

Annexe : Les opérations ensemblistes

par Pierre Caboche.

Le langage SQL est un langage extrêmement expressif. Aussi, il est très facile d'exprimer des opérations complexes sur des ensembles au moyen de requêtes SQL. Il est alors extrêmement tentant de traduire un cahier des charges de manière littérale en langage SQL, grâce aux nombreuses instructions du langage (`WHERE ... IN`, `WHERE ... NOT IN`, `EXCEPT`, `INTERSECT`, etc.) ainsi qu'à l'aide de requêtes imbriquées. C'est alors que survient le danger d'écrire des requêtes à la fois peu performantes et souvent fort peu lisibles.

Par ailleurs, MySQL est un SGBD extrêmement jeune, et les différentes fonctionnalités du SQL standard y sont ajoutées au fil des versions. Certains opérateurs comme `EXCEPT` ou `INTERSECT` ne sont pas du tout reconnus et il est souvent nécessaire de trouver des solutions alternatives à ces problèmes.

Dans ce document, nous allons présenter les différentes opérations ensemblistes au travers d'exemples concrets.

Ensembles de référence

Pour illustrer ce document, nous nous appuyerons sur les mêmes ensembles de référence, tirés du livre « Le SQL de MySQL » écrit par Christian Soutou.

On disposera des ensembles suivants :

- Les avions d'Air France (`AvionsdeAF`)
- Les avions de la compagnie Singapour Airline (`AvionsdeSing`)
- Les pilotes

Script de création des tables
<pre>CREATE TABLE AvionsdeAF (immat CHAR(6), typeAvion CHAR(10), nbHVol DECIMAL(10,2), CONSTRAINT pk_AvionsdeAF PRIMARY KEY (immat)); CREATE TABLE AvionsdeSING (immatriculation CHAR(6),typeAv CHAR(10),prixAchat DECIMAL(14,2),CONSTRAINT pk_AvionsdeSING PRIMARY KEY(immatriculation)); CREATE TABLE Pilote (brevet VARCHAR(6) PRIMARY KEY, nom VARCHAR(16),nbHVol DECIMAL(7,2),compa CHAR(4));</pre>

Dans ces tables, nous allons insérer les données suivantes :

Insertion des données
<pre>INSERT INTO AvionsdeAF VALUES ('F-WTSS', 'Concorde', 6570); INSERT INTO AvionsdeAF VALUES ('F-GLFS', 'A320', 3500); INSERT INTO AvionsdeAF VALUES ('F-GTMP', 'A340', NULL);</pre>

```

INSERT INTO AvionsdeSING VALUES ('S-ANSI', 'A320', 104500);
INSERT INTO AvionsdeSING VALUES ('S-AVEZ', 'A320', 156000);
INSERT INTO AvionsdeSING VALUES ('S-MILE', 'A330', 198000);
INSERT INTO AvionsdeSING VALUES ('F-GTMP', 'A340', 204500);

INSERT INTO Pilote VALUES ('PL-1', 'Gratien Viel', 450, 'AF');
INSERT INTO Pilote VALUES ('PL-2', 'Richard Grin', 1000, 'SING');
INSERT INTO Pilote VALUES ('PL-3', 'Placide Fresnais', 2450, 'CAST');
INSERT INTO Pilote VALUES ('PL-4', 'Daniel Vielle', 5000, 'AF');

```

Les requêtes imbriquées

Pour pallier à l'absence d'instructions telles que `INTERSECT` ou `EXCEPT`, de nombreux utilisateurs ont recours à des instructions telles que `WHERE ... IN` ou `WHERE ... NOT IN`, ce qui n'est pas toujours très judicieux.

En effet, l'usage de telles instructions nécessite le recours à des requêtes imbriquées, qui non seulement ne sont pas disponibles dans les versions de MySQL antérieures à la 4.1, mais en plus entraînent une baisse significative des performances. Par ailleurs, le fait d'avoir une requête à l'intérieur d'une autre requête rend la lecture difficile et par conséquent le code devient plus difficile à maintenir.

Dans de nombreux cas, il est tout à fait possible de se passer des requêtes imbriquées. C'est ce que nous exposerons dans ce document.

L'intersection

En MySQL, l'opérateur `INTERSECT` n'est pas implémenté.

Le résultat de l'intersection entre les ensembles A et B est l'ensemble des éléments qui appartiennent à la fois à l'ensemble A et à l'ensemble B. Partant de cette définition, de nombreux utilisateurs ont recours à l'instruction `WHERE ... IN`. Il est toutefois préférable d'avoir recours à une jointure de type `INNER`.

Exemples :

Requête non optimisée	Requête optimisée
<pre> SELECT DISTINCT typeAvion FROM AvionsdeAF WHERE typeAvion IN (SELECT typeAv FROM AvionsdeSING) ; </pre>	<pre> SELECT DISTINCT typeAvion FROM AvionsdeAF INNER JOIN AvionsdeSING ON (AvionsdeAF.typeAvion = AvionsdeSING.typeAv) </pre>
<pre> SELECT DISTINCT typeAv FROM AvionsdeSING </pre>	<pre> SELECT DISTINCT typeAv FROM AvionsdeSING </pre>

WHERE typeAv IN (SELECT typeAvion FROM AvionsdeAF);	INNER JOIN AvionsdeAF ON (AvionsdeAF.typeAvion = AvionsdeSING.typeAv)
SELECT DISTINCT immatriculation, typeAv FROM AvionsdeSING WHERE immatriculation IN (SELECT immat FROM AvionsdeAF) AND typeAv IN (SELECT typeAvion FROM AvionsdeAF);	SELECT DISTINCT immatriculation, TypeAv FROM AvionsdeSING INNER JOIN AvionsdeAF ON (AvionsdeAF.immat = AvionsdeSING.immatriculation AND AvionsdeAF.typeAvion = AvionsdeSING.typeAv) ;

L'union

L'opérateur d'UNION est parfaitement implémenté sous MySQL.

Il existe deux types d'union :

- L'union avec suppression des doublons, notée UNION (version abrégée) ou UNION DISTINCT
- L'union sans suppression des doublons, notée UNION ALL

L'union sans suppression de doublons (UNION ALL) est donc beaucoup plus rapide et doit être privilégiée chaque fois que cela est possible (quand on sait que l'on n'aura pas de doublons). Par défaut (UNION), l'union supprime les doublons.

Exemples :

Requête	Commentaires
SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING;	Supprime les doublons
SELECT typeAvion FROM AvionsdeAF UNION DISTINCT	Pareil que la requête précédente Ne marche pas sur toutes les versions de MySQL

<pre>SELECT typeAv FROM AvionsdeSING;</pre>	
<pre>SELECT typeAvion FROM AvionsdeAF UNION ALL SELECT typeAv FROM AvionsdeSING;</pre>	Ne supprime pas les doublons (plus rapide)

Au niveau du nommage des colonnes, voici comment se comporte l'opérateur d'union : l'UNION prend le nom des colonnes de la première requête, renvoie le résultat de la première requête et y ajoute les lignes de la deuxième requête, pour peu que les types concordent. Aussi, il est possible de définir des requêtes dont les données renvoyées n'ont strictement aucun sens :

Requête	Commentaires
<pre>SELECT nbhVol FROM AvionsdeAF UNION SELECT prixAchat FROM AvionsdeSING ;</pre>	<p>La requête n'a aucun sens : on mélange le nombre d'heure de vol des avion d'Air France au prix d'achat des avions de Sigapour Airline.</p>

La différence

La différence s'obtient grâce à l'opérateur EXCEPT (aussi connu sous le nom de MINUS dans certains SGBD). Cet opérateur n'est pas implémenté sous MySQL.

Pour pallier à cette absence, de nombreux utilisateurs ont recours à l'instruction WHERE ... NOT IN. Il est toutefois préférable d'avoir recours à une jointure de type LEFT OUTER et de faire un test WHERE ... IS NULL.

Requête non optimisée	Requête optimisée
<pre>SELECT DISTINCT typeAvion FROM AvionsdeAF WHERE typeAvion NOT IN (SELECT typeAv FROM AvionsdeSING);</pre>	<pre>SELECT DISTINCT typeAvion FROM AvionsdeAF LEFT OUTER JOIN AvionsdeSING ON (AvionsdeAF.typeAvion = AvionsdeSING.typeAv) WHERE AvionsdeSING.typeAv IS NULL</pre>
<pre>SELECT DISTINCT typeAv FROM AvionsdeSING WHERE typeAv NOT IN (SELECT typeAvion FROM AvionsdeAF);</pre>	<pre>SELECT DISTINCT typeAv FROM AvionsdeSING LEFT OUTER JOIN AvionsdeAF ON (AvionsdeSING.typeAv = AvionsdeAF.typeAvion) WHERE AvionsdeAF.typeAvion IS NULL</pre>

Le produit cartésien

Le produit cartésien est un opération qui consiste à faire toutes les combinaisons possibles entre les enregistrements de deux tables. Il est aussi appelé `CROSS JOIN`.

On obtient les combinaisons possibles entre les pilotes et les avions d'Air France de cette façon :

Requête	Commentaires
<pre>SELECT p.brevet, avAF.immat FROM Pilote p, AvionsdeAF avAF WHERE p.compa = 'AF' ;</pre>	Toutes les combinaisons possibles entre les pilotes et les avions d'Air France



Le produit cartésien est souvent dénigré car il renvoie énormément de lignes : le nombre de lignes renvoyées est égal au nombre d'enregistrements dans la première table, multiplié par le nombre d'enregistrements dans la deuxième table.

mais...



Le produit cartésien est très utile en association avec un `LEFT JOIN` pour avoir la liste des combinaisons qui n'existent pas dans une association (par exemple, lorsque l'on veut avoir une liste des conditions qui ne sont pas remplies).

Imaginons que l'on ait une association `A_Pilote_AF` entre les pilotes et les avions d'Air France. Pour avoir la liste des combinaisons pilote/avion qui n'apparaissent pas dans cette association, on fait :

Requête	Commentaires
<pre>SELECT p.brevet, avAF.immat FROM Pilote p, AvionsdeAF avAF LEFT OUTER JOIN A_Pilote_AF ap ON (ap.Pilote = nom AND ap.immat = avAF.immat) WHERE p.compa = 'AF' AND ap.Pilote IS NULL;</pre>	La condition <code>IS NULL</code> permet de ne conserver que les combinaisons qui n'apparaissent pas dans <code>A_Pilote_AF</code> .

Ordonner un résultat

La clause `ORDER BY` permet d'ordonner les résultats d'une requête.

Comme nous l'avons vu dans le cas de l'`UNION`, le nom des colonnes sont ceux issus de la première requête. De là, plusieurs façon de faire :

- se baser sur les noms des colonnes de la première requête :

Requête	Commentaires
<pre>SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING ORDER BY typeAvion DESC;</pre>	La condition <code>IS NULL</code> permet de ne conserver que les combinaisons qui n'apparaissent pas dans <code>A_Pilote_AF</code> .

- faire référence au numéro de la colonne :

Requête	Commentaires
<pre>SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING ORDER BY 1 DESC;</pre>	Il est ambigu de faire référence aux colonnes en se basant sur leur numéro.

Cette manière de faire est déconseillée (on doit pouvoir décider à tout moment de changer l'ordre des colonnes sans que cela n'ait d'influence sur le traitement)

- pour plus de clarté, effectuer un renommage des colonnes, surtout celles de la deuxième requête

Requête	Commentaires
<pre>(SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv AS typeAvion FROM AvionsdeSING) ORDER BY typeAvion DESC ;</pre>	Ici, on renomme les colonnes de la deuxième requête pour qu'elles correspondent à celles de la première requête.

La dernière est préférable, car à la fois claire (code plus facile à maintenir) et sans ambiguïté (les numéros de colonne peuvent changer).

Il est parfaitement inapproprié d'avoir recours à des requêtes imbriquées dans le seul but d'effectuer un tri :

Requête à ne pas reproduire	Requête équivalente
<pre>SELECT T.typeAvion FROM (SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING) T ORDER BY T.typeAvion DESC;</pre>	<pre>(SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv AS typeAvion FROM AvionsdeSING) ORDER BY typeAvion DESC ;</pre>

L'Ajout de colonnes

Pour certaines requêtes SQL, il est parfois nécessaire d'ajouter artificiellement des colonnes, dont la valeur sera donnée par une expression mathématique (basée ou non sur la valeur d'autres colonnes).

Cela se produit par exemple dans le cas :

- d'une UNION où l'une des colonnes manque dans une des requêtes :

Requête
<pre>SELECT immatriculation, prixAchat FROM AvionsdeSING UNION SELECT immat, 0 AS prixAchat FROM AvionsdeAF ;</pre>

- dans le cas d'un INSERT ... SELECT, quand on veut que les valeurs d'une certaine colonne soient toutes identiques :

Requête	Commentaires
<pre>INSERT INTO Pilote(nom, immat, heuresVol) SELECT p.brevet, avAF.immat, 0 AS heuresVol FROM Pilote p, AvionsdeAF avAF LEFT OUTER JOIN A_Pilote_AF ap ON (ap.Pilote = nom AND ap.immat = avAF.immat) WHERE p.compa = 'AF' AND ap.Pilote IS NULL ;</pre>	<p>On ajoute les combinaisons qui n'apparaissent pas dans A_Pilote_AF et on met 0 au nombre d'heures de vol.</p>



Il est vivement conseillé de spécifier le nom de la colonne ajoutée. On peut en effet constater des bugs sur certaines versions (lors d'un INSERT ... SELECT, par exemple)