UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

LABORATORIO DISEÑO DE BASE DE DATOS: ENTREGA 2

Grupo 3

Integrantes: Fabián Escobar
Gonzalo Estay
Richard Torti
Julio Vásquez
Gerardo Zuñiga

Profesores: Carolina Bonacic
Pablo Gonzalez

Ayudantes: Nestor Mora
Daniel Vargas

04 de Noviembre del 2016

1. CONSULTAS

A continuación se presentan todas las consultas que se realizan a la base de datos y que se consideran importantes para el enfoque del proyecto.

1.1 Listar historial de misiones de cada voluntario

```
SELECT USERS.ID_USER AS ID_USER,
       USERS.NOMBRE_USUARIO AS NOMBRE_USUARIO,
       MISSIONS.NOMBRE_MISION AS NOMBRE_MISION
FROM REALIZA
     JOIN USERS ON REALIZA.ID_USER = USERS.ID_USER
     JOIN TASKS ON REALIZA.ID_TASK = TASKS.ID_TASK
     JOIN MISSIONS ON TASKS.ID_MISSION = MISSIONS.ID_MISSION
ORDER BY USERS.ID_USER, TASKS.ID_TASK;
```

1.2 Listar documentación de cada tarea

```
SELECT GENERA.ID_TASK AS ID_TASKS,
       DOCUMENTATIONS.ID_DOC AS ID_DOC,
       DOCUMENTATIONS.NOMBRE_DOC AS NOMBRE_DOC,
       DOCUMENTATIONS.URL_DOC AS URL_DOC
FROM DOCUMENTATIONS
     JOIN GENERA ON DOCUMENTATIONS.ID_DOC = GENERA.ID_DOC
ORDER BY GENERA.ID_TASK, DOCUMENTATIONS.NOMBRE_DOC;
```

1.3 Historial de emergencias de una región

```
SELECT REGIONS.ID_REGION AS ID_REGION,
       REGIONS.NOMBRE_REGION AS NOMBRE_REGION,
       COMMUNES.ID_COMMUNE AS ID_COMMUNE,
       COMMUNES.NOMBRE_COMUNA AS NOMBRE_COMUNA,
       EMERGENCIES.NOMBRE AS NOMBRE_EMERGENCIA,
       EMERGENCIES.ESTADO_EMERGENCIA AS ESTADO_EMERGENCIA,
       EMERGENCIES.FECHA_EMERGENCIA AS FECHA_EMERGENCIA
FROM REGIONS
     JOIN COMMUNES ON REGIONS.ID_REGION = COMMUNES.ID_REGION
     JOIN EMERGENCIES ON COMMUNES.ID_COMMUNE = EMERGENCIES.ID_COMMUNE
ORDER BY  REGIONS.ID_REGION,
          COMMUNES.ID_COMMUNE,
          EMERGENCIES.FECHA_EMERGENCIA DESC;
```

1.4 Listado de administradores

```
SELECT USERS.ID_USER AS ID_USER,
       USERS.NOMBRE_USUARIO AS NOMBRE_USUARIO
FROM USERS
WHERE USERS.ADMIN = 1;
```

1.5 Listado de usuario según calificación promedio

```
SELECT USERS.ID_USER AS ID_USER,
       USERS.NOMBRE_USUARIO AS NOMBRE_USUARIO,
       PROM_TABLE.PROMEDIO AS PROMEDIO,
       PROM_TABLE.NRO_TASKS AS NRO_TASKS
FROM USERS
     JOIN (
          SELECT REALIZA.ID_USER AS ID_USER,
                 AVG(REALIZA.VALORACION) AS PROMEDIO,
                 COUNT(*) AS NRO_TASKS
          FROM REALIZA
          GROUP BY REALIZA.ID_USER
          ) PROM_TABLE ON USERS.ID_USER = PROM_TABLE.ID_USER
ORDER BY PROM_TABLE.PROMEDIO DESC;
```

1.6 Listar voluntarios que cumplen con al menos un requisito de tarea para una tarea específica de id = ID

```
SELECT USERS.ID_USER AS ID_USER,
       USERS.NOMBRE_USUARIO AS NOMBRE_USUARIO,
       COUNT(*) AS HABILIDADES
FROM REQUIERE
     JOIN ABILITIES ON REQUIERE.ID_TASK = ID USING(ID_ABILITY)
     JOIN TIENE USING(ID_ABILIIY)
     JOIN USERS USING(ID_USER)
GROUP BY USERS.ID_USER
```

1.8 Listar misiones administradas por un usuario

```
SELECT USERS.ID_USER AS ID_USER,
       USERS.NOMBRE_USUARIO AS NOMBRE_USUARIO,
       MISSIONS.NOMBRE_MISION AS NOMBRE_MISION
FROM REALIZA
     JOIN USERS ON REALIZA.ID_USER = USERS.ID_USER
     JOIN TASKS ON REALIZA.ID_TASK = TASKS.ID_TASK
     JOIN MISSIONS ON TASKS.ID_MISSION = MISSIONS.ID_MISSION
ORDER BY USERS.ID_USER, TASKS.ID_TASK;
```

1.9 Listar encargados de emergencia por emergencia

```
SELECT MISSIONS.ID_EMERGENCY AS id_emergency,
       EMERGENCIES.NOMBRE AS nombre,
       MISSIONS.ID_USER AS id_user,
       USERS.NOMBRE_USUARIO AS nombre_usuario,
       MISSIONS.ID_MISSION AS id_mission,
       MISSIONS.NOMBRE_MISION AS nombre_mision
FROM USERS
     JOIN MISSIONS ON USERS.ID_USER = MISSIONS.ID_USER
     JOIN EMERGENCIES ON MISSIONS.ID_EMERGENCY = EMERGENCIES.ID_EMERGENCY
ORDER BY MISSIONS.ID_EMERGENCY , MISSIONS.ID_USER , MISSIONS.ID_MISSION;
```

1.10 Listar voluntarios disponibles, junto a sus habilidades y el nivel de cada una.

Por medio de una consulta se crea una vista que permite listar todos los voluntarios que están actualmente disponibles y sus habilidades con el nivel que cada una tiene, de manera que al momento de necesitar voluntarios para una tarea de una misión, estos se pueden filtrar desde esta tabla.

```
SELECT USERS.ID_USER, USERS.NOMBRE_USUARIO, TIENE.ID_ABILITY,
TIENE.NIVEL_ABILITY
FROM USERS, TIENE
WHERE USERS.DISPONIBILIDAD = 1 AND TIENE.ID_USER = USERS.ID_USER;
```

2. PROCEDIMIENTOS ALMACENADOS

En la presente sección del documento se listan y explican los diferentes procedimientos almacenados implementados.

2.1 Crear, modificar y eliminar un usuario.

- Mediante el procedimiento crearUsuario, se insertan los datos de entrada (id_comuna, nombre_usr, pass, disp, administrador, mail y telefono) en las tablas USERS, EMAILS y PHONE_NUMBERS (en estas dos últimas tablas van los 2 últimos parámetros respectivamente). La id del usuario se genera por la misma tabla USERS, teniendo esta última autoincremento activado para las IDs. Previamente a la inserción se verifica si el e-mail a insertar ya existe, de manera que a cada usuario le corresponda un sólo e-mail.
- El procedimiento modificarUsuario permite realizar un UPDATE a los datos de un usuario, ingresando los datos del usuario junto con la ID correspondiente de este. Los datos se actualizan en las tablas USERS, EMAILS y PHONE_NUMBERS. Es necesario ingresar todos los datos para realizar la inserción.
- Con el procedimiento eliminarUsuario es posible eliminar un usuario de la base de datos ingresando únicamente la ID del usuario. Los datos se eliminan de casi todas las tablas a través de CASCADE DELETE, ya que para algunas tablas se implementan triggers que actúan al momento de eliminar un usuario de la BD.

2.2 Crear, modificar y eliminar una región.

- Mediante el procedimiento crearRegion, es posible insertar en la tabla REGIONS la región perteneciente a un país, en la cual se solicita sólo tener el nombre de dicha región, bajo la restricción de que este no se encuentra ya ocupado.
- modificarRegion permite modificar el nombre de una región que ya se encuentra ingresada en la tabla REGIONS, bajo la restricción de que este no se encuentra ya ocupado.

- eliminarRegion, permite eliminar de la tabla la región dada la id de la región. Se debe tener en consideración, de que al eliminar una región, se están eliminando sus comunas asociadas también.

2.3 Crear, modificar y eliminar una comuna.

- crearComuna, inserta en la tabla COMMUNES una comuna, dada la id de la región a la cual pertenece y el nombre, donde id de la región debe existir en la tabla REGIONS y el nombre no se encuentre repetido en la tabla COMMUNES.
- modificarComuna, modifica el nombre de la comuna en la tabla COMMUNES, siempre y cuando no se encuentre el nombre repetido en la tabla.
- eliminarComuna, elimina de la tabla COMMUNES una comuna, solicitando su id de comuna.

2.4 Crear, modificar y eliminar una emergencia.

- crearEmergencia, crea una emergencia en la tabla EMERGENCIES, donde se debe entregar la id del usuario (que debe existir en la tabla USERS), la id de la comuna (que debe existir en la tabla COMMUNES), el nombre de la emergencia, gravedad, estado y descripción. Como protección (en caso de que el procedimiento se duplique o que el usuario ingrese más de una vez una emergencia), no es posible ingresar emergencias que tengan o compartan iguales valores en los campos id de comuna, nombre y fecha.
- modificarEmergencia, modifica en la tabla EMERGENCIES, los campos de nombre, gravedad, estado y descripción de una emergencia, dado un id de emergencia.
- eliminarEmergencia, dado un id de emergencia, elimina de la tabla EMERGENCIES una emergencia.

2.5 Asignar un voluntario disponible a una tarea.

Se asigna un voluntario disponible a una tarea, verificando que este posea una de las habilidades requeridas y que esta habilidad cumpla con el nivel mínimo que se exige.

3. TRIGGERS

En esta última porción del documento, se presentan los triggers añadidos a la base de datos y se explica la funcionalidad de cada uno de ellos.

3.1 Crear una notificación cuando se elimina un administrador (encargado de emergencia) o encargado de misión.

El primer trigger implementado en la base de datos se encarga de generar una notificación cuando se elimina un voluntario que actualmente trabaja en una emergencia y que es administrador o encargado de emergencia de ella. De esta manera, se notifica a los otros

voluntarios correspondientes que se necesita un administrador o un encargado de emergencia, ya que el voluntario que previamente cumplía con este cargo, ya no existe en el sistema.

El trigger en cuestión, en SQL queda de la siguiente manera:

```
CREATE TRIGGER `notificacion_delete_user` BEFORE DELETE ON `USERS`
 FOR EACH ROW
BEGIN
IF OLD.ADMIN=1 THEN
      SET @bool = (SELECT COUNT(*) FROM ENCARGADO_EMERGENCIA
      WHERE ENCARGADO_EMERGENCIA.id_user=OLD.id_user);
      IF @bool > 0 THEN
            INSERT INTO NOTIFICATIONS(URL, CONTENIDO, FECHA)
            SELECT '', 'Se necesita un administradorde emergencia', NOW();
            SET @id_notificacion_2 =
            ( SELECT ID_NOTIFICATION FROM NOTIFICATIONS
            ORDER BY ID_NOTIFICATION DESC LIMIT 1);
            INSERT INTO RECIBE_NOTIFICACION(ID_USER, ID_NOTIFICATION)SELECT
            ID_USER, @id_notificacion_2 FROM ADMINS;
      END IF;
ELSE
      SET @bool = (SELECT COUNT(*) FROM ENCARGADOS_MISSION
      WHERE ENCARGADOS_MISSION.id_user=OLD.id_user);
      IF @bool > 0 THEN
            INSERT INTO NOTIFICATIONS(URL, CONTENIDO, FECHA)
            SELECT '', 'Se necesita un encargado de misión', NOW();
            SET @id_notificacion_2 =
            ( SELECT ID_NOTIFICATION FROM NOTIFICATIONS
            ORDER BY ID_NOTIFICATION DESC LIMIT 1);
            INSERT INTO RECIBE_NOTIFICACION(ID_USER, ID_NOTIFICATION)SELECT
            DISTINCT(ID_USER), @id_notificacion_2 FROM ENCARGADOS_MISSION;
      END IF;
END IF;
END
```

3.2 Generar una notificación cuando se envía una solicitud para una tarea.

El segundo trigger implementado se encarga de generar una notificación cuando se envía una solicitud a un voluntario de manera que este sepa que se necesita de su ayuda en una emergencia.

El código en SQL corresponde al siguiente:

```
CREATE TRIGGER `notificacion_request` AFTER INSERT ON `REQUESTS`
 FOR EACH ROW
BEGIN
SET @id_user_2 = (SELECT NEW.id_user);
SET @cont_notif = (SELECT NEW.nombre_solicitud);
INSERT INTO NOTIFICATIONS(URL, CONTENIDO, FECHA)
SELECT '', @cont_notif, NOW();
SET @id_notificacion_2 = ( SELECT ID_NOTIFICATION FROM NOTIFICATIONS
```

```
ORDER BY ID_NOTIFICATION DESC LIMIT 1);
INSERT INTO RECIBE_NOTIFICACION(ID_USER, ID_NOTIFICATION)
SELECT @id_user_2, @id_notificacion_2;
END
```

4. ANEXO

4.1 Modelo Físico

A continuación se incluye el modelo desde el cual se genera la base de datos sobre la que se realizan las consultas, procedimientos almacenados y triggers creados.