Systèmes d'information décisionnels (Data Warehouse)

Dr Dendani-Hadiby Nadjette
Université Badji Mokhtar Annaba
Département d'Informatique
n_dendani@yahoo.fr

2021/2022

SQL- OLAP

Evolution de SQL pour l'OLAP

- Evolution de la norme SQL : OLAP SQL 3 / SQL 99 extensions
- Nouvelles fonction SQL d'agrégation : Rank, N_tile, N_tile , rank, dense_rank, every, any, some…
- Nouvelles fonctions de la clause GROUP BY : CUBE, ROLLUP, GROUPING SETS
- Fonction GROUPING
- Fenêtre glissante

  WINDOWS/OVER/PARTITION, ...

Evolution de SQL pour l'OLAP

Fonctions classiques d'agrégation de SQL : Count, Max, Min, Sum, Avg

Ajout de nouvelles fonctions pour faire des calculs comme c'est possible dans les tableurs :

- N_tile(expression, n):
  - On calcule le domaine de l'expression en fonction des valeurs qui apparaissent dans la colonne.
  - Ce domaine est alors divisé en n intervalles de tailles approximativement égales.
  - La fonction retourne alors le numéro de l'intervalle qui contient la valeur de l'expression.
- *Ex1 : si le solde du compte est parmi les 10% les plus élevés*, N_tile(compte.solde, 10) retourne 10.
- *Ex2 : Trouver le min, le max et la moyenne des températures parmi les 10% les plus élevées.*

```
SELECT Percentile, MIN(Temp), AVG(Temp), MAX(Temp)
FROM Weather
GROUP BY N_tile(Temp,10) as Percentile
HAVING Percentile = 10;
```

NOUVELLES FONCTIONS SQL DE CLASSEMENT

rank(expression): rang de l'expression dans l'ensemble de toutes les valeurs du domaine.

Par exemple, s'il y a N valeurs dans une colonne, et si l'expression est la plus grande alors N, si c'est la plus petite alors 1

Soit la table : population(country, number)

Ex. : Trouver le classement de chaque pays :

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

La clause ORDER BY est nécessaire pour retourner les résultats triés :

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

Une utilisation de **rank** : trouver les n-premiers (syntaxe DB2) :

Ex. : Trouver les 5 premiers pays les plus peuplés :

```
SELECT country, rank() OVER (ORDER BY number DESC) AS n_rank
FROM population
ORDER BY n_rank
FETCH FIRST 5 ROWS ONLY
```

AUTRES NOUVELE FONCTION SQL

dense_rank(expression): 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 rand n+1.

Ex. : Trouver le classement de chaque pays :

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

Every(expression) : vrai ssi la valeur de son argument (expression) est vraie pour tous les tuples, sinon faux (∀)

Soit la table : Film_v(titre, type_film, année, magasin, qté_stock, qté_vendus)

Ex. : Trouver la quantité maxi et mini de films vendus à plus de 10 exemplaires :

```
SELECT COUNT(*), MAX(qté-vendus), MIN(qté-vendus),
SUM(qté-vendus), AVG(qté-vendus)
FROM Film_v
GROUP BY magasin Having EVERY(qté-vendus>=10)
```

Any(expression) ou Some(expression) : vrai ssi la valeur de son argument (expression) est vraie pour au moins un tuple, sinon faux (∃)

NOUVELLES FONCTIONS DE LA CLAUSE GROUP BY

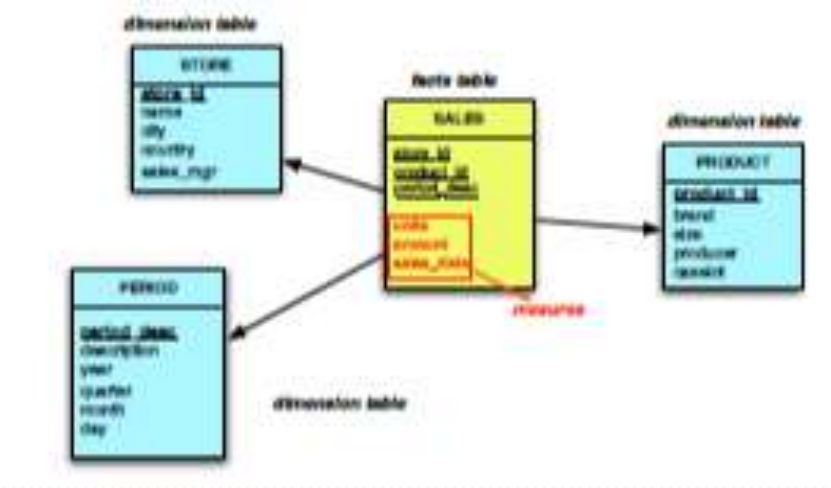
CUBE, ROLLUP et GROUPING SETS produisent un ensemble de tuples, équivalent à

un UNION ALL de tuples groupés différemment :

- ROLLUP calcule des agrégats (SUM, COUNT, MAX, MIN, AVG) à différents niveaux d'agrégation
- CUBE est similaire à ROLLUP mais permet de calculer toutes les combinaisons d'agrégations
- GROUPING SETS permet d'éviter le calcul du cube, quand il n'est pas globalement nécessaire
- Les fonctions GROUPING précise le groupe d'appartenance de chaque tuple pour calculer les sous-totaux et les filtres

UN DATAWAREHOUSE

- Schéma en étoile:



Création du cube « Sales » :

```
CREATE VIEW Sales AS
    (SELECT ds.*, YEAR(sales_date) AS year, MONTH(sales_date) AS
                month, DAY(sales_date) AS day
FROM (Sales NATURAL JOIN Store NATURAL JOIN Product
          NATURAL JOIN Period) ds
```

GROUP BY ROLLUP (1)

- calcule des agrégats (SUM, COUNT, MAX, MIN, AVG) à tous les niveaux de totalisation sur une hiérarchie de dimensions
- et calcule le total général :
  - Selon l'ordre de gauche à droite dans la clause GROUP BY
  - S'il y a n colonnes de regroupements, GROUP BY ROLLUP génère n+1 niveaux de totalisation

Exemples :

GROUP BY ROLLUP (year, month, day)

GROUP BY ROLLUP (country, city)

GROUP BY ROLLUP (month, city, producer)

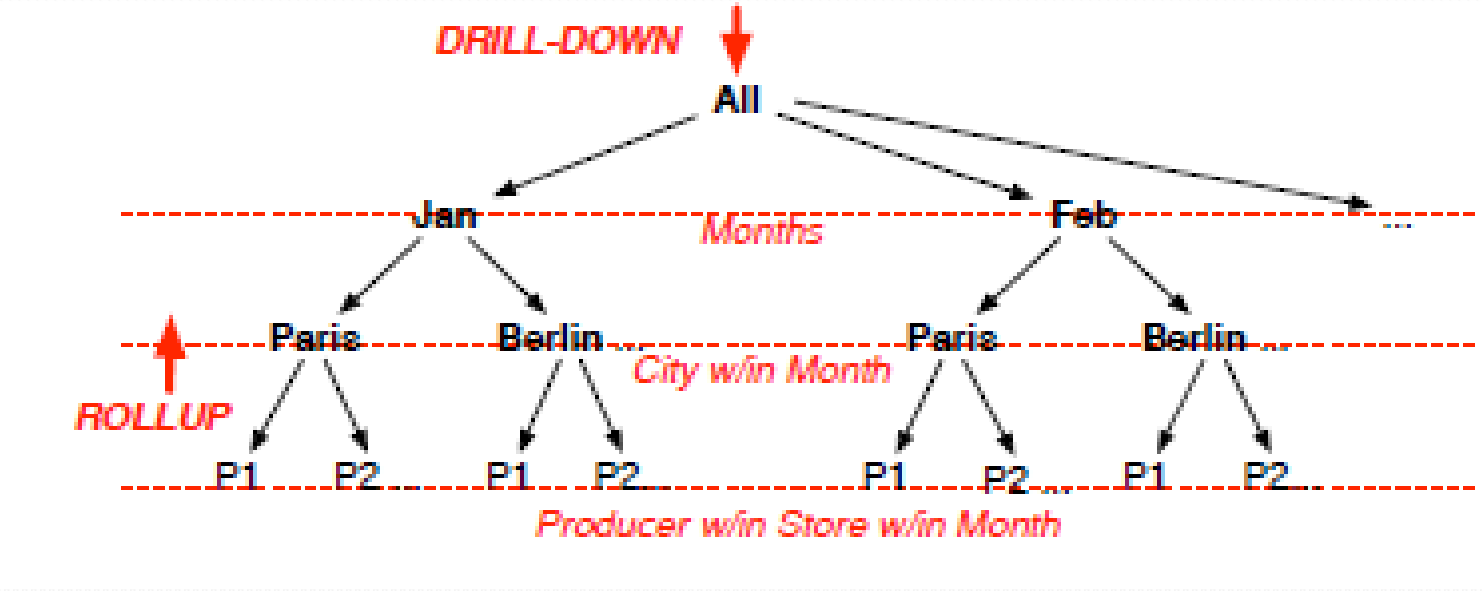
…

Simplifie et accélère la maintenance des tables de synthèse.

GROUP BY ROLLUP (2)

Ex (1) :

```
SELECT month, city, producer, SUM(units) AS sum_units
FROM Sales
WHERE year = 2014
GROUP BY ROLLUP (month, city, producer) ;
```



GROUP BY ROLLUP (3)

Ex 1. : Total des ventes par pays et responsable des ventes pour chaque mois de 2014, avec sous-totaux pour chaque mois, et grand total :

```
SELECT month, country, sales_mgr, SUM(amount)
FROM Sales
WHERE year = 2014
GROUP BY ROLLUP(month, country, sales_mgr)
```

Month	Country	Sales-mgr	SUM(amoun	
April	France	Martin	25000	
April	France	Smith	15000	
April	France	-	40000	*Total France/April*
April	Germany	Smith	15000	
April	Germany	-	15000	*Total Germany/April*
April	-	-	55000	*Tot. April*
May	France	Martin	25000	
May	France	-	25000	*Total France/May*
May	Germany	Smith	15000	
May	Germany	-	15000	*Total France/May*
May	-	-	40000	*Total May*
-	-	-	95000	*Grand Total*

GROUPE BY CUBE (1)

GROUP BY CUBE : calcule des agrégats (SUM, COUNT, MAX, MIN, AVG) à différents niveaux d'agrégation comme ROLLUP mais de plus permet de calculer toutes les combinaisons d'agrégations :

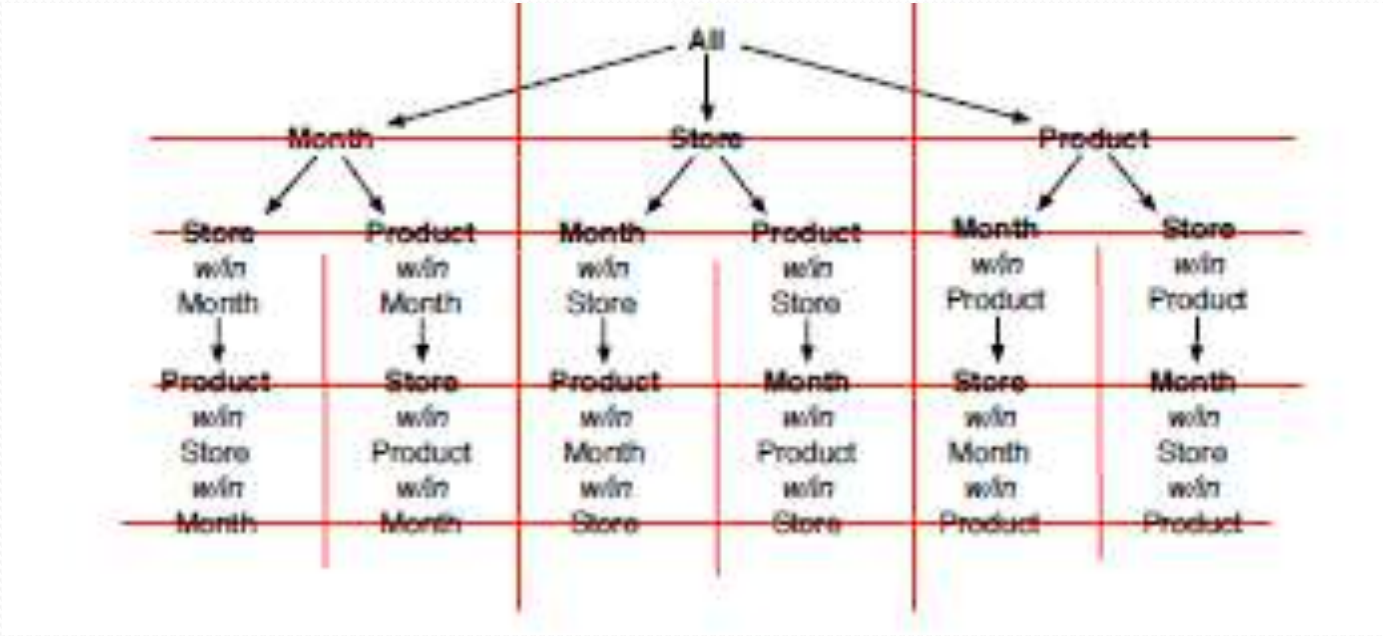
- GROUPE BY CUBE créé des sous-totaux pour toutes les combinaisons possibles d'un ensemble de colonnes de regroupement
- Si la clause CUBE contient n colonnes, CUBE calcule 2^n combinaisons de totaux
- Intéressant pour des colonnes représentant des dimensions appartenant à des hiérarchies différentes

GROUP BY CUBE est une alternative plus performante au UNION ALL

GROUPE BY CUBE (2)

Ex 1. : Tous les totaux et sous totaux des quantités de produits vendus par ville et produit pour chaque mois de 2014 :

```
SELECT month, city, product_id, SUM(units)
FROM Sales
WHERE year = 2014
GROUP BY CUBE (month, city, product.id
```



GROUPE BY CUBE (3)

Ex.2 : Tous les totaux et sous totaux des ventes par pays et responsable des ventes pour chaque mois de 2014

```
SELECT month, country, sales_mgr, SUM(amount)
FROM Sales
WHERE year = 2014
GROUP BY CUBE(month, country, sales_mgr)
```

Month	Country	Sales-mgr	SUM(amount)	
April	France	Martin	25000	
April	France	Smith	15000	
April	France	-	40000	Total France/April
April	Germany	Smith	15000	
April	Germany	-	15000	Total Germany/April
April	-	Martin	25000	Total Martin/April
April	-	Smith	30000	Total Smith/April
April	-	-	55000	Total April
May	France	Martin	25000	
May	France	-	25000	Total France/May
May	Germany	Smith	15000	
May	Germany	-	15000	Total Germany/May
May	-	Martin	25000	Total Martin/May
May	-	Smith	15000	Total Smith/May
May	-	-	40000	Total May
-	France	Martin	50000	Total Martin/France
-	France	Smith	15000	Total Smith/France
-	France	-	65000	Total France
-	Germany	Smith	30000	Total Smith/Germany
-	Germany	-	30000	Total Germany
-	-	Martin	50000	Total Martin
-	-	Smith	45000	Total Smith
-	-	-	95000	Grand Total

GROUP BY GROUPING SETS (1)

GROUP BY GROUPING SETS :

- Utilisé avec les agrégations usuelles (MAX, MIN, SUM, AVG, COUNT, ...),
- permet des groupements multiples (month, country) et (month, sales_mgr) ; les résultats peuvent être restreints par une clause HAVING

Ex. : Total des ventes pour chaque mois de l'année 2014, par pays et par responsable des ventes :

```
SELECT month, country, sales_mgr, SUM(amount)
FROM Sales
WHERE year = 2014
GROUP BY GROUPING SETS((month, country),(month, sales_mgr))
```

Month	Country	Sales-mgr	SUM(amount)	
April	France	-	40000	Total France/April
April	Germany	-	15000	Total Germany/April
April	-	Martin	25000	Total Martin/April
April	-	Smith	30000	Total Smith/April
May	France	-	25000	Total France/May
May	Germany	-	15000	Total Germany/May
May	-	Martin	25000	Total Martin/May
May	-	Smith	15000	Total Smith/May

GROUP BY GROUPING SETS (2)

Tuple de grand total : Une syntaxe particulière permet d'inclure un tuple « grand total » dans les résultats :

- les grands totaux » sont générés implicitement avec ROLLUP et CUBE
- la syntaxe permet aux grands totaux d'être générés sans agrégat supplémentaire

Ex. 2 : Total de ventes par mois, pays, et responsable de vente, et aussi grand total de vente :

```
SELECT month, country, sales_mgr, SUM(amount)
FROM Sales
WHERE year = 2014
GROUP BY GROUPING SETS((month, country), ( ) )
```

Month	Country	Sales-mgr	SUM(amount)
April	France	Martin	25000
April	France	Smith	15000
April	Germany	Smith	15000
May	France	Martin	25000
May	Germany	Smith	15000
-	-	-	95000

La fonction GROUPING

la fonction GROUPING : créé une nouvelle colonne avec des 0 et des 1,

permettant la détection de tuples de total qui sont générés lors de l'exécution de CUBE ou ROLLUP (1 pour les tuples de total et 0 pour les autres)

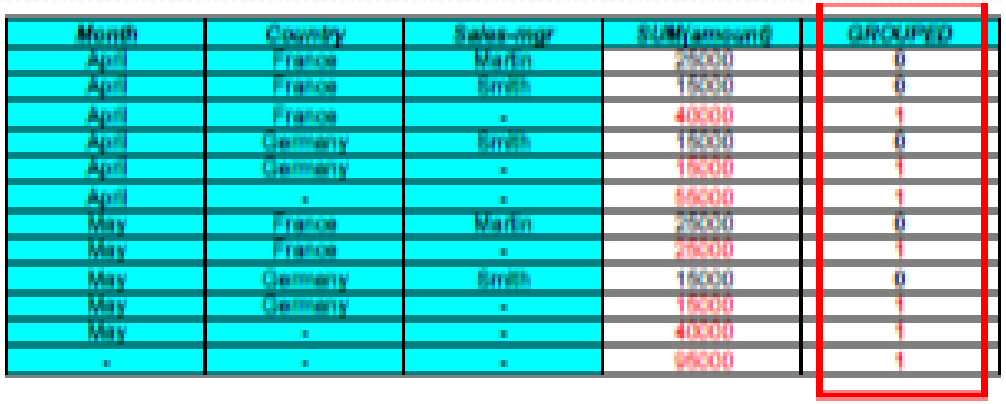
Ex. : Tous les totaux et sous totaux des ventes par pays et responsable des ventes pour chaque mois de 2014

SELECT month, country, sales_mgr, SUM(amount), GROUPING(sales_mgr)

FROM Sales

WHERE year = 2014

GROUP BY ROLLUP (month, country, sales_mgr)



Fenêtre glissante (1)

Permet de répondre à des questions comme :

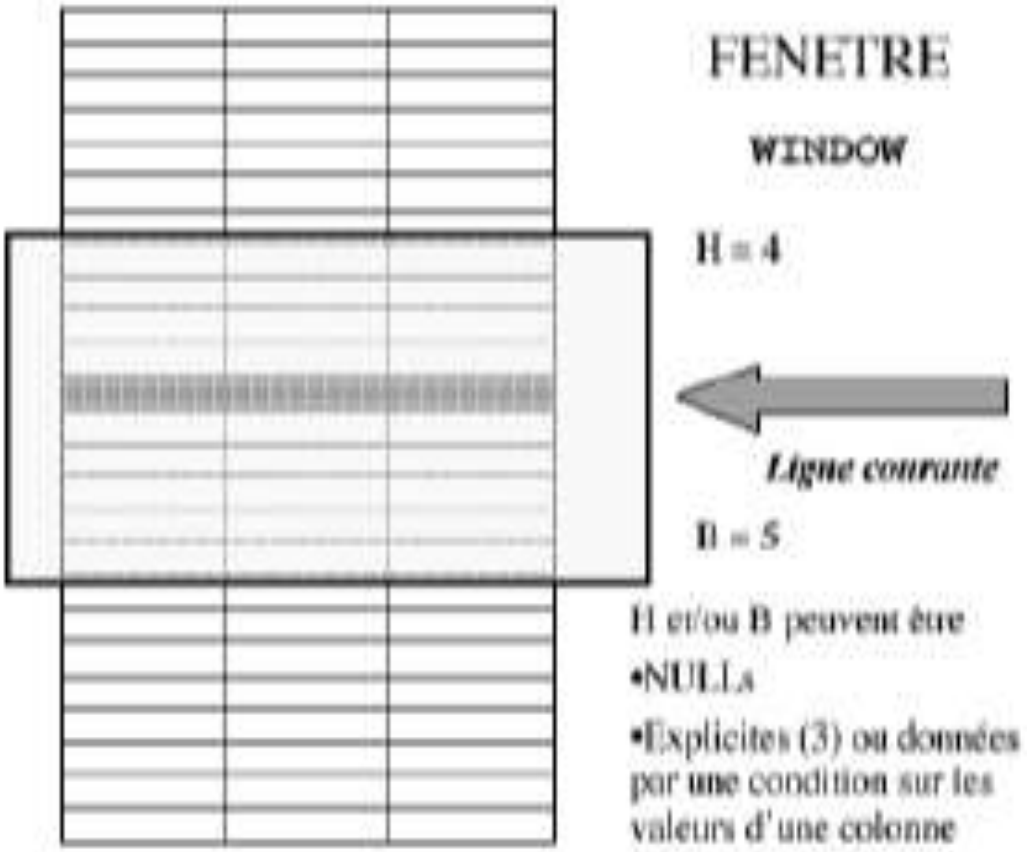
- *Trouver pour chaque mois, les ventes de chaque magasin en faisant la moyenne entre le mois courant et les deux mois précédents : faire la moyenne sur les* 3 derniers mois
- *Trouver pour chaque magasin, les ventes cumulées pour chaque mois de l'an dernier, ordonnées par mois*
- *Comparer les ventes de chaque mois avec la moyenne des ventes de chaque magasin pendant les 3 mois précédents*

=> Il faut définir une fenêtre glissante du temps et y effectuer des cumuls, moyennes, dérivations, …

Notion de FENETRE (WINDOW) :

- Dans une requête, une fenêtre est une sélection de lignes (tuples) utilisée pour un certain calcul par rapport à la ligne courante
- La taille de la fenêtre est un nombre de lignes mais peut être exprimée de différentes façons (nombre entier ou par une condition)
- Une fenêtre se déclare soit explicitement par une clause WINDOW ou directement dans le SELECT où elle est utilisée

Fenêtre glissante (2)



Fenêtre glissante (3)

3 paramètres :

- Partitionnement PARTITION : Chaque ligne de la partition a des valeurs égales sur un ensemble de colonnes (analogue au GROUP BY)
- Ordre ORDER : Ordre des lignes dans la partition comme un tri partiel sur 1 ou plusieurs colonnes (attention aux NULLs)
- Cadre FRAMING : Définit la taille de la fenêtre de manière physique (nombre de lignes ROWS) ou logique (condition, plage de valeurs RANGE) - Ex: ROWS 2 PRECEDING

Ex.1: Soit la table : population(country, year, number)

Trouver les populations de chaque pays et année, calculer la moyenne du taux de croissance de la population pour chaque pays et année, sur la base des années courante, précédente et suivante :

```
SELECT country, year, avg(population)
    OVER (ORDER BY country, year
      ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS p_avg
FROM population
ORDER BY country, year, p_avg;
```

Fenêtre glissante (4)

Ex.2 : Soit table Fvm(magasin, anmois, qté_totale) (source : Adiba & Fauvet, 2005)

Calculer pour chaque magasin et pour chaque mois du dernier tiers de 2001 le total des ventes, ainsi que la moyenne glissante des quantités vendues avec les 2 mois précédents.

```
SELECT h.magasin, h.anmois, h.qté_totale,
AVG(h.qté_totale) OVER w AS moygliss
FROM Fvm h
WHERE h.anmois BETWEEN 200109 AND 200112
WINDOW w AS (PARTITION BY h.magasin ORDER BY h.anmois ROWS 2 PRECEDING)
```

Magasin	anmois	Qté_totale	moygliss
M1	200109	46936	46936
M1	200110	22842	34889
M1	200111	30927	33568
M1	200112	51751	35173
M2	200109	20554	20554
M2	200110	9418	14986
M2	200111	13600	14524
M2	200112	24977	15998
M3	200109	23157	23157
M3	200110	26591	24874
M3	200111	31565	27104
M3	200112	46153	35436

Septembre

Moyenne sept & oct

Sept, oct, nov

Oct, nov, décembre

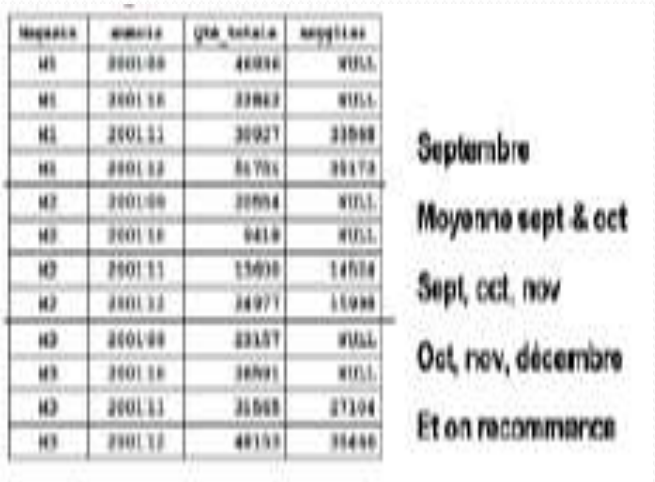
Et on recommence

Fenêtre glissante(window: groupe Incomplet(5)

Ex.2 : Soit table Fvm(magasin, anmois, qté_totale) *(source : Adiba & Fauvet, 2005)*

Calculer pour chaque magasin et pour chacun des 4 derniers mois 2001, le total des ventes ainsi que la moyenne glissante des quantités vendues sur les 2 mois précédents à condition qu'il y ait bien 2 mois, sinon NULL

```
SELECT h.magasin, h.anmois, h.qté_totale,
        CASE WHEN Count(*) OVER w < 3 THEN NULL
       ELSE AVG(h.qté_totale) OVER w End AS moygliss
 FROM Fvm h Where h.anmois BETWEEN 200109 AND 200112
  WINDOW w AS
   (PARTITION BY h.magasin ORDER BY h.anmois ROWS 2 PRECEDING)
```



Ce cour reprend les fonctions avancées du cour de Bernard ESPINASSE
« Extensions du langage SQL (SQL-3/SQL-99) pour l'OLAP »

FIN