1. Introduction

Commencez par créer un dossier TP4 dans votre compte, ouvrez le navigateur de fichiers dedans.

Ouvrez un terminal et tapez `ssh hadoop`, connectez-vous, puis tapez `cd TP4`.

Le but du TP est d'apprendre le langage Pig Latin qui permet de spécifier des jobs MapReduce plus facilement qu'en Java. Pig Latin est un drôle de langage [1]. Certaines de ses instructions ressemblent à SQL mais le principe est différent. SQL décrit les données à sélectionner tandis que Pig spécifie les traitements à effectuer. En SQL, vous ne savez pas comment les calculs sont faits : utilisation ou non des index, boucles internes... c'est caché. En Pig, au contraire, c'est vous qui spécifiez la séquence des calculs.

Pig repose sur plusieurs mécanismes :

- Les données sont appelées relations. Ce sont des listes de n-uplets, on peut voir ça comme des tables.
- Chaque relation est modélisée par un schéma. Le schéma indique les noms et types de champs des n-uplets.
- Les instructions Pig Latin transforment les relations. Seules les instructions d'affichage déclenchent effectivement les calculs.

La documentation de Pig est sur cette page. Choisir à gauche par exemple Pig Latin Basics pour des explications sur les éléments du langage.

2. Tutoriel

On va travailler avec le fichier des arbres remarquables de Paris. Il est sur HDFS `/share/paris/arbres.csv`. C'est un fichier CSV, ses colonnes sont séparées par un `;` et contiennent :

champ	exemple	signification
géopoint	`(48.8571, 2.29533)`	latitude, longitude
arrondissement	`7`	
genre	`Maclura`	
espèce	`pomifera`	
famille	`Moraceae`	
année plantation	`1935`	
hauteur	`13.0`	
circonférence		certains champs sont vides
adresse	`Quai Branly`	
nom commun	`Oranger des Osages`	
variété		
objectid	`6`	identifiant de l'arbre
nom_ev	`Parc du Champ de Mars`	

[1] (extrait de wikipedia) Le pig latin est un argot principalement utilisé en anglais, équivalent au louchébem (largonji) en français : un mélange de verlan (pour le renversement des syllabes) et de javanais (pour l'ajout systématique d'une syllabe). Donc, c'est un langage pour écrire des cochonneries.

Son schéma est donc :

```
(geopoint:tuple(lat:double,lon:double), arron:long, genre:chararray,
espece:chararray, famille:chararray, annee:long, hauteur:double,
circonf:double, adresse:chararray, commun:chararray, variete:chararray,
id:long, nomev:chararray)
```

On doit employer des `double` pour les réels, des `long` pour les entiers et des `chararray` pour les chaînes. Vous pouvez essayer des `float` et des `int` mais vous aurez des avertissements de conversions.

2.1. Chargement de données

Voici donc comment charger ce fichier dans Pig. Lancez le shell de Pig en tapant `pig`, puis copiez-collez cette instruction dedans :

```
arbres_bruts = LOAD '/share/paris/arbres.csv'                                --**--
    USING PigStorage(';')
    AS (geopoint:tuple(lat:double,lon:double),arron:long,genre:chararray,
        espece:chararray,famille:chararray,annee:long,hauteur:double,circonf:double,
        adresse:chararray,commun:chararray,variete:chararray,id:long,nomev:chararray);
```

Le commentaire `--**--` vous signale que cette instruction est indispensable pour la suite de ce tutoriel. D'autres instructions vont ré-utiliser cette relation.

Cette instruction est généralement la première d'un programme Pig. Elle déclare une relation, son schéma et son contenu et met le tout dans un *alias* qu'on peut employer dans les calculs suivants.

Deux instructions sont absolument indispensables à connaître :

- `DESCRIBE arbres_bruts;` qui affiche le schéma de la relation. C'est à faire pour vérifier à chaque fois qu'on construit une relation ayant un schéma différent (projection, groupement...).
- `DUMP arbres_bruts;` qui affiche le contenu de la relation. C'est uniquement cette instruction qui lance les calculs. Ça fait créer un job *MapReduce*.

Chaque fois que vous lancerez un *MapReduce*, il va y avoir un fichier de log. Vous pourrez tous les supprimer à la fin du TP.

2.2. Filtrage

Le problème de ce fichier, c'est qu'il y a une ligne de titres. Si on la traite avec les données, ça risque de casser les calculs. Il faut donc filtrer la relation et la condition est d'avoir, par exemple un geopoint valide. Pour cela, il suffit de tester si le champ `geopoint` vaut `NULL` :

```
arbres = FILTER arbres_bruts BY geopoint IS NOT NULL;                        --**--
```

Attention, ce n'est pas la même chose que tester si `geopoint` est une chaîne vide.

Vous devrez vérifier le schéma (`DESCRIBE`), puis afficher le résultat (`DUMP`).

Pig affiche deux `WARN` (`no job jar file set` à ignorer) à chaque fois qu'il lance un *MapReduce*. C'est comme ça que vous pouvez savoir ce qu'il fait. Une autre manière consiste à afficher le plan sous-jacent, mais c'est incompréhensible : `EXPLAIN arbres;` Cherchez le paragraphe concernant le *Map Reduce Plan*.

Sortez du shell de Pig en tapant `exit` ou `CTRL D`. Placer les deux instructions `LOAD` puis `FILTER` dans un fichier appelé `arbres.pig`. Revenez dans le shell de Pig et tapez `run arbres.pig`. Ensuite vous pourrez continuer à taper des instructions dans le shell.

Écrivez un filtre qui ne laisse passer que les lignes ayant à la fois une hauteur et une année de plantation. Les connecteurs logiques sont `AND`, `OR`, `NOT`. Les opérateurs de comparaison sont les mêmes qu'en Java : `==` et `!=` par exemple.

Pour accéder aux champs d'un tuple, lui-même champ d'une relation, on écrit `champ.souschamp`. Par exemple, les arbres situés à l'ouest de 2.45° :

```
resultat = FILTER arbres BY geopoint.lon > 2.45;
```

N'oubliez pas que pour afficher les n-uplets, il faut employer `DUMP`.

Il est indispensable de donner des noms parlants aux relations, en particulier quand elles doivent être ré-utilisées ensuite. Ce n'est pas le cas ici parce qu'on ne va rien faire avec le résultat.

2.3. Classement

Pour classer les n-uplets de la relation sur un critère :

```
arbres_avec_annee = FILTER arbres BY annee IS NOT NULL;                --**--
arbres_classes = ORDER arbres_avec_annee BY annee ASC;
```

Vous devrez vérifier le schéma (`DESCRIBE`), puis afficher le résultat (`DUMP`).

Pour classer dans l'ordre décroissant, il faut mettre `DESC` à la place de `ASC`.

2.4. Premiers n-uplets

Pour ne garder que les 5 premiers arbres parmi ceux classés par année :

```
vieux_arbres = LIMIT arbres_classes 5;
```

Les deux instructions précédentes : `ORDER` et `LIMIT` permettent de sélectionner les meilleurs n-uplets sur un critère. Une variante consiste à rajouter un rang à chaque n-uplet, basé sur le critère, puis à filtrer les résultats :

```
rangs_arbres = RANK arbres_avec_annee BY annee ASC;
DESCRIBE rangs_arbres;
vieux_arbres2 = FILTER rangs_arbres BY rank_arbres_avec_annee <= 5L;
```

Avez-vous étudié le schéma de la relation `rangs_arbres` ? Le rang est un champ qui a été rajouté et il a un nom particulier, basé sur la relation initiale. Ce champ est de type `long`, donc il est préférable de le comparer à entier long.

Avez-vous examiné les résultats ? Les arbres qui ont la même année reçoivent le même rang. Ça peut être souhaitable... ou non. D'autre part, dans cette réponse, les rangs sont présents dans

les données. On peut faire une projection pour les enlever.

2.5. Projection et calculs

Une projection consiste à ne garder que certaines colonnes dans une relation. On peut aussi faire des calculs au passage :

```
genres_ages = FOREACH arbres GENERATE genre, 2019L-annee;
```

Vous devrez vérifier le schéma, puis afficher le résultat.

Les champs sont nommés soit par leur nom dans le schéma, soit par leur position `$n°` avec n° commençant à 0 pour le premier champ :

```
genres_ages = FOREACH arbres GENERATE $2, 2019L-$5;
```

Il est possible de renommer les champs dans le résultat :

```
genres_ages = FOREACH arbres GENERATE genre, 2019L-annee AS age;          --**--
DESCRIBE genres_ages;
```

Le cours mentionne une syntaxe basée sur `..` (2 points) pour écrire facilement une énumération de champs. Voici comment enlever le rang de `vieux_arbres2` sans lister tous les champs :

```
vieux_arbres2_ok = FOREACH vieux_arbres2 GENERATE geopoint ..;
```

2.6. Éléments distincts

Voici comment éliminer les doublons (n-uplets entièrement égaux) dans une relation :

```
annees_toutes = FOREACH arbres_avec_annee GENERATE annee;
annees = DISTINCT annees_toutes;
```

Vous remarquerez que les années ressortent classées. Est-ce que vous avez vérifié qu'il y a quelques années en double dans `annees_toutes` ?

2.7. Groupement

Les groupements sont les concepts les plus complexes de Pig. La complexité vient d'une part la transformation elle-même, et aussi de ses possibilités d'emplois.

```
genres_ages_connus = FILTER genres_ages BY age IS NOT NULL;               --**--
DESCRIBE genres_ages_connus;
ages_par_genre = GROUP genres_ages_connus BY genre;                       --**--
DESCRIBE ages_par_genre;
```

Analysez bien le schéma de la relation produite :

```
ages_par_genre: {group: chararray,genres_ages_connus: {(genre: chararray,age: long)}}
```

C'est un ensemble de deux éléments. Le premier s'appelle `group` et c'est un entier. En fait, c'est le genre issu de la relation d'entrée. Le second est appelé `genres_ages_connus`, c'est à dire

exactement comme la relation qu'on vient de grouper. Regardez bien le type de ce second champ. Il y a des `{...}` pour signaler que c'est un ensemble de tuples `(genre, age)`.

Il vous reste à afficher le contenu de cette relation pour voir ce que sont les groupements. Voici un extrait. C'est un peu redondant car on retrouve les n-uplets ayant tous le même genre associé à leur âge :

```
(Celtis,{(Celtis,110)})
(Ginkgo,{(Ginkgo,121),(Ginkgo,137),(Ginkgo,103),(Ginkgo,151),(Ginkgo,123)})
(Corylus,{(Corylus,134),(Corylus,137)})
```

Une variante du groupement consiste à grouper tous les n-uplets dans un seul résultat.

```
ages = FOREACH genres_ages_connus GENERATE age;
DESCRIBE ages;
groupe_ages = GROUP ages ALL;                                    --**--
DESCRIBE groupe_ages;
```

Cette fois, le premier champ s'appelle `all` et le second champ regroupe tous les n-uplets (réduits ici à seulement l'âge des arbres).

2.8. Parcours d'un groupement

Lorsqu'on fait une boucle `FOREACH` sur une relation issue d'un groupement d'une autre relation, comme :

```
groupement = GROUP relation BY champ1;
resultat = FOREACH groupement GENERATE group, relation.champ2;
```

Pour la clause `GENERATE`, on dispose des champs :

- `group` qui vaut le nom du groupement courant, ce sont les différentes valeurs du `champ`
- `relation` qui contient la liste des n-uplets. On peut accéder à ses champs en écrivant `relation.champ`, et alors ça extrait la liste des valeurs.

```
resultat = FOREACH ages_par_genre GENERATE group, genres_ages_connus.age;
```

2.9. Agrégation

Le premier usage des groupements est pour faire différents calculs de synthèse sur les données : comptage, moyennes, maximas... On est obligé de grouper les données parce que les opérateurs ne travaillent pas sur des n-uplets différents ; il faut qu'ils soient dans la même liste.

Alors il faut faire quelque chose de très bizarre : grouper les données, puis faire une boucle dessus (une itération par groupe) et générer l'agrégation souhaitée sur les n-uplets du groupe.

```
nombre = FOREACH groupe_ages GENERATE COUNT(ages.age);
age_moyen = FOREACH groupe_ages GENERATE AVG(ages.age);
age_maximal = FOREACH groupe_ages GENERATE MAX(ages.age);
```

Si on avait fait un `GROUP ALL`, on n'obtiendra qu'une seule réponse, sinon ça sera une réponse par catégorie :

```
age_maximal = FOREACH ages_par_genre GENERATE group, MAX(genres_ages_connus.age);
```

Il est quasiment obligatoire d'afficher le schéma du groupement (`DESCRIBE`) pour bien comprendre ce qu'on calcule, comment s'appellent les champs.

2.10. Dégroupement

L'inverse de `GROUP` s'appelle `FLATTEN`. En fait, ça revient à générer un n-uplet par élément du groupe.

```
plat = FOREACH ages_par_genre GENERATE FLATTEN(genres_ages_connus);
```

Examiner le schéma pour voir que le nommage des champs est devenu compliqué.

2.11. Imbrications

Pour finir, le plus compliqué, les traitements imbriqués. Soit un groupement, par exemple `ages_par_genre`. Son schéma est :

```
ages_par_genre: {group: chararray,genres_ages_connus: {(genre: chararray,age: int)}}
```

On peut faire un parcours des valeurs ayant le même type :

```
tmp = FOREACH ages_par_genre {
    vieux = FILTER genres_ages_connus BY age > 150L;
    GENERATE group AS genre, COUNT(vieux) AS nombre;
}
resultat = FILTER tmp BY nombre>=2L;
```

Ce programme affiche les noms des genres d'arbres ayant au moins deux exemplaires de plus de 150 ans.

3. Travail à faire

Vous allez travailler avec le fichier des stations météo `/share/noaa/isd-history.csv`. Il décrit les stations météorologiques qui fournissent leurs données à la NOAA. Il n'y a pas de ligne de titre. Ses champs sont séparés par des `;` et sont :

champ	exemple	signification
id	`071180`	identifiant de la station
name	`LANNION`	nom de la station
country	`FR`	code pays
state		état américain
icao	`LFRO`	code aéroport international
lat	`48.754`	latitude
lon	`-3.472`	longitude
ele	`88.4`	altitude
begin	`(2001,09,19)`	date du premier relevé
end	`(2019,03,08)`	date du dernier

Vous devez créer un fichier `questionI.pig` pour chaque question, en remplaçant la lettre I par le numéro de la question. À chaque fois, faites ce que vous pouvez, faites pour le mieux.

Pour chacune de ces questions, veillez à ne pas traiter des données incorrectes ou incomplètes.

Votre premier travail va être de pouvoir lire le fichier en tant que relation. Vous avez donc son schéma à définir en fonction des données. La même instruction de chargement va être mise dans chacune de vos réponses.

1. Afficher les informations des stations nommées “LANNION”.
2. Afficher les informations des stations qui étaient actives en 1920. Vous verrez que leur répartition n’est pas homogène, et donc ça va être difficile de voir une évolution globale pour la Terre.
3. Afficher les codes des pays, en un seul exemplaire des stations de l’hémisphère sud.
4. Afficher le code du pays qui a le plus de stations, afficher aussi le nombre de stations qu’il possède.
5. Afficher le nombre total de stations différentes par leur code ICAO du fichier. De nombreuses stations sont des doublons (ou n’ont pas de code ICAO). Elles ont généré des relevés à des périodes discontinues.
6. Afficher l’altitude moyenne des stations.

4. Travail à rendre

Supprimez tous les fichiers de log de pig. Puis compressez le dossier tp4 en `tp4.tar.gz`. Il doit contenir les scripts pig demandés. Déposez l’archive sur Moodle.