city, list the requesting skills from those receiving the highest number of requests to those receiving the lowest number of requests (even skills receiving no requests).

4.9.1 Purpose

- Query 9a: By looking at the number of requests received, we can identify which cities are most in need of assistance. This query can be useful for:
  - ✓ Resource Allocation and Prioritization: Understanding the number of requests helps in allocating resources (volunteers) more effectively to cities with the highest demand.
  - ✓ Trend Analysis: Monitoring the number of requests over time can reveal trends and changes in the needs (through request volume) of different cities.

- Query 9b: By looking at the number of requests received per skill, we can analyse whether all cities have the same needs in terms of skills. Also, this query can be served for:
  - ✓ Needs Assessment: It provides a detailed overview of the specific skills that are in demand in different cities.
  - ✓ Customized Support: It helps in tailoring support and training programs to address the specific skill needs of each city. F

In general, these queries provide a comprehensive view of both the volume and type of requests across cities.

4.9.2 Code Explanation

Basic - Query 9

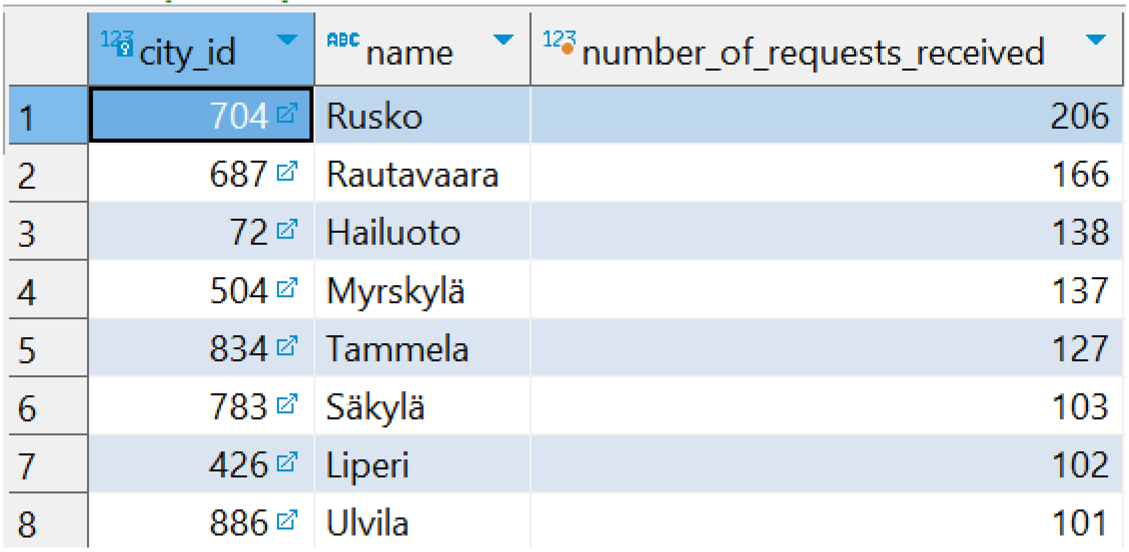
- Query 9a: This query counts the number of unique service requests received by each city. It joins the ‘request_location’ table with the ‘city’ table to obtain city names and aggregates the number of distinct requests per city. The results are then ordered by the number of requests in descending order.

- Query 9b: This query lists the skills requested in each city, ordered by the number of requests for each skill. It first generates all possible city-skill combinations using a cross join of ‘city’ table and ‘skill’ table. It then counts the number of unique requests for each skill in each city. Finally, it combines these results to ensure all combinations are included, even if there are no requests for some city-skill pairs, and orders the results by city and request count.

4.9.3 Results and Interpretation

4.9.3.1 Query 9a

- Task: For each city, count the number of requests that this city received.



	city_id	name	number_of_requests_received
1	704	Rusko	206
2	687	Rautavaara	166
3	72	Hailuoto	138
4	504	Myrskylä	137
5	834	Tammela	127
6	783	Säkylä	103
7	426	Liperi	102
8	886	Ulvila	101

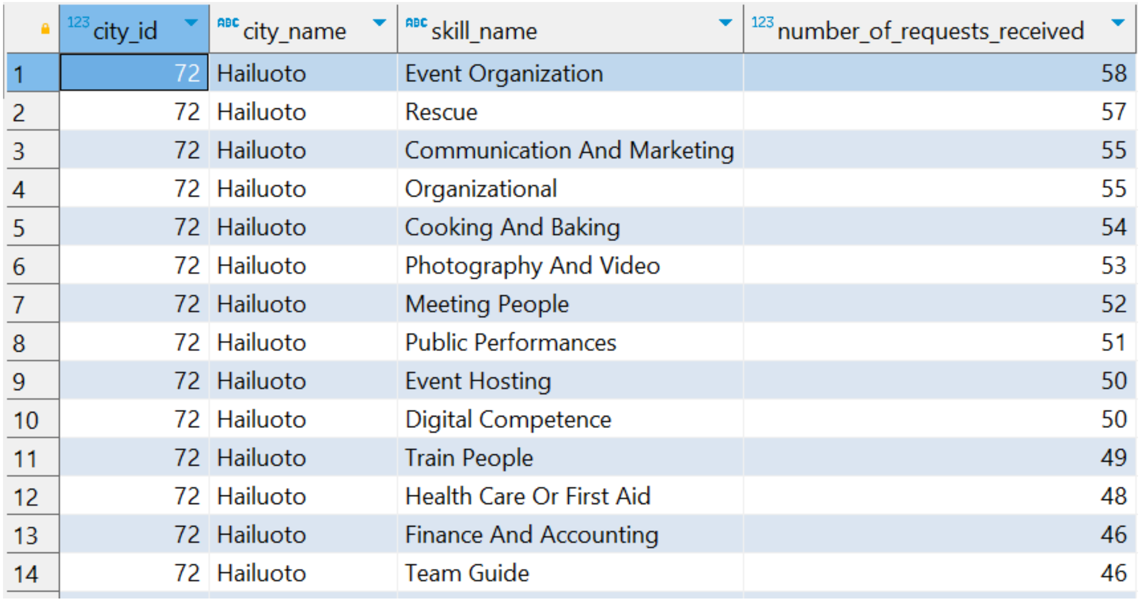
Interpretation:

- Rusko (city ID = 704) receives the highest number of requests, indicating a significant need for support in this city.
- Ulvila (city ID = 886) receives the lowest number of requests, suggesting that this city has a lower level of need compared to others.

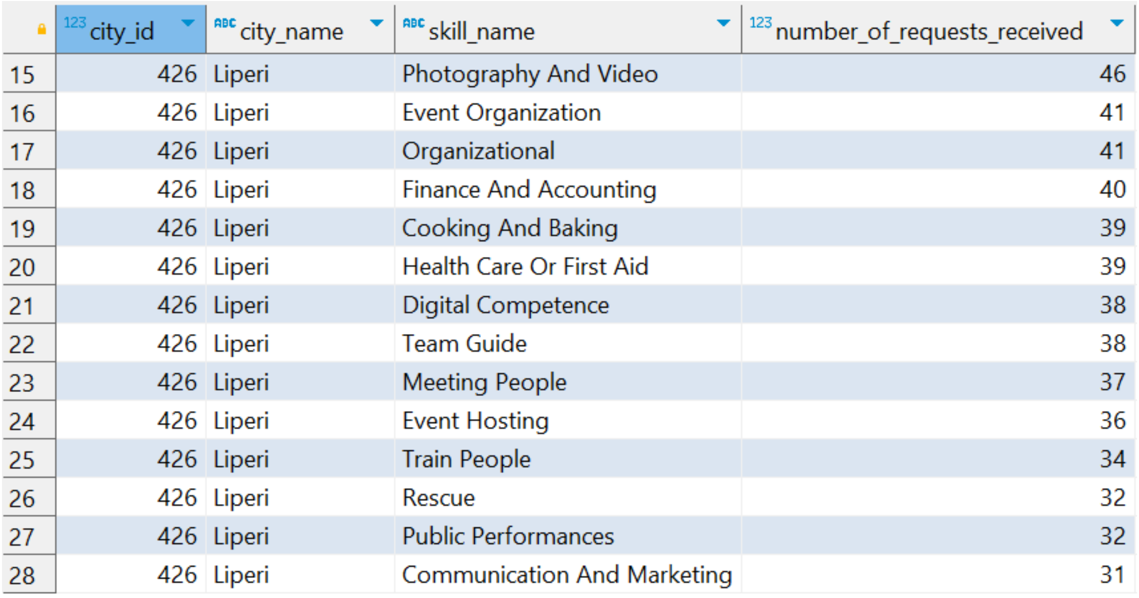
4.9.3.2 Query 9b

- Task: For each city, list the requesting skills from those receiving the highest number of requests to those receiving the lowest number of requests (even skills receiving no requests).

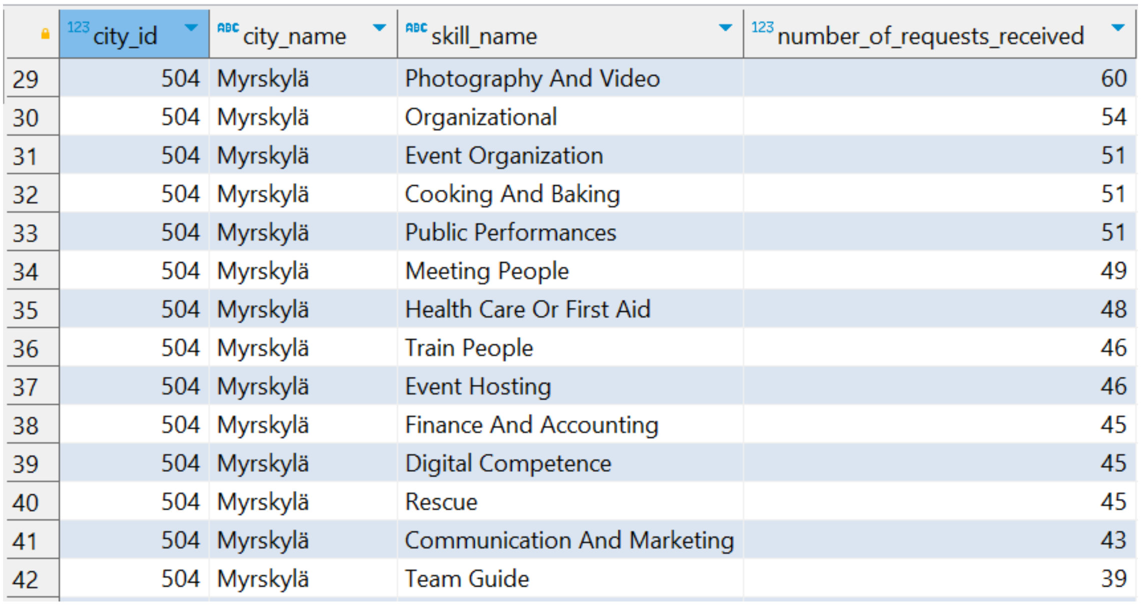
  Note: Since one request can request multiple skill, simply summing number of requests received per skills can be misleading, as is not equal to the total number of requests that one city received.



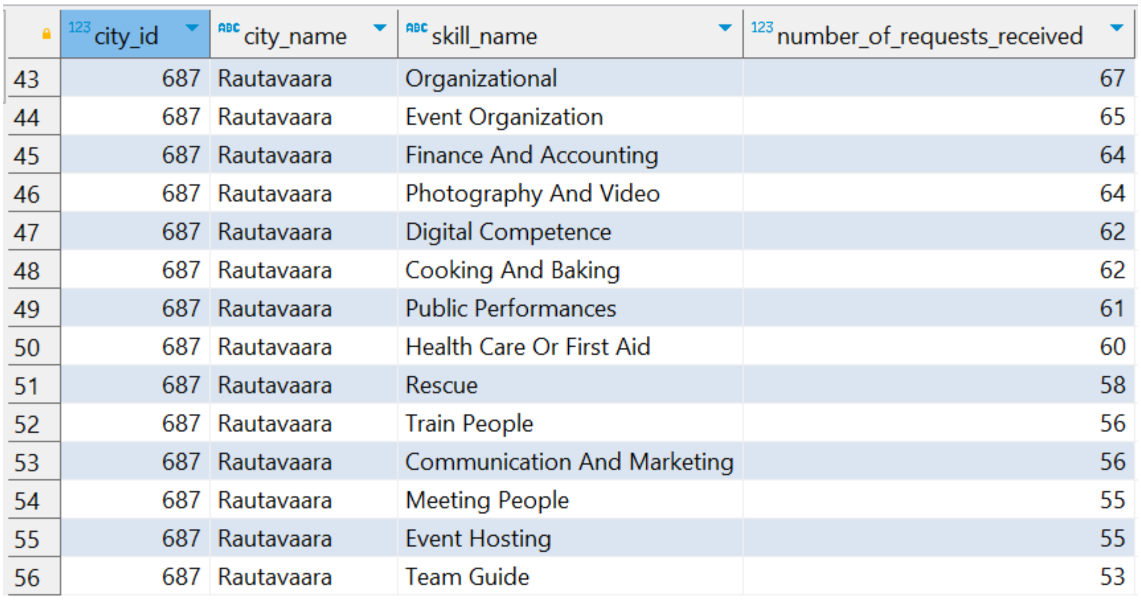
	city_id	city_name	skill_name	number_of_requests_received
1	72	Hailuoto	Event Organization	58
2	72	Hailuoto	Rescue	57
3	72	Hailuoto	Communication And Marketing	55
4	72	Hailuoto	Organizational	55
5	72	Hailuoto	Cooking And Baking	54
6	72	Hailuoto	Photography And Video	53
7	72	Hailuoto	Meeting People	52
8	72	Hailuoto	Public Performances	51
9	72	Hailuoto	Event Hosting	50
10	72	Hailuoto	Digital Competence	50
11	72	Hailuoto	Train People	49
12	72	Hailuoto	Health Care Or First Aid	48
13	72	Hailuoto	Finance And Accounting	46
14	72	Hailuoto	Team Guide	46



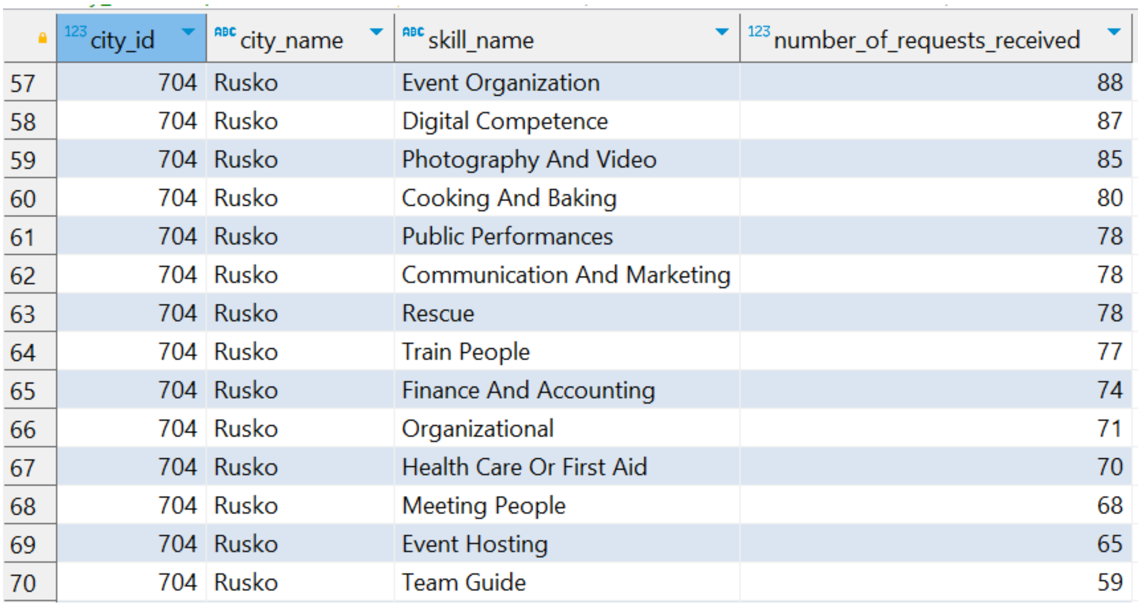
	city_id	city_name	skill_name	number_of_requests_received
15	426	Liperi	Photography And Video	46
16	426	Liperi	Event Organization	41
17	426	Liperi	Organizational	41
18	426	Liperi	Finance And Accounting	40
19	426	Liperi	Cooking And Baking	39
20	426	Liperi	Health Care Or First Aid	39
21	426	Liperi	Digital Competence	38
22	426	Liperi	Team Guide	38
23	426	Liperi	Meeting People	37
24	426	Liperi	Event Hosting	36
25	426	Liperi	Train People	34
26	426	Liperi	Rescue	32
27	426	Liperi	Public Performances	32
28	426	Liperi	Communication And Marketing	31



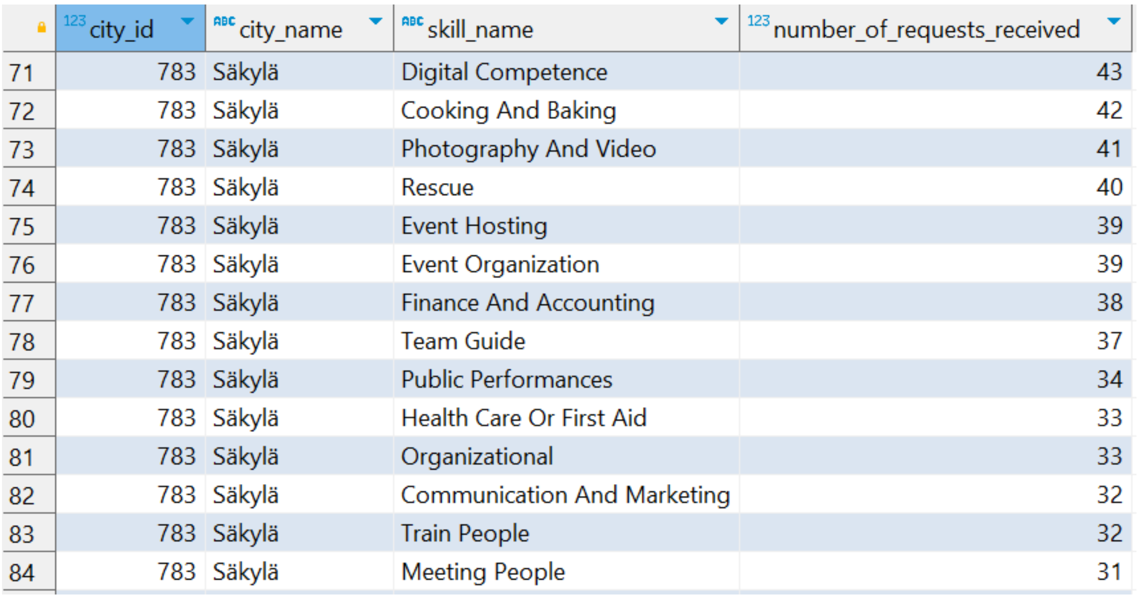
	city_id	city_name	skill_name	number_of_requests_received
29	504	Myrskylä	Photography And Video	60
30	504	Myrskylä	Organizational	54
31	504	Myrskylä	Event Organization	51
32	504	Myrskylä	Cooking And Baking	51
33	504	Myrskylä	Public Performances	51
34	504	Myrskylä	Meeting People	49
35	504	Myrskylä	Health Care Or First Aid	48
36	504	Myrskylä	Train People	46
37	504	Myrskylä	Event Hosting	46
38	504	Myrskylä	Finance And Accounting	45
39	504	Myrskylä	Digital Competence	45
40	504	Myrskylä	Rescue	45
41	504	Myrskylä	Communication And Marketing	43
42	504	Myrskylä	Team Guide	39



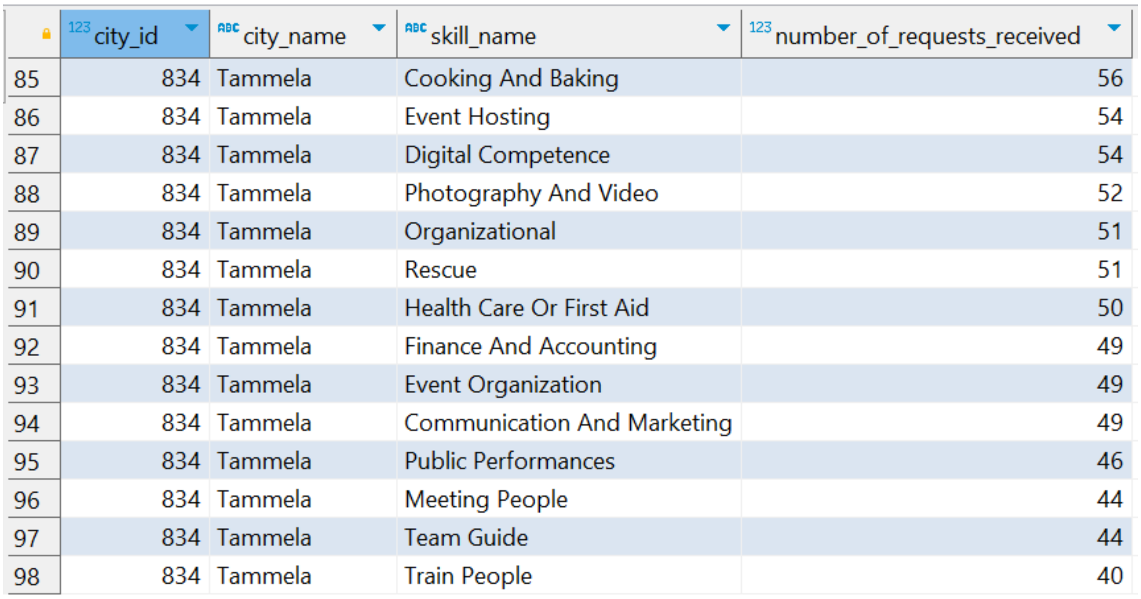
	city_id	city_name	skill_name	number_of_requests_received
43	687	Rautavaara	Organizational	67
44	687	Rautavaara	Event Organization	65
45	687	Rautavaara	Finance And Accounting	64
46	687	Rautavaara	Photography And Video	64
47	687	Rautavaara	Digital Competence	62
48	687	Rautavaara	Cooking And Baking	62
49	687	Rautavaara	Public Performances	61
50	687	Rautavaara	Health Care Or First Aid	60
51	687	Rautavaara	Rescue	58
52	687	Rautavaara	Train People	56
53	687	Rautavaara	Communication And Marketing	56
54	687	Rautavaara	Meeting People	55
55	687	Rautavaara	Event Hosting	55
56	687	Rautavaara	Team Guide	53



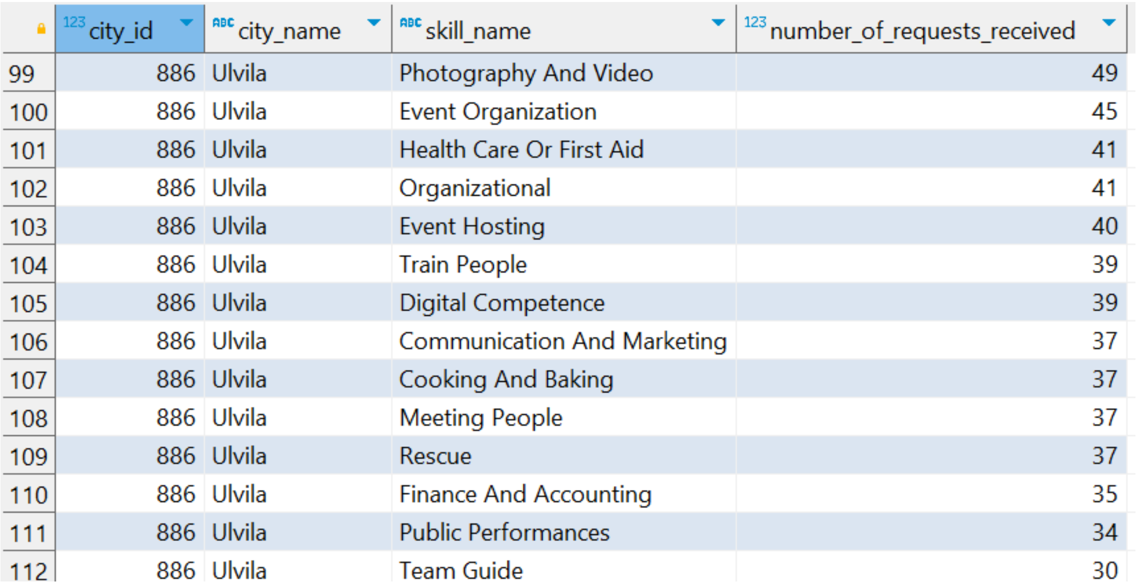
	city_id	city_name	skill_name	number_of_requests_received
57	704	Rusko	Event Organization	88
58	704	Rusko	Digital Competence	87
59	704	Rusko	Photography And Video	85
60	704	Rusko	Cooking And Baking	80
61	704	Rusko	Public Performances	78
62	704	Rusko	Communication And Marketing	78
63	704	Rusko	Rescue	78
64	704	Rusko	Train People	77
65	704	Rusko	Finance And Accounting	74
66	704	Rusko	Organizational	71
67	704	Rusko	Health Care Or First Aid	70
68	704	Rusko	Meeting People	68
69	704	Rusko	Event Hosting	65
70	704	Rusko	Team Guide	59



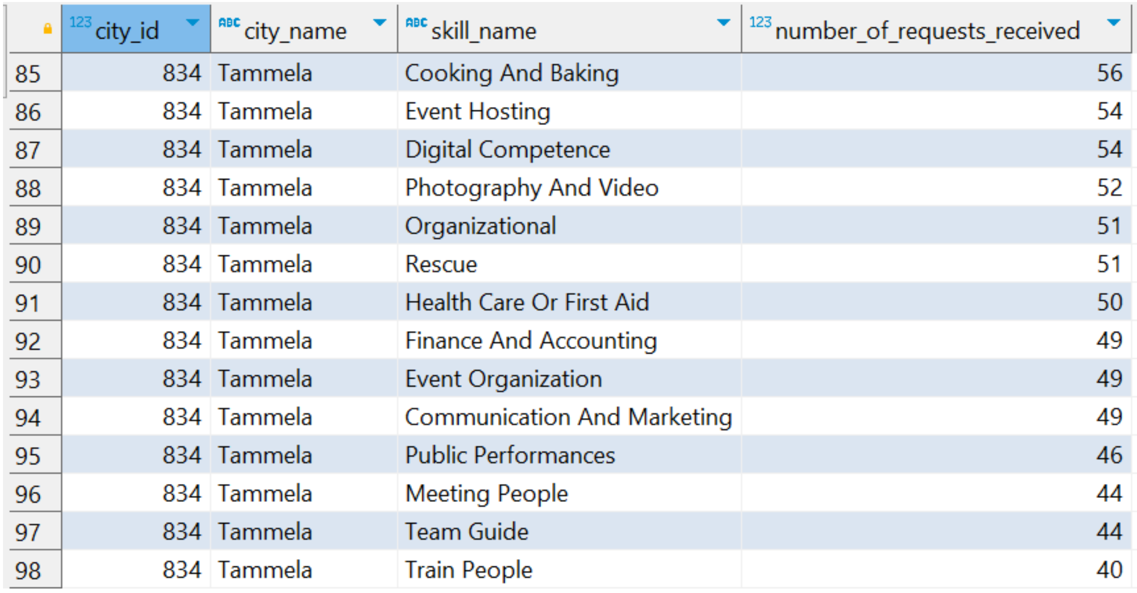
	city_id	city_name	skill_name	number_of_requests_received
71	783	Säkylä	Digital Competence	43
72	783	Säkylä	Cooking And Baking	42
73	783	Säkylä	Photography And Video	41
74	783	Säkylä	Rescue	40
75	783	Säkylä	Event Hosting	39
76	783	Säkylä	Event Organization	39
77	783	Säkylä	Finance And Accounting	38
78	783	Säkylä	Team Guide	37
79	783	Säkylä	Public Performances	34
80	783	Säkylä	Health Care Or First Aid	33
81	783	Säkylä	Organizational	33
82	783	Säkylä	Communication And Marketing	32
83	783	Säkylä	Train People	32
84	783	Säkylä	Meeting People	31



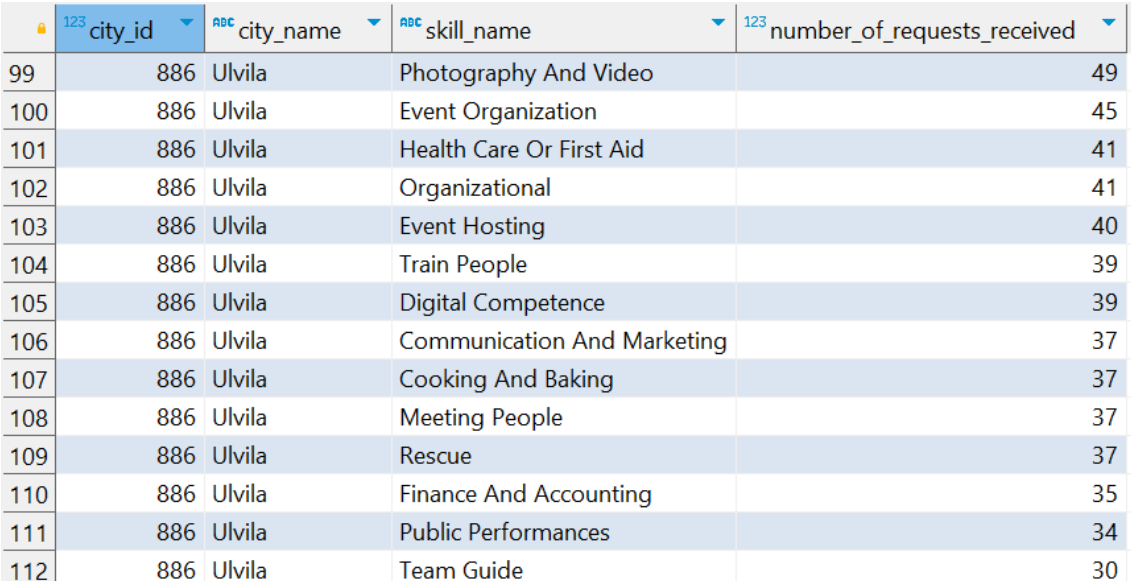
	city_id	city_name	skill_name	number_of_requests_received
85	834	Tammela	Cooking And Baking	56
86	834	Tammela	Event Hosting	54
87	834	Tammela	Digital Competence	54
88	834	Tammela	Photography And Video	52
89	834	Tammela	Organizational	51
90	834	Tammela	Rescue	51
91	834	Tammela	Health Care Or First Aid	50
92	834	Tammela	Finance And Accounting	49
93	834	Tammela	Event Organization	49
94	834	Tammela	Communication And Marketing	49
95	834	Tammela	Public Performances	46
96	834	Tammela	Meeting People	44
97	834	Tammela	Team Guide	44
98	834	Tammela	Train People	40



	city_id	city_name	skill_name	number_of_requests_received
99	886	Ulvila	Photography And Video	49
100	886	Ulvila	Event Organization	45
101	886	Ulvila	Health Care Or First Aid	41
102	886	Ulvila	Organizational	41
103	886	Ulvila	Event Hosting	40
104	886	Ulvila	Train People	39
105	886	Ulvila	Digital Competence	39
106	886	Ulvila	Communication And Marketing	37
107	886	Ulvila	Cooking And Baking	37
108	886	Ulvila	Meeting People	37
109	886	Ulvila	Rescue	37
110	886	Ulvila	Finance And Accounting	35
111	886	Ulvila	Public Performances	34
112	886	Ulvila	Team Guide	30



	city_id	city_name	skill_name	number_of_requests_received
85	834	Tammela	Cooking And Baking	56
86	834	Tammela	Event Hosting	54
87	834	Tammela	Digital Competence	54
88	834	Tammela	Photography And Video	52
89	834	Tammela	Organizational	51
90	834	Tammela	Rescue	51
91	834	Tammela	Health Care Or First Aid	50
92	834	Tammela	Finance And Accounting	49
93	834	Tammela	Event Organization	49
94	834	Tammela	Communication And Marketing	49
95	834	Tammela	Public Performances	46
96	834	Tammela	Meeting People	44
97	834	Tammela	Team Guide	44
98	834	Tammela	Train People	40



	city_id	city_name	skill_name	number_of_requests_received
99	886	Ulvila	Photography And Video	49
100	886	Ulvila	Event Organization	45
101	886	Ulvila	Health Care Or First Aid	41
102	886	Ulvila	Organizational	41
103	886	Ulvila	Event Hosting	40
104	886	Ulvila	Train People	39
105	886	Ulvila	Digital Competence	39
106	886	Ulvila	Communication And Marketing	37
107	886	Ulvila	Cooking And Baking	37
108	886	Ulvila	Meeting People	37
109	886	Ulvila	Rescue	37
110	886	Ulvila	Finance And Accounting	35
111	886	Ulvila	Public Performances	34
112	886	Ulvila	Team Guide	30

Interpretation:

- The distribution of **requested skills varies across different cities**, indicating unique local needs and priorities.
- **'Photography and Video' stands out as the most requested skill in three cities** (Liperi, Myrskylä, and Ulvila**) and maintains consistently high demand across all cities**. This skill involves capturing compelling visuals to document humanitarian efforts and inspire action and empathy, highlighting its importance in raising awareness and documenting impact.
- **'Team Guide' generally receives low requests in all cities**. This skill focuses on leading and mentoring teams to collaborate effectively and achieve

shared humanitarian goals, which might indicate a current sufficiency or less perceived urgency in leadership needs.

- **The top five skills** with the highest number of requests across all cities are predominantly **related to event organization** (i.e. event organization, organizational, photography) **and digital competence**. This highlights the increasing reliance on digital tools. Also, it indicates high needs of volunteers who can capture compelling visuals to document humanitarian efforts and inspire action and empathy.

4.10 Query 10 (Free choice)

Task: List requests that need more volunteers before registration deadlines.

4.10.1 Purpose

- The query helps identify which requests are currently undersubscribed in terms of volunteer support. By comparing the number of volunteers needed against the number currently registered (valid application only), it pinpoints requests that require more attention.
- By knowing which requests need more volunteers, organizations can focus their efforts on promoting these opportunities to potential volunteers. This ensures that resources (volunteer time, effort, etc.) are allocated efficiently where they are most needed.
- The query provides valuable data for decision-making processes within organization. It allows management board to quickly identify gaps in volunteer support and take proactive measures to address them, such as extending deadlines, increasing promotion efforts, or reallocating resources.

4.10.2 Code Explanation

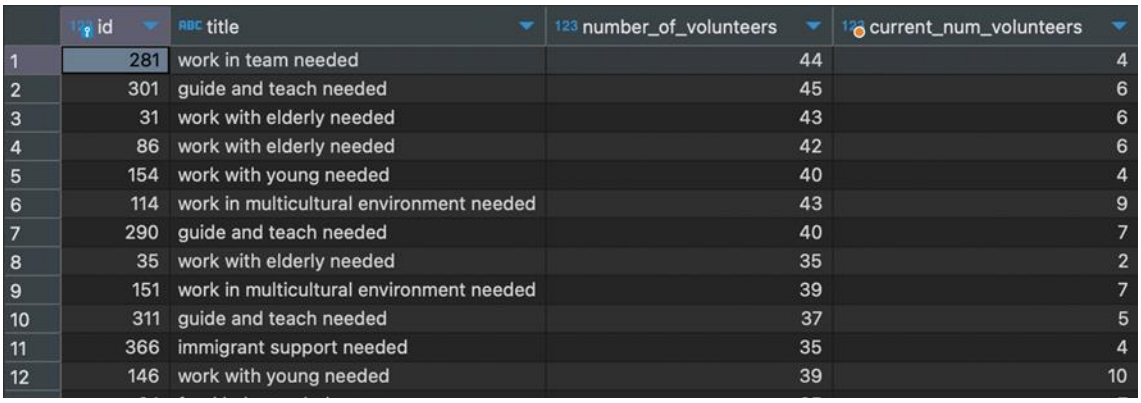
Basic - Query 10

The query specifies the columns that will be returned in the result set. It selects the request ID, title, number of volunteers needed, and the count of valid volunteer applications for each request.

Then it defines the tables from which the data will be retrieved. It selects from two tables: ‘request’ and ‘application’. ‘Left Join’ ensures that all rows from the ‘request’ table are returned, along with matching rows from the 'volunteer_application' table if they exist. The query filters the rows to only include requests with a registration deadline 'register_by_date' that is today or in the future and the grouped results where the count of valid volunteer applications (is_valid = True) is less than the number of volunteers needed for the request.

The result set is subsequently ordered by the difference between the number of volunteers needed and the current number of volunteers in descending order. It ensures that requests with the largest deficit of volunteers appear first.

4.10.3 Results and Interpretation



	id	title	number_of_volunteers	current_num_volunteers
1	281	work in team needed	44	4
2	301	guide and teach needed	45	6
3	31	work with elderly needed	43	6
4	86	work with elderly needed	42	6
5	154	work with young needed	40	4
6	114	work in multicultural environment needed	43	9
7	290	guide and teach needed	40	7
8	35	work with elderly needed	35	2
9	151	work in multicultural environment needed	39	7
10	311	guide and teach needed	37	5
11	366	immigrant support needed	35	4
12	146	work with young needed	39	10

Interpretation:

- The **most pressing volunteer requests**, which face shortages before their registration deadlines**, include opportunities to** 'work in teams', 'guide and teach', 'assist the elderly', and 'engage with young individuals'.
- By considering only requests with deadlines in the future (register_by_date >= CURRENT_DATE), the query ensures that the focus is on requests that still have time to attract more volunteers before their registration deadline expires. This helps in meeting the needs of requests within their specified timeframes.
- The query sorts the requests based on the deficit of volunteers, **allowing organizations to prioritize their efforts on the requests with the most urgent need for additional volunteers**. This ensures that critical requests are addressed promptly.

4.11 Query 11 (Free choice)

Task: Calculate the average age of volunteers and number of volunteers in each city based on their address

4.11.1 Purpose

- By calculating the average age of volunteers in each city, organizations gain insights into the demographic composition of their volunteer base. This information can help them understand the age distribution of volunteers and tailor their programs or outreach strategies accordingly.
- Knowing the average age of volunteers in each city allows organizations to target their recruitment efforts more effectively. For example, if a city has a higher average age of volunteers, the organization may focus on reaching out to younger demographics to diversify their volunteer pool
- Understanding the demographics of volunteers in each city can inform resource allocation decisions. For instance, if certain cities have a larger population of younger volunteers, the organization may allocate resources towards programs or initiatives that resonate more with that demographic.

4.11.2 Code Explanation

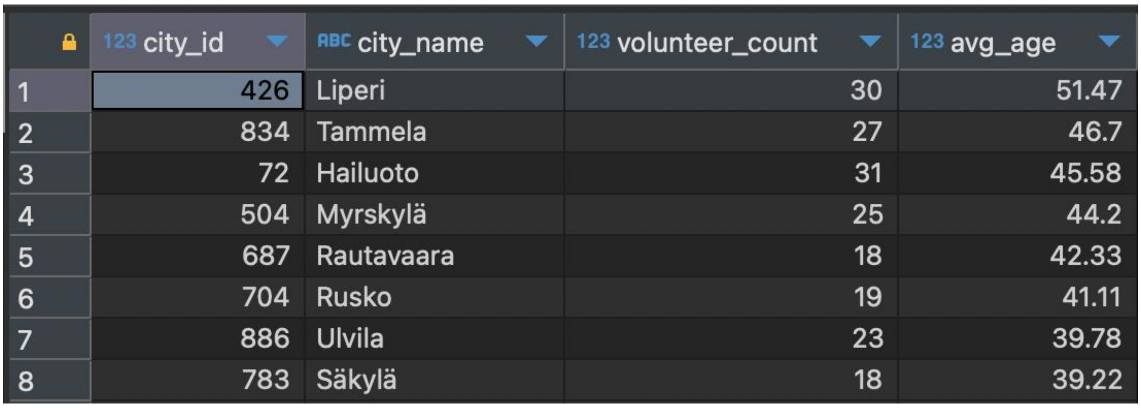
Basic - Query 11

The query selects the city ID, city name, and the average age of volunteers in each city. The ‘CAST’ function is used to convert the result of the age calculation to a decimal with a precision of 10 and a scale of 2. The average age is calculated by subtracting the birth year of each volunteer from the current year, then calculating the average of these differences. This gives an approximation of the average age of volunteers in each city.

The query joins the CTE ‘VolunteerAge’ table with the CTE ‘CityVolunteer’ table based on the ‘city_id’ column. This allows the query to retrieve information about volunteers and their respective cities. The set is grouped by ‘city name’ and ‘id’ to ensure that the average age calculation is performed for each city separately.

The result set is ordered by the average age of volunteers and number of volunteers in descending order. This helps identify cities with older volunteer populations at the top of the list.

4.11.3 Results



	city_id	city_name	volunteer_count	avg_age
1	426	Liperi	30	51.47
2	834	Tammela	27	46.7
3	72	Hailuoto	31	45.58
4	504	Myrskylä	25	44.2
5	687	Rautavaara	18	42.33
6	704	Rusko	19	41.11
7	886	Ulvila	23	39.78
8	783	Säkylä	18	39.22

Interpretation:

The provided result set offers insights into the average age of volunteers in various cities.

- Volunteers in Liperi have an average age of 51.47 years. This suggests that the volunteer base in Liperi tends to be older, possibly consisting of individuals who are more experienced or retired.
- Similar to Liperi, Tammela also has a relatively older volunteer base, although slightly younger on average.
- Ulvila marks a significant shift towards a younger volunteer population compared to the previous cities, with an average age below 40 years.
- Säkylä continues the trend of having a relatively younger volunteer base, with an average age around 39 years.

The interpretation suggests that there is variability in the age demographics of volunteers across different cities. Some cities have older volunteer populations, while others have younger ones. Understanding these age demographics can help organizations tailor their recruitment and engagement strategies to better target and serve the needs of their volunteer base in each city.

4.12 Query 12 (Free choice)

Task: Determine the most active volunteers by counting the number of valid applications they have submitted for requests. Order the list from the most active volunteers to the least active.

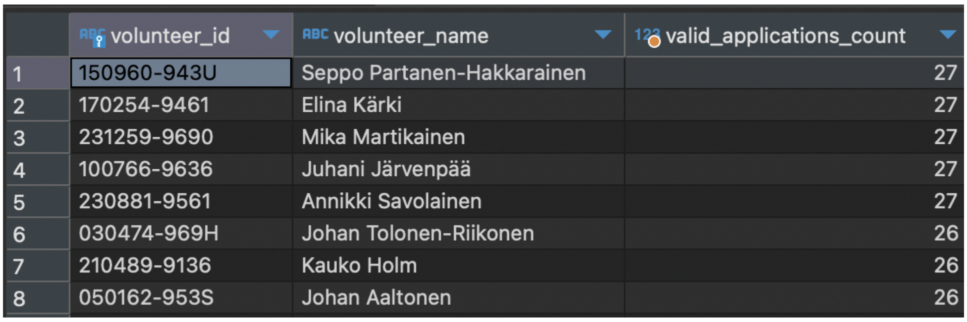
4.12.1 Purpose

This query identifies the most active volunteers based on their valid applications. It's important for the VMS because it helps recognize active volunteers, enhancing their satisfaction and retention. Active volunteers can be assigned to critical tasks, leveraging their dedication. Tracking activity provides valuable data to assess and improve recruitment and management strategies. Insights into volunteer activity guide training, communication, and engagement strategies.

4.12.2 Code Explanation

Basic - Query 12

4.12.3 Results



	volunteer_id	volunteer_name	valid_applications_count
1	150960-943U	Seppo Partanen-Hakkarainen	27
2	170254-9461	Elina Kärki	27
3	231259-9690	Mika Martikainen	27
4	100766-9636	Juhani Järvenpää	27
5	230881-9561	Annikki Savolainen	27
6	030474-969H	Johan Tolonen-Riikonen	26
7	210489-9136	Kauko Holm	26
8	050162-953S	Johan Aaltonen	26

5 Advanced Query

5.1 Views

5.1.1 View 1

Task: Create a view that lists next to each beneficiary the average number of volunteers that applied, the average age that applied, and the average number of volunteers they need across all of their requests.

Problem interpretation:

This SQL query creates a view by consolidating data from multiple tables to analyze volunteer engagement with requests from beneficiaries. It encompasses key metrics like the average count of volunteers who applied, the average age of applying volunteers, and the average number of volunteers required. There is some assumption required for this query:

- In PostgreSQL, if the birthdate is later than the current date, indicating that the volunteer's birthday has not occurred yet this year, the age is calculated to be one year less. In order to simplify the calculation, the query determines the age based on the year of birth of the volunteer extracted from 'birthdate' and the current year.
- As one volunteer can apply to many requests for one beneficiary, each volunteer is counted only once per beneficiary to maintain consistency and avoid duplication.
- Only valid applications are considered

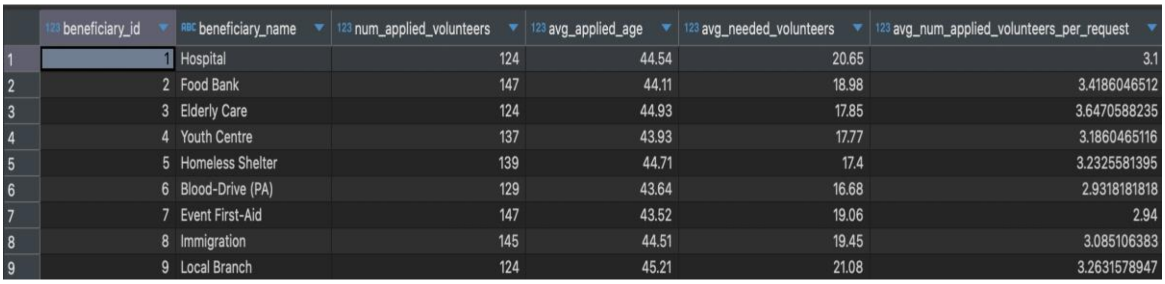
5.1.1.1 Code Explanation

Advanced - View 1

The code calculates the age of each volunteer who has applied for a request. It determines the age based on the birthdate of the volunteer extracted from the 'volunteer' table and the current year. Each volunteer is counted only once per beneficiary to maintain consistency. Only valid applications are considered. It computes statistics related to volunteer applications for each beneficiary. It tallies the distinct count of volunteers who have applied and computes the average age of these volunteers. Then counts the total number of requests for each beneficiary_ID in the request table to get the average number of volunteers applied per request for each beneficiary. These metrics are grouped by beneficiary ID.

Then the query calculates the average number of volunteers required for each beneficiary across all their requests. It derives this value from the average of the 'number_of_volunteers' column in the 'request' table, grouping the data by 'beneficiary_id'. The COALESCE function is used to handle NULL values, replacing them with 0 where appropriate.

5.1.1.2 Results and Interpretation



	beneficiary_id	beneficiary_name	num_applied_volunteers	avg_applied_age	avg_needed_volunteers	avg_num_applied_volunteers_per_request
1	1	Hospital	124	44.54	20.65	3.1
2	2	Food Bank	147	44.11	18.98	3.4186046512
3	3	Elderly Care	124	44.93	17.85	3.6470588235
4	4	Youth Centre	137	43.93	17.77	3.1860465116
5	5	Homeless Shelter	139	44.71	17.4	3.2325581395
6	6	Blood-Drive (PA)	129	43.64	16.68	2.9318181818
7	7	Event First-Aid	147	43.52	19.06	2.94
8	8	Immigration	145	44.51	19.45	3.085106383
9	9	Local Branch	124	45.21	21.08	3.2631578947

Interpretation:

- Most of the values are around 3, indicating a consistent pattern where roughly three unique volunteers apply to each request for these beneficiaries. This consistency suggests a stable level of engagement across different beneficiaries. The highest value, 3.65, suggests that one of the beneficiaries tends to attract more volunteers per request compared to others. This could indicate higher interest in that beneficiary's causes, better visibility or outreach efforts, or more compelling volunteer opportunities. The lower values, around 2.93 and 2.94, show that some beneficiaries receive slightly fewer volunteer applications per request. This could point to various factors, such as less appealing requests, lower visibility, or competition from other beneficiaries for volunteers.
- **Across most beneficiaries, the average age of volunteers who applied falls within a relatively narrow range, typically between 43 and 45 years old**. This suggests a consistent demographic trend among volunteers across different types of beneficiaries. The average ages may align with the nature of the beneficiaries' missions. For example, beneficiaries like the Elderly Care facility and the Hospital, which cater to older populations or individuals with health needs, attract volunteers with slightly higher average ages.
- **Despite the consistency in average ages, beneficiaries cater to diverse volunteer opportunities**, ranging from healthcare (Hospital) to social services (Food Bank, Homeless Shelter) and community events (Event First-Aid). This suggests that age may not be a significant barrier to volunteering across different sectors.
- **There is variability in the average number of volunteers required across different beneficiaries**. For instance, the Local Branch requires the highest average number of volunteers (21.08), while the Blood-Drive (PA) requires the lowest (16.68). Beneficiaries with higher average volunteer needs may be experiencing growth or expansion in their services, prompting the need

for additional volunteer support. Furthermore, higher volunteer needs may offer a wider range of opportunities or more flexible scheduling options to accommodate a larger volunteer workforce. Conversely, beneficiaries with lower average needs may have more stable operations or may need to explore strategies to attract more volunteers.

5.1.2 View 2 – Free choice

Task: For each request, find all nearest volunteers based on the distance between their volunteer_range and request_location. Only consider distance larger than 0.

Problem interpretation:

This view shows volunteers whose distance between volunteer_range and request_location is closest to where help is needed. It looks at requests volunteers applied to and checks if their application is valid.

Furthermore, in most real-world scenarios, a distance of 0 or less would indicate that the volunteer's location is identical or extremely close to the request location. In the context of finding the nearest volunteers for each request, it is reasonable to exclude volunteers whose range overlaps with the request location, providing more meaningful results for volunteer assignments. The goal is often to find those who need to travel the least distance to reach the request location. If a volunteer is already within the request location's range, their distance to the request is effectively zero. Including these volunteers in a distance-based calculation could skew the results and make it difficult to distinguish volunteers who genuinely need to travel to the location from those who are already nearby. Volunteers with overlapping ranges are essentially already in the desired location. Including them in the nearest distance calculations doesn't provide new information. Excluding them helps focus on volunteers who are not in the immediate vicinity but are still nearby and potentially available for the request.

5.1.2.1 Purpose

- Understanding the distance between volunteers and request locations can help in matching volunteers to requests more effectively. Volunteers who are closer to the request location may be prioritized for assignments, especially if the urgency or nature of the request requires immediate action.
- Volunteers may specify their preferred range of locations where they are willing to offer assistance. Analyzing the distance between volunteer ranges and request

locations can reveal patterns in volunteer preferences and help in aligning requests with volunteers who are most likely to accept assignments.

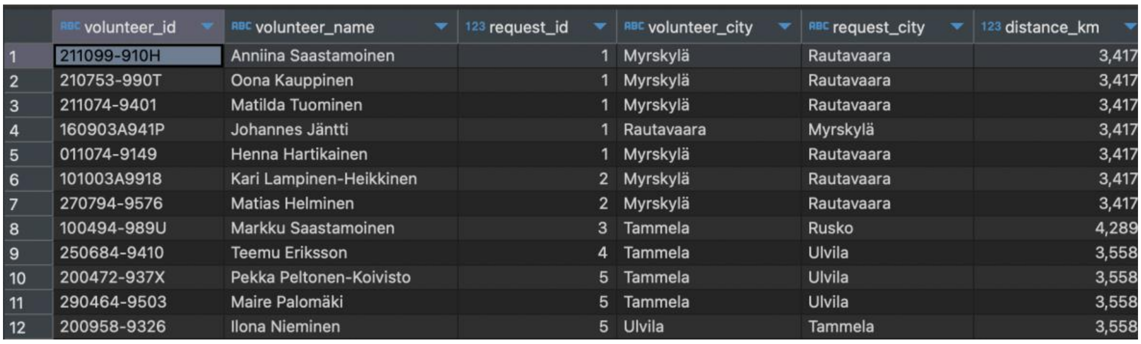
5.1.2.2 Code Explanation

Advanced - View 2

The query calculates the distance in kilometers (calculated from latitude and longitude of cities) between the volunteer's location and the location of the request they applied to assist with. DENSE_RANK() function assigns a rank to each volunteer based on their distance to each request. Volunteers with the same distance receive the same rank, ensuring that ties are handled properly.

The WHERE clause filters the results to include only volunteers who are ranked first (nearest_volunteer_rank = 1), meaning they are the closest volunteers to each request location and ensures that the distance calculated between the volunteer and the request is greater than 0.

5.1.2.3 Results and Interpretation



	volunteer_id	volunteer_name	request_id	volunteer_city	request_city	distance_km
1	211099-910H	Anniina Saastamoinen	1	Myrskylä	Rautavaara	3,417
2	210753-990T	Oona Kauppinen	1	Myrskylä	Rautavaara	3,417
3	211074-9401	Matilda Tuominen	1	Myrskylä	Rautavaara	3,417
4	160903A941P	Johannes Jäntti	1	Rautavaara	Myrskylä	3,417
5	011074-9149	Henna Hartikainen	1	Myrskylä	Rautavaara	3,417
6	101003A9918	Kari Lampinen-Heikkinen	2	Myrskylä	Rautavaara	3,417
7	270794-9576	Matias Helminen	2	Myrskylä	Rautavaara	3,417
8	100494-989U	Markku Saastamoinen	3	Tammela	Rusko	4,289
9	250684-9410	Teemu Eriksson	4	Tammela	Ulvila	3,558
10	200472-937X	Pekka Peltonen-Koivisto	5	Tammela	Ulvila	3,558
11	290464-9503	Maire Palomäki	5	Tammela	Ulvila	3,558
12	200958-9326	Ilona Nieminen	5	Ulvila	Tammela	3,558

Interpretation:

The final output is a list of volunteers who are closest to the location of the requests they have applied to other than those who locate exactly in the request location. For instance, with request_id # 1, there are 5 volunteers whose distance between in their volunteer_range and the request_location is closest.

5.2 Trigger and Functions

5.2.1 Trigger 1

Task: Create a check constraint for the volunteer table with a function that validates a volunteer ID when a new volunteer is inserted.

Problem Interpretation: Create a check constraint that validates a volunteer_id if it satisfies:

- Length: The volunteer ID should be exactly 11 characters long.
- Separator: The 7th character in the ID must be one of these: "+", "-", "A", "B", "C", "D", "E", "F", "X", "Y", "W", "V", "U".
- Control character: The control character is either a number or a letter. The value of the division's remainder determines the control character. Depending on the remainder, the control characters must be one of these: {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,H,J,K,L,M,N,P,R,S,T,U,V,W,X,Y}

5.2.1.1 Code Explanation

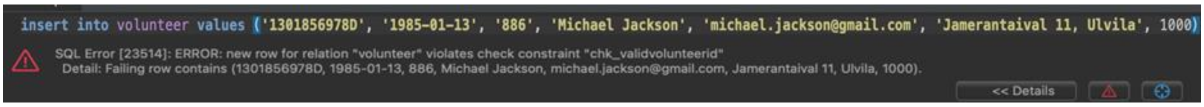
Advanced - Trigger 1

The code defines a PostgreSQL function 'calculate_control_character' and then adds a check constraint to the 'volunteer' table to ensure that the volunteer IDs meet certain criteria. The function takes a 'volunteer_id' as input (of type text) and calculates a control character based on the first 6 characters and the characters at positions 8 to 10 of the ID. It returns the calculated control character as text. The calculation is done using an array lookup based on the modulo operation of the concatenated integer value of the specified positions. Finally, the query adds a check constraint (chk_validvolunteerid) to the volunteer table. It ensures that:

- The length of the id is 11 characters.
- The character at position 7 is one of the specified characters.
- The last character of the id is equal to the control character calculated by the calculate_control_character function.

5.2.1.2 Testing and Results

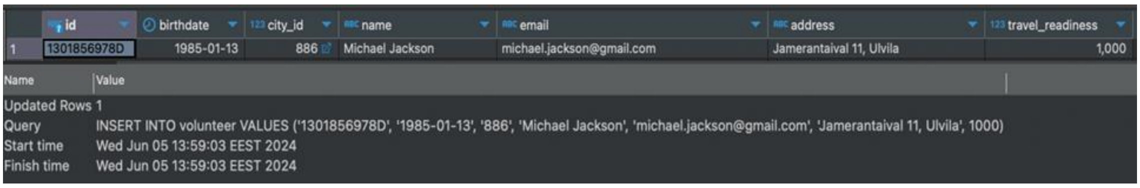
With active constraint, test command launches the constraint:



```
insert into volunteer values ('1301856978D', '1985-01-13', '886', 'Michael Jackson', 'michael.jackson@gmail.com', 'Jamerantaival 11, Ulvila', 1000)
SQL Error [23514]: ERROR: new row for relation "volunteer" violates check constraint "chk_validvolunteerid"
 Detail: Failing row contains (1301856978D, 1985-01-13, 886, Michael Jackson, michael.jackson@gmail.com, Jamerantaival 11, Ulvila, 1000).
```

- The test volunteer_id is ‘1301856978D’. This volunteer’s date of birth is ‘1985-01-13’ so the format of first 6 digits in volunteer_id would be 130185. The first digits of test value satisfies this condition.
- For the 7th digit, the volunteer was born in 1900 (‘1985’) so the corresponding character must be either ‘-‘ , ‘+’ or { ‘Y’, ‘X’, ‘W’, ‘V’, ‘U’). It is 6 in test value so the 7th digit violates the second condition of check constraint.
- The individualized string is an even number (‘978’) indicating that the volunteer’s gender is women.
- The remainder of 130185978 % 31 is 21 which corresponds to control character ‘N’. However, it is ‘D’ in testing value which violates the third condition.

<u>Without active constraint, test command does not cause any error:</u>



	id	birthdate	city_id	name	email	address	travel_readiness
1	1301856978D	1985-01-13	886	Michael Jackson	michael.jackson@gmail.com	Jamerantaival 11, Ulvila	1,000

```
Name          Value
Updated Rows  1
Query         INSERT INTO volunteer VALUES ('1301856978D', '1985-01-13', '886', 'Michael Jackson', 'michael.jackson@gmail.com', 'Jamerantaival 11, Ulvila', 1000)
Start time    Wed Jun 05 13:59:03 EEST 2024
Finish time   Wed Jun 05 13:59:03 EEST 2024
```

The test command with incorrect volunteer_id works well and the new row with new volunteer_id can be inserted into volunteer_table.

5.2.2 Trigger 2

<u>Task:</u> Create a trigger that updates the number of volunteers for a request whenever the minimum need for any of its skill requirements is changed. The total number of volunteers needed for each request is calculated as the sum of unskilled volunteers (those without any skill requirements) and the minimum need for each required skill.

<u>Note:</u> For the sake of simplicity, we consider only the ‘updating’ scenario (the trigger will be launched whenever the minimum need for any of its skill requirements is ***updated***).

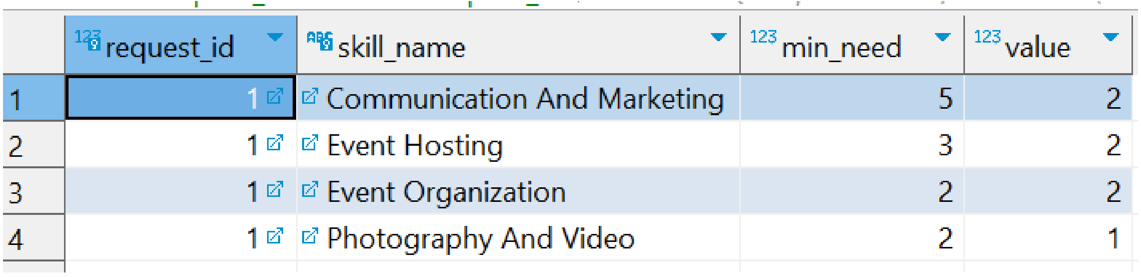
5.2.2.1 Code Explanation

Advanced - Trigger 2

5.2.2.2 Testing and Results

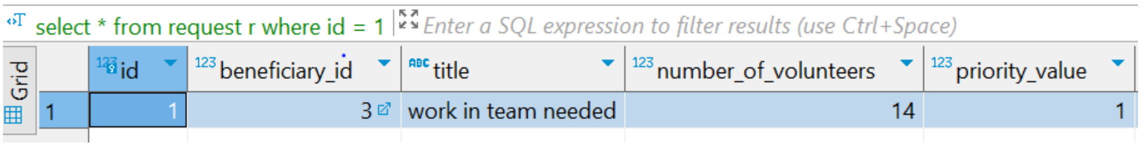
Before the trigger:

```
/* Update the minimal_need for a 'Event Hosting' skill of Request 1 in the request_skill
table. Currently, min_need of this skill = 3, number of volunteers needed for Request 1
is 14. */
select *
from request r
where id = 1;
```



	request_id	skill_name	min_need	value
1	1	Communication And Marketing	5	2
2	1	Event Hosting	3	2
3	1	Event Organization	2	2
4	1	Photography And Video	2	1

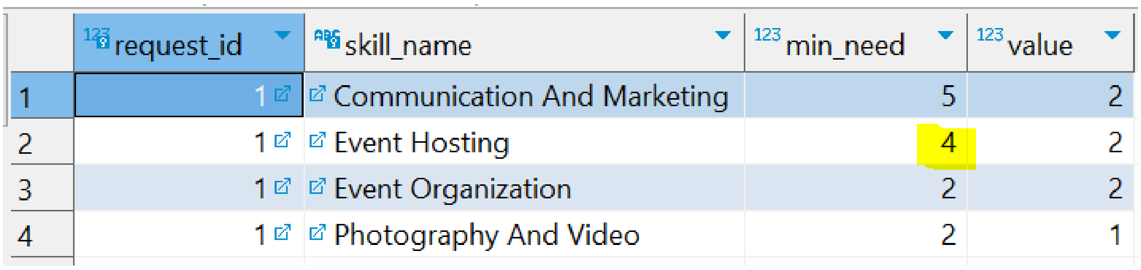
```
select * from request_skill rs
where request_id = 1;
```



select * from request r where id = 1

	id	beneficiary_id	title	number_of_volunteers	priority_value
1	1	3	work in team needed	14	1

```
/* Now we will update min_need to 4 */

UPDATE request_skill
SET min_need = 4
WHERE request_ID = 1 AND skill_name = 'Event Hosting';
```



	request_id	skill_name	min_need	value
1	1	Communication And Marketing	5	2
2	1	Event Hosting	4	2
3	1	Event Organization	2	2
4	1	Photography And Video	2	1

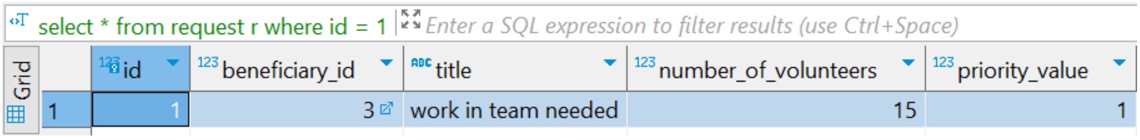
After the trigger:

Since request 1 has two ‘unskilled’ volunteers (= 14 – 5 – 3 – 2 – 2), after the trigger, we should have the following results:

- New sum(min_need of all requesting skills) = 5 + 4 + 2 + 2 = 13
- number of unskilled volunteers = 2

Hence, 'number_of_volunteers' = 13 + 2 = 15

```
-- Check if the trigger has updated the request table
SELECT * FROM request WHERE ID = 1;
```



```
select * from request r where id = 1
```

	id	beneficiary_id	title	number_of_volunteers	priority_value
1	1	3	work in team needed	15	1

5.3 Transactions

5.3.1 Transaction 1 - Optional

5.3.1.1 Code Explanation

Advanced - Transaction 1

The query includes the below steps:

Step 1: Creation of volunteer_assignment Table:

The query utilizes the CREATE TABLE IF NOT EXISTS statement to create the volunteer_assignment table. This ensures that the table is created if it does not already exist. The table includes fields such as request_id, volunteer_id, and is_accepted, with the PRIMARY KEY constraint defined on the combination of request_id and volunteer_id.

Step 2: Definition of assign_volunteer Function:

The function assign_volunteer is defined using the CREATE OR REPLACE FUNCTION statement. This function automates the process of assigning volunteers to requests.

Step 3: Initialization and Variable Declaration:

Within the function, variables are initialized and declared using the DECLARE keyword to store critical information, such as the total number of volunteers, the current date, and the registration date for the request.

Step 4: Validity Check for Request ID:

The function employs the IF NOT EXISTS condition within a SELECT statement to check if the provided request ID exists in the request table. If the request ID does not exist, it raises an exception using the RAISE EXCEPTION statement to notify the user.

Step 5: Skill-Based Volunteer Assignment:

To assign volunteers with matching skills, the function utilizes a nested FOR loop along with various JOIN conditions to iterate through requesting skills and select volunteers with matching skills.

Step 6: General Volunteer Assignment:

Remaining volunteers, not matched to specific skills, are assigned using a separate loop and JOIN conditions.

Step 7: Calculation of Assigned Volunteers:

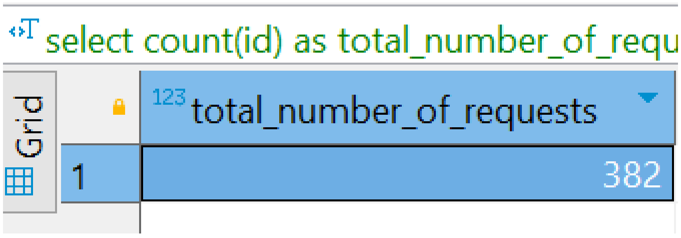
The function calculates the total number of assigned volunteers for the request using the SELECT INTO statement.

Step 8: Transaction Handling:

Finally, the function includes transaction handling logic using IF statements to check conditions for committing or rolling back the transaction based on the current date, registration date, and the fulfillment of volunteer requirements. It utilizes RAISE NOTICE and RAISE EXCEPTION statements to provide feedback and handle exceptions accordingly.

5.3.1.2 Testing & Results

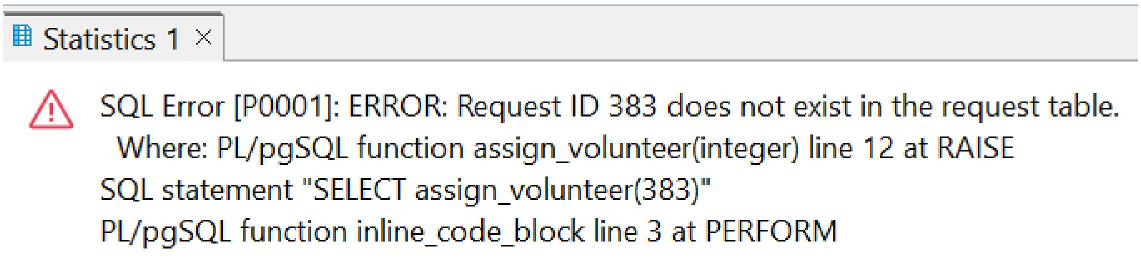
```
/* First, we tested with the request_id that does not exists in 'Request' table.
 * Currently, we have only 382 requests.
 */

select count(id) as total_number_of_requests
from request r;
```



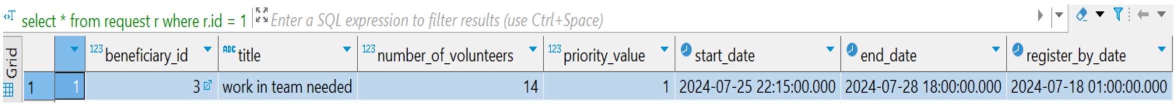
```
select count(id) as total_number_of_requ
```

	total_number_of_requests
1	382

```sql
-- Now if we enter id = 383, the transaction should be rolled back with the error meessage:
DO $$
BEGIN
   PERFORM assign_volunteer(383);
END $$;
```



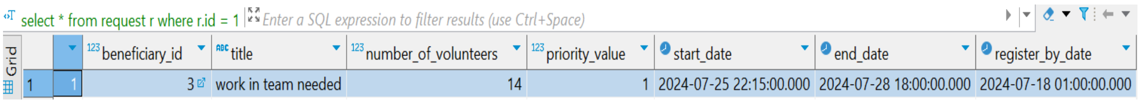
```
SQL Error [P0001]: ERROR: Request ID 383 does not exist in the request table.
  Where: PL/pgSQL function assign_volunteer(integer) line 12 at RAISE
SQL statement "SELECT assign_volunteer(383)"
PL/pgSQL function inline_code_block line 3 at PERFORM
```



select * from request r where r.id = 1
Enter a SQL expression to filter results (use Ctrl+Space)
Grid

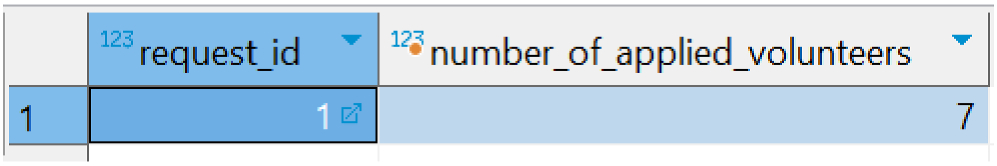
		beneficiary_id	title	number_of_volunteers	priority_value	start_date	end_date	register_by_date
1	1	3	work in team needed	14	1	2024-07-25 22:15:00.000	2024-07-28 18:00:00.000	2024-07-18 01:00:00.000

```sql
/* Scenario 1: The registration deadline is not past AND minimum volunteer requirement is not met.
 * We tested with Request 1 */
select *
from request r
where r.id = 1;

-- Number of applied volunteers (valid application only):
select request_id, count(volunteer_id) as number_of_applied_volunteers
from volunteer_application
where request_id = 1 and is_valid = 1
group by request_id;
```



select * from request r where r.id = 1
Enter a SQL expression to filter results (use Ctrl+Space)
Grid

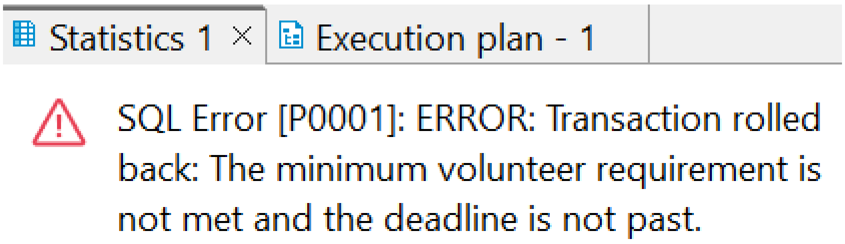
		beneficiary_id	title	number_of_volunteers	priority_value	start_date	end_date	register_by_date
1	1	3	work in team needed	14	1	2024-07-25 22:15:00.000	2024-07-28 18:00:00.000	2024-07-18 01:00:00.000



	request_id	number_of_applied_volunteers
1	1	7

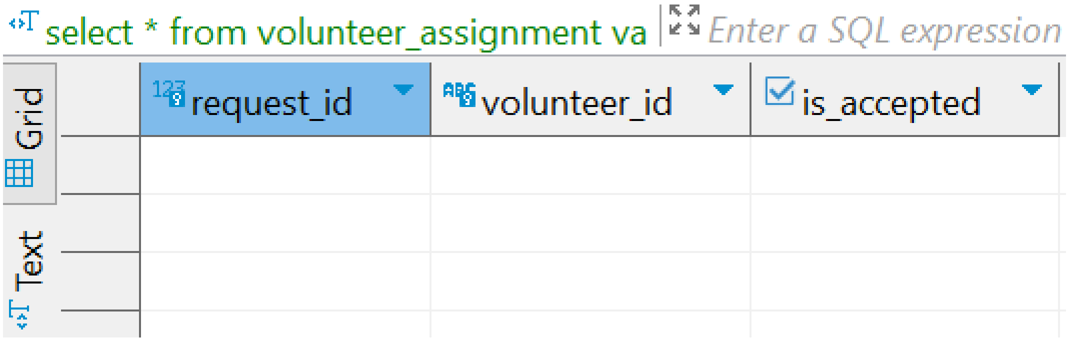
```sql
/* Request 1: Its 'register_by_date' (deadline) is not past, and only 7 volunteers applied ( < min_need = 14).
 * Hence the transaction should be rolled back.
 */
DO $$
BEGIN
   PERFORM assign_volunteer(1);
```

```
END $$;
```



SQL Error [P0001]: ERROR: Transaction rolled back: The minimum volunteer requirement is not met and the deadline is not past.

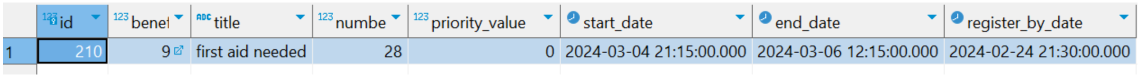
```
-- Re-check Volunteer_assignment table:
select *
from volunteer_assignment va;
```



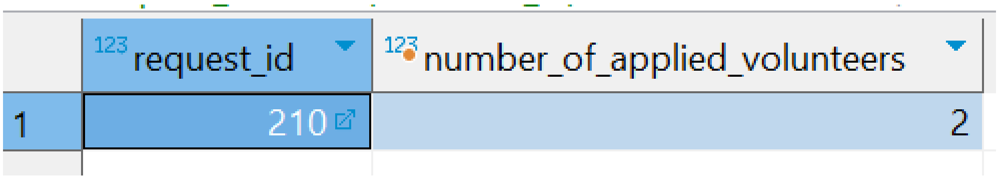
select * from volunteer_assignment va

	request_id	volunteer_id	is_accepted

```
/* Scenario 2 The registration deadline is past AND minimum volunteer requirement is
not met.
 * We tested with Request ID 210. */

select *
from request r
where r.id = 210;

-- Number of applied volunteers (valid application only):
select request_id, count(volunteer_id) as number_of_applied_volunteers
from volunteer_application
where request_id = 210 and is_valid = 1
group by request_id;
```



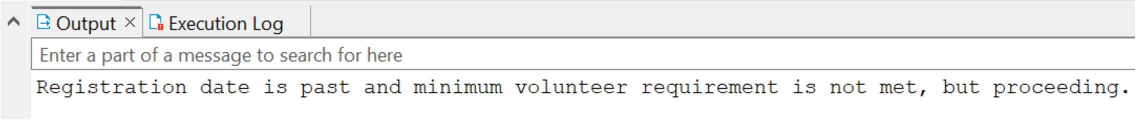
	id	benei	title	numbe	priority_value	start_date	end_date	register_by_date
1	210	9	first aid needed	28	0	2024-03-04 21:15:00.000	2024-03-06 12:15:00.000	2024-02-24 21:30:00.000



	request_id	number_of_applied_volunteers
1	210	2

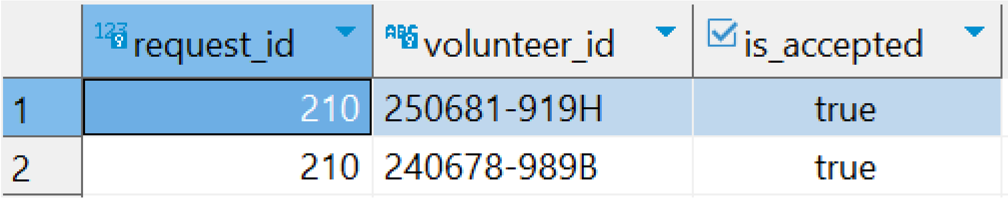
```
/* Deadline is past, and only 2 volunteers applied, which is lower than number of
volunteers needed (28).
```

```
However, the transaction should commit as we accept the number of volunteers as they
are */
DO $$
BEGIN
   PERFORM assign_volunteer(210);
END $$;
```



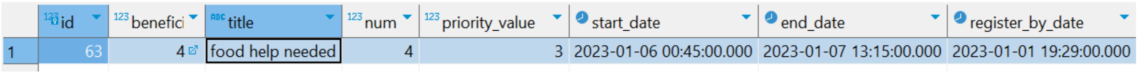
```
Registration date is past and minimum volunteer requirement is not met, but proceeding.
```

```
-- Re-check Volunteer_assignment table:
select *
from volunteer_assignment va
where request_id = 210;
```

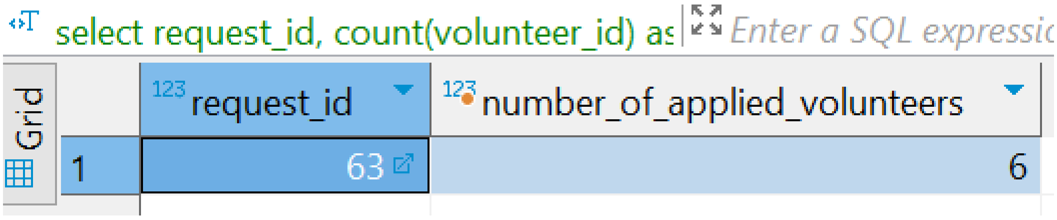


	request_id	volunteer_id	is_accepted
1	210	250681-919H	true
2	210	240678-989B	true

```
/* Scenario 3 the register_by_date is not past or the minimum number of volunteers is
meet.
 * We tested with Request ID 63. */

select *
from request r
where r.id = 63;

-- Number of applied volunteers (valid application only):
select request_id, count(volunteer_id) as number_of_applied_volunteers
from volunteer_application
where request_id = 63 and is_valid = 1
group by request_id;
```



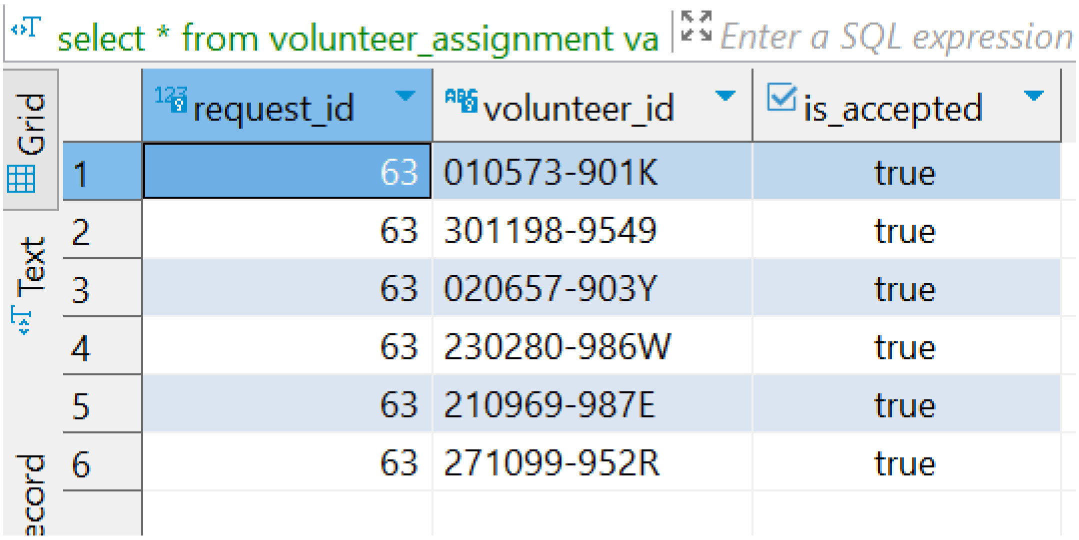
	id	benefici	title	num	priority_value	start_date	end_date	register_by_date
1	63	4	food help needed	4	3	2023-01-06 00:45:00.000	2023-01-07 13:15:00.000	2023-01-01 19:29:00.000



```
select request_id, count(volunteer_id) as
```

	request_id	number_of_applied_volunteers
1	63	6

```
/* Deadline is past (01.01.2023) but the minimum number of volunteers is meet (there
are totally 6 volunteers applied while the minimum number of volunteers required is
4).
 * Hence, the transaction should commit */
DO $$
BEGIN
   PERFORM assign_volunteer(63);
END $$;

-- Re-check Volunteer_assignment table:
select *
from volunteer_assignment va
where request_id = 63;
```



```
select * from volunteer_assignment va
```

	request_id	volunteer_id	is_accepted
1	63	010573-901K	true
2	63	301198-9549	true
3	63	020657-903Y	true
4	63	230280-986W	true
5	63	210969-987E	true
6	63	271099-952R	true

5.3.2 Transaction 2 - Free choice

5.3.2.1 Purpose

This transaction is designed to notify volunteers one week before the application deadline ('register_by_date' of the requests they have applied for). This is achieved by inserting notifications into the notifications table for volunteers with valid applications.

This transaction includes the below steps:

- Begin the Transaction: Start the transaction to ensure that all steps are executed as a single unit of work. If any step fails, the transaction can be rolled back to maintain data integrity.
- Identify Relevant Requests: Query the request table to find requests with a register_by_date that is one week from today. This ensures that we are targeting requests that are close to their registration deadlines.

- Identify Volunteers with Valid Applications: For each identified request, query the volunteer_application and volunteer tables to find volunteers who have applied and have valid applications. This step ensures that only valid and interested volunteers are notified.
- Insert Notifications: Insert this message into the notifications table along with the volunteer ID, email, request ID, and register_by_date. This step ensures that volunteers receive timely reminders about upcoming registration deadlines.
- Commit the Transaction/Rollback on Error: If all steps are successful, commit the transaction to make the changes permanent. If any error occurs during the process, rollback the transaction.

Below are some benefits of this transaction:

- Timely Reminders: Volunteers are notified a week before the registration deadline, giving them adequate time to take any necessary action. This could include completing/updating any missing application details.
- Increased Participation: Timely notifications can lead to increased participation as volunteers are less likely to forget about their application. This can help to meet the required number of volunteers for each request.
- Better Volunteer Experience: Volunteers are more likely to have a positive experience if they are kept informed about important deadlines. This can lead to higher satisfaction and potentially more long-term engagement with the Finnish Red Cross organization.

5.3.2.2 Code Explanation

Advanced - Transaction 2

The query includes below steps:

Step 1: Creation of ‘notifications’ Table:

The query creates a table named notifications using the CREATE TABLE IF NOT EXISTS statement. This table is designed to store notification details, including an auto-incremented ID, volunteer_ID, volunteer_email, request_ID, register_by_date, message, and noti_send_date fields.

Step 2: Creation of ‘notify_volunteers_before_registration’ function:

The function notify_volunteers_before_registration is defined to automate the process of notifying volunteers before the registration deadline of requests.

Function Overview:

- Initialization and Variable Declaration: The function initializes record variables req and app to hold request and volunteer application details.
- Request Loop: The function loops through requests that have a registration deadline 7 days from the current date.
- Application Loop: For each qualifying request, the function loops through valid volunteer applications associated with that request.
- Notification Message Creation: It creates a notification message reminding volunteers about the registration deadline for the respective request. This message includes the request ID and registration date formatted as 'DD.MM.YY'.
- Insertion into notifications Table: The function inserts the notification message along with volunteer and request details into the notifications table.

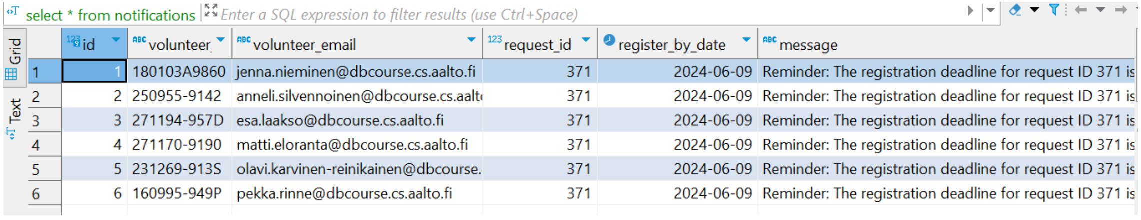
Operators/Functions Used:

- CREATE TABLE IF NOT EXISTS: Creates the notifications table if it does not already exist.
- FOR Loop: Iterates over the qualifying requests and volunteer applications.
- SELECT INTO Statement: Assigns values from the query result to record variables.
- INSERT INTO Statement: Inserts notification details into the notifications table.
- TO_CHAR Function: Formats the registration date as 'DD.MM.YY' for inclusion in the notification message.
- CURRENT_DATE Function: Retrieves the current date for comparison and default value assignment.
- INTERVAL Operator: Calculates the registration deadline 7 days from the current date.
- JOIN Clause: Joins the Volunteer table to acquire volunteer email addresses associated with their applications.

5.3.2.3 Testing & Results

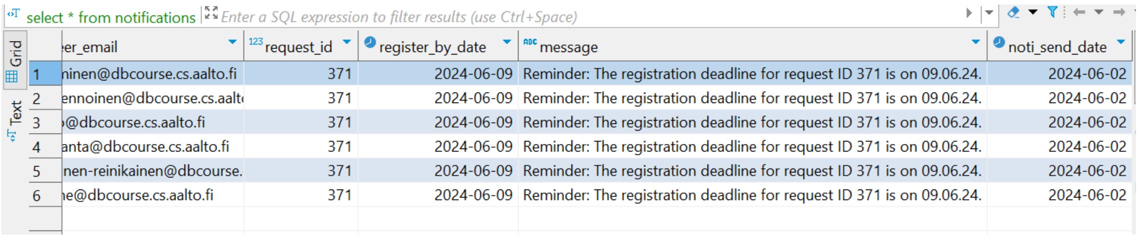
```
-- Test the function: Today is 02.06.2024 - so all volunteers applying for requests
having 'register_by_date' = 09.06.2024 must be notified.

DO $$
BEGIN
```

```
  PERFORM notify_volunteers_before_registration();
END $$;

-- Result:
select * from notifications;
```



select * from notifications | Enter a SQL expression to filter results (use Ctrl+Space)

	id	volunteer	volunteer_email	request_id	register_by_date	message
1	1	180103A9860	jenna.nieminen@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is
2	2	250955-9142	anneli.silvennoinen@dbcourse.cs.aalt	371	2024-06-09	Reminder: The registration deadline for request ID 371 is
3	3	271194-957D	esa.laakso@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is
4	4	271170-9190	matti.eloranta@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is
5	5	231269-913S	olavi.karvinen-reinikainen@dbcourse.	371	2024-06-09	Reminder: The registration deadline for request ID 371 is
6	6	160995-949P	pekka.rinne@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is



select * from notifications | Enter a SQL expression to filter results (use Ctrl+Space)

	er_email	request_id	register_by_date	message	noti_send_date
1	ninen@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02
2	ennoinen@dbcourse.cs.aalt	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02
3	@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02
4	anta@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02
5	nen-reinikainen@dbcourse.	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02
6	ne@dbcourse.cs.aalto.fi	371	2024-06-09	Reminder: The registration deadline for request ID 371 is on 09.06.24.	2024-06-02

5.4 Data Analysis

The below file contains all code for data analysis part.

Group 20-project
part 2-data analysis.

5.4.1 Question 1

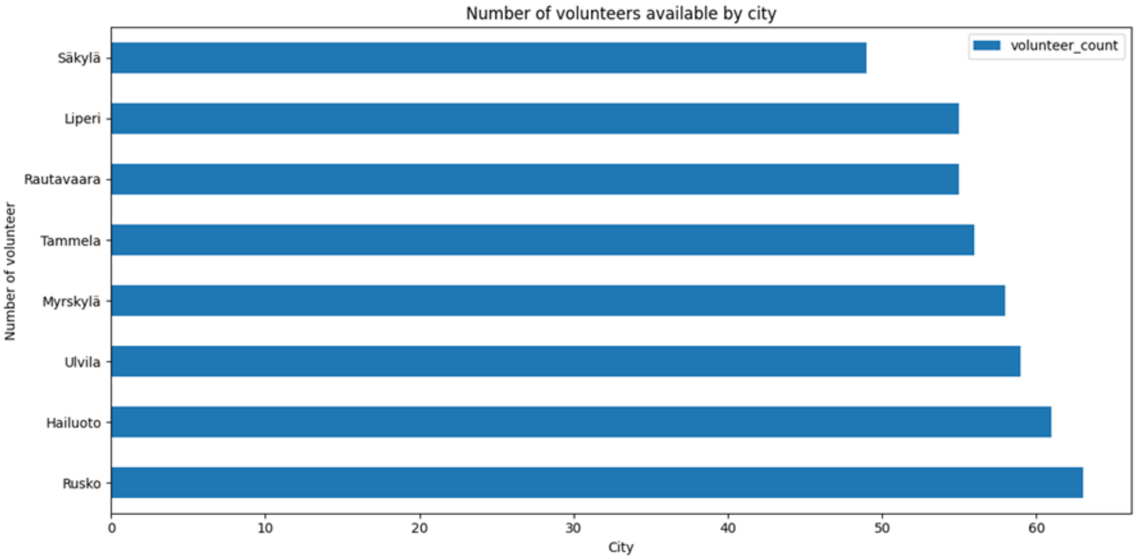
5.4.1.1 Code explanation

In this code for this question, three queries are performed to analyse and visualize volunteer data by city including Query for Volunteer Count by City, Query for Volunteer Applications by City, and Combined Query for Volunteers and Applications by City.

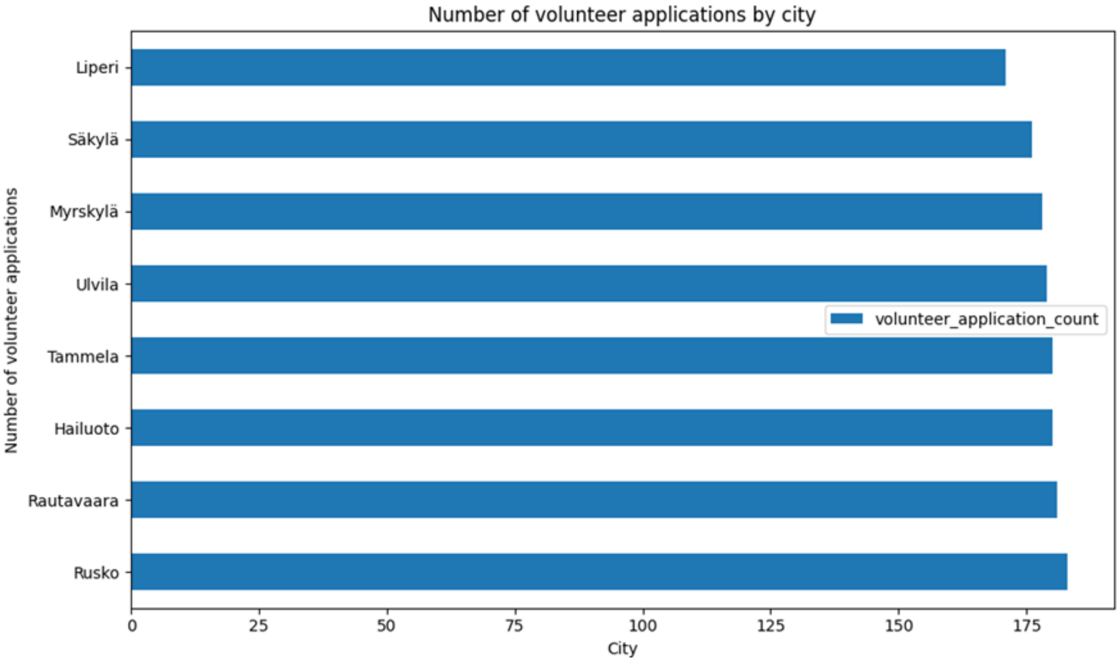
Then the query result is stored in corresponding data frame for visualization.

Note: The Combined Query for Volunteers and Applications by City is plotted as stacked bar chart using seaborn for better visualization. The data frame is melted to a long format for easier plotting with Seaborn.

5.4.1.2 Result

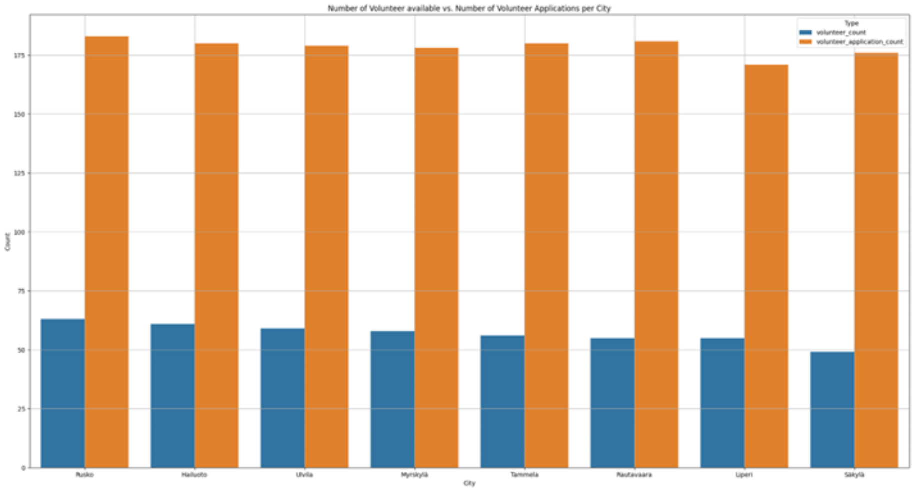


- The city with the most (top 2 volunteers available) are Rusko and Hailuoto.
- The city with the least (bottom 2 volunteers available) are Säkylä and Liperi.



- The city with the most (top 2) volunteer applications are Rusko and Rautavaara.
- The city with the least (bottom 2) volunteer applications are Säkylä and Liperi.

The below chart shows Number of Volunteer available vs. Number of Volunteer Applications per City.



5.4.2 Question 2

5.4.2.1 Assumption

Below scoring system for volunteer with valid application with corresponding weight is used to suggest top 5 candidates for each request.

Maximum matching score (80) is the total sum of maximum scores for each component including Skill Matches (40), Travel Readiness (20), and Interest Matches (20).

- Skill Matches Score is calculated based on the number of matching skills.
- Travel Readiness Score is calculated based on the volunteer's willingness to travel and the proximity to the request city.
- Interest Matches Score is calculated based on whether the volunteer's interests include the best matching interest for the request.

Volunteer matching score percentage is normalized by dividing volunteer matching score by the maximum possible score (80).

Below is the detailed rule how each component in matching score is determined.

- Skill Matches (max 40 points, min 0 points):
  - Full match: if a volunteer has all three required skills, they receive the highest score (40 points).

    - Partial match: if a volunteer has one or two of the required skills, their score is calculated based on the proportion of skills matched.
    - No match: If a volunteer has none of the required skills, they receive lowest score (0 points).

- Travel Readiness Score:
    - Highest score (20 points) if the volunteer and request cities are the same.
    - High score (20 points) if the request city is within volunteer city range and the volunteer is willing to travel (at least more than 10 minutes).
    - Moderate score (10 points) if the request city is within volunteer city range but the volunteer is not willing to travel (at least more than 10 minutes).
    - Moderate (10 points) to low score (5 points) based on the willingness to travel if the request city is outside the reasonable range. To be specific, volunteer is willing to travel long distance if the volunteer is willing to travel more than 120 minutes.
    - Zero score if the volunteer is not willing to travel at all.

- Interest Matches score: using cosine similarity function, request title is compared with all interest names, and the interest's name that has the highest similarity score above a set threshold (0.49) is selected and received the highest matching score. If the volunteer has an interest that matches the best interest for the request, they receive highest score (20 points), if the volunteer does not have an interest that matches the best interest for the request, they receive lowest score (0 points).

5.4.2.2 Code explanation

This code preprocesses data and calculates matching scores between volunteers and requests based on various criteria, ultimately suggesting the top candidates for each request.

Summary of the main actions:

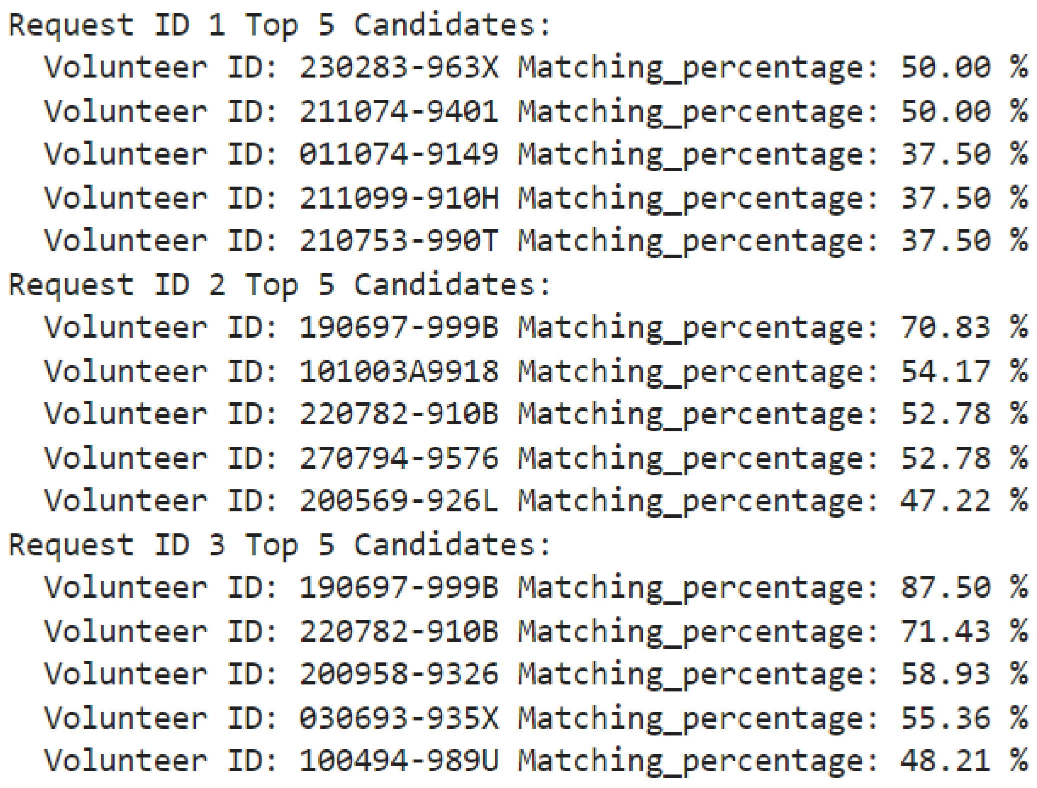
1. Data Preprocessing:
    - Normalize interest names in interest_df and interest_assignment_df to lowercase.
    - Remove "needed" from the end of strings in the title column of request_df.
    - Select only volunteer with valid application.

2. Defines a function calculate_similarity to compute cosine similarity between two sentences, used to match interests and request titles.
3. Calculate Score according to scoring scheme.
4. Suggest the top 5 volunteer candidates for each request based on the total scores and display result.

5.4.2.3 Result

According to this system, the suggested top 5 candidates result matches with candidates found in past questions (to be specific query 2 in part 4.2 find volunteers whose skill assignments match the requesting skills).

Top 5 candidates result using scoring system which matches the list of candidates in the result in query 2 part 4.2. However, the position of candidates is different due to the difference in scoring system. In this system, the matching skill weights 50%, query 2 in part 4.2 weights the matching skill 100%.



```
Request ID 1 Top 5 Candidates:
  Volunteer ID: 230283-963X Matching_percentage: 50.00 %
  Volunteer ID: 211074-9401 Matching_percentage: 50.00 %
  Volunteer ID: 011074-9149 Matching_percentage: 37.50 %
  Volunteer ID: 211099-910H Matching_percentage: 37.50 %
  Volunteer ID: 210753-990T Matching_percentage: 37.50 %
Request ID 2 Top 5 Candidates:
  Volunteer ID: 190697-999B Matching_percentage: 70.83 %
  Volunteer ID: 101003A9918 Matching_percentage: 54.17 %
  Volunteer ID: 220782-910B Matching_percentage: 52.78 %
  Volunteer ID: 270794-9576 Matching_percentage: 52.78 %
  Volunteer ID: 200569-926L Matching_percentage: 47.22 %
Request ID 3 Top 5 Candidates:
  Volunteer ID: 190697-999B Matching_percentage: 87.50 %
  Volunteer ID: 220782-910B Matching_percentage: 71.43 %
  Volunteer ID: 200958-9326 Matching_percentage: 58.93 %
  Volunteer ID: 030693-935X Matching_percentage: 55.36 %
  Volunteer ID: 100494-989U Matching_percentage: 48.21 %
```

5.4.3 Question 3

5.4.3.1 Assumption

We assume that valid requests are requests where the range of start date and end date is in the month considered.

5.4.3.2 Code explanation

The code analyses and visualizes the number of valid requests and valid volunteer applications per month, as well as the deviation between them.

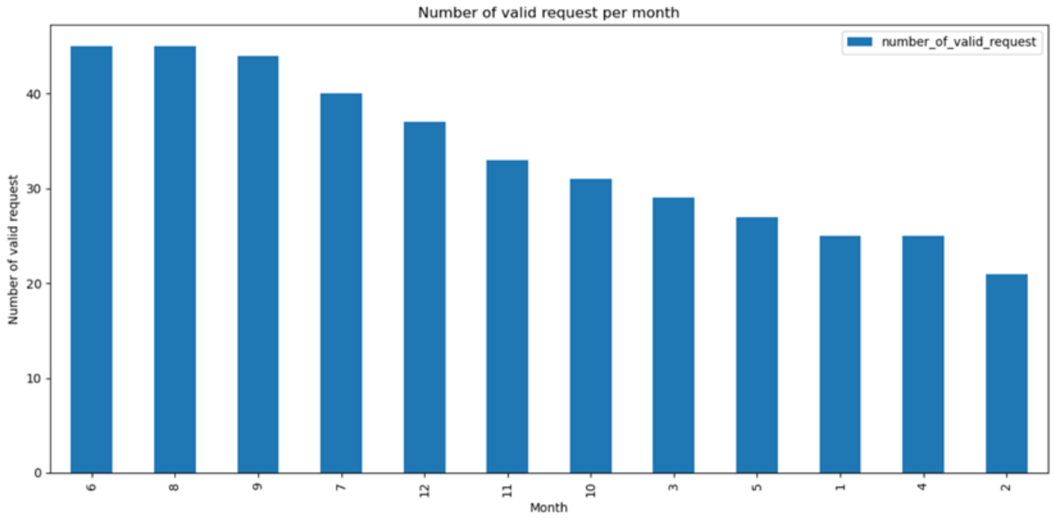
Summary of the main actions:

1. Data Preprocessing for Requests: Extracts relevant columns, then defines get_valid_months to determine all valid months for each request and stores results in a new DataFrame (valid_requests_per_month_df)
2. Count and Plot Valid Requests per Month
3. Data Preprocessing for Volunteers and Count and Plot Valid Volunteer Applications per Month
4. The result of combination of Requests and Volunteer Applications is plotted as stacked bar chart using seaborn for better visualization. The data frame is melted to a long format for easier plotting with Seaborn.
5. Computes and plots a heatmap showing the correlation between the number of requests and volunteer applications per month.

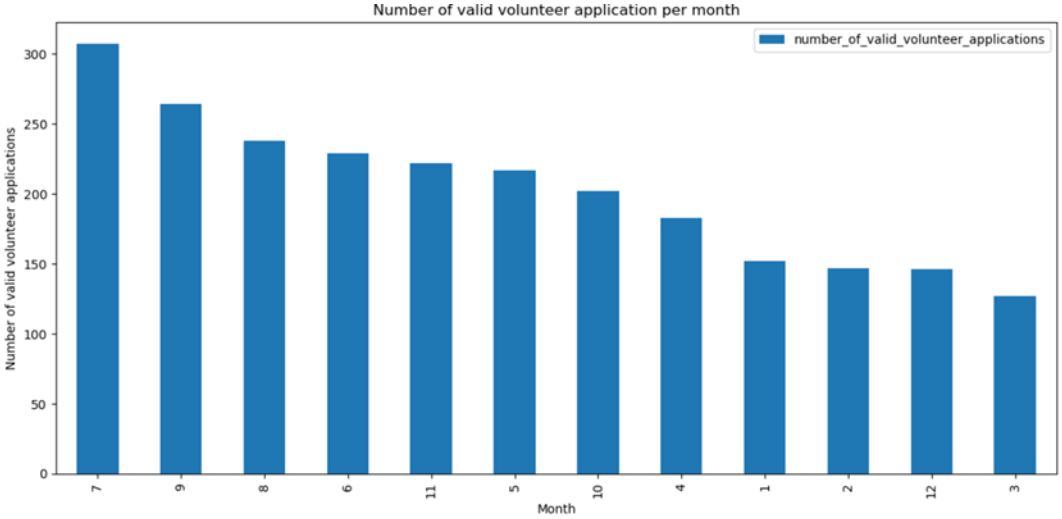
5.4.3.3 Result

This comprehensive analysis helps identify patterns and trends in volunteer activity and request needs throughout the year.

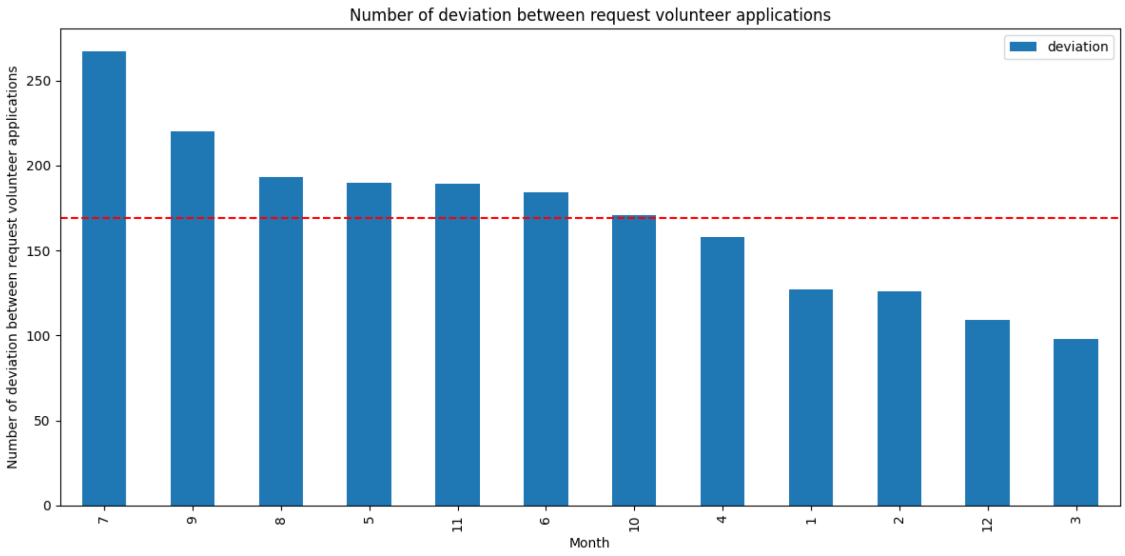
It seems that there are more requests and applications during the summer months and less requests and applications during the winter months and spring months.



- Months with the most valid volunteer applications (top 2) are 6 (June) and 8 (August).
- Months with the least valid volunteer applications (bottom 2) are 4 (April) and 2 (February).

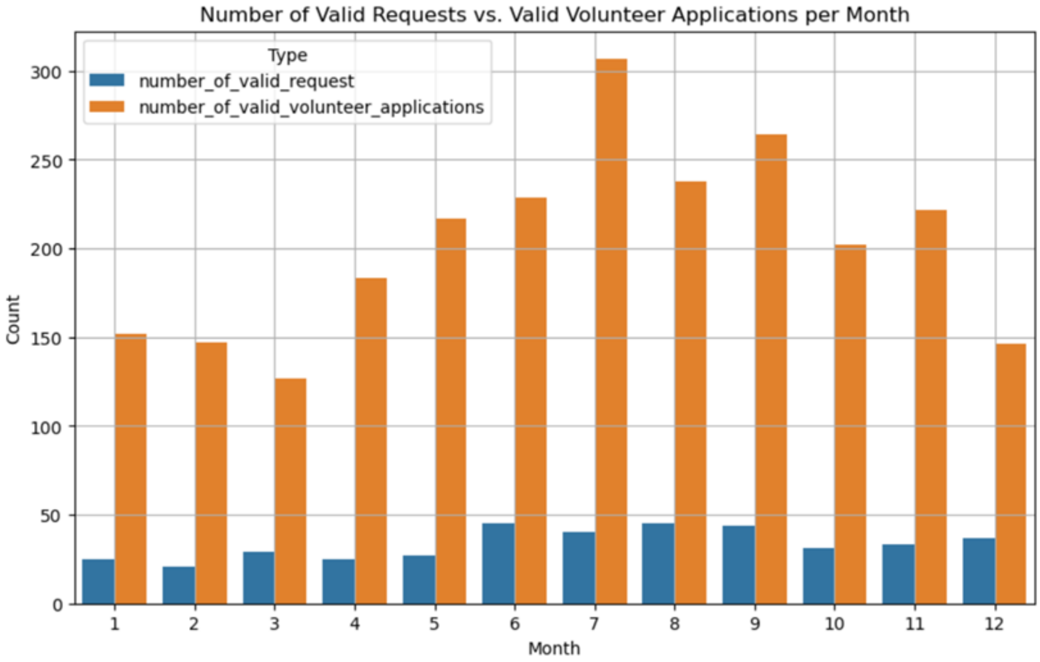


- Months with the most (highest number of) valid request (top 2) are 7 (July) and 9 (September).
- Months with the least (lowest number of) valid request (bottom 2) are 12 (December) and 3 (March).



- Months have the most different between the requests and volunteers for each month (top 2) are 7 (July) and 9 (September).
- Months have the least different between the requests and volunteers for each month (bottom 2) are 12 (December) and 3 (March).

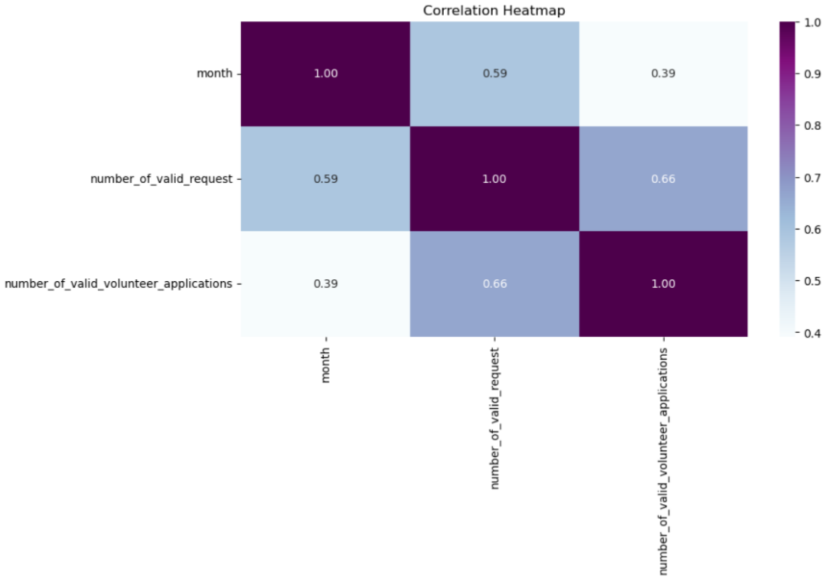
Overview of Valid Requests vs. Valid Volunteer Applications per Month:



- July is the month with highest number of valid volunteer applications.
- June to September is the period with highest number of valid volunteer applications.

- Also, there is more valid requests in period from June to September.

Next, there is moderate correlation (due to correlation score lies between 0.5 and 0.7) between the time of the year and number of requests and volunteers according to below heatmap.



5.4.4 Question 4

5.4.4.1 Free choice analysis: Identifying High-Demand, Low-Supply Skills

Purpose: to prepare resource for skills that are in high demand and low supply

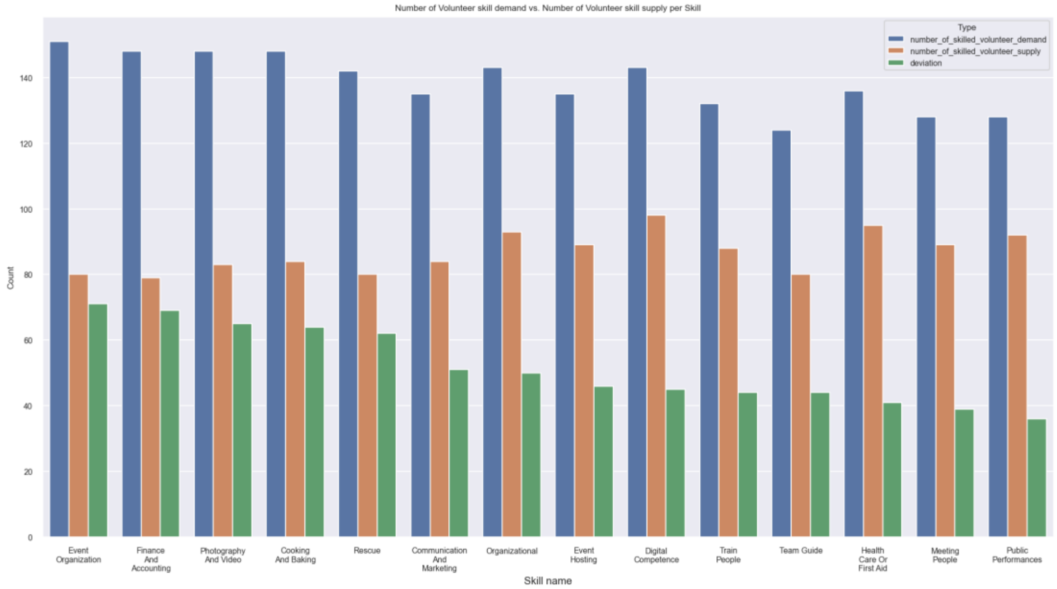
5.4.4.2 Code explanation

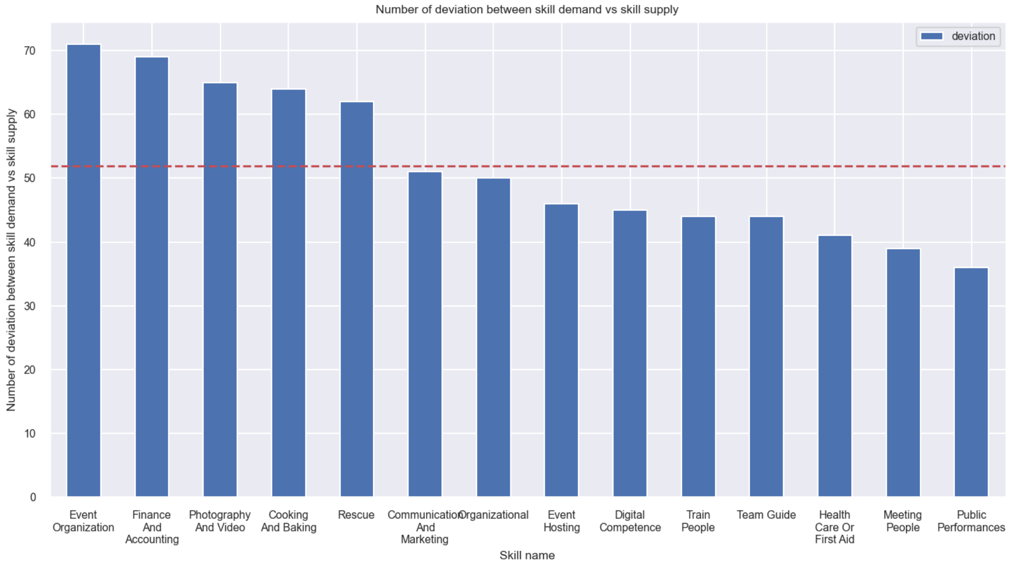
The code performs a free choice analysis to identify high-demand, low-supply skills among volunteers by comparing the number of volunteers with specific skills against the demand for those skills. Here's a summary of the main actions:

- Perform a Query to Determine Skill Demand and Supply and store the result to a data frame
- Melt the DataFrame to a long format suitable for Seaborn plotting and plot the data
- Define and apply a helper function wrap_labels to wrap long text labels on the x-axis to improve readability

5.4.4.3 Result

This analysis helps visualize the discrepancy between the demand and supply of volunteer skills, highlighting areas where there is a shortage of specific skills. The chart below shows that most skills are in shortage, especially Photography and Video, Event Organization, and Finance and Accounting.





6 Other optional additions

6.1 Trigger: Check 'up-to-20-application' requirement

6.1.1 Purpose

As outlined in the project description, each volunteer is allowed to submit up to 20 applications. To enforce this limit, it is crucial to create a trigger that prevents a volunteer from adding new applications if they already have 20 'open' applications. Implementing this trigger offers several significant benefits:

- Controlled volume: By limiting the number of applications per volunteer, we can prevent data overflow and maintain the integrity of the application data. This ensures that our database remains manageable and efficient.
- Preventing spam: This trigger helps to limit the potential for spam or abuse where a volunteer might otherwise apply to an excessive number of requests without genuine intent. This ensures that applications are meaningful and relevant.

Definition of 'Open' application: An 'open' application is defined by two criteria:

- Valid Application: The application must be valid, indicated by 'is_valid' = True (or = 1 as we converted True/False to 1/0).
- Current Request: The application must be for a request with a deadline that has not yet passed, indicated by 'register_by_date' >= current_date().

Our rationale for these criteria:

We chose to follow these criteria when counting the number of applications to enforce the limit. If we counted all applications regardless of their status or the request's deadline, the 'up-to-20-application' rule would be too strict. This could restrict potential volunteers from applying to more requests they are interested in. From the organization's perspective, the Red Cross wants volunteers to have more opportunities to apply and assist beneficiaries.

6.1.2 Code Explanation

Step 1: Creation of check_application_limit Function:

The script creates a function named check_application_limit to enforce a limit on the number of volunteer applications allowed per volunteer.

Function Overview:

- Initialization and Variable Declaration:

The function initializes a variable named application_count using the DECLARE statement to store the count of valid volunteer applications.

- Application Count Query:
  - It queries the database to count the number of valid applications made by the volunteer for requests with registration deadlines that have not passed, using the SELECT INTO statement to assign the count to the application_count variable.
  - Operators/Functions Used: SELECT INTO, COUNT, JOIN, CURRENT_DATE.
- Application Count Check:
  - If the count of valid applications for the volunteer exceeds or equals 20, an exception is raised to indicate that the volunteer has reached the application limit using the RAISE EXCEPTION statement.
  - Operators/Functions Used: RAISE EXCEPTION.
- Insertion Allowance:
  - If the application count is less than 20, the function allows the insertion of the new volunteer application by returning NEW.
  - Operators/Functions Used: RETURN NEW.

Step 2: Creation of application_limit_trigger Trigger:

The script creates a trigger named application_limit_trigger to execute the check_application_limit function before each insertion into the volunteer_application table.

Trigger Overview:

- Trigger Creation:

The CREATE TRIGGER statement creates a trigger named application_limit_trigger.

- Trigger Type and Timing:

The BEFORE INSERT clause specifies that the trigger is triggered before inserting a new row into the volunteer_application table.

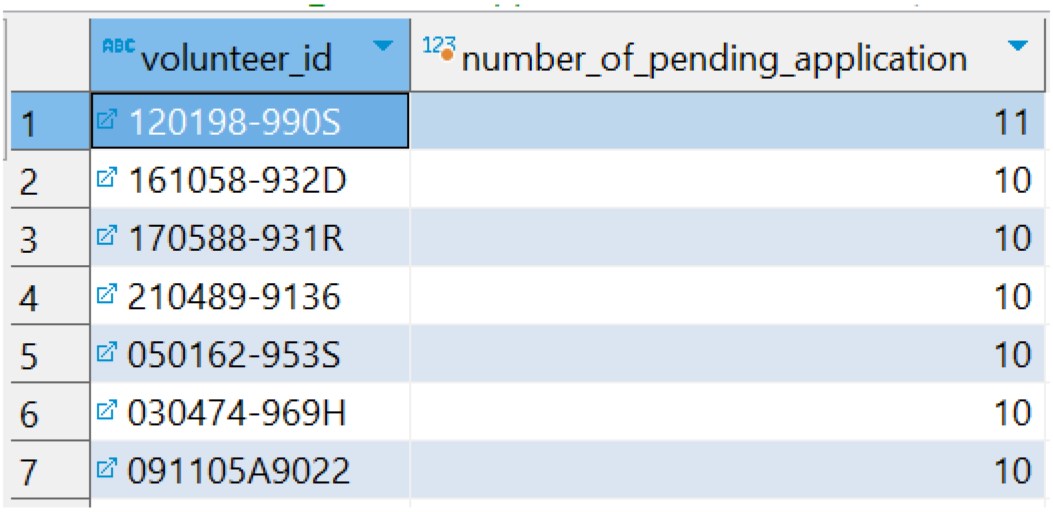
- Trigger Execution:

The trigger is configured to execute the check_application_limit function for each row insertion into the volunteer_application table using the EXECUTE FUNCTION clause.

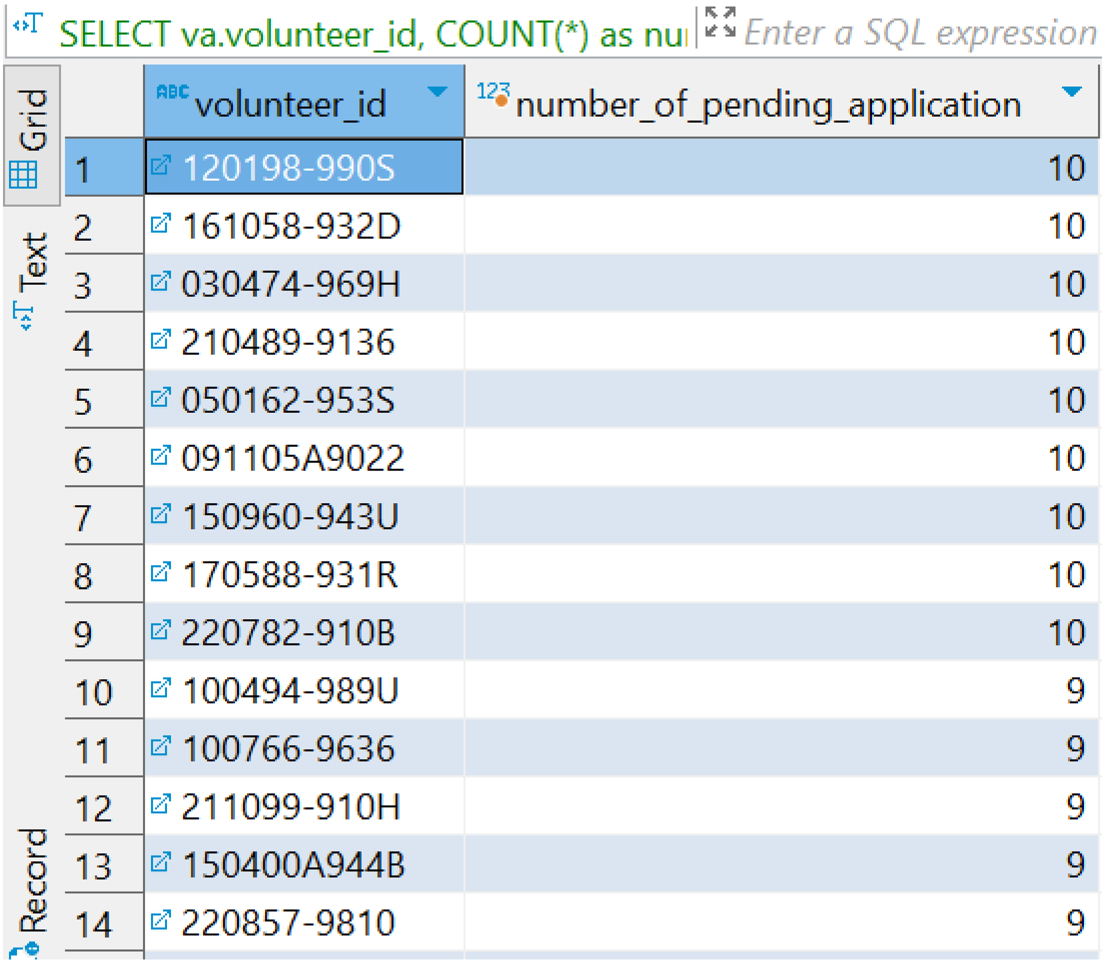
Operators/Functions Used: CREATE TRIGGER, BEFORE INSERT ON, FOR EACH ROW, EXECUTE FUNCTION.

6.1.3 Testing & Results

```
-- Test the trigger:
-- Show all volunteers with their open applications (valid application and with request
that is not overdue). Note: CURRENT_DATE = 07.06.2024 (= the date this test was
run)
SELECT va.volunteer_id, COUNT(*) as number_of_pending_application
   FROM volunteer_application va
   JOIN request r ON va.request_id = r.id
   WHERE va.is_valid = 1 AND r.register_by_date >= CURRENT_DATE
          group by va.volunteer_id
          order by number_of_pending_application DESC;
```



	volunteer_id	number_of_pending_application
1	120198-990S	11
2	161058-932D	10
3	170588-931R	10
4	210489-9136	10
5	050162-953S	10
6	030474-969H	10
7	091105A9022	10

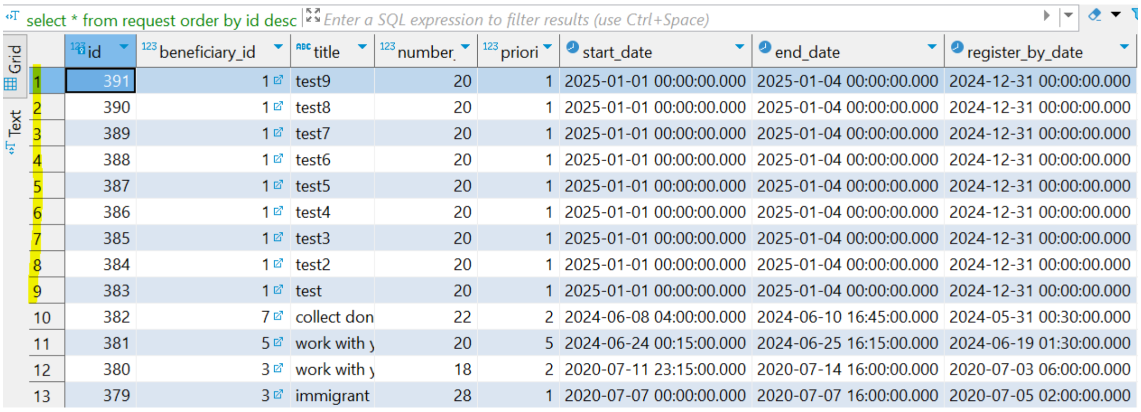


```
SELECT va.volunteer_id, COUNT(*) as nu
```

	volunteer_id	number_of_pending_application
1	120198-990S	10
2	161058-932D	10
3	030474-969H	10
4	210489-9136	10
5	050162-953S	10
6	091105A9022	10
7	150960-943U	10
8	170588-931R	10
9	220782-910B	10
10	100494-989U	9
11	100766-9636	9
12	211099-910H	9
13	150400A944B	9
14	220857-9810	9

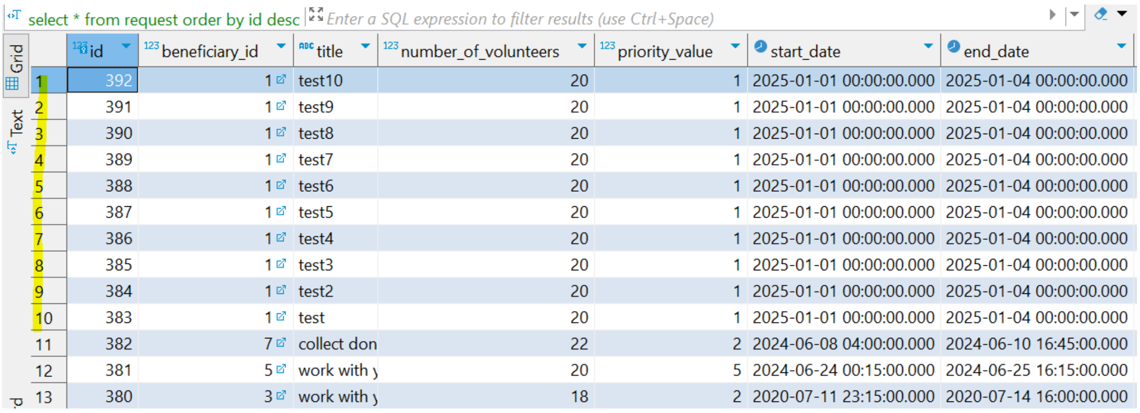
```
-- Since the volunteer id '120198-990S' is having 10 pending applications (the highest),
we will insert 10 more test applications.
/* However, to avoid messing up the original dataset, first we add 10 more request (id =
383, 384, .. 3911 as currently, the original dataset only has 382 requests.
 * Why we did so? Because we set the trigger that 1 volunteer cannot apply for the same
request if currently he/she has a valid application (see Additional Trigger 2)
 * We set their register_by_date in the future, so that their deadline are not past (i.e.
2024-12-31) and all applications submitted to these request is considered as 'pending'
 */

insert into request (beneficiary_id, title, number_of_volunteers, priority_value,
start_date,
   end_date, register_by_date) values
  (1, 'test', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test2', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test3', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test4', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test5', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test6', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test7', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test8', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test9', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31'),
  (1, 'test10', 20, 1, '2025-01-01', '2025-01-04', '2024-12-31')
 ;
```

```
-- Re-check if these requests are added:
select *
from request
order by id desc;
```



select * from request order by id desc | Enter a SQL expression to filter results (use Ctrl+Space)

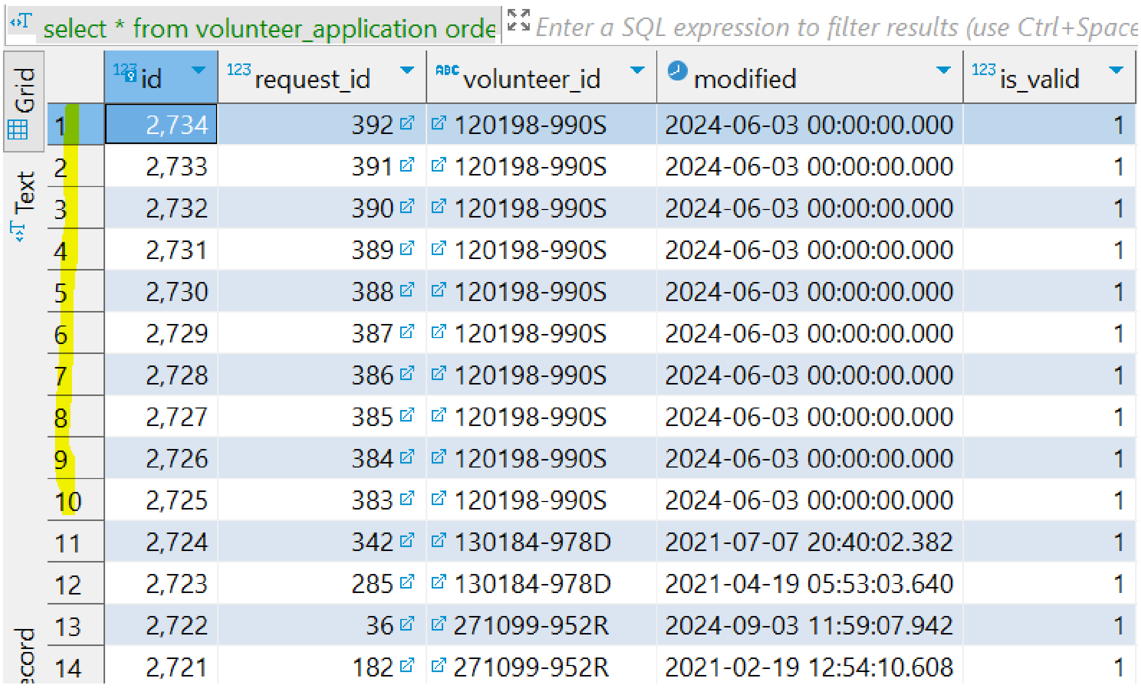
	id	beneficiary_id	title	number_	priori	start_date	end_date	register_by_date
1	391	1	test9	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
2	390	1	test8	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
3	389	1	test7	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
4	388	1	test6	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
5	387	1	test5	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
6	386	1	test4	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
7	385	1	test3	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
8	384	1	test2	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
9	383	1	test	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000	2024-12-31 00:00:00.000
10	382	7	collect don	22	2	2024-06-08 04:00:00.000	2024-06-10 16:45:00.000	2024-05-31 00:30:00.000
11	381	5	work with y	20	5	2024-06-24 00:15:00.000	2024-06-25 16:15:00.000	2024-06-19 01:30:00.000
12	380	3	work with y	18	2	2020-07-11 23:15:00.000	2020-07-14 16:00:00.000	2020-07-03 06:00:00.000
13	379	3	immigrant	28	1	2020-07-07 00:00:00.000	2020-07-07 16:00:00.000	2020-07-05 02:00:00.000



select * from request order by id desc | Enter a SQL expression to filter results (use Ctrl+Space)

	id	beneficiary_id	title	number_of_volunteers	priority_value	start_date	end_date
1	392	1	test10	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
2	391	1	test9	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
3	390	1	test8	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
4	389	1	test7	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
5	388	1	test6	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
6	387	1	test5	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
7	386	1	test4	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
8	385	1	test3	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
9	384	1	test2	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
10	383	1	test	20	1	2025-01-01 00:00:00.000	2025-01-04 00:00:00.000
11	382	7	collect don	22	2	2024-06-08 04:00:00.000	2024-06-10 16:45:00.000
12	381	5	work with y	20	5	2024-06-24 00:15:00.000	2024-06-25 16:15:00.000
13	380	3	work with y	18	2	2020-07-11 23:15:00.000	2020-07-14 16:00:00.000

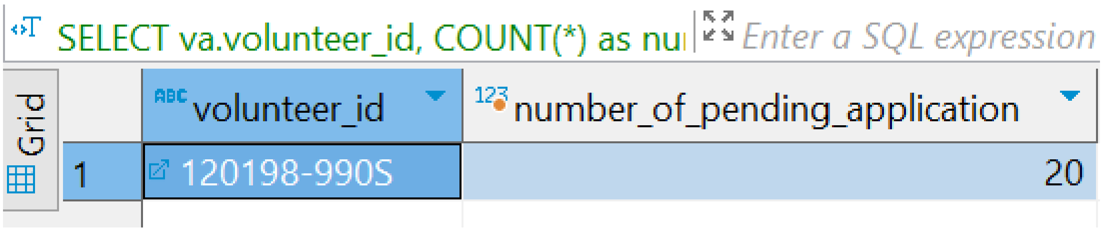
```
-- Now, the volunteer id '120198-990S' apply for these 9 newly-created request:
INSERT INTO volunteer_application (request_id, volunteer_id, modified, is_valid)
VALUES
(383, '120198-990S', '2024-06-03', 1),
(384, '120198-990S', '2024-06-03', 1),
(385, '120198-990S', '2024-06-03', 1),
(386, '120198-990S', '2024-06-03', 1),
(387, '120198-990S', '2024-06-03', 1),
(388, '120198-990S', '2024-06-03', 1),
(389, '120198-990S', '2024-06-03', 1),
(390, '120198-990S', '2024-06-03', 1),
(391, '120198-990S', '2024-06-03', 1),
(392, '120198-990S', '2024-06-03', 1);

-- Re-check if all aplications are added:
select * from volunteer_application
order by id desc;
```



```
select * from volunteer_application orde
```

	id	request_id	volunteer_id	modified	is_valid
1	2,734	392	120198-990S	2024-06-03 00:00:00.000	1
2	2,733	391	120198-990S	2024-06-03 00:00:00.000	1
3	2,732	390	120198-990S	2024-06-03 00:00:00.000	1
4	2,731	389	120198-990S	2024-06-03 00:00:00.000	1
5	2,730	388	120198-990S	2024-06-03 00:00:00.000	1
6	2,729	387	120198-990S	2024-06-03 00:00:00.000	1
7	2,728	386	120198-990S	2024-06-03 00:00:00.000	1
8	2,727	385	120198-990S	2024-06-03 00:00:00.000	1
9	2,726	384	120198-990S	2024-06-03 00:00:00.000	1
10	2,725	383	120198-990S	2024-06-03 00:00:00.000	1
11	2,724	342	130184-978D	2021-07-07 20:40:02.382	1
12	2,723	285	130184-978D	2021-04-19 05:53:03.640	1
13	2,722	36	271099-952R	2024-09-03 11:59:07.942	1
14	2,721	182	271099-952R	2021-02-19 12:54:10.608	1

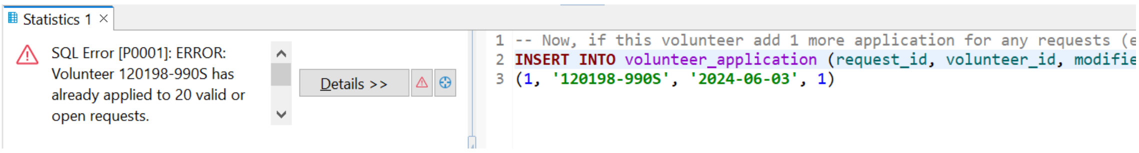
```
-- Re-check if volunteer ID '120198-990S' has now 20 'pending' application:
SELECT va.volunteer_id, COUNT(*) as number_of_pending_application
   FROM volunteer_application va
   JOIN request r ON va.request_id = r.id
   WHERE va.is_valid = 1 AND r.register_by_date >= CURRENT_DATE and
va.volunteer_id = '120198-990S'
          group by va.volunteer_id;
```



```
SELECT va.volunteer_id, COUNT(*) as nu
```

	volunteer_id	number_of_pending_application
1	120198-990S	20

```
-- Now, if this volunteer add 1 more application for any requests (ex: request ID = 1),
the trigger should be activated:
INSERT INTO volunteer_application (request_id, volunteer_id, modified, is_valid)
VALUES
(1, '120198-990S', '2024-06-03', 1);

-- Note: here we did not consider the scenario, where the volunteer set the application
as 'invalid' (in_valid = False), because in that sense, the application will not be counted.
```



SQL Error [P0001]: ERROR: Volunteer 120198-990S has already applied to 20 valid or open requests.

```
-- Now, if this volunteer add 1 more application for any requests (e
INSERT INTO volunteer_application (request_id, volunteer_id, modifie
(1, '120198-990S', '2024-06-03', 1)
```

```
-- Once done with the testing, we delete these test tuples, return to the original dataset:

delete from volunteer_application
where request_id > 382;

delete from request
where id > 382;

SELECT setval('volunteer_application_id_seq', (SELECT MAX(id) FROM
volunteer_application));
SELECT setval('request_id_seq', (SELECT MAX(id) FROM request));
```

6.2 Trigger: Prevent duplicate applications for the same request

6.2.1 Purpose

This trigger is designed to check whether the volunteer is currently having the valid application for the same request, before allowing the volunteer to submit the new application. If he/she already has the valid application for the request, the volunteer cannot apply to the same request. This trigger is necessary for below reasons:

- Data consistency and Avoid redundancy: It helps in maintaining consistent data by preventing duplicate active applications, which could otherwise lead to confusion and mismanagement. Also, it prevents redundant data entries that could lead to data bloat and inefficient queries.
- Optimized storage: This trigger prevents unnecessary use of storage resources by avoiding multiple records for the same application.
- Better user experience: From volunteers' point of view, volunteers will have a clear understanding of their application status, avoiding confusion caused by multiple active applications to the same request.

6.2.2 Code Explanation

Step 1: Creation of check_duplicate_application Function:

This code segment establishes a function named check_duplicate_application to forestall duplicate volunteer applications for the same request by the same volunteer.

- Duplicate Check: The function employs an IF EXISTS statement with a subquery to check for any existing valid application for the same request by the same volunteer.

- Exception Handling: If such an application already exists, the function raises an exception using the RAISE EXCEPTION statement, alerting about the duplication.
- Insertion Allowance: If no duplicate application is found, the function allows the insertion of the new volunteer application by returning NEW.
- Operators/Functions Used: IF EXISTS, SELECT, RAISE EXCEPTION, RETURN NEW.

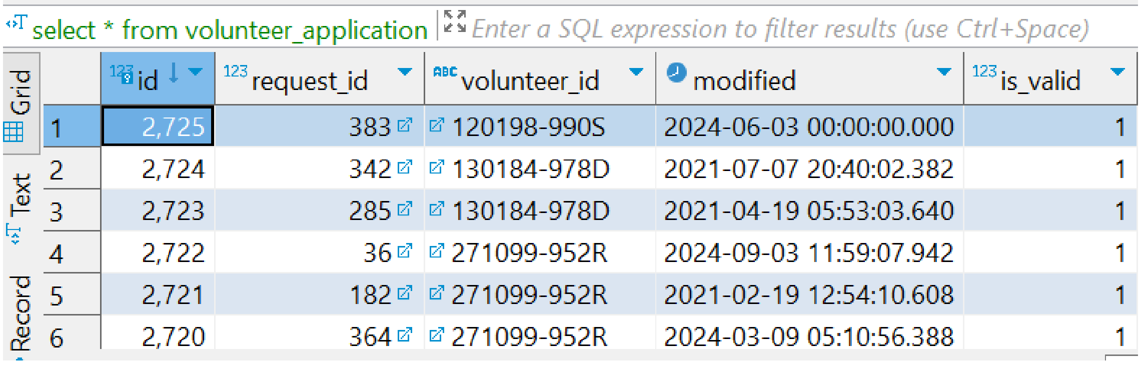
<u>Step 2: Creation of before_application_insert Trigger:</u>

This portion creates a trigger named before_application_insert to execute the check_duplicate_application function before each insertion into the volunteer_application table.

- Trigger Type and Timing: The trigger, specified as BEFORE INSERT, ensures that it fires before a new row is inserted into the volunteer_application table.
- Trigger Execution: Configured to execute the check_duplicate_application function for each row insertion, the trigger ensures that duplicate applications are caught in advance.
- Operators/Functions Used: CREATE TRIGGER, BEFORE INSERT ON, FOR EACH ROW, EXECUTE FUNCTION.

6.2.3 Testing & Results

```
-- Test the trigger:
/* To avoid messing up the original dataset, we use the new request (id = 383 as
created before).
 * Now for this request, volunteer with ID = '120198-990S' now submit 2 valid
applications.
 * The trigger should be activated.
 */
-- First application (is_valid = 1 indicates the application is valid). The first application
is added succesfully:
INSERT INTO volunteer_application (request_id, volunteer_id, modified, is_valid)
VALUES (383, '120198-990S', '2024-06-03', 1);
```

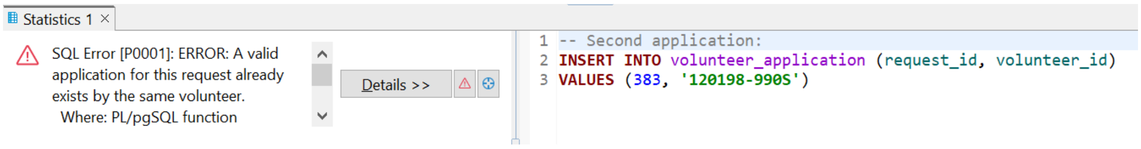


select * from volunteer_application

Enter a SQL expression to filter results (use Ctrl+Space)

	id	request_id	volunteer_id	modified	is_valid
1	2,725	383	120198-990S	2024-06-03 00:00:00.000	1
2	2,724	342	130184-978D	2021-07-07 20:40:02.382	1
3	2,723	285	130184-978D	2021-04-19 05:53:03.640	1
4	2,722	36	271099-952R	2024-09-03 11:59:07.942	1
5	2,721	182	271099-952R	2021-02-19 12:54:10.608	1
6	2,720	364	271099-952R	2024-03-09 05:10:56.388	1

```
-- Second application. Now the trigger is activated and prevent the volunteer to submit an application to the same request:
INSERT INTO volunteer_application (request_id, volunteer_id)
VALUES (383, '120198-990S');
```



SQL Error [P0001]: ERROR: A valid application for this request already exists by the same volunteer.
  Where: PL/pgSQL function

```
-- Second application:
INSERT INTO volunteer_application (request_id, volunteer_id)
VALUES (383, '120198-990S')
```

```
-- Once done with the testing, we delete these test tuples, return to the original dataset:

delete from volunteer_application
where request_id > 382;

delete from request
where id > 382;

SELECT setval('volunteer_application_id_seq', (SELECT MAX(id) FROM
volunteer_application));
SELECT setval('request_id_seq', (SELECT MAX(id) FROM request));
```

6.3 Another proposal

Dynamic matching of volunteers to beneficiaries based on demand requires a sophisticated approach that balances the ever-changing needs of diverse organizations with the available pool of volunteers.

By integrating clustering and predictive methods, our group suggests a model which aims to create a dynamic system that efficiently matches volunteer supply to the varied and changing demands of beneficiaries. Clustering beneficiaries into similar demand profiles allows for targeted and efficient resource planning, while time series predictions enable proactive management of volunteer resources. This dual approach minimizes the risks of oversupply and undersupply, ensuring that volunteer efforts are optimally utilized to meet the specific needs of each beneficiary cluster.

6.3.1 Clustering Methods

Employing K-means clustering to segment beneficiaries based on a comprehensive set of metrics to ensure that volunteer allocation aligns with the specific requirements of each beneficiary. The goal of the clustering exercise was to study the relationship between the vulnerability of a beneficiary and the volatility of demand for volunteers.

The k-means clustering approach was chosen due to the ease of experimentation and implementation of this method. The ideal number of clusters was not known prior to this analysis, and k-means clustering allowed for simple analysis to find the desired quantity of clusters.

6.3.1.1 Metric Selection and Preprocessing

When using K-means clustering to group beneficiaries, choosing the right metrics is crucial to effectively reflect their characteristics and demand patterns. Here are some metrics that can be used to group beneficiaries:

** Request Metrics:*

- Average Number of Volunteers Needed per Request: Indicates the typical volunteer demand for each beneficiary.
- Frequency of Requests: Measures how often each beneficiary submits requests, highlighting their consistency and volume of needs.
- Priority Value Distribution: Assesses the urgency and importance of requests, which could indicate more critical or routine operations.
- Typical Request Duration: Captures the average time span of requests (end_date - start_date), showing how long volunteers are needed.
- Lead Time (Register_by_Date): Measures how far in advance beneficiaries plan their volunteer needs, affecting the flexibility required in volunteer scheduling.

** Geographic Metrics:*

- **City_id**: Reflects the location of the beneficiaries, which is important for clustering those in close proximity or similar regional characteristics.
- Request Location Diversity: Captures how many different locations a beneficiary operates in, indicating operational scope.

** Beneficiary-Specific Metrics:*

- Beneficiary Type: Encodes the type of beneficiary (e.g., Hospital, Food Bank, Elderly Care) to see if certain types of group together naturally based on their volunteer demand patterns.
- Operational Scale: Measures the scale or size of operations, which could be inferred from the total number of volunteers requested over time or the geographic spread.

** Skill Requirements:*

- **Diversity of Skills Needed**: Assesses the variety of skills requested by the beneficiary, indicating more complex or specialized volunteer needs.
- **Skill Intensity**: Measures the average minimum need for each skill per request, reflecting the skill-specific demand.

* ***Temporal Metrics:***

- **Seasonality of Requests**: Identifies patterns in request submissions over time, such as peaks during certain seasons or consistent year-round needs.
- **Peak Demand Periods**: Captures specific times when volunteer demand is highest, which could affect grouping based on operational cycles.

* ***Demographic and Socioeconomic Factors:***

- **City Socioeconomic Indicators**: Incorporates data like average income, population density, or unemployment rates of the city_ID, which could impact the type and volume of volunteer needs.
- **Beneficiary Demographics**: Factors like the population served (e.g., age group for Youth Centres or Elderly Care) could provide insights into the type and frequency of volunteer requirements.

6.3.1.2 Normalize data

Before beginning the clustering analysis, a Min Max Scaler was applied to the variables to normalize the variables and ensure that each variable is equally scaled and weighted within the clustering analysis, preventing any single metric from disproportionately influencing the results.

6.3.1.3 Dimensionality Reduction (if needed):

Use techniques like PCA (Principal Component Analysis) to reduce the number of variables while preserving the variance, simplifying the clustering process.

6.3.1.4 Implementation

Apply the K-means algorithm to the preprocessed metrics to segment beneficiaries into clusters. The number of clusters (k) should be determined using methods like the Elbow Method or Silhouette Analysis, which evaluate the optimal k based on the compactness and separation of clusters.

6.3.1.5 Evaluation

Assess the coherence and practical relevance of the clusters. For instance, clusters should group beneficiaries with similar volunteer demands, geographic proximity, and operational patterns.

6.3.2 Predictive methods

Once beneficiaries are grouped, time series analysis can be employed to forecast volunteer demand trends, considering historical patterns and anticipated future changes. This predictive capability enables proactive volunteer recruitment and training, mitigating the risk of oversupply or undersupply.

Here's a detailed breakdown of the steps for applying time series analysis to predict future volunteer demand:

6.3.2.1 Historical Data Collection

- **Data Gathering**:
  - Collect comprehensive historical data on volunteer requests from the database. This includes data from the request table (request dates, number of volunteers needed, priority, etc.) and any additional relevant information from linked tables like request_skill and request_location.
  - Ensure the dataset spans a significant period to capture seasonal and cyclical trends (e.g., several years of data, if available).
- **Data Aggregation**:
  - Aggregate the data by relevant time intervals (e.g., daily, weekly, or monthly) depending on the granularity required. For instance, sum up the total number of volunteers needed per week or month.
  - If possible, tag each record with the cluster it belongs to, as determined from the clustering step.
- **Data Cleaning**:
  - Handle missing values or anomalies (e.g., sudden spikes that don't match historical patterns). This may involve interpolation, imputation, or removing outliers.
  - Ensure consistency in the data format, such as unified date formats and consistent time intervals.

- **Data Enrichment**:
  - Enhance the dataset with additional relevant features, such as public holidays, local events, or economic indicators that might influence volunteer demand.
  - Include metadata like beneficiary type or location to provide context for the demand patterns.

6.3.2.2 Modeling

* ***Choose forecasting methods***

LSTM (Long Short-Term Memory networks): A deep learning model ideal for capturing complex patterns and dependencies in time series data, especially with long-term dependencies and non-linear relationships.

* ***Model implementation***

LSTM: Prepare the data by transforming it into sequences suitable for LSTM. Build and train the LSTM network on these sequences and optimize the network's hyperparameters (use cross-validation and techniques like grid search to fine-tune model parameters and select the best-performing configuration).

* ***Validation***

- **Train-Test Split**:
  - Split the data into training and test sets, typically using a time-based split (e.g., using the first 80% of data for training and the remaining 20% for testing).
  - Ensure the test set is sufficiently recent to reflect current trends and patterns.
- **Model Evaluation**:
  - Compare model predictions against the test set to evaluate accuracy. Use metrics like MAE, MSE, or RMSE to quantify prediction errors.
  - Visualize the forecasted vs. actual values to assess the model's performance and identify areas for improvement.
- **Cross-Validation**:
  - Implement rolling-origin or time series cross-validation to evaluate the model's stability and performance over different periods.

- **Anomaly Detection**:
  - Analyze residuals (differences between actual and predicted values) to detect and understand anomalies or shifts in the pattern that the model may not have captured.
  - Adjust the model to better handle or anticipate such anomalies in future predictions.
- **Model Refinement**:
  - Based on validation results, refine the model by adjusting parameters, adding new features, or choosing a different modeling approach if necessary.

7 Discussion and Recommendations

In this chapter, we will provide a comprehensive reflection on the entire process, beginning with an identification of the limitations of our model. We will offer suggestions for improvement, discuss the key lessons learned, and outline the challenges we faced.

7.1 Limitations

Our model has several limitations that must be addressed for scalability and practical use by the Finnish Red Cross.

- Scalability issues: We have not yet assessed the performance and response times of the system under high-load conditions. The model requires further research and adjustments to handle larger databases.
- Dataset limitation: due to the limited and synthetic dataset, our interpretation of result queries might not accurately reflect the real volume of applications and volunteers in different cities.
- Lack of cost consideration: We did not consider the costs associated with storing, reading, and writing data when running each query, so no indexing was used.

Addressing these limitations and making necessary modifications will be crucial for the successful implementation of the model.

7.2 Recommendations

In our evaluation of the Volunteer Matching System, we identified key areas for improvement to enhance both functionality and compliance.

First, we recommend replacing geolocation with zip codes to increase granularity in our data. This adjustment will allow us to divide large cities into smaller, more manageable sections, thereby making travel times more feasible for volunteers. Geolocations can be accurately converted to zip codes using resources such as the Google Maps API or publicly available bounding box data.

Additionally, we propose the implementation of synthetic IDs for volunteer identification to ensure compliance with GDPR regulations and to protect personal data. The current design, which uses personal social security numbers (SSNs) as primary keys, directly violates GDPR Article 4[1], which mandates strict guidelines on the use of personal data. By generating synthetic IDs, we can safeguard personal information and uphold the required legal standards.

These recommendations, once implemented, will significantly improve the system's effectiveness and reliability, aligning it more closely with the operational needs of the Finnish Red Cross.

7.3 Key insights

Through this project, our group has gained valuable insights into the following areas:

- End-to-End Workstream of Database Solution:

The project highlighted the importance of integrating various components seamlessly from data collection to storage, ensuring robust and efficient database management. This comprehensive approach underscored the necessity of understanding the entire workflow and the interplay between different stages.

Working on designing a Volunteer Matching System (VMS) with the Finnish Red Cross (FRC) taught us the importance of smoothly connecting different parts of the project.

[1] Source: https://gdpr-info.eu/art-4-gdpr/

From connecting PostgreSQL database, loading the synthetic data to storing them securely, every step had to be well-coordinated.

- Practical Case Implementation:

Implementing theoretical knowledge in practical scenarios was a crucial aspect of the project. This application allowed us to bridge the gap between academic learning and real-world challenges. By designing a system that real volunteers and organizations will use, we learned how to solve actual problems and make our ideas come to life. Also, we were able to refine our skills in SQL and Python (especially in data visualization) and adapt theoretical concepts to practical, workable solutions. This experience emphasized the value of practical application in solidifying our understanding and improving our problem-solving abilities.

- Virtual Team Collaboration:

Our team used digital tools (Git, Teams, Telegram) to work together, despite being in different locations. This project showed us how important clear and consistent communication is for a virtual team. We had to stay in touch regularly, share updates, and make sure everyone understood their tasks. This teamwork was key to successfully completing the Volunteer Matching System for the Finnish Red Cross.

7.4 Challenges encountered

The project requirements were high-level, complex and difficult to interpret, leading to misunderstandings in initial planning stages and later implementing stages.

Also, as mentioned before, we faced challenges due to knowledge gap in designing a scalable database solution. This impacted our ability to accommodate growing data volumes while maintaining optimal performance.

7.5 Group Member Contribution

The workload was divided equally among group members, considering each person's interests and strengths. While each member was responsible for leading and finalizing specific tasks, all members contributed and provided support throughout the process.

Task name	**In charge**
UML diagrams and update documentation	Dung Tran
BCNF and Anomalies	Zong-Rong Yang
Create tables into database	Zong-Rong Yang
Load data from Excel file to database	Trang Le Forsell and Tien Huynh
Data pre-processing and cleaning	Tien Huynh
Basic queries	Tien Huynh, Dung Tran, and Zong-Rong Yang
Advanced queries	Tien Huynh and Dung Tran
Transactions	Dung Tran
Data analysis	Trang Le Forsell
Optional additions	Tien Huynh and Dung Tran
Update final report	Dung Tran
Presentation slides	Trang Le Forsell