# Ruby on Rails 實戰聖經

*使用 Rails 3.2 及 Ruby 1.9.3*

SEARCH

本書尚未完成，如果您有任何意見、鼓勵或勘誤，歡迎來信給我，謝謝。

## ActiveRecord 資料表關係

回首頁

本章目錄

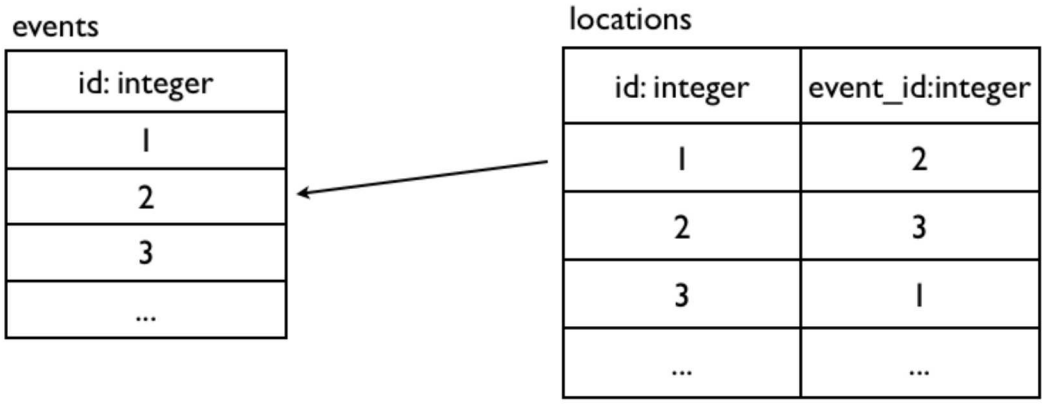


*ActiveRecord* 可以用 *Associations* 來定義資料表之間的關聯性，這是最被大家眼睛一亮 *ORM* 功能。到目前為止我們學會了用 *ActiveRecord* 來操作資料庫，但是還沒充分發揮關聯式資料庫的特性，那就是透過 *primary key* 和 *foreign keys* 將資料表互相關連起來。

> *Primary Key* 主鍵是一張資料表可以用來唯一識別的欄位，而 *Foreign Key* 外部鍵則是用來指向別張資料表的 *Primary Key*，如此便可以產生資料表之間的關聯關係。了解如何設計正規化關聯式資料庫請參考附錄基礎。

*Primary Key* 這個欄位在 *Rails* 中，照慣例叫做 *id*，型別是整數且遞增。而 *Foreign Key* 欄位照慣例會叫做 `{model_name}_id`，型別是整數。

### 一對一關聯 *one-to-one*




延續 *Part1* 的 *Event Model* 範例，假設一個 *Event* 擁有一個 *Location*。來新增一個 *Location Model*，其中的 *event_id* 就是外部鍵欄位：

```
rails g model location name:string event_id:integer
```

執行 `bundle exec rake db:migrate` 產生 *locations* 資料表。

分別編輯 *app/models/event.rb* 和 *app/models/location.rb*：

```
class Event < ActiveRecord::Base
    has_one :location # 單數
    #...
end

class Location < ActiveRecord::Base
    belongs_to :event # 單數
end
```

*belongs_to* 和 *has_one* 這兩個方法，會分別動態新增一些方法到 *Location* 和 *Event Model*

上，讓我們進入 `rails console` 實際操作資料庫看看，透過 *Associations* 你會發現操作關聯的物件非常直覺：

範例一，建立 *Location* 物件並關聯到 *Event*：

```
e = Event.first
l = Location.new( :name => 'Hsinchu', :event => e )
# 等同於 l = Location.new( :name => 'Hsinchu', :event_id => e.id )
l.save
e.location
l.event
```

`Event.first` 會撈出 *events table* 的第一筆資料，如果你第一筆還在，那就會是 `Event.find(1)`。同理，`Event.last` 會撈出最後一筆。

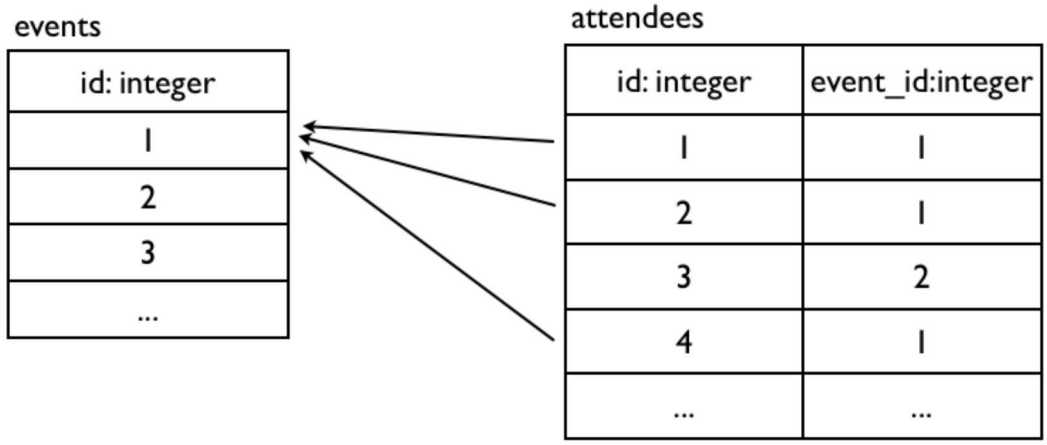
範例二，從 *Event* 物件中建立一個 *Location*：

```
e = Event.first
l = e.build_location( :name => 'Hsinchu' )
l.save
e.location
l.event
```

範例三，直接從 *Event* 物件中建立一個 *Location*：

```
e = Event.first
l = e.create_location( :name => 'Hsinchu' )
e.location
l.event
```

## 一對多關聯 *one-to-many*




一對多關聯算是最常用的，例如一個 *Event* 擁有很多 *Attendee*，來新增 *Attendee Model*：

```
rails g model attendee name:string event_id:integer
```

執行 `bundle exec rake db:migrate` 產生 *attendees* 資料表。

分別編輯 *app/models/event.rb* 和 *app/models/attendee.rb*：

```
class Event < ActiveRecord::Base
    has_many :attendees # 複數
    #...
end

class Attendee < ActiveRecord::Base
    belongs_to :event # 單數
end
```

同樣地， *belongs_to* 和 *has_many* 這兩個方法，會分別動態新增一些方法到 *Attendee* 和 *Event Model* 上，讓我們進入 `rails console` 實際操作資料庫看看：

### 範例一，建立 *Attendee* 物件並關聯到 *Event*：

```
e = Event.first
a = Attendee.new( :name => 'ihower', :event => e )
# 或 a = Attendee.new( :name => 'ihower', :event_id => e.id )
a.save
e.attendees # 這是陣列
e.attendees.size
Attendee.first.event
```

### 範例二，從 *Event* 物件中建立一個 *Attendee*：

```
e = Event.first
a = e.attendees.build( :name => 'ihower' )
a.save
e.attendees
```

### 範例三，直接從 *Event* 物件中建立一個 *Attendee*：

```
e = Event.first
a = e.attendees.create( :name => 'ihower' )
e.attendees
```

### 範例四，先建立 *Attendee* 物件再放到 *Event* 中:

```
e = Event.first
a = Attendee.create( :name => 'ihower' )
e.attendees << a
e.attendees
```

### 範例五，根據特定的 *Event* 查詢 *Attendee*

```
e = Event.first
e.id # 1
a = e.attendees.find(3)
attendees = e.attendees.where( :name => 'ihower' )
```

這樣就可以寫出限定在某個 *Event* 下的條件查詢，用這種寫法可以避免一些安全性問題，不會讓沒有權限的使用者搜尋到別的 *Event* 的 *Attendee* 。

### 範例六，刪除

```
Event.attendees.destroy_all # 會一筆筆觸發Attendee的destroy回呼
Event.attendees.delete_all # 不會觸發Attendee的destroy回呼
```

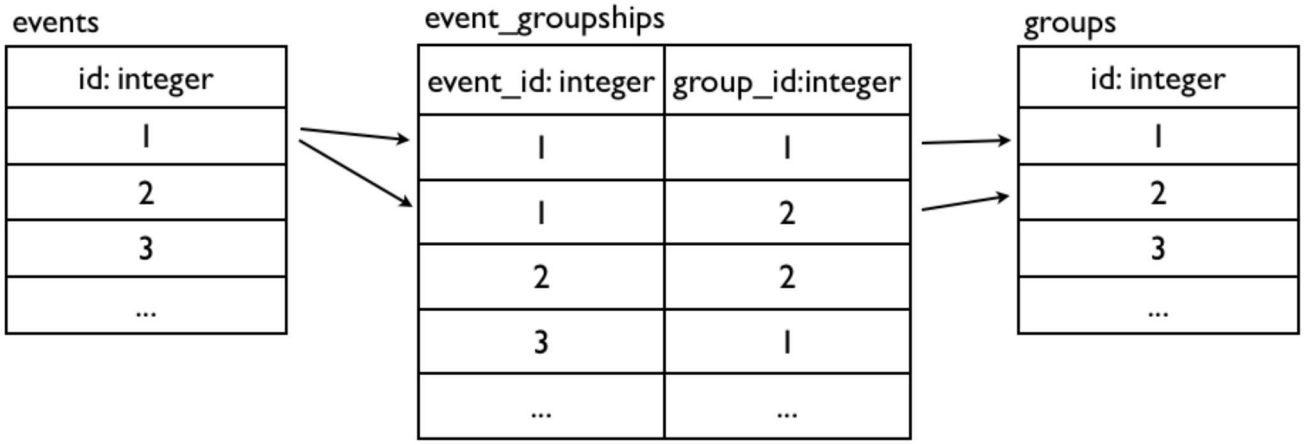
> 有個口訣可以記起來：有 *Foreign Key* 的 *Model* ，就是設定 `belongs_to` 的Model。

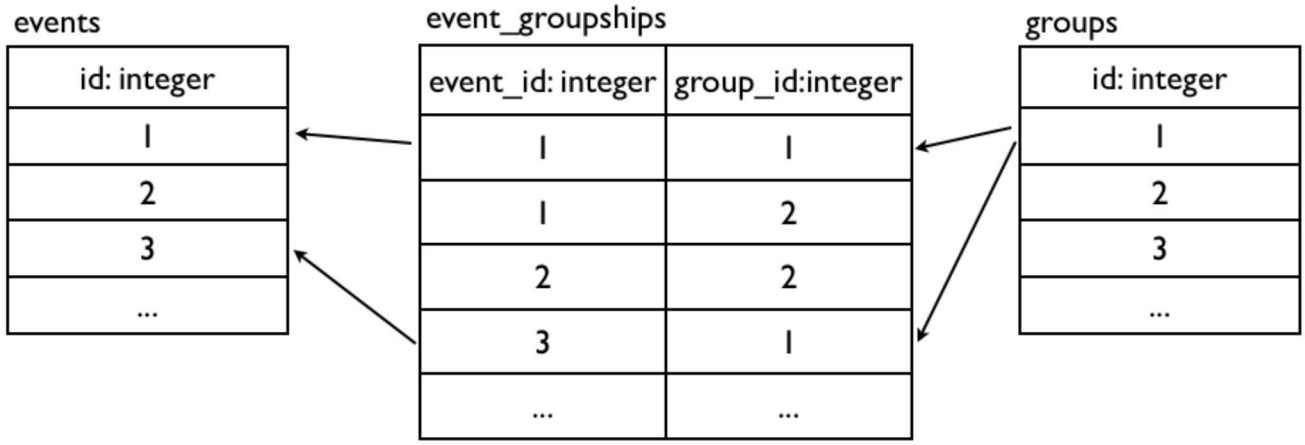
學到這裡，還記得上一章建立的 `Category` 嗎? 它也要跟 `Event` 是一對多的關係，讓我們補上程式吧：

```
class Category < ActiveRecord::Base
    has_many :events
end

class Event < ActiveRecord::Base
  belongs_to :category
  # ...
end
```

## 多對多關聯 *many-to-many*




另一種常見的關聯模式則是多對多，一筆資料互相擁有多筆資料，例如一個 *Event* 有多個 *Group*，一個 *Group* 有多個 *Event*。多對多關聯的實作必須多一個額外關聯用的資料表 (又做作 *Join table*)，讓我們來建立 *Group Model* 和關聯用的 *EventGroupship Model*，其中後者定義了兩個 *Foreign Keys*：

```
rails g model group name:string
rails g model event_groupship event_id:integer group_id:integer
```

執行 `bundle exec rake db:migrate` 產生這兩個資料表。

分別編輯 *app/models/event.rb*、*app/models/group.rb* 和 *app/models/event_groupship.rb*：

```
class Event < ActiveRecord::Base
    has_many :event_groupships
    has_many :groups, :through => :event_groupships
end

class EventGroupship < ActiveRecord::Base
    belongs_to :event
    belongs_to :group
end

class Group < ActiveRecord::Base
    has_many :event_groupships
    has_many :events, :through => :event_groupships
end
```

這個 *Join table* 筆者的命名習慣會是 *ship* 結尾，用以凸顯它的關聯性質。另外，除了定義 *Foreign Keys* 之外，你也可以自由定義一些額外的欄位，例如記錄是哪位使用者建立關聯。

*blongs_to* 和 *has_many* 我們見過了，這裡多一種 *has_many :through* 方法，可以神奇地把 *Event* 和 *Group* 關聯起來，讓我們進入 `rails console` 實際操作資料庫看看：

### 範例，建立雙向關聯記錄:

```
g = Group.create( :name => 'ruby taiwan' )
e1 = Event.first
e2 = Event.create( :name => 'ruby tuesday' )
EventGroupship.create( :event => e1, :group => g )
EventGroupship.create( :event => e2, :group => g )
g.events
e1.groups
e2.groups
```

> *Rails* 還有一種舊式的 *has_and_belongs_to_many* 方法也可以建立多對多關係，不過已經很少使用，在此略過不提。

## 關連的參數

以上的關聯方法 *blongs_to* 、 *has_one* 和 *has_many* 都還有一些可以客製的參數，讓我們來介紹幾個常用的參數，完整的參數請查詢 *API* 文件：

### class_name

可以變更關聯的類別名稱，例如：

```
class Event < ActiveRecord::Base
    belongs_to :manager, :class_name => "User" # 外部鍵是user_id
end
```

### foreign_key

可以變更 *Foreign Key* 的欄位名稱，例如改成 `manager_id`：

```
class Event < ActiveRecord::Base
    belongs_to :manager, :class_name => "User", :foreign_key => "manager_id"
end
```

### order

`has_many` 可以透過 `:order` 參數指定順序：

```
class Event < ActiveRecord::Base
    has_many :attendees, :order => "id desc"
    #...
end
```

### dependent

可以設定當物件刪除時，也會順便刪除它的 `has_many` 物件：

```
class Event < ActiveRecord::Base
    has_many :attendees, :dependent => :destroy
 end
```

`:dependent` 可以有三種不同的刪除方式，分別是：

- `:destroy` 會執行 *attendee* 的 *destroy* 回呼
- `:delete` 不會執行 *attendee* 的 *destroy* 回呼
- `:nullify` 這是預設值，不會幫忙刪除 *attendee*

> 要不要執行 *attendee* 的刪除回呼效率相差不少，如果需要的話，必須一筆筆把 *attendee* 讀取出來變成 *attendee* 物件，然後呼叫它的 *destroy* 。如果用 `:delete` 的話，只需要一個 *SQL* 語句就可以刪除全部 *attendee* 。

## joins 和 includes 查詢

針對 *Model* 中的 `belongs_to` 和 `has_many` 關連，可以使用 `joins`，也就是 *INNER JOIN*

```
Event.joins(:category)
# SELECT "events".* FROM "events" INNER JOIN "categories" ON "categories"."id" =
"events"."category_id"
```

可以一次關連多個：

```
Event.joins(:category, :location)
```

*joins* 主要的用途是來搭配 *where* 的條件查詢：

```
Event.joins(:category).where("categories.name is NOT NULL")
# SELECT "events".* FROM "events" INNER JOIN "categories" ON "categories"."id" =
"events"."category_id" WHERE (categories.name is NOT NULL)
```

透過 *joins* 抓出來的 *event* 物件是沒有包括其關連物件的。如果需要其關連物件的資料，會使用 `includes`。*includes* 可以預先將關連物件的資料也讀取出來，避免 *N+1* 問題(見效能一章)

```
Event.includes(:category)
# SELECT * FROM events
# SELECT * FROM categories WHERE categories.id IN (1,2,3...)
```

同理，也可以一次載入多個關連：

```
Event.includes(:category, :attendees)
# SELECT "events".* FROM "events"
# SELECT "categories".* FROM "categories" WHERE "categories"."id" IN (1,2,3...)
# SELECT "attendees".* FROM "attendees" WHERE "attendees"."event_id" IN (4, 5, 6,
7, 8...)
```

`includes` 方法也可以加上條件：

```
Event.includes(:category).where( :category => { :position => 1 } )
```

## 更多線上資源

- A Guide to Active Record Associations http://guides.rubyonrails.org/association_basics.html