Pregunta:

Existe otra tabla permisos que se relaciona con la tabla users según la siguiente estructura de tablas de SQL. ¿Cómo cambiaría el código para tener en cuesta esta relación y que el listado de usuarios contuviese otra columna con los permisos que tiene?

-- -----------------------------------------------------

-- Table `permisos`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `permisos` (

`id` INT NOT NULL AUTO\_INCREMENT,

`tipo` VARCHAR(30) NOT NULL,

`descripcion` VARCHAR(50) NOT NULL,

`restringido` TINYINT(1) NOT NULL DEFAULT FALSE,

PRIMARY KEY (`id`))

ENGINE = MyISAM;

-- -----------------------------------------------------

-- Table `permisos\_users`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `permisos\_users` (

`id` INT NOT NULL AUTO\_INCREMENT,

`permiso\_id` INT NULL,

`usuario\_id` INT NULL,

`restriccion` VARCHAR(40) NULL,

PRIMARY KEY (`id`))

ENGINE = MyISAM;

Respueta:

¡Excelente! Añadir relaciones entre tablas es el siguiente paso lógico y fundamental en cualquier API. La relación que describes entre users y permisos a través de una tabla intermedia permisos\_users es un ejemplo clásico de una **relación "Muchos a Muchos" (Many-to-Many)**. Un usuario puede tener muchos permisos y un permiso puede ser asignado a muchos usuarios.

TypeORM y NestJS hacen que gestionar este tipo de relaciones sea muy elegante. A continuación, te muestro cómo modificar el código paso a paso para incorporar esta nueva tabla y mostrar los permisos en el listado.

### **Parte 1: Modificaciones en el Backend (NestJS y TypeORM)**

Aquí es donde definiremos la relación para que TypeORM sepa cómo unir las tablas automáticamente.

#### **Paso 1.1: Crear la Entidad y Módulo para Permiso**

Así como creamos una entidad para User, necesitamos una para Permiso.

1. **Genera el módulo, controlador y servicio** para permisos por buena práctica, aunque no vayamos a crearle endpoints por ahora. En la terminal, dentro de mi-api-gestion:  
   Bash  
   nest g module permisos  
   nest g service permisos --no-spec  
   nest g controller permisos --no-spec
2. **Crea el fichero de la entidad**. Dentro de la nueva carpeta src/permisos, crea una subcarpeta entities y dentro de ella el fichero permiso.entity.ts.  
   **mi-api-gestion/src/permisos/entities/permiso.entity.ts**  
   TypeScript  
   import { User } from '@/users/entities/user.entity';  
   import { Entity, PrimaryGeneratedColumn, Column, ManyToMany } from 'typeorm';  
     
   @Entity({ name: 'permisos' })  
   export class Permiso {  
    @PrimaryGeneratedColumn()  
    id: number;  
     
    @Column({ type: 'varchar', length: 30 })  
    tipo: string;  
     
    @Column({ type: 'varchar', length: 50 })  
    descripcion: string;  
     
    @Column({ type: 'tinyint', width: 1, default: false })  
    restringido: boolean;  
     
    // Definimos el lado inverso de la relación muchos a muchos.  
    // Esto es opcional pero útil si alguna vez necesitas buscar usuarios desde un permiso.  
    @ManyToMany(() => User, (user) => user.permisos)  
    users: User[];  
   }
3. **Configura el PermisosModule** para que reconozca la entidad.  
   **mi-api-gestion/src/permisos/permisos.module.ts**  
   TypeScript  
   import { Module } from '@nestjs/common';  
   import { PermisosService } from './permisos.service';  
   import { PermisosController } from './permisos.controller';  
   import { TypeOrmModule } from '@nestjs/typeorm';  
   import { Permiso } from './entities/permiso.entity';  
     
   @Module({  
    imports: [TypeOrmModule.forFeature([Permiso])], // ¡Importante!  
    controllers: [PermisosController],  
    providers: [PermisosService],  
   })  
   export class PermisosModule {}
4. **Registra el PermisosModule en el AppModule**.  
   **mi-api-gestion/src/app.module.ts** (extracto)  
   TypeScript  
   // ...otras importaciones  
   import { UsersModule } from './users/users.module';  
   import { PermisosModule } from './permisos/permisos.module'; // Importa el nuevo módulo  
     
   @Module({  
    imports: [  
    // ... ConfigModule y TypeOrmModule.forRootAsync  
    UsersModule,  
    PermisosModule, // ¡Añade esta línea!  
    ],  
    // ...  
   })  
   export class AppModule {}

#### **Paso 1.2: Actualizar la Entidad User para definir la Relación**

Este es el paso más importante. Le diremos a la entidad User cómo se relaciona con Permiso a través de la tabla permisos\_users.

Modifica el fichero **mi-api-gestion/src/users/entities/user.entity.ts**:

TypeScript

import { Permiso } from '@/permisos/entities/permiso.entity'; // Importa la entidad Permiso  
import {  
 Entity,  
 PrimaryGeneratedColumn,  
 Column,  
 UpdateDateColumn,  
 ManyToMany, // Importa ManyToMany  
 JoinTable, // Importa JoinTable  
} from 'typeorm';  
  
@Entity({ name: 'users' })  
export class User {  
 // ... (id, username, password, email, etc. se mantienen igual)  
 @PrimaryGeneratedColumn()  
 id: number;  
 // ... resto de las columnas ...  
 @UpdateDateColumn()  
 modified: Date;  
  
 // --- AÑADIR ESTO ---  
 // Definición de la relación Muchos a Muchos  
 @ManyToMany(() => Permiso, { eager: false }) // eager: false es mejor para el rendimiento, los cargaremos cuando los necesitemos  
 @JoinTable({  
 name: 'permisos\_users', // El nombre de tu tabla intermedia  
 joinColumn: {  
 // Configuración de la clave foránea que apunta a esta entidad (User)  
 name: 'usuario\_id',  
 referencedColumnName: 'id',  
 },  
 inverseJoinColumn: {  
 // Configuración de la clave foránea que apunta a la otra entidad (Permiso)  
 name: 'permiso\_id',  
 referencedColumnName: 'id',  
 },  
 })  
 permisos: Permiso[];  
 // --- FIN DE LA ADICIÓN ---  
}

#### **Paso 1.3: Actualizar UsersService para Cargar los Permisos**

Ahora, al buscar usuarios, le diremos a TypeORM que también traiga sus permisos asociados. Esto se hace de forma muy sencilla usando la opción relations.

Modifica el fichero **mi-api-gestion/src/users/users.service.ts**:

TypeScript

// ...  
import { User } from './entities/user.entity';  
  
@Injectable()  
export class UsersService {  
 constructor(  
 @InjectRepository(User)  
 private usersRepository: Repository<User>,  
 ) {}  
  
 async findAll(): Promise<User[]> {  
 return this.usersRepository.find({  
 // El select se mantiene igual para seguridad  
 select: {  
 id: true,  
 username: true,  
 email: true,  
 activated: true,  
 banned: true,  
 last\_login: true,  
 created: true,  
 modified: true,  
 // No es necesario seleccionar 'permisos' aquí, 'relations' se encarga de ello  
 },  
 // ¡Añadimos la relación que queremos cargar!  
 relations: {  
 permisos: true,  
 },  
 order: {  
 id: 'ASC',  
 },  
 });  
 }  
}

¡Eso es todo por el backend! TypeORM ahora entiende la relación y, gracias a la opción relations, ejecutará un JOIN para traer los permisos de cada usuario en una sola consulta eficiente.

### **Parte 2: Modificaciones en el Frontend (Next.js y MUI)**

Ahora que la API devuelve los permisos dentro de cada objeto de usuario, vamos a mostrarlos.

#### **Paso 2.1: Definir los Tipos en el Frontend**

1. Crea un nuevo fichero de tipo para los permisos.  
   **mi-app-gestion/src/types/permiso.ts**  
   TypeScript  
   export interface Permiso {  
    id: number;  
    tipo: string;  
    descripcion: string;  
   }
2. Actualiza el tipo User para que incluya un array de Permiso.  
   **mi-app-gestion/src/types/user.ts**  
   TypeScript  
   import { Permiso } from './permiso'; // Importa el nuevo tipo  
     
   export interface User {  
    id: number;  
    username: string;  
    email: string;  
    activated: boolean;  
    banned: boolean;  
    last\_login: string;  
    created: string;  
    modified: string;  
    permisos: Permiso[]; // ¡Añade esta línea!  
   }

#### **Paso 2.2: Actualizar la Página de la Lista de Usuarios**

Finalmente, modifica el componente para añadir la nueva columna y renderizar los permisos, usando Chip de MUI para que se vea bien.

Modifica el fichero **mi-app-gestion/src/app/users/page.tsx**:

TypeScript

'use client';  
  
// ... (importaciones de react, query, api, etc.)  
import { User } from '@/types/user';  
  
// ... (importaciones de MUI)  
import {  
 // ... Container, Typography, Table, etc.  
 Box, // Importa Box para el layout de los chips  
 Chip,  
} from '@mui/material';  
  
// ... (función fetchUsers sin cambios)  
  
export default function UsersListPage() {  
 const { data: users, isLoading, isError, error } = useQuery<User[], Error>({  
 queryKey: ['users'],  
 queryFn: fetchUsers,  
 });  
  
 // ... (código de isLoading e isError sin cambios)  
  
 return (  
 <Container maxWidth="lg" sx={{ mt: 4, mb: 4 }}>  
 <Typography variant="h4" component="h1" gutterBottom>  
 Lista de Usuarios  
 </Typography>  
 <TableContainer component={Paper}>  
 <Table sx={{ minWidth: 650 }} aria-label="simple table">  
 <TableHead sx={{ backgroundColor: 'grey.200' }}>  
 <TableRow>  
 <TableCell>ID</TableCell>  
 <TableCell>Username</TableCell>  
 <TableCell>Email</TableCell>  
 <TableCell align="center">Estado</TableCell>  
 <TableCell>Permisos</TableCell> {/\* <-- NUEVA COLUMNA \*/}  
 </TableRow>  
 </TableHead>  
 <TableBody>  
 {users?.map((user) => (  
 <TableRow key={user.id} hover>  
 <TableCell component="th" scope="row">{user.id}</TableCell>  
 <TableCell>{user.username}</TableCell>  
 <TableCell>{user.email}</TableCell>  
 <TableCell align="center">  
 {/\* ... (código del chip de estado sin cambios) ... \*/}  
 </TableCell>  
  
 {/\* --- NUEVA CELDA PARA PERMISOS --- \*/}  
 <TableCell>  
 <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 0.5 }}>  
 {user.permisos.map((permiso) => (  
 <Chip key={permiso.id} label={permiso.tipo} size="small" variant="outlined" />  
 ))}  
 </Box>  
 </TableCell>  
 {/\* --- FIN DE LA NUEVA CELDA --- \*/}  
  
 </TableRow>  
 ))}  
 </TableBody>  
 </Table>  
 </TableContainer>  
 </Container>  
 );  
}

### **Resumen de Cambios y Ejecución**

1. **Backend**: Creamos una entidad Permiso, la enlazamos con User a través de una relación @ManyToMany y le dijimos al UsersService que cargara esta relación.
2. **Frontend**: Actualizamos nuestros tipos para reflejar los nuevos datos y modificamos la tabla de MUI para añadir una columna que renderiza los permisos de cada usuario usando componentes Chip.

Para verlo en acción, reinicia tu entorno de Docker para asegurarte de que el backend se ejecuta con los últimos cambios en el código.

Bash

docker-compose up --build

Ahora, cuando navegues a **http://localhost:3000/users**, la tabla tendrá una nueva columna "Permisos", y cada usuario mostrará sus permisos asignados en la base de datos como pequeñas etiquetas.