

# Webエンジニアコース フルタイム

---

Ruby on Railsアソシエーション編



DIVE INTO CODE



# はじめに

---

この講義の様子は、今後の運営改善に役立てるため、録画をいたします。

映像は、弊社YouTubeアカウントに**限定公開**で保存され、一般に公開されることはありません。

ご理解ご協力のほど、よろしくお願いいたします。



# 構成

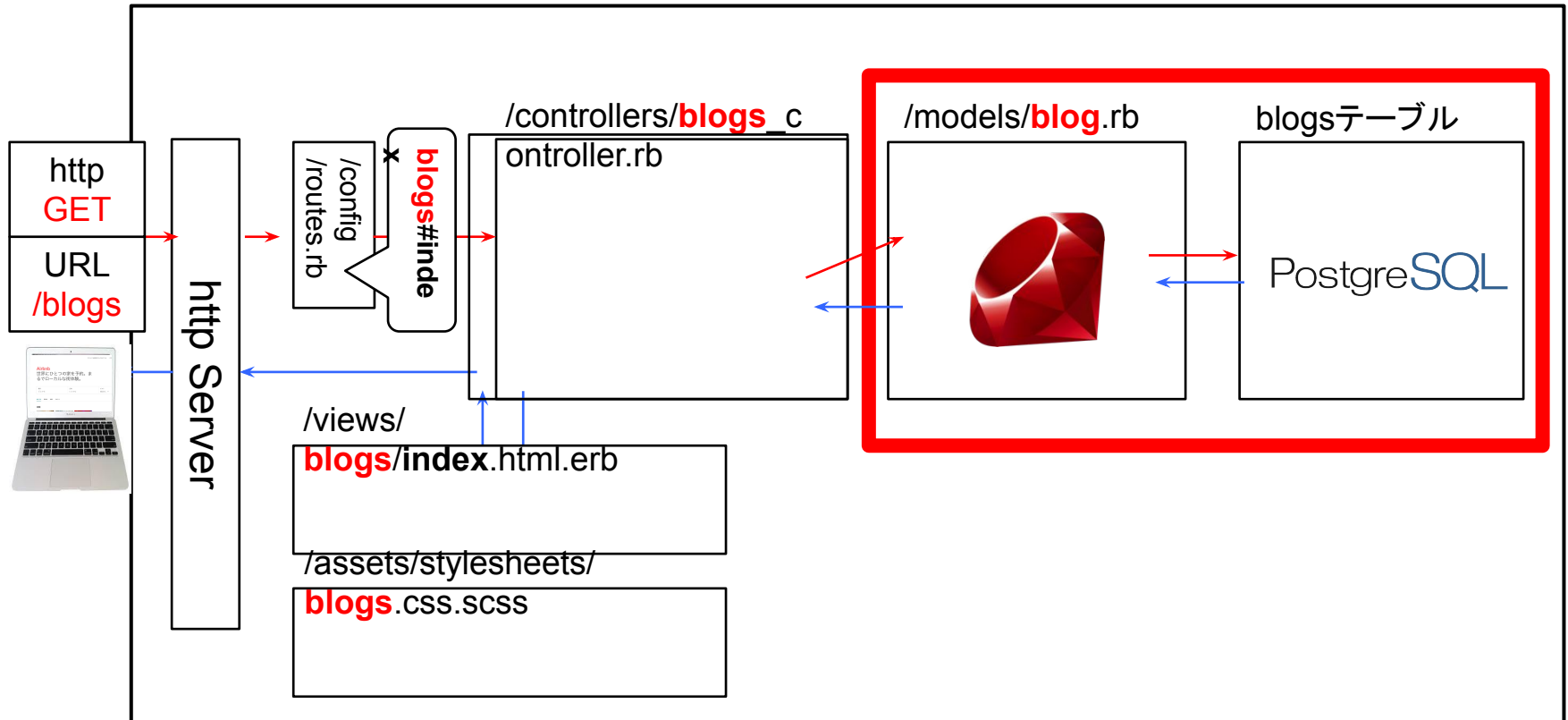
---

1. テーブルとアソシエーション
2. グループワーク
3. まとめ



# テーブルとアソシエーション

テーブルは、RDBMS内のデータを格納する一領域。アソシエーションは、Railsのモデルに関連する言葉。





# テーブルとアソシエーション

アソシエーションは、Ruby on Rails の複数モデル間の関係をあらわすことができる仕組みのこと。

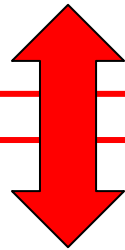
## Ruby on Rails

```
class Blog
```

```
end
```

```
class User
```

```
end
```



## RDBMS

blogs

users

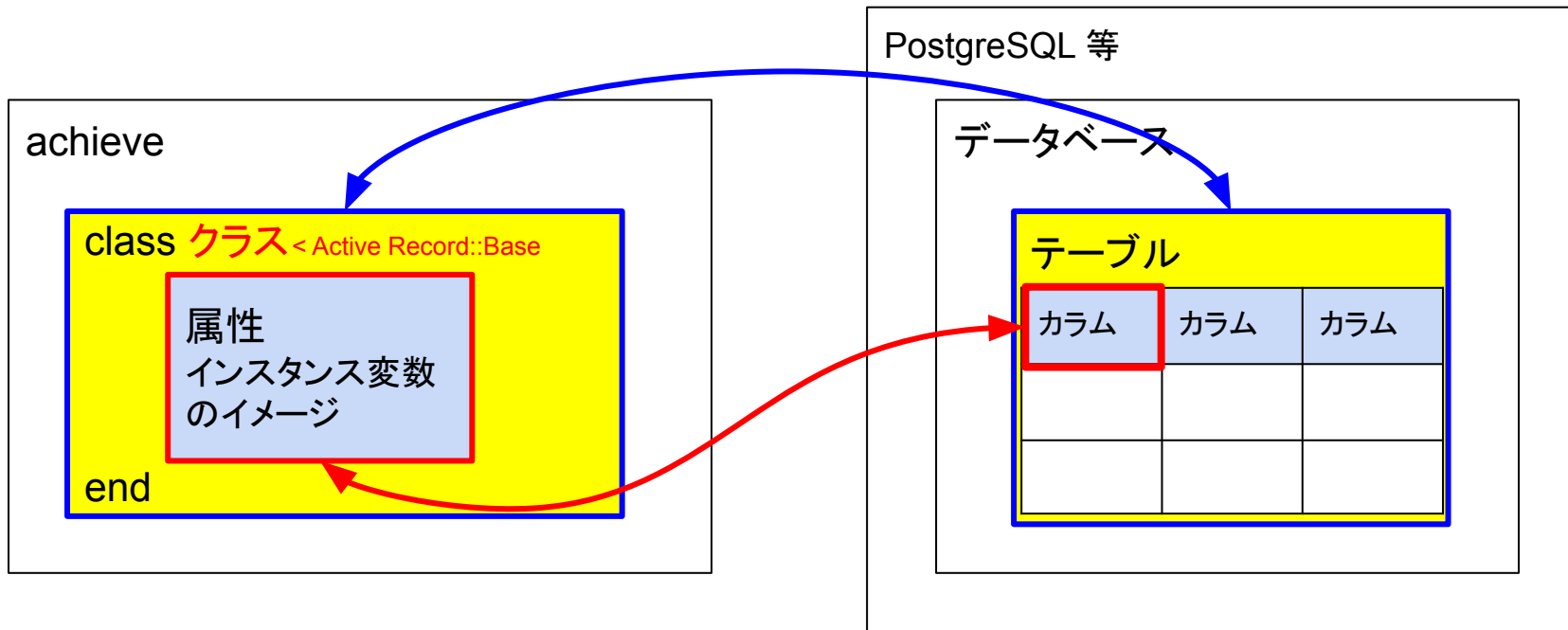


# テーブルとアソシエーション

オブジェクト指向言語のクラスは DBMS のテーブル、メソッドはテーブルのカラムに 1対1 に対応させて”翻訳”している。

## Ruby on Rails

## RDBMS





# テーブルとアソシエーション

O/Rマッパーは、Ruby と SQL を通訳する。

## Ruby on Rails

```
class Blog
```

```
end
```

**Blog.all**

## RDBMS

blogs

id	title	content

**SELECT \* FROM blogs;**



# テーブルとアソシエーション

SQL から Ruby に通訳された結果は、そのモデルクラスのインスタンスとなり、変数に代入して使われる。

## Ruby on Rails

```
@blogs = Blog.all
```

実行結果

```
=> #<ActiveRecord::Relation [#<Blog id: 9, title: "DIVE INTO", content: "9", created_at: "2017-09-22 09:11:57", updated_at: "2017-09-22 10:41:38">, #<Blog id: 10, title: "DIVE INTO", content: "10", created_at: "2017-09-22 10:35:50", updated_at: "2017-09-22 10:41:53">, #<Blog id: 11, title: "DIVE INTO ", content: "11", created_at: "2017-09-22 10:36:14", updated_at: "2017-09-22 10:42:05">]>
```

## RDBMS

```
SELECT * FROM blogs;
```

実行結果

id	title	content
9	DIVE INTO	9
10	DIVE INTO	10
11	DIVE INTO	11
(3 rows)		





# テーブルとアソシエーション

【ワーク】rails c と rails db で実行しよう。

## Ruby on Rails

rails c

Blog.all

Rubyの実行結果を確認しよう

exit

## RDBMS

rails db

SELECT \* FROM blogs;

SQLの実行結果を確認しよう

\q



# テーブルとアソシエーション

モデルの基本は、モデル 1つに対してテーブル 1つが対応する。そのため異なる概念のデータは異なるテーブル、モデルで表す。

## Ruby on Rails

```
class Blog  
end
```

**Blog.all**

```
class User  
end
```

**User.all**

## RDBMS

blogs

**SELECT \* FROM blogs;**

users

**SELECT \* FROM users;**



# テーブルとアソシエーション

SQLの結果とRailsメソッドの結果は見た目は異なるが内容は一緒。インスタンス化された内容を Rails内で使う。

## Ruby on Rails

@blogs = **Blog.all**

```
=> #<ActiveRecord::Relation [#<Blog id: 9, title: "DIVE INTO ",
content: "9", created_at: "2017-09-22 09:11:57", updated_at: "2017-09-22 10:41:38">, #<Blog id: 10, title: "DIVE INTO ",
content: "10", created_at: "2017-09-22 10:35:50", updated_at: "2017-09-22 10:41:53">, #<Blog id: 11, title: "DIVE INTO ",
content: "11", created_at: "2017-09-22 10:36:14", updated_at: "2017-09-22 10:42:05">]>
```

@users = **User.all**

```
=> #<ActiveRecord::Relation [#<User id: 39,
email: "J0zYXWcg.Ffal/ZI40kfaZ9SxCAk5X...", reset_password_token: " ",
n_count: 1, current_sign_in_at: "2016-05-20 00:26:36",
last_sign_in_at: "2016-05-20 00:26:36", last_sign_in_ip: "192.168.1.1">]>
```

## RDBMS

SELECT \* FROM blogs;

id	title	content
9	DIVE INTO	9
10	DIVE INTO	10
11	DIVE INTO	11

(3 rows)

SELECT \* FROM users;

id	email
1	3305717336-twitter@example.com

(1 row)



# テーブルとアソシエーション

関連付けは、本来RDBMS内で行う必要がある。テーブルに関連付け用のカラムを追加することで実現する。

blogsテーブル

usersテーブル

関連付けなし

blogs		
id	title	content

users

users		
id	email	password

関連付けあり

blogs			
id	title	content	user_id
1	DIVE	INTO 1	1
2	DIVE	INTO 2	2

users

users		
id	email	password
1	***	*****
2	***	*****



# テーブルとアソシエーション

主となるテーブルにある識別用カラムを主キー、それに関連づく副となるテーブルの識別用カラムを外部キーと言う。

主となるテーブル

users		
id	email	password
1	***	*****
2	***	*****

主キー  
Primary Key

副となるテーブル

blogs			
id	title	content	user_id
1	DIVE	INTO 1	1
2	DIVE	INTO 2	1

外部キー  
Foreign Key



# テーブルとアソシエーション

複数テーブル間のデータを関連付け(結合し)て取得するためには、結合文 SQL を使う必要がある。

主となるテーブル

users		
id	email	password
1	***	*****
2	***	*****

副となるテーブル

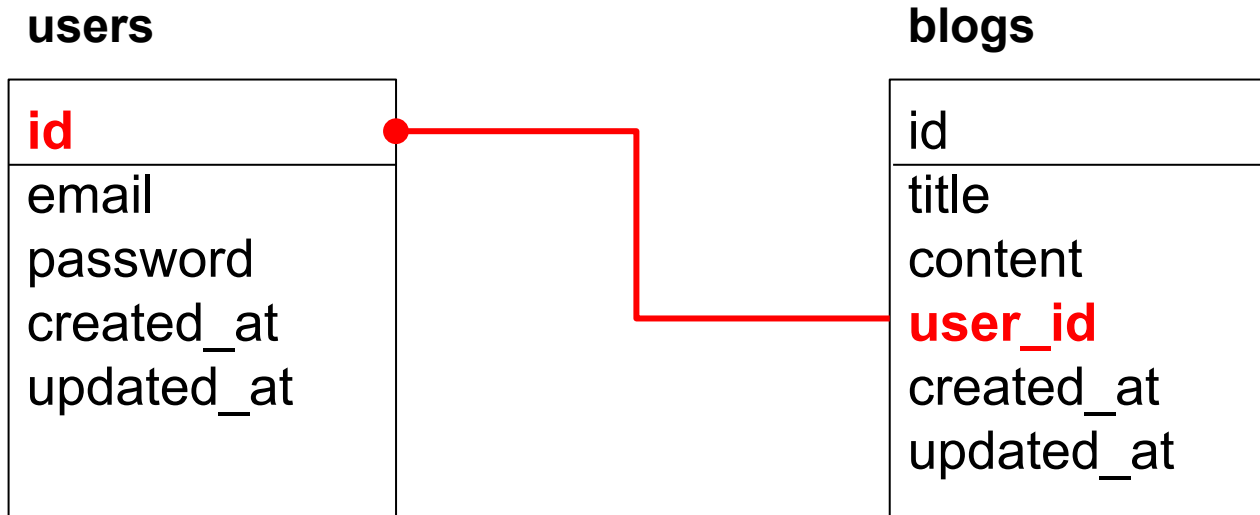
blogs			
id	title	content	user_id
1	DIVE	INTO 1	1
2	DIVE	INTO 2	2

```
SELECT title, users.id  
FROM blogs, users  
WHERE blogs.user_id = users.id;
```



# テーブルとアソシエーション

テーブル間の相関関係を図式化する手法「ERD (Entity-Relationship Diagram)」を知っておこう。





# テーブルとアソシエーション

結合文 SQL を直接記述したり、RDBMSごとの書き方の違いを意識せずに関連づけられる仕組みがアソシエーション。

## Ruby on Rails

アソシエーション

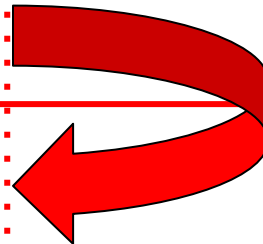
```
class Blog  
end
```

```
class User  
end
```

## RDBMS

blogs

users







# テーブルとアソシエーション

それぞれのモデルクラスに相手を参照するためのコードを記述すると関連付けられる。

## 主となるモデルクラス

```
class User
  has_many :blogs
```

## 副となるモデルクラス

```
class Blog
  belongs_to :user
```

上記は、下記オプションの指定は不要  
class\_name: **Blog**  
foreign\_key: **blogs.user\_id**

上記は、下記オプションの指定は不要  
class\_name: **User**  
foreign\_key: **blogs.user\_id**

公式ドキュメント: ActiveRecord::Associations::ClassMethods

<http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html>



# テーブルとアソシエーション

Rails内部でアソシエーション経由でデータを取得する場合、定義したアソシエーション名をメソッドとして使う。

## アソシエーションの定義

```
class User  
  has_many :blogs
```

```
class Blog  
  belongs_to :user
```

## 使い方の例

```
@user = User.find(1)  
@user.blogs
```

```
@blog = Blog.find(1)  
@blog.user
```

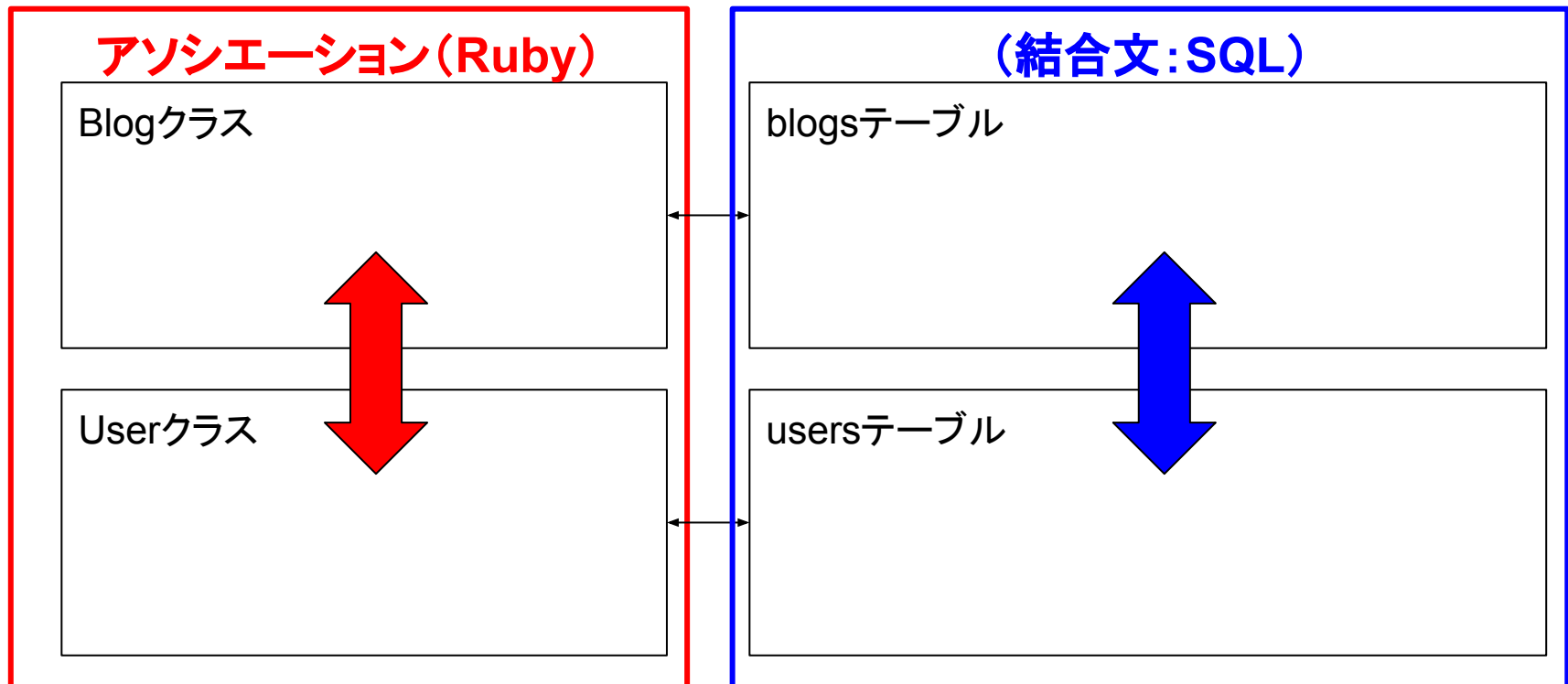


# テーブルとアソシエーション

アソシエーションは、データベース内のテーブルの外部結合を SQLを書かずに Rubyで実装できるようにする仕組み。

## Ruby on Rails内のクラス

## データベース内のテーブル





# ワーク

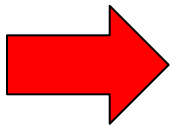
## ペアプログラミング



「二人一組になり、**一つの画面・キーボードを共有して実装する**」

1人がドライバとしてコードを書き、もう一人がオブザーバとしてコードを見ながらアドバイスをすることで知識の共有を促進します。

- 現役エンジニアに学ぶ「ペアプログラミング実践中に重要なポイントとは？」



挨拶をするかのごとく、自然とやろう



# グループワーク

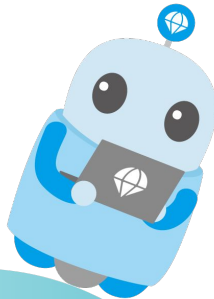
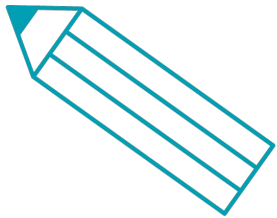
**テーマ:「投稿者の名前が表示されるブログアプリケーションを開発してください。」**

## 【流れ】

1. ペアプロで取り組むこと
2. BlogのCRUD機能、Userの新規登録・ログイン機能必須
3. BlogとUserを紐づけること
4. ブログお気に入り機能を実装すること
5. お気に入りをしたら、お気に入りボタンの表示が「お気に入り解除」に変わることに
6. 自分のブログには、お気に入りボタンは表示されないこと
7. 自分がお気に入りしたブログ一覧を、自分のshowページで見られること  
(新しいルーティングを組む必要はない)
8. 要件通りの実装ができたなら、GitHubにプッシュし、SlackにURL共有する
9. 余力があれば、追加課題に取り組んでみることに

追加課題詳細は、次のスライド👉

# 追加課題1



ちょっと難しいよ。サンプル  
解答は24ページから用意  
してあるよ。





## 追加課題1

ログインしているユーザの**全投稿に対して、お気に入りされた合計数**は、アソシエーションを使うとどのような実装になるか？

ゴール：

ユーザのShow画面で、合計数を表示する。

前提：

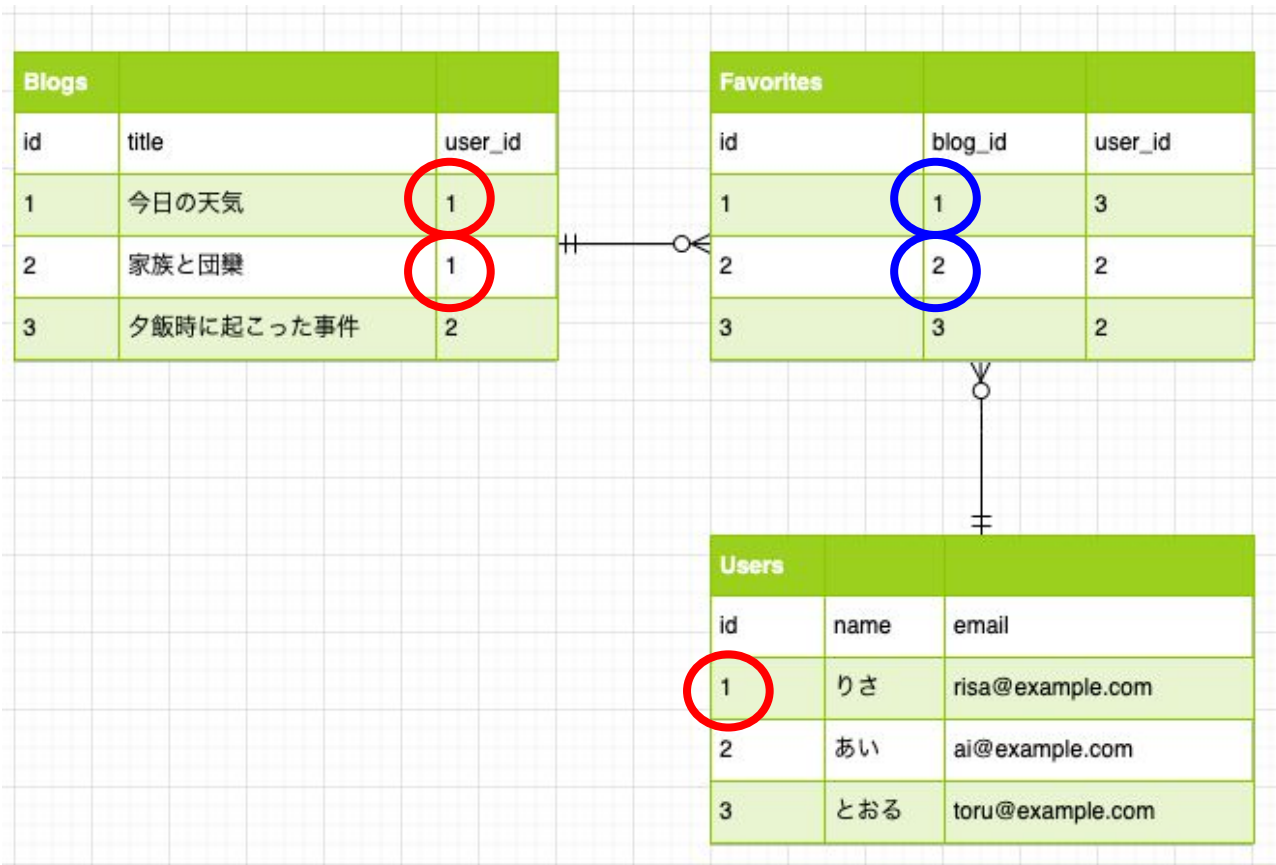
BlogとUser、Favoriteのアソシエーションが多対多で組まれていること。



# グループワーク

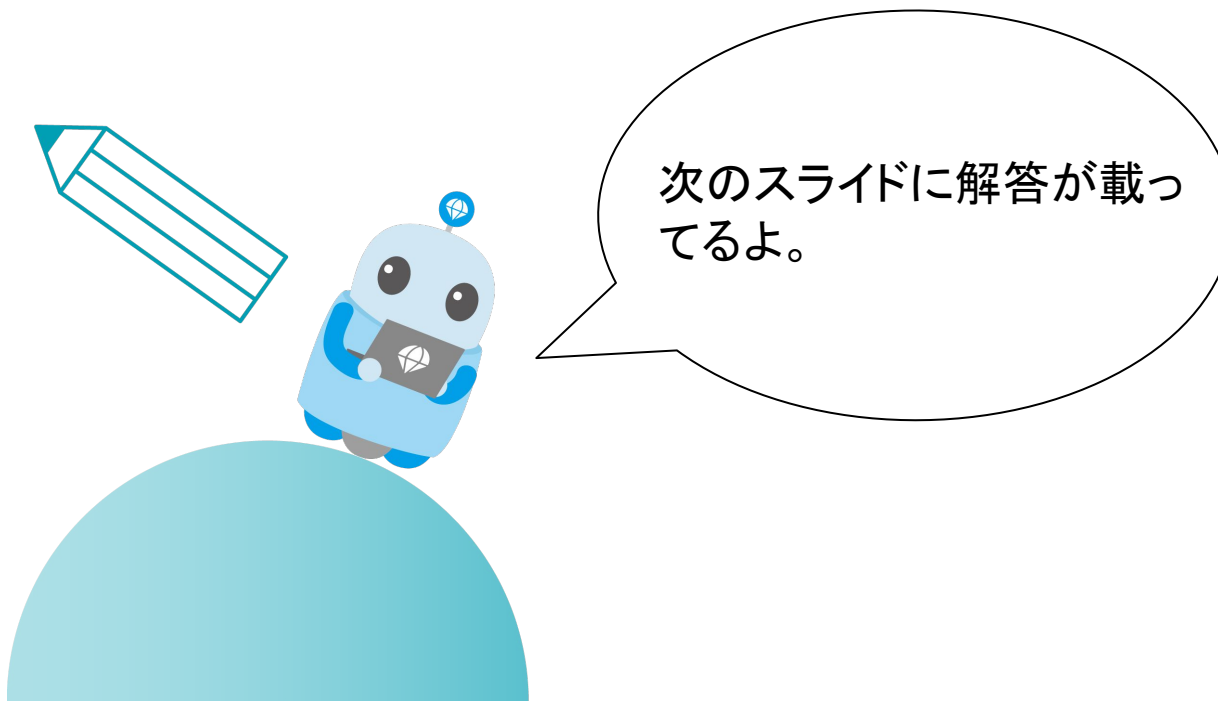
ヒント:ER図を書いて、整理してみよう。

例:りさが投稿した全ブログを対象に、お気に入りされた総数を数える。(青丸の数を数えます)





# 追加課題1のサンプル解答





## 追加課題1のサンプル解答

**Usersテーブルからアクセスした場合**

`controllers/users/users_controller.rb`

割愛

```
def show
```

```
  @user = User.find(params[:id])
```

```
  @blogs = @user.favorites
```

```
  @login_user_blogs = current_user.blogs ◀ 追記
```

```
end
```

最後の`@login_user_blogs = current_user.blogs`で、ログインユーザーが投稿したblogを取得しています。



## 追加課題1のサンプル解答

### Usersテーブルからアクセスした場合

views/users/show.html.erb

割愛

```
<% count = 0 %>
```

```
<% @login_user_blogs.each do |blog| %>
```

```
  <% count += blog.favorites.count if  
  blog.favorites.present? %>
```

```
<% end %>
```

```
<p>お気に入りされた合計数: <%= count %></p>
```



## 追加課題1のサンプル解答

### Blogsテーブル側からアクセスした場合

controllers/users/users\_controller.rb

省略

```
def show
```

```
  @user = User.find(params[:id])
```

```
  @pictures = @user.favorites
```

```
  @blog_ids = Blog.where(user_id: current_user.id).ids ◀ 追  
記
```

```
end
```

最後の@blog\_ids = Blog.where(user\_id: current\_user.id).idsで、ログインユーザーが投稿したブログのidを配列で取得しています。



## 追加課題1のサンプル解答

### Usersテーブルからアクセスした場合

views/users/show.html.erb

割愛

```
<% count = 0 %>
```

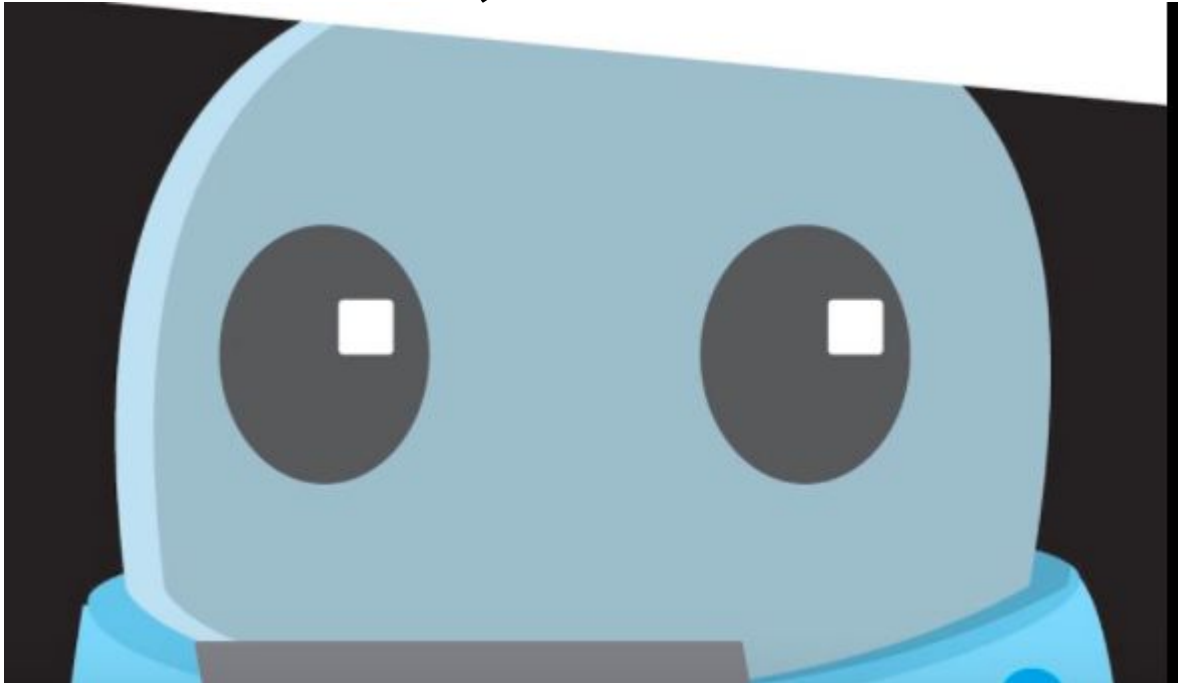
```
<% @blog_ids.each do |picture_id| %>
```

```
  <% count += Favorite.where(picture_id:  
picture_id).count if Favorite.where(picture_id:  
picture_id).present? %>
```

```
<% end %>
```

```
<p>お気に入りされた合計数: <%= count %></p>
```

だがしかし・・・





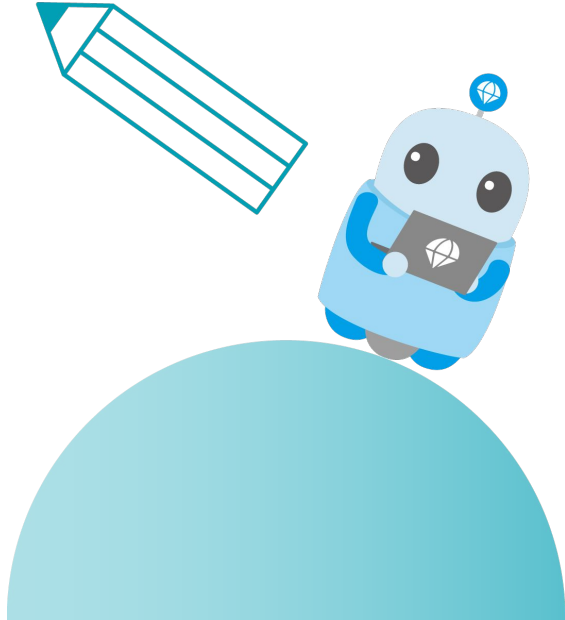
## グループワーク

このやり方ですと、データが億万存在した場合、億万回eachが回ることになります。

データは一回で値が取得できるようにしておくと、速度が上がります。

そこで、DBのテーブルを結合し、数を数えるという方針に切り替えてみましょう。(DBの言語であるSQLは、数を数えるのが得意です。)

## 追加課題2



DBのテーブルを結合し、数を数えてみよう。





# グループワーク

## ヒント1:

- ①FavoritesテーブルにBlogsテーブルを結合させる(赤枠)
- ②user\_id = 1 のデータを取得し、数をカウントする(青枠)
- ③まずは、どんなSQL文になるか、SQLから考えてみる
- ④railsに合うようにSQL文を書き換える

Favorites			Blogs			Users		
id	user_id	blog_id	id	title	user_id	id	name	email
1	3	1	1	今日の天気	1	1	りさ	risa@dic.com
2	2	2	2	家族と団欒	1	2	あい	ai@dic.com
3	2	3	3	夕飯時に起こった事件	2	3	さとり	satoru@dic.com

※ 画像は、りささんの全投稿に対してのお気に入りの総合計数を出そうとしています。

# 追加課題2 SQLのサンプル解答





```
SELECT COUNT(*) FROM favorites  
JOIN blogs ON favorites.blog_id = blogs.id  
JOIN users ON blogs.user_id = users.id  
WHERE users.id = current_user.id
```

※ usersテーブルも結合させています。

上記のSQLをrails用に書き換えます。ヒント👉



# グループワーク

---

## ヒント2:

**joinsメソッドを使ってテーブルを結合させます。**

railsガイド: [https://railsguides.jp/active\\_record\\_querying.html#joins](https://railsguides.jp/active_record_querying.html#joins)

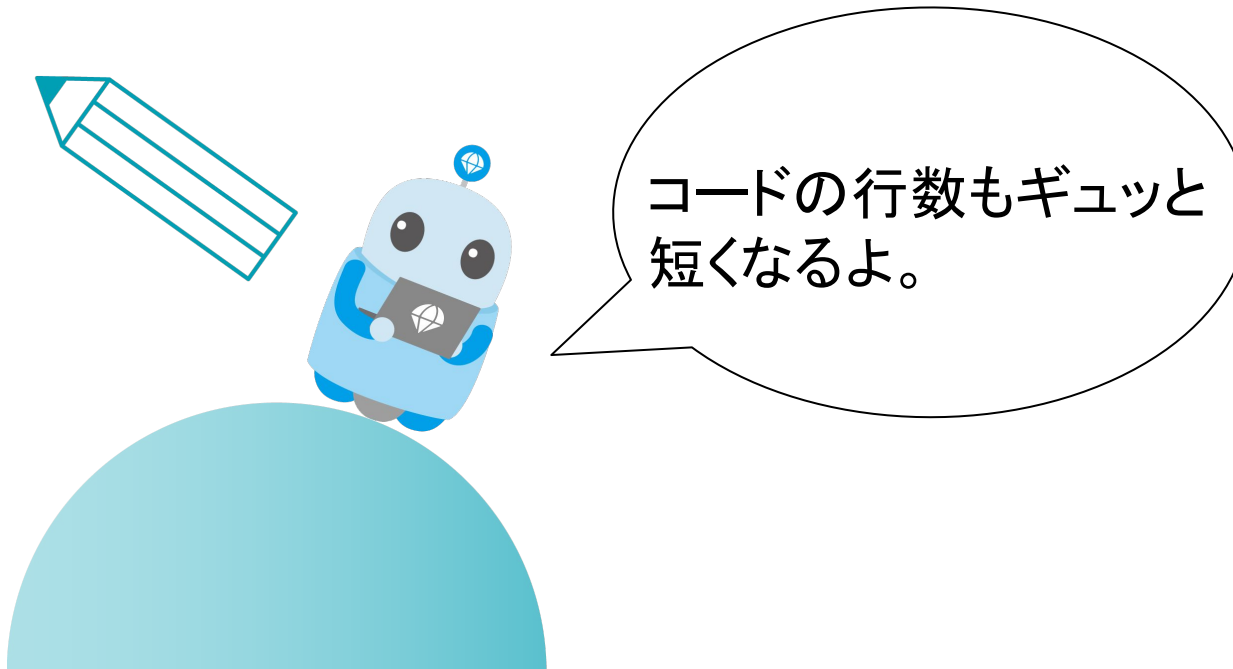
## ヒント3:

**whereメソッドの後に、ユーザのidを持っているテーブルを指定しないとエラーになります。**

railsガイド

: [https://railsguides.jp/active\\_record\\_querying.html#%E3%83%8D%E3%82%B9%E3%83%88%E3%81%97%E3%81%9F%E9%96%A2%E9%80%A3%E4%BB%98%E3%81%91%E3%82%92%E7%B5%90%E5%90%88%E3%81%99%E3%82%8B-%E8%A4%87%E6%95%B0%E3%83%AC%E3%83%99%E3%83%AB](https://railsguides.jp/active_record_querying.html#%E3%83%8D%E3%82%B9%E3%83%88%E3%81%97%E3%81%9F%E9%96%A2%E9%80%A3%E4%BB%98%E3%81%91%E3%82%92%E7%B5%90%E5%90%88%E3%81%99%E3%82%8B-%E8%A4%87%E6%95%B0%E3%83%AC%E3%83%99%E3%83%AB)

## 追加課題2 サンプル解答(完成版)





controllers/users/users\_controller.rb

**BlogsテーブルとUsersテーブルを結合する場合**

```
@favorited_total_counts = Favorite.joins(blog:  
:user).where(users: {id: current_user.id}).count
```

**Blogsテーブルのみ結合する場合(今回はこちらでok)**

```
@favorited_total_counts =  
Favorite.joins(:blog).where(blogs: {user_id:  
current_user.id}).count
```



# グループワーク

---

view/users/show.html.erb

```
<%= @favorited_total_counts %>
```

コードがとってもスッキリしました！



# まとめ

アソシエーションは、Ruby on Rails の複数モデル間の関係をあらわすことができる仕組みのこと。

- SQLの結果とRailsメソッドの結果は見た目は異なるが内容は一緒。インスタンス化された内容を Rails内で使う。
- データが格納される大元となるRDBMS内で関連付けを行う必要がある。テーブルに関連付け用のカラムを追加する。
- 主となるテーブルにある識別用カラムを主キー、それに関連づく副となるテーブルの識別用カラムを外部キーと言う。
- アソシエーションは、データベース内のテーブルの外部結合を SQLを書かずに Rubyで実装できるようにする仕組み。