Bases de données (1/2) : écosystème et découverte de SQL

D'après un TP de Stéphane Gonnord .

Buts du TP

- Apprendre a accéder à une base de données, via une application graphique et via la bibliothèque `sqlite3`.
- Comparer recherche dans des fichiers avec Python et interrogation d'une base de données avec SQL
- Faire quelques requêtes élémentaires, à nouveau via une application graphique et depuis un script python.
- Commencer à s'habituer à la syntaxe SQL, écrire des requêtes avec des sélections, projections ou jointures de tables

EXERCICE 1 Copier/Coller le répertoire du TP récupéré dans le répertoire `Donnees` de la classe. Tous les fichiers nécessaires (bases de données, fichiers textes, images, scripts ...) se trouvent dans le sous-répertoire `materiel` :

`cinema843.db` `exo2PartieA-eleve.py` `personne.txt` `jouer.txt` `film.txt` `projection.txt` `cinema.txt.` `base_triangles.db` `communes-regions-departements.db`

1 Fichiers plats vs. Base de donnée : au cinéma

Où l'on fait des recherches dans des fichiers textes puis dans une base de données.

EXERCICE 2 *Cet exercice et l'exercice 6 s'inspirent d'un TP du livre Bases de données : de la modélisation au SQL de Laurent Audibert qui est édité chez Ellipses et d'un TP posé en 2013/2014 par M.Ridde en classe de 833.*

Lancer `sqliteman` (qui se trouve sous scribe dans le répertoire commun `T:`) ; charger la base de données `cinema843.db`, constituée de 5 tables dont le schéma relationnel est donné ci-dessous. Pour certaines tables, chaque ligne/enregistrement possède un identifiant unique (appelé clef primaire) qui est souligné.

- `personne(idp:integer, nom:text, prenom:text)`
- `jouer(ida:integer, idf:integer, role:texte)`
- `film(idf:integer, idr:integer, titre:text, genre:text, annee:integer)`
- `projection(idc:integer, idf:integer, jour:text)`
- `cinema(idc:integer, nom:text, adresse:text)`

Le fichier `cinema843.db` contient une instance de ce schéma relationnel qui est représentée dans le fichier `image-cinema`.
Par ailleurs les contenus des cinq tables sont stockés séparément dans les fichiers textes `personne.txt`, `jouer.txt`, `film.txt`, `projection.txt` et `cinema.txt`.
On va rechercher des informations similaires :

➟ d'abord par des recherches séquentielles dans les fichiers textes à l'aide du *langage impératif* `Python`

➟ puis par des requêtes sur la base de données exprimées dans le *langage déclaratif* `SQL`

Partie A : recherche dans des fichiers textes

1. Ouvrir le fichier `exo2PartieA-eleve.py` avec `Pyzo`, il contient déjà une fonction `lecture`.
   Expliquer son fonctionnement. Quel est le rôle de l'instruction `ligne.rstrip().split('\t')` ?


```
def lecture(fichier):
    f = open(fichier)
    collecteur = []
    for ligne in f:
       champs = tuple(ligne.rstrip().split('\t'))
       collecteur.append(champs)
    f.close()
    return collecteur
```

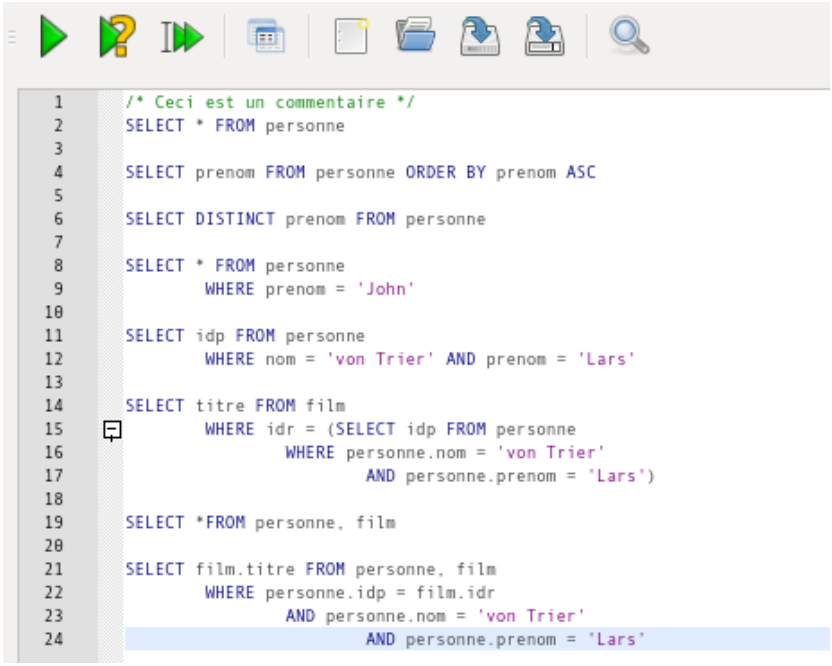
2. Avec cette fonction, récupérer dans des listes, les contenus des fichiers `personne.txt` et `film.txt`.
3. Écrire une fonction `requete1()` qui retourne la liste des personnes dont le prénom est John :

```
>>> requete1()
 [('05', 'Travolta', 'John'), ('12', 'Wayne', 'John'), ('18', 'Glen', 'John')]
```

4. Écrire une fonction `requete2()` qui retourne la liste de tous les prénoms dans l'ordre alphabétique.
   Écrire une fonction `requete3()` qui retourne la liste de tous les prénoms dans l'ordre alphabétique sans doublons.
5. Écrire une fonction `requete4()` qui retourne la liste de tous les films réalisés par Lars von Trier.

Partie B : requêtes sur une base de données

On utilise désormais `Sqliteman` pour interroger la base de données `cinema843.db` : les requêtes SQL seront écrites dans la fenêtre d'édition en haut à droite et les résultats s'afficheront en-dessous. La fenêtre de gauche affiche les différentes tables (ou relations) de la base de données.



```
1  /* Ceci est un commentaire */
2  SELECT * FROM personne
3
4  SELECT prenom FROM personne ORDER BY prenom ASC
5
6  SELECT DISTINCT prenom FROM personne
7
8  SELECT * FROM personne
9          WHERE prenom = 'John'
10
11 SELECT idp FROM personne
12         WHERE nom = 'von Trier' AND prenom = 'Lars'
13
14 SELECT titre FROM film
15         WHERE idr = (SELECT idp FROM personne
16                 WHERE personne.nom = 'von Trier'
17                         AND personne.prenom = 'Lars')
18
19 SELECT *FROM personne, film
20
21 SELECT film.titre FROM personne, film
22         WHERE personne.idp = film.idr
23                 AND personne.nom = 'von Trier'
24                         AND personne.prenom = 'Lars'
```

1. Créer et enregistrer un fichier de requêtes SQL nommé `exo2-requetes.sql`.
2. Saisir les requêtes de la capture d'écran précédente puis les exécuter une par une en les sélectionnant puis en appuyant sur F9.
   (a) Décrire l'objectif de chaque requête et comparer si possible avec une fonction Python écrite en Partie A.
   (b) Pour chaque requête, identifier les opérations de sélection et projection de l'algèbre relationnelle.
   (c) Quelle requête réalise un produit cartésien de deux tables ?
   (d) Quelle requête réalise une jointure entre deux tables sans utiliser l'opérateur `JOIN` ? Revenir à cette question dans l'exercice 5.
3. Ecrire des requêtes SQL pour :
   (a) Déterminer les titres de films dont le genre est *Drame*.
   (b) Déterminer les titres des films réalisés dans les années 80.
   (c) Déterminer le nombre de total projections de films (utiliser la fonction `COUNT`).
   (d) Déterminer les titres de films dont le genre est *Drame* et qui ont été réalisés dans les années 2000.
   (e) Déterminer les rôles joués par Kevin Spacey.

2 Clic-clic-clic vs. tip-tip-tip : des triangles

Où on apprend à consulter une base de données...

EXERCICE 3 Charger dans `sqliteman` la base de données `base_triangles.db`, constituée d'une seule table, dont le schéma relationnel est :

`triangle(idt:integer, ab:integer, ac:integer, bc:integer)`

Chaque enregistrement/ligne représente les longueurs d'un triangle ABC, ainsi qu'une *clé primaire*.
Créer et enregistrer un fichier de requêtes SQL nommé `exo3-requetes.sql`.

1. Lancer (successivement !) les requêtes SQL suivantes, en réfléchissant à leur signification/objectif :

```
SELECT COUNT(*) FROM triangles
SELECT * FROM triangles WHERE ab+ac+bc=100
SELECT ab*ac*bc FROM triangles WHERE ab+ac+bc>=100
```

2. Insérer deux lignes supplémentaires dans la table avec les requêtes SQL suivantes :

```
INSERT INTO triangles VALUES (100001,50, 35, 16);
INSERT INTO triangles VALUES (100001,35,50, 16);
```

3. Supprimer les deux lignes insérés avec l'opérateur DELETE .
4. Déterminer, à l'aide de requêtes SQL :
   (a) tous les triangles équilatéraux ;
   (b) tous les triangles rectangles en A ;
   (c) le nombre de tels triangles ;
   (d) le maximum des périmètres des triangles rectangles en A ;
   (e) la plus petite valeur des produits $AB.AC.BC$, pour les triangles ABC de périmètre supérieur ou égal à 100 ;

   (f) les longueurs correspondants au(x) triangle(s) pour le(s)quel(s) le minimum précédent est atteint (il faut utiliser la requête précédente comme sous-requête ...) ;
   (g) l'ensemble des triangles plats et placer les longueurs de ces triangles dans un fichier [1] (texte ou csv).

EXERCICE 4 Lancer `Pyzo`, sauvegarder immédiatement le fichier édité au bon endroit. Écrire une commande absurde, de type `print(5*3)` dans l'éditeur, sauvegarder et exécuter après un petit coup de F6.

1. Taper les lignes suivantes dans le script python. Exécuter.

```
import sqlite3

base = sqlite3.connect('base_triangles.db')
curseur = base.cursor()

res = curseur.execute("""SELECT * FROM triangles WHERE ab+bc+ac=100""")
foo = res.fetchall()

base.close()
```

2. Que vaut maintenant `res` ? Et `foo` ?

3 Bien faire la jointure

Où l'on découvre des jonctions de tables.

EXERCICE 5 Lancer `sqliteman` ; charger la base de données `communes-regions-departements.db`, constituée de trois tables appelées `communes`, `departements` et `regions`.
Créer et enregistrer un fichier de requêtes SQL nommé `exo5-requetes.sql`.

1. Écrire *sur papier* les schémas relationnels de ces tables.
2. Lancer la requête :

```
SELECT c.nom, d.nom
FROM communes AS c JOIN departements AS d
ON c.dep = d.id
```

   Cette requête nous a permis de joindre les deux tables `communes` et `departements`. Au passage, on a utilisé des alias (renommage). On aurait également pû écrire :

```
SELECT communes.nom, departements.nom
FROM communes JOIN departements
ON communes.dep = departements.id
```

3. En s'inspirant du modèle précédent, donner la liste de toutes les communes avec pour chacune, son département, sa région et sa population.
   La première ligne est :

```
L'Abergement-Clémenciat,Ain,Rhône-Alpes,784
```

4. (a) Donner la liste des villes de plus de 100000 habitants, ainsi que leur population et leur région.
   (b) Trier la liste précédente par ordre décroissant de la population. On pourra pour cela faire une recherche Google pour trouver la bonne fonction `SQL` à utiliser.

1. Il y a un bouton pour ça : *Export Data*, juste dessus les résultats.

5. Donner la liste des communes (nom et population) dont le nom commence par `Pa` et se finit par `is`. On pourra pour cela utiliser la commande `LIKE` (Google est encore votre ami).

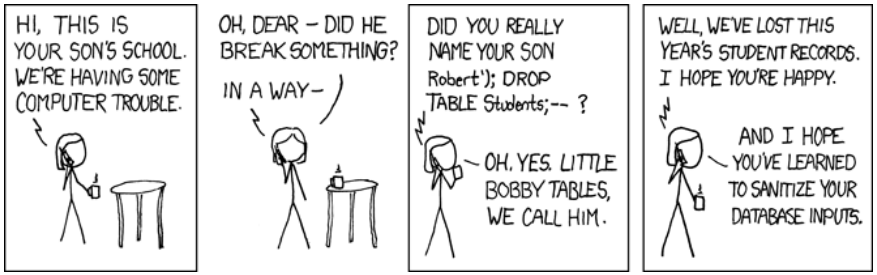
6. Déterminer les communes qui ont strictement plus de lettres dans leur nom que leur nombre d'habitants.

NB : une jointure peut également être réalisée via la syntaxe suivante (question B. 2. (d) de l'exo 2) :

```
SELECT c.nom, d.nom FROM communes AS c, departements AS d WHERE c.dep = d.id
```

EXERCICE 6 Lancer `sqliteman`; charger de nouveau la base de données `cinema843.db` qui a été présentée dans l'exercice 1.
Ecrire dans un fichier `exo6-requetes.sql` des requêtes SQL (avec jointures de tables) permettant de répondre aux questions suivantes :

1. Quels sont les titres des films réalisés par Lars von Trier?
2. Quels sont les titres des films où Kevin Spacey a joué?
3. Quels sont les drames que l'on a pu voir après le 1er janvier 2002?
4. Quels sont les noms et prénoms des personnes qui sont des acteurs? Ordonner la liste par ordre alphabétique croissant des noms.
5. Quels sont les noms et prénoms des personnes qui sont des réalisateurs?
6. Quels sont les noms et prénoms des personnes qui sont à la fois des acteurs et des réalisateurs?
   Faire l'intersection des tables générées par les deux requêtes précédentes à l'aide du mot clef `INTERSECT`.
7. Quels sont les noms et prénoms des acteurs qui ne sont pas des réalisateurs?
   Faire la différence entre deux tables à l'aide du mot clef `EXCEPT`.
8. Quels sont les noms et prénoms des réalisateurs qui ont réalisé des films policier?
9. Quels sont les noms et prénoms des réalisateurs qui ont réalisé des films d'épouvante et des films policier?
10. Quelle est la liste de toutes les interprétations possibles en précisant le nom et le prénom de l'acteur ainsi que le rôle et le titre du film? La liste doit être classée dans l'ordre alphabétique décroissant des noms.
11. Quels sont les titres des films réalisés par David Cronenberg qui ont été projetés au cinéma UGC?
12. Quels sont les acteurs qui ont joué dans des films projetés au cinéma UGC depuis l'an 2000?
13. Quels sont les titres des films où Stellan Skarsgard a joué un rôle et qui ont été projetés au cinéma UGC?



Corrigé du TP 14 Base de Données n° 1

Chenevois-Jouhet-Junier

1 Exercice 2 Partie A

```
def lecture(fichier):
    """Ouvre un fichier texte où chaque ligne est constituee de champs
    separes par des tabulations et se termine par un saut de ligne.
    Lit le fichier ligne par ligne et collecte les differents champs dans
    un tuple puis insere ce tuple dans une liste"""
    f = open(fichier)
    collecteur = []
    for ligne in f:
        champs = tuple(ligne.rstrip().split('\t'))
        collecteur.append(champs)
    f.close()
    return collecteur

film = lecture('film.txt')
jouer = lecture('jouer.txt')
projection = lecture('projection.txt')
personne = lecture('personne.txt')

"""
In [1]: film[0]
Out[1]: ('01', '15', 'Crash', 'Drame', '1996')

In [2]: jouer[0]
Out[2]: ('01', '05', 'Grace')

In [3]: projection[0]
Out[3]: ('02', '05', '01-05-2002')

In [4]: personne[0]
Out[4]: ('01', 'Kidman', 'Nicole')
"""

def requete1():
```

```
    """Retourne la liste des personnes dont le prenom est John"""
    t = []
    for p in personne:
        if p[-1] == 'John':
            t.append(p)
    return t

"""
In [8]: requete1()
Out[8]: [('05', 'Travolta', 'John'), ('12', 'Wayne', 'John'),
('18', 'Glen', 'John')]
"""

def requete2():
    """Retourne la liste de tous les prenoms dans l'ordre alphabetique"""
    prenom = []
    for p in personne:
        prenom.append(p[-1])
    prenom.sort()
    return prenom

"""
In [14]: requete2()
Out[14]:
['Angelina', 'Bruce',... ,'Stellan']
 """
def requete3():
    """Retourne la liste de tous les prenoms dans l'ordre alphabetique
    sans doublons"""
    prenom = requete2()
    newprenom = [prenom[0]]
    #on utilise le fait que prenom est deja dans l'ordre alphabetique
    for p in prenom[1:]:
        if p != newprenom[-1]:
            newprenom.append(p)
    return newprenom

"""
In [2]: len(requete2())
Out[2]: 22

In [3]: len(requete3())
Out[3]: 19
"""

def requete4():
    """Retourne la liste de tous les films realises par Lars Von trier"""
    #recherche de l'identifiant de Lars Von Trier
    for p in personne:
        if p[1] == 'von Trier':
```



```
            id = p[0]
            break
    filmTrier = []
    for f in film:
        if f[1] == id:
            filmTrier.append(f[2])
    return filmTrier

"""
In [17]: requete4()
Out[17]: ['Breaking the waves', 'Dogville']
"""
```

2 Exercice 2 Partie B Base : `cinema843.db`

Exo 2 : Toutes les lignes avec toutes les colonnes de la table personne

```
SELECT * FROM personne

(1, Kidman, Nicole)
(2, Bettany, Paul)
(3, Watson, Emily)
(4, Skarsgard, Stellan)
... (18 de plus)
```

Exo 2 : Tous les prenoms de la table personne dans l'ordre croissant des prenoms

```
SELECT prenom FROM personne ORDER BY prenom ASC

(Angelina)
(Bruce)
(Clint)
(David)
... (18 de plus)
```

Exo 2 : Tous les prenoms de la table personne avec supression des doublons. Les deux requetes donnent le meme resultat.

```
SELECT DISTINCT prenom FROM personne ORDER BY prenom ASC

SELECT DISTINCT prenom FROM personne

(Nicole)
(Paul)
(Emily)
```

personne		
idp	nom	prenom
1	Kidman	Nicole
2	Bettany	Paul
3	Watson	Emily
4	Skarsgard	Stellan
5	Travolta	John
6	L. Jackson	Samuel
7	Willis	Bruce
8	Irons	Jeremy
9	Spader	James
10	Hunter	Holly
11	Arquette	Rosanna
12	Wayne	John
13	von Trier	Lars
14	Tarantino	Quentin
15	Cronenberg	David
16	Mazursky	Paul
17	Jones	Grace
18	Glen	John
19	Eastwood	Clint
20	Spacey	Kevin
21	Mendes	Sam
22	Jolie	Angelina

film				
idf	idr	titre	genre	annee
1	15	Crash	Drame	1996
2	15	Faux-Semblants	Epouvante	1988
3	14	Pulp Fiction	Policier	1994
4	13	Breaking the waves	Drame	1996
5	13	Dogville	Drame	2002
6	12	Alamo	Western	1960
7	18	Dangereusement vôtre	Espionnage	1985
8	19	Chasseur blanc, coeur noir	Drame	1989
9	19	Minuit dans le jardin du bien et du mal	Policier	1998
10	21	American Beauty	Drame	1999
11	19	L'Echange	Drame	2008

jouer		
ida	idf	role
1	5	Grace
2	5	Tom Edison
3	4	Bess
4	4	Jan
5	3	Vincent Vega
6	3	Jules Winnfield
7	3	Butch Coolidge
8	2	Beverly & Elliot Mantle
9	1	James Ballard
10	1	Helen Remington
11	1	Gabrielle
4	5	Chuck
16	7	May Day
19	8	John Wilson
20	9	Jim Williams
20	10	Lester Burnham

projection		
idc	idf	jour
2	5	2002-05-01
2	5	2002-05-02
2	5	2002-05-03
2	4	1996-12-02
1	1	1996-05-07
2	7	1985-05-09
1	4	1996-08-02
4	3	1994-04-08
3	6	1990-12-02
2	2	1990-12-08
3	3	1994-11-05
4	3	1994-11-06
1	6	1980-07-05
2	4	1996-09-02
4	6	2002-08-01
3	6	1960-11-09
1	2	1988-03-12
2	8	1989-02-01
2	1	1997-05-11
2	3	1994-07-05
2	6	2002-08-01
1	3	1994-03-02
2	9	2008-12-02
1	10	2000-10-03
2	11	2004-03-02

cinema		
idc	nom	adresse
2	UGC	Part-Dieu
1	Pathé	Bellecour
3	Astoria	Cours Vitton
4	Comedia	Avenue Berthelot

Figure 1: Schéma relationnel de la table `cinema843.db`


```
(Stellan)
... (15 de plus)
```

Exo 2 : Toutes les lignes de la table personne qui contiennent le prenom ‘John’

```
SELECT * FROM personne WHERE prenom = 'John'
```

```
(5, Travolta, John)
(12, Wayne, John)
(18, Glen, John)
```

Exo 2 : Récupère l’idp de Lars von Trier

```
SELECT idp FROM personne
    WHERE nom = 'von Trier' AND prenom = 'Lars'
```

```
(13)
```

Exo 2 : Liste des titres de films réalisés par Lars von Trier

```
SELECT titre FROM film
    WHERE idr = (SELECT idp FROM personne
        WHERE personne.nom = 'von Trier'
            AND personne.prenom = 'Lars')
```

```
(Breaking the waves)
(Dogville)
```

Exo 2 : Déterminer les titres de films dont le genre est Drame

```
SELECT titre FROM film WHERE genre = 'Drame'
```

```
(Crash)
(Breaking the waves)
(Dogville)
... (3 de plus)
```

Exo 2 : Déterminer les titres des films réalisés dans les années 80

```
SELECT titre FROM film
    WHERE  1980 <= annee AND annee <= 1990
```

```
(Faux-Semblants)
(Dangereusement vôtre)
... (1 de plus)
```

Exo 2 : Déterminer le nombre de projections de films

```
SELECT COUNT(*) FROM projection
```

```
(25)
```

Exo 2 : Déterminer les rôles joués par Kevin Spacey

```
SELECT role FROM jouer
    WHERE ida = (SELECT idp FROM personne
            WHERE nom = 'Spacey' AND prenom = 'Kevin')

(Jim Williams)
(Lester Burnham)
```

3 Exercice 3 Base : `base_triangles.db`

Exo 3 : Nombre total de triangles

```
SELECT COUNT(*) FROM triangles

(100000)
```

Exo 3 : Tous les triangles dont le périmètre est 100

```
SELECT * FROM triangles WHERE ab+ac+bc=100

(471, 47, 50, 3)
(607, 35, 32, 33)
... (226 de plus)
```

Exo 3 : Tous les produits des cotes des triangles de périmètre >= 100

```
SELECT ab*ac*bc FROM triangles WHERE ab+ac+bc>=100

(12187500)
(50150)
... (91627 de plus)
```

Exo 3 : Insertion d'une ligne supplémentaire

```
INSERT INTO triangles VALUES (100001,1, 99, 1)
```

Exo 3 : Insertion d'une ligne supplémentaire

```
INSERT INTO triangles VALUES (100002,99, 1, 1)
```

Exo 3 : Tous les triangles équilatéraux


```
SELECT * FROM triangles WHERE ab = ac AND ab = bc

(15, 31, 31, 31)
(2045, 6, 6, 6)
... (17 de plus)
```

Exo 3 : Tous les triangles rectangles en A

```
SELECT  * FROM triangles  WHERE bc*bc=ab*ab+ac*ac

(0, 125, 325, 300)
(127, 70, 74, 24)
... (64 de plus)
```

Exo 3 : Le nombre de triangles rectangles en A

```
SELECT  COUNT(*)  FROM triangles  WHERE bc*bc=ab*ab+ac*ac

(66)
```

Exo 3 : Le maximum des périmètres des triangles rectangles en A

```
SELECT MAX(ab+bc+ac) FROM triangles  WHERE bc*bc=ab*ab+ac*ac

(1400)
```

Exo 3 : La plus petite valeur des produits AB.AC.BC, pour les triangles ABC de périmètre supérieur ou égal à 100

```
SELECT MIN(ab*bc*ac) FROM triangles  WHERE  ab+ac+bc>=100

(2450)
```

Exo 3 : Les longueurs correspondants au(x) triangle(s) pour le(s)quel(s) le minimum précédent est atteint

```
SELECT ab,ac,bc ,ab*ac*bc AS produit FROM triangles
    WHERE  ab+ac+bc>=100  AND ab*ac*bc=(
        SELECT  MIN(ab*ac*bc)    FROM triangles
            WHERE ab+ac+bc>=100
        )

(50, 1, 49, 2450)
```

Exo 3 : Ensemble des triangles plats

```
SELECT * FROM triangles WHERE ab+ac=bc OR ab+bc=ac OR ac+bc=ab

(20, 16, 55, 39)
```

```
(38, 45, 97, 52)
... (2877 de plus)
```

Exo 3 : Suppression des lignes insérées en 2)

```
DELETE FROM triangles WHERE idt > 100000
```

4 Exercice 4 Base : base_triangles.db

```
import sqlite3
base = sqlite3.connect('base_triangles.db')
curseur = base.cursor()
res = curseur.execute("""SELECT * FROM triangles WHERE ab+bc+ac=100""")
foo = res.fetchall()
curseur.close()
base.close()
print('res=',res,end='\n\n')
print('foo=',foo,end='\n\n')
print(len(foo),end='\n\n')

"""
res= <sqlite3.Cursor object at 0xb44a09e0>

foo= [(24, 51, 12, 37), ..., (99955, 24, 72, 4)]

474
"""
```

5 Exercice 5 Base : communes-regions-departements.db

Exo 5 : Jointure entre les tables communes et departements

```
SELECT c.nom, d.nom
FROM communes AS c JOIN departements AS d
ON c.dep = d.id

(L'Abergement-Clémenciat, Ain)
(L'Abergement-de-Varey, Ain)
... (36703 de plus)
```

Exo 5 : Liste de toutes les communes avec pour chacune, son département, sa région et sa population

```
SELECT  c.nom, d.nom, r.nom, c.pop
FROM  communes  AS c JOIN departements AS d JOIN regions AS r
ON c.dep=d.id AND d.reg=r.id

(L'Abergement-Clémenciat, Ain, Rhône-Alpes, 784)
(L'Abergement-de-Varey, Ain, Rhône-Alpes, 221)
... (36703 de plus)
```

Exo 5 : Liste des villes de plus de 100000 habitants, ainsi que leur population et leur région

```
SELECT  c.nom, r.nom, c.pop
FROM  communes  AS c JOIN departements AS d JOIN regions AS r
ON c.dep=d.id AND d.reg=r.id
WHERE c.pop >= 100000
ORDER BY c.pop DESC

(Paris, Île-de-France, 2243833)
(Marseille, Provence-Alpes-Côte d'Azur, 850726)
... (39 de plus)
```

Exo 5 : Liste des communes (nom et population) dont le nom commence par 'Pa' et se finit par 'is'

```
SELECT  c.nom, r.nom, c.pop
FROM  communes  AS c JOIN departements AS d JOIN regions AS r
ON c.dep=d.id AND d.reg=r.id
WHERE c.nom LIKE 'Pa%is'

(Pargny-les-Bois, Picardie, 136)
(Passy-en-Valois, Picardie, 164)
... (4 de plus)
```

Exo 5 : Liste des communes qui ont strictement plus de lettres dans leur nom que leur nombre d'habitants

```
SELECT  nom, pop FROM communes WHERE  LENGTH(nom )>pop

(Majastres, 2)
(Saint-Martin-lès-Seyne, 18)
... (52 de plus)
```

6 Exercice 6 Base : cinema843.db

Exo 6 : Quels sont les titres des films réalisés par Lars von Trier ?

```
SELECT film.titre FROM film
    JOIN personne
        ON personne.idp = film.idr
            WHERE personne.nom = 'von Trier' AND personne.prenom = 'Lars'

(Breaking the waves)
(Dogville)
```

Exo 6 : Quels sont les titres des films où Kevin Spacey a joué ?

```
SELECT film.titre FROM personne
    JOIN jouer ON personne.idp = jouer.ida
        JOIN film ON jouer.idf = film.idf
        WHERE personne.prenom = 'Kevin' AND personne.nom = 'Spacey'

(Minuit dans le jardin du bien et du mal)
(American Beauty)
```

Exo 6 : Quels sont les drames que l'on a pu voir après le 1er janvier 2002 ?

```
SELECT film.titre, film.genre, projection.jour
    FROM film JOIN projection
        ON film.idf = projection.idf
            WHERE film.genre = 'Drame'
                AND projection.jour >= '2002-01-01'

(Dogville, Drame, 2002-05-01)
(Dogville, Drame, 2002-05-02)
... (2 de plus)
```

Exo 6 : Quels sont les noms et prénoms des personnes qui sont des acteurs ? Ordonner la liste par ordre alphabétique croissant des noms.

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN jouer
        ON personne.idp = jouer.ida
            ORDER BY personne.nom

(Arquette, Rosanna)
(Bettany, Paul)
... (12 de plus)
```

Exo 6 : Quels sont les noms et prénoms des personnes qui sont des réalisateurs ?

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr

(Wayne, John)
(von Trier, Lars)
... (5 de plus)
```

Exo 6 : Quels sont les noms et prénoms des personnes qui sont à la fois des acteurs et des réalisateurs ?

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN jouer
        ON personne.idp = jouer.ida
```



```
INTERSECT
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr


(Eastwood, Clint)
```

Exo 6 : Quels sont les noms et prénoms des acteurs qui ne sont pas des réalisateurs ?

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN jouer
        ON personne.idp = jouer.ida
EXCEPT
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr

(Arquette, Rosanna)
(Bettany, Paul)
... (11 de plus)
```

Exo 6 : Noms et prénoms des réalisateurs qui ont réalisé des films policier.

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr
            WHERE genre = 'Policier'

(Tarantino, Quentin)
(Eastwood, Clint)
```

Exo 6 : Noms et prénoms des réalisateurs qui ont réalisé des films policier et des films dramatiques.

```
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr
            WHERE genre = 'Policier'
INTERSECT
SELECT DISTINCT personne.nom, personne.prenom
    FROM personne JOIN film
        ON personne.idp = film.idr
            WHERE genre = 'Drame'

(Eastwood, Clint)
```

Exo 6 : Liste de toutes les interprétations possibles en précisant le nom et le prénom de l'acteur ainsi que le role et le titre du film

```
SELECT personne.nom, personne.prenom, film.titre, jouer.role
    FROM personne JOIN jouer ON personne.idp = jouer.ida
        JOIN film ON jouer.idf = film.idf
            ORDER BY personne.nom DESC

(Willis, Bruce, Pulp Fiction, Butch Coolidge)
(Watson, Emily, Breaking the waves, Bess)
... (14 de plus)
```

Exo 6 : Titres des films réalisés par David Cronenberg qui ont été projetés au cinéma UGC

```
SELECT film.titre, personne.nom, personne.prenom, cinema.nom, projection.jour
    FROM personne JOIN film ON personne.idp = film.idr
            JOIN projection ON film.idf = projection.idf
                JOIN cinema ON projection.idc = cinema.idc
                    WHERE personne.nom = 'Cronenberg'
                        AND personne.prenom = 'David'
                            AND cinema.nom = 'UGC'

(Faux-Semblants, Cronenberg, David, UGC, 1990-12-08)
(Crash, Cronenberg, David, UGC, 1997-05-11)
```

Exo 6 : Acteurs qui ont joué dans des films projetés au cinéma UGC depuis l'an 2000

```
SELECT  DISTINCT personne.nom, personne.prenom
    FROM personne JOIN jouer JOIN film JOIN projection JOIN cinema
        ON personne.idp = jouer.ida AND jouer.idf = film.idf
        AND film.idf = projection.idc AND projection.idc = cinema.idc
            WHERE  projection.jour >= '2000-01-01'
            AND cinema.nom = 'UGC'

(Irons, Jeremy)
```

Exo 6 : Titres des films où Stellan Skarsgard a joué un role et qui ont été projetés au cinéma UGC

```
SELECT DISTINCT film.titre, personne.nom, personne.prenom
    FROM personne JOIN jouer ON personne.idp = jouer.ida
        JOIN film ON jouer.idf = film.idf
            JOIN projection ON jouer.idf = projection.idf
                JOIN cinema ON projection.idc = cinema.idc
                    WHERE personne.nom = 'Skarsgard'
                        AND personne.prenom = 'Stellan'
                            AND cinema.nom = 'UGC'
                                ORDER BY film.titre

(Breaking the waves, Skarsgard, Stellan)
(Dogville, Skarsgard, Stellan)
```