

EXERCISES QUE TRABAJAREMOS EN EL CUE:

- **EXERCISE 1: CREAR LA ESTRUCTURA DEL PROYECTO E INICIALIZAR LA CONEXIÓN A LA DB.**
- **EXERCISE 2: CREACIÓN DE LOS MODELOS DE LAS ENTIDADES.**
- **EXERCISE 3: CREACIÓN DEL CONTROLADOR.**
- **EXERCISE 4: REALIZAR LAS DISTINTAS CONSULTAS EN EL MODELO RELACIONAL.**

El objetivo de este ejercicio, es plantear una guía paso a paso para implementar un modelo relacional de muchos a muchos (`many_to_many`) entre dos entidades, haciendo uso de Sequelize en `node.js`.

En el análisis de sistemas, una relación de muchos a muchos se refiere a la que existe entre dos entidades A y B, donde un elemento de A puede estar vinculado a muchos elementos de B, y un elemento en B puede estar vinculado a varios elementos de A.

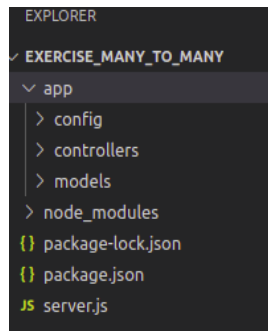
Por ejemplo: suponemos que desea diseñar un modelo de datos de proyectos de usuarios; se puede pensar que un usuario tiene muchos proyectos, y que también el proyecto posee varios usuarios.

Este tipo de relaciones se representa en una base de datos, mediante la creación de una tabla unión. Ejemplo: **`user_project`**, que se conocen como tabla puente, unión o enlace.

Entonces, la relación entre la entidad Usuario y la entidad Proyecto es de uno a muchos.

EXERCISE 1: CREAR LA ESTRUCTURA DEL PROYECTO E INICIALIZAR LA CONEXIÓN A LA DB

Crearemos la estructura inicial del proyecto principal de nombre **`exercise_many_to_many`**, y dentro del mismo, tendremos la carpeta `app` con los directorios `{config, controllers, models}`, esto es:



Procedemos a construir los parámetros de conexión a nuestra base de datos. Para ello, creamos el archivo **db.config.js** dentro de **/app/config**, con el siguiente contenido:

```
1 module.exports = {  
2   HOST: 'localhost',  
3   USER: 'node_user',  
4   PASSWORD: 'node_password',  
5   DB: 'db_node',  
6   dialect: 'postgres',  
7   pool: {  
8     max: 5,  
9     min: 0,  
10    acquire: 30000,  
11    idle: 10000  
12  }  
13 }
```

EXERCISE 2: CREACIÓN DE LOS MODELOS DE LAS ENTIDADES

Creación del modelo de usuario:

Creamos el archivo **user.model.js** dentro de **/app/models**, con el siguiente código:

```
1 module.exports = (sequelize, DataTypes) => {  
2   const User = sequelize.define('users', {  
3     name: {  
4       type: DataTypes.STRING  
5     }  
6   })  
7  
8   return User
```

Creación del modelo de proyecto:

Creamos el archivo **project.model.js** dentro de **/app/models**, con el siguiente código:

```
1 module.exports = (sequelize, DataTypes) => {
2   const Project = sequelize.define('projects', {
3     name: {
4       type: DataTypes.STRING
5     }
6   })
7
8   return Project
9 }
```

Ahora procedemos a crear la conexión a la base de datos, y la relación al modelo. Para ello, generamos un archivo **index.js** dentro de **/app/models**, con el siguiente código:

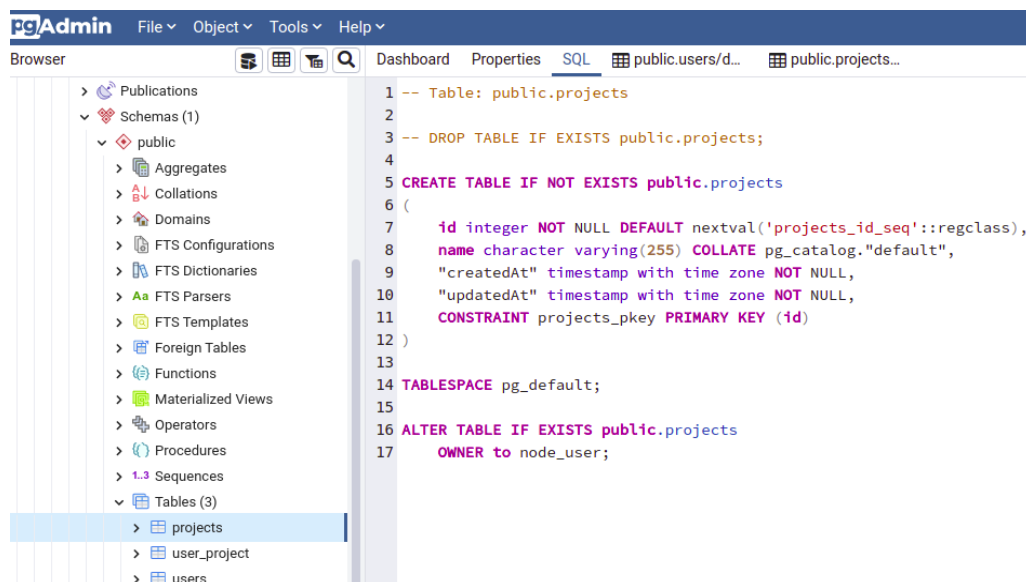
```
1 const dbConfig = require('../config/db.config')
2 const Sequelize = require('sequelize')
3
4 const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER,
5 dbConfig.PASSWORD, {
6   host: dbConfig.HOST,
7   dialect: dbConfig.dialect,
8   operatorAliases: false,
9
10  pool: {
11    max: dbConfig.max,
12    min: dbConfig.min,
13    acquire: dbConfig.acquire,
14    idle: dbConfig.idle
15  }
16 })
17
18 const db = {}
19
20 db.Sequelize = Sequelize
21 db.sequelize = sequelize
22
23 db.users = require('./user.model')(sequelize, Sequelize)
24 db.projects = require('./project.model')(sequelize, Sequelize)
25
26 db.users.belongsToMany(db.projects, {
27   through: "user_project",
28   as: "projects",
29   foreignKey: "user_id",
```

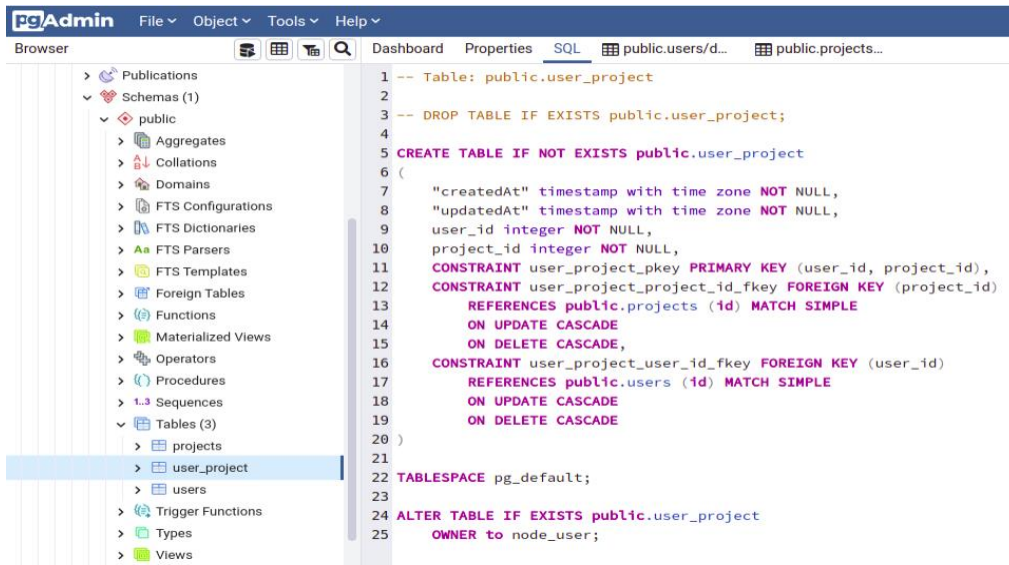
```
30 });  
31 db.projects.belongsToMany(db.users, {  
32   through: "user_project",  
33   as: "users",  
34   foreignKey: "project_id",  
35 });  
36  
37 module.exports = db
```

Para verificar que el modelo está funcionando, procedemos a crear el archivo **server.js** en la carpeta principal del proyecto, con el siguiente código:

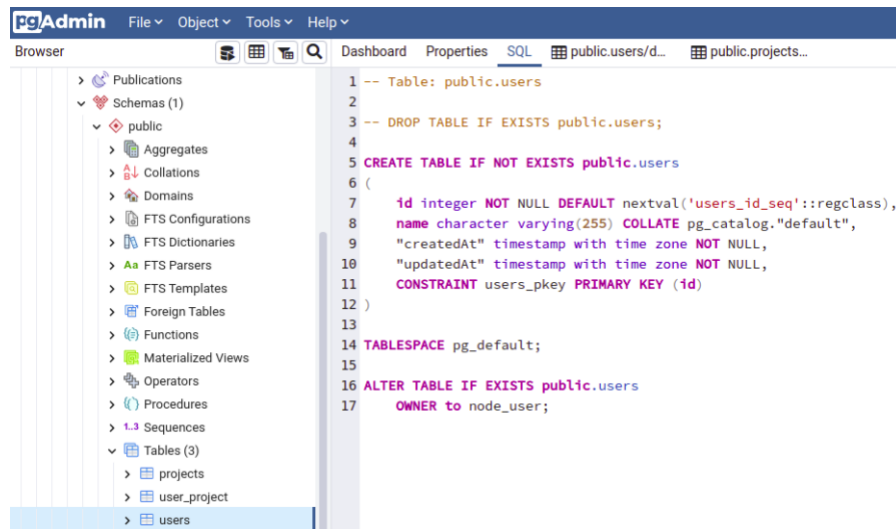
```
1 const db = require('./app/models')  
2  
3 const run = async () => {  
4  
5 }  
6  
7 // db.sequelize.sync()  
8 db.sequelize.sync({  
9   force: true  
10 }).then(() => {  
11   console.log('Eliminando y resincronizando la base de datos.')  
12   run()  
13 })
```

Verificamos en el pgAdmin que se han creado las tablas y las relaciones respectivamente:





```
1 -- Table: public.user_project
2
3 -- DROP TABLE IF EXISTS public.user_project;
4
5 CREATE TABLE IF NOT EXISTS public.user_project
6 (
7     "createdAt" timestamp with time zone NOT NULL,
8     "updatedAt" timestamp with time zone NOT NULL,
9     user_id integer NOT NULL,
10    project_id integer NOT NULL,
11    CONSTRAINT user_project_pkey PRIMARY KEY (user_id, project_id),
12    CONSTRAINT user_project_project_id_fkey FOREIGN KEY (project_id)
13        REFERENCES public.projects (id) MATCH SIMPLE
14        ON UPDATE CASCADE
15        ON DELETE CASCADE,
16    CONSTRAINT user_project_user_id_fkey FOREIGN KEY (user_id)
17        REFERENCES public.users (id) MATCH SIMPLE
18        ON UPDATE CASCADE
19        ON DELETE CASCADE
20 )
21
22 TABLESPACE pg_default;
23
24 ALTER TABLE IF EXISTS public.user_project
25     OWNER to node_user;
```



```
1 -- Table: public.users
2
3 -- DROP TABLE IF EXISTS public.users;
4
5 CREATE TABLE IF NOT EXISTS public.users
6 (
7     id integer NOT NULL DEFAULT nextval('users_id_seq'::regclass),
8     name character varying(255) COLLATE pg_catalog."default",
9     "createdAt" timestamp with time zone NOT NULL,
10    "updatedAt" timestamp with time zone NOT NULL,
11    CONSTRAINT users_pkey PRIMARY KEY (id)
12 )
13
14 TABLESPACE pg_default;
15
16 ALTER TABLE IF EXISTS public.users
17     OWNER to node_user;
```

EXERCISE 3: CREACIÓN DEL CONTROLADOR

Para la construcción del controlador User, creamos el archivo **user.controller.js**, dentro de **/app/controllers**, en el cual se definen: los métodos para crear un usuario, crear un proyecto, consultas de cómo buscar proyectos según el id del usuario, buscar los proyectos por id, obtener todos los Usuarios incluyendo los proyectos, entre otros.

Allí, agregamos al archivo **user.controller.js** el siguiente código:

```
1 const {
2   users
3 } = require('../models')
4 const db = require('../models')
5 const User = db.users
6 const Project = db.projects
7
8 // Crear y Guardar Usuarios
9 exports.createUser = (user) => {
10   return User.create({
11     name: user.name
12   })
13   .then(user => {
14     console.log(`>>> Se ha creado el usuario: ${JSON.stringify(user,
15 null, 4)}`)
16     return user
17   })
18   .catch(err => {
19     console.log(`>>> Error al crear el usuario ${err}`)
20   })
21 }
22
23 // obtener los proyectos de un usuario
24 exports.findUserById = (userId) => {
25   return User.findPk(userId, {
26     include: [{
27       model: Project,
28       as: "projects",
29       attributes: ["id", "name"],
30       through: {
31         attributes: [],
32       }
33     }, ],
34   })
35   .then(users => {
36     return users
37   })
38   .catch(err => {
39     console.log(`>>> Error mientras se encontraba los usuarios:
40 ${err}`)
41   })
42 }
43
44 // obtener todos los Usuarios incluyendo los proyectos
45 exports.findAll = () => {
46   return User.findAll({
47     include: [{
48       model: Project,
```

```
49     as: "projects",
50     attributes: ["id", "name"],
51     through: {
52       attributes: [],
53     },
54   }, ],
55 }).then(users => {
56   return users
57 })
58 }
```

Ahora, para la construcción del controlador Project, generamos el archivo **project.controller.js** dentro de **/app/controllers**, en el cual se definen los métodos para crear un proyecto:

```
1 const {
2   users,
3   projects
4 } = require('../models')
5 const db = require('../models')
6 const Project = db.projects
7 const User = db.users
8
9 // Crear y guardar un nuevo proyecto
10 exports.createProject = (project) => {
11   return Project.create({
12     name: project.name,
13   })
14   .then(project => {
15     console.log(`>>> Creado el proyecto: ${JSON.stringify(project,
16 null, 4)}`)
17     return project
18   })
19   .catch(err => {
20     console.log(`>>> Error al crear el proyecto: ${err}`)
21   })
22 }
23
24 // Agregar un Usuario al Proyecto
25 exports.addUser = (projectId, userId) => {
26   return Project.findById(projectId)
27   .then((project) => {
28     if (!project) {
29       console.log("No se encontro el proyecto!");
30       return null;
31     }
32     return User.findById(userId).then((user) => {
```

```
33     if (!user) {
34         console.log("Usuario no encontrado!");
35         return null;
36     }
37     project.addUser(user);
38     console.log('*****')
39     console.log(`>> Agregado el usuario id=${user.id} al proyecto
40 con id=${project.id}`);
41     console.log('*****')
42     return project;
43 });
44 })
45 .catch((err) => {
46     console.log(">> Error mientras se estaba agregando Usuario al
47 Proyecto", err);
48 });
49 };
50
51
52 // obtener los proyectos por id
53 exports.findById = (Id) => {
54     return Project.findByPk(Id, {
55         include: [{
56             model: User,
57             as: "users",
58             attributes: ["id", "name"],
59             through: {
60                 attributes: [],
61             }
62         }, ],
63     })
64     .then(project => {
65         return project
66     })
67     .catch(err => {
68         console.log(`>> Error mientras se encontraba el proyecto:
69 ${err}`)
70     })
71 }
72
73 // obtener todos los Usuarios incluyendo los comentarios
74 exports.findAll = () => {
75     return Project.findAll({
76         include: [{
77             model: User,
78             as: "users",
79             attributes: ["id", "name"],
80             through: {
```



```
81     attributes: [],  
82   }  
83 }, ],  
84 }).then(projects => {  
85   return projects  
86 }).catch((err) => {  
87   console.log(">> Error Buscando los proyectos: ", err);  
88 });  
89 }
```

Adecuamos al archivo **server.js**, agregando el controlador:

```
1 const db = require('./app/models')  
2 const userController = require('./app/controllers/user.controller')  
3 const projectController =  
4 require('./app/controllers/project.controller')  
5  
6 const run = async () => {  
7   // db.sequelize.sync()  
8   db.sequelize.sync({  
9     force: true  
10  }).then(() => {  
11    console.log('Eliminando y resincronizando la base de datos.')  
12    run()  
13  })
```

EXERCISE 4: REALIZAR LAS DISTINTAS CONSULTAS EN EL MODELO RELACIONAL

Verificamos la creación e inserción de usuarios, adecuando el archivo **server.js** con el siguiente código:

```
1 // Crear un Usuario  
2 const user1 = await userController.createUser({  
3   name: 'José Alberto',  
4 })  
5  
6 const user2 = await userController.createUser({  
7   name: 'Carlos Mejias',  
8 })
```

Observamos la salida en la terminal:

```
1 >> Se ha creado el usuario: {
2   "id": 1,
3   "name": "José Alberto",
4   "updatedAt": "2022-03-25T15:29:19.077Z",
5   "createdAt": "2022-03-25T15:29:19.077Z"
6 }
7 Executing (default): INSERT INTO "users"
8 ("id","name","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3)
9 RETURNING "id","name","createdAt","updatedAt";
10 >> Se ha creado el usuario: {
11   "id": 2,
12   "name": "Carlos Mejias",
13   "updatedAt": "2022-03-25T15:29:19.101Z",
14   "createdAt": "2022-03-25T15:29:19.101Z"
15 }
```

Creando los proyectos, se adecua el archivo server.js agregando:

```
1 // Crear un proyecto
2 const project1 = await projectController.createProject({
3   name: 'Proyecto A',
4 })
5
6 const project2 = await projectController.createProject({
7   name: 'Proyecto X',
8 })
```

Salida en la terminal:

```
1 >> Creado el proyecto: {
2   "id": 1,
3   "name": "Proyecto A",
4   "updatedAt": "2022-03-25T15:30:20.878Z",
5   "createdAt": "2022-03-25T15:30:20.878Z"
6 }
7 Executing (default): INSERT INTO "projects"
8 ("id","name","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3)
9 RETURNING "id","name","createdAt","updatedAt";
10 >> Creado el proyecto: {
11   "id": 2,
12   "name": "Proyecto X",
13   "updatedAt": "2022-03-25T15:30:20.900Z",
```

```
14   "createdAt": "2022-03-25T15:30:20.900Z"  
15 }
```

Agregando las relaciones:

```
1 await projectController.addUser(project1.id, user1.id);  
2 await projectController.addUser(project1.id, user2.id);  
3 await projectController.addUser(project2.id, user1.id);
```

Realizamos las consultas, adecuando en el archivo **server.js**:

```
1 // Consultado el proyecto(id) incluyendo los usuarios  
2 const _project1 = await projectController.findById(project1.id);  
3 console.log(" proyecto  ", JSON.stringify(_project1, null, 2));
```

Salida en la terminal:

```
1 >> Creado el proyecto: {  
2   "id": 1,  
3   "name": "Proyecto A",  
4   "updatedAt": "2022-03-25T15:32:54.665Z",  
5   "createdAt": "2022-03-25T15:32:54.665Z"  
6 }  
7 Executing (default): INSERT INTO "projects"  
8 ("id","name","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3)  
9 RETURNING "id","name","createdAt","updatedAt";  
10 >> Creado el proyecto: {  
11   "id": 2,  
12   "name": "Proyecto X",  
13   "updatedAt": "2022-03-25T15:32:54.690Z",  
14   "createdAt": "2022-03-25T15:32:54.690Z"  
15 // Obtener los proyectos por ID  
16 const projectData = await controller.findProjectById(project1.id)  
17 console.log(  
18   '>> Proyecto id=' + project1.id,  
19   JSON.stringify(projectData, null, 2)  
20 )  
21  
22 // Consultado todos los proyectos  
23 const projects = await projectController.findAll();  
24 console.log(" Proyectos: ", JSON.stringify(projects, null, 2));
```

Salida en la terminal:

```
1 Proyectos: [
2   {
3     "id": 1,
4     "name": "Proyecto A",
5     "createdAt": "2022-03-25T15:33:26.171Z",
6     "updatedAt": "2022-03-25T15:33:26.171Z",
7     "users": [
8       {
9         "id": 1,
10        "name": "José Alberto"
11      },
12      {
13        "id": 2,
14        "name": "Carlos Mejias"
15      }
16    ]
17  },
18  {
19    "id": 2,
20    "name": "Proyecto X",
21    "createdAt": "2022-03-25T15:33:26.184Z",
22    "updatedAt": "2022-03-25T15:33:26.184Z",
23    "users": [
24      {
25        "id": 1,
26        "name": "José Alberto"
27      }
28    ]
29  }
30 ]
31 // Consultado los usuarios (id) incluyendo los proyectos
32 const _user = await userController.findUserById(user1.id);
33 console.log(" user1: ", JSON.stringify(_user, null, 2));
```

Salida en la terminal:

```
1 Proyectos: [
2   {
3     "id": 1,
4     "name": "Proyecto A",
5     "createdAt": "2022-03-25T15:35:27.519Z",
6     "updatedAt": "2022-03-25T15:35:27.519Z",
7     "users": [
8       {
```

```
9      "id": 1,
10     "name": "José Alberto"
11   },
12   {
13     "id": 2,
14     "name": "Carlos Mejias"
15   }
16 ]
17 },
18 {
19   "id": 2,
20   "name": "Proyecto X",
21   "createdAt": "2022-03-25T15:35:27.525Z",
22   "updatedAt": "2022-03-25T15:35:27.525Z",
23   "users": [
24     {
25       "id": 1,
26       "name": "José Alberto"
27     }
28   ]
29 }
30 ]
31
32
33 //Consultado todos los usuarios con sus proyectos
34 const users = await userController.findAll();
35 console.log(">> usuarios: ", JSON.stringify(users, null, 2));
36 }
```

Salida en la terminal:

```
1 >> usuarios: [
2   {
3     "id": 1,
4     "name": "José Alberto",
5     "createdAt": "2022-03-25T15:36:44.871Z",
6     "updatedAt": "2022-03-25T15:36:44.871Z",
7     "projects": [
8       {
9         "id": 1,
10        "name": "Proyecto A"
11      },
12      {
13        "id": 2,
14        "name": "Proyecto X"
15      }
16    ]
17   }
18 ]
```

```
16 ]
17 },
18 {
19   "id": 2,
20   "name": "Carlos Mejias",
21   "createdAt": "2022-03-25T15:36:44.926Z",
22   "updatedAt": "2022-03-25T15:36:44.926Z",
23   "projects": [
24     {
25       "id": 1,
26       "name": "Proyecto A"
27     }
28   ]
29 }
30 ]
```