Corrigé TP SQL N° 2 et 3

R1) *Quelles sont les personnes, les cafés qu'ils fréquentent, et les boissons servies par ces cafés.*

Deux tables suffisent: *frequente* et *sert* (colonne commune *cafe*)

```
 select f.personne, f.cafe, s.boisson
 from frequente  f, sert s
 where f.cafe=s.cafe
 order by 1,2;
```

```
+----------+----------+----------+
| personne | cafe     | boisson  |
+----------+----------+----------+
| ali      | atlas    | 7up      |
| ali      | atlas    | oulmes   |
| ali      | rif      | coca     |
| amine    | commerce | coca     |
| amine    | commerce | oulmes   |
| amine    | commerce | 7up      |
| amine    | commerce | orangina |
| aziz     | atlas    | 7up      |
| aziz     | atlas    | oulmes   |
| aziz     | commerce | orangina |
| aziz     | commerce | coca     |
| aziz     | commerce | oulmes   |
| aziz     | commerce | 7up      |
| aziz     | rif      | coca     |
| said     | atlas    | 7up      |
| said     | atlas    | oulmes   |
+----------+----------+----------+
```

`Order by` pour la présentation.

R2) *Quelles sont les personnes qui fréquentent des cafés qui servent des boissons qu'ils aiment.*

Il faut trois tables: jointure sur toutes les 3 colonnes communes

```
select f.personne, f.cafe, s.boisson
from frequente  f, sert s, aime a
where f.cafe = s.cafe
and  f.personne = a.personne
and s.boisson = a.boisson
order by 1,2
```

```
+----------+----------+---------+
| personne | cafe     | boisson |
+----------+----------+---------+
| ali      | atlas    | 7up     |
| ali      | atlas    | oulmes  |
| ali      | rif      | coca    |
| amine    | commerce | oulmes  |
| amine    | commerce | coca    |
| aziz     | atlas    | oulmes  |
| aziz     | atlas    | 7up     |
| aziz     | commerce | 7up     |
| aziz     | commerce | oulmes  |
+----------+----------+---------+
9 rows in set (0.00 sec)
```

Comparer par rapport à 1)

- `Amine` fréquente commerce, mais n'aime que `oulmes` et `coca`.
- `Said` *ne* fréquente *pas* de café qui sert une boisson qu'il aime (ne figure pas dans cette dernière table).
- `Ali` a le même nombre de lignes dans les deux résultats. Il *ne* fréquente *que* les cafés qui servent des boissons qu'il aime.

Question subsidiaire:

R2') *Quelles sont les personnes qui fréquentent des cafés (au moins un) qui servent des boissons qu'ils n'aiment pas.*

Idem que ci-dessus, mais avec la relation *aimePas*.

R2' - a) Créer la relation aimePas.

Il faut faire les hypothèses :
1) dans la table *aime* toutes les personnes sont listées et toutes les boissons aussi,
2) si une personne *p* n'est pas dans la table *aime* avec une boisson *b*, alors *p* n'aime pas *b*. (Hypothèse du monde clos, tout

ce qui n'est pas énoncé est faux)

Expression algébrique : *DIFF* (*CART* (*PROJECT* (aime, personne), *PROJECT* (aime, boisson)), aime)

On créera une vue pour cela :

```
create view aimePas as
select distinct a1.personne, a2.boisson
from  aime a1, aime a2
where not exists (select * from aime a3
                  where a1.personne=a3.personne
                  and a2.boisson=a3.boisson )

 select * from aimePas order by 1;

+----------+----------+
| personne | boisson  |
+----------+----------+
| ali      | orangina |
| amine    | orangina |
| amine    | 7up      |
| aziz     | coca     |
| aziz     | orangina |
| said     | 7up      |
| said     | oulmes   |
+----------+----------+
```

R2' - b) Résultat : on crée une relation *X* qui répond à la requête « *Quelles sont les personnes qui fréquentent des cafés (au moins un) qui servent des boissons qu'ils n'aiment pas* ».

```
create view X as select f.personne, f.cafe, s.boisson
     from frequente  f, sert s, aimePas a
     where f.cafe = s.cafe
     and  f.personne = a.personne
     and s.boisson = a.boisson;

 select * from X;
+----------+----------+----------+
| personne | cafe     | boisson  |
+----------+----------+----------+
| amine    | commerce | orangina |
| amine    | commerce | 7up      |
| aziz     | commerce | coca     |
| aziz     | rif      | coca     |
| aziz     | commerce | orangina |
| said     | atlas    | 7up      |
| said     | atlas    | oulmes   |
+----------+----------+----------+
```

Où on voit que, par rapport à la requête **R2**, Ali n'appartient pas à *X* (ne fréquente que les cafés qui servent des boissons qu'il aime :-)), et Said appartient à *X* mais pas à **R2** (ne fréquente que les cafés qui servent des boissons qu'il *n*'aime *pas* :-( )

R3) *Quels sont les café servant toutes les boissons.*

Ici, c'est une simple division relationnelle.

Expression algébrique: *DIV* (*sert* , *PROJECT* (sert, boisson))

```
select distinct cafe
from sert x
where not exists (select * from sert y
                  where not exists (select * from sert z
                                    where x.cafe = z.cafe
                                    and z.boisson = y.boisson));

+----------+
| cafe     |
+----------+
| commerce |
+----------+
1 row in set (0.00 sec)
```

R4) *Quelles sont les personnes qui ne fréquentent que les cafés qui servent des boissons qu'ils aiment (Ali).*

Soit:
X = Personnes fréquentant un café servant au moins une boisson qu'elles n'aiment pas (R2') et
Y = Personnes fréquentant un café servant au moins une boisson qu'elles aiment (R2)

réponse = Y - X (opérateur de différence, NOT EXiSTS)

```
select distinct personne
from Y
```

```
where not exists (select *
                  from X
                  where  X.personne = Y.personne)

+----------+
| personne |
+----------+
| ali      |
+----------+
```

R5) *Quelles sont les personnes qui ne fréquentent que les cafés qui servent des boissons qu'ils n'aiment pas (Said).*

réponse = X - Y

```
select distinct personne
from X
where not exists (select *
                  from Y
                  where  X.personne = Y.personne)

+----------+
| personne |
+----------+
| said     |
+----------+
```

/fin

Corrigé TP SQL N° 3

Relation parent initiale

```
parent
+--------+--------+
| parent | enfant |
+--------+--------+
| Ali    | Fatima |
| Ali    | Kacem  |
| Fatima | Amina  |
| Fatima | Aziz   |
| Kacem  | Aziza  |
| Aziz   | Saida  |
| Saida  | Farid  |
+--------+--------+
```

La relation frere

```
create view frere (f1, f2) as
select a.enfant, b.enfant
from parent a, parent b
where a.parent = b.parent
      and a.enfant > b.enfant;

select * from frere;

+-------+--------+
| f1    | f2     |
+-------+--------+
| Kacem | Fatima |
| Aziz  | Amina  |
+-------+--------+
```

Relation cousin

```
create view cousin (c1, c2) as
select a.enfant, b.enfant
from  parent a,  parent b, frere f
where a.parent = f.f2
     and b.parent = f.f1;

select * from cousin;

+-------+-------+
| c1    | c2    |
+-------+-------+
| Amina | Aziza |
```

```
| Aziz  | Aziza |
+-------+-------+
```

Relation oncle (ou tante)

```
create view oncle (o, n) as
select a.parent, b.enfant
from  parent a,  parent b, frere f
where (b.parent = f.f2
       and a.parent = f.f1)
     or (b.parent = f.f1
        and a.parent = f.f2);

select distinct * from oncle;

+--------+-------+
| o      | n     |
+--------+-------+
| Kacem  | Amina |
| Kacem  | Aziz  |
| Fatima | Aziza |
+--------+-------+
```

Pour avoir les oncles rajouter dans la clause where "`and a.parent in (select * from male);`"

```
create view oncle (o, n) as
       select a.parent, b.enfant
       from  parent a,  parent b, frere f
       where (b.parent = f.f2
              and a.parent = f.f1)
             or (b.parent = f.f1
                and a.parent = f.f2)
       and a.parent in (select * from male);


mysql> select * from oncle;
+-------+-------+
| o     | n     |
+-------+-------+
| Kacem | Amina |
| Kacem | Aziz  |
+-------+-------+
2 rows in set (0.00 sec)
```

Relation Grand parent `GP`

```
create view gp (gp, pf) as
select a.parent, b.enfant
from parent a, parent b
where a.enfant = b.parent;

select * from gp;

+--------+-------+
| gp     | pf    |
+--------+-------+
| Ali    | Amina |
| Ali    | Aziz  |
| Ali    | Aziza |
| Fatima | Saida |
| Aziz   | Farid |
+--------+-------+
```

Relation ancêtre

Rappel: Relation *ancêtre* définie récursivement par :

ancêtre (x, y) = *parent* (x, y) ou $\exists$ z , *ancêtre* (x, z) et *parent* (z, y)

Ici, nous avons affaire à une relation obtenue par fermeture transitive. un ancêtre est un parent ou un grand-parent ou un arrière grand-parent à un niveau quelconque. C'est ce niveau quelconque qui est indéfini à l'avance.

Méthode de calcule

Le calcul consiste donc à progresser, chercher le grand-parent, ensuite le parent du grand-parent, le parent de ce dernier, etc. Jusqu'à ne plus rien obtenir quand on a atteint le dernier ancêtre connu qui n'a donc pas de parent (connu) dans la base.

Reprenons la table `parent` ci-dessus et cherchons un à un les ancêtres, en commençant par le parent.

Cela donne :

- Ancêtre niveau 0, c'est à dire parent

```
CREATE VIEW anc0 (anc, des) AS SELECT * FROM parent;
```

```
+--------+--------+
| anc    | des    |
+--------+--------+
| Ali    | Fatima |
| Ali    | Kacem  |
| Aziz   | Saida  |
| Fatima | Amina  |
| Fatima | Aziz   |
| Kacem  | Aziza  |
| Saida  | Farid  |
+--------+--------+
7 rows in set (0.00 sec)
```

- Ancêtre niveau 1, grand-parent (ou parent de anc_0)

```
CREATE VIEW anc1 (anc, des) AS
  SELECT g.parent, p.des
  FROM parent g, anc0 p
  WHERE g.enfant = p.anc;
```

```
+--------+-------+
| anc    | des   |
+--------+-------+
| Ali    | Amina |
| Ali    | Aziz  |
| Ali    | Aziza |
| Aziz   | Farid |
| Fatima | Saida |
+--------+-------+
5 rows in set (0.00 sec)
```

- Ancêtre niveau 2, grand-grand-parent (ou parent de anc_1)

```
CREATE VIEW anc2 (anc, des) AS
  SELECT g.parent, p.des
  FROM parent g, anc1 p
  WHERE g.enfant = p.anc;
```

```
+--------+-------+
| anc    | des   |
+--------+-------+
| Ali    | Saida |
| Fatima | Farid |
+--------+-------+
2 rows in set (0.00 sec)
```

- Ancêtre niveau 3, parent de anc_2

```
CREATE VIEW anc3 (anc, des) AS
  SELECT g.parent, p.des
  FROM parent g, anc2 p
  WHERE g.enfant = p.anc;
```

```
+-----+-------+
| anc | des   |
+-----+-------+
| Ali | Farid |
+-----+-------+
1 row in set (0.00 sec)
```

- Ancêtre niveau 4 parent de anc_3

```
CREATE VIEW anc4 (anc, des) AS
  SELECT g.parent, p.des
  FROM parent g, anc3 p
  WHERE g.enfant = p.anc;
```

```
Empty set (0.00 sec)
```

- Pas d'ancêtre niveau 4.

La liste finale de tous les ancêtres est l'union de anc_0 à anc_4

```
SELECT * FROM anc0
UNION SELECT * FROM anc1
UNION SELECT * FROM anc2
UNION SELECT * FROM anc3
UNION SELECT * FROM anc4
order by 1;
```

```
+--------+--------+
| anc    | des    |
+--------+--------+
| Ali    | Fatima |
```

```
| Ali    | Saida  |
| Ali    | Kacem  |
| Ali    | Amina  |
| Ali    | Aziz   |
| Ali    | Farid  |
| Ali    | Aziza  |
| Aziz   | Saida  |
| Aziz   | Farid  |
| Fatima | Farid  |
| Fatima | Amina  |
| Fatima | Aziz   |
| Fatima | Saida  |
| Kacem  | Aziza  |
| Saida  | Farid  |
+--------+--------+
15 rows in set (0.00 sec)
```

En fait, c'est l'union ensembliste des relations $\boldsymbol{ancêtre}_i$ où :

- ancêtre$_1$ de niveau 1 (le parent du parent)
- ancêtre$_2$ de niveau 2 (le parent du ancêtre$_1$)
- ...
- ancêtre$_n$ de niveau n (le parent du ancêtre$_{n-1}$)

jusqu'à ce qu'on obtienne une relation vide, i.e. on a atteint le dernier ancêtre connu.

Forme Algébrique

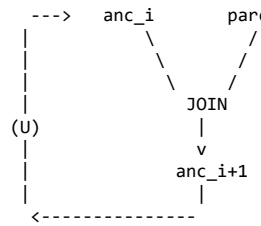
Soit $\boldsymbol{ancêtre}_i$ (*anc*, *desc*) la table mettant en relation an ancêtre avec un descendant.

- $\boldsymbol{ancêtre_0}$ = *parent*
- $\boldsymbol{ancêtre_1}$ = *project* (*join* (*parent a, parent b, a.enfant = b.parent*), *a.parent, b.enfant*)
- $\boldsymbol{ancêtre_2}$ = *project* (*join* ($\boldsymbol{ancêtre_1}$ *a, parent b, a.enfant = b.parent*), *a.parent, b.enfant*)
- ...
- $\boldsymbol{ancêtre_n}$ = *Project* (*join* ($\boldsymbol{ancêtre_{n-1}}$ *a, parent b, a.enfant = b.parent*), *a.parent, b.enfant*)

Résultat = $\cup_{i=0..n}$ $\boldsymbol{ancêtre_n}$.

Programmation avec SQL

Il faut faire un programme (e.g. PLSQL, ESQL ou php-MySQL) pour calculer cette relation, par récursion ou boucle *while*.



On va initialement créer la table finale `ancetre (ancetre, descendant)`, qui sera alimentée au fur et à mesure par les tuples de anc_i calculés à chaque itération. Représenté par `(U)` sur la figure.

Voici un programme PHP-MySQL qui calcule cette table `ancêtre`. Dans ce listing, `last_anc` et `new_anc` sont respectivement les deux table "variables de contrôle" `anc_i` et `anc_i+1` de la figure ci-dessus.

```
<?php

    //
    // On se connecte au serveur
    //

    $link = mysql_connect('localhost', 'Najib');

    //
    // On choisit la base
    //

    $c = mysql_select_db("Famille",$link);

    //
    // On Traite
    //

        mysql_query("drop table if exists ancetres;", $link);
        mysql_query("drop table if exists  last_anc;", $link);
        mysql_query("drop view if exists  new_anc;", $link);
```

```
/********* table ancetres initialement vide ************/
    mysql_query("create table ancetres (ancetre text(10), descendant text(10));", $link);

/*************** initialisation: last_anc := parent  *********************/
    mysql_query(" create table last_anc (anc text(10), des text(10));", $link);
    mysql_query(" insert into last_anc select * from parent;",$link);

/* A cumuler sur ancetre */
    mysql_query("insert into ancetres select * from last_anc; ",$link);

/***************  boucle  *********************/

/*****      On calcule new_anc      *****/
    while (1) {
        mysql_query("drop table if exists new_anc; ", $link);

        mysql_query("create table new_anc (anc text(10), des text(10));", $link);
        mysql_query("insert into new_anc select g.parent, p.des from parent g, last_anc p where g.enfant = p.anc;", $link);


        /***** on teste si le résultat new_anc n'est pas vide **********/
        $result = mysql_query("SELECT count(*) as cpt FROM new_anc", $link);
        $n = mysql_result($result, 0, cpt);   // $n contient le nombre de tuples trouvés

        if ($n > 0) {
               /** il y a résultat  **/
               /***   On cumule sur ancetres        ***/
              mysql_query("insert into ancetres select * from new_anc;", $link);

              /***  last_anc := new_anc, et on continue  ***/
              mysql_query("delete from last_anc;", $link);
              mysql_query("insert into last_anc select * from new_anc;", $link);
        } else {
              break; /* resultat: ancetres */
        }
     }
/*********  fin boucle **********/

/*******  affichage   *******/
 afficherRelation ("ancetres", $link);


 /******** Fonction  afficherRelation ********/
 function afficherRelation($table, $link)
 {   /*** affiche une table new_anc (anc, des) ***/

     //
     // On interroge
     //
     echo "<b>$table</b>";
     $result = mysql_query("SELECT * FROM $table order by 1",$link);

     //
     // On presente le resultat en table
     // Usage de fetch_row et de row[n° colonne]
     //

     printf( " <table border=1 cellspacing=0 cellpadding=3>");
     printf("<tr>\n");
     printf( "<th>anc</th><th>des</th>");
     printf( "</tr>\n");

     while ($row = mysql_fetch_row($result)) {
     printf("<tr><td>%s</td><td>%s</td></tr><br />\n",
            $row[0], $row[1]);
     }

     printf( "</table>");

     return;
 }
 ?>
```

Dans ce programme, le résultat est affiché dans une page HTML. Seule la partie `<table>...</table>` est considérée.

/That's all folks.