



D05 - Formation Ruby on Rails

SQL

Staff 42 bocal@staff.42.fr

Stéphane Ballet balletstephane.pro@gmail.com

Résumé: SQL (sigle de Structured Query Language, en français langage de requête structurée) est un langage informatique normalisé servant à exploiter des bases de données relationnelles. Rails fournit un formidable support à ce système, il le supporte mais l'utilise et, donc repose dessus. Il vous faut donc savoir de quoi il retourne.

Table des matières

I	Préambule	2
II	Consignes	3
III	Règles spécifiques de la journée	5
IV	Exercice 00 : CRUD starts here	6
V	Exercice 01 : Seeds and migrations	8
VI	Exercice 02 : Creer et Lire	10
VII	Exercice 03 : Active Record Associations	12
VIII	Exercice 04 : Création dynamique	13
IX	Exercice 05 : Validation	14
X	Exercice 06 : 3D	16
XI	Exercice 07 : Méthodes Modèles	17
XII	Exercice 08 : Select	18
XIII	Exercice 09 : Scope	19
XIV	Exercice 10 : CRUD Ends Here	20

Chapitre I

Préambule

01010111 01101000 01111001 00100000 00110100 00110010 00111111
00100010 01010100 01101000 01100101 00100000 01100001 01101110 01110011 01110111
01100101 01110010 00100000 01110100 01101111 00100000 01110100 01101000 01101001
01110011 00100000 01101001 01110011 00100000 01110110 01100101 01110010 01111001
00100000 01110011 01101001 01101101 01110000 01101100 01100101 00101110 00100000
01001001 01110100 00100000 01110111 01100001 01110011 00100000 01100001 00100000
01101010 01101111 01101011 01100101 00101110 00100000 01001001 01110100 00100000
01101000 01100001 01100100 00100000 01110100 01101111 00100000 01100010 01100101
00100000 01100001 00100000 01101110 01110101 01101101 01100010 01100101 01110010
00101100 00100000 01100001 01101110 00100000 01101111 01110010 01100100 01101001
01101110 01100001 01110010 01111001 00101100 00100000 01110011 01101101 01100001
01101100 01101100 01101001 01110011 01101000 00100000 01101110 01110101 01101101
01100010 01100101 01110010 00101100 00100000 01100001 01101110 01100100 00100000
01001001 00100000 01100011 01101000 01101111 01110011 01100101 00100000 01110100
01101000 01100001 01110100 00100000 01101111 01101110 01100101 00101110 00100000
01000010 01101001 01101110 01100001 01110010 01111001 00100000 01110010 01100101
01110000 01110010 01100101 01110011 01100101 01101110 01110100 01100001 01110100
01101001 01101111 01101110 01110011 00101100 00100000 01100010 01100001 01110011
01100101 00100000 00110001 00110011 00101100 00100000 01010100 01101001 01100010
01100101 01110100 01100001 01101110 00100000 01101101 01101111 01101110 01101011
01110011 00100000 01100001 01110010 01100101 00100000 01100001 01101100 01101100
00100000 01100011 01101111 01101101 01110000 01101100 01100101 01110100 01100101
00100000 01101110 01101111 01101110 01110011 01100101 01101110 01110011 01100101
00101110 00100000 01001001 00100000 01110011 01100001 01110100 00100000 01100001
01110100 00100000 01101101 01111001 00100000 01100100 01100101 01110011 01101011
00101100 00100000 01110011 01110100 01100001 01110010 01100101 01100100 00100000
01101001 01101110 01110100 01101111 00100000 01110100 01101000 01100101 00100000
01100111 01100001 01110010 01100100 01100101 01101110 00100000 01100001 01101110
01100100 00100000 01110100 01101000 01101111 01110101 01100111 01101000 01110100
00100000 00100111 00110100 00110010 00100000 01110111 01101001 01101100 01101100
00100000 01100100 01101111 00100111 00101110 00100000 01001001 00100000 01110100
01111001 01110000 01100101 01100100 00100000 01101001 01110100 00100000 01101111
01110101 01110100 00101110 00100000 01000101 01101110 01100100 00100000 01101111
01100110 00100000 01110011 01110100 01101111 01110010 01111001 00101110 00100010

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez assumer les versions de langages suivantes :
 - Ruby `>= 2.3.0`
 - pour d09 Rails `> 5`
 - mais pour tous les autres jours Rails `4.2.7`
 - HTML `5`
 - CSS `3`
- Nous vous interdisons FORMELLEMENT d'utiliser les mots clés `while`, `for`, `redo`, `break`, `retry` et `until` dans les codes sources Ruby que vous rendrez. Toute utilisation de ces mots clés est considérée comme triche (et/ou impropre), vous donnant la note de -42.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas, nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépôt GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement spécifiés par les énoncés des exercices.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google` / `man` / `Internet` /
- Pensez à discuter sur le forum Piscine de votre Intra, ou Slack, ou IRC...
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne

sont pas autrement précisées dans le sujet...

- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Tout le travail de la journée sera à rendre dans le projet rails fourni dans les ressources.
- Toute Gem non autorisée par l'énoncé est interdite
- Toute variable globale non autorisée par l'énoncé est interdite
- Des tests sont fournis pour tous les exercices, vous devez passer les tests qui concernent l'exercice en cours.
- Vous devez impérativement gérer les erreurs. Aucune page d'erreur Rails ne sera tolérée en correction, même avec des valeurs erronées comme une db inexistante, une table mal formatée, etc...
- Les tests et la seed sont à laisser tels quels
- Vous ne devez avoir AUCUNE 'offense' signalée par rubocop
- Vous devez LAISSER intact le fichier ".rubocop.yml" (il contient une config adoucie de Rubocop spécialement pour le d05)

On peut lancer des tests un par un pour plus de clarté avec la commande suivante

```
ruby -I"lib:test" test/chemin\_du\_test/nom\_du\_fichier.rb -n "test\_nom"
```


Vous veillerez à remplacer "nom" par le nom du test à effectuer. Il se peut que pour des raisons de dépendances, la commande ne fonctionne pas. Cela ne sera pas accepté pour les corrections, vous devez savoir gérer ces erreurs.



NB: la lib sqlite3 est très sensible à la corruption. N'hésitez pas à la réinstaller.

Chapitre IV

Exercice 00 : CRUD starts here

	Exercice : 00
Exercice 00 : CRUD starts here	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées : \$db	
Remarques : n/a	

Bonjour à tous, j'espère que le rush du weekend vous a éclairé sur le fonctionnement de Rails, si vous ne l'avez pas fait, honte à vous! Mais j'ai eu pitié et dans ma grande mansuétude, vous trouverez dans les ressources, une base sur laquelle travailler.

Vous y trouverez des bases sommaires et un controller avec quelques fonction déjà nommées. Afin de formaliser les travaux rendus, vous devez garder les noms des routes et des méthodes.

En effet, des tests ont été rédigé pour vous. D'ailleurs, vous pourrez les lancer avec la commande :

```
rake test
```

Il est possible de lancer un test à la fois, je vous laisse le soin de parcourir la doc (oui, on est lundi matin, je sais, c'est cruel...), ils ont été nommés selon les méthodes qu'ils testent, à vous de vous adapter...

Sachez que votre objectif pour la journée est de passer tous les tests. D'ailleurs, vous n'aurez la chance de faire un grande chelem sur le banc de test qu'une fois les exercices terminés... Enfin, peut-être, hahaha.

Bon, trêve de piallage matinal, let's get started.

Lancez le serveur de l'application fournie, allez sur `localhost:3000` et vous aurez l'immense privilège d'admirer une superbe liste de boutons qui ne font... rien... pour l'instant.

Pour cet exercice, on va commencer soft, et simplement implémenter le code des trois premières méthodes du controller `ft_query`.

Le premier : **'create_db'** doit créer le fichier qui sera utilisé par sqlite pour la suite. Ce fichier DOIT s'appeler `"ft_sql"`. La méthode doit créer une instance de la classe `SQLite3::Database` et la stocker dans la variable globale `"$db"`. Oui, encore une variable globale pour le coup mais c'est pour des questions de scope (je pense que vous avez un peu compris le but). Vous allez avoir encore d'autres globales de débloquentes durant cette journée. Aussi, aucune erreur ne doit apparaître si vous cliquez sur **'create_db'** plusieurs fois.

Le deuxième : **'create_table'** doit créer deux tables dans notre fichier `'ft_sql'`, une nommée `'clock_watch'` et une autre nommée `'race'`. Ces deux tables devront comporter (dans cet ordre) :

- **'clock_watch'** :
 - une clé primaire auto incrementée : `ts_id`
 - un entier : `day`
 - un entier : `month`
 - un entier : `year`
 - un entier : `hour`
 - un entier : `min`
 - un entier : `sec`
 - un entier : `race`
 - une string (50 Chars max) : `name`
 - un entier : `lap`
- **'race'** :
 - une clé primaire auto incrementée : `r_id`
 - une string (50 Chars max) : `start`

Le troisième : **'drop_table'** doit supprimer entièrement les tables créées par la méthode **'create_table'**.


Encore une fois, aucun bug ne sera toléré en cas de violent martelage du bouton.

Pour vous convaincre du bon fonctionnement de vos deux méthodes, la commande `rake test` doit valider les tests `'db_file_creation'`, `'db_table_creation'` et `'db_drop_table'`.

À vous d'aller checker le code !

Chapitre V

Exercice 01 : Seeds and migrations

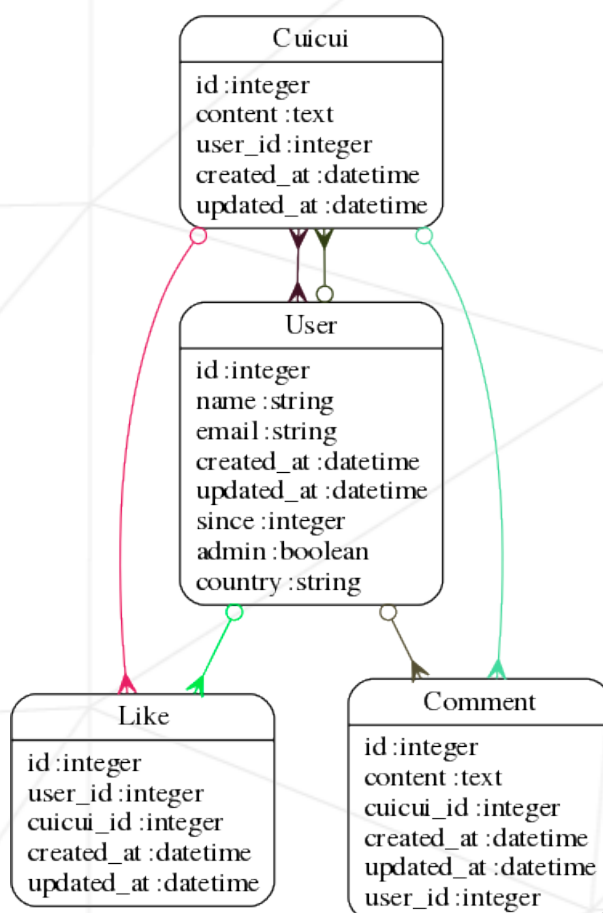
	Exercice : 01
Exercice 01 : Seeds and migrations	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : <code>tweetos_aka_hello_rails</code>	
Fonctions Autorisées :	
Remarques : n/a	

Vous commencez à voir un peu le concept du SQL, mais dans Rails, on bénéficie de tout un tas d'outils tous plus 'smart' les uns que les autres. Par exemple, les **migrations** qui vous permettent de scripter de manière versionnée la conception de vos tables. Ou alors, les **seeds** qui permettent, elles, de scripter la population de départ de vos tables.

Je vous laisse imaginer à quel point ces deux outils sont utiles et à ne pas prendre à la légère. Ici, une application en devenir vous est fournie, une seed y est présente, à vous de faire les bonnes migrations qui vont créer les tables adéquates à la population de la seed.

Spécifiez aussi une route par défaut, avec la macro "root" dans le fichier "config/routes.rb". Soyez judicieux.

Vous devrez créer les migrations spécifiques au schéma de base de données que vous trouverez ci contre :



Pour l'instant, les relations entre tables ne sont pas demandées. En revanche, pour valider l'exercice, vous devrez valider le test `migrations_test.rb` sans aucune erreur.


Utilisez [GIT](#) : les [migrations](#) sont sensibles et peuvent être 'rollback' pour leur modification mais pas toujours.



Renseignez vous sur les différentes options de rails generate. Vous verrez que vous avez pleins de générateurs différents (un peu comme scaffold, qui a l'avantage de vous faire le café et vous apporter le petit déjeuner au lit). Enfin, moi je dis ça mais c'est comme vous voulez.

Chapitre VI

Exercice 02 : Creer et Lire

	Exercice : 02
Exercice 02 : Creer et Lire	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées : <code>\$runner_1</code> , <code>\$runner_2</code> , <code>\$runner_3</code> , <code>\$runner_4</code> , <code>\$time_stamps</code>	
Remarques : n/a	

Ca va ? Les migrations n'ont pas eu raison de vous ? Parfait.

Retournons donc au [SQL](#) pur et dur...

Occupons nous maintenant du bouton "Subscribe / Start race". Vous l'aurez compris, cliquer une fois sur celui-là doit avoir pour effet de matérialiser :

- dans la table "clock_watch"
 - 4 nouvelles entrées dans clock_watch avec les 4 noms renseignés dans les encarts à droite
 - Le moment exact du clic pour chaque entrée, dans les champs day, month, year, hour, min, sec correspondants
 - La valeur de lap à 0
 - Une id unique dans la table clock_watch pour ts_id
 - Une id unique pour race correspondant à la bonne r_id
- dans la table "race" :
 - Une id unique (dans sa table) pour r_id
 - La valeur de start renseignée avec la string issue de (Time.now), qui doit correspondre avec les valeurs du lap '0' créés dans clock_watch (la course démarre au moment où les coureurs partent)

Vous devrez renseigner les 4 nouvelles variables globales avec les valeurs des boites de la premiere ligne des quatre inputs. Si des noms venaient à manquer, votre méthode doit assigner automatiquement ces champs avec "anonymous"

Ce qui fait qu'en remplissant les deux premières box avec "foo" et "bar" et en cliquant sur start un dimanche soir, vous devriez avoir quelque chose de ce style :

```
table clock_watch:
  ts_id  day   month  year   hour  min  sec  race  name      lap
-----
3       3     7     2016   18    7    40   0     anonymous  0
2       3     7     2016   18    7    40   0     anonymous  0
1       3     7     2016   18    7    40   0     bar       0
0       3     7     2016   18    7    40   0     foo       0

table race :
r_id  start
-----
0     2016-07-03 18:07:40 +0200
```

De toute façon, les tests sont là pour vous dire si vous êtes dans le vrai ou pas. Dans le doute, les tests auront toujours raison.

Si vous utilisez les variables correctement, vous devez voir l'ensemble des 4 fields se remplir automatiquement quand vous cliquez sur "Subscribe / Start"



NB: Récupérez dans vos méthodes, les noms passés en paramètres par les boutons comme ceci : `query_params[:name_1]`, `query_params[:name_2]`...

Pour pouvoir contempler vos résultats, décommentez les tableaux dans la vue en supprimant les lignes avec le commentaire `<!-- uncomment at ex02 -->` :


```
in [your_path_to_app]/app/views/ft_query/index.html.erb:
.....
<%if false%> <!-- uncomment at ex02 -->
.....
<%end%> <!-- uncomment at ex02 -->
```

Renseignez dans la méthode index du controller `ft_query`, la variable `$time_stamps` avec le contenu de la table "clock_watch". Si votre table est vide ou si le fichier `ft_sql` n'existe pas encore, renseignez la variable avec une string adaptée comme, par exemple : `['Database is empty or an other error occured']`
Et pour le moment, renseignez `$all` avec `['Not so fast, young padawan']`.

Vous aurez réussi votre exercice si vous passez le test "start_race", si vous remplissez toutes les consignes de la liste ci-dessus et que votre résultat ressemble à l'exemple de tableau en dessous de la liste.

Chapitre VII

Exercice 03 : Active Record Associations

	Exercice : 03
Exercice 03 : Active Record Associations	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : tweetos_aka_hello_rails	
Fonctions Autorisées :	
Remarques : n/a	


Active Record Associations n'est pas un terme au hasard.
Là aussi, pour réussir cet exercice, il vous faudra passer les tests suivants :

- Comment's relations methods
- Cuicui's relations methods
- Like's relations methods
- User's relations methods

Pour avoir un aperçu des données utilisée pour les tests, allez faire un tour du côté de "test/test_helper.rb". En effet, la db de test est indépendante de celle de développement et de production.

Chapitre VIII

Exercice 04 : Création dynamique

	Exercice : 04
Exercice 04 : Création dynamique	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées :	
Remarques : n/a	

Sa syntaxe délicieuse nous manquait, donc retournons au SQL. (prononcez : six-coups-elle, ou à la française : est-ce qu'eut aile).


Vous allez implémenter la méthode bien nommée "insert_time_stamp", appelée par les quatres boutons "time", qui ajoute une entrée à "clock_watch" avec :

- Une nouvelle ts_id unique
- Une valeur pour le champ race qui correspond a la dernière id de la table "race"
- Le nom qui correspond à la box à droite du bouton time qui viens d'être pressée.
- La valeur de lap qui est incrementée. Chaque clic sur Time chronomètre le tour et comptabilise le nombre de laps sous la colonne si bien intitulée : "lap"
- Le moment exact du click pour chaque entrée, dans les champs day, month, year, hour, min et sec correspondants

Pour valider cet exercice, vous devez remplir les conditions ci-dessus ET passer le test "insert_time_stamp".

Chapitre IX

Exercice 05 : Validation

	Exercice : 05
Exercice 05 : Validation	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : <code>tweetos_aka_hello_rails</code>	
Fonctions Autorisées :	
Remarques : n/a	

Tout vos modèles sont maintenant interconnectés. Vous pouvez maintenant décommenter les vues. Il y en a beaucoup, donc pour vous faciliter la vie, utilisez plutôt le script à la racine du projet rails comme ceci :

```
ruby views_com.rb unco
```

Contemplez toute cette donnée fraîche et qui n'a encore JA-MAIS servie : une occasion en or !

(Si des erreurs figurent encore, c'est que les précédents exercices qui traitent de cette application ne sont pas correctement faits.)

Elle tient toute sa valeur parce qu'elle est ordonnée, quelques doublons se sont déjà glissés dans les contenus, mais il se pourrait que cela dégenère très vite et avec les formulaires mis à disposition. Donc, pour protéger votre capital, je vous "invite" à créer de la validation de données `#lisezLaDocDeRailsCPasComplicqué`.

Pour réussir votre challenge économique, vous devez garantir :

- L'unicité des users par rapport à leur nom et à leur mail
- La longueur minimale du nom (soit 2 char)
- L'interdiction des noms suivants : ("42", "lancelot du lac", "Ruby") avec comme message " <name> is banished "
- La présence du nom de l'email, du since et du pays pour chaque user

- L'unicité des likes par rapport à l'association des id (un seul like par cuicui et par users ca semble logique)
- La validité syntaxique des emails
- La présence d'user_id sur création de cuicui
- La présence d'user_id sur création de comment
- La présence de contenu sur création de cuicui
- La présence de contenu sur création de comment
- L'unicité du contenu pour les cuicuis
- L'unicité du contenu pour les comments
- TOUTES les id doivent être strictement numeriques
- TOUTES les id doivent être présentes
- TOUTES les id doivent être uniques
- Les id doivent être valides (existantes pour le model associé)


Et comme les id des associations sont inscrites a la main, par exemple "http ://localhost :3000/cuicuis/new" vous laissez entrer dans la boîte "User" une string et ou une id invalide, il vous faut aussi assurer la validité de ces informations. Donc, que ce soit pour update ou new et que ce soit pour les likes, comments et cuicuis, vous devez aussi créer les validations qui assurent que les id associées soient existantes.

Votre banque d'informations sera bien gardée et de qualité garantie au fil du temps, plus besoin de surveiller / modérer.

Les tests pour cet exercice sont dans "test/models/* "

Chapitre X

Exercice 06 : 3D

	Exercice : 06
Exercice 06 : 3D	
Dossier de rendu : <i>ex06/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées :	
Remarques : n/a	


Vous avez implémenté '**drop_table**', vous allez maintenant implémenter le code des méthodes `delete_all` et `delete_last`.

- `delete_last` détruit le dernier enregistrement dans la table "time_stamp".
- `delete_all` détruit TOUTES les lignes de la table.

Des tests sont aussi présents pour cet exercice.

Chapitre XI

Exercice 07 : Méthodes Modèles

	Exercice : 07
Exercice 07 : Méthodes Modèles	
Dossier de rendu : <i>ex07/</i>	
Fichiers à rendre : tweetos_aka_hello_rails	
Fonctions Autorisées :	
Remarques : n/a	

Vous allez pouvoir décommenter la dernière partie dans "http ://localhost :3000/users/<userid>" qui correspond à la page "show" de l'utilisateur dont l'id est mis à la place de <userid> dans l'URL ci-dessus.


Supprimez les lignes de la vue correspondante où figurent : <!-- à décommenter dans l'ex07 -->

Vous avez plein d'erreurs. C'est normal. Votre exercice n'est pas fini.
Vous devez ouvrir le modèle User et implémenter du code dans les méthodes vides :

- fame : retourne un entier qui est la somme de tous les likes attribués à tous les utilisateurs de l'utilisateur.
- senior ? : retourne true si le champ since date de plus de 10 ans
- junior ? : retourne false si le champ since date de plus de 10 ans
- responses : Liste les 5 derniers commentaires des utilisateurs de l'utilisateur (les derniers en création pas en modification)
- top_cuicui : Liste les utilisateurs de l'utilisateur triés par nombre de like

Chapitre XII

Exercice 08 : Select

	Exercice : 06
Exercice 08 : Select	
Dossier de rendu : <i>ex06/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées : \$all	
Remarques : n/a	

Vous commencez à avoir l'habitude, vous allez implémenter le code des méthodes "all_by_name" et "all_by_race". De plus, une nouvelle variable globale vous est mise à disposition : "\$all".


Pour la peine, vous devrez faire en sorte que :

- "all_by_name" : stocke toutes les entrées de votre table "time_stamp" par nom de coureur dans la variable globale "\$all"
- "all_by_race" : stocke toutes les entrées de votre table "time_stamp" par ID de course dans la variable globale "\$all"

Bien sur, il y a encore une fois des tests pour vous aider. A vous de jouer pour n'avoir aucune erreur.

Chapitre XIII

Exercice 09 : Scope


	Exercice : 09
Exercice 09 : Scope	
Dossier de rendu : <i>ex09/</i>	
Fichiers à rendre : <code>tweetos_aka_hello_rails</code>	
Fonctions Autorisées :	
Remarques : n/a	

Dans le modèle Cuicui a la ligne : `"# scope :top, lambda implémentez ici "`, remplacez "implémentez ici" par votre code de sorte que, dans la page à laquelle mène le bouton "like", se liste les cuicuis les plus likés.

Servez vous des tests pour comprendre la composition de la collection attendue.

Chapitre XIV

Exercice 10 : CRUD Ends Here

	Exercice : 10
Exercice 10 : CRUD Ends Here	
Dossier de rendu : <i>ex10/</i>	
Fichiers à rendre : Seek_well	
Fonctions Autorisées :	
Remarques : n/a	

Vous allez implémenter la dernière étape du CRUD : Update

Vos coureurs doivent pouvoir changer leur nom à la volée pour tous les `time_stamps` de la course en cours, sauf si il etait anonyme. Dans ce cas, il le restera.

Des tests sont à votre disposition pour cet exercice.