Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Proyecto Final

Administración y Diseño de Bases de Datos

Título del proyecto.

Gestión de una red social

Héctor Abreu Acosta
(alu0101068855@ull.edu.es)
Manuel Ramón Regalado Peraza
(alu0100283433@ull.edu.es)
Marcos Jesús Santana Ramos
(alu0101033471@ull.edu.es)

Índice:

Objetivos del proyecto

Los objetivos de esta base de datos de gestión de una red social son:

- Almacenar y organizar toda la información necesaria para el correcto funcionamiento de la red social.
- Proporcionar una fuente centralizada y confiable de información para la red social.
- Mejorar la eficiencia y productividad al evitar la necesidad de ingresar y mantener datos de manera manual.
- Facilitar el control y la administración de los datos a través de permisos de acceso y restricciones.

Además, la base de datos permite hacer seguimientos y registro de cambios en los datos a lo largo del tiempo, lo que podría ser útil para hacer un análisis y mejorar la experiencia de los usuarios en la red social.

Contexto de la base de datos

La base de datos está diseñada para una red social en línea que permite a los usuarios conectarse con sus amigos y compartir contenido. Entre sus principales características, los usuarios pueden crear perfiles con información personal, y pueden enviar y recibir mensajes privados de otros usuarios. También pueden publicar mensajes y ver las publicaciones de sus amigos. Los usuarios también pueden comentar en las publicaciones de otros usuarios y expresar su apreciación por ellos a través de un sistema de "me gusta". Además de la participación en grupos, la creación de sugerencias y el registro de estadísticas sencillas.

Sobre el contenido de la base de datos, se pretende almacenar la siguiente información:

- Usuarios: Incluir información sobre los usuarios de la red social, como sus nombres de usuario, direcciones de correo electrónico, números de teléfono, fechas de nacimiento.
- Amigos: Información sobre las conexiones de amigos entre los usuarios.

- Publicaciones: Información sobre las publicaciones de los usuarios, como el texto del mensaje.

- Comentarios: Incluir información sobre los comentarios de los usuarios en las publicaciones de otras personas.

- Mensajes privados: Información sobre los mensajes privados enviados entre los usuarios.

- Grupos: Información sobre los grupos creados por los usuarios.

- Notificaciones: Incluir información sobre las notificaciones enviadas a los usuarios, como la fecha y hora y el tipo de notificación.

- Me gustas: Información sobre los me gusta a publicaciones de otros usuarios.

- Bloqueados: Almacenaría información sobre los usuarios bloqueados por otros usuarios.

- Sugerencias: Información sobre las sugerencias enviadas por los usuarios.

- Estadística: Almacenaría información estadística sobre el uso de la red social, como el número de usuarios registrados y el número de publicaciones realizadas.

En cuanto a los requisitos de la base de datos, se espera que cumpla con los siguientes modelos de datos:

- Entidades débiles: Una entidad débil podría ser la entidad de estadísticas, su existencia depende de la existencia de otra entidad en el modelo. En este caso, la entidad de estadísticas estaría relacionada con la entidad de usuario a través de una relación uno a uno.

- Relaciones triples: Una relación triple podría ser la relación entre los usuarios, las publicaciones y los comentarios. Esta relación se podría expresar de la siguiente manera:

  Un usuario tiene muchas publicaciones.
  Una publicación tiene muchos comentarios.
  Un usuario tiene muchos comentarios.

  Esta relación triple indica que cada usuario puede tener muchas publicaciones y comentarios, y que cada publicación puede tener muchos

comentarios. Además, cada comentario está relacionado con un usuario y una publicación en particular.

Por ejemplo, si Juan hace una publicación en su perfil y Maria hace un comentario en esa publicación, entonces se establece una relación triple entre Juan (usuario), la publicación de Juan (publicación) y el comentario de Maria (comentario).

- Tipos IS_A: Cada notificación puede ser de un tipo determinado, como comentario, mensaje, solicitud de amistad, etc. Se podría implementar mediante una IS_A que esté relacionado con la entidad "Notificaciones".

- Relaciones 1:N: La relación entre Usuarios y Publicaciones es un ejemplo de relación 1:N, ya que cada usuario puede publicar varias publicaciones, pero cada publicación solo puede tener un usuario asociado.

- Relaciones M:M: La relación entre Usuarios y grupos es un ejemplo de relación M:M, ya que cada usuario puede tener varios grupos y cada grupo puede tener varios usuarios.

- En cuanto a los casos de inclusión, inclusividad, exclusión y exclusividad. Se aporta un caso de restricción de inclusividad, en el cual, un usuario para poder realizar una sugerencia, primero debe de haber creado una publicación. También se creó un caso de restricción de exclusión, se plantea que el usuario puede enviar o recibir una notificación en concreto, no puede realizar ambas sobre la misma notificación. Es decir, la entidad usuario sólo puede combinarse con la entidad notificación utilizando una interrelación (enviar o recibir).

Modelo Entidad Relación

Entidades

A continuación se representa toda la información aportada dentro del modelo entidad relación creado. Entidades con sus atributos y atributo principal (en negrita):

- Usuarios: **id_usuario**, nombre, correo electrónico, contraseña, fecha de nacimiento y teléfono.

- Amigos (Entidad débil): id del usuario que envía la solicitud, el id del usuario que recibe la solicitud y el estado de la solicitud (enviada, aceptada, rechazada).

- Publicaciones: **id de la publicación**, id del usuario que ha publicado, el contenido, la fecha y hora de publicación.

- Comentarios: **id del comentario,** el id del usuario que ha comentado, el id de la publicación a la que se refiere el comentario, el contenido y la fecha y hora de publicación.

- Mensajes privados: **id del mensaje**, id del usuario que envía el mensaje, el id del usuario que recibe el mensaje, el contenido y la fecha y hora de envío.

- Grupos: **id del grupo**, el nombre del grupo, la descripción, la fecha de creación y la lista de id de usuarios.

- Notificaciones: **id de la notificación,** la fecha y hora de envío, el tipo de notificación (comentario, mensaje, solicitud de amistad) y el id del usuario.

- Me gustas (Entidad débil): id del usuario que ha realizado el "me gusta" y el id de la publicación que ha recibido el "me gusta".

- Bloqueados (Entidad débil): el id del usuario bloqueador y el id del usuario bloqueado.

- Sugerencias: **id de la sugerencia**, id del usuario que ha enviado la sugerencia, el tipo de sugerencia (mejora, error, nueva función, etc.) y el contenido de la sugerencia.

- Estadísticas (Entidad débil): el número de publicaciones realizadas, el número de comentarios y "me gusta" recibidos. Los tres atributos son **atributos derivados**, ya que se generan automáticamente dependiendo de las publicaciones, comentarios o me gusta de un usuario.

Relaciones entre entidades

1. Usuarios - Amigos: Relación 1:N, desde Usuarios hacia Amigos. Cada usuario puede enviar o recibir varias solicitudes de amistad, pero cada solicitud está asociada a un único usuario.

2. Publicaciones - Comentarios: Relación 1:N, desde Publicaciones hacia Comentarios. Cada publicación puede tener varios comentarios, pero cada comentario está asociado a una única publicación.

3. Usuarios - Mensajes privados: Relación 1:N. Cada usuario puede tener varios mensajes privados y cada mensaje puede tener un usuario que lo recibe.

4. Grupos - Usuarios: Relación M:M, mediante la entidad intermedia Miembros. Cada grupo puede tener varios miembros y cada usuario puede pertenecer a varios grupos.

5. Publicaciones - Me gustas: Relación 1:N, desde Publicaciones hacia Me gustas. Cada publicación puede tener varios "me gusta", pero cada "me gusta" está asociado a una única publicación.

6. Usuarios - Notificaciones: Relación 1:N, desde Usuarios hacia Notificaciones. Cada usuario puede enviar o recibir varias notificaciones con **restricción de exclusión**, pero cada notificación está asociada a un único usuario.

7. Usuarios - Bloqueados: Relación 1:N, desde Usuarios hacia Bloqueados. Cada usuario puede tener varios bloqueos, pero cada bloqueo está asociado a un único usuario.

8. Usuarios - Sugerencias: Relación 1:N, desde Usuarios hacia Sugerencias. Cada usuario puede enviar varias sugerencias, pero cada sugerencia está asociada a un único usuario.

9. Usuarios - Estadísticas: Relación 1:1, desde Usuarios hacia Estadísticas. Cada usuario tiene una única fila de estadísticas asociada a él.

10. Usuarios - Publicaciones: Relación 1:N, desde Usuarios hacia Publicaciones. Cada usuario puede publicar varias publicaciones.

11. Usuarios - Comentario: Relación 1:N, desde Usuarios hacia Comentario. Cada usuario puede publicar o recibir varios comentarios.

Descripción de la base de datos en SQL e implementación en el Sistema Gestor de Bases de Datos Relacional PostgreSQL

Creación de la base de datos

Una vez se cambia de usuario a *postgres*, se inicia el *psql* y se crea la base de datos “redsocial” con el comando:

```
postgres-# CREATE DATABASE redsocial;
```

A continuación se crean las tablas de la base de datos. Tabla **usuarios**:

```
redsocial-#
CREATE TABLE usuarios (
id_usuario SERIAL PRIMARY KEY,
nombre VARCHAR(20) NOT NULL,
correo VARCHAR(50) UNIQUE NOT NULL,
contraseña VARCHAR(50) NOT NULL,
fecha_nacimiento DATE NOT NULL,
telefono VARCHAR(15) UNIQUE NOT NULL,
);
```

Tabla **notificaciones**:

```
CREATE TABLE notificaciones (
id_notificacion serial PRIMARY KEY,
id_usuario integer NOT NULL,
fecha_envio date DEFAULT CURRENT_DATE NOT NULL,
tipo char(20) NOT NULL,
recibe_envia char(10) NOT NULL,
CONSTRAINT fk_usuarios13
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario)
);
```

Tabla **bloqueados**:

```
CREATE TABLE bloqueados (
id_usuario_bloqueado integer,
id_usuario_bloqueador integer,
CONSTRAINT fk_usuarios
FOREIGN KEY(id_usuario_bloqueado)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_usuarios2
FOREIGN KEY(id_usuario_bloqueador)
REFERENCES usuarios(id_usuario)
);
```

Tabla **grupos**:

```
CREATE TABLE grupos (
id_grupo serial PRIMARY KEY,
nombre VARCHAR(50),
descripcion VARCHAR(50));
```

Tabla **amigos**: La restricción del tipo de solicitud se crea más adelante

```
CREATE TABLE amigos (
id_usuario_envia integer NOT NULL,
id_usuario_recibe integer NOT NULL,
es_solicitud varchar(9) NOT NULL,
CONSTRAINT fk_usuarios4
FOREIGN KEY(id_usuario_envia)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_usuarios5
FOREIGN KEY(id_usuario_recibe)
REFERENCES usuarios(id_usuario)
);
```

Tabla **estadisticas**:

```
CREATE TABLE estadisticas (
id_usuario integer NOT NULL,
num_comentarios integer NOT NULL DEFAULT 0,
num_megustas integer NOT NULL DEFAULT 0,
num_publicaciones integer NOT NULL DEFAULT 0,
CONSTRAINT fk_usuarios6
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario)
);
```

Tabla **miembros_grupos**:

```
CREATE TABLE miembros_grupos (
id_grupo int NOT NULL,
id_usuario int NOT NULL,
CONSTRAINT fk_usuarios3
```

```
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_grupo
FOREIGN KEY(id_grupo)
REFERENCES grupos(id_grupo)
);
```

Tabla **mensajes_privados**:

```
CREATE TABLE mensajes_privados (
id_mensaje serial PRIMARY KEY,
id_usuario_envia integer NOT NULL,
id_usuario_recibe integer NOT NULL,
id_notificacion integer,
contenido VARCHAR(250) NOT NULL,
fecha date DEFAULT CURRENT_DATE NOT NULL,
CONSTRAINT fk_usuarios7
FOREIGN KEY(id_usuario_envia)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_usuarios8
FOREIGN KEY(id_usuario_recibe)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_noti
FOREIGN KEY(id_notificacion)
REFERENCES notificaciones(id_notificacion)
);
```

NOTA: el *id_notificación* no tiene NOT NULL, para generarlo previamente antes de crear la publicación y evitar futuros errores.

Tabla **sugerencias**:

```
CREATE TABLE sugerencias (
id_sugerencia serial PRIMARY KEY,
id_usuario integer NOT NULL,
tipo char(10) NOT NULL,
contenido VARCHAR(250) NOT NULL,
CONSTRAINT fk_usuarios9
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario)
);
```

Tabla **publicaciones**:

```
CREATE TABLE publicaciones (
id_publicacion serial PRIMARY KEY,
id_usuario integer NOT NULL,
id_notificacion integer,
contenido VARCHAR(250) NOT NULL,
fecha date DEFAULT CURRENT_DATE NOT NULL,
CONSTRAINT fk_usuarios10
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_noti2
FOREIGN KEY(id_notificacion)
REFERENCES notificaciones(id_notificacion)
);
```

NOTA: el *id_notificacion* no tiene NOT NULL, para generarlo previamente antes de crear la publicación y evitar futuros errores.

Tabla **comentarios**:

```
CREATE TABLE comentarios (
id_comentario serial PRIMARY KEY,
id_usuario integer NOT NULL,
id_publicacion integer NOT NULL,
id_notificacion integer,
contenido VARCHAR(250) NOT NULL,
fecha date DEFAULT CURRENT_DATE NOT NULL,
CONSTRAINT fk_usuarios11
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_publicacion
FOREIGN KEY(id_publicacion)
REFERENCES publicaciones(id_publicacion),
CONSTRAINT fk_noti3
FOREIGN KEY(id_notificacion)
REFERENCES notificaciones(id_notificacion)
);
```

NOTA: el *id_notificacion* no tiene NOT NULL, para generarlo previamente antes de crear la publicación y evitar futuros errores.

Tabla **me_gustas**:

```
CREATE TABLE me_gustas (
id_usuario integer NOT NULL,
id_publicacion integer NOT NULL,
CONSTRAINT fk_usuarios12
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario),
CONSTRAINT fk_publicacion2
FOREIGN KEY(id_publicacion)
REFERENCES publicaciones(id_publicacion),
UNIQUE (id_usuario, id_publicacion)
);
```

Como se puede comprobar, las foreign keys no fueron creadas con **DELETE CASCADE**, esto sería un error muy grave al intentar borrar por ejemplo un usuario con el cual el campo id_usuario es utilizado en distintas tablas. Para solucionar este error se crearon **CONSTRAINT** que añadían el **DELETE CASCADE** en las **FOREIGN KEYS**, igualmente la mejor solución es aplicar esta restricción en el momento que se crea la tabla.

```
–Como se solucionó–
alter table comentarios
add constraint fk_notificacion3
foreign key (id_notificacion)
references notificaciones(id_notificacion)
on delete cascade;
```

```
–Como se debía haber hecho–
CREATE TABLE comentarios (
id_comentario serial PRIMARY KEY,
id_usuario integer NOT NULL,
id_publicacion integer NOT NULL,
id_notificacion integer NOT NULL,
contenido VARCHAR(250) NOT NULL,
fecha date DEFAULT CURRENT_DATE NOT NULL,
```

```
CONSTRAINT fk_usuarios11
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario) ON DELETE CASCADE,
CONSTRAINT fk_publicacion
FOREIGN KEY(id_publicacion)
REFERENCES publicaciones(id_publicacion) ON DELETE CASCADE,
CONSTRAINT fk_noti3
FOREIGN KEY(id_notificacion)
REFERENCES notificaciones(id_notificacion) ON DELETE CASCADE
);
```

Restricciones

Las restricciones que se muestran a continuación tienen una captura de su comprobación en el github.

Restricción para respetar el formato de un correo electrónico en la tabla usuarios.

```
ALTER TABLE usuarios
ADD CONSTRAINT ck_email_valido
CHECK (correo ~* '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$');
```

Restricción para que la contraseña sea de tamaño 4 mínimo (PIN):

```
ALTER TABLE usuarios
ADD CONSTRAINT password_al_menos_4
CHECK (
char_length(contraseña) >= 4
);
```

Restricción para que los tipos de sugerencias sean de los tres tipos posibles (mejora, error, nueva funcion):

```
ALTER TABLE sugerencias
ADD CONSTRAINT ck_tipo_sugerencia
CHECK (tipo IN ('mejora', 'error', 'nueva funcion'));
```

Restricción para que los tipos de notificaciones sean de sus posibles tipos(mensaje, comentario o publicación): Igualmente, la tabla notificaciones se autogenera sola y no se debería de meter valores de manera manual.

```
ALTER TABLE notificaciones
ADD CONSTRAINT restriccion_tipo_notificacion
CHECK (tipo IN (‘mensaje’, 'comentario’, 'publicacion’));
```

Restricción para que los tipos de solicitudes de amistad sean de los tipos enviado, aceptado o rechazado):

```
ALTER TABLE amigos
ADD CONSTRAINT restriccion_tipo_solicitud
CHECK (es_solicitud IN (‘enviado’, 'aceptado’, 'rechazado’));
```

Disparadores

Disparador (y función) que no permite mandar mensajes privados a un usuario si éste ha bloqueado al emisor.

```
CREATE OR REPLACE FUNCTION restriccion_bloqueados()
RETURNS trigger AS
$$
DECLARE
  bloqueado BOOLEAN;
BEGIN
SELECT EXISTS(
  SELECT 1
  FROM bloqueados
  WHERE id_usuario_bloqueador = NEW.id_usuario_recibe AND
    id_usuario_bloqueado = NEW.id_usuario_envia
) INTO bloqueado;
  IF bloqueado THEN
    RAISE EXCEPTION 'No se puede enviar el mensaje: el usuario que lo recibe ha
bloqueado al usuario que lo envía';
  END IF;
  RETURN NEW;
END;
```

```
$$
LANGUAGE plpgsql;

CREATE TRIGGER restriccion_bloqueados
BEFORE INSERT ON mensajes_privados
FOR EACH ROW
EXECUTE PROCEDURE restriccion_bloqueados();
```

Una persona bloqueada no puede dar me gusta a una publicación de un usuario que le ha bloqueado.

```
CREATE OR REPLACE FUNCTION comprobar_bloqueado() RETURNS TRIGGER AS
$$
BEGIN
   IF (
     SELECT COUNT(*)
     FROM bloqueados
     WHERE id_usuario_bloqueado = NEW.id_usuario AND
       id_usuario_bloqueador = (
        SELECT id_usuario
        FROM publicaciones
        WHERE id_publicacion = NEW.id_publicacion
   )) > 0 THEN
   RAISE EXCEPTION 'No se puede dar "me gusta" a una publicación de un usuario que
te ha bloqueado';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER bloquear_me_gustas
BEFORE INSERT ON me_gustas
FOR EACH ROW
EXECUTE PROCEDURE comprobar_bloqueado();
```

Una persona bloqueada no puede comentar una publicación de un usuario que le ha bloqueado.

```
CREATE OR REPLACE FUNCTION bloquear_comentarios_bloqueados()
RETURNS TRIGGER AS $$
BEGIN
  -- Comprueba si el usuario que está realizando el comentario está bloqueado por el usuario que ha publicado la publicación
  IF EXISTS (
    SELECT *
    FROM bloqueados
    WHERE id_usuario_bloqueado = NEW.id_usuario AND
      id_usuario_bloqueador = (
        SELECT id_usuario
        FROM publicaciones
        WHERE id_publicacion = NEW.id_publicacion
  )) THEN
    -- Lanza una excepción si el usuario está bloqueado
      RAISE EXCEPTION 'No puedes comentar una publicación de un usuario que te ha bloqueado';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER bloquear_comentarios_bloqueados
BEFORE INSERT ON comentarios
FOR EACH ROW EXECUTE PROCEDURE bloquear_comentarios_bloqueados();
```

Una persona bloqueada no puede enviar una solicitud de amistad a un usuario que le ha bloqueado y viceversa.

```
CREATE OR REPLACE FUNCTION bloquear_solicitudes_bloqueados()
RETURNS TRIGGER AS $$
BEGIN
  -- Comprueba si el usuario que está enviando la solicitud está bloqueado por el usuario al que se le está enviando la solicitud
  IF EXISTS (
  SELECT *
  FROM bloqueados
  WHERE id_usuario_bloqueado = NEW.id_usuario_envia AND
    id_usuario_bloqueador = NEW.id_usuario_recibe
```

```
) THEN
    -- Lanza una excepción si el usuario está bloqueado
     RAISE EXCEPTION 'No puedes enviar una solicitud de amistad a un usuario que te ha bloqueado';
  END IF;

  IF EXISTS (
   SELECT *
   FROM bloqueados
  WHERE id_usuario_bloqueador = NEW.id_usuario_envia AND
    id_usuario_bloqueado = NEW.id_usuario_recibe
  ) THEN
    -- Lanza una excepción si el usuario está bloqueado
     RAISE EXCEPTION 'No puedes enviar una solicitud de amistad a un usuario que has bloqueado';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER bloquear_solicitudes_bloqueados
BEFORE INSERT ON amigos
FOR EACH ROW EXECUTE PROCEDURE bloquear_solicitudes_bloqueados();
```

Automatizar las notificaciones:

En el caso de que se añada una publicación, la tabla notificaciones se actualiza con el id del usuario que hace la publicación, el campo tipo = “publicación” y enviar_recibir = “envía”. A la vez dentro de la inserción de la publicación se autocompleta el campo id_notificaciones con el id de la notificación que se ha autogenerado.

```
CREATE OR REPLACE FUNCTION sp_insert_notificaciones()
RETURNS TRIGGER AS
$BODY$
BEGIN
INSERT INTO notificaciones (id_usuario, tipo, recibe_envia)
VALUES (NEW.id_usuario, 'publicacion', 'envia');
UPDATE publicaciones
SET id_notificacion = (
  SELECT id_notificacion
  FROM notificaciones
```

```
  ORDER BY id_notificacion
  DESC LIMIT 1
)
WHERE id_publicacion = NEW.id_publicacion;
RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER tr_insert_notificaciones
AFTER INSERT ON publicaciones
FOR EACH ROW
EXECUTE PROCEDURE sp_insert_notificaciones();
```

En el caso de que se añada un comentario, la tabla notificaciones se actualiza con el id del usuario que hace el comentario, el campo tipo = “comentario” y enviar_recibir = “envía”. A la vez dentro de la inserción de el comentario se autocompleta el campo id_notificaciones con el id de la notificación que se ha autogenerado.

```
CREATE OR REPLACE FUNCTION sp_insert_notificacionesCom()
RETURNS TRIGGER AS
$BODY$
BEGIN
INSERT INTO notificaciones (id_usuario, tipo, recibe_envia)
VALUES (NEW.id_usuario, 'comentario', 'envia');
UPDATE comentarios
SET id_notificacion = (
  SELECT id_notificacion
  FROM notificaciones
  ORDER BY id_notificacion
  DESC LIMIT 1
)
WHERE id_comentario = NEW.id_comentario;
RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER tr_insert_notificacionesCom
AFTER INSERT ON comentarios
FOR EACH ROW
EXECUTE PROCEDURE sp_insert_notificacionesCom();
```

En el caso de que se añada un mensaje privado, la tabla notificaciones se actualiza con el id del usuario que hace el mensaje_privado, el campo tipo = “mensaje” y enviar_recibir = “envía”. A la vez dentro de la inserción de el mensaje_privado, se autocompleta el campo id_notificaciones con el id de la notificación que se ha autogenerado.

```
CREATE OR REPLACE FUNCTION sp_insert_notificacionesMP()
RETURNS TRIGGER AS
$BODY$
BEGIN
INSERT INTO notificaciones (id_usuario, tipo, recibe_envia)
VALUES (NEW.id_usuario_envia, 'mensaje', 'envia');
UPDATE mensajes_privados
SET id_notificacion = (
  SELECT id_notificacion
  FROM notificaciones
  ORDER BY id_notificacion
  DESC LIMIT 1
)
WHERE id_mensaje = NEW.id_mensaje;
RETURN NULL;
END;
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER tr_insert_notificacionesMP
AFTER INSERT ON mensajes_privados
FOR EACH ROW
EXECUTE PROCEDURE sp_insert_notificacionesMP();
```

Otro disparador creado, cumple con el objetivo de autorellenar la tabla estadísticas con un contador de me gustas, comentarios y publicaciones de cada usuario. Cada vez que se crea un usuario se establece una fila en la tabla estadísticas con el id_usario y los valores de los campos a 0 y cada vez que se realiza una acción de las anteriores se actualiza dicho campo en la tabla.

```
CREATE OR REPLACE FUNCTION actualizar_estadisticas()
RETURNS TRIGGER AS $$
BEGIN
```

```
IF (TG_OP = 'INSERT' AND TG_TABLE_NAME = 'publicaciones') THEN
    UPDATE estadisticas SET num_publicaciones = num_publicaciones + 1 WHERE id_usuario = NEW.id_usuario;
 ELSIF  (TG_OP = 'INSERT' AND TG_TABLE_NAME = 'me_gustas') THEN
   UPDATE estadisticas SET num_megustas = num_megustas + 1 WHERE id_usuario = NEW.id_usuario;
 ELSIF (TG_OP = 'INSERT' AND TG_TABLE_NAME = 'comentarios') THEN
    UPDATE estadisticas SET num_comentarios = num_comentarios + 1 WHERE id_usuario = NEW.id_usuario;
 ELSEIF (TG_OP = 'INSERT' AND TG_TABLE_NAME = 'usuarios') THEN
       INSERT INTO estadisticas (id_usuario, num_comentarios, num_megustas, num_publicaciones) VALUES (NEW.id_usuario, 0, 0, 0);
 END IF;
 RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER actualizar_estadisticas
AFTER INSERT ON publicaciones
FOR EACH ROW
EXECUTE PROCEDURE actualizar_estadisticas();

CREATE TRIGGER actualizar_estadisticas
AFTER INSERT ON me_gustas
FOR EACH ROW
EXECUTE PROCEDURE actualizar_estadisticas();

CREATE TRIGGER actualizar_estadisticas
AFTER INSERT ON comentarios
FOR EACH ROW
EXECUTE PROCEDURE actualizar_estadisticas();

CREATE TRIGGER actualizar_estadisticas
AFTER INSERT ON usuarios
FOR EACH ROW
EXECUTE PROCEDURE actualizar_estadisticas();
```

No permitir que un usuario de más de un me gusta a la misma publicación:

```
CREATE OR REPLACE FUNCTION evitar_duplicados_me_gustas()
RETURNS TRIGGER AS $$
```

```
BEGIN
      -- Comprueba si ya existe una fila con los mismos valores para id_usuario y id_publicacion
    IF EXISTS (SELECT * FROM me_gustas WHERE id_usuario = NEW.id_usuario AND id_publicacion = NEW.id_publicacion) THEN
      -- Si existe, se cancela la operación
      RAISE EXCEPTION 'No se puede dar mas de un me gusta a la misma publicacion';
      RETURN NULL;
   ELSE
      -- Si no existe, se permite la operación y se devuelve el nuevo valor
      RETURN NEW;
   END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER evitar_duplicados_me_gustas
BEFORE INSERT ON me_gustas
FOR EACH ROW EXECUTE PROCEDURE evitar_duplicados_me_gustas();
```

Condición para poder realizar la **restricción de inclusividad**, la cual no permitía crear una sugerencia a menos que ya el usuario hubiese creado una publicación primero.

```
-- Una persona no puede hacer una sugerencia si no tiene primero una publicación hecha.
CREATE OR REPLACE FUNCTION comprobar_publi_suge() RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT COUNT(*) FROM publicaciones WHERE id_usuario = NEW.id_usuario ) <= 0 THEN
      RAISE EXCEPTION 'No se puede hacer una sugerencia sin antes haber creado una publicación ';
   END IF;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER bloquear_sugerencia
BEFORE INSERT ON sugerencias
FOR EACH ROW
EXECUTE PROCEDURE comprobar_publi_suge();
```

Carga de datos de ejemplo

Scripts implementación de la base de datos en PostgreSQL

Inserción de 3 usuarios de prueba.

```
insert into
   usuarios(nombre, correo, contraseña, fecha_nacimiento, telefono)
   values ('Hector','hector@gmail.com','1234','1998-12-17', '656-778-899');

insert into
   usuarios(nombre, correo, contraseña, fecha_nacimiento, telefono)
   values ('Ramon', 'ramon@gmail.com', '4321', '1998-12-17', '656-411-899');

insert into
   usuarios(nombre, correo, contraseña, fecha_nacimiento, telefono)
   values ('Marcos', 'marcos@gmail.com', '3333', '1998-12-17', '656-448-899');
```

Bloqueo de 2 usuarios:

```
insert into
   bloqueados(id_usuario_bloqueado, id_usuario_bloqueador)
   values (4, 3);
insert into
   bloqueados(id_usuario_bloqueado, id_usuario_bloqueador)
   values (3, 5);
```

Generar notificaciones de manera manual:

```
insert into
   notificaciones(id_usuario, tipo, recibe_envia)
   values(4, 'mensaje','envia');

insert into
   notificaciones(id_usuario, tipo, recibe_envia)
   values(3, 'mensaje','envia');
```

Envío de mensajes privados:

```
insert into
    mensajes_privados(id_usuario_envia, id_usuario_recibe, id_notificacion, contenido)
    values (4, 5, 2, 'esto es un mensaje privado');
```

El siguiente mensaje falla ya que salta la función **restriccion_bloqueados()**:

```
insert into
    mensajes_privados(id_usuario_envia, id_usuario_recibe, id_notificacion, contenido)
    values (4, 3, 2, 'esto es un mensaje privado');
```

Insertando comentarios:

```
insert into comentarios (id_usuario, id_publicacion, contenido)
values (3,5,'Mola');
```

Diseño de consultas

Creación de vista de todos los usuarios:

```
CREATE VIEW vista_usuarios AS
SELECT id_usuario, nombre, correo, fecha_nacimiento, telefono
FROM usuarios;
```

Creación de vista de las publicaciones del usuario con id 9:

```
CREATE VIEW publicaciones_usuario_9 AS
SELECT * FROM publicaciones WHERE id_usuario = 9;
```

Creación de vista de todos las publicaciones:

```
CREATE VIEW vista_publicaciones AS
```

```
SELECT *
FROM publicaciones;
```

Implementación de un API REST mediante Flask

Se realiza la instalación del framework Flask y la biblioteca psycopg2-binary.
Pasos :
Dentro del directorio PROYECTOFINAL se
comprueba si el módulo “venv” está instalado:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv |-> Comprobando si esta el modulo venv instalado
$ sudo apt install python3.8-venv | -> Instalando el módulo ‘venv’
$ python3 -m venv venv |-> Ejecución del módulo ‘venv’.
$ . venv/bin/activate |-> Activar y entrar en el entorno. ‘(venv)’ aparecerá en el prompt $
pip3 install Flask |-> Instalando Flask en el entorno (y el directorio)
$ pip install psycopg2-binary |-> Adaptador para poder conectarse a PostgreSQL usando
scripts de Python.
```

En cuanto al contenido importante de la carpeta venv, se encuentra el fichero **app.py** y las plantillas html creadas en la carpeta **templates**.

En cuanto al fichero **app.py**, contiene la conexión con la base de datos y los métodos creados para las consultas de las distintas tablas.

Conexión base de datos

```
# Instancia de la clase Flask que vigila en el directorio template las plantillas a renderizar
app = Flask(__name__)

# Conexión
def get_db_connection():
    conn = psycopg2.connect(host='localhost',
        database="redsocial",
        # user=os.environ['DB_USERNAME'],
                    user="postgres",
                    # password=os.environ['DB_PASSWORD']
        password="postgres")
    return conn
```

A continuación se muestra de ejemplo, los métodos que muestran el contenido de la tabla usuario y permiten crear editar o eliminar un usuario:

```
@app.route('/usuarios/')
def usuarios():
    conn = get_db_connection()
    cur = conn.cursor()
    cur.execute('SELECT * FROM usuarios')
    usuarios = cur.fetchall()
    cur.close()
    conn.close()
    return render_template('usuarios.html', usuarios=usuarios)


@app.route('/nuevo_usuario/', methods=('GET', 'POST'))
def nuevo_usuario():
    if request.method == 'POST':
        try:
            nombre = request.form['nombre']
            correo = request.form['correo']
            passwd = request.form['contrasena']
            fechan = request.form['fecha_nac']
            telefono = request.form['telefono']

            conn = get_db_connection()
            cur = conn.cursor()
                        cur.execute('INSERT INTO usuarios (nombre, correo, contraseña,
fecha_nacimiento, telefono)'
                    'VALUES (%s, %s, %s, %s, %s)',
                    (nombre, correo, passwd, fechan, telefono))
            conn.commit()
        except psycopg2.Error as err:
            print("Ocurrió un error al añadir valores:\n", err)
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('usuarios'))

    return render_template('nuevo_usuario.html')


@app.route('/editar_usuario/', methods=('GET', 'POST'))
def editar_usuario():
    if request.method == 'POST':
```

```
        try:
            idusuario = request.form['identificador']
            nombre = request.form['nombre']
            correo = request.form['correo']
            passwd = request.form['contrasena']
            fechan = request.form['fecha_nac']
            telefono = request.form['telefono']

            conn = get_db_connection()
            cur = conn.cursor()

             cur.execute('UPDATE usuarios SET nombre = %s, correo = %s, contraseña = %s,
fecha_nacimiento = %s, telefono = %s WHERE id_usuario = %s',
                        (nombre, correo, passwd, fechan, telefono, idusuario))

            conn.commit()
        except psycopg2.Error as err:
            print("Ocurrió un error al editar valores:\n", err)
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('index'))

    return render_template('editar_usuario.html')


@app.route('/eliminar_usuario/', methods=('GET', 'POST'))
def eliminar_usuario():
    if request.method == 'POST':
        try:
            id_usuario = request.form['id_usuario']
            conn = get_db_connection()
            cur = conn.cursor()
            cur.execute(
                'DELETE FROM usuarios WHERE id_usuario = %s;',
                (id_usuario,)
            )
            conn.commit()
        except psycopg2.Error as err:
            print("Ocurrió un error al eliminar el usuario:\n", err)
        finally:
            cur.close()
            conn.close()
        return redirect(url_for('usuarios'))
```

```
    return render_template('eliminar_usuario.html')
```

Para llevar a cabo estos métodos se han creado las siguientes plantillas:

usuarios.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEd
K2Kadq2F9CUG65" crossorigin="anonymous">
</head>

   <body>

      <div class="col align-self-center">

         {% extends 'base.html' %}
         {% block content %}
         <div class="container">
            <div class="row">
               <div class="col align-self-center">
                  <div class="alert alert-info" role="alert">
                     <div class="col">
                        <h3>{% block title %} Usuarios {% endblock %}</h3>
                     <div class="col">
                                 <a href="{{ url_for('nuevo_usuario') }}" role="button" class="btn
btn-primary" >Crear</a>
                                  <a href="{{ url_for('editar_usuario') }}" role="button" class="btn
btn-warning">Editar</a>
                               <a href="{{ url_for('eliminar_usuario') }}" role="button" class="btn
btn-danger">Eliminar  </a>
                                 <a href="{{ url_for('buscar_usuario') }}" role="button" class="btn
btn-success"> Buscar por id_usuario </a>
                     </div>
                     </div>
                  </div>

                  {% for usuario in usuarios %}
                     <div class='usuario'>
```

```
                    <div class="alert alert-success" role="alert">
                        <h4>ID_Usuario {{ usuario[0] }} </h4>
                    </div>

                    <i><p> <b>Nombre:</b> {{ usuario[1] }} </p></i>
                    <i><p> <b>Fecha_nacimiento:</b> {{ usuario[4] }} </p></i>
                    <i><p> <b>Correo:</b> {{ usuario[2] }} </p></i>
                    <i><p> <b>Contraseña:</b> {{ usuario[3] }} </p></i>
                    <i><p> <b>Telefono:</b> {{ usuario[5] }} </p></i>
                    <hr>

                </div>
            {% endfor %}

        {% endblock %}

        </div>
    </div>
</div>
</div>


                                                                    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+jjXkk+Q2h455rYXK/7HAu
oJl+0I4" crossorigin="anonymous"></script>

</body>


</html>
```

nuevo_usuario.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEd
K2Kadq2F9CUG65" crossorigin="anonymous">
</head>
```

```
<body>
    <div class="col align-self-center">
      {% extends 'base.html' %}
      {% block content %}
      <div class="container">
        <div class="row">
          <div class="col align-self-center">
            <div class="alert alert-primary" role="alert">
              <div class="col">
                <h1>{% block title %} Nuevo usuario {% endblock %}</h1>
              </div>
            </div>
                <form method="post">
                  <div class="mb-3">
                    <label for="nombre">Nombre</label>
                    <input type="text"class="form-control" name="nombre"
                      placeholder="nombre de usuario">
                    </input>
                  </div>

                  <div class="mb-3">
                    <label for="correo">Correo</label>
                    <input type="text"class="form-control" name="correo"
                      placeholder="example@gmail.com">
                    </input>
                  </div>

                  <div class="mb-3">
                    <label for="contrasena">Contraseña</label>
                                    <input  type="password"  class="form-control"
name="contrasena"
                      placeholder="**************">
                    </input>
                  </div>



                  <p>
                    <label for="fecha_nac">Fecha nacimiento</label>
                    <input type="date" name="fecha_nac"
                      placeholder="Fecha de nacimiento">
                    </input>
                  </p>
```

```
                        <div class="mb-3">
                            <label for="telefono">Telefono</label>
                            <input type="text" class="form-control" name="telefono"
                                placeholder="690009870">
                            </input>
                        </div>

                        <button type="submit" class="btn btn-success">Guardar</button>

                    </form>
                {% endblock %}
            </div>
        </div>
    </div>
</body>

</html>
```

editar_usuario.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEd
K2Kadq2F9CUG65" crossorigin="anonymous">
</head>

<body>
    <div class="col align-self-center">
        {% extends 'base.html' %}
        {% block content %}
        <div class="container">
            <div class="row">
                <div class="col align-self-center">
                    <div class="alert alert-warning" role="alert">
                        <div class="col">
                            <h1>{% block title %} Editar Usuario por ID {% endblock %}</h1>
                        </div>
```

```
                </div>
                    <form method="post">
                        <div class="mb-3">
                            <label for="identificador">ID Usuario</label>
                                    <input type="text"class="form-control" name="identificador"
required
                                placeholder="id de usuario a modificar">
                            </input>
                        </div>
                        <div class="mb-3">
                            <label for="nombre">Nombre</label>
                            <input type="text"class="form-control" name="nombre" required
                                placeholder="nombre de usuario">
                            </input>
                        </div>

                        <div class="mb-3">
                            <label for="correo">Correo</label>
                            <input type="text"class="form-control" name="correo" required
                                placeholder="example@gmail.com">
                            </input>
                        </div>

                        <div class="mb-3">
                            <label for="contrasena">Contraseña</label>
                                                <input  type="password"  class="form-control"
name="contrasena" required
                                placeholder="**************">
                            </input>
                        </div>
                        <p>
                            <label for="fecha_nac">Fecha nacimiento</label>
                            <input type="date" name="fecha_nac" required
                                placeholder="Fecha de nacimiento">
                            </input>
                        </p>

                        <div class="mb-3">
                            <label for="telefono">Telefono</label>
                            <input type="text" class="form-control" name="telefono" required
                                placeholder="690009870">
                            </input>
                        </div>

                        <button type="submit" class="btn btn-success">Guardar</button>
```

```
                    </form>
                {% endblock %}
            </div>
        </div>
    </div>
</body>
</html>
```

eliminar_usuario.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
        <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEd
K2Kadq2F9CUG65" crossorigin="anonymous">
</head>

<body>
    <div class="col align-self-center">
        {% extends 'base.html' %}
        {% block content %}
        <div class="container">
            <div class="row">
                <div class="col align-self-center">
                    <div class="alert alert-danger" role="alert">
                        <div class="col">
                            <h1>{% block title %} Eliminar Usuario {% endblock %}</h1>
                        </div>
                    </div>
                        <form method="post">
                            <div class="mb-3">
                                    <label for="id_usuario">Introduce el ID del Usuario a
Eliminar</label>
                                <input type="text"class="form-control" name="id_usuario"
                                    placeholder="ID Usuario">
                                </input>
                            </div>
                            <button type="submit" class="btn btn-success">Eliminar</button>
                        </form>
```

```
                {% endblock %}
            </div>
          </div>
        </div>
      </div>
 </body>
</html>
```

Para las distintas tablas se ha llevado a cabo la misma metodología. Sin embargo, hay tablas que no tienen implementada métodos para editar o crear ya que esos pasos son automáticos por la base de datos, como en el caso de notificaciones que se autogeneran y solo se ha permitido la opción de eliminar.

El comando para lanzar la aplicación mediante Flask mediante la configuración del puerto de acceso y su dirección.

```
~/ProyectoFinal/venv$ flask --app app run --host=0.0.0.0 --port=8080
```

Complicaciones y Conclusiones

Como se explicó antes, cuando se intentó eliminar un usuario, otras tablas dependían del identificador del usuario (*id_usuario*) y además ya se tenía datos del usuario a eliminar en tales tablas. Se intentó escribir múltiples operaciones de *DELETE* de todas las tablas dependientes de la tabla “Usuarios” en la función de *Flask* que hace la eliminación de Usuarios, pero sólo dejaba hacer como mucho dos operaciones PostgreSQL a la vez.

La solución fue añadir a todas las referencias de clave externa de *id_usuario* en cada tabla que apareciera las palabras clave *ON DELETE CASCADE*. Así, por ejemplo en la tabla *me_gusta* se añadió lo anterior -> REVISAR

```
CONSTRAINT fk_usuarios12
FOREIGN KEY(id_usuario)
REFERENCES usuarios(id_usuario) ON DELETE CASCADE,
```

En cuanto a las notificaciones, se pretendía elaborar disparadores para autocompletar la tabla notificaciones. Si bien los disparadores del tipo enviar se realizaron, los disparadores del tipo recibir en el que se pretendía que se autogeneraran notificaciones del tipo recibir cada vez que la tabla notificaciones recibía una nueva notificación del tipo

enviar, diferenciando entre los tres tipos de notificaciones (publicaciones, comentarios o mensaje privado), no se lograron realizar.

También se encontró dificultad a la hora de crear disparadores para evitar la duplicidad de solicitudes de amistad a la misma vez que se borrasen las solicitudes enviadas una vez aceptada o rechazada junto con sólo permitir aceptar o rechazar solicitudes si previamente existía una solicitud enviada por el id_usuario_recibe.

Igualmente, los disparadores creados demuestran la comprensión de esta técnica por parte del grupo.