Carlos Amador A01329447
Mónica Pérez A01329619

Bases de datos avanzadas

Tarea 2.3

10.50.67.83

Gimnasio

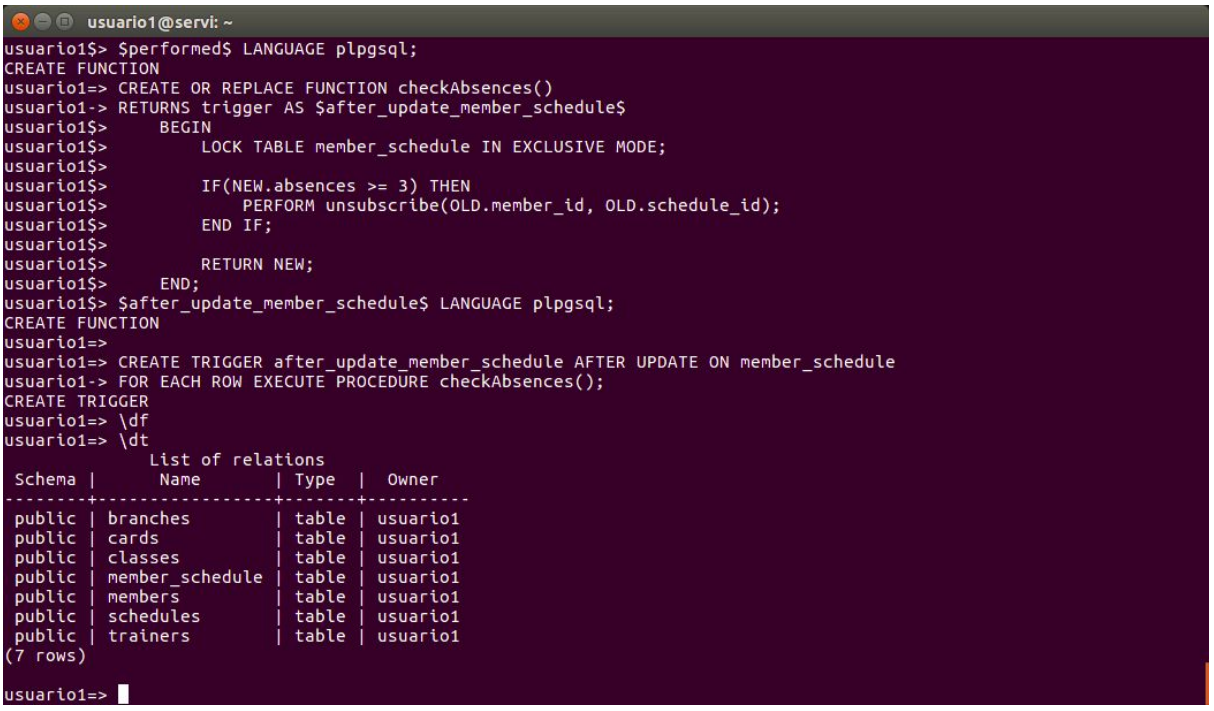
Índice

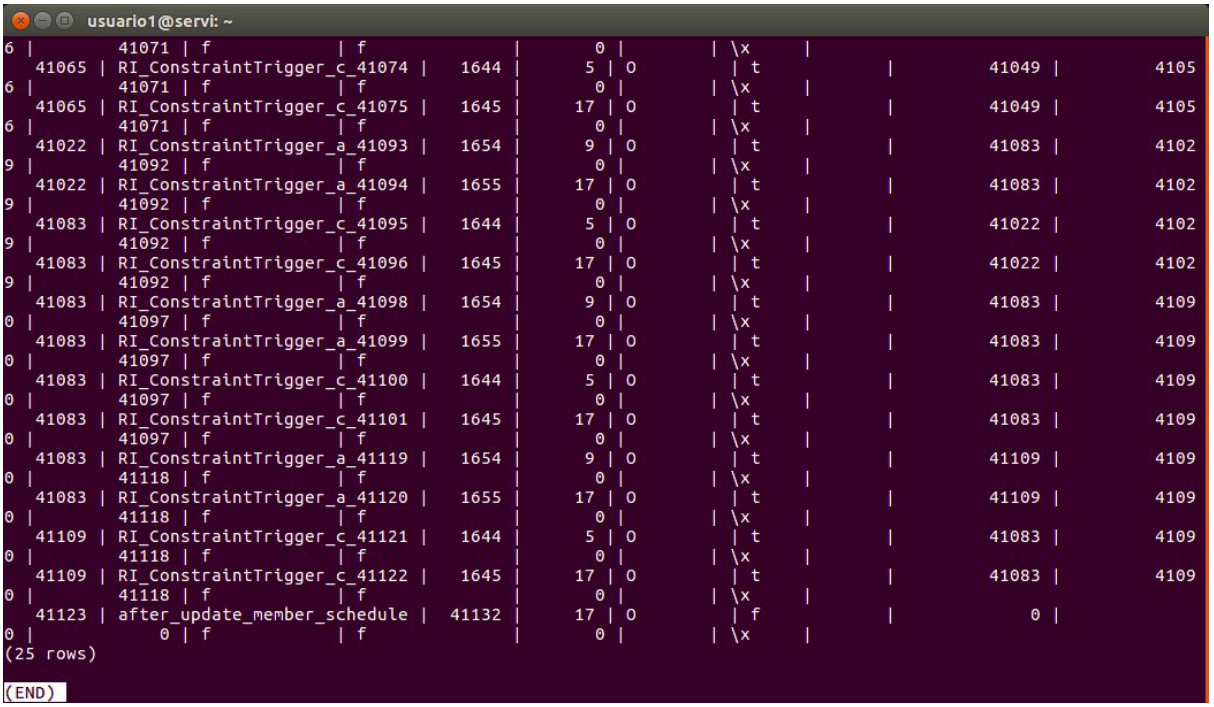
Descripción

Es una base de datos modelada a partir de la idea de un gimnasio. La base de datos está implementada en Postgresql.

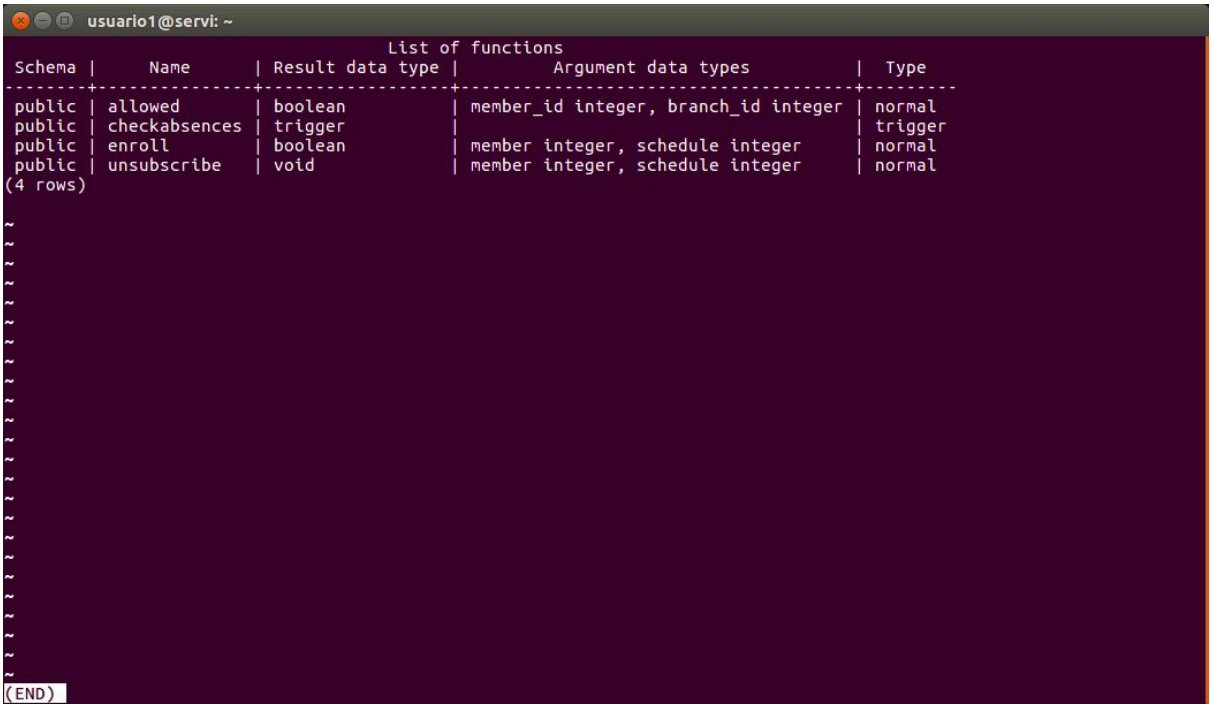
Montado de base en servidor

Para montar la base de datos en el servidor, hicimos ssh, accedimos a postgresql, y luego a la base “usuario1”. Finalmente, insertamos los comandos necesarios. Capturas de pantalla:



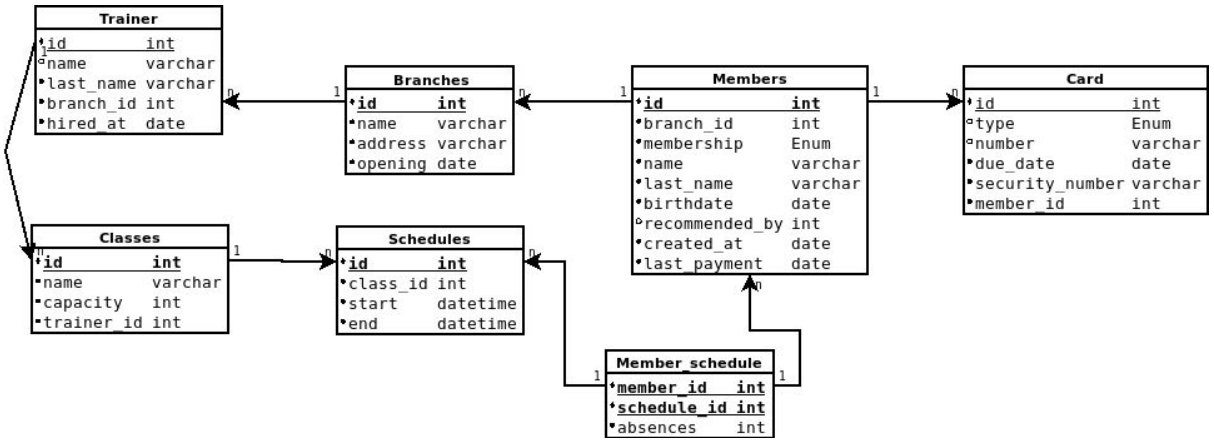


Al final de la captura anterior, se puede ver el trigger insertado.



Base de datos

Este es el diagrama relacional de la base de datos.



En cada tabla se cuenta con una llave primaria de tipo serial (único y que se incrementa cada vez que se inserta en la tabla), podemos observar que todos los atributos de la tabla tienen la restricción de valor no nulo, excepto el atributo recommended_by en la tabla de miembros, ya que puede no haber llegado por recomendación de alguien. También utilizamos dos enumerados para guardar los tipos de tarjeta y de membresía.

Los comandos utilizados para crear las tablas de la base son los siguientes:

- Tabla sucursales.

```
CREATE TABLE branches(
    id serial PRIMARY KEY,
    name text NOT NULL,
    address text NOT NULL,
    opening date NOT NULL
);
```

- Tabla entrenadores.

```
CREATE TABLE trainers(
    id serial PRIMARY KEY,
    name text NOT NULL,
    last_name text NOT NULL,
    branch_id int REFERENCES branches(id),
    hired_at date NOT NULL
);
```

- Tabla de clases.

```
CREATE TABLE classes(
    id serial PRIMARY KEY,
    name text NOT NULL,
    capacity int NOT NULL,
    trainer_id int REFERENCES trainers(id)
);
```

- Tabla de horarios de clase.

```
CREATE TABLE schedules(
    id serial PRIMARY KEY,
    class_id int REFERENCES classes(id),
    start_time timestamp NOT NULL,
    end_time timestamp NOT NULL
);
```

- Tabla de miembros del gimnasio.

```
CREATE TABLE members(
    id serial PRIMARY KEY,
    branch_id int REFERENCES branches(id),
    membership membership_type NOT NULL,
    name text NOT NULL,
    last_name text NOT NULL,
    birthdate date NOT NULL,
    recommended_by int REFERENCES members(id) NULL,
    created_at timestamp,
    last_payment date
);
```

- Tabla de tarjetas de los miembros.

```
CREATE TABLE cards(
    id serial PRIMARY KEY,
    card_t card_type NOT NULL,
    number text NOT NULL,
    due_date date NOT NULL,
    security_number text NOT NULL,
    member_id int REFERENCES members(id)
);
```

- Tabla cruce de miembros y horarios.

```
CREATE TABLE member_schedule(
    member_id int NOT NULL,
    schedule_id int NOT NULL,
    absences int NOT NULL DEFAULT (0),
    PRIMARY KEY (member_id, schedule_id)
);
```

- Enumerado de tipos de membresía.

```
CREATE TYPE membership_type AS ENUM ('standard', 'premium');
```

- Enumerado de tipos de tarjeta.

```
CREATE TYPE card_type AS ENUM ('debit', 'credit');
```

Código almacenado

Functions

- allowed (member_id INTEGER, branch_id INTEGER): Existen dos tipos de membresía, la standard y la premium. Si un usuario tiene membresía standard, sólo se puede inscribir a clases de su gimnasio; si es premium, puede en el que sea. De acuerdo a lo anterior, retorna true o false si un miembro puede atender a un branch de un gimnasio específico.
- checkAbsences(): es la función del trigger.

Ejemplo

```sql
/*Función que determina si el usuario puede ingresar o no a una sucursal dependiendo del tipo de membresía que tinene.*/
CREATE OR REPLACE FUNCTION allowed (member_id INTEGER, branch_id INTEGER)
RETURNS boolean AS $allowed$
    DECLARE
        allowed boolean;
        member_branch_id integer;
        membership membership_type;
    BEGIN
        SELECT m.branch_id, m.membership
        INTO member_branch_id, membership
        FROM members m
        WHERE m.id = member_id;

        allowed := true;

        IF(membership = 'premium'::membership_type) THEN
            allowed := true;
        ELSIF (member_branch_id = branch_id) THEN
            allowed := true;
        ELSE
            allowed:= false;
        END IF;

        RETURN allowed;
    END;
$allowed$ LANGUAGE plpgsql;
```

Procedures

- enroll (member INTEGER, schedule INTEGER): De acuerdo al id de un miembro y el de un schedule, intenta inscribir a un miembro a un schedule, lo cual implica verificar que se pueda inscribir en el gimnasio en el que está el schedule, así como verificar si hay cupo. Incluye una transacción para no hacer dirty reads del cupo.
- unsubscribe (member INTEGER, schedule INTEGER): Realiza lo opuesto a enroll, dando de baja a un miembro de un schedule.

Ejemplo

```
/*Desinscribe a un miembro de una clase*/
CREATE OR REPLACE FUNCTION unsubscribe(member INTEGER, schedule INTEGER)
RETURNS void AS $$
    BEGIN
        LOCK TABLE member_schedule IN EXCLUSIVE MODE;

        DELETE FROM member_schedule ms
        WHERE ms.member_id = member AND ms.schedule_id = schedule;

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

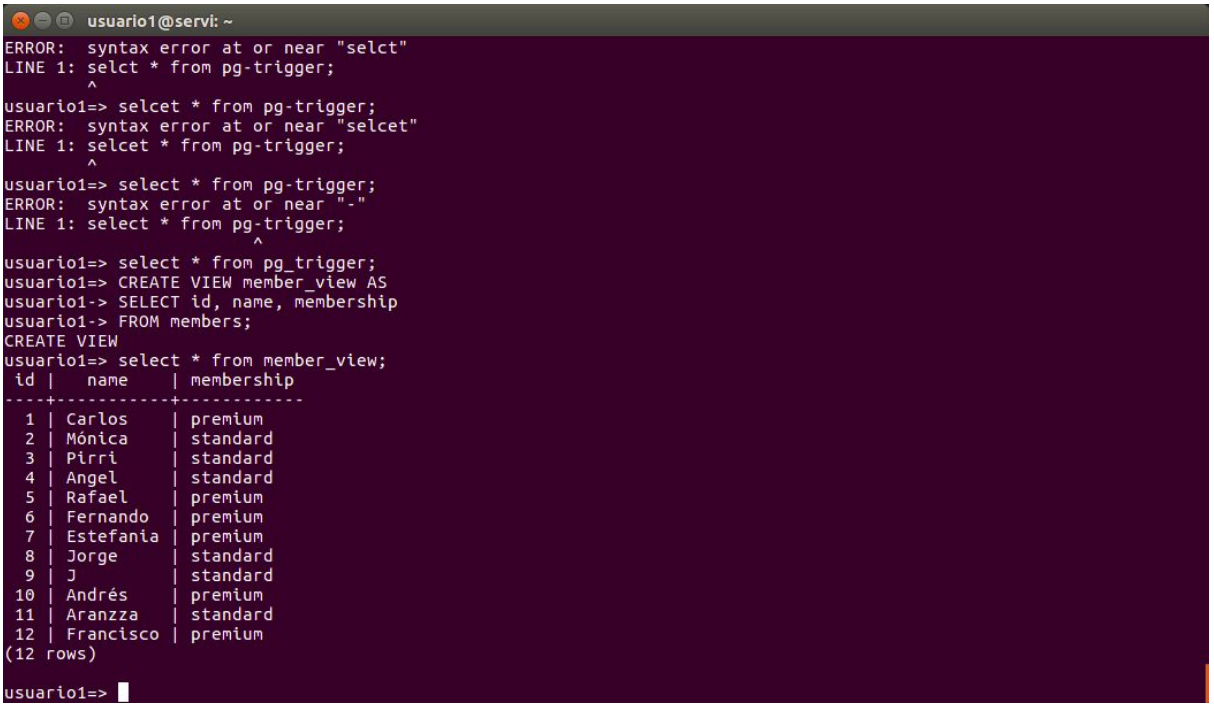
Triggers

- after_update_member_schedule: cuando un miembro llega a 3 faltas en un schedule, el sistema lo da de baja utilizando este trigger, llamando checkAbsences para contar las faltas.

```
/*Desinscribe a un miembro de una clase*/
CREATE OR REPLACE FUNCTION checkAbsences()
RETURNS trigger AS $after_update_member_schedule$
    BEGIN
        LOCK TABLE member_schedule IN EXCLUSIVE MODE;

        IF(NEW.absences >= 3) THEN
            PERFORM unsubscribe(OLD.member_id, OLD.schedule_id);
        END IF;

        RETURN NEW;
    END;
$after_update_member_schedule$ LANGUAGE plpgsql;

CREATE TRIGGER after_update_member_schedule AFTER UPDATE ON member_schedule
FOR EACH ROW EXECUTE PROCEDURE checkAbsences();
```

Views

Realizamos una view que permite visualizar de manera compacta la lista de los miembros del gimnasio, mostrando el id, nombre y tipo de membresía de los mismos.


usuario1@servi: ~
ERROR: syntax error at or near "selct"
LINE 1: selct * from pg-trigger;
        ^
usuario1=> selcet * from pg-trigger;
ERROR: syntax error at or near "selcet"
LINE 1: selcet * from pg-trigger;
        ^
usuario1=> select * from pg-trigger;
ERROR: syntax error at or near "-"
LINE 1: select * from pg-trigger;
                      ^
usuario1=> select * from pg_trigger;
usuario1=> CREATE VIEW member_view AS
usuario1-> SELECT id, name, membership
usuario1-> FROM members;
CREATE VIEW
usuario1=> select * from member_view;
 id | name | membership
----+-----------+------------
  1 | Carlos | premium
  2 | Mónica | standard
  3 | Pirri | standard
  4 | Angel | standard
  5 | Rafael | premium
  6 | Fernando | premium
  7 | Estefania | premium
  8 | Jorge | standard
  9 | J | standard
 10 | Andrés | premium
 11 | Aranzza | standard
 12 | Francisco | premium
(12 rows)
usuario1=>