EXERCICE 3 (8 points)

Cet exercice porte sur les bases de données relationnelles, les requêtes SQL et la programmation en Python.

L'énoncé de cet exercice utilise des mots-clés du langage SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN... ON`, `UPDATE... SET`, `INSERT INTO... VALUES...`, `COUNT`, `ORDER BY`.

La clause `ORDER BY` suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut précisé. `SELECT COUNT(*)`renvoie le nombre de lignes d'une requête.

Amélie souhaite organiser sa collection de CD. Elle a commencé par enregistrer toutes les informations sur un fichier CSV mais elle trouve que la recherche d'informations est longue et fastidieuse. Elle repense à son cours sur les bases de données et elle se dit qu'elle doit pouvoir utiliser une base de données relationnelle pour organiser sa collection.

Partie A

Dans cette partie on utilise une seule table.

Voici un extrait de la table `Chanson`.

`Chanson`			
`id`	`titre`	`album`	`groupe`
1	Sunburn	Showbiz	Muse
2	Muscle Museum	Showbiz	Muse
3	Showbiz	Showbiz	Muse
4	New Born	Origin of Symmetry	Muse
5	Sing for Absolution	Absolution	Muse
6	Hysteria	Absolution	Muse
7	Welcome too the Jungle	Appetite for Destruction	Guns N' Roses
8	Muscle Museum	Hullabaloo	Muse
9	Showbiz	Hullabaloo	Muse

1. L'attribut `titre` peut-il être une clé primaire pour la table `Chanson` ? Justifier.

2. Donner le résultat de la requête suivante :

```
SELECT titre, album
FROM Chanson
WHERE groupe = 'Guns N’Roses';
```

3. Écrire une requête SQL permettant d’obtenir tous les titres des chansons de l’album Showbiz dans l’ordre croissant.

4. Écrire une requête SQL permettant d’ajouter la chanson dont le titre est Megalomania de l’album Hullabaloo du groupe Muse.

Amélie a remarqué une faute de frappe dans la chanson Welcome too the Jungle qui s’écrit normalement Welcome to the Jungle.

5. Écrire une requête SQL permettant de corriger cette erreur.

Partie B

Dans cette partie on utilise trois tables.

Voici des extraits des trois tables `Chanson`, `Album` et `Groupe`.

Chanson		
id	titre	id_album
1	Sunburn	1
2	Muscle Museum	1
3	Showbiz	1
4	New Born	2
5	Sing for Absolution	4
6	Hysteria	4
7	Welcome to the Jungle	5
8	Muscle Museum	3
9	Showbiz	3

Album			
id	titre	année	id_groupe
1	Showbiz	1999	1
2	Origin of Symmetry	2001	1
3	Hullabaloo	2002	1
4	Absolution	2003	1
5	Appetite for Destruction	1987	2

Groupe	
id	nom
1	Muse
2	Guns N' Roses

6. Expliquer l'intérêt d'utiliser trois tables `Chanson`, `Album` et `Groupe` au lieu de regrouper toutes les informations dans une seule table.

7. Expliquer le rôle de l'attribut `id_album` de la table `Chanson`.

8. Proposer alors un schéma relationnel pour cette version de la base de données. On pensera à bien spécifier les clés primaires en les soulignant et les clés étrangères en les faisant précéder par le symbole `#`.

9. Écrire une requête SQL permettant d'obtenir tous les noms des albums contenant la chanson Showbiz.

10. Écrire une requête SQL permettant d'obtenir tous les titres avec le nom de l'album des chansons du groupe Muse.

11. Décrire par une phrase ce qu'effectue la requête SQL suivante :

```
SELECT COUNT(*) AS tot
FROM Album AS a
JOIN Groupe AS g ON a.id_groupe = g.id
WHERE g.nom = 'Muse';
```

Partie C

Dans cette partie, on utilise Python.

Amélie a remarqué que son professeur ne parle jamais d'ordre alphabétique mais d'ordre lexicographique lorsqu'il fait une requête avec `ORDER BY`.

Elle a compris qu'il s'agissait de l'ordre du dictionnaire mais elle se demande comment elle pourrait elle-même écrire une fonction `ordre_lex(mot1, mot2)` de comparaison entre deux chaînes de caractères en utilisant l'ordre lexicographique. La fonction

`ordre_lex(mot1, mot2)` prend en arguments deux chaines de caractères et renvoie un booléen. Une rapide recherche lui permet de trouver le résultat suivant :

Lorsque l'on compare deux chaînes de caractères suivant l'ordre lexicographique, on commence par comparer les deux premiers caractères de chacune des deux chaînes, puis en cas d'égalité on s'intéresse au second, et ainsi de suite. Le classement est donc le même que celui d'un dictionnaire. Si lors de ce procédé on dépasse la longueur d'une seule des deux chaînes, elle est considérée plus petite que l'autre. Lorsqu'on dépasse la longueur des deux chaînes au même moment, elles sont nécessairement égales

Amélie commence par écrire quelques assertions que sa fonction devra vérifier.

12. Compléter les assertions suivantes :

```
1  assert ordre_lex("", "a") == True
2  assert ordre_lex("b", "a") == ...
3  assert ordre_lex("aaa", "aaba") == ...
```

On suppose que les chaînes de caractères `mot1` et `mot2` ne sont composées que des lettres de l'alphabet, en minuscule, et la comparaison entre deux lettres peut se faire avec les opérateurs classiques == et <.
Par exemple :

```
1  >>> "" < "a"
2  True
3  >>> "b" == "a"
4  False
```

Enfin, le slice `mot1[1:]` renvoie la chaîne de caractère de `mot1` privée de son premier caractère.
Par exemple :

```
1  >>> mot1 = "abcde"
2  >>> mot2 = mot1[1:]
3  >>> mot2
4  "bcde"
```

13. Recopier et compléter la fonction récursive `ordre_lex` ci-dessous qui prend pour paramètre deux chaînes de caractères `mot1` et `mot2` et qui renvoie `True` si `mot1` précède `mot2` dans l'ordre lexicographique.

```
1  def ordre_lex(mot1, mot2):
2      if mot1 == "":
3          return True
4      elif mot2 == "":
5          return False
6      else:
7          c1 = mot1[0]
8          c2 = mot2[0]
9          if c1 < c2:
10             return ...
11         elif c1 > c2:
12             return ...
13         else:
14             return ...
```

14. Proposer une version itérative de la fonction `ordre_lex`.