

# SQL OLAP

---

Jacky Akoka

Isabelle Comyn-Wattiau

# SQL

- Langage de base de données relationnelles
- Développé chez IBM (1970-80)
- Devenu une norme (ANSI/ISO) en 1986
- A la fois LDD (Langage de Définition de Données) et LMD (Langage de Manipulation de Données)
- Toute interface SQL à un SGBD est une adaptation de la norme à ce SGBD
- Utilisable en mode interactif comme dans un langage de programmation
- Langage assertionnel (non procédural) : on décrit les caractéristiques des données recherchées et non le chemin d'accès
- ici, ORACLE SQL incluant OLAP SQL : extensions SQL pour l'OLAP

# Les standards SQL

- SQL86
- SQL89
- SQL92 ou SQL2
- SQL99 ou SQL3 : contient l'extension OLAP

---

# ***SQL ORACLE***

## ***Manipulation des Données***

# ***SQL ORACLE***

## ***Manipulation des Données***

INTERROGATION :

SELECT

MISE A JOUR :

INSERT

ajout

UPDATE

modification

DELETE

suppression

# *Schéma relationnel servant pour les exemples*

*Table FOURNISSEUR :*

F(FNO,FNOM,STATUT,VILLE)

*Table PIECE :*

P(PNO,PNOM,COULEUR,POIDS,VILLE)

*Table PROJET :*

J(JNO,JNOM,VILLE)

*Table FOURNISSEUR-PIECE-PROJET :*

FPJ(FNO,PNO,JNO,QTE)

# *1. Interrogation sur une table*

## *1.1. Interrogations simples*

SELECT liste-colonnes

FROM nomtable

WHERE condition;

**Sélectionne :**

- ❑ les colonnes précisées dans le SELECT (avec alias : nomcol nouveaunom (Oracle) ou nomcol AS nouveaunom (norme SQL))
- ❑ provenant de la table précisée dans le FROM
- ❑ dont les lignes vérifient la condition précisée dans le WHERE.
- ❑ **Remarques :**
- ❑ si on veut toutes les colonnes : \*
- ❑ si on ne veut pas les doubles : DISTINCT
- ❑ dans le SELECT, on peut aussi mettre des expressions calculées ou des chaînes de caractères (entre ' ')
- ❑ si la condition est complexe, on la construit à l'aide des opérateurs AND, OR et NOT

---

F(FNO,FNOM,STATUT,VILLE)  
P(PNO,PNOM,COULEUR,POIDS,VILLE)  
J(JNO,JNOM,VILLE)  
FPJ(FNO,PNO,JNO,QTE)

- a) Numéros et statuts des fournisseurs localisés à Paris

```
SELECT FNO,STATUT  
FROM F  
WHERE VILLE='Paris';
```

- b) Liste des numéros des pièces

```
SELECT PNO FROM P;
```

- c) Liste des numéros de pièces effectivement fournies

```
SELECT DISTINCT PNO FROM FPJ;
```

- d) Numéros des pièces et poids en grammes

```
SELECT PNO,'Poids en grammes : ', POIDS*454 FROM P;
```



# *Les opérateurs de comparaison*

=

!= ^= <> (différent)

>= > <= <

BETWEEN valeur minimale AND valeur maximale (au sens large)

IS NULL (ne contient pas de valeur)

IS NOT NULL (a une valeur, même 0)

LIKE chaîne de caractères (avec jokers)

*caractères jokers :* % remplace n'importe quelle suite de caractères

'\_' (souligné) remplace n'importe quel caractère

IN ( , , , ) inclusion d'une valeur dans une liste NOT IN ( , , , )

= ANY ( , , , ) équivaut à IN < > ALL ( , , , ) équivaut à NOT IN

*on peut accoler n'importe quel opérateur de comparaison simple (=, !=, >=, >, <=, <) avec ANY ou ALL*

# *1. Interrogation sur une table (suite)*

## *1.2. Interrogations avec tri du résultat*

SELECT liste-colonnes

FROM nomtable

WHERE condition

ORDER BY liste-colonnes;

Dans la clause ORDER BY, on peut avoir des :

- ❑ des noms de colonnes
- ❑ des expressions avec noms de colonnes
- ❑ des numéros de position des colonnes dans la clause SELECT.

On peut préciser le sens (croissant ou décroissant) : ASC ou DESC. Par défaut, c'est croissant.

Les valeurs nulles sont à la fin par ordre croissant, au début par ordre décroissant.

Si ORDER BY et DISTINCT sont spécifiés, la clause ORDER BY ne peut pas se référer à des colonnes non mentionnées dans la clause SELECT.

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

- o) Numéros des fournisseurs localisés à Paris, dans l'ordre décroissant des statuts**

```
SELECT FNO  
FROM F  
WHERE VILLE='Paris'  
ORDER BY STATUT DESC;
```

- p) Numéros des fournisseurs dans l'ordre décroissant des statuts et, pour les fournisseurs de même statut, par ordre alphabétique des noms**

```
SELECT FNO  
FROM F  
ORDER BY STATUT DESC, FNOM ASC;
```

# ***1. Interrogation sur une table (suite)***

## ***1.3. Interrogations avec fonctions***

### *Fonctions numériques*

ABS(n)	valeur absolue
CEIL(n)	partie entière + 1
FLOOR(n)	partie entière
MOD(m,n)	modulo n
POWER(m,n)	puissance n
ROUND(n[,m])	arrondi à m décimales
SIGN(n)	signe de n : -1, 0 ou 1
SQRT(n)	racine carrée
TRUNC(n[,m])	tronqué à m décimales

# ***1. Interrogation sur une table (suite)***

## ***1.3. Interrogations avec fonctions (suite)***

### ***Quelques fonctions sur les caractères***

<b>CHR(n)</b>	<b>caractère de code n</b>
<b>INITCAP(c)</b>	<b>transforme la chaîne en 'Titre' (1ère lettre de chaque mot en majuscules)</b>
<b>LENGTH(c)</b>	<b>longueur de c</b>
<b>LOWER(c)</b>	<b>tout en minuscules</b>
<b>REPLACE(c,chaîne à remplacer,chaîne de remplac.)</b>	
<b>SOUNDEX(c)</b>	<b>représentation phonétique</b>
<b>SUBSTR(c,m,n)</b>	<b>extraît n caractères à partir du m-ième caractère</b>
<b>TRANSLATE(c,c1,c2)</b>	<b>remplace partout dans c le caractère c1 par le caractère c2</b>
<b>UPPER(c)</b>	<b>tout en majuscules</b>

# ***1. Interrogation sur une table (suite)***

## ***1.3. Interrogations avec fonctions***

### ***Quelques fonctions sur les dates***

**MONTHS\_BETWEEN(d1,d2)**

nombre de mois entre 2 dates

**SYSDATE**                      date système

### ***Quelques autres fonctions***

**TO\_DATE(chaine,format)** convertit la chaîne en date dans le format spécifié

ex de format : **DD/MM/YY**

**TO\_CHAR(date,format)** convertit la date en chaîne

**TO\_NUMBER(chaine)** convertit une chaîne en numérique

# ***1. Interrogation sur une table (suite)***

## ***1.3. Interrogations avec fonctions***

***Un exemple de fonction de présentation du résultat***

***DECODE(expression1, expression2, expression3,  
expression4)***

**Une forme de Si ... Alors ... Sinon**

**Si expression1 = expression2, alors afficher expression3 sinon afficher  
expression4**

**Exemple : DECODE(sexe,'H','homme','femme')**

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

- r) **Statut du fournisseur Dupont (on ne sait pas si les noms ont été saisis en minuscules ou majuscules)**

**SELECT STATUT**

**FROM F**

**WHERE UPPER (FNOM)='DUPONT';**

**(ou LOWER (FNOM)='dupont')**

- s) **Supposons que l'on ajoute la date de fin de projet DATPROJ dans la table J, rechercher la liste des projets terminés**

**SELECT \***

**FROM J**

**WHERE DATPROJ < SYSDATE;**



# ***1. Interrogation sur une table (suite)***

## ***1.4. Interrogations avec agrégats de lignes***

**SELECT ... FROM ... WHERE ...**

**GROUP BY liste-colonnes**

**HAVING condition;**

**Les lignes ayant les mêmes valeurs pour l'ensemble des colonnes du GROUP BY sont regroupées.**

**Le SELECT contient une fonction qui porte sur un ensemble de valeurs.**

**Le HAVING permet de tester une condition contenant une fonction agrégat.**

**Liste des principales fonctions agrégats :**

**AVG(colonne)**

**COUNT(DISTINCT colonne) COUNT(\*) COUNT(colonne)**

**MAX(colonne) MIN(colonne)**

**SUM(colonne)**

# *1. Interrogation sur une table (suite)*

*Les fonctions agrégats peuvent figurer dans le SELECT ou dans le HAVING*

- calcul statistique sur un groupe de lignes vérifiant une condition

```
SELECT fonction statistique  
FROM table  
WHERE condition;
```

- calcul sur tous les groupes

```
SELECT fonction statistique  
FROM table  
GROUP BY col1, col2, ...;
```

# *1. Interrogation sur une table (suite)*

- calcul sur différents groupes avec condition sur le groupe  
SELECT ...  
FROM table  
GROUP BY colonnes  
HAVING condition;

## *Remarques :*

- pas de HAVING sans GROUP BY
- quand il y a GROUP BY, la clause SELECT ne peut contenir que des calculs statistiques et/ou les colonnes du GROUP BY

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

t) ~~FPJ(FNO, PNO, JNO, QTE)~~  
**Nombre total de fournisseurs**

```
SELECT COUNT(*)  
FROM F;
```

u) **Nombre total de fournisseurs qui fournissent effectivement des pièces**

```
SELECT COUNT(DISTINCT FNO)  
FROM FPJ;
```

v) **Nombre de fournisseurs qui fournissent des pièces de numéro 'P2'**

```
SELECT COUNT(DISTINCT FNO)  
FROM FPJ  
WHERE PNO='P2';
```

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

**w) Quantité totale de pièce 'P2' vendue**

```
SELECT SUM(QTE)  
FROM FPJ  
WHERE PNO='P2';
```

**x) Valeurs minimale et maximale du statut de fournisseur**

```
SELECT MIN(STATUT),MAX(STATUT)  
FROM F;
```

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

---

FPJ(FNO,PNO,JNO,QTE)

y) **Quantité totale par type de pièce**

**SELECT PNO, SUM(QTE) FROM FPJ  
GROUP BY PNO;**

z) **Quantité moyenne par type de pièce et par projet**

**SELECT PNO,JNO,AVG(QTE) FROM FPJ  
GROUP BY PNO,JNO;**

aa) **Numéros des pièces fournies par plus d'un fournisseur**

**SELECT PNO FROM FPJ  
GROUP BY PNO  
HAVING COUNT(DISTINCT FNO) > 1;**

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

---

**ab) ~~FPI(FNO,PNO,JNO,QTE)~~ Villes dont les fournisseurs ont tous le même statut**

**SELECT VILLE FROM F**

**GROUP BY VILLE**

**HAVING COUNT(DISTINCT STATUT)=1;**

**ac) Villes ayant des fournisseurs d'au moins 2 statuts**

**SELECT VILLE FROM F**

**GROUP BY VILLE**

**HAVING COUNT(DISTINCT STATUT)>1;**

**ou**

**SELECT VILLE FROM F**

**GROUP BY VILLE**

**HAVING MIN(STATUT)<>MAX(STATUT);**

# Commentaires sur les agrégats

- On doit parfois mettre une colonne dans le GROUP BY pour pouvoir l'afficher
- Le WHERE est appliqué avant le HAVING
- Généralisations du GROUP BY : CUBE et ROLLUP



## ***2. Interrogation sur plusieurs tables***

### ***2.1. Sous-interrogations***

Dans la clause WHERE, on peut faire référence à une clause SELECT.

**ad) Noms des fournisseurs qui fournissent la pièce 'P2'**

```
SELECT FNOM FROM F  
WHERE FNO IN (SELECT FNO FROM FPJ  
WHERE PNO='P2');
```

**ae) Numéros des fournisseurs localisés dans la même ville que 'F1'**

```
SELECT FNO FROM F  
WHERE VILLE = (SELECT VILLE FROM F  
WHERE FNO='F1');
```

## *2. Interrogation sur plusieurs tables (suite)*

### *2.2. Jointures*

#### *SYNTAXE Oracle :*

Dans la clause FROM, on précise la liste des tables à joindre.

Dans la clause WHERE, on précise, en plus de la restriction, les critères de jointure.

On peut donner éventuellement un nom d'alias à chaque table.

Le nom d'alias est obligatoire pour les auto-jointures (jointures d'une table avec elle-même).

## *2. Interrogation sur plusieurs tables (suite)*

### *2.2. Jointures (suite)*

#### ***SYNTAXE SQL2 :***

SELECT listecolonne FROM table1 NATURAL JOIN table2;

SELECT listecolonne FROM table1 JOIN table2 ON  
critèredejointure;

SELECT listecolonne FROM table1 CROSS JOIN table2;

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

---

**ah) Noms des fournisseurs qui fournissent la pièce 'P2'**

**SELECT F.FNOM FROM F,FPJ  
WHERE F.FNO=FPJ.FNO AND FPJ.PNO='P2';**

**ai) Numéros des fournisseurs localisés dans la même ville que 'F1'**

**SELECT FOUR2.FNO FROM F FOUR1, F FOUR2  
WHERE FOUR2.VILLE=FOUR1.VILLE  
AND FOUR1.FNO='F1';**

**aj) Noms des fournisseurs qui fournissent au moins une pièce rouge**

**SELECT F.FNOM FROM F,P,FPJ  
WHERE F.FNO=FPJ.FNO AND P.PNO=FPJ.PNO  
AND P.COULEUR='rouge';**

---

## *2. Interrogation sur plusieurs tables (suite)*

### *2.3. Jointures externes*

Permettent d'ajouter (avec des valeurs nulles) les lignes n'ayant pas de correspondant dans l'autre table

-> ajouter (+) après la colonne à conserver (syntaxe Oracle)

aq) Liste des quantités fournies par pièce avec les caractéristiques de la pièce

```
SELECT P.*, FNO, JNO  
FROM P, FPJ  
WHERE P.PNO=FPJ.PNO;
```

ne donne pas ces quantités si la pièce n'est pas décrite dans la table P

## *2. Interrogation sur plusieurs tables (suite)*

### *2.3. Jointures externes (suite) – syntaxe Oracle*

```
SELECT P.*, FPJ.*  
FROM P, FPJ  
WHERE P.PNO=FPJ.PNO (+);
```

donne toutes les pièces  
même si elles n'ont pas  
été commandées

```
SELECT P.*, FPJ.*  
FROM P, FPJ  
WHERE P.PNO (+)=FPJ.PNO;
```

donne toutes les commandes  
même celles de pièces non  
référéncées dans P

```
SELECT P.*, FPJ.*  
FROM P, FPJ  
WHERE P.PNO (+)=FPJ.PNO(+);
```

interdit : une seule  
jointure externe par  
prédicat

## *2. Interrogation sur plusieurs tables (suite)*

### *2.3. Jointures externes (suite) – syntaxe norme SQL*

SELECT *	donne toutes les pièces
FROM P NATURAL LEFT OUTER JOIN FPJ;	même si elles n'ont pas été commandées

SELECT *	donne toutes les
FROM P NATURAL RIGHT OUTER JOIN FPJ;	commandes même celles de pièces non référéncées dans P

SELECT * FROM P	les deux
NATURAL FULL OUTER JOIN FPJ;	

## ***2. Interrogation sur plusieurs tables (suite)***

### ***2.4. Opérateur EXISTS***

Avec l'opérateur EXISTS, on peut tester le contenu d'une clause SELECT. Il vaut vrai si le résultat du SELECT contient au moins une ligne, faux sinon.

**ar) Noms des fournisseurs qui fournissent la pièce 'P2'**

```
SELECT FNOM  
FROM F  
WHERE EXISTS (SELECT *  
               FROM FPJ  
               WHERE FNO=F.FNO  
               AND PNO='P2');
```



---

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

**as) Noms des fournisseurs qui ne fournissent pas 'P2'**

```
SELECT FNOM
FROM F
WHERE NOT EXISTS (SELECT * FROM FPJ
                  WHERE FNO=F.FNO
                  AND PNO='P2');
```

**at) Noms des fournisseurs qui fournissent toutes les pièces**

```
SELECT FNOM
FROM F
WHERE NOT EXISTS (SELECT * FROM P
                  WHERE NOT EXISTS
                    (SELECT * FROM FPJ
                     WHERE FNO=F.FNO
                     AND PNO=P.PNO));
```

---

### *3. Opérations ensemblistes sur les interrogations*

On peut réaliser des opérations ensemblistes sur les clauses SELECT.

#### *3 opérations ensemblistes*

UNION	union de deux ensembles
INTERSECT	intersection de deux ensembles
MINUS	différence de deux ensembles (norme : EXCEPT)

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

**au) Numéros des pièces fournies par le fournisseur F2 ou qui pèsent plus de 16 livres**

*avec opérateur  
ensemble*

```
SELECT PNO
FROM P
WHERE POIDS > 16
UNION
SELECT PNO
FROM FPJ
WHERE FNO = 'F2';
```

*sans opérateur  
ensemble*

```
SELECT PNO
FROM P
WHERE POIDS > 16
OR PNO IN (SELECT PNO
           FROM FPJ
           WHERE FNO = 'F2');
```

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

## Variante UNION ALL : garde les doublons

au\*) **Numéros des pièces fournies par le fournisseur F2 ou qui pèsent plus de 16 livres**

*avec opérateur  
ensemble*

```
SELECT PNO  
FROM P  
WHERE POIDS > 16  
UNION ALL  
SELECT PNO  
FROM FPJ  
WHERE FNO = 'F2';
```

*sans opérateur  
ensemble*

```
SELECT PNO  
FROM P  
WHERE POIDS > 16  
OR PNO IN (SELECT PNO  
FROM FPJ  
WHERE FNO = 'F2');
```

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

**av) Numéros des pièces fournies par le fournisseur F2 et qui pèsent plus de 16 livres**

*avec opérateur  
ensemble*

```
SELECT PNO
FROM P
WHERE POIDS > 16
INTERSECT
SELECT PNO
FROM FPJ
WHERE FNO = 'F2';
```

*sans opérateur  
ensemble*

```
SELECT PNO
FROM P
WHERE POIDS > 16
AND PNO IN (SELECT PNO
FROM FPJ
WHERE FNO = 'F2');
```

F(FNO,FNOM,STATUT,VILLE)

P(PNO,PNOM,COULEUR,POIDS,VILLE)

J(JNO,JNOM,VILLE)

FPJ(FNO,PNO,JNO,QTE)

**aw) Numéros des fournisseurs qui ne fournissent pas la pièce 'P2'**

*avec opérateur  
ensembliste*

```
SELECT FNO
FROM F
MINUS
SELECT FNO
FROM FPJ
WHERE PNO='P2';
```

*sans opérateur  
ensembliste*

```
SELECT FNO
FROM F
WHERE FNO NOT IN
      (SELECT FNO FROM FPJ
       WHERE PNO='P2');
```

## 5. Autres possibilités de SQL

- **SELECT dans le FROM : sélection de sélection**
  - **SELECT ncom FROM**  
((SELECT ncom,npro FROM lignecommande)  
MINUS  
(SELECT ncom,npro FROM lignelivraison));  
Commandes dont une ligne au moins n'a pas de livraison  
Permet de simplifier le résultat

## 5. Autres possibilités de SQL

- Requête récursive : courses cyclistes

Etape (villedépart, villearrivée)

```
SELECT villearrivée FROM Etape
```

```
START WITH villedépart='Dunkerque'
```

```
CONNECT BY PRIOR villearrivée=villedépart;
```

Donne toutes les villes d'arrivée après Dunkerque

Idem avec LEVEL dans le SELECT qui donne le niveau de l'élément dans le parcours



## 6. SQL OLAP

- Pour optimiser l'agrégation dans le contexte décisionnel
- La clause GROUP BY a été enrichie par :
  - CUBE
  - ROLLUP
  - Fonctions de GROUPING
  - L'expression GROUPING SETS
- Objectif : faciliter le requêtage et la fabrication de rapports

# Principe

- 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 permettent le groupe d'appartenance de chaque tuple pour calculer les sous-totaux et les filtres

# Exemple utilisé pour OLAP

## 6.1 : GROUP BY ROLLUP

- Le GROUP BY ROLLUP calcule 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 :
  - ❑ ROLLUP(année, mois, jour)
  - ❑ ROLLUP (pays, état, ville)
- Simplifie et accélère la maintenance des tables de synthèse

## 6.1 : GROUP BY ROLLUP

EXEMPLE a : La requête

```
SELECT channel_desc,calendar_month_desc, country_iso_code,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id  
AND sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id = channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY  
ROLLUP(channel_desc, calendar_month_desc, country_iso_code);
```

# 6.1 : GROUP BY ROLLUP

EXEMPLE a : Le résultat

CHANNEL_DESC	CALENDAR	CO	SALES\$
-----	-----	--	-----
Internet	2000-09	GB	16,569
Internet	2000-09	US	124,224
Internet	2000-09		140,793
Internet	2000-10	GB	14,539
Internet	2000-10	US	137,054
Internet	2000-10		151,593
Internet			292,387
Direct Sales	2000-09	GB	85,223
Direct Sales	2000-09	US	638,201
Direct Sales	2000-09		723,424
Direct Sales	2000-10	GB	91,925
Direct Sales	2000-10	US	682,297
Direct Sales	2000-10		774,222
Direct Sales			1,497,646
			1,790,032

# 6.1 : GROUP BY ROLLUP

- Calcule le niveau
  - ❑ GROUP BY (channel\_desc,  
calendar\_month\_desc country\_iso\_code)
  - ❑ Premier niveau de totalisation tous pays confondus pour toutes les combinaisons de channel\_desc et calendar\_month\_desc
  - ❑ Deuxième niveau de totalisation par canal de distribution
  - ❑ Total général

## 6.2. GROUP BY ROLLUP partiel

- Permet de totaliser à certains niveaux sur une dimension
- GROUP BY expr1, ROLLUP(expr2, expr3)
  - ❑ Crée  $2+1 = 3$  niveaux d'agrégation
  - ❑ (expr1, expr2, expr3)
  - ❑ (expr1, expr2)
  - ❑ (expr1)
  - ❑ Pas de total général



## 6.2 : GROUP BY ROLLUP partiel

EXEMPLE b : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id= channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY channel_desc,  
         ROLLUP(calendar_month_desc, country_iso_code);
```

## 6.2 : GROUP BY ROLLUP partiel

EXEMPLE b : Le résultat

CHANNEL_DESC	CALENDAR	CO	SALES\$
-----	-----	--	-----
Internet	2000-09	GB	16,569
Internet	2000-09	US	124,224
Internet	2000-09		140,793
Internet	2000-10	GB	14,539
Internet	2000-10	US	137,054
Internet	2000-10		151,593
Internet			292,387
Direct Sales	2000-09	GB	85,223
Direct Sales	2000-09	US	638,201
Direct Sales	2000-09		723,424
Direct Sales	2000-10	GB	91,925
Direct Sales	2000-10	US	682,297
Direct Sales	2000-10		774,222
Direct Sales			1,497,646

## 6.3. GROUP BY CUBE

- GROUP BY CUBE
  - Crée des sous-totaux pour toutes les combinaisons possibles d'un ensemble de colonnes de regroupement
  - CUBE sur les dimensions temps, géographie et canal de distribution calcule tous les sous-totaux des ventes pour toutes les combinaisons
  - 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
- Le GROUP BY CUBE est une alternative plus performante que le UNION ALL

## 6.3 : GROUP BY CUBE

EXEMPLE c : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND  
       sales.cust_id=customers.cust_id AND  
       sales.channel_id= channels.channel_id  
AND customers.country_id = countries.country_id  
AND channel_desc IN ('Direct Sales', 'Internet') AND  
   calendar_month_desc IN ('2000-09', '2000-10') AND  
   country_iso_code IN ('GB', 'US')  
GROUP BY CUBE(channel_desc, calendar_month_desc,  
               country_iso_code);
```

## 6.3 : GROUP BY CUBE

CHANNEL_DESC	CALENDAR	CO	SALES\$
-----	-----	--	-----
			1,790,032
		GB	208,257
		US	1,581,775
	2000-09		864,217
	2000-09	GB	101,792
	2000-09	US	762,425
	2000-10		925,815
	2000-10	GB	106,465
	2000-10	US	819,351
Internet			292,387
Internet		GB	31,109
Internet		US	261,278
Internet	2000-09		140,793
Internet	2000-09	GB	16,569
Internet	2000-09	US	124,224
Internet	2000-10		151,593
Internet	2000-10	GB	14,539
Internet	2000-10	US	137,054
Direct Sales			1,497,646
Direct Sales		GB	177,148
Direct Sales		US	1,320,497
Direct Sales	2000-09		723,424
Direct Sales	2000-09	GB	85,223
Direct Sales	2000-09	US	638,201
Direct Sales	2000-10		774,222
Direct Sales	2000-10	GB	91,925
Direct Sales	2000-10	US	682,297

EXEMPLE c : Le résultat

## 6.4 : GROUP BY CUBE partiel

- GROUP BY expr1, CUBE(expr2, expr3)
  - Calcule 4 niveaux de regroupement :
    - (expr1, expr2, expr3)
    - (expr1, expr2)
    - (expr1, expr3)
    - (expr1)
  - Ne calcule pas de total général

## 6.4 : GROUP BY CUBE partiel

EXEMPLE d : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id = times.time_id  
AND sales.cust_id = customers.cust_id  
AND customers.country_id=countries.country_id  
AND sales.channel_id = channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY channel_desc,  
         CUBE(calendar_month_desc, country_iso_code);
```

## 6.4 : GROUP BY CUBE partiel

EXEMPLE d : Le résultat

CHANNEL_DESC	CALENDAR	CO	SALES\$
-----	-----	--	-----
Internet			292,387
Internet		GB	31,109
Internet		US	261,278
Internet	2000-09		140,793
Internet	2000-09	GB	16,569
Internet	2000-09	US	124,224
Internet	2000-10		151,593
Internet	2000-10	GB	14,539
Internet	2000-10	US	137,054
Direct Sales			1,497,646
Direct Sales		GB	177,148
Direct Sales		US	1,320,497
Direct Sales	2000-09		723,424
Direct Sales	2000-09	GB	85,223
Direct Sales	2000-09	US	638,201
Direct Sales	2000-10		774,222
Direct Sales	2000-10	GB	91,925
Direct Sales	2000-10	US	682,297



# Remarques générales sur CUBE et ROLLUP

- Attention, les lignes de totalisation contiennent des valeurs nulles
- Problème si la table initiale contient aussi des valeurs nulles
- On a besoin parfois d'autres fonctions que des sommes
- Pour exploiter le résultat d'un CUBE ou d'un ROLLUP, on a besoin de savoir quelles sont les lignes de totalisation et combien de niveaux de totalisation comporte chaque ligne

## 6.5. Fonction GROUPING

- GROUPING est une fonction qui s'insère dans le SELECT
- GROUPING a un argument qui est un nom de colonne
- Elle renvoie 1 quand elle rencontre une valeur nulle créée par ROLLUP ou CUBE
- Sinon elle renvoie 0
  - Valeur non nulle
  - Valeur nulle initialement contenue dans la table

## 6.5. Fonction GROUPING

EXEMPLE e : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$,  
GROUPING(channel_desc) AS Ch,  
GROUPING(calendar_month_desc) AS Mo, GROUPING(country_iso_code) AS Co  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id  
AND sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id= channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY ROLLUP(channel_desc, calendar_month_desc, country_iso_code);
```

## 6.5. Fonction GROUPING

EXEMPLE e : Le résultat

CHANNEL_DESC	CALENDAR	CO	SALES\$	CH	MO	CO
-----	-----	--	-----	-----	-----	-----
Internet	2000-09	GB	16,569	0	0	0
Internet	2000-09	US	124,224	0	0	0
Internet	2000-09		140,793	0	0	1
Internet	2000-10	GB	14,539	0	0	0
Internet	2000-10	US	137,054	0	0	0
Internet	2000-10		151,593	0	0	1
Internet			292,387	0	1	1
Direct Sales	2000-09	GB	85,223	0	0	0
Direct Sales	2000-09	US	638,201	0	0	0
Direct Sales	2000-09		723,424	0	0	1
Direct Sales	2000-10	GB	91,925	0	0	0
Direct Sales	2000-10	US	682,297	0	0	0
Direct Sales	2000-10		774,222	0	0	1
Direct Sales			1,497,646	0	1	1
			1,790,032	1	1	1

## 6.5. Fonction GROUPING

- Combiné avec la fonction Oracle DECODE, permet d'améliorer la lisibilité
- Exemple f : la requête

```
SELECT DECODE(GROUPING(channel_desc), 1, 'Multi-channel sum',  
             channel_desc) AS Channel, DECODE (GROUPING (country_iso_code),  
             1, 'Multi-country sum',country_iso_code) AS Country,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id  
AND sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id= channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc= '2000-09'  
AND country_iso_code IN ('GB', 'US')  
GROUP BY CUBE(channel_desc, country_iso_code);
```

## 6.5. Fonction GROUPING

EXEMPLE f : Le résultat

CHANNEL	COUNTRY	SALES\$
-----	-----	-----
Multi-channel sum	Multi-country sum	864,217
Multi-channel sum	GB	101,792
Multi-channel sum	US	762,425
Internet	Multi-country sum	140,793
Internet	GB	16,569
Internet	US	124,224
Direct Sales	Multi-country sum	723,424
Direct Sales	GB	85,223
Direct Sales	US	638,201

## 6.5. Fonction GROUPING

- GROUPING avec HAVING permet de sélectionner les niveaux de regroupement à afficher
- Par exemple, tous les niveaux sauf le moins agrégé
- Résultat exemple g :

CHANNEL_DESC C	CO	SALES\$	CH	MO	CO
-----	--	-----	-----	-----	-----
	US	1,581,775	1	1	0
	GB	208,257	1	1	0
Direct Sales		1,497,646	0	1	1
Internet		292,387	0	1	1
		1,790,032	1	1	1

## 6.5. Fonction GROUPING

EXEMPLE g : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code, TO_CHAR(  
SUM(amount_sold), '9,999,999,999') SALES$, GROUPING(channel_desc) CH,  
GROUPING  
(calendar_month_desc) MO, GROUPING(country_iso_code) CO  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id= channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY CUBE(channel_desc, calendar_month_desc, country_iso_code)  
HAVING (GROUPING(channel_desc)=1 AND  
GROUPING(calendar_month_desc)= 1  
AND GROUPING(country_iso_code)=1) OR (GROUPING(channel_desc)=1  
AND GROUPING (calendar_month_desc)= 1) OR  
(GROUPING(country_iso_code)=1  
AND GROUPING(calendar_month_desc)= 1);
```



## 6.6. Fonction GROUPING\_ID

- 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

## 6.6. Fonction GROUPING\_ID

- CUBE(a,b)

Niveau d'agrégation	Vecteur de bits	GROUPING_ID
a,b	0 0	0
a	0 1	1
b	1 0	2
Total général	1 1	3

- Très utile pour rafraîchir les vues matérialisées

## 6.6. Fonction GROUPING\_ID

- | EMPLOYEE_ID | DIV | JOB | FIRST_NAME | LAST_NAME | SALARY |
|-------------|-----|-----|------------|-----------|--------|
| 1           | BUS | PRE | James      | Smith     | 800000 |
| 2           | SAL | MGR | Ron        | Johnson   | 350000 |
| 3           | SAL | WOR | Fred       | Hobbs     | 140000 |
| 4           | SUP | MGR | Susan      | Jones     | 200000 |
| 5           | SAL | WOR | Rob        | Green     | 350000 |

GROUPING\_ID() calcule la somme des niveaux de regroupement

- ```

SELECT division_id, job_id, GROUPING_ID(division_id, job_id) AS grp_id,
SUM(salary)
FROM employee
GROUP BY CUBE(division_id, job_id)
HAVING GROUPING_ID(division_id, job_id) > 0;

```

| DIV | JOB | GROUPING_ID | SUM(SALARY) |
|-----|-----|-------------|-------------|
| BUS | MGR | 3           | 1840000     |
|     |     | 2           | 550000      |
|     |     | 2           | 800000      |
|     |     | 2           | 490000      |
|     | PRE | 1           | 800000      |
|     |     | 1           | 840000      |
| SAL | WOR | 1           | 200000      |

## 6.7. Fonction GROUP\_ID

- La variété des possibilités du GROUP BY peut conduire à dupliquer des niveaux d'agrégation
- La fonction GROUP\_ID affecte la valeur 0 à tous les tuples, qui sont le premier calcul et 1, voire 2 ou plus, aux calculs redondants
- Fonction sans argument GROUP\_ID( )
- Un exemple sera fourni avec GROUPING SETS

## 6.8 : GROUPING SETS

- Permet de spécifier des ensembles précis de regroupements
- `SELECT manager_id, hire_date, count(*)  
FROM employees GROUP BY GROUPING  
SETS (manager_id, hire_date);`
- Équivalent à l'union suivante :
  - ❑ `SELECT manager_id, null hire_date, count(*)  
FROM employees GROUP BY manager_id, 2  
UNION ALL SELECT null, hire_date, count(*)  
FROM employees GROUP BY 1, hire_date;`

## 6.8 : GROUPING SETS

EXEMPLE h : La requête

```
SELECT country_iso_code, SUBSTR(cust_state_province,1,12),  
       SUM(amount_sold),  
       GROUPING_ID(country_iso_code, cust_state_province)  
       GROUPING_ID, GROUP_ID()  
FROM sales, customers, times, countries  
WHERE sales.time_id=times.time_id AND  
       sales.cust_id=customers.cust_id  
AND customers.country_id=countries.country_id AND  
       times.time_id= '30-OCT-00'  
AND country_iso_code IN ('FR', 'ES')  
GROUP BY GROUPING SETS (country_iso_code,  
ROLLUP(country_iso_code, cust_state_province));
```

## 6.8 : GROUPING SETS

EXEMPLE h : Le résultat

| CO  | SUBSTR(CUST_ | SUM(AMOUNT_SOLD) | GROUPING_ID | GROUP_ID() |
|-----|--------------|------------------|-------------|------------|
| --- | -----        | -----            | -----       | -----      |
| ES  | Alicante     | 135.32           | 0           | 0          |
| ES  | Valencia     | 4133.56          | 0           | 0          |
| ES  | Barcelona    | 24.22            | 0           | 0          |
| FR  | Centre       | 74.3             | 0           | 0          |
| FR  | Aquitaine    | 231.97           | 0           | 0          |
| FR  | Rhône-Alpes  | 1624.69          | 0           | 0          |
| FR  | Ile-de-Franc | 1860.59          | 0           | 0          |
| FR  | Languedoc-Ro | 4287.4           | 0           | 0          |
|     |              | 12372.05         | 3           | 0          |
| ES  |              | 4293.1           | 1           | 0          |
| FR  |              | 8078.95          | 1           | 0          |
| ES  |              | 4293.1           | 1           | 1          |
| FR  |              | 8078.95          | 1           | 1          |

---

## 6.8 : GROUPING SETS

EXEMPLE i : La requête

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND  
       sales.cust_id=customers.cust_id AND  
       sales.channel_id= channels.channel_id AND channel_desc IN  
('Direct Sales', 'Internet') AND calendar_month_desc IN  
('2000-09', '2000-10') AND country_iso_code IN ('GB', 'US')  
GROUP BY GROUPING SETS((channel_desc, calendar_month_desc,  
                          country_iso_code),  
                          (channel_desc, country_iso_code), (calendar_month_desc,  
                          country_iso_code));
```



---

## 6.8 : GROUPING SETS

- Calcule les agrégats pour 3 regroupements :
  - ❑ (channel\_desc, calendar\_month\_desc, country\_iso\_code)
  - ❑ (channel\_desc, country\_iso\_code)
  - ❑ (calendar\_month\_desc, country\_iso\_code)

## 6.8 : GROUPING SETS

- Équivalent à la requête :  
SELECT channel\_desc, calendar\_month\_desc, country\_iso\_code,  
TO\_CHAR(SUM(amount\_sold), '9,999,999,999') SALES\$,  
GROUPING\_ID(channel\_desc, calendar\_month\_desc, country\_iso\_code) gid  
FROM sales, customers, times, channels, countries  
WHERE sales.time\_id=times.time\_id AND sales.cust\_id=customers.cust\_id AND  
sales.channel\_id= channels.channel\_id AND  
channel\_desc IN ('Direct Sales', 'Internet') AND calendar\_month\_desc IN  
( '2000-09', '2000-10') AND country\_iso\_code IN ('GB', 'US')  
GROUP BY CUBE(channel\_desc, calendar\_month\_desc, country\_iso\_code)  
HAVING  
GROUPING\_ID(channel\_desc, calendar\_month\_desc, country\_iso\_code)=0  
OR GROUPING\_ID(channel\_desc, calendar\_month\_desc, country\_iso\_code)=2  
OR GROUPING\_ID(channel\_desc, calendar\_month\_desc, country\_iso\_code)=4;

## 6.8 : GROUPING SETS

- La requête précédente calcule 8 regroupements ( $2*2*2$ ) alors que seuls 3 niveaux sont requis
- Utilise des colonnes composites : voir plus loin
- Equivalent à trois GROUP BY reliés par des UNION ALL
- Les UNION ALL non optimisés génèrent autant de parcours complets de tables

## 6.8 : GROUPING SETS

| <b>GROUPING SETS</b>                                | <b>GROUP BY équivalent</b>                                    |
|-----------------------------------------------------|---------------------------------------------------------------|
| <b>GROUP BY GROUPING SETS (a, b, c)</b>             | GROUP BY a UNION ALL<br>GROUP BY b UNION ALL<br>GROUP BY c    |
| <b>GROUP BY<br/>GROUPING SETS (a, b, (b, c))</b>    | GROUP BY a UNION ALL<br>GROUP BY b UNION ALL<br>GROUP BY b, c |
| <b>GROUP BY GROUPING SETS ((a, b, c))</b>           | GROUP BY a, b, c                                              |
| <b>GROUP BY GROUPING SETS (a,(b),())</b>            | GROUP BY a UNION ALL<br>GROUP BY b UNION ALL<br>GROUP BY ( )  |
| <b>GROUP BY<br/>GROUPING SETS (a, ROLLUP(b, c))</b> | GROUP BY a UNION ALL<br>GROUP BY ROLLUP (b, c)                |

## 6.9 : Colonnes composites

- C'est une collection de colonnes qui est traitée comme un tout dans les regroupements
- Il suffit de les mettre entre parenthèses
- Exemple : ROLLUP (année, (trimestre, mois), jour)
  - Calcule les regroupements suivants :
    - (année, trimestre, mois, jour)
    - (année, trimestre, mois)
    - (année)
    - ( )
- Permet de « sauter » certains niveaux de regroupement

## 6.9 : Colonnes composites

```
SELECT channel_desc, calendar_month_desc, country_iso_code,  
TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND  
      sales.cust_id=customers.cust_id  
AND customers.country_id = countries.country_id  
AND sales.channel_id= channels.channel_id  
AND channel_desc IN ('Direct Sales', 'Internet')  
AND calendar_month_desc IN ('2000-09', '2000-10')  
AND country_iso_code IN ('GB', 'US')  
GROUP BY ROLLUP(channel_desc, calendar_month_desc,  
      country_iso_code);
```

## 6.9 : Colonnes composites

- La requête précédente calcule les regroupements ci-dessous :
  - (channel\_desc, calendar\_month\_desc, country\_iso\_code)
  - (channel\_desc, calendar\_month\_desc)
  - (channel\_desc)
  - ( )
- Si on ne veut pas le deuxième niveau, on va mettre :
  - GROUP BY ROLLUP(channel\_desc, (calendar\_month\_desc, country\_iso\_code));

## 6.10 : Groupements concaténés

- On peut combiner plusieurs GROUPING SETS, ainsi que des ROLLUP et CUBE dans un même GROUP BY
- GROUP BY GROUPING SETS (a, b), GROUPING SETS (c, d)
  - Effectue les regroupements (a, c), (a, d), (b, c), (b, d)
  - Évite l'énumération de tous les regroupements
  - Par exemple, un GROUPING SET sur chaque hiérarchie de dimension



## 6.10 : Groupements concaténés

```
SELECT channel_desc, calendar_year, calendar_quarter_desc,  
       country_iso_code, cust_state_province,  
       TO_CHAR(SUM(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id = times.time_id AND sales.cust_id =  
       customers.cust_id  
AND sales.channel_id = channels.channel_id AND  
       countries.country_id =  
       customers.country_id AND channel_desc IN  
('Direct Sales', 'Internet') AND calendar_month_desc IN ('2000-09',  
'2000-10') AND country_iso_code IN ('GB', 'FR')  
GROUP BY channel_desc, GROUPING SETS  
       (ROLLUP(calendar_year, calendar_quarter_desc),  
       ROLLUP(country_iso_code, cust_state_province));
```

## 6.10 : Groupements concaténés

- Effectue les regroupements
  - ❑ (channel\_desc, calendar\_year, calendar\_quarter\_desc)
  - ❑ (channel\_desc, calendar\_year)
  - ❑ (channel\_desc)
  - ❑ (channel\_desc, country\_iso\_code, cust\_state\_province)
  - ❑ (channel\_desc, country\_iso\_code)
  - ❑ (channel\_desc)
- On peut filtrer le doublon avec une fonction GROUP\_ID

## 6.10 : Groupements concaténés

```
SELECT country_iso_code, cust_state_province, calendar_year,  
calendar_quarter_desc, TO_CHAR(SUM(amount_sold), '9,999,999,999')  
    SALES$  
FROM sales, customers, times, channels, countries  
WHERE sales.time_id=times.time_id AND  
    sales.cust_id=customers.cust_id AND  
countries.country_id=customers.country_id AND  
sales.channel_id= channels.channel_id AND channel_desc IN  
('Direct Sales', 'Internet') AND calendar_month_desc IN  
('2000-09', '2000-10') AND country_iso_code IN ('GB', 'FR')  
GROUP BY GROUPING SETS (country_iso_code, cust_state_province),  
GROUPING SETS (calendar_year, calendar_quarter_desc);
```

## 6.10 : Groupements concaténés

- Effectue les regroupements
  - ❑ (country\_iso\_code, year)
  - ❑ (country\_iso\_code, calendar\_quarter\_desc)
  - ❑ (cust\_state\_province, year)
  - ❑ (cust\_state\_province, calendar\_quarter\_desc)

## 6.10 : Groupements concaténés

- Permet de générer exactement les regroupements souhaités
- Exemple :
  - Temps : année, trimestre, mois, jour
  - Produit : catégorie, sous-catégorie, nom produit
  - Géographie : région, sous-région, pays, état, ville
  - 12 colonnes de dimensions + colonne montant des ventes

## 6.10 : Groupements concaténés

```
SELECT calendar_year, calendar_quarter_desc, calendar_month_desc,  
country_region, country_subregion, country_iso_code,  
cust_state_province, cust_city, prod_category_desc, prod_subcategory_desc,  
prod_name, TO_CHAR(SUM  
(amount_sold), '9,999,999,999') SALES$  
FROM sales, customers, times, channels, countries, products  
WHERE sales.time_id=times.time_id AND sales.cust_id=customers.cust_id AND  
sales.channel_id= channels.channel_id AND sales.prod_id=products.prod_id AND  
customers.country_id=countries.country_id AND channel_desc IN  
( 'Direct Sales', 'Internet') AND calendar_month_desc IN  
( '2000-09', '2000-10') AND prod_name IN ('Envoy Ambassador',  
'Mouse Pad') AND country_iso_code IN ('GB', 'US')  
GROUP BY ROLLUP(calendar_year, calendar_quarter_desc,  
calendar_month_desc),  
ROLLUP(country_region, country_subregion, country_iso_code,  
cust_state_province, cust_city),  
ROLLUP(prod_category_desc, prod_subcategory_desc, prod_name);
```

## 6.10 : Groupements concaténés

- Le GROUP BY CUBE sur les 12 dimensions générerait  $2^{12} = 4096$  niveaux de regroupement
- Alors que la requête crée  $4 \times 4 \times 6 = 96$  regroupements :

| ROLLUP temps         | ROLLUP produit              | ROLLUP géographie                                                                                          |
|----------------------|-----------------------------|------------------------------------------------------------------------------------------------------------|
| Year, quarter, month | Category, subcategory, name | Region, subregion, country, state, city<br>Region, subregion, country, state<br>Region, subregion, country |
| Year, quarter        | Category, subcategory       | Region, subregion                                                                                          |
| Year                 | Category                    | Region                                                                                                     |
| ()                   | ()                          | ()                                                                                                         |

## 6.11. Les hiérarchies d'agrégation

- Les expressions CUBE et ROLLUP ne prennent pas du tout en compte les méta-données
- Elles calculent selon l'ordre des colonnes, indépendamment de tout critère sémantique
- Attention au nombre de combinaisons générées :
  - En théorie, on peut spécifier 255 colonnes dans un GROUP BY
  - Mais n colonnes dans le GROUP BY génèrent  $2^n$  combinaisons dans l'ensemble résultat !
- Un HAVING éventuel porte sur toutes les lignes du GROUP BY, quel que soit leur niveau de totalisation
- Un ORDER BY éventuel est appliqué après le GROUP BY
- Utiliser les fonctions GROUPING dans la clause ORDER BY pour différencier entre les niveaux de totalisation
- Autres fonctions : COUNT, AVG, MIN, MAX, STDDEV, VARIANCE



## 6.12. Exemple avec ORDER

- Permet de trier des lignes de sous-totaux et de filtrer les résultats

```
SELECT employee.emp_id AS Employee, year(order_date)
      AS Year, COUNT(*) AS Orders, GROUPING ( Employee )
      AS GE, GROUPING ( Year ) AS GY
FROM employee
LEFT OUTER JOIN sales_order
ON employee.emp_id = sales_order.sales_rep
WHERE employee.sex IN ('F')
AND employee.state IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (Year, Employee)
HAVING GE=1 OR GY=1;
```

## 6.13 : Clause WITH

- Clause de factorisation : permet de réutiliser le même bloc dans un SELECT s'il est nécessaire plusieurs fois
- Standard SQL99
- Le bloc est calculé une seule fois
- Oracle stocke le résultat dans un tablespace temporaire
- Oracle n'autorise pas l'utilisation récursive de la clause WITH

## 6.13 : Clause WITH

```
WITH channel_summary AS (SELECT channel_desc,  
    SUM(amount_sold)  
AS channel_total FROM sales, channels  
WHERE sales.channel_id = channels.channel_id GROUP BY  
    channels.channel_desc)  
SELECT channel_desc, channel_total  
FROM channel_summary WHERE channel_total > (SELECT  
    SUM(channel_total) * 1/3 FROM channel_summary);
```

| CHANNEL_DESC | CHANNEL_TOTAL |
|--------------|---------------|
| -----        | -----         |
| Direct Sales | 57875260.6    |

- Somme des ventes pour chaque canal dans channel\_summary, puis recherche les canaux qui réaliseraient, à eux seuls, plus du tiers des ventes

## 6.13 : Utilisation des cubes hiérarchiques

- On peut emboîter des SELECT

```
SELECT month, division, sum_sales FROM  
(SELECT year, quarter, month, division, brand, item, SUM(sales)  
    sum_sales,  
    GROUPING_ID(grouping-columns) gid  
FROM sales, products, time  
WHERE join-condition  
GROUP BY ROLLUP(year, quarter, month),  
ROLLUP(division, brand, item))  
WHERE division = 25 AND month = 200201 AND gid = gid-for-  
Division-Month;
```

- N'affiche que les régions du cube souhaitées

## 6.13 : Utilisation des cubes hiérarchiques

- Optimisation : la requête est transformée comme ci-dessous :

```
SELECT month, division, sum_sales
FROM (SELECT null, null, month, division, null, null, SUM(sales) sum_sales,
GROUPING_ID(grouping-columns) gid
FROM sales, products, time WHERE join-condition
GROUP BY month, division)
WHERE division = 25 AND month = 200201 AND gid = gid-for-Division-Month;
```