Gestion de la base de données

1. Suppression et création de la base PostGIS

a. Suppression de la base

Pour se connecter à la base Postgres :
1. Ouvrir un terminal de commande
2. *Se connecter à l'environnement virtuel :*

```
$ ssh sysop@pitonpartage
$ cd /opt/mapobs
$ source /opt/mapobs/bin/activate
$ cd mapobs
```

Se connecter en superutilisateur :

```
(mapobs) sysop@pitonpartage:/opt/mapobs/mapobs$ su root
```

Pour se connecter à postgres :

```
root@pitonpartage:/opt/mapobs/mapobs$ sudo -u postgres psql
```

Pour se connecter à la base MAPOBS :

```
root@pitonpartage:/opt/mapobs/mapobs$ sudo -u postgres psql mapobs
```

Pour supprimer la BDD, il suffit de taper la commande suivante dans Postgres :

```
postgres=# DROP DATABASE mapobs;
```

Si un message d'erreur signale que la base est en cours d'utilisation, tapez la commande suivante :

```
postgres=# SELECT pid, pg_terminate_backend(pid)
postgres=# FROM pg_stat_activity
postgres=# WHERE dataname = current_database() AND pid <> pg_backend_pid();
```

b. Création de la base :

Pour créer une base de données mapobs, il suffit de taper la ligne de commande suivante dans Postgres :

```
postgres=# CREATE DATABASE mapobs;
```

Pour attribuer à l'utilisateur sysop l'ensemble des droits sur la base :

```
postgres=# ALTER ROLE sysop SUPERUSER;
postgres=# ALTER ROLE sysop SET client_encoding TO 'utf8';
postgres=# ALTER ROLE sysop SET default_transaction_isolation TO 'read committed';
postgres=# ALTER ROLE sysop SET timezone TO 'UTC';

postgres=# GRANT ALL PRIVILEGES ON DATABASE mapobs TO sysop;

postgres=# ALTER DATABASE mapobs OWNER to sysop;
```

Pour quitter la base :

```
postgres=# \q
```

Pour repasser en utilisateur sysop :

```
root@pitonpartage:/opt/mapobs/mapobs$ su sysop
```

2. Peuplement de la base de données :

a. Création des modèles

Une fois la base de données créée, afin de créer les tables présentes dans le fichier models.py, quittez postgres et tapez la commande suivante :

```
(mapobs) sysop@pitonpartage:/opt/mapobs/mapobs$ python manage.py makemigrations app
```

S'il s'agit d'une remise à zéro de la base, une erreur devrait apparaître. Elle est dû au fait que l'application va lire les fichiers faisant appel aux modèles alors qu'ils ne sont pas encore créés. Pour éviter cette erreur, commentez les lignes du fichier mapobs/urls.py (mapobs étant le nom de notre projet Django) se trouvant entre les crochets de la fonction *urlpatterns*.

Tapez ensuite :

```
python manage.py migrate
```

Pour se connecter à l'application, il faut réinitialiser le mot de passe (même si celui-ci reste inchangé) en tapant la commande suivante :

```
python manage.py createsuperuser
```

N'oubliez pas de décommenter les lignes commentées dans mapobs/urls.py

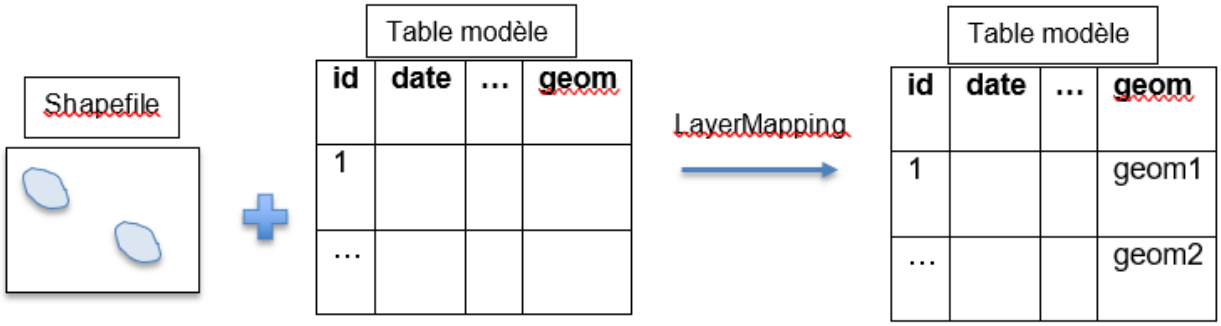
b. Conversion de shapefile ESRI en entités PostGIS

La plupart des données de l'observatoire sont stockées dans des shapefile. On peut trouver des couches de coulées, fissures et cônes.

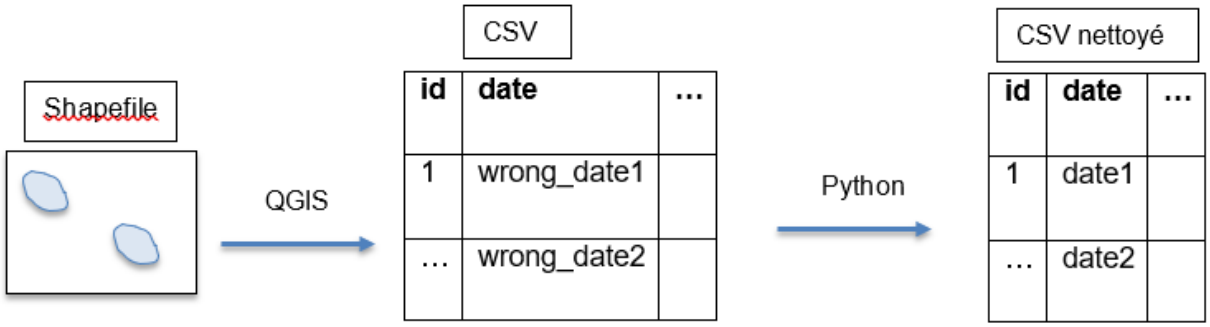
Afin de les ajouter au modèle de données, j'ai utilisé l'utilitaire d'importation de données LayerMapping de Django qui permet de faire correspondre le contenu d'une couche vectorielle à un modèle Django. Il faut pour cela lui fournir le modèle à remplir, le chemin vers le fichier shape et un dictionnaire, dans lequel, les clés correspondent aux champs du modèle et les valeurs aux champs de la source de données.

Les couches dont on dispose possèdent des attributs de date hétérogènes. On trouve des dates au format DD/MM/YYYY, d'autres au format YYYY ou encore sous forme de chaîne de caractère (ex : Début Août 1956). Celles de notre modèle sont au format DateTime (YYYY/MM/DD hh :mm) qui n'est pas disponible sous QGIS, le mapping de ces champs s'est donc avéré impossible.
La méthode choisie pour lier le shapefile au modèle est donc la suivante :

- On commence par ajouter au modèle la géométrie des entités grâce au LayerMapping.



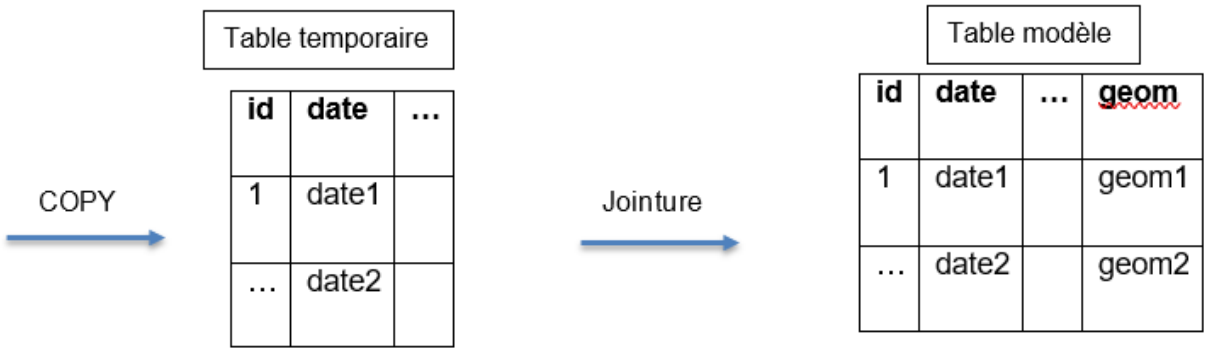
- Sous QGIS, on enregistre le shapefile au format CSV.



- On modifie ce CSV grâce à des fonctions Python que j'ai rédigé qui mettent les dates au format correct et qui ne gardent que les champs utiles au modèle. Si le temps n'était pas défini, il a été mis par défaut à 00 :00. De même si on ne possède que la donnée de l'année, on définit la date au premier jour du premier mois de l'année.

- Dans la base de données, on crée une table temporaire qui possède l'ensemble des champs du fichier tout juste généré et dans laquelle on copie son contenu.

- On met à jour la table du modèle grâce à une jointure des entités par l'identifiant



Pour reproduire cette méthode :
Avant toute chose, vérifiez la validité des géométries des shapefile de coulées, fissures et cônes. Pour cela, dans QGIS, lancez dans l'onglet Vecteur > Géométries > Vérifier la validité. Enregistrez la couche de géométries valides.

A l'aide de FileZilla, copier les shapefile adéquats dans l'environnement virtuel.
Depuis l'environnement virtuel, modifiez le fichier app/load.py, en y renseignant les chemins où sont enregistrés les Shapefile (shp_datatype : en remplaçant datatype par flow, fissure ou cone).

Afin d'ajouter les géométries des entités dans la base PostGIS, connectez-vous au shell Python en lançant la commande suivante :

```
>> python manage.py shell
```

Puis dans le shell tapez :

```
>>> from app import load
>>> load.run_datatype()(en remplaçant datatype par flow, fissure ou cone)
```

La fonction run_datatype va enregistrer les géométries d'un Shapefile dans la table PostGIS correspondante grâce au LayerMapping Django.

Pour ajouter les autres attributs, enregistrer le shapefile au format CSV avec QGIS : clic droit sur la couche > Enregistrer sous > Mettre le format en CSV séparé par une virgule.
Supprimer la colonne cb_ids du fichier CSV. Il s'agit d'une clé étrangère qui oblige à la création d'une table intermédiaire supplémentaire.

I. Flow

Pour ajouter la couche des coulées, dans la base postgres tapez : à noter qu'on ne crée pas de colonne géométrie dans la base.

```
mapobs=# CREATE TABLE "flow_temp" ("flow_id" serial NOT NULL PRIMARY KEY, "stime"
timestamp with time zone NULL, "stime_bc" integer NULL, "stime_unc" timestamp with
time zone NULL, "etime" timestamp with time zone NULL, "etime_bc" integer NULL,
"etime_unc" timestamp with time zone NULL, "final_vers" boolean NOT NULL, "com"
varchar(255) NULL, "loaddate" timestamp with time zone NULL, "pubdate" timestamp
with time zone NULL, "cc_load_id" integer NULL, "cc_pub_id" integer NULL,
"fissure_id" integer NULL);


mapobs=# COPY flow_temp(stime, stime_bc, stime_unc, etime, etime_bc, etime_unc,
final_vers, loaddate, pubdate, fissure_id, cc_load_id, cc_pub_id, com, flow_id)
FROM '/home/sysop/CouleesOVPF/couleesOVPFvalides.csv' DELIMITER ',' CSV HEADER;

mapobs=# UPDATE flow
SET stime = flow_temp.stime, stime_bc = flow_temp.stime_bc, etime =
flow_temp.etime, etime_bc = flow_temp.etime_bc, etime_unc = flow_temp.etime_unc,
final_vers = flow_temp.final_vers, com = flow_temp.com, loaddate =
flow_temp.loaddate, pubdate = flow_temp.pubdate
FROM flow_temp
WHERE flow.flow_id = flow_temp.flow_id;
```

Pour compléter la date de publication et la date d'ajout, lancez les commandes suivantes :

```
mapobs=# UPDATE flow SET loaddate=current_timestamp;
mapobs=# UPDATE flow SET pubdate=current_timestamp;
```

II. Fissure

Un premier nettoyage des dates a été effectué manuellement. En effet, certaines dates se présentaient sous la forme

- *« Aout 1956 » ce qui a été modifié par 01/08/1956.*
- *Les dates avec un intervalle de plusieurs mois ont vu leur opentime défini au 1er jour du 1er mois et leur opentime_bc au 1er jour du 2^{e} mois.*

Pour adapter totalement les dates de la couche de fissures tapez :

```
>>> from app import utils
>>> utils.run_fissure('path_to_input_cone_file', 'path_to_the_output_file')


Ex :
>>> utils.run_fissure('/home/sysop/FissuresOVPF/FissuresOVPF.csv',
'/home/sysop/FissuresOVPF/FissuresOVPF_correct_date.csv')
```

Une fois le CSV généré, dans la table mapobs tapez :

```
mapobs=# CREATE TABLE "fissure_temp" ("fissure_id" serial NOT NULL PRIMARY KEY,
"opentime" varchar(255) NULL, "opentime_unc" varchar(255) NULL, "com" varchar(255)
NULL, "com4" varchar(255) NULL, "com5" varchar(255) NULL);

mapobs=# COPY fissure_temp(sfissure_id, opentime, opentime_unc, com, com4, com5)
FROM '/home/sysop/FissuresOVPF/FissuresOVPF_correct_date.csv' DELIMITER ',' CSV
HEADER;
```

```
mapobs=# UPDATE fissure
SET opentime=nullif(fissure_temp.opentime,'')::timestamptz,
opentime_unc=nullif(fissure_temp.opentime_unc,'')::timestamptz,
com=fissure_temp.com, com4=fissure_temp.com4, com5=fissure_temp.com5
FROM fissure_temp
WHERE fissure.fissure_id=fissure_temp.fissure_id;
```

Ne pas oublier de définir les loaddate et pubdate à la date et au temps courant.

Si jamais le besoin d'ajouter un autre type d'entité apparaît, taper la commande suivante :

```
python manage.py sqlmigrate app 0001
```

Elle affiche les commandes postgres utilisée pour créer les modèles. Cherchez la table voulu, n'oubliez pas de supprimer la colonne géométrie.

III. Cone

Pour les cônes, le shapefile utilisé n'étant pas au bon format, on ne dispose que de l'année d'accrétion. Une fonction a donc été écrite et écrit un nouveau fichier CSV dans lequel la date d'accrétion est au 1er jour du 1er mois de l'année à l'heure 00 :00. Pour générer ce nouveau fichier :

```
>>> from app import utils
>>> utils.run_cone('path_to_input_cone_file', 'path_to_the_output_file')
```

```
Ex :
>>> utils.run_cone('/home/sysop/ConesOVPF/coneOVPF.csv',
'/home/sysop/ConesOVPF/conesOVPF_correct_date.csv')
```

Une fois le CSV généré, dans la table mapobs tapez :

```
mapobs=# CREATE TABLE "cone_temp" ("cone_id" serial NOT NULL PRIMARY KEY, "name"
varchar(60) NULL, "accretion_date" varchar(255) NULL, "com" varchar(255) NULL,
"com2" varchar(255) NULL, "com3" varchar(255) NULL);
```

Ici aussi on définit des attributs censés être timestamp en chaîne de caractère pour palier le fait que certaines colonnes du csv sont vides.

```
mapobs=# COPY cone_temp(cone_id, name, accretion_date, com, com2, com3) FROM
'/home/sysop/ConesOVPF/conesOVPF_correct_date.csv' DELIMITER ',' CSV HEADER;
```

```
mapobs=# UPDATE cone SET name=cone_temp.name,
accretion_date=nullif(cone_temp.accretion_date,'')::timestamptz, com=cone_temp.com,
com2=cone_temp.com2, com3=cone_temp.com3 FROM cone_temp WHERE
cone.cone_id=cone_temp.cone_id;
```

Ne pas oublier de définir les loaddate et pubdate à la date et au temps courant.

IV. Cas particulier des éruptions

La table des éruptions a été peuplée à partir du tableau de Geneviève qu'il a d'abord fallu simplifier afin de ne conserver que les attributs intéressants.
Les attributs sélectionnés sont présentés sur la figure suivante :

		CHARACTERISTICS OF ERUPTIONS, SEISMIC CRISES AND COLLAPSES																
Event		Eruption timing						Fissures			Seismic Crisis start dur			Seismic Swarm				Comment
		Start		End		DUR		N_f	vol	flux	SSC		DSC	SSS	ESS	DSS		
F	date	year.day	hh:mm	date	hh:mm	days	notes		10^6 m^3	$m^3.s^{-1}$	year.day	hh:mm	hh:mm	hh:mm	hh:mm	hh:mm		

Un script a été écrit afin de nettoyer les données et de les mettre au bon format. Par exemple, les dates n'étant pas au format DateTime ont été modifiées. *Pour le lancer, dans le shell Python tapez :*

```
>>> from app import utils
>>> utils.run('path_to_input_gene_file', 'path_to_the_output_file')

Ex :
>>> utils.run('/home/sysop/TableauEruptionGene_Simpli.csv',
'/home/sysop/TableauEruptionGene_Final.csv')
```

Les colonnes du tableau final doivent ensuite être enregistrées dans une table PostGIS qu'il faut créer. *Pour la créer, il faut taper dans la base de données mapobs la commande suivante :*

```
mapobs=# CREATE TABLE "gene_tab" ("event_f" varchar(255) NULL, "event_date"
varchar(255) NULL, "ed_stime" varchar(255) NULL, "ed_etime" varchar(255) NULL,
"com" varchar(255) NULL, "Nf" varchar(255) NULL, "vol" varchar(255) NULL, "flux"
varchar(255) NULL, "seismic_crisis_stime" varchar(255) NULL, "seismic_crisis_etime"
varchar(255) NULL);
```

Ici, toutes les colonnes de type timestamp with time zone, float ou integer sont créées au format varchar. En effet, dans le csv, ces attributs présentent pour certaines lignes des valeurs nulles qui sont considérées par Postgres comme la valeur `""` ce qui n'est pas une valeur valide pour un timestamp. Python ne faisant aucune distinction entre une valeur mise à *None* et une chaîne de caractère vide, il est impossible de modifier cette valeur. On définit donc le type de la colonne en chaîne de caractère que l'on va ensuite caster en *timezone* au moment de faire la jointure avec la table des éruptions.
Afin de copier le csv dans la table PostGIS :

```
mapobs=# COPY gene_tab FROM '/home/sysop/TableauEruptionGene_Final.csv' DELIMITER
',' CSV HEADER;
```

C'est ici que l'on caste les dates en timestamp with time zone.

```
mapobs=# INSERT INTO ed(ed_stime, ed_etime, ed_com) SELECT
nullif(gene_tab.ed_stime,'')::timestamptz,
nullif(gene_tab.ed_etime,'')::timestamptz, gene_tab.com FROM gene_tab WHERE
gene_tab.event_f LIKE 'E%';
```

Ne pas oublier de définir les loaddate et pubdate à la date et au temps courant.

V. Seismic event

Les données concernant les crises sismiques se trouvent dans la table gene_tab. Pour compléter la table des fissures tapez dans la base mapobs :

```
mapobs=# INSERT INTO sd_ev(sd_ev_stime, sd_ev_etime) SELECT
nullif(gene_tab.seismic_crisis_stime,'')::timestamptz,
nullif(gene_tab.seismic_crisis_etime,'')::timestamptz FROM gene_tab WHERE
gene_tab.event_f LIKE 'SC%';
```

VI. Mise à jour des tables

a. Flux des coulées

La table Flow dispose d'un attribut flux. Sa valeur est stockée dans la table de Geneviève.
Pour récupérer sa valeur et l'ajouter à la table des coulées tapez:

```
mapobs=# UPDATE flow SET flux=nullif(gene_tab.flux,'')::float FROM gene_tab WHERE
flow.stime=nullif(gene_tab.event_date,'')::timestamptz;
```

b. Associer les éruptions aux fissures

Pour associer les éruptions aux fissures par la date tapez :

```
mapobs=# UPDATE fissure as f SET ed_id = ed.ed_id FROM ed WHERE (ed.ed_stime,
ed.ed_etime) OVERLAPS (f.opentime,f.opentime);
```

On utilise ici, une méthode qui permet de vérifier que deux dates se superposent. Ainsi si la durée d'une éruption se superpose à l'apparition d'une fissure, on lui associe cette fissure.
778 fissures sur 993 ont été associées à une éruption.

c. Associer les coulées aux fissures

Pour associer les coulées aux fissures tapez :

```
mapobs=# UPDATE flow SET fissure_id = fissure.fissure_id FROM fissure WHERE
(flow.stime, flow.etime) OVERLAPS (fissure.opentime, fissure.opentime);
```

71 coulées sur 87 ont pu être associées à une fissure.
Ces résultats pourraient être dû au fait que la table des coulées (jusqu'à 2017) est plus à jour que celle des fissures. Aussi, certaines entités n'ont pas de date de début ou de formation.

d. Associer les cônes aux fissures

Afin d'associer une fissure à un cône, il faut que ce dernier intersecte une seule et unique fissure. La fissure peut en revanche intersecter plusieurs cônes comme sur la figure suivante, si on considère que la forme bleue représente un cône et chaque ligne, une fissure:

On commence par créer une table qui contient toutes les géométries qui s'intersectent :

```
mapobs=# CREATE TABLE q1 AS (SELECT f.fissure_id, c.cone_id FROM fissure AS f, cone
AS c WHERE ST_Intersects(f.geometry, c.geometry));
SELECT 730
```

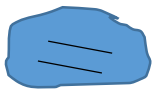
On stocke dans une nouvelle table le nombre d'occurrence d'un cône dans la table précédente :

```
mapobs=# CREATE TABLE q2 AS (SELECT c.cone_id, COUNT(*) FROM fissure as f, cone as
c WHERE ST_Intersects(f.geometry, c.geometry) GROUP BY cone_id);
SELECT 629
```

On ne garde les cônes qui intersectent une unique fissure :

```
mapobs=# CREATE TABLE q3 AS (SELECT q1.fissure_id, q1.cone_id FROM q1, q2 WHERE
q1.cone_id = q2.cone_id AND q2.count = 1 ORDER BY q1.cone_id);
SELECT 543
```

On a ainsi dans cette table l'ensemble des cônes qui intersectent une unique fissure. Dans les 2 figures suivantes, les deux cônes intersectent chacun plusieurs fissures alors que chaque fissure intersecte un unique cône, on ne sait donc pas de quelle fissure est issue chaque cône et ces derniers ne sont pas appariés :



Grâce à une jointure, on est ainsi capable d'attribuer une fissure à certains cônes :

```
mapobs=# UPDATE cone SET fissure_id=q3.fissure_id FROM q3 WHERE cone.cone_id =
q3.cone_id;
UPDATE 543

On a ainsi pu apparier 543 des 653 cônes.
```

e. Associer les évènements sismiques aux éruptions

Pour associer un évènement sismique à une éruption, tapez la commande suivante :

```
mapobs=# UPDATE sd_ev SET ed_id = ed.ed_id FROM ed WHERE (sd_ev_stime, sd_ev_etime)
OVERLAPS (ed.ed_stime, ed.ed_etime);
```

Avec cette technique on ne parvient à apparier que 2 des 30 évènements sismiques.

f. Ajouter des entités de volume

Pour créer des entités volume à partir du volume des coulées contenu dans la table de Géneviève :

On commence par créer une table intermédiaire volume_temp

```
CREATE TABLE "volume_temp"("flow_id" integer NULL, "vol" float NULL, ed_stime ti-
mestamp with time zone NULL, ed_etime timestamp with time zone NULL);
```

On ajoute à cette table les données de volume, de durée et de fin d'éruption :

```
INSERT INTO volume_temp(vol, ed_stime, ed_etime) SELECT nullif(g.vol,'')::float,
nullif(g.ed_stime,'')::timestamptz, nullif(g.ed_etime,'')::timestamptz FROM
gene_tab as g WHERE vol!='';
INSERT 79
```

On y ajoute les identifiants des coulées dont les dates de début et de fin se superposent avec celles des éruptions :

```
UPDATE volume_temp SET flow_id=f.flow_id FROM flow as f WHERE (f.stime, f.etime)
OVERLAPS (ed_stime, ed_etime);
UPDATE 44
```

On insère dans la table de volume les lignes pour laquelle le flow_id n'est pas nulle (les volumes ayant trouvé une correspondance dans la table des coulées) :

```
INSERT INTO volume(value_measured, flow_id) SELECT nullif(v.vol, '')::float,
v.flow_id FROM volume_temp as v WHERE flow_id;
INSERT 44
```

Ne pas oublier de définir les loaddate et pubdate à la date et au temps courant.