Comment fonctionne la requête SELECT en SQL

By Thierry SPRIET thierry.spriet@univ-avignon.fr

La requête SELECT est composée de 5 clauses, seules les 2 premières sont obligatoires.

SELECT la liste des informations que l'on veut
FROM la liste des tables où se trouvent les infos
WHERE Conditions sur les tuples
GROUP BY attributs pour agrégation
HAVING Conditions sur les groupes
ORDER BY attributs pour tri de l'affichage

Les principes de base (squelette[1] 1)

select empno,ename from emp ;
c'est une **projection** sur la table emp, on ne garde que 2 colonnes, mais toutes les lignes.

*Select * from emp where deptno=20 ;*
c'est une **restriction**, on garde toutes les colonnes, mais QUE les lignes où le numéro de département est égal à 20.

Comment cela se passe t'il ?

SQL gère un pointeur implicite sur la table qui la parcoure ligne à ligne. **Pour chacune des lignes** la condition du where est évaluée. Si elle est vraie, la ligne est solution et donc affichée (du moins ce qui est demandé dans la clause select) ; le pointeur passe alors à la ligne suivante

Les jointures (squelette 2)

Quand l'information voulue provient de **plusieurs tables** on doit faire une jointure.
c'est à dire utiliser plusieurs tables. Pour l'algorithme, imaginez que SQL crée une table virtuelle qui est le résultat de cette jointure puis exécute sa requête sur cette table. Nous sommes alors dans le cas de base.
Pour créer cette table virtuelle, il y a plusieurs possibilités :

- jointure INTERNE (par défaut)

Sans condition elle correspond au «**produit cartésien**» des tables.
Avec condition elle permet d'associer uniquement certaines lignes de la première table avec certaines lignes de la seconde. Il y a 2 façon de faire :
la plus mauvaise est d'indiquer la condition dans la clause WHERE. En effet dans cette solution, la table virtuelle utilisée est le produit cartésien des tables pour ensuite supprimer les lignes qui ne respectent pas la condition de jointure. **La bonne façon** est donc d'indiquer dès la clause FROM la jointure et la condition de jointure; cela se fait avec :
emp JOIN dept ON condition
ou
emp JOIN dept USING (attribut)
ou
emp NATURAL JOIN dept

- jointure EXTERNE

Elle est utilisée quand on veut forcer l'association des lignes d'une table quand elles ne trouve pas leur association dans l'autre table. Par exemple :

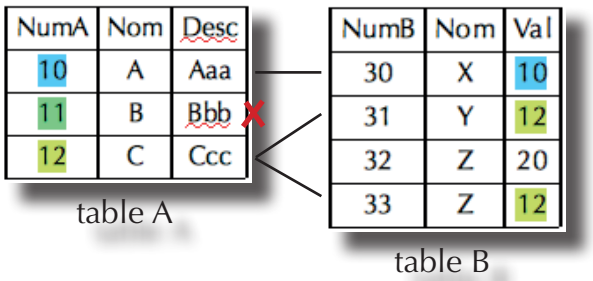

table A

NumA	Nom	Desc
10	A	Aaa
11	B	Bbb
12	C	Ccc

table B

NumB	Nom	Val
30	X	10
31	Y	12
32	Z	20
33	Z	12

Donne la jointure : A LEFT JOIN B ON NumA=Val1

NumA	A.Nom	Desc	NumB	B.Nom	Val
10	A	Aaa	30	X	10
11	B	Bbb	NULL	NULL	NULL
12	C	Ccc	31	Y	12
12	C	Ccc	33	Z	12

Dans ce cas, la ligne 11 de la table A s'associe avec un tuple «vide» (un tuple de valeurs NULL) de la table B car il ne trouve pas d'association possible (pas de VAL1 égale à 11). La jointure

1 les squelettes principaux de la requête SELECT peuvent être retrouvés dans le document «les squelettes du SELECT»

EXTERNE permet de forcer cette association. Les jointures externes se font en rajoutant un mot clef **RIGHT** ou **LEFT** devant JOIN.
RIGHT indique que l'on veut forcer l'association des tuples de la table de droite , donc qu'ils s'associent avec un tuple NULL dans l'autre table. LEFT permet de faire dans l'autre sens.

La condition de jointure peut être exprimée avec un ON suivi d'une condition (NumA=Val1 dans notre exemple) ou par un USING suivi du ou des attributs sur lesquels on veut faire la jointure (par exemple USING Nom) ou encore avec l'usage d'une jointure naturelle (FROM A NATURAL RIGHT JOIN B) qui fait la jointure sur les attributs de même nom.

- sous requête

cela consiste à remplacer la clause FROM par le résultat d'une requête SQL (Attention, cette méthode n'est pas à privilégier pour des raisons d'optimisation lors de l'exécution de vos requêtes).

Cette méthode consiste donc à réduire le champ de recherche pour la requête principale. Cela revient souvent à remonter les conditons de jointure dans une sous requête de clause FROM alors que la même requête peut être faite soit avec une jointure conditionnelle (ON ou USING) ou une condition de la clause WHERE.

Par exemple la requête sur les tables A et B précédente permettant de *donner le nom des A ayant un NumA qui apparait comme Val1 dans la table B* et dont le NumA est supérieur à 15.

```
SELECT C.nom
FROM (SELECT nom,NumA
      FROM A
      where Numa>15) as C
 JOIN B ON (NumA=Val1) ;
```

peut s'écrire plus simplement comme cela :

```
SELECT A.nom
FROM A  JOIN B ON (NumA=Val1)
where Numa>15;
```

A noter, que lorsqu'on utilise une sous-requête dans la clause FROM, il faut la renommer afin que les attributs puissent être préfixés si besoin (as C dans notre exemple)

L'utilisation de sous requête dans la clause FROM peut cependant trouver son intérêt pour contourner la non prise en charge par certains SGBD (dont PostgreSQL) de l'imbrication des fonctions de groupes (voir ci-dessous).

Les conditions du WHERE (squelette 3)

La clause WHERE est évaluée à chaque ligne de la table obtenue par la clause FROM. La condition qui y est décrite doit donc avoir un sens booléen pour chacune de ces lignes et doit donc concerner des valeurs attributs de la ligne courante.

squelette 3.1

```
SELECT nom
FROM A
where NumA>15;
```

Si l'on veut comparer avec des valeurs autres, il va falloir utiliser une sous requête.

Par exemple pour savoir si le NumA de la ligne courante est plus grand que la moyenne des Val1 de la table B :

squelette 3.2

```
SELECT nom
FROM A
WHERE NumA > (SELECT avg(Val1) FROM B);
```

Dans les sous requêtes on peut utiliser des attributs de la requête principale (plus généralement d'une requête de niveau supérieur). Il faut cependant faire attention à lever l'ambiguïté sur la provenance des atrtributs (si ambiguïté il y a).

Ainsi si dans l'exemple précéden, on veut exclure du calcul de la moyenne des Val1 les valeurs correspondantes au même nom que le NumA que l'on traite on pourra faire :

```
SELECT nom
FROM A
WHERE NumA > (SELECT avg(Val1) FROM B
              WHERE A.nom<>B.nom);
```

Attention la portée des attributs ne marche pas dans l'autre sens, dans la requête précédente nous n'aurions pas pu utiliser les attributs de la table B dans la requête principale (ce qui explique qu'il n'y a pas d'ambiguïté dans la clause SELECT avec le nom ; le seul nom connu à ce niveau est bien celui de la table A)

- le cas particulier du exists

A utiliser dans la clause WHERE, la commande EXISTS permet de tester l'existence de valeurs particulières (ou la non existence avec NOT EXISTS). Elle est bien adaptée à des requête du style « *Les nom de A pour lesquels il n'existe pas dans B de Val1 correpondant à leur NumA* »

```
SELECT nom
FROM A
WHERE not exists (SELECT NumB FROM B
                  WHERE NumA=Val1);
```

Le GROUP BY (squelette 4)

Dans un select classique les résultats sont donnés à et pour chaque ligne solution. Par exemple :

```
SELECTnom FROM A ;
```

donne le nom de tous les personnes de la table A. On peut aussi utiliser des fonctions dans le champ SELECT par exemple :

```
SELECT upper(nom) FROM A ;
```

Ces fonctions s'appliquent à chaque tuple. (ici elle met en majuscule les noms avant de les afficher.

Quand on utilise la clause GROUP BY, les résultats sont donnés **pour chaque groupe de lignes**.

Ce qui veut dire que dans la clause SELECT on ne peut demander **QUE** des résultats concernant le groupe complet de lignes, c'est à dire qui ne renvoie **QU'UNE SEULE** valeur par groupe de lignes.

Par exemple : « quel est la valeur maximum de Val1 par nom de B»

```
SELECT max(val1)
FROM B
GROUP BY nom ;
```

La clause SELECT renvoie bien ici une seule valeur pour chaque groupe de lignes, on pourrait aussi demander l'affichage le numéro NumB

```
SELECT max(val1), nom
FROM B
GROUP BY nom ;
```

puisque cette information est aussi unique par groupe.

Par contre impossible de demander l'affichage, dans cette requête du NumB correspondant

```
SELECT max(val1),nom,NumB
FROM B
GROUP BY nom;
```

En effet, pour chaque groupe de nom de B il peut y avoir plusieurs valeurs de NumB.

La clause HAVING (squelette 5)

La clause HAVING permet dans une requête avec un GROUP BY de ne garder que certains groupes de ligne comme solution. Par exemple : «*quels sont les* ***nom*** *de B qui plus de 20 pour la totalité de leurs* ***Val1***»

```
SELECT sum(val1),nom
FROM B
GROUP BY nom
HAVING sum(val1) > 20;
```

squelette 5.1

La clause having ne se substitue pas à la clause where, elles se complètent. Ainsi si l'on veut : «*quels sont les* ***nom*** *de B qui plus de 20 pour la totalité de leurs* ***Val1*** *excepté pour les* ***NumB*** *> 32*»

Il faut garder les tuples ayant *NumB<=32* grâce à une condition WHERE et s'occuper de la somme supérieur à 20 dans la clause HAVING.

```
SELECT sum(val1),nom
FROM B
WHERE NumB <=32
GROUP BY nom
HAVING sum(val1) > 20;
```

La clause HAVING peut aussi contenir une sous requête. (**squelette 5.2**)

Les conditions de la clause HAVING portant sur les agrégations (groupes de lignes) la sous-requête est la plus part du temps elle même une requête avec une clause GROUP BY.

Ce squelette 5.2 répond souvent à des questions du type : « quels sont les groupes qui ont le plus (ou moins ou la moyenne ou la somme) de quelque chose que les autres groupes. Cela demande en effet de comparer le résultat d'une fonction de groupe (sum, max, min, avg, count, stdder) d'un groupe par rapport aux autres groupes. exemple : «*quel est le* nom *de B ayant la moyenne des* val1 *la plus grande*»

```
SELECT avg(val1),nom
FROM B
GROUP BY nom
HAVING avg(val1) > ( SELECT avg(val1)
                     FROM B
                     GROUP BY nom) ;
```

On remarque dans cet usage que la sous requête est quasi la même que la requête principale.

C'est ainsi que l'on peut résoudre le problème évoqué plus haut sur la concaténation des fonctions de groupes MAX(AVG()) qu'il n'est pas possible de faire directement dans la clause SELECT.

Par contre si nous voulons imbriquer AVG(COUNT()) on ne peut pas marcher utiliser la même construction car aucun groupe de lignes aura comme résultat la moyenne des COUNT. Dans ce cas, on doit utiliser une table temporaire dans la clause FROM à l'aide d'une sous requête.

xemple : «*la moyenne du nombre de lignes dans la table B par* nom»

```
SELECT avg(TT.nbl) FROM (SELECT count(*) as nbl
                         FROM B
                         GROUP BY nom) as TT ;
```

Attention à ne pas oublier de nommer la sous requête (as TT) ainsi que ses résulats à réutiliser (as nbl).