

SQL déclaratif

du standard à la pratique

Langage d'interrogation des Bases de
Données

Partie 2

DRL – DATA RETRIEVAL LANGUAGE

La clause « SELECT »

Limiter et ordonner

Les fonctions

GROUP BY

Jointures

Sous-requêtes

La clause « **SELECT** »

```
SELECT colonne1, colonne2, colonne3, ...  
FROM nom_table
```

- La casse n'a pas d'importance, mais on prendra l'habitude d'écrire les mots-clés du langage en majuscules
- On écrira généralement les clauses (« **SELECT** », « **FROM** », etc.) sur des lignes différentes afin d'indenter au mieux le code

La clause « SELECT »

Sélectionner toutes les colonnes et toutes les lignes

```
SELECT *  
FROM student
```

Sélectionner toutes les lignes de certaines colonnes uniquement

```
SELECT first_name, last_name, year_result  
FROM student
```

La clause « SELECT » : Alias

```
SELECT first_name as [Prénom]  
      , last_name 'Nom de famille'  
      , section_id Section  
      , year_result as "Résultat annuel"  
FROM student
```

- Les alias servent à renommer l'intitulé des colonnes à l'affichage des données. Cela ne change rien au niveau des données contenues dans les tables, bien entendu
- Le mot-clé « **as** » n'est pas obligatoire
- **Sous SQL-Server**, les alias doivent être accompagnés de guillemets simples, doubles ou de crochets s'ils contiennent des caractères spéciaux, des accents ou des espaces

Prénom	Nom de famille	Section	Résultat annuel
Georges	Lucas	1320	10
Clint	Eastwood	1010	4
Sean	Connery	1020	12

La clause « SELECT » :

Opérations Arithmétiques

```
SELECT first_name, year_result  
      , (year_result/20)*100 as [Nouveau Résultat]  
FROM student
```

Opérateurs autorisés

Opérateur	Signification
/	Division
*	Multiplication
+	Addition / Concaténation
-	Soustraction

first_name	year_result	Nouveau Résultat
Georges	10	50
Clint	4	20
Sean	12	60
Robert	3	15
Kevin	16	80
Kim	19	95
Johnny	11	55
Julia	17	85
Natalie	4	20
Georges	4	20
Andy	19	95

La clause « SELECT » : Concaténation

```
SELECT first_name + ' ' + last_name as [Nom complet]  
      , [login] + student_id as 'Code étudiant'  
FROM student
```

Nom complet	Code étudiant
Georges Lucas	glucas1
Clint Eastwood	ceastwoo2
Sean Connery	sconnery3
Robert De Niro	rde niro4
Kevin Bacon	kbacon5
Kim Basinger	kbasinge6
Johnny Depp	jdepp7
Julia Roberts	jroberts8
Natalie Portman	nportman9
Georges Clooney	gclooney10
Andy Garcia	agarcia11

La clause « SELECT » : DISTINCT

```
SELECT DISTINCT first_name  
FROM student
```

Sans
DISTINCT

first_name
Alyssa
Andy
Bruce
Clint
David
Georges
Georges
Halle
Jennifer
Johnny

Avec
DISTINCT

first_name
Alyssa
Andy
Bruce
Clint
David
Georges
Halle
Jennifer
Johnny
Julia

La clause « SELECT » : DISTINCT

```
SELECT DISTINCT first_name, year_result  
FROM student
```

Sans
DISTINCT

first_name	year_result
Tom	4
Tom	4
Sophie	6
Shannen	2
Sean	12
Sarah	7
Sandra	2
Robert	3
Reese	7
Natalie	4

Avec
DISTINCT

first_name	year_result
Tom	4
Sophie	6
Shannen	2
Sean	12
Sarah	7
Sandra	2
Robert	3
Reese	7
Natalie	4
Michael .I	3

La clause « SELECT » : SANS FROM

```
SELECT col1 as alias_col1, col2, col3, ...
```

Sous SQL Server, la clause « **SELECT** » peut s'utiliser seule, si le but est simplement d'afficher un résultat structuré dans un tableau

```
SELECT GETDATE() as [Date du jour]  
      , 'Vive le SQL !'
```

Date du jour	(No column name)
2014-06-30 15:18:00.263	Vive le SQL !

Limiter et ordonner

```
SELECT colonne1, colonne2, colonne3, ...  
FROM nom_table  
WHERE conditions  
ORDER BY liste_colonnes
```

- La clause « **WHERE** » permet de limiter le nombre de lignes sélectionnées
- La clause « **ORDER BY** » permet de trier les résultats affichés, selon une ou plusieurs colonnes données
- Comme vu précédemment, ***ces clauses ne sont pas obligatoires*** mais si elles sont présentes, elles apparaissent dans cet ordre
- ***Il n'y a qu'une seule clause de chaque type.*** Il n'y aura donc jamais deux « **WHERE** » dans une même requête

Limiter et ordonner : « **WHERE** »

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE year_result >= 16
```

Opérateurs de comparaison

Opérateur	Signification
>	Plus grand
<	Plus petit
>=	Plus grand ou égal
<=	Plus petit ou égal
<>	Différent
!	Négation (« !> » = « pas plus grand »)

student_id	first_name	last_name	year_result
5	Kevin	Bacon	16
6	Kim	Basinger	19
8	Julia	Roberts	17
11	Andy	Garcia	19
18	Jennifer	Gamer	18
25	Halle	Berry	18

Limiter et ordonner : **BETWEEN**

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
WHERE year_result BETWEEN 10 AND 16
```

Liste des étudiants ayant obtenu un résultat annuel compris entre 10 et 16, ces valeurs incluses

Cela revient à demander la liste des étudiants qui ont un résultat
plus grand ou égal à 10 ET plus petit ou égal à 16

student_id	first_name	last_name	year_result
1	Georges	Lucas	10
3	Sean	Connery	12
5	Kevin	Bacon	16
7	Johnny	Depp	11
23	Keanu	Reeves	10

Limiter et ordonner : BETWEEN

```
SELECT first_name, last_name, birth_date
FROM student
WHERE birth_date BETWEEN '1960-01-01' AND '1970-12-31'
```

Les bornes de l'intervalle doivent être du même type que la valeur comparée. Dans cet exemple, les chaînes de caractères **'1960-01-01'** et **'1970-12-31'** seront automatiquement converties en dates afin de pouvoir être comparées aux valeurs de la colonne « **birth_date** »

first_name	last_name	birth_date
Johnny	Depp	1963-06-09 00:00:00.000
Julia	Roberts	1967-10-28 00:00:00.000
Georges	Clooney	1961-05-06 00:00:00.000
Tom	Cruise	1962-07-03 00:00:00.000
Sophie	Marceau	1966-11-17 00:00:00.000
Michael J.	Fox	1969-06-20 00:00:00.000
Sandra	Bullock	1964-07-26 00:00:00.000
Keanu	Reeves	1964-09-02 00:00:00.000
Halle	Berry	1966-08-14 00:00:00.000

Limiter et ordonner : IN

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
WHERE year_result IN(12,16,18)
```

Liste des étudiants dont le résultat annuel est ***égal à 12*** OU ***égal à 16*** OU ***égal à 18***

student_id	first_name	last_name	year_result
3	Sean	Connery	12
5	Kevin	Bacon	16
18	Jennifer	Gamer	18
25	Halle	Berry	18

Limiter et ordonner : IN

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE first_name IN('Tom', 'Jennifer', 'Halle')
```

L'opérateur « **IN** » permet de comparer tous types de données, tant que les valeurs entre parenthèses sont bien du même type que la valeur comparée. ***La casse n'a PAS d'importance***

student_id	first_name	last_name	year_result
13	Tom	Cruise	4
18	Jennifer	Gamer	18
20	Tom	Hanks	8
25	Halle	Berry	18

Limiter et ordonner : **LIKE**

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE first_name LIKE 'j%'
```

L'opérateur « **LIKE** » est utilisé pour comparer des chaînes de caractères entre elles

Le symbole « **%** » peut être utilisé pour remplacer **de 0 à N caractères**

Le symbole « **_** » peut être utilisé pour remplacer **1 caractère**

student_id	first_name	last_name	year_result
7	Johnny	Depp	11
8	Julia	Roberts	17
18	Jennifer	Gamer	18

Limiter et ordonner : LIKE

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE last_name LIKE '%oo_'
```

Liste des étudiants dont les trois dernières lettres du nom de famille sont « o », « o » et un caractère indéfini

student_id	first_name	last_name	year_result
2	Clint	Eastwood	4
14	Reese	Witherspoon	7

Limiter et ordonner : **NOT**

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE year_result NOT BETWEEN 10 AND 15
```

L'opérateur « **NOT** » marque la négation des opérateurs « **BETWEEN** », « **IN** » et « **LIKE** »

student_id	first_name	last_name	year_result
2	Clint	Eastwood	4
4	Robert	De Niro	3
5	Kevin	Bacon	16
6	Kim	Basinger	19
8	Julia	Roberts	17
9	Natalie	Portman	4
10	Georges	Clooney	4
11	Andy	Garcia	19
12	Bruce	Willis	6
12	Tom	Cruise	4

Limiter et ordonner : NOT

Liste des étudiants dont le nom de famille ne contient pas de « e »

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
WHERE last_name NOT LIKE '%e%'
```

Liste des étudiants dont le résultat annuel est différent de 12, 16 ou 18

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
WHERE year_result NOT IN (12,16,18)
```

Limiter et ordonner : **IS (NOT) NULL**

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
WHERE year_result IS NULL
```

Afin de déterminer si une valeur est « **NULL** » ou non, il faudra utiliser la syntaxe « **IS NULL** » dont la négation sera « **IS NOT NULL** »

student_id	first_name	last_name	year_result
5	Kevin	Bacon	NULL
7	Johnny	Depp	NULL
10	Georges	Clooney	NULL

Limiter et ordonner : **AND**

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE first_name like 'J%'
      AND year_result >= 10
```

L'opérateur « **AND** » permet de combiner plusieurs conditions en même temps

Une ligne doit répondre simultanément à toutes les conditions pour faire partie du résultat

student_id	first_name	last_name	year_result
7	Johnny	Depp	11
8	Julia	Roberts	17
18	Jennifer	Gamer	18

Limiter et ordonner : **OR**

```
SELECT student_id, first_name, last_name, year_result
FROM student
WHERE first_name like 'J%'
      OR year_result >= 10
```

Tout comme son compère « **AND** », l'opérateur « **OR** » permet également de combiner plusieurs conditions en même temps

Il suffit qu'une ligne réponde à l'une des conditions pour faire partie du résultat

student_id	first_name	last_name	year_result
1	Georges	Lucas	10
3	Sean	Connery	12
5	Kevin	Bacon	16
6	Kim	Basinger	19
7	Johnny	Depp	11
8	Julia	Roberts	17
11	Andy	Garcia	19

Limiter et ordonner : Précédence

Il est conseillé d'**utiliser des parenthèses** afin de forcer le système à tester les conditions dans l'ordre souhaité. Sous SQL-Server, si aucune parenthèse n'est utilisée, l'ordre de précedence suivant est appliqué

Ordre	Opérateurs évalués
1	Multiplication, Division, Modulo
2	Addition, Soustraction, Concaténation
3	Opérateurs de comparaisons (=,>,<!=,...)
4	NOT
5	AND
6	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
7	Affectation (=)

<http://msdn.microsoft.com/fr-fr/library/ms190276.aspx>

Limiter et ordonner : « ORDER BY »

```
SELECT student_id, first_name, last_name, year_result  
FROM student  
ORDER BY last_name ASC
```

La clause « **ORDER BY** » permet de trier le résultat d'une requête selon une ou plusieurs colonnes

Le tri peut se faire de façon croissante (« **ASC** » – ascendant) ou décroissante (« **DESC** » – descendant) sur les valeurs. La valeur par défaut est « **ASC** »

Il est possible de trier selon une colonne qui n'est pas affichée

student_id	first_name	last_name	year_result
5	Kevin	Bacon	16
6	Kim	Basinger	19
25	Halle	Berry	18
22	Sandra	Bullock	2
10	Georges	Clooney	4
3	Sean	Connery	12
12	Tom	Cruise	4

Limiter et ordonner : « ORDER BY »

```
SELECT section_id  
       , first_name + ' ' + last_name as 'Nom complet'  
FROM student  
ORDER BY section_id, 'Nom complet' DESC
```

Il est possible de trier les résultats *en fonction d'un alias* de colonne ainsi que sur *une combinaison de plusieurs colonnes*

section_id	Nom complet
1010	Sandra Bullock
1010	Natalie Portman
1010	Clint Eastwood
1010	Bruce Willis
1020	Tom Hanks
1020	Tom Cruise
1020	Sean Connery
1020	Sarah Michelle Gellar

Auto-Evaluation

N'oubliez pas de prendre le temps d'évaluer le niveau de maîtrise que vous estimez avoir acquis personnellement concernant les notions abordées dans ce module !

Rappel de la signification des lettres dans les tableaux d'auto-évaluation :

- **Parfait (P)** : vous avez parfaitement compris cette notion et vous vous sentez à votre aise
- **Satisfaisant (S)** : vous avez compris de quoi il s'agit mais la pratique vous manque
- **Vague (V)** : vous savez de quoi il s'agit, mais cela reste un peu vague dans votre esprit. Une explication supplémentaire du formateur ou une bonne révision de votre part s'impose
- **Insatisfaisant (I)** : Vous n'avez pas du tout compris la notion abordée, il faut tout faire pour y remédier !

Auto-Evaluation

Notions à évaluer

Notions	P	S	V	I
Ordre « SELECT ... FROM »				
Alias				
Concaténation et conversion de type de données (CONVERT)				
Clause « WHERE »				
Opérations arithmétiques				
Opérateurs « BETWEEN », « IN », « LIKE » et leur négation				
Comparaison avec une valeur « NULL »				
Opérateurs « AND » et « OR »				
Clause « ORDER BY »				

Les fonctions

Une fonction est un ensemble de lignes de code stocké dans le système, qui exécute à la demande, une tâche pour l'utilisateur et renvoie un résultat. Une fonction demande souvent que **des paramètres soient fournis** en entrée.

Le résultat renvoyé doit ensuite être affiché ou inclus dans une expression ou une requête. Une fonction peut bien sûr utiliser le résultat d'une autre fonction

- Le nom de la fonction est ***toujours suivi de parenthèses***, même si aucun paramètre n'est fourni ou attendu
- Les fonctions renvoient des valeurs de types divers. Le tableau suivant classe les fonctions présentées dans cette formation, selon le type de valeur qu'elles renvoient

Type retourné	Noms des fonctions
numérique	datepart, charindex, len, abs, modulo
chaîne de caractères	substring, upper, lower, replace, trim
datetime	getdate()

Les fonctions : **CONVERT**

CONVERT (NOUVEAU_TYPE, valeur_à_convertir)

CONVERT (TYPE_CHAÎNE_DE_CARACTÈRES, date_à_convertir, format_date)

La fonction « **CONVERT** » attend 2 paramètres en entrée (éventuellement 3 lorsque l'élément à convertir est une date) et *renvoie une valeur correspondant au deuxième paramètre dans le type spécifié par le premier* (et dans le format précisé par le troisième, le cas échéant **[100-114]**)

```
SELECT CONVERT (varchar, birth_date, 110)
           as [Date de naissance]
FROM student
```

Date de naissance
05-17-1944
05-31-1930
08-25-1930
00-17-1942

Les fonctions : **GETDATE()**

```
SELECT  GETDATE ()  
        , CONVERT (varchar, GETDATE (), 109)  
        , CONVERT (date, GETDATE ())  
        , CONVERT (time, GETDATE ())
```

Sous SQL-Server, la fonction « **GETDATE()** » renvoie la date et l'heure actuelles

Type retourné : DATETIME

Date du jour	Date du jour formatée	Date uniquement	Heure uniquement
2014-05-14 14:21:09.210	May 14 2014 2:21:09:210PM	2014-05-14	14:21:09.2130000

Les fonctions : **DATEPART**

DATEPART (*partie_de_date_à_extraire, date_traitée*)

La fonction « **DATEPART** » extrait une partie d'une date donnée

Type retourné : NOMBRE

```
SELECT DATEPART (mm, GETDATE ())  
      , DATEPART (dy, GETDATE ())  
      , DATEPART (ns, GETDATE ())
```

Mois	Jour de l'année	Nanosecondes
5	134	153000000

<http://msdn.microsoft.com/fr-be/library/ms174420.aspx>

Les fonctions : **CHARINDEX**

CHARINDEX (*chaine_de_caractères_recherchée, valeur_à_évaluer*)

La fonction « **CHARINDEX** » renvoie la position du début de l'occurrence d'une chaîne de caractère dans une autre

Type retourné : NOMBRE

```
SELECT CHARINDEX('i', 'Kim Basinger')  
      , CHARINDEX('08', 'Basinger 08/12/1953')  
      , CHARINDEX('y', 'Kim Basinger')  
      , CHARINDEX('', 'Kim Basinger')
```

position du premier i	position du 08	pas de y	chaîne vide
2	10	0	0

Les fonctions : **LEN**

LEN (*chaine_de_caractères_à_mesurer*)

La fonction « **LEN** » renvoie le nombre de lettres composant une chaine de caractères donnée, espaces blancs compris

Type retourné : NOMBRE

```
SELECT LEN('Kim Basinger')
```

Longueur de la chaine de caractères

12

Les fonctions : **ABS**

ABS (*nombre*)

La fonction « **ABS** » renvoie la valeur absolue du nombre passé en paramètre

Type retourné : NOMBRE

```
SELECT ABS (-1.0) , ABS (0.0) , ABS (1.0)  
SELECT ABS (-2147483648)
```

val1	val2	val3
1.0	0.0	1.0

```
Msg 8115, Level 16, State 2, Line 1  
Arithmetic overflow error converting expression to data type int.
```

Les fonctions : Modulo

dividende % diviseur

Le « % », qui représente la fonction « **modulo** » que l'on rencontre fréquemment dans d'autres langages également, **renvoie le reste de la division ENTIÈRE du premier nombre (dividende) par le second (diviseur)**. Cette fonction permet de savoir si le premier chiffre est multiple du second

Type retourné : NOMBRE

```
SELECT 38 / 5, 38 % 5
```

$$\begin{array}{r} 38,0 \\ 5 \overline{) 38,0} \\ \underline{35} \\ 30 \\ \underline{30} \\ 0 \end{array}$$

Division entière	Reste
7	3

Les fonctions : **SUBSTRING**

SUBSTRING (*chaine_de_caractères, position_départ, nombre_caractères*)

La fonction « **SUBSTRING** » renvoie une chaîne de caractère d'une longueur souhaitée, à partir d'une position donnée, à l'intérieur d'une chaîne de caractères passée en paramètre

Type retourné : CHAÎNE DE CARACTÈRES

```
SELECT SUBSTRING ('Basinger', 4, 3)
SELECT last_name, SUBSTRING(first_name, 1, 1)
FROM student
```

Caractères 4, 5 et 6
ing

last_name	Initiale du prenom
Lucas	G
Eastwood	C
Connery	S
De Niro	R
Bacon	K

Les fonctions : **UPPER** et **LOWER**

UPPER (*chaîne_de_caractères*)

LOWER (*chaîne_de_caractères*)

Les fonctions « **UPPER** » et « **LOWER** » renvoient la chaîne de caractères passée en paramètres, respectivement en majuscules ou en minuscules

Type retourné : CHAÎNE DE CARACTÈRES

```
SELECT UPPER(last_name), LOWER(first_name)
FROM student
WHERE UPPER(first_name) LIKE 'TOM'
```

Nom de famille	Prénom
CRUISE	tom
HANKS	tom

Les fonctions : **REPLACE**

REPLACE (*chaine_de_caractères_traitée, caract_à_remplacer, nouveau_caract*)

La fonction « **REPLACE** » remplace les caractères demandés par d'autres

Type retourné : CHAÎNE DE CARACTÈRES

```
SELECT REPLACE(' Kim Basinger ', ' ', '')  
      , REPLACE('11110000101010', '1', '0')
```

Sans espaces	Sans 1
KimBasinger	0000000000000000

Les fonctions : **LTRIM** et **RTRIM**

LTRIM (*chaine_de_caractères*)

RTRIM (*chaine_de_caractères*)

Les fonctions « **LTRIM** » et « **RTRIM** » renvoient la chaîne de caractères passée en paramètres épurée des espaces blancs éventuellement présents en début ou en fin de chaîne, respectivement

Type retourné : CHAÎNE DE CARACTÈRES

```
SELECT LTRIM('    Kim Basinger    ')  
      , RTRIM('    Kim Basinger    ')  
      , LTRIM(RTRIM('    Kim Basinger    '))
```

LTRIM	RTRIM	LTRIM de RTRIM
Kim Basinger	Kim Basinger	Kim Basinger

Les fonctions : Agrégations

Une fonction d'agrégation est une fonction particulière qui attend comme paramètres **un ensemble de valeurs** (une colonne) et qui présente en retour **un seul** résultat agrégé (regroupé) sur ces valeurs. Sauf exceptions, les valeurs **NULL** ne sont pas prises en compte

Fonctions d'agrégation principales

Fonction	Description
COUNT	Nombre total de valeurs contenues dans la table/colonne
MAX	Valeur numérique la plus élevée
MIN	Plus petite valeur numérique disponible
SUM	Somme de l'ensemble des valeurs de la colonne
AVG	Moyenne de l'ensemble des valeurs de la colonne

Les fonctions : **COUNT**

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

La fonction d'agrégation « **COUNT** » renvoie le nombre total de valeur contenues dans la table ou la colonne à laquelle on applique la fonction. Les valeurs « **NULL** » ne sont prises en compte que dans l'utilisation du « **COUNT(*)** »

Type retourné : NOMBRE (INTEGER)

```
SELECT COUNT(*), COUNT(first_name), COUNT(DISTINCT first_name)  
FROM student
```

Total des lignes	Total des prénoms	Total des prénoms sans doublons
25	25	23

Les fonctions : **MAX** et **MIN**

MAX (*colonne*)

MIN (*colonne*)

Les fonctions d'agrégation « **MAX** » et « **MIN** » renvoient respectivement la plus grande ou la plus petite des valeurs contenues dans une colonne donnée

Type retourné : NOMBRE

```
SELECT MAX (year_result), MIN (year_result*5)  
      , MAX (LEN(last_name))  
FROM student
```

Résultat le plus élevé	Pourcentage le plus faible	Taille du nom le plus long
19	10	15

Les fonctions : **SUM**

SUM (*colonne*)

La fonction « **SUM** » renvoie la somme des valeurs d'une colonne

Type retourné : NOMBRE

```
SELECT SUM (year_result), SUM (year_result) / COUNT (*)  
FROM student
```

Somme des résultats annuels	Moyenne générale
219	8

Les fonctions : **AVG**

AVG (colonne)

La fonction « **AVG** » renvoie la moyenne de l'ensemble des valeurs contenues dans une colonne

Type retourné : NOMBRE

```
SELECT AVG (year_result)
        , AVG (DATEPART(yy,GETDATE()) - DATEPART(yy,birth_date))
FROM student
```

Moyenne générale	Moyenne d'âge
8	53

Les fonctions : **CASE**

```
CASE  
  WHEN expression1 THEN valeur1  
  WHEN expression2 THEN valeur2  
  ...  
  WHEN expressionN THEN valeurN  
  ELSE valeur_par_défaut  
END
```

- **L'instruction « CASE »** peut être utilisée afin de modifier l'affichage des éléments d'une colonne selon ce que l'on souhaite
- Dès qu'une expression contenue dans l'une des clauses « **WHEN** » est évaluée à « **TRUE** », la valeur contenue après la clause « **THEN** » est affichée dans la colonne et **l'instruction « CASE » se termine** et est réévaluée en totalité pour la ligne suivante
- Si aucune des expressions évaluées après les clauses « **WHEN** » n'est validée, la valeur affichée dans la colonne correspond à la valeur présentée dans la clause « **ELSE** »

Les fonctions : CASE

```
SELECT last_name, first_name, year_result,  
CASE  
    WHEN year_result BETWEEN 18 AND 20 THEN 'Excellent'  
    WHEN year_result BETWEEN 16 AND 17 THEN 'Très Bien'  
    WHEN year_result BETWEEN 14 AND 15 THEN 'Bien'  
    WHEN year_result BETWEEN 12 AND 13 THEN 'Suffisant'  
    WHEN year_result BETWEEN 10 AND 11 THEN 'Faible'  
    WHEN year_result BETWEEN 8 AND 9 THEN 'Insuffisant'  
    ELSE 'Insuffisance Grave'  
END AS [Note globale]  
FROM student
```

last_name	first_name	year_result	Note globale
Lucas	Georges	10	Faible
Eastwood	Clint	4	Insuffisance Grave
Connery	Sean	12	Suffisant
De Niro	Robert	3	Insuffisance Grave
Bacon	Kevin	16	Très Bien
Basinger	Kim	19	Excellent
Dann	Johnny	11	Faible

Les fonctions : CASE

```
CASE colonne_à_évaluer  
  WHEN valeur_de_comparaison1 THEN valeur1  
  ...  
  ELSE valeur_par_défaut  
END
```

Lorsque les expressions à évaluer sont des **égalités strictes**, il est possible de simplifier l'écriture du « **CASE** » en reprenant le nom de la colonne à évaluer directement après le mot-clé « **CASE** »

```
SELECT student_id, first_name,  
CASE section_id  
  WHEN 1010 THEN 'BSc Management'  
  WHEN 1320 THEN 'MA Sociology'  
  ELSE NULL  
END AS [Nom de section section]  
FROM student
```

last_name	first_name	Nom de section section
Lucas	Georges	MA Sociology
Eastwood	Clint	BSc Management
De Niro	Robert	NULL
Depp	Johnny	NULL
Portman	Natalie	BSc Management
Garcia	Andy	NULL
Willie	Bruce	BSc Management

Auto-Evaluation

N'oubliez pas de prendre le temps d'évaluer le niveau de maîtrise que vous estimez avoir acquis personnellement concernant les notions abordées dans ce module !

Rappel de la signification des lettres dans les tableaux d'auto-évaluation :


- **Parfait (P)** : vous avez parfaitement compris cette notion et vous vous sentez à votre aise
- **Satisfaisant (S)** : vous avez compris de quoi il s'agit mais la pratique vous manque
- **Vague (V)** : vous savez de quoi il s'agit, mais cela reste un peu vague dans votre esprit. Une explication supplémentaire du formateur ou une bonne révision de votre part s'impose
- **Insatisfaisant (I)** : Vous n'avez pas du tout compris la notion abordée, il faut tout faire pour y remédier !

Auto-Evaluation

Notions à évaluer

Notions	P	S	V	I
Fonction (fonctionnement interne, utilité, mise en pratique)				
Imbrication de fonctions				
Fonctions d'agrégation				
Expression « CASE »				

GROUP BY



```
SELECT colonnes, fonction_agrégation(colonne)
FROM table
WHERE condition_affichage_lignes
GROUP BY sous_groupes_agrégation
HAVING condition_affichage_groupes
ORDER BY ordre_tri_affichage
```

COUNT
MAX
MIN
SUM
AVG

- La clause « **GROUP BY** » permet de créer des sous-regroupements de lignes au niveau de la table, afin de leur appliquer une même fonction d'agrégation
- La clause « **HAVING** » ne peut être présente que si la clause « **GROUP BY** » est présente également. Le « **HAVING** » pose une condition d'affichage sur les groupes créés par la clause « **GROUP BY** ». Cette condition doit porter sur une fonction d'agrégation également
- Première règle d'or
*Dès que la clause « **SELECT** » combine l'affichage d'une ou plusieurs fonctions d'agrégation **ET** des colonnes non-agrégées, la clause « **GROUP BY** » est obligatoire*
- Seconde règle d'or
*Toutes les colonnes non-agrégées présentes dans la clause « **SELECT** » doivent impérativement se retrouver dans la clause « **GROUP BY** »*

GROUP BY

```
SELECT section_id, AVG(year_result)
FROM student
GROUP BY section_id
```

section_id	Moyenne par section
1010	4
1020	7
1110	8
1120	17
1310	11
1320	10

Sans le « **GROUP BY** », le système produit l'erreur suivante :

Column 'student.section_id' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

GROUP BY : + WHERE

```
SELECT section_id, AVG(year_result)
FROM student
WHERE LEFT(last_name,1) IN ('B', 'C', 'D')
GROUP BY section_id
```

section_id	last_name	year_result
1020	Connery	12
1110	De Niro	3
1120	Bacon	16
1310	Basinger	19
1110	Depp	11
1020	Clooney	4
1020	Cruise	4
1010	Bullock	2
1320	Doherty	2
1320	Berry	18

Moyenne = 6,67

section_id	Moyenne par section
1010	2
1020	6
1110	7
1120	16
1310	19
1320	10

Solution de la requête

Ensemble de lignes triées grâce à la clause « **WHERE** »
et sur lequel la clause « **GROUP BY** » sera appliquée

GROUP BY : + WHERE + HAVING

```
SELECT section_id, AVG(year_result)
FROM student
WHERE LEFT(last_name,1) IN ('B', 'C', 'D')
GROUP BY section_id
HAVING AVG(year_result) >= 10
```

section_id	last_name	year_result
1020	Connery	12
1110	De Niro	3
1120	Bacon	16
1310	Basinger	19
1110	Depp	11
1020	Clooney	4
1020	Cruise	4
1010	Bullock	2
1320	Doherty	2
1320	Berry	18

« WHERE » uniquement

section_id	Moyenne par section
1010	2
1020	6
1110	7
1120	16
1310	19
1320	10

WHERE + GROUP BY

section_id	Moyennes > 10
1120	16
1310	19
1320	10

WHERE + GROUP BY
+ HAVING

GROUP BY : colonnes multiples

```
SELECT section_id, course_id, AVG(year_result)
FROM student
WHERE section_id IN (1010, 1020)
GROUP BY section_id, course_id
HAVING SUM(year_result) >= 2
ORDER BY section_id
```

- *Toutes les colonnes non-agrégées présentes dans la clause « **SELECT** » doivent impérativement se retrouver dans la clause « **GROUP BY** »*
- La condition du « **HAVING** », portant sur l'affichage des groupes créés par la clause « **GROUP BY** », peut utiliser d'autres fonctions d'agrégation et d'autres colonnes que celles utilisées dans la clause « **SELECT** »

section_id	course_id	Moyenne
1010	EG1020	2
1010	EG2210	4
1020	EG1020	7
1020	EG2110	7
1020	EG2210	10

GROUP BY : ROLLUP et CUBE

```
SELECT colonnes, fonction_agrégation(colonne)
FROM table
GROUP BY ROLLUP (sous_groupes_agrégation)
```

```
SELECT colonnes, fonction_agrégation(colonne)
FROM table
GROUP BY CUBE (sous_groupes_agrégation)
```

- Les mots-clés « **ROLLUP** » ou « **CUBE** » peuvent être rajoutés à la clause « **GROUP BY** » de façon à afficher des sous-totaux
- « **ROLLUP** » applique un sous-total en remontant dans les colonnes indiquées, présentant un sous-total à partir de la colonne la plus détaillée, en remontant vers la colonne présentant des résultats groupés de façon plus vaste (sous-total par section et global)
- « **CUBE** » permet d'appliquer la fonction d'agrégation sur tout regroupement possible au niveau des données agrégées (sous-total par section, global et par cours, sans tenir compte des sections). Le « **CUBE** » englobe le « **ROLLUP** »

GROUP BY : ROLLUP

```
SELECT section_id, course_id, AVG(year_result)
FROM student
WHERE section_id IN (1010, 1020)
GROUP BY ROLLUP (section_id, course_id)
```

section_id	course_id	Moyenne
1010	EG1020	2
1010	EG2210	4
1020	EG1020	7
1020	EG2110	7
1020	EG2210	10

Sans « ROLLUP »

section_id	course_id	Moyenne
1010	EG1020	2
1010	EG2210	4
1010	NULL	4
1020	EG1020	7
1020	EG2110	7
1020	EG2210	10
1020	NULL	7
NULL	NULL	6

Total section 1010

Total section 1020

Total général

Avec « ROLLUP »

GROUP BY : CUBE

```
SELECT section_id, course_id, SUM(year_result)
FROM student
WHERE section_id IN (1010, 1020)
GROUP BY CUBE (section_id, course_id)
```

section_id	course_id	Somme
1010	EG1020	2
1020	EG1020	7
1020	EG2110	35
1010	EG2210	14
1020	EG2210	10

Sans « CUBE »

section_id	course_id	Somme
1020	EG2210	10
1020	EG2110	35
1020	EG1020	7
1020	NULL	52
1010	EG2210	14
1010	EG1020	2
1010	NULL	16
NULL	EG2210	24
NULL	EG2110	35
NULL	EG1020	9
NULL	NULL	68

Total section 1020

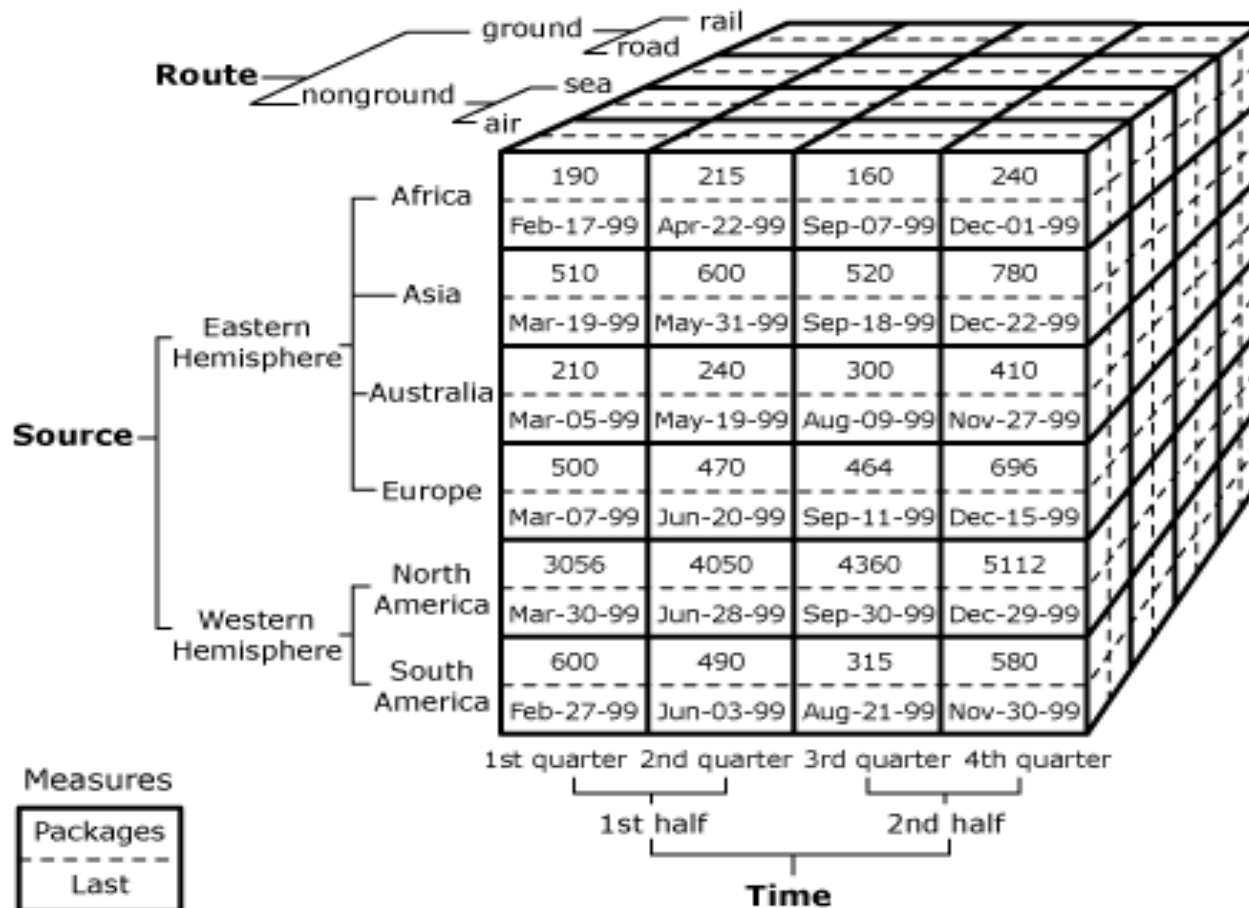
Total section 1010

Total par cours

Total général

Avec « CUBE »

GROUP BY : CUBE OLAP



Auto-Evaluation

N'oubliez pas de prendre le temps d'évaluer le niveau de maîtrise que vous estimez avoir acquis personnellement concernant les notions abordées dans ce module !

Rappel de la signification des lettres dans les tableaux d'auto-évaluation :

- **Parfait (P)** : vous avez parfaitement compris cette notion et vous vous sentez à votre aise
- **Satisfaisant (S)** : vous avez compris de quoi il s'agit mais la pratique vous manque
- **Vague (V)** : vous savez de quoi il s'agit, mais cela reste un peu vague dans votre esprit. Une explication supplémentaire du formateur ou une bonne révision de votre part s'impose
- **Insatisfaisant (I)** : Vous n'avez pas du tout compris la notion abordée, il faut tout faire pour y remédier !

Auto-Evaluation

Notions à évaluer

Notions	P	S	V	I
Clause « GROUP BY ... HAVING » et règles d'or				
Différence entre les clauses « WHERE » et « HAVING »				
« GROUP BY » sur plusieurs colonnes				
Clause « ROLLUP »				
Clause « CUBE »				

Jointures

Jointures horizontales

```
SELECT table1.col1, table1.col2, table2.col1, table2.col2  
FROM table1 JOIN table2 ON table1.col1 = table2.col2  
WHERE ... GROUP BY ... ORDER BY ...
```

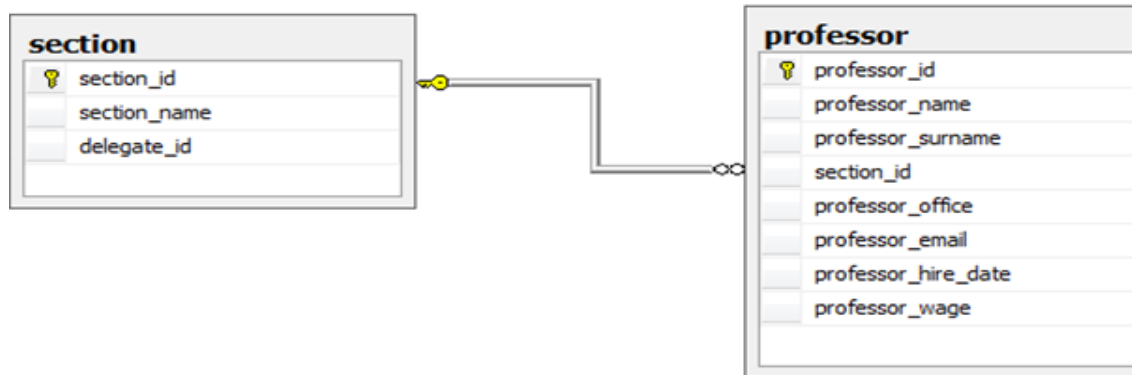
Comparaison des colonnes des tables entre elles

Jointures verticales

```
SELECT ... FROM ... WHERE ... GROUP BY ...  
opérateur_comparaison_requêtes  
SELECT ... FROM ... WHERE ... GROUP BY ...
```

Comparaison du résultat de deux requêtes entre eux

Jointures horizontales



- La jointure horizontale compare **deux colonnes** entre elles et affiche les colonnes souhaitées pour chaque concordance trouvée
- La condition de la jointure (c'est-à-dire la comparaison à faire) utilisera souvent les **clés primaires et étrangères** liant les tables (mais ce n'est pas obligatoire)
- Si les colonnes utilisées dans la requête ont le même nom dans plus d'une table participant à la jointure, il faudra faire précéder ces colonnes du nom de la table. Nous prendrons donc l'habitude de **donner un alias aux tables** et de faire précéder chaque colonne d'un alias de table créé
- Lorsqu'une table a reçu un alias, il n'est plus possible d'utiliser le nom de la table dans la requête car le système travaille désormais avec une copie de la table d'origine, portant l'alias comme nom

Jointures : CROSS JOIN

```
SELECT T1.col1, T1.col2, T2.col1, T2.col2  
FROM table1 T1 CROSS JOIN table2 T2
```

Le « **CROSS JOIN** » effectue simplement un produit cartésien des lignes de chacune des tables

T1.col1	T1.col2
10	15
20	25
30	35

3

X

2

T2.col1	T2.col2
10	BB
15	DD



T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB
10	15	15	DD
20	25	10	BB
20	25	15	DD
30	35	10	BB
30	35	15	DD

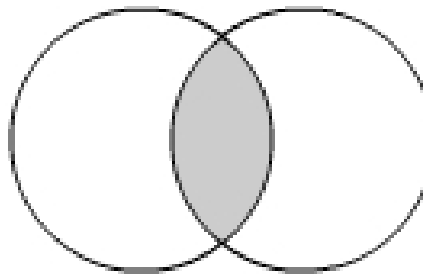


Jointures : **INNER JOIN**

```
SELECT *  
FROM table1 T1 JOIN table2 T2 ON T1.col1 = T2.col1
```

Le « **INNER JOIN** » compare les éléments des colonnes indiquées après le « **ON** » et affiche les informations demandées à chaque correspondance (mot-clé « **INNER** » facultatif *sous SQL-Server*)

T1.col1	T1.col2
10	15
20	25
30	35



T2.col1	T2.col2
10	BB
15	DD



T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB



Jointures : INNER JOIN

```
SELECT S.section_id, S.section_name, P.professor_name
FROM section S JOIN professor P
ON S.section_id = P.section_id
```

Table
« SECTION »

section_id	section_name	delegate_id
1010	BSc Management	12
1020	MSc Management	9
1110	BSc Economics	15
1120	MSc Economics	6
1310	BA Sociology	23
1320	MA Sociology	8

Table
« PROFESSOR »

professor_id	professor_name	section_id
1	zidda	1020
2	decrop	1120
3	giot	1310
4	lecourt	1310
5	scheppens	1020
6	louveaux	1110

Aucun professeur n'appartient aux sections 1010 et 1320
2 professeurs font partie des sections 1020 et 1310

section_id	section_name	professor_name
1020	MSc Management	zidda
1120	MSc Economics	decrop
1310	BA Sociology	giot
1310	BA Sociology	lecourt
1020	MSc Management	scheppens
1110	BSc Economics	louveaux

Résultat de la jointure

Jointures : INNER JOIN

```
SELECT S.section_id, S.section_name, P.professor_name  
FROM section S, professor P  
WHERE S.section_id = P.section_id
```



```
SELECT S.section_id, S.section_name, P.professor_name  
FROM section S JOIN professor P  
ON S.section_id = P.section_id
```

Sans faire précéder la colonne « ***section_id*** » de l'alias de l'une ou l'autre table, le système produit l'erreur suivante :

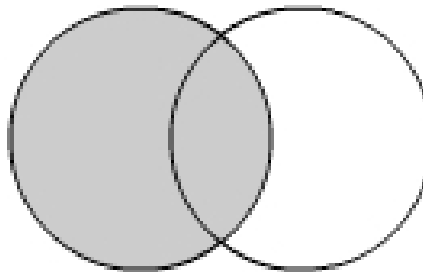
Ambiguous column name 'section_id'.

Jointures : **LEFT OUTER JOIN**

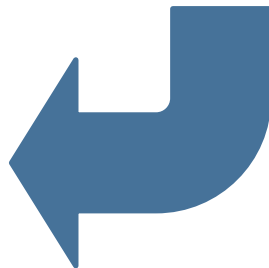
```
SELECT *  
FROM table1 T1 LEFT JOIN table2 T2 ON T1.col1 = T2.col1
```

Le « **LEFT OUTER JOIN** » affiche les informations demandées à chaque correspondance, mais affiche aussi toutes les lignes de la première table, même si elles n'ont pas de correspondance dans la seconde (mot-clé « **OUTER** » facultatif *sous SQL-Server*)

T1.col1	T1.col2
10	15
20	25
30	35



T2.col1	T2.col2
10	BB
15	DD



T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB
20	25	NULL	NULL
30	35	NULL	NULL



Jointures : LEFT OUTER JOIN

```
SELECT S.section_id, S.section_name, P.professor_name  
FROM section S LEFT JOIN professor P  
ON S.section_id = P.section_id
```

Aucun professeur n'appartient aux sections 1010 et 1320
2 professeurs font partie des sections 1020 et 1310

section_id	section_name	professor_name
1010	BSc Management	NULL
1020	MSc Management	zidda
1020	MSc Management	scheppens
1110	BSc Economics	louveaux
1120	MSc Economics	decrop
1310	BA Sociology	giot
1310	BA Sociology	lecourt
1320	MA Sociology	NULL

On désire afficher les informations sur toutes les sections, qu'il y ai un professeur qui y soit inscrit ou non

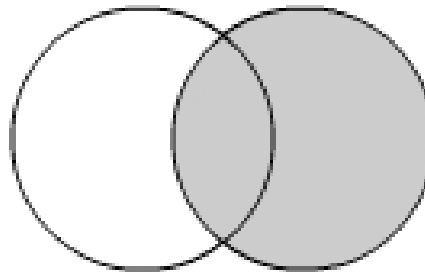
Liste de toutes les sections avec les professeurs qui y sont inscrits, s'il y en a

Jointures : **RIGHT OUTER JOIN**

```
SELECT *  
FROM table1 T1 RIGHT JOIN table2 T2 ON T1.col1 = T2.col1
```

Le « **RIGHT OUTER JOIN** » fonctionne de la même manière que le **LEFT**, mais concerne la seconde table de la jointure (mot-clé « **OUTER** » facultatif *sous SQL-Server*)

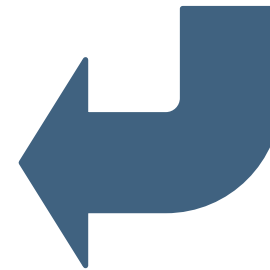
T1.col1	T1.col2
10	15
20	25
30	35



T2.col1	T2.col2
10	BB
15	DD



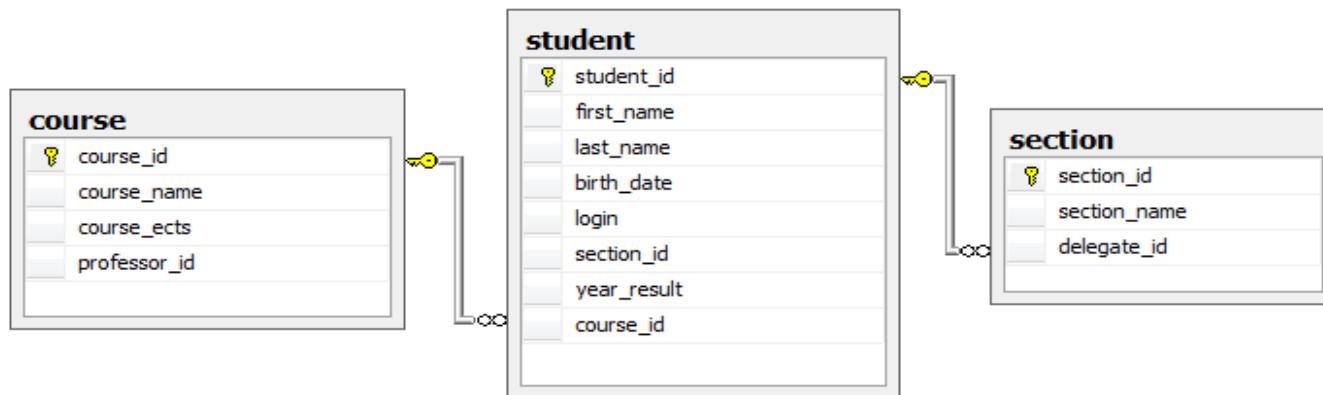
T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB
NULL	NULL	15	DD



Jointures : RIGHT OUTER JOIN

```
SELECT first_name + ' ' + last_name  
      , section_name, course_name  
FROM   course C RIGHT JOIN student St  
      ON St.course_id = C.course_id  
   LEFT JOIN section S  
      ON St.student_id = S.delegate_id
```

Liste des étudiants, la section dont ils sont *éventuellement* délégués ainsi que le cours auquel ils sont *éventuellement* inscrits



Jointures : RIGHT OUTER JOIN

course_id	course_name	course_ects	professor_id
EG1020	Derivatives	3.0	3
EG2110	Marketing management	3.5	2
EG2210	Financial Management	4.0	3
EING2283	Marketing engineering	4.0	1
EING2383	Supply chain management et e-business	2.5	5

student_id	first_name	last_name	course_id
1	Georges	Lucas	EG2210
2	Clint	Eastwood	EG2210
3	Sean	Connery	EG2110
4	Robert	De Niro	EG2210
5	Kevin	Bacon	0
6	Kim	Basinger	0
7	Johnny	Depp	EG2210

section_id	section_name	delegate_id
1010	BSc Management	12
1020	MSc Management	9
1110	BSc Economics	15
1120	MSc Economics	6
1310	BA Sociology	23
1320	MA Sociology	6

```

SELECT first_name + ' ' + last_name
, section name, course name
FROM course C RIGHT JOIN student St
ON St.course id = C.course id
LEFT JOIN section S
ON St.student id = S.delegate_id
    
```

Nom étudiant	Section représentée	Cours choisi
Robert De Niro	NULL	Financial Management
Kevin Bacon	NULL	NULL
Kim Basinger	MSc Economics	NULL
Kim Basinger	MA Sociology	NULL
Johnny Depp	NULL	Financial Management
Julia Roberts	NULL	NULL
Natalie Portman	MSc Management	Financial Management
Georges Clooney	NULL	Marketing management

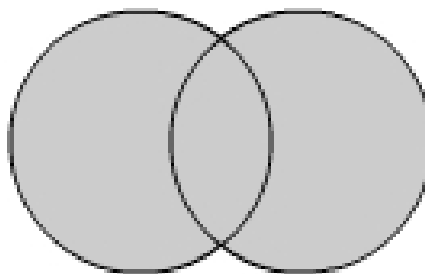
Certains étudiants ne sont pas délégué de section, certains sont délégués de 2 sections, certains étudiants ne sont inscrits dans aucun cours, mais la liste de tous les étudiants doit apparaître quoiqu'il en soit

Jointures : **FULL OUTER JOIN**

```
SELECT *  
FROM table1 T1 FULL JOIN table2 T2 ON T1.col1 = T2.col1
```

Le « **FULL OUTER JOIN** » est une combinaison du **LEFT** et du **RIGHT** qui met en relation les lignes qui ont des éléments communs dans les colonnes indiquées et affiche toutes les autres lignes des deux tables, même si elles n'ont pas de point commun (mot-clé « **OUTER** » facultatif *sous SQL-Server*)

T1.col1	T1.col2
10	15
20	25
30	35



T2.col1	T2.col2
10	BB
15	DD



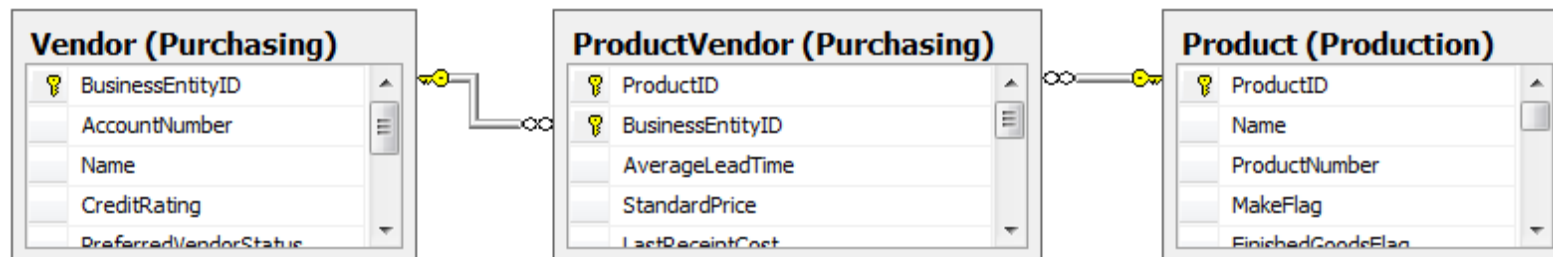
T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB
20	25	NULL	NULL
30	35	NULL	NULL
NULL	NULL	15	DD



Jointures : SELF-JOIN

```
SELECT *  
FROM table1 T1 JOIN table1 T2 ON T1.col1 = T2.col1
```

Le « **SELF-JOIN** » n'est rien d'autre qu'un « **INNER JOIN** » dans lequel les deux tables sont des copies de la même table d'origine. On utilise un « **SELF-JOIN** » lorsqu'on compare des éléments au sein de la même table. Les alias font en sorte que le système traite la requête comme un « **INNER JOIN** » classique, considérant les alias comme deux tables distinctes



La table « **ProductVendor** » de la base de données « **AdventureWorks** », représente le lien « **Many-to-Many** » entre les tables « **Vendor** » et « **Product** », mettant en relation quel vendeur a vendu quel produit
À partir de la table « **ProductVendor** », nous aimerions savoir quel produit a été vendu par plus d'un vendeur

Jointures : SELF-JOIN

```
SELECT pv1.ProductID, pv1.BusinessEntityID
FROM Purchasing.ProductVendor pv1
     INNER JOIN Purchasing.ProductVendor pv2
     ON pv1.ProductID = pv2.ProductID
        AND pv1.BusinessEntityID <> pv2.BusinessEntityID
```

Numero produit	Numero vendeur
1	1580
2	1688
4	1650
317	1578
317	1678
318	1578
318	1678
319	1556
319	1578
319	1678
320	1514
320	1602

Table « pv1 »

Numero Produit	Numero vendeur
317	1578
317	1678
318	1578
318	1678
319	1556
319	1578
319	1678
320	1602
320	1604
320	1514
321	1514
321	1602

Liste des produits vendus par plus d'un vendeur

Numero produit	Numero vendeur
1	1580
2	1688
4	1650
317	1578
317	1678
318	1578
318	1678
319	1556
319	1578
319	1678
320	1514
320	1602

Table « pv2 »

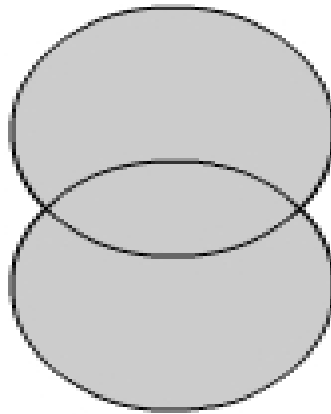
Jointures verticales

```
SELECT ... FROM ... WHERE ... GROUP BY ...  
opérateur_comparaison_requêtes  
SELECT ... FROM ... WHERE ... GROUP BY ...  
ORDER BY ...
```

- Les jointures verticales comparent le résultat de **deux requêtes indépendantes**
- La comparaison des requêtes n'est possible que si chacune d'elles renvoie ***le même nombre de colonnes*** et que les colonnes en vis-à-vis sont du ***même type***
- L'affichage final résultant utilisera le nom des colonnes ou des alias utilisés ***dans la première requête***, il n'est donc pas nécessaire de donner des alias aux colonnes de la seconde
- Chaque requête peut contenir ***autant de clauses nécessaires*** à sa bonne réalisation (SELECT, FROM + JOIN, WHERE, GROUP BY, ...) à l'exception de la clause « **ORDER BY** » qui, si elle est utilisée, ***triera le résultat final*** résultant de la comparaison des deux requêtes. Il faudra toujours placer la clause « **ORDER BY** » à la suite de la deuxième requête

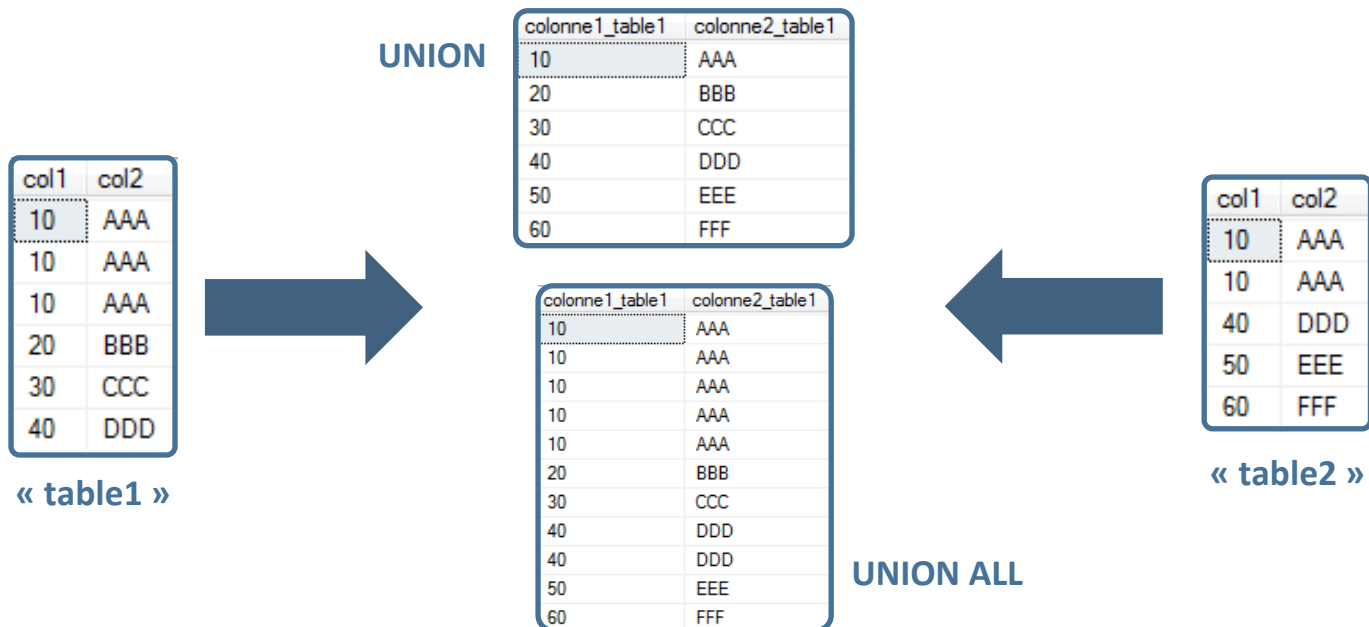
Jointures : **UNION [ALL]**

- L'opérateur « **UNION** » applique un « **DISINCT** » aux résultats des deux requêtes et ajoute ensuite les lignes renvoyées par la seconde requête à celles présentées par la première, si elles sont différentes
- Le mot clé « **ALL** » peut être ajouté à l'opérateur « **UNION** » afin qu'absolument toutes les lignes ramenées par chacune des requêtes soient affichées, lignes déjà présentes dans le résultat de la première requête et doublons compris



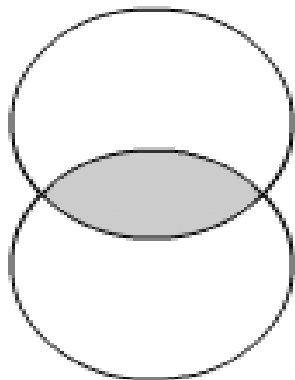
Jointures : UNION [ALL]

```
SELECT col1 as [colonne1_table1], col2 as [colonne2_table1]
FROM table1
UNION
SELECT col1 as [colonne1_table2], col2 as [colonne2_table2]
FROM table2
ORDER BY [colonne2_table1]
```

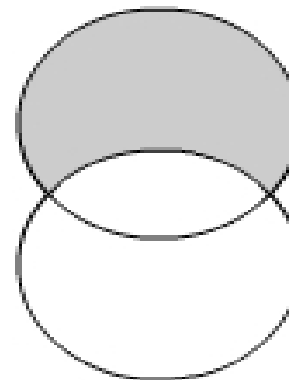


Jointures : **INTERSECT** et **EXCEPT**

- L'opérateur « **INTERSECT** » n'affiche les lignes de la première requête que si elles se retrouvent également dans la seconde. **Les lignes en double ne sont comparées qu'une seule fois**
- L'opérateur « **EXCEPT** » n'affiche les lignes de la première requête que si elles **NE** se retrouvent **PAS** dans la seconde. **Les lignes en double ne sont comparées qu'une seule fois**
- « **INTERSECT ALL** » et « **EXCEPT ALL** » ne sont pas reconnus



INTERSECT



EXCEPT

Jointures : INTERSECT

```
SELECT * FROM table1  
INTERSECT  
SELECT * FROM table2
```

col1	col2
10	AAA
10	AAA
10	AAA
20	BBB
30	CCC
40	DDD

« table1 »



col1	col2
10	AAA
40	DDD



col1	col2
10	AAA
10	AAA
40	DDD
50	EEE
60	FFF

« table2 »

The 'ALL' version of the INTERSECT operator is not supported.

Jointures : EXCEPT

```
SELECT * FROM table1  
EXCEPT  
SELECT * FROM table2
```

La version Oracle de l'opérateur « EXCEPT » est « MINUS »

col1	col2
10	AAA
10	AAA
10	AAA
20	BBB
30	CCC
40	DDD

« table1 »



col1	col2
20	BBB
30	CCC

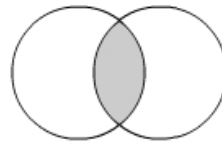


col1	col2
10	AAA
10	AAA
40	DDD
50	EEE
60	FFF

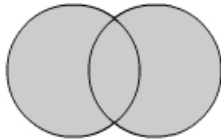
« table2 »

The 'ALL' version of the EXCEPT operator is not supported.

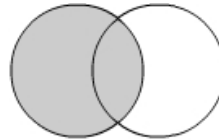
Jointures : Résumé



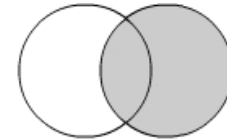
INNER JOIN



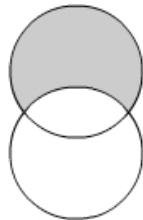
FULL OUTER JOIN



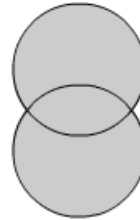
LEFT OUTER JOIN



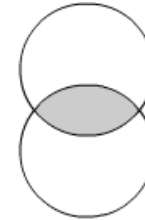
RIGHT OUTER JOIN



EXCEPT



UNION



INTERSECT

Auto-Evaluation

N'oubliez pas de prendre le temps d'évaluer le niveau de maîtrise que vous estimez avoir acquis personnellement concernant les notions abordées dans ce module !

Rappel de la signification des lettres dans les tableaux d'auto-évaluation :

- **Parfait (P)** : vous avez parfaitement compris cette notion et vous vous sentez à votre aise
- **Satisfaisant (S)** : vous avez compris de quoi il s'agit mais la pratique vous manque
- **Vague (V)** : vous savez de quoi il s'agit, mais cela reste un peu vague dans votre esprit. Une explication supplémentaire du formateur ou une bonne révision de votre part s'impose
- **Insatisfaisant (I)** : Vous n'avez pas du tout compris la notion abordée, il faut tout faire pour y remédier !

Auto-Evaluation

Notions à évaluer

Notions	P	S	V	I
Différence entre jointures « horizontales » et « verticales »				
« CROSS JOIN »				
Equi-join (« INNER JOIN » entre plusieurs tables)				
« LEFT/RIGHT/FULL OUTER JOIN »				
SELF-JOIN				
Opérateurs « UNION »				

Sous-Requêtes

```
SELECT ... FROM ...  
WHERE (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...)  
GROUP BY ... ORDER BY ...
```

```
SELECT ...  
FROM (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...) AS T1  
WHERE ... GROUP BY ... ORDER BY ...
```

```
SELECT ... FROM ... WHERE ...  
GROUP BY ... HAVING (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...)  
GROUP BY ... ORDER BY ...
```

Sous-Requêtes

- Une « **sous-requête** » est une **requête évaluée à l'intérieur d'une autre** requête et dont le résultat influence le résultat de la requête principale
- Une sous-requête est **toujours placée entre parenthèses**
- Il est possible d'utiliser une sous-requête **dans la clause « FROM », « WHERE » ou « HAVING »**
- Il est important de **bien visualiser le résultat produit par la requête imbriquée** afin de l'utiliser correctement dans la requête principale. Dans un premier, n'oublions pas qu'il est possible de n'exécuter qu'**une partie du code en le surlignant**, lorsqu'on travaille avec Microsoft SQL Server Management Studio

Sous-Requêtes : **WHERE** et **HAVING**

```
SELECT ... FROM ...  
WHERE (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...)  
GROUP BY ... ORDER BY ...
```

```
SELECT ... FROM ... WHERE ...  
GROUP BY ... HAVING (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...)  
GROUP BY ... ORDER BY ...
```

- Lors de l'utilisation de requêtes imbriquées dans les conditions posées par le « **WHERE** » ou le « **HAVING** », il est indispensable que les données renvoyées soient **cohérente avec l'expression** dans laquelle elles sont utilisées (nombre de valeurs et type)
- Les données renvoyées seront de trois types :
 - **Scalaires** (une seule valeur)
 - **Multi-valeurs** (un ensemble de données scalaires, soit une colonne entière)
 - **Tabulaire** (un ensemble de lignes et de colonnes)

Sous-Requêtes : WHERE et HAVING

Scalar-valued subquery : « WHERE »

```
SELECT last_name, year_result
FROM student
WHERE year_result >= (SELECT year_result FROM student
                      WHERE last_name LIKE 'Bacon')
```

Si la valeur renvoyée par la sous-requête est **une valeur scalaire**, alors il est tout à fait possible d'utiliser les **opérateurs classiques d'inégalité** dans l'expression

last_name	year_result
Bacon	16
Basinger	19
Roberts	17
Garcia	19
Gamer	18
Berry	18

← Valeur renvoyée par la sous-requête

Liste des étudiants dont le résultat annuel est plus grand ou égal au résultat de M. Bacon

Sous-Requêtes : WHERE et HAVING

Scalar-valued subquery : « WHERE »

```
SELECT last_name, year_result
FROM student
WHERE year_result > (SELECT AVG(year_result)
                     FROM student)
```

→ 8

Une agrégation globale fonctionne bien également puisqu'elle renvoie *une seule valeur*

last_name	year_result
Lucas	10
Connery	12
Bacon	16
Basinger	19
Depp	11
Roberts	17
Garcia	19
Gamer	18
Reeves	10
Berry	18

Liste des étudiants ayant un résultat plus élevé que la moyenne

Sous-Requêtes : WHERE et HAVING

Scalar-valued subquery : « HAVING »

```
SELECT section_id, AVG(year_result) as [Moyenne]
FROM student
GROUP BY section_id
HAVING AVG(year_result) > (SELECT AVG(year_result)
                           FROM student)
```

section_id	Moyenne
1120	17
1310	11
1320	10

Liste des sections dont la moyenne est plus grande que la moyenne globale


Sous-Requêtes : WHERE et HAVING

Multi-valued subquery : « [NOT] IN » operator

```
SELECT last_name, year_result
FROM student
WHERE year_result IN (SELECT MAX(year_result)
                      FROM student
                      GROUP BY section_id)
```

Si la sous-requête renvoie plus d'une valeur, il devient impossible d'utiliser les **opérateurs classiques d'inégalité**. L'opérateur « **IN** » permettra de comparer la valeur d'une colonne à chaque élément de la liste des valeurs renvoyées par la sous-requête, par exemple

```
SELECT MAX(year_result)
FROM student
GROUP BY section_id
```



6
12
19
18
19
18

Si le résultat annuel de l'étudiant est égal à au moins l'une des valeurs renvoyées par la sous-requête, les données sont affichées

last_name	year_result
Connery	12
Basinger	19
Garcia	19
Willis	6
Marceau	6
Gamer	18
Berry	18


Sous-Requêtes : WHERE et HAVING

Multi-valued subquery : « ANY » operator

```
SELECT last_name, year_result
FROM student
WHERE year_result > ANY (SELECT MAX(year_result)
                        FROM student
                        GROUP BY section_id)
```

L'opérateur « **ANY** » peut être utilisé en plus des opérateurs d'*inégalité classiques* afin de comparer la valeur d'une colonne individuellement à chacune des valeurs de la liste renvoyée par la sous-requête. Si ***au moins l'une des comparaisons*** renvoie **TRUE**, les données sont affichées

```
SELECT MAX(year_result)
FROM student
GROUP BY section_id
```



6
12
19
18
19
18

Si le résultat annuel de l'étudiant est supérieur à au moins l'une des valeurs renvoyées par la sous-requête, les données sont affichées

last_name	year_result
Witherspoon	7
Michelle Gellar	7
Milano	7
Hanks	8
Reeves	10
Lucas	10
Depp	11
Copper	12

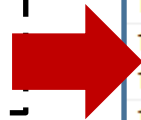
Sous-Requêtes : WHERE et HAVING

Multi-valued subquery : « ALL » operator

```
SELECT last_name, year_result
FROM student
WHERE year_result >= ALL (SELECT MAX(year_result)
                           FROM student
                           GROUP BY section_id)
```

L'opérateur « **ALL** » peut être utilisé en plus des opérateurs d'*inégalité classiques* afin de comparer la valeur d'une colonne individuellement à chacune des valeurs de la liste renvoyée par la sous-requête. Si **toutes les comparaisons** renvoient **TRUE**, les données sont affichées

```
SELECT MAX(year_result)
FROM student
GROUP BY section_id
```



Maximum par section	
6	
12	
19	
18	
19	
18	

last_name	year_result
Basinger	19
Garcia	19

Si le résultat annuel de l'étudiant est supérieur ou égal à toutes les valeurs renvoyées par la sous-requête, les données sont affichées

Sous-Requêtes : [NOT] EXISTS

```
SELECT last_name  
FROM student as s  
WHERE EXISTS (SELECT * FROM inscriptions as i  
              WHERE i.student_id = s.student_id  
              AND i.course_id = 'EING2234')
```

- L'opérateur « **EXISTS** » permet de n'afficher les données demandées que si le résultat de la sous-requête produit au moins une ligne de données (le nombre de lignes renvoyées par la sous-requête n'a pas d'importance)
- Ce résultat est donc de **type tabulaire** et bien souvent **corrélé**, c'est-à-dire qu'il tient compte des données contenues dans la requête principale
- Le mot-clé « **NOT** » peut être ajouté devant l'opérateur « **EXISTS** » afin de formuler la négation

Sous-Requêtes : [NOT] EXISTS

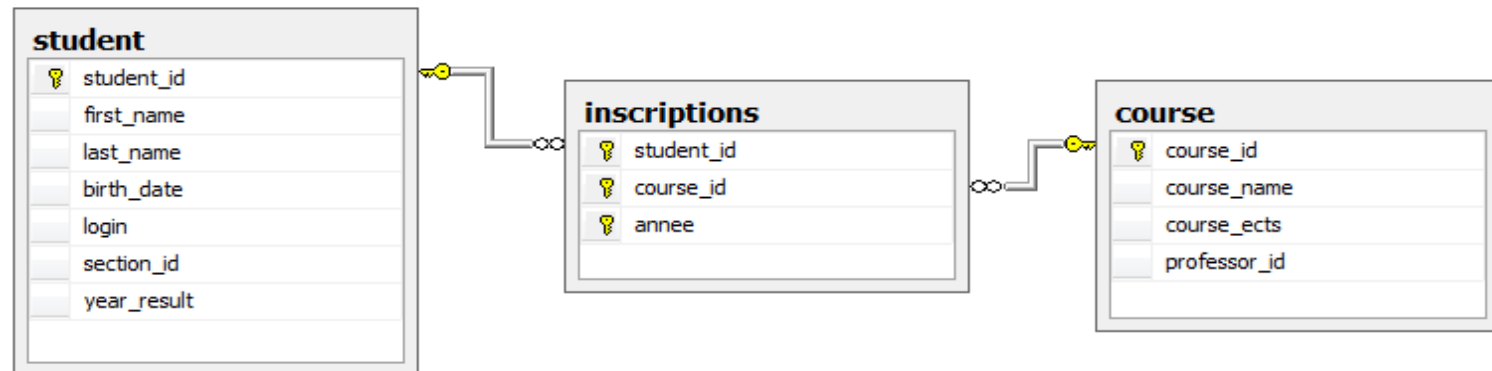


Table
« STUDENT »

student_id	last_name
1	Lucas
2	Eastwood
3	Connery
4	De Niro
5	Bacon
6	Basinger
7	Depp
8	Roberts
9	Portman
10	Clooney

student_id	course_id	annee
1	ECGE2183	1960-09-01
1	FING2283	1960-09-01
3	EING2234	1960-09-01
3	EING2283	1960-09-01
3	EING2383	1960-09-01
4	EING2283	1960-09-01
6	EING2234	1960-09-01
9	EING2234	1960-09-01
9	EING2383	1960-09-01

Table
« INSCRIPTIONS »

Étudiants inscrits au cours EING2234

```
SELECT student_id, last_name
FROM student as s
WHERE EXISTS (
  SELECT *
  FROM inscriptions as i
  WHERE i.student_id = s.student_id
  AND i.course_id = 'EING2234')
```

student_id	last_name
3	Connery
6	Basinger
9	Portman

Sous-Requêtes : **FROM** et **WITH**

```
SELECT ...  
FROM (SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...) AS T1  
WHERE ... GROUP BY ... ORDER BY ...
```



```
WITH table_CTE (nom_col1, nom_col2, nom_col3, ..., nom_colN)  
AS  
(SELECT ... FROM ... WHERE ... GROUP BY ... ORDER BY ...)  
SELECT ... FROM table_CTE WHERE ... GROUP BY ... ORDER BY ...
```

CTE = Common Table Expression

Sous-Requêtes : FROM et WITH

- Une sous-requête de **type tabulaire** (renvoyant plusieurs lignes et plusieurs colonnes) peut être **traitée comme une table** à part entière et servir de référence pour la requête principale
- Dans le cas où la sous-requête est utilisée dans une clause « **FROM** », il est **nécessaire de lui donner un alias** afin de l'utiliser comme un nom de table dans la requête principale
- Il faudra toujours donner un alias aux colonnes affichant le résultat d'une expression
- Lors de l'utilisation d'une requête imbriquée avec la clause « **WITH** », la requête sert à fournir la table pré-déclarée et doit renvoyer le même nombre de colonnes qu'annoncé dans la clause « **WITH** »
- La plupart des systèmes montrent de **meilleures performances** avec l'utilisation de la clause « **WITH** », mais cela ne doit pas devenir une généralité. Certains cas d'utilisation peuvent démontrer le contraire au sein du même système

Sous-Requêtes : FROM et WITH

```
SELECT section_name as [Section], Nbr as [Nombre d'étudiants]
FROM (SELECT section_id, COUNT(*) as [Nbr] FROM student
      GROUP BY section_id) as std
JOIN section as s ON s.section_id = std.section_id
WHERE Nbr > 5
```



```
WITH std (section_id, Nbr)
AS
( SELECT section_id, COUNT(*) as [Nbr]
  FROM student
 GROUP BY section_id)
SELECT section_name as [Section], Nbr as [Nombre d'étudiants]
FROM std JOIN section as s ON s.section_id = std.section_id
WHERE Nbr > 5
```

Liste des sections contenant plus de 5 étudiants

Sous-Requêtes : FROM et WITH

```
WITH DirectReports(Name, Title, EmployeeID, EmployeeLevel, Sort, ManagerID)
AS (SELECT CONVERT(varchar(255), e.FirstName + ' ' + e.LastName),
      e.Title, e.EmployeeID, 1,
      CONVERT(varchar(255), e.FirstName + ' ' + e.LastName),
      ManagerID
FROM dbo.MyEmployees AS e
WHERE e.ManagerID IS NULL
UNION ALL
SELECT CONVERT(varchar(255), REPLICATE ('|      ', EmployeeLevel) +
      e.FirstName + ' ' + e.LastName),
      e.Title, e.EmployeeID, EmployeeLevel + 1,
      CONVERT (varchar(255), RTRIM(Sort) + '|      ' +
      FirstName + ' ' + LastName),
      e.ManagerID
FROM dbo.MyEmployees AS e
JOIN DirectReports AS d ON e.ManagerID = d.EmployeeID
)
SELECT EmployeeID, Name, Title, EmployeeLevel, ManagerID
FROM DirectReports
ORDER BY Sort
```

Clause « **WITH** » utilisée dans le cadre de l'**affichage hiérarchique** des employés d'une société

Sous-Requêtes : FROM et WITH

Table
« MYEMPLOYEES »

EmployeeID	FirstName	LastName	Title	DeptID	ManagerID
1	Ken	Sánchez	Chief Executive Officer	16	NULL
16	David	Bradley	Marketing Manager	4	273
23	Mary	Gibson	Marketing Specialist	4	16
273	Brian	Welcker	Vice President of Sales	3	1
274	Stephen	Jiang	North American Sale...	3	273
275	Michael	Blythe	Sales Representative	3	274
276	Linda	Mitchell	Sales Representative	3	274
285	Syed	Abbas	Pacific Sales Manager	3	273
286	Lynn	Tsoflias	Sales Representative	3	285

EmployeeID	Name	Title	EmployeeLevel	ManagerID
1	Ken Sánchez	Chief Executive Officer	1	NULL
273	Brian Welcker	Vice President of Sales	2	1
16	David Bradley	Marketing Manager	3	273
23	Mary Gibson	Marketing Specialist	4	16
274	Stephen Jiang	North American Sales Manager	3	273
276	Linda Mitchell	Sales Representative	4	274
275	Michael Blythe	Sales Representative	4	274
285	Syed Abbas	Pacific Sales Manager	3	273
286	Lynn Tsoflias	Sales Representative	4	285

Résultat de
la requête du slide
précédent

Sous-Requêtes : JOIN VS Sous-requête

```
SELECT DISTINCT course_name  
  FROM course  
 WHERE course_id IN (SELECT course_id FROM inscriptions )
```

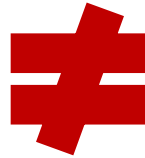


```
SELECT DISTINCT course_name  
FROM course C JOIN inscriptions I  
  ON C.course_id = I.course_id
```

Sous-Requêtes : JOIN VS Sous-requête

```
SELECT DISTINCT course_name  
FROM course  
WHERE course_id NOT IN (SELECT course_id FROM inscriptions )
```

Retourne le nom du cours de la table « **Course** » s'il n'existe pas dans la table « **Inscriptions** »



```
SELECT DISTINCT course_name  
FROM course C JOIN inscriptions I  
ON C.course_id <> I.course_id
```

Retourne le nom du cours de la table « **Course** » s'il est différent de l'un des cours de la table « **Inscriptions** »

Auto-Evaluation

N'oubliez pas de prendre le temps d'évaluer le niveau de maîtrise que vous estimez avoir acquis personnellement concernant les notions abordées dans ce module !

Rappel de la signification des lettres dans les tableaux d'auto-évaluation :

- **Parfait (P)** : vous avez parfaitement compris cette notion et vous vous sentez à votre aise
- **Satisfaisant (S)** : vous avez compris de quoi il s'agit mais la pratique vous manque
- **Vague (V)** : vous savez de quoi il s'agit, mais cela reste un peu vague dans votre esprit. Une explication supplémentaire du formateur ou une bonne révision de votre part s'impose
- **Insatisfaisant (I)** : Vous n'avez pas du tout compris la notion abordée, il faut tout faire pour y remédier !

Auto-Evaluation

Notions à évaluer

Notions	P	S	V	I
Types de sous-requêtes (scalaire, multi-valeur, tabulaire)				
Sous-requêtes dans les clauses « WHERE » et « HAVING »				
Opérateurs « ALL » et « ANY »				
Opérateur « EXISTS »				
Sous-requêtes dans la clause « FROM »				
Clause « WITH »				