Project_part2

Linda Srbova, Juri Voloskin

A Queries

1.

```
UPDATE request
SET title = title || ' from '|| start_date|| ' to ' || end_date;
```

2.

```
SELECT va.volunteer_id
FROM request r
JOIN volunteer_application va ON va.request_id = r.id
LEFT JOIN request_skill rs ON rs.request_id = r.id -- joins requests with skills, if
there are no requested skills assigned, then it is assigned to be NULL
LEFT JOIN skill_assignment sa ON sa.volunteer_id = va.volunteer_id AND sa.skill_name
= rs.skill_name -- left join tries to include volunteers skills by matching the
volunteer id. The sa.skill_name will be kept as it is, if its matches with request
skills, otherwise it will contain NULL values
WHERE va.is_valid = TRUE -- condition to select only valid applications
GROUP BY va.volunteer_id -- groups the rows by volutneer id
ORDER BY COUNT(sa.skill_name) DESC; -- ordering from highest to lowest number of
matching skills per volunteer. Since column of skill name contains null values
whenever it does not match the requested skill, the function count (does not count
NULL values) can be used to count the number of matching skills
```

3.

```
select rs.request_id, rs.skill_name, (rs.min_need - COUNT(DISTINCT sa.volunteer_id))
AS missing_skills_count
-- reporting request id for each skill (thus each request id will have as many rows
as there is the number of assigned skills to it)
-- reporting the number of people that are still needed
from request_skill rs
join volunteer_application va on rs.request_id =va.request_id -- joining volunteer
applications with requests for skills
join skill_assignment sa on sa.volunteer_id = va.volunteer_id -- joining volunteer
skills by volunteer id
where rs.skill_name =sa.skill_name OR rs.skill_name IS null -- selecting rows where
there is a match between the skills, of the volunteer who applied, and the listed
skill needed
group by rs.request_id,rs.skill_name, rs.min_need -- group by request id
HAVING
(rs.min_need - COUNT(DISTINCT sa.volunteer_id)) > 0; -- filter to show only the
requests and the related skills for which there are volunteers still missing
-- reducing the number of volunteers needed for the skills of the specific request by
each distinct volunteer that applied and matches the skill
```

4.

```sql
SELECT
r.title , b."name"
FROM request r
JOIN beneficiary b ON r.beneficiary_id = b.id -- Join request and beneficiary by matching the beneficiary id
where r.register_by_date > current_date -- removing those requests that have already passed the registration deadline
ORDER BY
r.priority_value DESC, -- Ordering by request's priority from the highest
r.register_by_date ASC; -- Sort by register by the closest register date - the most urgent
```

5.

```sql
SELECT vr.volunteer_id, rl.request_id,
from request r
JOIN request_location rl ON r.id = rl.request_id --joining request and range by id
join volunteer_range vr on vr.city_id = rl.city_id -- joining volunteer range by city
JOIN skill_assignment sa ON vr.volunteer_id = sa.volunteer_id -- joining volunteer skills by matching volunteer ids
LEFT JOIN request_skill rs ON rs.request_id = rl.request_id -- joining requested skills by request id; left join ensures that if no request skills are for the given request, then the skills column is null
and rs.skill_name = sa.skill_name -- Adjusted LEFT JOIN condition, the matching is only if the requested and volunteer skills match
GROUP BY vr.volunteer_id, rl.request_id -- grouping
HAVING
COUNT(*) >= 2 --for each volunteer and request, count those that have more than 2 rows (ie fulfilling the conditions of matching and range)
```

6.

#adding a new empty column in request

```
ALTER TABLE request ADD COLUMN normalized_title VARCHAR(255);
```

#the column filled up with edited title to be same format as interest_name (selecting string until 'need' - to remove the dates and include only info that is indicated in volunteers' assignment of interest; first letter of word in caps and removing space between words)

```
update request
SET normalized_title =
REGEXP_REPLACE(
INITCAP(
CASE
WHEN POSITION('needed' IN title) > 0 THEN
LEFT(title, POSITION('needed' IN title) - 1)
ELSE
title
end),'[^a-zA-Z0-9]+', '', 'g');
```

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

```
select r.title
from interest_assignment ia
JOIN request r ON ia.interest_name LIKE r.normalized_title
WHERE r.register_by_date > CURRENT_DATE;
```

7.

```
select r.id, v."name",v.email -- listing request ID, volunteer name and email
from request r            -- first table is request
join request_location rl on rl.request_id = r.id -- joining info on the location at which the request is needed, it can be at multiple distinct locations
inner join volunteer_application va on r.id =va.request_id -- joining request table and volunteer application only for rows where the request id is matching. One request has often many applications, even from the same person
join volunteer v on v.id = va.volunteer_id -- joining volunteer table by matching volunteer id with volunteer application table. One volunteer, with specific name, email, and travel_readiness, usually submits many applications to different requests
join volunteer_range vr on vr.volunteer_id = va.volunteer_id -- adding information on the volunteer location
where vr.city_id != rl.city_id -- selecting only the rows where the volunteer and request locations do not match
group by r.id, va.request_id, va.id, v."name",v.email, v.travel_readiness -- Each volunteer can have many locations and each request can have many locations.
```

```
Therefore, for each request, all the possible combinations, where these don't match,
are listed. To not list them separately by locations, we group them.
order by v.travel_readiness ; -- ordering the list by travel_readiness which is
specific for each volunteer
```

8.

```
SELECT rs.skill_name -- listing the skills in order
FROM request_skill rs -- from table request_skill, which already contains all the
skills requested by beneficiaries
GROUP BY rs.skill_name -- we group it by skills, because the skills are repated in
many requests
ORDER BY avg(rs.value) DESC; -- and for each skill we calculate the average val
```

Free queries

```
-- find volunteers and beneficiaries that reside in the same city, given that
volunteers applied for the requests made by those beneficiaries
select distinct b.name as ben_name, v.name as vol_name,  c."name" as city_name
from volunteer v, volunteer_application va, request r, beneficiary b, city c
where v.id = va.volunteer_id and va.request_id = r.id and r.beneficiary_id = b.id
and v.city_id = b.city_id and v.city_id = c.id;

-- find volunteers who do not want to travel outside the city where they reside
select v_home."name", v_home.id, c."name"
from city c, (
    -- among volunteers, find only those who chose only their home city as the
volunteer_range
    select v."name", v.id, v.city_id
    from volunteer v
    except (
        -- cut off volunteers who chose cities different from their home city
        select distinct  v.name, v.id, vr.city_id
        from volunteer v, volunteer_range vr
        where v.id=vr.volunteer_id and v.city_id <> vr.city_id)
        ) as v_home
where c.id = v_home.city_id;

-- see how many skills volunteers have relative to the number of requests they
applied for
select v."name", sa.volunteer_id, vol_req_count.req_count, count(sa.skill_name)
as skill_count
from volunteer v, skill_assignment sa, (
    -- count the nr of requests each volunteer applied for
    select va.volunteer_id, count(va.request_id) as req_count
    from volunteer_application va
```

```
    group by va.volunteer_id) as vol_req_count
where v.id = sa.volunteer_id and v.id = vol_req_count.volunteer_id
group by sa.volunteer_id, vol_req_count.req_count, v."name";



-- for each request, see the number of applications compared to the min_number of
people needed (skills not verified)
select rs.request_id, sum(rs.min_need) as ppl_needed, req_app_cnt.appls_count
from request_skill rs, (
    -- count the nr of applications submitted for each request
    select va.request_id, count(id) as appls_count
    from volunteer_application va
    group by request_id) as req_app_cnt
where rs.request_id = req_app_cnt.request_id
group by rs.request_id, req_app_cnt.request_id, req_app_cnt.appls_count;
```

B. ADVANCED

a)VIEWS

VIEW 1

```
------------------
-- view
-----------------
--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
CREATE view v1_avg_ppl_supdem as
select r.beneficiary_id, avg(r.number_of_volunteers) as avg_ppl_needed,
avg(apps_per_req.app_count) as avg_ppl_applied
from request r, (
    -- nr of applications per request
    select va.request_id, count(id) as app_count
    from volunteer_application va
    group by va.request_id) as apps_per_req
where r.id  = apps_per_req.request_id
group by r.beneficiary_id;
```

VIEW 2

```
CREATE VIEW Beneficiaries_With_most_match AS
SELECT b."name", count(distinct va.id) as matching_applications -- listing
beneficiaries from the highest to lowest number of failed applications
FROM beneficiary b
JOIN request r ON b.id = r.beneficiary_id -- joining beneficiaries and requests by
matching beneficiary id
JOIN volunteer_application va ON r.id = va.request_id -- joining volunteer
applications by matching request ids
join request_skill rs on rs.request_id = r.id -- joining requested skills by matching
request ids
join skill_assignment sa on sa.volunteer_id = va.volunteer_id -- joining volunteer
skills by matching volunteer id
join interest_assignment ia on ia.volunteer_id = va.volunteer_id
WHERE r.register_by_date < current_date
and r.register_by_date >= va.modified
and sa.skill_name =rs.skill_name
and ia.interest_name = r.normalized_title
and va.is_valid = TRUE-- selecting rows for the registration date has passed, the
application wasnt modified after the deadline, the skills matched, the applicant's
interests matched, the application was valid,
GROUP BY b.name
ORDER BY count(distinct va.id) DESC;
```

b)Trigger and Functions

Trigger 1:

```
CREATE OR REPLACE FUNCTION parse_vol_id(vol_id TEXT)
RETURNS TABLE(old_id, true_c_char text, extract_num int, found_c_char text) AS $$
BEGIN
    RETURN QUERY
    SELECT
        CAST(split_part(date_string, '-', 1) AS INT) AS year,
        CAST(split_part(date_string, '-', 2) AS INT) AS month,
        CAST(split_part(date_string, '-', 3) AS INT) AS day;
END;
$$ LANGUAGE plpgsql;



CREATE OR REPLACE FUNCTION parse_id(input_text TEXT)
RETURNS TABLE(birthdate_int text, separator text, code_int text, last_char text)
AS $$
BEGIN
    RETURN QUERY
    SELECT
```

```
        substring(input_text FROM 1 FOR 6) AS birthdate_int,
        substring(input_text FROM 7 for 1) as separator,
        substring(input_text FROM 8 for 3) as code_int,
        right(input_text, 1) AS last_char;

END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION id_code(vol_id TEXT)
RETURNS INTEGER AS $$
DECLARE
    bdate TEXT;
    code TEXT;
    code_int INTEGER;
    c_char_code INTEGER;
BEGIN
    bdate := substring(vol_id FROM 1 FOR 6);
    code  := substring(vol_id FROM 8 for 3);
    code_int := cast( (bdate || code) as integer);
    c_char_code := code_int % 31;

    RETURN c_char_code;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION predict_control_char(vol_id TEXT)
RETURNS TEXT AS $$
DECLARE
    bdate TEXT;
    code TEXT;
    code_int INTEGER;
    c_char_code INTEGER;
BEGIN
    bdate := substring(vol_id FROM 1 FOR 6);
    code  := substring(vol_id FROM 8 for 3);
    code_int := cast( (bdate || code) as integer);
    c_char_code := code_int % 31;

    RETURN substring('0123456789ABCDEFHJKLMNPRSTUVWXY' FROM c_char_code+1 FOR 1);

END;
$$ LANGUAGE plpgsql;
```

```
-- check constraint for the volunteer table
-- The ID is valid if they satisfies: len(id) = 11, valid separator char, correct
control character is used
CREATE OR REPLACE FUNCTION check_all_(vol_id TEXT)
RETURNS BOOL AS $$
DECLARE
    bdate TEXT;
    sepchar text;
    code TEXT;
    code_int INTEGER;
    c_char_code INTEGER;
    control_char text;
    predict_c_char text;

    len_check BOOL;
    sepchar_check bool;
    control_c_check bool;

BEGIN
    bdate := substring(vol_id FROM 1 FOR 6);
    sepchar := substring(vol_id FROM 7 for 1);
    code  := substring(vol_id FROM 8 for 3);
    control_char := right(vol_id, 1);

    -- compute the control char based on the integer content of the id
    code_int := cast( (bdate || code) as integer);
    c_char_code := code_int % 31;
    predict_c_char := substring('0123456789ABCDEFHJKLMNPRSTUVWXY' FROM
c_char_code+1 FOR 1);

    len_check := length(vol_id) = 11;
    sepchar_check := sepchar in ('+', '-', 'A', 'B', 'C', 'D', 'E', 'F', 'X',
'Y', 'W', 'V', 'U');
    control_c_check = (control_char = predict_c_char);


    RETURN len_check and sepchar_check and control_c_check;

END;
$$ LANGUAGE plpgsql;
```

```
SELECT v.id, parsed.last_char, predict_control_char(v.id), check_all_(v.id)
FROM volunteer v, lateral parse_id(v.id) as parsed;



-- trigger function definition
CREATE OR REPLACE FUNCTION check_valid_input()
RETURNS TRIGGER AS $$
declare
    vol_id text;
    bdate TEXT;
    sepchar text;
    code TEXT;
    code_int INTEGER;
    c_char_code INTEGER;
    control_char text;
    predict_c_char text;

    len_check BOOL;
    sepchar_check bool;
    control_c_check bool;

begin
    vol_id := new.id;

    -- extract the elements of volunteer id required for validation
    bdate := substring(vol_id FROM 1 FOR 6);
    sepchar := substring(vol_id FROM 7 for 1);
    code  := substring(vol_id FROM 8 for 3);
    control_char := right(vol_id, 1);

    -- compute the control char based on the integer content of the id
    code_int := cast( (bdate || code) as integer);
    c_char_code := code_int % 31;
    predict_c_char := substring('0123456789ABCDEFHJKLMNPRSTUVWXY' FROM
c_char_code+1 FOR 1);

    -- compute boolean checkers
    len_check := length(vol_id) = 11;
    sepchar_check := sepchar in ('+', '-', 'A', 'B', 'C', 'D', 'E', 'F', 'X',
'Y', 'W', 'V', 'U');
    control_c_check := (control_char = predict_c_char);

    -- process each of the checks one by one
    -- raise an exception if any of them is wrong
```

```
    -- print the reason for a failure
    IF not len_check THEN
        RAISE EXCEPTION 'id length is not exactly 11!';
    END IF;

    IF not sepchar_check THEN
        RAISE EXCEPTION 'invalid separator character used!';
    END IF;

    IF not control_c_check THEN
        RAISE EXCEPTION 'control character is invalid!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


-- trigger definition
CREATE TRIGGER new_volunteer_validation
BEFORE INSERT ON volunteer
FOR EACH ROW
EXECUTE FUNCTION check_valid_input();


-- examples of incorrect ids (prevented by the trigger)
INSERT INTO volunteer (id) VALUES ('121191123-419H'); -- invalid length
INSERT INTO volunteer (id) VALUES ('121191Q419H'); -- invalid separator
INSERT INTO volunteer (id) VALUES ('121191-419H'); -- invalid control char

-- tested with valid inputs as well (the trigger lets them through)
```

c) Transactions

Transactions 1.

```
CREATE TABLE volunteer_assigned (
assignment_id SERIAL PRIMARY KEY,
request_id INT,
volunteer_id INT,
volunteer_skill text,
requested_skill text);

BEGIN TRANSACTION;

-- extracting the number of volunteers needed for the request and the registration
deadline
SELECT r.number_of_volunteers, r.register_by_date
INTO total_needed, deadline
FROM request r
WHERE r.id = :request_id; -- dynamic assignment of the request id of interest



WITH prioritizing_skills_order -- prioritize skills by the value of importance;
listing request id, skill name, minimum number of people needed
AS (SELECT
rs.request_id,
rs.skill_name,
rs.min_need,
ROW_NUMBER() OVER (PARTITION BY rs.request_id ORDER BY rs.value DESC) AS
skill_priority_request -- within request id ordering the skills by importance
FROM request_skill rs),
applicant_skills_order -- matching volunteers with skills and requests and ordering
by the skills importance
AS     (SELECT
va.volunteer_id,
va.request_id,
sa.skill_name,
ROW_NUMBER() OVER (PARTITION BY rs.request_id ORDER BY rs.value DESC) AS
skill_usefulness -- for each request, if the application is valid, the volunteers's
skills are ordered by value of importance for the request
FROM volunteer_application va
JOIN skill_assignment sa ON va.volunteer_id = sa.volunteer_id
JOIN request_skill rs ON va.request_id = rs.request_id AND sa.skill_name =
rs.skill_name
WHERE va.is_valid = TRUE),
chosen -- matching the volunteers with the requests in order of how well their skills
meet the needs
AS (SELECT
```

```
aso.volunteer_id,
pso.request_id,
aso.skill_name
FROM applicant_skills_order aso
JOIN prioritizing_skills_order pso ON aso.request_id = pso.request_id AND
siao.skill_name = siro.skill_name
WHERE aso.skill_usefulness <= pso.min_need) -- the rows are combined for each request
as long as the integer in the column 'skill_usefulness' is lower or equal to the
number of people needed for the skills

-- Insert the information from 'chosen' (assigned volunteers) into the previously
created volunteer_assigned table
INSERT INTO volunteer_assigned (request_id, volunteer_id, volunteer_skill,
requested_skill)
SELECT CAST(c.request_id AS INT),
CAST(c.volunteer_id AS INT),
c.skill_name
FROM chosen_c;

-- saving how many volunteers match the needed skills, from the data saved in the new
table
SELECT COUNT(*) AS volunteer_count,
INTO volunteer_count
FROM volunteer_assigned
WHERE va.request_id = :request_id;

-- If the conditions are met, commit the transaction, else roll back
IF (volunteer_count < total_needed AND CURRENT_TIMESTAMP < deadline) THEN
ROLLBACK;
ELSE
COMMIT;
END_IF;

END_TRANSACTION;
```

Transaction 2.

```
-- let beneficiaries update the end and regby dates of their requests
-- however, if start-date is older than the new end-date or if the start-date is
later than the new regby date
-- or the new end-date is earlier than the new regby date, then rollover the
transaction

begin;

DO $$
DECLARE
    new_end_date date := '2024-01-01';
    req_id_used integer := 1; -- set here the request id you're willing to change
    check_end_date bool;
begin

    WITH req_cte AS (
        SELECT * FROM request
        where request.id = req_id_used )

    SELECT new_end_date < req_cte.start_date INTO check_end_date from req_cte;


    -- Raise the exception if the new end-date ends up being earlier than the
existing start-date
    IF check_end_date THEN
        RAISE EXCEPTION 'The new end-date cannot be earlier than the start-date';

    ELSE
        -- If the condition is satisfied, update the row
        UPDATE request
        SET end_date = new_end_date
        where id = req_id_used;

    END IF;
END $$;

-- Commit the transaction if no exception was raised
COMMIT;
```

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

d)Analysis

1. VISUALIZATION

```
query_available = " SELECT c.name as city_name, count(distinct vr.volunteer_id)
as available_count FROM city as c JOIN volunteer_range as vr ON vr.city_id = c.id
GROUP BY c.name ORDER BY count(distinct vr.volunteer_id) DESC"
# joining city and request location by matching city_id, joining the volunteer
range on volunteer id
# groupping the data by the name of the city, for which then it is counted how
many volunteers are available
# ordering from the cities with most available volunteers

#Executing the query and reading the results into a DataFrame
available_volunteers = pd.read_sql_query(query_available, engine)

query_applied = "SELECT c.name as city_name, count(distinct vr.volunteer_id) as
applied_count FROM request_location as rl JOIN city as c ON c.id = rl.city_id
JOIN volunteer_application as va ON va.request_id = rl.request_id JOIN
volunteer_range as vr ON vr.volunteer_id = va.volunteer_id WHERE rl.city_id  =
vr.city_id GROUP BY c.name ORDER BY count(distinct vr.volunteer_id) DESC "
# joining city and request location by matching city_id, joining the volunteer
application (to include the info on the applicants) on matching request id,
joining the volunteer range on volunteer id
# filtering the rows where the volunteer range actually matches the request
location to filter out those who applied but do not match
# groupping the data by the name of the city, for which then it is counted how
many volunteers applied
# ordering from the cities with highest number of applicants with suitable skills

#Executing the query and reading the results into a DataFrame
applied_volunteers = pd.read_sql_query(query_applied, engine)

import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(8, 6))  # setting the figure width, height in inches
x = np.arange(len(applied_volunteers['city_name']))
bar_width = 0.30 # setting the width of one bar
```

```
# Plotting histograms
plt.bar(x - bar_width/2, available_volunteers['available_count'],
width=bar_width, color='green', label='Available volunteers')
plt.bar(x + bar_width/2, applied_volunteers['applied_count'], width=bar_width,
color='red', label='Applied volunteers')

# Add labels and legend
plt.xlabel('City', fontsize=14)
plt.ylim(40,65)
plt.ylabel('Number of volunteers', fontsize=14) #setting axis label
plt.legend() # add legend
plt.xticks(x, applied_volunteers['city_name'], rotation=15, ha='center',
fontsize=13) #add labels of cities under angle in the center of the bar
plt.yticks( fontsize=13) #add labels of cities under angle
plt.gca().xaxis.set_tick_params(which='both', bottom=False, top=False)  # Remove
ticks

plt.gcf().set_size_inches(10, 6)

# Show plot
plt.show()
```