1. Introduction

Le langage SQL (Structured Query Language) est issu des travaux menés par IBM autour du langage prototype SEQUEL (Semi Query Language) pour le SGBD Système R. Aujourd'hui SQL est devenu un standard (normalisé par l'ANSI depuis 1986) disponible sur presque tous les SGBD relationnels (DB2, ORACLE, INFORMIX,...). SQL n'est pas simplement un langage de requête ou d'interrogation d'une base de données relationnelle. Il supporte aussi bien les fonctions de description de données que celles de manipulation de données. La liste suivante permet de donner une idée sur les opérations qui peuvent être réalisées avec SQL

• Créer le schéma de la base(Créer , supprimer une table ou un index, etc.)
• Modifier le schéma de la base (ajouter une nouvelle table, modifier le format d'une colonne, etc.)
• Interroger la base de données
• Définir des vues sur la base de données
• Spécifier les droits d'accès d'un utilisateur vis à vis des objets de la base (relations et vues)
• Mettre en place un mécanisme de contrôle d'intégrité des données
• Invoquer les commandes à partir d'un langage de programmation (Embedded SQL)

2. Requête en SQL

2.1 Structure d'une requête

Une requête d'interrogation en SQL est composée de trois clauses : SELECT , FROM et WHERE

SELECT A1 , A2 , ..., An
FROM R1 , R2 , ..., Rm
WHERE C

Où chaque Ai représente un attribut, chaque Ri représente une relation et C représente une condition ou une qualification. Une telle requête est équivalente à l'expression de l'algèbre relationnelle suivante :

Π A1 , A2 , ..., An (σC (R1 x R2 x .....x Rm))

Afin de bien illustrer les possibilités de SQL en matière d'interrogation d'une base de données relationnelle, nous allons nous servir du schéma relationnel suivant :

EMPLOYES (Num_Emp, Nom_Emp, Fonction, Salaire, Prime, Num_Resp , Num_Dept)
DEPARTEMENTS (Num_Dept, Nom_Dept , Ville)
et nous considérerons les extensions suivantes :

Num_Dept	Nom_Dept	Ville
10	Comptabilité	Khenchela
20	Recherches	Khenchela
30	Ventes	Alger
40	Fabrication	Oran

Relation DEPARTEMENTS

Num_Emp	Nom_Emp	Fonction	Num-Resp	Salaire	Prime	Num_Dept
1	Ali	Ingénieur	13	800		20
2	Rachid	Vendeur	6	1600	300	30
3	Kamel	Vendeur	6	1250	500	30
4	Omar	Directeur	9	2975		20
5	Brahim	Vendeur	6	1250	1400	30
6	Farouk	Directeur	9	2850		30
7	Lyes	Directeur	9	2450		10
8	Tahar	Analyste	4	3000		20
9	Malik	Président		5000		10
10	Louisa	Vendeur	6	1500	0	30
11	Fatma	Ingénieur	8	1100		20
12	Nadia	Ingénieur	6	950		30
13	Ahmed	Analyste	4	3000		20
14	Kader	Ingénieur	7	1300		10

Relation EMPLOYES

3. Expression de la projection avec SQL

Une projection consiste à extraire des colonnes (attributs) spécifiques d'une relation puis à éliminer les tuples en double pouvant apparaître dans la relation résultat. Cette opération s'exprime a l'aide de SQL par la clause :

SELECT liste d'attributs
FROM nom de relation

En pratique, SQL n'élimine pas les tuples en double car ces tuples ne gênent pas l'utilisateur et leur élimination implicite par SQL peut entraîner une perte de temps. Cependant SQL offre à l'utilisateur la possibilité de demander une élimination explicite de ces doubles grâce a un mot clé du langage qui est **: DISTINCT (ou parfois UNIQUE).**

Q1: Donner les noms de tous les employés et le salaire de chacun d'eux ?

```
SELECT Nom_Emp , Salaire
FROM EMPLOYES
```

Q2: Liste de tous les Départements dans lesquels travaille au moins un employé?

```
SELECT  DISTINCT Num_Dept
FROM EMPLOYES
```

On a rajouté le mot clé DISTINCT car le même numéro de Département peut apparaître plusieurs fois dans le résultat du fait que plusieurs Employés différents peuvent travailler dans un même département.

Q3: Lister tous les tuples de la relation DEPARTEMENTS ?

```
SELECT Num_Dept , Nom_Dept , Ville
FROM DEPARTEMENTS
```

On voit que la liste des attributs après la clause SELECT inclut tous les attributs de la relation DEPARTEMENTS. Pour simplifier l'écriture d'une telle requête, SQL permet de les formuler comme
suit :

```
SELECT *
FROM DEPARTEMENTS
```

ou le SELECT * signifie : " Sélectionner tous les attributs de la relation DEPARTEMENTS "

4. Expression de la sélection avec SQL

L'opération algébrique de sélection s'exprime dans SQL par un bloc du type :

```
SELECT liste d'attributs
FROM nom de relation
WHERE condition
```

Q4: Quels sont les employés travaillant dans le département numéro 10

```
SELECT *
FROM EMPLOYES
WHERE Num_Dept = 10
```

Avec cette requête, on aura comme résultat une relation ayant le même schéma que la relation EMPLOYES (i.e. mêmes attributs) et contenant tous les tuples de la relation EMPLOYES pour lesquels l'attribut Num_Dept a pour valeur 10.

Q5: Quels sont les numéros et les noms des employés travaillant dans le département numéro 10 ?

```
SELECT Num_Emp, Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10
```

On peut éventuellement ordonner le résultat d'une requête grâce à la clause ORDER BY. Pour cela, il suffit de spécifier dans cette clause le ou les attributs selon lesquels on désire ordonner le résultat ainsi que le critère ascendant ou descendant. La condition de la clause WHERE peut utiliser les opérateurs tels que: = , > , ≥ , < , ≤ , != , AND , OR , NOT , BETWEEN , LIKE , IN , ANY, ALL, EXIST, ... }

BETWEEN : Test d'appartenance à un intervalle
IN : Test d'appartenance d'une valeur à un ensemble
LIKE : Test de ressemblance de chaînes de caractères
ANY : Comparaison d'une valeur à une valeur quelconque d'un ensemble
ALL : Comparaison d'une valeur à toutes les valeurs d'un ensemble
EXIST : Test d'existence d'un tuple dans une relation (Quantificateur existentiel)

Q6: Quels sont les numéros et les noms des employés du département numéro 10 et qui ont un salaire supérieur à 1000 ?

```
SELECT Num_Emp, Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10  AND Salaire > 1000
```

Q7: Quels sont les fonctions exercées par les employés des départements 10 et 20 en éliminant les tuples en double du résultat?

(a) : avec **IN**

```
SELECT  DISTINCT   Fonction
FROM               EMPLOYES
WHERE              Num_Dept  IN (10 , 20 )
```

(b): avec **OR**

```
SELECT DISTINCT  Fonction
FROM             EMPLOYES
WHERE            Num_Dept  = 10 OR Num_Dept  = 20
```

Q8: Quels sont les fonctions dont le salaire est compris entre 2000 et 3000?

(a) : avec **BETWEEN**

```
SELECT  DISTINCT   Fonction
FROM               EMPLOYES
WHERE              Salaire
BETWEEN            2000 AND 3000
```

(b): avec **AND**

```
SELECT DISTINCT  Fonction
FROM             EMPLOYES
WHERE            Salaire >= 2000
AND              Salaire < = 3000
```

Q9: Quels sont les employés du département numéro 10 dont le nom commence par la lettre 'B' ?

```
SELECT Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10  AND Nom_Emp LIKE  'B%'
```

Le symbole % représente n'importe quelle chaîne de caractères. Il existe d'autres possibilités de comparaisons avec l'opérateur LIKE. Par exemple, pour rechercher les employés dont le nom se termine par la lettre R , il suffit d'utiliser '%R' comme argument de LIKE.

Q10: Quels sont les employés du département numéro 10 dont la fonction n'est ni 'Ingénieur' ni 'Analyste' ?

```
SELECT Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10
AND  NOT (Fonction = 'Ingénieur' OR Fonction = 'Analyste' )
```

5. Expression de l'union avec SQL

L'opération d'union algébrique s'exprime dans SQL par un bloc du type :

```
SELECT liste d'attributs
FROM nom de relation
WHERE condition
UNION
SELECT liste d'attributs
FROM nom de relation
WHERE condition
```

Q11: Quels sont les fonctions exercées par les employés des départements 10 et 20 en éliminant les tuples en double du résultat? (même que Q7)

(a) : Pas de tuples en double

```
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 10
UNION
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 20
```

(b) : Avec tuples en double

```
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 10
UNION ALL
SELECT    Fonction
FROM      EMPLOYES
WHERE     Num_Dept = 20
```

Par défaut, l'opération d'UNION (requête (a)) élimine les tuples en double du résultat. Si on veut les garder, on doit utiliser UNION ALL (requête (b)) à la place de UNION tout court comme ci-dessus. Cette question étant la même que Q7, nous avons vu que dans la requête correspondant à Q7,on avait explicitement demandé l'élimination des tuples en double grâce au mot clé DISTINCT.

6. Expression du produit cartésien avec SQL

Le produit cartésien entre deux relations est un cas particulier de jointure ou la condition de jointure est absente Par exemple le produit cartésien des relations EMPLOYES et DEPARTEMENTS s'obtient à l'aide de la requête suivante :

```
SELECT *
FROM EMPLOYES , DEPARTEMENTS
```

On remarque l'absence de la clause WHERE puisque comme il a été précisé on n'a pas besoin de spécifier une condition pour avoir le produit cartésien.
Il est aussi tout à fait possible de préfixer dans une requête un attribut apparaissant dans une clause SELECT ou une clause WHERE avec le nom de sa relation comme dans la requête suivante (même que Q6) :

```
SELECT EMPLOYES.Num_Emp, EMPLOYES.Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10  AND Salaire > 1000
```

7. Expression de l'intersection avec SQL

On sait que cette opération a comme arguments deux relations et donne comme résultat une relation contenant l'ensemble des tuples qui appartiennent en même temps à la première relation et à la deuxième. Un exemple de question illustrant cette opération est :
Q12: Quels sont les numéros des départements dans lesquels travaillent des VENDEURS et des 'INGENIEURS'.

```
SELECT Num_Dept
FROM EMPOYES
WHERE Fonction  ='VENDEUR'
INTERSECT
SELECT Num_Dept
FROM EMPOYES
WHERE Fonction  ='INGENIEUR'
```

8. Expression de la différence avec SQL

On sait que cette opération a comme arguments deux relations et donne comme résultat une relation contenant l'ensemble des tuples appartenant à la première relation mais pas à la deuxième. Un exemple de question illustrant cette opération est:
Q13: Quels sont les numéros des départements dans lesquels travaillent des 'VENDEURS' mais pas des 'INGENIEURS' ?

```
SELECT Num_Dept
FROM EMPOYES
WHERE Fonction  ='VENDEUR'
MINUS
SELECT Num_Dept
FROM EMPOYES
WHERE Fonction  ='INGENIEUR'
```

9. Expression des jointures avec SQL

9.1 Jointure avec qualification

SQL n'offre pas une opération spécifique pour exprimer la jointure avec qualification ou avec condition. Comme cette opération algébrique est une opération complémentaire, elle peut s'exprimer en fonction des opérations de algébriques de base à savoir : la sélection et le produit cartésien. On sait que :

JOIN C (R,S) = Select C (R x S) ou C représente la condition ou qualification de jointure faisant intervenir des opérateurs de comparaison.

SQL n'offre pas une opération spécifique pour exprimer la jointure. Il faudra donc l'exprimer comme une sélection selon une condition sur le produit cartésien des relations R et S .

Q14: Quels sont les numéros et les noms des employés qui travaillent à 'KHENCHELA' ?

```
SELECT Num_Emp , Nom_Emp
FROM EMPLOYES , DEPARTEMENTS
WHERE EMPLOYES•Num_Dept=DEPARTEMENTS•Num_Dept
AND Ville = 'KHENCHELA'
```

Q15: Quels sont les noms ,les fonctions et les salaires des employés du département 'RECHERCHES' dont le salaire est supérieur à 1000 ?

```
SELECT Nom_Emp , Fonction , Salaire
FROM EMPLOYES , DEPARTEMENTS
WHERE EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND Nom_Dept = 'RECHERCHE'
AND Salaire > 1000
```

Q16: Quels sont les noms et les fonctions des employés travaillant à 'KHENCHELA' et ayant la même fonction que l'employé 'AHMED' ?

Cette jointure peut s'exprimer sous forme de blocs SELECT imbriqués.

```
SELECT Nom_Emp , Fonction
FROM EMPLOYES , DEPARTEMENTS
WHERE Ville = 'KHENCHELA'
AND EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
AND Fonction IN
( SELECT Fonction
  FROM EMPLOYES
  WHERE Nom_Emp = 'AHMED' )
```

9.2 Jointure Naturelle

SQL n'offre pas une opération spécifique pour exprimer la jointure naturelle. Il faudra là aussi traduire en utilisant l'expression de la jointure naturelle en fonction des opérations algébriques de sélection , projection et produit cartésien et qui est :JOIN (R,S) = ΠV (σ C (R x S))

L'expression de la jointure naturelle entre les relations EMPLOYES et DEPARTEMENTS qui ont un seul attribut peut s'écrire : ΠV (σ C (EMPLOYES X DEPARTEMENTS)) où :

V = (Num_Emp,Nom_Emp,Salaire, Fonction, Prime, Num_Dept,Nom_Dept,Ville) et la condition de sélection C est : EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept d'où la requête SQL équivalente :

```
SELECT Num_Emp,Nom_Emp,Salaire, Fonction, Prime, Num_Dept,Nom_Dept,Ville
FROM EMPLOYES , DEPARTEMENTS
WHERE EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
```

9.3 Equi-Jointure

Cette opération n'existe pas aussi dans SQL. Cependant, il est possible de l'exprimer de la même manière que la jointure avec qualification puisque la seule différence est que dans l'équi-jointure l'opérateur de comparaison θ utilisé dans la condition de jointure est l'opérateur d'égalité (=).

9.4 Jointure d'une relation avec elle même (Auto-Jointure)

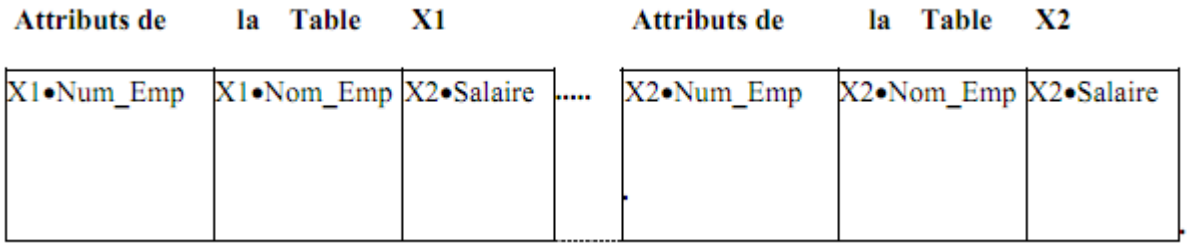
Pour répondre à certaines questions, on peut parfois être amené à faire la jointure d'une table avec elle-même. Le problème qui se pose alors est comment distinguer les attributs. La solution qui est largement utilisée est de désigner dans la clause FROM de la requête SQL la relation (ou table) par deux noms différents appelés synonymes.

Avec SQL, l'association d'un nom synonyme à une table consiste à faire suivre le nom de la table par le nom synonyme qui n'est autre qu'un identificateur au sens informatique du terme.

Q17: Quels sont les noms et les fonctions des employés ayant un salaire supérieur à celui de l'employé dont le nom est 'ALI'

```
SELECT X1.Nom_Emp , X1.Fonction
FROM EMPLOYES  X1 , EMPLOYES X2
WHERE X1.Salaire >  X2.Salaire
AND X2.Nomp_Emp = 'ALI'
```

X1 et X2 sont les noms synonymes désignant la table EMPLOYES. Cette requête peut être vue comme une requête ayant pour arguments dans la clause FROM deux tables X1 et X2 ayant le mêmes attributs et la même extension que EMPLOYES. L'évaluation de la requête va consister à calculer le produit cartésien entre deux les tables X1 et X2, puis de sélectionner les tuples du résultat pour lesquels la colonne X1.Salaire > X2.Salaire et la colonne X2.Nomp_Emp = 'ALI' . Enfin, on fait une projection sur les colonnes X1.Nom_Emp et X1.Fonction qui sera alors la relation répondant à cette question. Ainsi, le schéma de la relation résultat du produit cartésien X1 x X2 sera :



Après sélection des lignes ou tuples satisfaisant la condition :

X1•Salaire > X2•Salaire AND X2•Nomp_Emp = 'ALI' , la projection permet de retenir les colonnes X1•Nom_Emp et X1•Fonction

Un autre exemple d'auto-jointure est :

Q18: Quels sont les noms des employés ayant le même responsable ?

```
SELECT EMP1•Nom_Emp , EMP2•Nom_Emp
FROM EMPLOYES  EMP1 , EMPLOYES EMP2
WHERE EMP1•Num_Resp =  EMP2•Num_Resp
```

Dans cette requête EMP1 et EMP2 sont les noms synonymes de la table EMPLOYES. Les étapes d'évaluation de la requête sont les mêmes que celles de Q14. Cependant, il faut remarquer ici que dans le résultat on peut avoir des tuples en double et des tuples symétriques traduisant les situations qui suivent :

Dans notre exemple, les employés Tahar et Ahmed ont le même responsable (Num_Resp = 4 qui correspond à l'employé Omar). Dans le résultat de notre requête, on aura une relation ayant deux attributs et contenant parmi ses tuples les suivants :

EMP1•Nom_Emp	EMP2•Nom_Emp
Tahar	Tahar
Tahar	Ahmed
Ahmed	Ahmed
Ahmed	Tahar

Les tuples 2 et 4 traduisent la même information : Tahar et Ahmed ont le même responsable. Il faudrait donc ne garder qu'un seul de ces deux tuples. Les tuples 1 et 3 traduisent le fait que tout employé a le même responsable que lui-même. Ils n'apportent aucune information et doivent être éliminés du résultat.

Pour traiter ces cas il suffit de rajouter dans la clause WHERE une condition supplémentaire :
EMP1•Nom_Emp < EMP2•Nom_Emp (ou >)

```
SELECT EMP1•Nom_Emp , EMP2•Nom_Emp
FROM EMPLOYES  EMP1 , EMPLOYES EMP2
WHERE EMP1•Num_Resp =  EMP2•Num_Resp
AND EMP1•Nom_Emp <  EMP2•Nom_Emp
```

Essayons de voir l'effet de cette condition supplémentaire sur le résultat final. L'application de cette condition a lieu au moment de la sélection des lignes à partir de la relation résultant du produit cartésien entre EMP1 et EMP2. Voyons quelles sont parmi les lignes qui contenaient dans les colonnes EMP1.Nomp_Emp et EMP2.Nom_Emp les valeurs Tahar et Ahmed celles qui seront retenues par l'opération de sélection :

- Tahar n'est pas < à Tahar : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Tahar et dans la colonne EMP2.Nom_Emp la valeur Tahar aussi ne figurera pas dans la relation résultat de la sélection
- Tahar n'est pas < à Ahmed : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Tahar et dans la colonne EMP2.Nom_Emp la valeur Ahmed ne figurera pas dans la relation résultat de la sélection.
- Ahmed n'est pas < à Ahmed : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Ahmed et dans la colonne EMP2.Nom_Emp la valeur Ahmed aussi ne figurera pas dans la relation résultat de la sélection.
- Ahmed est < à Tahar : donc la ligne qui contient dans la colonne EMP1.Nom_Emp la valeur Ahmed et dans la colonne EMP2.Nom_Emp la valeur Tahar figurera dans la relation résultat de la sélection

Donc après projection on aura uniquement le tuple (Ahmed,Tahar) dans le résultat final et correspondant au fait que Tahar et Ahmed ont le même responsable. Il en sera de même pour les autres employés ayant le même responsable.

10. 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 seule valeur (une seule ligne), ou un ensemble de valeurs.
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.

10.1 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'.
L'opérateur de comparaison utilisé est celui de l'égalité (=) parce qu'on est sûr d'avance que le résultat retourné par la sous requête comporte une seule valeur.
On aurait pu répondre à cette question en utilisant les deux requêtes séparées suivantes :

requête (1)

```
SELECT   Fonction
FROM     EMPLOYES
WHERE    Nomp_Emp  ='OMAR'
```

requête (2)

```
SELECT   Nom_Emp
FROM     EMPLOYES
WHERE    Fonction = 'DIRECTEUR'
```

La requête (1) va retourner : 'DIRECTEUR' . On l'utilise alors pour exprimer la requête (2).

10.2 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 )
```

10.3 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 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.

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' )
```

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' )
```

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 vue 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 qui sera expliquée dans la suite) 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.

12. Les fonctions de groupes

12.1 aperçu général

SQL offre aussi un certain nombre de fonctions dites fonctions de groupes qui peuvent être utilisées dans l'expression d'une requête. les plus importantes sont :

AVG : permet de calculer une MOYENNE
SUM : permet de calculer une SOMME
COUNT : permet de compter des tuples
MAX : permet de calculer un maximum
MIN : permet de calculer un minimum
GROUP BY : permet de créer des sous-ensembles de tuples
HAVING : permet de tester si une condition est vérifiée par un groupe de tuples

Q29: Quels sont les numéros et salaires des employés dont le salaire est supérieur à 10% de la moyenne des salaires de son département ?

```
SELECT Nom_Emp , Salaire
FROM EMPLOYES , X
WHER Salaire >
( SELECT AVG(Salaire) * 0.10
FROM EMPLOYES
WHERE X•Num_Detp  = Num_Dept )
```

Q30: Quel est le nom , la fonction et le salaire de(s) l'employé(s) ayant le salaire le plus élevé

```
SELECT Nom_Emp , Fonction , Salaire
FROM EMPLOYES
WHER Salaire =
( SELECT MAX(Salaire)
FROM EMPLOYES)
```

La sous requête est évalué en premier et permet de calculer le salaire maximum des employés. La requête principale va parcourir les lignes de la table EMPLOYES et comparer le salaire de chaque employé avec le résultat de la sous requête. S'il y a égalité, le nom , la fonction et le salaire de cet employé figurera dans le résultat final.

Q31: Quel est le salaire maximum , le salaire minimum et la différence entre ces deux valeurs ?

```
SELECT MAX(Salaire) , MIN(Salaire) , MAX(Salaire) - MIN(Salaire)
FROM EMPLOYES
```

Q32: Quel est le nombre d'employés percevant une prime ?

```
SELECT COUNT(Prime)
FROM EMPLOYES
```

Q33: Quel est le nombre de fonctions différentes exercées dans le département 30 ?

```
SELECT COUNT(DISTINCT  Fonction)
FROM EMPLOYES
WHERE Num_Dept  = 30
```

Le mot DISTINCT clé permet de ne compter une même valeur qu'une seule fois. Le nombre retourné sera bien le nombre de valeurs différentes de l'attribut Fonction.

12.2 Manipulation de groupes : la clause GROUP BY

La clause GROUP BY permet de partitionner les tuples d'une relation de façon à former des groupes de tuples. Chaque groupe de tuples est caractérisé par le fait que les tuples qu'il contient possèdent les mêmes caractéristiques.

La manipulation des groupes implique donc que les requêtes peuvent comporter des clauses WHERE , GROUP BY, des fonctions de groupes (AVG, SUM, MAX, MIN) et HAVING. La clause WHERE est utilisée pour sélectionner dans une table les lignes (tuples) qui satisfont une condition.

Elle ne s'applique donc pas aux groupes. La clause HAVING est utilisée pour sélectionner dans une table les groupes de lignes (tuples) qui satisfont une condition. Elle ne s'applique donc pas aux lignes et s'utilise avec une clause GROUP BY.

Dans une requête manipulant des fonctions de groupe, il existe un ordre logique d'exécution de la requête et qui est :
1- La clause WHERE est appliquée en premier pour qualifier les lignes
2- Les groupes de lignes sont formés (GROUP BY)
3- Les fonctions de groupes sont appliquées (AVG, MIN, MAX,..)
4- Enfin la clause HAVING est appliquée pour choisir les groupes répondant au critère spécifié dans cette clause.

Q34 : Quelle est la moyenne des salaires du département 10 ?

```
SELECT AVG (Salaire)
FROM EMPLOYES
WHERE Num_Dept  = 10
```

Si on veut connaître la moyenne des salaires des départements 20 et 30, il suffit d'utiliser la même requête en remplaçant dans la clause WHERE Num_Dept = 10 par Num_Dept = 20 puis par Num_Dept = 30 comme suit : moyenne des salaires du département 20 moyenne des salaires du département 30

```
SELECT AVG (Salaire)
FROM        EMPLOYES
WHERE Num_Dept  = 20
SELECT AVG (Salaire)
FROM EMPLOYES
WHERE Num_Dept  = 30
```

Au lieu d'utiliser 3 requêtes séparées, on peut obtenir la moyenne des salaires par département
avec une seule requête en utilisant une clause GROUP BY comme suit :

```
SELECT Num_Dept , AVG (Salaire)
FROM EMPLOYES
GROUP BY  Num_Dept
```

Q35: Quels est pour chaque département et chaque fonction le salaire annuel (12 mois) moyen par fonction et le nombre d'employés exerçant cette fonction?

```
SELECT Num_Dept, Fonction , Count (*) , AVG (Salaire) * 12
FROM EMPLOYES
GROUP BY Num_Dept , Fonction
```

Q36: même question que mais donner le nom du département au lieu du numéro ce qui va nécessiter de faire une jointure

```
SELECT Nom_Dept, Fonction , Count (*) , AVG (Salaire) * 12
FROM EMPLOYES , DEPARTEMENTS
WHERE EMPLOYES•Num_Dept = DEPARTEMENTS•Num_Dept
GROUP BY Num_Dept , Fonction
```

Q37: Quels est pour chaque département le salaire annuel (12 mois) moyen de tous ses employés sauf ceux ayant une fonction de 'DIRECTEUR' ou 'PRESIDENT'?

```
SELECT Num_Dept, AVG (Salaire) * 12
FROM EMPLOYES
WHERE Fonction NOT IN ('DIRECTEUR' , 'PRESIDENT')
GROUP BY Num_Dept
```

Q38: Quels sont les numéros des départements ayant plus de 10 employés ?

```
SELECT Num_Dept
FROM EMPLOYES
GROUP BY Num_Dept
HAVING COUNT(*) > 10
```

Q39: Quelles sont les différentes fonctions exercées dans l'ensemble des départements et la moyenne des salaires par fonction ?

```
SELECT Fonction , AVG (Salaire)
FROM EMPLOYES
GROUP BY Fonction
```

Q40: Quelles sont les différentes fonctions exercées dans l'ensemble des départements dont la moyenne des salaires par fonction est supérieure à 10.000 ?

```
SELECT Fonction , AVG (Salaire)
FROM EMPLOYES
GROUP BY Fonction
HAVING AVG (Salaire) > 10.000
```

Q41: Quels sont les numéros des départements au moins deux employés exerçant la fonction de 'Vendeur'?

```
SELECT Num_Dept
FROM EMPLOYES
WHERE Fonction = 'Vendeur'
GROUP BY Num_Dept
HAVING COUNT(*) > 2
```

Q42: Quels sont les numéros des départements dans lesquels la moyenne des primes est supérieur à 10% de la moyenne des salaires?

```
SELECT Num_Dept , AVG (Salaire) , AVG (Prime) , AVG (Salaire) * 0.10
FROM EMPLOYES
GROUP BY Num_Dept
HAVING AVG (Prime) >  AVG (Salaire) * 0.10
```

Q43: Liste des employés du département 10 par ordre décroissant de leur salaire?

```
SELECT Num_Dept , Nom_Emp, Salaire
FROM EMPLOYES
WHERE Num_Dept = 10
ORDER BY Salaire DESC
```

Q44: Liste par ordre alphabétique des employés du département 10?

```
SELECT Nom_Emp
FROM EMPLOYES
WHERE Num_Dept = 10
ORDER BY Nom_Emp
```

Il est aussi possible de trier le résultat retourné par une requête en fonction de plusieurs attributs.

Q45: Liste des employés triés selon un ordre alphabétique de leur fonction et a l'intérieur de chaque fonction les trier selon un salaire décroissant.

```
SELECT Nom_Emp, Fonction, Salaire
FROM EMPLOYES
ORDER BY Fonction ASC, Salaire DESC
```

Q46: Liste des employés n'ayant pas reçu de prime.

```
SELECT Nom_Emp, Salaire, Prime
FROM EMPLOYES
WHERE Prime IS NULL
```

13. La description des données avec SQL

SQL offre un certain nombre de commandes pour créer des tables et pour modifier leurs structures.

Les principales commandes sont :

CREATE TABLE : ajouter une table à la B.D.

ALTER TABLE : modifier une colonne ou ajouter une nouvelle colonne à une table

DROP TABLE: supprimer une table