



Bases de Datos: Sequelize

5-3-2020 - V6

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Bases de Datos y ORM sequelize - Índice

1. Relaciones entre modelos	3
2. Proyecto user-quiz: Tabla Quizzes, Modelo, Relación y Comandos	12
3. Proyecto user-quiz: Relación Favourites, Modelo y Comandos	23
4. Proyecto user-quiz: Sequelize-cli, migraciones y seeders	32



Relaciones entre modelos

Juan Quemada, DIT - UPM
Santiago Pavón, DIT - UPM

Relaciones o asociaciones entre modelos

◆ Una **relación** define una asociación entre modelos

- Hay tres tipos de relaciones básicas: **1-1**, **1-N** y **N-N**
 - ◆ <https://sequelize.org/master/manual/assocs.html>
- Las relaciones se definen combinando estos **4 métodos**
 - ◆ `AhasOne(B)`, `A.belongsTo(B)`, `AhasMany(B)`, `A.belongsToMany(B)`

◆ **Relación 1-1:** asocia 1 elemento con un solo elemento de otra tabla

- Por ejemplo, una **usuario** tiene asociado un solo **DNI**
 - ◆ Definición: `User.hasOne(DNI)` y `DNI.belongsTo(User)`

◆ **Relación 1-N:** asocia 1 elemento con N elementos en otra tabla

- Por ejemplo, una **usuario** es autor de varios **quizzes**
 - ◆ Definición: `UserhasMany(Quiz)` y `Quiz.belongsTo(User)`

◆ **Relación N-N:** asocia N elementos con N elementos en otra tabla

- Por ejemplo, cada **usuario** tienen varios **quizzes favoritos**
 - ◆ Definición: `User.belongsToMany(Quiz, ...)` y `Quiz.belongsToMany(User, ...)`

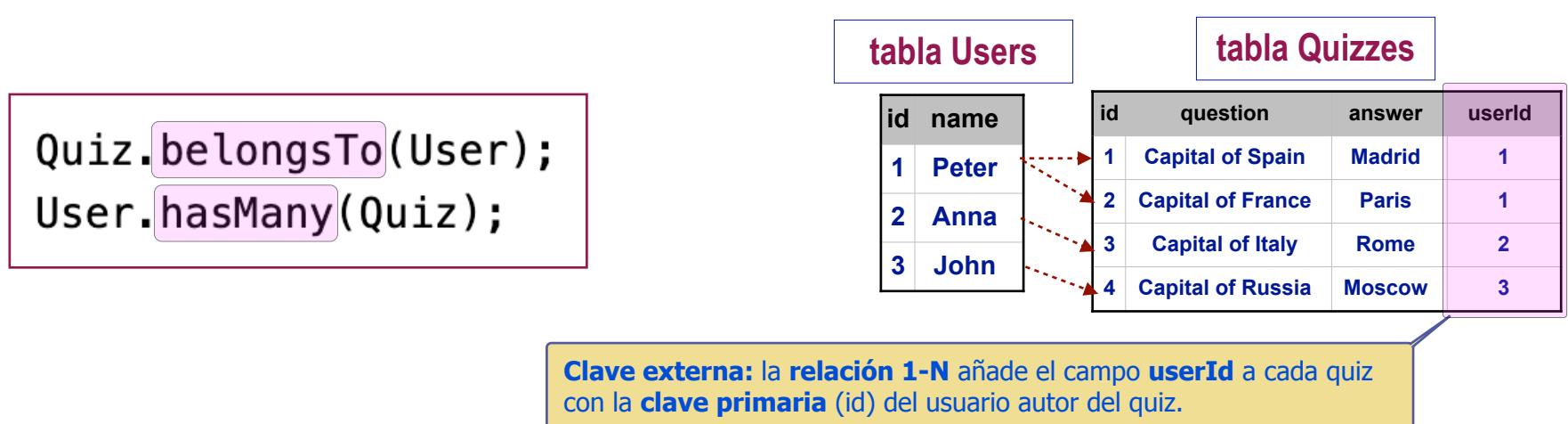
Relación 1-N

◆ La relación "autor de quizzes" es una relación 1-N

- Relaciona a un usuario con los quizzes que ha creado
 - ♦ La relación tiene dos sentidos, que se definen con `UserhasMany(Quiz)` y `Quiz.belongsTo(User)`
 - <https://sequelize.org/master/manual/assocs.html#one-to-many-relationships>

◆ La relación añade una columna `userId` a la tabla `Quizzes`

- Los campos contienen la **clave externa** del autor del quiz



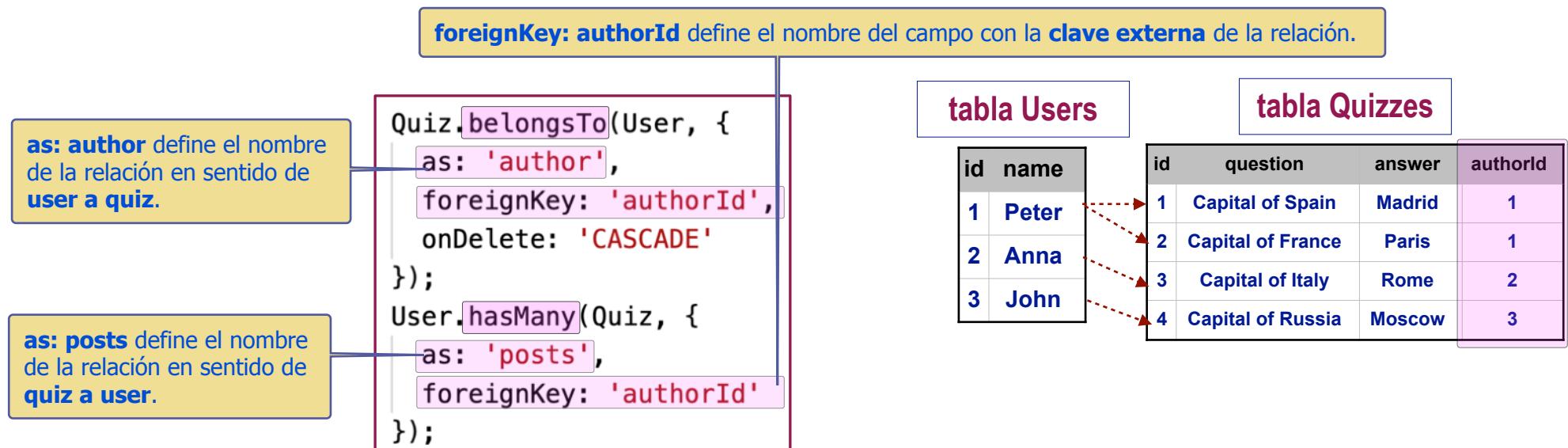
Relación autor con alias

◆ La relación y la clave externa pueden tener **alias**

- Los **alias** se definen en las **opciones** de belongsTo(..) y B.hasMany(..)
 - ◆ Los alias son necesarios para diferenciar relaciones entre los mismos modelos
 - <https://sequelize.org/master/manual/assocs.html#one-to-many-relationships>

◆ Las opciones definen los siguientes alias

- "as:" define el nombre de las relaciones **author** y **posts**, en vez de **User** y **Quiz** (modelos)
- "foreignKey:" define el nuevo nombre **authorId** de la **clave externa** (en vez de **userId**)



Borrado en Cascada

◆ Al borrar o actualizar registros hay que decidir que se hace con sus relacionados

- SQL permite diversas estrategias
 - ◆ 'CASCADE': borrar o actualizar también los elementos relacionados
 - ◆ 'SET NULL': configurar como nulos los identificadores de la relación
 - ◆

▪ Doc:

- ◆ <https://sequelize.org/master/manual/assocs.html#one-to-many-relationships>
- ◆ <https://sequelize.org/master/class/lib/associations/base.js~Association.html>

◆ En la relación definida usamos la estrategia 'CASCADE' al borrar un autor

- "onDelete: CASCADE" indica que al borrar el usuario deben borrarse además sus quizzes

onDelete: 'CASCADE'
configura la relación para que la eliminación de un usuario elimine también todos sus quizzes.

```
Quiz.belongsTo(User, {  
  as: 'author',  
  foreignKey: 'authorId',  
  onDelete: 'CASCADE'  
});  
UserhasMany(Quiz, {  
  as: 'posts',  
  foreignKey: 'authorId'  
});
```

tabla Users

	id	name
1	Peter	
2	Anna	
3	John	

tabla Quizzes

	id	question	answer	authorId
1	Capital of Spain	Madrid	1	
2	Capital of France	Paris	1	
3	Capital of Italy	Rome	2	
4	Capital of Russia	Moscow	3	

Carga ansiosa (eager)



Relación posts-author

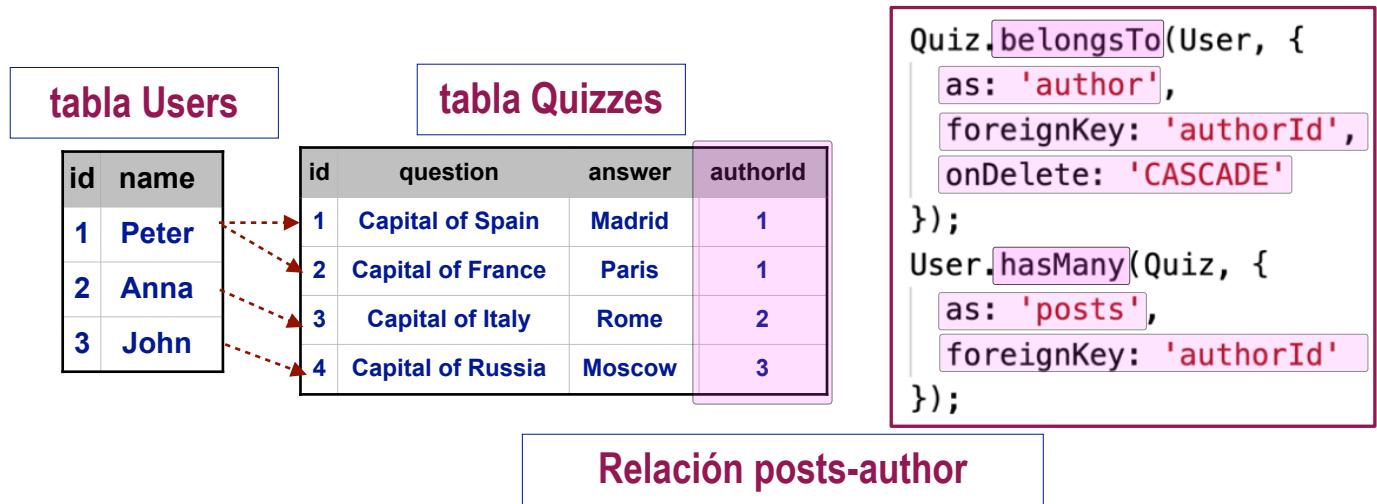
◆ Carga ansiosa

- Carga de los elementos del **modelo** y de los **relacionados** en una misma query mas compleja
 - ◆ La propiedad **include** indica los **elementos relacionados** a cargar
 - Por ejemplo: `include:[User]` (sin alias) o `include:[{ model: User, as: 'author'}]`
- <https://sequelize.org/master/manual/assocs.html#fetching-associations---eager-loading-vs-lazy-loading>

◆ Ejemplos de **carga ansiosa**

- `User.findAll({ include: [{ model: Quiz, as: 'posts'}]})`
 - ◆ Busca todos los usuarios incluyendo en cada uno una propiedad **posts** con sus quizzes
- `User.findOne({ where: {name: 'Peter'}, include: [{ model: Quiz, as: 'posts'}]})`
 - ◆ Busca el usuario 'Peter' incluyendo una propiedad **posts** con sus quizzes
- `Quiz.findOne({ where: {question: 'Capital of Spain'}, include: [{ model: User, as: 'author'}]})`
 - ◆ Busca la pregunta 'Capital of Spain' incluyendo una propiedad **author** con el usuario autor

Carga perezosa (lazy)



◆ Carga perezosa

- Los **elementos relacionados** se gestionan con **métodos invocados en instancias** del modelo
 - ◊ La definición de la relación crea los métodos de instancia que permiten gestionar la relación
- <https://sequelize.org/master/manual/assocs.html#special-methods-mixins-added-to-instances>
- <https://sequelize.org/master/manual/assocs.html#fetching-associations---eager-loading-vs-lazy-loading>

◆ Métodos de instancia creados para posts-author por `hasMany(..)` y `belongsTo(..)`

- `user.getPosts()`: obtiene los quizzes de user ([q1, q2, ...])
- `user.hasPost(quiz)`: indica si quiz pertenece a user
- `user.addPost(quiz)`: user pasa a ser autor de quiz
- `user.removePost(quiz)`: user deja de ser autor de quiz
- `user.createPost({question: .., answer: ..})`: crea un quiz asociado a user
-
- `quiz.setAuthor(user)`: configura user como autor de quiz
- `quiz.getAuthor()`: obtiene el autor de quiz
-

Relación N-N

◆ La relación "favoritos" es una relación N-N

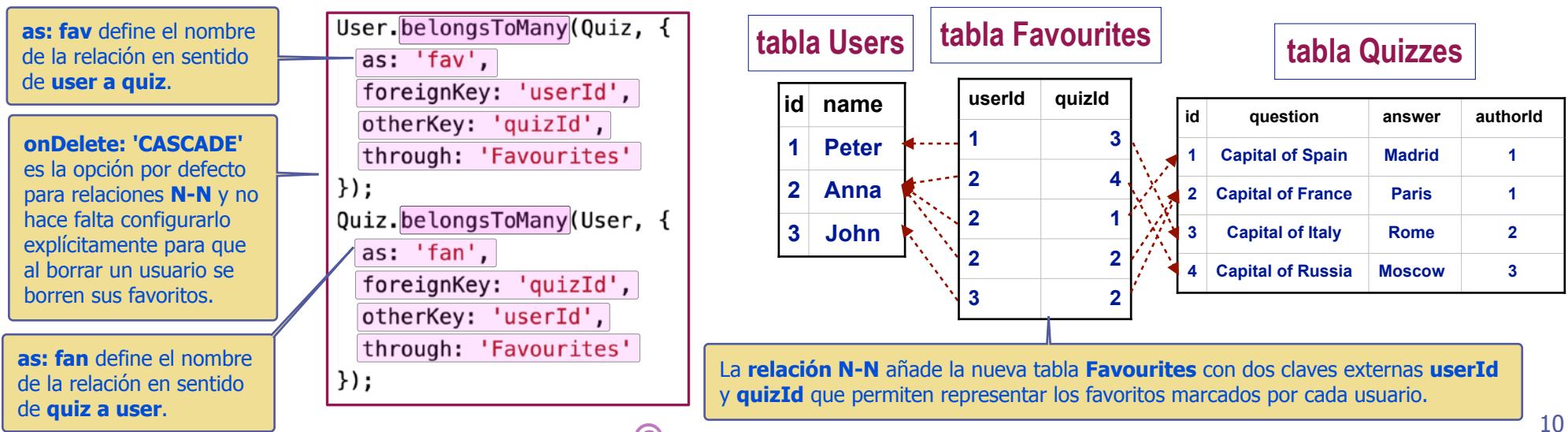
- Relaciona a un usuario con sus quizzes favoritos
- Relaciona a un quiz con sus fans, los usuarios que lo han marcado como favorito

◆ Ambos sentidos de la relación se definen con el método **belongsToMany(..)**

- La gestión de la relación necesita crear la nueva tabla **Favourites**, de tipo join, con dos claves externas
 - <https://sequelize.org/master/manual/assocs.html#many-to-many-relationships>

◆ Las opciones definen los siguientes alias

- "**as:**" define el nombre de las relaciones **fav** y **fan**, en vez de **User** y **Quiz** (modelos)
- "**foreignKey:** y **otherKey**" definen las **claves externas** en la tabla **Favourites**
- "**through: 'Favourites'**" define el nombre en la tabla join **Favourites**
- "**onDelete: CASCADE**" es la opción por defecto y no se explica (eliminar un usuario elimina sus favs)



Alias y relaciones

```
User.belongsToMany(Quiz, {
    as: 'fav',
    foreignKey: 'userId',
    otherKey: 'quizId',
    through: 'Favourites'
});
Quiz.belongsToMany(User, {
    as: 'fan',
    foreignKey: 'quizId',
    otherKey: 'userId',
    through: 'Favourites'
});
```

tabla Favourites

userId	quizId
1	3
2	4
2	1
2	2
3	2

tabla Quizzes

id	question	answer	authorId
1	Capital of Spain	Madrid	1
2	Capital of France	Paris	1
3	Capital of Italy	Rome	2
4	Capital of Russia	Moscow	3

Relación fav-fan

id	name
1	Peter
2	Anna
3	John

Relación posts-author

```
Quiz.belongsTo(User, {
    as: 'author',
    foreignKey: 'authorId',
    onDelete: 'CASCADE'
});
UserhasMany(Quiz, {
    as: 'posts',
    foreignKey: 'authorId'
});
```

tabla Users

◆ Relaciones User-Quiz

- Los alias diferencian ambas relaciones, tanto en cargas ansiosas, como perezosas

◆ Ejemplos de **cargas ansiosas** involucrando varias relaciones

- User.findAll({ include: [{ model: Quiz, as: 'posts' }, { model: Quiz, as: 'fav' }]})**
 - Cargar los usuarios incluyendo en cada uno una propiedad **posts** con sus quizzes o otra **fav** con sus favoritos
- User.findAll({ include: [{ model: Quiz, as: 'fav' }, include: [{ model: Quiz, as: 'author' }]}])**
 - Cargar los usuarios incluyendo en cada uno una propiedad **fav** con sus favoritos y en cada favorito el autor

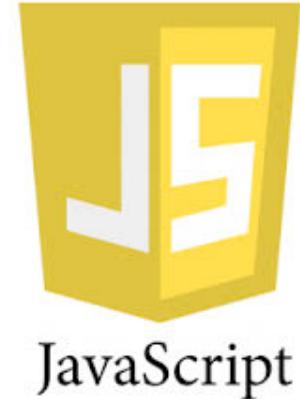
◆ Ejemplos de **cargas perezosas**

- user.addFav(quiz)**

añade quiz como favorito de user

- user.removeFav(quiz)**

elimina quiz de los favoritos de user



Proyecto user-quiz: Tabla Quizzes, Modelo, Relación y Comandos

Juan Quemada, DIT - UPM

Quizzes

README.md (M): doc e instrucciones.

main.js (M): programa principal.

verde: modificado (M).

Download, instalation and usage

This branch can be downloaded, installed and run as follows:

```
$ git clone -b quiz https://github.com/CORE-UPM/user_quiz  
$ cd user_quiz
```

```
$ npm install
```

git clone: descarga un proyecto (directorio) con los ficheros de la captura.

```
$ npm start
```

```
## or 'node main'
```

```
....
```

```
> h
```

Commands (params are requested after):

```
> h ## show help  
>  
> lu | ul | u ## users: list all  
> cu | uc ## user: create  
> ru | ur | r ## user: read (show age)  
> uu ## user: update  
> du | ud ## user: delete  
>  
> lq | ql | q ## quizzes: list all  
> cq | qc ## quiz: create  
> tq | qt | t ## quiz: test (play)  
> uq | qu ## quiz: update  
> dq | qd ## quiz: delete  
>  
> e ## exit & return to shell
```

rosa: añadido.

Name

```
 README.md  
 package.json  
 package-lock.json  
 model.js  
 main.js  
 cmd_user.js  
 cmd_quiz.js
```

cmd_quiz.js: comandos de quiz.

cmd_user.js (M): comandos de user.

model.js (M): Definición de tablas
Users y **Quizzes** y de relación 1:N

id	name
1	Peter
2	Anna
3	John

id	question	answer	authorId
1	Capital of Spain	Madrid	1
2	Capital of France	Paris	1
3	Capital of Italy	Rome	2
4	Capital of Russia	Moscow	3

Paquete en GitHub: https://github.com/CORE-UPM/user_quiz/tree/quiz

```
const { Sequelize, Model, DataTypes } = require('sequelize');

const options = { logging: false};
const sequelize = new Sequelize("sqlite:db.sqlite", options);
```

```
class User extends Model {}
class Quiz extends Model {}
```

Crear clase **Quiz** extendiendo **Model**.

```
User.init(...)
```

Definir tabla **User** con campos **question** y **answer** junto con validaciones y restricciones.

```
Quiz.init(
  { question: {
    type: DataTypes.STRING,
    unique: { msg: "Quiz already exists"}
  },
  answer: DataTypes.STRING
},
{ sequelize }
);
```

Definir relación **1:N** de autor de quiz entre **User** y **Quiz**.

```
Quiz.belongsTo(User, {
  as: 'author',
  foreignKey: 'authorId',
  onDelete: 'CASCADE'
});
UserhasMany(Quiz, {
  as: 'posts',
  foreignKey: 'authorId'
});
```

onDelete:'CASCADE' indica que se deben borrar los quizzes de un user al borrarle.

```
// Initialize the database
```

```
(async () => {
  try {
    await sequelize.sync();
    let count = await User.count();
    let count1 = await Quiz.count();
    if (count==0) {
      let c = await User.bulkCreate([
        { name: 'Peter', age: 22 },
        { name: 'Anna', age: 23 },
        { name: 'John', age: 30 }
      ]);
    }
  }
});
```

```
let q = await Quiz.bulkCreate([
  { question: 'Capital of Spain', answer: 'Madrid', authorId: 1 },
  { question: 'Capital of France', answer: 'Paris', authorId: 1 },
  { question: 'Capital of Italy', answer: 'Rome', authorId: 2 },
  { question: 'Capital of Russia', answer: 'Moscow', authorId: 3 }
])
```

```
process.stdout.write(` DB created (${c.length} users, ${q.length} quizzes)\n`);
return;
} else {
  process.stdout.write(` DB exists (${count} users, ${count1} quizzes)\n`);
}
} catch (err) {
  console.log(` ${err}`);
}
})());
```

```
module.exports = sequelize;
```

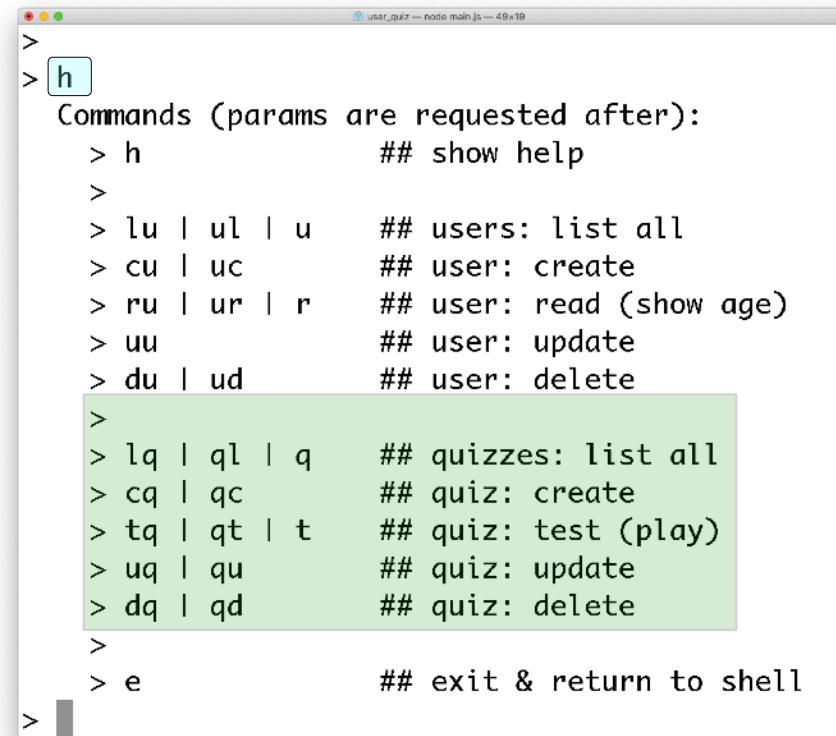
model.js

cmds_user.js: comando help

```
const {User, Quiz} = require("./model.js").models;  
...  
  
exports.help = (rl) =>  
  rl.log(  
    ` Commands (params are requested after):  
    > h          ## show help  
    >  
    > lu | ul | u    ## users: list all  
    > cu | uc      ## user: create  
    > ru | ur | r    ## user: read (show age)  
    > uu          ## user: update  
    > du | ud      ## user: delete  
    >  
    > lq | ql | q    ## quizzes: list all  
    > cq | qc      ## quiz: create  
    > tq | qt | t    ## quiz: test (play)  
    > uq | qu      ## quiz: update  
    > dq | qd      ## quiz: delete  
    >  
    > e          ## exit & return to shell`  
)
```

El comando **help** envía al stream de salida del interfaz **rl** un string con la ayuda. Este describe la lista de comandos y su función.

La función **help** no necesita ser **async** porque no tiene callbacks, ni sincronizaciones.



A screenshot of a terminal window titled "user_quiz — node main.js — 49x19". The window shows the command "h" being typed, followed by the generated help text. The help text lists commands for users and quizzes, each with a description and a parameter placeholder "(params are requested after)". The "quizzes" section is highlighted in green.

```
>  
> h  
Commands (params are requested after):  
  > h          ## show help  
  >  
  > lu | ul | u    ## users: list all  
  > cu | uc      ## user: create  
  > ru | ur | r    ## user: read (show age)  
  > uu          ## user: update  
  > du | ud      ## user: delete  
  >  
  > lq | ql | q    ## quizzes: list all  
  > cq | qc      ## quiz: create  
  > tq | qt | t    ## quiz: test (play)  
  > uq | qu      ## quiz: update  
  > dq | qd      ## quiz: delete  
  >  
  > e          ## exit & return to shell
```

cmds_user.js: comando read

tabla Users

id	name
1	Peter
2	Anna
3	John

tabla Quizzes

id	question	answer	authorId
1	Capital of Spain	Madrid	1
2	Capital of France	Paris	1
3	Capital of Italy	Rome	2
4	Capital of Russia	Moscow	3

```
// Show user's age & quizzes
exports.read = async (rl) => {

  let name = await rl.questionP("Enter name");
  if (!name) throw new Error("Response can't be empty!");

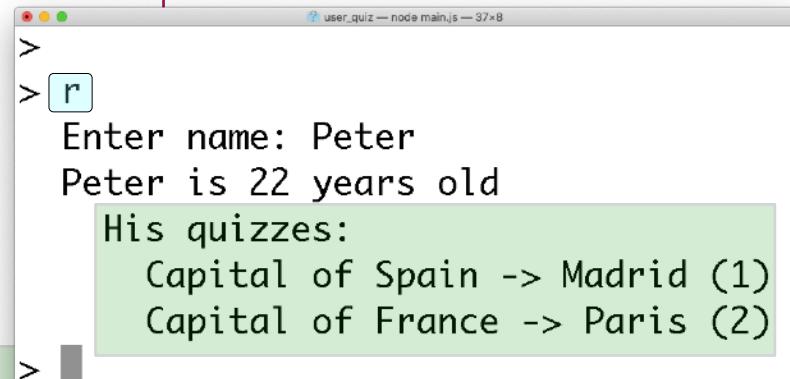
  let user = await User.findOne(
    { where: {name},
      include: [{ model: Quiz, as: 'posts' }]}
  );
  if (!user) throw new Error(`'${name}' is not in DB`);

  rl.log(` ${user.name} is ${user.age} years old`);

  rl.log(` His quizzes:`)
  user.posts.forEach(
    (quiz) => rl.log(` ${quiz.question} -> ${quiz.answer} (${quiz.id})`)
  );
}
```

Petición de **búsqueda ansiosa** por **nombre** a la BBDD, que carga la instancia del usuario, junto con los los quizzes creados por ese autor:
`{ where:{name},
 include: [{model: Quiz, as: 'posts'}]}
}).`

Recordar que **{name}** es equivalente en ES6 a **{name: name}**.



```
user_quiz — node main.js — 37×8
>
> r
Enter name: Peter
Peter is 22 years old
His quizzes:
  Capital of Spain -> Madrid (1)
  Capital of France -> Paris (2)
```

La búsqueda ansiosa del usuario carga en la propiedad **user.posts** un array con todos los quizzes creados por este usuario.

cmds_quiz.js: comando create

create añade un nuevo **quiz** a la tabla **Quizzes**.

```
// Create quiz with <question> and <answer> in the DB  
exports.create = async (rl) => {
```

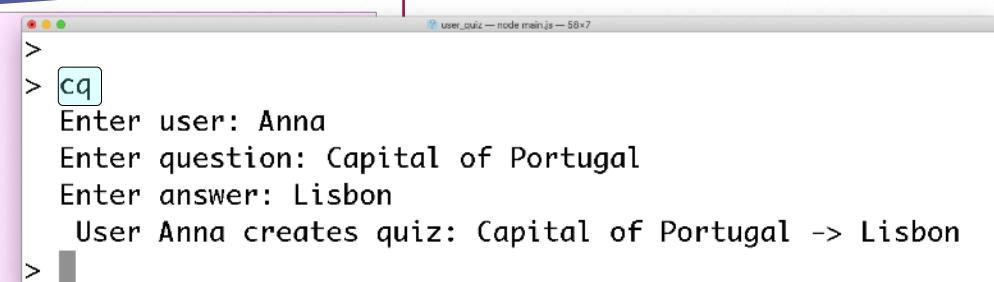
```
let name = await rl.questionP("Enter user");  
let user = await User.findOne({where: {name}});  
if (!user) throw new Error(`User '${name}' doesn't exist!`);  
  
let question = await rl.questionP("Enter question");  
if (!question) throw new Error("Response can't be empty!");  
  
let answer = await rl.questionP("Enter answer");  
if (!answer) throw new Error("Response can't be empty!");
```

Pedir los parámetros necesarios:
user, question y answer.

Petición de crear un nuevo **quiz**:
Quiz.create({question, answer, user.id});

```
await Quiz.create(  
  { question,  
    answer,  
    authorId: user.id  
  }  
);  
rl.log(`  User ${name} creates quiz: ${question} -> ${answer}`);
```

Añade un nuevo **quiz** a la tabla con el autor indicado, e informa a través de la consola.



The screenshot shows a terminal window titled "user_quiz — node main.js — 58x7". The user types "cq" and the program asks for a user name ("Enter user: Anna"). It then asks for a question ("Enter question: Capital of Portugal") and an answer ("Enter answer: Lisbon"). Finally, it logs the message "User Anna creates quiz: Capital of Portugal -> Lisbon" to the console.

cmds_quiz.js: comando list

Importa los modelos **User** y **Quiz** para que los comandos de cmds_quiz.js puedan acceder a las tablas correspondientes.

```
const {User, Quiz} = require("./model.js").models;

// Show all quizzes in DB including <id> and <author>
exports.list = async (rl) => {

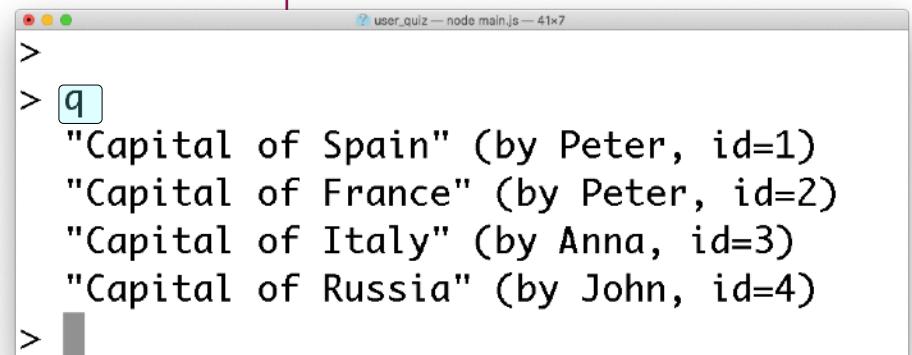
    let quizzes = await Quiz.findAll(
        { include: [
            { model: User,
              as: 'author'
            }
        ]
    );
    quizzes.forEach(
        q => rl.log(`"${q.question}" (by ${q.author.name}, id=${q.id})`)
    );
}
```

list lista el contenido de la tabla **Quizzes** por el stream de salida del interfaz **rl**.

La función **list** necesita ser **async** porque los accesos a la BBDD deben ser sincronizados, porque incluyen esperas en los accesos a la BBDD.

Petición de **búsqueda ansiosa** de todos los **quizzes** de la tabla **Quiz**, incluyendo con cada quiz el autor en la propiedad **author**:

```
Quiz.findAll (
    { include: [
        { model: User,
          as: 'author'
        }
    ]
)
```



The terminal window shows the command `node main.js` being run. The output is:

```
>
> q
"Capital of Spain" (by Peter, id=1)
"Capital of France" (by Peter, id=2)
"Capital of Italy" (by Anna, id=3)
"Capital of Russia" (by John, id=4)
>
```

El iterador `forEach` envía al stream de salida una línea, con `question`, `autor` e `id`, para cada quiz.

cmds_quiz.js: comando test

test presenta la pregunta de un **quiz**, identificado por su **id**, para que el usuario lo conteste.

La función **list** necesita ser **async** porque los accesos a la BBDD deben ser sincronizados, porque incluyen esperas en los accesos a la BBDD.

```
// Test (play) quiz identified by <id>
exports.test = async (rl) => {

  let id = await rl.questionP("Enter quiz Id");
  let quiz = await Quiz.findByPk(Number(id));
  if (!quiz) throw new Error(` Quiz '${id}' is not in DB`);

  let answered = await rl.questionP(quiz.question);

  if (answered.toLowerCase().trim() === quiz.answer.toLowerCase().trim()) {
    rl.log(` The answer "${answered}" is right!`);
  } else {
    rl.log(` The answer "${answered}" is wrong!`);
  }
}
```

Se comprueba si la respuesta es correcta o no.

Petición de búsqueda por id de un quiz:

Quiz.findByPk(Number(id))

```
> t
Enter quiz Id: 2
Capital of France: Paris
The answer "Paris" is right!
```

```
> t
Enter quiz Id: 2
Capital of France: Rome
The answer "Rome" is wrong!
```

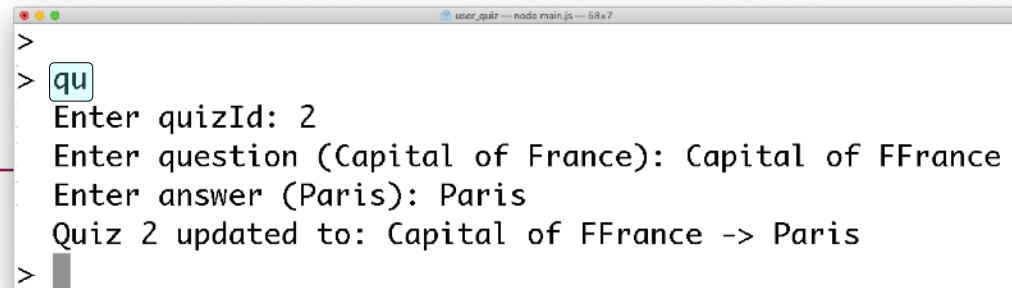
cmds_quiz.js: comando update

update actualiza un **quiz** existente en la tabla **Quizzes**.

```
// Update quiz (identified by <id>) in the DB  
exports.update = async (rl) => {
```

```
let id = await rl.questionP("Enter quizId");  
let quiz = await Quiz.findByPk(Number(id));  
  
let question = await rl.questionP(`Enter question (${quiz.question})`);  
if (!question) throw new Error("Response can't be empty!");  
  
let answer = await rl.questionP(`Enter answer (${quiz.answer})`);  
if (!answer) throw new Error("Response can't be empty!");  
  
quiz.question = question;  
quiz.answer = answer;  
await quiz.save({fields: ["question", "answer"]});  
  
rl.log(` Quiz ${id} updated to: ${question} -> ${answer}`);
```

Actualiza el **quiz** indicado, e informa a través de la consola.



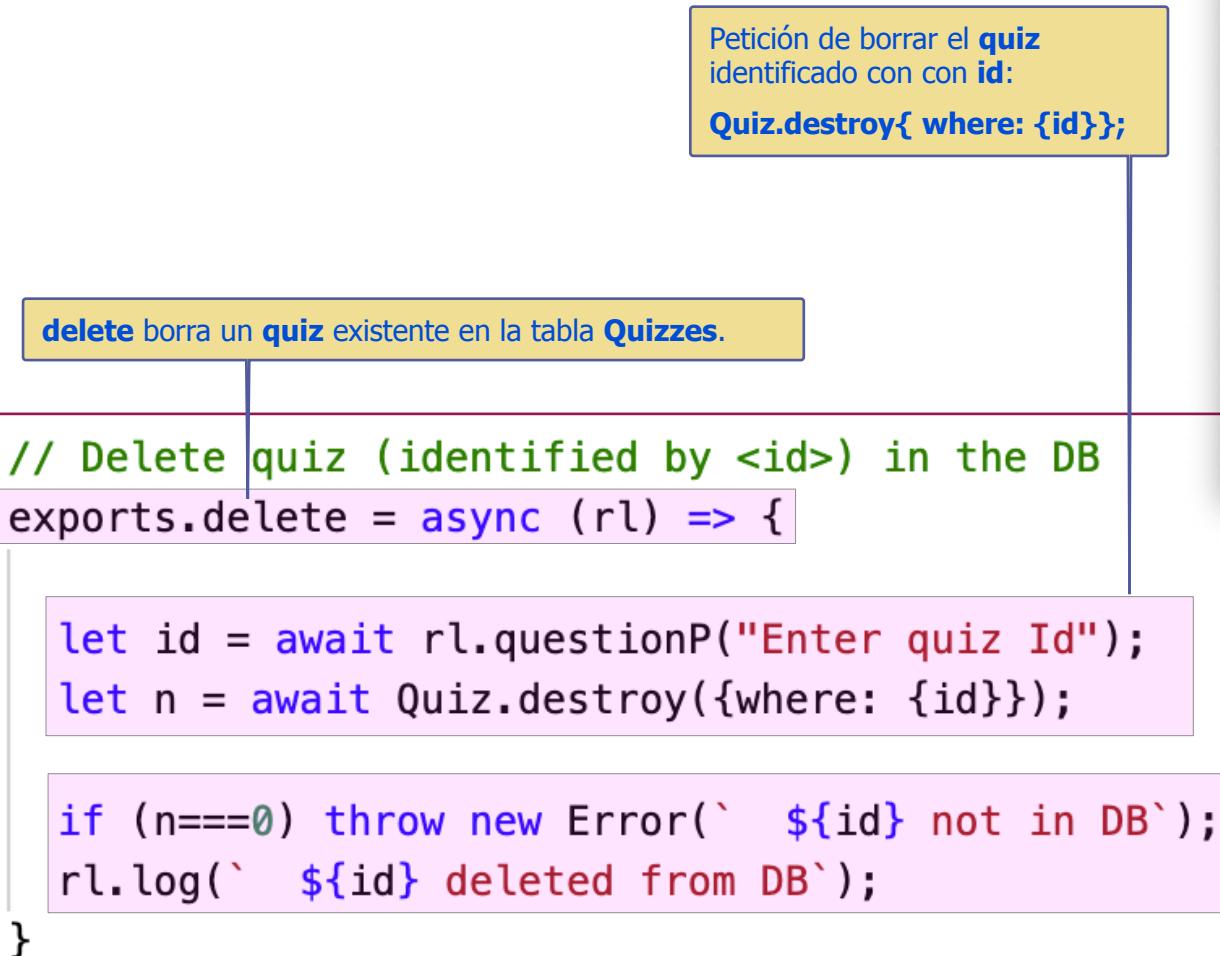
A screenshot of a terminal window titled "user_quiz — node main.js — 68x7". The window shows the following interaction:

```
>  
> qu  
Enter quizId: 2  
Enter question (Capital of France): Capital of FFrance  
Enter answer (Paris): Paris  
Quiz 2 updated to: Capital of FFrance -> Paris  
>
```

Pedir los parámetros necesarios: **quizId**, **question** y **answer**.

Petición de actualizar un **quiz**:
quiz.save({ fields: ["question", "answer"]});

cmds_quiz.js: comando delete



```
>
> q
"Capital of Spain" (by Peter, id=1)
"Capital of France" (by Peter, id=2)
"Capital of Italy" (by Anna, id=3)
"Capital of Russia" (by John, id=4)
>
> dq
Enter quiz Id: 2
2 deleted from DB
>
> q
"Capital of Spain" (by Peter, id=1)
"Capital of Italy" (by Anna, id=3)
"Capital of Russia" (by John, id=4)
>
```

```
>
> dq
Enter quiz Id:
Error:      not in DB
>
```

```

const user = require("./cmds_user.js");
const quiz = require("./cmds_quiz.js");
const readline = require('readline');

const rl = readline.createInterface({ ... });
rl.log = (msg) => console.log(msg); // Add log to rl interface
rl.questionP = function (string) { // Add questionP to rl interface... };
};

rl.prompt();

rl.on('line', async (line) => {
  try{
    let cmd = line.trim()

    if ('' ===cmd) {}
    else if ('h' ===cmd) { user.help(rl);}

    else if (['lu', 'ul', 'u'].includes(cmd)) { await user.list(rl);}
    else if (['cu', 'uc'].includes(cmd)) { await user.create(rl);}
    else if (['ru', 'ur', 'r'].includes(cmd)) { await user.read(rl);}
    else if (['uu'].includes(cmd)) { await user.update(rl);}
    else if (['du', 'ud'].includes(cmd)) { await user.delete(rl);}

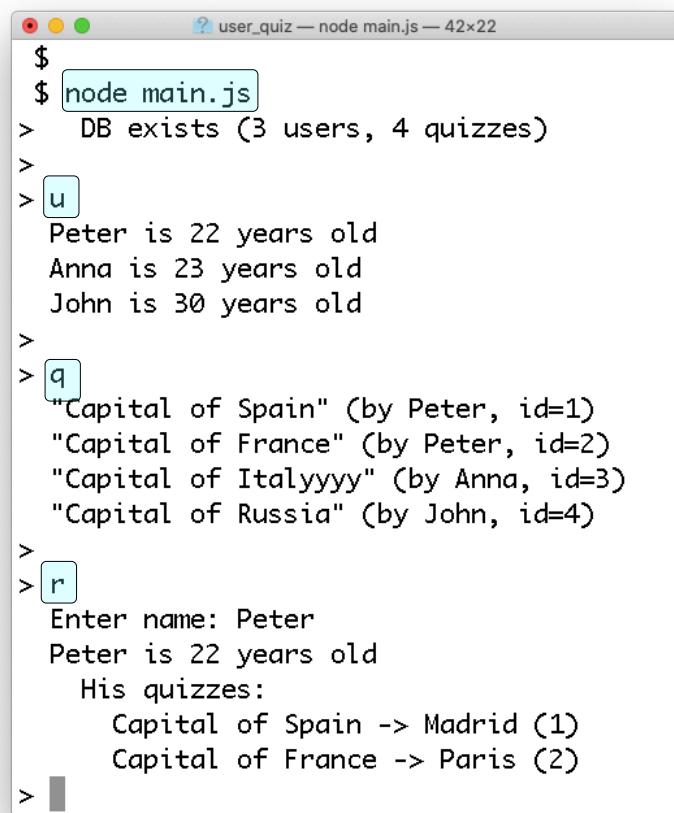
    else if (['lq', 'ql', 'q'].includes(cmd)) { await quiz.list(rl);}
    else if (['cq', 'qc'].includes(cmd)) { await quiz.create(rl);}
    else if (['tq', 'qt', 't'].includes(cmd)) { await quiz.test(rl);}
    else if (['uq', 'qu'].includes(cmd)) { await quiz.update(rl);}
    else if (['dq', 'qd'].includes(cmd)) { await quiz.delete(rl);}

    else if ('e'==cmd) { rl.log('Bye!'); process.exit(0);}
    else { rl.log('UNSUPPORTED COMMAND!'); user.help(rl);};
  } catch (err) { rl.log(` ${err}`);}
  finally { rl.prompt(); }
});

```

Importar modulo:
- **cmds_quiz** comandos del programa.

El Programa Principal: main.js



```

$ node main.js
> DB exists (3 users, 4 quizzes)
>
> u
Peter is 22 years old
Anna is 23 years old
John is 30 years old
>
> q
"Capital of Spain" (by Peter, id=1)
"Capital of France" (by Peter, id=2)
"Capital of Italyyyy" (by Anna, id=3)
"Capital of Russia" (by John, id=4)
>
> r
Enter name: Peter
Peter is 22 years old
His quizzes:
  Capital of Spain -> Madrid (1)
  Capital of France -> Paris (2)
>

```

Al teclear alguno de los comandos se invoca el método asociado, que se ha importado del **módulo cmds_quiz** que implementa los comandos.



Proyecto user-quiz: Relación Favourites, Modelo y Comandos

Juan Quemada, DIT - UPM

Favourites

Download, instalation and usage

This branch can be downloaded, installed and run as follows:

```
$ git clone -b favourite https://github.com/CORE-UPM/user_quiz
$ cd user_quiz
$ npm install
$ npm start ## or 'node main'
.....
> h
Commands (params are requested after):
  > h          ## show help
  >
  > lu | ul | u ## users: list all
  > cu | uc      ## user: create
  > ru | ur | r ## user: read (show age)
  > uu          ## user: update
  > du | ud      ## user: delete
  >
  > lq | ql | q ## quizzes: list all
  > cq | qc      ## quiz: create
  > tq | qt | t ## quiz: test (play)
  > uq | qu      ## quiz: update
  > dq | qd      ## quiz: delete
  >
  > lf | fl | f ## favourites: list all
  > cf | fc      ## favourite: create
  > df | fd      ## favourite: delete
  >
  > e          ## exit & return to shell
```

README.md (M): doc e instrucciones.

main.js (M): programa principal.

rosa: añadido.

git clone: descarga un proyecto (directorio) con los ficheros de la captura.

cmds_fav.js: comandos de favourites.

verde: modificado (M).

Name

READMD.md
package.json
package-lock.json
model.js
main.js
cmds_user.js
cmds_quiz.js
cmds_favs.js

cmds_user.js (M): comandos de user.

model.js (M): Definición de relación Favourite entre Users y Quizzes.

id	name
1	Peter
2	Anna
3	John

tabla Users

userId	quizId
1	3
2	4
2	1
2	2
3	2

tabla Favourites

id	question	answer	authorId
1	Capital of Spain	Madrid	1
2	Capital of France	Paris	1
3	Capital of Italy	Rome	2
4	Capital of Russia	Moscow	3

tabla Quizzes

```
$ node main
> DB created: (3 users, 4 quizzes, 5 favs)
> f
Quiz 3 (Capital of Italy) favourite of Peter
Quiz 1 (Capital of Spain) favourite of Anna
Quiz 2 (Capital of France) favourite of Anna
Quiz 4 (Capital of Russia) favourite of Anna
Quiz 2 (Capital of France) favourite of John
>
> r
Enter name: Anna
Anna is 23 years old
Quizzes:
  Capital of Italy -> Rome (3)
Favourite quizzes:
  Capital of Spain -> Madrid (Peter, 1)
  Capital of France -> Paris (Peter, 2)
  Capital of Russia -> Moscow (John, 4)
>
> r
Enter name: Peter
Peter is 22 years old
Quizzes:
  Capital of Spain -> Madrid (1)
  Capital of France -> Paris (2)
Favourite quizzes:
  Capital of Italy -> Rome (Anna, 3)
```

Paquete en GitHub: https://github.com/CORE-UPM/user_quiz/tree/favourite

model.js

```
const { Sequelize, Model, DataTypes } = require('sequelize');  
  
const options = { logging: false};  
const sequelize = new Sequelize("sqlite:db.sqlite", options);
```

```
class User extends Model {}  
class Quiz extends Model {}
```

```
User.init( ...  
);
```

Definir la relación **N:N Favourite** de **User** a **Quiz**. Esta relación necesita la tabla no explícita **Favourite**, que se define con **through: 'Favourite'**.

```
Quiz.init( ...  
);
```

```
Quiz.belongsTo(User, { ...  
});  
UserhasMany(Quiz, { ...  
});
```

```
// N:N default is -> onDelete: 'CASCADE'
```

```
User.belongsToMany(Quiz, {  
  as: 'fav',  
  foreignKey: 'userId',  
  otherKey: 'quizId',  
  through: 'Favourites'  
});
```

Las relaciones N:N tienen por defecto **onDelete: 'CASCADE'**, por lo que no hace falta configurarla.

```
Quiz.belongsToMany(User, {  
  as: 'fan',  
  foreignKey: 'quizId',  
  otherKey: 'userId',  
  through: 'Favourites'  
});
```

```
// Initialize the database  
(async () => {  
  try {  
    await sequelize.sync();  
    let count = await User.count();  
    let count1 = await Quiz.count();  
    let count2 = await sequelize.models.Favourites.count();  
    if (count==0) {  
      let c = await User.bulkCreate([  
        { name: 'Peter', age: 22},  
        { name: 'Anna', age: 23},  
        { name: 'John', age: 30}  
      ]);  
      let q = await Quiz.bulkCreate([  
        { question: 'Capital of Spain', answer: 'Madrid', authorId: 1},  
        { question: 'Capital of France', answer: 'Paris', authorId: 1},  
        { question: 'Capital of Italy', answer: 'Rome', authorId: 2},  
        { question: 'Capital of Russia', answer: 'Moscow', authorId: 3}  
      ]);  
      let f = await sequelize.models.Favourites.bulkCreate([  
        { userId: 1, quizId: 3},  
        { userId: 2, quizId: 4},  
        { userId: 2, quizId: 1},  
        { userId: 2, quizId: 2},  
        { userId: 3, quizId: 2}  
      ]);  
      process.stdout.write(` DB created: (${c.length} users, ${q.length} quizzes, ${f.length} favs)\n`);  
    } else {  
      process.stdout.write(` DB exists: (${count} users, ${count1} quizzes, ${count2} favs)\n`);  
    }  
  } catch (err) {  
    console.log(` ${err}`);  
  }  
})();  
  
module.exports = sequelize;
```

Se cuentan los favoritos de la tabla **Favourites**.

Se crean estos 5 favoritos.

Se indica también el número de favoritos

Se indica también el número de favoritos

```

const {User, Quiz} = require("./model.js").models;

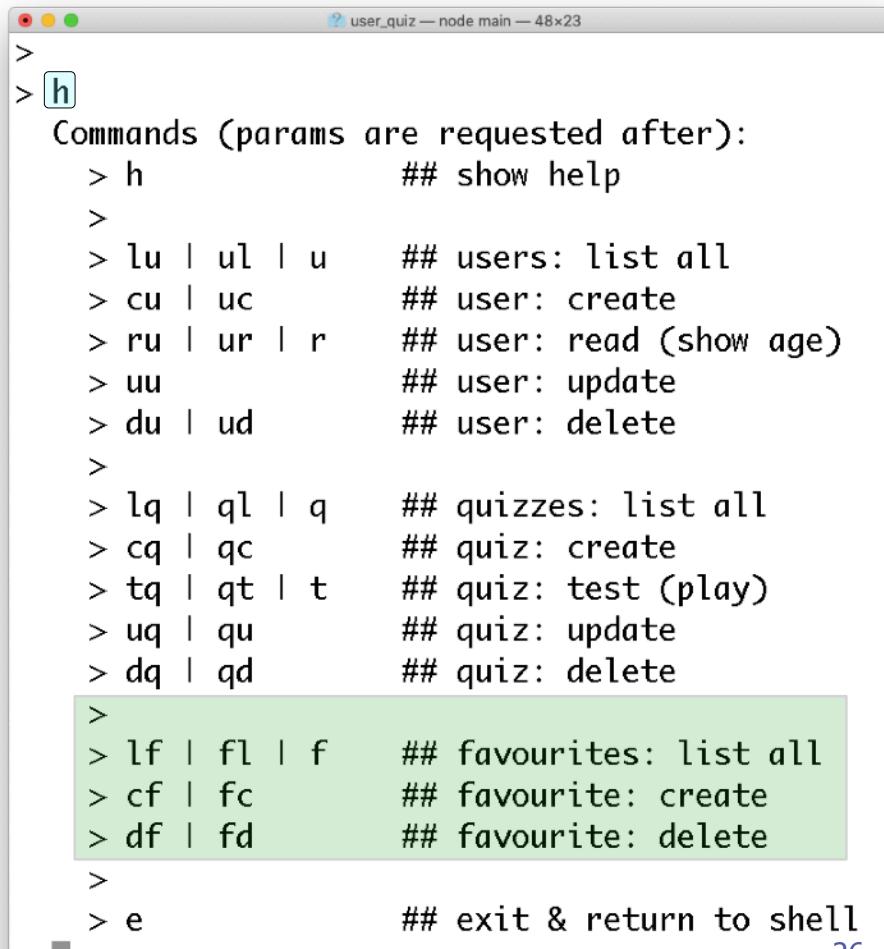
exports.help = (rl) =>
  rl.log(
    ` Commands (params are requested after):
      > h          ## show help
      >
      > lu | ul | u ## users: list all
      > cu | uc     ## user: create
      > ru | ur | r ## user: read (show age)
      > uu          ## user: update
      > du | ud     ## user: delete
      >
      > lq | ql | q ## quizzes: list all
      > cq | qc     ## quiz: create
      > tq | qt | t ## quiz: test (play)
      > uq | qu     ## quiz: update
      > dq | qd     ## quiz: delete
      >
      > lf | fl | f ## favourites: list all
      > cf | fc     ## favourite: create
      > df | fd     ## favourite: delete
      >
      > e          ## exit & return to shell` )
  )

```

cmds_user.js: comando help

El comando **help** envía al stream de salida del interfaz **rl** un string con la ayuda. Este describe la lista de comandos y su función.

La función **help** no necesita ser **async** porque no tiene promesas.



```

>
> [h]
Commands (params are requested after):
  > h          ## show help
  >
  > lu | ul | u ## users: list all
  > cu | uc     ## user: create
  > ru | ur | r ## user: read (show age)
  > uu          ## user: update
  > du | ud     ## user: delete
  >
  > lq | ql | q ## quizzes: list all
  > cq | qc     ## quiz: create
  > tq | qt | t ## quiz: test (play)
  > uq | qu     ## quiz: update
  > dq | qd     ## quiz: delete
  >
  > lf | fl | f ## favourites: list all
  > cf | fc     ## favourite: create
  > df | fd     ## favourite: delete
  >
  > e          ## exit & return to shell

```

cmds_user.js: comando read

```
// Show user's age, quizzes & favourites
exports.read = async (rl) => {

  let name = await rl.question("Enter name");
  if (!name) throw new Error("Response can't be empty!");

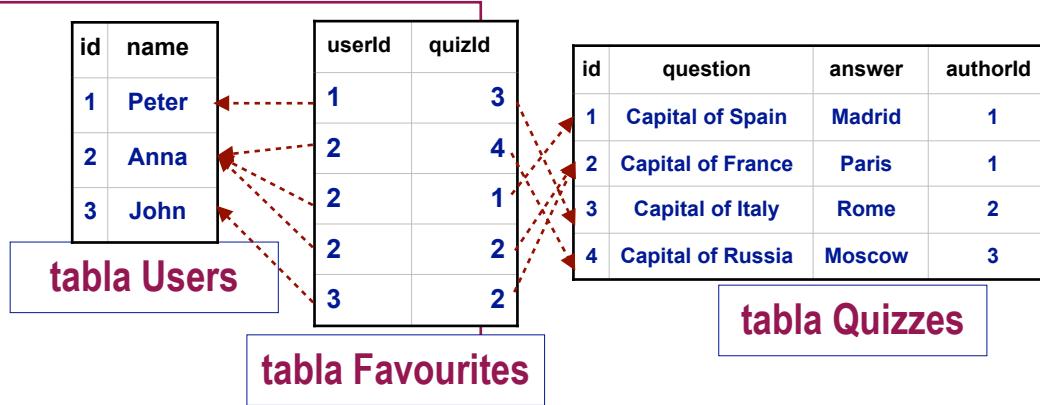
  let user = await User.findOne({
    where: {name},
    include: [
      { model: Quiz, as: 'posts'},
      { model: Quiz, as: 'fav',
        include: [{ model: User, as: 'author'}]
      }
    ]
  });

  if (!user) throw new Error(`'${name}' is not in DB`);

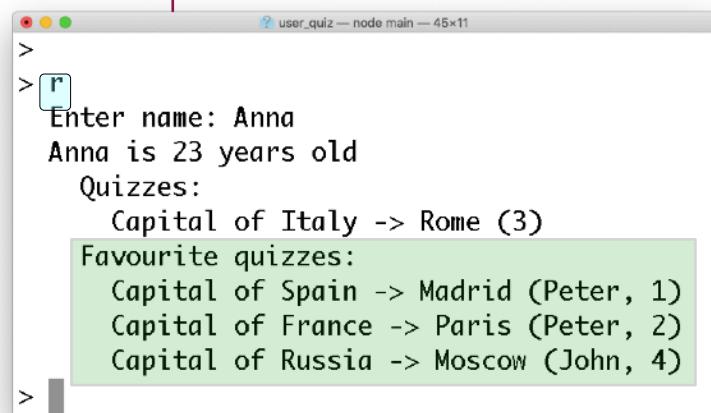
  rl.log(` ${user.name} is ${user.age} years old`);

  rl.log(` Quizzes:`)
  user.posts.forEach(
    (quiz) => rl.log(` ${quiz.question} -> ${quiz.answer} (${quiz.id})`)
  );

  rl.log(` Favourite quizzes:`)
  user.fav.forEach( (quiz) =>
    rl.log(` ${quiz.question} -> ${quiz.answer} (${quiz.author.name}, ${quiz.id})`)
  );
}
```



Búsqueda ansiosa por **nombre**, que carga la instancia del usuario, junto con sus quizzes y favoritos (incluyendo autor).



La búsqueda ansiosa del usuario carga en la propiedad **user.fav** un array con todos los quizzes creados por este usuario.

cmds_favs.js: comando list

Importa los modelos **User** y **Quiz** para que los comandos de cmd_quiz.js puedan acceder a las tablas correspondientes.

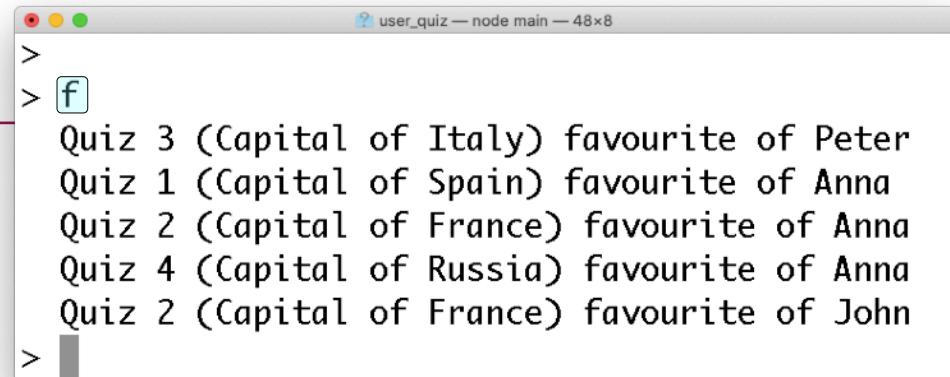
```
const {User, Quiz} = require("./model.js").models;  
  
// Show all favourites in DB including <user> and <id>  
exports.list = async (rl) => {
```

```
let users = await User.findAll({  
  include: [{model: Quiz, as: 'fav'}]});  
  
users.forEach(u => {  
  if (u.fav.length !== 0) {  
    u.fav.forEach(q =>  
      rl.log(` Quiz ${q.id} (${q.question}) favourite of ${u.name}`)  
    )  
  }  
});
```

El iterador forEach envía al stream de salida una línea, con id (question) y usuario.

list lista el contenido de la tabla **Favourites** por el stream de salida del interfaz **rl**.

La función **list** necesita ser **async** porque los accesos a la BBDD deben ser sincronizados, porque incluyen esperas en los accesos a la BBDD.



```
>  
> f  
Quiz 3 (Capital of Italy) favourite of Peter  
Quiz 1 (Capital of Spain) favourite of Anna  
Quiz 2 (Capital of France) favourite of Anna  
Quiz 4 (Capital of Russia) favourite of Anna  
Quiz 2 (Capital of France) favourite of John  
>
```

Como favoritos no es una tabla explícita, debemos buscar todos los usuarios incluyendo sus quizzes favoritos con **búsqueda ansiosa**.

cmds_favs.js: comando create

create añade un nuevo **favorito** a la tabla **Favourites**.

// Create favourite in the DB

```
exports.create = async (rl) => {
```

```
let name = await rl.question("Enter name");
let id = await rl.question("Enter quiz id");
if (!name || !id) throw new Error("Response can't be empty!");
```

```
let user = await User.findOne({where: {name}});
if (!user) throw new Error(` ${name} not in DB`);
```

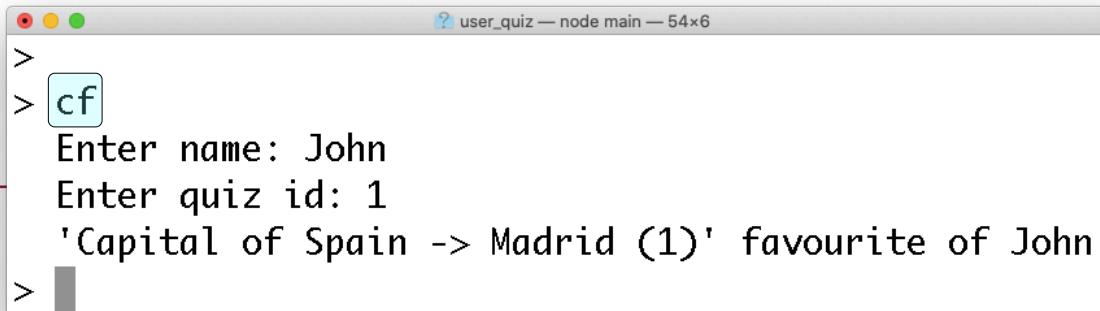
```
let quiz = await Quiz.findByPk(Number(id));
if (!quiz) throw new Error(` Quiz not in DB`);
```

```
await user.addFav(quiz)
```

Petición perezosa de crear un nuevo **favorito**:
`user.addFav(quiz);`

```
rl.log(` ${quiz.question} -> ${quiz.answer} (${id})' favourite of ${name}`);
```

```
}
```



```
>
> cf
Enter name: John
Enter quiz id: 1
'Capital of Spain -> Madrid (1)' favourite of John
>
```

Pedir los parámetros necesarios:
user, **question** y **answer**.

cmds_favs.js: comando delete

delete borra el **favorito** indicado en la tabla **Favourites**.

```
// Delete favourite in the DB
```

```
exports.delete = async (rl) => {
```

```
let name = await rl.question("Enter name");
let id = await rl.question("Enter quiz id");
if (!name || !id) throw new Error("Response can't be empty!");
```

```
let user = await User.findOne({where: {name}});
if (!user) throw new Error(` ${name} not in DB`);
```

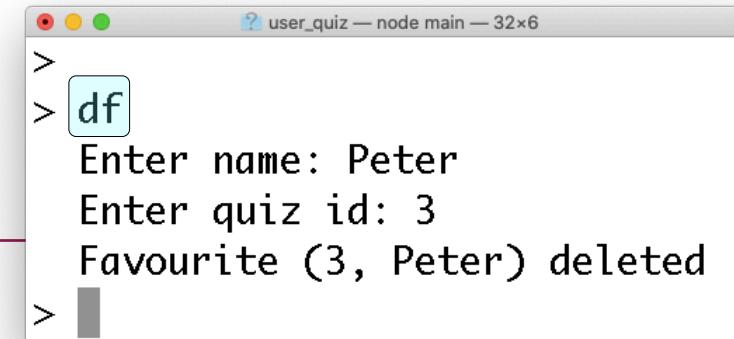
```
let quiz = await Quiz.findByPk(Number(id));
if (!quiz) throw new Error(` Quiz not in DB`);
```

```
await user.removeFav(quiz);
```

Borrado perezoso del **quiz** con identificador **id** favorito de **user**:
user.removeFav(quiz);

```
rl.log(` Favourite (${id}, ${name}) deleted`);
```

Si el **quiz** se ha borrado, informa a través de la consola.



```
> df
Enter name: Peter
Enter quiz id: 3
Favourite (3, Peter) deleted
>
```

Pedir parámetros necesarios:
name e **id (quiz)**.

obtener instancias de user y quiz.

```
}
```

```

const user = require("./cmds_user.js");
const quiz = require("./cmds_quiz.js");
const favs = require("./cmds_favs.js");
const readline = require('readline');

const rl = readline.createInterface({
});

rl.log = (msg) => console.log(msg); // Add log to rl interface
rl.questionP = function (string) { // Add questionP to rl interface...
};

rl.prompt();

rl.on('line', async (line) => {
  try{
    let cmd = line.trim()

    if ('' ===cmd) {}
    else if ('h' ===cmd) { user.help(rl);}

    else if (['lu', 'ul', 'u'].includes(cmd)) { await user.list(rl);}
    else if (['cu', 'uc'].includes(cmd)) { await user.create(rl);}
    else if (['ru', 'ur', 'r'].includes(cmd)) { await user.read(rl);}
    else if (['uu'].includes(cmd)) { await user.update(rl);}
    else if (['du', 'ud'].includes(cmd)) { await user.delete(rl);}

    else if (['lq', 'ql', 'q'].includes(cmd)) { await quiz.list(rl);}
    else if (['cq', 'qc'].includes(cmd)) { await quiz.create(rl);}
    else if (['tq', 'qt', 't'].includes(cmd)) { await quiz.test(rl);}
    else if (['uq', 'qu'].includes(cmd)) { await quiz.update(rl);}
    else if (['dq', 'qd'].includes(cmd)) { await quiz.delete(rl);}

    else if (['lf', 'fl', 'f'].includes(cmd)) { await favs.list(rl);}
    else if (['cf', 'fc'].includes(cmd)) { await favs.create(rl);}
    else if (['df', 'fd'].includes(cmd)) { await favs.delete(rl);}

    else if ('e'==cmd) { rl.log('Bye!'); process.exit(0);}
    else { rl.log('UNSUPPORTED COMMAND!'); user.help(rl);
    };
  } catch (err) { rl.log(` ${err}`);}
  finally { rl.prompt(); }
});

```

Importar modulo:
- **cmds_favs**: comandos del programa.

El Programa Principal: main.js

```

$ node main
> DB created: (3 users, 4 quizzes, 5 favs)
>
> f
Quiz 3 (Capital of Italy) favourite of Peter
Quiz 1 (Capital of Spain) favourite of Anna
Quiz 2 (Capital of France) favourite of Anna
Quiz 4 (Capital of Russia) favourite of Anna
Quiz 2 (Capital of France) favourite of John
>
> r
Enter name: Anna
Anna is 23 years old
Quizzes:
  Capital of Italy -> Rome (3)
Favourite quizzes:
  Capital of Spain -> Madrid (Peter, 1)
  Capital of France -> Paris (Peter, 2)
  Capital of Russia -> Moscow (John, 4)
>
> r
Enter name: Peter
Peter is 22 years old
Quizzes:
  Capital of Spain -> Madrid (1)
  Capital of France -> Paris (2)
Favourite quizzes:
  Capital of Italy -> Rome (Anna, 3)
>

```

Al teclear alguno de los comandos se invoca el método asociado, que se ha importado del **módulo cmds_favs** que implementa los comandos.



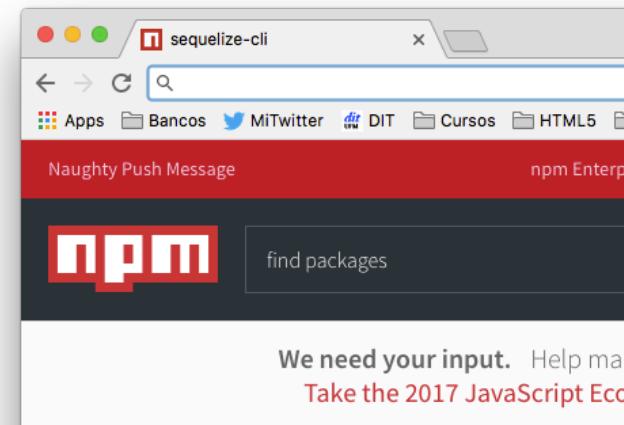
Proyecto user-quiz: Sequelize-cli, migraciones y seeders

Juan Quemada, DIT - UPM

Migración de la BBDD con sequelize-cli

- ◆ La **estructura** y los **modelos** de datos de una base de datos **sufren cambios** durante su vida operativa
 - Estos cambios deben realizarse minimizando las perdidas de datos y preservando su significado lo mejor posible
- ◆ Las técnicas de **migración** de una BBDD permiten una gestión **incremental** y **reversible** de cambios en los modelos o esquemas de datos utilizados
 - https://en.wikipedia.org/wiki/Schema_migration
- ◆ **sequelize-cli** (Sequelize Command Line Interpreter)
 - Herramienta de sequelize para realizar migraciones
 - <https://www.npmjs.com/package/sequelize-cli>
 - Instrucciones para realizar migraciones con sequelize:
 - <https://sequelize.org/master/manual/migrations.html>

```
..$ ## instalar sequelize-cli en directorio node_modules con npm  
..$  
..$ npm install sequelize-cli@5.5.0      ## instala version 5.5.0  
..$
```



sequelize-cli public
The Sequelize Command Line Interface (CLI)

sequelize-cli

```
$  
$ npx sequelize --help
```

npx ejecuta comandos de S.O. y paquetes npm. Por ej.

..\$ npx sequelize --help

invoca el comando instalado con **npm** que también se invoca con

..\$ node_modules/.bin/sequelize --help

Sequelize CLI [Node: 12.14.0, CLI: 5.5.1, ORM: 5.21.4]

sequelize [command]

Commands:

sequelize db:migrate	Run pending migrations
sequelize db:migrate:schema:timestamps:add	Update migration table to have timestamps
sequelize db:migrate:status	List the status of all migrations
sequelize db:migrate:undo	Reverts a migration
sequelize db:migrate:undo:all	Revert all migrations ran
sequelize db:seed	Run specified seeder
sequelize db:seed:undo	Deletes data from the database
sequelize db:seed:all	Run every seeder
sequelize db:seed:undo:all	Deletes data from the database
sequelize db:create	Create database specified by configuration
sequelize db:drop	Drop database specified by configuration
sequelize init	Initializes project
sequelize init:config	Initializes configuration
sequelize init:migrations	Initializes migrations
sequelize init:models	Initializes models
sequelize init:seeders	Initializes seeders
sequelize migration:generate	Generates a new migration file [aliases: migration:create]
sequelize model:generate	Generates a model and its migration [aliases: model:create]
sequelize seed:generate	Generates a new seed file [aliases: seed:create]

Options:

--help Show help
--version Show version number



[boolean]
[boolean]

\$

Migration

Download, instalation and usage

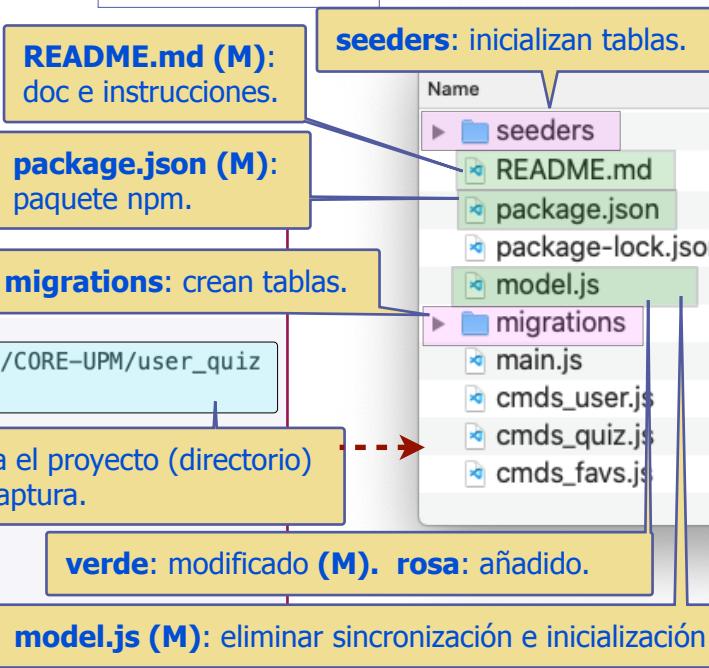
This branch can be downloaded, installed and run as:

```
$ git clone -b migration https://github.com/CORE-UPM/user_quiz
$ cd user_quiz
$ npm install
$ npm run migrate
$ npm run seed
$ npm start ## or 'node main'
....
```

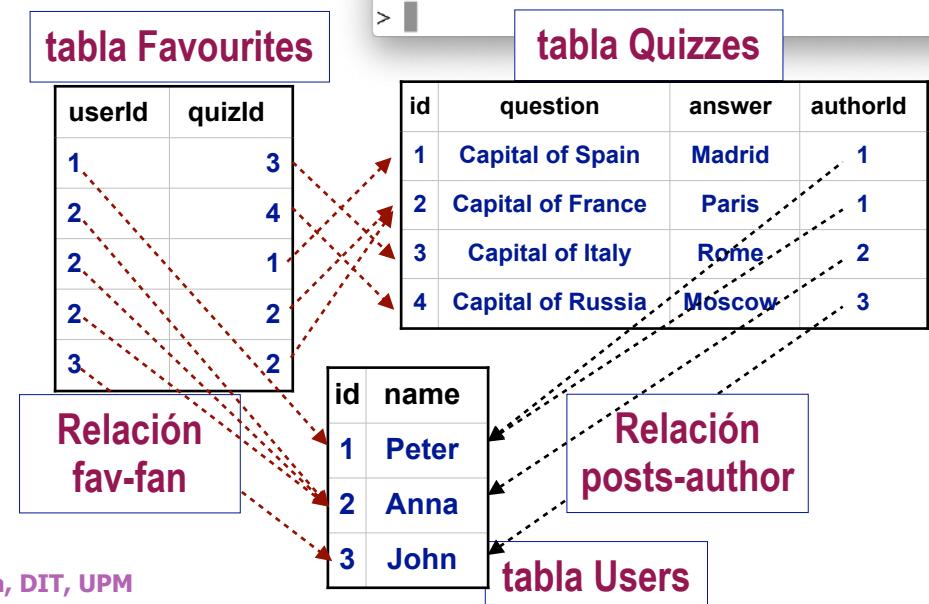
git clone: descarga el proyecto (directorio) con los ficheros de la captura.

verde: modificado (M). **rosa:** añadido.

Paquete en GitHub: https://github.com/CORE-UPM/user_quiz/tree/migration



```
$ node main
> u
Peter is 22 years old
Anna is 23 years old
John is 30 years old
>
> q
"Capital of Spain" (by Peter, id=1)
"Capital of France" (by Peter, id=2)
"Capital of Italy" (by Anna, id=3)
"Capital of Russia" (by John, id=4)
>
> f
Quiz 3 (Capital of Italy) favourite of Peter
Quiz 1 (Capital of Spain) favourite of Anna
Quiz 2 (Capital of France) favourite of Anna
Quiz 4 (Capital of Russia) favourite of Anna
Quiz 2 (Capital of France) favourite of John
>
> r
Enter name: John
John is 30 years old
Quizzes:
    Capital of Russia -> Moscow (4)
Favourite quizzes:
    Capital of France -> Paris (Peter, 2)
```



```
{
  "name": "user_quiz",
  "version": "1.0.0",
  "description": "Educational node.js project to start with DBs, sequelize & sockets",
  "main": "main.js",
  "scripts": {
    "migrate": "sequelize db:migrate --url sqlite://$(pwd)/db.sqlite",
    "seed": "sequelize db:seed:all --url sqlite://$(pwd)/db.sqlite",
    "migrate_win": "sequelize db:migrate --url sqlite://%cd%/db.sqlite",
    "seed_win": "sequelize db:seed:all --url sqlite://%cd%/db.sqlite",
    "start": "node main.js"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/CORE-UPM/user_quiz"
  },
  "author": "Juan Quemada",
  "license": "GNUnv3 – General Public License version 3",
  "bugs": {
    "url": "https://github.com/CORE-UPM/user_quiz/issues"
  },
  "homepage": "https://github.com/CORE-UPM/user_quiz#readme",
  "dependencies": {
    "sequelize": "^5.21.3",
    "sequelize-cli": "^5.5.0",
    "sqlite3": "^4.0.9"
  }
}
```

Se añaden estos scripts a **package.json** para **UNIX/Linux**

\$ npm run migrate ## Ejecuta las migraciones pendientes en orden de creación
\$ npm run seed ## Ejecuta las seeds pendientes en orden de creación

Se añaden estos scripts a **package.json** para **Windows**

\$ npm run migrate ## Ejecuta las migraciones pendientes
en orden de creación
\$ npm run seed ## Ejecuta las seeds pendientes
en orden de creación

package.json

La dependencia del paquetes **sequelize-cli** se añaden a package.json automáticamente al instalarlo:

\$
\$ npm install sequelize-cli@5.5.0
\$

model.js

```
const { Sequelize, Model, DataTypes } = require('sequelize');
...
const options = { logging: false};
const sequelize = new Sequelize("sqlite:db.sqlite", options);

class User extends Model {}
class Quiz extends Model {}

User.init(...);
Quiz.init(...);

Quiz.belongsTo(User, {...});
UserhasMany(Quiz, {...});

// N:N default is -> onDelete: 'CASCADE'
User.belongsToMany(Quiz, {
  as: 'fav',
  foreignKey: 'userId',
  otherKey: 'quizId',
  through: 'Favourites'
});
Quiz.belongsToMany(User, {
  as: 'fan',
  foreignKey: 'quizId',
  otherKey: 'userId',
  through: 'Favourites'
});

```

```
// Initialize the database
(async () => {
  try {
    await sequelize.sync();
    let count = await User.count();
    let count1 = await Quiz.count();
    let count2 = await sequelize.models.Favourites.count();
    if (count==0) {
      let c = await User.bulkCreate([
        { name: 'Peter', age: 22 },
        { name: 'Anna', age: 23 },
        { name: 'John', age: 30 }
      ]);
      let q = await Quiz.bulkCreate([
        { question: 'Capital of Spain', answer: 'Madrid', authorId: 1 },
        { question: 'Capital of France', answer: 'Paris', authorId: 1 },
        { question: 'Capital of Italy', answer: 'Rome', authorId: 2 },
        { question: 'Capital of Russia', answer: 'Moscow', authorId: 3 }
      ]);
      let f = await sequelize.models.Favourites.bulkCreate([
        { userId: 1, quizId: 3 },
        { userId: 2, quizId: 4 },
        { userId: 2, quizId: 1 },
        { userId: 2, quizId: 2 },
        { userId: 3, quizId: 2 }
      ]);
      process.stdout.write(` DB created: (${c.length} users, ${q.length} quizzes, ${f.length} favs)\n`);
    } else {
      process.stdout.write(` DB exists: (${count} users, ${count1} quizzes, ${count2} favs)\n`);
    }
  } catch (err) {
    console.log(` ${err}`);
  }
})();

module.exports = sequelize;
```

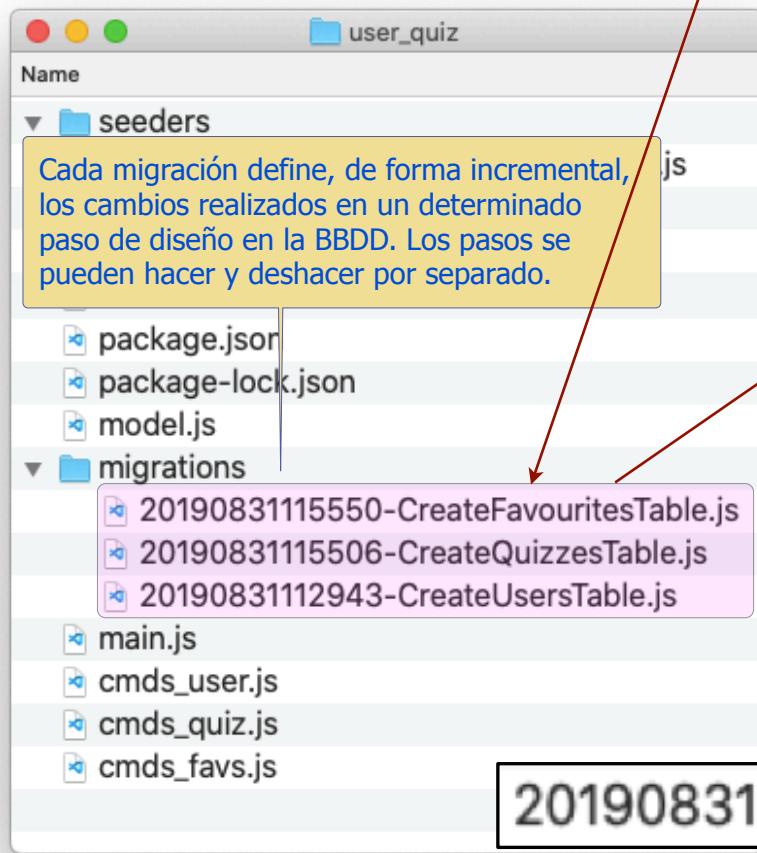
Crear esqueletos de ficheros de migración

Crear el directorio **migrations** y el esqueleto de las migraciones de las tablas users, quizzes y favourites con sequelize.

```
..$  
..$ npx sequelize migration:create --name CreateUsersTable ←  
..$  
..$ npx sequelize migration:create --name CreateQuizzesTable ←  
..$  
..$ npx sequelize migration:create --name CreateFavouritesTable ←
```

Sequelize crea el esqueleto de los ficheros de migración con un nombre que incluye la fecha de creación. Esta fecha es importante, porque las migraciones se aplican siguiendo el orden de creación.

El esqueleto incluye los métodos **up** y **down** para hacer o deshacer la migración de la tabla especificada.



```
'use strict';

module.exports = {
  up: function (queryInterface, Sequelize) {
    /*
      Add altering commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.createTable('users', { id: Sequelize.INTEGER });
    */
  },
  down: function (queryInterface, Sequelize) {
    /*
      Add reverting commands here.
      Return a promise to correctly handle asynchronicity.

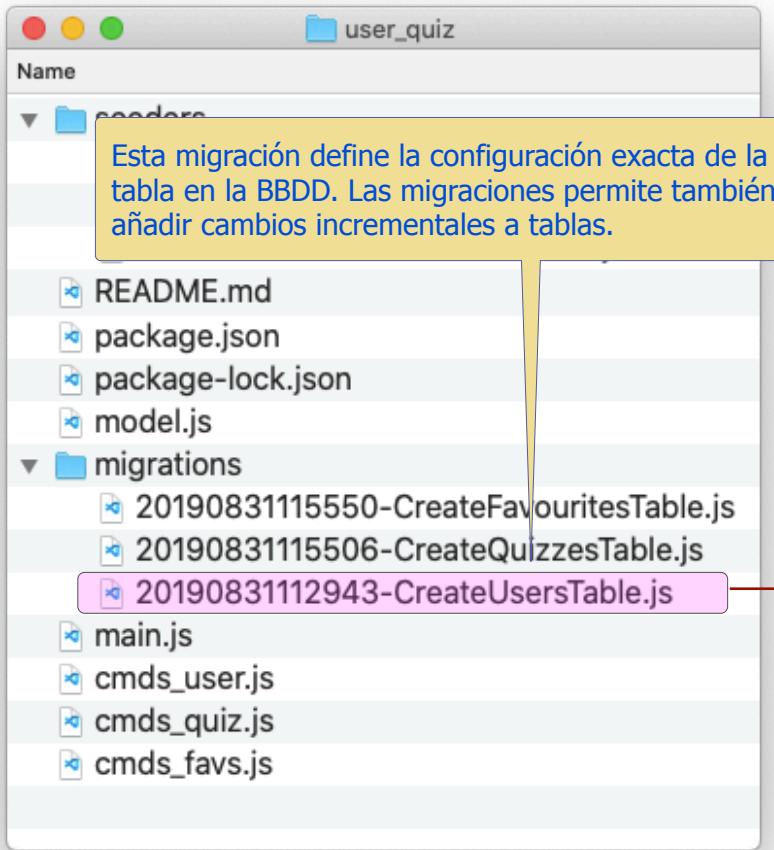
      Example:
      return queryInterface.dropTable('users');
    */
  }
};
```

20190831115550-CreateFavouritesTable.js

38

Estructura del nombre de una migración: <año><mes><día><hora><minuto><segundo>-<nombre>

Migración: 20190831112943- CreateUsersTable.js



```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable(
      'Users',
      {
        id: {
          type: Sequelize.INTEGER,
          allowNull: false,
          primaryKey: true,
          autoIncrement: true,
          unique: true
        },
        name: {
          type: Sequelize.STRING,
          unique: true
        },
        age: Sequelize.INTEGER,
        createdAt: {
          type: Sequelize.DATE,
          allowNull: false
        },
        updatedAt: {
          type: Sequelize.DATE,
          allowNull: false
        }
      }
    );
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('Users');
  }
};
```

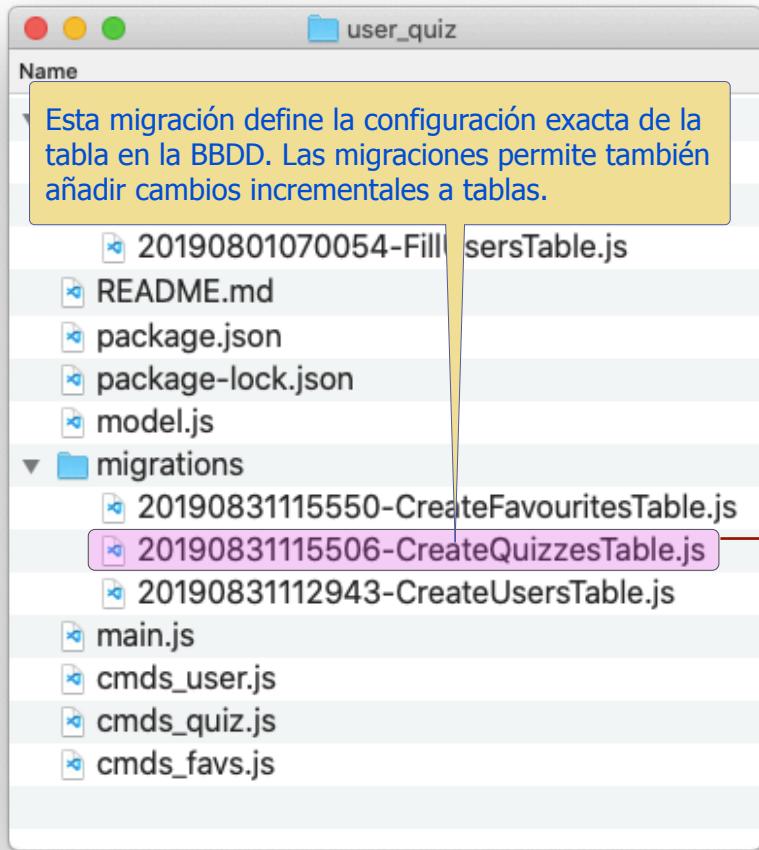
La función **up** define como realizar los cambios en la BBDD. Se debe completar manualmente incluyendo con el editor todos los campos que de la nueva tabla.

La migración debe incluir solo las opciones que definen la columna, pero **no** deben incluir las **validaciones**.

sync({force: true}) indica forzar los cambios al arrancar la aplicación si hay alguna incompatibilidad o error.

La función **down** define como deshacer los cambios en la BBDD..

Migración: 20190831115506- CreateQuizzesTable.js



authId es la clave externa de la relación posts-author.

references indica que referencia la clave primaria (**id**) de la tabla **Users**.

```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {

    return queryInterface.createTable(
      'Quizzes',
      {
        id: {
          type: Sequelize.INTEGER,
          allowNull: false,
          primaryKey: true,
          autoIncrement: true,
          unique: true
        },
        question: {
          type: Sequelize.STRING,
          unique: true
        },
        answer: Sequelize.STRING,
        authorId: {
          type: Sequelize.INTEGER,
          references: {
            model: "Users",
            key: "id"
          },
          onUpdate: 'CASCADE',
          onDelete: 'CASCADE'
        },
        createdAt: {
          type: Sequelize.DATE,
          allowNull: false
        },
        updatedAt: {
          type: Sequelize.DATE,
          allowNull: false
        }
      },
      {
        sync: {force: true}
      }
    );
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('Quizzes');
  }
};
```

La función **up** define como realizar los cambios en la BBDD. Se debe completar manualmente incluyendo con el editor todos los campos que de la nueva tabla.

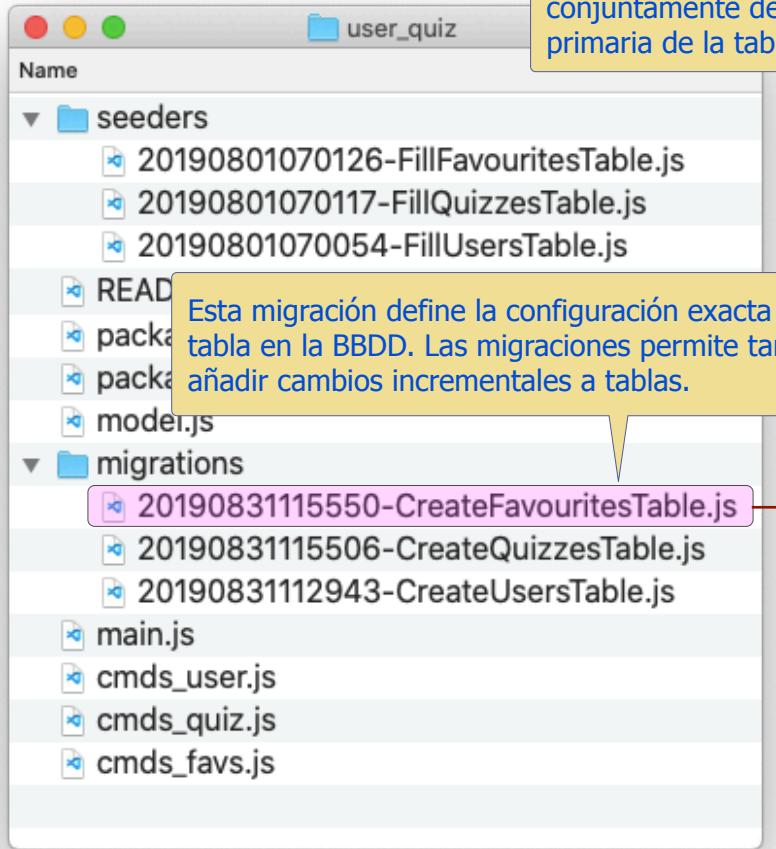
La migración debe incluir solo las opciones que definen la columna, pero **no** deben incluir las **validaciones**.

onUpdate y **onDelete** se configuran en modo '**CASCADE**', tal y como indica el modelo.

sync({force: true}) indica forzar los cambios al arrancar la aplicación si hay alguna incompatibilidad o error.

La función **down** define como deshacer los cambios en la BBDD..

Migración: 2019083115550- CreateFavouritesTable.js



```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('Favourites',
      {
        userId: {
          type: Sequelize.INTEGER,
          primaryKey: true,
          unique: "compositeKey",
          allowNull: false,
          references: {
            model: "Users",
            key: "id"
          },
          onUpdate: 'CASCADE',
          onDelete: 'CASCADE'
        },
        quizId: {
          type: Sequelize.INTEGER,
          primaryKey: true,
          unique: "compositeKey",
          allowNull: false,
          references: {
            model: "Quizzes",
            key: "id"
          },
          onUpdate: 'CASCADE',
          onDelete: 'CASCADE'
        },
        createdAt: {
          type: Sequelize.DATE,
          allowNull: false
        },
        updatedAt: {
          type: Sequelize.DATE,
          allowNull: false
        }
      },
      {
        sync: {force: true}
      }
    );
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('Favourites');
  }
};
```

userId y **quizId** actúan conjuntamente de clave primaria de la tabla.

La función **up** define como realizar los cambios en la BBDD. Se debe completar manualmente incluyendo con el editor todos los campos que de la nueva tabla.

La migración debe incluir solo las opciones que definen la columna, pero **no** deben incluir las **validaciones**.

onUpdate y **onDelete** se configuran en modo '**CASCADE**', tal y como estaban en el commit anterior.

sync({force: true}) indica forzar los cambios al arrancar la aplicación si hay alguna incompatibilidad o error.

La función **down** define como deshacer los cambios en la BBDD..

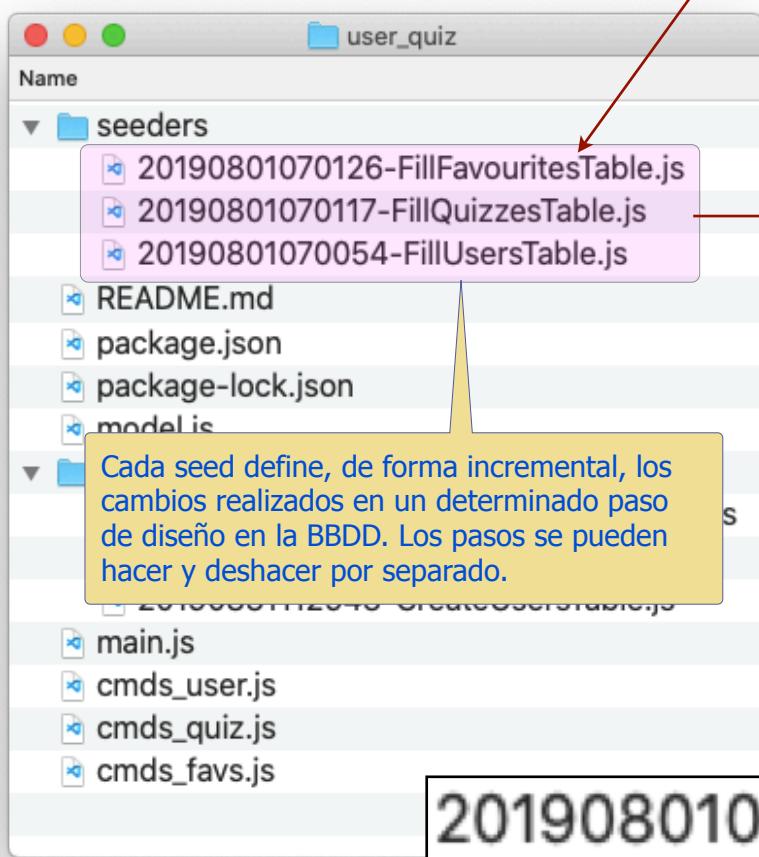
Crear esqueletos de seeders (ficheros semilla)

Crear el directorio **seeds** y el esqueleto de los los ficheros de seeds de las tablas users, quizzes y favourites con sequelize.

```
..$  
..$ npx sequelize seed:create --name FillUsersTable  
..$  
..$ npx sequelize seed:create --name FillQuizzesTable  
..$  
..$ npx sequelize seed:create --name FillFavouritesTable
```

Sequelize crea el esqueleto de los ficheros de seeds con un nombre que incluye la fecha de creación. Esta fecha es importante, porque los seeds se aplican siguiendo el orden de creación.

El esqueleto incluye los métodos **up** y **down** para hacer o deshacer los seeds de la tabla especificada.



```
'use strict';

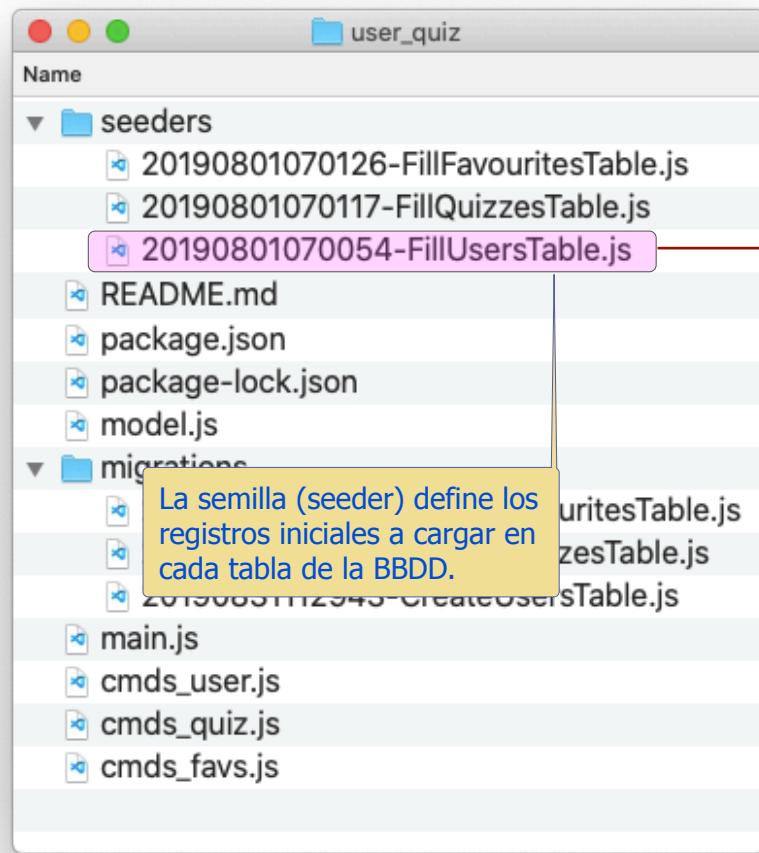
module.exports = {
  up: function (queryInterface, Sequelize) {
    /*
      Add altering commands here.
      Return a promise to correctly handle asynchronicity.

      Example:
      return queryInterface.bulkInsert('Person', [
        {
          name: 'John Doe',
          isBetaMember: false
        }
      ], {});
    },
    down: function (queryInterface, Sequelize) {
      /*
        Add reverting commands here.
        Return a promise to correctly handle asynchronicity.

        Example:
        return queryInterface.bulkDelete('Person', null, {});
      }
    };
  }
};
```

20190801070126-FillFavouritesTable.js

Seeder: 20190801070054-FillUsersTable.js



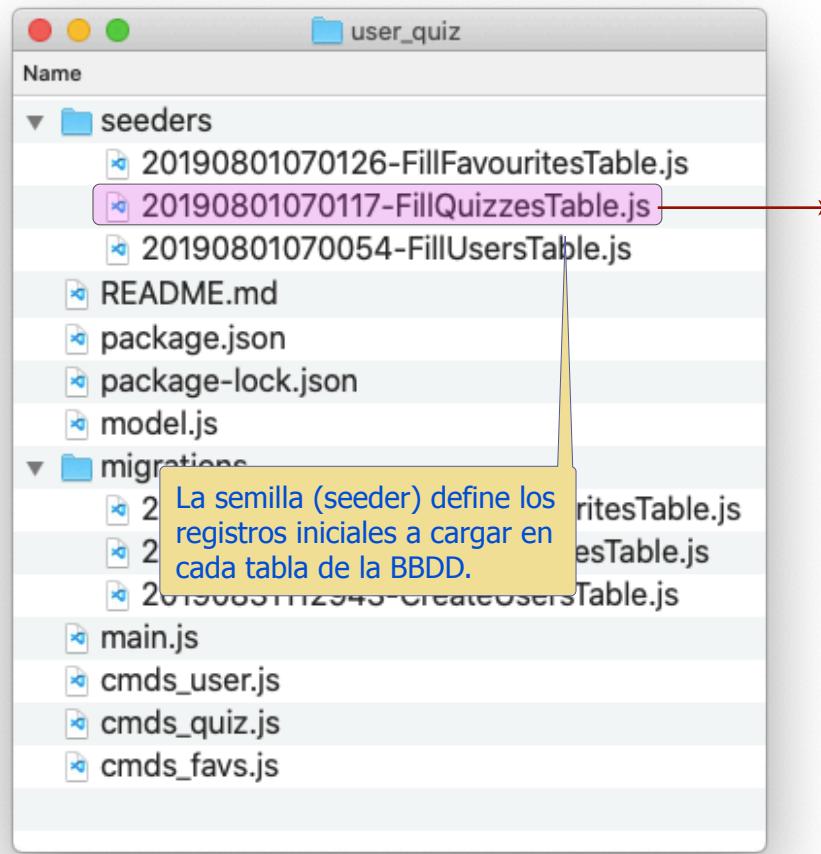
```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.bulkInsert('Users', [
      {
        name: 'Peter',
        age: 22,
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        name: 'Anna',
        age: 23,
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        name: 'John',
        age: 30,
        createdAt: new Date(),
        updatedAt: new Date()
      }
    ]);
  },
  down: function (queryInterface, Sequelize) {
    return queryInterface.bulkDelete('Users', null, {});
  }
};
```

La función **up** define como inicializar la tabla con la semilla. Debe completarse manualmente.

La función **down** define como eliminar los seeds de la tabla.

Seeder: 20190801070117-FillQuizzesTable.js



```
'use strict';

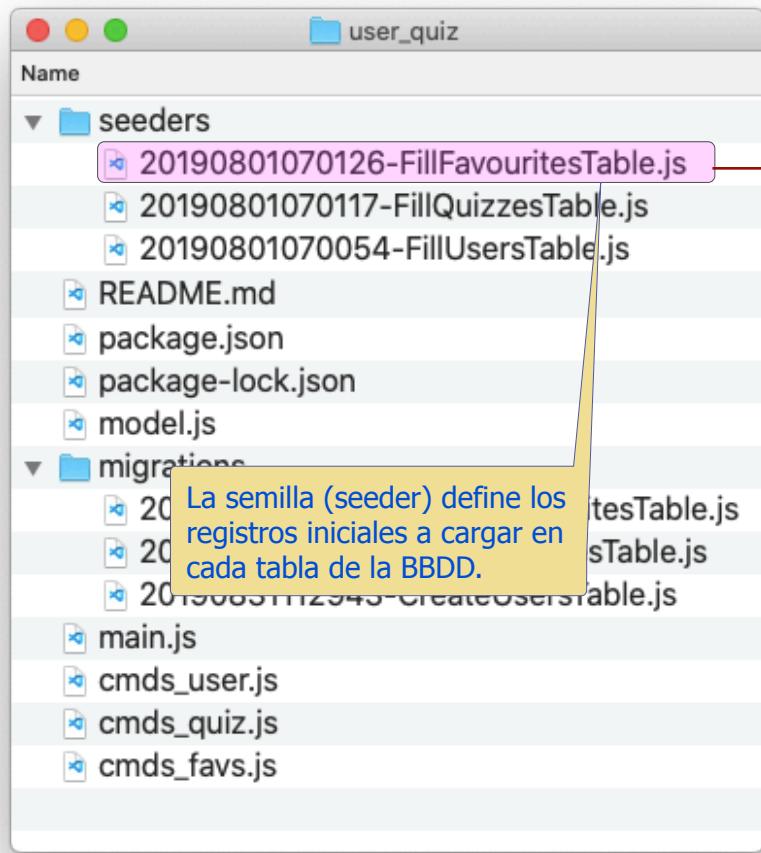
module.exports = {
  up: (queryInterface, Sequelize) => {

    return queryInterface.bulkInsert('Quizzes', [
      {
        question: 'Capital of Spain',
        answer: 'Madrid',
        authorId: 1,
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        question: 'Capital of France',
        answer: 'Paris',
        authorId: 1,
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        question: 'Capital of Italy',
        answer: 'Rome',
        authorId: 2,
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        question: 'Capital of Russia',
        answer: 'Moscow',
        authorId: 3,
        createdAt: new Date(),
        updatedAt: new Date()
      }
    ]);
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.bulkDelete('Quizzes', null, {});
  }
};
```

La función **up** define como inicializar la tabla con la semilla. Debe completarse manualmente.

La función **down** define como eliminar los seeds de la tabla.

Seeder: 20190801070126-FillFavouritesTable.js



La semilla (seeder) define los registros iniciales a cargar en cada tabla de la BBDD.

```
'use strict';

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.bulkInsert('Favourites', [
      {
        userId: '1',
        quizId: '3',
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        userId: '2',
        quizId: '4',
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        userId: '2',
        quizId: '1',
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        userId: '2',
        quizId: '2',
        createdAt: new Date(),
        updatedAt: new Date()
      },
      {
        userId: '3',
        quizId: '2',
        createdAt: new Date(),
        updatedAt: new Date()
      }
    ]);
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.bulkDelete('Favourites', null, {});
  }
};
```

La función **up** define como inicializar la tabla con la semilla. Debe completarse manualmente.

La función **down** define como eliminar los seeds de la tabla.



Final del tema
Muchas gracias!