BROHAN Romain
DROMIGNY--CHEVREUIL Ivan
GASCOIN-FONTAINE Samy

Rapport de Base de données évoluée

Entrepôts de données

2017-2018

I – Présentation du dataset

Ce projet a pour but la mise en place d'un entrepôt de données sous Oracle avec un sujet spécifique. Dans le cas présent, l'entrepôt de données est basé sur les mouvements sociaux (grèves) de la SNCF depuis 2002 à travers toute la France.

Celui-ci a été récupéré sur le site d'open data de la SNCF. Ci-contre le lien vers le dataset : https://data.sncf.com/explore/dataset/mouvements-sociaux-depuis-2002/?sort=date_de_debut.

Un second dataset avait également été trouvé mais nous avons finalement décidé de ne pas l'incorporer dans notre entrepôt de données car nous avons admis avoir suffisamment de données avec le premier.

II – Conception de l'entrepôt de données

L'entrepôt de données a, comme expliqué précédemment, été implémenté via Oracle. Une étude sur le dataset a d'abord été effectuée afin de déterminer la table de faits ainsi que les différentes dimensions de l'entrepôt de données.

Voici la liste des attributs présents dans le dataset : *date_de_debut, date_de_fin, Motif exprimé, Organisations syndicales, Métiers ciblés par le préavis, Population devant travailler ciblée par le préavis, Nombre de gréviste du préavis* et *Taux de grévistes*.

A partir de cette liste d'attribut nous avons créé une table de fait ainsi que 5 dimensions : *Motifs, Temps, Organisations, Metiers_Cibles, Nb_Travailleurs.*

Nous avons également décidé d'ajouter des attributs supplémentaires afin d'améliorer et de compléter notre entrepôt de données :

- Nous avons rajouté un attribut *saison* dans la dimension *Temps.*
- Un attribut *categorie_greve* dans la dimension *Motifs*, pouvant avoir pour valeur : *Salaire, Conditions de travail, Retraite.* Des catégories supplémentaires beaucoup plus spécifiques aurait pu également être déterminées, bien que ces trois soit amplement suffisantes pour notre entrepôt de données.
- Un attribut *annee* dans la table de faits afin de pouvoir récupérer cette information plus facilement.
- Un atttribut *duree_en_jour* dans la table de faits afin de pouvoir récupérer le nombre de jours (ou la durée) d'une grève plus facilement.
- Le pourcentage de grévistes (attribut *taux_de_greviste*) a également été mis dans la table de fait pour les mêmes raisons que celles énoncées précédemment.

En ce qui concerne l'automatisation de la création des tuples, nous avons décidé de l'effectuer au moyen d'un script C++ que nous avons implémenté nous même. Nous sommes conscients de l'existance de logiciels permettant cette automatisation, tel que Talend, mais nous avons préféré nous tourné vers une technologie que nous connaissions déjà afin de ne pas perdre de temps dans l'apprentissage d'une nouvelle (cela incluant apprendre son fonctionnement, ainsi que toutes les erreurs basiques pouvant survenir).

Ayant convenu que nous laission les multi values dans notre entrepôt de données, nous nous sommes de ce fait dit que Talend n'était pas forcément obligatoire d'utilisation.

De plus, les postes de l'université étant parfois quelque peu compliqué en ce qui concerne l'installation de nouveaux logiciels dessus, nous avions un peu peur en ce qui concernait Talend, tandis que cela ne posait aucun problème pour un programme en C++.

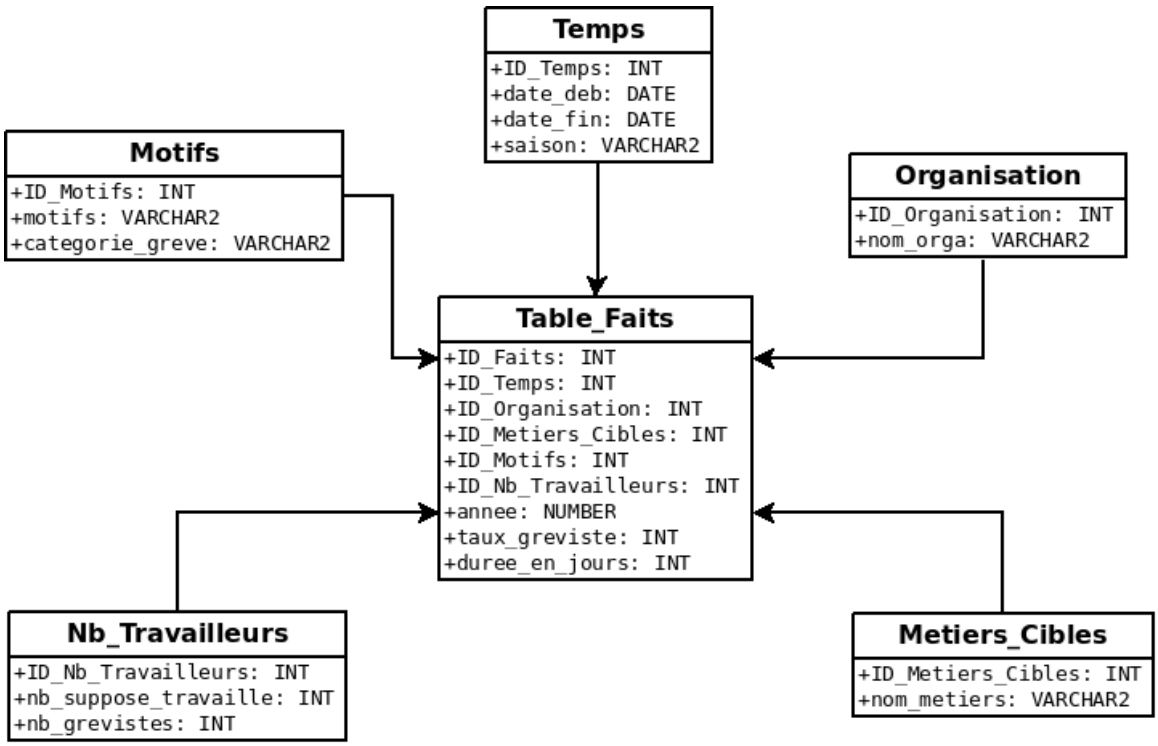

Figure 1- UML de l'entrepôt de données

III - Les requêtes

Dans le cadre de ce projet, nous devions concevoir une dizaine de requêtes en utilisant les opérateurs d'OLAP (GROUP BY, GROUP BY CUBE/ROLLUP, RANK, etc..) :

Requête n°1 :

En utilisant ROLLUP, affiche le nombre de grévistes total par grève, puis par année, puis le nombre de grévistes total jusqu'en 2009.

```
SELECT annee, date_deb, SUM(nb_grevistes) AS nb_grevistes, GROUPING_ID(annee, date_deb)
FROM (SELECT annee, date_deb, date_fin, nb_grevistes
        FROM Table_Faits NATURAL JOIN Temps NATURAL JOIN Nb_Travailleurs
        GROUP BY (annee, date_deb, date_fin, nb_grevistes)
WHERE nb_grevistes IS NOT NULL AND annee <= 2009
GROUP BY ROLLUP (annee, date_deb);
```

Dans cette requête, "nb_grevistes IS NOT NULL" nous permet de ne pas prendre en compte les grèves dont le nombre de grévistes n'est pas renseigné.

La requête imbriquée nous permet de traiter tout les tuples valides et d'éviter les doublons grâce au GROUP BY (annee, date_deb, date_fin, nb_grevistes). Cela rend ainsi le calcul du SUM exact, puisque dans notre dataset il n'y a pas deux grèves différentes qui se soit déroulées le même jour sur la même période de temps et avec exactement le même nombre de grévistes : si jamais c'est le cas, c'est qu'il s'agit de la même grève.

Le GROUP BY ROLLUP nous permet d'afficher les agrégats correspondant à ce que nous voulons, à savoir, le total par grève, par année, puis le total complet. Le GROUPING_ID permet de mieux différencier les agrégats des tuples résultants.

Requête n°2 :

A l'aide de ROWNUM et RANK, afficher le top 10 des plus grosses grèves en terme de nombre de grévistes depuis 2002.

```
SELECT *
FROM
    (SELECT date_deb, date_fin, nb_grevistes, RANK() over(ORDER BY nb_grevistes DESC) AS Top_10
    FROM Table_Faits NATURAL JOIN Temps NATURAL JOIN Nb_Travailleurs
    WHERE nb_grevistes IS NOT NULL
    GROUP BY (date_deb, date_fin, nb_grevistes))
WHERE ROWNUM <=10 ;
```

Pour cette requête il nous a fallut utiliser une requête imbriquée. En effet, on utilise ici un *ORDER BY* ainsi qu'un ROWNUM, or il s'avère que si ces deux opérateurs se retrouvent dans la même requête, ROWNUM va s'appliquer avant ORDER BY, ce qui nous a retourné non pas le top 10 des plus grosses grèves, mais le classement par ordre décroissant du nombre de gréviste des 10 premiers tuples ajoutés à la table ! C'est pourquoi nous avons imbriqué l'ORDER BY afin qu'il agisse avant le ROWNUM.

Une fois de plus, IS NOT NULL permet d'éviter les grèves dont le nombre de grévistes n'a pas été renseigné. De plus avec un ORDER BY, les tuples dont la valeur est null sont placés en premier, ce qui nous donnerait un Top 10 de valeurs null si nous n'utilisions pas cet opérateur.

RANK() nous permet d'ajouter une colonne indiquant la place de la grève dans le top 10.

Requête n°3 :

A l'aide de GROUPING SET, affiche le nombre total de grèvistes par an pour chaque année.

```
SELECT annee, SUM(nb_grevistes) AS nb_grevistes
FROM Table_Faits NATURAL JOIN Nb_Travailleurs
 GROUP BY GROUPING SETS (annee)
ORDER BY annee ASC;
```

GROUPING SETS permet ici d'afficher uniquement les aggrégats voulu, c'est à dire ici le total par an.

Requête n°4 :

Affiche le pourcentage de grève pour chaque catégorie pour chaque année à l'aide de PARTITION BY et RATIO_TO_REPORT.

```
SELECT annee, categorie_greve, ROUND(100*RATIO_TO_REPORT(Total) over (PARTITION BY annee),2)
ratio_type_greve
FROM (SELECT annee, 'Salaire' AS categorie_greve, COUNT(*) AS Total
        FROM Table_Faits NATURAL JOIN Motifs
        WHERE categorie_greve LIKE('%Salaire%')
        GROUP BY annee
        UNION
        SELECT annee, 'Retraite' AS categorie_greve, COUNT(*) AS Total
        FROM Table_Faits NATURAL JOIN Motifs
        WHERE categorie_greve LIKE('%Retraite%')
        GROUP BY annee
        UNION
        SELECT annee, 'Condition de travail' AS categorie_greve, COUNT(*) AS Total
        FROM Table_Faits NATURAL JOIN Motifs
        WHERE categorie_greve LIKE('%Conditions de travail%')
        GROUP BY annee
        UNION
        SELECT annee, 'Not defined' AS categorie_greve, COUNT(*) AS Total
        FROM Table_Faits NATURAL JOIN Motifs
        WHERE categorie_greve IS NULL
        GROUP BY annee)
GROUP BY (annee, categorie_greve, Total)
ORDER BY annee;
```

RATIO_TO_REPORT est une fonction analytique qui prend en paramètre une colonne et calcule la part que représente ce paramètre par rapport à la somme des paramètres des autres lignes telles que définies dans la clause WHERE. Ici, grâce à de multiples requêtes imbriquées, nous appliquons RATIO_TO_REPORT à la colonne Total qui est le fruit de l'union des quatre tables recensant toutes les grèves de catégories Retraite, Condition de travail et Salaire, ainsi que celles n'ayant pas de catégorie définie.

PARTITION BY sert ici à créer des partitions par année, et ainsi créer un pourcentage pour chaque catégorie pour chaque année.

L'union des quatre tables provient du fait que nous avons du multi values dans nos colonnes. Nous sommes donc obligés d'effectuer différentes requêtes visant à récupérer différentes informations et à l'assembler au moyen d'UNION. En fonction de ce que l'on récupère, nous allons ainsi rentrer une valeur dans une colonne temporaire qui aura le même nom pour toutes les parties à unir. (exemple : 'Not defined' AS categorie_greve ou 'Retraite' AS categorie_greve). Dans le cas présent la colonne temporaire va avoir le même nom qu'une colonne présente dans la dimension *Motifs* mais cela ne pose pas de problèmes.

Requête n°5 :

Affiche le nombre de gréviste par an et par association (uniquement pour CGT, SUD et CFDT) entre 2002 et 2005 en utilisant grouping sets.

```
SELECT annee, Syndics, SUM(DISTINCT nb_grevistes) AS nb_grevistes
FROM (SELECT annee, 'CGT' AS Syndics, nom_orga, nb_grevistes
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles NATURAL JOIN
        Nb_Travailleurs
        WHERE nom_orga LIKE('%CGT%')
        GROUP BY annee, nom_orga, nb_grevistes
        UNION
        SELECT annee, 'SUD' AS Syndics, nom_orga, nb_grevistes
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles NATURAL JOIN
        Nb_Travailleurs
        WHERE nom_orga LIKE('%SUD%')
        GROUP BY annee, nom_orga, nb_grevistes
        UNION
        SELECT annee, 'CFDT' AS Syndics, nom_orga, nb_grevistes
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles NATURAL JOIN
        Nb_Travailleurs
        WHERE nom_orga LIKE('%CFDT%')
        GROUP BY annee, nom_orga, nb_grevistes)
WHERE annee >= 2002 and annee <= 2005
GROUP BY GROUPING SETS((annee, Syndics), (annee), ())
ORDER BY annee, Syndics;
```

Comme pour la requête précédente, le problème du multi values est toujours présent. Nous avons donc ici choisi de ne récupérer que les syndicats CGT, SUD et CFDT, nous avons stocké les résultats dans une colonne temporaire appellé *Syndics*.

Requête n°6 :

A l'aide de ROLLUP, affiche le nombre de grèves par saison, ainsi que le pourcentage de gréviste.

```
SELECT saison, AVG(taux_grevistes) AS Pourcentage, COUNT(*) AS nb_greves
FROM Table_Faits NATURAL JOIN Temps
GROUP BY ROLLUP (saison) ;
```

La fonction AVG( ) permet de calculer la moyenne sur un ensemble de données numériques.

GROUP BY ROLLUP nous permet d'afficher le nombre de grévistes et le taux moyen par saison, ainsi qu'au cours de l'année.

Requête n°7 :

A l'aide du GROUP BY CUBE affiche tous les agrégats existant entre les métiers concernés, les syndicats présents et l'année des grèves.

```
SELECT annee, Syndics, Metier, COUNT(*) AS Total
FROM
        (SELECT annee, 'CGT' AS Syndics, 'tractionnaire' AS Metier
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles
        WHERE nom_orga LIKE('%CGT%') AND nom_metier LIKE('%tractionnaire%')
        UNION
        SELECT annee, 'SUD' AS Syndics, 'tractionnaire' AS Metier
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles
        WHERE nom_orga LIKE('%SUD%') AND nom_metier LIKE('%tractionnaire%')
        UNION
        SELECT annee, 'CGT' AS Syndics, 'agent de manoeuvre' AS Metier
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles
        WHERE nom_orga LIKE('%CGT%') AND nom_metier LIKE('%agent de manoeuvre%')
        UNION
        SELECT annee, 'SUD' AS Syndics, 'agent de manoeuvre' AS Metier
        FROM Table_Faits NATURAL JOIN Organisations NATURAL JOIN Metiers_Cibles
        WHERE nom_orga LIKE('%SUD%') AND nom_metier LIKE('%agent de manoeuvre%'))
WHERE annee >= 2010
GROUP BY CUBE (annee, Syndics, Metier)
ORDER BY annee;
```

Pour cette requête nous avons encore le problème d'atomicité. Afin de palier à ce problème il a donc fallut faire (n*m) – 1 UNION, où n est le nombre d'attributs différents de la dimension *Organisations* que l'on veut utiliser et m le nombre d'attributs différents de la dimension *Metier_Cibles* que l'on veut utiliser.

Requête n°8 :

Top 5 des grèves les plus longues pour lesquelles la CGT était présente.

```
SELECT *
FROM (SELECT duree_en_jour, annee, nb_grevistes
        FROM Table_Faits NATURAL JOIN Nb_Travailleurs NATURAL JOIN Organisations
        WHERE nb_grevistes IS NOT NULL AND nom_orga LIKE('%CGT%')
        ORDER By duree_en_jour DESC)
WHERE ROWNUM <= 5;
```

Le top 5 est ici effectué sans utilisé le RANK( ) et uniquement avec un ORDER BY ainsi qu'un ROWNUM. Les résultats seront donc bien classés par ordre décroissant mais sans annotation de rang.

Requête n°9 :

A l'aide du GROUP BY CUBE affiche tous les agrégats existant entre les différentes catégories de grève, la saison et l'année des grèves.

```
SELECT annee, categorie_greve, saison, COUNT(*) AS Total
FROM (SELECT annee, 'Salaire' AS categorie_greve, saison
                FROM Table_Faits NATURAL JOIN Motifs NATURAL JOIN Temps
                WHERE categorie_greve LIKE('%Salaire%')
                UNION
                SELECT annee, 'Retraite' AS categorie_greve, saison
                FROM Table_Faits NATURAL JOIN Motifs NATURAL JOIN Temps
                WHERE categorie_greve LIKE('%Retraite%')
                UNION
                SELECT annee, 'Condition de travail' AS categorie_greve, saison
                FROM Table_Faits NATURAL JOIN Motifs NATURAL JOIN Temps
                WHERE categorie_greve LIKE('%Conditions de travail%'))
WHERE annee >= 2008
GROUP BY CUBE (annee, categorie_greve, saison)
ORDER BY annee;
```

Requête n°10 :

Usage d'une fenêtre mobile pour afficher un cumul du nombre de grévistes par an.

```
SELECT annee, nb_grevistes, SUM(nb_grevistes) over (ORDER BY annee rows unbounded preceding) AS Cumul
FROM Table_Faits NATURAL JOIN Nb_Travailleurs
GROUP BY annee, nb_grevistes;
```

Une simple requête qui sert d'exemple pour la clause OVER bien que nous n'en ayons pas vraiment vu l'utilité dans notre dataset.

IV – Conclusion

Plusieurs améliorations sont possibles en ce qui concerne cet entrepôt de données :

- L'utilisation de Talend permettrait une meilleurs récupération et création des tuples. Entre autre la suppression des multi values de manière automatisé.
- Il aurait également été possible de créer des vues pouvant contenir les différents éléments souhaités pour les attributs contenants des multi values. Cela aurait permis de les réutiliser dans les différentes requêtes si plusieurs de ces requêtes avaient eu besoin des mêmes informations. Dans notre cas, nous avions toujours besoins d'informations bien spécifiques, ce qui n'aurait pas forcément été très optimisé de créer des vues pour une seule requête à chaque fois.