

Dans ce compte rendu, vous trouverez les  
réponses au TP2.

# Compte rendu du TP2

SQL OLAP

Mustapha NEZZARI

---



## Sommaire

---

Modélisation dimensionnelle .....	4
L'entrepôt de données .....	5
La table des faits Vente .....	5
Les mesures .....	5
Script SQL de la vue matérialisée .....	5
La dimension Produit .....	6
Script SQL de la vue matérialisée .....	6
Script SQL de la dimension .....	6
Hiérarchie .....	6
La dimension Client .....	8
Script SQL de la vue matérialisée .....	8
Explications .....	8
Script SQL de la dimension .....	8
Hiérarchie .....	8
La dimension Lieu .....	9
Script SQL de la vue matérialisée .....	9
Explications .....	9
Script SQL de la dimension .....	9
Hiérarchie .....	9
La dimension Temps .....	11
Script SQL de la vue matérialisée .....	11
Explications .....	11
Script SQL de la dimension .....	12
Hiérarchie .....	12
Les vues matérialisées .....	13
Mode de rafraichissement .....	13
Tests .....	13
Les clés primaires et étrangères .....	14
Optimisations .....	15
Les index .....	15
Script SQL .....	15
Explications .....	15
Questions d'exploitation .....	16
Question 1 .....	16
Script SQL .....	16

Résultat .....	16
Explications.....	16
Question 2 .....	17
Script SQL .....	17
Résultat .....	17
Explications.....	17
Question 3 .....	18
Script SQL .....	18
Résultat .....	18
Explications.....	18
Question 4 .....	19
Script SQL .....	19
Résultat .....	19
Explications.....	19

## Modélisation dimensionnelle

Dans le cas présenté, l'entreprise cherche à analyser ses ventes selon des mesures différentes. Elle s'intéresse aux **quantités** vendues, au **prix** de ventes (avec éventuellement des **remises**) et aux **chiffres d'affaire**.

Elle souhaite aussi pouvoir avoir des informations sur les produits vendus comme leur **désignation**, leur **catégorie** et leur **sous-catégorie**.

Elle s'intéresse aussi aux caractéristiques de ses clients. Les questions qu'elle se pose montrent un intérêt pour leur **âge**, leur **tranche d'âge** et leur **sexe**. Afin de cibler ses campagnes promotionnelles, elle s'intéresse à des critères géographiques locaux et internationaux comme le **pays**, la **ville** ou encore le **code postal**.

Enfin, elle veut archiver le tout afin de pouvoir effectuer des analyses temporelles sur certaines **années**, **mois**, **semaines** et **jours**.

En tenant compte de toutes ces informations, j'ai réalisé le schéma en étoile Figure 1.

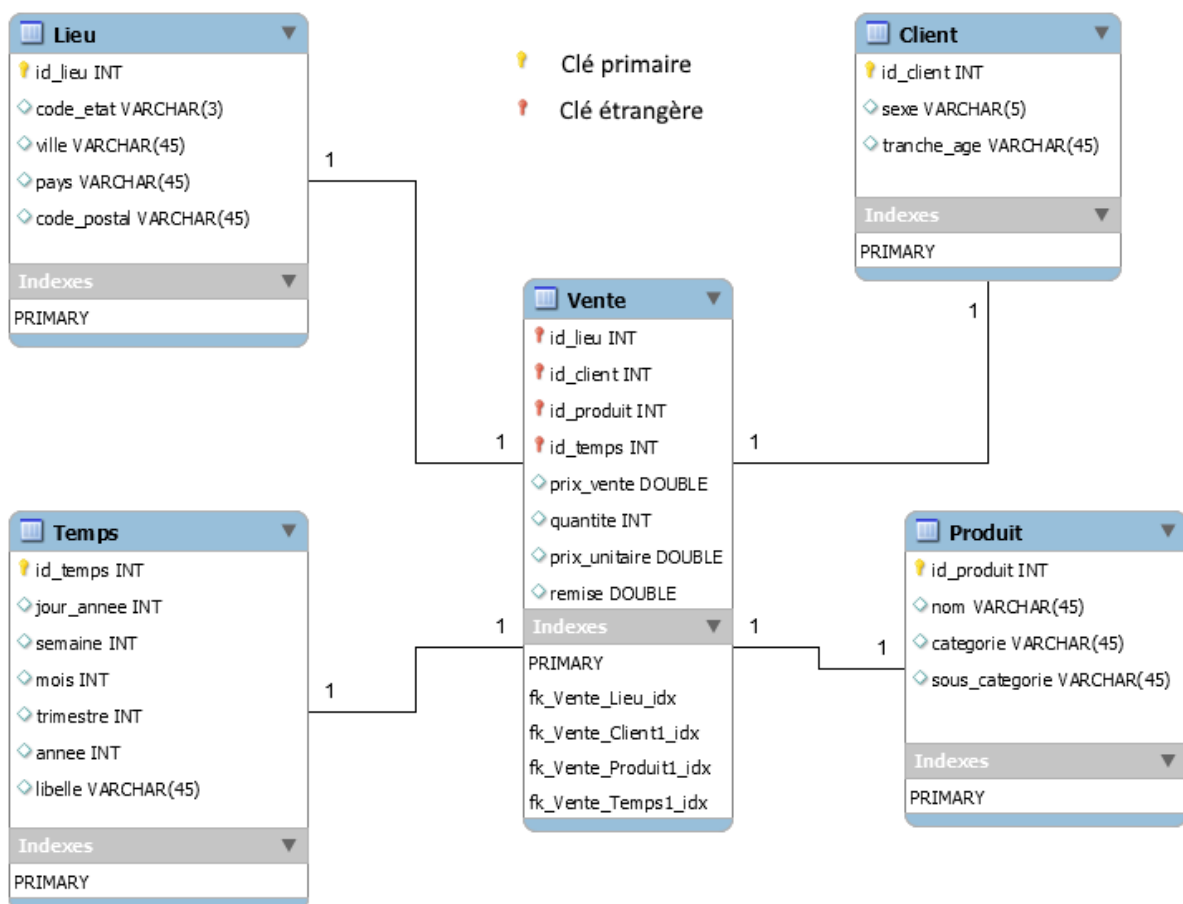


Figure 1

On peut y voir la table de fait Vente et les différentes dimensions qui représentent les axes possibles d'analyse à savoir Client, Lieu, Produit et Temps.

## L'entrepôt de données

Afin de créer cet entrepôt de données, j'ai utilisé les vues matérialisées. Il s'agit ici d'expliquer certains choix ainsi que certaines méthodes utilisées.

### La table des faits Vente

La table des faits Ventes va contenir une entrée par ventes d'un produit en stockant les références du client, du produit vendu, du lieu de vente ainsi que de la date.

Les mesures

En plus des références précédentes, elle va permettre de stocker différentes mesures qui sont :

- La quantité du produit vendu
- Le prix du produit au moment de la vente
- La remise s'il y en a eu une
- Le prix total de ventes qui correspond à la formule : Quantité x Prix unitaire de vente à la date de vente x remise en pourcentage.
- L'âge du client au moment de la vente. J'ai décidé de ne pas mettre l'âge directement dans la dimension Client car on aurait perdu cette notion d'archivage. De plus je pense qu'il est plus pertinent de connaître l'âge de la personne au moment de la vente car l'achat peut être influencé par l'âge et donc la mentalité et le contexte. Par exemple, on pourrait croire qu'une publicité a plus touché les personnes jeunes plutôt que les personnes âgées et l'on pourrait alors le savoir en mettant cette donnée dans la table des ventes.

Script SQL de la vue matérialisée

```
create materialized view vente_vm
REFRESH FORCE
ON DEMAND
Enable query rewrite
as select distinct
  c.id as id_client,
  p.id as id_produit,
  l.id as id_lieu,
  concat(to_char(t.jour_annee), concat('-', to_char(t.annee))) as id_temps,
  (lf.qte * (pud.prix * (1 - pud.remise / 100))) as prix_vente,
  lf.qte as quantite,
  pud.prix as prix_unitaire,
  pud.remise as remise,
  trunc((f.date_etabli - c2.date_nais) / 365.25) as age
from ligne_facture lf
join facture f on lf.facture = f.num
join temps_vm t on concat(to_char(to_number(to_char(f.DATE_ETABLI, 'DDD'))),
concat('-', to_char(to_number(to_char(f.DATE_ETABLI, 'YYYY'))))) = t.id
join client_vm c on f.client = c.id
join client c2 on f.client = c2.num
join lieu_vm l on c.id = l.id
join produit_vm p on lf.produit = p.id
join prix_date pud on lf.id_prix = pud.num;
```

Dans cette requête, la jointure avec la vue Temps se fait en reconstruisant de la même manière qu'il que dans la dimension temps.

L'âge correspond à la formule : date de la facture – date de naissance du client.

## La dimension Produit

Cette dimension représente un produit à vendre.

Script SQL de la vue matérialisée

```
create materialized view produit_vm
REFRESH FORCE
ON DEMAND
Enable query rewrite
as select NUM as id,
    REGEXP_SUBSTR(designation, '^[^.]+' , 1) as nom,
    REGEXP_SUBSTR(designation, '^[^.]+' , INSTR(designation, '.', 1, 1) + 1) as categorie,
    case
        when INSTR(designation, '.', 1, 2) > 0 then REGEXP_SUBSTR(designation, '^[^.]+' ,
INSTR(designation, '.', 1, 2) + 1)
        else null
    end
    as sous_categorie
from produit;
```

### Explications

Dans le SIO on dispose de la table Produit qui possède trois attributs :

- Sa référence (qui ne changera pas dans le SID)
- Sa désignation qui est composé du nom du produit, de sa catégorie et, s'il y en a une, de sa sous-catégorie.
- Son stock

Il a donc fallu découper la désignation pour séparer le nom, la catégorie et la sous-catégorie grâce aux regex. Ici j'ai utilisé la regex '^[^.]+' qui permet de dire « un ou plusieurs caractères excepté le point ».

La fonction NVL () a permis d'insérer une valeur nulle lorsqu'il n'y a pas de sous-catégorie.

Dans la dimension produit je n'ai pas mis le stock car je pense que pour les questions qui permettent de savoir comment améliorer les ventes, on n'a pas besoin de connaître le stock du produit dans le temps. On pourrait le rajouter plus tard si l'on veut mais je pense que c'est une donnée « inutile » puisqu'on a déjà les quantités vendues.

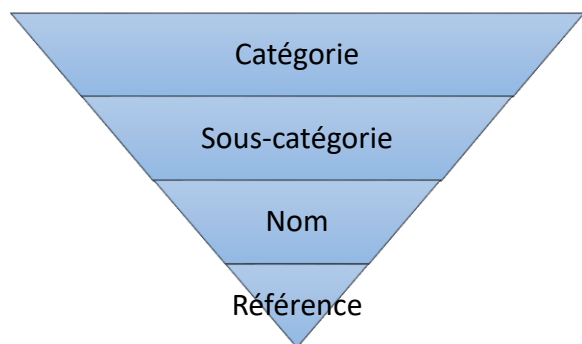
Script SQL de la dimension

```
create dimension produit_dim
    level id is (produit_vm.id)
    level nom is (produit_vm.nom)
    level categorie is (produit_vm.categorie)
    level sous_categorie is (produit_vm.sous_categorie)

    hierarchy produit_rollup (
        id
        child of nom
        child of sous_categorie
        child of categorie
    );
```

### Hiérarchie

La hiérarchie choisi (du plus grand au plus petit niveau) est : Catégorie > Sous-catégorie > Nom > id





## La dimension Client

Cette dimension représente un client qui a acheté au moins une fois.

Script SQL de la vue matérialisée

```
create materialized view client_vm
REFRESH FORCE
ON DEMAND
Enable query rewrite
as select distinct c.num as id, c.sexe,
case
  when trunc(((select max(f.date_etabli) from facture f where f.client = c.num) -
c.date_nais) / 365.25) < 30 then '<30 ans'
  when trunc(((select max(f.date_etabli) from facture f where f.client = c.num) -
c.date_nais) / 365.25) >= 30 and trunc(((select max(f.date_etabli) from facture f
where f.client = c.num) - c.date_nais) / 365.25) <= 45 then '30-45 ans'
  when trunc(((select max(f.date_etabli) from facture f where f.client = c.num) -
c.date_nais) / 365.25) > 45 and trunc(((select max(f.date_etabli) from facture f
where f.client = c.num) - c.date_nais) / 365.25) <= 60 then '46-60 ans'
  else '>60 ans'
end as tranche_age
from client c
join facture f on c.num = f.client;
```

### Explications

Les champs retenus dans la vue sont :

- La référence du client (la même que dans la table du SIO)
- Le sexe (idem)
- La tranche d'âge calculée à partir de la date maximum de vente pour le client. Il pourrait être plus performant d'utiliser la du jour (SYSDATE) plutôt que la date maximum des factures du client.

Le reste des champs de la table Client du SIO ont été utilisé dans la dimension Lieu car la localité est un axe d'analyse.

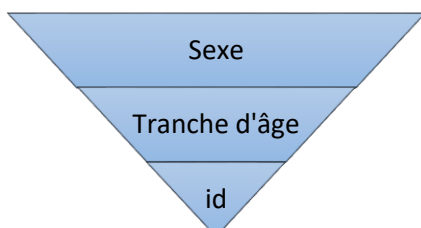
Script SQL de la dimension

```
create dimension client_dim
level id is (client_vm.id)
level sexe is (client_vm.sexe)
level tranche_age is (client_vm.tranche_age)

hierarchy client_rollup (
  id
  child of tranche_age
  child of sexe
);
```

### Hiérarchie

La hiérarchie choisi (du plus grand au plus petit niveau) est : sexe > tranche d'âge > id



## La dimension Lieu

Cette dimension représente le lieu où une (ou plusieurs) vente a été effectuée.

Script SQL de la vue matérialisée

```
create materialized view lieu_vm
REFRESH FORCE
ON DEMAND
Enable query rewrite
as select num as id,
    SUBSTR(REGEXP_SUBSTR(c2.adresse, '^[^,]+', INSTR(c2.adresse, ',', 1, 1) + 1, 3),
1, 3) as code_etat,
    REGEXP_SUBSTR(adresse, '^[^,]+', INSTR(adresse, ',', 1, 1) + 1, 3) as pays,
    REGEXP_SUBSTR(adresse, '^[^,]+', INSTR(adresse, ',', 1, 1) + 1, 2) as ville,
    REGEXP_SUBSTR(adresse, '^[^,]+', INSTR(adresse, ',', 1, 1) + 1, 1) as code_postal
from client;
```

### Explications

Cette vue utilise les informations de la table Client du SIO. Elle est composée :

- D'un id qui correspond à la référence client car chaque client a une référence différente et est associé à une adresse. Même en utilisant l'adresse complète pour l'id, on n'aurait pas pu garantir l'unicité des clés car si deux clients sont deux personnes mariées et habitent à la même adresse, il y aurait des conflits de clés. C'est pourquoi il est plus judicieux de prendre l'id du client comme id ici.
- Un code d'état
- Le pays
- La ville
- Le code postal

Il a fallu découper l'adresse pour séparer les trois derniers champs de la même manière que la désignation des produits.

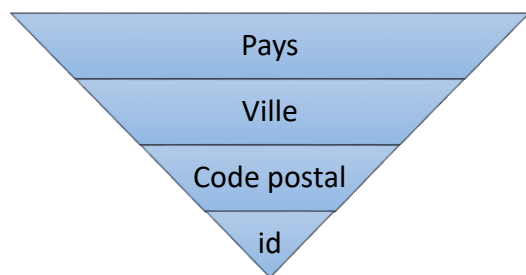
Script SQL de la dimension

```
create dimension lieu_dim
    level id is (lieu_vm.id)
    level pays is (lieu_vm.pays)
    level ville is (lieu_vm.ville)
    level code_postal is (lieu_vm.code_postal)

    hierarchy lieu_rollup (
        id
        child of code_postal
        child of ville
        child of pays
    );
```

### Hiérarchie

La hiérarchie choisi (du plus grand au plus petit niveau) est : Pays > Ville > Code postal > id



## La dimension Temps

Cette dimension est importante car elle permet l'historisation des données.

Script SQL de la vue matérialisée

```
create materialized view temps_vm
REFRESH FORCE
ON DEMAND
Enable query rewrite
as select concat(to_char(to_number(to_char(date_, 'DDD'))), concat('-',
to_char(to_number(to_char(date_, 'YYYY')))) as id,
           to_number(to_char(date_, 'DDD')) as jour_annee,
           to_number(to_char(date_, 'MM')) as mois,
           to_number(to_char(date_, 'YYYY')) as annee,
           to_number(to_char(date_, 'Q')) as trimestre,
           to_number(to_char(date_, 'WW')) as semaine,
           to_char(date_, 'DAY') as libelle
from (select (level + date_minimum - 1) as date_
      from (select
              min(date_etabli) as date_minimum,
              max(date_etabli) as date_maximum
            from facture)
      connect by level < (date_maximum - date_minimum + 2));
```

### Explications

Cette vue est constitué des champs suivants :

- L'id du temps qui est la concaténation du numéro du jour de l'année et de l'année (Par exemple : 365-2017)
- Du numéro du jour de l'année (de 1 à 365)
- Du mois (de 1 à 12)
- De l'année
- Du trimestre (de 1 à 4) que j'ai choisi de rajouter car on pourrait plus tard en avoir besoin
- Du libellé (Du Lundi au Dimanche) que j'ai aussi choisi de rajouter

Pour générer cette table, on utilise la sous requête avec la clause CONNECT BY qui permet de générer des dates entre deux dates. Le minimum et le maximum sont les dates de la plus ancienne et la plus récentes facture.

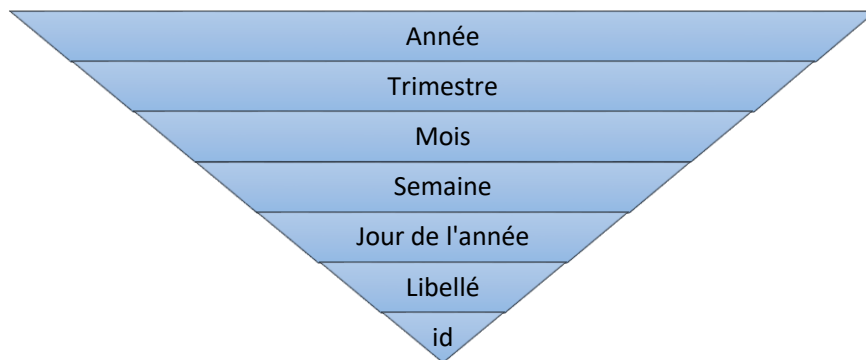
Script SQL de la dimension

```
create dimension temps_dim
  level id is (temps_vm.id)
  level jour_annee is (temps_vm.jour_annee)
  level mois is (temps_vm.mois)
  level annee is (temps_vm.annee)
  level trimestre is (temps_vm.trimestre)
  level semaine is (temps_vm.semaine)
  level libelle is (temps_vm.libelle)

  hierarchy temps_rollup (
    id
    child of libelle
    child of jour_annee
    child of semaine
    child of mois
    child of trimestre
    child of annee
  );
```

Hiérarchie

La hiérarchie choisi (du plus grand au plus petit niveau) est : Année > Trimestre > Mois > Semaine > Jour de l'année > Libellé > id



## Les vues matérialisées

### Mode de rafraichissement

Selon moi, il n'est pas nécessaire de rafraichir les vues à chaque fois qu'il y a une entrée de fait dans le SIO car les données du SID seront utilisées ponctuellement et non de façon continue. J'ai donc décidé d'utiliser la méthode REFRESH FORCE ON DEMAND sur toutes mes vues, qui me permet de manuellement rafraîchir mes vues de façon incrémentale (si c'est possible, sinon toute la vue est rafraîchie).

Il faut donc utiliser les méthodes suivantes pour rafraîchir les vues à chaque fois que l'on veut s'en servir pour prendre des décisions :

```
execute dbms_mview.refresh('CLIENT_VM');  
execute dbms_mview.refresh('LIEU_VM');  
execute dbms_mview.refresh('PRODUIT_VM');  
execute dbms_mview.refresh('TEMPS_VM');  
execute dbms_mview.refresh('VENTE_VM');
```

Figure 2

### Tests

Afin de tester si mes vues fonctionnent, j'ai créé une procédure qui permet d'insérer 4 ventes dans les tables du SIO. Le script de cette procédure est disponible dans le fichier *procedure\_insert.sql*.

Il suffit d'appeler la procédure avec la commande « exec INSERER\_DONNEES; » et de rafraichir les vues grâce au commande de la Figure 2.

## Les clés primaires et étrangères

---

Dans notre cas, nous utilisons des vues matérialisée, il n'est donc pas nécessaires de créer des clés primaires puisqu'elles utilisent les clés des tables du SIO à part pour la table Temps où l'on génère nous même une clé. La création d'un index pourra apporter cette notion d'unicité (si l'on utilise un index unique).

# Optimisations

---

## Les index

Script SQL

```
create unique index client_vm_index on client_vm (id);  
create bitmap index CLIENT_VM_INDEX_TRANCHE_AGE ON client_vm (tranche_age);  
create bitmap index CLIENT_VM_INDEX_SEXE ON client_vm (sexe);  
create index lieu_vm_index on lieu_vm (id);  
create unique index temps_vm_index on temps_vm (id);
```

Explications

Les index sont en général utilisés pour indexer des données (à l'aide de hashmaps par exemple) afin de les retrouver plus rapidement. On les utilise par exemple dans le cas où l'on a des jointures sur plusieurs tables sur lesquelles on a besoin de moins de 15% des données ou lorsqu'on utilise certaines colonnes dans des clauses WHERE.

Normalement, les vues matérialisées créent automatiquement des index sur les clés primaires du SIO. Cependant dans mon cas, seule celle sur la table produit est créée automatiquement. Je pense que c'est à cause des requêtes complexes pour créer les vues matérialisées que les index ne sont pas créés automatiquement pour les autres tables. J'ai donc dû les créer moi-même.

De plus j'ai créé un bitmap index pour le sexe du client car c'est un champ qui n'a que deux valeurs possibles ce qui satisfait les prérequis d'utilisation des bitmap index. J'ai fait de même pour la tranche d'âge qui n'a que quatre valeurs possibles.

Enfin je pense que comme nous voudrions faire des recherches selon toutes les granularités, on peut rajouter les champs suivants :

- Dans lieu : Ville, pays et code postal
- Dans produit : le nom, la catégorie et la sous-catégorie
- Dans temps : Tous les champs.

Cependant je ne le fais pas car je n'ai pas un jeu de données assez grand pour faire mes tests.



## Questions d'exploitation

Les captures d'écran ont été prises en local car le VPN n'a pas voulu fonctionner lorsque j'ai voulu les faire sur ma base filora.

### Question 1

Script SQL

```
SELECT p.id,  
        p.nom,  
        SUM(v.prix_vente) AS CA  
FROM    vente_vm v  
        join produit_vm p  
        ON v.id_produit = p.id  
GROUP BY p.id,  
          p.nom;
```

Résultat



	ID	NOM	CA
1	2	Chang	59508
2	3	Aniseed Syrup	34200
3	4	Chef Anton s Cajun Seasoning	24552
4	5	Chef Anton s Gumbo Mix	30770
5	6	Grandma s Boysenberry Spread	27800
6	7	Uncle Bob s Organic Dried Pears	34680
7	8	Northwoods Cranberry Sauce	102720
8	9	Mishi Kobe Niku	132696
9	10	Ikura	11780
10	11	Queso Cabrales	48790
11	12	Queso Manchego La Pastora	106400
12	13	Konbu	7392
13	14	Tofu	72168
14	15	Genen Shouyu	21514
15	16	Pavlova	8131,5

Explications


Cette requête donne le chiffre d'affaire par produit. On sélectionne simplement la somme des prix de ventes en groupant sur les prix produits. Bien sûr comme sur toutes les requêtes écrites jusqu'à présent, on fait les jointures nécessaires afin d'avoir les informations comme le nom etc ....

## Question 2

Script SQL

```
SELECT p.categorie,  
        t.mois,  
        SUM(v.prix_vente) AS CA  
FROM   vente_vm v  
        join produit_vm p  
        ON v.id_produit = p.id  
        join temps_vm t  
        ON v.id_temps = t.id  
GROUP BY rollup ( p.categorie, t.mois );
```

Résultat

 SQL | Toutes les lignes extraites : 109 en 0,008 secondes

CATEGORIE	MOIS	CA
91 Pâtes e...	9	32765
92 Pâtes e...	10	67096
93 Pâtes e...	11	38879
94 Pâtes e...	12	80814
95 Pâtes e...	(null)	715287
96 Poisson...	1	43459
97 Poisson...	2	18183
98 Poisson...	3	18937
99 Poisson...	4	19116
100 Poisson...	5	57675
101 Poisson...	6	18632
102 Poisson...	7	23740
103 Poisson...	8	30462
104 Poisson...	9	45696
105 Poisson...	10	32080
106 Poisson...	11	21541
107 Poisson...	12	36554
108 Poisson...	(null)	366075
109 (null)	(null)	4043085,5

## Explications

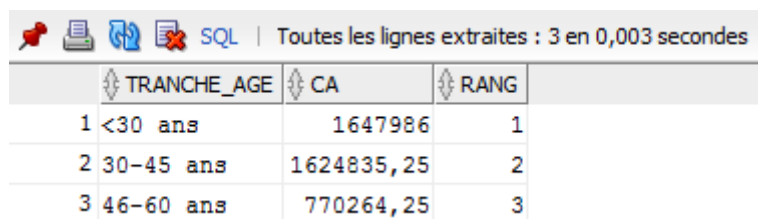
Cette requête donne le chiffre d'affaire par catégorie et mois, par catégorie et toute catégorie et mois confondus. Ceci est fait grâce à un rollup, comme on a maintenant l'habitude de faire.

### Question 3

Script SQL

```
SELECT c.tranche_age,  
        SUM(v.prix_vente) AS CA,  
        Rank()  
        over (  
            ORDER BY SUM(v.prix_vente) DESC) AS RANG  
FROM   vente_vm v  
        join client_vm c  
        ON v.id_client = c.id  
GROUP BY c.tranche_age;
```

Résultat



	TRANCHE_AGE	CA	RANG
1	<30 ans	1647986	1
2	30-45 ans	1624835,25	2
3	46-60 ans	770264,25	3

### Explications

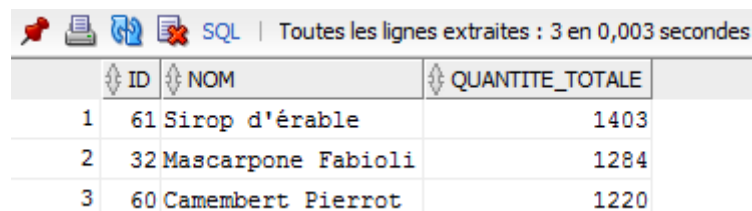
Cette requête donne le chiffre d'affaire par tranche d'âge, en donnant le rang de chaque tranche (1 pour celle qui a le plus grand chiffre d'affaire). Ici on utilise la fonction rank() qui va classer les chiffre d'affaire dans l'ordre décroissant. On affiche aussi le chiffre d'affaire grâce à la fonction SUM () et la clause GROUP BY.

## Question 4

Script SQL

```
SELECT id,
       nom,
       QUANTITE_TOTALE
FROM   (SELECT p.id,
              p.nom,
              SUM(v.quantite) AS QUANTITE_TOTALE,
              Rank()
                over (
                  ORDER BY SUM(v.quantite) DESC) AS RANG
FROM   vente_vm v
       join produit_vm p
       ON v.id_produit = p.id
GROUP BY p.id,
         p.nom)
WHERE  rang BETWEEN 1 AND 3;
```

Résultat



ID	NOM	QUANTITE_TOTALE
1	61 Sirop d'érable	1403
2	32 Mascarpone Fabioli	1284
3	60 Camembert Pierrot	1220

## Explications

Cette requête donne les 3 produits les plus vendus en quantité. Pour ce faire, on fait une sous requête ou l'on utilise la fonction rank () en classant les quantités vendues et en groupant ces quantités par produit. Ensuite la requête principale sélectionne les 3 premiers produits en filtrant sur les rangs entre 1 et 3.