

Corrigé Exercices sur SQL

Soit une base de données conçue par une agence immobilière lui permettant de gérer les appartements dont elle a la responsabilité :

Immeuble	NomImmeuble	Adresse	NBEtages	AnnéeConstruction	NomGerant
	Koudalou	3 rue Blanche	15	1975	Doug
	Barabas	2 Allée Nikos	2	1973	Ross

Appart	NomImmeuble	NoAppart	Superficie	Etage
	Koudalou	1	150	14
	Koudalou	34	50	15
	Koudalou	51	200	2
	Koudalou	52	50	5
	Barabas	1	250	1
	Barabas	2	250	2

Personne	Nom	Age	Profession
	Ross	51	Informaticien
	Alice	34	Cadre
	Rachel	23	Stagiaire
	William	52	Acteur
	Doug	34	Rentier

Occupant	NomImmeuble	NoAppart	NomOccupant	AnneeArrivée
	Koudalou	1	Rachel	2012
	Barabas	1	Doug	2014
	Barabas	2	Ross	2014
	Koudalou	51	William	2016
	Koudalou	34	Alice	2013

Pour simplifier, nous faisons les hypothèses suivantes :

- Deux personnes ont forcément des noms différents
- Deux immeubles ont forcément des noms différents
- Deux appartements situés dans le même immeuble ont forcément des numéros (NoAppart) différents.
- Un appart ne peut être occupé que par une personne
- Une personne ne peut occuper qu'un seul appartement

Un enregistrement dans la table Immeuble, Appart ou Personne décrit un immeuble, un appartement et une personne. Un enregistrement dans la table Occupant dit que telle personne (NomOccupant) occupe tel appartement (NomImmeuble et NoAppart). Cette même personne occupe cet appartement depuis une certaine Année (Année d'arrivée).

1. Pour chacune des requêtes suivantes exprimées en SQL, si elle est correcte alors donner son résultat, sinon expliquer pourquoi elle n'est pas correcte. (Noter que vous pouvez tester directement sur la base de données Immo).

- a. `SELECT NomImmeuble FROM Appart`

Correcte. Son résultat est :

NomImmeuble
Koudalou
Koudalou
Koudalou
Koudalou
Barabas
Barabas

On remarque que les doublons ne sont pas éliminés. Il faut aussi noter que l'ordre d'affichage de ces noms d'immeubles est quelconque. Ce n'est pas parce qu'on insère d'abord les 4 apparts de Koudalou puis les 2 de Barabas que lors de l'évaluation de cette requête, on aura forcément 4 Koudalou puis 2 Barabas. Vous pouvez re-tester cette requête quand vous aurez créé la clé primaire de la table Appart (exos plus loin) et vous verrez que l'ordre d'affichage changera.

- b. `SELECT NomImmeuble, AnnéeArrivée FROM Occupant`

Correcte. Résultat :

NomImmeuble	AnneeArrivée
Koudalou	2012
Barabas	2014
Barabas	2014
Koudalou	2016
Koudalou	2013

- c. `SELECT DISTINCT Nom FROM Personne`

Nom
Ross
Alice
Rachel
William
Doug

Comme on a supposé que les personnes ont des noms différents, le DISTINCT ici n'a aucun effet.

- d. `SELECT * FROM Appart, Immeuble`

Appart. NomImmeuble	No Appart	Superficie	Etage	Immeuble. NomImmeuble	Adresse	Etages	Année Construction	Nom Gérant
Koudalou	1	150	14	Koudalou	3 rue Blanche	15	1975	Doug
Koudalou	34	50	15	Koudalou	3 rue Blanche	15	1975	Doug
Koudalou	51	200	2	Koudalou	3 rue Blanche	15	1975	Doug

Koudalou	52	50	5	Koudalou	3 rue Blanche	15	1975	Doug
Barabas	1	250	1	Koudalou	3 rue Blanche	15	1975	Doug
Barabas	2	250	2	Koudalou	3 rue Blanche	15	1975	Doug
Koudalou	1	150	14	Barabas	2 Allée Nikos	2	1973	Ross
Koudalou	34	50	15	Barabas	2 Allée Nikos	2	1973	Ross
Koudalou	51	200	2	Barabas	2 Allée Nikos	2	1973	Ross
Koudalou	52	50	5	Barabas	2 Allée Nikos	2	1973	Ross
Barabas	1	250	1	Barabas	2 Allée Nikos	2	1973	Ross
Barabas	2	250	2	Barabas	2 Allée Nikos	2	1973	Ross

Chaque Appart est associé à tous les immeubles. On observe que du fait que l'attribut NomImmeuble soit présent dans les 2 tables, quand on fait leur produit (sous SQL), les deux attributs sont présents dans le résultat. Pour les distinguer, leurs noms sont précédés du nom de la table d'où ils proviennent.

- e. `SELECT NomImmeuble FROM Immeuble MINUS SELECT NomImmeuble FROM Appart` (remplacer MINUS par EXCEPT si votre SGBD l'utilise)

Correcte : Le résultat est

NomImmeuble

Le résultat ne contient donc aucun enregistrement. Le sens de cette requête est : afficher les noms des immeubles qui sont décrits dans la table Immeuble mais qui n'ont aucun appart décrit dans Appart. L'inverse (Appart moins Immeuble) donne aussi le même résultat car il n'y a aucun appartement qui se trouve dans un immeuble non décrit dans la table Immeuble.

- f. `SELECT NomImmeuble FROM Personne`

Erreur : il n'y a pas d'attribut NomImmeuble dans la table Personne.

2. Exprimer chacune des requêtes suivantes en utilisant SQL (jusqu'à i, ce sont les mêmes requêtes que pour l'algèbre) :

- a. Afficher le nom du gérant de l'immeuble qui s'appelle Koudalou

```
SELECT    NomGerant
FROM      Immeuble
WHERE     NomImmeuble = 'Koudalou'
```

- b. Afficher la profession du gérant de Koudalou.

```
SELECT    Profession
FROM      Personne, Immeuble
WHERE     Nom = NomGerant
```

En utilisant l'opération INNER JOIN, cette requête sera écrite

```
SELECT Profession
FROM Personne INNER JOIN Immeuble ON Nom = NomGerant
```

On vous laisse le soin d'expliquer aux élèves l'intérêt de cette deuxième notation.

- c. Afficher la profession du gérant de l'immeuble où Rachel habite.

```
SELECT Profession
FROM Personne, Occupant O, Immeuble I
WHERE NomGerant=Nom AND
      O.NomImmeuble = I.NomImmeuble AND
      NomOccupant = 'Rachel'
```

Noter l'utilisation des variables/alias O et I pour désigner respectivement Occupant et Immeuble. On aurait pu s'en passer et écrire Occupant.NomImmeuble = Immeuble.NomImmeuble. C'est juste plus long. Noter aussi qu'on a écrit « Occupant O » non pas « Occupant AS O ». Les deux notations sont correctes (généralement).

- d. Afficher la superficie de l'appartement occupé par Rachel ainsi que la profession du gérant de l'immeuble où elle habite.

```
SELECT Superficie, Profession
FROM Personne, Occupant O, Immeuble I, Appart A
WHERE NomGerant=Nom AND
      O.NomImmeuble = I.NomImmeuble AND
      NomOccupant = 'Rachel' AND
      O.NomImmeuble=A.NomImmeuble AND
      O.NoAppart = A.NoAppart
```

On pourrait remplacer la condition « O.NomImmeuble=A.NomImmeuble » par « I.NomImmeuble=A.NomImmeuble » car on a de toute façon la condition « O.NomImmeuble=I.NomImmeuble ».

- e. Afficher le numéro et le nom d'immeuble des appartements occupés.

```
SELECT NomImmeuble, NoAppart
FROM Occupant
```

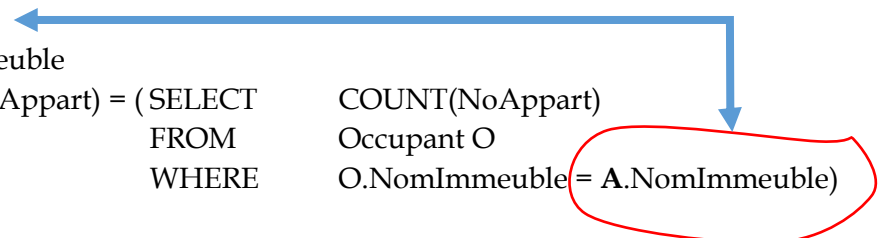
- f. Afficher le numéro et le nom d'immeuble des appartements inoccupés.

```
SELECT NomImmeuble, NoAppart
FROM Appart
EXCEPT
SELECT NomImmeuble, NoAppart
FROM Occupant
```

- g. Afficher le nom du ou des immeubles qui ont tous leurs appartements occupés (sur la base qui est donnée, c'est clair que c'est Barabas).

Principe : Pour chaque immeuble, on compte le nombre de ces appart. Si ce nombre est égal au nombre de ses apparts occupés, alors on affiche le nom de cet immeuble.

```
SELECT NomImmeuble
FROM Appart A
GROUP BY NomImmeuble
HAVING Count(NoAppart) = (SELECT COUNT(NoAppart)
                          FROM Occupant O
                          WHERE O.NomImmeuble = A.NomImmeuble)
```



On regroupe les apparts par nom d'immeuble. Pour chacun des groupes (donc pour chaque immeuble), on compte le nombre d'apparts. Si ce nombre (condition HAVING) est égal au nombre d'apparts occupés dans ce même immeuble (condition O.NomImmeuble=A.NomImmeuble), alors l'immeuble en question est affiché.

Le principe de cette requête SQL est quelque peu différent de celui que nous avons vu en algèbre. On peut cependant exprimer la requête vue en algèbre comme suit :

```

SELECT    NomImmeuble
FROM      Appart

```

} ← Tous les immeubles

```

EXCEPT

SELECT    NomImmeuble
FROM      (SELECT    NomImmeuble, NoAppart
FROM      Appart
EXCEPT
SELECT    NomImmeuble, NoAppart
FROM      Occupant) AS Libre(NomImmeuble, NoAppart)

```

← Immeubles où il y a au moins un appartement libre

} ← Apparts libres. Cf. question f.

La « nouveauté » avec cette requête est de constater que dans la clause FROM, on peut non seulement mentionner des tables, mais de plus on peut utiliser des requêtes. Pour être plus exact : le résultat d'une requête. On sait que la requête en bleu retourne une table. On a juste « nommé » le résultat par « Libre » et on a précisé le nom des attributs de cette table « éphémère ». Elle est éphémère car une fois la requête exécutée, on ne pourra pas réutiliser la table « Libre » comme une table quelconque.

Pour illustrer encore ce cas d'utilisation de requête dans FROM, supposons que l'on veuille afficher le nom de gérants d'immeubles où il y a des appartements libres. Une formulation possible de la requête est :

```

SELECT    NomGerant
FROM      Immeuble, (SELECT    NomImmeuble, NoAppart
FROM      Appart
EXCEPT
SELECT    NomImmeuble, NoAppart
FROM      Occupant) AS Libre(NomImmeuble, NoAppart)
WHERE     Immeuble.NomImmeuble = Libre.NomImmeuble

```

- h. Donner le nom des occupants qui sont arrivés après Alice (exprimer cette requête sans utiliser la constante 2013 qui est l'année d'arrivée d'Alice).

```

SELECT    NomOccupant
FROM      Occupant
WHERE     AnneeArrivee > (
SELECT    AnneeArrivee
FROM      Occupant
WHERE     NomOccupant = 'Alice')

```

- i. Donner les paires de noms d'occupants qui correspondent à des personnes habitant le même immeuble. La paire (Alice, William) en est un exemple car les deux habitent Koudalou. Remarque : essayer d'éviter de retourner des paires inutiles, ex : (Alice, Alice) est inutile car toute personne habite forcément le même immeuble qu'elle-même et (ii) ne pas retourner à la fois (Alice, William) et (William, Alice) car il s'agit en fait de la même paire. Indication : On peut

utiliser le comparateur < entre des chaînes de caractère. Dans ce cas, c'est l'ordre lexicographique qui est pris en compte. Par exemple, Alice < William mais pas l'inverse.

```
SELECT      O1.NomOccupant AS Occupant1, O2.NomOccupant AS Occupant2
FROM        Occupant O1, Occupant O2
WHERE       O1.NomImmeuble = O2.NomImmeuble AND
            O1.NomOccupant < O2.NomOccupant
```

- j. Afficher le nombre d'appartements

```
SELECT      COUNT(NoAppart)
FROM        Appart
```

Noter que la requête

```
SELECT      COUNT(NoAppart)
FROM        Appart
```

Est aussi correcte. Toutes les deux comptent le nombre de valeurs prises par chacun des attributs. En l'absence de valeurs NULL, dans les deux cas, ça revient à compter le nombre de lignes dans la table Appart. En général, quand on veut compter le nombre de lignes dans une table, on préférera la formulation

```
SELECT      COUNT(*)
FROM        Appart.
```

Supposons que l'on voulait compter le nombre de superficies distinctes. A ce moment, on n'a pas d'autre choix que d'utiliser l'attribut Superficie en le faisant précéder de DISTINCT.

```
SELECT      COUNT(DISTINCT Superficie)
FROM        Appart
```

Vu que maintenant on sait que dans FROM, on peut utiliser des requêtes, celle-ci-dessus peut être formulée par :

```
SELECT      COUNT(*)
FROM        (SELECT      DISTINCT Superficie
             FROM        Appart) AS LesSuperficies(Aire)
```

On compte le nombre de lignes qu'il y a dans la table éphémère LesSuperficies et qui est obtenue par une requête.

- k. Afficher la superficie moyenne des appartements

```
SELECT      AVG(Superficie)
FROM        Appart
```

C'est la même chose que de faire

```
SELECT      SUM(Superficie)/COUNT(*)
FROM        Appart
```

- l. Afficher les appartements (tous leurs attributs) qui ont une superficie inférieure à la moyenne.

```
SELECT      *
FROM        Appart
WHERE       Superficie > (SELECT      AVG(Superficie)
                         FROM        Appart)
```

- m. Afficher les appartements qui ont la superficie maximale

Même chose que la précédente, il suffit de remplacer > par = et AVG par MAX.

- n. Pour chaque immeuble, afficher son nom ainsi que le nombre de ses appartements.

```
SELECT    NomImmeuble, COUNT(*)
FROM      Appart
GROUP BY  NomImmeuble.
```

Remarque importante: Quand on a une requête avec GROUP BY, dans la clause SELECT et mis à part des fonctions d'agrégation, on ne peut mentionner que des attributs qu'on a utilisés dans GROUP BY. Par exemple,

```
SELECT    A.NomImmeuble, NBEtages, COUNT(*)
FROM      Appart A, Immeuble I
WHERE     A.NomImmeuble=I.NomImmeuble
GROUP BY  I.NomImmeuble.
```

Intuitivement, on cherche à afficher pour chaque immeuble, son nom, le nombre de ses étages et le nombre de ses appartements. Le nombre de ses étages est une info qui se trouve dans Immeuble. La formulation paraît logique et pourtant la requête ci-dessus contient 2 erreurs : (i) NBEtages n'est pas mentionné dans GROUP BY (ii) on a regroupé en fonction de I.NomImmeuble et on demande à afficher A.NomImmeuble. Même si WHERE impose l'égalité entre les deux, on doit utiliser I.NomImmeuble dans SELECT.

Autre subtilité, supposons que dans Immeuble on ait un troisième immeuble pour lequel il n'y a aucun appartement dans Appartement et supposons que pour ce cas, on veuille afficher la valeur 0 pour count pour bien signifier que l'immeuble existe et l'agence ne gère (plus ?) aucun de ses appart. Dans ce cas, la requête sera bien plus complexe.

- o. Pour chaque immeuble, afficher son nom ainsi que le nombre de ses occupants arrivés après 2013.

```
SELECT    NomImmeuble, COUNT(*)
FROM      Occupant
WHERE     AnneeArrivée > 2013
GROUP BY  NomImmeuble
```

- p. Pour chaque personne, afficher son nom, sa profession, la superficie de l'appartement qu'elle occupe ainsi que le nom du gérant de son immeuble.

```
SELECT    Nom, Profession, Superficie, NomGerant
FROM      Personne P, Occupant O, Immeuble I, Appart A
WHERE     P.Nom=O.NomOccupant AND
          O.NomImmeuble=I.NomImmeuble AND
          O.NoAppart = A.NoAppart AND
          A.NomImmeuble=I.NomImmeuble
```

Observer ici qu'aucun des attributs que l'on veut afficher ne se trouve dans la table Occupant. Pourtant nous avons besoin de cette table pour faire le lien entre une personne et l'appart qu'elle occupe pour obtenir la superficie et une personne et l'immeuble où elle habite pour obtenir le nom du gérant.

- q. Pour chaque immeuble ayant au moins 3 appartements afficher son nom ainsi que la superficie moyenne de ses appartements ayant au moins 100 m2.

```
SELECT    NomImmeuble, AVG(Superficie)
```

```

FROM      Appart
WHERE     Superficie >= 100 AND
          NomImmeuble IN (SELECT  NomImmeuble
                          FROM      Appart
                          GROUP BY  NomImmeuble
                          HAVING    COUNT(*) >=3)

GROUP BY  NomImmeuble

```

3. Pour chacune des opérations de mise à jours ci-dessous, l'exprimer en utilisant SQL

- a. Modifier l'âge de Doug : son véritable âge est de 67.
 UPDATE Personne SET Age=67 WHERE Nom='Doug'
- b. Ajouter une nouvelle personne qui s'appelle Bob et dont la profession est dentiste. Par contre, on ne connaît pas son âge.
 INSERT INTO Personne VALUES('Bob', NULL, 'Dentiste')
 Ici, on n'a pas précisé les attributs et donc c'est l'ordre par default qui est l'ordre respecté lors de la creation de la table Personne : d'abord Nom, puis Age et enfin Profession. Une autre manière de procéder est
 INSERT INTO Personne(Profession, Nom) VALUES('Dentiste', 'Bob').
- c. Ajouter un occupant (David) qui occupe un appartement dont le numéro est 25 et qui est situé dans un immeuble dont le nom est Glycines. Il serait arrivé en 2020. Constaté que maintenant, l'agence est censée gérer un locataire qui habite un appartement dont elle n'a pas connaissance (il ne figure pas dans la table Appart) situé dans un immeuble sur lequel elle n'a aucune information.
 INSERT INTO Occupant Values('Glycines',25,'David',2020)
- d. Ajouter une personne qui s'appelle Alice âgée de 22 et qui est étudiante. Constaté que maintenant qu'on a deux personnes qui s'appellent Alice. On ne sait plus qui des deux occupe l'appartement 34 de Koudalou.
 INSERT INTO Personne VALUES('Alice', 22, 'Etudiante')
- e. Supprimer l'immeuble Koudalou de la table Immeuble. Constaté que maintenant on n'a plus aucune information sur cet immeuble alors qu'il y a des locataires qui habitent des appartements qui s'y trouvent.
 DELETE FROM Immeuble WHERE NomImmeuble='Koudalou'
- f. Annuler les 5 modifications que vous venez d'effectuer (remodifier quand il y a eu modification, supprimer quand il y a eu insertion et insérer quand il y a eu suppression).
- g. Supprimer les appartements non occupés (un peu difficile je vous l'accorde).
 DELETE FROM Appart
 WHERE (NomImmeuble,NoAppart) IN (
 SELECT NomImmeuble, NoAppart
 FROM Appart
 EXCEPT
 SELECT NomImmeuble, NoAppart
 FROM Occupant)

On a gardé la couleur Bleu pour bien montrer qu'il s'agit d'une requête qu'on a rencontrée auparavant. L'idée est qu'un appartement est identifié par son numéro et le nom de l'immeuble où

il est situé. Si ce couple fait partie de l'ensemble des couples identifiant les appartements libres, alors l'appartement en question est supprimé.

Certains systèmes n'acceptent pas de condition IN qui porte simultanément sur 2 attributs. Une solution pour y remédier est de « concaténer » les deux attributs pour n'en former qu'un :

```
DELETE FROM Appart
WHERE CONCAT(NomImmeuble,'@',NoAppart) IN (
                SELECT      CONCAT(NomImmeuble,'@', NoAppart)
                FROM        Appart
                EXCEPT
                SELECT      CONCAT(NomImmeuble,'@', NoAppart)
                FROM        Occupant)
```

Par exemple, la paire ('Koudalou',34) sera transformée en 'Koudalou@34'.

4. Que doit-on faire pour empêcher qu'un utilisateur puisse insérer une nouvelle personne ayant un nom qui existe déjà dans la table personne.

Il faut que soit (i) l'attribut Nom soit déclaré comme clé primaire de la table Personne :

```
ALTER TABLE Personne ADD CONSTRAINT ma_contrainte_cle PRIMARY KEY(Nom)
```

Ou bien (ii), déclarer l'attribut Nom comme unique (donc clé secondaire)

```
ALTER TABLE Personne ADD CONSTRAINT ma_contrainte_unicite_nom CHECK UNIQUE(Nom)
```

5. Que doit-on faire pour empêcher que quelqu'un puisse supprimer un immeuble tant qu'on a des locataires qui y habitent ?

Il faut créer une contrainte de référence, autrement dit une clé étrangère dans la table Occupant pour que chaque occupant soit lié à un immeuble qui existe dans Immeuble :

```
ALTER TABLE Occupant ADD CONSTRAINT fk1 FOREIGN KEY(NomImmeuble) REFERENCES
Immeuble(NomImmeuble).
```

Si vous procédez de la sorte, la grande majorité de systèmes vont retourner un message d'erreur. En effet, généralement, une clé étrangère ne peut faire référence qu'à une clé primaire ou une clé secondaire.

Ainsi, avant d'ajouter la clé étrangère, il faut d'abord créer la clé primaire de la table Immeuble qui est NomImmeuble :

```
ALTER TABLE Immeuble ADD CONSTRAINT ma_contrainte_cle2 PRIMARY KEY(NomImmeuble)
```

Après cela, on peut créer la clé étrangère.

Noter que la présence de la clé étrangère va :

- Empêcher la suppression d'un immeuble tant qu'il a des occupants
- Empêcher d'insérer un occupant s'il n'habite pas un immeuble répertorié dans Immeuble
- Empêcher les modifications de nom d'immeuble tant qu'il y a des occupants associés à l'ancien nom.

Par exemple, une fois la clé étrangère créée, la commande :

```
UPDATE Immeuble SET NomImmeuble='Kadala' WHERE NomImmeuble='Koudalou'
```

Sera refusé car il y a des occupants qui sont associés à Koudalou et si le système accepte la modification, alors on va se retrouver avec des occupants qui habitent Koudalou mais on n'a aucune information sur un immeuble avec ce nom.

6. Que doit-on faire pour que si l'on supprime un immeuble (où il n'y a pas d'habitants) tous les appartements qui s'y trouvent soient eux aussi automatiquement supprimés ? (Indication : regarder à ON DELETE CASCADE).

```
ALTER TABLE Apart ADD CONSTRAINT fk2 FOREIGN KEY(NomImmeuble) REFERENCES  
Immeuble(NomImmeuble) ON DELETE CASCADE
```

Tout simplement.

Avec la commande

```
ALTER TABLE Apart ADD CONSTRAINT fk2 FOREIGN KEY(NomImmeuble) REFERENCES  
Immeuble(NomImmeuble) ON DELETE CASCADE, ON UPDATE CASCADE
```

Les modifications de NomImmeuble dans Immeuble sont répercutées dans la table Apart.