

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 ...
```

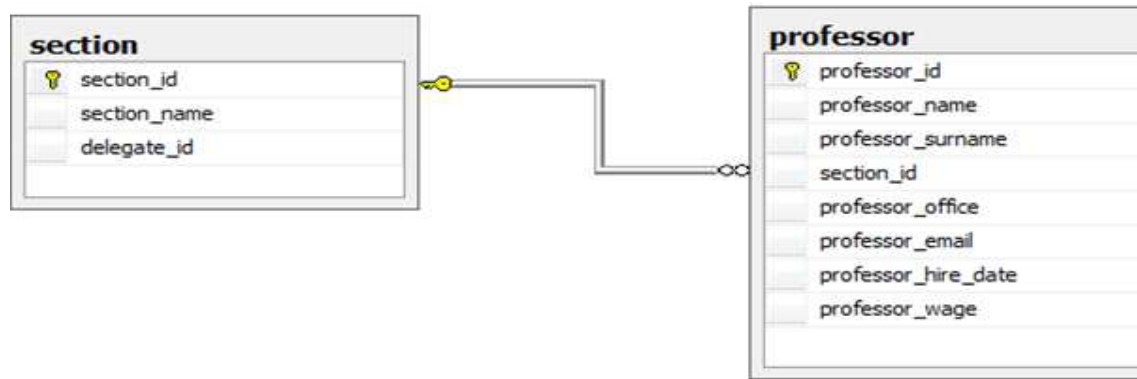
Comparaison des colonnes des tables entre elles

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

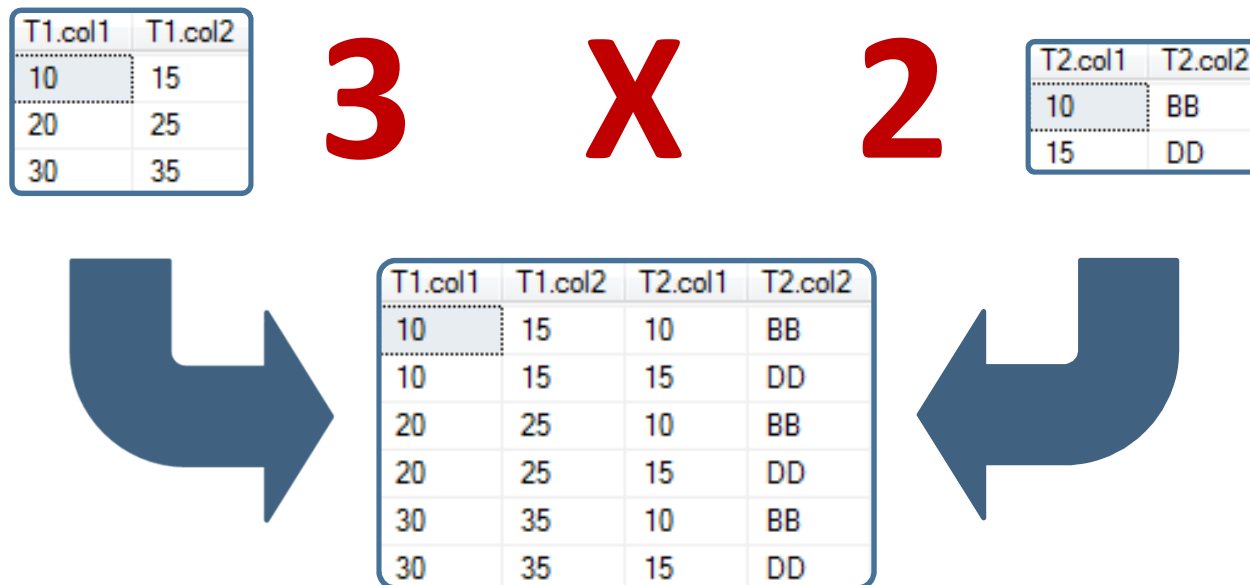


- La jointure horizontale compare deux colonnes entre elles et affiche les colonnes souhaitées pour chaque concordance trouvée
- La condition de la jointure (c'est-à-dire la comparaison à faire) utilisera souvent les **clés primaires et étrangères** liant les tables (mais ce n'est pas obligatoire)
- 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 travail 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

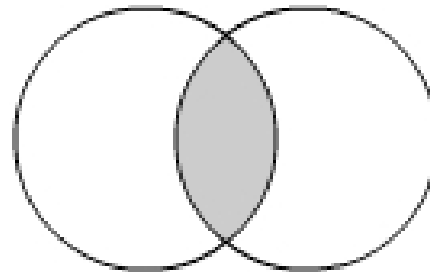


Jointures : **INNER JOIN**

```
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*)

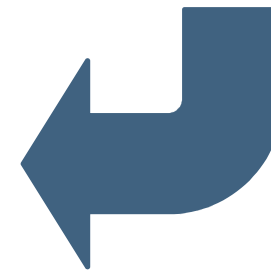
T1.col1	T1.col2
10	15
20	25
30	35



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 :

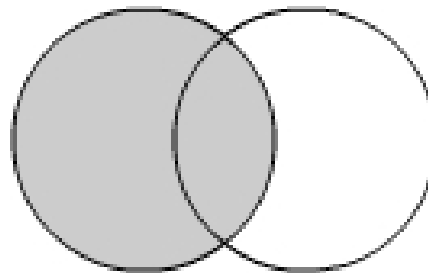
Ambiguous column name 'section_id'.

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*)

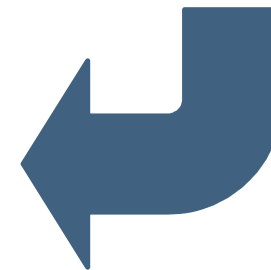
T1.col1	T1.col2
10	15
20	25
30	35



T2.col1	T2.col2
10	BB
15	DD



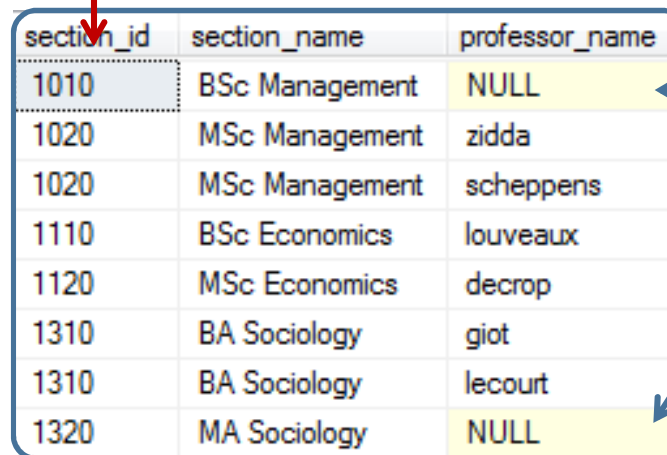
T1.col1	T1.col2	T2.col1	T2.col2
10	15	10	BB
20	25	NULL	NULL
30	35	NULL	NULL



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

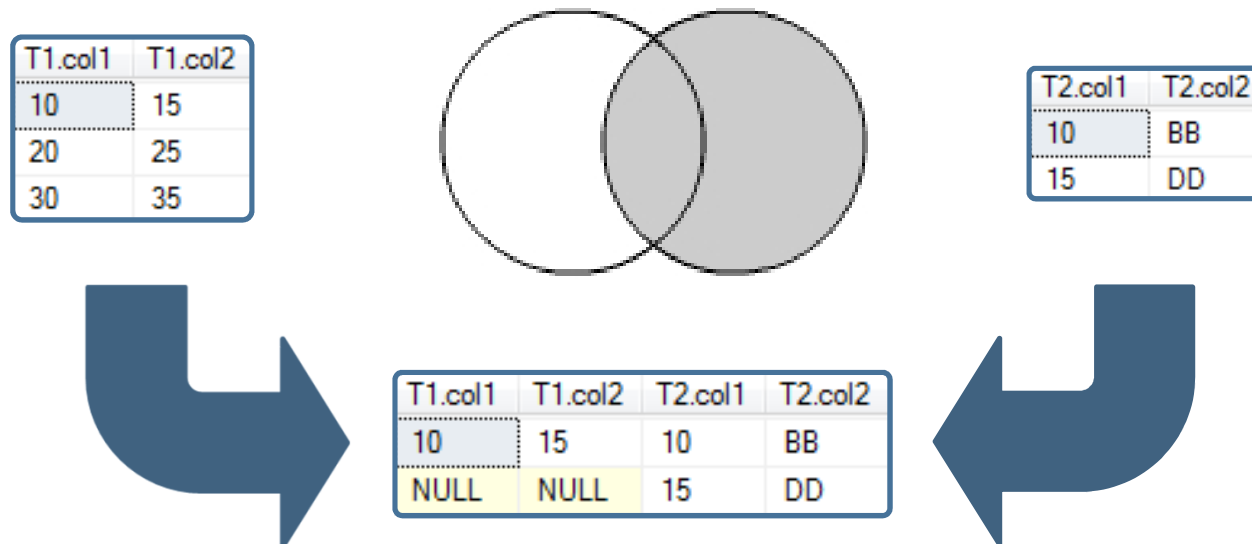
On désire afficher les informations sur toutes les sections, qu'il y ai un professeur qui y soit inscrit ou non

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
```

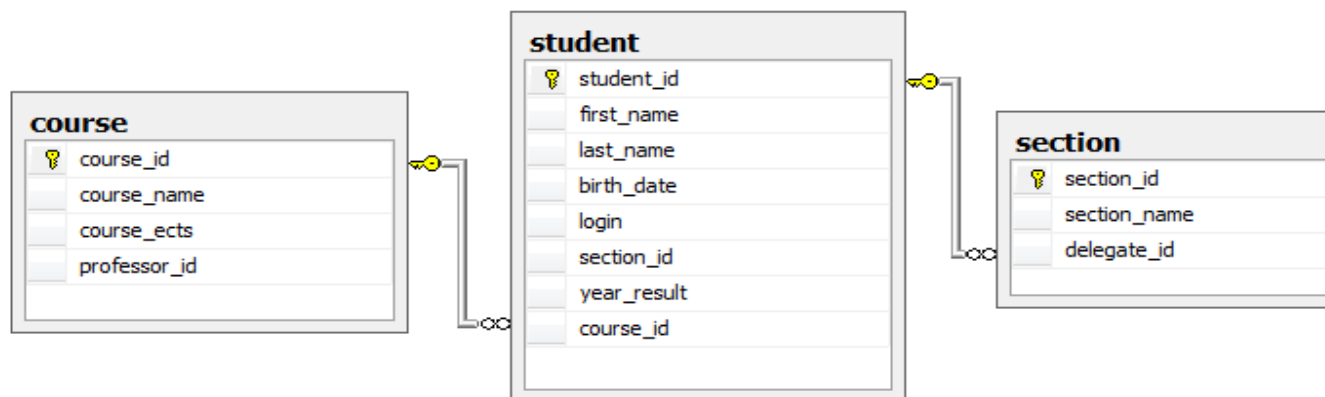
Le « **RIGHT OUTER JOIN** » fonctionne de la même manière que le **LEFT**, mais concerne la seconde table de la jointure (mot-clé « **OUTER** » facultatif *sous SQL-Server*)



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
    
```

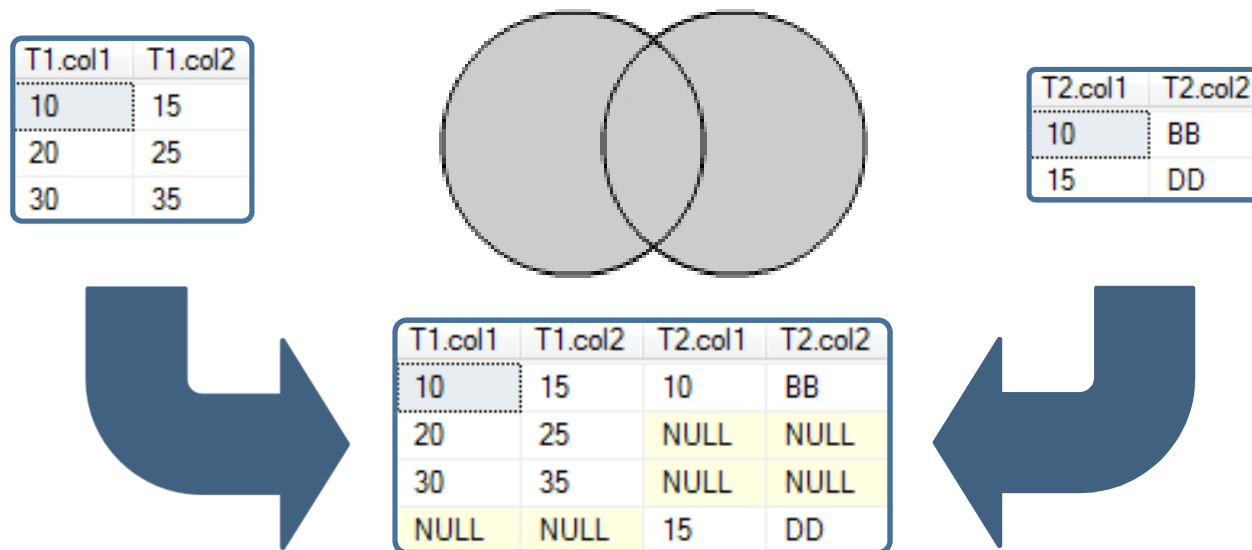
Nom étudiant	Section représentée	Cours choisi
Robert De Niro	NULL	Financial Management
Kevin Bacon	NULL	NULL
Kim Basinger	MSc Economics	NULL
Kim Basinger	MA Sociology	NULL
Johnny Depp	NULL	Financial Management
Julia Roberts	NULL	NULL
Natalie Portman	MSc Management	Financial Management
Georges Clooney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



Jointures : EQUI-JOIN

```
SELECT C.course_name, P.professor_name, S.section_name
FROM course C, professor P, section S
WHERE C.professor_id = P.professor_id
      AND P.section_id = S.section_id
```



```
SELECT C.course_name, P.professor_name, S.section_name
FROM (course C JOIN professor P
      ON C.professor_id = P.professor_id)
     JOIN section S
      ON P.section_id = S.section_id
```

Un « ***ÉQUI-JOIN*** » est le terme employé lorsque la condition de jointure est basée sur des ***égalités strictes*** entre les colonnes comparées

Jointures : EQUI-JOIN

course_id	course_name	professor_id
ECGE2183	Financial Management	3
ECGE2184	Marketing management	2
EING2234	Derivatives	3
EING2283	Marketing engineering	1
EING2383	Supply chain management et e-business	5

Table « COURSE »

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

Table
« PROFESSOR »

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

Table
« SECTION »

```

SELECT C.course_name, P.professor_name, S.section_name
FROM (course C JOIN professor P
      ON C.professor_id = P.professor_id)
      JOIN section S
      ON P.section_id = S.section_id
    
```



course_name	prof_name	section_name
Financial Management	giot	BA Sociology
Marketing management	decrop	MSc Economics
Derivatives	giot	BA Sociology
Marketing engineering	zidda	MSc Management
Supply chain manage...	scheppens	MSc Management

Jointures : **NON EQUI-JOIN**

```
SELECT S.last_name, S.year_result, G.Grade  
FROM Grade G, student S  
WHERE S.year_result BETWEEN G.Lower_bound AND G.Upper_bound
```



```
SELECT S.last_name, S.year_result, G.Grade  
FROM Grade G JOIN student S  
ON S.year_result BETWEEN G.Lower_bound AND G.Upper_bound
```

Un « **NON ÉQUI-JOIN** » est le terme employé lorsque la condition de jointure n'est pas basée sur des **égalités strictes** entre les colonnes comparées

Jointures : NON EQUI-JOIN

```
SELECT S.last_name, S.year_result, G.Grade
FROM Grade G JOIN student S
ON S.year_result BETWEEN G.Lower_bound AND G.Upper_bound
```

Table
« STUDENT »

last_name	year_result
Lucas	10
Eastwood	4
Connery	12
De Niro	3
Bacon	16
Basinger	19
Depp	11

Table
« GRADE »

lower_bound	upper_bound	grade
14	15	B
18	20	E
10	11	F
8	9	I
0	7	IG
12	13	S
16	17	TB

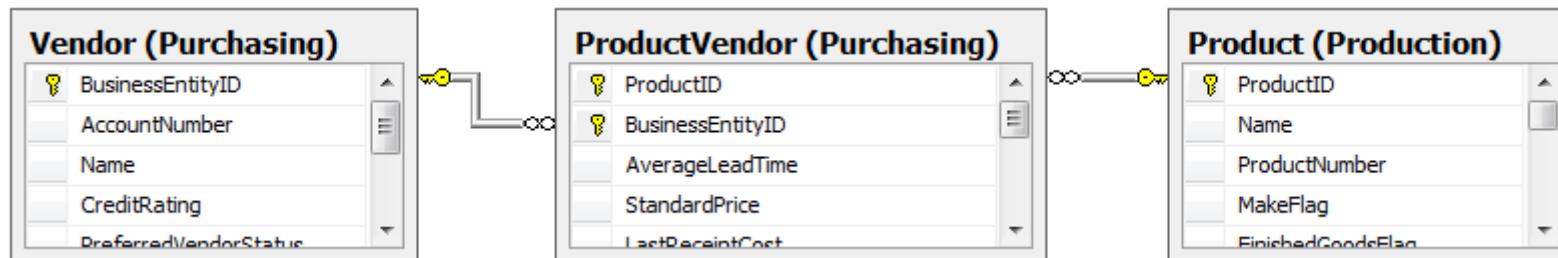


last_name	year_result	Grade
Basinger	19	E
Garcia	19	E
Gamer	18	E
Berry	18	E
Lucas	10	F
Depp	11	F
Reeves	10	F
Hanks	8	I
Eastwood	4	IG
De Niro	3	IG
Portman	4	IG
Clooney	4	IG

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 »

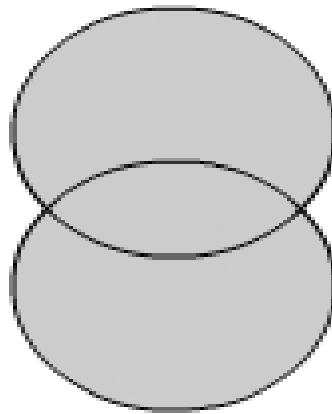
Jointures verticales

```
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

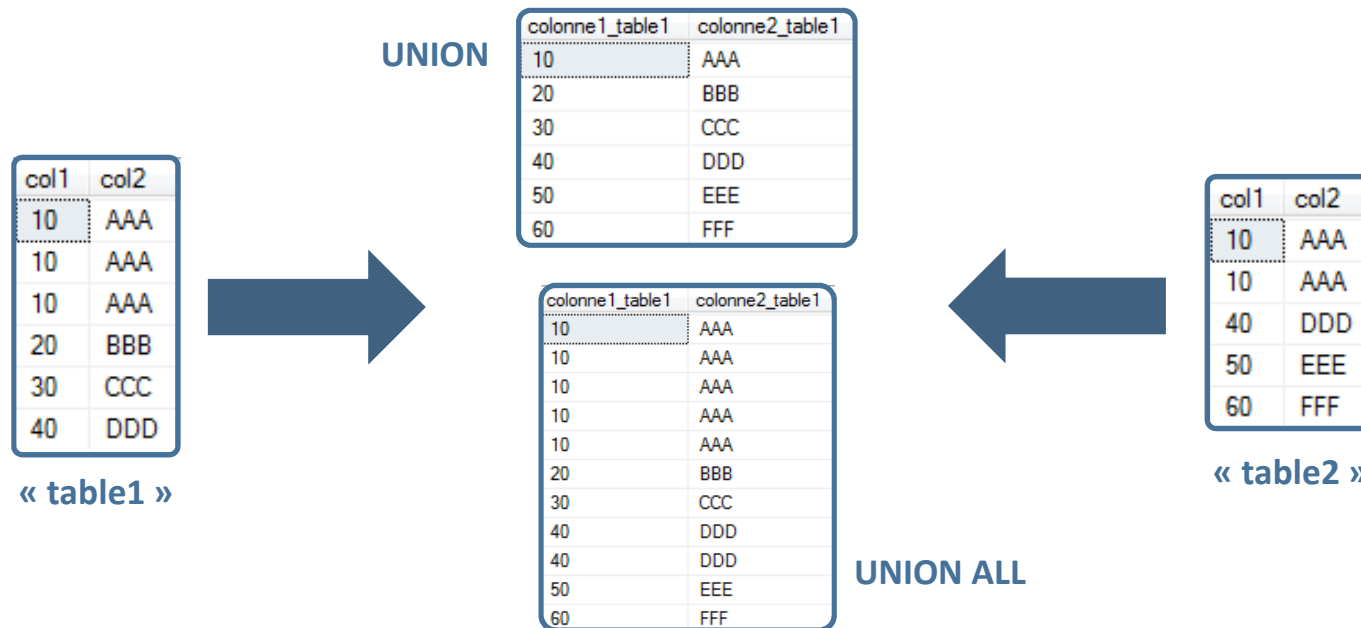
Jointures : **UNION [ALL]**

- 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



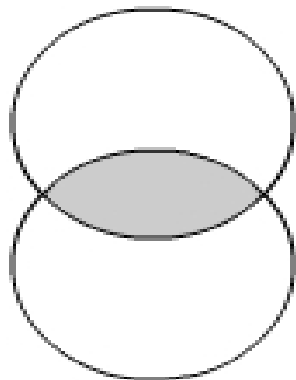
Jointures : UNION [ALL]

```
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]
```

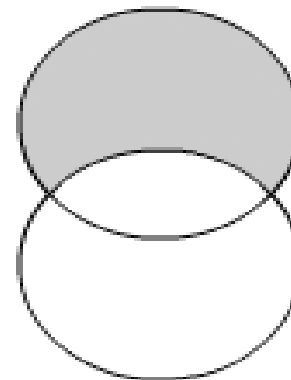


Jointures : **INTERSECT** et **EXCEPT**

- 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

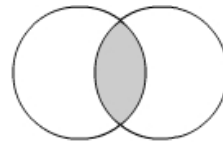


col1	col2
10	AAA
10	AAA
40	DDD
50	EEE
60	FFF

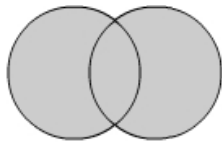
« table2 »

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

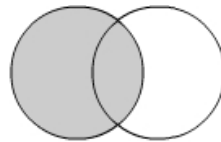
Jointures : Résumé



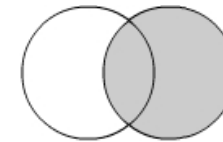
INNER JOIN



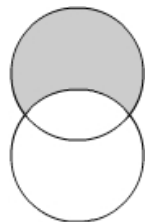
FULL OUTER JOIN



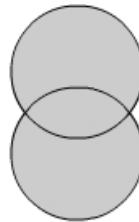
LEFT OUTER JOIN



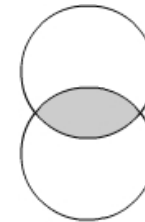
RIGHT OUTER JOIN



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 », « INTERSECT », « EXCEPT »				