

JOINS com Sequelize, MySQL e Node.js — parte 2



Edu Marcelino

May 28 · 7 min read

• • •



Na parte 1 deste artigo falamos sobre alguns tipos de consultas do tipo JOIN.

SELECT TWO TABLES

INNER JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

Nesta segunda parte iremos nos aprofundar em mais tipos de Join com sequelize.

Recomendo ler a parte 1, se já lei, vamos em frente!

JOINS para a parte 2

Nesta parte do artigo veremos estes tipos de JOIN:

SEMI JOIN

ANTI SEMI JOIN

FULL OUTER JOIN

FULL OUTER JOIN EXCLUSION

TABELAS

Para você não se perder, os exemplos serão sempre com as tabelas criadas de exemplo: movie, category e parental

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 Fantasma	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

Tabela 1: movies

id	name
1	Ação
2	Animação
3	Aventura
4	Cinema de arte

1	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 2: category

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

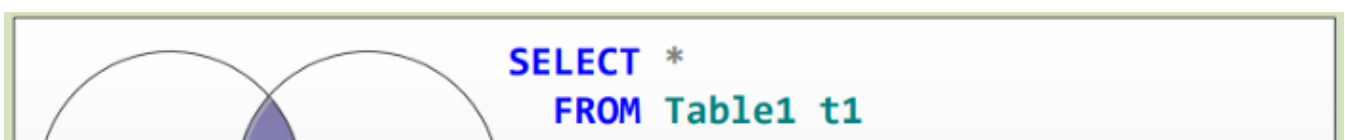
Tabela 3: parental

Vamos lá!

SEMI JOIN

Ou também chamado de **LEFT SEMI JOIN Subqueries**.

O objetivo é o mesmo do INNER JOIN em listar os registros presentes nas duas tabelas relacionadas por suas chaves primaria e estrangeira, ou seja, deve existir nas duas tabelas para ser listado. Porém, no INNER JOIN é possível incluir no resultado as colunas da tabela da direita, o que é possível no SEMI JOIN. Em outras palavras, as únicas colunas que serão exibidas pertencem à tabela da esquerda, enquanto que a tabela da direita serve de filtro da seleção.





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

Note que a palavra reservada JOIN não aparece na estrutura, ao invés disso existe apenas um select que retornará dados, e outro select de filtragem.

No sequelize ficará assim:

```
db.movies.findAll({
  raw: true,
  attributes: attributes,
  where: {
    $and: sequelize.literal(`exists (
      select 1 from parental
      where id = parentalRatingID`)
  },
  order: [['id', 'ASC']],
}).then(movies => console.table(movies))
```

Observe que a única coisa que é passada para o primeiro where é se existe um **movie.parentalRatingID** na tabela **parental.id**, não importando mais nada.

Na saída teremos:

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

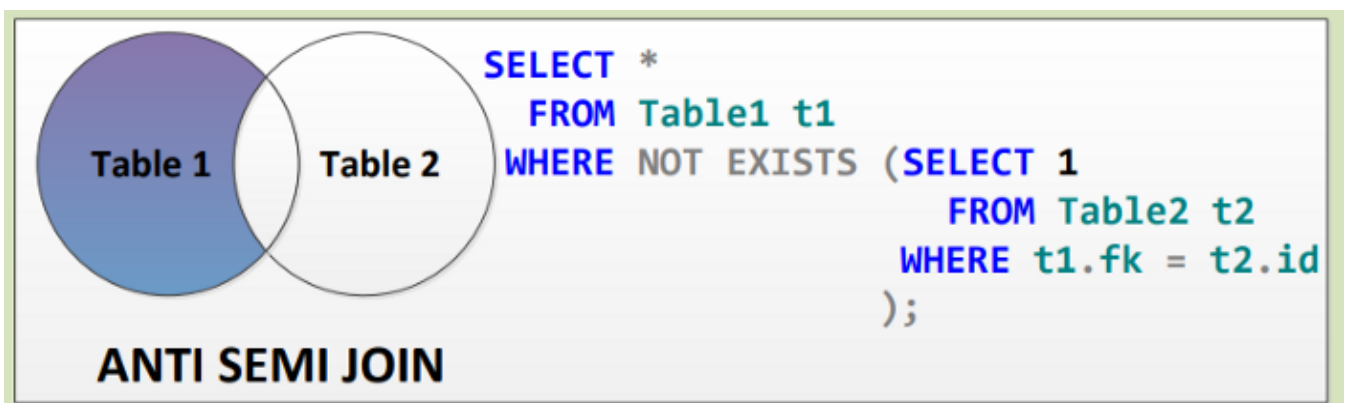
Veja a saída do INNER JOIN na parte 1 deste artigo e compare. Adianto que são iguais exceto que neste SEMI JOIN não é possível exibir a coluna **parental.name**.

Vamos ver como ficou o log do sequelize:

```
Executing (default): SELECT `id`, `name`, `categoryId`, `dubLeg` FROM `movies`
AS `movies` WHERE exists (select 1 from parental where id = parentalRatingID )
ORDER BY `movies`.`id` ASC;
```

ANTI SEMI JOIN

Este tipo é o oposto do SEMI JOIN, ou seja, selecionar os registros da tabela da esquerda onde NÃO existam registros com o mesmo valor de chave estrangeira.



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

No sequelize ficará assim:

```
db.movies.findAll({
  raw: true,
  attributes: attributes,
  where: {
    $and: sequelize.literal(`not exists (
                                select 1 from parental
                                where id = parentalRatingID )`)
  },
  order: [['id', 'ASC']],
}).then(movies => console.table(movies))
```

No ANTI SEMI JOIN incluímos o **not exists** na cláusula where, o que fará retornar verdadeiro se o movies.id existir em parental.parentalRatingID.

Na saída teremos:

(index)	id	name	categoryId	dubLeg
0	1	1301 - O 3º Motor	12	'DubLegend'

1	5	'13 Fantasma'	18	'Dublado'
2	18	'A cor de um crime'	14	'Legendado'
3	22	'A Estranha Perfeita'	9	'Legendado'
4	32	'A Lenda de Beowulf'	16	'Dublado'
5	33	'A Liga Extraordinária'	8	'Legendado'
6	35	'A mulher do açougueiro'	24	'Legendado'
7	36	'A Mumia 3 - A Tumba do Imperador Dragão'	2	'Dublado'
8	40	'A Rede 2.0'	7	'Legendado'
9	41	'A Super Máquina - Knight Rider'	22	'Dublado'
10	47	'Acampamento do Papai'	3	'Dublado'
11	48	'Adivinhe quem vem para morrer'	3	'Dublado'
12	53	'Albergue 2 - Hostel 2'	23	'Legendado'
13	54	'Além das Linhas Inimigas'	17	'Legendado'
14	55	'Alien - O Oitavo Passageiro'	19	'Legendado'

Veja retornou apenas os registro que estão com a chave estrangeira null.

id	name	categoryId	parentalRatingID
1	007 Die Another day	22	6
2	10.000 A.C.	11	6
3	100 Escovadas antes de dormir	13	5
4	101 Dalmatas	13	NULL
5	13 Fantasma	18	NULL
6	13 Homens e um novo segredo	2	5
7	13º Guerreiro - Decimo Terceiro Guerreiro	9	6
8	1408	17	6
9	16 Quadras	24	2
10	300	2	5
11	3.10	NULL	2
12	6º Dia - Sexto Dia	3	6
13	60 Segundos	13	2
14	A Bela e a Fera	21	2
15	A Bussola de Ouro	12	1
16	A Caverna	22	6
17	A Cela	NULL	5
18	A cor de um crime	14	NULL
19	A Dama e o Vagabundo	11	6
20	A Dinastia da Espada	22	2
21	A Era do Gelo	3	1
22	A Estranha Perfeita	9	NULL
23	A Experiência	16	1
24	A Experiência 3	2	5
25	A Família da Noiva	NULL	1
26	A Fonte	8	2

Vamos ver como ficou o log do sequelize:

```
Executing (default): SELECT `id`, `name`, `categoryId`, `dubLeg` FROM `movies`
AS `movies` WHERE not exists (
select 1 from parental
where id = parentalRatingID ) ORDER BY `movies`.`id` ASC;
```

FULL OUTER JOIN

Apesar de estar especificado no padrão SQL, este tipo de JOIN não é suportado pelo MySQL e forks como o MariaDB. Mesmo se fosse possível, o Sequelize não possibilitaria este tipo de operação.

Na especificação do SQL o comando tem este formato:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

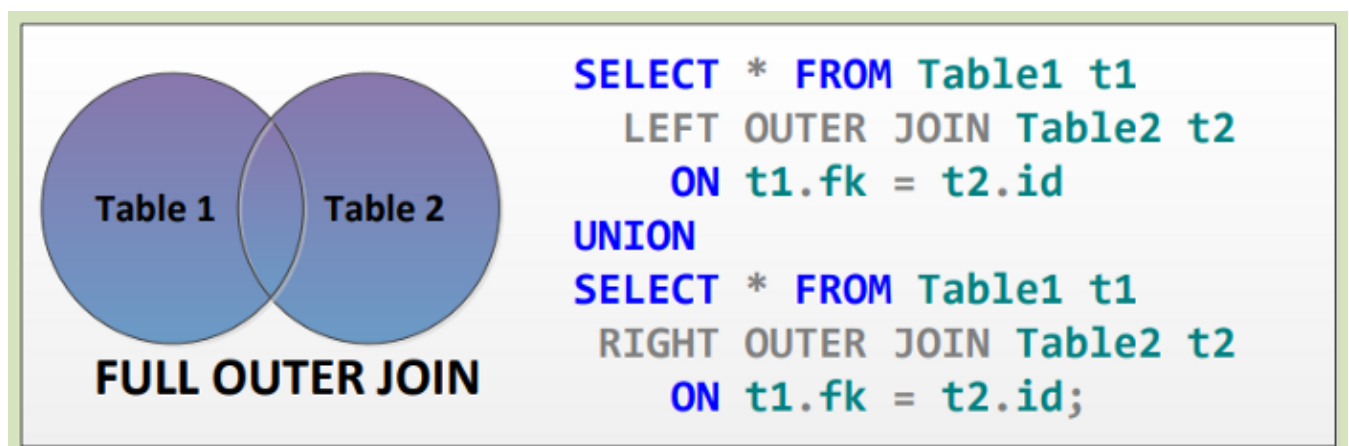
https://www.w3schools.com/sql/sql_join_full.asp

O FULL OUTER JOIN retorna todas as linhas da tabela da direita e todas as linhas da tabela da esquerda, tendo ou não relação de igualdade nas chaves.

É utilizado quando queremos unir duas tabelas obtendo o mesmo resultado de um LEFT OUTER JOIN e RIGHT OUTER JOIN juntos.

Por este motivo podemos simular o mesmo resultado utilizando a palavra reservada UNION.

UNION é capaz de unificar dois SELECTs conforme a figura abaixo.



Como já citei acima, o Sequelize não suporta operações FULL OUTER JOIN, e mesmo que suportasse teríamos erros no banco de dados pois o sequelize converte os seus comando em comandos SQL antes de enviar ao MySQL.

O Sequelize também não disponibiliza parâmetros para fazermos uma união direta de

selects.

Nesta situação teremos que recorrer às **RAW QUERIES**.

***RAW QUERIES** são formas de passar diretamente os comandos SQL, sem utilizar os recursos nativos do Sequelize.*

Exemplo de Raw query:

```
sequelize.query( "SELECT table1 WHERE id > 100")
```

Faremos então uma chamada UNION entre os SELECTs, simulando assim um FULL OUTER JOIN.

No sequelize ficará assim:

```
db.conn.query(  
  `SELECT t1.id, t1.name as Categ, t2.name as Rating  
    FROM category t1  
    LEFT OUTER JOIN parental t2 ON t1.id = t2.id  
  UNION  
    SELECT t1.id, t1.name as Categ, t2.name as Rating  
    FROM category as t1  
    RIGHT OUTER JOIN parental as t2 ON t1.id = t2.id  
  `.  
  .replace(/\s{2,}/g, ' ')  
)  
.then(function(rows){  
  console.table(rows[0].map(w=> w ))  
})
```

Vamos entender passo a passo este exemplo:

- 1- Listar o resultado do primeiro select, que é um LEFT JOIN entre as tabelas **category** e **parental**. Lembrando que LEFT JOIN retornara todas as linhas da tabela da esquerda.
- 2- Queremos ver as colunas id e name da tabela category e o campo name da tabela parental (t1.id, t1.name as Categ, t2.name as Rating)
- 3- Declaramos UNION, que solicita a união dos resultados.

4- Listar o resultado do segundo select, que é um RIGHT JOIN entre as tabelas **category** e **parental**. Lembrando que RIGHT JOIN retornara todas as linhas da tabela da direita.

5- Como um bônus, coloquei um replace para remover os espaços duplos deixados na formatação da string entre a crases ``. Mas isso você pode ignorar.

Na saída teremos:

(index)	id	Categ	Rating
0	1	'Ação'	'L'
1	2	'Animação'	'10'
2	3	'Aventura'	'12'
3	4	'Cinema de arte'	'14'
4	5	'Chanchada'	'16'
5	6	'Comédia'	'18'
6	7	'Comédia romântica'	null
7	8	'Comédia dramática'	null
8	9	'Comédia de ação'	null
9	10	'Dança'	null
10	11	'Documentário'	null
11	12	'Docuficção'	null
12	13	'Drama'	null
13	14	'Espionagem'	null
14	15	'Faroeste'	null
15	16	'Fantasia científica'	null
16	17	'Ficção científica'	null
17	18	'Filmes de guerra'	null
18	19	'Musical'	null
19	20	'Filme policial'	null
20	21	'Romance'	null
21	22	'Seriado'	null
22	23	'Suspense'	null
23	24	'Terror'	null
24	null	null	'21'
25	null	null	'24'

O que significa esse resultado ?

A tabela category possui 24 registros e está toda aí por conta do LEFT JOIN. Os ids de 1 a 6 também existem na tabela parental, e por isso a coluna Rating está populada (lembrando que renomeamos a coluna t2.name para Rating no select)

À partir do id 7, nenhum registro existe na coluna parental, e por isso recebe NULL. Já as últimas duas linhas possuem ids que não existem na tabela category, mas existe na tabela parental, e por isso estão ali informados e null nas demais colunas.

Releia, de uma respirada, porque é mesmo um pouco confuso. Mas entendendo bem o FULL OUTER JOIN, os outros todos ficam muito fáceis.

O sequelize nos retornará em log o mesmo select que enviamos, já que usamos RAW QUERY:

```
Executing (default): SELECT t1.id, t1.name as Categ, t2.name as Rating FROM category t1 LEFT OUTER JOIN parental t2 ON t1.id = t2.id UNION SELECT t1.id, t1.name as Categ, t2.name as Rating FROM category as t1 RIGHT OUTER JOIN parental as t2 ON t1.id = t2.id
```

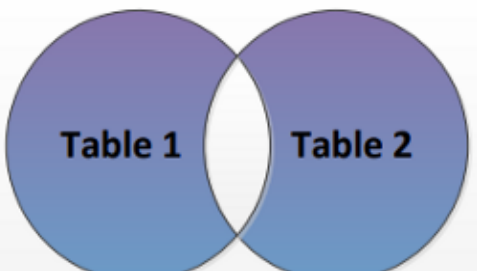
FULL OUTER JOIN EXCLUSION

Este comando frito minha cabeça e ainda tem mais?

Sim, mais um pouco, aguenta!

Você já sabe que o FULL OUTER JOIN traz tudo, e se não existir ele seta com null.

Mas agora, queremos todas as linhas de ambas as colunas excluindo as que não coincidem. Ou seja, se tem na tabela 1 mas não tem na tabela 2, não mostre, e vice-versa.



FULL OUTER JOIN with exclusion

```
SELECT * FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t2.id IS NOT NULL
UNION
SELECT * FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.fk = t2.id
WHERE t1.ID IS NOT NULL;
```

No sequelize ficará assim:

```
db.conn.query(  
  `SELECT t1.id, t1.name as Categ, t2.name as Rating  
    FROM category t1  
      LEFT OUTER JOIN parental t2 ON t1.id = t2.id  
      WHERE t2.id is not null  
    UNION  
    SELECT t1.id, t1.name as Categ, t2.name as Rating  
      FROM category as t1  
        RIGHT OUTER JOIN parental as t2 ON t1.id = t2.id  
        WHERE t1.id is not null  
  `.replace(/\s{2,}/g, ' ')  
)
```

Da mesma forma que o anterior usamos RAW QUERY, e apenas incluímos o WHERE com is not null, para não trazer valores nulos.

Na saída teremos:

(index)	id	Categ	Rating
0	1	'Ação'	'L'
1	2	'Animação'	'10'
2	3	'Aventura'	'12'
3	4	'Cinema de arte'	'14'
4	5	'Chanchada'	'16'
5	6	'Comédia'	'18'

Então aí está, todos os ids (pois a validação é por `t1.id = t2.id`), que são comuns a ambas tabelas estão presentes.

Vamos ver como ficou o log do sequelize:

```
Executing (default): SELECT t1.id, t1.name as Categ, t2.name as Rating FROM  
category t1 LEFT OUTER JOIN parental t2 ON t1.id = t2.id WHERE t2.id is not null  
UNION SELECT t1.id, t1.name as Categ, t2.name as Rating FROM category as t1
```

```
RIGHT OUTER JOIN parental as  
t2 ON t1.id = t2.id WHERE t1.id is not null
```

Ufa! esse foi mais complicado...

Espero que de alguma forma passa ter ajudado você a compreender melhor o mundo dos JOINS.

Todos os arquivos deste artigo estão disponíveis no github

<https://github.com/emarcelino3/sequelize-joins>

Deixe seu comentário e até a próxima!

Sequelize MySQL Nodejs

[About](#) [Help](#) [Legal](#)

Get the Medium app

