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'), ')');
```

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

		title	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-	

work with young needed (2023-05-28 - 2023-06-01)

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.

```sql
-- PART A.2
SELECT
    va.request_id,
    va.volunteer_id,
    COALESCE(ms.matching_skills, 0) AS matching_skills
FROM
    volunteer_application va
LEFT JOIN
    (
        SELECT
            v.id AS volunteer_id,
            r.id AS request_id,
            COUNT(sa.skill_name) AS matching_skills
        FROM
            request r
        JOIN
            volunteer_application va ON r.id = va.request_id
        JOIN
            volunteer v ON va.volunteer_id = v.id
        LEFT JOIN
            request_skill rs ON r.id = rs.request_id
        LEFT JOIN
            skill_assignment sa ON v.id = sa.volunteer_id AND rs.skill_name = sa.skill_name
        WHERE
            va.is_valid = TRUE
        GROUP BY
            v.id, r.id
    ) ms ON va.volunteer_id = ms.volunteer_id AND va.request_id = ms.request_id
WHERE
    va.is_valid = TRUE
ORDER BY
    va.request_id, matching_skills DESC, va.volunteer_id;
```

	request_id	volunteer_id	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

<u>Explanation:</u> 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.

```sql
-- PART A.3
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;
```

	request_id	skill_name	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'.

```sql
SELECT
    r.id AS request_id,
    r.priority_value AS request_priority,
    r.register_by_date AS register_by_date,
    b.id AS beneficiary_id
FROM
    request r
JOIN
    beneficiary b ON r.beneficiary_id = b.id
ORDER BY
    r.priority_value DESC,
    ABS(EXTRACT(DAY FROM r.register_by_date - CURRENT_DATE))  ASC;
```

st(+) 1

T r.id AS request_id, r.priority_valu | Enter a SQL expression to filter results (use Ctrl+Space)

request_id	request_priority	register_by_date	beneficiary_id
291	5	2024-06-04 05:30:00.000	8
341	5	2024-06-06 14:00:00.000	8
131	5	2024-05-25 15:29:00.000	7
8	5	2024-05-25 00:00:00.000	6
381	5	2024-06-19 01:30:00.000	5
157	5	2024-06-20 12:00:00.000	5
114	5	2024-06-22 12:00:00.000	3
136	5	2024-06-29 04:29:00.000	4
221	5	2024-07-08 06:00:00.000	3

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).

```sql
--5
SELECT v.id AS volunteer_id,
       r.id AS request_id,
       r.title
FROM volunteer v
JOIN volunteer_range vr ON v.id = vr.volunteer_id
JOIN request_location rl ON vr.city_id = rl.city_id
JOIN request r ON r.id = rl.request_id
LEFT JOIN request_skill rs ON r.id = rs.request_id
WHERE (r.interest IS NULL OR r.interest IN (SELECT interest_name FROM interest_assignment WHERE volunteer_id = v.id))
  AND (rs.skill_name IS NULL OR rs.skill_name IN (SELECT skill_name FROM skill_assignment WHERE volunteer_id = v.id))
GROUP BY v.id, r.id
HAVING COUNT(rs.skill_name) >= 2 OR COUNT(rs.skill_name) = 0;
```

volunteer(+) 1 | volunteer(+) 2

SELECT v.id AS volunteer_id, r.id AS request_id, r.t | Enter a SQL expression to filter results (use Ctrl+Space)

	volunteer_id	request_id	title
1	010469-981A	2	work with young needed (2022-05-31 to 2022-06-01)
2	010469-981A	12	guide and teach needed (2020-03-19 to 2020-03-20)
3	010469-981A	13	work in multicultural environment needed (2023-09-09 to 2023-09-11)
4	010469-981A	14	guide and teach needed (2022-08-17 to 2022-08-18)
5	010469-981A	19	guide and teach needed (2022-06-02 to 2022-06-02)
6	010469-981A	24	food help needed (2024-11-17 to 2024-11-17)
7	010469-981A	30	guide and teach needed (2022-07-24 to 2022-07-24)
8	010469-981A	34	guide and teach needed (2021-01-16 to 2021-01-20)
9	010469-981A	45	food help needed (2023-10-18 to 2023-10-19)
10	010469-981A	52	work with young needed (2022-12-31 to 2023-01-01)

<u>Explanation</u>: 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

```sql
--6
SELECT v.id AS volunteer_id,
       r.id AS request_id,
       r.title
FROM volunteer v
JOIN interest_assignment ia ON v.id = ia.volunteer_id
JOIN request r ON r.interest = ia.interest_name
WHERE r.register_by_date >= NOW()
ORDER BY v.id, r.id;
```

volunteer(+) 1 | volunteer(+) 2

SELECT v.id AS volunteer_id, r.id AS request_id, r. | Enter a SQL expression to filter results (use Ctrl+Space)

	volunteer_id	request_id	title
1	010469-981A	3	guide and teach needed (2024-07-22 to 2024-07-27)
2	010469-981A	4	guide and teach needed (2024-09-25 to 2024-09-25)
3	010469-981A	24	food help needed (2024-11-17 to 2024-11-17)
4	010469-981A	56	guide and teach needed (2024-09-13 to 2024-09-14)
5	010469-981A	62	work with young needed (2024-10-08 to 2024-10-12)
6	010469-981A	66	work with young needed (2024-06-16 to 2024-06-18)
7	010469-981A	80	guide and teach needed (2024-06-26 to 2024-06-29)
8	010469-981A	88	food help needed (2024-12-12 to 2024-12-12)
9	010469-981A	101	guide and teach needed (2024-09-21 to 2024-09-22)
10	010469-981A	106	work in multicultural environment needed (2024-10-03 to 2024-10-04)

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.

```sql
--7
SELECT r.id AS request_id,
       v.name AS volunteer_name,
       v.email AS volunteer_email,
       v.travel_readiness
FROM volunteer_application va
JOIN request r ON va.request_id = r.id
JOIN volunteer v ON va.volunteer_id = v.id
LEFT JOIN volunteer_range vr ON v.id = vr.volunteer_id AND vr.city_id IN (SELECT city_id FROM request_location WHERE request_id = r.id)
WHERE vr.city_id IS NULL
ORDER BY v.travel_readiness DESC;
```

olunteer(+) 1 | request(+) 2

SELECT r.id AS request_id, v.name AS volunteer_r | Enter a SQL expression to filter results (use Ctrl+Space)

	request_id	volunteer_name	volunteer_email	travel_readiness
1	143	Kalervo Rinne	kalervo.rinne@dbcourse.cs.aalto.fi	1,439
2	100	Karoliina Kallio	karoliina.kallio@dbcourse.cs.aalto.fi	1,429
3	99	Karoliina Kallio	karoliina.kallio@dbcourse.cs.aalto.fi	1,429
4	214	Karoliina Kallio	karoliina.kallio@dbcourse.cs.aalto.fi	1,429
5	113	Paavo Kuusisto	paavo.kuusisto@dbcourse.cs.aalto.fi	1,418
6	138	Paavo Kuusisto	paavo.kuusisto@dbcourse.cs.aalto.fi	1,418

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).

```sql
--8
SELECT skill_name,
       AVG(value) AS average_importance
FROM request_skill
GROUP BY skill_name
ORDER BY average_importance DESC;
```

olunteer(+) 1 | request_skill 2

SELECT skill_name, AVG(value) AS average_impo | Enter a SQL express

	skill_name	average_importance
1	CookingAndBaking	2.7027027027
2	HealthCareOrFirstAid	2.5514705882
3	PublicPerformances	2.546875
4	Rescue	2.5211267606
5	DigitalCompetence	2.5104895105
6	TeamGuide	2.5080645161
7	CommunicationAndMarketing	2.4888888889

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

```sql
SELECT
    rs.skill_name,
    COUNT(rs.request_id) AS demand_count
FROM
    request_skill rs
GROUP BY
    rs.skill_name
ORDER BY
    demand_count DESC;
```

st_skill 1

CT rs.skill_name, COUNT(rs.request | Enter a SQL expressio

skill_name	demand_count
EventOrganization	151
CookingAndBaking	148
PhotographyAndVideo	148
FinanceAndAccounting	148
Organizational	143
DigitalCompetence	143
Rescue	142
HealthCareOrFirstAid	136

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

```sql
SELECT
    EXTRACT(YEAR FROM va.modified) AS year,
    EXTRACT(MONTH FROM va.modified) AS month,
    COUNT(va.id) AS total_applications,
    COUNT(CASE WHEN va.is_valid = TRUE THEN 1 END) AS approved_applications,
    ROUND(COUNT(CASE WHEN va.is_valid = TRUE THEN 1 END)::NUMERIC / COUNT(va.id) * 100, 2) AS appro
FROM
    volunteer_application va
GROUP BY
    EXTRACT(YEAR FROM va.modified), EXTRACT(MONTH FROM va.modified)
ORDER BY
    year DESC, month DESC;
```

1

T EXTRACT(YEAR FROM va.modifie | Enter a SQL expression to filter results (use Ctrl+Space)

year	month	total_applications	approved_applications	approval_rate
2,024	12	40	34	85
2,024	11	115	103	89.57
2,024	10	107	96	89.72
2,024	9	172	152	88.37
2,024	8	104	91	87.5
2,024	7	148	134	90.54
2,024	6	128	119	92.97

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

```sql
SELECT
    v.id AS volunteer_id,
    v.name AS volunteer_name,
    sa.skill_name,
    r.id AS request_id,
    r.start_date
FROM
    volunteer v
JOIN
    skill_assignment sa ON v.id = sa.volunteer_id
JOIN
    request_skill rs ON sa.skill_name = rs.skill_name
JOIN
    request r ON rs.request_id = r.id
WHERE
    r.start_date > CURRENT_DATE
ORDER BY
    r.start_date ASC, v.id ASC;
```

er(+) 1

T v.id AS volunteer_id, v.name AS v | Enter a SQL expression to filter results (use Ctrl+Space)

volunteer_id	volunteer_name	skill_name	request_id	start_date
010573-901K	Juhani Pesonen	Organizational	8	2024-06-05 05:15:00.000
010784-9686	Christina Karjalainen	Organizational	8	2024-06-05 05:15:00.000
011095-974M	Ulla Saari-Tiainen	Organizational	8	2024-06-05 05:15:00.000
020500A905H	Mikko Heinonen	Organizational	8	2024-06-05 05:15:00.000
020572-947E	Alexander Ahonen	Organizational	8	2024-06-05 05:15:00.000
020798-936D	Katariina Nieminen-Nevalainen	Organizational	8	2024-06-05 05:15:00.000
030474-969H	Johan Tolonen-Riikonen	Organizational	8	2024-06-05 05:15:00.000
031161-910W	Leena Mäkinen	Organizational	8	2024-06-05 05:15:00.000
031185-956K	Kaarina Leppänen	Organizational	8	2024-06-05 05:15:00.000

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

```sql
SELECT b.id, b.name, SUM(EXTRACT(EPOCH FROM (r.end_date - r.start_date)) / 3600 * va.valid_volunteer_count) AS total_volunteer_hours
FROM beneficiary b
JOIN request r ON b.id = r.beneficiary_id
JOIN (
    SELECT va.request_id, COUNT(va.id) AS valid_volunteer_count
    FROM volunteer_application va
    WHERE va.is_valid = TRUE
    GROUP BY va.request_id
) va ON r.id = va.request_id
GROUP BY b.id, b.name
ORDER BY total_volunteer_hours desc;
```

:iary 1

T b.id, b.name, SUM(EXTRACT(EP(Enter a SQL expression to filter results (use Ctrl+Space)

id	name	total_volunteer_hours
8	Immigration	15,962.5
7	Event First-Aid	15,136
5	Homeless Shelter	13,632.25
4	Youth Centre	12,716.5
6	Blood-Drive (PA)	11,357.5
2	Food Bank	10,942
1	Hospital	10,802.5
9	Local Branch	10,655.25
3	Elderly Care	8,445.25

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:

```sql
-- View: beneficiary_statistics
CREATE VIEW beneficiary_statistics AS
SELECT
    b.id AS beneficiary_id,
    b.name AS beneficiary_name,
    ROUND(applications_per_request.avg_applications::numeric,3) AS avg_number_volunteers_applied,
    ROUND(AVG(volunteer_ages.age)::numeric,3) AS avg_age_applied,
    ROUND(AVG(rq.number_of_volunteers)::numeric,3) AS avg_number_volunteers_needed
FROM
    beneficiary b
JOIN
    request rq ON b.id = rq.beneficiary_id
-- Calculate the avg applications separately, and join by beneficiary_id
JOIN
    (SELECT
            sub.beneficiary_id,
            AVG(sub.count) as avg_applications
    FROM
        (SELECT
                r.beneficiary_id ,
                r.id ,
                COUNT(va.volunteer_id )
         FROM
                volunteer_application va
         JOIN
                request r on r.id = va.request_id
         GROUP BY r.id
         ORDER BY r.beneficiary_id,r.id
        ) as sub
        GROUP BY sub.beneficiary_id
    ) AS applications_per_request ON b.id = applications_per_request.beneficiary_id
-- List all the age of applicants for each request
JOIN
    (
        SELECT
            va.request_id,
            EXTRACT(YEAR FROM AGE(v.birthdate)) AS age
        FROM
            volunteer_application va
        JOIN
            volunteer v ON va.volunteer_id = v.id
    ) AS volunteer_ages ON rq.id = volunteer_ages.request_id
GROUP BY
    b.id,applications_per_request.avg_applications
order by b.id;
```

View:

beneficiary 1

	beneficiary_id	beneficiary_name	avg_number_volunteers_applied	avg_age_applied	avg_number_volunteers_needed
1	1	Hospital	6.325	42.881	20.783
2	2	Food Bank	7.605	45.792	19.618
3	3	Elderly Care	7.353	43.744	17.788
4	4	Youth Centre	7.512	44.824	16.644
5	5	Homeless Shelter	7.581	44.837	17.758
6	6	Blood-Drive (PA)	6.545	44.785	16.601
7	7	Event First-Aid	7	43.297	19.011
8	8	Immigration	7.511	44.671	19.85
9	9	Local Branch	6.684	46.531	19.689

Value: Select a cell to view/edit value. Press F7 to hide this panel

Refresh | Save | Cancel | Export data | 200 | 9 | 9 row(s) fetched - 0.005s, on 2024-06-04 at 13:31:32

<u>Explanation:</u> 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.

```sql
-- View: city_stats
CREATE VIEW city_request_stats AS
SELECT
    c.id AS city_id,
    c.name AS city_name,
    COUNT(r.id) AS num_requests,
    AVG(fulfilled.volunteer_count / r.number_of_volunteers::FLOAT) AS avg_fulfill_rate
FROM
    city c
JOIN
    request_location rl ON c.id = rl.city_id
JOIN
    request r ON rl.request_id = r.id
LEFT JOIN
    (
        SELECT
            ra.id,
            COUNT(va.id) AS volunteer_count
        FROM
            request ra
        LEFT JOIN
            volunteer_application va ON ra.id = va.request_id
        WHERE
            va.is_valid = TRUE
        GROUP BY
            ra.id
    ) fulfilled ON r.id = fulfilled.id
GROUP BY
    c.id, c.name;
```

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

city_id	city_name	num_requests	avg_fulfill_rate
426	Liperi	102	0.9608243531
783	Säkylä	103	0.7883950647
504	Myrskylä	137	0.886429371
704	Rusko	206	0.8755106838
72	Hailuoto	138	0.9779520118
687	Rautavaara	166	1.0151844746
886	Ulvila	101	0.9845221959
834	Tammela	127	0.8440681438

<u>Explanation</u>: 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:

```
CREATE OR REPLACE FUNCTION validate_ID(id VARCHAR) RETURNS BOOLEAN AS $$
DECLARE
    date_part VARCHAR(6);
    individual_part VARCHAR(3);
    control_character CHAR(1);
    separator CHAR(1);
    numeric_part BIGINT;
    expected_control CHAR(1);
    control_characters CONSTANT VARCHAR(31) := '0123456789ABCDEFHJKLMNPRSTUVWXY';
BEGIN
    -- Check length
    IF LENGTH(id) <> 11 THEN
        RETURN FALSE;
    END IF;

    -- Extract parts
    date_part := SUBSTRING(id FROM 1 FOR 6);
    separator := SUBSTRING(id FROM 7 FOR 1);
    individual_part := SUBSTRING(id FROM 8 FOR 3);
    control_character := SUBSTRING(id FROM 11 FOR 1);

    -- Check valid separators
    IF separator NOT IN ('+', '-', 'A', 'B', 'C', 'D', 'E', 'F', 'X', 'Y', 'W', 'V', 'U') THEN
        RETURN FALSE;
    END IF;

    -- Check numeric parts and convert to number
    IF date_part !~ '^[0-9]+$' OR individual_part !~ '^[0-9]+$' THEN
        RETURN FALSE;
    END IF;

    numeric_part := CAST(date_part || individual_part AS BIGINT);

    -- Calculate expected control character
    expected_control := SUBSTRING(control_characters FROM (numeric_part % 31)::integer + 1 FOR 1);

    -- Compare calculated control character with given control character
    IF control_character = expected_control THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;
ALTER TABLE volunteer
ADD CONSTRAINT check_id_validity
CHECK (validate_ID(id));
```

<u>Explanations:</u> 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:

```
-- Example 1: Valid HETU
--INSERT INTO volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
--VALUES ('150600A905P', '2000-06-15', 1, 'John Cena', 'john.cena@example.com', '123 hcm St', 30);

-- Example 2: Invalid HETU
INSERT INTO volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
VALUES ('150600A905X', '2000-06-15', 1, 'Carol Smith', 'carol.smith@example.com', '456 hanoi St', 45);
```

Execution Error

"check_id_validity"
Detail: Failing row contains (150600A905X, 2000-06-15, 1, Carol Smith, carol.smith@example.com, 456 hanoi St, 45).

Stop | Retry | << Details

```
SQL Error [23514]: ERROR: new row for relation "volunteer" violates check constraint "check_id_validity"
  Detail: Failing row contains (150600A905X, 2000-06-15, 1, Carol Smith, carol.smith@example.com, 456 ha
  ERROR: new row for relation "volunteer" violates check constraint "check_id_validity"
  Detail: Failing row contains (150600A905X, 2000-06-15, 1, Carol Smith, carol.smith@example.com, 456 ha
  ERROR: new row for relation "volunteer" violates check constraint "check_id_validity"  Detail: Failing row co
```

```
-- Example 1: Valid HETU
INSERT INTO volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
VALUES ('150600A905P', '2000-06-15', 72, 'John Cena', 'john.cena@example.com', '123 hcm St', 30);

-- Example 2: Invalid HETU
--INSERT INTO volunteer (id, birthdate, city_id, name, email, address, travel_readiness)
--VALUES ('150600A905X', '2000-06-15', 1, 'Carol Smith', 'carol.smith@example.com', '456 hanoi St', 45);
```

Statistics 1

Name	Value
Queries	2
Updated Rows	1
Execute time (ms)	3
Fetch time (ms)	0
Total time (ms)	3
Start time	2024-06-08 17:08:32.651
Finish time	2024-06-08 17:08:32.655

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

```
CREATE 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;


CREATE 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.

<u>Explanation</u>: 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 (2022-05-31 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:

```sql
-- Insert a request_skill with min_need of 5
INSERT INTO request_skill (request_id, skill_name, min_need, value) VALUES (2, 'Rescue', 5, 3);

-- Check the request table to see the updated number_of_volunteers
SELECT *
FROM request_skill rs
where request_id = '2';
```

equest_skill 1 × Statistics 1

SELECT * FROM request_skill rs where request_id = | Enter a SQL expression to filter results (use Ctrl+Space)

	request_id	skill_name	min_need	value
1	2	HealthCareOrFirstAid	1	4
2	2	CookingAndBaking	3	0
3	2	TeamGuide	4	0
4	2	TrainPeople	1	2
5	2	DigitalCompetence	3	0
6	2	MeetingPeople	3	1
7	2	EventOrganization	3	0
8	2	FinanceAndAccounting	3	2
9	2	PublicPerformances	3	5
10	2	Rescue	5	3

2	work with young needed (2022-05-31 to 2022-06-01)	9	30	3	2022-05-31 04:15:00.000	2022-06-01 17:00:00.0

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

```sql
-- Insert a request_skill with min_need of 5
--INSERT INTO request_skill (request_id, skill_name, min_need, value) VALUES (2, 'Rescue', 5, 3);
UPDATE request_skill SET min_need = 2 WHERE request_id = 2 AND skill_name = 'Rescue';

-- Check the request table to see the updated number_of_volunteers
SELECT *
FROM request_skill rs
where request_id = '2';
```

est_skill 1 × Statistics 1

CT * FROM request_skill rs where request_id = | Enter a SQL expression to filter results (use Ctrl+Space)

request_id	skill_name	min_need	value
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
2	Rescue	2	3

2	work with young needed (2022-05-31 to 2022-06-01)	9	27	3	2022-05-31 04:15:00.000	2022-06-01 17:00:00.0

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.

```sql
-- Insert a request_skill with min_need of 5
--INSERT INTO request_skill (request_id, skill_name, min_need, value) VALUES (2, 'Rescue', 5, 3);
--UPDATE request_skill SET min_need = 2 WHERE request_id = 2 AND skill_name = 'Rescue';
DELETE FROM request_skill WHERE request_id = 2 AND skill_name = 'Rescue';

-- Check the request table to see the updated number_of_volunteers
SELECT *
FROM request_skill rs
where request_id = '2';
```

uest_skill 1 × Statistics 1

ECT * FROM request_skill rs where request_id = Enter a SQL expression to filter results (use Ctrl+Space)

request_id	skill_name	min_need	value
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

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

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
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;

COMMIT;
```

Statistics 1

Name	Value
Updated Rows	66
Query	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

<u>Explanation:</u> 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.