

LE SQL de A à Z : 3e partie - les jointures

Table des matières

- I. Préambule
- II. Les jointures ou comment faire des requêtes sur plusieurs tables
 - II-A. Premiers essais de jointure
 - II-B. Différents type de jointures (naturelles, équi, non équi, auto, externes, hétérogènes, croisée et union)
- III. Syntaxe normalisée des jointures
 - III-A. Opérateur de jointure naturelle
 - III-B. Les jointures internes
 - III-C. Les jointures externes
 - III-D. Différence entre jointure externe et jointure interne
 - III-D-1. L'hypothèse du monde clos
 - III-D-2. Mécanisme en jeu
 - III-D-3. Discussion sur la jointure externe
 - III-E. La jointure croisée
 - III-F. La jointure d'union
- IV. Nature des conditions de jointures
 - IV-A. Équi-jointure
 - IV-B. Non équi-jointure
 - IV-C. Auto-jointure
 - IV-D. La jointure hétérogène
- V. Récapitulatif des jointures normalisées
 - V-A. Terminologie et syntaxe des jointures
 - V-B. Arbre de jointure
- VI. Note importante
- VII. Résumé

Dans le précédent article nous avons commencer à décortiquer le simple **SELECT**. Dans le présent, nous allons nous consacrer aux jointures entre tables. Toutes les jointures sous toutes les coutûres ! Autrement dit comment faire des requêtes portant sur plusieurs tables.

Article lu 815610 fois.

L'auteur

SQLPro 

L'article

Publié le 22 avril 2004 - Mis à jour le 12 juillet 2008

ePub, Azw et Mobi

Liens sociaux

 Partager

I. Préambule▲

La structure de la base de données exemple, ainsi qu'une version des principales bases utilisées sont disponibles dans la page "La base de données exemple"

II. Les jointures ou comment faire des requêtes sur plusieurs tables▲

Vous trouverez des compléments d'information sur le sujet aux pages 129 à 152 de l'ouvrage "SQL", collection "La Référence", Campus Press éditeur.

Les jointures permettent d'exploiter pleinement le modèle relationnel des tables d'une base de données. Elle sont faites pour mettre en relation deux (ou plus) tables concourant à rechercher la réponse à des interrogations. Une jointure permet donc de combiner les colonnes de plusieurs tables.

Il existe en fait différentes natures de jointures que nous expliciterons plus en détail. Retenez cependant que la plupart des jointures entre tables s'effectuent en imposant l'égalité des valeurs d'une colonne d'une table à une colonne d'une autre table. On parle alors de jointure naturelle ou équi-jointure. Mais on trouve aussi des jointures d'une table sur elle-même. On parle alors d'auto-jointure. De même, il arrive que l'on doive procéder à des jointures externe, c'est-à-dire joindre une table à une autre, même si la valeur de liaison est absente dans une table ou l'autre. Enfin, dans quelques cas, on peut procéder à des jointures hétérogènes, c'est-à-dire que l'on remplace le critère d'égalité par un critère d'inégalité ou de différence. Nous verrons au moins un cas de cette espèce.

Une jointure entre tables peut être mise en oeuvre, soit à l'aide des éléments de syntaxe SQL que nous avons déjà vu, soit à l'aide d'une clause spécifique du SQL, la clause **JOIN**. Nous allons commencer par voir comment à l'aide du SQL de base nous pouvons exprimer une jointure.

II-A. Premiers essais de jointure▲

Rappel de la syntaxe du **SELECT** :

Sélectionnez

SELECT [DISTINCT ou ALL] * ou liste_de_colonnes FROM nom_des_tables_ou_des_vues

C'est ici le pluriel de la partie **FROM** qui change tout!

Tâchons donc de récupérer les n° des téléphones associés aux clients.

Exemple 1 :

Sélectionnez	Séle
SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT, T_TELEPHONE	CLI_ ---- DUPC DUPC DUPC DUPC DUPC DUPC DUPC ...

Cette requête ne possède pas de critère de jointure entre une table et l'autre. Dans ce cas, le compilateur SQL calcule le produit cartésien des deux ensembles, c'est-à-dire qu'à chaque ligne de la première table, il accole l'ensemble des lignes de la seconde à la manière d'une "multiplication des petits pains" !
Nous verrons qu'il existe une autre manière, normalisée cette fois, de générer ce produit cartésien. Mais cette requête est à proscrire.
Dans notre exemple elle génère 17 400 lignes!

Il faut donc définir absolument un critère de jointure.

Dans le cas présent, ce critère est la correspondance entre les colonnes contenant la référence de l'identifiant du client (CLI_ID).

Exemple 2 :

Sélectionnez	Séle
SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT, T_TELEPHONE WHERE CLI_ID = CLI_ID	CLI_ ---- DUPC DUPC DUPC DUPC DUPC DUPC DUPC DUPC ...

Nous n'avons pas fait mieux, car nous avons créé une clause toujours vraie, un peu à la manière de 1 = 1 !
En fait il nous manque une précision : il s'agit de déterminer de quelles tables proviennent les colonnes CLI_ID de droite et de gauche. Cela se précise à l'aide d'une notation pointée en donnant le nom de la table.

Il est donc nécessaire d'indiquer au compilateur la provenance de chacune des colonnes CLI_ID et donc d'opérer une distinction entre l'une et l'autre colonne.
Ainsi, chaque colonne devra être précédée du nom de la table, suivi d'un point.

Exemple 3 :

Sélectionnez	Séle
SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT, T_TELEPHONE WHERE T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID	CLI_ ---- DUPC DUPC BOUV DUBC DREY DUHA BOUV MART MART ...

On tombe ici à 174 enregistrements dans la table !

Mais il existe une autre façon de faire, plus simple encore. On utilise la technique du "**surnommage**", c'est-à-dire que l'on attribue un surnom à chacune des tables présente dans la partie **FROM** du **SELECT** :

Exemple 4 :

	Séle
Sélectionnez	CLI_

	DUPC
	DUPC
	BOUV
	DUBC
	DREY
	DUHA
	BOUV
	MART
	...

Ici, la table T_CLIENT a été surnommée "C" et la table T_TELEPHONE "T".

Bien entendu, et comme dans les requêtes monotabulaires on peut poser des conditions supplémentaires de filtrage dans la clause **WHERE**. Cherchons par exemple les clients dont les numéros de téléphone correspondent à un fax :

Exemple 5 :

	Séle
Sélectionnez	CLI_

	DUPC
	MART
	DUHA
	DUPC
	MART
	DUBC
	DREY
	DUHA
	PHIL
	DAUM
	...

Le fait de placer comme critère de jointure entre les tables, l'opérateur logique "égal" donne ce que l'on appelle une **"équi-jointure"**.

Comme vous pouvez le constater, le nom du client "BOUVIER" n'apparaît pas. Il n'a pas été "oublié" par le traitement de la requête, mais le numéro de fax de ce client n'est pas présent dans la table T_TELEPHONE. Or le moteur SQL recherche les valeurs de la jointure par égalité. Comme l'ID_CLI de "BOUVIER" n'est pas présent dans la table T_TELEPHONE, il ne peut effectuer la jointure et ignore donc cette ligne. Nous verrons comment réparer cette lacune lorsque nous parlerons des jointures externes.

On peut aussi utiliser les surnoms dans la partie qui suit immédiatement le mot clef **SELECT**. Ainsi l'exemple 4, peut aussi s'écrire :

Exemple 6 :

	Séle
Sélectionnez	CLI_

	...

C'est particulièrement pratique lorsque l'on veut récupérer une colonne qui se retrouve dans les deux tables, ce qui est souvent le cas de la (ou les) colonne(s) de clef étrangère qui permet justement d'assurer la jointure.

Pour joindre plusieurs tables, on peut utiliser le même processus de manière répétitive...

Exemple 7 :

Sélectionnez
SELECT C.CLI_ID, C.CLI_NOM, T.TEL_NUMERO, E.EMPL_ADRESSE, A.ADR_VILLE FROM T_CLIENT C, T_TELEPHONE T, T_ADRESSE A, T_EMAIL E WHERE C.CLI_ID = T.CLI_ID AND C.CLI_ID = A.CLI_ID AND C.CLI_ID = E.CLI_ID
Sélectionnez

CLI_ID	CLI_NOM	TEL_NUMERO	EML_ADRESSE	ADR_VILLE
1	DUPONT	01-45-42-56-63	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	05-59-45-72-42	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	05-59-45-72-24	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	01-44-28-52-50	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	01-44-28-52-52	alain.dupont@wanadoo.fr	VERSAILLES
2	MARTIN	01-47-66-29-29	mmartin@transports_martin_fils.fr	VERGNOLLES CEDEX 452
2	MARTIN	01-47-66-29-55	mmartin@transports_martin_fils.fr	VERGNOLLES CEDEX 452
2	MARTIN	01-47-66-29-29	plongeur@aol.com	VERGNOLLES CEDEX 452
2	MARTIN	01-47-66-29-55	plongeur@aol.com	VERGNOLLES CEDEX 452
5	DREYFUS	04-92-19-18-58	pdreyfus@club-internet.fr	PARIS
...				

De même que nous l'avons vu dans l'exemple 2.4, ne sont visible ici que les lignes clients ayant **à la fois**, au moins une adresse, un e-mail et au moins un numéro de téléphone. Si nous avions voulu une liste complète des clients avec toutes les coordonnées disponibles, nous aurions du faire une requête externe sur les tables.

II-B. Différents type de jointures (naturelles, équi, non équi, auto, externes, hétérogènes, croisée et union)▲

Lorsque nous étudions le modèle relationnel de notre base de données exemple nous avons vu que le modèle physique des données, répercute les clefs des tables maîtres en tant que clefs étrangères des tables pour lesquelles une jointure est nécessaire. En utilisant la jointure entre clefs primaires et clefs secondaires basée sur l'égalité des valeurs des colonnes nous exécutons ce que les professionnels du SQL appelle une jointure naturelle.

Il est aussi possible de faire des équi-jointures qui ne sont pas naturelles, soit par accident (une erreur !), soit par nécessité.

Il est aussi possible de faire des non équi-jointures, c'est-à-dire des jointures basée sur un critère différent de l'égalité, mais aussi des auto-jointures, c'est-à-dire de joindre la table sur elle-même. Le cas le plus délicat à comprendre est celui des jointures externes, c'est-à-dire exiger que le résultat comprenne toutes les lignes des tables (ou d'au moins une des tables de la jointure), même s'il n'y a pas correspondance des lignes entre les différentes tables mise en oeuvre dans la jointure.

La jointure d'union consiste à ajouter toutes les données des deux tables à condition qu'elles soient compatibles dans leurs structures.

La jointure croisée permet de faire le produit cartésien des tables.

Enfin on peut effectuer des requêtes hétérogènes, c'est-à-dire de joindre une table d'une base de données, à une ou plusieurs autres base de données éventuellement même sur des serveurs différents, voire même sur des serveurs de différents types (par exemple joindre une table T_CLIENT de la base BD_COMMANDE d'un serveur Oracle à la table T_PROSPECT de la base BD_COMMERCIAL d'un serveur Sybase !).

Dans la mesure du possible, **utilisez toujours un opérateur de jointure normalisé Sql2 (mot clef JOIN)**.

En effet :

- Les jointures faites dans la clause **WHERE** (ancienne syntaxe de 1986 !) ne permettent pas de faire la distinction de prime abord entre ce qui relève du filtrage et ce qui relève de la jointure.
- Il est à priori absurde de vouloir filtrer dans le **WHERE** (ce qui restreint les données du résultat) et de vouloir "élargir" ce résultat par une jointure dans la même clause **WHERE** de filtrage.
- La lisibilité des requêtes est plus grande en utilisant la syntaxe à base de **JOIN**, en isolant ce qui est du filtrage et de la jointure, mais aussi en isolant avec clarté chaque condition de jointures entre chaque couples de table.
- L'optimisation d'exécution de la requête est souvent plus pointue du fait de l'utilisation du **JOIN**.
- Lorsque l'on utilise l'ancienne syntaxe et que l'on supprime la clause **WHERE** à des fins de tests, le moteur SQL réalise le produit cartésien des tables ce qui revient la plupart du temps à mettre à genoux le serveur !

III. Syntaxe normalisée des jointures▲



Vous trouverez des compléments d'information sur le sujet aux pages 136 à 152 de l'ouvrage "SQL", collection "La Référence", Campus Press éditeur.

Les jointures normalisées s'expriment à l'aide du mot clef **JOIN** dans la clause **FROM**. Suivant la nature de la jointure, on devra préciser sur quels critères se base la jointure.

Voici un tableau résumant les différents types de jointures normalisées :

Jointure interne	<p>Sélectionnez</p> <pre>SELECT ... FROM <table gauche> [INNER]JOIN <table droite> ON <condition de jointure></pre>
Jointure externe	<p>Sélectionnez</p> <pre>SELECT ... FROM <table gauche> LEFT RIGHT FULL OUTER JOIN <table droite> ON condition de jointure</pre>
Jointure	

naturelle	Sélectionnez SELECT ... FROM <table gauche> NATURAL JOIN <table droite> [USING <noms de colonnes>]
Jointure croisée	Sélectionnez SELECT ... FROM <table gauche> CROSS JOIN <table droite>
Jointure d'union	Sélectionnez SELECT ... FROM <table gauche> UNION JOIN <table droite>

Nous allons décrire en détail toutes ces jointures.

III-A. Opérateur de jointure naturelle▲



Il existe un opérateur normalisé pour effectué en SQL la jointure naturelle des tables :

Sélectionnez

```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM table1 NATURAL JOIN table2 [USING (colonne1 [, colonne2 ...])]
```

L'opérateur **NATURAL JOIN** permet d'éviter de préciser les colonnes concernées par la jointure.
Dans ce cas, le compilateur SQL va rechercher dans les 2 tables, les colonnes dont le nom est identique. Bien entendu, le type de données doit être le même !

NOTA : on veillera au niveau de la modélisation et notamment au niveau du MPD (Modèle Physique de Données) que les noms des colonnes de clefs en relation avec d'autres tables par l'intermédiaires des clefs étrangères soient strictement identiques.

Exemple 8 :

Sélectionnez SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT NATURAL JOIN T_TELEPHONE	Séle CLI_ --- DUPC DUPC DUPC BOUV DUBC DREY DUHA BOUV MART ...
--	--

Mais cette syntaxe est rarement acceptée par les moteurs SQL actuels !

La partie optionnelle **USING** permet de restreindre les colonnes concernées, lorsque plusieurs colonnes servent à définir la jointure naturelle. Ainsi la commande SQL :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT
      NATURAL JOIN T_TELEPHONE
      USING (CLI_ID)
```

Revient au même que la commande SQL de l'exemple 8.

III-B. Les jointures internes▲



Comme il s'agit de la plus commune des jointures c'est celle qui s'exerce par défaut si on ne précise pas le type de jointure. Après le mot clef **ON** on doit préciser le critère de jointure.

Reprenons notre exemple de départ :

Exemple 9 :

--	--

Sélectionnez	Séle
<pre>SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT INNER JOIN T_TELEPHONE ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID</pre>	CLI_ ----- DUPC DUPC DUPC BOUV DUBC DREY DUHA BOUV MART MART ...

Ou en utilisant le surnommage :

Exemple 10 :

Sélectionnez	Séle
<pre>SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT C INNER JOIN T_TELEPHONE T ON C.CLI_ID = T.CLI_ID</pre>	CLI_ ----- DUPC DUPC DUPC BOUV DUBC DREY DUHA BOUV MART MART ...

Plus pratique à écrire et aussi lisible sinon plus !

NOTA : le mot clef **INNER** est facultatif. Par défaut l'absence de précision de la nature de la jointure la fait s'exécuter en jointure interne. Ainsi on peut reformuler le requête ci-dessus en :

Exemple 11 :

Sélectionnez	Séle
<pre>SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT C JOIN T_TELEPHONE T ON C.CLI_ID = T.CLI_ID</pre>	CLI_ ----- DUPC DUPC DUPC BOUV DUBC DREY DUHA BOYE MART MART ...

III-C. Les jointures externes ▲



Les jointures externes sont extrêmement pratiques pour rapatrier le maximum d'informations disponible, même si des lignes de table ne sont pas renseignées entre les différentes tables jointes.

Procédons à l'aide d'un exemple pour mieux comprendre la différence entre une jointure interne et une jointure externe. Nous avons vu à l'exemple 9 que seul les clients dotés d'un numéro de téléphone étaient répertoriés dans la réponse. Ainsi, le client "BOUVIER" était absent.

Exemple 12 :

Sélectionnez	Séle
<pre>SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT C INNER JOIN T_TELEPHONE T ON C.CLI_ID = T.CLI_ID WHERE TYP_CODE = 'FAX'</pre>	CLI_ ----- DUPC MART DUHA DUPC MART DUBC DREY DUHA PHIL DAUM ...

Que faut-il modifier dans la requête pour obtenir une ligne "BOUVIER" avec aucune référence de téléphone associée dans la réponse ?
Il suffit en fait d'opérer à l'aide d'une jointure externe :

Exemple 13 :

Sélectionnez	Séle
<pre>SELECT CLI_NOM, TEL_NUMERO FROM T_CLIENT C LEFT OUTER JOIN T_TELEPHONE T ON C.CLI_ID = T.CLI_ID WHERE TYP_CODE = 'FAX' OR TYP_CODE IS NULL</pre>	CLI_ ---- DUPC DUPC MART BOUV DUBC DREY FAUR LACC DUHA DUHA ...

ou encore :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
      LEFT OUTER JOIN T_TELEPHONE T
        ON C.CLI_ID = T.CLI_ID AND TYP_CODE IS NULL
```

La syntaxe de la jointure externe est la suivante :

Sélectionnez

```
SELECT ...
FROM   <table gauche>
      LEFT | RIGHT | FULL OUTER JOIN <table droite 1>
        ON <condition de jointure>
      [LEFT | RIGHT | FULL OUTER JOIN <table droite 2>
        ON <condition de jointure 2>]
...
```

Les mots clefs **LEFT**, **RIGHT** et **FULL** indiquent la manière dont le moteur de requête doit effectuer la jointure externe. Il font référence à la table située à gauche (**LEFT**) du mot clef **JOIN** ou à la table située à droite (**RIGHT**) de ce même mot clef. Le mot **FULL** indique que la jointure externe est bilatérale.

Sélectionnez
<pre>SELECT colonnes FROM TGauche LEFT OUTER JOIN TDroite ON condition de jointure</pre>
On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche qui n'ont pas été prises en compte au titre de la satisfaction du critère.
Sélectionnez
<pre>SELECT colonnes FROM TGauche RIGHT OUTER JOIN TDroite ON condition de jointure</pre>
On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.
Sélectionnez
<pre>SELECT colonnes FROM TGauche FULL OUTER JOIN TDroite ON condition de jointure</pre>
On recherche toutes les valeurs satisfaisant la condition de jointure précisée dans prédicat, puis on rajoute toutes les lignes de la table TGauche et TDroite qui n'ont pas été prises en compte au titre de la satisfaction du critère.

- NOTA** : il existe des équivalences entre différentes expressions logiques à base de jointures externes. Les principales sont :
- la jointure externe droite peut être obtenue par une jointure externe gauche dans laquelle on inverse l'ordre des tables.
 - la jointure externe bilatérale peut être obtenue par la combinaison de deux jointures externes unilatérales avec l'opérateur ensemblistes **UNION**.

Remplacement d'une jointure externe droite par une jointure externe gauche.
L'équivalent logique de :

Exemple 14 :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       RIGHT OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

est :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_TELEPHONE T
       LEFT OUTER JOIN T_CLIENT C
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

Remplacement d'un **FULL OUTER JOIN** avec jointures externes gauche et droite :
L'équivalent logique de ...

Exemple 15 :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       FULL OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

est :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       LEFT OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
UNION
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       RIGHT OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

Remplacement d'un **FULL OUTER JOIN** avec jointures externes gauche uniquement :
L'équivalent logique de ...

Exemple 16 :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       FULL OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

est :

Sélectionnez

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       LEFT OUTER JOIN T_TELEPHONE T
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
UNION
SELECT CLI_NOM, TEL_NUMERO
FROM   T_TELEPHONE T
       LEFT OUTER JOIN T_CLIENT C
         ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'
```

III-D. Différence entre jointure externe et jointure interne▲

Pour bien comprendre la distinction entre les jointures internes et externes, nous devons consacrer quelques instants à aborder des problèmes de logique ensembliste sous un oeil pragmatique.

III-D-1. L'hypothèse du monde clos▲

Les jointures externes sont extrêmement pratiques pour rapatrier le maximum d'informations disponible, même si des lignes de table ne sont pas renseignées entre les différentes tables jointes.

Sans le savoir, nous faisons assez systématiquement l'hypothèse du monde clos. c'est-à-dire que nous considérons que l'absence d'information, n'est pas une information. Si vous

demandez à une secrétaire de vous communiquer les coordonnées des clients qui sont domiciliés à Paris, elle vous donnera une liste où figurera autant de fois le nom "Paris" qu'il y a de clients dans la liste, et ceci paraît bien normal ! Sauf que, comme l'aurait fait tout un chacun, votre secrétaire a fait l'hypothèse du monde clos sans le savoir en présumant que les clients pour lesquels l'adresse n'est pas renseignée ne sont pas domiciliés à PARIS !

C'est cela l'hypothèse du monde clos : considérer que l'absence d'information doit être synonyme de critère de discrimination... **La jointure externe permet de contrer l'hypothèse du monde clos** en considérant qu'en cas d'absence de jointure entre une table et l'autre, on ne supprime par pour autant l'information.

III-D-2. Mécanisme en jeu ▲

Lorsqu'une ligne d'une table figurant dans une jointure n'a pas de correspondance dans les autres tables, le critère d'équi-jointure n'est pas satisfait et la ligne est rejetée. C'est la jointure interne. Au contraire, la jointure externe permet de faire figurer dans le résultat les lignes satisfaisant la condition d'équi-jointure ainsi que celles n'ayant pas de correspondances.

Ainsi, si je veux contacter tous mes clients, quelque soit le mode de contact que je veux utiliser dans le cadre d'une campagne publicitaire, j'ai intérêt à obtenir une réponse contenant **tous** les clients, même ceux qui n'ont pas de téléphone, d'e-mail ou d'adresse.

Exemple 17 :

Sélectionnez				
<pre>SELECT CLI_ID, CLI_NOM, TYP_CODE ' : ' TEL_NUMERO AS TEL_CONTACT, EML_ADRESSE, ADR_VIL FROM T_CLIENT C LEFT OUTER JOIN T_TELEPHONE T ON C.CLI_ID = T.CLI_ID LEFT OUTER JOIN T_EMAIL E ON C.CLI_ID = E.CLI_ID LEFT OUTER JOIN T_ADRESSE A ON C.CLI_ID = A.CLI_ID</pre>				
Sélectionnez				
CLI_ID	CLI_NOM	TEL_CONTACT	EML_ADRESSE	ADR_VILLE
1	DUPONT	01-45-42-56-63	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	05-59-45-72-42	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	05-59-45-72-24	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	01-44-28-52-50	alain.dupont@wanadoo.fr	VERSAILLES
1	DUPONT	01-44-28-52-52	alain.dupont@wanadoo.fr	VERSAILLES
2	MARTIN	01-47-66-29-29	mmartin@transports_martin_fils.fr	VERGNOLLES CEDEX
2	MARTIN	01-47-66-29-55	mmartin@transports_martin_fils.fr	VERGNOLLES CEDEX
2	MARTIN	01-47-66-29-29	plongeur@aol.com	VERGNOLLES CEDEX
2	MARTIN	01-47-66-29-55	plongeur@aol.com	VERGNOLLES CEDEX
3	BOUVIER	GSM : 06-11-86-78-89	NULL	MONTMAIZIN
...				

NOTA : Sur certains moteurs SQL la jointure bilatérale externe (**FULL OUTER JOIN**) s'exprime :

Sélectionnez

```
SELECT colonnes
FROM TGauche FULL JOIN TDroite ON condition de jointure
```

D'anciennes syntaxes permettent de faire des jointures externes unilatérale. Par exemple, il n'est pas rare de rencontrer les syntaxes suivantes :

Sélectionnez

```
SELECT colonnes
FROM table_1 t1, table_2 t2
WHERE t1.id1 *= t2.id2
```

ou encore :

Sélectionnez

```
SELECT colonnes
FROM table_1 t1, table_2 t2
WHERE t1.id1 (+)= t2.id2
```

Elles sont bien évidemment à proscrire si la syntaxe SQL 2 est disponible !

III-D-3. Discussion sur la jointure externe ▲

La jointure externe est rarement bien comprise du premier coup. Si je vous propose de lire cette discussion qui a eût lieu sur un forum Internet, c'est parce quelle permet de mieux la comprendre.

Sélectionnez

Objet : Optimisation Jointure
 Date : 26 Dec 2002 11:54:07 +0100
 De : Laurent Moreau

Bonjour a tous,
 Je me pose une petite question de syntaxe SQL sur les jointures.
 Et comme Frédéric va surement me répondre, je prend un exemple de son site.
 Y-a t-il une syntaxe meilleure que l'autre (si oui pourquoi) ?

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       INNER JOIN T_TELEPHONE T
           ON C.CLI_ID = T.CLI_ID
WHERE  TYP_CODE = 'FAX'

SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
       INNER JOIN T_TELEPHONE T
           ON (C.CLI_ID = T.CLI_ID AND T.TYP_CODE = 'FAX' )
```

Merci,

Sélectionnez

Date : 26 Dec 2002 14:20:39 -0800
 De : Med Bouchenafa

Au niveau performance, il n'y a aucune différence, l'optimiseur de SQL/Server construit le même plan
 Par contre, au niveau conceptuel il est interessant de différencier ce qui appartient à la jointure et ce qui appartient à la limitation du jeu de résultat
 Cela facilite la maintenance et la lecture du source
 [...]

Sélectionnez

Date : 26 Dec 2002 16:31:14 +0100
 De : Frédéric BROUARD

Pour comprendre la différence entre le prédicat de la clause de jointure et le prédicat du filtre WHERE, je vous propose le petit test suivant :

```
-- test de jointure

CREATE TABLE TEST_JOIN1
(COL1 INT,
 COL2 CHAR(2))

CREATE TABLE TEST_JOIN2
(COL1 INT,
 COL2 CHAR(2))

INSERT INTO TEST_JOIN1 VALUES (101, 'AA')
INSERT INTO TEST_JOIN1 VALUES (102, 'AA')
INSERT INTO TEST_JOIN1 VALUES (103, 'BB')

INSERT INTO TEST_JOIN2 VALUES (101, 'AA')
INSERT INTO TEST_JOIN2 VALUES (102, 'AA')
INSERT INTO TEST_JOIN2 VALUES (201, 'BB')

-- équivalente ??? En apparence seulement !

SELECT TJ1.COL1, TJ1.COL2
FROM   TEST_JOIN1 TJ1
       JOIN TEST_JOIN2 TJ2
           ON TJ1.COL1 = TJ2.COL1
WHERE  TJ1.COL2 = 'AA'

SELECT TJ1.COL1, TJ1.COL2
FROM   TEST_JOIN1 TJ1
       JOIN TEST_JOIN2 TJ2
           ON TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'AA'

-- d'accord... même résultat !!!

--changeons le lien interne en lien externe...

SELECT TJ1.COL1, TJ1.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
           ON TJ1.COL1 = TJ2.COL1
WHERE  TJ1.COL2 = 'AA'

SELECT TJ1.COL1, TJ1.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
           ON TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'AA'

-- pas d'accord !!! résultat différent...

-- POURQUOI ???
-- je vous laisse méditer la chose !
```

Sélectionnez

Date : 26 Dec 2002 16:55:13 +0100
 De : Lionel Pénuchot

Salut Fred,

Tu peux alors m'expliquer pourquoi cette requête renvoie le même résultat que ta de

```
SELECT TJ1.COL1, TJ1.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
         ON TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'taratata' and 1=0
```

Cordialement

Sélectionnez

Date : 27 Dec 2002 10:17:10 +0100
De : Frédéric BROUARD

La clause WHERE est un filtre d'élimination. Il suppose que les différentes tables sont déjà jointes en une seule et parcourt l'ensemble des résultats à la recherche des lignes correspondant aux critères donnés.

En revanche la clause JOIN se comporte différemment. Elle agit AVANT que la jointure soit effective.

Dans le cadre d'une jointure INTERNE le comportement est similaire à celui d'une clause WHERE.

Dans le cas d'une jointure externe il faut décomposer la logique en plusieurs étapes :

- 1) évaluation des éléments du prédicat
- 2) application de la jointure : lignes jointe en 1)
+ lignes sans correspondance

Reprennons la clause de jointure de la requête donnée par Laurent :

```
TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'taratata' and 1=0
```

prédicat de jointure	prédicats hors jointure
----------------------	-------------------------

1° 1=0 => aucune ligne n'est récupérée
donc aucune ligne de TEST_JOIN1 n'a de correspondance avec TEST_JOIN2
conclusion : toutes les lignes de TEST_JOIN1 seront reprises du fait
de la jointure externe

2° T1.COL2 = 'taratata' => encore une fois aucune ligne n'est récupérée donc aucune ligne de TEST_JOIN1 n'a de correspondance avec TEST_JOIN2 conclusion : toutes les lignes de TEST_JOIN1 seront reprises du fait de la jointure externe

3° TJ1.COL1 = TJ2.COL1 => 2 lignes sont en correspondance avec TEST_JOIN2
conclusion, ces deux lignes seront récupérées du fait de la jointure
externe et la 3eme ligne aussi puisqu'elle n'a aucune correspondance

???

Pour comprendre la différence, il suffit de demander à voir les colonnes de la table TEST_JOIN2 dans les différents cas :

```
SELECT TJ1.COL1, TJ1.COL2, TJ2.COL1, TJ2.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
         ON TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'AA'
```

COL1	COL2	COL1	COL2
101	AA	101	AA
102	AA	102	AA
103	BB	NULL	NULL

```
SELECT TJ1.COL1, TJ1.COL2, TJ2.COL1, TJ2.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
         ON TJ1.COL1 = TJ2.COL1
WHERE  TJ1.COL2 = 'AA'
```

COL1	COL2	COL1	COL2
101	AA	101	AA
102	AA	102	AA

```
SELECT TJ1.COL1, TJ1.COL2, TJ2.COL1, TJ2.COL2
FROM   TEST_JOIN1 TJ1
       LEFT OUTER JOIN TEST_JOIN2 TJ2
         ON TJ1.COL1 = TJ2.COL1 AND TJ1.COL2 = 'taratata' and 1=0
```

COL1	COL2	COL1	COL2
101	AA	NULL	NULL
102	AA	NULL	NULL
103	BB	NULL	NULL

et voilà !

III-E. La jointure croisée▲



La jointure croisée n'est autre que le produit cartésien de deux tables. Rappelons que le produit cartésien de deux ensembles n'est autre que la multiplication généralisée. Dans le cas des tables, c'est le fait d'associer à chacune des lignes de la première table, toutes les lignes de la seconde. Ainsi, si la première table compte 267 lignes et la seconde 1214, on se retrouve avec un résultat contenant 324 138 lignes. Bien entendu, s'il n'y a aucun doublon dans les tables, toutes les lignes du résultat seront aussi uniques.

La jointure croisée peut s'écrire de deux manières différentes :

- à l'aide de l'opérateur normalisé :

Sélectionnez

```
SELECT colonnes
FROM table_1 CROSS JOIN table_2
```

- ou à l'aide d'une clause **FROM** simplifiée :

Sélectionnez

```
SELECT colonnes
FROM table_1, table_2
```

Ce qui est certainement l'expression la plus minimaliste de tous les ordres **SELECT** du SQL !

Nous voudrions savoir si notre table des tarifs de chambre (TJ_TRF_CHB) est complète, c'est-à-dire si l'on a bien toutes les chambres (T_CHAMBRE) pour toutes les dates de début de tarif (T_TARIF) :

Exemple 18 :

Sélectionnez

```
SELECT CHB_ID, TRF_DATE_DEBUT, 0 AS TRF_CHB_PRIX
FROM   T_TARIF, T_CHAMBRE
ORDER BY CHB_ID, TRF_DATE_DEBUT
```

Séle
CHB_

...

En comparant rapidement le contenu des deux tables on peut s'assurer que tous les tarifs ont bien été renseignés.

Avec la syntaxe normalisée, cette requête s'écrit :

Exemple 19 :

Sélectionnez

```
SELECT CHB_ID, TRF_DATE_DEBUT, 0 AS TRF_CHB_PRIX
FROM   T_TARIF CROSS JOIN T_CHAMBRE
ORDER BY CHB_ID, TRF_DATE_DEBUT
```

Séle
CHB_

...

et donne le même résultat !

III-F. La jointure d'union▲



La jointure d'union, permet de faire l'union de deux tables de structures quelconque. Elle s'exprime qu'à l'aide de l'opérateur normalisé SQL2 suivant :

Sélectionnez

```
SELECT colonnes
FROM table_1 UNION JOIN table_2
```

Effectuons par exemple la jointure d'union entre les tables T_TITRE ("M.", "Mme.", "Melle.") et T_TYPE ("FAX", "GSM", "TEL") :

Exemple 20 :

Sélectionnez

```
SELECT *
FROM T_TITRE UNION JOIN T_TYPE
```

Séle
TIT_

M.
Melle
Mme.
NULL
NULL
NULL

--	--

En fait c'est comme si l'on avait listé la première table, puis la seconde en évitant toute colonne commune et compléter les espaces vides des valeurs **NULL**.

Mais cette jointure est très rarement implantée dans les moteurs SQL.

NOTA : si l'opérateur **UNION JOIN** n'est pas présent dans votre moteur, vous pouvez le fabriquer comme suit, car elle (l'union) peut être facilement remplacée par un autre ordre SQL presque aussi simple :

Sélectionnez

```
SELECT *
FROM   <table gauche>
      FULL OUTER JOIN <table droite>
      ON <critère>
```

Où la condition <critère> est n'importe quel prédicat valant toujours faux comme "1=2".

En dehors du fait de linéariser des tables hétérogènes, il est franchement permis de douter de l'utilité d'un tel opérateur.

IV. Nature des conditions de jointures ▲

Nous allons maintenant analyser les différentes jointures basées sur la nature des conditions pour bien les distinguer.

IV-A. Équi-jointure ▲

L'équi-jointure consiste à opérer une jointure avec une condition d'égalité. Cette condition d'égalité dans la jointure peut ne pas porter nécessairement sur les clefs (primaires et étrangères).
Recherchons par exemple les clients dont le nom est celui d'une ville contenu dans la table des adresses :

Exemple 21 :

Sélectionnez	Séle
SELECT DISTINCT C.CLI_ID, C.CLI_NOM, A.ADR_VILLE	CLI_
FROM T_CLIENT C	----
JOIN T_ADRESSE A	...
ON C.CLI_NOM = A.ADR_VILLE	

Nous avons donc bien réalisé une équi-jointure, mais elle n'est pas naturelle parce qu'elle ne repose pas sur les clefs des tables.

Bien entendu, il existe un opérateur normalisé SQL 2 permettant de traiter le cas de l'équi-jointure :

Sélectionnez

```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM   table1 [INNER] JOIN table2 ON condition de jointure
```

Le mot clef **INNER** n'étant pas obligatoire, mais voulant s'opposer aux mot clefs **OUTER**, **UNION** et **CROSS**.
Ainsi, la requête précédente, s'écrit à l'aide de cette syntaxe :

Exemple 22 :

Sélectionnez	Séle
SELECT DISTINCT C.CLI_ID, C.CLI_NOM, A.ADR_VILLE	CLI_
FROM T_CLIENT C	----
INNER JOIN T_ADRESSE A	...
ON C.CLI_NOM = A.ADR_VILLE	

IV-B. Non équi-jointure ▲

Il s'agit là d'utiliser n'importe quelle condition de jointure entre deux tables, exceptée la stricte égalité. Ce peuvent être les conditions suivantes :

>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
<>	différent de
IN	dans un ensemble
LIKE	correspondance partielle

BETWEEN	entre deux
... AND	valeurs
...	
EXISTS	dans une table

En règle générale on trouve des non équi-jointures dans le cadre de comparaisons temporelles ou de mesures physiques. Par exemple on pourrait rechercher une pièce mécanique dans un stock qui soit de même nature ou de même fonction qu'une pièce donnée, mais plus légère.

Nous voulons obtenir les factures qui ont été émises avant que le prix des petits déjeuners n'atteigne 6 €.

Exemple 23 :

Sélectionnez	Séle
SELECT F.* FROM T_FACTURE F INNER JOIN T_TARIF T ON F.FAC_DATE < T.TRF_DATE_DEBUT WHERE TRF_PETIT_DEJEUNE >= 6	FAC_ ----- ...

NOTA : pour récupérer toutes les colonnes d'une table, on peut utiliser l'opérateur * suffixé par le nom de table, comme nous l'avons fait ici pour la table des factures.

Si notre table des tarifs avait été organisée par tranches, comme ceci :

Sélectionnez

TRF_DATE_DEBUT	TRF_DATE_FIN	TRF_TAUX_TAXES	TRF_PETIT_DEJEUNE
1999-01-01	1999-08-31	18,60	6,00 E
1999-09-01	1999-12-31	20,60	7,00 E
2000-01-01	2000-08-31	20,60	8,00 E
2000-09-01	2000-12-31	20,60	9,00 E
2001-01-01	2001-12-31	20,60	10,00 E

alors, récupérer le tarif des chambres pour chacune des dates du planning devient un exercice très simple :

Exemple 24 :

Sélectionnez	Séle
SELECT CPC.CHB_ID, CPC.PLN_JOUR, TC.TRF_CHB_PRIX FROM TJ_CHB_PLN_CLI CPC INNER JOIN T_TARIF T ON CPC.PLN_JOUR BETWEEN T.TRF_DATE_DEBUT AND T.TRF_DATE_FIN INNER JOIN TJ_TRF_CHB TC ON T.TRF_DATE_DEBUT = TC.TRF_DATE_DEBUT AND CPC.CHB_ID = TC.CHB_ID	CHB_ ----- ...

Nous avons donc à nouveau un exemple remarquable de non équi-jointure.

Constatons que la colonne TRF_DATE_FIN de cette nouvelle version de la table des tarifs implique une redondance de l'information. En effet, cette date de fin est déductible de la date de début de la ligne qui contient la date immédiatement postérieure avec un jour de moins. De plus le problème induit par cette organisation des données fait qu'il faut obligatoirement définir une date de fin des tarifs, même dans le futur, sinon certaines tarifications ne pourront être établies par cette requête. Il ne s'agit donc pas d'une modélisation correcte !

IV-C. Auto-jointure ▲

Le problème consiste à joindre une table à elle-même. Il est assez fréquent que l'on ait besoin de telles auto-jointures car elle permettent notamment de modéliser des structures de données complexes comme des arbres. Voici quelques exemples de relation nécessitant une auto-jointure de tables :

- dans une table des employés, connaître le supérieur hiérarchique de tout employé
- dans une table de nomenclature savoir quels sont les composants nécessaires à la réalisation d'un module, ou les modules nécessaires à la réalisation d'un appareil
- dans une table de personnes, retrouver l'autre moitié d'un couple marié.

La représentation d'une telle jointure dans le modèle de données, se fait par le rajout d'une colonne contenant une pseudo clef étrangère basée sur la clef de la table.

Dans ce cas, une syntaxe possible pour l'auto-jointure est la suivante :

Sélectionnez

```
SELECT [DISTINCT ou ALL] * ou liste de colonnes
FROM   laTable t1
       INNER JOIN laTable t2
       ON t1.laClef = t2.laPseudoClefEtrangère
```

C'est l'exemple typique où l'utilisation de surnoms pour les tables est obligatoire, sinon il y a risque de confusion pour le moteur SQL.

Pour donner un exemple concret à nos propos nous allons modéliser le fait qu'une chambre puisse communiquer avec une autre (par une porte). Dès lors, le challenge est de trouver quelles sont les chambres qui communiquent entre elles par exemple pour réaliser une sorte de suite. Pour ce faire, nous allons ajouter à notre table des chambres une colonne de clef étrangère basée sur la clef de la table. Dans ce cas, cette colonne doit obligatoirement accepter des valeurs nulles !

Voici l'ordre SQL pour rajouter la colonne CHB_COMMUNIQUE dans la table T_CHAMBRE :

Sélectionnez

```
ALTER TABLE T_CHAMBRE ADD CHB_COMMUNIQUE INTEGER
```

Alimentons là de quelques valeurs exemples en considérant que la 7 communique avec la 9 et la 12 avec la 14 :

Sélectionnez

```
UPDATE T_CHAMBRE SET CHB_COMMUNIQUE = 9 WHERE CHB_ID = 7
UPDATE T_CHAMBRE SET CHB_COMMUNIQUE = 7 WHERE CHB_ID = 9
UPDATE T_CHAMBRE SET CHB_COMMUNIQUE = 12 WHERE CHB_ID = 14
UPDATE T_CHAMBRE SET CHB_COMMUNIQUE = 14 WHERE CHB_ID = 12
```

Pour formuler la recherche de chambres communicantes, il suffit de faire la requête suivante :

Exemple 25 :

Sélectionnez	Séle
<pre>SELECT DISTINCT c1.CHB_ID, c1.CHB_COMMUNIQUE FROM T_CHAMBRE c1 INNER JOIN T_CHAMBRE c2 ON c1.CHB_ID = c2.CHB_COMMUNIQUE</pre>	CHB_ ----

où la table T_CHAMBRE figure deux fois par l'entremise de deux surnommages différents c1 et c2.

Notons que cette présentation n'est pas pratique car elle dédouble le nombre de couples de chambres communicantes, donnant l'impression qu'il y a 4 couples de chambres communicantes ce qui est faux. Il manque juste un petit quelque chose pour que cette requête soit parfaite. Il suffit en effet de ne retenir les solutions que pour des identifiants inférieurs ou supérieur d'une table par rapport à l'autre :

Exemple 26 :

Sélectionnez	Séle
<pre>SELECT DISTINCT c1.CHB_ID, c1.CHB_COMMUNIQUE FROM T_CHAMBRE c1 INNER JOIN T_CHAMBRE c2 ON c1.CHB_ID = c2.CHB_COMMUNIQUE AND c1.CHB_ID <= c2.CHB_ID</pre>	CHB_ ----

IV-D. La jointure hétérogène ▲

Une jointure hétérogène consiste à joindre dans une même requête, des tables provenant de bases de données différentes, voire de serveurs de données différents. Un tel type de jointure n'est possible que :

- entre deux bases d'un même serveur : que si le SQL du serveur le permet (par exemple deux bases Oracle, deux bases SQL Server, etc.)
- entre deux bases de deux serveurs différents : qu'en passant par un outil tiers de type "middleware" (par exemple une entre une base Oracle et une base Sybase). L'un des rares middleware à faire ce type de jointure est le BDE (Borland Database Engine) d'Inprise.

Dans ces deux cas, il faut bien vérifier la compatibilité des types car il se peut que des résultats surprenants apparaissent notamment dans le traitement des nombres réels.

Pour effectuer de telles requêtes, la syntaxe est toujours spécifique au moteur employé. Avec le BDE le niveau de SQL est SQL 2, cantonné aux fonctions de base.

Voici un exemple de requête entre deux serveurs à l'aide du BDE :

Exemple 27 :

Sélectionnez	Séle
SELECT C.CLI_NOM, T.TEL_NUMERO	NOM_
FROM ":ORACLE_CLIBD:T_CLIENT" C, ":SYBASE_TELBD:T_TELEPHONE" T	----
WHERE C.CLI_ID = T.CLI_ID	DUPC
	DUPC
	DUPC
	BOUV
	DUBC
	DREY
	DUHA
	BOYE
	MART
	...

En fait le BDE se sert d'alias permettant de définir à la fois le serveur et la base concerné. Ici les alias sont : ":ORACLE_CLIBD:" et ":SYBASE_TELBD:".

V. Récapitulatif des jointures normalisées▲

V-A. Terminologie et syntaxe des jointures▲



- **Jointure naturelle** : la jointure s'effectue sur les colonnes communes, c'est-à-dire celles de même nom et type :

Sélectionnez

```
SELECT colonnes
FROM table1 NATURAL JOIN table2 [USING col1, col2 ... ]
[WHERE prédicat] ...
```

Le mot clef **USING** permet de restreindre les colonnes communes à prendre en considération.

- **Jointure interne** : la jointure s'effectue entre les tables sur les colonnes précisées dans la condition de jointure :

Sélectionnez

```
SELECT colonnes
FROM table1 t1 [INNER ] JOIN table2 t2 ON condition
[WHERE prédicat] ...
```

- **Jointure externe** : la jointure permet de récupérer les lignes des tables correspondant au critère de jointure, mais aussi celle pour lesquelles il n'existe pas de correspondances.

Sélectionnez

```
SELECT colonnes
FROM table1 t1 [RIGHT OUTER | LEFT OUTER | FULL OUTER ] JOIN table2 t2 ON condition
[WHERE prédicat] ...
```

- **RIGHT OUTER** : la table à droite de l'expression clef "RIGHT OUTER" renvoie des lignes sans correspondance avec la table à gauche.
- **LEFT OUTER** : la table à gauche de l'expression clef "LEFT OUTER" renvoie des lignes sans correspondance avec la table à droite.
- **FULL OUTER** : les deux tables renvoient des lignes sans correspondance entre elles.
- **Jointure croisée** : la jointure effectue le produit cartésien (la "multiplication") des deux tables.
Il n'y a pas de condition.

Sélectionnez

```
SELECT colonnes
FROM table1 t1 CROSS JOIN table2 t2
[WHERE prédicat] ...
```

- **Jointure d'union** : la jointure concatène les tables sans aucune correspondances de colonnes.

Sélectionnez

```
SELECT colonnes
FROM table1 UNION JOIN table2
```

Il n'y a pas de critère de jointure.

Si votre SGBDR n'implémente pas la jointure externe droite, inversez l'ordre des tables et faire une jointure externe gauche lorsque cela est possible.

V-B. Arbre de jointure▲

La jointure de multiple tables peut se représenter sous la forme d'un arbre. Cet arbre possède donc une racine, c'est la table principale, celle d'où l'on veut que l'information

partie. Elle possède aussi des feuilles, c'est-à-dire des tables d'entités. Les tables situées entre la racine et les feuilles, sont souvent des tables de jointure, possédant en principe deux clef étrangères. Dans le principe toute table de jointure devrait être un noeud de l'arbre.

La représentation arborescente d'une jointure est un excellent moyen pour visualiser si la clause de jointure de votre requête est à priori correcte. En effet, une référence circulaire dans la clause de jointure ne peut pas être représentée sous la forme d'un arbre et il y a fort à parier que la requête soit incorrecte.

Voici par exemple la requête qui "met à plat", la base hotel :

Sélectionnez

```
SELECT *
FROM   T_CLIENT CLI                -- le client (racine de l'arbre)
      JOIN T_ADRESSE ADR           -- adresse, table d'entité (feuille de l'arbre)
        ON CLI.CLI_ID = ADR.CLI_ID
      JOIN T_TITRE TIT             -- titre, table d'entité (feuille de l'arbre)
        ON CLI.TIT_CODE = TIT.TIT_CODE
      JOIN T_EMAIL EML             -- mail, table d'entité (feuille de l'arbre)
        ON CLI.CLI_ID = EML.CLI_ID
      JOIN T_TELEPHONE TEL         -- téléphone, table d'entité servant de jointure (n
        ON CLI.CLI_ID = TEL.CLI_ID
      JOIN T_TYPE TYP             -- type de téléphone, table d'entité (feuille de l
        ON TEL.TYP_CODE = TYP.TYP_CODE
      JOIN TJ_CHB_PLN_CLI CPC       -- table de jointure (noeud dans l'arbre)
        ON CLI.CLI_ID = CPC.CLI_ID
      JOIN T_PLANNING PLN          -- date du planning, table d'entité (feuille de l'a
        ON CPC.PLN_JOUR = PLN.PLN_JOUR
      JOIN T_CHAMBRE CHB          -- chambre, table d'entité servant de jointure (noe
        ON CPC.CHB_ID = CHB.CHB_ID
      JOIN TJ_TRF_CHB TC          -- table de jointure (noeud dans l'arbre)
        ON CHB.CHB_ID = TC.CHB_ID
      JOIN T_TARIF TRF            -- tarif, table d'entité (feuille de l'arbre)
        ON TC.TRF_DATE_DEBUT = TRF.TRF_DATE_DEBUT
      JOIN T_FACTURE FAC          -- facture, table d'entité servant de jointure
        ON CLI.CLI_ID = FAC.CLI_ID
      JOIN T_LIGNE_FACTURE LIF     -- ligne de facture, table d'entité (feuille de l'a
        ON FAC.FAC_ID = LIF.FAC_ID
      JOIN T_MODE_PAIEMENT PMT     -- mode de paiement, table d'entité (feuille de l'a
        ON FAC.PMT_CODE = PMT.PMT_CODE
```

Correctement indenté on distingue déjà la structure arborescente. On peut la mettre en évidence en supprimant tout ce qui n'est pas un nom de table :

Sélectionnez

```
T_CLIENT CLI                -- le client (racine de l'arbre)
T_ADRESSE ADR              -- adresse, table d'entité (feuille de l'arbre)
T_TITRE TIT                -- titre, table d'entité (feuille de l'arbre)
T_EMAIL EML                -- mail, table d'entité (feuille de l'arbre)
T_TELEPHONE TEL           -- téléphone, table d'entité servant de jointure (n
T_TYPE TYP                 -- type de téléphone, table d'entité (feuille de l
TJ_CHB_PLN_CLI CPC         -- table de jointure (noeud dans l'arbre)
T_PLANNING PLN             -- date du planning, table d'entité (feuille de l'a
T_CHAMBRE CHB             -- chambre, table d'entité servant de jointure (noe
TJ_TRF_CHB TC              -- table de jointure (noeud dans l'arbre)
T_TARIF TRF                -- tarif, table d'entité (feuille de l'arbre)
T_FACTURE FAC              -- facture, table d'entité servant de jointure
T_LIGNE_FACTURE LIF        -- ligne de facture, table d'entité (feuille de l'a
T_MODE_PAIEMENT PMT        -- mode de paiement, table d'entité (feuille de l'a
```

Par cette indentation, il est facile de repérer les jointures entre les tables.

VI. Note importante▲

Les jointures ne sont pas la seule manière de mettre en relation différentes tables au sein d'une même requête SQL. On peut aussi joindre plusieurs tables à l'aide des sous-requêtes ainsi qu'à l'aide des opérateurs ensemblistes que nous allons voir aux deux prochains chapitres.

VII. Résumé▲

Voici les différences entre les moteurs des bases de données :

	Paradox	Access	PostgreSQL	Sybase	SQL Server 7	Oracle 8	DB2 (400)
INNER JOIN	Oui	Oui	Oui	Oui	Oui	Non (1)	Oui
OUTER JOIN	LEFT, RIGHT, FULL	LEFT, RIGHT	LEFT, RIGHT, FULL	LEFT, RIGHT, FULL	LEFT, RIGHT, FULL	Non (1)	LEFT (4)
UNION JOIN	Non (2)	Non	Non	Non (2)	Non (2)	Non (1) (2)	Non
CROSS JOIN	Non (3)	Non (3)	Oui	Non (3)	Oui	Non (1) (3)	Oui

(1) Oracle ne connaît toujours pas le **JOIN** (ça fait quand même plus de dix ans de retard pour cet éditeur pionnier qui semble s'endormir sur ses lauriers). Il faut donc utiliser une syntaxe propriétaire. Exception : la version 9 supporte enfin les jointures normalisées.

(2) Possible avec un **FULL OUTER JOIN**



(3) Possible avec l'ancienne syntaxe sans précision de critère **WHERE**

(4) de plus IBM DB2 (400) dispose d'un très intéressant "**exception join**" équivalent à :

Sélectionnez

```
SELECT *  
FROM tablegauche  
      LEFT OUTER JOIN tabledroite  
        ON références de jointure  
WHERE tabledroite.clef IS NULL
```

Données concernant DB2 400 aimablement communiquées par kerigan.

Vous avez aimé ce tutoriel ? Alors partagez-le en cliquant sur les boutons suivants :  

  Partager

Copyright © 2004 Frédéric Brouard. Aucune reproduction, même partielle, ne peut être faite de ce site ni de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Contacter le responsable de la rubrique SGBD & SQL

[Nous contacter](#) [Participez](#) [Hébergement](#) [Informations légales](#) [Partenaire : Hébergement Web](#)

© 2000-2019 - www.developpez.com