BNF 101 – compléments de cours :

Jointures et requêtes select imbriquées

Les exemples de requêtes ci-dessous fonctionnent sur la plupart des SGBD :
Oracle, MySQL, Postgres..

I/ Le produit cartésien

Soit 2 tables :
table1(attribut1,attribut2) et
table2(attribut2,attribut3)

Le produit cartésien produira :

table1*table2(table1.attribut1,table1.attribut2,table2.attribut2,table2.attribut3)

Avec des valeurs, on aura donc :
On considère que l'on 2 enregistrements (lignes) par table.
table1 (a 1 b 2)
table2 (1 300 2 400)

table1*table2
(a 1 1 300)
(a 1 2 400)
(b 2 1 300)
(b 2 2 400)

SELECT nomattribut1,...,nomattributN FROM nomdetable1,nomdetable2,....,nomdetableN;

Cependant cette méthode est à éviter car elle est lente et elle fournit des doublons.

II/ Les jointures

table1
a 1
b 2

table2
1 300
2 400

table jointure (résultat)
a 1 300
b 2 400

L'exemple ci-dessus est le cas d'une jointure dite NATURELLE, elle est associative et commutative.

Cas théorique d'une jointure naturelle, c'est à dire table1.attribut2=table2.attribut2

SELECT nomattribut1,...,nomattributN FROM nomdetable1 INNER JOIN nomdetable2 ON nomdetable1.nomattribut=nomdetable2.nomattribut WHERE conditions;

III/ Les requêtes imbriquées ou internes

Les requêtes imbriquées se lisent de bas en haut (on lit d'abord la dernière requête) et elles peuvent être remplacées par des jointures mais la réciproque est fausse.

SELECT nomattribut FROM nomdetable WHERE nomattribut IN (SELECT nomattribut FROM nomdetable WHERE condition);

Lorsqu'on demande de valider ou d'optimiser une requête, il faut toujours autant que possible utiliser les jointures. Spécialement lorsque la requête possède des sous-requêtes.
Voici une des raisons:

SELECT * FROM tableA

1. WHERE champ1
2. IN (SELECT champ1 FROM tableB WHERE champ2 = 14 OR champ2 = 16)

Ce genre de requêtes doivent être évitées.
La quasi totalité de ce genre de requêtes peuvent être modifiées pour utiliser une jointure (INNER JOIN).

SELECT * FROM tableA

1. INNER JOIN tableB ON (tableA.champ1 = tableB.champ1)
2. WHERE (champ2 = 14 OR champ2 = 16)

Les résultats des 2 requêtes sont identiques mais la deuxième est beaucoup plus rapide.

La raison est fort simple.
Avec la première requête, aucun index ne peut être utilisé pour trier les valeurs de la table A. Les résultats sont épurés dans la condition selon le résultat de la sous-requête (la sous-requête en soit est bien optimisée).

Avec la deuxième requête, il peut épurer les enregistrements de la table A en utilisant l'index sur champ1 étant donné que celui-ci est utilisé pour la jointure.

Selon les tests réalisés, la première s'exécute en 0.015 seconde en moyenne pour 376 résultats.
La deuxième s'exécute en 0.003 secondes en moyenne pour 376 résultats.

IV/ Précisions sur les cas où une requête imbriquée ne ramène pas les mêmes résultats que les jointures.

Ci-dessous un exemple simple :

Soit les tables suivantes, et leur jeu de données associés :

CREATE TABLE TAB1 (COL1 INT) CREATE TABLE TAB2 (COL2 INT)	INSERT INTO TAB1 VALUES (1) INSERT INTO TAB1 VALUES (1) INSERT INTO TAB1 VALUES (2) INSERT INTO TAB2 VALUES (1) INSERT INTO TAB2 VALUES (2) INSERT INTO TAB2 VALUES (2)

Quel est donc l'équivalent, sans sous-requête et exprimé à l'aide uniquement de jointure, de la requête ?

Exemple :

SELECT TAB1.COL1 AS COL FROM TAB1 WHERE TAB1.COL1 IN (SELECT TAB2.COL2 FROM TAB2)	COL --- 1 1 2

Une jointure interne simple :

Exemple :

SELECT TAB1.COL1 AS COL FROM TAB1 INNER JOIN TAB2 ON TAB1.COL1 = TAB2.COL2	COL --- 1 1 2 2

il y a une valeur 2 en plus.

Rajoutons le dédoublonnage :

Exemple :

SELECT DISTINCT TAB1.COL1 AS COL FROM TAB1 INNER JOIN TAB2 ON TAB1.COL1 = TAB2.COL2	COL --- 1 2

il y a une valeur 1 en moins.

Si on détaille l'exemple de requête imbriquée,

```
SELECT TAB1.COL1 AS COL
FROM   TAB1
WHERE  TAB1.COL1 IN (SELECT TAB2.COL2 FROM TAB2)
```

est équivalent à

```
SELECT TAB1.COL1 AS COL
FROM   TAB1
WHERE  TAB1.COL1 IN (1,2,2);
```

L'opérateur IN compare chaque valeur de la table 1 à chaque valeur de la liste ci-dessus. Chaque fois qu'il y a correspondance, l'opérateur renvoie « vrai » et le moteur SQL exécute la ligne suivante de la requête principale.
Cela donne :
1 versus 1 → vrai
1 versus 1 → vrai
2 versus 1 → faux
2 versus 2 → vrai
→ r**ésultat 1,1,2**

Attention aux valeurs NULL !
Si la sous-requête utilisée dans le prédicat IN renvoie une valeur NULL, même pour une seule ligne de l'ensemble, le résultat de l'opération IN ou (NOT IN) ne sera jamais vrai : il vaudra toujours NULL et la requête ne renverra pas le résultat attendu.