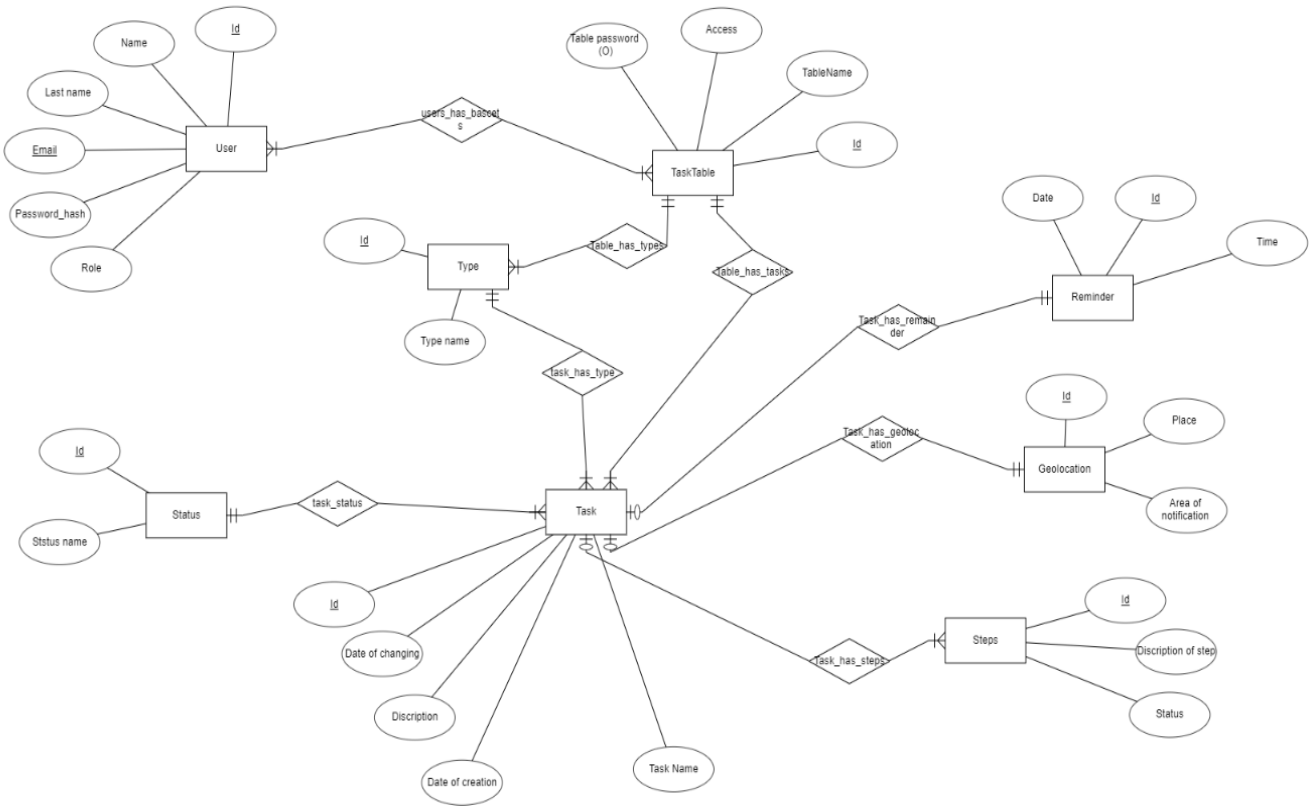
ZADANIE Z PREDMETU
DATABÁZOVÉ SYSTÉMY
NÁZOV DOMÉNY: PLÁNOVAČ DŇA

MENO A PRIEZVISKO: LIASHENKO OLEKSANDR
AK. ROK: 2023/2024

ENTITNO-RELAČNÝ DIAGRAM



OPIS ENTITNÝCH MNOŽÍN

Entita User má štyri atribúty:

- ID - Unikátny identifikátor usera,
- Name - meno používateľa
- Last name – prizvisko používateľa
- E-mail - jedinečný e-mail používateľa
- Heslo - heslo používateľa
- Role – role používateľa

Používateľ využíva našu aplikáciu

Enteti Task Table bude mať jedny atribút:

- ID - Unikátny identifikátor Task Table

Task Table sa používa na ukladanie úloh určitého usera, na zjednodušenie, zabezpečenie a oddelenie usera od úloh.

Enteti Task bude mať tri atribúty:

- ID - Unikátny identifikátor taska,
- Title - Názov našej úlohy
- Discription - Opis našej úlohy
- Date of creation
- Date of changing

Ulohy, ktoré bude mať náš používateľ

Enteti Type bude mať dva atribúty:

- ID - Unikátny identifikátor typu,
- Name - Názov typu

Typy úloh, ako napríklad: Práca, štúdium, cestovanie atď.

Enteti Status bude mať dva atribúty:

- ID - Unikátny identifikátor statusu,
- Name - Názov statusu

Status úlohy, napr: Ukončená, Odstránená, Prebieha atď.

Enteti Geolocation bude mať dva atribúty:

- ID - Unikátny identifikátor geolocationu,

- Place - miesto, kde bude používateľ dostávať oznámenia
- Area of notification - V akom okruhu od miesta dostane používateľ oznámenie

Geolocation je potrebný na pripomenutie používateľovi, aby niečo urobil v niejakom meste.

Enteti Remainder bude mať dva atribúty:

- ID - Unikátny identifikátor remaindru,
- Date_of_remainder - Dátum a čas, kedy sa má používateľovi pripomenúť táto úloha

Remaider je potrebný na pripomenutie používateľovi, aby niečo urobil (vypil vodu, čítal, zaplatil účet).

OPIS VZŤAHOV MEDZI ENTITNÝMI MNOŽINAMI

Každý User môže mať niekoľko Task Table a každý Task Table môže mať niekoľko Usera.

(Many-to-Many)

Task Table môže mať niekoľko Taskov, ale Task musí mať len jeden Task Table.

(One-to-Many)

Status musí mať niekoľko Taskov, ale Task musí mať len jeden Status.

(One-to-Many)

Type musí mať niekoľko Taskov, ale Task musí mať len jeden Type.

(One-to-Many)

Task Table môže mať niekoľko Typov, ale Type musí mať len jeden Task Table.

(One-to-Many)

Day môže mať niekoľko Taskov, ale Task musí mať len jeden Day na čo je naplánovaný.

(One-to-Many)

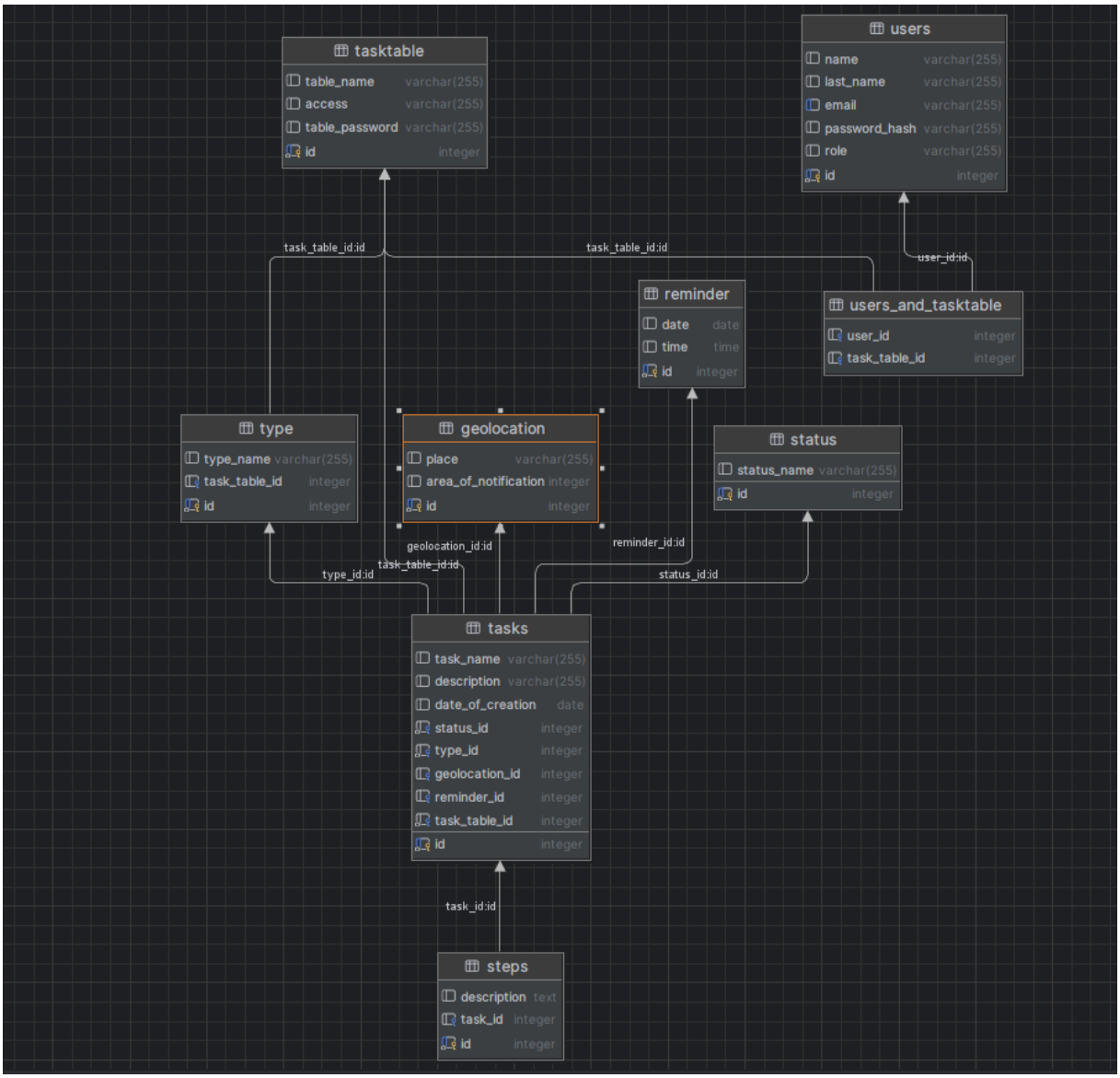
Task môže mať len jeden Remainder, a Remainder musí mať len jeden Task

(One-to-One)

Task môže mať len jeden Geolocation, a Geolocation musí mať len jeden Task

(One-to-One)

RELAČNÝ MODEL


tasktable
table_name varchar(255)
access varchar(255)
table_password varchar(255)
id integer
users
name varchar(255)
last_name varchar(255)
email varchar(255)
password_hash varchar(255)
role varchar(255)
id integer
task_table_id:id
task_table_id:id
user_id:id
reminder
date date
time time
id integer
users_and_tasktable
user_id integer
task_table_id integer
type
type_name varchar(255)
task_table_id integer
id integer
geolocation
place varchar(255)
area_of_notification integer
id integer
status
status_name varchar(255)
id integer
geolocation_id:id
task_table_id:id
reminder_id:id
type_id:id
status_id:id
tasks
task_name varchar(255)
description varchar(255)
date_of_creation date
status_id integer
type_id integer
geolocation_id integer
reminder_id integer
task_table_id integer
id integer
task_id:id
steps
description text
task_id integer
id integer

PRVÁ ODOVZDÁVKA - SQL SKRIPT NA VYTVORENIE DATABÁZY

```
CREATE TABLE Users (
    Id INT PRIMARY KEY,
    Name varchar(255),
    Last_name varchar(255),
    Email varchar(255) unique,
    Password_hash varchar(255),
    Role varchar(255) default 'User'
);

CREATE TABLE TaskTable (
    Id int primary key ,
    Table_Name varchar(255),
    Access varchar(255) default 'Public',
    Table_password VARCHAR(255) default null
);

CREATE TABLE Users_and_TaskTable(
    User_id INT REFERENCES Users(Id),
    Task_table_id INT REFERENCES TaskTable(Id)

);



CREATE TABLE Status (
    Id INT PRIMARY KEY,
    Status_name VARCHAR(255)
);

CREATE TABLE Type (
    Id INT PRIMARY KEY,
    Type_name VARCHAR(255),
    Task_table_id INT default null,
    FOREIGN KEY (Task_table_id) REFERENCES TaskTable(Id)
);

CREATE TABLE Geolocation (
    Id INT PRIMARY KEY,
    Place VARCHAR(255),
    Area_of_notification int default 100
);

CREATE TABLE Reminder (
    Id INT PRIMARY KEY,
    Date date,
    Time time
);




CREATE TABLE Tasks (
```

```
    Id int primary key,
    Task_Name varchar(255),
    Description varchar(255) default null,
    Date_of_creation date DEFAULT CURRENT_TIMESTAMP,
    Status_Id int not null default 1,
    Type_Id int not null default 1,
    Geolocation_Id int default null,
    Reminder_Id int default null,
    Task_table_id int not null,
    FOREIGN KEY (Task_table_id) REFERENCES TaskTable(Id),
    FOREIGN KEY (Status_Id) REFERENCES Status(Id),
    FOREIGN KEY (Type_Id) REFERENCES Type(Id),
    FOREIGN KEY (Reminder_Id) REFERENCES Reminder(Id),
    FOREIGN KEY (Geolocation_Id) REFERENCES Geolocation(Id)
);




CREATE TABLE Steps (
    Id INT PRIMARY KEY,
    Description TEXT,
    Task_Id INT,
    FOREIGN KEY (Task_Id) REFERENCES Tasks(Id)
);




TRUNCATE Table Users cascade;
INSERT INTO Users (Id, Name, Last_name, Email, Password_hash, Role)
VALUES
    (1, 'John', 'Doe', 'john@example.com', 'password_1',  'Admin'),
    (2, 'Jane', 'Smith', 'jane@example.com', 'password_2', default),
    (3, 'Alice', 'Johnson', 'alice@example.com', 'password_3', default),
    (4, 'Bob', 'Brown', 'bob@example.com', 'password_4', default),
    (5, 'Emily', 'Davis', 'emily@example.com', 'password_5', default);



TRUNCATE Table TaskTable cascade;
INSERT INTO TaskTable (Id, Table_Name)
VALUES
    (1, 'Jons Plans '),
    (2, 'Jane Plans '),
    (3, 'Family Plans'),
    (4, 'Work Plans'),
    (5, 'Bobs Plans');


TRUNCATE Table Users_and_TaskTable cascade;
INSERT INTO Users_and_TaskTable (User_id, Task_table_id)
VALUES
    (1, 1),
    (1, 3),
    (1, 4),
    (2, 2),
    (2, 3),
    (2, 4),
    (3, 3),
```

```
    (4, 3),
    (5, 4),
    (4,5);


TRUNCATE Table Type cascade;
INSERT INTO Type (Id, Type_name, Task_table_id)
VALUES
    (1, 'My Tasks', null),
    (2, 'Work', 4),
    (3, 'Study',3),
    (4, 'Health',2),
    (5, 'Gym',1),
    (6, 'Meeting',4),
    (7, 'Home',3),
    (8, 'Children', 2),
    (9, 'Study',1),
    (10, 'Plans to make 100000000 money',5);



TRUNCATE Table Status cascade;
INSERT INTO Status (Id, Status_name)
VALUES
    (1, 'In Progress'),
    (2, 'Completed'),
    (3, 'Canceled'),
    (4, 'Planned'),
    (5, 'Delayed'),
    (6, 'Deleted');

TRUNCATE Table Geolocation cascade;
INSERT INTO Geolocation (Id, Place)
VALUES
    (1, 'Home'),
    (2, 'Office'),
    (3, 'School'),
    (4, 'Gym'),
    (5, 'Supermarket'),
    (6, 'Park'),
    (7, 'Library');

TRUNCATE Table Reminder cascade;
INSERT INTO Reminder (Id, Date, Time)
VALUES
    (1, '2024-04-03', '08:00:00'),
    (2, '2024-04-04', '12:30:00'),
    (3, '2024-04-05', '18:00:00'),
    (4, '2024-04-06', '09:45:00'),
    (5, '2024-04-07', '15:20:00'),
    (6, '2024-04-08', '10:00:00'),
    (7, '2024-04-09', '18:00:00'),
    (8, '2024-08-15', '00:00:00');;

TRUNCATE Table Tasks cascade;
INSERT INTO Tasks (Id, Task_Name, Description, Date_of_creation, Status_Id,
Type_Id, Geolocation_Id, Reminder_Id, Task_table_id)
VALUES
    (1, 'Buy groceries', 'Milk, eggs, bread', '2024-04-03', 3, 5, 5, null,
3),
    (2, 'Finish report', 'Complete the quarterly report', '2024-04-04', 2,
```

```
2, null, 2, 4),
    (3, 'Gym workout', 'Cardio and weightlifting', '2024-04-05', 2, 4, 4,
3, 1),
    (4, 'Study for exam', 'Review chapters 1-5', '2024-04-06', 1, 3, 3, 4,
1),
    (5, 'Meeting with client', 'Discuss project updates', '2024-04-07', 2,
6, 2, 5, 4),
    (6, 'Clean house', 'Vacuum, dust, and mop', '2024-04-08', 1, 7, 1, 6,
3),
    (7, 'Read book', 'Chapter 6-10', '2024-04-09', default, default, 7 ,
2,2),
    (8, 'Buy Present', null, '2024-08-10', default, default, null, 8,2),
    (9, 'Make money', 'many money', '2024-04-09', default, default, null,
2, 5);



TRUNCATE Table Steps cascade;
INSERT INTO Steps (Id, Description, Task_Id)
VALUES
    (1, 'Buy milk', 1),
    (2, 'Buy eggs', 1),
    (3, 'Buy bread', 1),
    (4, 'Review previous reports', 2),
    (5, 'Gather data', 2),
    (6, 'Write analysis', 2),
    (7, 'Warm-up', 3),
    (8, 'Cardio session', 3),
    (9, 'Weightlifting', 3),
    (10, 'Review chapter 1', 4),
    (11, 'Review chapter 2', 4),
    (12, 'Review chapter 3', 4),
    (13, 'Review chapter 4', 4),
    (14, 'Review chapter 5', 4),
    (15, 'Prepare presentation slides', 5),
    (16, 'Prepare meeting agenda', 5),
    (17, 'Vacuum living room', 6),
    (18, 'Dust furniture', 6),
    (19, 'Mop floors', 6),
    (20, 'Read chapter 6', 7),
    (21, 'Read chapter 7', 7),
    (22, 'Read chapter 8', 7),
    (23, 'Read chapter 9', 7),
    (24, 'Read chapter 10', 7);



-- View 1: Find all users named 'Jane'
CREATE OR REPLACE VIEW Users_Jane AS
SELECT * FROM Users WHERE Name = 'Jane';

-- View 2: Find all tasks from the TaskTable where the name starts with
'Jons'
CREATE OR REPLACE VIEW TaskTable_Jons AS
SELECT * FROM TaskTable WHERE Table_Name LIKE 'Jons%';

-- View 3: Merge the Tasks and Status tables
CREATE OR REPLACE VIEW Tasks_Status AS
SELECT t.*, s.Status_name
FROM Tasks t
JOIN Status s ON t.Status_Id = s.Id;
```

```sql
--  View 4: Merge the Users, Users_and_TaskTable, and TaskTable tables
CREATE OR REPLACE VIEW Users_Tasks AS
SELECT u.Name, tt.Table_Name, t.Task_Name
FROM Users u
JOIN Users_and_TaskTable ut ON u.Id = ut.User_id
JOIN TaskTable tt ON ut.Task_table_id = tt.Id
JOIN Tasks t ON t.Task_table_id = tt.Id;

-- View 5: External join of Type, Tasks, and Geolocation tables
CREATE OR REPLACE VIEW Type_Tasks_Geolocation AS
SELECT t.Task_Name, ty.Type_name, g.Place
FROM Tasks t
LEFT JOIN Type ty ON t.Type_Id = ty.Id
LEFT JOIN Geolocation g ON t.Geolocation_Id = g.Id;

-- 6: Using the COUNT function to count tasks of each type
CREATE OR REPLACE VIEW Task_Count_By_Type AS
SELECT t.Type_Id, ty.Type_Name, COUNT(*) AS Task_Count
FROM Tasks t
LEFT JOIN Type ty ON t.Type_Id = ty.Id
GROUP BY t.Type_Id, ty.Type_Name;

--7
CREATE OR REPLACE VIEW detailed_tasks AS
SELECT t.task_name,
       t.description,
       count(st.description) AS step_count,
       s.status_name,
       ty.type_name,
       g.place               AS geolocation,
       u.name                AS assigned_user
FROM tasks t
         JOIN status s ON t.status_id = s.id
         JOIN type ty ON t.type_id = ty.id
         JOIN steps st ON t.id = st.task_id
         LEFT JOIN geolocation g ON t.geolocation_id = g.id
         JOIN users_and_tasktable ut ON t.task_table_id = ut.task_table_id
         JOIN users u ON ut.user_id = u.id
GROUP BY t.task_name, t.description, s.status_name, ty.type_name, g.place,
u.name;


--8
CREATE OR REPLACE VIEW tasks_completion_status AS
SELECT task_name,
       CASE
           WHEN status_id = 2 THEN 'Completed'::text
           ELSE 'Not Completed'::text
           END AS completion_status
FROM tasks t;


--9
CREATE OR REPLACE VIEW tasks_status AS
    SELECT t.id,
       t.task_name,
       t.description,
       t.date_of_creation,
       t.status_id,
       t.type_id,
```

```
       t.geolocation_id,
       t.reminder_id,
       t.task_table_id,
       s.status_name
FROM tasks t
JOIN status s ON t.status_id = s.id;

--10
CREATE VIEW Completed_Tasks AS
SELECT *
FROM Tasks
WHERE Status_Id = (SELECT Id FROM Status WHERE Status_name = 'Completed')
UNION
SELECT *
FROM Tasks
WHERE Status_Id = (SELECT Id FROM Status WHERE Status_name = 'Canceled');


--10
CREATE VIEW Work_Tasks AS
SELECT *
FROM Tasks
WHERE Task_table_id = (SELECT Id FROM TaskTable WHERE Table_Name = 'Work
Plans');

--11
CREATE OR REPLACE VIEW task_steps AS
    SELECT t.task_name,
       t.description,
       s.description AS step_description
FROM tasks t
         LEFT JOIN steps s ON t.id = s.task_id;



--12
CREATE OR REPLACE VIEW tasks_timeline AS
SELECT t.task_name,
       t.date_of_creation,
       s.status_name
FROM tasks t
         JOIN status s ON t.status_id = s.id
ORDER BY t.date_of_creation DESC;




--Ensures timely management of tasks by automatically marking them as
delayed if their reminder dates have passed.
CREATE OR REPLACE FUNCTION check_reminder_expiry() RETURNS TRIGGER AS $$
DECLARE
    reminder_date DATE;
BEGIN
    IF NEW.Reminder_Id IS NOT NULL THEN
        SELECT Date INTO reminder_date FROM Reminder WHERE Id =
NEW.Reminder_Id;

        IF CURRENT_DATE > reminder_date THEN
            NEW.Status_Id = 5;
        END IF;
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;



CREATE TRIGGER reminder_expiry_trigger
BEFORE INSERT OR UPDATE
ON tasks
FOR EACH ROW
EXECUTE FUNCTION check_reminder_expiry();

--INSERT INTO tasks (Id, Reminder_Id, Status_Id)
--VALUES (1, 1, 3);


--Ensures that steps associated with a task are properly cleaned up when
the task itself is deleted, preventing orphaned data.

CREATE OR REPLACE FUNCTION delete_associated_steps() RETURNS TRIGGER AS $$
BEGIN
    DELETE FROM Steps WHERE Task_Id = OLD.Id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_associated_steps_trigger
BEFORE DELETE ON tasks
FOR EACH ROW
EXECUTE FUNCTION delete_associated_steps();

--DELETE FROM tasks WHERE Id = 1;

-- Guarantees that each user's email address remains unique within the
system, preventing duplication and ensuring proper identification

CREATE OR REPLACE FUNCTION enforce_unique_email() RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Users
        WHERE Email = NEW.Email AND Id <> NEW.Id
    ) THEN
        RAISE EXCEPTION 'Email must be unique';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_unique_email_trigger
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW
EXECUTE FUNCTION enforce_unique_email();


-- INSERT INTO Users (Id, Email, Name)
-- VALUES (1, 'example@example.com', 'John Doe');
```

```
--Enhances data security by ensuring that tables with private access are
protected with a password, restricting unauthorized access.

CREATE OR REPLACE FUNCTION require_table_password() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Access = 'Private' AND NEW.Table_password IS NULL THEN
        RAISE EXCEPTION 'Table password is required for private access.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER table_password_requirement_trigger
BEFORE INSERT OR UPDATE ON tasktable
FOR EACH ROW
EXECUTE FUNCTION require_table_password();




CREATE SEQUENCE users_id_seq START 1;

CREATE SEQUENCE tasktable_id_seq START 1;

CREATE SEQUENCE users_and_tasktable_id_seq START 1;

CREATE SEQUENCE status_id_seq START 1;

CREATE SEQUENCE type_id_seq START 1;

CREATE SEQUENCE geolocation_id_seq START 1;

CREATE SEQUENCE reminder_id_seq START 1;

CREATE SEQUENCE tasks_id_seq START 1;

CREATE SEQUENCE steps_id_seq START 1;




-- These triggers auto-increment for  Users
CREATE OR REPLACE FUNCTION auto_increment_users()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Users);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_users
BEFORE INSERT ON Users
FOR EACH ROW
EXECUTE FUNCTION auto_increment_users();

-- These triggers auto-increment for  TaskTable
CREATE OR REPLACE FUNCTION auto_increment_tasktable()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM TaskTable);
    RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_tasktable
BEFORE INSERT ON TaskTable
FOR EACH ROW
EXECUTE FUNCTION auto_increment_tasktable();

-- These triggers auto-increment for Users_and_TaskTable
CREATE OR REPLACE FUNCTION auto_increment_users_and_tasktable()
RETURNS TRIGGER AS $$
BEGIN
    NEW.user_id := (SELECT COALESCE(MAX(user_id), 0) + 1 FROM
Users_and_TaskTable);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_users_and_tasktable
BEFORE INSERT ON Users_and_TaskTable
FOR EACH ROW
EXECUTE FUNCTION auto_increment_users_and_tasktable();

-- These triggers auto-increment for  Status
CREATE OR REPLACE FUNCTION auto_increment_status()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Status);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_status
BEFORE INSERT ON Status
FOR EACH ROW
EXECUTE FUNCTION auto_increment_status();

-- These triggers auto-increment for  Type
CREATE OR REPLACE FUNCTION auto_increment_type()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Type);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_type
BEFORE INSERT ON Type
FOR EACH ROW
EXECUTE FUNCTION auto_increment_type();

-- These triggers auto-increment for  Geolocation
CREATE OR REPLACE FUNCTION auto_increment_geolocation()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Geolocation);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_geolocation
```

```
BEFORE INSERT ON Geolocation
FOR EACH ROW
EXECUTE FUNCTION auto_increment_geolocation();

-- These triggers auto-increment for Reminder
CREATE OR REPLACE FUNCTION auto_increment_reminder()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Reminder);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_reminder
BEFORE INSERT ON Reminder
FOR EACH ROW
EXECUTE FUNCTION auto_increment_reminder();

--These triggers auto-increment for  Tasks
CREATE OR REPLACE FUNCTION auto_increment_tasks()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Tasks);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_tasks
BEFORE INSERT ON Tasks
FOR EACH ROW
EXECUTE FUNCTION auto_increment_tasks();

-- These triggers auto-increment for Steps
CREATE OR REPLACE FUNCTION auto_increment_steps()
RETURNS TRIGGER AS $$
BEGIN
    NEW.id := (SELECT COALESCE(MAX(id), 0) + 1 FROM Steps);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_auto_increment_steps
BEFORE INSERT ON Steps
FOR EACH ROW
EXECUTE FUNCTION auto_increment_steps();


--This procedure searches for tasks based on a provided keyword. It looks
for matches in the task name or description fields.
CREATE OR REPLACE PROCEDURE find_tasks_by_keyword(IN p_keyword VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
    EXECUTE 'SELECT * FROM Tasks WHERE Task_Name LIKE ''%' || p_keyword ||
'%'' OR Description LIKE ''%' || p_keyword || '%'';';

END;
$$;

--CALL find_tasks_by_keyword('Buy');
```

```sql
--his procedure retrieves tasks associated with a specific user email. It
fetches task names, descriptions, and their statuses for the given user
CREATE OR REPLACE PROCEDURE find_tasks_by_user_email(IN p_user_email
VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT t.Task_Name, t.Description, s.Status_name
    FROM Users u
    JOIN Users_and_TaskTable ut ON u.Id = ut.User_id
    JOIN TaskTable tt ON ut.Task_table_id = tt.Id
    JOIN Tasks t ON t.Task_table_id = tt.Id
    JOIN Status s ON t.Status_Id = s.Id
    WHERE u.Email = p_user_email;
END;
$$;


CREATE OR REPLACE PROCEDURE find_tasks_by_user_id(IN p_user_id INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT t.Task_Name, t.Description, s.Status_name
    FROM Users_and_TaskTable ut
    JOIN TaskTable tt ON ut.Task_table_id = tt.Id
    JOIN Tasks t ON t.Task_table_id = tt.Id
    JOIN Status s ON t.Status_Id = s.Id
    WHERE ut.User_id = p_user_id;
END;
$$;


CREATE OR REPLACE PROCEDURE find_tasks_with_overdue_reminders(IN p_today
DATE)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT *
    FROM Tasks t
    JOIN Reminder r ON t.Reminder_Id = r.Id
    WHERE r.Date < p_today;
END;
$$;


CREATE OR REPLACE PROCEDURE update_task_status(IN p_task_id INTEGER, IN
p_new_status_id INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Tasks
    SET Status_Id = p_new_status_id
    WHERE Id = p_task_id;
END;
$$;
--CALL update_task_status(2,1)

CREATE OR REPLACE PROCEDURE update_task_type(IN p_task_id INTEGER, IN
p_new_type_id INTEGER)
LANGUAGE plpgsql
AS $$
```

```
BEGIN
    UPDATE Tasks
    SET Type_Id = p_new_type_id
    WHERE Id = p_task_id;
END;
$$;

--CALL update_task_type(2,1)
```