Volunteer Matching System

SQL IMPLEMENTATION AND EXPLANATION

OUTLINE

Part A. Basic

Part B. Advanced

- a) Views
- b) Trigger and Functions
- c) Transactions
- d) Analysis (Shown in Jupyter Notebook file)

Part A. Basic

1. (2p) For each request, include the starting date and the end date in the title.

```
-- PART A.1

UPDATE request

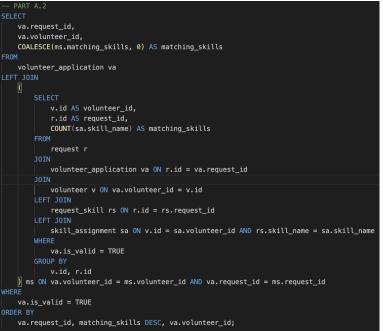
SET title = CONCAT(title, ' (', TO_CHAR(start_date, 'YYYY-MM-DD'), ' - ',

TO_CHAR(end_date, 'YYYY-MM-DD'), ')');
```

Explanation: Using CONCAT() to combine the string and update the title in request table

	•	RBC title	123 beneficiary_id	•
1	1	work in team needed (2024-07-25 - 2024-07-28)		3 🗹
2	2	work with young needed (2022-05-31 - 2022-06-01		9 🗹
3	3	guide and teach needed (2024-07-22 - 2024-07-27		9 🗹
4	4	guide and teach needed (2024-09-25 - 2024-09-25		5 🗹
5	5	work with elderly needed (2021-03-01 - 2021-03-07		4 🗹
6	6	work with elderly needed (2024-10-18 - 2024-10-18		3 🗹
7	7	organise activities needed (2023-09-15 - 2023-09-		9 🗹
8	8	help in crisis needed (2024-06-05 - 2024-06-07)		6 🗹
9	9	work with young needed (2023-05-28 - 2023-06-0		9 🗹
10	10	work in team needed (2024-C work with young needed	(2023-05-28 - 2023-0	06-01)

2. (3p) 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.



	123 request_id 🔻	RBC volunteer_id	•	123 matching_skills	•
1	1 ₫	☑ 230283-963X			3
2	1 ₫	011074-9149			1
3	1 ⊿	☑ 160903A941P			1
4	1 ☑	211074-9401			1
5	1 ⊿	☑ 211099-910H			1
6	1 ☑	☑ 250681-919H			1
7	1 ⊿	☑ 210753-990T			0
8	2 🗹	☑ 190697-999B			6
9	2 🗹	☑ 220782-910B			5
10	2 🗹	270794-9576			5
11	2 🗹	☑ 200569-926L			4
12	2 🗹	☑ 101003A9918			3

The output

Explanation: We use the subquery to find the number of skills each volunteer has that match the skills required by each request they have applied for. And then the main query retrieves all valid applications and includes the count of matching skills for each volunteer and request pair. The COALESCE is used to handle NULL value just in case.

3. (3p) 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.

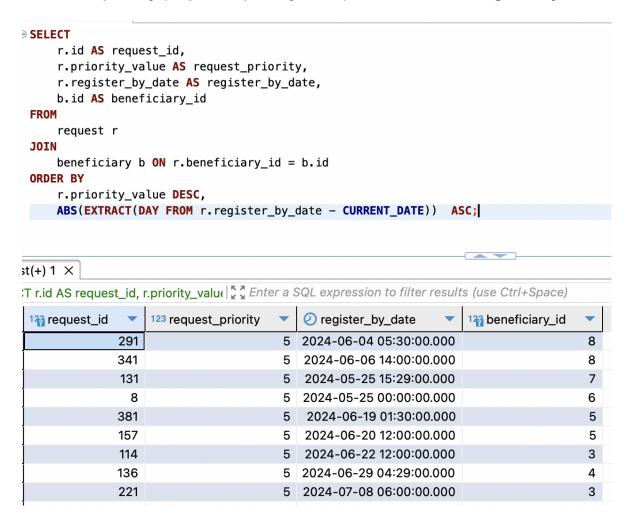
```
SELECT
    r.id AS request_id,
    rs.skill_name,
    GREATEST(SUM(rs.min_need) - COALESCE(COUNT(va.volunteer_id), 0), 0) AS missing_volunteers
FROM
    request r
JOIN
    request_skill rs ON r.id = rs.request_id
LEFT JOIN
    volunteer_application va ON r.id = va.request_id
LEFT JOIN
    skill_assignment sa ON va.volunteer_id = sa.volunteer_id AND rs.skill_name = sa.skill_name
WHERE
    va.is_valid = TRUE
GROUP BY
    r.id, rs.skill_name
ORDER BY
    r.id, rs.skill_name;
```

	¹⅔ request_id ▼	skill_name •	123 missing_volunteers	•
1	1	☑ CommunicationAndMarketing		28
2	1	☑ EventHosting		14
3	1	☑ EventOrganization		7
4	1	PhotographyAndVideo		7
5	2	☑ CookingAndBaking		10
6	2	☑ DigitalCompetence		10
7	2	☑ EventOrganization		10
8	2	☑ FinanceAndAccounting		10
9	2	☑ HealthCareOrFirstAid		0
10	2	☑ MeetingPeople		10
11	2	☑ PublicPerformances		10

The output

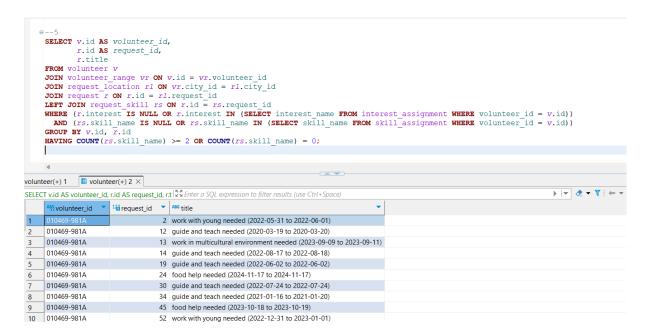
Explanation: The missing number of volunteers is calculated by subtracting the number of volunteers who have required skills from the minimum needed number of volunteers from the table request_skill. We use GREATEST() to handle negative values.

4. (3p) Sort requests and the beneficiaries who made them by the highest number of priority (request's priority value) and the closest 'register by date'.



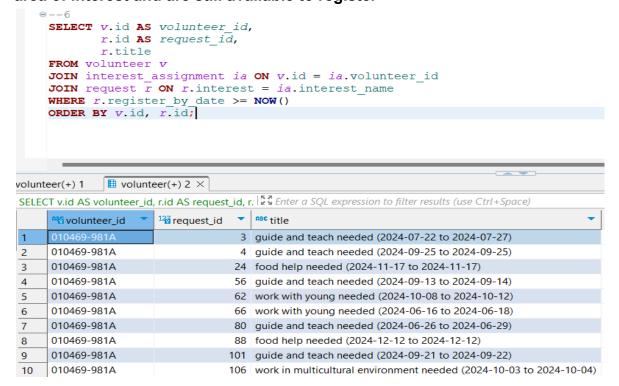
Explanation: We sort by the highest priority and the closest 'register by date', which we use the absolute difference between the current day and the register day, and ascend them. (The current_date at that moment is 2024-06-06)

5. (3p) 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).



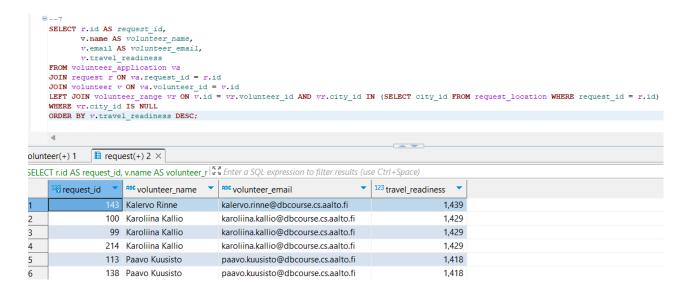
Explanation: This query joins the volunteer with their range, request location, and request. It checks for interest matches with required skills. The results are grouped by volunteer and request IDs and filtered by skill count >= 2 (volunteer has at least 2 of the required skills for the request) or = 0 (no required skill for the request)

6. (3p) For each volunteer, list all the requests where the title matches their area of interest and are still available to register



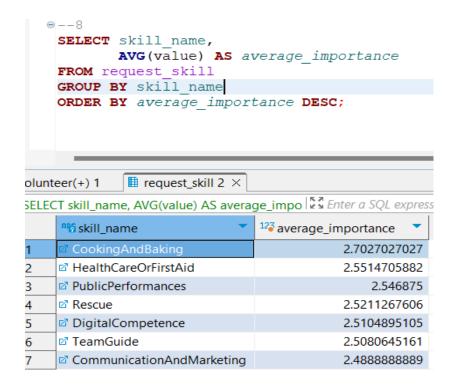
Explanation: This query joins the volunteer with their interest and with requests that have that interest. The results are filtered to include only requests that are open for registration and ordered by volunteerID and requestID.

7. (3p) 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.



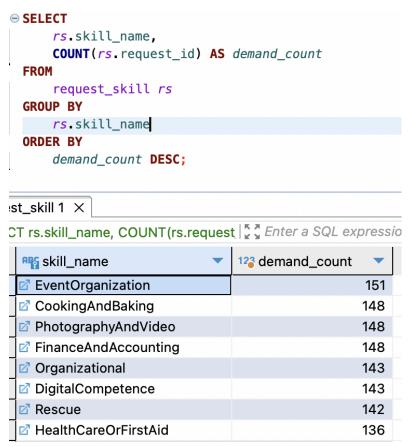
Explanation: This query joins volunteer applications with request and volunteer, and checks volunteer location ranges against request locations by left join. Results are filtered to include volunteers outside the request's location range and are ordered by travel readiness.

8. (3p) Order the skills overall (from all requests) in the most prioritized to least prioritized (average the importance value).



Explanation: This query calculates the average importance value for each skill from the 'request_skill' table. It groups the skills and orders them by their average importance value(from the most to least important).

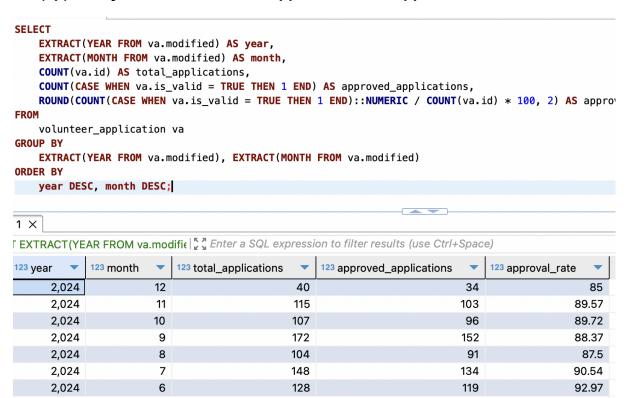
9. (3p) Query to Determine the Demand for Specific Skills Across All Requests



Explanation: We use this query to understand the demand for specific skills across all requests and help the organization identify which skills are most needed.

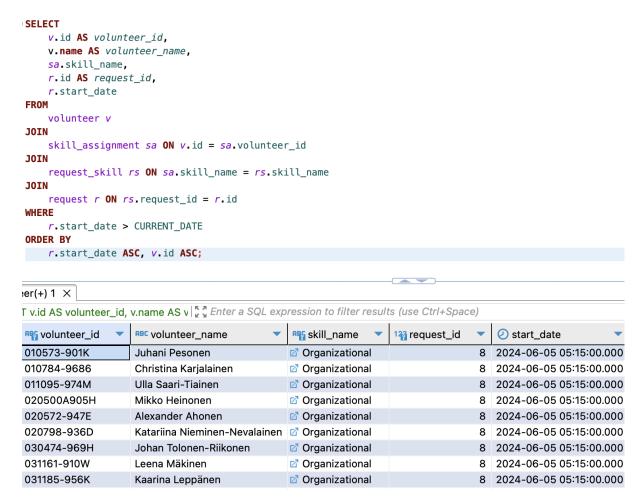
This can inform training programs and recruitment efforts to ensure a sufficient number of volunteers with these in-demand skills are available.

10. (3p) Query to Track Volunteer Application and Approval Rates Over Time



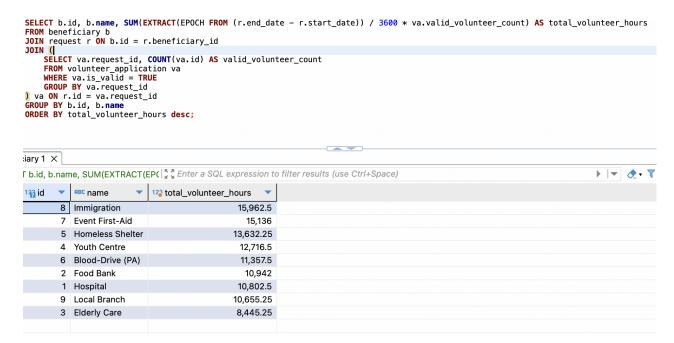
Explanation: This query tracks volunteer application and approval rates over time, providing insights into trends and the efficiency of the approval process. Monitoring these rates helps identify periods of high volunteer interest and ensures that applications are processed promptly, maintaining volunteer engagement.

11. (3p) Query to Identify Volunteers with Specific Skills for Upcoming Requests



Explanation: This query helps identify volunteers with specific skills needed for upcoming requests. By planning ahead and ensuring the right volunteers are assigned to tasks that require their skills, the organization can enhance task success rates and ensure that projects start on time with the necessary expertise.

12. (3p) List of Beneficiaries and Their Total Volunteer Hours



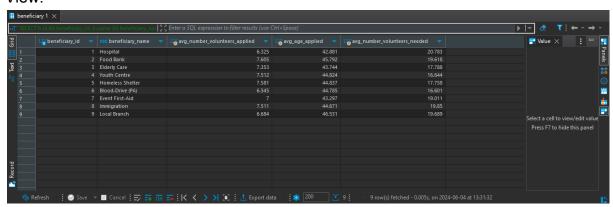
Explanation: This query calculates the total number of volunteer hours provided for each beneficiary's requests, assuming each valid application corresponds to a fixed number of volunteer hours. Thus, this provides insights into the total volunteer effort dedicated to each beneficiary, helping to evaluate the level of support beneficiaries are receiving.

Part B. Advance:

a) Views:

1. (5p) 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 Query:

View:

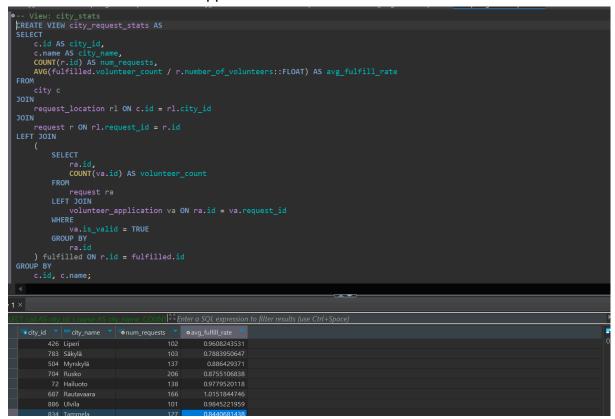


Explanation: In the FROM clause, we create two sub-table:

- applicants_per_request: the inner subquery is used to count the number of applicants for each request, then the average for each beneficiary is calculated and join by the id of each beneficiary
- volunteer_ages: It retrieves the age of each applicant from the volunteer table and then calculates the average age of applicants for each request.

The statistics are rounded to 3 decimal places.

2. (5p) Write a query that creates a view containing: location ID, city name, number of requests, avg_fulfill_rate avg_fulfill_rate is the fraction between the number of satisfied volunteers for the application and the total needed.



Explanation: This query creates a view that displays all the cities, with the name, total requests from each city and the fulfillment rate for each request. We believe that this statistic will help the organizers to identify which city would need the most help, and therefore make better resource allocation.

This query is joined between the request location, request, city and a table to show the number of applications for each volunteer.

We use a subquery **fulfilled** to calculate the number of valid volunteer applications for each request. We count the number of valid (is_valid = TRUE) volunteer applications and group by request_id.

b) Triggers and Functions

1. (5p) The code for the constraints:

```
PREATE OR REPLACE FUNCTION validate_ID(id VARCHAR) RETURNS BOOLEAN AS $$

DECLARS:

date_part VARCHAR(6);

idvidual_part VARCHAR(3);

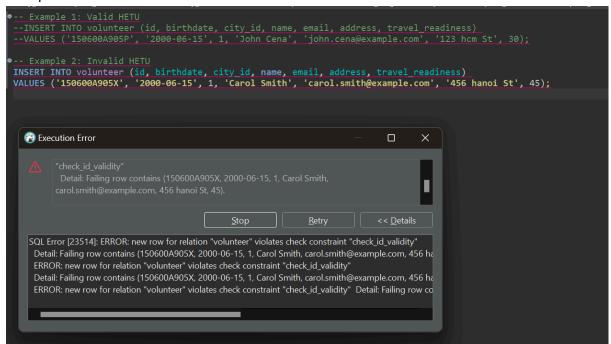
separator (IAR(1);

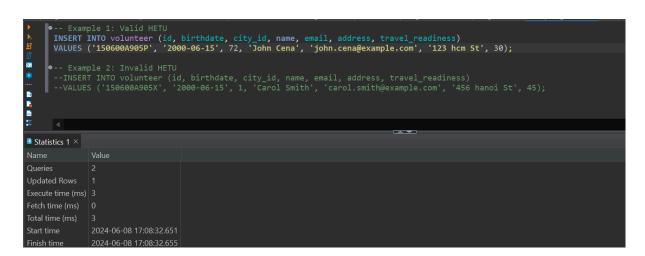
separator (I
```

Explanations: Function validate_ID is used for:

- Length Check: Ensures the ID has exactly 11 characters.
- Extract Parts: Separates the ID into date_part, separator, individual_part, and control character.
- Separator Check: Validates the separator character.
- Numeric Check: Ensures date part and individual part are numeric.
- Calculate Control Character: Concatenates date_part and individual_part into a numeric value, calculates the modulus 31, and finds the corresponding control character from control_characters. Explicitly casts the modulus result to integer for the SUBSTRING function.
- Validation: Compares the expected control character with the provided control character and returns TRUE or FALSE.

Sample run:





2. (5p) The code for the trigger:

```
OCREATE OR REPLACE FUNCTION adjust_volunteers_needed() RETURNS TRIGGER AS $$
DECLARE
    difference INT;
BEGIN
    -- Calculate the difference in min_need
    IF TG_OP = 'INSERT' THEN
        difference := NEW.min_need;
ELSIF TG_OP = 'UPDATE' THEN
        difference := NEW.min_need - OLD.min_need;
ELSIF TG_OP = 'DELETE' THEN
        difference := - OLD.min_need;
END IF;
    -- Update the number_of_volunteers in the request table
    UPDATE request
    SET number_of_volunteers = number_of_volunteers + difference
    WHERE id = COALESCE(NEW.request_id, OLD.request_id);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

OCREATE TRIGGER trg_adjust_volunteers_needed
after INSERT OR UPDATE OR DELETE ON request_skill
FOR EACH ROW
EXECUTE FUNCTION adjust_volunteers_needed();
```

*Assumption: We assume that the number_of_volunteer has already included the volunteer with the required skill, so the requirements for the unskilled volunteer will not change, thus this trigger will change the number_of_volunteer according to the change to the request_skill of the request.

Explanation: The trigger will calculate the changes to the required_skill table, and update the corresponding changes to the number_of_volunteer of the request.

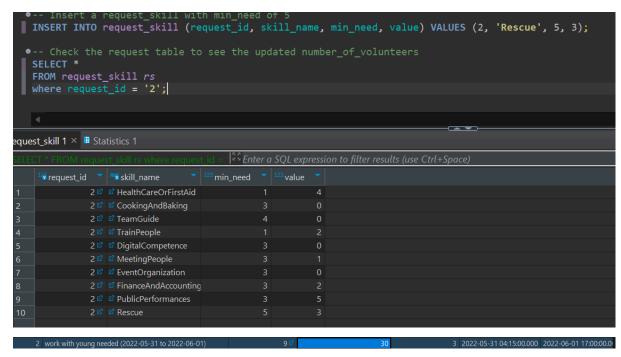
Example:

This is the information about request with request id= 2

2 work with young needed (2022-05	5-31 to 2022-06-01) 9 🗗	25	3 2022-05-	31 04:15:00.000 2022-06-01 17:00:00.0
2 ₺	☑ HealthCareOrFirstAid		1	4
2 ₺	☑ CookingAndBaking		3	0
2 ₺	☑ TeamGuide		4	0
2 ♂	☑ TrainPeople		1	2
2 ₺	☑ DigitalCompetence		3	0
2 ₺	☑ MeetingPeople		3	1
2 ₺	☑ EventOrganization		3	0
2 ₺	☑ FinanceAndAccounting		3	2
2 ♂	☑ PublicPerformances		3	5

We are going to update the following query to check INSERT:

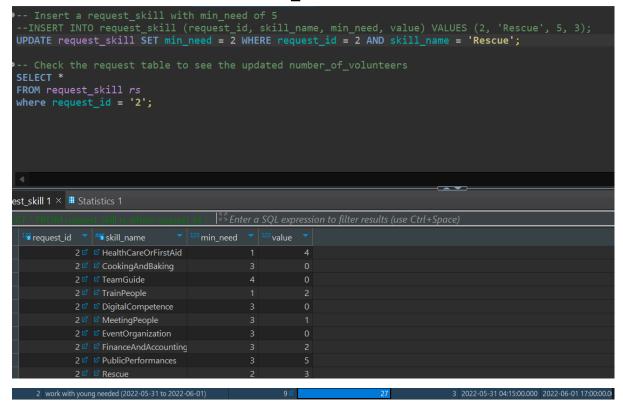
- After the query execution:



The volunteer needed for request 2 has increased from 25 to 30

Testing UPDATE, continue from the example above:

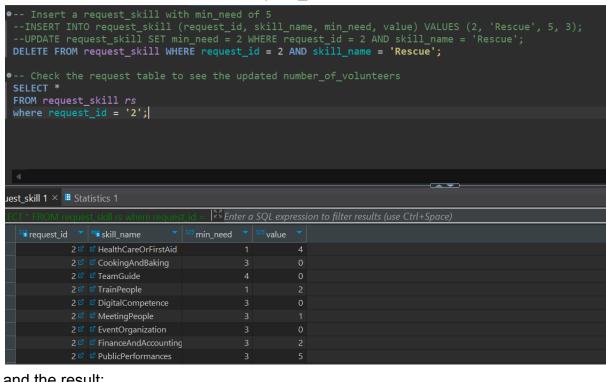
Now we decrease the added min need down to 2



Since the min_need of Rescue reduced by 3, the volunteer needed also decreased by 3.

Testing DELETE, continue from the example above:

Now we deleted the added request_skill.



and the result:

2 work with young needed (2022-05-31 to 2022-06-01)	9 ₺	25	3
---	-----	----	---

c) Transactions

We create a transaction to swap the value of skill 'HealthcareOrFirstAid' and 'TrainPeople' in all requests made by beneficiary 1.

```
--Swap the value of skill 'HealthCareOrFirstAid' and 'TrainPeople' in all requests made by beneficiary 1
).
D
        BEGIN;
                  orary table to hold swapped values
        CREATE TEMP TABLE temp_skill_swap AS
        SELECT rs.request_id, rs.skill_name, rs.value
        FROM request_skill rs
        JOIN request r ON rs.request_id = r.id
        WHERE r.beneficiary_id = 1
AND rs.skill_name IN ('HealthCareOrFirstAid', 'TrainPeople');
           Swap HealthCareOrFirstAid with TrainPeople
       SET value = (SELECT value FROM temp skill swap WHERE request_id = request_skill.request_id AND skill_name = 'TrainPeople')
WHERE skill_name = 'HealthCareOrFirstAid'
          AND request_id IN (SELECT request_id FROM temp skill swap);
       --- Swap TrainPeople with HealthCareOrFirstAid

UPDATE request_skill

SET value = (SELECT value FROM temp skill swap WHERE request_id = request_skill.request_id AND skill_name = 'HealthCareOrFirstAid')

WHERE skill_name = 'TrainPeople'
         AND request_id IN (SELECT request_id FROM temp skill swap);
       DROP TABLE temp skill swap;
       4
■ Statistics 1 ×
Updated Rows 66
            BEGIN;
              -- Temporary table to hold swapped values
             CREATE TEMP TABLE temp skill swap AS
             SELECT rs.request id, rs.skill name, rs.value
             FROM request skill rs
             JOIN request r ON rs.request id = r.id
             WHERE r.beneficiary id = 1
             AND rs.skill_name IN ('HealthCareOrFirstAid', 'TrainPeople');
              -- Swap HealthCareOrFirstAid with TrainPeople
             UPDATE request_skill
              SET value = (SELECT value FROM temp_skill_swap WHERE request_id = request_skill.request_id AND skill_name = 'TrainPeople')
              WHERE skill_name = 'HealthCareOrFirstAid'
               AND request_id IN (SELECT request_id FROM temp_skill_swap);
              -- Swap TrainPeople with HealthCareOrFirstAid
```

Explanation: First, we create a temporary table temp_skill_swap to store information of the two skills for all requests made by beneficiary 1. Next, we swap the values of the two skills with each other using updates. Then we drop the temporary table, cleaning up the temporary data used for the swap and commit.

In some cases, the requirements for certain roles might change over time. If the system initially matched a volunteer to a role based on a specific skill, but later the role's requirements change to a different skill, swapping values can help adjust the volunteer's profile to better match the new requirements.

Or sometimes, volunteers might be temporarily reassigned to different tasks. For instance, if a volunteer with HealthCareOrFirstAid skills is temporarily assigned to train others (TrainPeople), the system might need to reflect this temporary change without permanently altering the volunteer's skill set.