BASE DE DONNÉES ÉVOLUÉES

MISE EN PLACE D'UNE BASE EN XML

Boulahmel Amine
Faculté des sciences et techniques
Université de Nantes
amine.boulahmel@etu.univ-nantes.fr

Juzdzewski Matthieu
Faculté des sciences et techniques
Université de Nantes
matthieu.juzdzewski@etu.univ-nantes.fr

Jandu Harry
Faculté des sciences et techniques
Université de Nantes
harry.jandu@etu.univ-nantes.fr

15 septembre 2021

RÉSUMÉ

Dans le cadre du projet de *Base de données évoluées* de première année de Master à l'université de Nantes, nous avons construit un entrepôt de données à partir de datasets libres que nous avons ensuite interrogés pour établir des statistiques et des graphiques. Notre sujet concerne les salaires horaire nets moyens en France entre 2012 et 2016 pour chaque commune, chaque catégorie socioprofessionnelle et chaque genre. Nous avons procédé à un nettoyage et une transformation des données afin finalement d'obtenir un ensemble de données au format XML. Nous avons utilisé XSLT pour faire nos requêtes. Le github de notre groupe est complet et documenté et se pose en tant que soutient de ce rapport, notamment le read-me principal.

***M*ots-clés** XML · XSLT · DTD · XSD · aggrégats · OpenRefine · Rstudio · Saxon · xmlstarlet

1 Introduction

Le projet consiste à mettre en oeuvre une base de données traitant d'un sujet (libre de choix) en suivant plusieurs étapes pré-déterminées, dresser des exemples d'utilisation de la base et explorer les opportunités de cette dernière.
Afin de mieux visualiser le déroulement global de la mise en place de la base, nous avons synthétisé les différentes étapes afin de mieux comprendre le déroulement global du projet.

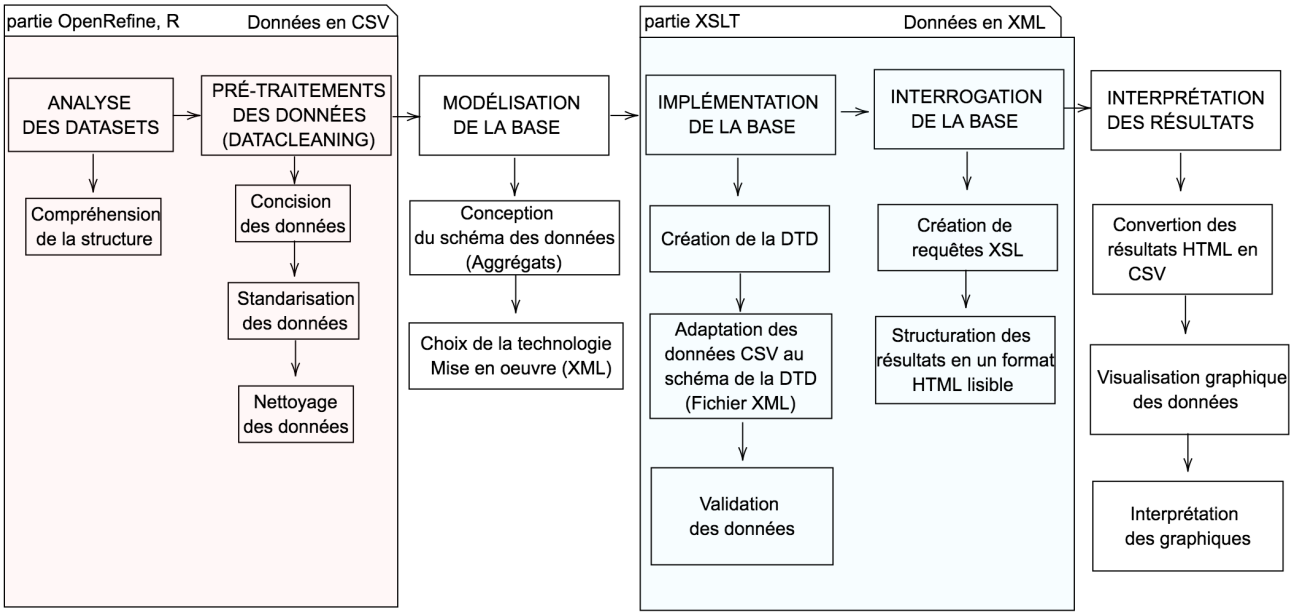


FIGURE 1 – Cartographie du déroulement de la mise en place de la base

2 Analyse des datasets

Les datasets traitent les **salaires nets horaire moyens selon la commune, sexe, catégorie socioprofessionnelle entre 2012 et 2016**.
Les différents datasets contiennent (en moyenne) 29 colonnes.
Les deux premières dénotent les informations sur la localisation et les 27 restantes les salaires.
Les 27 colonnes sont définies par la nomenclature des catégories socioprofessionnelles via une classification créée par l'Institut national de la statistique et des études économiques en 1982.
Les noms de chaque colonnes contiennent des informations sur le sexe, la catégorie socioprofessionnelle et l'année (eg. **SHMFC12** signifie **Salaire Horaire Moyen** d'une **Femme Cadre** en **2012**).
Cette structure est représenté ainsi :

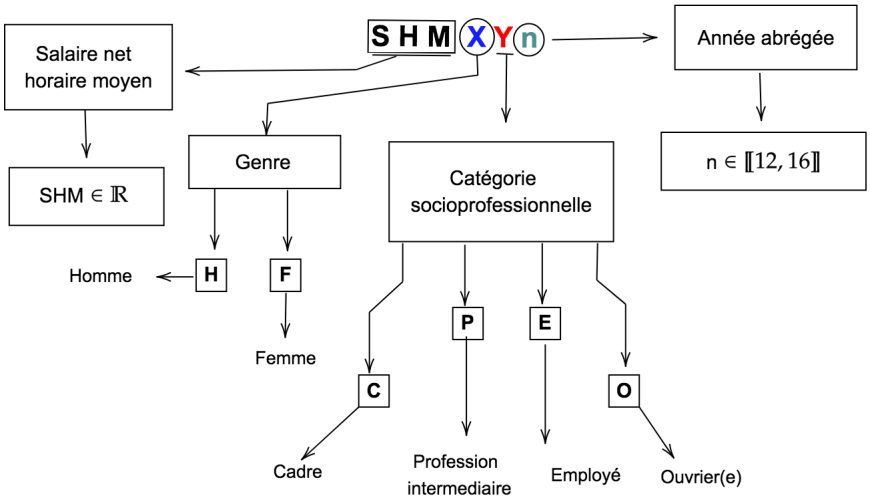


FIGURE 2 – Structure générale des colonnes traitant les salaires

3 Pré-traitements des données

À partir de la structure globale, une stratégie de raffinement est mise en place. Cette stratégie consiste à séparer toutes les colonnes de type **SHMXYN** en quatre colonnes : **Salaire, Genre, Catégorie et Année**. À partir de cette séparation, les salaires seront regroupés dans leurs colonnes respectives tout en gardant la cohérence des données. Une fois les entités séparées et les données regroupées, une seconde étape consistera à standardiser les données afin d'effectuer des requêtes de manière simple sans se soucier de conventions/notations propres aux datasets de base (eg. transformer les séparateurs ',' de nombres flottant en '.', etc.). Enfin, viendra une dernière étape de nettoyage qui consistera tout simplement à supprimer les valeurs manquantes.
La concision des données se fera via l'outil **OpenRefine**.
Les autres étapes se feront via le logiciel **Rstudio** et le langage **R**.

3.1 Pré-traitements avec OpenRefine

La première étape consiste à effectuer un premier *lissage* des données. Il faut supprimer les colonnes dont nous ne nous servirons pas, renommer certains labels et surtout procéder à la transformation de la forme des données : une transposition matricielle des colonnes en lignes puis une séparation en nos quatre colonnes : *Salaire, Genre, Catégorie et Année*. Pour cela, nous utilisons l'outil **openRefine**. La suppression et la nomination des colonnes sont des étapes simples et qui sont déjà expliquées sur notre Git (openRefine propose une interface facile d'utilisation pour effectuer ce type de tâches). La transposition matricielle se fait également uniquement à travers l'interface, comme suit :

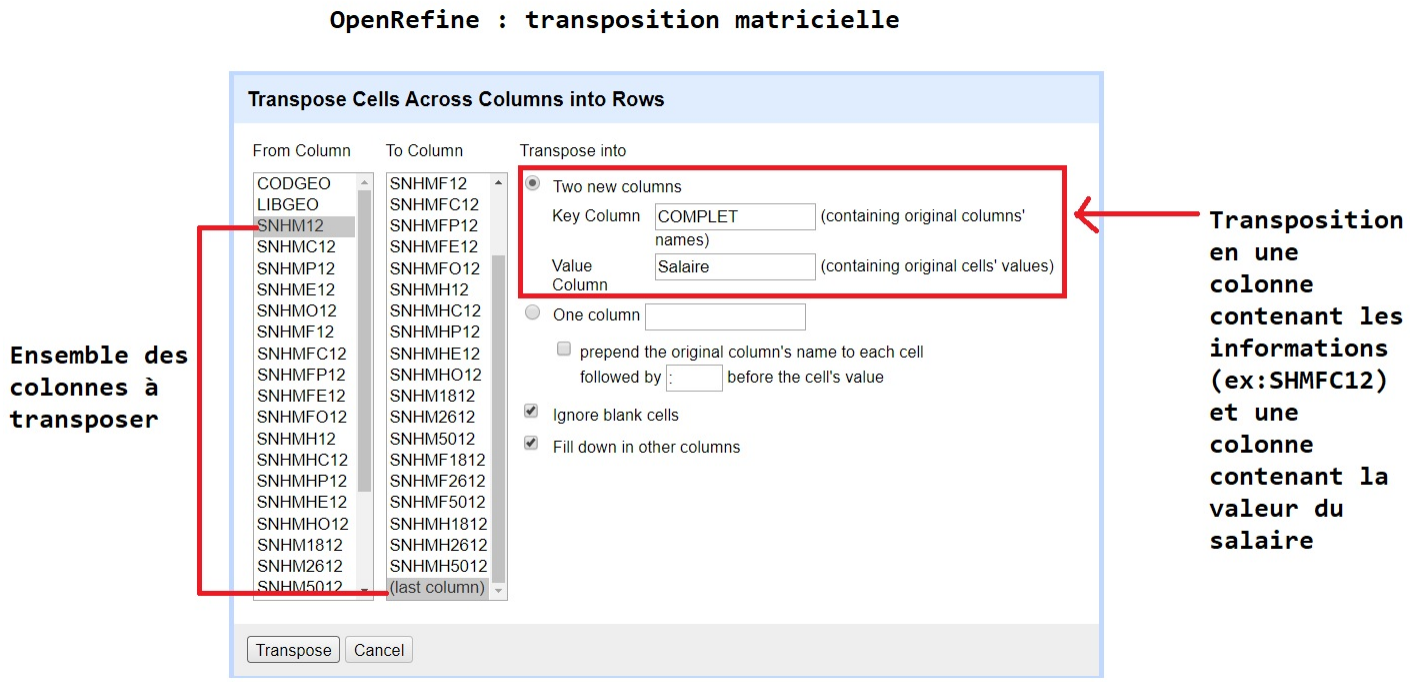


FIGURE 3 – Transposition matricielle des colonnes en lignes

Pour séparer les colonnes correctement, nous utilisons le **General Refine Expression Language** ou **GREL**. Les détails du processus sont expliqués sur le Git, mais voici par exemple le code qui permet de créer la colonne *Categorie* :

```
1  if(contains(value,"C"), "Cadre",
2  if(contains(value,"P"), "Profession",
3  if(contains(value,"E"), "Employé", "Ouvrier")))
```

Lorsque ceci est terminé, le dataset est prêt à être traité comme expliqué dans la partie suivante.

3.2 Pré-Traitements avec Rstudio

La colonne des salaires contient des flottants séparés par des virgules. Il est impératif de changer ces séparateurs afin de ne pas se retrouver avec des requêtes qui considèrent les chiffres comme étant des chaînes de caractères. Et donc, cette transformation se fera via la commande R suivante : Ensuite, nous ajouterons les colonnes *Région* et *Département* afin

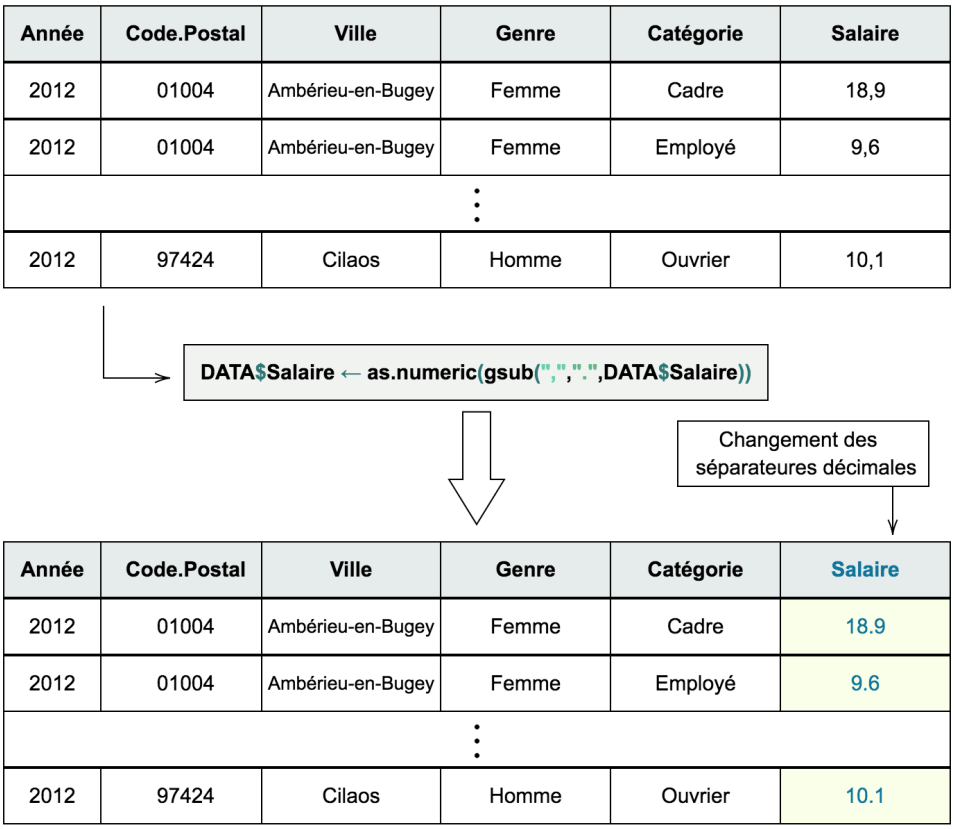


FIGURE 4 – Représentation du changement des séparateurs dans la colonne **Salaire**

de lier les données entre elles.
Afin de préserver la cohérence, il existe une fonction sous **R** dénommée "match" permettant de faire correspondre une ligne spécifique du dataset en liant une colonne dont la valeur est égale à la ligne provenant du deuxième dataset que l'on veut rajouter.
Il suffit donc de faire correspondre les deux premiers caractères du code postal (correspondants à la région) afin de lier les lignes entre elles ; et donc, ajouter la colonne avec ces nouvelles données. Malgré le traitement, quelques lignes se retrouvent avec des valeurs **NA** (eg. valeurs manquantes). Le problème est dû au fait que les matchs s'effectuent uniquement sur les deux premiers chiffres du code postal alors qu'en réalité, il existe des régions avec trois chiffres.

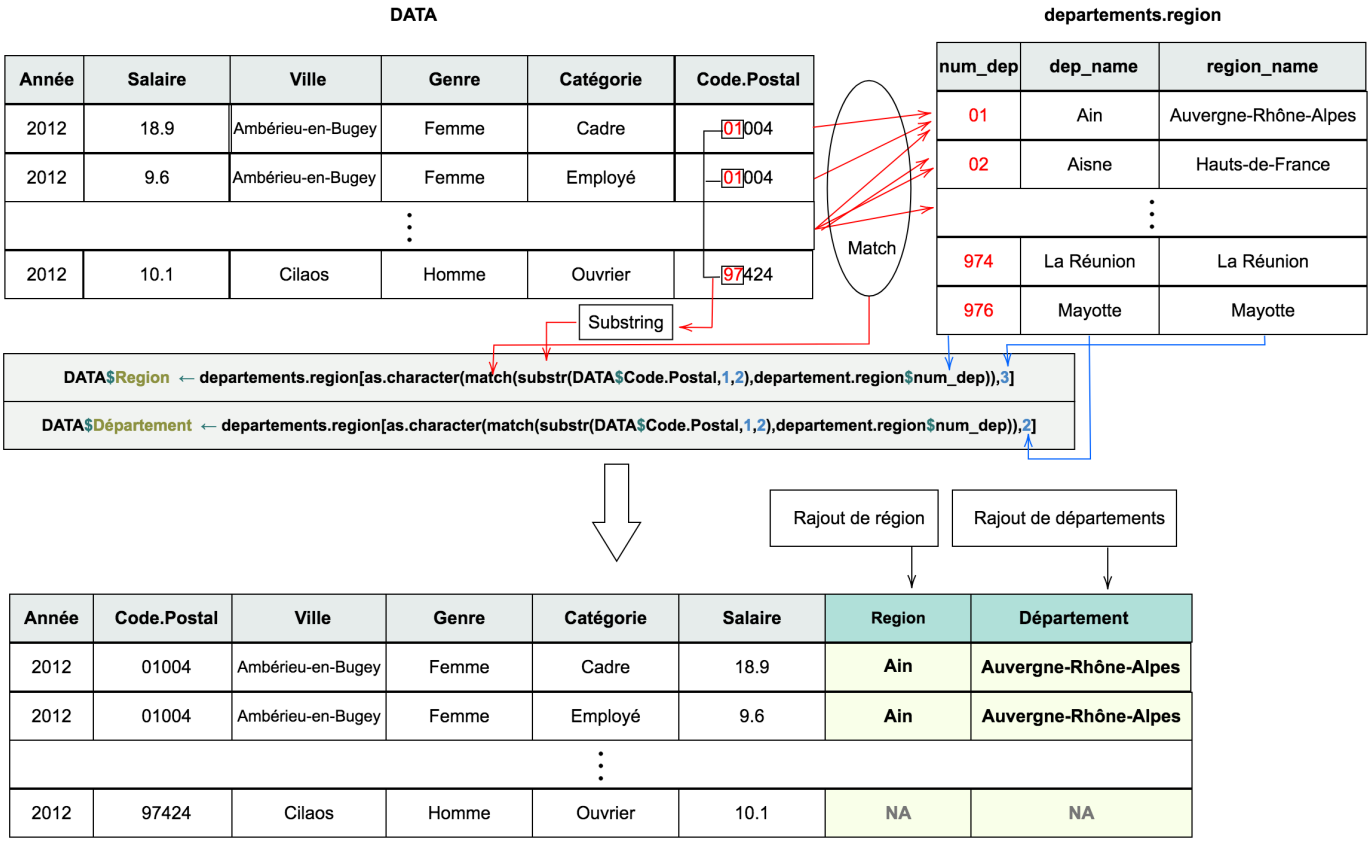


FIGURE 5 – Représentation du processus d'ajout des colonnes région et département

Pour cela, le problème est traité à part avec un code permettant de rajouter manuellement les régions et départements manquants.
Sous R, les données sont traitées dans une structure de données de type **DataFrame**.
Les **DataFrame** fonctionnent grâce à des facteurs afin de stocker des données de type **catégoriques**. Un rajout des données de façon brute est impossible sans avoir rajouter au préalable les différents niveaux.
Pour cela, la fonction *levels()* permet d'initialiser les niveaux d'une colonne du dataset.
Une fois les niveaux insérés, les traitements manuels peuvent procéder comme prévu.
Le procédé est simple : pour chaque ligne contenant des valeurs de type NA, on teste si les trois premiers chiffres du code postal correspondent à une valeur définie. Si c'est le cas, on initialisera la colonne département et région par les valeurs correspondantes.
Afin de trouver la première valeur de type **NA** dans le dataset, la fonction is.na() est utilisée. On recherche toujours le premier indice de ligne associé à la valeur **NA**. Ainsi, à chaque tour de boucle, la première colonne retournée sera la première ligne avec une valeur **NA**. Une fois cette colonne initialisée, le code va obligatoirement retourner la colonne suivante par principe de récurrence et le procédé continue jusqu'à la dernière valeur **NA**.

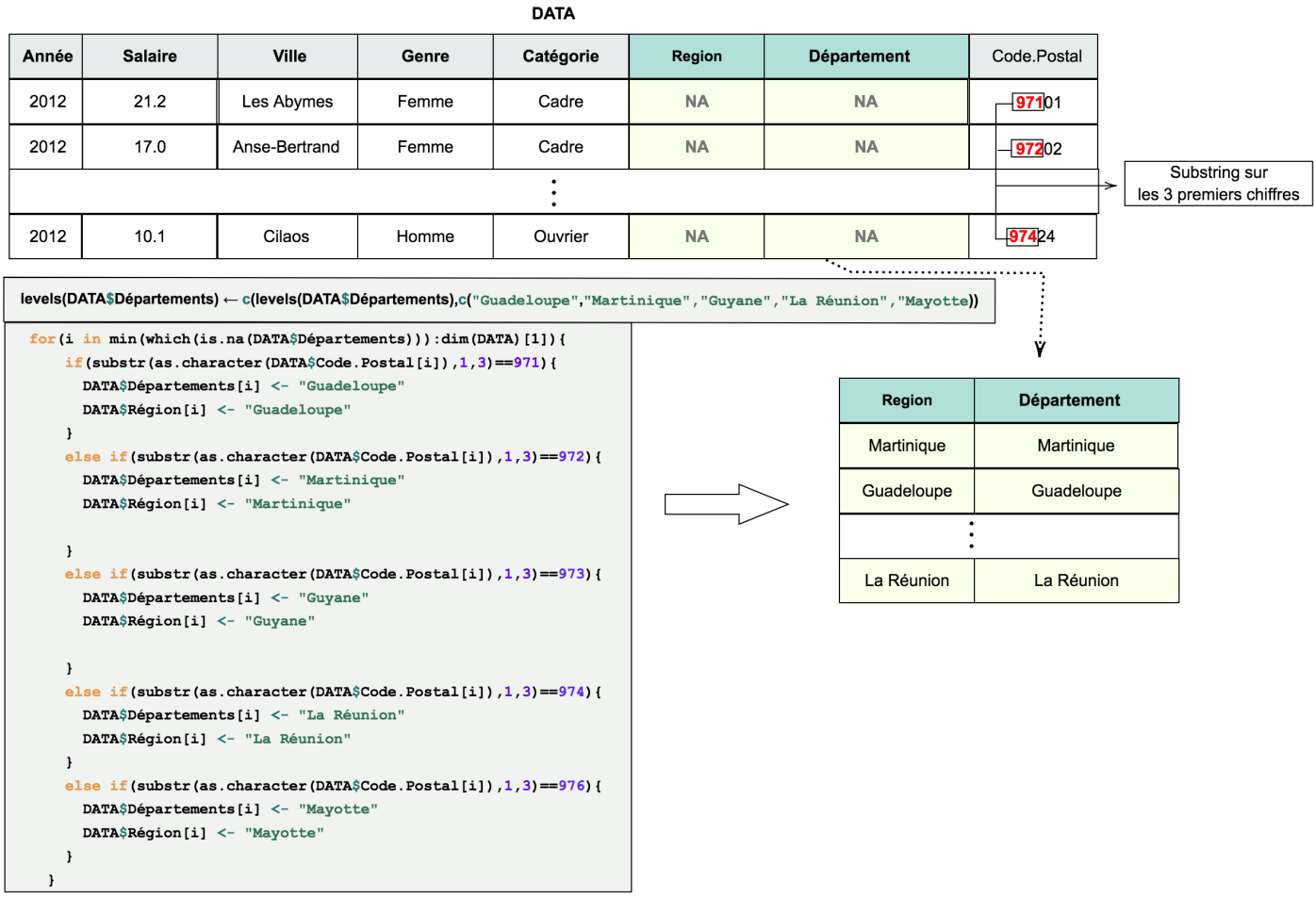
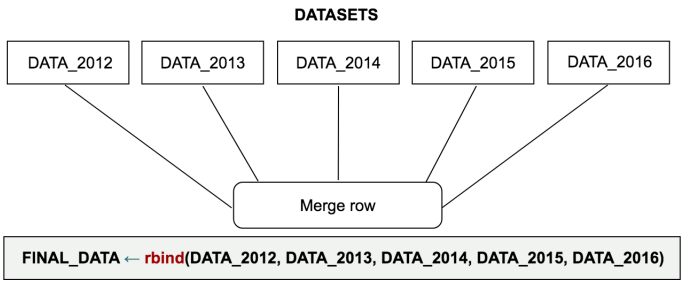


FIGURE 6 – Processus d'ajout des valeurs manquantes des lignes ayant une région à trois chiffres

Afin de simplifier l'interrogation de la base (ou effectuer de futurs liens avec d'autres datasets), une colonne contenant les codes de départements est ajoutée dans le dataset (en suivant le principe du matching évoqué précédemment).
Une fois ces étapes achevées sur le premier dataset (eg. données de 2012), les même processus sont ré-appliqués sur les autres datasets (eg. données de 2013, 2014, 2015 et 2016).
Une fois toutes ces étapes achevées, on effectue un **merge** (sur les lignes) de tous les datasets en un seul. La fonction RBIND() permet ceci de manière efficace. Enfin, la dernière étape consiste en une exportation du dataset au format CSV.



Année	Code.Postal	Ville	Genre	Catégorie	Salaire	Region	Département	CodeDép
2012	01004	Ambérieu-en-Bugey	Femme	Cadre	18.9	Ain	Auvergne-Rhône-Alpes	01
2012	01004	Ambérieu-en-Bugey	Femme	Employé	9.6	Ain	Auvergne-Rhône-Alpes	01
⋮								
2013	34154	Mauguio	Homme	Ouvrier	11.1	Occitanie	Hérault	34
⋮								
2014	22004	Bégard	Femme	Cadre	18.2	Bretagne	Côtes-d'Armor	22
⋮								
2015	44061	Frossay	Homme	Profession	15.5	Pays de la Loire	Loire-Atlantique	44
⋮								
2016	97423	Les Trois-Bassins	Homme	Ouvrier	10.7	La Réunion	La Réunion	974

FIGURE 7 – Merge de tous les datasets en un seul dataset : **FINAL_DATA**

4 Modélisation de la base

4.1 Schéma des données

Le schéma appliqué est en étoile afin de stocker les données agrégées. Il existe quatre dimensions (en plus de la table des faits) :

1. **FACTS** : la table des faits qui contiendra les différentes informations
2. **Genre** : répertoriant les sexes et contiendra deux valeurs possibles : Homme ou Femme,
3. **Année** : répertoriant les différentes années du dataset et contiendra des valeurs entières comprises entre 2012 et 2016,
4. **Catégorie** : répertoriant les différentes catégories du dataset et contiendra quatre valeurs possibles : {Cadre, Profession, Ouvrier, Employé}
5. **Localisation** : qui contiendra les informations sur les régions, départements et villes.

Chaque dimension aura sa propre clé primaire.

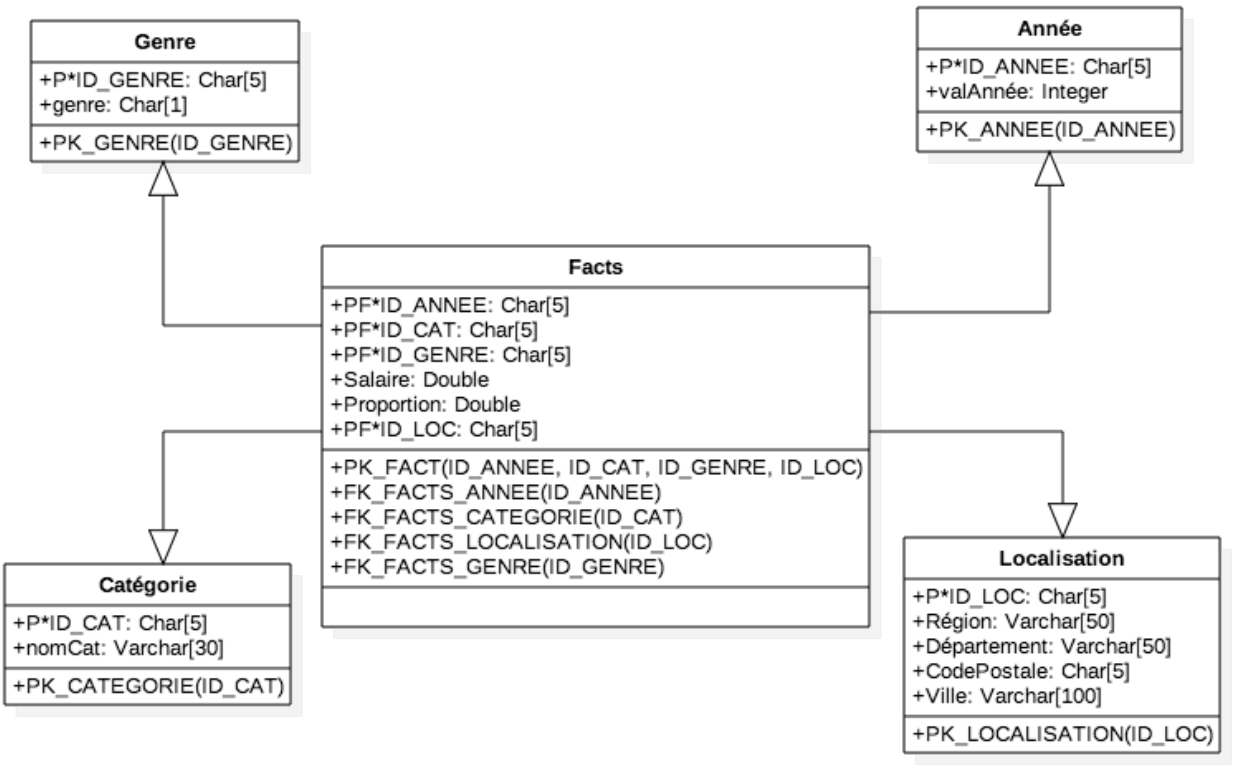


FIGURE 8 – Schéma des données

La technologie mise en oeuvre afin d'implémenter le schéma ci-dessus sera **XML** et sera détaillée dans la partie **Implémentation de la base**.

5 Implémentation de la base

5.1 Création de la DTD

Afin de mettre en place le schéma des données au format XML, il est impératif de créer un fichier **DTD** qui décrira le modèle proposé (eg. voir partie annexe).
La structure globale se présentera ainsi :

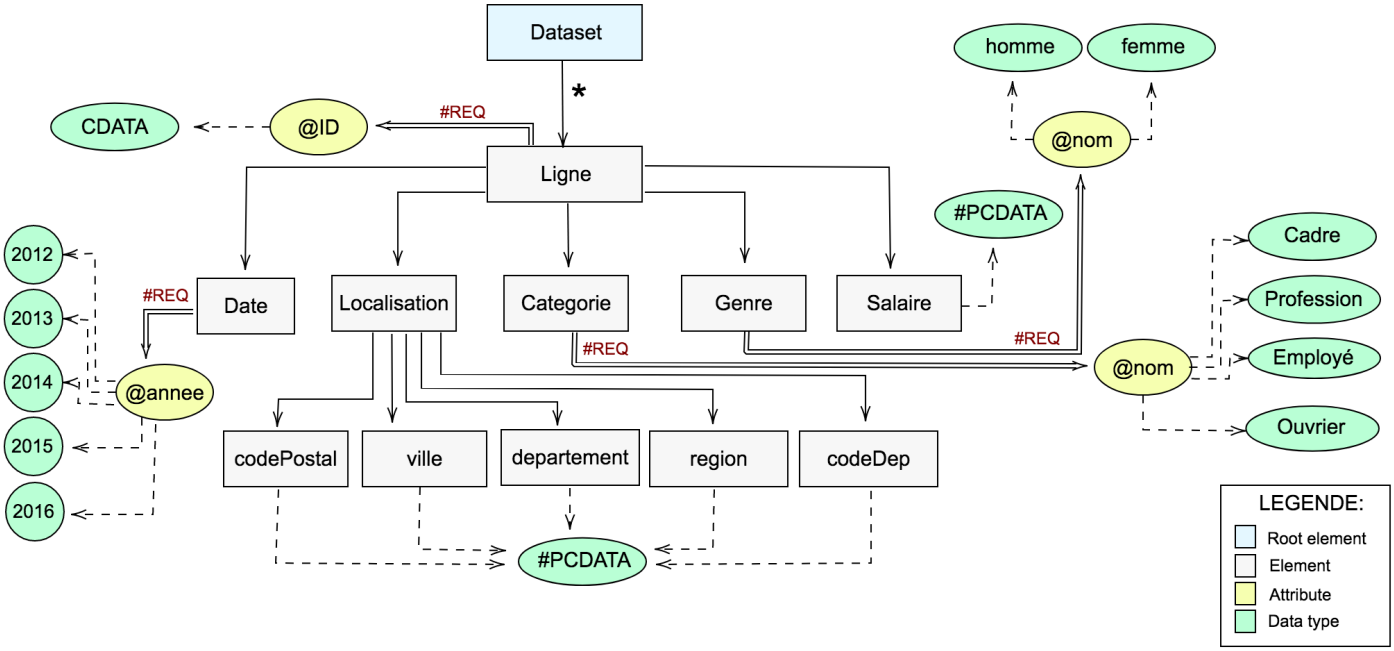


FIGURE 9 – Schéma de la DTD

5.2 Conversion du .csv en .xml

Pour convertir notre dataset du format *CSV* au format *XML*, nous avons écrit un programme en JAVA dont les détails se trouvent sur le Git. Voici cependant le modèle conceptuel de la conversion :

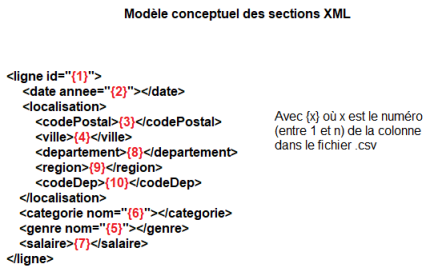


FIGURE 10 – Modèle conceptuel pour la conversion en xml

Enfin, les données XML sont prêtes à être validées grâce à l'outil **xmlstarlet** (eg. ou possibilité de valider avec d'autres outils traitant la technologie XML).

6 Interrogation de la base

Nous avons effectué nos requêtes en *XSLT 2.0*. Cette version permet d'utiliser des fonctionnalités avancées comme le *group-by* par exemple. Voici un exemple d'exécution d'une requête en ligne de commande (sous MAC) :

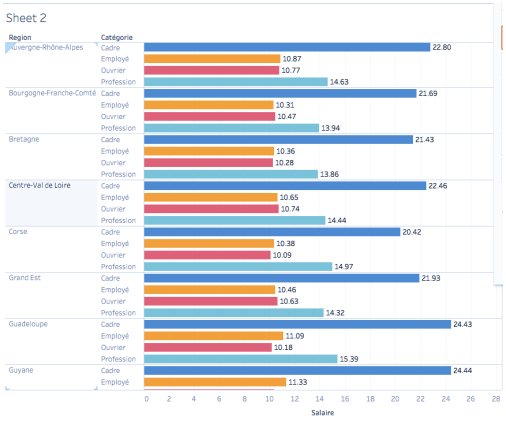
```
saxon    -s:XML_FINAL.xml    -xsl:REQ_2.1.xsl    -o:test_2.htm
```

6.1 Requête 1

L'objectif de cette requête est de donner, pour chaque région, la moyenne des salaires pour chaque catégorie socioprofessionnelle.

Region: Auvergne-Rhône-Alpes	
Categorie: Cadre	Total: 22.797476125511604
Categorie: Profession	Total: 14.625170532060006
Categorie: Employé	Total: 10.86991814461119
Categorie: Ouvrier	Total: 10.773874488403827
Region: Hauts-de-France	
Categorie: Cadre	Total: 21.573872180451104
Categorie: Profession	Total: 14.220770676691716
Categorie: Employé	Total: 10.56682330827068
Categorie: Ouvrier	Total: 10.503571428571412
Region: Provence-Alpes-Côte d'Azur	
Categorie: Cadre	Total: 22.904907975460137
Categorie: Profession	Total: 14.809815950920266
Categorie: Employé	Total: 10.634355828220867
Categorie: Ouvrier	Total: 10.4978527607362

(a) Résultat html



(b) Résultat graphique

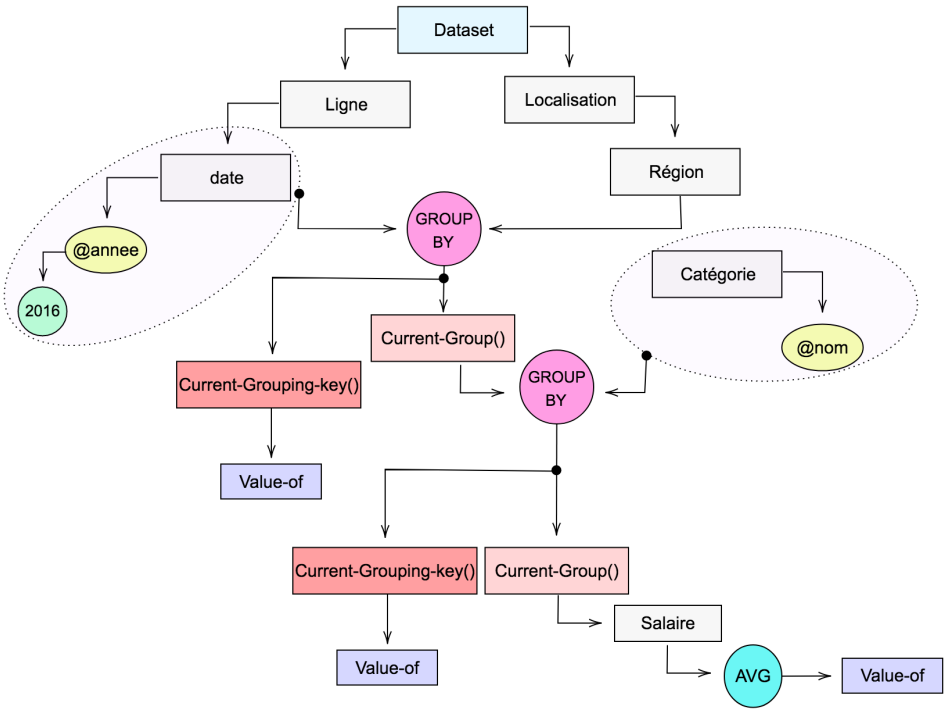


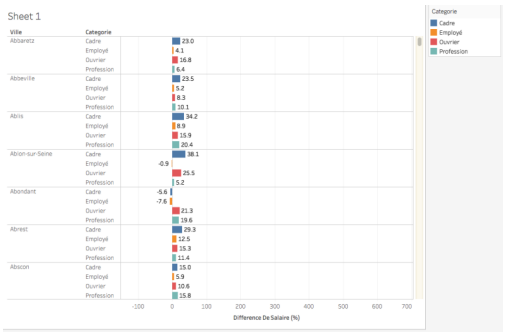
FIGURE 12 – Logique de la requête

6.2 Requête 2

L'objectif de cette requête est de donner, pour chaque ville et chaque catégorie socioprofessionnelle, la différence de salaire (en pourcentage) entre les hommes et les femmes.

Ville	Categorie	Difference de salaire (%)
Ambérieu-en-Bugey	Cadre	26.63
Ambérieu-en-Bugey	Profession	27.07
Ambérieu-en-Bugey	Employé	8.0
Ambérieu-en-Bugey	Ouvrier	18.56
Ambronay	Cadre	22.04
Ambronay	Profession	15.0
Ambronay	Employé	4.85
Ambronay	Ouvrier	20.83

(a) Résultat html



(b) Résultat graphique

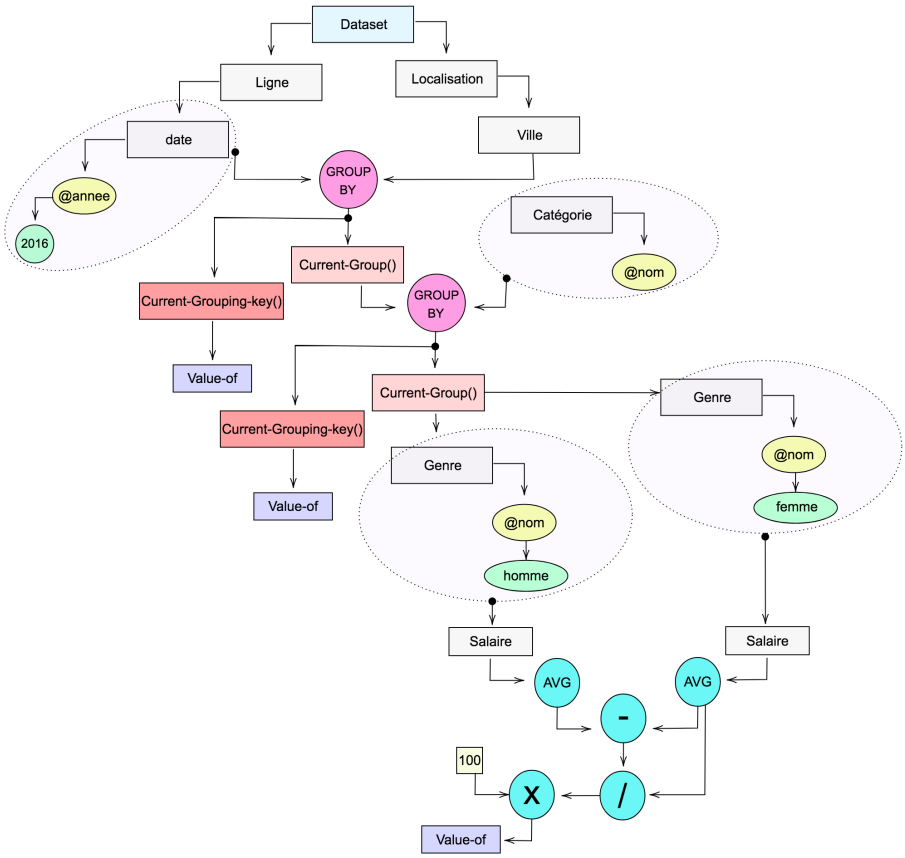


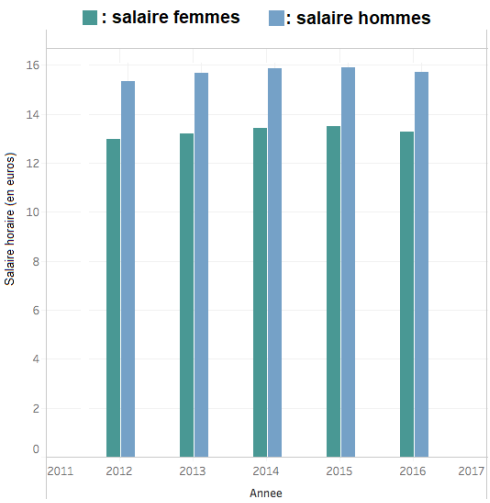
FIGURE 14 – Logique de la requête

6.3 Requête 3

L'objectif de cette requête est de montrer le salaire moyen des hommes et des femmes pour chaque année entre 2012 et 2016.

Annee	Total Salaire Femme	Total Salaire Homme
2012	12.992	15.333
2013	13.207	15.678
2014	13.408	15.864
2015	13.491	15.894
2016	13.282	15.73

(a) Résultat html



(b) Résultat graphique

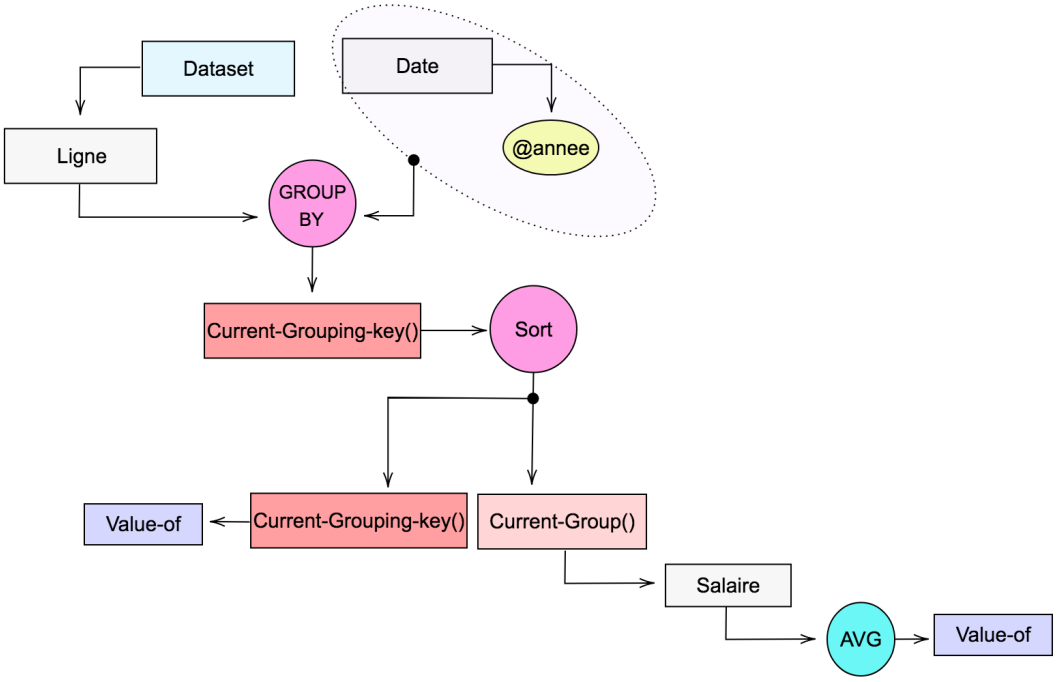


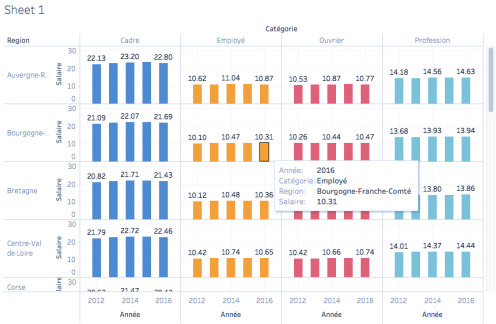
FIGURE 16 – Logique de la requête

6.4 Requête 4

L'objectif de cette requête est de montrer les salaires moyens par région et catégorie socioprofessionnelle entre 2012 et 2016. La requête est similaire à la requête 1, à ceci près que cette fois nous avons les données pour chaque année.

Region	Catégorie	Année	Salaire
Auvergne-Rhône-Alpes	Cadre	2012	22.125
Auvergne-Rhône-Alpes	Profession	2012	14.183
Auvergne-Rhône-Alpes	Employé	2012	10.623
Auvergne-Rhône-Alpes	Ouvrier	2012	10.528
Hauts-de-France	Cadre	2012	21.033
Hauts-de-France	Profession	2012	13.808
Hauts-de-France	Employé	2012	10.292
Hauts-de-France	Ouvrier	2012	10.349
[. . .]			
Auvergne-Rhône-Alpes	Cadre	2013	22.749
Auvergne-Rhône-Alpes	Profession	2013	14.435
Auvergne-Rhône-Alpes	Employé	2013	10.878
Auvergne-Rhône-Alpes	Ouvrier	2013	10.726
Hauts-de-France	Cadre	2013	21.498
Hauts-de-France	Profession	2013	14.073
Hauts-de-France	Employé	2013	10.508
Hauts-de-France	Ouvrier	2013	10.428

(a) Résultat html



(b) Résultat graphique

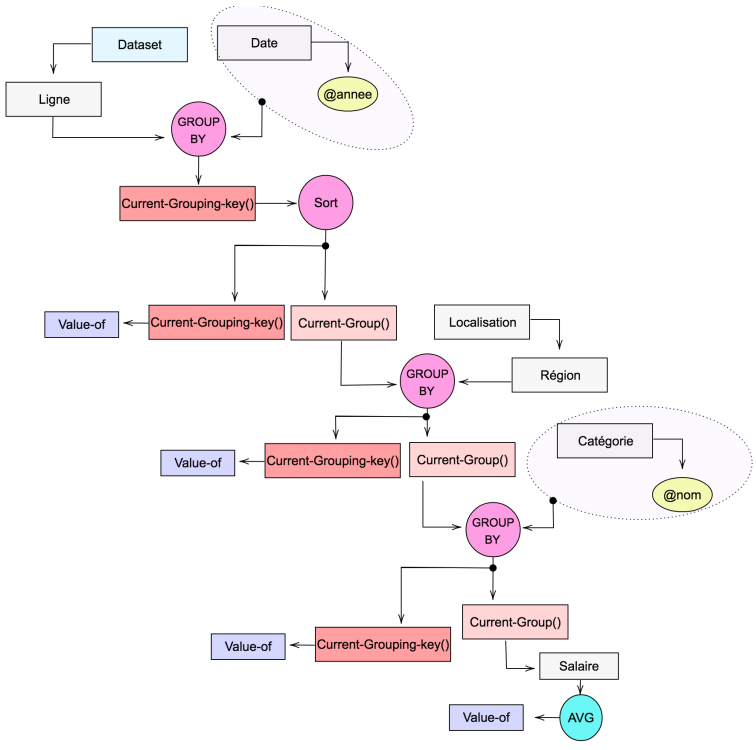


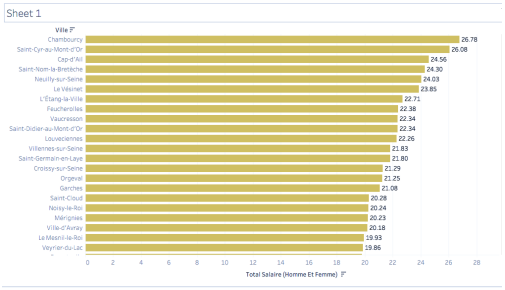
FIGURE 18 – Logique de la requête

6.5 Requête 5

L'objectif de cette requête est de montrer le salaire moyen par ville en 2016.

Ville	Total salaire (homme et femme)
Ambérieu-en-Bugey	14.518
Ambronay	14.218
Arbent	15.665
Attignat	14.062
Bâgé-la-Ville	14.129
Balan	14.645
Bellignat	14.028
Béligneux	14.83
Bellegarde-sur-Valserine	13.388

(a) Résultat html



(b) Résultat graphique

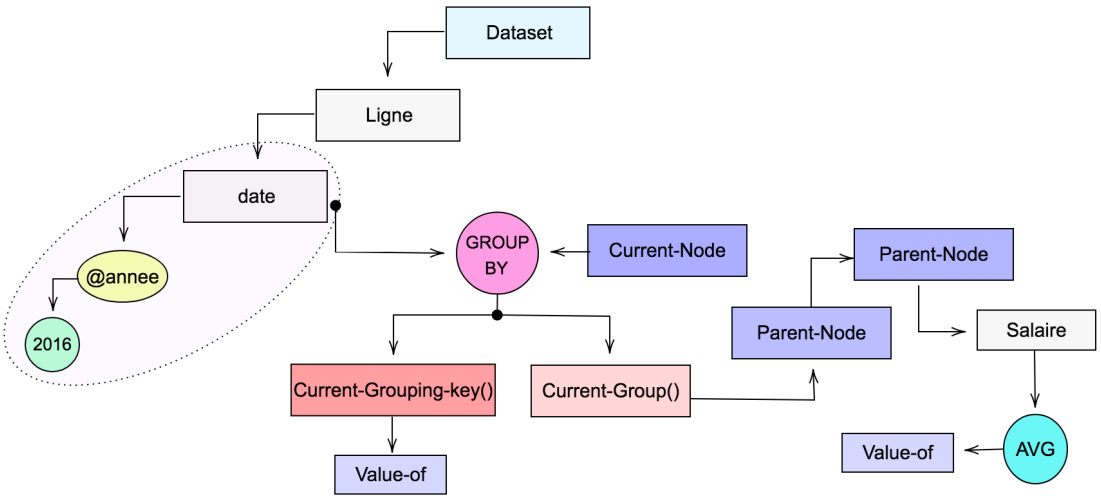


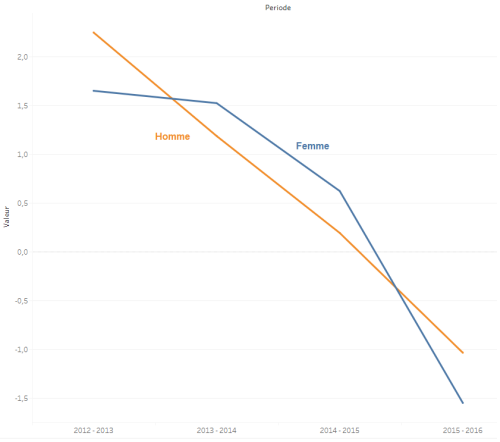
FIGURE 20 – Logique de la requête

6.6 Requête 6

L'objectif de cette requête est de montrer l'évolution globale de salaire (en %), par genre, entre 2012 et 2016.

Periode	Evolution femme	Evolution homme
2012 - 2013	1.649510177524533	2.247007119310494
2013 - 2014	1.523036679121369	1.1856275360609958
2014 - 2015	0.6222385232148946	0.192418079868924
2015 - 2016	-1.5499279658898988	-1.0352724872981593

(a) Résultat html



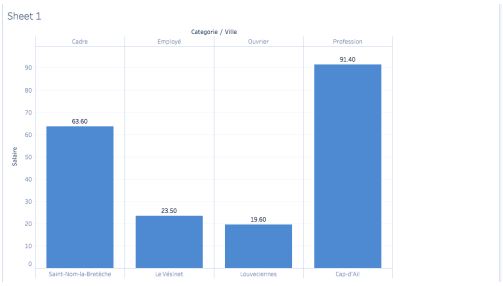
(b) Résultat graphique

6.7 Requête 7

L'objectif de cette requête est de donner, pour chaque catégorie socioprofessionnelle, la ville la plus intéressante en terme de salaire (en 2016).

Categorie	Ville	Salaire
Cadre	Saint-Nom-la-Bretèche	63.6
Profession	Cap-d'Ail	91.4
Employé	Le Vésinet	23.5
Ouvrier	Louveciennes	19.6

(a) Résultat html



(b) Résultat graphique

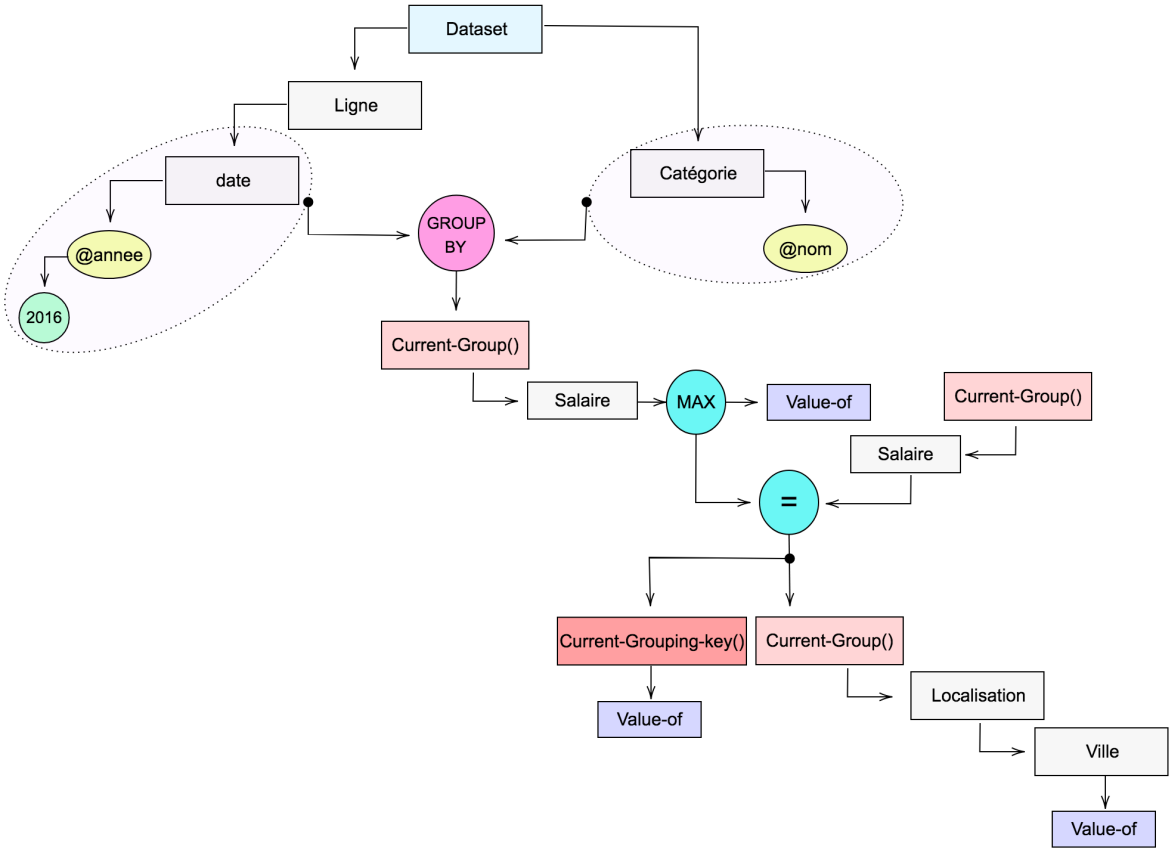


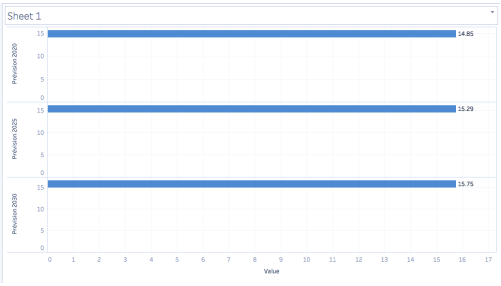
FIGURE 23 – Logique de la requête

6.8 Requête 8

L'objectif de cette requête est de tenter de prévoir, en utilisant les données des salaires de 2012 à 2016, la valeur du salaire moyen en 2020, 2025 et 2030.

Prévision 2020	Prévision 2025	Prévision 2030
14.850741458478316	15.293432779487372	15.74932045208902

(a) Résultat html



(b) Résultat graphique

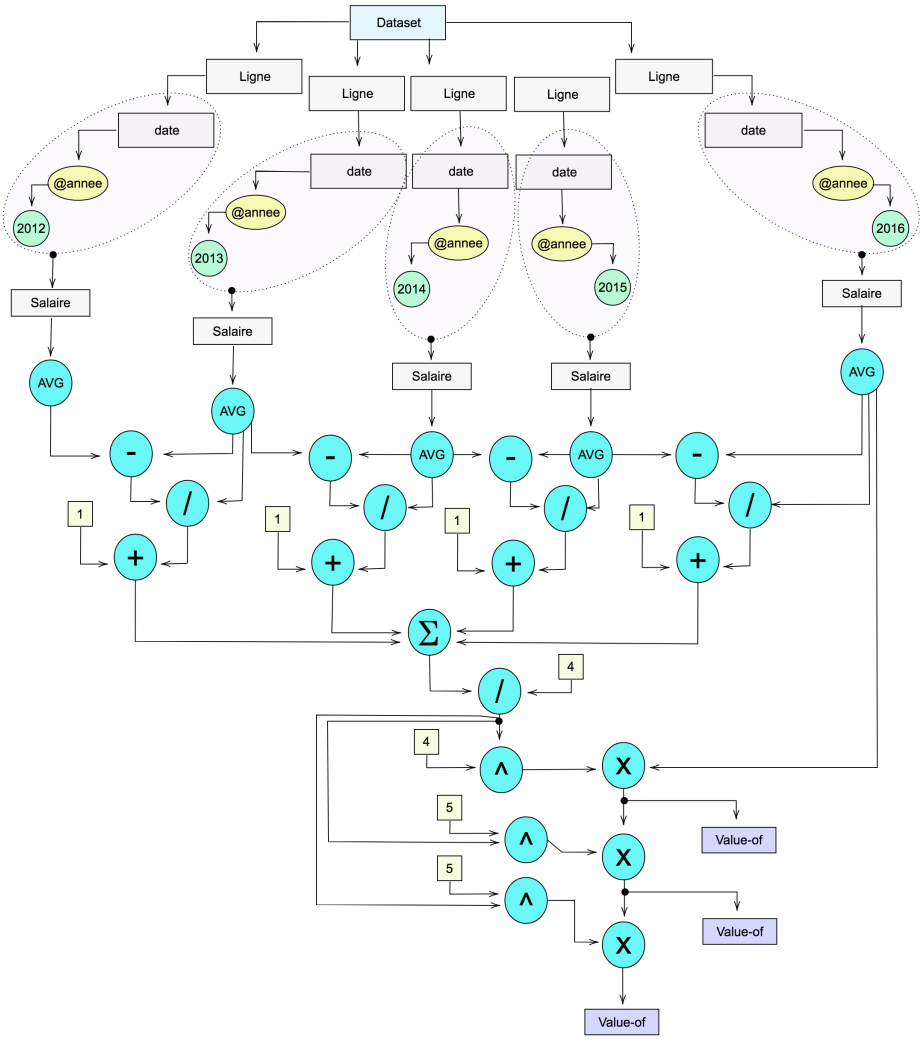


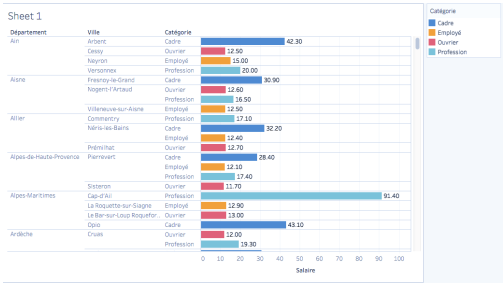
FIGURE 25 – Logique de la requête

6.9 Requête 9

L'objectif de cette requête est de montrer quelle est la ville la plus intéressante en terme de salaire (en 2016), et ce pour chaque département et chaque catégorie socioprofessionnelle.

Département	Catégorie	Ville	Salaire
Ain	Cadre	Arbent	42.3
Ain	Profession	Versonnex	20
Ain	Employé	Neyron	15
Ain	Ouvrier	Cessy	12.5
Aisne	Cadre	Fresnoy-le-Grand	30.9
Aisne	Profession	Nogent-l'Artaud	16.5
Aisne	Employé	Villeneuve-sur-Aisne	12.5
Aisne	Ouvrier	Nogent-l'Artaud	12.6

(a) Résultat html



(b) Résultat graphique

6.10 Requête 10

L'objectif de cette requête est de déterminer quelle est la ville qui propose le meilleur salaire en 2016, toute catégorie et genre confondus.

Meilleure Ville	Salaire
Cap-d'Ail	91.4

(a) Résultat html

(b) Résultat graphique

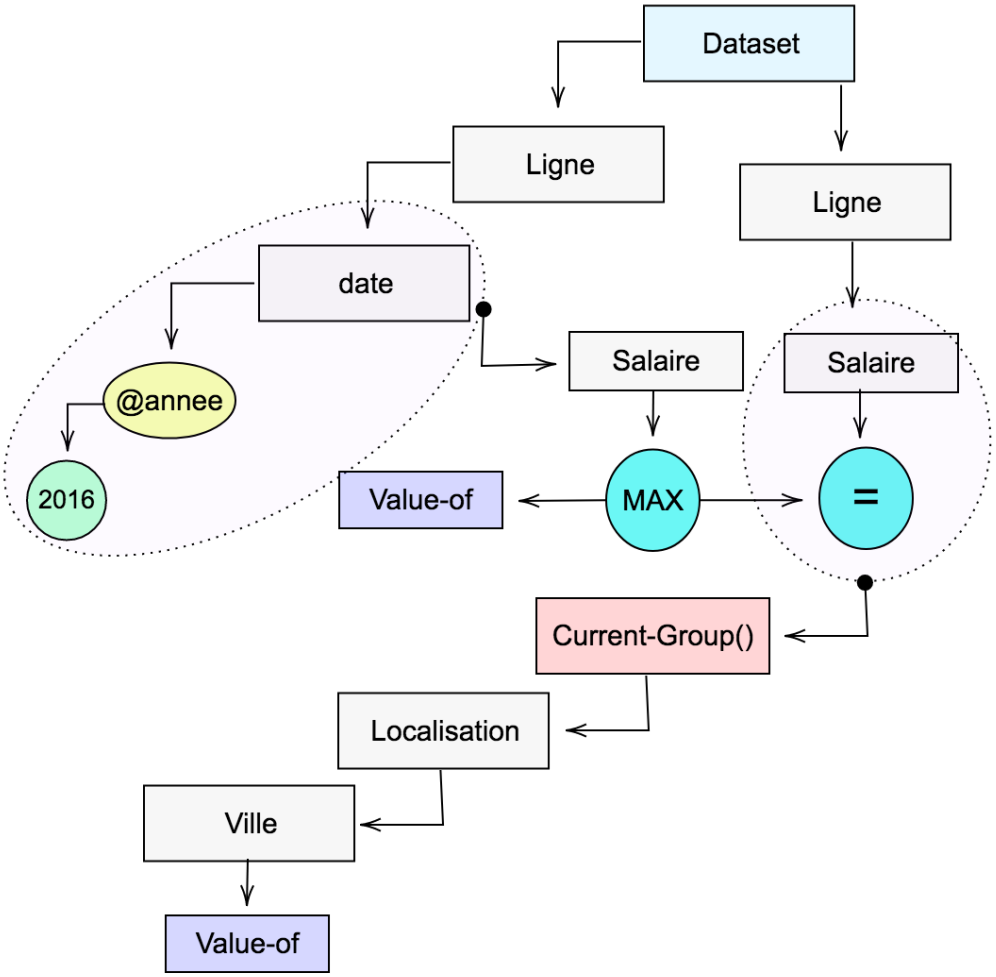


FIGURE 28 – Logique de la requête

7 Annexe

7.1 Changement des séparateurs décimales

```
DATA$Salaire <- as.numeric(gsub(",",".",DATA$Salaire))
```

7.2 Ajout des colonnes Région et Départements

Ajout de région

```
DATA$Region <- departements.region[as.
    character(match(substr(DATA$Code.Postal
    ,1,2),departement.region$num_dep)),3]
```

Ajout de Département

```
DATA$Departement <- departements.region[as.
    character(match(substr(DATA$Code.Postal
    ,1,2),departement.region$num_dep)),2]
```

7.3 Ajout des valeurs manquantes

Ajout de niveaux

```
levels(DATA$Departements) <- c(levels(
    DATA$Departements), c("Guadeloupe", "
    Martinique", "Guyane", "La Réunion", "
    Mayotte"))
.
```

Ajout des valeurs manquantes

```
for(i in min(which(is.na(DATA$Departements))):
    dim(DATA)[1]){
  if(substr(as.character(DATA$Code.Postal[i])
      ,1,3)==971){
    DATA$Departements[i] <- "Guadeloupe"
    DATA$Region[i] <- "Guadeloupe"
  }
  else if(substr(as.character(DATA$Code.Postal[i
      ]),1,3)==972){
    DATA$Departements[i] <- "Martinique"
    DATA$Region[i] <- "Martinique"

  }
  else if(substr(as.character(DATA$Code.Postal[i
      ]),1,3)==973){
    DATA$Departements[i] <- "Guyane"
    DATA$Region[i] <- "Guyane"

  }
  else if(substr(as.character(DATA$Code.Postal[i
      ]),1,3)==974){
    DATA$Departements[i] <- "La Réunion"
    DATA$Region[i] <- "La Réunion"
  }
  else if(substr(as.character(DATA$Code.Postal[i
      ]),1,3)==976){
    DATA$Departements[i] <- "Mayotte"
    DATA$Region[i] <- "Mayotte"
  }
}
```

7.4 Ajout des codes de départements

```
DATA$CodeDep <- departements.region[match(DATA$Departements,departements.region$dep_name),1]  #Rajout
    de code de departement
```

7.5 Exportation des données en format CSV

```
write.csv(DATA,"~/<Chemin>/<nom_dataset_ANNEE>.csv")
```

7.6 Merge des datasets

```
FINAL_DATA <- rbind(<Nom_DATASET_1>,<Nom_DATASET_2>,<Nom_DATASET_3>, <Nom_DATASET_4>, <Nom_DATASET_5>)
    #merge des datasets
```

7.7 fichier DTD

```
<!ELEMENT dataset (ligne*)>
  <!ELEMENT ligne (date,localisation,categorie,genre,salaire)>
  <!ATTLIST ligne id CDATA #REQUIRED>
  <!ELEMENT localisation (codePostal,ville,departement,region,codeDep)>
  <!ELEMENT categorie EMPTY>
      <!ATTLIST categorie nom (Cadre|Profession|Employé|Ouvrier) #REQUIRED>
  <!ELEMENT genre EMPTY>
      <!ATTLIST genre nom (Femme|Homme) #REQUIRED>
  <!ELEMENT date EMPTY>
      <!ATTLIST date annee (2012|2013|2014|2015|2016) #REQUIRED>
  <!ELEMENT salaire (#PCDATA)>
  <!ELEMENT codePostal (#PCDATA)>
  <!ELEMENT ville (#PCDATA)>
  <!ELEMENT departement (#PCDATA)>
  <!ELEMENT region (#PCDATA)>
  <!ELEMENT codeDep (#PCDATA)>
```

7.8 Conversion CSV en XML

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

public class ParserCSVXML {

  static String path = "C:\\Users\\Matt\\Desktop\\FACM2Data\\BDD\\FINALCSV\\SALAIREFINAL.txt";
  static String outputFile = "C:\\\\Users\\\\Matt\\\\Desktop\\\\FACM2Data\\\\BDD\\\\FINALCSV\\\\XMLres.
      xml";

  public static void main(String[] args) {
    BufferedReader reader;
    try {
      reader = new BufferedReader(new FileReader(path));
      String line = reader.readLine();
      String[] headers = line.split(",");
      try
      {
        Files.write(Paths.get(outputFile), "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<dataset>\n".
            getBytes(), StandardOpenOption.APPEND);
      }
      catch(IOException e) {}
      line = reader.readLine();
      while(line != null)
      {
          String[] data = line.split(",");
        if(data.length > 1)
        {
          String section = "    <ligne id=\"" + data[0].replaceAll("\"", "") + "\">\n        <date
              annee=\""+ data[1].replaceAll("\"", "") +
              "\"></date>\n        <localisation>\n            <codePostal>"
              + data[2].replaceAll("\"", "") + "</codePostal>\n            <ville>"+data[3].replaceAll(
                  "\"", "")
              +"</ville>\n            <departement>"+data[7].replaceAll("\"", "")+"</departement>\n
                          <region>"+data[8].replaceAll("\"", "")+"</region>\n            <codeDep>"
              + data[9].replaceAll("\"", "")+"</codeDep>\n        </localisation>\n        <categorie
                  nom=\""
              +data[5].replaceAll("\"", "")+"\"></categorie>\n        <genre nom=\""+data[4].replaceAll
                  ("\"", "")+"\"></genre>\n        <salaire>"+data[6].replaceAll("\"", "")+"</salaire
                  >\n    </ligne>\n";

          try {
            Files.write(Paths.get(outputFile), section.getBytes(), StandardOpenOption.APPEND);
          } catch(IOException e) {}
        }
        line = reader.readLine();
      }
      try{
        Files.write(Paths.get(outputFile), "</dataset>".getBytes(), StandardOpenOption.APPEND);
      } catch(IOException e) {}
    } catch(Exception e) {System.out.println("ERROR");}
  }
}
```

7.9 Requête 1

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  >
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template match="/">
    <table border="1">
      <!-- group by region -->
      <xsl:for-each-group select="dataset/ligne[date/@annee = '2016']" group-by="localisation/region">
        <tr>
          <th>Region: <xsl:value-of select="current-grouping-key()" /></th>
        </tr>
        <!-- group by category -->
          <xsl:for-each-group select="current-group()" group-by="categorie/@nom">
            <tr>
              <td>Categorie: <xsl:value-of select="current-grouping-key()" /></td>
              <!-- sum up the salary in the grouped category -->
              <td>Total: <xsl:value-of select="avg(current-group()/salaire)" /></td>
            </tr>
          </xsl:for-each-group>
      </xsl:for-each-group>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

7.10 Requête 2

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/1999/xhtml"
    >
    <xsl:output method="html" encoding="UTF-8"/>

    <xsl:template match="/">
        <table border="1">
            <thead>
                <tr>
                    <th> Ville</th>
                    <th> Categorie</th>
                    <th> Difference de salaire (%)</th>
                </tr>
            </thead>
               <tbody>
                <xsl:for-each-group select="dataset/ligne[date/@annee = '2016']" group-by="localisation
                    /ville">
                        <xsl:variable name="ville" select="current-grouping-key()" />
                        <xsl:for-each-group select="current-group()" group-by="categorie/@nom">
                            <tr>
                                <td><xsl:value-of select="$ville" /></td>
                                <td><xsl:value-of select="current-grouping-key()" /></td>
                                <xsl:variable name="homme_total" select="avg(current-group()[genre/@nom
                                    = 'Homme']/salaire)" />
                                <xsl:variable name="femme_total" select="avg(current-group()[genre/@nom
                                    = 'Femme']/salaire)" />
                                <td><xsl:value-of select="format-number((($homme_total - $femme_total)
                                    div $femme_total) * 100), '#.##')" /></td>
                            </tr>
                        </xsl:for-each-group>
                </xsl:for-each-group>
            </tbody>
        </table>
    </xsl:template>
</xsl:stylesheet>
```

7.11 Requête 3

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
```

```
  >
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template match="/">
    <html>
      <head>
        <meta charset="utf-8" />
        <style>
          .femme
          {
            color: blue;
          }
          .homme
          {
            color: red;
          }
        </style>
      </head>

      <body>
        <table border="1">
          <tr>
            <th>Annee</th>
            <th>Total Salaire Femme</th>
            <th>Total Salaire Homme</th>
          </tr>
          <xsl:for-each-group select="dataset/ligne" group-by="date/@annee">
            <xsl:sort select="current-grouping-key()" />
            <tr>
              <td><xsl:value-of select="current-grouping-key()" /></td>
              <xsl:for-each-group select="current-group()" group-by="genre/@nom">
                <td><xsl:value-of select="format-number((avg(current-group()/salaire)), '##.###')" /></
                    td>
              </xsl:for-each-group>
            </tr>
          </xsl:for-each-group>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.12 Requête 4

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  >
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template match="/">
    <table border="1">
      <thead>
            <tr>
                <th> Region</th>
                <th> Catégorie</th>
                <th> Année</th>
                <th> Salaire </th>
            </tr>
        </thead>
        <tbody>
        <xsl:for-each-group select="dataset/ligne" group-by="date/@annee">
          <xsl:sort select="current-grouping-key()" />
          <xsl:variable name="annee" select="current-grouping-key()" />
            <xsl:for-each-group select="current-group()" group-by="localisation/region">
              <xsl:variable name="region" select="current-grouping-key()" />
              <xsl:for-each-group select="current-group()" group-by="categorie/@nom">
                <tr>
                  <td><xsl:value-of select="$region" /></td>
                  <td><xsl:value-of select="current-grouping-key()" /></td>
                  <td><xsl:value-of select="$annee" /></td>
                  <td><xsl:value-of select="format-number(avg(current-group()/salaire), '##.###')" /></
                      td>
                </tr>
              </xsl:for-each-group>
          </xsl:for-each-group>
        </xsl:for-each-group>
```

```
      </tbody>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

7.13 Requête 5

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  >
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template match="/">
    <table border="1">
      <tr>
        <th>Ville</th>
        <th>Total salaire (homme et femme)</th>
      </tr>
      <xsl:for-each-group select="dataset/ligne[date/@annee = '2016']/localisation/ville" group-by=".">
        <tr>
          <td><xsl:value-of select="current-grouping-key()" /></td>
          <td><xsl:value-of select="format-number(avg(current-group()/../../salaire), '##.###')" /></td
              >
        </tr>
      </xsl:for-each-group>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

7.14 Requête 6

```
<?xml version='1.0' encoding='UTF-8'?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml"
  >
  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:template match="/">
    <!-- calcul evolution salaire 2012 - 2013 -->
    <xsl:variable name="salaire_total_homme_2012" select="avg(dataset/ligne[date/@annee = '2012' and
        genre/@nom = 'Homme']/salaire)" />
    <xsl:variable name="salaire_total_femme_2012" select="avg(dataset/ligne[date/@annee = '2012' and
        genre/@nom = 'Femme']/salaire)" />

    <xsl:variable name="salaire_total_homme_2013" select="avg(dataset/ligne[date/@annee = '2013' and
        genre/@nom = 'Homme']/salaire)" />
    <xsl:variable name="salaire_total_femme_2013" select="avg(dataset/ligne[date/@annee = '2013' and
        genre/@nom = 'Femme']/salaire)" />


    <!-- calculs evolution salaire 2013 - 2014 -->
    <xsl:variable name="salaire_total_homme_2014" select="avg(dataset/ligne[date/@annee = '2014' and
        genre/@nom = 'Homme']/salaire)" />
    <xsl:variable name="salaire_total_femme_2014" select="avg(dataset/ligne[date/@annee = '2014' and
        genre/@nom = 'Femme']/salaire)" />


    <!-- calculs evolution salaire 2014 - 2015 -->
    <xsl:variable name="salaire_total_homme_2015" select="avg(dataset/ligne[date/@annee = '2015' and
        genre/@nom = 'Homme']/salaire)" />
    <xsl:variable name="salaire_total_femme_2015" select="avg(dataset/ligne[date/@annee = '2015' and
        genre/@nom = 'Femme']/salaire)" />

    <!-- calculs evolution salaire 2015 - 2016 -->
    <xsl:variable name="salaire_total_homme_2016" select="avg(dataset/ligne[date/@annee = '2016' and
        genre/@nom = 'Homme']/salaire)" />
    <xsl:variable name="salaire_total_femme_2016" select="avg(dataset/ligne[date/@annee = '2016' and
        genre/@nom = 'Femme']/salaire)" />


    <table border="1">
      <tr>
        <th>Periode</th>
```

```
        <th>Evolution femme</th>
        <th>Evolution homme</th>
      </tr>
      <tr>
        <td>2012 - 2013</td>
        <td><xsl:value-of select="(($salaire_total_femme_2013 - $salaire_total_femme_2012) div
            $salaire_total_femme_2012) * 100" /></td>
        <td><xsl:value-of select="(($salaire_total_homme_2013 - $salaire_total_homme_2012) div
            $salaire_total_homme_2012) * 100" /></td>
      </tr>
      <tr>
        <td>2013 - 2014</td>
        <td><xsl:value-of select="(($salaire_total_femme_2014 - $salaire_total_femme_2013) div
            $salaire_total_femme_2013) * 100" /></td>
        <td><xsl:value-of select="(($salaire_total_homme_2014 - $salaire_total_homme_2013) div
            $salaire_total_homme_2013) * 100" /></td>
      </tr>
      <tr>
        <td>2014 - 2015</td>
        <td><xsl:value-of select="(($salaire_total_femme_2015 - $salaire_total_femme_2014) div
            $salaire_total_femme_2014) * 100" /></td>
        <td><xsl:value-of select="(($salaire_total_homme_2015 - $salaire_total_homme_2014) div
            $salaire_total_homme_2014) * 100" /></td>
      </tr>
      <tr>
        <td>2015 - 2016</td>
        <td><xsl:value-of select="(($salaire_total_femme_2016 - $salaire_total_femme_2015) div
            $salaire_total_femme_2015) * 100" /></td>
        <td><xsl:value-of select="(($salaire_total_homme_2016 - $salaire_total_homme_2015) div
            $salaire_total_homme_2015) * 100" /></td>
      </tr>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

7.15 Requête 7

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:output method="html" doctype-public="XSLT-compat" omit-xml-declaration="yes" encoding="UTF-8"
        indent="yes" />

    <xsl:template match="/">
      <html>
        <table border="1">
            <thead>
                <tr>
                    <th>Categorie</th>
                    <th>Ville</th>
                    <th>Salaire</th>
                </tr>
            </thead>
            <tbody>
                <xsl:for-each-group select="dataset/ligne[date/@annee = '2016']" group-by="categorie/
                    @nom">
                <xsl:variable name="m" select="max(current-group()/salaire)"/>
                <xsl:variable name="max" select="current-group()[salaire= $m]"/>
                <tr>
                    <td><xsl:value-of select="current-grouping-key()"/></td>

               <td><xsl:value-of select="$max/localisation/ville"/></td>
               <td><xsl:value-of select="$m"/></td>


                   </tr>
                </xsl:for-each-group>
            </tbody>
        </table>
        </html>
    </xsl:template>
</xsl:transform>
```

7.16 Requête 8

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
```

```
3
4       <xsl:output method="html" doctype-public="XSLT-compat" omit-xml-declaration="yes" encoding="UTF-8"
            indent="yes" />
5
6       <xsl:template match="/">
7         <html>
8           <table border="1">
9               <thead>
10                  <tr>
11                      <th>Prévision 2020</th>
12                      <th>Prévision 2025</th>
13                      <th>Prévision 2030</th>
14                  </tr>
15              </thead>
16              <tbody>
17                  <xsl:variable name="ST2012" select="avg(dataset/ligne[date/@annee = '2012']/salaire)"/>
18                  <xsl:variable name="ST2013" select="avg(dataset/ligne[date/@annee = '2013']/salaire)"/>
19                  <xsl:variable name="ST2014" select="avg(dataset/ligne[date/@annee = '2014']/salaire)"/>
20                  <xsl:variable name="ST2015" select="avg(dataset/ligne[date/@annee = '2015']/salaire)"/>
21                  <xsl:variable name="ST2016" select="avg(dataset/ligne[date/@annee = '2016']/salaire)"/>
22
23                  <xsl:variable name="Diff12_13" select="1 + (($ST2013 - $ST2012) div $ST2013)"/>
24                  <xsl:variable name="Diff13_14" select="1 + (($ST2014 - $ST2013) div $ST2014)"/>
25                  <xsl:variable name="Diff14_15" select="1 + (($ST2015 - $ST2014) div $ST2015)"/>
26                  <xsl:variable name="Diff15_16" select="1 + (($ST2016 - $ST2015) div $ST2016)"/>
27
28                  <xsl:variable name="AvgEvolution" select="($Diff12_13 + $Diff13_14 +
29                  $Diff14_15 + $Diff15_16) div 4"/>
30
31                  <xsl:variable name="Prevision2020" select="$ST2016 * ($AvgEvolution * $AvgEvolution *
                        $AvgEvolution
32                      * $AvgEvolution)"/>
33                  <xsl:variable name="Prevision2025" select="$Prevision2020 * ($AvgEvolution *
                        $AvgEvolution * $AvgEvolution
34                      * $AvgEvolution * $AvgEvolution)"/>
35                  <xsl:variable name="Prevision2030" select="$Prevision2025 *  ($AvgEvolution *
                        $AvgEvolution * $AvgEvolution
36                      * $AvgEvolution * $AvgEvolution)"/>
37
38                  <tr>
39                      <td><xsl:value-of select="$Prevision2020"/></td>
40                      <td><xsl:value-of select="$Prevision2025"/></td>
41                      <td><xsl:value-of select="$Prevision2030"/></td>
42                  </tr>
43
44              </tbody>
45          </table>
46          </html>
47      </xsl:template>
48  </xsl:transform>
```

7.17 Requête 9

```
1   <?xml version="1.0" encoding="UTF-8" ?>
2   <xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
3       <xsl:output method="html" doctype-public="XSLT-compat" omit-xml-declaration="yes" encoding="UTF-8"
            indent="yes" />
4
5       <xsl:template match="/">
6         <html>
7           <table border="1">
8               <thead>
9                   <tr>
10                      <th>Département</th>
11                      <th>Catégorie</th>
12                      <th>Ville</th>
13                      <th>Salaire</th>
14                  </tr>
15              </thead>
16              <tbody>
17                  <xsl:for-each-group select="dataset/ligne[date/@annee = '2016']"
18                  group-by="localisation/departement">
19
20
21                      <xsl:variable name="dep" select="current-grouping-key()"/>
22
23                  <xsl:for-each-group select="current-group()" group-by="categorie/@nom">
24                  <tr>
```

```
                    <xsl:variable name="m" select="max(current-group()/salaire)"/>
                    <xsl:variable name="max" select="current-group()[salaire= $m]"/>
                    <td><xsl:value-of select="$dep"/></td>
                    <td><xsl:value-of select="current-grouping-key()"/></td>
                    <td><xsl:value-of select="$max/localisation/ville"/></td>
                    <td><xsl:value-of select="$m"/></td>
                    </tr>
                </xsl:for-each-group>




                </xsl:for-each-group>
            </tbody>
        </table>
        </html>
    </xsl:template>
</xsl:transform>
```

7.18 Requête 10

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

    <xsl:output method="html" doctype-public="XSLT-compat" omit-xml-declaration="yes" encoding="UTF-8"
        indent="yes" />

    <xsl:template match="/">
      <html>
        <table border="1">
            <thead>
                <tr>
                    <th>Meilleure Ville</th>
                    <th>Salaire</th>
                </tr>
            </thead>
            <tbody>
                    <tr>
                    <xsl:variable name="m" select="max(dataset/ligne[date/@annee = '2016']/salaire)"/>
                    <xsl:variable name="max" select="dataset/ligne[salaire = $m]"/>
                        <td>
                            <xsl:value-of select="$max/localisation/ville"/>
                        </td>
                        <td>
                            <xsl:value-of select="$m"/>
                        </td>
                    </tr>
            </tbody>
        </table>
        </html>
    </xsl:template>
</xsl:transform>
```