Séquence 4 – Le langage SQL

Objectifs

1. Identifier les services rendus par un système : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.
2. Identifier les composants d'une requête.
3. Construire des requêtes d'interrogation à l'aide de : SELECT, FROM, WHERE, JOIN.
4. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.

Cette séquence s' appuie sur :

- https://pixees.fr/informatiquelycee/n_site/nsi_term_bd_intro.html
- https://pixees.fr/informatiquelycee/n_site/nsi_term_bd_rela.html
- https://pixees.fr/informatiquelycee/n_site/nsi_term_bd_sql.html

0 Rappels : Manipulation des fichiers csv

0.1 Les fichiers csv

Voici un parallèle feuille de calcul vs. Fichier csv :

	A	B	C	D
1	Nom	Anglais	Info	Maths
2	Joe	17	18	16
3	Zoé	15	17	19
4	Max	19	13	14

Le fichier CSV correspondant est :

```
'Nom','Anglais','Info','Maths'
'Joe','17','18','16'
'Zoé','15','17','19'
'Max','19','13','14'
```

0.2 Import d'un fichier csv

Pour l' import, on crée une liste de dictionnaires (un par ligne de la table).
La première ligne du fichier CSV est considérée comme la ligne des noms des attributs. fichier est une chaîne de caractères donnant le nom du fichier sans son extension. Par exemple `depuis_csv('Ma_Base')` pour charger le fichier `Ma_Base.csv`.

```
import csv
def depuis_csv(fichier) :
    mon_fichier = open(fichier + '.csv','r')
    lecteur = csv.DictReader(mon_fichier)
    ma_liste = [ ]
    for ligne in lecteur :
        dico_ligne = dict(ligne)
        ma_liste.append(dico_ligne)
    return ma_liste
```

```
>>> dict([('one',1),('two',2)])
  {'one': 1, 'two': 2}
```

Voici le code en plus condensé :

```
import csv
def depuis_csv(fichier) :
    lecteur = csv.DictReader(open(fichier + '.csv','r'))
    return [dict(ligne) for ligne in lecteur]
```

0.3 Export vers un fichier CSV

Pour l' export, on entre le nom de la table sous forme de chaîne. On donne l'ordre des colonnes sous forme d'une liste d' attributs.

```
def vers_csv(nom, ordre):
    with open(nom + '.csv', 'w') as fic:
        dic= csv.DictWriter(fic,fieldnames=ordre)
        table = eval(nom)
        dic.writeheader() # 1re ligne, celle des attributs
```

```
            for ligne in table:
                dic.writerow(ligne)
        return None
```

0.4 Opérations sur les tables

On peut donc créer des outils de manipulation des tables, comme la recherche et le tri.

0.4.1 Sélection de lignes vérifiant un critère

On cherche à créer une nouvelle table en extrayant d'une table les lignes satisfaisant une condition donnée sous la forme d'une fonction booléenne.

MOT CLÉ - Les opérateurs booléens habituels sont :

`<, >, <=, >=, ==, !=, in, not, and, or, is…`

On crée la fonction `select` ci-dessous qui prends en paramètre une table et un critère de sélection (critère) sous forme d'une chaîne de caractères contenant un test booléen prenant une ligne en argument. La fonction renvoie une liste construite par compréhension avec un filtre qui ne contient que les lignes de la table qui satisfont le critère donné en argument.

```
def select(table, critère):
    def test(ligne):
        return eval(critère)
    return [ligne for ligne in table if test(ligne)]
```

La fonction python `eval` est nouvelle.

Voici un exemple :

```
>>> eval("2==2")
True
>>> eval("2==3")
False
```

0.4.2 Sélection de colonnes

Pour sélectionner un ou plusieurs attributs (colonnes) d'une table (cette opération s'appelle « projection » dans le langage des bases de données), on va créer une nouvelle table qui ne contiendra que ces attributs :

```
def projection(table, liste_attributs):
    ma_liste=[ ]
    for ligne in table :
        mon_dico = {}
        for clé in ligne :
            if clé in liste_attributs :
                mon_dico[clé] = ligne[clé]
        ma_liste.append(mon_dico)
    return ma_liste
```

Voici le code en plus condensé :

```
def projection(table, liste_attributs):
return [{clé:ligne[clé] for clé in ligne if clé in liste_attributs} \
                for ligne in table]
```

0.4.3 Tri d'une table selon une colonne

Puisqu'une table est représentée par une liste, on peut la trier en utilisant la fonction de tri sorted. Un troisième argument, `reverse` (un booléen), permet de préciser si l'on veut le résultat par ordre croissant (par défaut) ou décroissant (en précisant `reverse=True`).

On peut alors créer une fonction `tri` qui trie n'importe quelle table en donnant l'attribut choisi pour le tri et en précisant si l'on veut obtenir le tri dans l'ordre décroissant.

```
def tri(table, attribut, decroit=False):
    def critère(ligne):
        return ligne[attribut]
    return sorted(table, key=critère, reverse=decroit)
```

A faire vous même 1.

- Téléchargez le fichier `base.csv` :
  http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/base.csv
- Reprenez les fonctions ci-dessus (`depuis_csv, select, projection et tri`) et testez-les avec `base.csv`

A faire vous même 2.

- Téléchargez le fichier localisation_colleges_cotes_d_armor.csv :
  http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/localisation_colleges_cotes_d_armor.csv
- Reprenez la fonction `select` ci-dessus et ressortez tous les établissements de Dinan
  ……
  ……
- Reprenez la fonction `projection` ci-dessus et ressortez toutes les données avec seulement les colonnes :
  - Communesrelevantdusecteurderecrutement
  - Detailsecteurscolaire
  - Commune
  - college
- Reprenez la fonction `tri` ci-dessus et triez les données par ordre alphabétique de la colonne Commune.

0.5 Jointures de tables

Lorsque l'on traite de grandes quantités de données, celles-ci sont souvent réparties dans plusieurs tables. On est donc souvent amené à regrouper des données dans une nouvelle table. Cette opération s'appelle la jointure de tables.

0.5.1 Fusion de deux tables pour un même attribut

On veut fusionner deux tables selon un attribut commun. On va sélectionner dans chaque table la ligne ayant la même valeur pour l'attribut choisi.
Exemple :

	Nom	Maths	Anglais	Info
0	Joe	16	17	18
1	Zoé	19	15	17
2	Max	14	19	13

	Nom	Âge	Courriel
0	Joe	16	joe@info.fr
1	Zoé	15	zoe@info.fr

Devient :

	Nom	Âge	Courriel	Maths	Info	Anglais
0	Joe	16	joe@info.fr	16	18	17
1	Zoé	15	zoe@info.fr	19	17	15

Remarque : Cependant, dans certaines tables, l'attribut commun peut avoir une autre appellation. On précisera alors l'attribut de la seconde table :

```
>>> jointure(Table1, Table2, 'Nom', 'Name')
```

Voici une proposition de code :

```
1  from copy import deepcopy
2  def jointure(table1, table2, cle1, cle2=None):
3      if cle2 is None:
4          cle2 = cle1
5      new_table = []
6      for ligne1 in table1:
7          for ligne2 in table2:
8              if ligne1[cle1] == ligne2[cle2]:
9                  new_line = deepcopy(ligne1)
10                 for cle in ligne2:
11                     if cle != cle2:
12                         new_line[cle] = ligne2[cle]
13                 new_table.append(new_line)
14     return new_table
```

Ligne 3 : par défaut, les clés de jointure portent le même nom.

Ligne 5 : la future table créée, vide au départ.
Ligne 8 : on ne considère que les lignes où les cellules de l'attribut choisi sont identiques.
Ligne 9 : on copie entièrement la ligne de table1.
Ligne 10 : on copie la ligne de table2 sans répéter la cellule de jointure.

À NOTER

En terminale, vous découvrirez la gestion des bases de données relationnelles, notamment à l'aide du langage SQL. Dans ce langage, la jointure donnée en exemple s'écrira :

```
SELECT Nom
FROM Table1 JOIN Table2
ON Table1.Nom = Table2.Nom
```

A faire vous même 3.

- Téléchargez le fichier `base2.csv` :
  http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/base2.csv
- Reprenez la fonction `jointure` ci-dessus et testez-la avec `base.csv` et `base2.csv`

0.6 Panda – Un module python pour aller plus loin dans la manipulation des fichiers csv

Pour traiter des données, nous allons utiliser la bibliothèque Python Pandas. Une bibliothèque Python permet de rajouter des fonctionnalités par rapport au langage de base. La bibliothèque Pandas est donc très utilisée pour tout ce qui touche au traitement des données.
Pour nos premiers pas avec Pandas, nous allons utiliser des données très simples au format CSV : ces données sont contenues dans le fichier `ident_virgule.csv` .

A faire vous même 4.

- Téléchargez le fichier `ident_virgule.csv` :
  http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/ident_virgule.csv
- Saisissez et exécutez le code Python suivant :

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
```

- Nous créons une variable `iden` qui va contenir les données du fichier csv
- Tapez alors `iden` dans la console python pour voir apparaître les données contenues dans `iden`

```
In [3]: iden
Out[3]:
      nom       prenom date_naissance
0  Durand  Jean-Pierre     23/05/1985
1  Dupont   Christophe     15/12/1967
2   Terta        Henry     12/06/1978
```

Il est possible de récupérer certaines données du tableau, par exemple, certaines lignes, certaines colonnes ou bien encore des valeurs uniques. Le principe de fonctionnement de `loc` est relativement simple puisque l'on aura une instruction de la forme
`loc[index_ligne,index_colonne]`

A faire vous même 5.

- Complétez avec la ligne suivante et testez le programme :

```
info=iden.loc[1,'prenom']
```

- Vérifiez que la variable `info` contient bien le prénom `christophe`
- Modifiez le programme pour que la variable `info` contienne `12/06/1978`

Il est possible de récupérer plusieurs ou toutes les lignes d'une colonne, il suffit de remplacer la partie `index_ligne` de `loc` par `:`
Il est possible de récupérer toutes les colonnes d'une ligne particulière, cette fois en remplaçant la partie `index_colonne` de `loc` par `:`

A faire vous même 6.

- Modifiez la dernière ligne et testez :

```
info=iden.loc[:,'nom']
```

- Vérifiez que la variable `info` contient bien toutes les données de la colonne d'index `nom`, autrement dit, tous les noms
- Modifiez la dernière ligne et testez :

```
info=iden.loc[2,:]
```

- Vérifiez que la variable `info` contient bien toutes les données de la dernière ligne (index 2)

Il est aussi possible de récupérer seulement certaines lignes et certaines colonnes en utilisant la notation suivante :

```
loc[[index_ligne_1,index_ligne_2,...],[index_colonne_1,index_colonne_2,...]]
```

A faire vous même 7.

- Modifiez la dernière ligne et testez :

```
info=iden.loc[[0,1],['nom','date_naissance']]
```

- Vérifiez que la variable `info` contient bien un tableau avec uniquement les colonnes `nom` et `date_naissance` de la 1ère ligne (index 0) et de la 2e ligne (index 1).

Nous allons maintenant introduire des conditions dans la sélection de villes. Imaginez par exemple que vous désirez obtenir un tableau contenant toutes les villes de France qui ont une altitude minimum supérieure à 1500 m

A faire vous même 8. Pour les plus rapides

- Téléchargez `villes_virgule.csv` : http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/villes_virgule.csv
- Saisissez et testez :

```
import pandas
info_villes=pandas.read_csv("villes_virgule.csv")
nom_alt=info_villes.loc[info_villes["alt_min"]>1500,["nom","alt_min"]]
```

- Dans le `loc`, l'expression `info_villes["alt_min"]>1500` est bien avant la virgule, elle concerne donc les index des lignes du tableau. On sélectionnera uniquement les lignes qui auront la valeur du descripteur `alt_min` supérieure à `1500`.
- Modifiez le programme qui permettra d'avoir les villes qui ont une densité d'habitant inférieure à 50 (on aura 3 colonnes : le nom de la ville, la densité de la population et l'altitude minimum)

A faire vous même 9. Pour les plus rapides

- Il est possible de combiner plusieurs facteurs de sélection en utilisant un "et"("&") ou un "ou"("|").
- Modifiez la dernière ligne et testez :

```
nom_alt=info_villes.loc[(info_villes["alt_min"]>1500) & \
                        (info_villes["dens"]>50),["nom","dens","alt_min"]]
```

- Il est aussi possible d'effectuer des calculs sur des colonnes, par exemple des moyennes. Il suffit d'utiliser l'instruction `mean` pour effectuer une moyenne
- Il est aussi possible de trier le tableau en fonction des valeurs d'un descripteur. Il suffit d'utiliser l'instruction `sort_values`
- Il est aussi possible de trier par ordre décroissant en ajoutant `ascending=False` :

Il est possible de fusionner 2 tableaux de données qui ont une colonne commune :
Afin de travailler sur cette fusion, nous allons travailler avec 2 fichiers au format CSV :
`fiches_client.csv` : http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/fiches_client.csv
et `fiches_com.csv`. : http://ninoo.fr/LC/Term_NSI/seq4_langage_sql/fiches_com.csv

A faire vous même 10.

- Après avoir téléchargé les 2 fichiers ci-dessus, testez le code suivant :

```
import pandas
client=pandas.read_csv("fiches_client.csv")
commande=pandas.read_csv("fiches_com.csv")
```

- Utilisez l'explorateur de variables de Spyder afin d'afficher le contenu des variables `client` et `commande`
- Testez le code suivant :

```
import pandas
client=pandas.read_csv("fiches_client.csv")
commande=pandas.read_csv("fiches_com.csv")
cl_com=pandas.merge(client, commande)
```

- Utilisez l'explorateur de variables afin d'afficher le contenu des variables

1 Introduction à la notion de bases de données

Dans une base de données, l'information est stockée dans des fichiers, mais il n'est pas possible de travailler sur ces données avec un simple éditeur de texte. Pour manipuler les données présentes dans une base de données (écrire, lire ou encore modifier), il est nécessaire d'utiliser un type de logiciel appelé "système de gestion de base de données" très souvent abrégé en SGBD. Il existe une multitude de SGBD : des libres et gratuites (mySQL, PostgreSQL, SQLight, ...), des payantes et propriétaires (Ms Access, Ms SQLserver, ...).

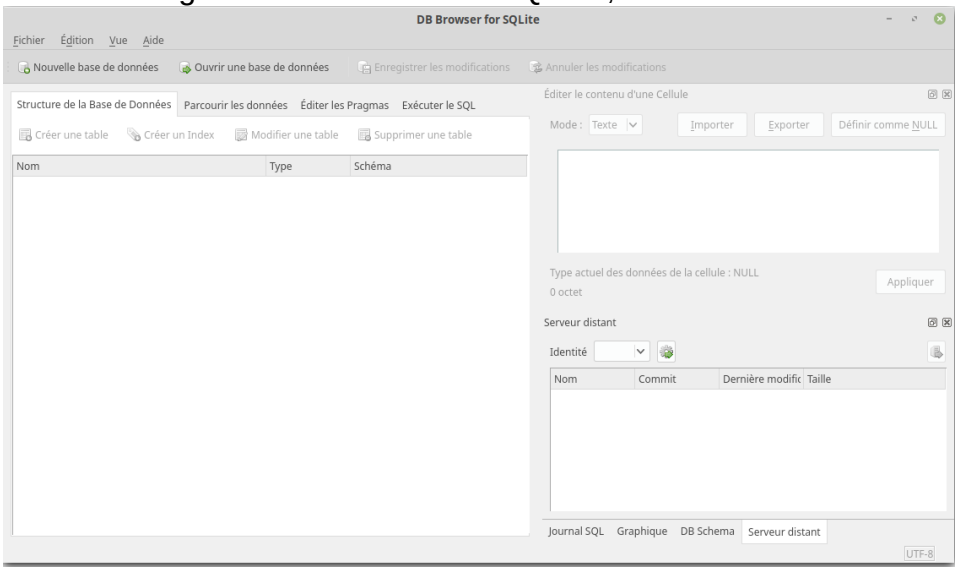
- les SGBD permettent de gérer la lecture, l'écriture ou la modification des informations
- les SGBD permettent de gérer les autorisations d'accès à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur A de lire et d'écrire alors que l'utilisateur B aura uniquement la possibilité de lire les informations
- les fichiers des bases de données sont stockés sur des disques durs dans des ordinateurs, ces ordinateurs peuvent subir des pannes. Il est souvent nécessaire que l'accès aux informations contenues dans une base de données soit maintenu, même en cas de panne matérielle. Les bases de données sont donc dupliquées sur plusieurs ordinateurs afin qu'en cas de panne d'un ordinateur A, un ordinateur B contenant une copie de la base de données présente dans A, puisse prendre le relais. Tout cela est très complexe à gérer, en effet toute modification de la base de données présente sur l'ordinateur A doit entraîner la même modification de la base de données présente sur l'ordinateur B. Cette synchronisation entre A et B doit se faire le plus rapidement possible, il est fondamental d'avoir des copies parfaitement identiques en permanence. C'est aussi les SGBD qui assurent la maintenance des différentes copies de la base de données.
- plusieurs personnes peuvent avoir besoin d'accéder aux informations contenues dans une base données en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent). Ces problèmes d'accès concurrent sont aussi gérés par les SGBD.

2 Les bases de données et le langage SQL

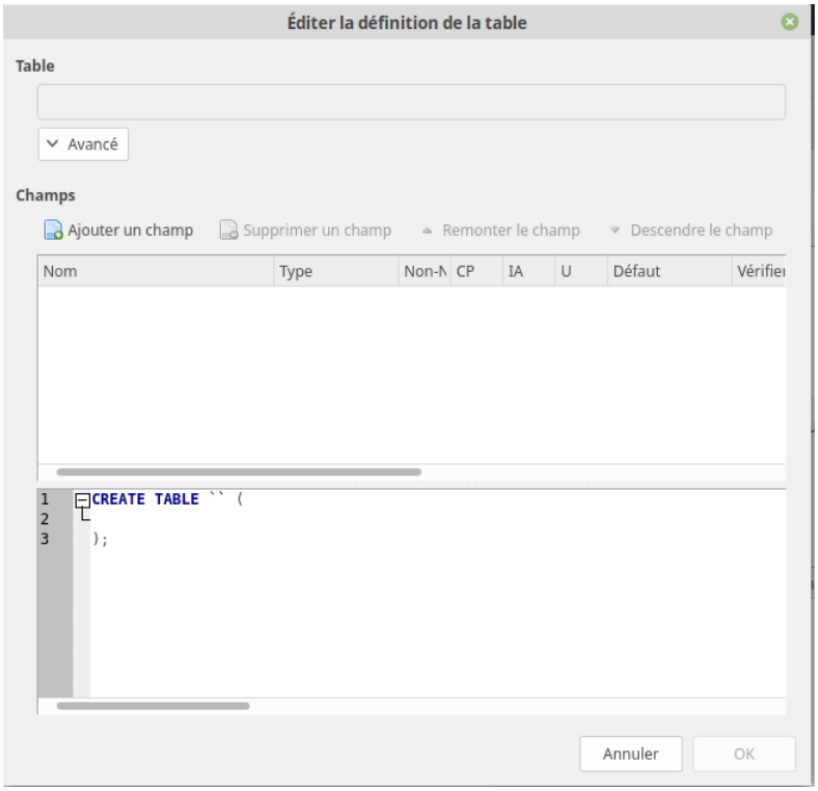
Nous allons maintenant apprendre à réaliser des requêtes, c'est-à-dire que nous allons apprendre à créer une base des données, créer des attributs, ajouter de données, modifier des données et enfin, nous allons surtout apprendre à interroger une base de données afin d'obtenir des informations.
Pour réaliser toutes ces requêtes, nous allons devoir apprendre un langage de requêtes : SQL (Structured Query Language). SQL est propre aux bases de données relationnelles, les autres types de bases de données utilisent d'autres langages pour effectuer des requêtes.
Pour créer une base de données et effectuer des requêtes sur cette dernière, nous allons utiliser le logiciel "DB Browser for SQLite" : https://sqlitebrowser.org/.
SQLite est un système de gestion de base de données relationnelle très répandu.
Nous allons commencer par créer notre base de données :

A faire vous même 11.

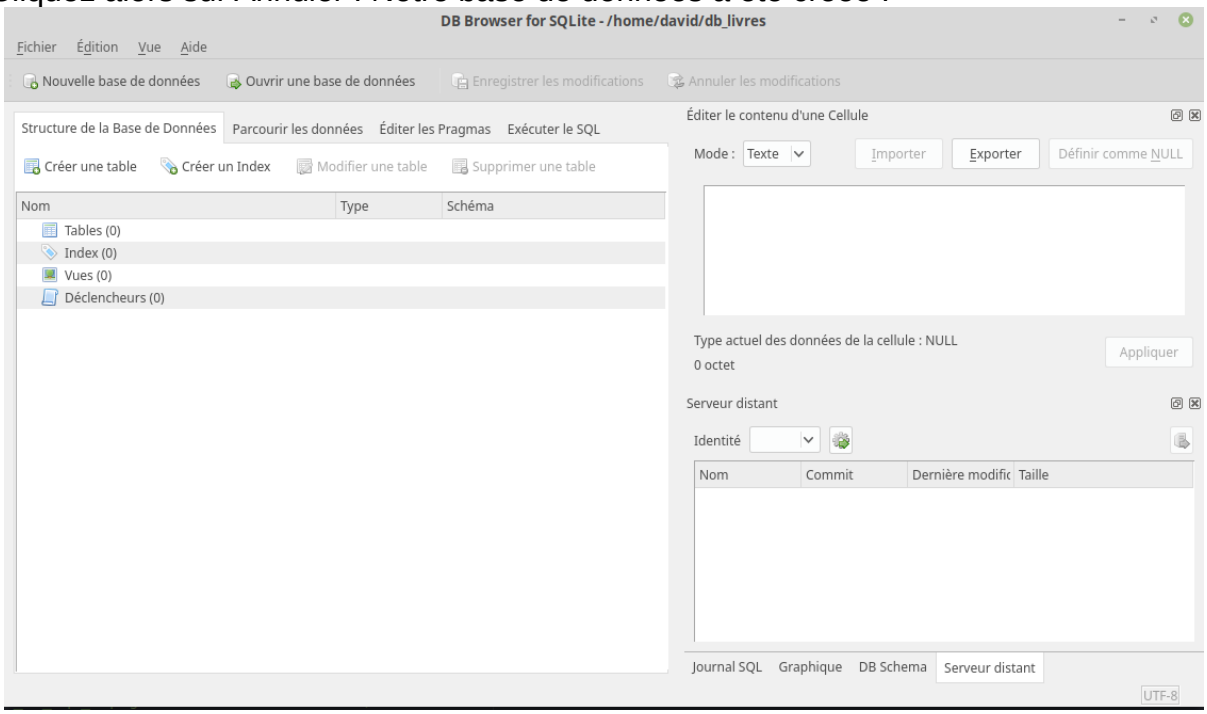
- Après avoir lancé le logiciel "DB Browser for SQLite", vous devriez obtenir ceci :



- Cliquez sur Nouvelle base de données. Après avoir choisi un nom pour votre base de données (par exemple `db_livres.db`), vous devriez avoir la fenêtre suivante :



- Cliquez alors sur Annuler . Notre base de données a été créée :



- Pour créer une table, cliquez sur l'onglet `Exécuter le SQL`.
- Copiez-collez le texte ci-dessous dans la fenêtre `SQL 1`

```
CREATE TABLE LIVRES
(id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT);
```

- Cliquez ensuite sur le petit triangle situé au-dessus de la fenêtre `SQL 1` (ou appuyez sur F5)
- Dans la fenêtre, vous devriez trouver "Requête exécutée avec succès" !

Vous venez de créer votre première table.

A faire vous même 12.

Nous créons ensuite les attributs : `id, titre, auteur, ann_pulbi, note`
Nous avons pour chaque attribut précisé son domaine : `id` : entier (INT), `titre` : chaîne de caractères (TEXT), `auteur` : chaîne de caractères, `ann_publi` : entier et `note` : entier
L'attribut `id` va jouer ici le rôle de clé primaire. On peut aussi, par souci de sécurité (afin d'éviter que

l'on utilise 2 fois la même valeur pour l'attribut `id`), modifier le l'instruction SQL vue ci-dessus, afin de préciser que l'attribut `id` est bien notre clé primaire :

```
CREATE TABLE LIVRES
       (id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT, PRIMARY KEY (id));
```

A faire vous même 13.

Toujours dans l'onglet `Exécuter le SQL`, après avoir effacé la fenêtre `SQL 1`, copiez-collez dans cette même fenêtre la requête ci-dessous :

```
INSERT INTO LIVRES
(id,titre,auteur,ann_publi,note)
VALUES
(1,'1984','Orwell',1949,10),
(2,'Dune','Herbert',1965,8),
(3,'Fondation','Asimov',1951,9),
(4,'Le meilleur des mondes','Huxley',1931,7),
(5,'Fahrenheit 451','Bradbury',1953,7),
(6,'Ubik','K.Dick',1969,9),
(7,'Chroniques martiennes','Bradbury',1950,8),
(8,'La nuit des temps','Barjavel',1968,7),
(9,'Blade Runner','K.Dick',1968,8),
(10,'Les Robots','Asimov',1950,9),
(11,'La Planète des singes','Boulle',1963,8),
(12,'Ravage','Barjavel',1943,8),
(13,'Le Maître du Haut Château','K.Dick',1962,8),
(14,'Le monde des Ā','Van Vogt',1945,7),
(15,'La Fin de l'éternité','Asimov',1955,8),
(16,'De la Terre à la Lune','Verne',1865,10);
```

Ici aussi, aucun problème, la requête a bien été exécutée :
La table LIVRES contient bien les données souhaitées (onglet `Parcourir les données`)

A faire vous même 14.

Nous allons apprendre à effectuer des requêtes d'interrogation sur la base de données que nous venons de créer.
Toutes les requêtes se feront dans la fenêtre `SQL 1` de l'onglet `Exécuter le SQL`

- Saisissez la requête SQL suivante :

```
SELECT id, titre, auteur, ann_publi, note
FROM LIVRES ;
```

- Puis appuyez sur le triangle (ou la touche F5)

A faire vous même 15.

Comme vous pouvez le constater, notre requête SQL a permis d'afficher tous les livres. Nous avons ici 2 mots clés du langage SQL *SELECT* qui permet de sélectionner les attributs qui devront être "affichés" (je mets "affichés" entre guillemets, car le but d'une requête sql n'est pas forcément d'afficher les données) et *FROM* qui indique la table qui doit être utilisée.
Il est évidemment possible d'afficher seulement certains attributs (ou même un seul).

- Saisissez la requête SQL suivante :

```
SELECT titre, auteur
FROM LIVRES ;
```

- Vérifiez que vous obtenez bien uniquement les titres et les auteurs des livres

A faire vous même 16.

- Écrivez et testez une requête permettant d'obtenir uniquement les titres des livres.

N.B. Si vous désirez sélectionner tous les attributs, vous pouvez écrire :

```
SELECT *
FROM LIVRES ;
```

à la place de :

```
SELECT id, titre, auteur, ann_publi, note
FROM LIVRES ;
```

Pour l'instant nos requêtes affichent tous les livres, il est possible d'utiliser la clause WHERE afin d'imposer une (ou des) condition(s) permettant de sélectionner uniquement certaines lignes.
La condition doit suivre le mot-clé `WHERE` :

A faire vous même 17.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre, ann_publi
FROM LIVRES
```

```
WHERE auteur='Asimov' ;
```

- Vérifiez que vous obtenez bien uniquement les livres écrits par Isaac Asimov.

A faire vous même 18.

- Écrivez et testez une requête permettant d'obtenir uniquement les titres des livres écrits par Philip K.Dick.

.....
.....
.....
.....

A faire vous même 19.

Il est possible de combiner les conditions à l'aide d'un `OR` ou d'un `AND`

- Saisissez et testez la requête SQL suivante :

```
SELECT titre, ann_publi
FROM LIVRES
WHERE auteur='Asimov' AND ann_publi>1953 ;
```

Vérifiez que nous obtenons bien le livre écrit par Asimov publié après 1953 (comme vous l'avez sans doute remarqué, il est possible d'utiliser les opérateurs d'inégalités).

A faire vous même 20.

D'après vous, quel est le résultat de cette requête :

```
SELECT titre
FROM LIVRES
WHERE auteur='K.Dick' OR note>=8 ;
```

.....
.....
.....

A faire vous même 21.

- Écrivez une requête permettant d'obtenir les titres des livres publiés après 1945 qui ont une note supérieure ou égale à 9.

.....
.....
.....
.....

Il est aussi possible de rajouter la clause SQL `ORDER BY` afin d'obtenir les résultats classés dans un ordre précis.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre
FROM LIVRES
WHERE auteur='K.Dick' ORDER BY ann_publi ;
```

Nous obtenons les livres de K.Dick classés du plus ancien ou plus récent.
Il est possible d'obtenir un classement en sens inverse à l'aide de la clause `DESC`

A faire vous même 22.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre
FROM LIVRES
WHERE auteur='K.Dick' ORDER BY ann_publi DESC ;
```

Nous obtenons les livres de K.Dick classés du plus récent au plus ancien.
Que se passe-t-il quand la clause `ORDER BY` porte sur un attribut de type TEXT ?

.....
.....

A faire vous même 23.

- Vous pouvez constater qu'une requête du type :

```
SELECT auteur
FROM LIVRES ;
```

affiche plusieurs fois certains auteurs (les auteurs qui ont écrit plusieurs livres présents dans la base de données)
Il est possible d'éviter les doublons grâce à la clause `DISTINCT`

A faire vous même 24. À faire vous-même 14

- Saisissez et testez la requête SQL suivante :

```
SELECT DISTINCT auteur
FROM LIVRES ;
```

Nous avons vu précédemment qu'une base de données peut contenir plusieurs relations (plusieurs tables).

A faire vous même 25.

- Créez une nouvelle base de données que vous nommerez par exemple `db_livres_auteurs.db`

A faire vous même 26.

- Créez une table AUTEURS à l'aide de la requête SQL suivante :

```
CREATE TABLE AUTEURS
(id INT, nom TEXT, prenom TEXT, ann_naissance INT, langue_ecriture
TEXT, PRIMARY KEY (id));
```

A faire vous même 27.

- Créez une table LIVRES à l'aide de la requête SQL suivante :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY KEY (id));
```

A faire vous même 28.

- Ajoutez des données à la table AUTEURS à l'aide de la requête SQL suivante :

```
INSERT INTO AUTEURS
(id,nom,prenom,ann_naissance,langue_ecriture)
VALUES
(1,'Orwell','George',1903,'anglais'),
(2,'Herbert','Frank',1920,'anglais'),
(3,'Asimov','Isaac',1920,'anglais'),
(4,'Huxley','Aldous',1894,'anglais'),
(5,'Bradbury','Ray',1920,'anglais'),
(6,'K.Dick','Philip',1928,'anglais'),
(7,'Barjavel','René',1911,'français'),
(8,'Boulle','Pierre',1912,'français'),
(9,'Van Vogt','Alfred Elton',1912,'anglais'),
(10,'Verne','Jules',1828,'français');
```

A faire vous même 29.

- Ajoutez des données à la table LIVRES à l'aide de la requête SQL suivante :

```
INSERT INTO LIVRES
(id,titre,id_auteur,ann_publi,note)
VALUES
(1,'1984',1,1949,10),
(2,'Dune',2,1965,8),
(3,'Fondation',3,1951,9),
(4,'Le meilleur des mondes',4,1931,7),
(5,'Fahrenheit 451',5,1953,7),
(6,'Ubik',6,1969,9),
```

```
  (7,'Chroniques martiennes',5,1950,8),
  (8,'La nuit des temps',7,1968,7),
  (9,'Blade Runner',6,1968,8),
  (10,'Les Robots',3,1950,9),
  (11,'La Planète des singes',8,1963,8),
  (12,'Ravage',7,1943,8),
  (13,'Le Maître du Haut Château',6,1962,8),
  (14,'Le monde des Ã',9,1945,7),
  (15,'La Fin de l'éternité',3,1955,8),
  (16,'De la Terre à la Lune',10,1865,10);
```

Nous avons 2 tables, grâce aux jointures nous allons pouvoir associer ces 2 tables dans une même requête.
En général, les jointures consistent à associer des lignes de 2 tables. Elles permettent d'établir un lien entre 2 tables. Qui dit lien entre 2 tables dit souvent clef étrangère et clef primaire.
Dans notre exemple l'attribut `id_auteur` de la tables LIVRES est bien une clé étrangère puisque cet attribut correspond à l'attribut `id` de la table "AUTEURS".
À noter qu'il est possible de préciser au moment de la création d'une table qu'un attribut jouera le rôle de clé étrangère. Dans notre exemple, à la place d'écrire :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY KEY (id));
```

On pourra écrire :

```
  CREATE TABLE LIVRES
  (id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY
KEY (id), FOREIGN KEY (id_auteur) REFERENCES AUTEURS(id));
```

grâce à cette précision, sqlite sera capable de détecter les anomalies au niveau de clé étrangère : essayez par exemple d'ajouter un livre à la table LIVRES avec l'attribut `id_auteur` égal à 11 !
Passons maintenant aux jointures :

A faire vous même 30.

- Saisissez et testez la requête SQL suivante :

```
SELECT *
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id ;
```

Le `FROM LIVRES INNER JOIN AUTEURS` permet de créer une jointure entre les tables LIVRES et AUTEURS ("rassembler" les tables LIVRES et AUTEURS en une seule grande table). Le `ON LIVRES.id_auteur = AUTEURS.id` signifie qu'une ligne quelconque A de la table LIVRES devra être fusionnée avec la ligne B de la table AUTEURS à condition que l'attribut `id` de la ligne A soit égal à l'attribut `id_auteur` de la ligne B.
Par exemple, la ligne 1 (id=1) de la table LIVRES (que l'on nommera dans la suite ligne A) sera fusionnée avec la ligne 1 (id=1) de la table AUTEURS (que l'on nommera dans la suite B) car l'attribut `id_auteur` de la ligne A est égal à 1 et l'attribut `id` de la ligne B est aussi égal à 1.
Autre exemple, la ligne 1 (id=1) de la table LIVRES (que l'on nommera dans la suite ligne A) ne sera pas fusionnée avec la ligne 2 (id=2) de la table AUTEURS (que l'on nommera dans la suite B') car l'attribut `id_auteur` de la ligne A est égal à 1 alors que l'attribut `id` de la ligne B' est égal à 2.
Cette notion de jointure n'est pas évidente, prenez votre temps pour bien réfléchir et surtout n'hésitez pas à poser des questions.

A faire vous même 31.

- Saisissez et testez la requête SQL suivante :

```
SELECT *
FROM AUTEURS
INNER JOIN LIVRES ON LIVRES.id_auteur = AUTEURS.id ;
```

Comme vous pouvez le constater, le résultat est différent, cette fois-ci ce sont les lignes de la table LIVRES qui viennent se greffer sur la table AUTEURS.
Dans le cas d'une jointure, il est tout à fait possible de sélectionner certains attributs et pas d'autres :

A faire vous même 32.

- Saisissez et testez la requête SQL suivante :

```
SELECT nom, prenom, titre
FROM AUTEURS
INNER JOIN LIVRES ON LIVRES.id_auteur = AUTEURS.id ;
```

A faire vous même 33.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id ;
```

Si un même nom d'attribut est présent dans les 2 tables (par exemple ici l'attribut id), il est nécessaire d'ajouter le nom de la table devant afin de pouvoir les distinguer (AUTEURS.id et LIVRES.id)

A faire vous même 34.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre,AUTEURS.id,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id ;
```

Il est possible d'utiliser la clause WHERE dans le cas d'une jointure :

A faire vous même 35.

- Saisissez et testez la requête SQL suivante :

```
SELECT titre,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id
WHERE ann_publi>1950 ;
```

Enfin, pour terminer avec les jointures, vous devez savoir que nous avons abordé la jointure la plus simple (`INNER JOIN`). Il existe des jointures plus complexes (`CROSS JOIN`, `LEFT JOIN`, `RIGHT JOIN`), ces autres jointures ne seront pas abordées ici.
Nous en avons terminé avec les requêtes d'interrogation, intéressons-nous maintenant aux requêtes de mise à jour (`INSERT`, `UPDATE`, `DELETE`).
Nous allons repartir avec une base de données qui contient une seule table :

A faire vous même 36.

- Créez une nouvelle base de données que vous nommerez par exemple `db_livres.db`

A faire vous même 37.

Créez une table LIVRES à l'aide de la requête SQL suivante :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT, PRIMARY
KEY (id));
```

A faire vous même 38.

Ajoutez des données à la table LIVRES à l'aide de la requête SQL suivante :

```
INSERT INTO LIVRES
(id,titre,auteur,ann_publi,note)
VALUES
(1,'1984','Orwell',1949,10),
(2,'Dune','Herbert',1965,8),
(3,'Fondation','Asimov',1951,9),
(4,'Le meilleur des mondes','Huxley',1931,7),
(5,'Fahrenheit 451','Bradbury',1953,7),
(6,'Ubik','K.Dick',1969,9),
```

```
(7,'Chroniques martiennes','Bradbury',1950,8),
(8,'La nuit des temps','Barjavel',1968,7),
(9,'Blade Runner','K.Dick',1968,8),
(10,'Les Robots','Asimov',1950,9),
(11,'La Planète des singes','Boulle',1963,8),
(12,'Ravage','Barjavel',1943,8),
(13,'Le Maître du Haut Château','K.Dick',1962,8),
(14,'Le monde des Ã','Van Vogt',1945,7),
(15,'La Fin de l'éternité','Asimov',1955,8),
(16,'De la Terre à la Lune','Verne',1865,10);
```

Nous avons déjà eu l'occasion de voir la requête permettant d'ajouter une entrée (utilisation d'INSERT)

A faire vous même 39.

- Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête `SELECT * FROM LIVRES`.

```
INSERT INTO LIVRES
(id,titre,auteur,ann_publi,note)
VALUES
(17,'Hypérion','Simmons',1989,8);
```

A faire vous même 40.

- Écrivez et testez une requête permettant d'ajouter le livre de votre choix à la table LIVRES.

`UPDATE` va permettre de modifier une ou des entrées. Nous utiliserons `WHERE`, comme dans le cas d'un `SELECT`, pour spécifier les entrées à modifier.

.....

.....

A faire vous même 41.

- Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête `SELECT * FROM LIVRES`.

```
UPDATE LIVRES
SET note=7
WHERE titre = 'Hypérion' ;
```

A faire vous même 42.

- Écrivez une requête permettant d'attribuer la note de 10 à tous les livres écrits par Asimov publiés après 1950. Testez cette requête.

`DELETE` est utilisée pour effectuer la suppression d'une (ou de plusieurs) entrée(s). Ici aussi c'est le `WHERE` qui permettra de sélectionner les entrées à supprimer.

.....

.....

A faire vous même 43.

- Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête `SELECT * FROM LIVRES`.

```
DELETE FROM LIVRES
WHERE titre='Hypérion' ;
```

A faire vous même 44.

- Écrivez une requête permettant de supprimer les livres publiés avant 1945. Testez cette requête.

.....

.....

Lire livre Prepabac P. 206-212

P. 214 ex 1

P. 214 ex 2

P. 214 ex 3

Cet exercice porte sur les bases de données relationnelles et le langage SQL.

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

La ligue féminine de basket-ball publie les données relatives à chaque saison sur le site web de la ligue. On y retrouve des informations concernant les équipes participantes, les calendriers et les résultats des matchs ainsi que les statistiques des joueuses. Dans cet exercice, nous allons nous intéresser à la base de données relationnelle LFB_2021_2022 permettant le stockage et la gestion des données de la saison régulière de basket-ball féminin 2021-2022.

1. Voici ci-dessous le contenu entier de la relation (table) `Equipe` :

id_equipe	nom	adresse	telephone
1	Saint-Amand	39 avenue du Clos, 59230 Saint-Amand-les-Eaux	03 04 05 06 07
2	Basket Landes	15 place Saint-Roch, 40000 Mont-De-Marsan	05 06 07 08 09
3	Villeneuve d'Ascq	2 rue Breughel, 59650 Villeneuve-d'Ascq	03 02 01 00 01
4	Tarbe	Quai de l'Adour, 65000 Tarbes	05 04 03 02 02
5	Lyon	451 cours Emile Zola, 69100 Villeurbanne	04 05 06 07 08
6	Bourges	6 rue du Pré Doulet, 18000 Bourges	02 03 04 05 06
7	Charleville-Mézières	Rue de la Vieille Meuse, 08000 Charleville-Mézières	03 05 07 09 01
8	Landerneau	Kerouel, 29410 Pleyber-Christ	02 04 06 08 00
9	Angers	330 rue Saint-Léonard, 49000 Angers	02 00 08 06 04
10	Lattes Montpellier	157 rue de la Porte Lombarde, 34970 Lattes	04 03 02 01 00
11	Charnay	Allée des Ecoliers, 71850 Charnay-lès-Mâcon	03 01 09 07 05
12	Roche Vendée	BP 151, 85004 La Roche-Sur-Yon Cedex	02 05 08 01 04

On donne ci-contre le schéma relationnel de la table `Equipe`.
Dans ce schéma, un attribut souligné indique qu'il s'agit d'une clé primaire.

Equipe	
id_equipe	INT
nom	VARCHAR(50)
adresse	VARCHAR(100)
telephone	VARCHAR(20)

a. Après le chargement de la table `Equipe`, expliquer pourquoi la requête suivante produit une erreur :

```
INSERT INTO Equipe
VALUES (11,"Toulouse","2 rue du Nord,40100 Dax","05 04 03 02 01");
```

b. Expliquer le choix du domaine pour l'attribut `telephone`.

c. Donner le résultat de la requête suivante :

```
SELECT nom, adresse, telephone FROM Equipe WHERE id_equipe = 5;
```

d. Donner et expliquer le résultat de la requête suivante :

```
SELECT COUNT(*) FROM Equipe;
```

e. Écrire la requête SQL permettant d'afficher les noms des équipes par ordre alphabétique.

f. Écrire la requête SQL permettant de corriger le nom de l'équipe dont l'`id_equipe` est égal à 4. Le nom correct est "Tarbes".

2. Sur le site web de la fédération de basket-ball féminin, nous pouvons consulter la composition des équipes. Pour chaque joueuse, on peut y lire en plus de son nom, sa date de naissance, sa taille ainsi que le poste occupé dans l'équipe. Ces informations sont présentées dans une page web dont le titre est « Fiche Joueuse », page construite à partir de la table `Joueuse` dont voici un extrait :

id_joueuse	nom	prenom	date_naissance	taille	poste	id_equipe
1	Berkani	Lisa	19/05/1997	176	2	7
2	Alexander	Kayla	05/01/1991	193	5	5
3	Magarity	Regan	30/04/1996	192	4	2
4	Muzet	Johanna	08/07/1997	183	3	11
5	Kalu	Ezinne	26/06/1992	173	2	8
6	Sigmundova	Jodie Cornelie	20/04/1993	193	5	9
7	Dumerc	Céline	09/07/1982	162	2	2
8	Slonjsak	Iva	16/04/1997	183	3	9
9	Michel	Sarah	10/01/1989	180	2	6
10	Lithard	Pauline	11/02/1994	164	1	1

On donne ci-contre le schéma relationnel de la table `Joueuse`.

Un attribut souligné indique qu'il s'agit d'une clé primaire. Le symbole # devant un attribut indique qu'il s'agit d'une clé étrangère.

La clé étrangère `Joueuse.id_equipe` fait référence à la clé primaire `Equipe.id_equipe` de la table `Equipe`.

Joueuse	
id_joueuse	INT
nom	VARCHAR(50)
prenom	VARCHAR(50)
date_naissance	DATE
taille	INT
poste	INT
#id_equipe	INT

a. Expliquer pourquoi l'attribut `id_equipe` a été déclaré clé étrangère.

b. On souhaite supprimer toutes les informations relatives à une équipe. Expliquer pourquoi on ne peut pas directement supprimer cette équipe dans la table `Equipe`.

c. Écrire la requête SQL qui permet d'afficher les noms et les prénoms des joueuses de l'équipe d'Angers par ordre alphabétique des noms. On supposera que l'utilisateur qui écrit cette requête ne connaît pas l'identifiant de l'équipe d'Angers.

3. Les résultats des matchs sont aussi publiés sur le site web de la ligue. Par exemple, pour le match n°10 qui a opposé l'équipe de Villeneuve d'Ascq à l'équipe de Bourges le 23/10/2021 on retrouve les informations suivantes :

Match n° 10

23/10/2021

Villeneuve d'Ascq 73 | 78 Bourges

Le score final du match a été de 73 points pour l'équipe de Villeneuve d'Ascq qui a joué à domicile (nom affiché à gauche sur la page) contre 78 points pour l'équipe de Bourges qui a joué en déplacement (nom affiché à droite sur la page).

a. À partir de l'analyse de cet exemple, proposer un schéma relationnel pour la table `Match`. Si des clés étrangères sont définies, préciser quelles tables et quels attributs elles référencent.

b. Écrire la requête SQL qui permet l'insertion dans la table `Match` de l'enregistrement correspondant à l'exemple donné ci-dessus.

4. En plus du score final, sur la page web sont affichées des informations relatives aux performances des joueuses pendant le match.
Nous allons retenir ici seulement 3 critères : le nombre de points marqués, les rebonds et les passes décisives effectués.

Voici un extrait des statistiques du match n°53 qui a opposé l'équipe de Landerneau à celle de Charleville-Mézières le 16/04/2022 :

Match n° 53

16/04/2022

Landerneau 56 | 64 Charleville-Mézières

Extrait des statistiques :

Equipe	Nom	Prénom	Points	Rebonds	Passes décisives
Charleville-Mézières	Pouye	Tima	18	6	2
Charleville-Mézières	Akhator	Evelyn	15	17	0
Charleville-Mézières	Bouderra	Amel	10	3	9
Landerneau	Mane	Marie	18	2	3
Landerneau	Amukamara	Promise	12	2	5
Landerneau	Geiselsoder	Luisa	4	10	2

a. Proposer un schéma relationnel pour stocker les informations relatives aux statistiques des joueuses dans la base de données, telles que présentées ci-dessus.

b. Écrire la requête SQL qui a été utilisée pour afficher la partie « Extrait des statistiques » de l'exemple ci-dessus.