

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer les composants d'accès
aux données »

SQL Server et Transact SQL
Le langage DML partie2
(select)

Apprentissage

Mise en pratique

Evaluation

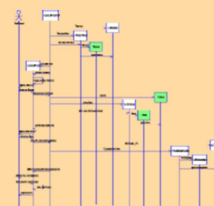
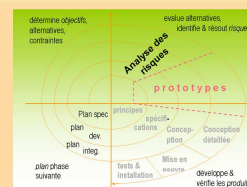
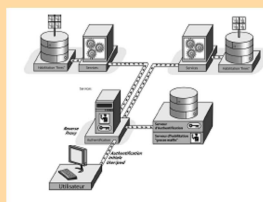


TABLE DES MATIERES

Table des matières	3
1. L'ORDRE SELECT	7
1.1 Syntaxe.....	7
1.2 Select avec expression	9
1.3 Select Option DISTINCT	10
1.4 La clause TOP	11
1.5 Select conditionnelle avec WHERE	12
a) Les opérateurs de comparaison	13
b) Le prédicat BETWEEN	14
c) La clause IN	15
d) L'opérateur LIKE	16
e) Le prédicat IS NULL	17
1.6 Les fonctions.....	17
a) Les fonctions d'agrégation	18
b) Les fonctions scalaires mathématiques	19
c) Les fonctions scalaires de chaîne	20
d) Les fonctions scalaires de date et heure	21
e) Les fonctions de conversion	23
f) Les fonctions de classement	24
1.7 Les autres clauses du SELECT.....	26
a) La clause ORDER BY	26
b) La clause GROUP BY simple	28
c) La clause HAVING	30
2. L'opération de JOINTURE	33
a) Utilisation des jointures internes	33
b) Utilisation des jointures externes gauche ou droite	35
c) Utilisation des jointures croisées	36
d) Jointure de plus de deux tables	37
3. POUR ALLER PLUS LOIN.....	38
3.1 L'opérateur UNION	38
3.2 Les sous requêtes.....	40
e) Sous requête imbriquée : Renvoi d'une valeur unique	40
f) Sous requête imbriquée : Renvoi d'une liste de valeurs	41

Le langage DML partie 2 – SQL server

g)	Sous requête EN CORRELATION	43
h)	Sous requête EN CORRELATION : Utilisation de EXISTS et NOT EXISTS	44
3.3	La clause GROUP BY générale	45
3.4	Les opérateurs EXCEPT ET INTERSECT	48

Objectifs

Après étude de ce document, le stagiaire sera capable de traduire en langage SQL un besoin de sélection de données stockées en tables de bases de données relationnelle.

Pré requis

Connaissance de base des SGBD relationnelles comme les tables, attributs ou champs, domaine et types de données.

Outils de développement

SQL Server Management Studio (version 2008 minimum).

Méthodologie

SQL Server permet d'effectuer l'interrogation des données soit directement à travers SQL Server Management Studio (menu contextuel, 'Sélectionner les 1000 lignes du haut'), soit à travers la commande de sélection de données SQL spécifique 'select' du LMD SQL.

Cette commande 'select' permet d'affiner précisément le résultat obtenu et de reconstituer l'information globale disséminée dans les différentes tables (notion de jointure). Pour ce faire, la commande SQL 'select' présente de nombreuses variantes et options. Ce document couvre l'ensemble des variantes et options de la commande 'select' mais tout cela demande un peu d'entraînement avant de pouvoir traduire correctement un besoin de sélection de données en ordres SQL.

Nous vous conseillons de parcourir une première fois le document dans son ensemble, du moins pour les chapitres 1 et 2, puis de rechercher au cas par cas les informations nécessaires lors de la résolution de requêtes.

Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

Ressources

L'aide en ligne Microsoft de référence accessible facilement par la touche F1 dans SQL Server Management Studio.

Lectures conseillées

1. L'ORDRE SELECT

1.1 SYNTAXE

Cette commande permet la **RECHERCHE** d'informations en sélectionnant les données selon divers critères.

```
SELECT <NOMS DE COLONNES OU EXPRESSIONS>
FROM   <NOMS DE TABLES>
WHERE  <CONDITIONS DE RECHERCHE>

      GROUP BY  <NOMS DE COLONNE DU SELECT>
      HAVING    <CONDITION DE RECHERCHE>
      ORDER BY  <NOM OU POSITION DE COLONNE
                DANS L'ORDRE SELECT>
```

Le résultat de la commande **SELECT** est une table, sous-ensemble de la (ou des) table(s) de départ :

- dont les colonnes dépendent des rubriques citées après **SELECT** (colonnes directement issues de la table d'origine, valeurs calculées, constantes, etc. ...)
- dont les lignes satisfont tant par leur contenu que pour leur présentation, aux options suivant, le cas échéant, le nom de la table.

EXEMPLE : Lister le contenu de la table EMPLOYES

```
SELECT * FROM EMPLOYES
```

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00010	PAUL	BALZAC	A00	1200
00020	ANNE	MARTIN	B00	1500
00030	ANDRE	BOULIN	C01	2100
00040	PIERRE	DURAND	E01	1530
00050	MARIE	VIALA	A01	1650
00120	JULIEN	DELMAS	E01	1834
00110	JEAN	CAZENEUVE	C01	
00160	ARTHUR	ZOLA	B00	1600

L'astérisque (*) inséré dans la liste de sélection permet d'extraire toutes les colonnes d'une table.

EXEMPLE : Afficher les noms des employés contenus dans la table employés.

```
SELECT NOM FROM EMPLOYES
```

NOM

BALZAC

MARTIN

BOULIN

DURAND

VIALA

DELMAS

CAZENEUVE

ZOLA

EXEMPLE : Lister le nom et le salaire des employés de la table EMPLOYES.

```
SELECT NOM, SALAIRE FROM EMPLOYES
```

NOM

SALAIRE

BALZAC 1200

MARTIN 1500

BOULIN 2100

DURAND 1530

VIALA 1650

DELMAS 1834

CAZENEUVE

ZOLA 1600

1.2 SELECT AVEC EXPRESSION

Une ou plusieurs expressions peuvent suivre le mot-clé `SELECT`. Une expression spécifie un mode de calcul.

Elle peut être :

- Un nom de colonne
- Une constante (numérique ou chaîne de caractères)
- Une fonction (`CURRENT_TIMESTAMP`, `GETDATE()` ...)
- Une combinaison de ces valeurs séparées :
 - par l'opérateur de concaténation `+` (chaînes de caractères)
 - par l'un des opérateurs arithmétiques binaires (`*`, `/`, `+`, `-` évalués dans cet ordre) ou unaires (`+`, `-`) (numériques).Si l'un des opérandes a la valeur `null`, le résultat de l'expression est `null`.
- Une suite d'expressions séparées par des parenthèses

Exemples :

```
'NOM : '
```

```
PRIX *QUANTITE
```

```
(PRIX *1.183) / (COLONNE_1 + COLONNE_2)
```

EXEMPLE : Lister le nom et le salaire en centimes de chaque employé.

```
SELECT NOM + ' ' + PRENOM, 'SALAIRE : ',  
       SALAIRE * 100, 'CENTIMES '  
FROM EMPLOYES
```

Expr1	Expr2	Expr3	Expr4
-----	-----	-----	-----
BALZAC PAUL	SALAIRE :	120000	CENTIMES
MARTIN ANNE	SALAIRE :	150000	CENTIMES
BOULIN ANDRE	SALAIRE :	210000	CENTIMES
DURAND PIERRE	SALAIRE :	153000	CENTIMES
VIALA MARIE	SALAIRE :	165000	CENTIMES
DELMAS JULIEN	SALAIRE :	183400	CENTIMES
CAZENEUVE JEAN	SALAIRE :		CENTIMES
ZOLA ARTHUR	SALAIRE :	160000	CENTIMES

La lisibilité des noms de colonne peut être améliorée en utilisant le mot clé AS : les noms de colonne par défaut seront remplacés par des **alias** dans la liste de sélection.

```
SELECT NOM + ' ' + PRENOM AS 'NOM DU SALARIE',
       SALAIRE * 100 AS SALAIRE, 'CENTIMES ' as 'CENTIMES'
FROM EMPLOYES
```

NOM DU SALARIE	SALAIRE	CENTIMES
-----	-----	-----
BALZAC PAUL	1200000	CENTIMES
MARTIN ANNE	1500000	CENTIMES
BOULIN ANDRE	2100000	CENTIMES
DURAND PIERRE	1530000	CENTIMES
.....

1.3 SELECT OPTION DISTINCT

```
SELECT [ALL] nom coll FROM nomtable
```

par opposition à

```
SELECT DISTINCT nomcoll FROM nomtable
```

L'option **ALL** est prise par défaut, toutes les lignes sélectionnées figurent dans le résultat.

L'option **DISTINCT** permet de ne conserver qu'un exemplaire de chaque ligne en double.

```
SELECT WDEPT FROM EMPLOYES      SELECT DISTINCT WDEPT FROM EMPLOYES
```

WDEPT	WDEPT
-----	-----
A00	A00
B00	B00
C01	C01
E01	E01
A01	A01
E01	
C01	
B00	

1.4 LA CLAUSE TOP

SELECT TOP n nom col1, nom col2 **FROM** nomtable

Exemple 1 : Lister les 5 premiers employés de la table **EMPLOYES**.

SELECT TOP 5 NOM, PRENOM **FROM** **EMPLOYES**

NOM	PRENOM
-----	-----
BALZAC	PAUL
MARTIN	ANNE
BOULIN	ANDRE
DURAND	PIERRE
VIALA	MARIE

Exemple 2 : Lister 20% de la table **EMPLOYES**.

SELECT TOP 20 PERCENT NOM, PRENOM **FROM** **EMPLOYES**

NOM	PRENOM
-----	-----
BALZAC	PAUL
MARTIN	ANNE

Depuis la version 2005, *n* peut être variable :

En utilisant les compléments de Transact SQL, l'exemple 1 peut être codé :

```
declare @n bigint
Set @n=5
SELECT TOP (@n) NOM, PRENOM FROM EMPLOYES
```

L'exemple 2 peut aussi être codé :

```
declare @n bigint
Set @n=20
SELECT TOP (@n) PERCENT NOM, PRENOM FROM EMPLOYES
```

La clause TOP peut accepter n'importe quelle expression scalaire ou expression plus élaborée telle que le résultat d'une sous-requête ou une fonction utilisateur UDF retournant une valeur scalaire.

1.5 SELECT CONDITIONNELLE AVEC WHERE

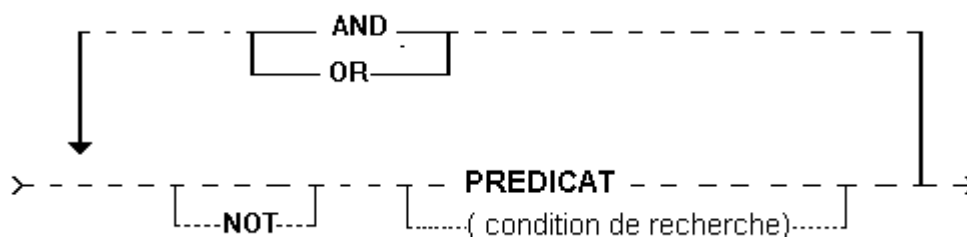
La clause **WHERE** permet de préciser les conditions de recherche sur les lignes de la table.

Les conditions peuvent contenir une liste illimitée de prédicats – expressions renvoyant la valeur TRUE (vrai), FALSE (faux) ou UNKNOWN (inconnu) -, combinés à l'aide des opérateurs logiques **AND** ou **OR** .

Exemples de prédicats :

```
SALAIRE = 15000  
NOM = 'DUPONT'  
NODEPT = NOGROUP
```

Conditions de recherche :



Exemples :

```
PRIX < 100.00 OR PRIX > 135.00  
NOT (AUTEUR = 'DUMAS')
```

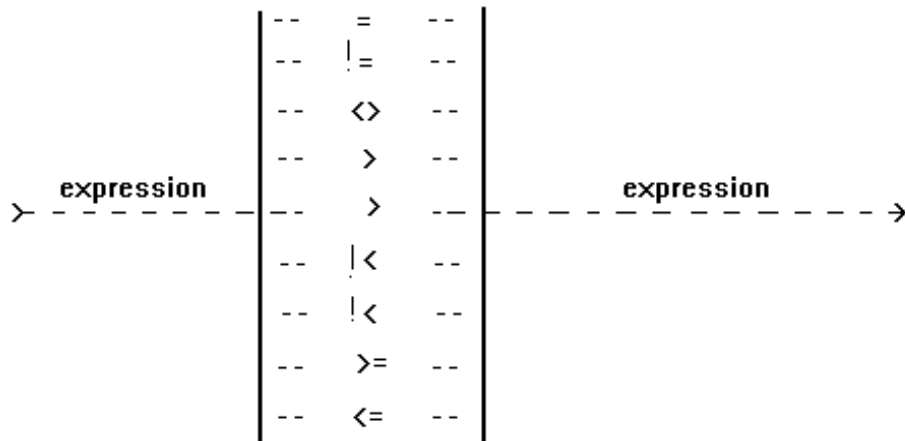
Conditions de recherche de la clause WHERE

Pour spécifier la condition de recherche dans la clause **WHERE**, on utilise indifféremment l'un des opérateurs conditionnels ci-après :

Description	Opérateurs conditionnels
Opérateurs de comparaison	=, <>, !=, >, >=, !>, <, <=, !<
Comparaisons de plage	BETWEEN et NOT BETWEEN
Comparaisons de listes	IN et NOT IN
Comparaisons de chaîne de caractères	LIKE et NOT LIKE
Valeurs inconnues	IS NULL et IS NOT NULL

!=, !>, !< sont des opérateurs non conformes à la norme SQL-92

a) Les opérateurs de comparaison



- Les 2 expressions doivent être de même type
- Les nombres sont comparés selon leur valeur algébrique (conversion)
- Les chaînes de caractères sont comparées de gauche à droite.
- Les données de type *char*, *nchar*, *varchar*, *nvarchar*, *text*, *datetime* et *smalldatetime* doivent être encadrées par des apostrophes.

Quelques exemples :

Rechercher dans la table EMPLOYES, les données concernant les employés qui travaillent dans le département A00 ou E01

```
SELECT * FROM EMPLOYES WHERE WDEPT = 'A00' OR WDEPT = 'E01'
```

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00010	PAUL	BALZAC	A00	1200
00040	PIERRE	DURAND	E01	1530
00120	JULIEN	DELMAS	E01	1834

Rechercher dans la table EMPLOYES, les données concernant les employés dont le matricule est supérieure à 00110 et dont le salaire est égal à 1834

```
SELECT NOM, SALAIRE, NOEMP FROM EMPLOYES
WHERE NOEMP > '00110' AND SALAIRE = 1834
```

NOM	SALAIRE	NOEMP
-----	-----	-----
DELMAS	18345	00120

Rechercher dans la table **EMPLOYES**, les données concernant les employés dont le matricule est supérieur ou égal à 00060 et dont le salaire est égal à 1834 ou 1700

```
SELECT * FROM EMPLOYES
WHERE (SALAIRE = 1834 OR SALAIRE = 1700)
AND NOEMP >= '00060'
```

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
-----	-----	-----	-----	-----
00120	JULIEN	DELMAS	E01	1834

b) Le prédicat **BETWEEN**

L'opérateur BETWEEN de la clause WHERE permet d'extraire des lignes appartenant à une plage de valeurs donnée.

Lister le prénom et le nom des employés dont le salaire est compris entre 12000 et 16000 francs.

```
SELECT PRENOM, NOM FROM EMPLOYES
WHERE SALAIRE BETWEEN 1200 AND 1600
```

PRENOM	NOM	SALAIRE
-----	-----	-----
PAUL	BALZAC	1200
ANNE	MARTIN	1500
PIERRE	DURAND	1530
ARTHUR	ZOLA	1600

- **BETWEEN** précise les bornes (comprises) entre lesquelles s'effectuera la sélection.
 - **NOT BETWEEN** précise les bornes en dehors desquelles s'effectuera la sélection.
- Attention : les conditions négatives ralentissent l'extraction des données.

On aurait pu écrire :

```
SELECT PRENOM, NOM FROM EMPLOYES
      WHERE SALAIRE > = 1200 AND SALAIRE < = 1600.
```

c) La clause **IN**

L'opérateur **IN** de la clause **WHERE** permet d'extraire des lignes correspondant à une liste de valeurs donnée.

Exemple 1:

```
SELECT * FROM EMPLOYES
      WHERE NOEMP IN ('00010', '00020', '00050' '00100')
```

Exemple 2:

```
SELECT * FROM EMPLOYES
      WHERE WDEPT IN (1, 2, 3, 4, 5)
ou
SELECT * FROM EMPLOYES
      WHERE WDEPT BETWEEN 1 AND 5
ou
SELECT * FROM EMPLOYES
      WHERE WDEPT >= 1 AND WDEPT <= 5
```

NB : On peut aussi utiliser **NOT IN**

Attention : les conditions négatives ralentissent l'extraction des données.

d) L'opérateur **LIKE**

L'opérateur **LIKE** de la clause **WHERE** conjointement aux caractères génériques, permet de comparer des chaînes de caractères *inexactes*.

Pour une comparaison de chaîne exacte, remplacer l'opérateur **LIKE** par un opérateur de comparaison, par exemple, utiliser **NOM = 'BALZAC'** plutôt que **NOM LIKE 'BALZAC'**

LIKE ne peut être utilisé qu'avec des données de type *char*, *nchar*, *varchar*, *nvarchar* ou *datetime*

Types de caractères génériques

Caractères génériques	Description
%	N'importe quelle chaîne comprise entre zéro et plusieurs caractères
-(trait de soulignement)	N'importe quel caractère unique
[]	N'importe quel caractère unique dans la plage (par exemple [s-w] ou [aeiouy])
[^]	N'importe quel caractère unique n'appartenant pas à la plage (par exemple [^s-w] ou [^aeiouy])

Exemple: Lister les données de la table **EMPLOYES** dont le nom commence par la lettre "B"

```
SELECT * FROM EMPLOYES
WHERE NOM LIKE 'B %'
```

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00010	PAUL	BALZAC	A00	1200
00030		BOULIN	C01	2100

LIKE 'BAL%'	tous les noms commençant par les lettres BAL
LIKE '%BAL%'	tous les noms contenant les lettres BAL
LIKE '--LZ--'	tous les noms de 6 caractères contenant LZ en 3ème et 4ème positions
LIKE '[S-V]ENT'	tous les noms de 4 lettres se terminant par les lettres ENT et commençant par n'importe quelle lettre comprise entre S et V.

NOT LIKE inverse de **LIKE**

e) Le prédicat **IS NULL**

L'opérateur **IS NULL** de la clause **WHERE** est utilisé pour extraire des lignes pour lesquelles il manque des informations dans une colonne donnée.

Une colonne prend la valeur **NULL** si aucune valeur n'y est entrée au moment de la saisie des données et si aucune valeur par défaut n'a été définie pour cette colonne (c'est dans l'instruction **CREATE TABLE** que les colonnes sont autorisées à recevoir des valeurs **NULL**).

Exemple: Lister le prénom et le nom des employés dont on ne connaît pas le salaire.

```
SELECT PRENOM, NOM FROM EMPLOYES
      WHERE SALAIRE IS NULL
```

PRENOM	NOM
-----	-----
JEAN	CASENEUVE

Le prédicat **IS (NOT) null** permet de tester la ou les valeurs **NULL** contenues dans une colonne.

1.6 LES FONCTIONS

SQL offre la possibilité d'intégrer dans les expressions des fonctions retournant :

- Une valeur dépendant du contenu de colonnes (fonction sur les colonnes)
- Une valeur dépendant d'opérandes (fonctions scalaires)

NB : à utiliser avec précaution ! Dans ce domaine des fonctions intégrées chaque SGBD apporte son lot d'extensions par rapport aux standards du langage SQL

Exemple 1: Lister le salaire maximum du département E01

```
SELECT MAX (SALAIRE) FROM EMPLOYES
      WHERE WDEPT = 'E01'
```

Lorsqu'une fonction d'agrégation est exécutée, SQL effectue la synthèse des valeurs d'une colonne spécifique pour la table complète ou pour des groupes de lignes de la table (clause **GROUP BY**) : une valeur d'agrégation unique est alors générée pour la table complète ou pour chaque groupe de lignes.

Exemple 2: Lister le nombre de caractères du nom des employés.

```
SELECT LEN(NOM) FROM EMPLOYES
```

Les fonctions scalaires effectuent une opération sur une valeur unique et renvoie ensuite une valeur unique. Elles peuvent être utilisées pour autant que l'expression est valide.

a) Les fonctions d'agrégation

Les fonctions d'agrégation effectuent un calcul sur un ensemble de valeurs et renvoient une valeur unique. A l'exception de la fonction `COUNT (*)`, les fonctions d'agrégation ignorent les valeurs `NULL`.

Pour les fonctions pour lesquelles ces valeurs sont précisées :

ALL applique la fonction d'agrégation à toutes les valeurs. (**ALL** est l'argument par défaut)

DISTINCT spécifie que la fonction doit seulement être appliquée à chaque instance unique d'une valeur, quel que soit le nombre d'occurrences de la valeur.

AVG ([**ALL** | **DISTINCT**] *expr*)

Calcul de la moyenne des valeurs de la collection de nombres précisés par l'expression entre parenthèses.

```
SELECT AVG(SALAIRE) FROM EMPLOYES
```

```
SELECT AVG(DISTINCT SALAIRE) FROM EMPLOYES
```

SUM ([**ALL** | **DISTINCT**] *expr*)

Somme des valeurs des nombres d'une colonne de type numérique

```
SELECT SUM(SALAIRE) FROM EMPLOYES
WHERE WDEPT = 'E01'
```

MAX(*expr*) et **MIN**(*expr*)

Obtention de la valeur maximum (minimum) d'une collection de valeurs.

COUNT (*)

Dénombrement d'une collection de lignes

```
SELECT COUNT ( * ) FROM EMPLOYES
```

La fonction `COUNT` renvoie le nombre d'employés.

La fonction `COUNT (*)` ne requiert aucun paramètre et calcule le nombre de lignes de la table spécifiée sans supprimer les doublons. Elle compte chaque ligne séparément, même les lignes contenant des valeurs `NULL`.

COUNT ([**ALL** | **DISTINCT**] *expr*)

Dénombrement de toutes les expressions non nulles ou non nulles uniques

```
SELECT COUNT (DISTINCT WDEPT) FROM EMPLOYES
```

La fonction `COUNT` renvoie le nombre de département non nuls uniques.

Le langage DML partie 2 – SQL server

Afpa © 2015 – Section Tertiaire Informatique – Filière « Etude et développement »

COUNT_BIG ([ALL | DISTINCT] expr)

Idem count, mais la valeur de retour est au format bigint au lieu de int.

VAR (expr)

Renvoie la variance de toutes les valeurs de l'expression numérique donnée

STDEV(expr)

Renvoie l'écart type de toutes les valeurs de l'expression numérique donnée

b) Les fonctions scalaires mathématiques

Les fonctions scalaires effectuent une opération sur une valeur ou sur un nombre fixe de valeurs, et non sur une collection de valeurs.

*Les fonctions scalaires **mathématiques** effectuent un calcul, généralement basé sur les valeurs d'entrée transmises comme arguments et elles renvoient une valeur numérique*

Mathématiques:

ABS (expr) valeur absolue

CEILING (expr), **FLOOR**(expr)

Plus petit (grand) entier supérieur(inférieur) ou égal à *expr*

SIGN (expr)

Renvoie 1 si *expr* positive, -1 si *expr* est négative, 0 si nulle

SQRT (expr) racine carrée

POWER (expr ,n) élève *expr* à la puissance n

SQUARE (expr) calcul le carré de *expr*

Trigonométriques :

SIN(expr), **COS**(expr), **TAN**(expr), **COTAN**(expr)

ASIN(expr), **ACOS**(expr), **ATAN**(expr)

PI() renvoie la valeur PI : 3.14159265358979

DEGREES(expr), **RADIANS**(expr)

Conversion de degrés (radians) en radians (degrés)

Logarithmiques :

LOG(expr), **EXP**(expr), **LOG10**(expr)

Diverses:

RAND (expr) rend un nombre aléatoire entre 0 et 1 , *expr* représente la valeur de départ

ROUND (expr , n) arrondit *expr* à la précision n

c) Les fonctions scalaires de chaîne

Les fonctions scalaires **de chaîne** effectuent une opération sur une valeur d'entrée de type chaîne et renvoient une valeur numérique ou de type chaîne.

CHAR (expr), **NCHAR** (expr)

Convertit *expr* en un caractère ASCII ou un caractère UNICODE

CHAR(65) donne A

ASCII (expr), **UNICODE** (expr)

Renvoie le code ASCII ou unicode de *expr*

ASCII (A) donne 65

LEN (expr)

Renvoie le nombre de caractères d'une expression de chaîne

LEN ('MARTIN') donne 6

LOWER (expr), **UPPER** (expr)

Renvoie *expr* après avoir transformé les caractères majuscules en caractères minuscules, ou monuscules en majuscules

LOWER ('MARTIN') donne martin

UPPER ('martin') donne MARTIN

LEFT (expr, n) **RIGHT** (expr, n)

Renvoie les n caractères les plus à gauche (droite) de *expr*.

LEFT ('MARTIN', 2) donne MA

RIGHT ('MARTIN', 2) donne IN

LTRIM (expr), **RTRIM** (expr)

Supprime les espaces à gauche (droite) de *expr*.

LTRIM (' MARTIN') donne MARTIN

RTRIM ('MARTIN ') donne MARTIN

SUBSTRING (expr, p, n)

Renvoie *n* caractères de la chaîne. *expr* à partir de la position spécifiée par *p*

SUBSTRING ('HUGO', 2, 3) donne 'UGO'

STUFF (*expr1*, *p*, *n*, *expr2*)

Supprime *n* caractères de *expr1* à partir d'une position donnée et insère *expr2*

STUFF ('ABCDEF', 2, 3, 'IJKLMN') renvoie AIJKLMNEF

REPLACE (*expr1*, *expr2*, *expr3*)

Remplace dans *expr1* toutes les occurrences de *expr2* par *expr3*

REPLACE ('ABCDAB', 'AB', 'IJK') renvoie IJKCDIJK

ISNULL (*expr1*, *expr2*)

Remplace un NULL détecté dans *expr1* par *expr2*

ISNULL (salaire, ' ') renvoie ' ' à la place de NULL chaque fois que null est détecté dans la colonne salaire

d) Les fonctions scalaires de date et heure

Les fonctions scalaires **de date et heure** effectuent une opération sur une valeur d'entrée de type date et heure et renvoient soit une valeur numérique, de type date ou heure, ou de type chaîne.

CURRENT_TIMESTAMP, **GETDATE** ()

Renvoie la date et l'heure courante

DAY (*expr*), **MONTH** (*expr*), **YEAR** (*expr*)

Obtention d'un jour, mois ou année à partir de *expr*

DAY (CURRENT_TIMESTAMP) donne le jour courant

YEAR (CURRENT_TIMESTAMP) extrait l'année courante

Les fonctions **DAY**, **MONTH** et **YEAR** sont synonymes de

DATEPART (*dd*, *date*), **DATEPART** (*mm*, *date*) et **DATEPART** (*yy*, *date*), respectivement.

Dans les fonctions suivantes, le paramètre *partie_de_date* indique la partie de date à renvoyer suivant les abréviations :

Partie de date	Nom Complet	Abréviation
Année	Year	yy, yyyy
Quart	Quart	qq, q
Mois	Month	mm, m
Quantième	DayofYear	dy
Jour	Day	d, dd
Semaine	Week	wk, ww
Jour de la semaine	WeekDay	dw
Heure	Hour	hh
Minute	Minute	mi
Secondes	Second	ss, s
Millisecondes	Millisecond	ms

DATEPART (partie_de_date, date)

Renvoie un entier représentant l'élément de date précisé dans la date spécifiée.

DATEPART (mm, CURRENT_TIMESTAMP) donne le mois courant
équivalent à

DATEPART (month, CURRENT_TIMESTAMP)

DATEPART (mm, '21/06/2001') renvoie 6

DATEPART (dw, '21/06/2001') renvoie 4

DATENAME (partie_de_date, date)

Renvoie le nom de l'élément de date précisé dans la date spécifiée.

DATENAME (mm, '21/06/2001') renvoie 'juin'

DATENAME (dw, '21/06/2001') renvoie 'jeudi'

DATEADD (partie_de_date, nombre, date)

Renvoie une nouvelle valeur **datetime** calculée en ajoutant un intervalle à la date spécifiée.

DATEADD (dd, 15, '21/06/2001') renvoie '06/07/2001'

DATEADD (mm, 7, '21/06/2001') renvoie '21/01/2002'

DATEDIFF (partie_de_date, date_début , date_fin)

Renvoie un entier spécifiant le nombre d'intervalles compris entre date_début et date_fin.

DATEDIFF (dd, '21/06/2001', 01/07/2001') renvoie 10

DATEDIFF (mm, '21/06/2001', '01/01/2002') renvoie 7

Il est possible de configurer le premier jour de la semaine par l'intermédiaire de la fonction

SET DATEFIRST (numero_jour)

Les jours sont numérotés de 1 pour le lundi à 7 pour le dimanche ; il est possible de connaître la configuration en interrogeant la fonction @@datefirst.

e) Les fonctions de conversion

Conversion explicite d'une expression d'un type de données en une expression d'un type de données différent.

CAST et CONVERT offrent la même fonctionnalité

CAST (expression as type_de_donnée)

Permet de convertir une valeur dans le type spécifié.

CAST (DATEPART (hh, DATE) as char(2))

renvoie le nombre d'heures sous forme d'une chaîne de caractères de longueur 2, extraite d'un champ DATE de type datetime.

CONVERT (type_de_donnée, expr, style)

Permet de convertir l'expression dans le type spécifié, avec la possibilité d'utilisation d'un style (facultatif)

CONVERT (char, DATE, 8)

renvoie sous la forme d'une chaîne de caractères (hh:mm:ss) l'heure d'un champ DATE de type datetime avec application du style 8

Sans siècle (aa)	Avec siècle (aaaa)	Standard	Entrée/Sortie
1	101	États-Unis	mm/jj/aaaa
2	102	ANSI	aa.mm.jj
3	103	Anglais/Français	jj/mm/aa
4	104	Allemand	jj.mm.aa
5	105	Italien	jj-mm-aa
6	106 ⁽¹⁾	-	jj mois aa
7	107 ⁽¹⁾	-	mois jj, aa
8	108	-	hh:mm:ss

Extrait de styles (Voir aide en ligne pour plus d'informations)

f) Les fonctions de classement

La fonction **ROW_NUMBER ()** permet de numéroté séquentiellement chaque ligne rendue par une instruction Select, sur la base d'un critère donné, codé sur une clause Over.

```
SELECT ROW_NUMBER ( ) OVER (ORDER BY SALAIRE DESC) AS  
NUMCOL, PRENOM, NOM, SALAIRE FROM EMPLOYES
```

NUMCOL	PRENOM	NOM	SALAIRE
1	ANDRE	BOULIN	2100
2	JULIEN	DELMAS	1834
3	MARIE	VIALA	1650
4	ARTHUR	ZOLA	1600
5	PIERRE	DURAND	1530
6	ANNE	MARTIN	1500
7	PAUL	BALZAC	1200
8	JEAN	CAZENEUVE	

On ne peut malheureusement pas faire de la sélection sur la colonne obtenue (clause **WHERE**) pour limiter le nombre de lignes renvoyées, en une seule instruction. Une solution est de passer par l'intermédiaire d'une table temporaire, au moyen d'une procédure stockée.

RANK ()

Avec la fonction **RANK**, le résultat obtenu est le même qu'avec **ROW_NUMBER** à la différence qu'une ligne peut se voir attribuer le même numéro (n) que la ligne qui la précède, la ligne suivante prenant le numéro (n+2).

DENSE_RANK ()

Le résultat obtenu est le même qu'avec **RANK** à la différence que la ligne suivant les 2 lignes dont le numéro de colonne est égal prendra le numéro (n+1).

NTILE(n)

Cette fonction permet de numérotter les lignes par paquet de n éléments.

```
SELECT  NTILE  (3)  OVER  (ORDER  BY  SALAIRE  DESC)  AS  
NUMCOL, PRENOM, NOM, SALAIRE FROM EMPLOYES
```

NUMCOL	PRENOM	NOM	SALAIRE
1	ANDRE	BOULIN	2100
1	JULIEN	DELMAS	1834
1	MARIE	VIALA	1650
2	ARTHUR	ZOLA	1600
2	PIERRE	DURAND	1530
2	ANNE	MARTIN	1500
3	PAUL	BALZAC	1200
3	JEAN	CAZENEUVE	

1.7 LES AUTRES CLAUSES DU SELECT

a) La clause ORDER BY

La clause ORDER BY permet de préciser une séquence de tri pour le résultat d'une requête.

- ASC séquence croissante (valeur par défaut)
- DESC séquence décroissante

Exemple 1: Lister le contenu de la table employé, trié par département croissant et nom décroissant

```
SELECT *FROM EMPLOYES  
      ORDER BY WDEPT ASC, NOM DESC
```

Equivalent à:

```
SELECT * FROM EMPLOYES  
      ORDER BY WDEPT, NOM DESC
```

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
-----	-----	-----	-----	-----
00140	HUBERT	REEVES	A00	2100
00130	PHILIPPE	LABRO	A00	1520
00010	PAUL	BALZAC	A00	1200
00060	REGINE	WIRTH	A01	1700
00160	ARTHUR	ZOLA	B00	1600
00020	ANNE	MARTIN	B00	1500
00030	ANDRE	BOULIN	C01	2100
00050	MARIE	VIALA	E01	1650
00040	PIERRE	DURAND	E01	1530
00120	JULIEN	DELMAS	E01	1834
00150	MARIE	CURIE	E01	3400

La clause ORDER BY doit être la dernière clause dans l'ordre SELECT et peut être spécifiée avec n'importe quelle colonne.

Exemple 2: Lister par salaire décroissant et par département, le salaire en centimes, le prénom, le nom, et le département des employés.

```
SELECT SALAIRE*100, PRENOM, NOM, WDEPT FROM EMPLOYES
ORDER BY 1 DESC, WDEPT
```

	PRENOM	NOM	WDEPT
-----	-----	-----	-----
	JEAN	CAZENEUVE	A00
300000	MARIE	CURIE	E01
210000	HUBERT	REEVES	A00
210000	REGINE	WIRTH	A01
183450	ANNE	MARTIN	B00
170000	ANDRE	BOULIN	C01
165000	MARIE	VIALA	E01
160000	ARTHUR	ZOLA	B00
153000	PIERRE	DURAND	E01
152000	PHILIPPE	LABRO	A00
150000	JULIEN	DELMAS	E01
120000	PAUL	BALZAC	A00

La valeur NULL occupe la position la plus élevée.

b) La clause GROUP BY simple

Ces options permettent de définir et de traiter des groupes afin de réaliser des sous-totaux par exemple.

Un groupe est formé à partir d'un ensemble de lignes d'une table ayant une ou plusieurs caractéristiques communes.

L'intérêt d'un groupe est de conserver la trace des éléments qu'il contient, par exemple pour les dénombrer ou effectuer des opérations telles que somme ou moyenne.

Exemple : Quel est le salaire moyen et le salaire minimum des employés à l'intérieur de chaque département pour les n° employés > 00010.

```
SELECT WDEPT, AVG (SALAIRE), MIN (SALAIRE) FROM EMPLOYES
WHERE NOEMP > 00010
GROUP BY WDEPT
```

WDEPT	Expr1	Expr2
-----	-----	-----
A01	1650	1650
B00	1550	1500
C01	2100	2100
E01	1682	1530

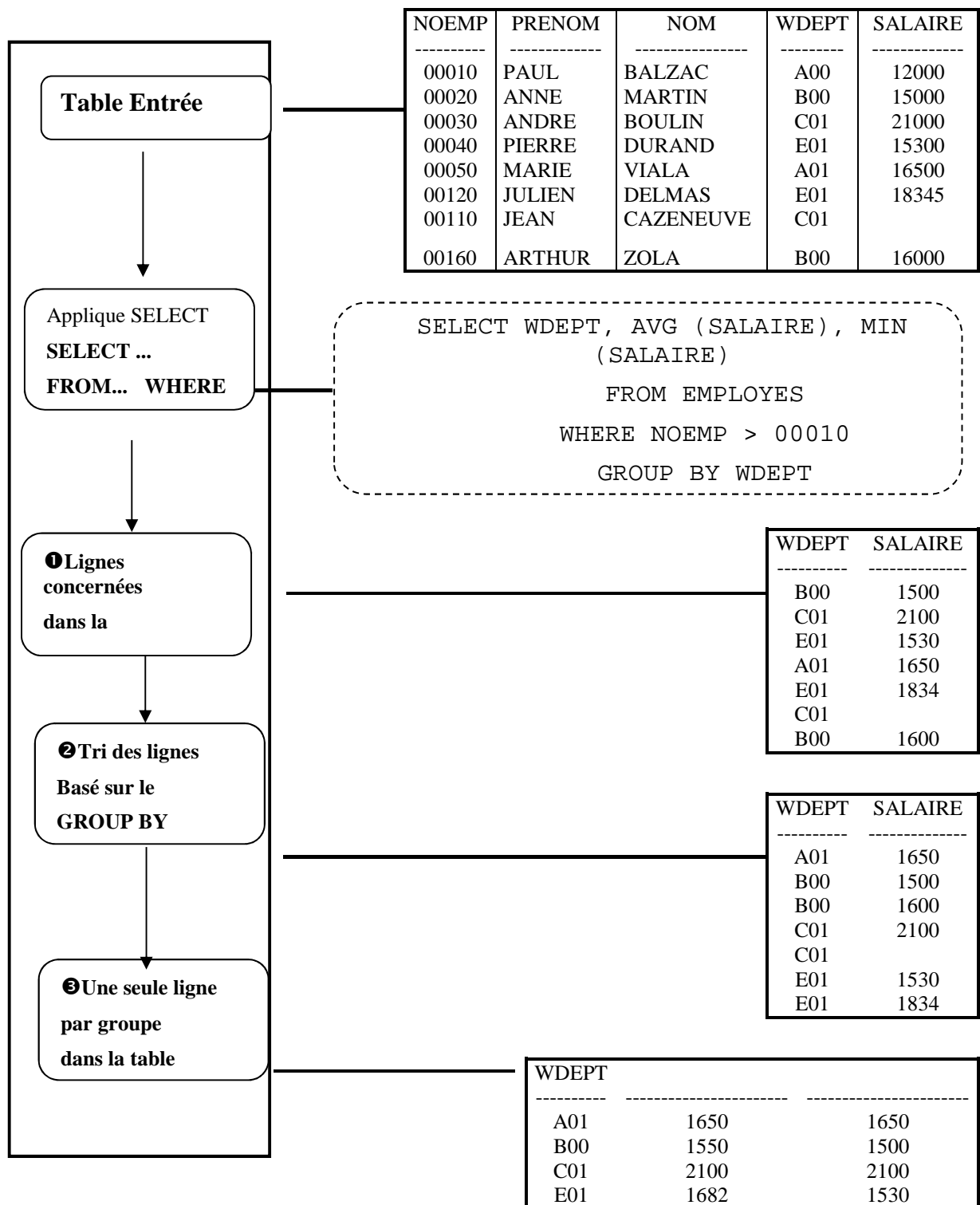
Il y a autant de groupes que de valeurs de WDEPT distinctes.

Ces groupes apparaissent en séquence croissante car un classement est nécessaire en interne pour constituer les groupes.

GROUP BY est suivi du nom d'une ou plusieurs colonnes présentes dans le **SELECT** appelées colonnes de regroupement.

La liste de colonnes suivant **SELECT** ne peut comporter que les noms des colonnes de regroupement, ou des noms de fonctions.

Analyse du GROUP BY



❶ Les colonnes WDEPT et SALAIRE sont sélectionnés seulement pour les N° d'employés supérieurs à 00010.

❷ Ces lignes sont triées dans la séquence GROUP BY

❸ La moyenne et le Min des salaires sont calculés, et une ligne par groupe est générée.

c) La clause HAVING

La clause HAVING est utilisée en conjonction avec la clause GROUP BY.

La clause HAVING agit comme critère de *sélection pour les groupes renvoyés avec la clause GROUP BY*.

Exemple 1 Quel est le salaire moyen et le salaire minimum des employés à l'intérieur de chaque département pour les n° employés > 00010 ?

Lister uniquement les groupes pour lesquels la moyenne est supérieure à 16 000

```
SELECT WDEPT, AVG (SALAIRE), MIN (SALAIRE) FROM EMPLOYES
WHERE NOEMP > 00010
GROUP BY WDEPT
HAVING AVG (SALAIRE) >= 16000
```

WDEPT

WDEPT	AVG (SALAIRE)	MIN (SALAIRE)
A01	1650	1650
C01	2100	2100
E01	1682	1530

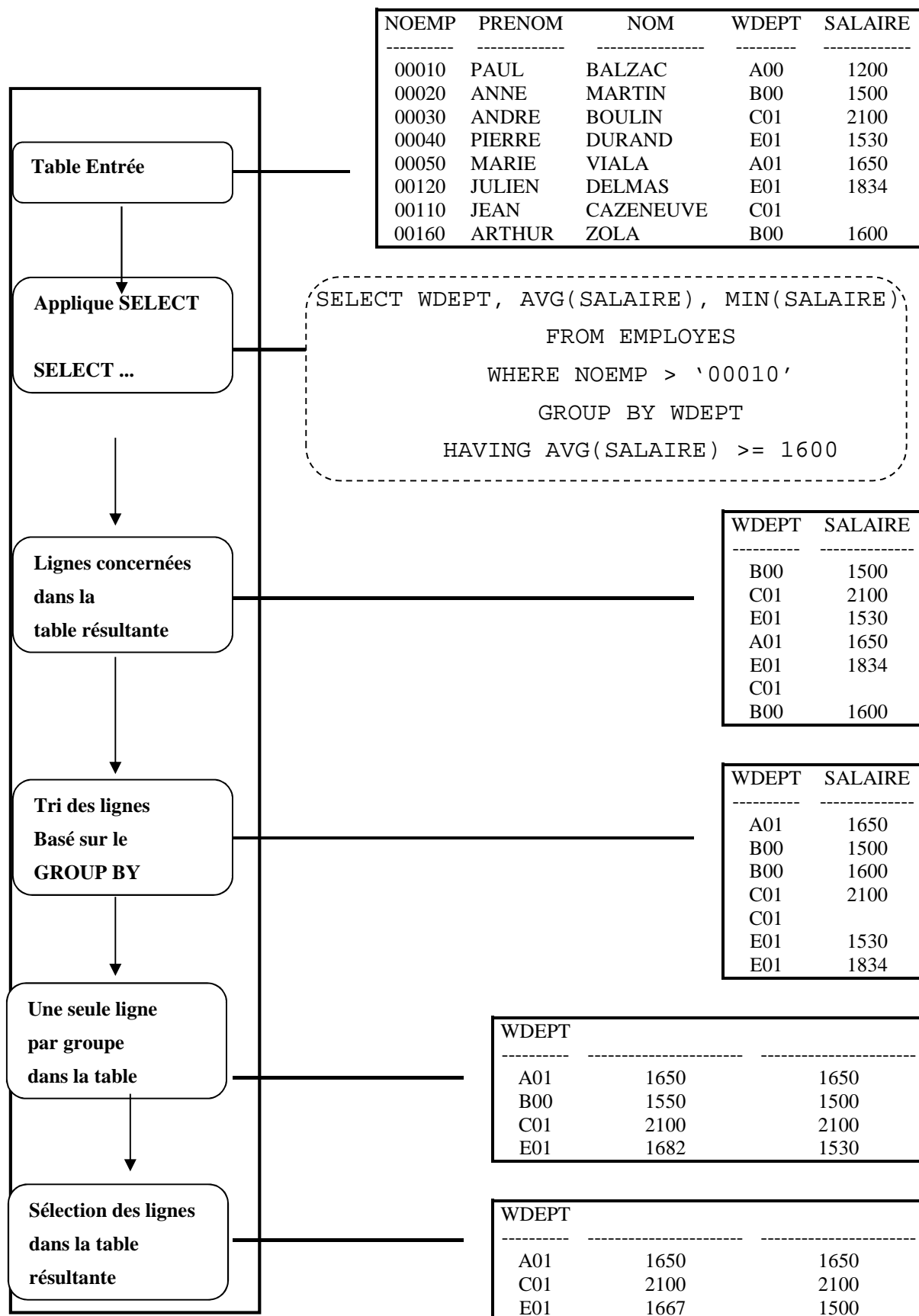
La condition de recherche suivant **HAVING** ne peut porter que sur des colonnes de regroupement définies par la clause **GROUP BY**, ou sur des fonctions.

Il ne faut pas confondre les clauses **WHERE** et **HAVING**.

WHERE permet de sélectionner des lignes avant la formation des groupes.

HAVING permet de ne retenir que certains des groupes constitués par la clause **GROUP BY**.

Analyse du HAVING



Les étapes suivantes sont réalisées :

1. Les colonnes WDEPT et SALAIRE sont sélectionnées seulement pour les N° d'employés supérieurs à 00010.
2. Ces lignes sont triées dans la séquence GROUP BY
3. La moyenne et le Min des salaires sont calculés.
4. Seuls les groupes ayant une moyenne des salaires supérieure à 1500 sont renvoyés.

Exemple 2 : Quelle est la moyenne des salaires, le salaire minimum des employés des départements ayant plus d'un salarié ?

```
SELECT WDEPT, AVG (SALAIRE), MIN (SALAIRE) FROM EMPLOYES
WHERE NOEMP > 00010
GROUP BY WDEPT
HAVING COUNT (*) > 1
```

WDEPT		1.1.1.1
-----	-----	-----
B00	15500	1500
C01	21000	2100
E01	16822	1530
		1.1.1.2

2. L'OPERATION DE JOINTURE

Une jointure est une opération qui permet d'interroger plusieurs tables pour obtenir un jeu de résultats unique intégrant des lignes et des colonnes de chaque table.

La plupart des conditions de jointure sont basées sur la clé primaire d'une table et la clé étrangère d'une autre table. Quand la jointure se fait sur des tables ayant des noms de colonne identiques, les noms en double doivent être préfixés par leur nom de table ou un nom associé.

Il existe trois types de jointure : les jointures internes, les jointures externes et les jointures croisées.

Syntaxe partielle:

```
SELECT nom_colonne1, nom_colonne2, ... nom_colonne n
FROM nom_table1
[INNER | {LEFT | RIGHT | FULL} [OUTER]] JOIN
nom_table2 ON conditions _recherche
```

- Le mot clé **JOIN** et ses options spécifient les tables à joindre et la manière de les joindre.
- Le mot clé **ON** spécifie les colonnes communes aux tables.
La plupart des conditions de jointure sont basées sur la clé primaire d'une table et la clé étrangère d'une autre table.

a) Utilisation des jointures internes

Exemple 1: Liste des employés avec le nom du département dans lequel ils sont affectés.

```
SELECT NOM, PRENOM, SALAIRE, NOMDEPT FROM EMPLOYES
      INNER JOIN DEPART
      ON WDEPT = NODEPT
```

NOM	PRENOM	SALAIRE	NOMDEPT
BALZAC	PAUL	1200	SERVICE INFORMATIQUE
MARTIN	ANNE	1500	PLANNING
ZOLA	ARTHUR	1600	PLANNING

La jointure interne renvoie uniquement les lignes en correspondance.

Analyse de la JOINTURE

Table EMPLOYES

NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00010	PAUL	BALZAC	A00	1200
00020	ANNE	MARTIN	B00	1500
00030	ANDRE	BOULIN	C01	2100
00040	PIERRE	DURAND	E01	1530
00050	MARIE	VIALA	A01	1650
00120	JULIEN	DELMAS	E01	1834
00110	JEAN	CAZENEUVE	C01	
00160	ARTHUR	ZOLA	B00	1600

Table DEPART

NODEPT	NOMDEPT	NOMGR
A00	SERVICE INFORMATIQUE	00010
B00	PLANNING	00020
E21	LOGICIEL	00030

```

SELECT NOM, PRENOM, SALAIRE, NOMDEPT
FROM EMPLOYES
INNER JOIN DEPART
ON WDEPT = NODEPT

```

NOM	PRENOM	SALAIRE	NOMDEPT
BALZAC	PAUL	1200	SERVICE INFORMATIQUE
MARTIN	ANNE	1500	PLANNING
ZOLA	ARTHUR	1600	PLANNING

On aurait pu coder la requête suivant la syntaxe suivante :

```
SELECT NOM, PRENOM, SALAIRE, NOMDEPT
      FROM EMPLOYES, DEPART
      WHERE WDEPT = NODEPT
```

b) Utilisation des jointures externes gauche ou droite

- La jointure externe gauche permet d'afficher toutes les lignes de la première table nommée (table située à gauche de la clause JOIN).
- La jointure externe droite permet d'afficher toutes les lignes de la seconde table nommée (table située à droite de la clause JOIN).
- La clause `LEFT OUTER JOIN` peut être abrégée en `LEFT JOIN` ;
La clause `RIGHT OUTER JOIN` peut être abrégée en `RIGHT JOIN` .
- La jointure externe complète (`FULL OUTER JOIN`) renvoie les lignes en correspondance (`INNER JOIN`) + les lignes sans correspondance des 2 tables.

Exemple 2: Liste des employés avec le nom du département dans lequel ils sont affectés.

```
SELECT NOM, PRENOM, SALAIRE, NOMDEPT FROM EMPLOYES
      LEFT JOIN DEPART
      ON WDEPT = NODEPT
```

NOM	PRENOM	SALAIRE	NOMDEPT
BALZAC	PAUL	1200	SERVICE INFORMATIQUE
MARTIN	ANNE	1500	PLANNING
BOULIN	ANDRE	2100	<NULL>
DURAND	PIERRE	1530	<NULL>
VIALA	MARIE	1650	<NULL>
DELMAS	JULIEN	1834	<NULL>
CAZENEUVE	JEAN		<NULL>
ZOLA	ARTHUR	1600	PLANNING

La jointure externe renvoie toutes les lignes de la table EMPLOYES. La colonne NOMDEPT contient la valeur NULL pour les départements n'existant pas dans la table DEPART.

c) Utilisation des jointures croisées

Les jointures croisées affichent toutes les combinaisons de toutes les lignes des tables jointes. Il n'est pas nécessaire de disposer de colonnes communes, et la clause **ON** n'est pas utilisée pour les jointures croisées.

Les jointures croisées sont rarement utilisées sur une base de données normalisée : Lors d'une jointure croisée, SQL génère un produit cartésien dans lequel le nombre de lignes du jeu de résultats est égal au nombre de lignes de la première table multiplié par le nombre de lignes de la deuxième table.

```
SELECT NOM, NODEPT FROM EMPLOYES  
CROSS JOIN DEPART
```

NOM	NODEPT
-----	-----
BALZAC	A00
MARTIN	A00
BOULIN	A00
DURAND	A00
VIALA	A00
DELMAS	A00
CAZENEUVE	A00
ZOLA	B00
BALZAC	B00
MARTIN	B00
BOULIN	B00
DURAND	B00
VIALA	B00
DELMAS	B00
CAZENEUVE	B00
ZOLA	B00
BALZAC	E21
MARTIN	E21
BOULIN	E21
DURAND	E21
VIALA	E21
DELMAS	E21
CAZENEUVE	E21
ZOLA	E21

d) Jointure de plus de deux tables

Il est possible de joindre jusqu'à 256 tables dans une seule requête.

L'utilisation de plusieurs jointures peut être ramenée à une combinaison de jointures indépendantes :

Exemple : Jointure entre les tables A, B et C

La première jointure combine les tables A et B pour produire un jeu de résultats, lui-même combiné à la table C dans la deuxième jointure pour produire le jeu de résultats final.

La requête pourra se coder sous la forme :

```
SELECT colonne_1, colonne_2, ..., colonne_n
FROM Table_1
JOIN Table_2
ON Table_1.colonne_i = Table_2.colonne_j
JOIN Table_3
ON Table_2.colonne_x = Table_3.colonne_y
```

La qualification des colonnes est nécessaire lorsque deux tables possèdent des colonnes de même nom.

a) Auto-jointure

Pour trouver des lignes ayant des valeurs en commun avec d'autres lignes de la même table, une table peut être jointe avec une autre instance d'elle-même ;

- Des alias de tables sont nécessaires pour référencer deux copies de la table : Un alias de table est spécifié dans la clause `FROM` après le nom de la table.
- Il peut être nécessaire d'utiliser des conditions dans la clause `WHERE` pour exclure les lignes en double.

Exemple : Liste des employés ayant le même prénom

```
SELECT A.NOM, NODEPT FROM EMPLOYES A
JOIN EMPLOYES B
ON A.PRENOM = B.PRENOM
WHERE A.NOM <> B.NOM
```

3. POUR ALLER PLUS LOIN

3.1 L'OPERATEUR UNION

L'opérateur `UNION` est utilisé pour combiner l'information retrouvée par 2, ou plus, ordres `SELECT`.

Syntaxe :

```
Instruction_select
```

```
UNION [ALL]
```

```
Instruction_select
```

Les colonnes des jeux de résultats doivent se correspondre. Il n'est pas nécessaire qu'elles portent le même nom, mais chaque jeu de résultats doit avoir le même nombre de colonnes, et ces colonnes doivent avoir des types de données compatibles et être dans le même ordre.

La liste des lignes issues des tables citées dans les `SELECT` de l'`UNION` ne comporte aucun doublon, sauf si l'option `ALL` est spécifiée.

Les noms de colonne du jeu de résultats sont ceux de la première instruction `SELECT`. Les alias de colonne s'ils sont à définir, seront donc définis dans la première instruction `SELECT`.

Si la clause `ORDER BY` doit être définie, elle sera définie à la suite de la dernière instruction `SELECT` (génération d'erreur sinon).

Exemple : Liste du département A00

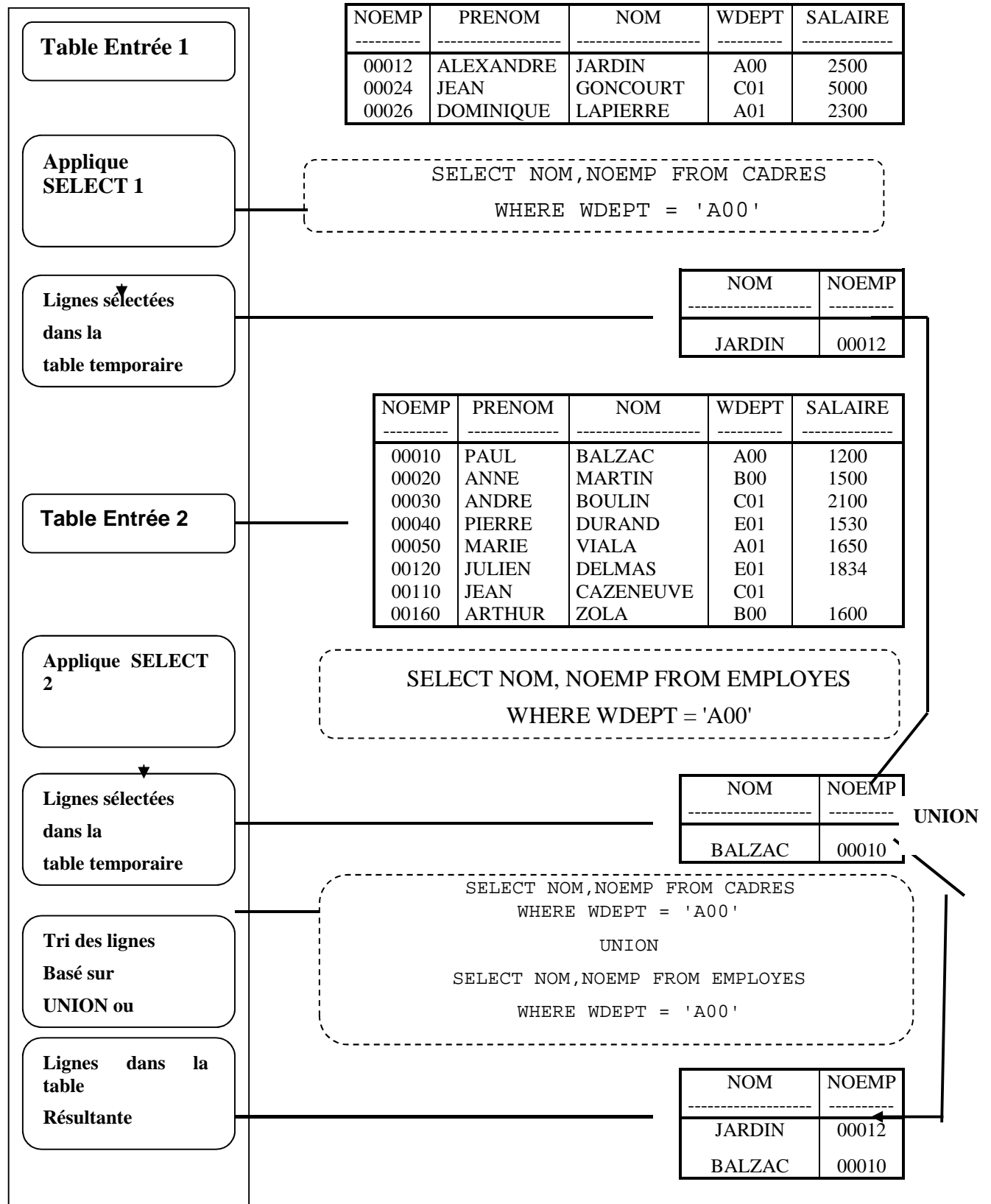
```
SELECT NOM, NOEMP FROM CADRES
      WHERE WDEPT = 'A00'

UNION

SELECT NOM, NOEMP FROM EMPLOYES
      WHERE WDEPT = 'A00'
```

NOM	NOEMP
-----	-----
JARDIN	00012
BALZAC	00010

Analyse de l'UNION



Les étapes suivantes sont réalisées :

1. Le premier `SELECT` est exécuté : les résultats sont sauvés dans une table temporaire.
2. Le deuxième `SELECT` est exécuté : les résultats sont sauvés dans une table temporaire
3. Les lignes sont triées suivant les clauses `UNION` ou `ORDER BY`.
4. Les doublons sont éliminés dans la table résultante.

3.2 LES SOUS REQUETES

Une sous-requête est une instruction SQL imbriquée dans une instruction `SELECT`, `INSERT`, `UPDATE` ou `DELETE`.

Elles permettent de scinder une requête complexe en une suite d'étapes logiques et de résoudre un problème avec une seule instruction.

Elles peuvent être imbriquées – s'exécutant une fois lorsque la requête externe s'exécute -ou en corrélation – s'exécutant une fois pour chaque ligne renvoyée lors de l'exécution de la requête externe.

- Elles doivent être encadrées par des parenthèses à droite de l'opérateur de la clause `WHERE`.
- Elles peuvent être imbriquées dans d'autres sous-requêtes
- Une sous-requête retourne toujours une seule colonne avec une ou plusieurs lignes
- La requête qui contient la sous-requête est généralement appelée *requête externe* ; la sous-requête est appelée *requête interne* ou *SubSelect*.

e) **Sous requête imbriquée** : Renvoi d'une valeur unique

Exemple: Liste des employés ayant un salaire supérieur à la moyenne des salaires.

```
SELECT NOM, PRENOM FROM EMPLOYES
```

```
WHERE SALAIRE > (SELECT AVG (SALAIRE) FROM EMPLOYES)
```

NOM	PRENOM
-----	-----
BOULIN	ANDRE
VIALA	MARIE
DELMAS	JULIEN

La sous requête est tout d'abord évaluée pour donner un résultat unique :

```
SELECT AVG (SALAIRE) FROM EMPLOYES
```

```
-----  
1630
```

Le salaire de chaque employé de la table EMPLOYES est alors évalué par rapport à cette valeur.

f) Sous requête imbriquée : Renvoi d'une liste de valeurs

Il est important en utilisant les sous-requêtes de savoir si le jeu de résultats comprendra plus d'une occurrence.

Si plus d'une occurrence est retournée, un des prédicats suivants doit être codé dans la clause externe `WHERE`.

- **IN** Contrôle si la valeur en cours du `SELECT` externe appartient à la liste des valeurs résultantes du SubSELECT.
- **ANY** ou **SOME** Contrôle si la valeur en cours du `SELECT` externe est `>`, `<`, `=`, `>=` ou `<=` à au moins une des valeurs de la liste des valeurs résultantes du SubSELECT.
- **ALL** Contrôle si la valeur en cours du `SELECT` externe est `>`, `<`, `=`, `>=` ou `<=` à toutes les valeurs de la liste des valeurs résultantes du SubSELECT.

Résultats SubSELECT > 1 ligne

Table EMPLOYES

NOEMP -----	PRENOM -----	NOM -----	WDEPT -----	SALAIRE -----
00010	PAUL	BALZAC	A00	1200
00020	ANNE	MARTIN	B00	1500
00030	ANDRE	BOULIN	C01	2100
00040	PIERRE	DURAND	E01	1530
00050	MARIE	VIALA	A01	1650
00120	JULIEN	DELMAS	E01	1834
00110	JEAN	CAZENEUVE	C01	
00160	ARTHUR	ZOLA	B00	1600

SELECT NOM, PRENOM
FROM EMPLOYES
WHERE SALAIRE

(SELECT SALAIRE FROM EMPLOYES
WHERE WDEPT = 'B00')

Résultats SubSELECT

SALAIRE -----
1500
1600

IN

Résultat final

NOM -----	PRENOM -----
MARTIN	ANNE
ZOLA	ARTHUR

> ANY

Résultat final

NOM -----	PRENOM -----
BOULIN	ANDRE
DURAND	PIERRE
VIALA	MARIE
DELMAS	JULIEN
ZOLA	ARTHUR

> ALL

Résultat final

NOM -----	PRENOM -----
BOULIN	ANDRE
VIALA	MARIE
DELMAS	JULIEN

g) Sous requête **EN CORRELATION**

Avec les sous-requêtes en corrélation, la requête interne se sert des informations de la requête externe et s'exécute pour chaque ligne de cette dernière.

Le traitement effectif fonctionne de la manière suivante :

- Chaque ligne de la table est utilisée en entrée pour les colonnes du `SELECT` primaire
- La ligne en cours du `SELECT` externe est utilisée pour fournir les valeurs pour le `SubSELECT`.
Le `SubSELECT` est évalué et un résultat est retourné.
Le résultat peut être une valeur simple ou plusieurs occurrences d'une seule colonne.
- La clause `WHERE` du `SELECT` primaire peut maintenant être évaluée en utilisant les valeurs retrouvées depuis le `SubSELECT`.

Exemple: Liste des employés dont le salaire est inférieur à la moyenne des salaires des employés de leur département.

```
SELECT NOM, PRENOM, SALAIRE, WDEPT FROM EMPLOYES A
      WHERE SALAIRE < (SELECT AVG (SALAIRE) FROM EMPLOYES B
                      WHERE A.WDEPT = B.WDEPT)
```

NOM	PRENOM	SALAIRE	WDEPT
MARTIN	ANNE	1500	B00
DURAND	PIERRE	1530	E01

Les alias de table sont obligatoires pour distinguer les noms des tables

Déroulement de l'évaluation de la requête :

- Une ligne de la table `EMPLOYEES` est lue la valeur de `WDEPT` est transmise à la requête interne.
- La requête interne est exécutée : la moyenne des salaires du département dont la valeur a été transmise est alors calculée et cette valeur est alors renvoyée à la requête externe.
- La clause `WHERE` de la requête externe est alors évaluée et la ligne est incluse ou non dans le jeu de résultats.
- Le traitement passe à la ligne suivante.

h) Sous requête EN CORRELATION : Utilisation de EXISTS et NOT EXISTS

Le mot clé EXISTS renvoie un indicateur vrai ou faux selon que la requête interne renvoie ou non des lignes.

La ligne de la table primaire sera incluse dans le jeu de résultats si

- La sous-requête a renvoyé au moins une ligne si EXISTS est codé
- La sous-requête n'a pas renvoyé de ligne si NOT EXISTS est codé

Exemple: Liste des employés qui ne sont pas affectés à un département existant.

```
SELECT NOM, PRENOM, SALAIRE, WDEPT FROM EMPLOYES
WHERE NOT EXISTS (SELECT * FROM DEPART
                  WHERE NODEPT = WDEPT)
```

NOM	PRENOM	WDEPT
-----	-----	-----
BOULIN	ANDRE	C01
DURAND	PIERRE	E01
VIALA	MARIE	A01
DELMAS	JULIEN	E01
CAZENEUVE	JEAN	C01

Il faut obligatoirement coder '**SELECT ***' dans le SubSELECT, puisque EXISTS ne retourne pas de données.

3.3 LA CLAUSE GROUP BY GÉNÉRALE

Une clause GROUP BY générale inclut des **GROUPING SETS**, **CUBE**, **ROLLUP**.

Exemple : On souhaite connaître le nombre de commandes par quantité pour chaque produit dans la table

DETAILCOMMANDE (IDProduit, QteCommandee, DateCde)

Avec une clause GROUP BY simple :

```
SELECT IDPRODUIT,
       SUM(QTECOMMANDEE) as 'QUANTITE TOTALE'
       COUNT(IDPRODUIT) AS NBCOMMANDES,
FROM DETAILCOMMANDE
GROUP BY IDPRODUIT, QTECOMMANDEE
```

Pour une table en entrée contenant :

IDPRODUIT	QTECOMMANDEE
-----	-----
P1	1
P2	10
P3	10
P1	5
P1	5

On obtient :

IDPRODUIT	QUANTITE TOTALE	NB COMMANDES
-----	-----	-----
P1	1	1
P1	10	2
P2	10	1
P3	10	1

En remplaçant GROUP BY IDPRODUIT, QTECOMMANDEE par

GROUP BY **ROLLUP** (IDPRODUIT, QTECOMMANDEE),

SQL Server génère les lignes du GROUP BY simple, et rajoute des lignes de sous-total et de total général

On obtient :

IDPRODUIT	QUANTITE TOTALE	NB COMMANDES
-----	-----	-----
P1	1	1
P1	10	2
P1	11	3
P2	10	1
P2	10	1
P3	10	1
P3	10	1
NULL	31	5

ROLLUP génère les lignes GROUP BY Simples, une ligne avec un sous-total pour chaque combinaison unique de valeurs d'IdProduit, et une ligne de total global.

En remplaçant GROUP BY IDPRODUIT, QTECOMMANDEE par
GROUP BY **CUBE** (IDPRODUIT, QTECOMMANDEE),

On obtient :

IDPRODUIT	QUANTITE TOTALE	NB COMMANDES
-----	-----	-----
P1	1	1
NULL	1	1
P1	10	2
NULL	10	2
P2	10	1
P3	10	1
NULL	20	2
NULL	31	5
P1	11	3
P2	10	1
P3	10	1

Cube génère des lignes GROUP BY simples, les lignes générées par ROLLUP et des lignes de tabulation croisée.

En utilisant la clause `GROUPING SETS`, on peut spécifier plusieurs regroupements de données dans une requête, sans les avoir tous comme dans les clauses `CUBE` ou `ROLLUP`. Les résultats sont l'équivalent de l'opération `UNION ALL` des groupes spécifiés.

```
SELECT IDPRODUIT,
       SUM(QTECOMMANDEE) as 'QUANTITE TOTALE'
       COUNT(IDPRODUIT) AS NBCOMMANDES,
FROM DETAILCOMMANDE
GROUP BY GROUPING SETS (IDPRODUIT, QTECOMMANDEE)
```

Équivaut à :

```
SELECT IDPRODUIT,
       SUM(QTECOMMANDEE) as 'QUANTITE TOTALE',
       COUNT (IDPRODUIT) AS NBCOMMANDES
FROM DETAILCOMMANDE
GROUP BY IDPRODUIT

UNION ALL

SELECT NULL AS IDPRODUIT,
       SUM(QTECOMMANDEE) as 'QUANTITE TOTALE',
       COUNT (IDPRODUIT) AS NBCOMMANDES
FROM DETAILCOMMANDE
GROUP BY QTECOMMANDEE
```

On obtient :

IDPRODUIT	QUANTITE TOTALE	NB COMMANDES
-----	-----	-----
P1	11	3
P2	10	1
P3	10	1
NULL	1	1
NULL	10	2
NULL	20	2

3.4 LES OPERATEURS EXCEPT ET INTERSECT

Les opérateurs ensemblistes `EXCEPT` et `INTERSECT` sont de nouveaux opérateurs permettant à l'utilisateur de retrouver des enregistrements communs à deux tables ou vues, ou des enregistrements appartenant à l'une sans appartenir à la deuxième.

Ces opérateurs respectent les mêmes règles que l'opérateur `UNION` : les 2 instructions `Select` disposent du même nombre de colonnes, et ces colonnes doivent être de type compatible (par exemple, `int` et `smallint` ou `char(10)` et `varchar(20)`).

Syntaxe :

`Instruction_select`

INTERSECT | EXCEPT

`Instruction_select`

`EXCEPT` renvoie toute valeur distincte de la requête de gauche mais non trouvée dans la requête de droite.

`INTERSECT` renvoie par contre toute valeur distincte renvoyée aussi bien par la requête à gauche que celle à droite de l'opérande `INTERSECT`.

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

E. Cattaneo – Formatrice

B. Hézard – Formateur

Ch. Perrachon – Ingénieure de formation

Date de mise à jour : 12/10/15

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »