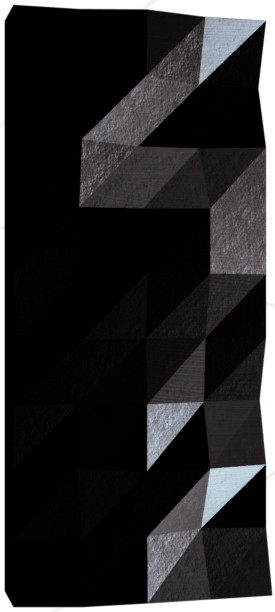

# Training Python-Django - 2

## ORM

*Summary: Today, we'll tackle the ORM of* `Django`.

*Version: 1*

# Contents

# Chapter I

# General rules

- Your project must be realized in a virtual machine.
- Your virtual machine must have all the necessary software to complete your project. These softwares must be configured and installed.
- You can choose the operating system to use for your virtual machine.
- You must be able to use your virtual machine from a cluster computer.
- You must use a shared folder between your virtual machine and your host machine.
- During your evaluations you will use this folder to share with your repository.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

# Chapter II

# Today's specific rules

- You must use a `python3` interpreter.
- Each exercise will be independant. If, among the required features, some have already been tackled in previous exercises, duplicate them in the current one.
- You must work in a `postgresql` database named `djangotraining` and create a role named `djangouser`, which password will be `"secret"`, that will have all the rights for it.
- Your repository folder must be a Django project. The project must be named after the current day.
- We will use Django's application concept to separate the exercises:
  Today's exercises must be located in a specific Django application named after the matching exercise and located at the root of the repo.
- The Django project must be properly configured to meet the exercise's requirements. You will not be able to change the configurations during evaluation.
- You can't turn-in any migration with your work.
- In each exercise which cart mentions `ORM`, you must use the `Django`'s ORM. You mustn't write any line of `SQL`.
- In each exercise which cart mentions `SQL`, you must use the `psycopg2` library and run all the requests in `SQL`.

Here is a typical structure example for the repo of a student named krichard, about the day d42, including 2 exercises:

```
|-- krichard
|   |-- .
|   |-- ..
|   |-- .git
|   |-- .gitignore
|   |-- d42
|   |   |-- __init__.py
|   |   |-- settings.py
|   |   |-- urls.py
|   |   |-- wsgi.py
|   |-- ex00
|   |   |-- admin.py
|   |   |-- apps.py
|   |   |-- forms.py
|   |   |-- __init__.py
|   |   |-- models.py
|   |   |-- tests.py
|   |   |-- urls.py
|   |   |-- views.py
|   |-- ex01
|   |   |-- admin.py
|   |   |-- apps.py
|   |   |-- forms.py
|   |   |-- __init__.py
|   |   |-- models.py
|   |   |-- tests.py
|   |   |-- urls.py
|   |   |-- views.py
|   |-- manage.py
```

.

Be smart: factor your code and make it easy to use. You'll save some time.

# Chapter III

# Exercise 00

|  | Exercise 00 |
|---|---|
| Exercice 00: SQL - building a table | |
| Turn-in directory : *ex*00/ | |
| Files to turn in : | |
| Allowed functions : | |

Create a Django application named `ex00` as well as an inside view that must be accessed at the following URL: `127.0.0.1:8000/ex00/init`.

This view must create an SQL table in Postgresql with the help of the `psycopg2` library and return a page containing "OK" if successful. Otherwise, it must return an error message describing the problem.

The SQL table must fit this description:

- It must be named `ex00_movies`.
- It must be created only if it doesn't already exist.
- It must only include the following fields:
  - `title`: unique, variable character chain, 64 byte maximum size, non null.
  - `episode_nb`: full, PRIMARY KEY.
  - `opening_crawl`: text, can be null, no size limit.
  - `director`: variable character chain, non null, 32 bytes maximum size.
  - `producer`: variable character chain, non null, 128 bytes maximum size.
  - `release_date`: date (without time), non null.

# Chapter IV

# Exercise 01

|  | Exercise 01 |
|---|---|
| Exercise 01: ORM - building a table | |
| Turn-in directory : $ex01/$ | |
| Files to turn in : | |
| Allowed functions : | |

Create an application named `ex01`. In it, create a `Django` model named `Movies` with this fields exactly:

- `title`: unique, variable character chain, 64 byte maximum size, non null.
- `episode_nb`: full, PRIMARY KEY.
- `opening_crawl`: text, can be null, no size limit.
- `director`: variable character chain, non null, 32 bytes maximum size.
- `producer`: variable character chain, non null, 128 bytes maximum size.
- `release_date`: date (without time), non null.

This model also must redefine the `__str__` method so that it sends back the `title` attribute.

# Chapter V

# Exercise 02

|  | Exercise 02 |
|---|---|
| Exercise 02: SQL - Data insertion | |
| Turn-in directory : $ex02/$ | |
| Files to turn in : | |
| Allowed functions : | |

You must create an Django application named `ex02`. In this application, the accessible views via the fololwing urls:

- `127.0.0.1:8000/ex02/init`: must create a table with the very same specifications as the one required for the `ex00` except it will be named `ex02_movies`.

  It must return a page displaying "OK" if successful. Otherwise, it should display a message describing the problem.

- `127.0.0.1:8000/ex02/populate`: must insert the following data in the table created by the previous view:

  - episode_nb: 1 - title: The Phantom Menace - director: George Lucas - producer: Rick McCallum - release_date: 1999-05-19
  - episode_nb: 2 - title: Attack of the Clones - director: George Lucas - producer: Rick McCallum - release_date: 2002-05-16
  - episode_nb: 3 - title: Revenge of the Sith - director: George Lucas - producer: Rick McCallum - release_date: 2005-05-19
  - episode_nb: 4 - title: A New Hope - director: George Lucas - producer: Gary Kurtz, Rick McCallum - release_date: 1977-05-25
  - episode_nb: 5 - title: The Empire Strikes Back - director: Irvin Kershner - producer: Gary Kurtz, Rick McCallum - release_date: 1980-05-17

- episode_nb: 6 - title: Return of the Jedi - director: Richard Marquand - producer: Howard G. Kazanjian, George Lucas, Rick McCallum - release_date: 1983-05-25
- episode_nb: 7 - title: The Force Awakens - director: J. J. Abrams - producer: Kathleen Kennedy, J. J. Abrams, Bryan Burk - release_date: 2015-12-11

It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex02/display`: must display all the data included in the `ex02_movies` table in an `HTML` table, including the eventual void fields.

  If there is no available data, or an error, the page must simply display `"No data available"`.

# Chapter VI

# Exercise 03

|  | Exercise 03 |
|---|---|
| Exercice 03: ORM - data insertion | |
| Turn-in directory : *ex*03/ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex03`. Inside it, create a Django model identical to the one created in the `ex01`.

This app must include the accessible views via the following URLs:

- `127.0.0.1:8000/ex03/populate`: must insert the model of this application, the following data:
    - episode_nb: 1 - title: The Phantom Menace - director: George Lucas - producer: Rick McCallum - release_date: 1999-05-19
    - episode_nb: 2 - title: Attack of the Clones - director: George Lucas - producer: Rick McCallum - release_date: 2002-05-16
    - episode_nb: 3 - title: Revenge of the Sith - director: George Lucas - producer: Rick McCallum - release_date: 2005-05-19
    - episode_nb: 4 - title: A New Hope - director: George Lucas - producer: Gary Kurtz, Rick McCallum - release_date: 1977-05-25
    - episode_nb: 5 - title: The Empire Strikes Back - director: Irvin Kershner - producer: Gary Kurtz, Rick McCallum - release_date: 1980-05-17
    - episode_nb: 6 - title: Return of the Jedi - director: Richard Marquand - producer: Howard G. Kazanjian, George Lucas, Rick McCallum - release_date: 1983-05-25

- episode_nb: 7 - title: The Force Awakens - director: J. J. Abrams - producer: Kathleen Kennedy, J. J. Abrams, Bryan Burk - release_date: 2015-12-11

It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex03/display` : must display all the data included in the `Movies` table in an `HTML` table, including the eventual void fields.

  If there is no available data, or an error, the page must simply display `"No data available"`.

`During the evaluation, the migration will be carried out before the tests.`

# Chapter VII

# Exercise 04

|  | Exercise 04 |
|---|---|
| Exercise 04: SQL - Data deleting | |
| Turn-in directory : $ex04/$ | |
| Files to turn in : | |
| Allowed functions : | |

Create an app named `ex04`. It must contain the accessible views via the following URLs:

- `127.0.0.1:8000/ex04/init`: must create a table with the very same specifications as the one required for the `ex00` app except it will be named `ex04_movies`.

  It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex04/populate`: must insert the data described in the exercise `ex02` in the table created in the previous view.

  This view must reinsert any deleted data.

  It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex04/display`: must display all the data included in the `ex04_movies` table in an `HTML` table, including the eventual void fields.

  If there is no available data, or an error, the page must simply display `"No data available"`.

- `127.0.0.1:8000/ex04/remove`: must display an `HTML` form containing a drop-down list of film titles and a `submit` button named `remove`.

The film titles are the ones contained in the `ex04_movies` table.

When the form is validated, the selected film is deleted from the database and the form is redisplayed with the updated list containing the remaining films.

If there is no available data, or an error, the page must simply display `"No data available"`.

# Chapter VIII

# Exercise 05

|  | Exercise 05 |
|---|---|
| Exercice 05: ORM - Deleting data | |
| Turn-in directory : $ex05/$ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex05`. Create a model identical to the one in ex01 inside it.

This app must include the accessible views via the following URLs:

- `127.0.0.1:8000/ex05/populate`: must insert the data described in the exercise `ex03` in the created application.

  This view must reinsert any deleted data.

  It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex05/display`: must display all the data included in the `Movies` table in an `HTML` table, including the eventual void fields.

  If there is no available data, or an error, the page must simply display `"No data available"`.

- `127.0.0.1:8000/ex05/remove`: must display an `HTML` form containing a drop-down list of film titles and a `submit` button named `remove`.

  The film titles are the ones contained in the `Movies` model of this application.

When the form is validated, the selected film is deleted from the database and the form is redisplayed with the updated list containing the remaining films.

If there is no available data, or an error, the page must simply display `"No data available"`.

`During the evaluation, the migration will be carried out before the tests.`

# Chapter IX

# Exercise 06

|  | Exercise 06 |
|---|---|
| Exercise 06: SQL - Updating a data | |
| Turn-in directory : $ex06/$ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex06`. It must contain the accessible views via the following URLs:

- `127.0.0.1:8000/ex06/init`: must create a table with the very same specifications as the one required for the `ex00` app except it will be named `ex06_movies` and will include the following additional fields:
  - `created` a datetime type (date and time), that, when created, must automatically set to the current date and time.
  - `updated` a datetime type (date and time), that, when created, must automatically set to the current date and time and automatically updates with each update thanks to the following trigger:

```
CREATE OR REPLACE FUNCTION update_changetimestamp_column()
RETURNS TRIGGER AS $$
BEGIN
NEW.updated = now();
NEW.created = OLD.created;
RETURN NEW;
END;
$$ language 'plpgsql';
CREATE TRIGGER update_films_changetimestamp BEFORE UPDATE
ON ex06_movies FOR EACH ROW EXECUTE PROCEDURE
update_changetimestamp_column();
```

- `127.0.0.1:8000/ex06/populate`: must populate the table created in the previous view with the data described in the exercise doit peupler la table `ex02`.

It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex06/display`: must display all the data included in the `ex06_movies` table in an `HTML` table.

  If there is no available data, or an error, the page must simply display `"No data available"`.

- `127.0.0.1:8000/ex06/update`: must manage the sending and receipt of a form. The latter must allow to choose a film in a drop- down menu containing the films included in the `ex06_movies` table and to write text in the second field. When validating the form, the view must replace the `opening_crawl` field of the selected film with the text typed in the form in the `ex06_movies` table.

  If there is no available data, or an error, the page must simply display `"No data available"`.

# Chapter X

# Exercise 07

|  | Exercise 07 |
|---|---|
| Exercise 07: ORM - Updating a data | |
| Turn-in directory : *ex*07/ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex07`.Create a model identical to the one in ex01 inside it, except you will add the following fields:

- `created` a datetime type (date and time), that, when created, must automatically set to the current date and time.

- `updated` a datetime type (date and time), that, when created, must automatically set to the current date and time and automatically updates with each update.

This app must contain the accessible views on the following URLs:

- `127.0.0.1:8000/ex07/populate`: populates the model previously created with the same data as `ex02`.

  It must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex07/display`: displays all the data included in the `Movies` table in an `HTML` table.

  If there is no available data, or an error, the page must simply display `"No data available"`.

- `127.0.0.1:8000/ex07/update`: must manage the sending and receipt of a form. The latter must allow to choose a film in a drop- down menu containing the films

included in the `Movies` table and to write text in the second field.
When validating the form, the view must modify the `opening_crawl` field of the selected film with the text typed in the form of the `Movies` model.

If there is no available data, or an error, the page must simply display `"No data available"`.

`During the evaluation, the migration will be carried out before the tests.`

# Chapter XI

# Exercise 08

|  | Exercise 08 |
|---|---|
| Exercise 08: SQL - Foreign Key | |
| Turn-in directory : *ex*08/ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex08`. This app must contain the accessible views via the following URLs:

- `127.0.0.1:8000/ex08/init`: Must create two tables.
  The first must be named `ex08_planets` and include the following fields:
  - `id`: serial, primary key
  - `name`: unique, variable character chain, 64 byte maximum size, non null.
  - `climate`: variable character chain.
  - `diameter`: whole.
  - `orbital_period`: whole.
  - `population`: large whole.
  - `rotation_period`: whole.
  - `surface_water`: real.
  - `terrain`: variable character chain, 128 bytes maximum size.

  The second one must be called `ex08_people` and include the following fields:
  - `id`: serial, primary key.
  - `name`: unique, variable character chain, 64 byte maximum size, non null.

  - `birth_year`: variable character chain, 32 byte maximum size.
  - `gender`: variable character chain, 32 byte maximum size.
  - `eye_color`: variable character chain, 32 byte maximum size.
  - `hair_color`: variable character chain, 32 byte maximum size.
  - `height`: whole.
  - `mass`: real.
  - `homeworld`: variable character chain, 64 byte maximum size, foreign key, referencing the `name` column of the `08_planets` table.

- `127.0.0.1:8000/ex08/populate`: must populate both tables copying the content of the `people.csv` and `planets.csv` files in the matching tables, respectively: `ex08_people` and `ex08_planets`.

  This view must return a page displaying "OK" for each successful insertion. Otherwise, it must display an error message stating the problem.

- `127.0.0.1:8000/ex08/display`: displays all the characters' names, their homeworld as well as the climate, which is `windy` or moderately `windy` sorted in character's name alphabetical order.

  If there is no available data, or an error, the page must simply display `"No data available"`.

`Get information about the psycopg2copy_from method`

# Chapter XII

# Exercise 09

|  | Exercise 09 |
|---|---|
| Exercise 09: ORM - Foreign Key | |
| Turn-in directory : *ex*09/ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app called `ex09` and create two models inside. The first one will be named `Planets` and will contain the following fields:

- `name`: unique, variable character chain, 64 byte maximum size, non null.
- `climate`: variable character chain.
- `diameter`: whole.
- `orbital_period`: whole.
- `population`: large whole.
- `rotation_period`: whole.
- `surface_water`: real.
- `terrain`: character chains.

- `created` a datetime type (date and time), that, when created, must automatically set to the current date and time.

- `updated` a datetime type (date and time), that, when created, must automatically set to the current date and time and automatically updates with each update.

This model also must redefine the `__str__()` method so that it returns the `name` attribute.

The second model you will create must be named `People` and contain the following fields:

- `name`: character chain, 64 byte maximum size, non null.
- `birth_year`: character chain, 32 byte maximum size.
- `gender`: character chain, 32 byte maximum size.
- `eye_color`: character chain, 32 byte maximum size.
- `hair_color`: character chain, 32 byte maximum size.
- `height`: whole.
- `mass`: real.
- `homeworld`: character chain, 64 byte maximum size, foreign key referencing the `name` column of this app's `Planets` table.
- `created` a datetime type (date and time), that, when created, must automatically set to the current date and time.
- `updated` a datetime type (date and time), that, when created, must automatically set to the current date and time and automatically updates with each update.

This model also must redefine the `__str__()` method so that it returns the `name` attribute.

In this app, you will also create a view that must be accessible at the following address: `127.0.0.1:8000/ex09/display`.
This view must display all the characters' names, their homeworld as well as the climate, which is `windy` or moderately `windy` sorted in character's name alphabetical order in an `HTML` table.

If there is no available data, the view must display the following text: `"No data available, please use the following command line before use:"` followed by a command line.

This command line must be the one to be executed from the root of your repo in order to insert all the data included in the `ex09_initial_data.json` file (provided with today's resources) in the models previously created.

You will have to provide this files with your repo.

`During the evaluation, the migration will be carried out before the tests.`

# Chapter XIII

# Exercise 10

|  | Exercise 10 |
|---|---|
| Exercise 10: ORM - Many to Many | |
| Turn-in directory : $ex10/$ | |
| Files to turn in : | |
| Allowed functions : | |

Create a new Django app named `ex10` and create 3 models inside:

- `Planets` and `People`: Both these models must be identical to the ones in `ex09`.
- `Movies`: This model must be identical to the one in the `ex01` except you must add the field `characters`.
  This is a `"many to many"` type with the `People` model. It allows the listing of every character in a film included in the `People` table.

The necessary fixtures to populate the models are included in the `ex10_initial_data.json` file provided with today's resources.

In this app, you will also create a view accessible via the following URL: `127.0.0.1:8000/ex10`. It must display a form featuring these mandatory fields:

- `Movies minimum release date` : date
- `Movies maximum release date` : date
- `Planet diameter greater than` : number
- `Character gender`: drop-down list showing the different values available in the `gender` field of the People model. The same value shouldn't be featured twice.

Once validated, the view must search, return and display the result(s).

A result is a character whose gender matches the `'character gender'` field, with a film they're in, if the film was released between `Movies minimum release date` and `Movies maximum release date`, their planet if its diameter is greater or equal `Planet diameter greater than`.

If your research gives no result, the message `"Nothing corresponding to your research"` must appear. Each result must be displayed on a line with these elements (not necessarily in this order):

- Character's name
- Their gender
- Film title
- Name of homeworld
- Homeworld's diameter

For instance: the results for the female characters whose films were released between "1900-01-01" and "2000-01-01" and whose homeworld has a diameter greater than 11000 are:

- A New Hope - Leia Organa - female - Alderaan - 12500
- The Phantom Menace - Padmé Amidala - female - Naboo - 12120
- Return of the Jedi - Leia Organa - female - Alderaan - 12500
- Return of the Jedi - Mon Mothma - female - Chandrila - 13500
- The Empire Strikes Back -Leia Organa - female - Alderaan - 12500

Several characters can be in the same film and one character can appear in several films. This is called a many to many relation. In this case, an intermediate table must be created between these tables. Each row of this intermediate table is a (unique) cross-reference: the first references a row in the films' table. The second references a row in the characters' table (or the other way round). Once your models are built and your migrations achieved, you will be able to see this table via the postgre console.

# Chapter XIV

# Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

`The evaluation process will happen on the computer of the evaluated group.`