Utilisation des sous-requêtes

- Le terme sous-requête désigne toute requête utilisée à l'intérieur d'une requête dite principale ou externe.
- Elle retourne comme toute requête un résultat qui est une relation qui peut comporter une ou plusieurs colonnes et aussi une ou plusieurs lignes
- La sous requête sert en général pour comparer un attribut ou un ensemble d'attributs de la requête principale au résultat retourné par la sous requête.
- Il faut remarquer au passage que les sous requêtes **peuvent aussi être utilisées** dans les opérations **d'insertion**, de mise à jour et de **suppression** de tuples dans une table.

❖ Sous-requête retournant une seule valeur

Q19: **Quels sont les noms des employés ayant la même fonction que l'employé 'OMAR'**

```
SELECT   Nom_Emp
FROM     EMPLOYES
WHERE    Fonction =
            ( SELECT    Fonction
              FROM      EMPLOYES
              WHERE     Nomp_Emp  ='OMAR' )
```

- La sous requête doit être placée entre parenthèses.
- Elle est évaluée en premier et va retourner une seule valeur 'DIRECTEUR'.
- Cette valeur sera utilisée pour constituer la condition de sélection de la clause WHERE de la requête principale qui sera donc :

Fonction = 'DIRECTEUR'.

❖ Sous-requête retournant un ensemble de valeurs

- Lorsqu'une sous requête retourne un ensemble de valeurs ,

- il faudra **faire précéder** l'opérateur de comparaison (= , != , > , < , <= , >=) de la clause WHERE de la requête principale avec un des **mots clefs ANY ou ALL**.

Q20: **Quels sont les noms des employés ayant un salaire supérieur à celui d'un employé quelconque du département 30 ?**

```
SELECT    Nom_Emp
FROM      EMPLOYES
WHERE     Salaire > ANY
          ( SELECT    Salaire
            FROM      EMPLOYES
            WHERE     Nump_Dept  = 30 )
```

- La sous requête va retourner un ensemble de valeurs représentant les salaires du département 30.

- Pour chaque employé, la requête principale va comparer son salaire avec l'ensemble des valeurs retournées par la sous requête.

- Si le salaire est supérieur à une valeur quelconque de cet ensemble, cet employé sera inclus dans le résultat.

Q21: **Quels sont les noms des employés ayant un salaire supérieur à celui de tous les employés du département 30 ?**

```
SELECT      Nom_Emp
FROM        EMPLOYES
WHERE       Salaire > ALL
                ( SELECT      Salaire
                  FROM        EMPLOYES
                  WHERE       Nump_Dept  = 30 )
```

On peut dans le cas ou la condition est = ANY , remplacer ce test par l'opérateur **IN**.

- De même que != ALL peut être remplacer par **NOT IN**.

Q22: **Quels sont les noms et les fonctions des employés du département 10 ayant la même fonction qu'un employé quelconque du département 30 ?**

```
SELECT      Nom_Emp , Fonction
FROM        EMPLOYES
WHERE       Num_Dept = 10
AND         Fonction IN
                ( SELECT      Fonction
                  FROM        EMPLOYES
                  WHERE       Nump_Dept  = 30 )
```

❖ Sous-requête retournant plusieurs colonnes

- Une sous requête peut retourner plus d'une colonne si la liste des attributs de sa clause SELECT comprend plus d'un attribut.
- Le nombre d'attributs de la requête principale à comparer avec l'ensemble des lignes (ou valeurs) retournées par la sous requête **doit être égale** au nombre de **colonnes** retournées par la **sous requête**.

Q23: **Quels sont les employés ayant le même salaire et la même fonction que l'employé 'Omar' ?**

```
SELECT    *
FROM      EMPLOYES
WHERE     (Salaire , Fonction) =
                    ( SELECT    Salaire , Fonction
                      FROM      EMPLOYES
                      WHERE     Nom_Emp  = 'Omar' )
```


- Il est aussi possible **d'utiliser plusieurs sous requêtes imbriquées** et construire ainsi des requêtes aussi complexes qu'on le souhaite. C'est ce qui fait la puissance du langage SQL.
- Le nombre autorisé de sous requêtes imbriquées dépend de l'implémentation du langage. A titre d'exemple les premières versions du SGBD ORACLE (V3 et V4 datant de 1982-1983) autorisait **jusqu'à 16 sous requêtes imbriquées** en plus de la requête principale.

Q24: **Quels sont les noms et les fonctions des employés ayant la même fonction que l'employé 'Omar' ou un salaire supérieur ou égal à celui de 'Rachid'?**

```
SELECT   Nom_Emp , Fonction
FROM     EMPLOYES
WHERE    Fonction =
                    ( SELECT   Fonction
                      FROM     EMPLOYES
                      WHERE    Nom_Emp = 'Omar' )

                    OR         Salaire > =

                    ( SELECT   Salaire
                      FROM     EMPLOYES
                      WHERE    Nom_Emp = 'Rachid' )
```


❖ Utilisation des sous requêtes dans une Jointure

- Une sous requête peut être aussi utilisée dans l'expression d'une jointure.
- L'évaluation de la sous requête dépend de la façon dont on structure la requête qui la contient.
- Les exemples suivants permettent de donner une idée sur quelques possibilités.

Q25: **Quels sont les noms, Salaires et fonctions des employés travaillant à 'Alger' et ayant la même fonction que 'Rachid'?**

```
SELECT  Nom_Emp , Fonction , Salaire
FROM    EMPLOYES , DEPARTEMENTS
WHERE   Ville  = 'Alger'
AND     EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND     Fonction IN
            ( SELECT    Fonction
              FROM      EMPLOYES
              WHERE     Nom_Emp  = 'Rachid' )
```

- Dans cette jointure, la sous requête est exécutée en premier et fournit comme résultat un ensemble de valeurs correspondant aux fonction des employés ayant pour nom 'Rachid'.

- Si on est sûr que deux employés ne peuvent pas avoir le même nom **alors le IN peut être remplacé par =.**
- Le résultat est utilisé pour constituer la partie de la condition de jointure :

  Fonction IN <Ensemble des valeurs retournées par la sous requête>.

- D'après l'exemple , l'ensemble retourné par la sous requête étant égal à 'Vendeur', la condition de jointure serait équivalente à :

```
Ville  = 'Alger'
AND EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND Fonction IN ('Vendeur' )
```

➢ Nous allons voir un autre exemple où l'évaluation de la sous requête ne se fait pas en une seule fois indépendamment de la requête principale mais est répétée plusieurs fois en fonction de certaines données provenant de la requête principale.

Q26: **Quels sont les noms et Salaires des employés qui ont un salaire supérieur à la moyenne des salaires de leur départements?**

❑ L'obtention d'une réponse à cette question va nécessiter les étapes suivantes :

- Parcourir la table EMPLOYES de façon à connaître le numéro **d** du département de l'employé et son salaire **s**
- Calculer la moyenne **md** des salaires du département **d** (dans une sous requête)
- Tester si le salaire **s est > md** et si oui inclure l'employé dans le résultat

- Une requête permettant de répondre à cette question serait :

```
SELECT  Nom_Emp , Salaire
FROM    EMPLOYES , X
WHERE   Salaire >
                ( SELECT        AVG(Salaire)
                  FROM          EMPLOYES
                  WHERE         X•Num_Detp  = Num_Dept )
```

- On remarque l'utilisation d'un nom **synonyme X** pour la table EMPLOYES dans la clause FROM de la requête principale.

- **X** peut être vu comme une variable qui parcourt les tuples de la relation EMPLOYES apparaissant dans la requête principale.

- La clause WHERE X•Num_Detp = Num_Dept de la sous requête lui permet en quelque sorte de **se synchroniser avec la requête principale**.

- Ainsi pour chaque tuple repéré par la variable X , la sous requête va calculer la moyenne (fonction AVG : Average) des salaires du département ayant le même numéro que celui du **tuple repéré par X** et qui correspond à un employé bien sûr.

- Le fait que la sous requête référence la même table EMPLOYES dans sa clause FROM ne gêne en aucun cas la requête principale.

❖ Autres possibilités d'interrogation de SQL

➢ En logique on utilise généralement les deux quantificateurs □ ∃ (il existe) et □ ∀ (quelque soit) qui s'appliquent à des propositions selon les deux formes :

□ ∃ x P(x)

et ∀ x P(x) .

- Dans le premier cas la P(x) est vraie s'il existe au moins une valeur de x qui la rend vraie

- et dans le deuxième cas P(x) est vraie si et seulement si elle est vraie pour toutes les valeurs de x .

➢ **SQL** offre un prédicat qui s'appelle **EXISTS** équivalent du quantificateur existentiel ∃.

- Il permet de tester si l'ensemble de valeurs (tuples) retourné par une requête (ou sous requête) est vide ou non.

- Ce test est évalué à vrai (TRUE) si l'ensemble contient au moins une ligne et à faux (FALSE) si l'ensemble retourné est vide.

☝ **SQL n'offre pas** de prédicat équivalent du quantificateur ∀.

- Cependant, ce quantificateur **peut être traduit** en utilisant l'opérateur EXISTS.
- En effet, Il suffit de remarquer l'équivalence entre les deux formules suivantes :

$$\forall x\, P(x) \equiv \neg \exists x \neg P(x)$$

- L'utilisation du quantificateur ∀ est très rare dans l'expression des requêtes.

- Il **sert surtout** pour **exprimer** l'opération algébrique de **DIVISION** dont la définition formelle utilise justement ce quantificateur.

❑ Expression de la différence avec EXISTS

➢ La différence entre deux relations peut s'exprimer grâce a l'opérateur **MINUS** dont l'utilisation est semblable a celle de l'union entre deux relations.

➢ Il est aussi possible **d'utiliser le prédicat EXISTS** pour réaliser cette opération.

Q27: **Quels sont les numéros des départements dans lesquels travaillent des 'VENDEURS' mais pas des 'INGENIEURS' ?**

```
SELECT    Num_Dept
FROM      DEPARTEMENT
WHERE     Fonction  ='VENDEUR'
AND       NOT EXISTS
              ( SELECT    Num_Dept
                FROM      DEPARTEMENT
                WHERE     Fonction  ='INGENIEUR' )
```


❖ On sait que la division d'une relation R de schéma R(A,B,C) par une relation S de

schéma S(C) est une relation Q de schéma Q(A, B) et telle que :

Q = { (a, b) ∈ à R / ∀ (c) ∈ à S , ∃ (a,b,c) ∈ à R) }

- en utilisant l'équivalence des formules : **∀ x P(x) ≡ ⌉∃ x ⌉P(x)**

on peut traduire la division comme suit :

(a,b) ∈ Q si et seulement si **il n'existe pas** de **c ∈ à S**
tel qu'il n'existe pas de (a,b,c) ∈ R.

Ainsi, la division va se traduire en SQL par l'utilisation de deux opérateurs NOT EXITS successifs.

- Nous allons illustrer la **mise en ouvre de la division** à l'aide des deux relations suivantes :

  PROFESSEURS(Num_Prof, Code_Mod)

  MODULES(Code_Mod , Spécialité)

- Pour répondre à la question suivante :

Q28: **Quels sont les numéros des enseignants qui dispensent TOUS les modules de la spécialité 'INFORMATIQUE'?**


```
SELECT    DISTINCT (Num_Prof)
FROM      PROFESSEURS X
WHERE     NOT EXISTS
           ( SELECT    *
           FROM        MODULES
           WHERE       MODULES.Spécialité ='INFORMATIQUE'
           AND         NOT EXISTS
                        ( SELECT   *
                        FROM       PROFESSEURS
                        WHERE      MODULES•Code_Mod=PROFESSEURS•Code_Mod
                        AND        PROFESSEURS•Num_Prof = X•Num_Prof ) )
```