HINTS

CONSEJOS CONCEPTUALES

- El método **findAll()** nos entrega todos los registros de la tabla, y para acceder a ellos, simplemente se ejecuta el método **toJSON()**, el cual nos da un objeto con la información. **findAll()** posee variadas opciones para hacer consultas más complejas.
- Sequelize mantendrá la conexión a la base de datos abierta por defecto, y utilizará la misma para todas las querys. Si es necesario cerrar la conexión, se invoca el método sequelize.close(), el cual es asíncrono y devolverá una promesa (promise).
- Recuerde que Sequelize se refiere a la biblioteca (library) en sí. Mientras que sequelize se refiere a una instancia dentro de Sequelize, la cual representa la conexión a una base de datos.
- Durante la definición de un modelo, el nombre de la tabla no es especificado explícitamente. Cuando el nombre de la tabla no está dado, Sequelize automáticamente pluraliza el nombre del modelo, y lo utiliza como nombre de la tabla. La pluralización se realiza a través de la biblioteca (library) inflection, por lo tanto, los plurales como person → people serían incorrectos. Sin embargo, este comportamiento es configurable.
- La auto-pluralización llevada a cabo por Sequelize, también puede ser detenida utilizando la opción freezeTableName: true. De esta manera, inferirá que el nombre de la tabla debe ser igual al del modelo.
- Es posible especificar el nombre de la tabla, directamente en la definición del modelo. Se hace de la siguiente manera:

```
sequelize.define('Usuario', {
 // ... (atributos)
}, {
 tableName: 'Empleados'
});
```

- Existe una gran variedad de tipos de datos en Sequelize que pueden ser utilizados. Se muestran a continuación:

  Strings:

- DataTypes.STRING -- VARCHAR(255)
- DataTypes.STRING(1234) -- VARCHAR(1234)
- DataTypes.STRING.BINARY -- VARCHAR BINARY
- DataTypes.TEXT -- TEXT
- DataTypes.TEXT('tiny') -- TINYTEXT
- DataTypes.CITEXT -- CITEXT PostgreSQL y SQLite solamente.
- DataTypes.TSVECTOR -- TSVECTOR PostgreSQL solamente.

Boolean:

- DataTypes.BOOLEAN -- TINYINT(1)

Números:

- DataTypes.INTEGER -- INTEGER
- DataTypes.BIGINT -- BIGINT
- DataTypes.BIGINT(11) -- BIGINT(11)
- DataTypes.FLOAT -- FLOAT
- DataTypes.FLOAT(11) -- FLOAT(11)
- DataTypes.FLOAT(11, 10) -- FLOAT(11,10)
- DataTypes.REAL -- REAL PostgreSQL solamente.
- DataTypes.REAL(11) -- REAL(11) PostgreSQL solamente.
- DataTypes.REAL(11, 12) -- REAL(11,12) PostgreSQL solamente.
- DataTypes.DOUBLE -- DOUBLE
- DataTypes.DOUBLE(11) -- DOUBLE(11)
- DataTypes.DOUBLE(11, 10) -- DOUBLE(11,10)
- DataTypes.DECIMAL -- DECIMAL
- DataTypes.DECIMAL(10, 2) -- DECIMAL(10,2)

Fechas:

- DataTypes.DATE -- TIMESTAMP WITH TIME ZONE para postgres
- DataTypes.DATEONLY -- DATE sin hora

- Para realizar consultas listas para la producción con Sequelize, se deben incluir transacciones, las cuales son importantes para garantizar la integridad de los datos, y proporcionar otros beneficios.
- Sequelize provee una gran cantidad de operadores:

```
const { Op } = require("sequelize");
Post.findAll({
        where: {
                [Op.and]: [{ a: 5 }, { b: 6 }], // (a = 5) AND (b =
6)
                [Op.or]: [{ a: 5 }, { b: 6 }], // (a = 5) OR (b = 6)
                someAttribute: {
                        // Basicos
                        [Op.eq]: 3, // = 3
                        [Op.ne]: 20, // != 20
                        [Op.is]: null, // IS NULL
                        [Op.not]: true, // IS NOT TRUE
                        [Op.or]: [5, 6], // (someAttribute = 5) OR
(someAttribute = 6) // Uso de identificadores de columna específicos
                        [Op.col]: 'user.organization_id', // =
"user"."organization_id" // Comparación de números
                        [Op.gt]: 6, // > 6
                        [Op.gte]: 6, // >= 6
                        [Op.lt]: 10, // < 10
                        [Op.lte]: 10, // <= 10
                        [Op.between]: [6, 10], // BETWEEN 6 AND 10
                        [Op.notBetween]: [11, 15], // NOT BETWEEN 11
AND 15 //
                        [Op.all]: sequelize.literal('SELECT 1'), // >
ALL (SELECT 1)
                        [Op.in]: [1, 2], // IN [1, 2]
                        [Op.notIn]: [1, 2], // NOT IN [1, 2]
                        [Op.like]: '%hat', // LIKE '%hat'
                        [Op.notLike]: '%hat', // NOT LIKE '%hat'
                        [Op.startsWith]: 'hat', // LIKE 'hat%'
                        [Op.endsWith]: 'hat', // LIKE '%hat'
                        [Op.substring]: 'hat', // LIKE '%hat%'
                        [Op.iLike]: '%hat', // ILIKE '%hat'
(mayúsculas y minúsculas) (PG solamente)
                        [Op.notILike]: '%hat', // NOT ILIKE '%hat'
(PG solamente)
                        [Op.regexp]: '^[h|a|t]', // REGEXP/~
```

```
'^[h|a|t]' (MySQL/PG solamente)
                        [Op.notRegexp]: '^[h|a|t]', // NOT REGEXP/!~
'^[h|a|t]' (MySQL/PG solamente)
                        [Op.iRegexp]: '^[h|a|t]', // ~* '^[h|a|t]'
(PG solamente)
                        [Op.notIRegexp]: '^[h|a|t]', // !~*
'^[h|a|t]' (PG solamente)
                        [Op.any]: [2, 3], // ANY ARRAY[2, 3]::INTEGER
(PG solamente)
                        [Op.match]: Sequelize.fn('to_tsquery', 'fat &
rat') // Búsqueda de texto de coincidencia para las cadenas 'fat'
and 'rat' (PG solamente) /
                        [Op.like]: { [Op.any]: ['cat', 'hat'] } //
LIKE ANY ARRAY['cat', 'hat']
                    }
            }
 });
```