

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 :
 - **Scalaire** (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

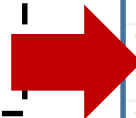
Multi-valued subquery : « [NOT] IN » operator

```
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
```

Maximum par section
6
12
19
18
19
18



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

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
```

Maximum par section
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

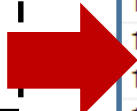
Sous-Requêtes : WHERE et HAVING

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

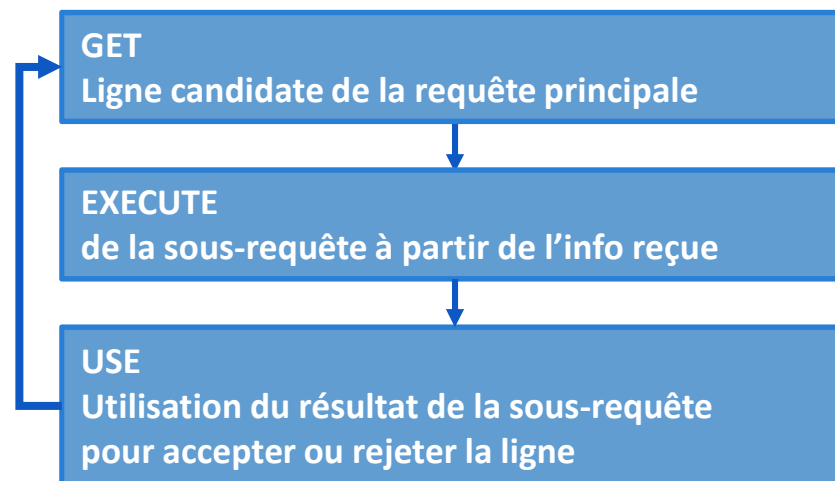
last_name	year_result
Basinger	19
Garcia	19

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 : Corrélation

```
SELECT ... FROM table1 as T1  
WHERE (SELECT ... FROM table1 as T2 WHERE T1.col1 = T2.col1 ...) ...  
GROUP BY ... ORDER BY ...
```

Une requête « **corrélée** » signifie que le résultat renvoyé par la sous-requête dépend directement de la ligne actuellement rencontrée par la requête principale. Le résultat de la sous-requête est donc réévalué et potentiellement différent à chaque ligne rencontrée dans la requête principale



Sous-Requêtes : Corrélation

```
SELECT last_name, section_id, year_result
FROM student AS s1
WHERE year_result > (SELECT AVG(year_result)
                     FROM student
                     WHERE section_id = s1.section_id )
```

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

Table « S1 »

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

Table « STUDENT »

section_id	Moyenne par section
1010	4
1020	7
1110	8
1120	17
1310	11
1320	10 < 18

Moyennes

last_name	section_id	year_result
Basinger	1310	19
Berry	1320	18
Connery	1020	12
Depp	1110	11
Garcia	1110	19
Gamer	1120	18
Hanks	1020	8
Reeves	1020	10
Willis	1010	6

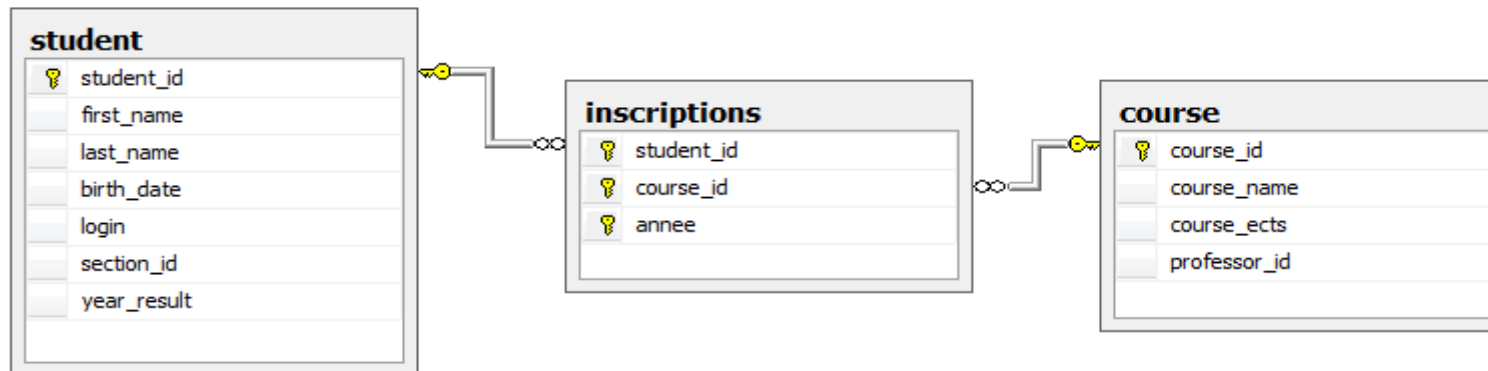
Résultat

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



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')
    
```

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 »

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) + ' ' +
      e.FirstName + ' ' + e.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 )
```

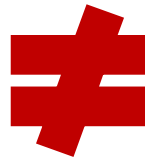


```
SELECT DISTINCT course_name  
FROM course C JOIN inscriptions I  
  ON C.course_id = I.course_id
```

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 »				
Sous-requête corrélée				
Opérateur « EXISTS »				
Sous-requêtes dans la clause « FROM »				
Clause « WITH »				