Paso 1: Creacion de base de datos, comandos de creacion, tablas y usuarios

Hemos creado la base de datos WePlanDB basado en que los usuarios puedan crear actividades y conectarse mediante aficiones a traves de diferentes grupos en eventos.

Contenido del Paso 1:

`--Creacion de la base de datos--` `CREATE DATABASE WePlanDB; USE WePlanDB;`

Tablas madres:

``` CREATE TABLE usuarios ( usuario_id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(50) NOT NULL, apellidos VARCHAR(150) NOT NULL, telefono VARCHAR(20), direccion VARCHAR(255), correo_electronico VARCHAR(150) UNIQUE NOT NULL, pais VARCHAR(50), contrasenya VARCHAR(255) NOT NULL, fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP, tipo_de_usuario VARCHAR(50) );

CREATE TABLE grupos ( grupo_id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(100) NOT NULL, descripcion TEXT, creador_id INT, fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP, max_miembros INT DEFAULT 50, FOREIGN KEY (creador_id) REFERENCES usuarios(usuario_id) );

CREATE TABLE categorias_actividades ( categoria_id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(50) NOT NULL, descripcion TEXT );

CREATE TABLE aficiones ( aficion_id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(100) NOT NULL, categoria VARCHAR(50) );

CREATE TABLE eventos ( evento_id INT PRIMARY KEY AUTO_INCREMENT, grupo_id INT NOT NULL, subcategoria_id INT, titulo VARCHAR(100) NOT NULL, fecha_hora DATETIME NOT NULL, lugar VARCHAR(200), FOREIGN KEY (grupo_id) REFERENCES grupos(grupo_id), FOREIGN KEY (subcategoria_id) REFERENCES subcategorias_actividades(subcategoria_id) ); ```

# Tablas relacionales:

``` CREATE TABLE usuario_grupo ( usuario_id INT NOT NULL, grupo_id INT NOT NULL, PRIMARY KEY (usuario_id, grupo_id), FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id), FOREIGN KEY (grupo_id) REFERENCES grupos(grupo_id) );

CREATE TABLE usuario_aficion ( usuario_id INT NOT NULL, aficion_id INT NOT NULL, PRIMARY KEY (usuario_id, aficion_id), FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id), FOREIGN KEY (aficion_id) REFERENCES aficiones(aficion_id) );

CREATE TABLE grupo_subcategoria ( grupo_id INT NOT NULL, subcategoria_id INT NOT NULL, PRIMARY KEY (grupo_id, subcategoria_id), FOREIGN KEY (grupo_id) REFERENCES grupos(grupo_id), FOREIGN KEY (subcategoria_id) REFERENCES subcategorias_actividades(subcategoria_id) ); ```

# Tablas hijas:

``` CREATE TABLE evento_participantes ( evento_id INT NOT NULL, usuario_id INT NOT NULL, PRIMARY KEY (evento_id, usuario_id), FOREIGN KEY (evento_id) REFERENCES eventos(evento_id), FOREIGN KEY (usuario_id) REFERENCES usuarios(usuario_id) );

CREATE TABLE subcategorias_actividades ( subcategoria_id INT PRIMARY KEY AUTO_INCREMENT, categoria_id INT NOT NULL, nombre VARCHAR(50) NOT NULL, descripcion TEXT, FOREIGN KEY (categoria_id) REFERENCES categorias_actividades(categoria_id) );

```

Creacion de usuario que podrá acceder a la base de datos:

``` CREATE USER 'usuario-weplan'@'localhost' IDENTIFIED BY 'Usuarioweplan123$'; GRANT USAGE ON . TO 'usuario-weplan'@'localhost'; ALTER USER 'usuario-weplan'@'localhost' REQUIRE NONE WITH MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;

GRANT ALL PRIVILEGES ON WePlanDB.* TO 'usuario-weplan'@'localhost'; FLUSH PRIVILEGES; ```

# Creacion de vistas:

-- Vista para ver grupos con número de miembros `CREATE VIEW vista_grupos_con_miembros AS SELECT g.grupo_id, g.nombre AS grupo, COUNT(ug.usuario_id) AS miembros, u.nombre AS creador, g.descripcion FROM grupos g LEFT JOIN usuario_grupo ug ON g.grupo_id = ug.grupo_id JOIN usuarios u ON g.creador_id = u.usuario_id GROUP BY g.grupo_id;`

-- Vista para eventos próximos `CREATE VIEW vista_eventos_proximos AS SELECT e.evento_id, e.titulo, e.fecha_hora, e.lugar, g.nombre AS grupo, s.nombre AS actividad FROM eventos e JOIN grupos g ON e.grupo_id = g.grupo_id LEFT JOIN subcategorias_actividades s ON e.subcategoria_id = s.subcategoria_id WHERE e.fecha_hora > NOW() ORDER BY e.fecha_hora;`

-- Vista para aficiones de usuarios `CREATE VIEW vista_usuario_aficiones AS SELECT u.usuario_id, u.nombre, u.apellidos, GROUP_CONCAT(a.nombre SEPARATOR ', ') AS aficiones FROM usuarios u LEFT JOIN usuario_aficion ua ON u.usuario_id = ua.usuario_id LEFT JOIN aficiones a ON ua.aficion_id = a.aficion_id GROUP BY u.usuario_id;`

## Paso 2: Contenido de las tablas y explicacion

Tenemos varias tablas madre: usuarios, categorias_actividades, aficiones y incluso grupos como eventos. La tabla usuario es para crear registros unicos que puedan acceder al sitio web, luego tenemos categorias_actividades para Categorias generales (Musica, Deporte, Educacion,etc), aficiones para caso mas especificos.

El flujo de datos entre tablas es el siguiente: `Un usuario se registra (usuarios)-> Define sus aficiones con un formulario web (aficiones y usuario_aficion)-> Busca grupo gracias a las aficiones(grupo y grupo_subcategoria) -> Se une a grupos (grupo y usuario_grupo) Un "host" del grupo crea un evento (grupo y evento) -> el usuario confirma asistencia (usuario,evento y evento_participantes)` ``` -Tabla usuarios- Dentro de la tabla usuarios, tenemos datos personales con nombre, apellidos, teléfono, dirección, credenciales con un correo único y contraseña y otros como
```

país, fecha_registro (util para saber cuestiones de seguridad). Su uso es para el perfil del usuario en el login y mostrar en grupos/eventos.

-Tabla grupos- La tabla grupos es para mostrar directamente en home.php y contiene nombre, descripcion. Para gestiornalo, tiene un creador_id (FK a usuarios), como fecha_creacion además de un limite de miembros con max_miembros.

-Tabla categorias_actividades- La tabla sirve para mostrar los tipos de categorias generales donde los grupos se clasifican. Contiene nombre y una descripción.

-Tabla aficiones- La tabla aficiones sirve para mostrar intereses mas especificos, personales que no pueden ser clasificados dentro de categorias_actividades. Contiene nombre y categoria

-Tabla eventos- La tabla contiene un titulo, fecha_hora y lugar para Programacion de horarios, luego algunas claves foraneas como grupo_id (el grupo que va a realizar el evento) y subcategoria_id (el tipo especifico de actividad a realizar). Es muy util para los grupos usar estos eventos que contiene un rango de actividades en vez de enforcarse en un solo. ```

Y luego tenemos varias tablas hijas: subcategorias_actividades, evento_participantes. ``` -Tabla subcategorias_actividades- La tabla permite que una subcategoria pertenezca a una categoria general. Por ejemplo Padel y Futbol estarian dentro de "Deportes". Sirve para hacer una jeraquía de actividades.

-Tabla evento_participantes- La tabla permite que un usuario participe en un evento. Esto es útil porque permite controlar las asistencias de los mismos, gestion de capacidad, etc. ```

Paso 3: Relaciones,FK y Peticiones JOIN

Relaciones

Algunas relaciones que hemos hecho han sido usuario_grupo, usuario_aficion y grupo_subcategorias para que logren conectarse segun la relacion de mucho a mucho (n-n) para los 3 casos (un grupo puede albergar muchos usuarios y un usuario puede estar dentro de muchos grupos).

Para la tabla usuario_grupo, tenemos que el usuario sabe a que grupo pertenece y viceversa. Para la tabla usuario_aficion tenemos que hay sistema de aficiones que el usuario puede usar y que permite conectar con otras personas. Para la tabla grupo_subcategoria, permite definir las actividades que hace cada grupo.

Foreign Keys

Para las Foreign Keys (Clave Foráneas), en la tabla grupos tenemos **creador_id** que depende de usuario_id. En eventos tenemos **grupo_id** por ser quien hará la actividad y subcategoria_id para describir la actividad a hacer en el evento. En evento_participantes depende de evento_id y usuario_id. En usuario_grupo tiene depedencia en grupo_id y usuario_id.

Como tenemos tablas relacionesles, dependerá de las tablas madres como usuario_aficion que depende de aficion_id y usuario_id y grupo_subcategoria que usa grupo_id y subcategoria_id.

Peticiones JOIN

Para las peticiones JOIN, las hemos usados como para crear las vistas que a continuación serán explicadas.

EL primer JOIN, de grupos, es para usuario_grupo que usara el grupo_id con creador_id. Esto permite traer todos los grupos incluso sin miembros y mostrar el host (creador del grupo) con su informacion de usuario.

Para el JOIN de Vista eventos_proximos usa grupo_id como subcategoria_id. Esto permite traer informacion del grupo organizador, ademas de que los eventos pueden no tener subcategoria asignada.

Y para la tercera vista usuario_aficiones, usa usuario_id como aficion_id. Esto permite trae todos los usuarios incluso sin aficiones (util para gestionar aficiones de usuarios si no logran saber como aplicarse aficiones a ellos mismos). También si el usuario tiene aficiones, trae su nombres; en cambio si no tiene aficiones, devuelve NULL.

Paso 4: Vistas y explicacion

La primera, vista_grupo_con_miembros, permite ver un grupo que alberga los miembros, su creador y descripcion para que pueda ser visto por todos dentro de home.html.

Estructura completa: ``` SELECT: Columnas a mostrar g.grupo_id - ID único del grupo g.nombre AS grupo - Nombre del grupo (renombrado) COUNT(ug.usuario_id) AS miembros - Cuenta miembros u.nombre AS creador - Nombre del usuario creador g.descripcion - Descripción del grupo

FROM grupos g: Tabla principal (alias g)

LEFT JOIN usuario_grupo ug: Une grupos con sus miembros (tabla puente) Usa LEFT para incluir grupos sin miembros

JOIN usuarios u: Une el creador del grupo con tabla usuarios INNER JOIN porque todo grupo tiene creador

GROUP BY g.grupo_id: Agrupa por grupo para que COUNT cuente miembros por grupo ```

La segunda vista_eventos_proximos, es para los usuarios saber cuales solo eventos a futuro, el grupo que lo organiza y la actividad a realizar, ademas puede ser visto por el usuario dentro de notificaciones.html.

Estructura completa: ``` SELECT: e.evento_id - ID del evento e.titulo - Título del evento e.fecha_hora - Cuándo ocurre e.lugar - Dónde ocurre g.nombre AS grupo - Grupo organizador s.nombre AS actividad - Tipo de actividad (puede ser NULL)

FROM eventos e: Tabla principal eventos JOIN grupos g: INNER JOIN obligatorio - todo evento tiene grupo LEFT JOIN subcategorias_actividades s: Eventos pueden no tener subcategoría definida Si no tiene, actividad será NULL

WHERE e.fecha_hora > NOW(): Solo eventos futuros Filtra eventos pasados

ORDER BY e.fecha_hora: Orden cronológico Los más próximos primero ```

Y por ultimo con la tercera vista, vista_usuario_aficiones, le permite al usuario ver su información personal como aficiones que ha selecionado para verlas en su usuario.

Estructura completa: ``` SELECT: u.usuario_id-ID del usuario u.nombre-Nombre del usuario u.apellidos-Apellidos del usuario GROUP_CONCAT(a.nombre SEPARATOR',')AS aficiones: Función

especial MySQL para concatena TODAS las aficiones en una sola cadena separadas por coma y espacio. Ejemplo:"Senderismo,Pintura,Lectura"

FROM usuarios u:Tabla principal usuarios LEFT JOIN usuario_aficion ua: Une usuario con tabla puente de aficiones Usa LEFT para incluir usuarios sin aficiones LEFT JOIN aficiones a: Une tabla puente con nombres de aficiones Segundo LEFT JOIN porque si no hay en la primera,no hay en la segunda

GROUP BY u.usuario_id: Agrupa por usuario Necesario para GROUP_CONCAT Une múltiples filas de aficiones en una sola ```

Paso 5: Comandos del panel de admin, uso y explicacion

Los comandos usar en el panel de admin son:

SHOW TABLES:

Para mostrar la lista de tablas de las base de datos dentro del nav que hemos creado. Muy util para mostrar tabla por tabla cuando se selecciona, es uno de los fundamentos para el panel admin. Util para el READ

SELECT * FROM 'tabla' LIMIT 1;

Sirve para mostrar dentro del main, el cotenido de la tabla entera por sus columnas como sus registros

CRUD:

SHOW COLUMNS FROM 'tabla' para mostrar solo las columnas de la 'tabla'. Util para el CREATE y UPDATE SHOW TABLE LIKE 'tabla' para verificar la existencia de la 'tabla'. Util para el UPDATE INSERT INTO 'tabla' VALUES 'valores' para meter dentro de una tabla los registros a crear. Util para el CREATE DELETE FROM 'tabla' WHERE id_column = "" sirve para borrar dentro de la 'tabla' una columna con id. Util para el DELETE y UPDATE UPDATE 'tabla' SET columna1 = "valor_nuevo" WHERE $id_column = "" sirve para actualizar el registro de una tabla usando su columna. Util para el UPDATE