0. Objectifs

- Prise en main d'un SG DB Browser (SQLite)

- Etablissement des requêtes SQL de base pour créer une BDD et ses tables suivant un cahier des charges.

Le langage SQL utilisé est SQLite une version « Lite » allégée / transportable du langage SQL. Ce qui induit des différences dans les appellations des types des attributs entre autres mais pas dans l'organisation des requêtes.

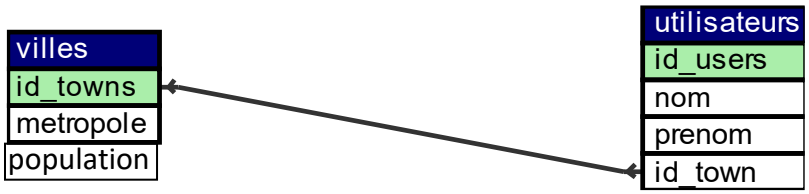
D'une manière générale chaque SGDB possède quelques particularités qui lui sont propre et différentes du SQL « standard » **SQL:2011** ou **ISO/CEI 9075:2011. Il faut donc ne pas hésiter à lire la documentation.**

La documentation se trouve à la page https://www.sqlitetutorial.net/

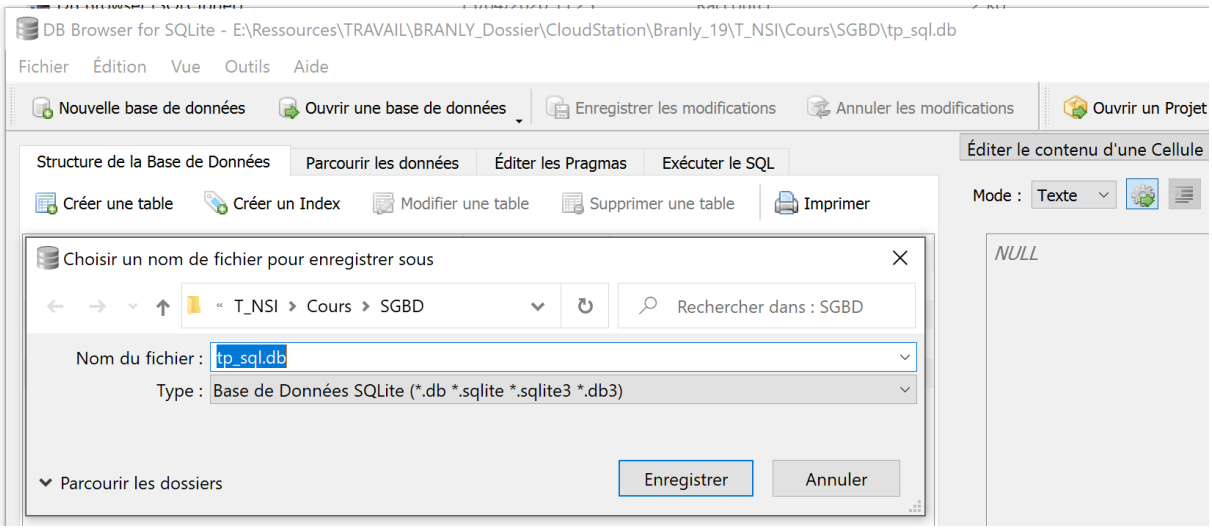
DB Brownser permet de créer graphiquement une base de données dans le TP on vous demande expressément d'utiliser des requêtes SQL .

1. Création de la base de données « tp_sql »

Un fournisseur internet a besoin de connaitre la localisation de ses clients. Il établit une base de données « tp_sql »



1.1. Créer une nouvelle base de données « tp_sql»

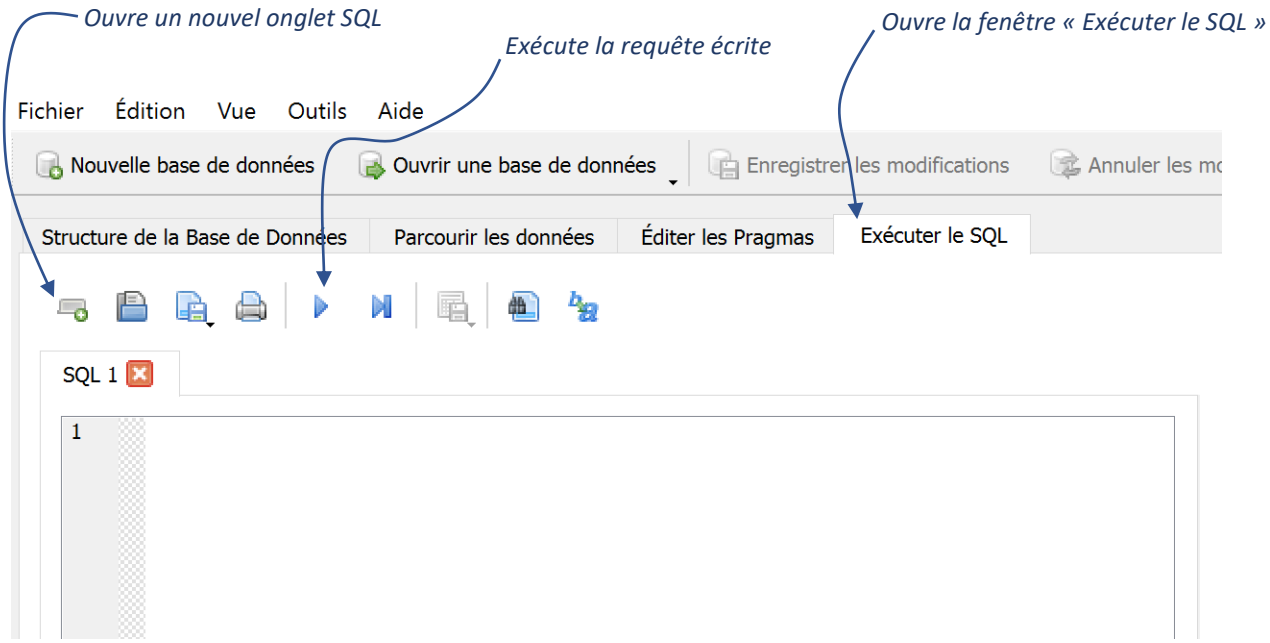


1.2. Insertion des tables dans la base de données

Ajouter deux tables « utilisateurs » et villes en utilisant une requête SQL dans l'ordre

villes (id_town INTEGER , metropole TEXT)

utilisateurs (id_user INTEGER , nom TEXT, prenom TEXT, genre TEXT, age INTEGER,# id_town)



⇨ Pour le compte-rendu de tp, écrire toutes les requêtes SQL demandées dans un fichier SQL ouvert avec *VisualStudioCode* que l'on uploadera sur nsibranly.fr avec le code **tp8**.

⇨ Sur DB Browser, chaque requête sera écrite dans un nouvel onglet et exécutée ensuite.

Requête 1.

```
CREATE TABLE villes (
    id_town INTEGER PRIMARY KEY,
    metropole TEXT,
    population INTEGER
) ;
```

Requête 2.

```
CREATE TABLE utilisateurs (
    id_user INTEGER PRIMARY KEY ,
    nom TEXT,
    prenom TEXT,
    genre TEXT,
    age INTEGER,
    id_towns INTEGER,
    FOREIGN KEY(id_towns) REFERENCES villes(id_town)
) ;
```

1.3. Remplissage des tables

⇨ Requête pour « villes » pour obtenir :

id_town	metropole	population
Filtre	Filtre	Filtre
1	LYON	513000
2	PARIS	2161000
3	MARSEILLE	861000
4	MACON	33000

Requête 3.

```
INSERT INTO villes VALUES
(1,'LYON',513000),
(2,'PARIS',2161000),
(3,'MARSEILLE',861000),
(4,'MACON',33000);
```

⇨ Requête pour « utilisateurs » pour obtenir :

id_user	nom	prenom	genre	age	id_towns
Filtre	Filtre	Filtre	Filtre	Filtre	Filtre
1	Chouhan	Jean	homme	50	1
2	Durand	Louis	homme	37	1
3	GranJean	Alice	femme	45	2
4	Bobet	Louison	homme	27	3

Requête 4.

```
INSERT INTO utilisateurs(id_user,nom,prenom,genre,age,id_towns) VALUES
(1,'Chouhan','Jean','homme',50,1),
(2,'Durand','Louis','homme',37,1),
(3,'GranJean','Alice','femme',45,2),
(4,'Bobet','Louison','homme',27,3);
```

1.4. Modification des tables

- Ajouter un 5ème enregistrement à la table « utilisateur » :

5	Champin	Arnaud	homme	23	1

Requête 5.

```
INSERT INTO utilisateurs VALUES
(5,'Champin','Arnaud','homme',23,1);
```

- Modifier l'âge de Durand par 83 :

Requête 6.

```
UPDATE utilisateurs SET age = 83 WHERE nom = 'Durand';
```

1.5. Opérations d'agrégation

- Calculer la somme de tous les âges et la ranger dans l'attribut « somme_age » :

	somme_age
1	228

Requête 7.

```
SELECT SUM(age) AS somme_age FROM utilisateurs;
```

- Donner le nombre d'utilisateurs :

	nb
1	5

Requête 8.

```
SELECT COUNT(*) AS nb FROM utilisateurs;
```

- Grouper ce résultat par genre :

	genre	nb
1	femme	1
2	homme	4

Requête 9.

```
SELECT genre , COUNT(*) AS nb FROM utilisateurs GROUP BY  genre   ;
```

- Donner la moyenne des âges des utilisateurs et ranger la dans l'attribut « ageMoyen » :

	ageMoyen
1	45.6

Requête 10.

```
SELECT AVG(age) AS ageMoyen FROM utilisateurs;
```

- Différencier suivant les genres :

	genre	ageMoyen
1	femme	45.0
2	homme	45.75

Requête 11.

```
SELECT genre,AVG(age) AS ageMoyen FROM utilisateurs GROUP BY genre;
```

- Donner l'âge maximum de chacun des genres :

	genre	age_maximum
1	femme	45
2	homme	83

Requête 12.

```
SELECT genre,MAX(age) AS age_maxi FROM utilisateurs GROUP BY genre;
```

1.6. Un truc rigolo quand on a bien rempli toute une table !

Crée à l'aide du logiciel une table « maTable » puis supprimer la :

Requête 13.

```
CREATE TABLE maTable  (
    id INTEGER
) ;
DROP TABLE maTable;
```

1.7. Mise en relation des deux tables

⇨ On désire obtenir les noms et prénoms des clients qui habitent LYON écrivez la requête correspondante. Le résultat devra faire apparaitre le prénom avant le nom :

	nom	prenom
1	Chouhan	Jean
2	Durand	Louis
3	Champin	Arnaud

Requête 14.

```
SELECT nom,prenom FROM utilisateurs
JOIN villes ON utilisateurs.id_towns = villes.id_town
WHERE metropole = 'LYON';
```

⇨ On désire obtenir le nombre d'habitants de la ville dans laquelle vit Durand :

	nom	prenom	population
1	Durand	Louis	513000

Requête 15.

```
SELECT u.nom,u.prenom, v.population FROM villes AS v
JOIN utilisateurs AS u ON v.id_town = u.id_towns
WHERE u.nom = 'Durand'
```

⇨ On désire obtenir le nom, prénom des utilisateurs qui vivent dans une ville de plus de 1 000 000 habitants :

	nom	prenom
1	GranJean	Alice

Requête 16.

```
SELECT u.nom,u.prenom FROM utilisateurs AS u
JOIN villes AS v ON u.id_towns = v.id_town
WHERE v.population > 1000000
```