

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 uno a muchos (`one_to_many`) entre dos entidades, haciendo uso de Sequelize en `node.js`.

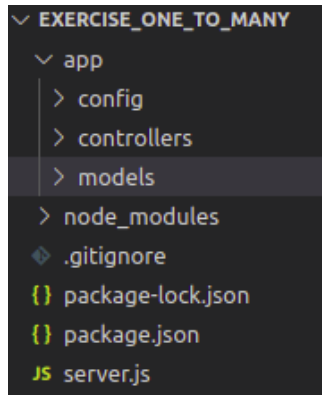
En el análisis de sistemas, una relación de uno a muchos se refiere a la que existe entre dos entidades A y B, en la que un elemento de A puede estar vinculado a muchos elementos de B, pero un miembro de B está vinculado a un solo elemento de A.

Por ejemplo: suponemos que se desea diseñar un modelo de datos de proyectos de usuarios; se puede pensar que un usuario tiene muchos proyectos, pero el proyecto solo pertenece a un usuario.

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_one_to_many`**, y dentro del mismo, poseemos 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 }
```

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:

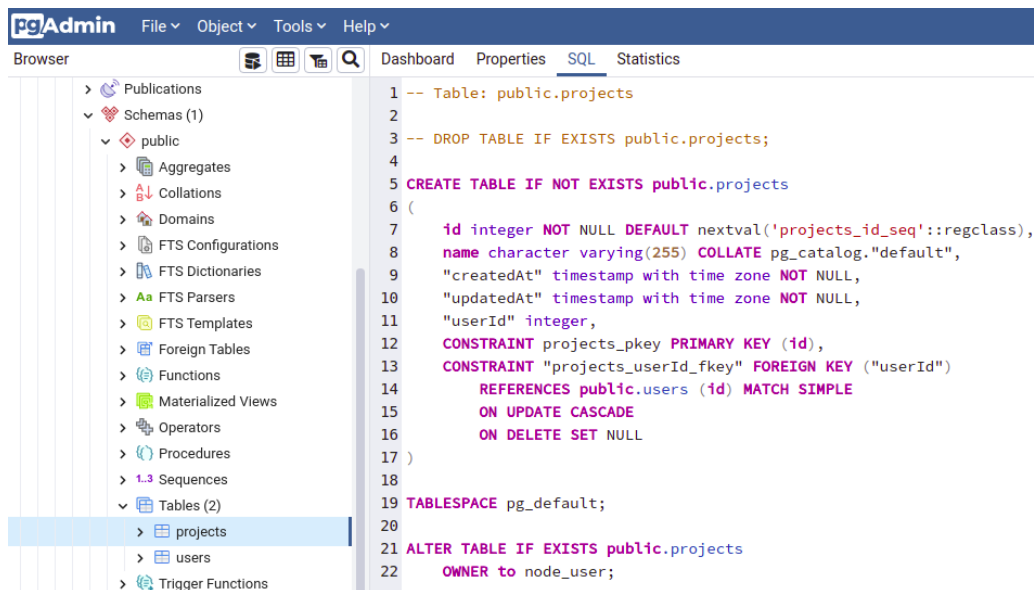
```
1 const dbConfig = require('../config/db.config')
2 const Sequelize = require('sequelize')
3
4 const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER,
5 dbConfig.PASSWORD, {
6   host: dbConfig.HOST,
7   dialect: dbConfig.dialect,
8   operatorAliases: false,
9
10  pool: {
11    max: dbConfig.max,
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.hasMany(db.projects, {
27   as: 'projects'
28 })
29
30 db.projects.belongsTo(db.users, {
```

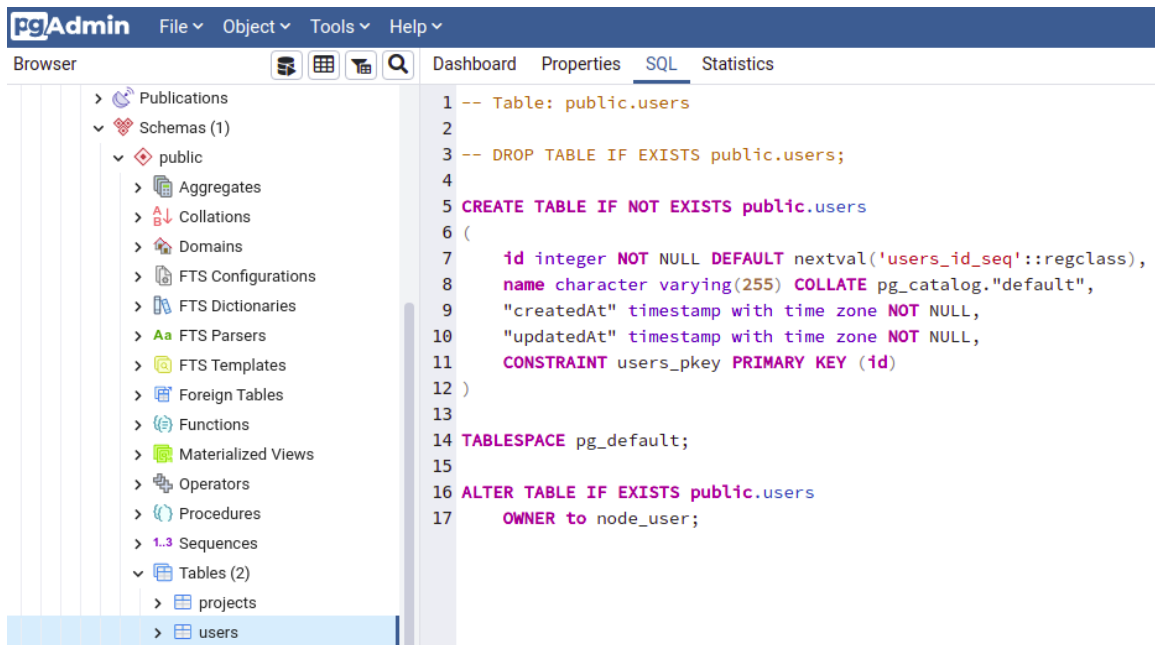
```
30   foreignkey: 'userId',
31   as: 'user'
32 })
33
34 module.exports = db
```

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 // db.sequelize.sync()
7 db.sequelize.sync({
8   force: true
9 }).then(() => {
10   console.log('Eliminando y resincronizando la base de datos.')
11   run()
12 })
```

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





The screenshot shows the pgAdmin interface with the SQL editor open. The left sidebar shows the database structure, including Schemas (1) and public. The SQL editor contains the following code:

```
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, creamos el archivo **user.controller.js** dentro de **/app/controllers**, en el cual se definirán los métodos para: crear un usuario, crear un proyecto, consultas sobre cómo buscar proyectos según el id del usuario, buscar los proyectos por id, y obtener todos los Usuarios incluyendo los proyectos.

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 // Crear y guardar un nuevo proyecto
24 exports.createProject = (userId, project) => {
25   return Project.create({
26     name: project.name,
27     userId: userId
28   })
29   .then(project => {
30     console.log(`>>> Creado el proyecto: ${JSON.stringify(project,
31 null, 4)}`)
32     return project
33   })
34   .catch(err => {
35     console.log(`>>> Error al crear el proyecto: ${err}`)
36   })
37 }
38
39 // obtener los proyectos de un usuario
40 exports.findUserById = (userId) => {
41   return User.findById(userId, {
42     include: ['projects']
43   })
44   .then(user => {
45     return user
46   })
47   .catch(err => {
48     console.log(`>>> Error mientras se encontraba los usuario:
```

```
49 ${err}`)
50   })
51 }
52
53 // Obtener los Proyectos por el id del proyecto
54 exports.findProjectById = (id) => {
55   return Project.findByPk(id, {
56     include: ['user']
57   })
58   .then(project => {
59     return project
60   })
61   .catch(err => {
62     console.log(`>> Error buscado los proyectos: ${err}`)
63   })
64 }
65
66 // obtener todos los Usuarios incluyendo los proyectos
67 exports.findAll = () => {
68   return User.findAll({
69     include: ['projects']
70   }).then(users => {
71     return users
72   })
73 }
```

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

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

EXERCISE 4: REALIZAR LAS DISTINTAS CONSULTAS EL MODELO RELACIONAL

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

```
1 const db = require('./app/models')
2 const controller = require('./app/controllers/user.controller')
3
4 const run = async () => {
5
6   // Crear un Usuario
7   const user1 = await controller.createUser({
8     name: 'José Alberto',
9   })
10
11   const user2 = await controller.createUser({
12     title: 'Carlos Mejias',
13   })
14 }
15
16 // db.sequelize.sync()
17 db.sequelize.sync({
18   force: true
19 }).then(() => {
20   console.log('Eliminando y resincronizando la base de datos.')
21   run()
22 })
```

Observamos la salida en la terminal:

```
1 >> Se ha creado el usuario: {
2   "id": 1,
3   "name": "José Alberto",
4   "updatedAt": "2022-03-25T13:34:08.501Z",
5   "createdAt": "2022-03-25T13:34:08.501Z"
6 }
7 Executing (default): INSERT INTO "users"
8 ("id","createdAt","updatedAt") VALUES (DEFAULT,$1,$2) RETURNING
9 "id","name","createdAt","updatedAt";
10 >> Se ha creado el usuario: {
11   "id": 2,
12   "name": null,
13   "updatedAt": "2022-03-25T13:34:08.528Z",
14   "createdAt": "2022-03-25T13:34:08.528Z"
```


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

```
1 // Crear un proyecto
2 const project1 = await controller.createProject(user1.id, {
3   name: 'Proyecto A',
4 })
5
6 await controller.createProject(user1.id, {
7   name: 'Proyecto B',
8 })
9
10 const project2 = await controller.createProject(user2.id, {
11   name: 'Proyecto X',
12 })
13
14 await controller.createProject(user2.id, {
15   name: 'Proyecto Z',
16 })
```

Salida en la terminal:

```
1 >> Creado el proyecto: {
2   "id": 1,
3   "name": "Proyecto A",
4   "userId": 1,
5   "updatedAt": "2022-03-25T13:38:29.980Z",
6   "createdAt": "2022-03-25T13:38:29.980Z"
7 }
8 Executing (default): INSERT INTO "projects"
9 ("id","name","createdAt","updatedAt","userId") VALUES
10 (DEFAULT,$1,$2,$3,$4) RETURNING
11 "id","name","createdAt","updatedAt","userId";
12 >> Creado el proyecto: {
13   "id": 2,
14   "name": "Proyecto B",
15   "userId": 1,
16   "updatedAt": "2022-03-25T13:38:29.995Z",
17   "createdAt": "2022-03-25T13:38:29.995Z"
18 }
19 Executing (default): INSERT INTO "projects"
20 ("id","name","createdAt","updatedAt","userId") VALUES
21 (DEFAULT,$1,$2,$3,$4) RETURNING
22 "id","name","createdAt","updatedAt","userId";
23 >> Creado el proyecto: {
24   "id": 3,
25   "name": "Proyecto X",
```

```
26   "userId": 2,  
27   "updatedAt": "2022-03-25T13:38:30.014Z",  
28   "createdAt": "2022-03-25T13:38:30.014Z"  
29 }  
30 Executing (default): INSERT INTO "projects"  
31 ("id","name","createdAt","updatedAt","userId") VALUES  
32 (DEFAULT,$1,$2,$3,$4) RETURNING  
33 "id","name","createdAt","updatedAt","userId";  
34 >> Creado el proyecto: {  
35   "id": 4,  
36   "name": "Proyecto Z",  
37   "userId": 2,  
38   "updatedAt": "2022-03-25T13:38:30.040Z",  
39   "createdAt": "2022-03-25T13:38:30.040Z"
```

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

```
1  // obtener los usuarios por id  
2  const userData = await controller.findUserById(user1.id)  
3  console.log(  
4    '>> Usuario id=' + userData.id,  
5    JSON.stringify(userData, null, 2)  
6  )
```

Salida en la terminal:

```
1  >> Usuario id=1 {  
2    "id": 1,  
3    "name": "José Alberto",  
4    "createdAt": "2022-03-25T13:41:51.668Z",  
5    "updatedAt": "2022-03-25T13:41:51.668Z",  
6    "projects": [  
7      {  
8        "id": 1,  
9        "name": "Proyecto A",  
10       "createdAt": "2022-03-25T13:41:51.705Z",  
11       "updatedAt": "2022-03-25T13:41:51.705Z",  
12       "userId": 1  
13     },  
14     {  
15       "id": 2,  
16       "name": "Proyecto B",  
17       "createdAt": "2022-03-25T13:41:51.714Z",  
18       "updatedAt": "2022-03-25T13:41:51.714Z",
```

```
19     "userId": 1
20   }
21 ]
22 }
23 // Obtener los proyectos por ID
24 const projectData = await controller.findProjectById(project1.id)
25 console.log(
26   '>> Proyecto id=' + project1.id,
27   JSON.stringify(projectData, null, 2)
28 )
```

Salida en la terminal:

```
1 >> Proyecto id=1 {
2   "id": 1,
3   "name": "Proyecto A",
4   "createdAt": "2022-03-25T13:42:39.569Z",
5   "updatedAt": "2022-03-25T13:42:39.569Z",
6   "userId": 1,
7   "user": {
8     "id": 1,
9     "name": "José Alberto",
10    "createdAt": "2022-03-25T13:42:39.535Z",
11    "updatedAt": "2022-03-25T13:42:39.535Z"
12  }
13 }
14
15
16 // obtener todos los Usuarios
17 const users = await controller.findAll()
18 console.log('>> Todos los Usuarios ', JSON.stringify(users, null, 2))
```

Salida en la terminal:

```
1 >> Todos los Usuarios [
2   {
3     "id": 1,
4     "name": "José Alberto",
5     "createdAt": "2022-03-25T13:43:44.996Z",
6     "updatedAt": "2022-03-25T13:43:44.996Z",
7     "projects": [
8       {
9         "id": 1,
10        "name": "Proyecto A",
```

```
11     "createdAt": "2022-03-25T13:43:45.033Z",
12     "updatedAt": "2022-03-25T13:43:45.033Z",
13     "userId": 1
14   },
15   {
16     "id": 2,
17     "name": "Proyecto B",
18     "createdAt": "2022-03-25T13:43:45.043Z",
19     "updatedAt": "2022-03-25T13:43:45.043Z",
20     "userId": 1
21   }
22 ]
23 },
24 {
25   "id": 2,
26   "name": null,
27   "createdAt": "2022-03-25T13:43:45.022Z",
28   "updatedAt": "2022-03-25T13:43:45.022Z",
29   "projects": [
30     {
31       "id": 3,
32       "name": "Proyecto X",
33       "createdAt": "2022-03-25T13:43:45.054Z",
34       "updatedAt": "2022-03-25T13:43:45.054Z",
35       "userId": 2
36     },
37     {
38       "id": 4,
39       "name": "Proyecto Z",
40       "createdAt": "2022-03-25T13:43:45.062Z",
41       "updatedAt": "2022-03-25T13:43:45.062Z",
42       "userId": 2
43     }
44   ]
45 }
46 ]
```