



## D05 - Ruby on Rails Training

### SQL

*Summary: SQL (for Structured Query Language) is a normalized computer language used to exploit relational databases. Rails amazingly supports this system. Not only does it support it, but it uses it and relies on it. Hence, you should know what it's all about.*

# Contents

<b>I</b>	<b>Preamble</b>	<b>2</b>
<b>II</b>	<b>Ocaml piscine, general rules</b>	<b>3</b>
<b>III</b>	<b>Today's specific instructions</b>	<b>5</b>
<b>IV</b>	<b>Exercise 00: CRUD starts here</b>	<b>6</b>
<b>V</b>	<b>Exercise 01: Seeds and migrations</b>	<b>8</b>
<b>VI</b>	<b>Exercise 02: Create and read</b>	<b>10</b>
<b>VII</b>	<b>Exercise 03: Active Record Associations</b>	<b>12</b>
<b>VIII</b>	<b>Exercise 04: Dynamic creation</b>	<b>13</b>
<b>IX</b>	<b>Exercise 05 : Validation</b>	<b>14</b>
<b>X</b>	<b>Exercise 06: 3D</b>	<b>16</b>
<b>XI</b>	<b>Exercise 07: Model Methods</b>	<b>17</b>
<b>XII</b>	<b>Exercise 08: Select</b>	<b>18</b>
<b>XIII</b>	<b>Exercise 09: Scope</b>	<b>19</b>
<b>XIV</b>	<b>Exercise 10: CRUD End Here</b>	<b>20</b>

# Chapter I

## Preamble

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

# Chapter II

## Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

# Chapter III

## Today's specific instructions

- Today's work will be turn-in in the Rails project provided with the resources.
- Any Gem that's not authorized by the wording is prohibited.
- Any global variable that's not authorized by the wording is prohibited.
- Tests are provided for all exercises. You must run the tests for each exercise.
- You have to manage the errors. No Rails error page will be tolerated during the evaluation, even with a flawed value such as an inexistent db, an ill formatted table, etc...
- The tests and the seed must be left as are.
- Rubocop must not report any 'offense'.
- You must not TOUCH the ".rubocop.yml" file. (it contains a soft config of Rubocop, especially crafted for the d05)

For the sake of clarity, you can run one test at a time.

```
ruby -I"lib:test" test/path\_to\_file.rb -n "file\_name"
```


You will replace "name" by the name of the test you will run. Because of dependancies, command might not run properly. This will not be tolerated during the evaluation. You must manage these errors.



NB: the sqlite3 lib is very sensitive to corruption. Don't be afraid to reinstall it.

# Chapter IV

## Exercise 00: CRUD starts here

	Exercise 00
Exercise 00: CRUD starts here	
Turn-in directory : <i>ex00/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions : <b>\$db</b>	

Hello there! I hope the weekend's rush helped you get familiar with Rails. If you did not take it, shame on you! But since I'm a model of mercifulness, you will find a base you can work on in the resources.

You will find simple bases and a controller with a few functions that have already been named. In order to formalize the works in your repo, you will have to keep the path's and methods' names.

As a matter of fact, tests have been redacted for you and you will be able to run them with the following command:

```
rake test
```

You can run one test at a time. I'll let you run through the doc (I know, it's Monday morning, I'm a cruel bastard...). They've been named according to the methods they test. Try to adapt...

Know that you're supposed to take all the tests within the day. Anyway, in order to score all the points during the evaluation, you will have to complete all the exercises... but will this be enough?

Ok, enough with all the Monday morning babble, let's get started.

Run the server of the provided application, go to `localhost:3000` and you will be granted with a wonderful list of buttons that are... absolutely useless... for now.

For this exercise, we'll start gently by simply implementing the code of the first three methods of the `ft_query` controller.

The first one: `'create_db'` must create the file that will be used by sqlite afterwards. This file **MUST** be named `"ft_sql"`. The method must create an instance of the `SQLite3::Database` class and stock it and the `"$db"` global variable. Yes, that's another global variable, but this time, it's a matter of scope (I believe you've figured out where it was going). You will have other unlocked globals today. Thus, no error should appear if you click many times on `'create_db'`.

The second one: `'create_table'` must create two tables in our `'ft_sql'` file. One is named `'clock_watch'`, the other `'race'`. Both these tables will include (in that order):

- `'clock_watch'` :
  - a self incremented primary key: `ts_id`
  - a whole: `day`
  - a whole: `month`
  - a whole: `year`
  - a whole: `hour`
  - a whole: `min`
  - a whole: `sec`
  - a whole: `race`
  - a string (50 Chars max): `name`
  - a whole: `lap`
- `'race'` :
  - a self incremented primary key: `r_id`
  - a string (50 Chars max): `start`

The third one: `'drop_table'` must entirely delete the tables created by the `'create_table'` method.


Once again, no bug will be tolerated when hammering the button.

To make sure both you methods work properly, the `rake test` command must validate the `'db_file_creation'`, `'db_table_creation'` and `'db_drop_table'` tests. Go check the code!



# Chapter V

## Exercise 01: Seeds and migrations

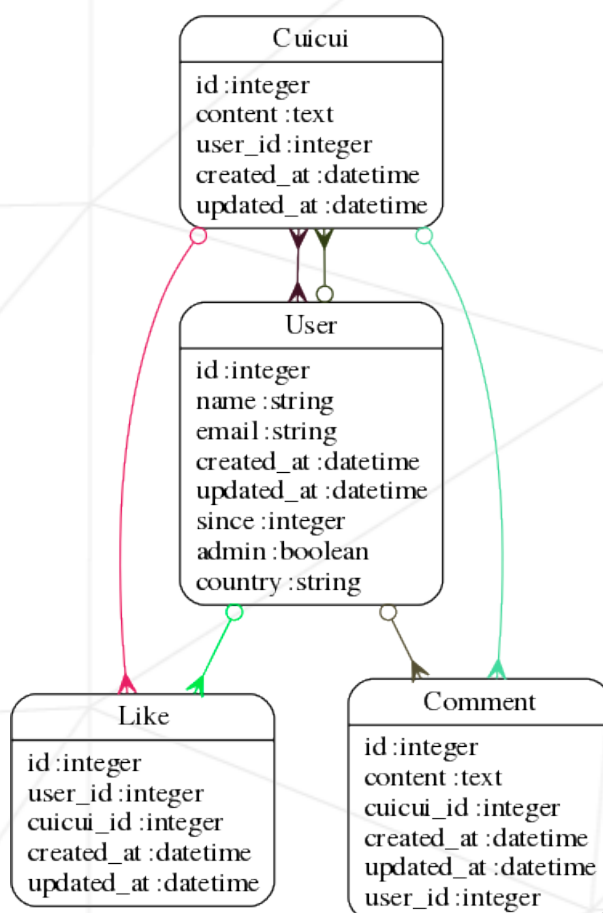
	Exercise 01
Exercise 01: Seeds and migrations	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <b>tweetos_aka_hello_rails</b>	
Allowed functions :	

You start digging the SQL concept, but in Rails, you will enjoy a whole lot of 'smart' tools. For instance, the **migrations** that will help you script you table's conception a versioned way. Or the **seeds** that help script the starting population of your tables.

You cannot fathom how much these tools will help you, so you shouldn't take them lightly. Here, you will be provided an application in the making that includes a seed. You will have to make the correct migrations that will create the tables matching the seed's population.

You will also specify a default path with the "root" macro in the "config/routes.rb" file. Act wisely.

You will have to create the migrations specific to the following basic database diagram:



For the moment, relations between tables are not required. However, for this exercise to be validated, you will validate the `migrations_test.rb` test properly.


Use [GIT](#): the [migrations](#) are sensitive and can be 'rollback' for their modifications. But not always.



Find out about the different options of `rails generate`. You will discover many different generators (kind of like `scaffold`, which comes with many perks like, making coffee and serving breakfast in bed). Now you know that, I'll let you choose your own way to do things.

# Chapter VI

## Exercise 02: Create and read

	Exercise 02
Exercise 02: Create and read	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions : <code>\$runner_1</code> , <code>\$runner_2</code> , <code>\$runner_3</code> , <code>\$runner_4</code> , <code>\$time_stamps</code>	

You're ok? Migrations did not make you sweat too much? Fine.

Let's go back to hardcore [SQL](#)...

Let's focus on the "Subscribe / Start race" button. As you found out, clicking it will materialize:

- in the "clock\_watch" table
  - 4 new inputs in clock\_watch with the 4 names indicated in the inserts on the right.
  - The exact moment of the click for each input in the matching day, month, year, hour, min, sec fields.
  - The lap value at 0
  - A `unique` in the table `clock_watch` id for `ts_id`
  - A `unique` id for race matching the correct `r_id`
- in the "race" table:
  - A `unique`(in its table) id for `r_id`
  - The start value indicated with the (Time.now) string issue that must match the lap '0' values created in clock\_watch (the race starts when the runners go).

You must fill in the 4 new global variables with the boxes' values of the first line of the 4 inputs. If some names are missing, your method automatically assign "anonymous" to these fields.

Thus, filling the first two boxes with "foo" and "bar" and clicking start on a Sunday evening, you will have something remotely looking like that:

```
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
```

Anyway, tests are made to let you know whether you got things right or wrong. If you wonder, the test will always be right.

If you properly use variables, you must see the whole 4 fields automatically filling up when you click "Subscribe / Start"



NB: Retrieve your methods, the names passed in parameters with buttons as follows: `query_params[:name_1]`, `query_params[:name_2]`...

In order to behold your results, uncomment the tables in the view deleting the lines with the comment `<!-- uncomment at ex02 -->` :

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


In the `ft_query` controller index method, fill in the `$time_stamps` variable with the "clock\_watch" table content. If your table is empty of the `ft_sql` file doesn't exist yet, fill in the variable with an accurate string such as: `['Database is empty or an other error occurred']`

And for now, fill in `$all` with `['Not so fast, young padawan']`.

Your exercise will be valid if you pass the "start\_race" test, if you've observed all the instructions above and your result looks like the table example below the list.

# Chapter VII

## Exercise 03: Active Record Associations

	Exercise 03
Exercise 03: Active Record Associations	
Turn-in directory : <i>ex03/</i>	
Files to turn in : <b>tweetos_aka_hello_rails</b>	
Allowed functions :	


Active Record Associations is not a random name.  
Once again, if you want to validate this exercise, you will have to run the following tests:

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

To get a quick peek at the data used for the tests, go check "test/test\_helper.rb". Indeed, the test db is independent from the development and production ones.

# Chapter VIII

## Exercise 04: Dynamic creation

	Exercise 04
Exercise 04: Dynamic creation	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions :	

Since we were missing its delightful syntax, let's go back to SQL. (fun fact: in French, SQL reads like "Is she...". Fun! Right? Right ?? ).


You will implement the aptly named "insert\_time\_stamp", called by the four "time" buttons, that add a "clock\_watch" input with:

- A new single ts\_id
- A value for the race field matching the final id in the "race" table.
- The name matching the box on the right of the time button that was just clicked.
- The incremented lap value. Each click on time times the lap and counts the number of laps in the o so aptly called "lap" column.
- The exact click moment for each input in the matching fields day, month, year, hour, min and sec.

To validate the exercise, your must observe all the conditions above AND run the "insert\_time\_stamp" test.

# Chapter IX

## Exercise 05 : Validation

	Exercise 05
Exercise 05 : Validation	
Turn-in directory : <i>ex05/</i>	
Files to turn in : <b>tweetos_aka_hello_rails</b>	
Allowed functions :	

All your models are now interconnected. You can uncomment the views. There are many. To make your life easier, use the script at the root of the rails project and proceed as follows:

```
ruby views_com.rb unco
```

Behold this fresh virgin data: such a rare opportunity!

(If you still get errors, it means the previous exercises tackling this application were not correctly achieved.)

It is valuable because it is tidy. There already are duplicates in the content, but it might degenerate quickly with the provided forms. That is why, in order to protect your capital, you should create data validation. *#Readtheraildocitsnotcomplicated*.

To achieve your financial challenge, you must guarantee:

- The unicity of users according to their names and mail addresses.
- The minimal length of a name (that is 2 char)
- The banning of the following names: ( "42", "lancelot du lac", "Ruby") with the message " <name> is banished ".
- For each user, the presence of the email name, the since and the country.
- The unicity of likes according to the id association (one like for each tweet and user. Logic prevails).

- Syntactic validity of emails.
- User\_id presence on each tweet creation.
- User\_id presence on each comment creation.
- Content presence on each tweet creation.
- Content presence on each comment creation.
- The unicity of content for the tweets.
- The unicity of contents for the comments.
- EACH id must strictly be digital.
- EACH id must be present.
- EACH id must be unique.
- Ids must be valid (existing for the associated model).

And since the associated ids were entered by hand, "http://localhost:3000/tweets/new" for instance, will let you enter a invalid string and/or id in the "user" box, we also need to confirm the validity of these informations. So, whether it's for update or new and whatever it is about likes, comments or tweets, you must also create validations that confirm that associated ids are existing.


From now on, your data bank will be thoroughly protected. No need to watch or moderate it.

Tests for this exercise are in "test/models/\* "



# Chapter X

## Exercise 06: 3D

	Exercise 06
Exercise 06: 3D	
Turn-in directory : <i>ex06/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions :	


You have implemented '**drop\_table**', you will now implement the `delete_all` and `delete_last` method codes.

- `delete_last` destroys the last record on the "time\_stamp" table.
- `delete_all` destroys ALL the lines in the table.

We do have tests for this exercise also!

# Chapter XI

## Exercise 07: Model Methods

	Exercise 07
Exercise 07: Model Methods	
Turn-in directory : <i>ex07/</i>	
Files to turn in : <b>tweetos_aka_hello_rails</b>	
Allowed functions :	


Now you can uncomment the last part in "http://localhost:3000/users/<userid>" that matches the "show" page of the user whose id is set instead of <userid> in the url above. Delete the lines of the views with the following statement: <!-- uncomment in ex07 -->

You have an awful lot of errors. Don't worry. Your exercise is not over yet. You must open a User model and implement code in the empty methods:

- fame: returns an integer that is the sum of all the likes given to all the user's tweets.
- senior?: returns true if the field since is more than 10 years old.
- junior?: returns false if the field since is less than 10 years old. (LE FRANÇAIS INDIQUAIT ENCORE "PLUS" DE 10 ANS MAIS J'IMAGINE QUE C'EST "MOINS")
- responses: lists the last 5 comments of the user's tweets.(created, not modified)
- top\_tweet: lists the user's tweets sorted in number of likes.

# Chapter XII

## Exercise 08: Select

	Exercise 06
Exercise 08: Select	
Turn-in directory : <i>ex06/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions : <b>\$all</b>	

You should be used to it by now. You will implement "all\_by\_name" and "all\_by\_race" methods codes. Besides, a new global variable is provided. It's called: "\$all".


This is why you will have to make sure that:

- "all\_by\_name": stores all the inputs of your "time\_stamp" table sorted by runner's names in the "\$all" global variable.
- "all\_by\_race": stores all the inputs of your "time\_stamp" table sorted by race ID in the "\$all" global variable.

Run the tests and make sure you get no error!

# Chapter XIII

## Exercise 09: Scope


	Exercise 09
Exercise 09: Scope	
Turn-in directory : <i>ex09/</i>	
Files to turn in : <b>tweetos_aka_hello_rails</b>	
Allowed functions :	

In the tweet model, on the line: `"# scope :top, lambda implement here"`, replace "implement here" by your code so that the list of the tweets with the most likes appears on the page the "like" button leads to.

Use the tests to understand the composition of the expected collection.

# Chapter XIV

## Exercise 10: CRUD End Here

	Exercise 10
Exercise 10: CRUD End Here	
Turn-in directory : <i>ex10/</i>	
Files to turn in : <b>Seek_well</b>	
Allowed functions :	

You're going to implement the last step of the CRUD: Update

Your racers must be able to change their names on the fly for each of the `time_stamps` of the ongoing race, except if they were anonymous. If so, they will remain anonymous.

Tests are available for this exercise.