Support of volunteer activity & Database

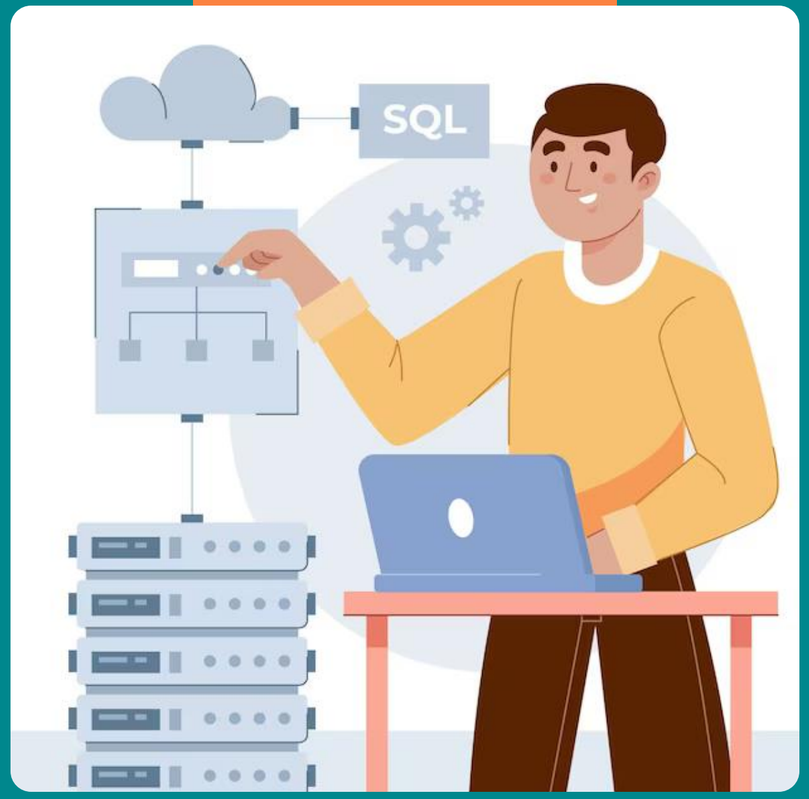
Made by: Borys Boiko IP-21 group
Supervised by: docent Lishchuk Katerina

Main goals

Simplicity

Flexibility

Adaptiveness



Work stages

Explore topic essentials along with future field of deployment

Build a conceptual entity relationship diagram

Move from paper draft to the real programmed database

Fix bugs and potential issues. Optimize common operations

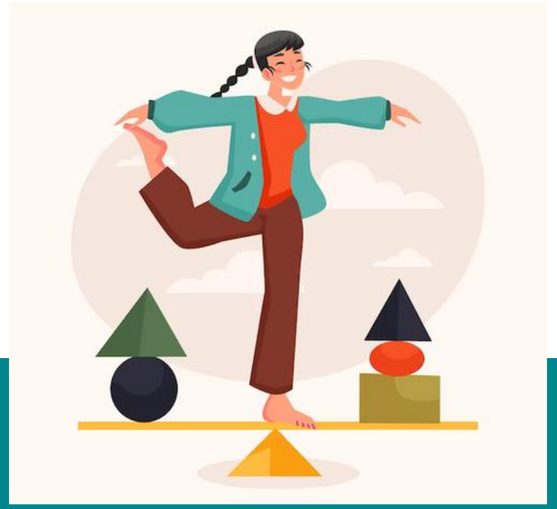
Volunteership

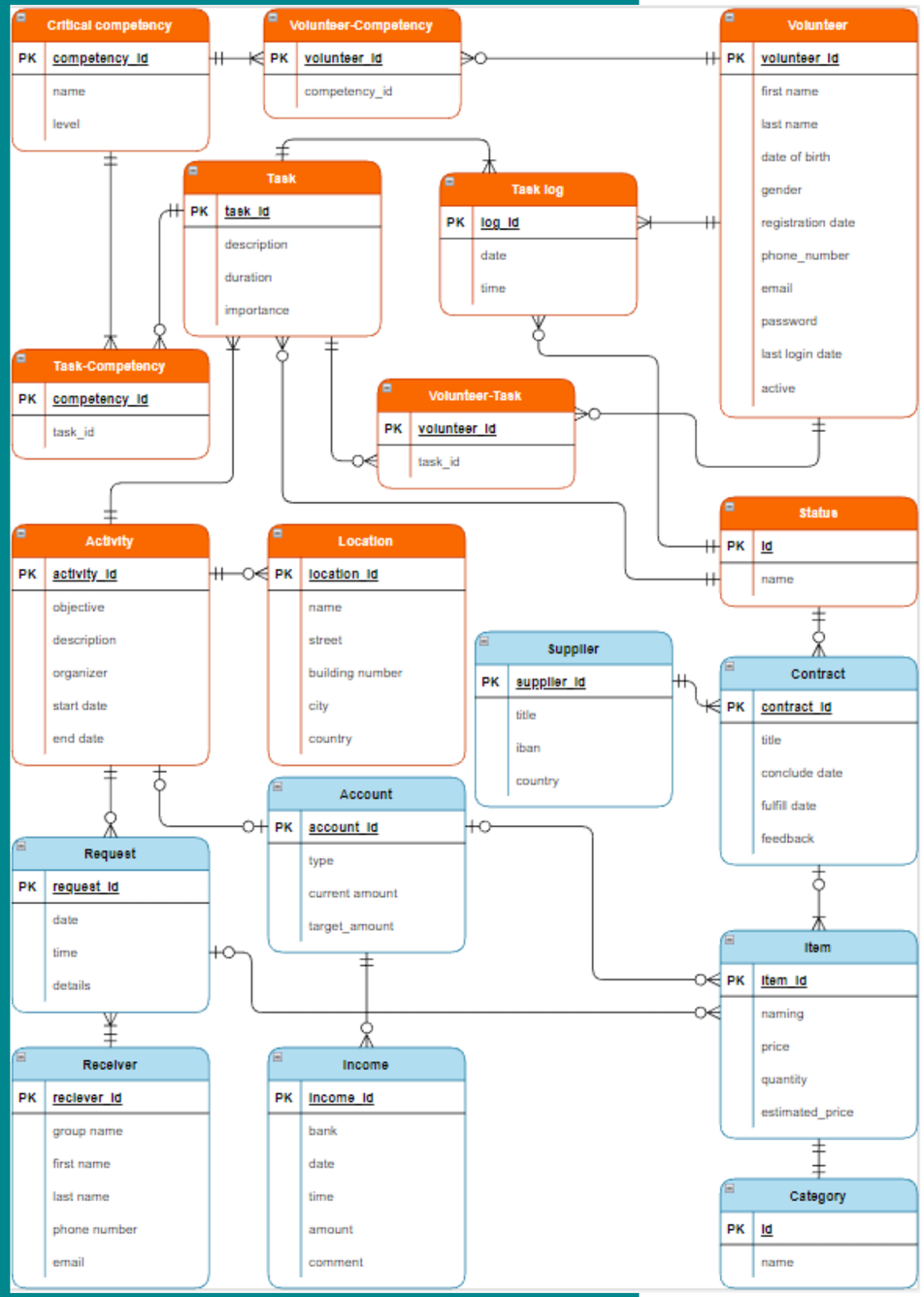
Volunteer`s activity is based on three pillars:

Voluntary: nobody can force other to help.

Non-profitable: one can not be paid for volunteering.

Socially beneficial: volunteers do not wait, they take lead.



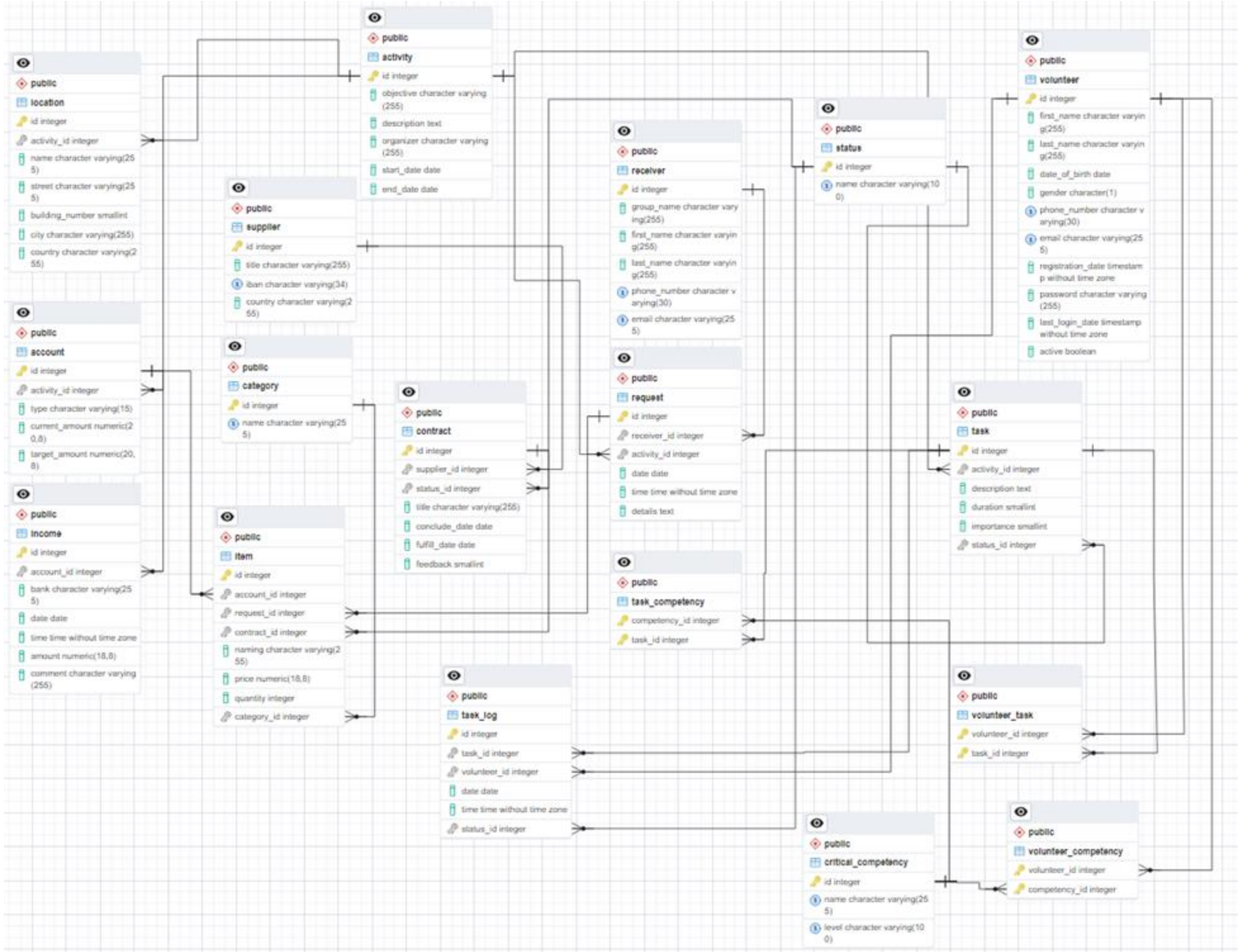


ER-diagram

The diagram is painted in two colors. Each color denotes different sphere of interest:

- fundraising
- task management

Scheme generated by pgadmin4



Views

Interaction with the database

```sql
CREATE OR REPLACE VIEW v_volunteer_leaderboard_top_50 AS
WITH id_points
AS (SELECT vol.id, SUM(t.duration/task_occurence)
    AS points FROM volunteer vol
JOIN task_log tl ON tl.volunteer_id = vol.id
JOIN task t ON t.id = tl.task_id
JOIN (SELECT task_id, COUNT(*) AS task_occurence FROM task_log
     WHERE task_log.status_id =
      (SELECT id FROM status WHERE name = 'Finished')
     GROUP BY task_id) AS task_count
ON task_count.task_id = t.id
WHERE EXTRACT(MONTH FROM tl.date) = EXTRACT(MONTH FROM CURRENT_DATE)
  AND EXTRACT(YEAR FROM tl.date) = EXTRACT(YEAR FROM CURRENT_DATE)
  AND tl.status_id = (SELECT id FROM status WHERE name = 'Finished')
  GROUP BY vol.id)
  SELECT volunteer.first_name ||' '||last_name AS full_name,
  points FROM id_points
  JOIN volunteer
  ON volunteer.id = id_points.id
  ORDER BY points DESC LIMIT 50;
```

full_name text	points numeric
John Doe	7
Michael Johnson	6
Emily Jones	3
Olivia Davis	3
David Brown	3
William Anderson	3
Licha Blewis	1
Sophia Hill	1

Procedures and Functions

Interaction with the database

Triggers and indexes

Interaction with the database

```
CREATE OR REPLACE FUNCTION task_update_trigger()
RETURNS TRIGGER AS
$body$
DECLARE
    temp_id_spec INT;
BEGIN
    FOR temp_id_spec IN SELECT DISTINCT vt.volunteer_id
    FROM volunteer_task vt
    WHERE task_id = NEW.id
    LOOP
        INSERT INTO task_log (task_id, volunteer_id,
                              date, time, status_id)
        VALUES (NEW.id, temp_id_spec, CURRENT_DATE,
                CURRENT_TIME, NEW.status_id);
        END LOOP;
IF NEW.status_id =(SELECT id FROM status WHERE name = 'Cancelled')
    OR NEW.status_id =(SELECT id FROM status WHERE name = 'Finished')
    THEN
    DELETE FROM volunteer_task WHERE task_id = NEW. id;
    END IF;  RETURN NEW;
END; $body$ LANGUAGE plpgsql;

CREATE TRIGGER after_task_update
AFTER UPDATE ON task FOR EACH ROW
EXECUTE FUNCTION task_update_trigger();
```

Planning Time: 0.130 ms

Execution Time: 0.041 ms

Planning Time: 0.129 ms

Execution Time: 0.321 ms

Thank you!