SQL

Ch 6 – union, intersection, différence, etc.

I. Introduction

Les requêtes utilisant l'ordre SQL SELECT ont permis la construction de réponses sous forme de lignes d'un certain nombre de colonnes issues d'une ou plusieurs tables. En effet, grâce au mécanisme des jointures, on a pu ajouter des colonnes provenant de plusieurs tables.
D'autres mots clefs liés à l'ordre SELECT permettent d'enrichir le vocabulaire permettant de construire des réponses en s'appuyant sur les opérateurs de l'algèbre relationnelle.

II. UNION

Objectif :
Mettre bout à bout les résultats d'exécution de requêtes SQL SELECT différentes mais **union compatible** :

- possédant **le même nombre de colonnes**
- et ayant **le même type de données pour chaque colonne**.

Syntaxe :

```
 requete SQL SELECT
UNION [ALL]
 requete SQL SELECT
[ ORDER BY . . . ]
;
```

L'ajout du mot clef ALL permet de conserver les doublons.

Exemple 1 :
On souhaite envoyer un courrier à tous les tiers avec lesquels on travaille (aussi bien clients que fournisseurs) ; il faut rassembler les adresses des clients et les adresses des fournisseurs et on souhaite conserver une indication du type de tiers :

```
  SELECT "C" AS type, numCLi, nomCli, adrCli
    FROM T_client
    WHERE codPaysCli = "FR"
UNION
  SELECT "F" AS type, numFou, nomFou, adrFou
    FROM T_fournisseur
```

```
    WHERE codPaysFou = "FR";
```

Exemple 2:
Idem. Ci-dessus mais avec classement du résultat final

```
  SELECT "C" AS type, numCLi, nomCli, adrCli
    FROM T_client
    WHERE codPaysCli = "FR"
UNION
  SELECT "F" AS type, numFou, nomFou, adrFou
    FROM T_fournisseur
    WHERE codPaysFou = "FR"
ORDER BY numCli;
```

Exemple 3 :
On souhaite envoyer un courrier à certains clients : ceux qui ont un chiffre d'affaires inférieur à 100 et ceux qui sont Français

```
  SELECT numCli, nomCli
    FROM T_client
    WHERE CAcli < 100
UNION
  SELECT numCli, nomCli
    FROM T_client
    WHERE codPaysCli = "FR"
;
```

Équivalent à :

```
  SELECT numCLient, nomClient
    FROM T_client
    WHERE chiffreDAffaires < 100
          OR codPays = "FR";
```

III.INTERSECT

Objectif :
Retourner les lignes qui sont communes à plusieurs requêtes SQL différentes mais **union compatible** :

- possédant **le même nombre de colonnes**
- et ayant **le même type de données pour chaque colonne**.

Syntaxe :

```
 requete SQL SELECT
INTERSECT
 requete SQL SELECT
[ ORDER BY . . . ]
;
```

Exemple 1 :

On souhaite obtenir les numéros et nom des clients qui seraient aussi des fournisseurs (à condition qu'ils aient le même numéro et le même nom en tant que client et fournisseur) :

```
  SELECT numCli, nomCli
    FROM T_client
INTERSECT
  SELECT numFou, nomFou
    FROM T_fournisseur ;
```

Équivalent à :

```
  SELECT numCLi, nomCli
    FROM T_client
    WHERE (numCli, nomCli) IN
              (SELECT numFou, nomFou
                 FROM T_fournisseur);
```

IV. MINUS

Objectif :

Retourner les lignes résultant de l'exécution d'une requête SQL sans les lignes résultant de l'exécution d'une seconde requête SQL. Les 2 requêtes doivent

- posséder **le même nombre de colonnes**
- et avoir **le même type de données pour chaque colonne**.

Syntaxe :

```
 requete SQL SELECT
MINUS
 requete SQL SELECT
[ ORDER BY . . . ]
;
```

Exemple 1:

On souhaite obtenir les numéros et nom des clients qui ne sont que clients et pas fournisseurs (à condition qu'un client possède le même numéro et le même nom s'il est également fournisseur) :

```
  SELECT numCli, nomCli
    FROM T_client
MINUS
  SELECT numFou, nomFou
    FROM T_fournisseur ;
```

Exemple 2:

On souhaite envoyer un courrier à certains clients : ceux qui sont français et qui ont un chiffre d'affaires inférieur à 100 (il faut donc enlever ceux qui ont un CA supérieur ou égal à 100) :

```
  SELECT numCli, nomCli
    FROM T_client
    WHERE codPaysCli = "FR"
MINUS
  SELECT numCli, nomCli
```

```
    FROM T_client
    WHERE CAcli >= 100;
```

Équivalent à :

```
  SELECT numCli, nomCli
    FROM T_client
    WHERE codPays = "FR"
          AND chiffreDAffaires < 100;
```

V. Requête de division

Un exemple pour simplifier l'explication :

- **Membre** (id_memb, nom_memb, prenom_memb)
- **Activite** (id_activ, lib_activ)
- **Participer** (#id_memb, #id_activ, date_activ)

L'opération relationnelle de division permet de répondre à une question du type :

- **Quels sont les membres qui ont participé à toutes les activités ?**

Il n'existe aucun mot clef SQL pour exprimer l'opération relationnelle de la division. Une sous-requête sera utilisée. Ainsi, pour répondre à la question précédente, nous nous poserons la question suivante :

- **Quels sont les membres pour lesquelles le nombre d'activités différentes auxquels ils ont participé est égale au nombre total d'activités différentes :**

Soit en SQL :

```
  SELECT id_memb, nom_memb, prenom_memb
    FROM Membre M INNER JOIN Participer P
                       ON M.id_memb = P.id_memb
    GROUP BY id_memb, nom_memb, prenom_memb
    HAVING COUNT(*) =
                (SELECT COUNT(DISTINCT id_activ)
                    FROM Activite)
;
```

Ou bien (attention au « coût » de ce type de requête – requête imbriquée « corrélée ») :

```
  SELECT id_memb, nom_memb, prenom_memb
    FROM Membre M
    WHERE (SELECT COUNT(DISTINCT id_activ)
             FROM Participer
             WHERE id_memb = M.id_memb)
       =  (SELECT COUNT(DISTINCT id_activ)
             FROM Activite)
;
```

Attention : Ces 2 requêtes fonctionnent correctement dans ce cas précis car les comptages sont réalisés sur des colonnes clefs primaires et clefs étrangères avec contrainte d'intégrité référentielle forte)

Attention aux pièges pouvant fournir des résultats erronés sur ce type de requêtes :

cf. page http://sqlpro.developpez.com/cours/divrelationnelle/