Rapport SAÉ 2.04

parcoursup

Entrez dans l'enseignement supérieur

Partie BDD

POLLET Florian | FREVILLE Cyprien

Exercice 1: Comprendre les données

1.1) Le premier objectif de cet exercice est de déterminer le nombre de lignes dans les fichiers "fr-esr-parcoursup.csv". Pour cela, on utilisera une commande permettant d'afficher le nombre de lignes d'un fichier.

```
wc -l <nom du fichier>
```

Dans le cas présent, nous allons effectuer la commande suivante :

```
wc -l fr-esr-parcoursup.csv
```

On obtient le résultat suivant :

Nom du fichier	Nombre de lignes
fr-esr-parcoursup.csv	13870

Nom du fichier	Nombre de lignes sans entête
fr-esr-parcoursup.csv	13869

1.2) Une ligne dans notre fichier représente une formation ainsi que ses différentes informations comme sa capacité en termes de place, sa localisation et autres.

1.3) Il y a **118 colonnes,** nous allons utiliser la commande suivante afin d'en avoir la preuve :

```
head -n <nombre de ligne> <nom du fichier>
```

La commande "**head**" permet de commencer la sélection au début du fichier et "**-n <nombre de ligne>**" indique le nombre de lignes qu'il faut sélectionner.
Pour trouver le séparateur de champs il suffit de regarder le fichier, dans notre cas le séparateur est un point-virgule, ce séparateur permet de délimiter chaque colonne du tableau.
Nous allons effectuer la commande suivante :

```
head -n 1 fr-esr-parcoursup.csv | tr ";" "\n" | wc -l
```

Nous obtenons le résultat suivant :

```
118
```

En résumé pour compter le nombre de colonnes nous avons sélectionné la première ligne du fichier avec “**head -n 1 fr-esr-parcoursup.csv”** puis nous avons mis un passage à la ligne à chaque fois qu’il y a un point-virgule avec “**tr ";" "\n"** ” ce qui nous a permis de mettre les colonnes en ligne et donc de simplifier le comptage des colonnes puis pour compter les lignes on a utilisé “**wc -l**” ce qui nous a retourné 118 colonnes.

1.4) La colonne identifiant un établissement est la suivante :

Nom de colonne	Numéro de colonne
Code UAI de l'établissement	colonne n3

1.5) La colonne identifiant une formation est la suivante :

Nom de colonne	Numéro de colonne
cod_aff_form	colonne n110

1.6) Nous avons **49 lignes** qui sont des BUT en informatique, nous avons pu obtenir cette valeur en effectuant la commande suivante :

```
grep "BUT - Informatique" fr-esr-parcoursup.csv | wc -l
```

1.7) La colonne identifiant un département est la suivante :

Nom de colonne	Numéro de colonne
Code départemental de l’établissement	colonne n5

1.8) Nous envisageons de créer une table “import” temporaire dans postgreSQL grâce à la commande suivante :

```
\copy import from fr-esr-parcoursup.csv with (format CSV, delimiter ';',
HEADER);
```

1.9)

Le premier problème est la redondance, par exemple on peut avoir une dizaine de fois le même numéro de département ou la même commune. Le second problème est qu'il y a de nombreuses colonnes qui n'ont pas à être présentes, ce sont des statistiques qui sont calculables. Les données en trop alourdissent la table.

Exercice 1.2: Importer les données

2.1) Le fichier se trouve dans le rendu et se nomme bien dico.xls. Ouverture du fichier avec Libreoffice, "contrôle + A" puis cliquez sur la première ligne afin de la de-sélectionner sur le petit "1" à gauche , "control + X" puis collage spéciale, transposée. Après cela il faut écrire n1 dans la cellule A2 et la faire glisser jusqu'en bas.

2.2/2.3) Voir parcoursup.sql

La table import fut créée grâce à ce programme Java, ce qui nous as permis de gagner énormément de temps,elle prend le fichier "dico.clsx" comme base et met les types automatiquement :

```java
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.util.List;

public class Main {
    public static void main(String[] args) throws IOException {
        PrintWriter writer = new PrintWriter( new File( pathname: "output.txt"));
        writer.println("create table import(");

        File file = new File( pathname: "dico.csv");
        List<String> lines = Files.readAllLines(file.toPath(), StandardCharsets.UTF_8);
        String lastLine = lines.get(lines.size() - 1);
        int i = 1;
        for (String line : lines) {
            String[] array = line.split( regex: ",");
            System.out.println(array[array.length - 1]);
            if(line != lastLine){
                writer.println("    n"+i+"  "+array[array.length - 1]+" default null,");
            }else{
                writer.println("    n"+i+"  "+array[array.length - 1]+" default null");
            }
            i++;
        }

        writer.println(");");
        writer.close();

    }
}
```

2.4)\copy import from .\fr-esr-parcoursup.csv with (format CSV, delimiter ';', HEADER);

Le chemin est très important, cette commande est dans le document parcoursup.sql et copie les données du fichier suivant : "fr-esr-parcoursup.csv".

2.5)

a) **Combien il y a de formations gérés par ParcourSup ?**

```
SELECT COUNT(*) FROM import;
```

Nous obtenons la réponse suivante :

```
 count
-------
 13869
(1 ligne)
```

b) **Combien il y a d'établissements gérés par ParcourSup ?**

```
SELECT COUNT(DISTINCT n3) FROM import;
```

Nous obtenons la réponse suivante :

```
 count
-------
  3965
(1 ligne)
```

c) **Combien il y a de formations pour l'université de Lille ?**

```
SELECT COUNT(n110) FROM import WHERE n9='Lille';
```

Nous obtenons la réponse suivante :

```
 count
-------
   198
(1 ligne)
```

d) **Combien il y a de formations pour notre IUT ?**

```
SELECT COUNT(n110) FROM import WHERE n3='0597215X';
```

Nous obtenons la réponse suivante :

```
 count
```

```
-------
    10
(1 ligne)
```

Ici “0597215X” est le numéro de notre IUT trouvé dans les différents documents à notre disposition en effectuant des “contrôle + f”.

e) Quel est le code du BUT Informatique de l'université de Lille ?

```
SELECT n110 FROM import WHERE n9='Villeneuve-d''Ascq' AND n10='BUT
- Informatique';
```

Nous obtenons la réponse suivante :

```
 n110
------
 6888
(1 ligne)
```

f) Citez 5 colonnes contenant des valeurs nulles :

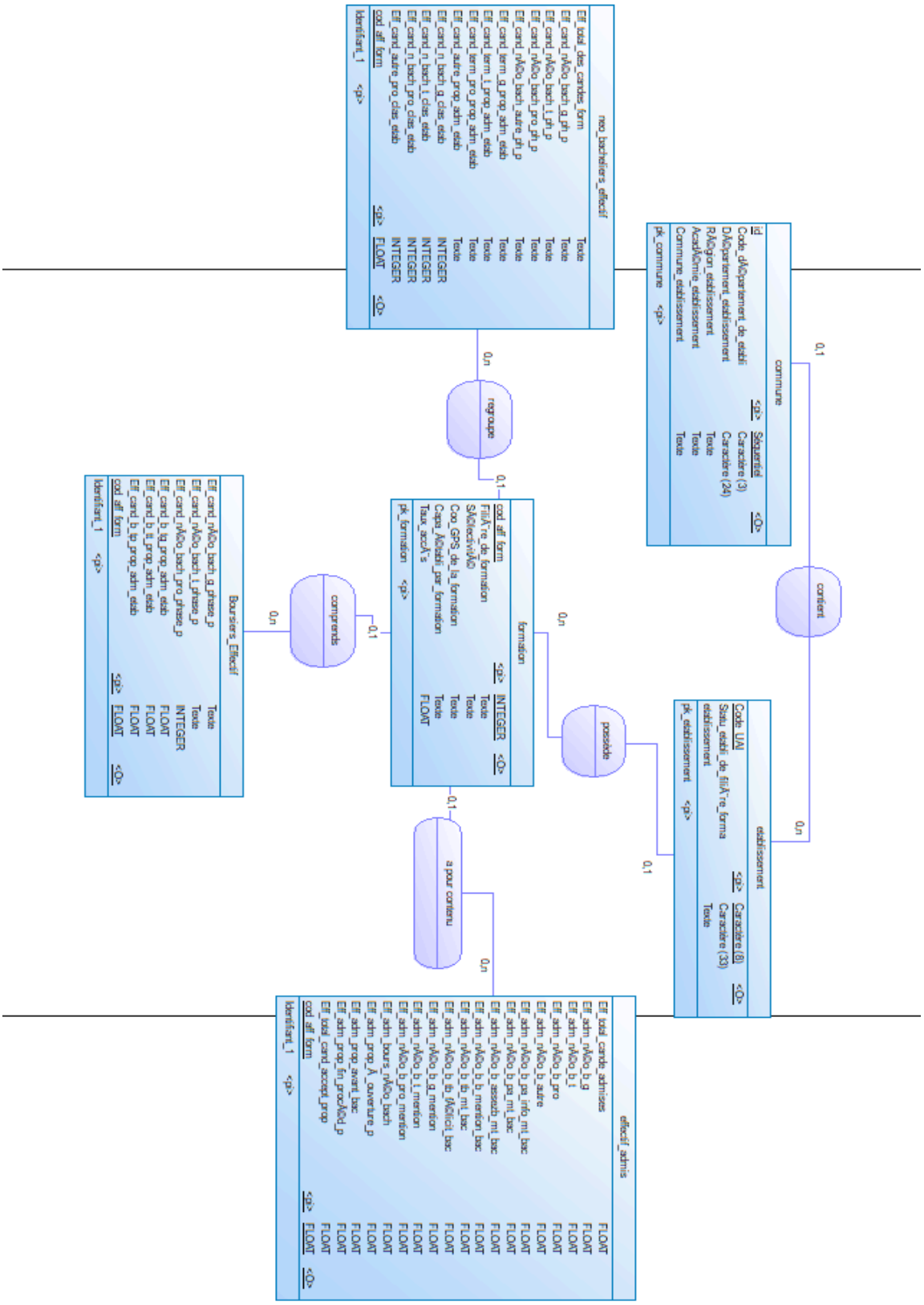
```
SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_NAME = 'import' AND COLUMN_DEFAULT IS NULL;
```

Nous obtenons la réponse suivante :

```
 column_name
-------------
 n1
 n4
 n7
 n8
 n9
(5 rows)
```

La requête n’est pas bonne ,nous avons eu beaucoup de mal à faire cette question, mais afin de trouver si une colonne contient une valeur nulle il faudrait pouvoir convertir un nom de colonne en colonne lors d’un where.

Exercice 2: Ventiler les données



Q2. Une question de taille !

1) **Quelle taille en octet fait le fichier récupéré ?**

```
wc -c fr-esr-parcoursup.csv
```

Nous obtenons la réponse suivante :

```
12423586 fr-esr-parcoursup.csv
```

`wc -c` nous donne la taille en octet d'un fichier.

2) **Quelle taille en octet fait la table import ?**

```
select pg_total_relation_size('import') as importsize;
```

Nous obtenons la réponse suivante :

```
 importsize
------------
   18644992
(1 row)
```

3) **Quelle taille en octet fait la somme des tables créées ?**

```
select (pg_total_relation_size('formation') +
pg_total_relation_size('etablissement') +
pg_total_relation_size('commune') +
pg_total_relation_size('effectif_admis') +
pg_total_relation_size('Boursiers_Effectif') +
pg_total_relation_size('neo_bacheliers_effectif') ) as newsize;
```

Nous obtenons la réponse suivante :

```
 newsize
---------
  9838592
(1 row)
```

Nous avons utilisé "pg_total_relation_size" afin de calculer la taille de chaque table et nous les avons additionnés.

4) **Quelle taille en octet fait la somme des tailles des fichiers exportés correspondant à ces tables ?**

```
\copy table to 'table.csv' with (format CSV, delimiter ';',
HEADER)
```

```
Puis

wc -c commune.csv

wc -c formation.csv

wc -c etablissement.csv

wc -c effectif_admis.csv

wc -c neo_bacheliers_effectif.csv

wc -c boursiers_effectif.csv
```

Nous obtenons les réponses suivantes :

```
94090 commune.csv

1681546 formation.csv

308533 etablissement.csv

695369 effectif_admis.csv

632395 neo_bacheliers_effectif.csv

286885 boursiers_effectif.csv
```

Nous avons exporté les tables grâce à la commande \copy, puis nous avons calculé les octets de chaque table avec wc -c et ensuite nous avons calculé à la main le total, nous avons 3698818 octets ce qui correspond à 3,698818 mo.

Exercice 3: Requêtage

`Le fichier requetes.sql est fourni avec ce rapport.`

Q1) Écrire une requête qui, à partir de import affiche le contenu de la colonne n56 et le re-calcul de celle-ci à partir d'autres colonnes de import (2 cols).

```
SELECT n56, n57 + n58 + n59 AS ResultByMe FROM import;
```

Nous obtenons la réponse suivante avec LIMIT 10:

```
 n56 | resultbyme
-----+------------
 122 |        122
 120 |        120
 139 |        139
  80 |         80
 241 |        241
 162 |        162
 629 |        629
  28 |         28
 137 |        137
  41 |         41
(10 lignes)
```

n56=Effectif des admis néo bacheliers

n57+n58+n59=Effectif des admis néo bacheliers généraux + Effectif des admis néo bacheliers technologiques + Effectif des admis néo bacheliers professionnels

Q2) **Quelle requête vous permet de savoir que ce re-calcul est parfaitement exact ?**

```
SELECT n56, n57 + n58 + n59 AS ResultByMe  FROM import WHERE n57 + n58 + n59 <> n56;
```

Nous obtenons la réponse suivante :

```
 n56 | resultbyme
-----+------------
(0 ligne)
```

Calcule la somme de trois colonnes (n57, n58 et n59) et la compare à la valeur d'une autre colonne (n56).Si la somme n'est pas correcte alors la requête renverraient ces lignes.Ce qui nous permet de déduire que ce re-calcul est exact.

Q3) **Écrire une requête qui, à partir de import affiche le contenu de la colonne n74 et le re-calcul de celle-ci à partir d'autres colonnes de import (2 cols).**

```
SELECT n74, ROUND((n51/(case WHEN n47 <> 0 THEN n47 ELSE 1 END))*100) AS ResultByMe FROM import;
```

Nous obtenons la réponse suivante avec LIMIT 10 :

```
 n74 | resultbyme
-----+------------
  40 |         40
  56 |         56
  33 |         33
  28 |         28
  47 |         47
```

```
  27 |         27
  72 |         72
  82 |         82
  79 |         79
  76 |         76
(10 lignes)
```

Divise n51 par n47 en gérant le cas où n47 serait zéro (évite la division par zéro) et puis sert à comparer ce calcul et la colonne n74.
n51=Dont effectif des admis ayant reçu leur proposition d'admission à l'ouverture de la procédure principale
n47=Effectif total des candidats ayant accepté la proposition de l'établissement (admis)
n=% d'admis ayant reçu leur proposition d'admission à l'ouverture de la procédure principale

Q4) Quelle requête vous permet de savoir que ce re-calcul est parfaitement exact ?

```
SELECT n74, ROUND((n51/(case WHEN n47 <> 0 THEN n47 ELSE 1
END))*100) AS ResultByMe FROM import GROUP BY n74, n47, n51 HAVING
ROUND((n51/(case WHEN n47 <> 0 THEN n47 ELSE 1 END))*100)<> n74;
```

Nous obtenons la réponse suivante :

```
 n74 | resultbyme
-----+------------
  58 |         57
  58 |         57
  58 |         57
(3 lignes)
```

Elle cherche des incohérences (résultat calculé vs valeur d'origine) par groupes de données. Hors comme on peut le voir il y en a.

Avec la commande suivante nous allons déterminer notre taux d'erreur :

```
SELECT (COUNT(*)::float / (SELECT COUNT(*) FROM import)) * 100 AS
TauxErreur FROM (SELECT n74, ROUND((n51/(CASE WHEN n47 <> 0 THEN
n47 ELSE 1 END))*100) AS ResultByMe FROM import GROUP BY n74, n47,
n51 HAVING ROUND((n51/(CASE WHEN n47 <> 0 THEN n47 ELSE 1
END))*100) <> n74 ) AS SousREQ;
```

```
      tauxerreur
----------------------
 0.021630975556997622
(1 row)
```

Q5) Écrire une requête qui, à partir de import affiche le contenu de la colonne n76 et le re-calcul de celle-ci à partir d'autres colonnes de import (2 cols). A partir de combien de décimales ces données sont exactes ?

```
SELECT n76, ROUND(n53/(case WHEN n47 <> 0 THEN n47 ELSE 1
END)*100) AS ResultsByMe FROM import;
```

Nous obtenons la réponse suivante avec LIMIT 10:

```
 n76 | resultsbyme
-----+-------------
 100 |         100
  89 |          89
  59 |          59
  77 |          77
  59 |          59
  97 |          97
  93 |          93
  56 |          56
 100 |         100
  91 |          91
(10 lignes)
```

La requête affiche n76 et son re-calcul à partir d'autres colonnes (n53 et n47).
n76=% d'admis ayant reçu leur proposition d'admission avant la fin de la procédure principale
n53=Dont effectif des admis ayant reçu leur proposition d'admission avant la fin de la procédure principale
n47=Effectif total des candidats ayant accepté la proposition de l'établissement (admis)

Q5 bonus) **Vérification:**

```
SELECT n76, ROUND(n53/(case WHEN n47 <> 0 THEN n47 ELSE 1
END)*100) AS ResultsByMe FROM import GROUP BY n76, n47, n53 HAVING
ROUND((n53/case WHEN n47 <> 0 THEN n47 ELSE 1 END)*100) <> n76;
```

Nous obtenons la réponse suivante :

```
 n76 | resultsbyme
-----+-------------
(0 ligne)
```

Le résultat vide indique qu'il n'y a aucune différence entre le recalcul arrondi (ResultsByMe) et la valeur d'origine (n76) pour aucun groupe (n76, n47, n53). Cela montre une cohérence entre le recalcul et les données originales.

Q6) **Fournir la même requête sur vos tables ventilées**

```
SELECT n76, ROUND(e.Eff_adm_prop_fin_procéd_p
/(case WHEN e.Eff_total_cand_accept_prop <> 0 THEN
e.Eff_total_cand_accept_prop ELSE 1 END)*100) as resultbyme
FROM import as i join effectif_admis as e on i.n110 =
e.cod_aff_form;
```

Nous obtenons la réponse suivante avec LIMIT 10 :

```
 n76 | resultsbyme
```

```
-----+-------------
 100 |         100
  89 |          89
  59 |          59
  77 |          77
  59 |          59
  97 |          97
  93 |          93
  56 |          56
 100 |         100
  91 |          91
(10 lignes)
```

La requête calcule et arrondit le nombre de candidats admis par rapport au total des candidats pour chaque valeur n76.

n76=% d'admis ayant reçu leur proposition d'admission avant la fin de la procédure principale

`Eff_adm_prop_fin_procéd_p`=Dont effectif des admis ayant reçu leur proposition d'admission avant la fin de la procédure principale

`Eff_total_cand_accept_prop`=Effectif total des candidats ayant accepté la proposition de l'établissement (admis)

Q7) Écrire une requête qui, à partir de import affiche la n81 et la manière de la recalculer. A partir de combien de décimales ces données sont exactes ?

```
SELECT n81, ROUND(n55/(case WHEN n56 <> 0 THEN n56 ELSE 1
END)*100) AS ResultByMe FROM import GROUP BY n81, n55, n56;
```

Nous obtenons la réponse suivante avec LIMIT 10 :

```
 n81 | resultbyme
-----+------------
  10 |         10
  17 |         17
   0 |          0
  26 |         26
  51 |         51
  27 |         27
  19 |         19
  12 |         12
   0 |          0
  15 |         15
(10 lignes)
```

La requête affiche n81 et son re-calcul à partir de n55 et n56 (division avec gestion du zéro et arrondi à 2 décimales)

n55=Dont effectif des admis boursiers néo bacheliers

n56=Effectif des admis néo bacheliers

n81=% d'admis néo bacheliers boursiers

Q7 bonus) Vérification:

```
SELECT n81, ROUND(n55/(case WHEN n56 <> 0 THEN n56 ELSE 1
END)*100) AS ResultByMe FROM import GROUP BY n81, n55, n56 HAVING
ROUND((n55/case WHEN n56 <> 0 THEN n56 ELSE 1 END)*100) <> n81;
```

Nous obtenons la réponse suivante :

```
 n81 | resultbyme
-----+------------
(0 ligne)
```

Le résultat vide signifie qu'il n'y a aucune différence entre le recalcul arrondi (ResultByMe) et la valeur d'origine (n81) pour aucun groupe (n81, n55, n56).

Q8) **Fournir la même requête sur vos tables ventilées**

```
SELECT i.n81, ROUND(e.Eff_adm_bours_néo_bach/(case WHEN
(e.Eff_adm_néo_b_g + e.Eff_adm_néo_b_t +e.Eff_adm_néo_b_pro) <> 0
THEN (e.Eff_adm_néo_b_g + e.Eff_adm_néo_b_t +e.Eff_adm_néo_b_pro)
ELSE 1 END)*100) as resultbyme FROM import as i join
effectif_admis as e on i.n110 = e.cod_aff_form;
```

Nous obtenons la réponse suivante avec LIMIT 10 :

```
 n81 | resultbyme
-----+------------
  10 |         10
  17 |         17
   0 |          0
  26 |         26
  51 |         51
  27 |         27
  19 |         19
  12 |         12
   0 |          0
  15 |         15
(10 lignes)
```

On peut voir ici que “n81” est égal à “resultbyme”, ce qui est le cas pour toutes les autres lignes.

Conclusion :

En résumé, l'application des consignes de ventilation des données nous a permis de diviser la taille de la base de données, tout en facilitant la création de vues d'ensemble et l'organisation des données grâce à des requêtes simples.