TP n° 2 : Requêtes, dépendances fonctionnelles et normalisation

Remarques En cas de difficulté à vous connectez rapportes-vous à la section « Préparation de l'environnement de travail » de la série de TP n° 1.

Dans le précédent TP, vous avez utilisé *SQLPLUS*. Je vous invite, au cours de ce TP à découvrir le client graphique *SQLDevloper*, que vous pouvez lancer, en exécutant la commande suivante :

```
syoucef$ sqldevloper
```

Vous déposez le travail réalisé sur l'ENT.

Exercice n° 1

1. Afficher le nom du département qui a le budget le plus élevé.
2. Afficher les salaires et les noms des enseignants qui gagnent plus que le salaire moyen.
3. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignent ainsi que le nombre total de cours suivis par chaque étudiant, en utilisant la clause *HAVING*.
4. Pour chaque enseignant, afficher tous les étudiants qui ont suivi plus de deux cours dispensés par cet enseignent ainsi que le nombre total de cours suivis par chaque étudiant, sans utiliser la clause *HAVING*.
5. Afficher les identifiants et les noms des étudiants qui n'ont pas suivi de cours avant 2010.
6. Afficher tous les enseignants dont les noms commencent par E.
7. Afficher les salaires et les noms des enseignants qui perçoivent le quatrième salaire le plus élevé.
8. Afficher les noms et les salaires des trois enseignants qui perçoivent les salaires les moins élevés. Les afficher par ordre décroissant.
9. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause *IN*.
10. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause *SOME*.
11. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la jointure naturelle (*NATURAL INNER JOIN*).
12. Afficher les noms des étudiants qui ont suivi un cours en automne 2009, en utilisant la clause *EXISTS*.
13. Afficher toutes les paires des étudiants qui ont suivi au moins un cours ensemble.
14. Afficher pour chaque enseignant, qui a effectivement assuré un cours, le nombre total d'étudiant qui ont suivi ses cours. Si un étudiant a suivi deux cours différents avec le même enseignant, on le compte deux fois. Trier le résultat par ordre décroissant.
15. Afficher pour chaque enseignant, même s'il n'a pas assuré de cours, le nombre total d'étudiant qui ont suivi ses cours. Si un étudiant a suivi deux fois un cours avec le même enseignant, on le compte deux fois. Trier le résultat par ordre décroissant.
16. Pour chaque enseignant, afficher le nombre total de *grades A* qu'il a attribué.
17. Afficher toutes les paires enseignant-élèves où un élève a suivi le cours de l'enseignant, ainsi que le nombre de fois que cet élève a suivi un cours dispensé par cet enseignant.
18. Afficher toutes les paires enseignant-élève où un élève a suivi au moins deux cours dispensé par l'enseignant en question.

Exercice n° 2

Donner la forme la plus avancée des schémas de relations suivants, munis de l'ensemble de dépendances fonctionnelles F. Si une relation n'est pas normalisée, la décomposer pour atteindre la forme normale la plus avancée.

1. $R(A, B, C)$ et $F = \{A \longrightarrow B; B \longrightarrow C\}$.
2. $R(A, B, C)$ et $F = \{A \longrightarrow C; A \longrightarrow B\}$.
3. $R(A, B, C)$ et $F = \{A, B \longrightarrow C; C \longrightarrow B\}$.

Exercice n° 3

1. Soit une relation $R(A, B, C, D, E)$ et un ensemble de dépendances fonctionnelles $F = \{A \longrightarrow B, C; C, D \longrightarrow E; B \longrightarrow D; E \longrightarrow A\}$. Déduire au moins 16 dépendances fonctionnelles, en utilisant le système d'Amstrong (règles de réflexivité, augmentation et transitivité).
2. Soit une relation $R(A, B, C, D, E, F)$ et un ensemble de dépendances fonctionnelles $F = \{A \longrightarrow B, C, D; B, C \longrightarrow D, E; B \longrightarrow D; D \longrightarrow A\}$.
   (a) Calculer la fermeture de l'attribut B et de l'ensemble $\{A, B\}$
   (b) Montrer que que $\{A, F\}$ est une super-clé.
   (c) Est-elle en BCNF (*Boyce–Codd Normal Form*) ? Si R n'est pas en BCNF, la décomposer pour atteindre cette forme normale.
3. Soit une relation $R(A, B, C, D, E)$ que l'on décompose en deux relations
   (a) $R_1(A, B, C)$ et $R_2(A, D, E)$. Montrer que cette décomposition est sans perte d'information.
   (b) $R_1(A, B, C)$ $R_2(C, D, E)$. Montrer que cette décomposition est avec perte d'information.

Exercice n° 4

Pour répondre aux questions suivantes, vous pouvez utiliser le langage de programmation de votre prédilection. Je vous donne les réponses des premières quartions, pour ne pas perdre trop de temps dans le choix des structures de données à utiliser. Elles sont décrites ci-après [1] :

— un ensemble pour représenter une relation et chaque élément est une chaîne de caractères. L'exemple suivant est une déclaration de deux relations (tableau) constituées respectivement des éléments suivants $\{A, B, C, G, H, I\}$ et $\{X, Y\}$:

```
myrelations = [
                {'A', 'B', 'C', 'G', 'H', 'I'},
                {'X', 'Y'}
              ]
```

— et les dépendances est une liste de paire de deux ensembles. L'exemple suivant est une déclaration de l'ensemble des dépendances suivantes $A \longrightarrow B$; $A \longrightarrow C$; $CG \longrightarrow H$; $CG \longrightarrow I$ et $B \longrightarrow H$.

```
mydependencies = [
                   [ {'A'}, {'B'} ],  #A->B
                   [ {'A'}, {'C'} ],  #A->C
                   [ {'C', 'G'}, {'H'} ],  #CG->H
                   [ {'C', 'G'}, {'I'} ],  #CG->I
                   [ {'B'}, {'H'} ] #B->H
                 ]
```

1. Écrire une procédure qui permet de prendre en paramètre une liste de dépendances fonctionnelles et les affiche.

```
def printDependencies(F: "list of dependencies"):

    for alpha,beta in F:
        print("\t", alpha, " --> ", beta)
```

1. Quelque soit le langage de programmation que vous avez choisi, utiliser les mêmes structures de données

2. Écrire une procédure qui permet de prendre en paramètre un ensemble de relations T et les affiche.

```
def printRelations(T: "list of relations"):

    for R in T:
        print("\t", R)
```

3. Écrire une fonction qui, prenant en paramètre un ensemble *inputset*, revoie tous ces sous-ensembles. Rappelons qu'un sous-ensemble S d'un ensemble E est un ensemble tel que $S \subseteq E$. L'ensemble qui a pour éléments tous les sous-ensemble de E est noté $P(E)$. Et $card(P(E)) = 2^n$, où $n = card(E)$ (la cardinalité de E). N'oubliez pas l'instruction permettant d'importer *itertools*.

```
import itertools
```

```
def powerSet(inputset: "set"):
    _result = []
    for r in range(1, len(inputset)+1):
        _result += map(set, itertools.combinations(inputset, r))

    return _result
```

4. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F et un ensemble d'attributs K, de retourner la fermeture (clôture) de K.
5. Écrire une fonction qui permet, étant donné un ensemble de dépendances fonctionnelles F, de retourner la clôture de F. Rappel : la clôture de F est un ensemble constitué de toutes les dépendances fonctionnelles que l'on peut déduire de F.
6. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F et deux ensembles d'attributs α et β , de retourner vrai si α détermine fonctionnement β.
7. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F, une relation R et un ensemble d'attributs K, de retourner vrai si K est une super-clé.
8. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F, une relation R et un ensemble d'attributs K, de retourner vrai si K est une clé candidate.
9. Écrire une fonction qui, étant donné une relation R et un ensemble de dépendances fonctionnelles F, de retourner la liste de toutes les clés candidates.
10. Écrire une fonction qui, étant donné une relation R et un ensemble de dépendances fonctionnelles F, de retourner la liste de toutes les super-clés.
11. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F et une relation R, de retourner une clé candidate.
12. Écrire une fonction qui permet, étant donnée une relation R et un ensemble de dépendances fonctionnelles F, de retourner vrai si cette relation est en BCNF.
13. Écrire une fonction qui permet, étant donnée un ensemble de relations T et une liste de dépendances fonctionnelles F, de retourner vrai si le schéma défini par ces relations est en BCNF.
14. Écrire une fonction qui permet, étant donnée un ensemble de dépendances fonctionnelles F et un ensemble de relations T, d'implémenter l'algorithme de décomposition en BCNF, vu en cours.