

---

RAILS



## EXERCICE PRATIQUE SUR LES ASSOCIATIONS

### PARTIE II – ASSOCIATIONS 1-n & N-N

*ELS – MARS 2016*

---

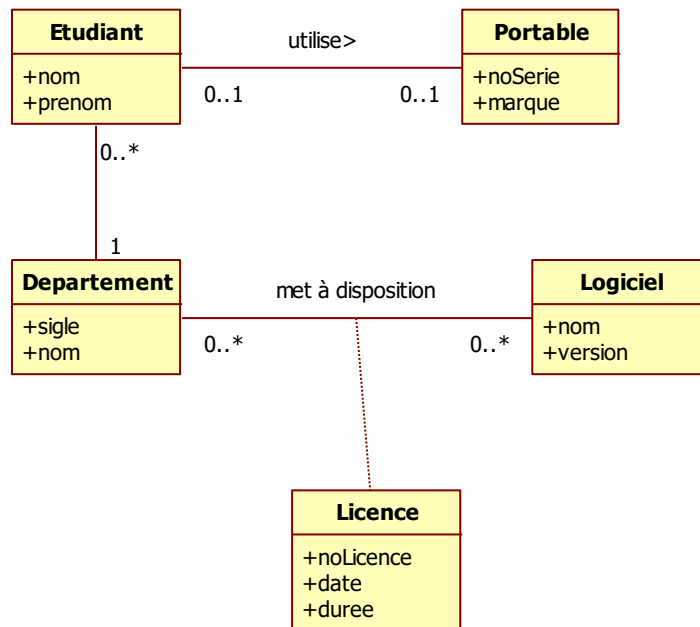
<b>1</b>	<b>Modèle de la BD (Rappel)</b>	<b>3</b>
<b>2</b>	<b>Association 1-n : Etudiants – Départements</b>	<b>4</b>
2.1	Créer la ressource <code>Departement</code>	4
2.2	Rajouter la <code>foreign_key</code> <code>departement_id</code> dans la table <code>etudiants</code>	4
2.3	Adapter les modèles pour prendre en compte l'association 1-N	4
2.4	Compléter l'interface utilisateur	5
<b>3</b>	<b>Association N-N: Departements – Logiciels</b>	<b>6</b>
3.1.1	Compléter l'interface utilisateur	8
3.1.2	Accorder une nouvelle licence pour un département donné	9

---

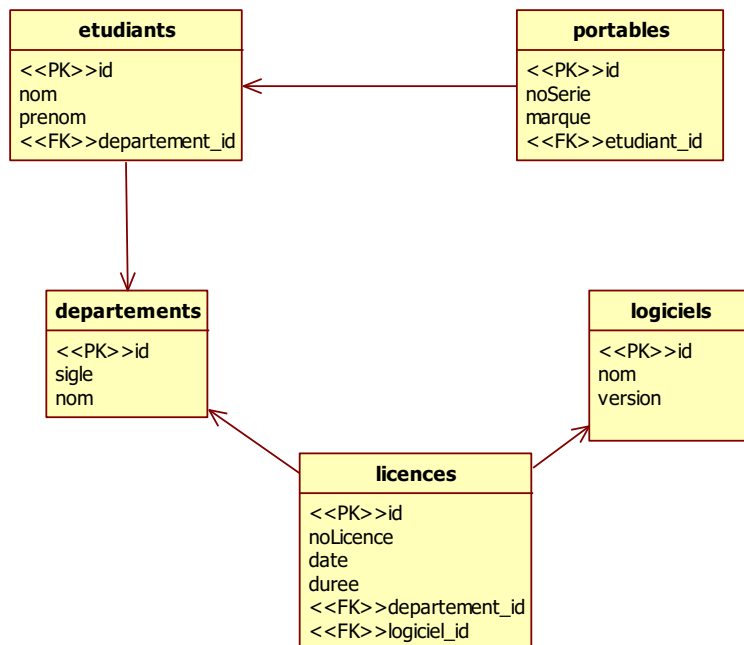


# 1 MODELE DE LA BD (RAPPEL)

## Modèle conceptuel



## Modèle relationnel



## 2 ASSOCIATION 1-N : ETUDIANTS – DEPARTEMENTS

Cette partie du laboratoire va s'intéresser la mise en œuvre de la ressource « `Département` » et de l'association 1-n entre étudiants et départements.

### 2.1 CREER LA RESSOURCE `DEPARTEMENT`

En utilisant la même démarche que pour les étudiants et portables, créer la ressource `Département` (contrôleur, modèle et vues) et la table correspondante. Peupler cette table avec au moins deux départements TIC & TIN (en utilisant la techniques des « fixtures »).

### 2.2 RAJOUTER LA `FOREIGN_KEY DEPARTEMENT_ID` DANS LA TABLE `ETUDIANTS`

👉 Rajouter cette `foreign_key` par le biais d'un nouveau fichier de migration, comprenant uniquement un « `add_column` ».

*Les vues associés à la ressource `Etudiant` – déjà complétés par nos soins dans la première partie du labo – devront être adaptées pour tenir compte de la nouvelle colonne. On le fera plus tard, « manuellement », au fur et à mesure des besoins..*

👉 Ajuster les données de test associées aux étudiants (`etudiants.yml`) de manière à prendre en compte les départements.

### 2.3 ADAPTER LES MODELES POUR PRENDRE EN COMPTE L'ASSOCIATION 1-N

👉 On peut d'ores et déjà adapter les modèles `Etudiant` et `Département` de manière à prendre en compte l'association 1-N. Faites en sorte également à vous assurer que le champ « `sigle` » d'un département est un champ requis.

😊 **Solution : Voir étape 1**

## 2.4 COMPLETER L'INTERFACE UTILISATEUR

### DU COTE « ETUDIANTS »

En premier lieu, compléter la fin du contrôleur `etudiants_controller.rb` en autorisant le nouveau paramètre `:departement_id` qui viendra du remplissage des formulaires:

```
[...]  
private  
  [...]
  
  # Never trust parameters from the scary internet, only allow the white  
  # list through.  
  def etudiant_params  
    params.require(:etudiant).permit(:nom, :prenom, :departement_id)  
  end  
end
```

Puis, adaptez les vues en faisant le nécessaire au niveau du contrôleur

- Afficher son département

### Etudiants

Prenom	Nom	Departements	
Alain	Martin	TIN	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
Claude	Dupont	***	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
Mat	Maillol	TIC	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>
Marc	Lenfant	***	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[New etudiant](#) [MENU PRINCIPAL](#)

### "Show" d'un etudiant"

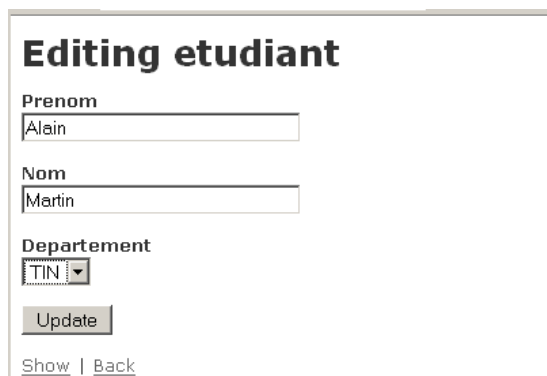
**Prenom:** Alain

**Nom:** Martin

**Departement:** TIN

[Edit](#) | [Back](#)

- Pouvoir changer le département dans lequel il est inscrit (liste déroulante)



**Editing etudiant**

Prenom  
Alain

Nom  
Martin

Departement  
TIN

Update

[Show](#) | [Back](#)

😊 *Solution : Voir étape 2*

## DU COTE « DEPARTEMENTS »: AFFICHER LA LISTE DE SES ETUDIANTS INSCRITS



← → ↻ 127.0.0.1:3000/departements/1

Apps Rails start Oops! Google Chro... Apps

**Sigle:** TIC

**Nom:** Techno Info

**Etudiants inscrits**  
Alain Martin  
Claude Dupont

[Edit](#) | [Back](#)

[Gestion des portables](#)  
[Gestion des etudiants](#)  
[Gestion des departements](#)

😊 *Solution : Voir étape 3*

## 3 ASSOCIATION N-N: DEPARTEMENTS — LOGICIELS

Le traitement des associations N-N s'opère de deux manières différentes suivant que la table d'associations possède ou non des attributs propres.

Deux cas possibles:

- **Cas No1**: Table d'associations sans attribut  
⇒ Il suffit alors d'utiliser des relations `has_and_belongs_to_many`
- **Cas No2**: Table d'associations avec attribut(s)  
⇒ On utilise plutôt des relations `has_many` avec clause « `through` »

Comme notre association contiendra des attributs, nous nous placerons dans le cas no 2.

👉 A faire par vous ?

1/ Créer la ressource `Logiciel` (génération de scaffold) avec les deux attributs `nom` et `version`

2/ Peupler la table `logiciels` avec des données de test (fixtures)

3/ Créer le nécessaire pour les licences, à savoir :

- Une table d'associations `licences`, avec deux clés étrangères (integer) et un attribut `no_licence` (string)
- Un modèle dans un fichier « `licence.rb` » pour cette table, le modèle étant nécessaire de manière à pouvoir - dans le code - accéder à l'attribut `no_licence`.
- Peupler cette table avec des données de test (fixtures) en créant un fichier supplémentaire dans les fixtures : `licences.yml`

Ces opérations peuvent être réalisées « à la main », pas à pas..

Toutefois, la commande `>rails generate model`, - et non pas `rails generate scaffold` -, met en place un certain nombre de choses qui permettent d'accélérer le travail :

- Génération d'un fichier modèle `Licence.rb`
- Génération du fichier de migration `create_licences` pour créer la table
- Création du fichier de fixtures `licences.yml`

Il restera à lancer la migration (`rake db:migrate`) et à compléter `licences.yml` avec des données de test.

En bref.. saisissez la commande :

```
>rails generate model departement_id :integer,  
                        logiciel_id :integer, no_licence :string
```

5/ Spécifier le modèle conceptuel (clauses `has_many`) dans les modèles correspondants

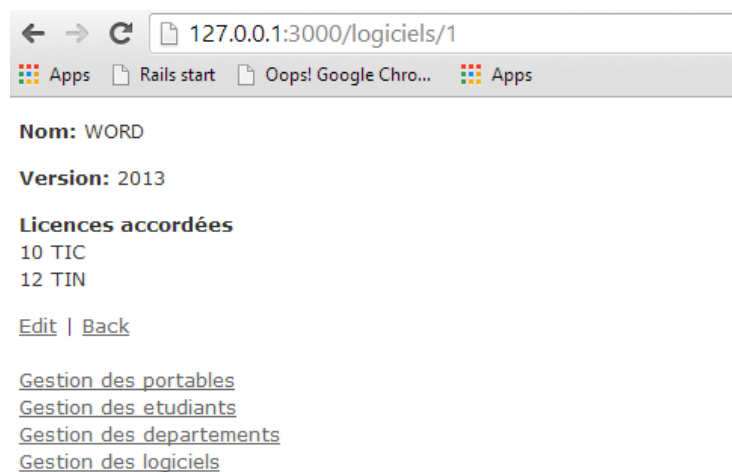
Voir les slides ORM et complétez les modèles `departement.rb`, `logiciel.rb` et `licence.rb`

😊 *Solution : Voir étape 4*

### 3.1.1 COMPLETER L'INTERFACE UTILISATEUR

Dans cette étape, nous allons utiliser les méthodes générées par Rails par le biais des clauses `has_and_belongs_to_many` (voir cours théorique, chap.2. Le modèle), et qui permettent d'accéder à tous les logiciels d'un département donné : `un_departement_donne.logiciels`, ou à tous les départements d'un logiciel donné : `un_logiciel_donne.departements` sans avoir à opérer des jointures.

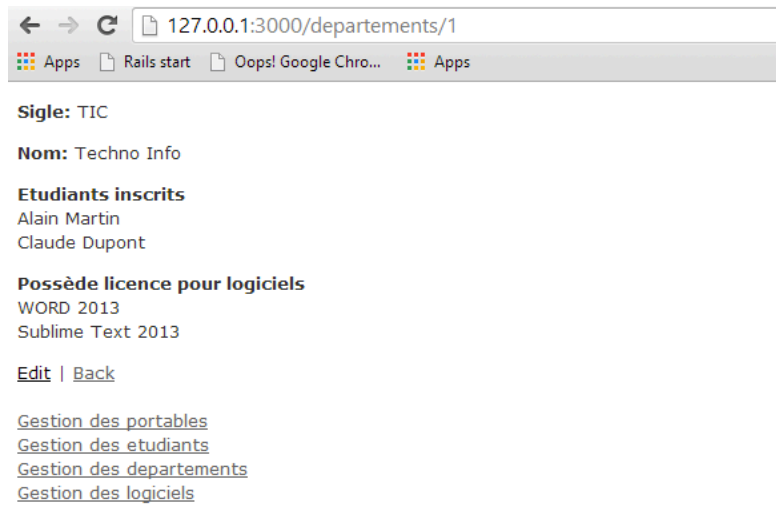
#### 1/ POUR UN LOGICIEL DONNE (SHOW) ⇒ AFFICHER TOUTES LES LICENCES ACCORDEES AVEC LEUR NO



😊 *Solution : Voir étape 5*



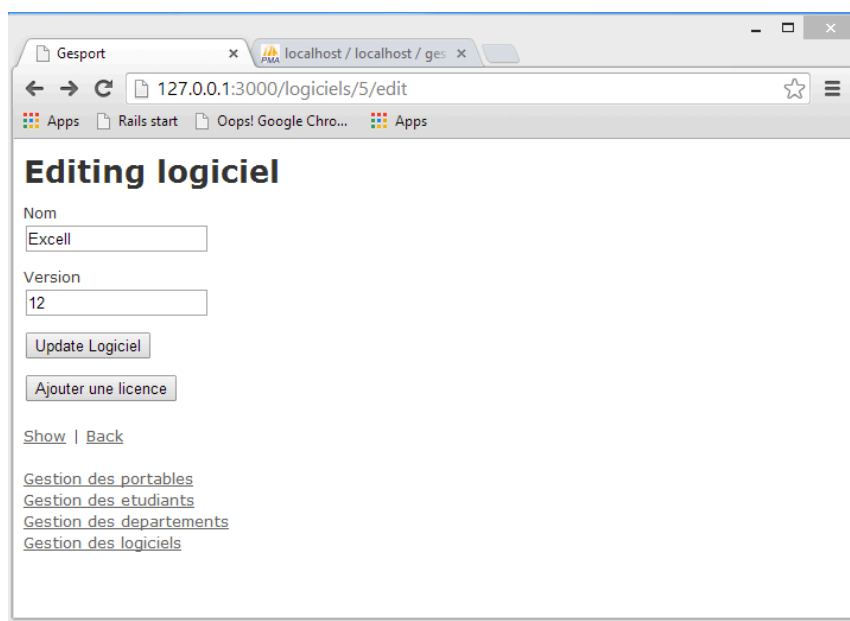
## 2/ POUR UN DEPARTEMENT DONNE (SHOW) ⇒ AFFICHER TOUS LES LOGICIELS POSSEDES



😊 **Solution : Voir étape 6**

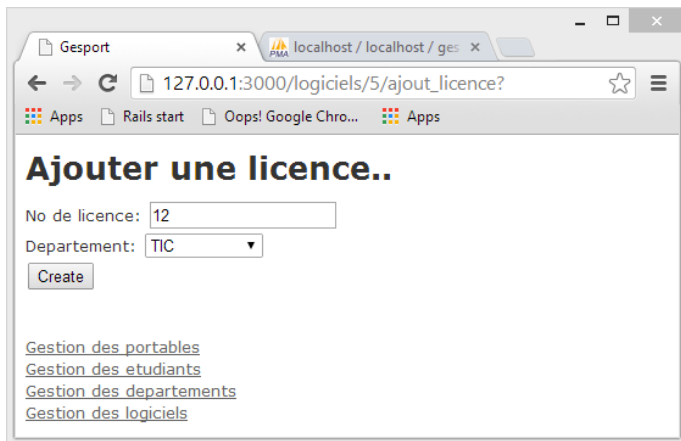
### 3.1.2 ACCORDER UNE NOUVELLE LICENCE POUR UN DEPARTEMENT DONNE

Dans l'action « edit » du contrôleur des logiciels, ajouter un bouton permettant de créer une licence (ajouter une action correspondante dans le même contrôleur).



Ce bouton aura pour effet de lancer une nouvelle action : `ajout_licence`, qui aura pour effet de proposer un formulaire comprenant :

- une liste déroulante permettant de choisir le département concerné
- un champ de saisie pour le no de licence



## 😊 Allons-y en trois étapes..

### Etape 1 : Ajouter le bouton « Ajout licence »

La pression du bouton lancera une requête vers une action **`ajout_licence`** du contrôleur des logiciels. L'url aura la forme `logiciels/id_logiciel/ajout_licence`, avec le verbe http «GET », ou `id_logiciel` correspond à l'identifiant du logiciel dont on veut rajouter une licence.

Pour ce faire, référez-vous à la 2<sup>ème</sup> méthode présentée en page 11 de « En voiture Part II ».

- a) Dans le fichier de Routage « `routes.rb` », rajouter une « **route de membre** » à la ressource logiciel :

```
resources :logiciels
```

devient :

```
resources :logiciels do
  member do
    get :ajout_licence
  end
end
```

Cela a pour effet de générer une méthode nommée `ajout_licence_logiciel_path` qui permet de générer l'url correspondant :

`ajout_licence` > nom de l'action

`logiciel` > s'appliquant à un logiciel (nom du contrôleur)

`path` > chemin d'accès

Cette méthode s'utilise **avec un paramètre** : le logiciel concerné

- b) L'ajout un bouton est une instruction identique à « `link_to` », en remplaçant `link_to` par « `button_to` »

```
<% button_to label_du_bouton, url %>
```

A la différence d'un `link_to`, la requête générée par `button_to` utilise le verbe http PUT en lieu et place du verbe GET. Il nous faut donc, soit changer la route de membre et écrire `put :ajout_licence` en lieu et place de `get :ajout_licence`, ou alors rajouter une option à la commande `button_to` pour lui demander d'utiliser le verbe « get ».

Ce qui donnerait :

```
<% button_to label_du_bouton, url, { :method => :get } %>
```



Essayez avec une action `ajout_licence`, rajoutée au contrôleur, vide pour l'instant. La pression du bouton devrait indiquer une erreur signalant qu'il manque la vue (le template) pour l'action correspondante.

## Etape 2 : Création d'une vue avec formulaire pour ajouter la licence

1. Dans la nouvelle action du contrôleur, récupérer le logiciel concerné dans une variable `@logiciel` à partir des paramètres de la requête. Pour ce faire, voir la méthode privée « `set_logiciel` » dans le bas du contrôleur.
2. Récupérer également dans une variable `@departements` la liste de tous les départements

3. Créer la vue `ajout_licence.html.erb` qui contiendra le formulaire de création de licence

**Indications :** Pour le formulaire utilisé pour spécifier la nouvelle licence, utiliser la méthode `form_for`

```
<%= form_for :xxx, :url => url_requete, :method => verbe_http do |f| %>

  <%= f.text_field :no_licence %>
  <%= f.collection_select :departement_id ..à compléter.. %>

  <%= f.submit "Create" %>

<% end %>
```

Pour créer un formulaire s'appuyant sur :

- `xxx`

A la soumission du formulaire, les paramètres retournés seront accessibles dans une variable « params ».

Cette variable est un hash qui contiendra un champ nommé `xxx`, lui-même un hash contenant tous les champs de saisie du formulaire (`:no_licence`, `:departement_id`).

Pour une question de lisibilité, utilisez quelque chose de plus significatif que `:xxx`, comme par exemple `:licence`

- `url_requete, verbe_http`

Url utilisé dans la requête de soumission du formulaire et verbe http utilisé pour la soumission du formulaire.

La soumission du formulaire devrait avoir pour effet d'invoquer une nouvelle action spécifique du contrôleur de logiciels, comme par exemple l'action « `créer_licence` ». Cette action aura la responsabilité de récupérer les informations saisies dans le formulaire et d'enregistrer la nouvelle licence dans la base de données.

En l'état, ajoutez cette nouvelle action dans le contrôleur, vide de toute instruction, en ajoutant la nouvelle route correspondante dans le fichier `routes.rb`, et en utilisant la méthode auto-générée pour construire `url_requete`.



Essayez en vérifiant que le formulaire de saisie arrive correctement à l'écran

**Etape 3 : Enregistrer la nouvelle licence dans la base de données**

Il ne s'agit plus que d'écrire le contenu de l'action « `créer_licence` ».

Il faut créer une nouvelle licence à partir des données saisies dans le formulaire (`no_licence` et `departement_id`), associer cette licence avec le logiciel concerné, puis la sauvegarder dans la base de données.

Indications :

- `params[:id]`

→ retourne l'identificateur du logiciel

- `params.require(:xxx).permit(:no_licence, :departement_id)`

→ retourne le hash contenant les champs `no_licence` et `departement_id`

→ « `require` » retournera une erreur si le champ `xxx` est absent, et seuls les champs `no_licence` et `departement_id` de `xxx` seront retenus (sécurité internet)

😊 **Solution : Voir étape 7**