Corrigé du TD N°2 : Bases de Données (4ème année)

Exercice 1

1. ***CREATE TABLE Etudiant (NumEtud Number (6) PRIMARY KEY, NomEtud Varchar(20),***
   PrenEtud Varchar(15), AnneeUniv Number(4), Moyenne Number(5,2) ,
   CONSTRAINT nn_Etudiant_NomEtud NOT NULL (NomEtud)) ;
2. ***ALTER TABLE Etudiant***
   ADD CONSTRAINT ck_Etudiant_Moyenne CHECK (Moyenne BETWWEN 0 AND 20) ;
3. ***ALTER TABLE Etudiant***
   ADD (DateNaisEtud Date NULL) ;
4. ***ALTER TABLE Etudiant***
   DROP COLUMN DateNaisEtud ;

Exercice 2

1. ***INSERT INTO Etudiant VALUES (3618, 'Alami', 'Mohamed', 2020, 15.25) ;***
2. ***ALTER TABLE Etudiant***
   DISABLE CONSTRAINT nn_Etudiant_NomEtud ;
3. ***INSERT INTO Etudiant (NumEtud, PrenEtud, Moyenne) VALUES (4253, 'Said', 13.75) ;***
4. ***ALTER TABLE Etudiant***
   ENABLE CONSTRAINT nn_Etudiant_NomEtud ;

   Un message d'erreur indiquant que la contrainte ne peut être activée s'affichera, car il y a un enregistrement (une ligne) déjà insérée dans la table qui ne la vérifie pas. Pour la réactiver, il faudrait d'abord, supprimer l'enregistrement (4253, 'Said', 13.75), puis relancer la requête d'activation.
5. ***ALTER TABLE Etudiant***
   DROP CONSTRAINT nn_Etudiant_NomEtud ;
6. ***DROP TABLE Etudiant ;***

Exercice 3

Employes *(NumEmp, NomEmp, Fonction, Salaire, DateRecrut, NumSup#, CodeFil#)*
Filiale *(CodeFil, NomFil, DateCreatFil, Secteur, CapitalFil, VilleFil)*

1. ***CREATE TABLE EmpInovIndus AS***
   SELECT * FROM Employes
   WHERE CodeFil IN (SELECT CodeFil FROM Filiale WHERE NomFil = 'InovIndus');
2. ***SELECT CodeFil, NomFil FROM Filiale***
   WHERE CodeFil NOT IN (SELECT CodeFil FROM Employes) ;

3. ***SELECT * FROM Employes***
   WHERE CodeFil IS NULL ;

4. ***SELECT * FROM Employes***
   WHERE CodeFil IN (SELECT CodeFil FROM Filiale WHERE VilFiliale = 'Tanger') ;

   - Ici, on a mis "**IN**" et non pas "**=**", car on peut avoir plusieurs filiales dans la ville de *Tanger*, et la requête interne retournera alors, un ensemble de valeurs de *CodeFil* qui sera comparé avec une seule valeur du champ *CodeFil* de la requête externe.

   - Si l'on n'est pas sûr de l'orthographe de la ville de Tanger (majuscule, minuscule ou un mélange des deux), on convertit le contenu du champ *VilFiliale* en majuscule avant de le comparer avec la valeur 'TANGER' :

     SELECT * FROM Employes
     WHERE CodeFil IN (SELECT CodeFil FROM Filiale UPPER(VilFiliale) = 'TANGER') ;

5. ***SELECT Employes.NomEmp, Employes.Fonction AS Fonct_Employe, Emp1.NomEmp, Emp1.Fonction AS Fonct_Sup FROM Employes, Employes Emp1***
   WHERE Employes.NumSup=Emp1.NumEmp ;

   Explication : On fait une copie *Emp1* (dans la mémoire) de la table *Employes* et on réalise une jointure entre la table et sa copie. Ceci donne le produit cartésien des 2 tables. Ensuite, on garde uniquement les couples *(ligneEmployes, ligneEmp1)* qui vérifient : *ligneEmploye.NumSup= ligneEmp1.NumEmp.*

   AS Fonct_Employe, permet de renommer (seulement au moment de l'affichage) la colonne *Employes.Fonction* en *Fonct_Employe.*

   La requête précédente contient une jointure naturelle entre la table *Employes* et sa copie *Emp1*. Ainsi, elle affichera chaque employé avec son supérieur, sauf le PDG qui, lui, n'a pas de supérieur hiérarchique. Pour l'afficher, on peut fait une jointure externe à droite, mais le nom et la fonction de son supérieur resteront vides :

   SELECT Employes.NomEmp, Employes.Fonction AS Fonct_Employe, Emp1.NomEmp, Emp1.Fonction AS Fonct_Sup from Employes, Employes Emp1
   WHERE Employes.NumSup=Emp1.NumEmp (+) ;

   On peut aussi, trier, par ordre alphabétique des noms des employés, le résultat de la requête :

   SELECT Employes.NomEmp, Employes.Fonction AS Fonct_Employe, Emp1.NomEmp, Emp1.Fonction AS Fonct_Sup FROM Employes, Employes Emp1
   WHERE Employes.NumSup=Emp1.NumEmp (+)
   ORDER BY Employes.NomEmp ;

6. Cas d'un seul employé dont le nom = 'Alami' :

   SELECT NomEmp, Salaire FROM Employes
   WHERE Salaire > (SELECT Salaire FROM Employes WHERE UPPER(NomEmp) = 'ALAMI') ;

   Cas de plusieurs employés dont le nom = 'Alami' :

   SELECT NomEmp, Salaire FROM Employes
   WHERE Salaire > ALL (SELECT Salaire FROM Employes WHERE UPPER(NomEmp) = 'ALAMI') ;

7. ***SELECT Employes.NomEmp, Employes.Salaire FROM Employes, Employes EmpCopie***
   WHERE Employes.Salaire > EmpCopie.Salaire AND UPPER (EmpCopie.NomEmp) = 'ALAMI' ;
8. ***SELECT NomEmp FROM Employes Emp***
   WHERE CodeFil != (SELECT CodeFil FROM Employes WHERE NumEmp = Emp.NumSup) ;
9. ***SELECT NumEmp, NomEmp FROM Employes CopieEmp***
   WHERE EXISTS (SELECT * FROM Employes WHERE NumSup = CopieEmp.NumEmp) ;

   C'est une requête composée de deux requêtes dont l'exécution est synchronisée : La requête externe parcourt la table *CopieEmp* (copie de la table *Employes*) ligne par ligne. Pour chaque ligne parcourue, il y a exécution de la requête interne qui opère sur la table (originale) *Employes*. Une ligne parcourue dans *CopieEmp* est sélectionnée (affichée) si la requête interne retourne (sélectionne) au moins une ligne de la table *Employes* vérifiant la condition.

   Si l'on veut afficher les employés qui n'ont pas de subordonnés, on n'a qu'à remplacer *EXISTS* par *NOT EXISTS*.

10. ***SELECT NumEmp, NomEmp, Salaire FROM Employes***
    ORDER BY Salaire DESC, NomEmp ASC;

Exercice 4

1- ***SELECT CodeFil, Fonction FROM Employes***
GROUP BY CodeFil, Fonction ;

On peut trier le résultat par ordre croissant des *CodeFil*, et en cas d'égalité par rapport au champ *CodeFil*, on trie par ordre croissant de *Fonction* :

SELECT CodeFil, Fonction FROM Employes
GROUP BY CodeFil, Fonction
ORDER BY CodeFil, Fonction ;

2- ***SELECT CodeFil, COUNT(*) FROM Employes***
GROUP BY CodeFil
ORDER by CodeFil ;

3- ***SELECT COUNT (DISTINCT Fonction) AS nbEmployes FROM Employes ;***

Ou bien :

SELECT COUNT(COUNT(*)) AS nbEmployes FROM Employes
GROUP BY Fonction ;

4- ***SELECT Fonction, COUNT(*) FROM Employes***
GROUP BY Fonction;

5- ***SELECT Fonction, COUNT(*) FROM Employes***
WHERE DateRecrut > '01-01-2012'
GROUP BY Fonction ;

Noter que la clause *WHERE* vient avant *GROUP BY*. La requête élimine d'abord, toutes les lignes ne vérifiant pas la condition *WHERE*, puis regroupe, par *Fonction*, les lignes retenues. Enfin, elle calcule à l'aide de *COUNT* le nombre de lignes de chaque groupe formé précédemment.

6- ***SELECT CodeFil, COUNT(DISTINCT Fonction) AS nbFonctions FROM Employes***
GROUP BY CodeFil
ORDER BY CodeFil ;

7- ***SELECT NomFil, COUNT(DISTINCT Fonction) FROM Employes, Filiale***
WHERE Employes.CodeFil = Filiale.CodeFil (On peut utiliser Natural Join)
GROUP BY NomFil
ORDER BY NomFil ;

8- ***SELECT CodeFil, MIN(Salaire), MAX(Salaire), SUM(Salaire), AVG(Salaire) FROM Employes***
GROUP BY CodeFil
ORDER BY CodeFil ;

9- ***SELECT Fonction, Min(Salaire) AS SalaireMin, AVG(Salaire) AS SalaireMoy***
FROM Employes
GROUP BY Fonction
ORDER BY Fonction ;

10- Si l'on veut afficher le numéro de l'employé avec le nombre de ses subordonnés, la requête sera :

SELECT NumSup, Count(*) FROM Employes
WHERE NumSup IS NOT NULL
GROUP BY NumSup ;

Ici, on a exclu le PDG (plus haut supérieur hiérarchique) à l'aide de la condition *NumSup IS NOT NULL*, sinon il s'affichera comme subordonné du vide !

Si l'on veut afficher le nom de l'employé au lieu de son numéro avec le nombre de ses subordonnés, la requête sera :

SELECT e.NomEmp, COUNT(*) FROM Employes, Employes e
WHERE Employes.NumSup = e.NumEmp
GROUP BY e.NomEmp ;

On peut trier le résultat des deux requêtes précédentes, en ajoutant *ORDER BY NumSup* pour la première et *ORDER BY e.NomEmp* pour la seconde.

11- ***SELECT CodeFil, COUNT(*) FROM Employes***
GROUP BY CodeFil
HAVING COUNT(*)> 100
ORDER BY CodeFil ;

12- ***SELECT CodeFil, COUNT(*) FROM Employes***
WHERE UPPER(Fonction) = 'INGENIEUR'
GROUP BY CodeFil
HAVING COUNT(*) > 50
ORDER BY CodeFil ;

Exercice 5

1- ***UPDATE Employes***
SET Salaire = Salaire*1.5
WHERE Salaire < 15000 AND CodeFil IN (SELECT CodeFil FROM Filiale WHERE
UPPER(VilleFil)='CASABLANCA') ;

2- ***DELETE FROM Employes***
WHERE CodeFil = (SELECT CodeFil FROM Filiale
WHERE NomFil = 'InovIndus') ;

Ici, on a mis '=' au lieu de '*IN'* car la requête interne retournera une seule valeur, du fait que l'on a une seule filiale qui porte le nom '*InovIndus'*.

Exercice 6 :

1-

- ***CREATE TABLE Patient (NUMPAT Number (8) PRIMARY KEY, NOMPAT Varchar(20),***
PRENPAT Varchar(20), DATENAISPAT Date, VILLE varchar(25))

- ***CREATE TABLE Operation (CODEOP Varchar(15) PRIMARY KEY, NOMOP Varchar(20),***
PRIXOP Number (8,2)) ;

- ***CREATE TABLE Faire_Op (NUMPAT Number (8) REFERENCES Patient (NUMPAT),***
CODEOP Varchar(15) REFERENCES Operation (CODEOP),
DATEOP Date DEFAULT sysdate,
DUREEOP Number (4),
CODEMED NUMBER(4) REFERENCES Medecin (CODEMED),
PRIMARY KEY (NUMPAT, CODEOP, DATEOP)) ;

NB : On peut utiliser le type ***INT*** à la place de *Number* (?) et ***TEXT*** à la place de varchar(?)

2- ***SELECT * FROM Operation***
ORDER BY PRIXOP Desc, NOMOP ;

3- ***SELECT * FROM Operation***
WHERE NOMOP LIKE '%fibroscopie%'
AND PRIXOP > 1000 ;

4- UPDATE Operation
SET PRIXOP = 0.9*PRIXOP
WHERE CODEOP NOT IN (SELECT CODEOP FROM Faire_OP) ;

5- SELECT CODEOP, COUNT(*) FROM Faire_OP
GROUP BY CODEOP ;

6- SELECT NOMOP, NOMPAT, NOMMED FROM Operation, Patient, Medecin, Faire_Op,
WHERE Operation.CODEOP= Faire_Op.CODEOP
AND Faire_Op.NUMPAT = Patient.NUMPAT
AND Faire_Op.CODEMED = Medecin.CODEMED ;

Ou bien

Utiliser *NATURAL JOIN* entre les tables.

7- SELECT NOMPAT, PRENPAT FROM Patient
WHERE NUMPAT IN (SELECT NUMPAT FROM Faire_OP WHERE DUREEOP>90
AND CODEOP IN (SELECT CODEOP FROM Operation
WHERE PRIXOP<= 5000)) ;

8- SELECT NOMED, COUNT(*) FROM Medecin NATURAL JOIN Faire_OP NATURAL JOIN Patient
WHERE UPPER(VILLE)='MEKNES'
AND DATEOP BETWEEN '01-01-2019' AND '31-12-2019'
GROUP BY CODEMED
HAVING COUNT(*) > 20 ;