JOINs com Sequelize, MySQL e Node.Js — parte 1

Edu Marcelino
May 27 · 7 min read

sequelize.query

Quem está acostumado com queries SQL e se depara com esse método do sequelize deve sentir um sorriso de orelha à orelha. Afinal se aproveitar das chamadas **raw queries** é uma mão na roda em casos em que o sequelize não suporte nativamente uma operação ou mesmo quando você ainda não aprendeu a fazer do modo nativo entregue pela ferramenta.

Se você está neste último caso, as próximas dicas são para você.

> ***Disclaimer e agradecimento***
>
> *Os conceitos demonstrados foram baseados no poster criado para mySQL por* ***Steve Stedman.*** *Deixo o link abaixo para quem se interessar*
>
> *http://stevestedman.com/2015/03/mysql-join-types-poster/*

Portanto vamos tentar reproduzir as ilustrações do poster no Sequelize, e exporar os muitos tipos de joins.

Configuração básica

Parto do princípio que você já tem em mãos Sequelize, Node.js e uma base de dados SQL, tudos instalados e prontos para os estudos.

Utilizarei três tabelas de exemplo:

Tabela 1: movies

Populada com 584 registros de filmes com as colunas abaixo com os seguintes campos:

name: Nome do filme
categoryId: Chave estrangeira para a tabela com as categorias de filme (gênero), como ação, comédia, drama, etc.
parentalRatingId: Chave estrangeira para a tabela de classificação parental, ou seja, se o filme é livre ou se existe classificação de idade.
directorName: Nome do diretor
dubLeg: Informa se os filmes possuem versão legendada

id	name	categoryId	parentalRatingID	releaseYear	directorName	dubLeg
1	007 Die Another day	22	6	2008	Akira Kurosawa	Legendado
2	10.000 A.C.	11	6	2008	Alfred Hitchcock	Dublado
3	100 Escovadas antes de dormir	13	5	2008	Bernardo Bertolucci	Dublado
4	101 Dalmatas	13	NULL	2008	Billy Wilder	Dublado
5	13 Fantasmas	18	NULL	2008	Charlie Chaplin	Dublado
6	13 Homens e um novo segredo	2	5	2008	D.W. Griffith	Legendado
7	13º Guerreiro - Decimo Terceiro Guerreiro	9	6	2008	David Lean	Legendado
8	1408	17	6	2008	Douglas Sirk	Legendado
9	16 Quadras	24	2	2008	Ernst Lubitsch	Legendado
10	300	2	5	2008	F.W. Murnau	Legendado
11	3.10	NULL	2	2008	Federico Fellini	Legendado
12	6º Dia - Sexto Dia	3	6	2008	Francis Ford Coppola	Legendado
13	60 Segundos	13	2	2008	François Truffaut	Dublado
14	A Bela e a Fera	21	2	2008	Frank Capra	Dublado
15	A Bussola de Ouro	12	1	2008	Fritz Lang	Legendado
16	A Caverna	22	6	2005	George Cukor	Legendado
17	A Cela	NULL	5	2005	Henry King	Legendado
18	A cor de um crime	14	NULL	2005	Howard Hawks	Legendado
19	A Dama e o Vagabundo	11	6	2005	Ingmar Bergman	Dublado
20	A Dinastia da Espada	22	2	2005	Jacques Tati	Legendado
21	A Era do Gelo	3	1	2005	Jean Renoir	Dublado
22	A Estranha Perfeita	9	NULL	2005	Jean-Luc Godard	Legendado
23	A Experiência	16	1	2005	John Ford	Legendado
24	A Experiência 3	2	5	2005	John Huston	Legendado
25	A Família da Noiva	NULL	1	2005	Jonathan Demme	Dublado
26	A Fonte	8	2	2005	Josef von Sternberg	Dublado

Os dados não representam a realidade. Servem apenas para estudo.

Tabela 2: category

id	name
1	Ação
2	Animação
3	Aventura
4	Cinema de arte
5	Chanchada
6	Comédia
7	Comédia romântica
8	Comédia dramática
9	Comédia de ação
10	Dança
11	Documentário
12	Docuficção
13	Drama
14	Espionagem
15	Faroeste
16	Fantasia científica
17	Ficção científica
18	Filmes de guerra
19	Musical
20	Filme policial
21	Romance
22	Seriado
23	Suspense
24	Terror

Tabela 3: parental

id	name
1	L
2	10
3	12
4	14
5	16
6	18
25	21
26	24

MODELS

Aqui criamos um model para cada tabela em um arquivo de models.

> modules.js

```
    id:         { type: Sequelize.INTEGER(2), autoIncrement: true, primaryKey: true },
    name:       { type: Sequelize.STRING(80) },
}, { freezeTableName: true });

const parental = conn.define('parental', {
    id:         { type: Sequelize.INTEGER(2), autoIncrement: true, primaryKey: true },
    name:       { type: Sequelize.STRING(80) },
}, { freezeTableName: true });

const movies = conn.define('movies', {
    id:                 { type: Sequelize.INTEGER(2), autoIncrement: true, primaryKey: true },
    name:               { type: Sequelize.STRING(80) },
    categoryId:         { type: Sequelize.INTEGER(2) },
    parentalRatingID:   { type: Sequelize.INTEGER(2) },
    releaseYear:        { type: Sequelize.INTEGER(4), validate: { min: 1800, max: 2100 } },
    directorName:       { type: Sequelize.STRING(80) },
    dubLeg:             { type: Sequelize.STRING(10) },
}, { freezeTableName: true });

movies.belongsTo(category, { foreignKey: 'categoryId', allowNull: true });
movies.belongsTo(parental, { foreignKey: 'parentalRatingID', allowNull: true });
```

Observem que utilizamos uma associação chamada **belongsTo.** Quer dizer que criamos uma relação de chaves estrangeiras entre as tabelas.
Então, no exemplo abaixo a tabela **movies** pertence à tabela categoria através do campo **categoryId**. Ou seja, a tabela 'dona' da informação "categorias" é a tabela **category** e nela existe a chave primária (**primary key**) no campo **id**.
A tabela **movies** utiliza desta informação em seu campo **categoryId**. Desta forma os dois campos, **movies.categoryId** e **category.id**, possuem um relacionamento **belongsTo**.

```
movies.belongsTo(category, { foreignKey: 'categoryId', allowNull: true });
```

O comando **allowNull** informa ao banco de dados para que permita a criação de um registro em **movies** com o campo **categoryId** nulo (vazio).

Mantenha seus models em arquivos separados para uma boa organização, e exporte-os para estarem disponíveis em outro arquivo .js

```
module.exports = { movies, category, parental }
```

CRIANDO O DATABASE E TABELAS

Crie um database no seu banco mySQL. Você pode fazer isso de várias formas. A que eu acho mais simples é utilizando a lib do seu banco. No meu caso, uso o MySQL então basta instalar o mysql2

```
npm install mysql2
```

Agora chamamos o mysql2 e podemos criar uma nova database:

```
const mysql2 = require("mysql2")
const connectionSql = mysql2.createConnection({
host: host,
user: user,
password: password
});
```

Com a conexão pronta, é só utilizar o comando.
Meu database chama-se sandbox

```
connectionSql.query("CREATE DATABASE IF NOT EXISTS sandbox", err=> )
```

Agora podemos iniciar o sequelize:

```
const Sequelize = require('sequelize')
const conn = new Sequelize(schema, user, password, params)
```

O sequelize pode criar as tabelas no banco de dados baseando-se nas definições dos Models, utilizando para o método sync.

```
conn.sync()
```

ok, Tudo pronto

Agora em outro arquivo .js basta chamar o arquivo dos models, onde exportamos os objetos

```
const db = require("./modules")
```

Agora vamos à parte interessante!

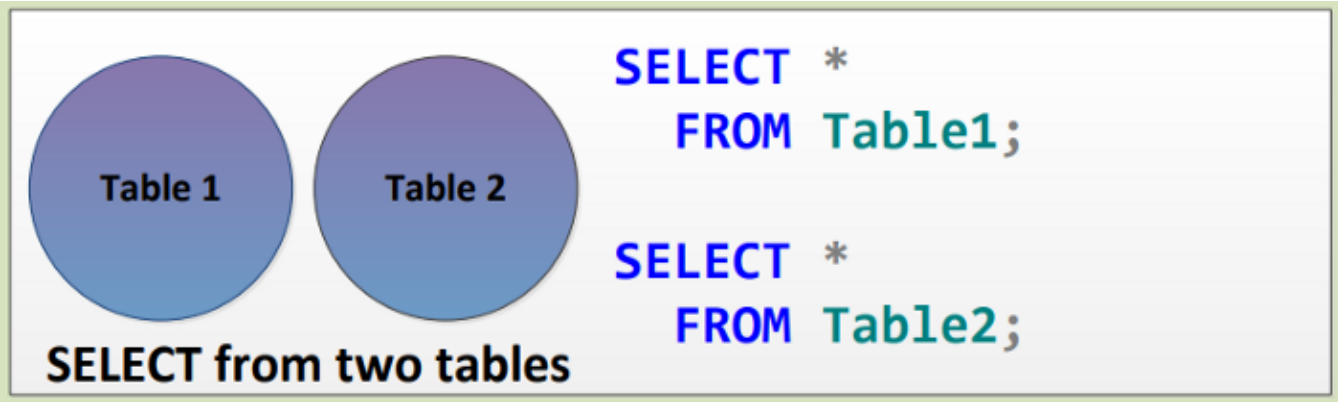
JOINS

> *A palavra reservada JOIN é utilizada para listar registros de duas ou mais tabelas de um banco de dados. Para utilizar o JOIN as tabelas precisam ser relacionadas. Em Bancos de Dados as tabelas são relacionadas umas com as outras através de* ***chaves primárias e estrangeiras****.*

> *Como você pode ver nas tabelas que criamos, a tabela* ***movies*** *possui uma chave primária chamada id, mas também possui as chaves estrangeiras* ***categoriId*** *e* ***ParentalRatingID.*** *Vamos trabalhar com isso daqui pra frente.*
>
> ***Veja nas ilustrações de Steve Stedman que as tabelas devem ser visualizadas na forma de conjuntos, onde uma está à esquerda e outra à direita da operação. Então quando falarmos em Direita/Right, falamos da tabela 1, e Esquerda/Left falamos da tabela 2.***

SELECT TWO TABLES

Vamos iniciar com o mais básico de todos. Uma seleção separada de duas tabelas.



http://stevestedman.com/wp-content/uploads/VennDiagram2.pdf

No sequelize ficará assim:

```
db.parental.findAll({ attributes: attributes, raw: true, })
    .then(parent => { console.table(parent) })

db.category.findAll({ attributes: attributes, raw: true, limit:10})
    .then(categs => { console.table(categs) })
```

Ok, não é join! Mas ilustra que podemos juntar tudo na mão utilizando as variáveis parent e categs

Vamos ver a resposta no log do sequelize:

```
Executing (default): SELECT `id`, `name` FROM `parental` AS `parental`;
Executing (default): SELECT `id`, `name` FROM `category` AS `category` LIMIT 10;
```

Aqui o log mostra exatamente o que o sequelize repassou ao mySql. Ele converteu o comando findAll em SELECT.

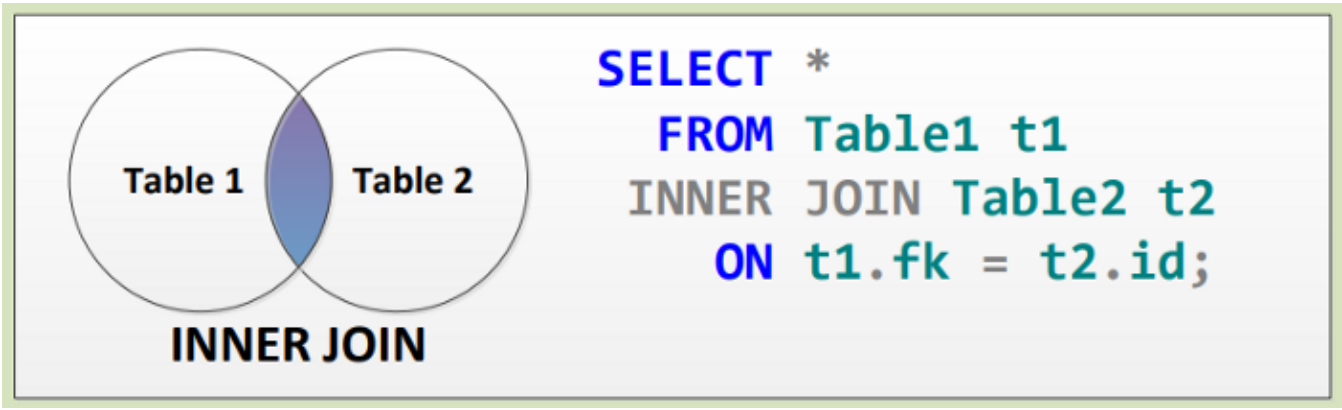
E ai está a saída:

(index)	id	name
0	1	'L'
1	2	'10'
2	3	'12'
3	4	'14'
4	5	'16'
5	6	'18'
6	25	'21'
7	26	'24'

(index)	id	name
0	1	'Ação'
1	2	'Animação'
2	3	'Aventura'
3	4	'Cinema de arte'
4	5	'Chanchada'
5	6	'Comédia'
6	7	'Comédia romântica'
7	8	'Comédia dramática'
8	9	'Comédia de ação'
9	10	'Dança'

INNER JOIN

O Inner Join consiste em unir/juntar informações de duas tabelas considerando apenas os dados que existam ao mesmo tempo nas duas através de suas chaves estrangeira e primária.



http://stevestedman.com/wp-content/uploads/VennDiagram2.pdf

No sequelize ficará assim:

```
db.movies.findAll({
    raw: true,
    attributes: attributes,
```

```
        include: [{
            model: db.parental,
            required: true,
            attributes: ['name'],
        }],
        order: [['id', 'ASC']],
    }).then(movies => console.table(movies))
}
```

Finalmente temos um primeiro JOIN! Que só foi possível graças ao parâmetro include, onde a segunda tabela deve ser informada, e ao parâmetro **required: true**, que explico melhor abaixo.

O objeto do do inner join e listar apenas as coincidências entre as tabelas, então o parâmetro **required: true** elimina da lista os registros que não foram encontrados em na tabela parental. (null não existe lá!)

O parâmetro atributes informa quais colunas queremos, unir no resultado.

> attributes: [‘name’],

Na saída teremos:

(index)	id	name	categoryId	dubLeg	parental.name
0	1	'007 Die Another day'	22	'Legendado'	'18'
1	2	'10.000 A.C. '	11	'Dublado'	'18'
2	3	'100 Escovadas antes de dormir'	13	'Dublado'	'16'
3	6	'13 Homens e um novo segredo'	2	'Legendado'	'16'
4	7	'13º Guerreiro - Decimo Terceiro Guerreiro'	9	'Legendado'	'18'
5	8	'1408'	17	'Legendado'	'18'
6	9	'16 Quadras'	24	'Legendado'	'10'
7	10	'300'	2	'Legendado'	'16'
8	11	'3.10'	null	'Legendado'	'10'
9	12	'6º Dia - Sexto Dia'	3	'Legendado'	'18'
10	13	'60 Segundos'	13	'Dublado'	'10'
11	14	'A Bela e a Fera'	21	'Dublado'	'10'
12	15	'A Bussola de Ouro'	12	'Legendado'	'L'

Veja que na tabela movies, o campo parentalRatinfID não está preenchido para o filme 4 e 5. E é isso que o **required: true** elimina.

id	name	categoryId	parentalRatingID	releaseYear
1	007 Die Another day	22	6	2008
2	10.000 A.C.	11	6	2008
3	100 Escovadas antes de dormir	13	5	2008
4	101 Dalmatas	13	NULL	2008
5	13 Fantasmas	18	NULL	2008
6	13 Homens e um novo segredo	2	5	2008

7	13° Guerreiro - Decimo Terceiro Guerreiro	9	6	2008

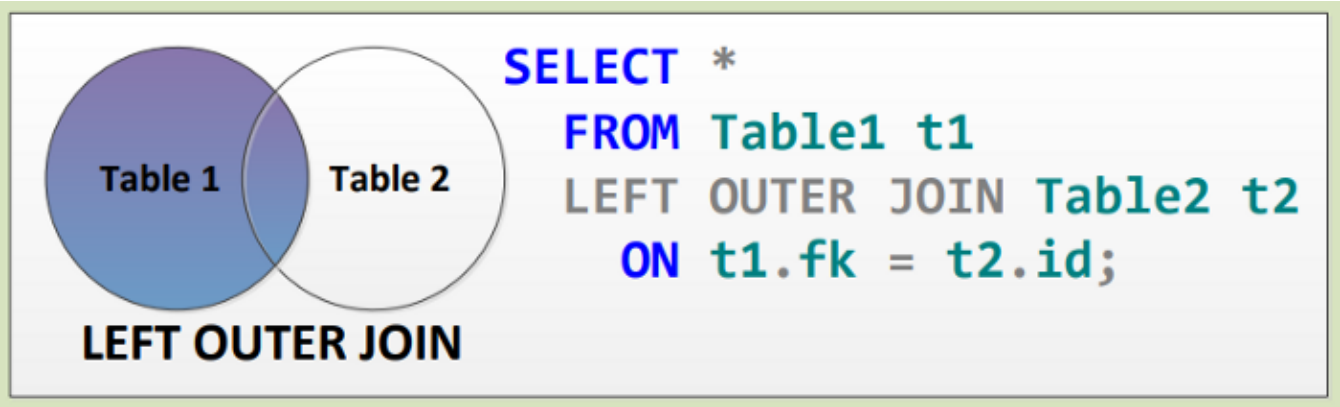
> **Mas como o sequelize sabe que era pra comparar utilizando o campo parentalRatingID ??**
>
> *Calma… Lembra da associação belongsTo que fizemos entre os campos?*

Vamos ver como ficou o log do sequelize, que prova que a operação foi mesmo um INNER JOIN:

```
Executing (default): SELECT `movies`.`id`, `movies`.`name`,
`movies`.`categoryId`, `movies`.`dubLeg`, `parental`.`name` AS `parental.name`
FROM `movies` AS `movies` INNER JOIN `parental` AS `parental` ON
`movies`.`parentalRatingID` = `parental`.`id` ORDER BY `movies`.`id` ASC;
```

LEFT OUTER JOIN

Enquanto o INNER JOIN se preocupa em demonstrar apenas as linhas coincidentes entre as tabelas movies e parental, o LEFT OUTEER JOIN que trazer todas. A tabela da esquerda/LEFT precisa ser listada integralmente, e a da direita aparecerá null se não existir.



http://stevestedman.com/wp-content/uploads/VennDiagram2.pdf

No sequelize ficará assim:

```
db.movies.findAll({
    raw: true,
    attributes: attributes,
    include: [{
        model: db.parental,
        required: false,
```

```
        attributes: ['name'],
    }],
    order: [['id', 'ASC']],
}).then(movies => console.table(movies))
```

Observe que a única diferença no comando em relação ao INNER JOIN é o parâmetro required que agora é false.

Na saída teremos:

(index)	id	name	categoryId	dubLeg	parental.name
0	1	'007 Die Another day'	22	'Legendado'	'18'
1	2	'10.000 A.C. '	11	'Dublado'	'18'
2	3	'100 Escovadas antes de dormir'	13	'Dublado'	'16'
3	4	'101 Dalmatas'	13	'Dublado'	null
4	5	'13 Fantasmas'	18	'Dublado'	null
5	6	'13 Homens e um novo segredo'	2	'Legendado'	'16'
6	7	'13º Guerreiro - Decimo Terceiro Guerreiro'	9	'Legendado'	'18'
7	8	'1408'	17	'Legendado'	'18'
8	9	'16 Quadras'	24	'Legendado'	'10'
9	10	'300'	2	'Legendado'	'16'
10	11	'3.10'	null	'Legendado'	'10'
11	12	'6º Dia - Sexto Dia'	3	'Legendado'	'18'
12	13	'60 Segundos'	13	'Dublado'	'10'
13	14	'A Bela e a Fera'	21	'Dublado'	'10'
14	15	'A Bussola de Ouro'	12	'Legendado'	'L'
15	16	'A Caverna'	22	'Legendado'	'18'
576	577	'X-Men - 5 Temp - Ep. 1 ao 14'	1	'Dublado'	null
577	578	'X-Men Evolution - Ep. 1 ao 42'	11	'Dublado'	null
578	579	'Xeque-Mate'	7	'Legendado'	'16'
579	580	'Xuxa Abracadabra'	1	'Legendado'	'10'
580	581	'Xuxa circo 5 SPB'	22	'Legendado'	null
581	582	'Xuxa só para baixinhos 3'	7	'Legendado'	'10'
582	583	'Zathura'	2	'Legendado'	'10'
583	584	'Zoolander'	16	'Legendado'	null

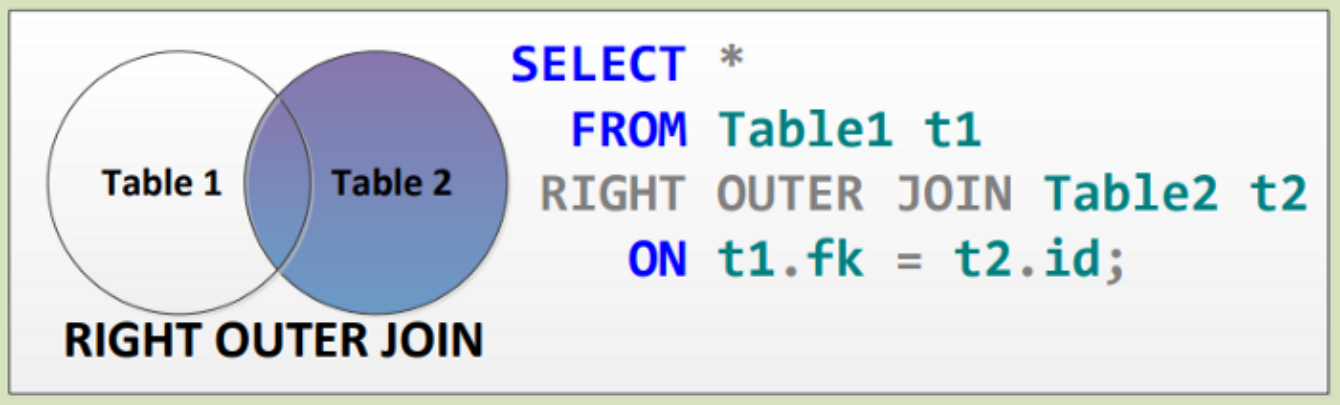
Veja que a coluna parental.name, que foi unida à lista pelo JOIN, possui registros nulos. Isso era o esperado para este tipo de JOIN.

Vamos ver como ficou o log do sequelize, que prova que a operação foi mesmo um LEFT OUTER JOIN:

> Executing (default): SELECT `movies`.`id`, `movies`.`name`, `movies`.`categoryId`, `movies`.`dubLeg`, `parental`.`name` AS `parental.name` FROM `movies` AS `movies` **LEFT OUTER JOIN** `parental` AS `parental` ON `movies`.`parentalRatingID` = `parental`.`id` ORDER BY `movies`.`id` ASC;

RIGHT OUTER JOIN

Vamos pensar na tabela da direita. Queremos aqui uma lista com todos os registros da tabela da direita, mesmo que não exista relação com a tabela da esquerda. Aqui a prioridade é a da direita.



http://stevestedman.com/wp-content/uploads/VennDiagram2.pdf

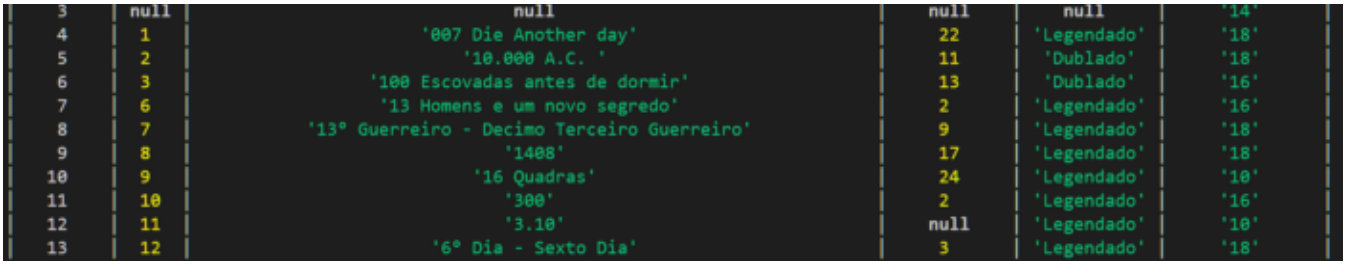
No sequelize ficará assim:

```
db.movies.findAll({
    raw: true,
    attributes: attributes,
    include: [{
        model: db.parental,
        required: false,
        right: true,
        attributes: ['name'],
    }],
    order: [['id', 'ASC']]
}).then(movies => console.table(movies))
```

A primeira coisa que vemos é que não é requerido que exista a informação preenchida na chave estrangeira com o campo da tabela da direita, então o parâmetro **required: false**.
E para definir qual tabela é de listagem obrigatória, marcamos o parâmetro **right: true**

Na saída teremos:

(index)	id	name	categoryId	dubLeg	parental.name
0	null	null	null	null	'24'
1	null	null	null	null	'12'
2	null	null	null	null	'21'



3	null	null	null	null	'14'
4	1	'007 Die Another day'	22	'Legendado'	'18'
5	2	'10.000 A.C. '	11	'Dublado'	'18'
6	3	'100 Escovadas antes de dormir'	13	'Dublado'	'16'
7	6	'13 Homens e um novo segredo'	2	'Legendado'	'16'
8	7	'13º Guerreiro - Decimo Terceiro Guerreiro'	9	'Legendado'	'18'
9	8	'1408'	17	'Legendado'	'18'
10	9	'16 Quadras'	24	'Legendado'	'10'
11	10	'300'	2	'Legendado'	'16'
12	11	'3.10'	null	'Legendado'	'10'
13	12	'6º Dia - Sexto Dia'	3	'Legendado'	'18'

Note que nenhum filme foi classificado com classificação parental 12, 14, 21 e 21 na tabela movies, e por isso temos todos os campos da tabela da esquerda com null, estando presente a tabela da direita em **parental.name**.

Vamos ver como ficou o log do sequelize, que prova que a operação foi mesmo um RIGHT OUTER JOIN:

```
Executing (default): SELECT `movies`.`id`, `movies`.`name`,
`movies`.`categoryId`, `movies`.`dubLeg`, `parental`.`name` AS `parental.name`
FROM `movies` AS `movies` RIGHT OUTER JOIN `parental` AS `parental` ON
`movies`.`parentalRatingID` = `parental`.`id` ORDER
BY `movies`.`id` ASC;
```

Na segunda parte deste artigo, falarei mais sobre os outros sabores do JOIN!

PARTE 2

Será que o sequelize (ou mySQL) atende a todos ?!

Sequelize Join MySQL