Mustapha NEZZARI

Dans ce compte rendu, vous trouverez les réponses aux exercices 1 et 2 du TP1. Chaque réponse est composée de la requête SQL, d’une capture d’écran du résultat et d’une explication de la requête.

Compte rendu du TP1

SQL OLAP

Sommaire

[Exercice 1 4](#_Toc467346025)

[Question a 4](#_Toc467346026)

[Requête 4](#_Toc467346027)

[Résultat 4](#_Toc467346028)

[Explications 4](#_Toc467346029)

[Question b 5](#_Toc467346030)

[Requête 5](#_Toc467346031)

[Résultat 5](#_Toc467346032)

[Explications 5](#_Toc467346033)

[Question c 6](#_Toc467346034)

[Requête 6](#_Toc467346035)

[Résultat 6](#_Toc467346036)

[Explications 6](#_Toc467346037)

[Question d 7](#_Toc467346038)

[Requête 7](#_Toc467346039)

[Résultat 7](#_Toc467346040)

[Explications 7](#_Toc467346041)

[Question e 8](#_Toc467346042)

[Question f 9](#_Toc467346043)

[Requête 9](#_Toc467346044)

[Résultat 9](#_Toc467346045)

[Explications 9](#_Toc467346046)

[Question g 10](#_Toc467346047)

[Requête 10](#_Toc467346048)

[Résultat 10](#_Toc467346049)

[Explications 10](#_Toc467346050)

[Exercice 2 12](#_Toc467346051)

[Question 1 12](#_Toc467346052)

[Requête 12](#_Toc467346053)

[Résultat 12](#_Toc467346054)

[Explications 12](#_Toc467346055)

[Question 2 14](#_Toc467346056)

[Requête 14](#_Toc467346057)

[Résultat 14](#_Toc467346058)

[Explications 14](#_Toc467346059)

[Question 3 15](#_Toc467346060)

[Requête 15](#_Toc467346061)

[Résultat 15](#_Toc467346062)

[Explications 16](#_Toc467346063)

[Question 4 17](#_Toc467346064)

[Requête 17](#_Toc467346065)

[Résultat 17](#_Toc467346066)

[Explications 17](#_Toc467346067)

[Question 5 19](#_Toc467346068)

[Requête 19](#_Toc467346069)

[Résultat 19](#_Toc467346070)

[Explications 19](#_Toc467346071)

[Question 6 20](#_Toc467346072)

[Requête 20](#_Toc467346073)

[Résultat 20](#_Toc467346074)

[Explications 21](#_Toc467346075)

[Question 7 22](#_Toc467346076)

[Requête 22](#_Toc467346077)

[Résultat 22](#_Toc467346078)

[Explications 22](#_Toc467346079)

[Question 8 23](#_Toc467346080)

[Requête 23](#_Toc467346081)

[Résultat 23](#_Toc467346082)

[Explications 24](#_Toc467346083)

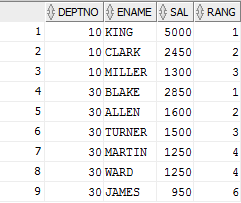
# Exercice 1

## Question a

### Requête



### Résultat



### Explications

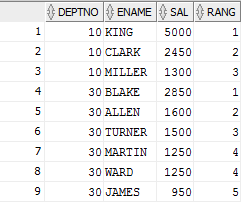
Cette requête classe les salaires des employés pour les départements 10 et 30. Ce classement est réalisé grâce à la fonction fenêtrée Rank () qui utilise la clause partition **BY** pour faire des groupes de classements par département. La clause **ORDER** **BY** sal permet de définir sur quelle colonne le tri est fait dans chaque groupe, ici le salaire par ordre décroissant. On filtre enfin sur les départements 10 et 30 grâce à la clause **WHERE** deptno = 10 **OR** deptno = 30.

## Question b

### Requête



### Résultat



### Explications

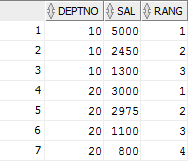
Cette requête effectue la même chose que la requête précédente à la seule différence que les trous sont comblés dans le rang grâce à la fonction fenêtrée Dense\_rank ().

## Question c

### Requête



### Résultat



### Explications

Cette requête classe le salaire des employés par ordre décroissant pour les départements 10 et 20. On utilise la même requête qu’à la question précédente sauf que l’on filtre cette fois sur les départements 10 et 30 et que l’on a une clause **DISTINCT** pour ne pas avoir de doublons au niveau des salaires si par exemple deux employés gagnent le même salaire. Enfin la clause **ORDER** **BY** deptno permet d’ordonner le tout par département.

## Question d

### Requête

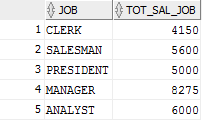
#### Avec le Group By



#### Avec le Partition by



### Résultat



### Explications

Cette requête permet de connaître le salaire total versé par profession. Il est possible de l’avoir par deux méthodes. La première en utilisant la fonction SUM () sur les salaires et en faisant une agrégation sur les jobs. La deuxième en utilisant la fonction SUM () et la CLAUSE OVER qui va nous permettre de définir une zone sur laquelle nous allons travailler. Cette zone est définie par la clause PARTITION BY qui permet de réaliser des groupes. Ici on fait la somme des salaires par job, on veut donc un groupe par job pour pouvoir ensuite faire la somme des salaires dans chacun de ces groupes. Comme la clause PARTITION BY n’affecte pas le nombre de lignes retournées (cf question e), on a une ligne par entrée dans la table EMP. Il faut donc utiliser la clause DISTINCT pour ne pas avoir de doublon.

## Question e

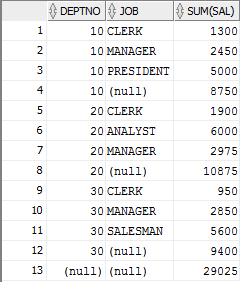
La différence entre la clause GROUP BY et PARTITION BY est que la clause GROUP BY permet de faire une agrégation selon certaines colonnes. Cela affecte donc le nombre de lignes retournées. La clause PARITION BY n’affecte pas le nombre de lignes retournées car elle n’agrège pas les lignes. Elle définit plutôt des groupes selon les colonnes sélectionnées. C’est pour cela qu’on l’utilise souvent avec la clause DISTINCT pour ne pas avoir de doublon.

## Question f

### Requête



### Résultat



### Explications

Cette requête permet de connaître le montant total des salaires versés :

* Tout département et job confondu (en orange)
* Par département (en vert)
* Par département et job : (en bleu)

Ceci est possible grâce à la clause **GROUP** **BY** rollup ( deptno, job ) qui va permettre à la fonction SUM () de faire la somme sur des granularité de plus en plus grande, c’est-à-dire d’abord sur l’ensemble bleu, puis vert puis orange.

## Question g

### Requête

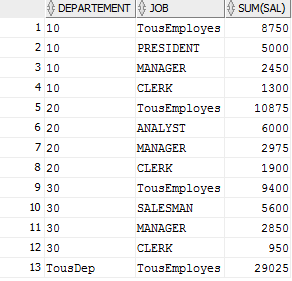
#### Avec NVL ()



#### Avec DECODE ()



### Résultat



### Explications

Les deux requêtes font exactement la même chose qu’à la question précédente si ce n’est que les valeurs (null) sont remplacées par du texte grâce aux fonctions NVL () et DECODE ().

La fonction NVL (texte, defaut) prend en argument :

* texte : Le texte à afficher
* defaut : La valeur par défaut à afficher si le premier argument est (null)

La fonction DECODE (expression, [recherche, resultat]+, defaut) prend en argument :

* expression : L’expression dans laquelle on va rechercher des valeurs
* [recherche, resultat] : Un bloc qui peut se répéter, qui comprend deux arguments :
  + recherche : La valeur à rechercher (et à remplacer par l’argument suivant) dans l’expression
  + resultat : La valeur qui remplacera l’argument précédent s’il est trouvé
* defaut : La valeur par défaut si aucune des expressions du bloc précédent n’est trouvée.

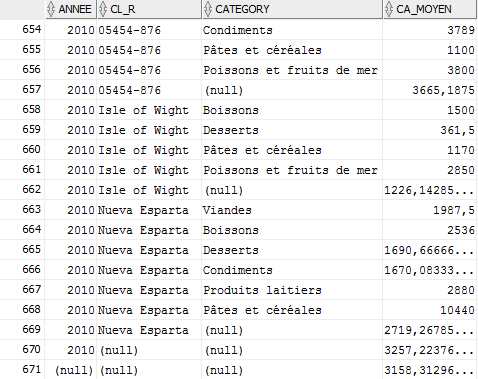
# Exercice 2

## Question 1

### Requête



### Résultat



### Explications

Cette requête permet de connaître la moyenne des ventes par :

* année, région et catégorie (bleu)
* année et région (vert)
* année (orange)
* Toute année région et catégorie confondue (jaune)

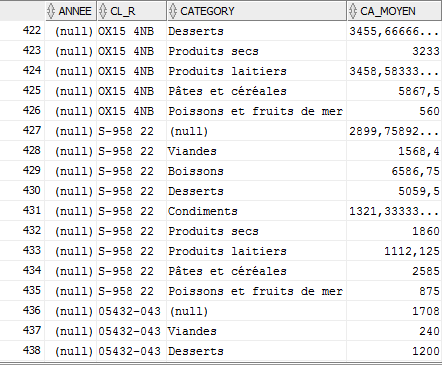
pour les années 2009 et 2010. Pour ça on utilise la clause **GROUP** **BY** rollup ( annee, cl\_r, category ). Le filtre sur les années 2009 et 2010 est réalisé grâce à la clause **WHERE** annee = 2009 **OR** annee = 2010. Enfin pour pouvoir utiliser les informations des autres tables, on utilise des jointures sur les id.

## Question 2

### Requête



### Résultat



### Explications

A COMPLETER

## Question 3

### Requête



### Résultat



### Explications

Cette requête permet de connaître pour chaque année le produit le mieux vendu par catégorie. Pour cela, on fait dans un premier temps la somme des ventes par produit, par catégorie et pour chaque année et dans un second temps on prend le produit où le chiffre d’affaire est le plus grand par catégorie et par année.

La première étape est réalisée grâce à une sous requête qui classe les produits par catégorie et par année selon leur chiffre d’affaire. Ce classement est fait par la fonction Rank() over ( PARTITION **BY** t.annee, p.category **ORDER** **BY** SUM(qte\*pu) **DESC**) RANG. La clause PARITION BY permet de faire des groupe d’année et dans chacun de ces groupes, des groupes de catégorie. On peut ensuite faire le classement des produits, dans chaque groupe de catégorie, par leur chiffre d’affaire grâce à la clause **ORDER** **BY** SUM(qte\*pu) **DESC**.

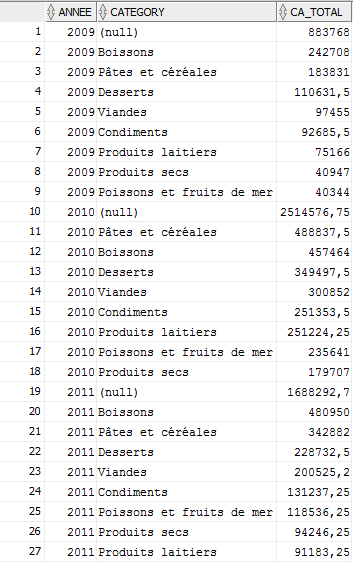
La deuxième étape consiste à ne prendre que les premières lignes de chaque classement qui correspondent aux produits ou le chiffre d’affaire est le plus grand par année et par catégorie.

## Question 4

### Requête



### Résultat



### Explications

Cette requête permet de connaitre le total du CA par :

* année et catégorie (bleu)
* année (vert)

Cela a comme jusqu’à présent été possible grâce à la clause ROLLUP. Cependant pour enlever le sous-total « toute année et catégorie confondue » il a fallu utiliser les GROUPING\_ID qui permettent de connaître le niveau d’agrégation pour chaque ligne de sous-total. Ici on peut représenter notre cas avec ce tableau :

|  |  |  |
| --- | --- | --- |
| Niveau d’agrégation | Bit vector | GROUPING\_ID |
| Année, catégorie | 0 0 | 0 |
| Année | 0 1 | 1 |
| Catégorie | 1 0 | 2 |
| Grand total | 1 1 | 3 |

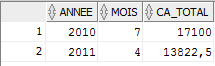
Ici nous voulons cacher le Grand total, il faut donc utiliser une clause qui exclut le GROUPING\_IG n° 3 comme la clause **HAVING** Grouping\_id(t.annee, p.category) != 3.

## Question 5

### Requête



### Résultat



### Explications

Cette requête permet de connaître le meilleur mois de vente du produit « Sirop d’érable » pour chaque année. Pour cela, on fait dans un premier temps la somme des ventes par mois pour chaque année et dans un second temps on prend le mois où le chiffre d’affaire est le plus grand par année.

La première étape est réalisée grâce à une sous requête qui fait la somme par mois et par année et pour le produit Sirop d’érable (**WHERE** p.pname **LIKE** 'Sirop d érable'). On ajoute ensuite une colonne qui permet de voir le classement des mois. Ceci est fait par la fonction RANK qui donne le rang des mois par année grâce à la clause PARTITION BY. Ce rang est trié par le chiffre d’affaire pour placer le mois qui a le plus grand chiffre d’affaire en tête, grâce à la clause **ORDER** **BY** SUM(qte\*pu) **DESC**.

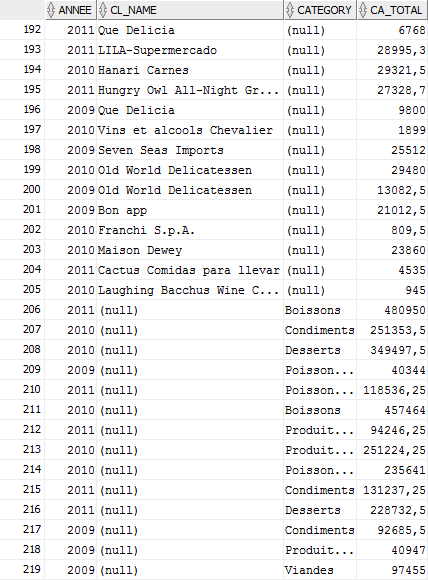
La deuxième étape consiste à ne prendre que les premières lignes de chaque classement qui correspondent aux mois ou le chiffre d’affaire est le plus grand par année pour le produit demandé.

## Question 6

### Requête



### Résultat



### Explications

Cette requête permet de donner tous les totaux de ventes par année et nom du client puis par année et catégorie. Ceci est possible grâce à la clause **GROUP** **BY** grouping sets ( ( t.annee, c.cl\_name ), ( t.annee, p.category ) ).

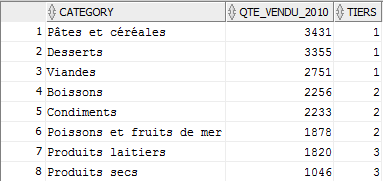
On Remarque donc que l’on a utilisé la clause GROUPING SETS qui permet de préciser les sous-totaux que l’on veut calculer contrairement à la clause ROLLUP qui calcule un jeu prédéfini de sous-totaux.

## Question 7

### Requête



### Résultat



### Explications

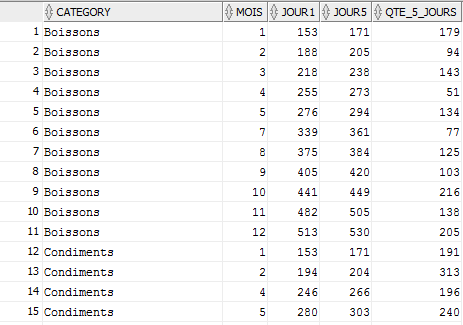
Cette requête donne la répartition par tiers des catégories selon leurs quantités totales vendues en 2010. Pour cela, on filtre les quantité vendues en 2010, on en fait la somme par catégorie puis on répartie ces sommes dans 3 groupes grâce à la fonction Ntile(3) over ( **ORDER** **BY** SUM(qte) **DESC**) **AS** TIERS.

## Question 8

### Requête



### Résultat



### Explications

Cette requête permet de connaître la quantité de produits vendus pour chaque catégorie, les 5 premiers jours de chaque mois de 2010. J’ai choisi de prendre les 5 premiers jours de ventes et de ne prendre que les mois où j’ai vendu au moins pendant 5 jours. Enfin on connaît aussi la quantité vendu le premier jour (JOUR1) et le 5ème jour (JOUR5) ce qui n’aurait pas été possible si l’on n’avait que 3 jours de ventes par exemple (JOUR5 aurait été a null).

Dans un premier temps, on fait la somme des quantités vendues par jour et on affiche le rang de ces jours par catégorie et par mois grâce à la fonction Dense\_rank() over ( PARTITION **BY** p.category, t.mois **ORDER** **BY** t.jour) **AS** RANG.

Dans un second temps on filtre :

* Pour n’avoir que l’année 2010
* Pour n’avoir que les mois où l’on a au moins 5 jours de vente

Puis on fait la somme des quantités des 5 premiers jours de vente grâce à la fonction SUM(qte\_jours) **AS** QTE\_5\_JOURS et la clause **WHERE** rang **BETWEEN** 1 **AND** 5 **GROUP** **BY** category, mois.