Preload, Eagerload, Includes and Joins

By **Neeraj Singh** in **Rails** *on July 01, 2013*

Rails provides four different ways to load association data. In this blog we are going to look at each of those.

Preload

Preload loads the association data in a separate query.

```
User.preload(:posts).to_a

# =>
SELECT "users".* FROM "users"
SELECT "posts".* FROM "posts"  WHERE "posts"."user_id" IN (1)
```

This is how `includes` loads data in the default case.

Since `preload` always generates two sql we can't use `posts` table in where condition. Following query will result in an error.

```
User.preload(:posts).where("posts.desc='ruby is awesome'")

# =>
SQLite3::SQLException: no such column: posts.desc:
SELECT "users".* FROM "users"  WHERE (posts.desc='ruby is awesome')
```

With preload where clauses can be applied.

```
User.preload(:posts).where("users.name='Neeraj'")

# =>
SELECT "users".* FROM "users"  WHERE (users.name='Neeraj')
SELECT "posts".* FROM "posts"  WHERE "posts"."user_id" IN (3)
```

Includes

Includes loads the association data in a separate query just like `preload`.

However it is smarter than `preload`. Above we saw that `preload` failed for query `User.preload(:posts).where("posts.desc='ruby is awesome'")`. Let's try same with includes.

```
User.includes(:posts).where('posts.desc = "ruby is awesome"').to_a

# =>
SELECT "users"."id" AS t0_r0, "users"."name" AS t0_r1, "posts"."id" AS
t1_r0,
       "posts"."title" AS t1_r1,
       "posts"."user_id" AS t1_r2, "posts"."desc" AS t1_r3
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user_id" = "users"."id"
WHERE (posts.desc = "ruby is awesome")
```

As you can see `includes` switches from using two separate queries to creating a single `LEFT OUTER JOIN` to get the data. And it also applied the supplied condition.

So `includes` changes from two queries to a single query in some cases. By default for a simple case it will use two queries. Let's say that for some reason you want to force a simple `includes` case to use a single query instead of two. Use `references` to achieve that.

```
User.includes(:posts).references(:posts).to_a

# =>
SELECT "users"."id" AS t0_r0, "users"."name" AS t0_r1, "posts"."id" AS
t1_r0,
       "posts"."title" AS t1_r1,
       "posts"."user_id" AS t1_r2, "posts"."desc" AS t1_r3
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user_id" = "users"."id"
```

In the above case a single query was done.

Eager load

eager loading loads all association in a single query using `LEFT OUTER JOIN`.

```
User.eager_load(:posts).to_a

# =>
SELECT "users"."id" AS t0_r0, "users"."name" AS t0_r1, "posts"."id" AS
t1_r0,
       "posts"."title" AS t1_r1, "posts"."user_id" AS t1_r2,
"posts"."desc" AS t1_r3
FROM "users" LEFT OUTER JOIN "posts" ON "posts"."user_id" = "users"."id"
```

This is exactly what `includes` does when it is forced to make a single query when `where` or `order` clause is using an attribute from `posts` table.

Joins

Joins brings association data using `inner join`.

```
User.joins(:posts)

# =>
SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" =
"users"."id"
```

In the above case no posts data is selected. Above query can also produce duplicate result. To see it let's create some sample data.

```
def self.setup
  User.delete_all
  Post.delete_all

  u = User.create name: 'Neeraj'
  u.posts.create! title: 'ruby', desc: 'ruby is awesome'
  u.posts.create! title: 'rails', desc: 'rails is awesome'
  u.posts.create! title: 'JavaScript', desc: 'JavaScript is awesome'

  u = User.create name: 'Neil'
  u.posts.create! title: 'JavaScript', desc: 'Javascript is awesome'

  u = User.create name: 'Trisha'
end
```

With the above sample data if we execute `User.joins(:posts)` then this is the result we get

```
#<User id: 9, name: "Neeraj">
#<User id: 9, name: "Neeraj">
#<User id: 9, name: "Neeraj">
#<User id: 10, name: "Neil">
```

We can avoid the duplication by using `distinct` .

```
User.joins(:posts).select('distinct users.*').to_a
```

Also if we want to make use of attributes from `posts` table then we need to select them.

```
records = User.joins(:posts).select('distinct users.*, posts.title as
posts_title').to_a
records.each do |user|
  puts user.name
  puts user.posts_title
end
```

Note that using `joins` means if you use `user.posts` then another query will be performed.

By **Neeraj Singh** In **Rails**
on July 01, 2013