

# Requêtes OLAP

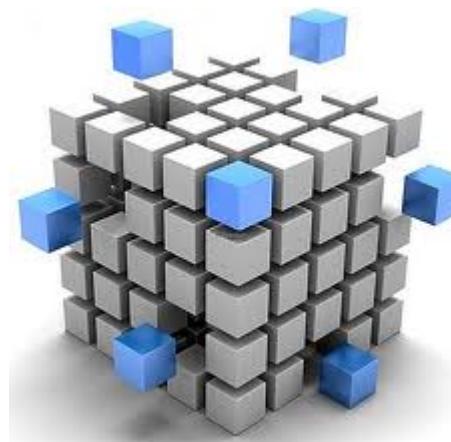
# Rappels

Au cours précédent, on a vu

- La distinction OLTP / OLAP
- Le modèle multidimensionnel, et son implémentation en relationnel (schéma en étoile/flocon)
- La représentation des faits et des dimensions à l'aide d'un cube

# Rappels

- Une mesure = une valeur selon n dimensions, d'où la notion de (*hyper*)**cube** de données.
- Un fait est un cube atomique à l'intérieur d'un plus grand cube.
- Requête OLAP : manipulation de ce cube.



# Requêtes SIO/SID

- Dans un SIO :
  - Requêtes transactionnelles (cf cours SGBD)
  - Insert, update, delete, select
  - Requêtes portant sur peu de tuples
- Dans un SID :
  - Requêtes analytiques
  - insert (par lots) et select
  - Manipulation du cube et des dimensions avec leurs hiérarchies : drill down, slice, dice,...

# Exemples de requêtes OLAP

- Navigation dans un cube
  - « Nombre de livres pour enfants vendus en janvier, indépendamment du lieu »
- Navigation à travers les hiérarchies
  - « Montant des ventes par famille de produits, par trimestre et région »
- Classement
  - « Les 10 livres les mieux vendus en 2013, par région »

# Manipulation du cube

Une requête OLAP permet de naviguer dans le modèle multi dimensionnel :

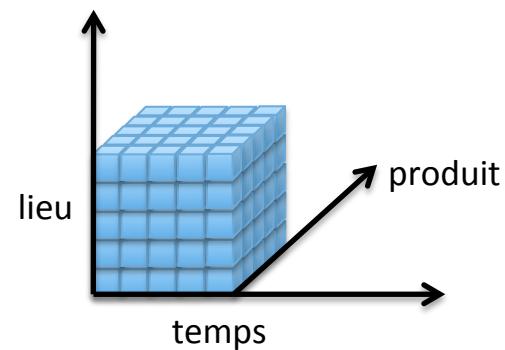
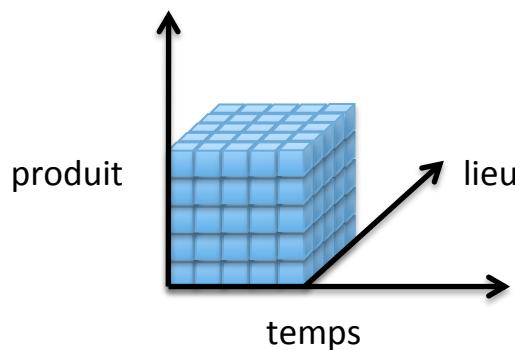
- **Rotate** : Choisir le pivot d'analyse en faisant tourner le cube
- Application d'un critère de sélection :
  - **Slice** : sélection sur une dimension
  - **Dice** : sélection sur plusieurs dimensions
- Navigation dans les hiérarchies :
  - **Roll up** : naviguer vers une granularité plus grossière
  - **Drill down** : vers une granularité plus fine (zoom)
- Modification du nombre de dimensions :
  - **Drill out/split** : ajouter des dimensions
  - **Drill in/merge** : enlever des dimensions
- *Nous ne verrons ici que des opération portant sur 1 cube, il existe aussi des opérations permettant de faire le lien entre plusieurs cubes.*

# Rotation

Change l'orientation dimensionnelle de l'analyse. Par exemple :

	2009	2010	2011
Oeuf	221	263	139
Viande	275	257	116

	2009	2010	2011
IDF	101	120	52
Nord	395	400	203

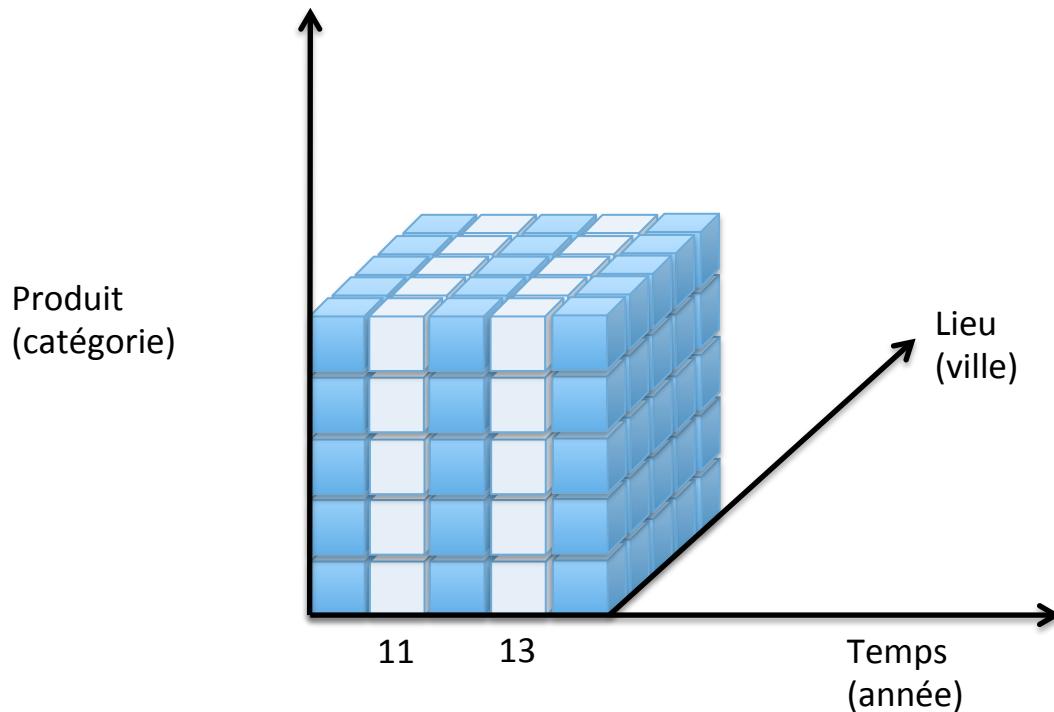


# Slice

- Permet d'extraire des informations représentées par un cube, en fonction des termes définis pour une dimension.
- Le paramètre de l'opérateur **slice** est un critère de sélection portant sur une dimension.
- Exemples :
  - (mois > 2) and (mois <= 11) and (mois <> 4)
  - Pays = ‘DE’ and code\_postal = 10179 (ici, dimension géographique avec plusieurs niveaux)
- Notation : slice(F,D) sélection des « tranches » qui satisfont la formule F sur la dimension D.

# Slice - Exemple

- Slice(année = 2011 ou année = 2013, Temps)

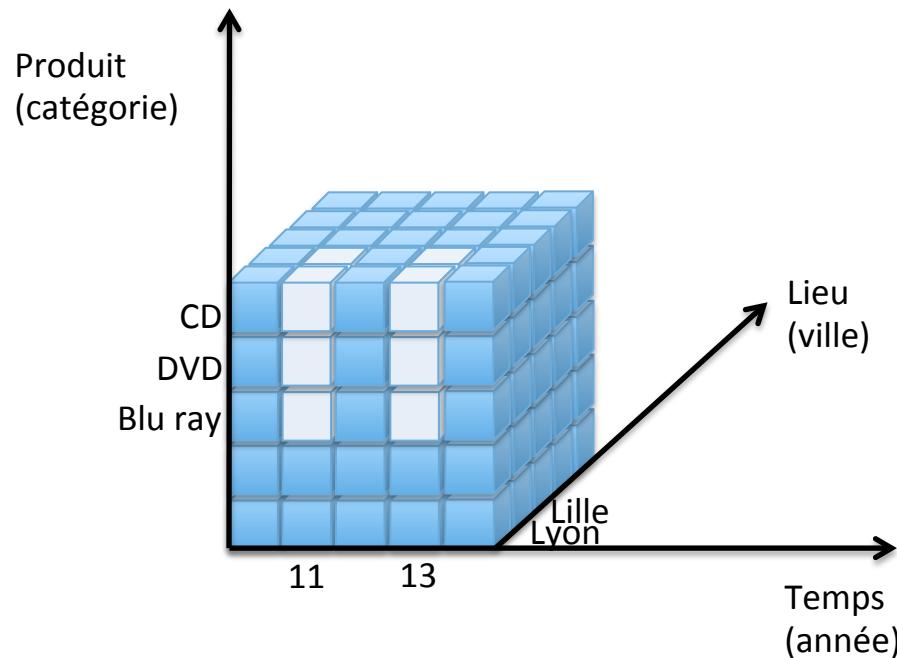


# Dice

- Application d'une sélection sur plusieurs dimensions, donc peut-être vu comme la combinaison de plusieurs slices.
- Slice produit des « tranches » du cube initial,  
Dice produit des « sous-cubes »

# Dice - Exemple

`Slice(année=2011 or année = 2013, Temps)`  
`and slice(famille = 'médias',Produit)`  
`and slice(ville='Lille' or ville = 'Lyon', Lieu)`

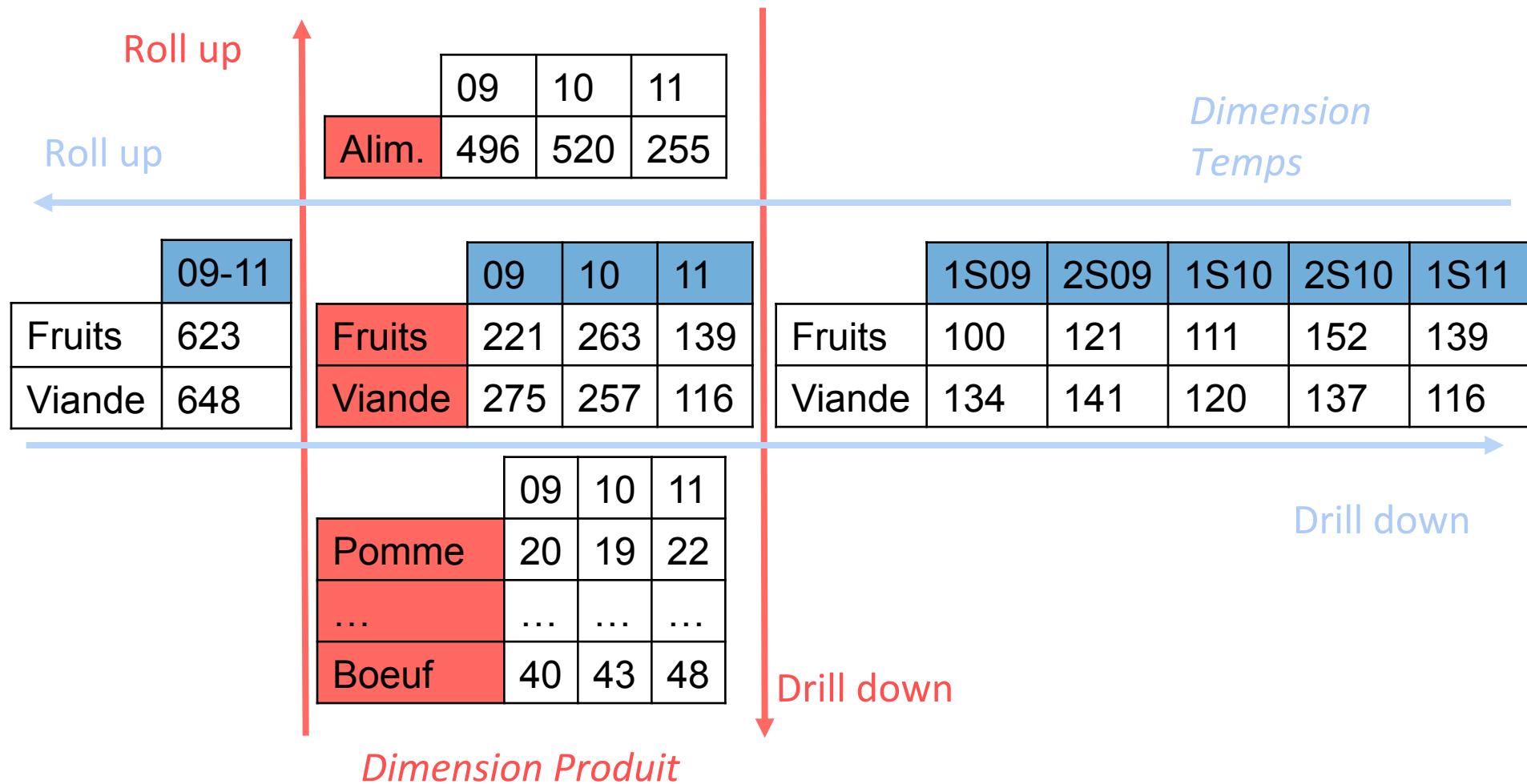


# Roll up, Drill down

Opération modifiant la granularité :

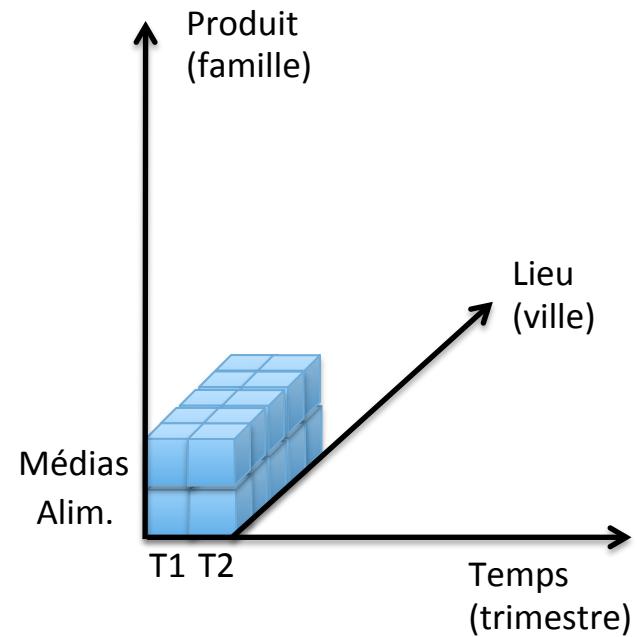
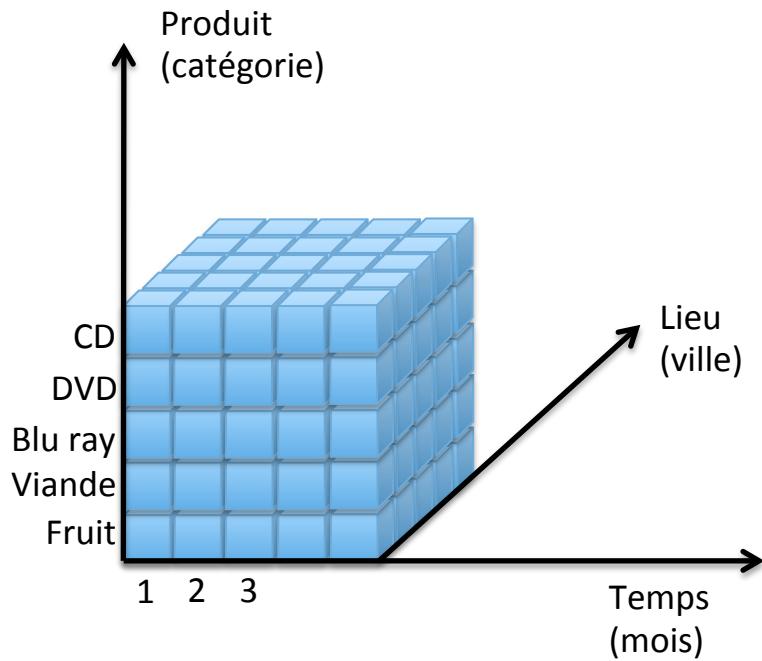
- Forage vers le haut, **roll up** : comme on va vers une granularité plus grossière, il faut utiliser des fonctions d'agrégation pour calculer les nouvelles mesures (attention aux mesures non additives ou semi additives)
- Forage vers le bas, **drill down** : on obtient des données plus détaillées, jusqu'à la granularité de l'entrepôt.

# Roll up, Drill down



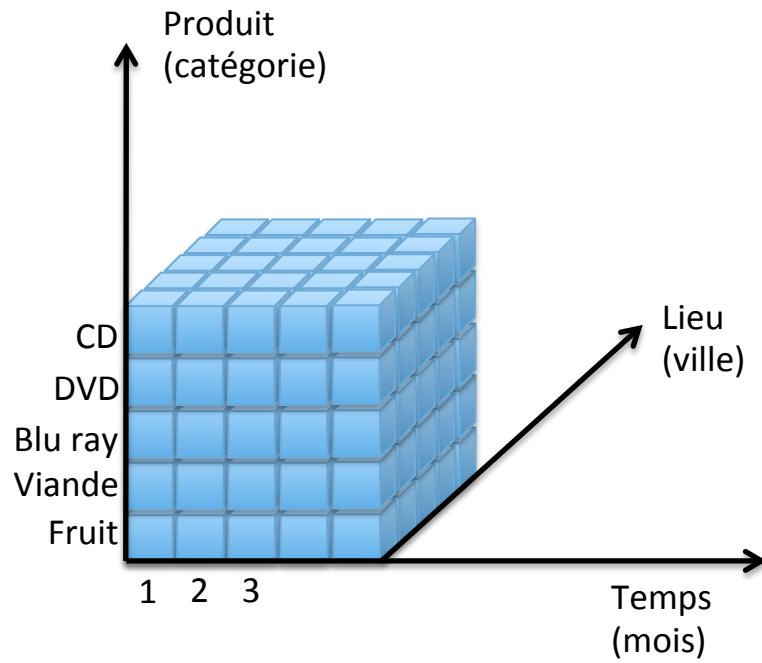
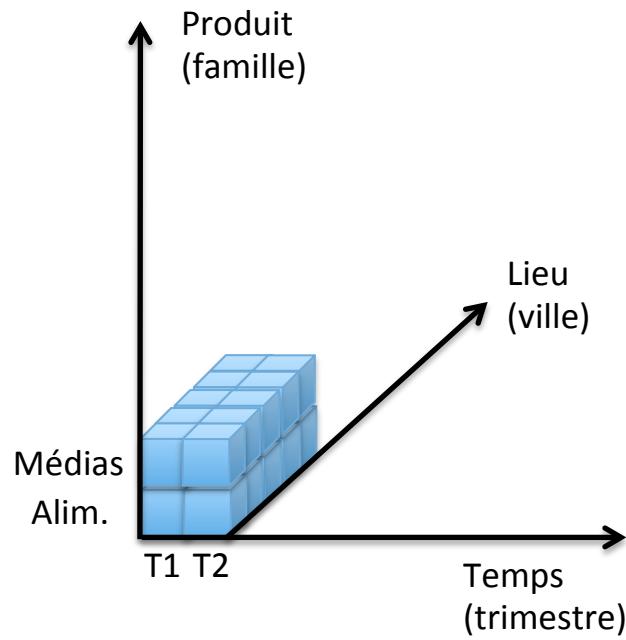
# Roll up – Exemple

Roll up de (catégorie/mois/ville) vers (famille/trimestre/ville)



# Drill down - Exemple

Drill down de (famille/trimestre/ville) vers (catégorie/mois/ville)

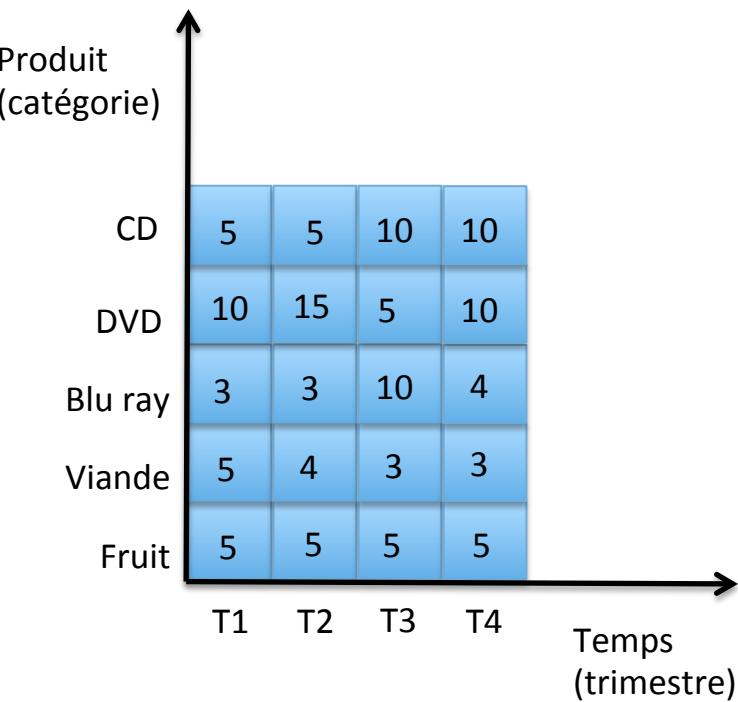
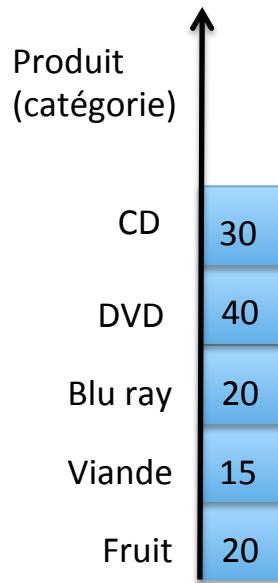


# Split / Merge

- Split :
  - un cube à n dimensions est complété par k dimensions.
  - On obtient des données plus détaillées, pas à cause du changement de granularité mais parce qu'on détaille en fonction de nouvelles dimensions
- Merge :
  - On supprime des dimensions
  - On obtient des données moins détaillées, parce qu'il y a moins de dimensions (mais pas de changement de granularité)

# Split - exemple

- Split de la dimension Temps (passage de 1 à 2 dimensions)



# Merge - exemple

- Merge de la dimension Temps (passage de 2 à 1 dimensions)

	Produit (catégorie)			
	T1	T2	T3	T4
CD	5	5	10	10
DVD	10	15	5	10
Blu ray	3	3	10	4
Viande	5	4	3	3
Fruit	5	5	5	5

Temps (trimestre)

	Produit (catégorie)
CD	30
DVD	40
Blu ray	20
Viande	15
Fruit	20

# Implémentations

- Il existe principalement 2 approches
  - Extension du langage SQL (SQL-OLAP), pour un cube stocké dans un SGBD relationnel, sous la forme de schéma en étoile ou en flocon. On parle de modèle **ROLAP** ou Relational - OLAP
  - Expression multidimensionnelle (ex : MDX de Microsoft) s'appliquant directement à un SGBD multidimensionnel. On parle de modèle **MOLAP** ou Multidimensional – OLAP

*Il existe aussi un modèle hybride entre ces 2 solutions, appelé **HOLAP** pour Hybrid - OLAP*

# SQL-OLAP

- Extension de SQL 3 (SQL'99).
- On utilise les opérateurs SQL classiques :
  - Jointures (tables de faits / dimensions), sélections, projections, group by, ...
- SQL est enrichi avec de nouveaux opérateurs :
  - Clause Group by, avec Cube, Rollup et Grouping sets
  - Fonction grouping
  - Expressions analytiques :
    - Fenêtre glissante
    - Fonctions de classement Rank, NTile

# Group By

- Group by  $att_1, \dots, att_n$  permet de partitionner les données en mettant dans le même groupe les tuples qui ont la même valeur pour  $(att_1, \dots, att_n)$
- Dans le select, peuvent figurer  $att_1, \dots, att_n$  et des fonctions de groupe (count, sum, avg, min, max) portant sur n'importe quel attribut.
- Clause Having pour sélectionner les groupes
- Exemple : (on suppose une granularité jour/produit)

```
Select mois, annee, sum(CA)
From Ventes V
Join Temps T on V.TID = T.TID
Join Produit P on V.PID = P.PID
Where P.famille = 'Médias' --> slice dimension Produit
Group by mois, annee --> rollup sur la dimension temps
```

# Exemple

```
SELECT fact1_id,  
fact2_id,  
COUNT(*) AS num,  
SUM(sales_value) AS  
sales_val  
FROM fact_tab  
GROUP BY  
fact1_id,fact2_id  
ORDER BY  
fact1_id,fact2_id;
```

Fact1_id	Fact2_id	Num	Sales_val
1	1	83	4363.55
1	2	96	4794.76
1	3	93	4718.25
1	4	105	5387.45
1	5	101	5027.34
2	1	109	5652.84
2	2	96	4583.02
2	3	110	5555.77
2	4	113	5936.67
2	5	94	4508.74

# Group by rollup

- Calcule les fonctions (sum, count, ...) à tous les niveaux d'une hiérarchie d'une dimension, ainsi que le résultat final.
  - Group by rollup ( $att_1, att_2, att_3$ ) définit des groupements par ( $att_1, att_2, att_3$ ), par ( $att_1, att_2$ ), par ( $att_1$ ) et par ().
  - S'il y a n attributs dans le group by rollup, on aura donc  $n+1$  niveaux d'agrégation.

# Exemple

```
SELECT fact1_id,  
fact2_id,  
SUM(sales_value) AS  
sales_val  
FROM fact_tab  
GROUP BY ROLLUP  
(fact1_id,fact2_id)  
ORDER BY  
fact1_id,fact2_id;
```

Fact1_id	Fact2_id	Sales_val
1	1	4363.55
1	2	4794.76
1	3	4718.25
1	4	5387.45
1	5	5027.34
<b>1</b>		<b>24291.35</b>
2	1	5652.84
2	2	4583.02
2	3	5555.77
2	4	5936.67
2	5	4508.74
<b>2</b>		<b>26237.04</b>
		<b>50528.39</b>

# Group by cube

- Calcule les fonctions (sum, count, ...) pour toutes les combinaisons possibles de groupement
  - Group by cube ( $\text{att}_1, \text{att}_2, \text{att}_3$ ) définit des groupements par  $(\text{att}_1, \text{att}_2, \text{att}_3)$ ,  $(\text{att}_1, \text{att}_2)$ ,  $(\text{att}_2, \text{att}_3)$ ,  $(\text{att}_1, \text{att}_3)$ ,  $(\text{att}_1)$ ,  $(\text{att}_2)$ ,  $(\text{att}_3)$  et  $()$ .
  - S'il y a  $n$  attributs dans le group by cube, on aura donc  $2^n$  niveaux d'agrégation.

# Exemple

```
SELECT fact1_id,  
fact2_id,  
SUM(sales_value) AS  
sales_val  
FROM fact_tab  
GROUP BY CUBE  
(fact1_id, fact2_id)  
ORDER BY fact1_id,  
fact2_id;
```

Fact1_id	Fact2_id	Sales_val
1	1	4363.55
1	2	4794.76
1	3	4718.25
1	4	5387.45
1	5	5027.34
<b>1</b>		<b>24291.35</b>
2	1	5652.84
2	2	4583.02
2	3	5555.77
2	4	5936.67
2	5	4508.74
<b>2</b>		<b>26237.04</b>
	1	<b>10016.39</b>
	2	<b>9377.78</b>
	3	<b>10274.02</b>
	4	<b>11324.12</b>
	5	<b>9536.08</b>
		<b>50528.39</b>

# Group by grouping sets

- Permet des groupements multiples
- Exemple : grouper par (mois, region) et (mois, produit)
- Le groupe () représente la relation entière
- Le rollup et le cube peuvent être écrits avec un grouping sets.

# Exemple

```
SELECT fact1_id,  
fact2_id, fact3_id,  
SUM(sales_value) AS  
sales_val,  
FROM fact_tab  
GROUP BY GROUPING SETS  
( (fact1_id, fact2_id),  
(fact1_id, fact3_id) )  
ORDER BY fact1_id,  
fact2_id, fact3_id;
```

Fact1_id	Fact2_id	Fact3_id	Sales_val
1	1		4363.55
...	...		...
1	5		5027.34
1		1	2737.4
...		...	...
1		10	2334.06
2	1		5652.84
...	...		...
2	5		4508.74
2		1	3512.69
...		...	...
2		10	2027.16

# Groupements

- On peut utiliser des ensembles d'attributs dans les expressions de groupe :
  - GROUP BY ROLLUP ((a, b), c)  
On groupe selon (a, b, c) (a, b) () pas(a)
  - GROUP BY CUBE ((a, b), c)  
On groupe selon (a, b, c) (a, b) (c) () pas (a, c) (a) (b, c) (b)
- On peut concaténer des groupements en les séparant par des “,” ce qui revient à grouper selon le produit cartésien des différents groupements :
  - GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)  
On groupe selon (a, c) (a, d) (b, c) (b, d)
  - GROUP BY A, GROUPING SETS ((B), (C), ())  
Équivalent à GROUPING SETS ((A,B), (A,C), (A))

# Fonction grouping

- On veut déterminer pour 1 attribut de groupement si une ligne est un total
- **GROUPING(att)** renvoie 1 si on a une valeur « null » due à un groupement pour cet attribut (donc un sous-total)
- Utilisée dans une clause HAVING, cette fonction permet donc de choisir le type de groupement

# Exemple

```
SELECT fact1_id, fact2_id,  
SUM(sales_value) AS  
sales_val,  
GROUPING(fact1_id) AS f1g,  
GROUPING(fact2_id) AS f2g  
FROM fact_tab  
GROUP BY CUBE (fact1_id,  
fact2_id)  
HAVING  
GROUPING(fact1_id) = 1 OR  
GROUPING(fact2_id) = 1  
ORDER BY GROUPING(fact1_id),  
GROUPING(fact2_id);
```

Fact1_id	Fact2_id	Sales_val	f1g	f2g
1		24291.35	0	1
2		26237.04	0	1
	4	11324.12	1	0
	3	10274.02	1	0
	2	9377.78	1	0
	1	10016.39	1	0
	5	9536.08	1	0
		50528.39	1	1

Grouping(fact1\_id) renvoie 1 quand fact1\_id vaut null suite au groupement

# Fonction grouping\_id (Oracle)

- **GROUPING\_ID** renvoie un nombre qui permet d'identifier le type de groupement de chaque ligne.
- GROUPING porte sur une seule colonne, pour tester les différents critères d'agrégation, il faut autant de colonnes de GROUPING que de colonnes de regroupement
- La fonction GROUPING\_ID synthétise en une seule fonction l'état d'agrégation sur les ensembles de critères

Par exemple :

CUBE ( a , b )

Niveau de groupement	Bit vector (grouping)	Grouping_id
a,b	0 0	0
a	0 1	1
b	1 0	2
total	1 1	3

# Expressions analytiques

- Syntaxe Oracle :

```
fonction (expression)
over ([clause de partitionnement] [clause d'ordre [clause de fenêtrage]])
```

- La **clause de partitionnement** : de la forme **PARTITION BY ...**

- Elle définit un découpage des données suivant les valeurs des colonnes du PARTITION BY.
  - Chaque valeur définit un groupe logique à l'intérieur duquel est appliquée la fonction analytique.

- La **clause d'ordre** : de la forme **ORDER BY ...**

- Elle indique comment les données sont triées à l'intérieur de chaque partition.

- La **clause de fenêtrage** indique l'ensemble des lignes sur lequel doit être appliquée la fonction.

- Cette clause définit une fenêtre glissante, i.e. une suite de lignes autour de (avant/après) la ligne courante.
  - Si une clause de fenêtrage est spécifiée, une clause d'ordre doit obligatoirement l'être aussi.
  - *Lire la documentation pour la syntaxe ...*

# Exemple

```
table COTATION (
    isin varchar2 (20), -- identifiant de la société
    dte date, -- date de cotation
    ouv number, -- premier cours de la journée
    max number, -- cours le plus haut de la journée
    min number, -- cours le plus bas de la journée
    clo number, -- dernier cours de la journée (clôture)
    volume number, -- nb d'actions échangées
)
-- (isin, dte) clé primaire

select isin, dte,
    avg(clo) over(partition by isin order by dte rows 4 preceding)
    mma_5
from COTATION
--> Moyenne des cours de clôture sur les 5 derniers jours, par
société
```

# RANK, DENSE\_RANK

- Fonction analytique, donc à utiliser dans une expression analytique
- Rank() donne le rang d'un enregistrement à l'intérieur de son groupe (cf clause de partition) en fonction de l'ordre défini (cf clause d'ordre)
- avec la fonction rank(), s'il y a 2 tuples qui ont les mêmes valeurs et sont classés de rang n, le tuple suivant est de rang n+2. Avec dense\_rank() le tuple suivant a le rang n+1.

# Exemple

Table POPULATION(country, number)

```
SELECT country,  
rank() OVER (ORDER BY number DESC) AS n_rank  
FROM population
```

Donne chaque pays avec son rang (rang 1 pour le plus peuplé).

Le résultat n'est pas trié (utiliser une clause Order by dans le select).

# NTILE

- La fonction NTILE ordonne les lignes en N paquets de même taille (à 1 près), N étant un entier passé en paramètre.
  - La clause order by est donc obligatoire.
- NTILE renvoie pour chaque ligne le numéro du paquet auquel cette ligne appartient.

# Exemple

```
SELECT last_name, salary,  
NTILE(4) OVER (ORDER BY salary DESC) AS quartile  
FROM employees  
WHERE department_id = 100;
```

Comme il y a 6 lignes, et 6 n'est pas divisible par 4, les paquets 3 et 4 ont 1 ligne de moins que les paquets 1 et 2.

Last_name	salary	quartile
Greenberg	12000	1
Faviet	9000	1
Chen	8200	2
Urman	7800	2
Sciarra	7700	3
Popp	6900	4