Université Paris Nanterre

Intégration de données

Cours 6- SQL Analytique

Sonia GUEHIS
Sonia.guehis@parisnanterre.fr

Objectif

Le but des extensions SQL et fonctions analytiques est d'optimiser les performances liées aux requêtes d'analyse et de génération de rapports

Exemple GROUP BY

PC	Paris	12
DVD	Paris	18
TV	Rome	4
PC	Rome	14
DVD	Naples	9
PC	Naples	5
TV	Naples	1
TV	Rome	7
PC	Rome	3
DVD	Naples	9
PC	Naples	15
TV	Naples	10

SELECT Produit, Ville, SUM(Qte) as Quantite
FROM Fait_Ventes
GROUP BY Produit, Ville

Produit	Ville	Quantite
DVD	Paris	18
DVD	Naples	18
PC	Paris	12
PC	Naples	20
PC	Rome	17
TV	Naples	11
TV	Rome	11

Extensions à la clause GROUP BY

- ROLLUP et CUBE
- GROUPING
- GROUPING_ID et DECODE

Extension de SQL

- ROLLUP:

  Créé des agrégats à n+1 niveaux. ROLLUP commence par calculer les agrégats spécifiés dans la clause GROUP BY, ensuite les sous totaux plus élevés sont créés progressivement, en parcourant de droite à gauche les attributs regroupés.

  SELECT <liste d'attributs>, <liste d'agrégats>

  FROM <table…>

  GROUP BY ROLLUP(liste d'attributs);

Exemple ROLLUP

PC	Paris	12
DVD	Paris	18
TV	Rome	4
PC	Rome	14
DVD	Naples	9
PC	Naples	5
TV	Naples	1
TV	Rome	7
PC	Rome	3
DVD	Naples	9
PC	Naples	15
TV	Naples	10

SELECT Produit, Ville, SUM(Qte) as Quantite
FROM Fait_Ventes
GROUP BY ROLLUP (Produit, Ville)

DVD	Paris	18
DVD	Naples	18
DVD	-	36
PC	Paris	12
PC	Naples	20
PC	Rome	17
PC	-	49
TV	Naples	11
TV	Rome	11
TV	-	22
-	-	107

Extension de SQL

- CUBE:

  Créé toutes combinaisons d'agrégats

  SELECT <liste d’attributs>, <liste d’agrégats>

  FROM <table…>

  GROUP BY CUBE(liste d’attributs);

Exemple CUBE

PC	Paris	12
DVD	Paris	18
TV	Rome	4
PC	Rome	14
DVD	Naples	9
PC	Naples	5
TV	Naples	1
TV	Rome	7
PC	Rome	3
DVD	Naples	9
PC	Naples	15
TV	Naples	10

SELECT Produit, Ville, SUM(Qte) as Quantite
FROM Fait_Ventes
GROUP BY CUBE (Produit, Ville)

Produit	Lieu	Quantite
DVD	Paris	18
DVD	Naples	18
DVD	-	36
PC	Paris	12
PC	Naples	20
PC	Rome	17
PC	-	49
TV	Naples	11
TV	Rome	11
TV	-	22
-	Paris	30
-	Naples	49
-	Rome	28
-	-	107

Quand faut-il utiliser Rollup ou Cube ?

On peut utiliser ROLLUP lorsque des taches incluent des sous-totaux.

• dans une dimension hiérarchique de type temporelle ou géographique.

•Dans l'emploi de tables de type Summary tables (vues matérialisées)

On peut utiliser Cube dans toutes les situations ou l'on requiert des rapports de tableaux-croisés.

•Dans l'emploi de tables de type Summary tables (vues matérialisées)

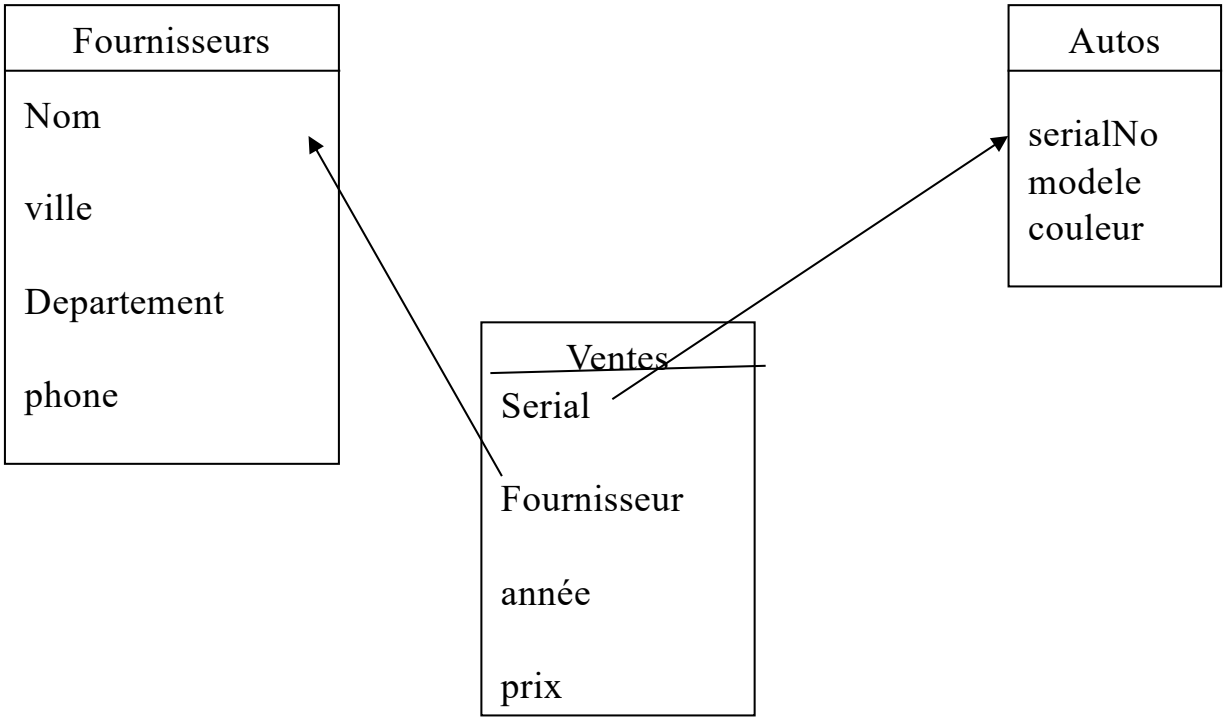
•Convient particulièrement dans les requêtes qui utilisent des colonnes provenant de plusieurs dimensions plutôt que de colonnes représentant différents niveaux d'une seule dimension.

Fonctions GROUPING

- Il est très difficile de déterminer au niveau de la présentation du résultat quels sont les sous totaux ou quel est le niveau exact d'agrégation

- La fonction GROUPING permet d'identifier les lignes super agrégées à l'intérieur d'une expression GROUP BY (ROLLUP et CUBE). GROUPING retourne la valeur 1, lorsque cette ligne est un agrégat ou un sous totaux; dans le cas contraire, la valeur retournée est 0.

Exemple


Fournisseurs
Nom
ville
Departement
phone
Autos
serialNo
modele
couleur
Ventes
Serial
Fournisseur
année
prix

Exemple CUBE avec GROUPING

```
SELECT année, fournisseur, COUNT(*),
        GROUPING(année) AS group_année,
        GROUPING(fournisseur) AS grp_fournisseur
FROM Ventes, Autos
WHERE année=2006 OR année= 2005
GROUP BY CUBE (année, fournisseur);
```

Exemple CUBE avec GROUPING

année	fournisseur	COUNT(*)	GROUP_année	GRP_fournisseur
		3922	**1**	**1**
	fournisseur1	1110	1	0
	fournisseur2	518	1	0
	fournisseur3	1258	1	0
	fournisseur5	370	1	0
	fournisseur6	148	1	0
	fournisseur7	518	1	0
2005		**1036**	**0**	**1**
2005	fournisseur1	296	0	0
2005	fournisseur2	74	0	0
2005	fournisseur3	222	0	0
2005	fournisseur5	222	0	0
2005	fournisseur7	222	0	0
2006		**2886**	**0**	**1**
2006	fournisseur1	814	0	0
2006	fournisseur2	444	0	0
2006	fournisseur3	1036	0	0
2006	fournisseur5	148	0	0
2006	fournisseur6	148	0	0
2006	fournisseur7	296	0	0

Fonction GROUPING_ID

•La fonction GROUPING_ID permet d'économiser de l'espace lors de l'identification des lignes super agrégées.

•A l'appel de la fonction GROUPING, cette fonction identifie ces lignes et renvoie le niveau GROUP BY comme bit vector.

•GROUPIND_ID prend toutes les valeurs 1 et 0 qui ont été créées lors de l'utilisation de la fonction GROUPING et génere à partir de celles-ci un bit-vector par agrégat. Le bit-vector est traité comme un nombre binaire et la valeur est retournée par la fonction GROUPING_ID

Fonction GROUPING_ID (Exemple)

Si on utilise un agrégat avec l'expression CUBE(a, b), les valeurs possibles sont les suivantes :

Niveau d'agrégat	Bit-vector	GROUPING_ID
a, b	0 0	0
a	0 1	1
b	1 0	2
Total	1 1	3

Fonction GROUPING_ID (Exemple)

```
SELECT année,
fournisseur,
COUNT(*),
GROUPING (année)AS GRP_année,
GROUPING (fournisseur) AS GRP_fournisseur,
GROUPING_ID(année, fournisseur) AS grp_id
FROM  Ventes, Autos
WHERE année=2006 OR année= 2005
GROUP BY CUBE (année,fournisseur);
```

Fonction GROUPING_ID (Exemple)

année	fournisseur	COUNT(*)	GRP_année	GRP_fournisseur	GRP_ID
		3922	1	1	3
	fournisseur1	1110	1	0	2
	fournisseur2	518	1	0	2
	fournisseur3	1258	1	0	2
	fournisseur5	370	1	0	2
	fournisseur6	148	1	0	2
	fournisseur7	518	1	0	2
2005		1036	0	1	1
2005	fournisseur1	296	0	0	0
2005	fournisseur2	74	0	0	0
2005	fournisseur3	222	0	0	0
2005	fournisseur5	222	0	0	0
2005	fournisseur7	222	0	0	0
2006		2886	0	1	1
2006	fournisseur1	814	0	0	0
2006	fournisseur2	444	0	0	0
2006	fournisseur3	1036	0	0	0
2006	fournisseur5	148	0	0	0
2006	fournisseur6	148	0	0	0
2006	fournisseur7	296	0	0	0

Fonction GROUPING et DECODE (Exemple)

•On peut augmenter la lisibilité en utilisant les fonctions GROUPING et DECODE.

•La fonction DECODE opère sur le résultat de la fonction GROUPING

```
SELECT DECODE(GROUPING_ID(année, fournisseur),
              1, 'Sous total année',
              2, 'Sous total fournisseur',
              3, 'Total General',
             null ) AS SOMMES,
 année,
 fournisseur,
 COUNT(*)
FROM sales
GROUP BY CUBE ( année,fournisseur);
```

Fonction GROUPING et DECODE (Exemple)

SOMMES	année	fournisseur	COUNT(*)
Total General			57
Sous total fournisseur		fournisseur1	15
Sous total fournisseur		fournisseur2	7
Sous total fournisseur		fournisseur3	21
Sous total fournisseur		fournisseur5	5
Sous total fournisseur		fournisseur6	2
Sous total fournisseur		fournisseur7	7
Sous total année	2003		4
	2003	fournisseur3	4
Sous total année	2005		14
	2005	fournisseur1	4
	2005	fournisseur2	1
	2005	fournisseur3	3
	2005	fournisseur5	3
	2005	fournisseur7	3
Sous total année	2006		39
	2006	fournisseur1	11
	2006	fournisseur2	6
	2006	fournisseur3	14
	2006	fournisseur5	2
	2006	fournisseur6	2
	2006	fournisseur7	4

Fonctions analytiques :

- *Rankings et percentiles*
- *Moving window calculations*
- *Lag/lead analyse* : Trouver une valeur dans une ligne, un nombre spécifique de lignes à partir de la ligne courante.
- *First/last analyse* : Première ou dernière valeur dans un groupe ordonné
- *Statistiques de régression linéaire* :

Fonctions analytiques :

•*Rankings et percentiles* : Répond aux questions du style « montrer les 10 premiers et 10 derniers vendeurs par région » ou « montrer pour chaque région, les vendeurs qui ont fait plus de 25% des ventes »

•*Moving window calculations* :

- •Calcul cumulatif et déplacement d'agrégats.
- •Fonctionne avec les fonctions SUM, AVG, MIN, MAX, COUNT, Variance, STDDEV, FIRST_VALUE, LAST_VALUE.
- •Répond aux questions style « Quel est le ***moving average*** de la 2ème semaine ? » ou « quelle est la somme cumulée des ventes pour chaque région ? »

Fonctions analytiques :

•*Lag/lead analyse* : Trouver une valeur dans une ligne, un nombre spécifique de lignes à partir de la ligne courante.

•*First/last analyse* : Première ou dernière valeur dans un groupe ordonné

•*Statistiques de régression linéaire* :

Fonctions analytiques : syntaxe

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

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

fonction est le nom de la fonction analytique.
over indique qu'on utilise une fonction analytique.

Les trois clauses : de partitionnement, d'ordre et de fenêtrage sont facultatives en général mais souvent nécessaires suivant la fonction utilisée.

Fonctions analytiques : syntaxe

La clause de partitionnement

- est 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.
- C'est analogue au GROUP BY.

La clause d'ordre

- est de la forme **ORDER BY ... [NULLS FIRST|LAST]**
- Elle indique comment les données sont triées à l'intérieur de chaque partition.
- L'emploi de NULLS FIRST/LAST est facultatif : NULLS FIRST indique que les valeurs nulles apparaissent d'abord tandis que NULL LAST indique que ces valeurs apparaissent à la fin.

Fonctions analytiques : syntaxe

La clause de fenêtrage indique l'ensemble des lignes sur laquelle doit être appliquée la fonction. Si une clause de fenêtrage est spécifiée, une clause d'ordre doit obligatoirement l'être aussi.
La clause de fenêtrage est de la forme :

```
{ ROWS | RANGE }
{ BETWEEN
{ UNBOUNDED PRECEDING
 | CURRENT ROW
 | value_expr { PRECEDING | FOLLOWING }
}  AND
{ UNBOUNDED FOLLOWING
 | CURRENT ROW
 | value_expr { PRECEDING | FOLLOWING }
}
 |          { UNBOUNDED PRECEDING
 | CURRENT ROW
 | value_expr PRECEDING
}
 }
```

Fonctions analytiques : syntaxe

ROWS indique qu'on définit une fenêtre en terme de ligne, par exemple. **ROWS 5.**

PRECEDING signifie qu'on prend toutes les lignes comprises entre la ligne courante et celles qui précèdent jusqu'à la 5ème incluse.

RANGE définit une fenêtre en terme de valeur : par exemple pour avoir les dates à moins de 100 jours d'une date de référence indiquée dans la clause *order by* on utiliserait **RANGE 100 PRECEDING**.

RANGE ne fonctionne qu'avec des colonnes de type numérique ou date; la colonne de référence est celle indiquée dans le *order by*, il ne peut y avoir qu'une unique colonne dans le *order by*.

Fonctions analytiques : RANK

RANK () over ([clause de partitionnement] clause d'ordre)

Exemple :

Montrer les 3 premiers fournisseurs et leur chiffre d'affaires

```
SELECT * FROM
(SELECT
 fournisseur, TO_CHAR(SUM(price) , '9,999,999,999') chiffre_d,
            RANK ( ) OVER (ORDER BY SUM(price) DESC ) AS PRICE_RANG
FROM sales, Autos
WHERE serialNo=serial
GROUP BY fournisseur
)
WHERE PRICE_RANG <4;
```

fournisseur	CHIFFRE_D	PRICE_RANG
fournisseur3	1,005,000	1
fournisseur1	562,000	2
fournisseur7	385,000	3

Fonctions analytiques : CUME_DIST

CUME_DIST () over ([clause de partitionnement] clause d'ordre)

Calcule la position d'une valeur relative spécifique par rapport à un jeu de valeurs. La rangée de valeurs pour CUME_DIST est plus grande que 0, jusqu'à 1.

CUME_DIST(x) = nombre de valeurs dans S venant avant et incluant x, dans l'ordre spécifié par ORDRE /N

Exemple : Montre une répartition cumulative des ventes par fournisseur pour chaque année

```
SELECT
   année, fournisseur,
  TO_CHAR(SUM(price), '9,999,999,999') ventes,
  CUME_DIST() OVER ( PARTITION BY année ORDER BY SUM(price)) AS CUME_DIST_ventes
FROM sales, Autos
WHERE serialNo=serial AND année IN ('2006', '2005')
GROUP BY (année,fournisseur);
```

Fonctions analytiques : CUME_DIST

année	fournisseur	VENTES	CUME_DIST_VENTES
2005	fournisseur2	30,000	,2
2005	fournisseur5	90,000	,4
2005	fournisseur1	159,000	,6
2005	fournisseur7	165,000	,8
2005	fournisseur3	180,000	1
2006	fournisseur5	106,000	,166666667
2006	fournisseur6	110,000	,333333333
2006	fournisseur7	220,000	,5
2006	fournisseur2	233,000	,666666667
2006	fournisseur1	403,000	,833333333
2006	fournisseur3	585,000	1

Fonctions analytiques : MAX et MIN

MAX ( ) over ([clause de partitionnement] clause d'ordre)

Exemple : Montre pour chaque fournisseur ses ventes par an ainsi que le maximum et minimum des 3 dernières années où il a eu des ventes.

```
SELECT
            année,
            fournisseur,
 SUM(price) as ventes,
MAX(SUM(price) ) OVER ( PARTITION BY fournisseur ORDER BY année rows 2
             preceding) max_3,
MIN(SUM(price) ) OVER ( PARTITION BY fournisseur ORDER BY
             année rows 2 preceding) min_3
FROM sales, Autos
WHERE serialNo=serial GROUP BY (année, fournisseur);
```

Fonctions analytiques : MAX et MIN

année	fournisseur	VENTES	MAX_3	MIN_3
2005	fournisseur1	159000	159000	159000
2006	fournisseur1	403000	403000	159000
2005	fournisseur2	30000	30000	30000
2006	fournisseur2	233000	233000	30000
2003	fournisseur3	240000	240000	240000
2005	fournisseur3	180000	240000	180000
2006	fournisseur3	585000	585000	180000
2005	fournisseur5	90000	90000	90000
2006	fournisseur5	106000	106000	90000
2006	fournisseur6	110000	110000	110000
2005	fournisseur7	165000	165000	165000
2006	fournisseur7	220000	220000	165000

Fonctions analytiques : LAG et LEAD

LAG ( ) over ([clause de partitionnement] clause d'ordre)

LAG permet de "regarder" en arrière tandis que LEAD permet de "regarder" en avant.

Exemple : . Obtenir pour le fournisseur3 sa vente précédente ainsi que la suivante

```
SELECT
        datesales,
        fournisseur,
SUM(price) as ventes,
LAG(SUM(price), 1 ) OVER ( PARTITION BY fournisseur ORDER BY datesales)
            precedente,
LEAD(SUM(price), 1 ) OVER ( PARTITION BY fournisseur ORDER BY      datesales)
            suivante
FROM sales, Autos
WHERE serialNo=serial and fournisseur='fournisseur3'
GROUP BY ( datesales, fournisseur);
```

Fonctions analytiques : LAG et LEAD

DATESALE	fournisseur	VENTES	PRECEDENTE	SUIVANTE
11/09/03	fournisseur3	240000		180000
24/09/05	fournisseur3	180000	240000	60000
11/04/06	fournisseur3	60000	180000	60000
11/06/06	fournisseur3	60000	60000	149000
30/08/06	fournisseur3	149000	60000	198000
12/09/06	fournisseur3	198000	149000	118000
24/09/06	fournisseur3	118000	198000	

Fonctions analytiques : autres

- **VARIANCE et STDDEV :** VARIANCE donne la variance et STDDEV l'écart-type. Ce sont des mesures de la volatilité.

- **CORRELATION** : CORR renvoie le coefficient de corrélation entre deux variables. Ce coefficient est compris entre -1 (corrélation négative, les deux variables varient en sens inverse) et +1 (corrélation positive, les deux variables varie dans le même sens).

- **FIRST_VALUE et LAST_VALUE** : renvoie la première et la dernière valeur d'une expression pour une fenêtre donnée