Concepteur Développeur en Informatique

Assurer la persistance des données

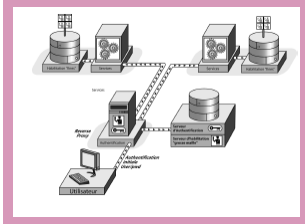
SQL – Manipulation des données - Sélection

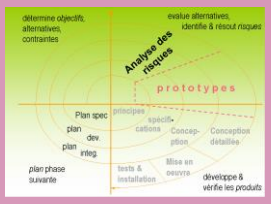
Accueil

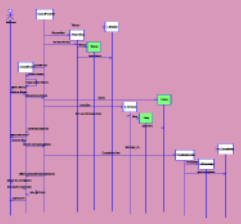
Apprentissage

PAE

Evaluation







Module 3 : Développer la Persistance des données

Sommaire

1 Introduction

Ce support a pour but de vous présenter le sous ensemble du langage SQL utilisé pour manipuler des données au sein d'un Système de Gestion de Bases de Données Relationnel.
On utilisera le terme de Langage de Manipulation de Données (LMD) pour qualifier ce sous-ensemble (de DML Data Management Language en anglais).
Avant d'aborder le langage SQL proprement dit, vous pouvez prendre connaissance du document sur l'algèbre relationnel qui traite de notions sous-jacentes aux opérations traitées dans ce document.

Seront traités dans ce document :

- Eléments syntaxiques relatifs à SQL
- Exemples de manipulation de données au travers des opérations de Sélection, Suppression, Mise à jour et Ajout.
- Différences syntaxiques au sein des différents dialectes SQL (document joint)

1.1 Bref Historique

S.Q.L. est un langage structuré qui permet :

- de créer la structure d'une base de données relationnelle (base, tables, …)
- d'interroger et de modifier les données contenues dans ce type de base de données.

S.Q.L. signifie Structured Query Language.
Il est issu de SEQUEL : Structured English Query Language.
C'est le premier langage pour les S.G.B.D. Relationnels.
Il a été développé par IBM en 1970 pour système R, son 1er SGBDR.

S.Q.L. a été reconnu par l'ANSI (Association de Normalisation des Systèmes d'Information) puis s'est imposé comme norme. Il n'existe pas de S.G.B.D. Relationnel sans S.Q.L..

Malgré la présente d'une norme S.Q.L., il existe un ensemble de dialectes. Les différences entre ces différents dialectes sont souvent minimes et tous respectent un minimum commun.
Il existe aujourd'hui une troisième version de SQL, SQL 3, qui propose de nouveaux éléments syntaxiques mieux à même de prendre en compte les évolutions technologiques liées au développement des applications développés selon un modèle objet.

1.2 Définition

S.Q.L. est un langage relationnel qui permet d'effectuer les tâches suivantes :

- Définition et modification de la structure de la base de données
- Interrogation et modification non procédurale (c'est à dire interactive) de la base de données
- Contrôle de sécurité et d'intégrité de la base
- Sauvegarde et restauration des bases

S.Q.L. est un langage interactif, mais il peut aussi être intégré dans un langage de programmation pour le développement d'applications.
S.Q.L. est un standard qui permet d'assurer une portabilité importante des bases de données relationnelles.
Il est tout à fait déconseiller de développer ses requêtes dans SQL server à l'aide d'un langage évolué tel que C# même si l'intégration du Framework Dot Net dans le moteur du SGBDR SQL Server 2005 et ses versions ultérieures le permet.

2 Opération de sélection : SELECT

L'instruction SELECT permet d'extraire des données et de les présenter triées et/ou regroupées suivant certains critères. Les enregistrements extraits devront donc vérifier certains critères exprimés dans des expressions conditionnelles.

La syntaxe de l'instruction SELECT présente les différentes clauses utilisables dans les sélections. Certaines clauses sont facultatives, mais il est important de respecter l'ordre d'apparition des clauses dans la requête.

```
SELECT  <NOMS DE COLONNES OU EXPRESSIONS>
FROM    <NOMS DE TABLES>
WHERE   <CONDITIONS DE RECHERCHE>
        GROUP BY    <NOMS DE COLONNE DU SELECT>
        HAVING      <CONDITION DE REGROUPEMENT>
        ORDER BY    <NOM OU POSITION DE COLONNE
                     DANS L'ORDRE SELECT>
```

Le résultat de la clause 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
```

2.1 Select avec expression

Une ou plusieurs expressions peuvent suivre le mot-clé SELECT. Une expression spécifie une valeur.

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

EXEMPLE :

```
'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
```

NOM	PRENOM	SALAIRE	
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	CENTIMESCEN
CAZENEUVE JEAN	SALAIRE :		TIMESCENTIME
ZOLA ARTHUR	SALAIRE :	160000	S

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 '

FROM EMPLOYES

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

2.2 Select Option DISTINCT

SELECT [ALL] nom col1 **FROM** nomtable
par opposition à

SELECT DISTINCT nomcol1 **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.

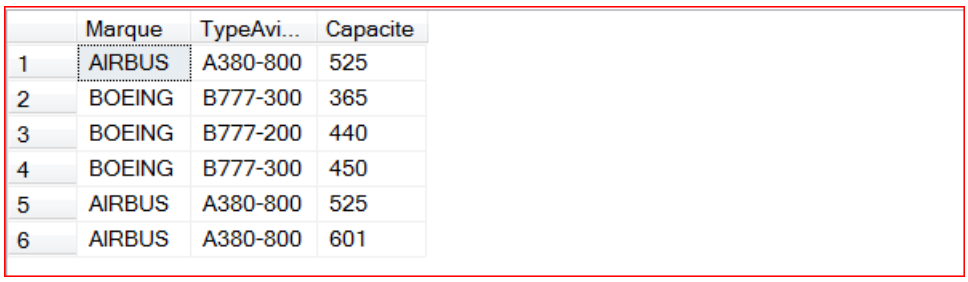
SELECT **WDEPT** FROM EMPLOYES SELECT **DISTINCT WDEPT** FROM EMPLOYES

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

Le résultat d'un SELECT étant un ensemble, il peut y avoir des doublons. Le mot clé DISTINCT permet de préciser que l'on ne veut qu'un seul exemplaire des lignes retenues dans la sélection.

Ensemble des types d'avions dont la capacité est supérieure à 250 passagers.

```
SELECT      Marque,TypeAvion, Capacite
FROM        AVION
WHERE       Capacite > 250
```



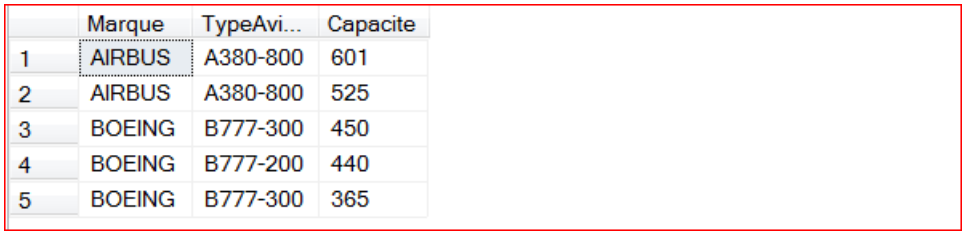
	Marque	TypeAvi...	Capacite
1	AIRBUS	A380-800	525
2	BOEING	B777-300	365
3	BOEING	B777-200	440
4	BOEING	B777-300	450
5	AIRBUS	A380-800	525
6	AIRBUS	A380-800	601

Figure 1 : Types d'avions dont la capacité est supérieure à 250

Nous ajoutons le mot clé **DISTINCT** à la sélection et ordonnons les données par capacités décroissantes :

```
SELECT DISTINCT Marque,TypeAvion,Capacite
FROM AVION
WHERE Capacite > 250
ORDER BY Capacite DESC
```

Remarques : la clause **ORDER BY** doit toujours être exprimée en dernier. Elle peut contenir plusieurs facteurs séparés par une virgule et l'ordre peut être croissant **ASC** (par défaut) ou décroissant **DESC**.
DISTINCT concerne la totalité des valeurs des attributs : ici, Marque, TypeAvion et Capacite.



	Marque	TypeAvi...	Capacite
1	AIRBUS	A380-800	601
2	AIRBUS	A380-800	525
3	BOEING	B777-300	450
4	BOEING	B777-200	440
5	BOEING	B777-300	365

Figure 2 : Sélection ordonnée de lignes distinctes

2.3 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

Avec la version 2005, n peut être variable :
En utilisant les compléments de Transact SQL, l'exemple 1 peut être codé :

Declare @n bigint (**@**n est une variable locale : le nom d'une variable locale est toujours préfixée par un @)
Set @n=5

SELECT TOP (@n) NOM, PRENOM FROM **EMPLOYES**

L'exemple 2 peut aussi être code :

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.

2.4 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** .

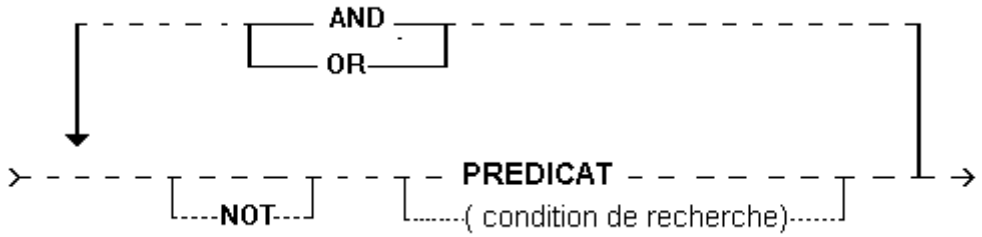
Exemples de prédicats :

SALAIRE = 15000

NOM = 'DUPONT'

NODEPT = NOGROUP

Conditions de recherche :



Exemples :

PRIX < 100.00 OR PRIX > 135.00

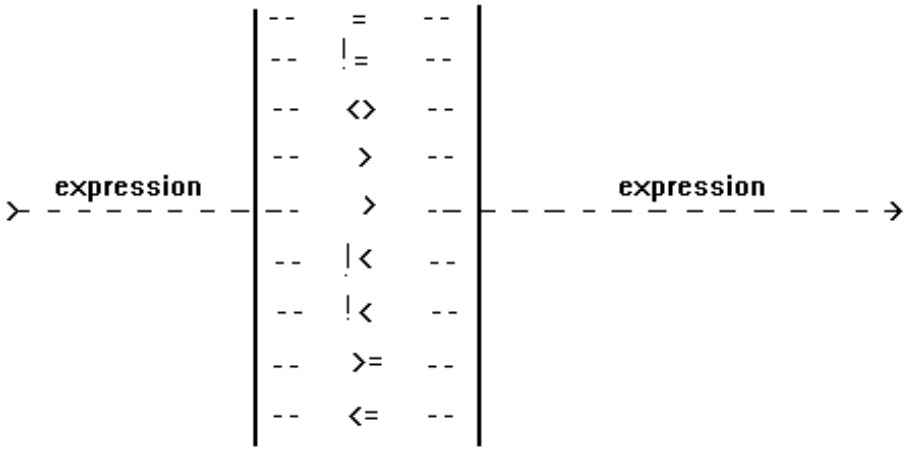
NOT (AUTEUR = 'DUMAS')

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**

2.4.1 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.

EXEMPLES :

Rechercher dans la table EMPLOYES, les données concernant les employés qui travaillent dans les départements 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**, **NOEM**P FROM **EMPLOYES**
WHERE **NOEMP > '00110' AND SALAIRE = 18345**

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
00120				

2.4.2 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 1200 et 1600 euros.

```
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.
```

2.4.3 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.

2.4.4 La clause 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	ANDRE	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

2.4.5 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.

2.5 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)

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.

2.5.1 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.

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

2.5.2 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*

Algébriques:

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

2.5.3 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

2.5.4 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, ‘12/06/2001’) renvoie 6

Le Day of Week renvoie la position du jour dans la semaine.

DATEPART (dw, ‘12/06/2013’) 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, '12/06/2013') renvoie 'mercredi'
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 variable système @@datefirst.

2.5.5 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

2.5.6 Les fonctions de classement

La fonction ROW_NUMBER permet de numéroter 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. ***Nous reviendrons sur cette problématique par la suite.***

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).

Dans l'exemple si deux salariés ont le même tarif horaire.

```
select [EmployeeID],Rate, RANK() over (order by Rate desc)
from [HumanResources].[EmployeePayHistory]
```

Résultats | Messages

	EmployeeID	Rate	Rang
5	140	60,0962	5
6	158	50,4808	6
7	42	50,4808	6
8	140	48,5577	8
9	268	48,101	9
10	284	48,101	9
11	288	48,101	9
12	3	43,2692	12
13	71	43,2692	12

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 numéroter 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	

2.6 Les clauses du SELECT

2.6.1 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
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

2.6.2 La clause GROUP BY

Ces options permettent de définir et de traiter des groupes.

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		
A01	1650	1650
B00	1550	1500
C01	2100	2100
E01	1682	1530

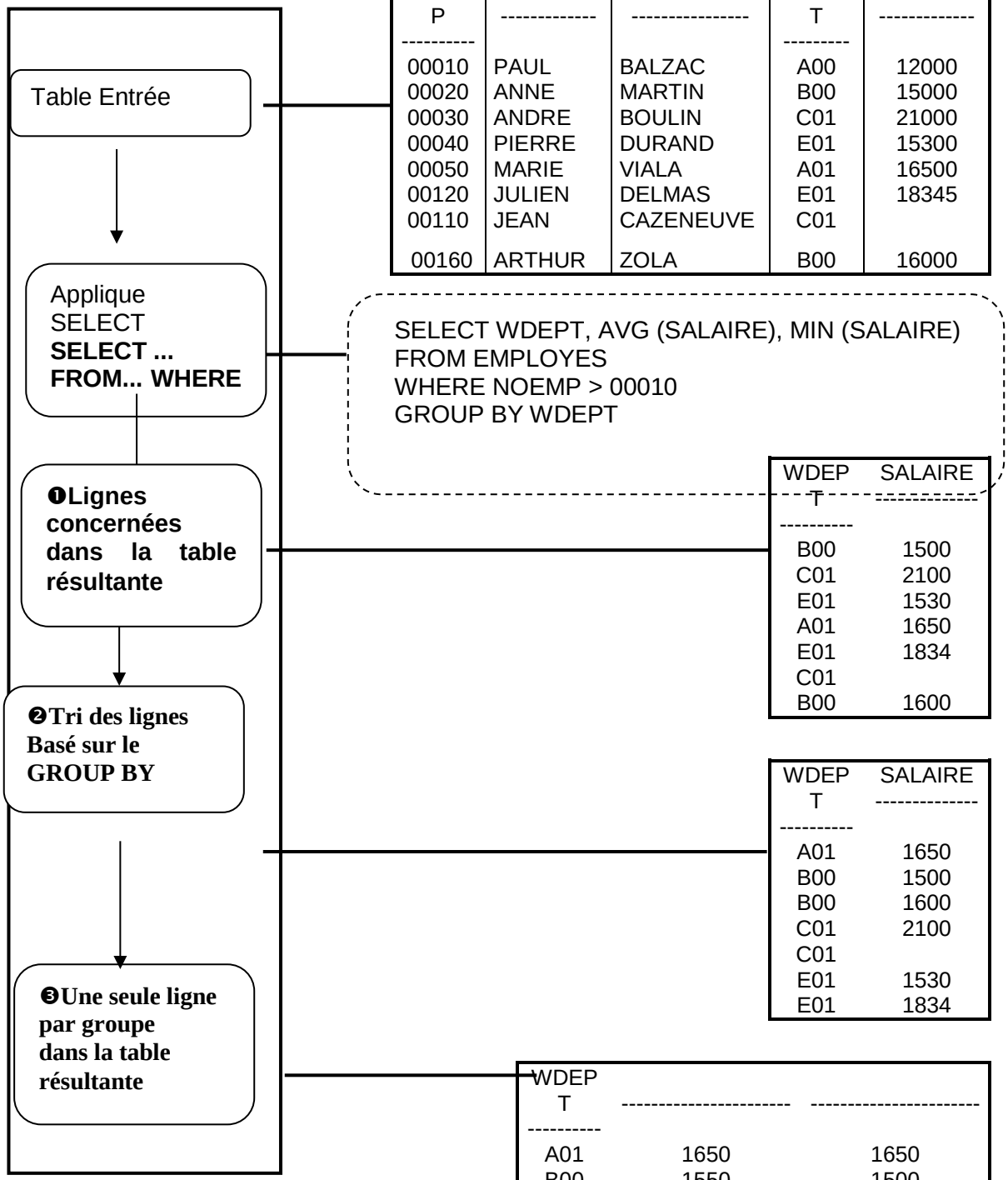
Il y a autant de groupes que de valeurs de WDEPT distinctes.

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.

2.6.2.2 Analyse du GROUP BY



NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00010	PAUL	BALZAC	A00	12000
00020	ANNE	MARTIN	B00	15000
00030	ANDRE	BOULIN	C01	21000
00040	PIERRE	DURAND	E01	15300
00050	MARIE	VIALA	A01	16500
00120	JULIEN	DELMAS	E01	18345
00110	JEAN	CAZENEUVE	C01	
00160	ARTHUR	ZOLA	B00	16000

WDEPT	SALAIRE
B00	1500
C01	2100
E01	1530
A01	1650
E01	1834
C01	
B00	1600

WDEPT	SALAIRE
A01	1650
B00	1500
B00	1600
C01	2100
C01	
E01	1530
E01	1834

WDEPT		
A01	1650	1650
B00	1550	1500
C01	2100	2100
E01	1682	1530

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

❷Ces lignes sont triées dans la séquence GROUP BY

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

2.6.3 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 a 16 000

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

WDEPT		
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.

AUTRES EXEMPLES

Nombre d'avions dans la table AVION

```
SELECT COUNT(AV#)
FROM AVION
```

Le résultat est 16 (avec mon jeu d'essai)

Je peux aussi donner un nom de colonne au résultat de l'expression **COUNT** en ayant recours au mécanisme des alias : NomColonne **AS** NomAlias. Nous verrons que nous devrons recourir de nouveau aux alias dans d'autres contextes.

```
SELECT COUNT(AV#) as "Nombre Avions"
FROM AVION
```

	Nombre Avions
1	16

Je peux aussi vouloir comptabiliser le nombre d'avions par marque.
La clause GROUP BY me permet alors de définir les conditions de regroupement des calculs récapitulatifs :

```
SELECT COUNT(AV#) as "Nombre Avions"
FROM AVION
GROUP BY Marque
```

	Nombre Avions
1	8
2	3
3	5

Il est préférable de faire figurer le nom de la marque dans le résultat et de trier les valeurs récapitulatives :

```
SELECT Marque,COUNT(AV#) as "Nombre Avions"
FROM AVION
GROUP BY Marque
ORDER BY "Nombre Avions" DESC
```

Qui peut s'écrire aussi :

```
SELECT Marque,COUNT(AV#) as "Nombre Avions"
FROM AVION
GROUP BY Marque
ORDER BY COUNT(AV#) DESC
```

	Marque	Nombre Avions
1	AIRBUS	8
2	BOEING	5
3	ATR	3

Je peux aussi ne pas vouloir conserver dans mon résultat les marques dont le nombre d'avions est inférieur à 4.
J'introduis alors dans ma requête une clause **HAVING** qui exprime une condition sur une opération de regroupement.

```
SELECT Marque,COUNT(AV#) as "Nombre Avions"
FROM AVION
GROUP BY Marque
HAVING COUNT(AV#) > 3  -- on ne peut pas utiliser "Nombre Avions"
ORDER BY COUNT(AV#) DESC
```

	Marque	Nombre Avions
1	AIRBUS	8
2	BOEING	5

Attention à ne pas confondre la clause **HAVING** et la clause **WHERE** qui filtre les lignes retenues pour les calculs.
Ainsi, si je souhaite ne pas retenir les avions localisés à Toulouse, je constate que les résultats diffèrent :

```
SELECT Marque,COUNT(AV#) as "Nombre Avions"
FROM AVION
WHERE  localisation != 'Toulouse'  -- Expression de différent
GROUP BY Marque
HAVING COUNT(AV#) > 3
ORDER BY COUNT(AV#) DESC
```

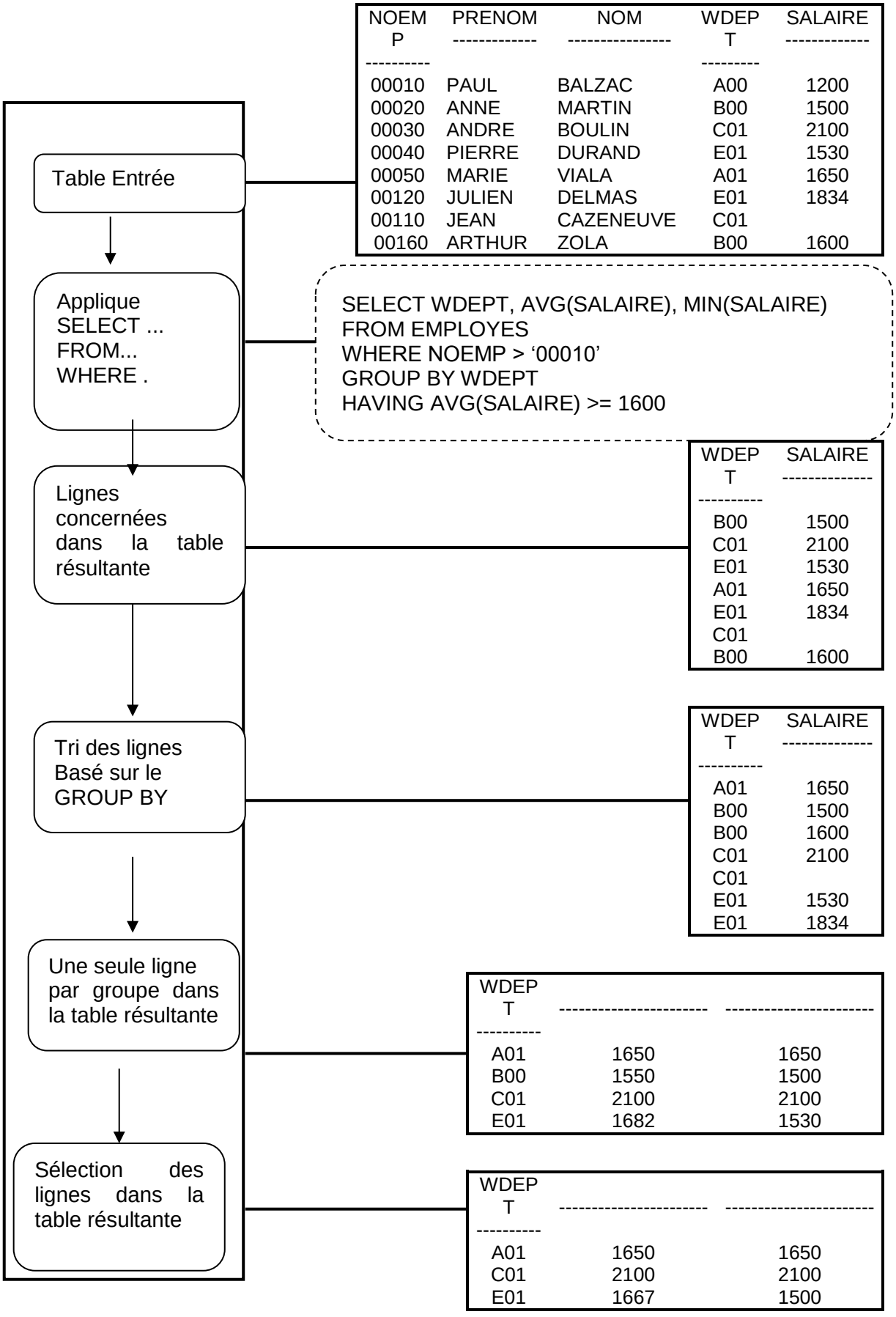
	Marque	Nombre Avions
1	AIRBUS	6
2	BOEING	5

Capacités minimum et maximum des Boeing.

```
SELECT MIN(Capacite) as "Capacite Mini Boeing",
 MAX(Capacite) as "Capacite Maxi Boeing"
FROM AVION
WHERE Marque = 'BOEING'
```

	Capacite Mini Boeing	Capacite Maxi Boeing
1	110	450

2.6.3.1 Analyse du HAVING



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

WDEPT	SALAIRE
B00	1500
C01	2100
E01	1530
A01	1650
E01	1834
C01	
B00	1600

WDEPT	SALAIRE
A01	1650
B00	1500
B00	1600
C01	2100
C01	
E01	1530
E01	1834

WDEPT		
A01	1650	1650
B00	1550	1500
C01	2100	2100
E01	1682	1530

WDEPT		
A01	1650	1650
C01	2100	2100
E01	1667	1500

Les étapes suivantes sont réalisées :

1. Les colonnes WDEPT et SALAIRE sont sélectées seulement pour les N° d'employées 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		
B00	15500	1500
C01	21000	2100
E01	16822	1530

2.6.4 La clause UNION

La clause UNION est utilisée 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. Les structures des tables temporaires issues du Select doivent être équivalentes.

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.

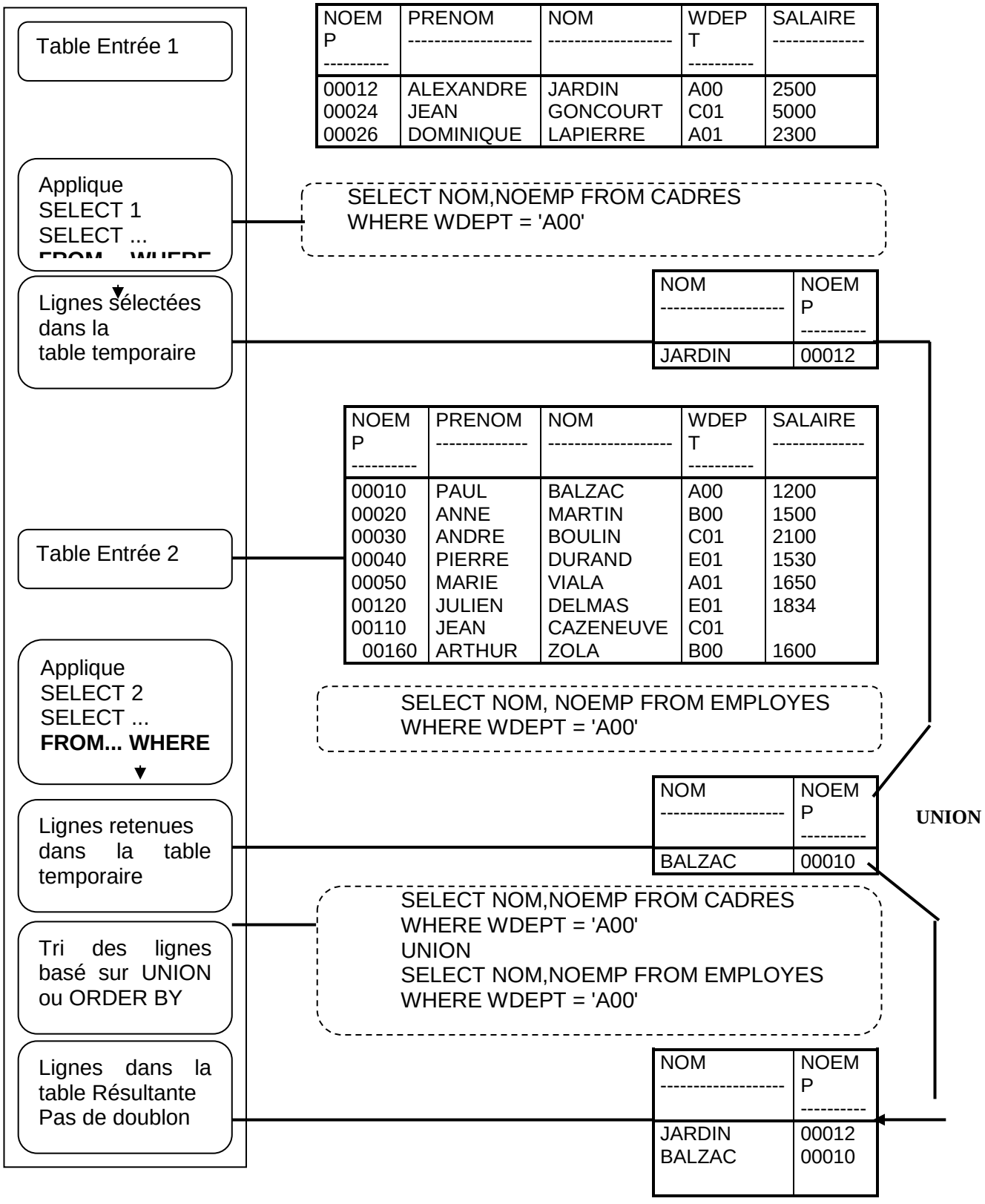
Si la clause ORDER BY doit être définie, elle sera définie à la suite de la dernière instruction SELECT (génération d'erreur sinon).

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

2.6.4.1 Analyse de l'UNION



NOEMP	PRENOM	NOM	WDEPT	SALAIRE
00012	ALEXANDRE	JARDIN	A00	2500
00024	JEAN	GONCOURT	C01	5000
00026	DOMINIQUE	LAPIERRE	A01	2300

NOM	NOEMP
JARDIN	00012

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

NOM	NOEMP
BALZAC	00010

NOM	NOEMP
JARDIN	00012
BALZAC	00010

2.6.4.2 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 Les jointures

Ce chapitre présente les différents types de jointure, les opérations de jointure et d'une manière générale, le travail avec des données issues de plusieurs tables.
Seront traités dans ce document :

- Les opérateurs ensemblistes
- La méthode des prédicats
- Les différentes natures de jointure (equi-jointure, jointure droite, gauche et totale)

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.

3.1 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.

3.1.1.1 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	00010
B00	INFORMATIQUE	00020
E21	PLANNING	00030
	LOGICIEL	

```
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

3.1.2 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 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.

3.2 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	A00
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

3.3 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.

3.4 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
```

3.5 Les sous requêtes

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*.

3.6 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.

3.7 SOUS REQUETE IMBRIQUEE : 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.

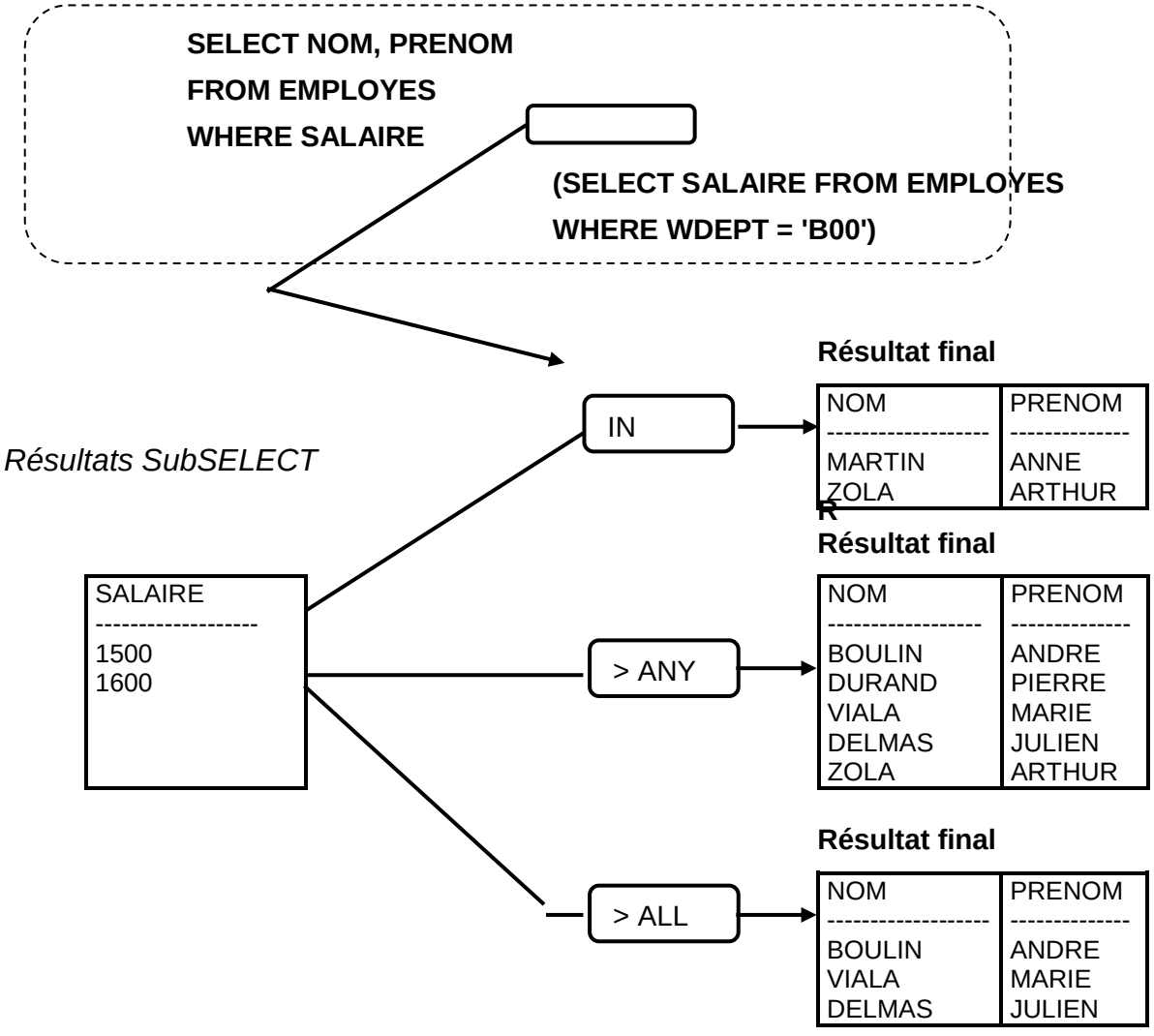
Si plus d'une occurrence est retournée, un des prédicats suivants doit être codé dans la clause externe WHERE.

- **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.

3.7.1.1 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



3.8 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.

3.9 Sous requête EN CORRELATION : Utilisation de EXISTS et NOT EXISTS

Le mot clé EXIST 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.

```
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.

3.10 Les opérateurs 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.

3.11 Autres exemples sur utilisation méthode ensembliste

Schéma de construction :

```
SELECT liste d'attributs
FROM table1
WHERE attribut de jointure
IN   (SELECT attribut de jointure
FROM table2
WHERE condition)
```

La requête à l'intérieur des parenthèses est dite requête interne ou sous-requête. Elle est évaluée en premier, constituant ainsi un premier ensemble dont on réalisera l'intersection (IN) avec l'ensemble issu de l'évaluation de la requête externe.
Les attributs sélectionnés, et retenus dans le jeu de résultat, sont ***nécessairement*** *issus de la requête externe. Il s'agit donc d'une méthode assez restrictive.*

D'une manière générale, l'exécution des requêtes construites selon la méthode ensembliste demande plus de ressources au système. Elles peuvent toutefois être plus faciles à réaliser et bien adaptées à certains cas de figure.

Exemple : Liste des pilotes assurant un vol au départ de Paris

```
SELECT PIL# as "Code Pilote",NOM
FROM PILOTE
WHERE PIL# IN (SELECT PILOTE
FROM VOL
WHERE Villedepart = 'Paris');
```

	Code Pilote	NOM
1	1	Tanguy
2	2	Morillon
3	7	Bertrand
4	9	Luc

Figure 3 : Illustration de la méthode ensembliste

Autre exemple : Traitement du sauf (négation NOT) ou différence.
Liste des pilotes qui ne sont pas affectés à des vols.

```
SELECT PIL# as "Code Pilote",NOM
FROM PILOTE
WHERE PIL# NOT IN (SELECT PILOTE
FROM VOL)
```

L'attribut de jointure est une valeur scalaire et ne peut donc être une valeur vectorielle. Ainsi, vous ne pouvez pas écrire une requête sous la forme qui suit :

```
SELECT PIL# as "Code Pilote",NOM
FROM PILOTE
WHERE (PIL#,VILLE) NOT IN (SELECT PILOTE,VILLEDEPART
FROM VOL)
```

On ne pourra pas non plus écrire la requête suivante :

```
SELECT PIL# as "Code Pilote",NOM,VILLEDEPART
FROM PILOTE
WHERE PIL# IN (SELECT PILOTE FROM VOL)
```

Car l'attribut VILLEDEPART n'appartient pas à la table sur laquelle porte la requête externe. Vous obtenez un message d'erreur :

Serveur : Msg 207, Niveau 16, État 3, Ligne 1
'VILLEDEPART' : nom de colonne incorrect.

On peut par contre joindre un attribut avec un attribut qui résulte d'une opération récapitulative.
Nous pouvons ainsi obtenir la liste des avions dont la capacité en passagers est égale à la capacité maximum.

```
SELECT AV#,MARQUE,TYPEAVION,CAPACITE
FROM AVION
WHERE CAPACITE IN
    (SELECT MAX(CAPACITE) FROM AVION)
```

	AV#	MARQUE	TYPEAVION	CAPACITE
1	170	AIRBUS	A380-800	601

Figure 4 : Liste des avions dont la capacité est égale à la capacité maximum

Si nous souhaitons préciser qu'il s'agit en fait des avions qui ont la capacité maximum du type, nous modifierons instinctivement la requête comme suit, afin de calculer le maximum selon le type :

```
SELECT AV#,MARQUE,TYPEAVION,CAPACITE
FROM AVION
WHERE CAPACITE IN
      (SELECT MAX(CAPACITE) FROM AVION GROUP BY
 MARQUE,TYPEAVION)
```

Ce qui donne pour résultat:

	AV#	MARQUE	TYPEAVION	CAPACITE
1	102	AIRBUS	A321-111	185
2	103	ATR	ATR42-300	48
3	104	BOEING	B777-200	440
4	105	AIRBUS	A321-211	199
5	106	ATR	ATR42-300	48
6	107	BOEING	B777-300	450
7	108	BOEING	B737-600	110
8	109	AIRBUS	320-212	164
9	120	ATR	ATR72-200	74
10	151	BOEING	B737-700	149
11	155	AIRBUS	A320-214	180
12	170	AIRBUS	A380-800	601
13	171	AIRBUS	A380-800	450

Ce résultat n'exprime aucune information vraiment exploitable.
L'avion 171 est retenu car il a la capacité max du 107 qui est un Boeing.

*En fait, pour obtenir un résultat exact, il nous faut **corréler** les lignes de la requête externe avec celles de la requête interne. Tous les SGBDR ne sont pas en capacité de réaliser ce type d'opérations.*
Elles sont à utiliser avec parcimonie dans la mesure du possible car coûteuses en ressources et en temps de traitement.

Dans l'exemple suivant, vous remarquerez l'utilisation d'un alias pour distinguer deux occurrences d'une même table.

Requêtes corrélées

```
SELECT AV#,MARQUE,TYPEAVION,CAPACITE
FROM AVION as "AV1"
WHERE CAPACITE IN
        (SELECT MAX(CAPACITE)
                FROM AVION
WHERE MARQUE = AV1.MARQUE AND TYPEAVION = AV1.TYPEAVION GROUP BY
MARQUE, TYPEAVION)
```

*Le résultat est alors correct et l'avion **171** éliminé du jeu de résultats.*

	AV#	MARQUE	TYPEAVION	CAPACITE
1	102	AIRBUS	A321-111	185
2	103	ATR	ATR42-300	48
3	104	BOEING	B777-200	440
4	105	AIRBUS	A321-211	199
5	106	ATR	ATR42-300	48
6	107	BOEING	B777-300	450
7	108	BOEING	B737-600	110
8	109	AIRBUS	320-212	164
9	120	ATR	ATR72-200	74
10	151	BOEING	B737-700	149
11	155	AIRBUS	A320-214	180
12	170	AIRBUS	A380-800	601

Figure 5 : Exemple de requête corrélée

Comparaison d'une valeur avec l'ensemble des valeurs d'un attribut d'une requête interne.

*On peut utiliser les opérateurs **SOME**, **ANY**, ou **ALL** pour comparer la valeur d'une expression avec les valeurs d'un attribut d'une requête interne.*

Exemple : Liste des avions dont la capacité est supérieure à toute capacité des avions de marque Boeing.

```
SELECT AV#, MARQUE, TYPEAVION, CAPACITE
FROM AVION
WHERE CAPACITE > ALL
     (SELECT CAPACITE FROM AVION WHERE MARQUE='Boeing')
```

La liste de valeurs générée par la requête interne est

	CAPACITE
1	365
2	440
3	450
4	110
5	149

Le résultat de ALL est le suivant : Uniquement les avions dont la capacité est supérieure à 450

	AV#	MARQUE	TYPEAVION	CAPACITE
1	100	AIRBUS	A380-800	525
2	160	AIRBUS	A380-800	525
3	170	AIRBUS	A380-800	601

Figure 6 : Comparer une valeur à un ensemble avec ALL

En modifiant la requête et en remplaçant ALL par ANY ou SOME, nous obtenons la liste de tous les avions dont la capacité est > 110.

	AV#	MARQUE	TYPEAVION	CAPACITE
1	100	AIRBUS	A380-800	525
2	101	BOEING	B777-300	365
3	102	AIRBUS	A321-111	185
4	104	BOEING	B777-200	440
5	105	AIRBUS	A321-211	199
6	107	BOEING	B777-300	450
7	109	AIRBUS	320-212	164
8	150	AIRBUS	A321-111	181
9	151	BOEING	B737-700	149
10	155	AIRBUS	A320-214	180
11	160	AIRBUS	A380-800	525
12	170	AIRBUS	A380-800	601
13	171	AIRBUS	A380-800	450

Figure 7 : Comparer une valeur à un ensemble avec SOME ou ANY

4 Précision et échelle des valeurs numériques

Lorsque nous effectuons des opérations arithmétiques au sein des requêtes SQL, il est important de s'approprier les règles de détermination de la précision et de l'échelle des valeurs numériques.

Tout d'abord précisons les notions de précision et d'échelle d'un nombre :

- La précision est le nombre de chiffres qui composent un nombre.
- L'échelle est le nombre de chiffres à droite de la virgule décimale dans un nombre

Ainsi, le nombre 235,56 a une précision de 5 et une échelle de 2 sous sa forme actuelle.

Dans la version actuelle de SQL Server, la précision maximale d'un nombre au format numeric ou decimal est de 38 chiffres.

La détermination de la précision et de l'échelle d'un nombre résultant d'une opération est fonction du type, de la précision et de l'échelle des opérandes.
La précision et l'échelle des types de données numériques sont constantes, sauf pour le type decimal.

Si un opérateur arithmétique agit sur deux expressions du même type, le résultat est du même type de données, avec la précision et l'échelle définies pour ce type.

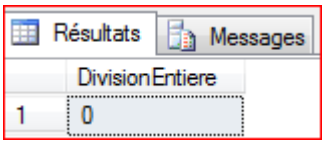
Ainsi, le code suivant sélectionne la division de deux entiers.

```
Declare @entierB int;
Declare @entierA int;

Set @entierA = 2;
Set @entierB = 3;

select @entierA/@entierB as DivisionEntiere;
```

Cette opération produit un entier, preuve en image :



4.1 Priorité des types de données

Si un opérateur agit sur **deux expressions** de **types de données numériques différents**, les règles de priorité des types de données déterminent le type du résultat. Le résultat a la précision et l'échelle définies pour son type de données.

1. types de données définis par l'utilisateur
2. **sql_variant**
3. **xml**
4. **datetime**
5. **smalldatetime**
6. **float**
7. **real**
8. **decimal**
9. **money**
10. **smallmoney**
11. **bigint**
12. **int**
13. **smallint**

14. **tinyint**
15. **bit**
16. **ntext**
17. **text**
18. **image**
19. **timestamp**
20. **uniqueidentifier**
21. **nvarchar** (y compris **nvarchar(max)**)
22. **nchar**
23. **varchar** (y compris **varchar(max)**)
24. **char**
25. **varbinary**(y compris **varbinary(max)**)
26. **binary** (plus bas niveau de priorité)

Exécutons la requête suivante :

```
Declare @reelB real;
set @reelB = 3;

select @entierA/@reelB as DivisionReel;
```

Ainsi, compte tenu des règles énoncées précédemment, le résultat sera de la précision du type réel.

Pour un réel la précision est de 7. Résultat en image :

	DivisionReel
1	0,6666667

4.2 Les nombres approximatifs

Attention à la définition des nombres approximatifs comme réels ou flottants.

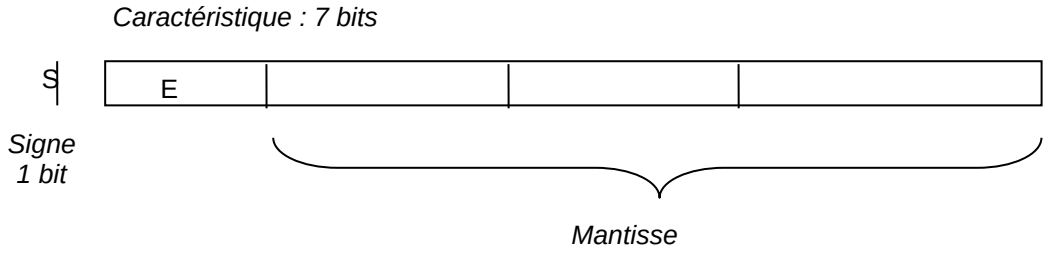
Elle impacte la précision et la taille nécessaire au stockage.

Le type float peut avoir une précision simple ou double. Un nombre de type float se déclare comme float(n) où *n* est le nombre de bits utilisé pour stocker la mantisse.

Soit le nombre float F, nous avons F = S - M * 10E où :

- S est le signe du nombre
- M est la mantisse
- E est l'exposant encore appelé caractéristique

On utilise alors 4 ou 8 octets suivant la précision souhaitée, simple ou double, pour coder le nombre.



La mantisse fait donc 24 bits en simple précision et 56 bits en double précision.

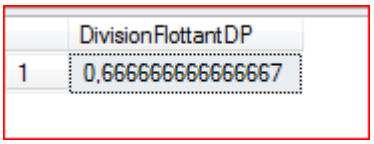
Par défaut le type float est en double précision.

Observons alors le résultat de notre opération en changeant le type réel en type float.

```
Declare @floatB float;
set @floatB = 3;

select @entierA/@floatB as DivisionFlottantDP;
```

Nous obtenons le résultat suivant :



	DivisionFlottantDP
1	0,666666666666667

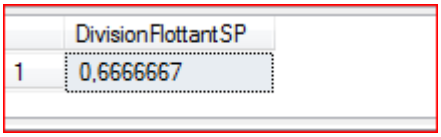
La précision est donc ici de 15 et non de 7. En fait le type real correspond au type float(24).

Modifions et exécutons notre code pour le vérifier.

```
Declare @float24B float(24);
set @float24B = 3;

select @entierA/@float24B as DivisionFlottantSP;
```

Nous obtenons un résultat identique à celui obtenu avec un réel.



	DivisionFlottantSP
1	0,6666667

4.3 Echelle et précision des résultats avec des nombres décimaux

Le cas des opérations dont le résultat est de type décimal.

Le tableau suivant montre le calcul de la précision et de l'échelle lorsque le résultat d'une opération est du type decimal.
Le résultat est de type decimal lorsqu'une des conditions suivantes est remplie :

- Les deux expressions sont de type decimal ;
- Une expression est de type decimal et l'autre est d'un type de données avec une priorité moins élevée comme par exemple une valeur entière.

Les expressions des opérandes sont notées e1 et e2, avec respectivement les précisions p1 et p2 et les échelles s1et s2.

Opération	Précision du résultat	Échelle du résultat *
e1 + e2	max(s1, s2) + max(p1-s1, p2-s2) + 1	max(s1, s2)
e1 - e2	max(s1, s2) + max(p1-s1, p2-s2) + 1	max(s1, s2)
e1 * e2	p1 + p2 + 1	s1 + s2
e1 / e2	p1 - s1 + s2 + max(6, s1 + p2 + 1)	max(6, s1 + p2 + 1)

Vérification par quelques exemples :

```
declare @decA decimal(3,2)
declare @decB decimal(4,3)
set @decA = 5
set @decB = 5

select @decA+@decB as AdditionDec,
@decA/@decB as DivisionDec,
@decA*@decB as MultiplicationDec
```

Précision de l'addition est égale à : max((2,3) + max(1,1) + 1 = 3 + 1+ 1 = 5

Echelle de l'addition est égale à : max(2,3) = 3

Le résultat est bien en (5,3)

	AdditionDec
1	10.000

La précision de la multiplication est égale à : 3+4+1 = 8
L'échelle est de : 2 + 3 = 5
Test avec opérande A tendant vers max et opérande B tendant vers max :

```
declare @decA decimal(3,2)
declare @decB decimal(4,3)
set @decA = 9.99
set @decB = 9.999
```

Le résultat est bien en (8,5)

MultiplicationDec
99.89001

Précision de la division est égale à 3 - 2 + 3 + max(6,2+4+1) = 4 + 7 = 11
Echelle de la division est max(6,2+4+1) = 7

Test avec opérande A tendant vers max et opérande B tendant vers 0 :

```
declare @decA decimal(3,2)
declare @decB decimal(4,3)
set @decA = 9.99
set @decB = 0.001
```

Le résultat est bien en (11,7)

DivisionDec
9990.0000000