Travaux pratiques - Langage SQL

Pour les exercices 1, 2 et 3, vous travaillerez avec la base de données `disquaire.db` que vous trouverez dans le répertoire de ce chapitre. Vous utiliserez le logiciel DB Browser for SQLite dans lesquels vous aurez au préalable ouvert la base de données en question.

On rappelle que cette base de données possède le schéma relationnel suivant :

```
Album(id_album INT, titre TEXT, annee INT, dispo BOOL)
```

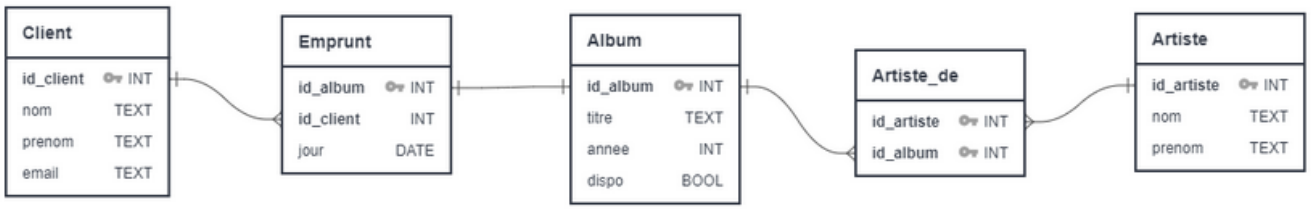
```
Artiste(id_artiste INT, nom TEXT, prenom TEXT)
```

```
Artiste_de(#id_artiste INT, #id_album INT)
```

```
Client(id_client INT, nom TEXT, prenom TEXT, email TEXT)
```

```
Emprunt(#d_client INT, #id_album INT, date DATE)
```

On peut aussi représenter graphiquement ce schéma par le diagramme suivant :



Réalisé avec l'application quickdatabasediagrams.com

Exercice 1 - Requêtes portant sur une table

1. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT * FROM Album;
```

2. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT titre, dispo FROM Album;
```

3. Écrivez et testez une requête permettant de récupérer uniquement les titres et l'année de sortie de chaque album.
4. Écrivez et testez une requête permettant de récupérer tous les attributs des clients.
5. Écrivez et testez une requête permettant de récupérer uniquement l'id_client, le nom et le prénom de chaque client.
6. Écrivez et testez une requête permettant de récupérer uniquement le nom et le prénom de chaque artiste.

En plus de sélectionner des colonnes, on peut sélectionner certaines lignes en utilisant la clause WHERE suivie de la condition de sélection.

7. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT titre FROM Album WHERE dispo=0;
```

8. Écrivez et testez une requête permettant de récupérer les titres de tous les albums sortis en 2000 ou après.
   Résultat attendu : 12 enregistrements.
9. Écrivez et testez une requête permettant de récupérer tous les albums sortis en 1970.
   Résultat attendu : 3 enregistrements.
10. Écrivez et testez une requête permettant de récupérer les titres de tous les albums sortis avant 1950 ou après 2010.
    Résultat attendu : 6 enregistrements.
11. Écrivez et testez une requête permettant de récupérer tous les albums disponibles et sortis avant 1990.
    Résultat attendu : 13 enregistrements.
12. Écrivez et testez une requête permettant de récupérer tous les clients dont le nom de famille est "Petit".
    Résultat attendu : 2 enregistrements.
13. Écrivez et testez une requête permettant de récupérer tous les clients dont le nom de famille est "Chartier".
    Résultat attendu : 0 enregistrements.
14. Écrivez et testez une requête permettant de récupérer les noms des artistes ne possédant pas de prénom.
    Résultat attendu : 6 enregistrements.

On peut trier des données en utilisant ORDER BY à la fin d'une requête, suivi de l'attribut à trier et de ASC (pour un tri croissant) ou DESC (pour un tri décroissant).

15. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT * FROM Artiste ORDER BY nom ASC;
```

16. Écrivez et testez une requête permettant de récupérer les albums triés dans l'ordre décroissant de leur année de sortie.
17. Écrivez et testez une requête permettant de récupérer les titres par ordre alphabétique des albums sortis entre 1980 et 2010. Résultat attendu : 10 enregistrements.
18. Écrivez et testez une requête permettant de récupérer les noms et prénoms de tous les clients, triés par ordre alphabétique des noms.
19. Écrivez et testez une requête permettant de récupérer les noms et prénoms de tous les clients, triés par ordre alphabétique des noms, puis des prénoms en cas de noms identiques.
    Vous vérifierez bien que les clients portant le même nom sont bien triés alphabétiquement selon leur prénom (ce qui n'était pas le cas de la précédente requête).

On peut supprimer les doublons grâce à DISTINCT.

20. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT DISTINCT id_client FROM Emprunt;
```

21. Écrivez et testez une requête permettant de récupérer les titres distincts des albums.
Résultat attendu : 29 enregistrements (car un album était en double).

    On peut compter le nombre d'enregistrements (ou tuples) en utilisant la fonction COUNT().

22. Traduisez par une phrase la requête suivante. Vérifiez ensuite votre réponse en l'exécutant.

```
SELECT COUNT(*) AS nbClients FROM Client;
```

    On rappelle que AS permet de nommer le résultat de la requête, ici on le nomme nbClients. Essayez d'enlever le AS nbClients et observez le résultat de la requête.

23. Écrivez et testez une requête permettant de récupérer le nombre d'emprunts en cours, que l'on notera nbEmprunts.
Résultat attendu : 11.
24. Écrivez et testez une requête permettant de récupérer le nombre de clients ayant le nom "Petit", que l'on notera nbPetit.
Résultat attendu : 2.
25. Écrivez et testez une requête permettant de récupérer le nombre de titres différents d'albums, que l'on notera nbAlbumsDistincts.
Résultat attendu : 29.

    On peut effectuer des requêtes effectuant des recherches de certains motifs.

    Par exemple, on peut chercher tous les clients dont l'adresse email contient le domaine "domaine.net". La requête s'écrirait :

```
SELECT * FROM Client WHERE email LIKE "%domaine.net%";
```

    Analyse :
    On a remplacé le = qui fait une recherche exacte par LIKE. Ainsi, email LIKE "%domaine.net%" est évaluée à vrai si et seulement si l'attribut email correspond au motif "%domaine.net%". Dans un motif le symbole % est un joker et peut être substitué par n'importe quelle chaîne de caractères.

26. Écrivez et testez une requête permettant de récupérer les titres de tous les albums contenant le mot "Love".
Résultat attendu : 2 enregistrements.
27. Écrivez et testez une requête permettant de récupérer le nombre d'albums dont le titre contient la lettre "a".
Résultat attendu : 16 enregistrements.
28. Écrivez et testez une requête permettant de récupérer les noms et prénoms de tous les artistes commençant par la lettre "M".
Résultat attendu : 3 enregistrements.

Exercice 2 - Requêtes croisées

1. Écrivez et testez une requête permettant de récupérer l'adresse email de chaque client ayant un emprunt en cours.
   Résultat attendu : 11 enregistrements.
2. Écrivez et testez une requête permettant de récupérer les noms, prénoms et adresses email des clients ayant un emprunt en cours et dont la date d'emprunt est postérieure au 2 octobre 2021.
   Résultat attendu : 5 enregistrements.
3. En réalité, l'attribut dispo (de la table Album) n'est pas utile car on peut retrouver tous les albums empruntés avec une jointure. En supposant que l'attribut dispo n'existe pas, écrivez et testez une requête permettant de récupérer le titre de tous les albums empruntés.
   Résultat attendu : 11 enregistrements.
4. Toujours sans utiliser l'attribut dispo, écrivez et testez une requête permettant de récupérer le titre de tous les albums empruntés par le client dont l'attribut id_client vaut 7.
   Résultat attendu : 3 enregistrements.
5. Toujours sans utiliser l'attribut dispo, écrivez et testez une requête permettant de récupérer le nom, le prénom et le titre de tous les albums empruntés par le client dont l'attribut id_client vaut 7.
   Résultat attendu : 3 enregistrements (comme à la question précédenet, le client en question est Alain Moreau).
6. Écrivez et testez une requête permettant de récupérer les titres ainsi que les noms et prénoms des artistes de chaque album.
   Résultat attendu : 34 enregistrements.
7. Écrivez et testez une requête permettant de récupérer les titres et l'année de sortie de tous les albums de Michael Jackson.
   Résultat attendu : 3 enregistrements.
8. Écrivez et testez une requête permettant de récupérer les titres et l'année de sortie de tous les albums de Sting, rangés par ordre croissant d'année de sortie.
   Résultat attendu : 3 enregistrements.
9. Écrivez et testez une requête permettant de récupérer les noms et prénoms des artistes de l'album initulé "Don't Explain".
   Résultat attendu : 2 enregistrements.

Exercice 3 - Modification de la base du disquaire

1. Le prénom du client "Robert Pascal" a mal été saisi dans la base de données. En effet, son prénom n'est pas Pascal mais Pascale. Écrivez et testez une requête permettant corriger cette erreur. Vous vérifierez que la correction a bien été effectuée !
2. Un titre d'album a mal été saisi dans la base de données. Il faut remplacer Riding With The King par Riding with the King. Écrivez et testez une requête permettant corriger cette erreur. Vous vérifierez que la correction a bien été effectuée !
3. Le client "Durand Julien" souhaite que les informations personnelles le concernant soient supprimées de la base de données (c'est son droit avec le RGPD). Écrivez et testez la requête permettant d'effectuer ces suppressions. Vous vérifierez que la suppression a bien été effectuée !
4. Essayez de supprimer le client "Dupont Florine". Quelle est l'erreur provoquée ? Expliquez-la (voir Cours si nécessaire).

Exercice 4 - Créer et modifier une table

1. Ouvrez DB Browser for SQLite : cliquez sur ”Nouvelle Base de Données” puis appelez-la bac.db et validez. Fermez ensuite la fenêtre de l'assistant de création de table. Vous êtes désormais prêt pour la suite !
2. Écrivez la requête permettant de créer une table Note correspondant à la relation suivante. Vous vérifierez ensuite dans la structure de la base de données que la table a bien été créée avec les bons attributs et les bons types.

```
Note(id_eleve INT, nom TEXT, prenom TEXT, maths INT, anglais INT,
info INT)
```

3. Écrivez la requête permettant d'insérer les trois enregistrements suivants dans la table Note.

id_eleve	nom	prenom	maths	anglais	info
1	Marchand	Alice	14	13	11
2	Muller	Marie	10	18	17
3	Prenel	Laura	13	14	15

4. Écrivez et testez une requête permettant d'afficher tous les enregistrements de la table Note afin de vérifier que les données ont bien été ajoutées.
5. Écrivez et testez la requête permettant d'ajouter l'enregistrement `(3, 'Dupont', 'Arthur', 18, 14, 13)` à la table Note. Quelle est l'erreur provoquée ? Expliquez-la (voir Cours si nécessaire).
6. Écrivez et testez une requête permettant d'ajouter l'enregistrement concernant Arthur mais avec la clé primaire `id_eleve` égale à 4. Vous vérifierez que l'enregistrement a bien été ajouté !
   À ce stade, vous devriez obtenir la table Note suivante :

id_eleve	nom	prenom	maths	anglais	info
1	Marchand	Alice	14	13	11
2	Muller	Marie	10	18	17
3	Prenel	Laura	13	14	15
4	Dupont	Arthur	18	14	13

7. Il y a eu une erreur dans la saisie de l'élève numéro 3 : son prénom n'est pas Laura mais Laure. Écrivez et testez une requête permettant d'effectuer la correction. Vous vérifierez que la modification a bien été effectuée !.
8. Un autre erreur a été détectée : les notes de Mathématiques et d'Anglais de Marchand Alice ont été inversées. Écrivez et testez une requête permettant d'effectuer la correction. Vous vérifierez que la modification a bien été effectuée !.
9. Pour des raisons d'équité entre établissement, il a été décidé que les notes de mathématiques de tous les élèves devaient augmenter d'un point. Écrivez et testez une requête permettant d'effectuer la correction. Vous vérifierez que la modification a bien été effectuée !.

Exercice 5 **Murder mystery**

https ://qkzk.xyz/docs/nsi/cours_terminale/bdd/murder_mystery/

Exercice 6 SQL avec Python

Création et connexion avec la base de données

Premièrement, on doit créer une nouvelle base de données et s'y connecté pour permettre au module `sqlite3` d'intéragir avec elle.

On utilise `sqlite3.connect()` pour créer la connexion avec la base de données `eleves.db` dans le répertoire courant, cette base étant créée si elle n'existe pas.

```
import sqlite3
conn = sqlite3.connect('eleves.db')
```

L'objet `conn` représente la connexion avec la base de données sur le disque.

Une fois la connexion établie, il est nécessaire de créer un curseur grâce à `conn.cursor()` (c'est un objet de la classe `Cursor`).

```
cur = conn.cursor()
```

Maintenant que nous avons la connexion et un curseur on peut exécuter des requêtes SQL et récupérer les résultats grâce au curseur.

```
cur.execute("""CREATE TABLE Eleve (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT,
    prenom TEXT,
    age INTEGER,
    classe TEXT);
    """)
```

Remarques :

- On peut utiliser des triples guillemets pour écrire les chaînes de requêtes sur plusieurs lignes et mieux les formater visuellement.
- SQLite ne prévoit que 5 domaines pour les attributs d'une table : NULL, INTEGER, REAL, TEXT, BLOB. Voir ce lien pour plus de détails : https ://www.sqlite.org/datatype3.html
- On a utilisé AUTOINCREMENT pour ne plus avoir besoin de spécifier l'id comme clé primaire : celle-ci s'autoincrémente à chaque nouvel enregistrement dans la base. C'est très utile car la plupart du temps on ne sait pas quelle clé primaire utiliser pour ajouter un nouvel enregistrement.

Une fois la requête exécutée, il ne faut pas oublier de stopper la connexion avec la base de données. On utilise pour cela la méthode `close` :

```
conn.close()
```

On peut créer une fonction qui permet de créer notre table, en la supprimant si elle existe déjà (pour mieux faire nos essais) :

```
def creer_table():
    conn = sqlite3.connect('eleves.db')
    cur = conn.cursor()
    cur.execute("DROP TABLE IF EXISTS Eleve;")
    cur.execute("""CREATE TABLE Eleve (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nom TEXT,
        prenom TEXT,
        age INTEGER,
        classe TEXT);
        """)
    conn.close()
```

Insérer des enregistrements

Pour insérer des enregistrements, on utilise notre curseur pour exécuter des requêtes SQL.

Insérer un enregistrement

Si on souhaite insérer un élève dans la table `Eleve` on appelle la méthode `cur.execute()` en lui passant en paramètre une chaîne de caractères correspondant à une requête SQL :

```
cur.execute("INSERT INTO Eleve (id, nom, prenom, age, classe) VALUES (1, 'Dupont',
    'Jean', 15, '2A')")
conn.commit()
```

Remarque : L'instruction INSERT crée ce qu'on appelle une **transaction**, qui doit être validée par `conn.commit()` afin que les changements soient enregistrés en base de données !

Dans le cas précédent, on les données à insérer étaient directement écrites dans la chaîne de requête. Or, bien souvent les données que l'on veut utiliser dans une requête sont stockées dans des variables et on peut utiliser ces variables pour construire la chaîne de requête. On appelle cela une **requête préparée**, qui fait appel au caractère `?` de la façon suivante :

```
eleve_2 = ('Dupont', 'Jeanne', 17, 'TG2')
cur.execute("INSERT INTO Eleve (nom, prenom, age, classe) VALUES (?, ?, ?, ?)",
    eleve_2)
conn.commit()
```

On passe un `tuple` en deuxième paramètre à notre méthode `execute` : ce tuple contient les valeurs par lesquelles SQLite doit remplacer les `?`.

Cette manière de faire permet de se prémunir d'une faille de sécurité appelée injection SQL dont nous reparlerons par la suite.

Il est également possible de construire une requête à partir de données mémorisées dans un dictionnaire. Ainsi, on peut insérer un troisième élève de cette façon :

```
eleve_3 = {'nom': 'Marchand', 'prenom': 'Marie', 'age': 15, 'classe': '2A'}
cur.execute("""INSERT INTO Eleve (nom, prenom, age, classe) VALUES (:nom, :prenom,
    :age, :classe)""", eleve_3)
conn.commit()
```

De cette façon, SQLite va remplacer :nom, :prenom, :age et :classe dans la chaîne de requête par les valeurs associées aux clés 'nom', 'prenom', 'age' et 'classe' du dictionnaire passé en deuxième argument.

L'utilisation des clés des dictionnaires a l'avantage d'être plus explicite et de ne plus se soucier de l'ordre des données (comme pour un tuple par exemple)

Insérer plusieurs enregistrements

Si on dispose de données sur plusieurs élèves, on peut utiliser une boucle `for` en construisant la chaîne de requête SQL pour chaque élève de la façon suivante :

```
autres_eleves = [
    ('Martin', 'Adeline', 16, '1G1'),
    ('Dupont', 'Lucas', 15, '2A'),
    ('Richard', 'Louise', 15, '1G2'),
    ('Rouger', 'Marius', 16, '1G2'),
    ('Louapre', 'Lola', 18, 'TG2'),
    ('Boudou', 'Lise', 17, 'TG1'),
    ('Dupont', 'Aurelien', 16, '1G1'),
    ('Laurent', 'Diego', 17, '1G2'),
    ('Martin', 'Anaelle', 16, 'TG1')
]

for eleve in liste_eleves:
    cur.execute("INSERT INTO Eleve (nom, prenom, age, classe) VALUES (?, ?, ?, ?)",
        eleve)
conn.commit()
```

En réalité, le code précédent peut se raccourcir en utilisant la méthode `cur.executemany()` qui permet justement d'insérer plusieurs lignes en base de données directement. On préférera donc écrire le code équivalent qui suit :

```
cur.executemany("INSERT INTO Eleve (nom, prenom, age, classe) VALUES (?, ?, ?, ?)",
    autres_eleves)
conn.commit()
```

Il suffit de passer en deuxième paramètre à cette méthode une liste de séquences d'éléments (ici une liste de tuples).

Récupérer des données

Les méthodes fetchall, fetchone et fetchmany

On peut exécuter des requêtes d'interrogation de la base de données. On utilise pour cela également la méthode `cur.execute()`. On peut affecter le résultat de la requête dans une variable `res`, qui est alors un itérable que l'on peut utiliser pour afficher les résultats.

Pour renvoyer tous les résultats, on applique la méthode `fetchall()` à `res` :

```
res = cur.execute("SELECT * FROM Eleve")
print(res.fetchall())
#OUTPUT
[(1, 'Dupont', 'Jean', 15, '2A'), (2, 'Dupont', 'Jeanne', 17, 'TG2'), (3, 'Marchand
    ', 'Marie', 15, '2A'),
(4, 'Martin', 'Adeline', 16, '1G1'), (5, 'Dupont', 'Lucas', 15, '2A'), (6, 'Richard
    ', 'Louise', 15, '1G2'),
(7, 'Rouger', 'Marius', 16, '1G2'), (8, 'Louapre', 'Lola', 18, 'TG2'), (9, 'Boudou
    ', 'Lise', 17, 'TG1'),
(10, 'Dupont', 'Aurelien', 16, '1G1'), (11, 'Laurent', 'Diego', 17, '1G2')]
```

Vous remarquerez que la méthode `fetchall` renvoie une liste de tuples. À ce titre on peut aussi simplement convertir l'itérable `res` en une liste :

```
res = cur.execute("SELECT * FROM Eleve")
l = list(res)
print(l)
#OUTPUT
[(1, 'Dupont', 'Jean', 15, '2A'), (2, 'Dupont', 'Jeanne', 17, 'TG2'), (3, 'Marchand
    ', 'Marie', 15, '2A'),
(4, 'Martin', 'Adeline', 16, '1G1'), (5, 'Dupont', 'Lucas', 15, '2A'), (6, 'Richard
    ', 'Louise', 15, '1G2'),
(7, 'Rouger', 'Marius', 16, '1G2'), (8, 'Louapre', 'Lola', 18, 'TG2'), (9, 'Boudou
    ', 'Lise', 17, 'TG1'),
(10, 'Dupont', 'Aurelien', 16, '1G1'), (11, 'Laurent', 'Diego', 17, '1G2')]
```

Pour renvoyer un résultat, on applique la méthode `fetchone()` à `res` :

```
res = cur.execute("SELECT * FROM Eleve")
print(res.fetchone())
#OUTPUT
(1, 'Dupont', 'Jean', 15, '2A')
```

La méthode `fetchone` renvoie en réalité la prochaine ligne du résultat de la requête. Par exemple, si on rappelle à nouveau la méthode `fetchone`, elle renvoie le deuxième élève :

```
print(res.fetchone())
#OUTPUT
(2, 'Dupont', 'Jeanne', 17, 'TG2')
```

Pour choisir le nombre de résultats à renvoyer, on applique la méthode fetchmany() à res en lui passant en paramètre le nombre souhaité :

```
res = cur.execute("SELECT * FROM Eleve")
print(res.fetchmany(3))
#OUTPUT
[(1, 'Dupont', 'Jean', 15, '2A'), (2, 'Dupont', 'Jeanne', 17, 'TG2'), (3, 'Marchand
    ', 'Marie', 15, '2A')]

print(res.fetchmany(3))
#OUTPUT
[(4, 'Martin', 'Adeline', 16, '1G1'), (5, 'Dupont', 'Lucas', 15, '2A'), (6, '
    Richard', 'Louise', 15, '1G2')]
```

Remarque : Pour les requêtes d'interrogation, il est inutile d'utiliser `conn.commit()` puisque ces requêtes n'entraînent aucun changement de la base de données.

Requêtes avec paramètres

On peut bien sûr construire des requêtes d'interrogation avec des paramètres. Pour cela il faut écrire une requête préparée comme vu précédemment. Par exemple, si on veut récupérer tous les élèves ayant pour nom Dupont, on écrira :

```
>>> res = cur.execute("SELECT id, nom, prenom FROM Eleve WHERE nom = ?", ('Dupont
    ',))
>>> res.fetchall()
[(1, 'Dupont', 'Jean'), (2, 'Dupont', 'Jeanne'),
(5, 'Dupont', 'Lucas'), (10, 'Dupont', 'Aurelien')]
```

Remarques

- Vous noterez que l'on a écrit ('Dupont',) en deuxième paramètre pour bien passer un tuple à la méthode execute (et non ('Dupont') qui n'en serait pas un).
- on aurait aussi pu utiliser la syntaxe avec les dictionnaires : cur.execute("SELECT id, nom, prenom FROM Eleve WHERE nom = :nom", 'nom' : 'Dupont')

Imaginons que notre application possède un champ permettant à l'utilisateur de saisir un nom pour chercher un élève dans la base de données, on peut créer une fonction qui serait appelée lorsque la saisie est validée.

En effet, on peut facilement écrire une telle fonction qui renvoie la liste de tous les élèves dont le nom lui est passé en paramètre. Cela ressemble à quelque chose comme ci-dessous :

```
def recuperer_eleves_par_nom(nom):
    conn = sqlite3.connect('eleves.db')
    cur = conn.cursor()
    res = cur.execute("SELECT id, nom, prenom FROM Eleve WHERE nom = ?", (nom, ))
    eleves = res.fetchall()  # on stocke les resultats pour pouvoir les renvoyer
    conn.close()
    return eleves  # apres avoir ferme la connexion
```

On peut bien sûr appeler cette fonction :

```
>>> recuperer_eleves_par_nom('Martin')
[(4, 'Martin', 'Adeline', '1G1'), (12, 'Martin', 'Anaelle', 'TG1')]
>>> recuperer_eleves_par_nom('Richard')
[(6, 'Richard', 'Louise', '1G2')]
>>> recuperer_eleves_par_nom('Obama')
[]
```

PARTIE A

1. Écrire une fonction `recuperer_eleves_par_classe` qui prend un paramètre `classe` et qui renvoie la liste de tous les élèves de la table `Eleve` qui sont dans la classe classe.

   Exemple :

   ```
   >>> recuperer_eleves_par_classe('1G1')
   [(4, 'Martin', 'Adeline', 16, '1G1'), (10, 'Dupont', 'Aurelien', 16, '1G1')]
   ```

2. Écrire une fonction `recuperer_eleve` qui prend un entier `id_eleve` en paramètre et qui renvoie l'élève (les attributs nom, prenom, classe uniquement) ayant pour identifiant `id_eleve`.

Exemple :

```
>>> recuperer_eleve(3)
('Marchand', 'Marie', '2A')
```

3. Écrire une fonction `changer_classe` qui prend en paramètre un entier `id_eleve` correspondant à l'attribut id d'un élève, et une chaîne de caractère `nouvelle_classe` et qui modifie la classe de l'élève en question en base de données.
   Exemple :

```
>>> changer_classe(3, '2C')
>>> recuperer_eleve(3)
('Marchand', 'Marie', '2C')
```

4. Écrire une fonction `recuperer_liste_alphabetique_par_classe` qui prend un paramètre classe et qui renvoie la liste triée par ordre alphabétique de tous les élèves de la table Eleve qui sont dans la classe classe.
   Le tri doit être réalisé par la requête SQL (vous pourrez dans un second temps essayer de le réaliser avec Python si vous le souhaitez en triant la liste des résultats).
   Exemple :

```
>>> recuperer_liste_alphabetique_par_classe('1G2')
[('Laurent', 'Diego', 17, '1G2'), ('Richard', 'Louise', 15, '1G2'), ('Rouger',
   'Marius', 16, '1G2')]
```

PARTIE B

1. Écrire une classe Python appelée `Eleve` dont les instances possèdent 4 attributs : `nom` (de type str), `prenom` (de type str), `age` (de type int) et `classe` (de type str).
2. Quelle instruction permet de créer un objet de la classe `Eleve` vous correspondant ?
3. Écrivez une fonction `ajouter_eleve(eleve)` qui prend en paramètre un objet `eleve` de la classe `Eleve` et qui ajoute cet élève en base de données.
4. Utilisez cette fonction pour ajouter l'élève créé à la question 2 en base de données puis vérifiez que l'ajout a bien été réalisé.

Sources :

- Lycée Mounier - Angers