

1.3

Le langage SQL

NSI TERMINALE - JB DUTHOIT

Nous avons étudié la structure d'une base de données relationnelle, et nous allons maintenant apprendre à :

- Créer une base de données
- créer des attributs
- Ajouter des données
- Modifier des données
- Interroger une base de données afin d'obtenir les informations souhaitées

Pour cela, il nous faut apprendre un langage de requêtes : SQL : Structured Query Language.

Nous allons utiliser le logiciel "DB Browser for SQLite" : <https://sqlitebrowser.org/>.

Définition

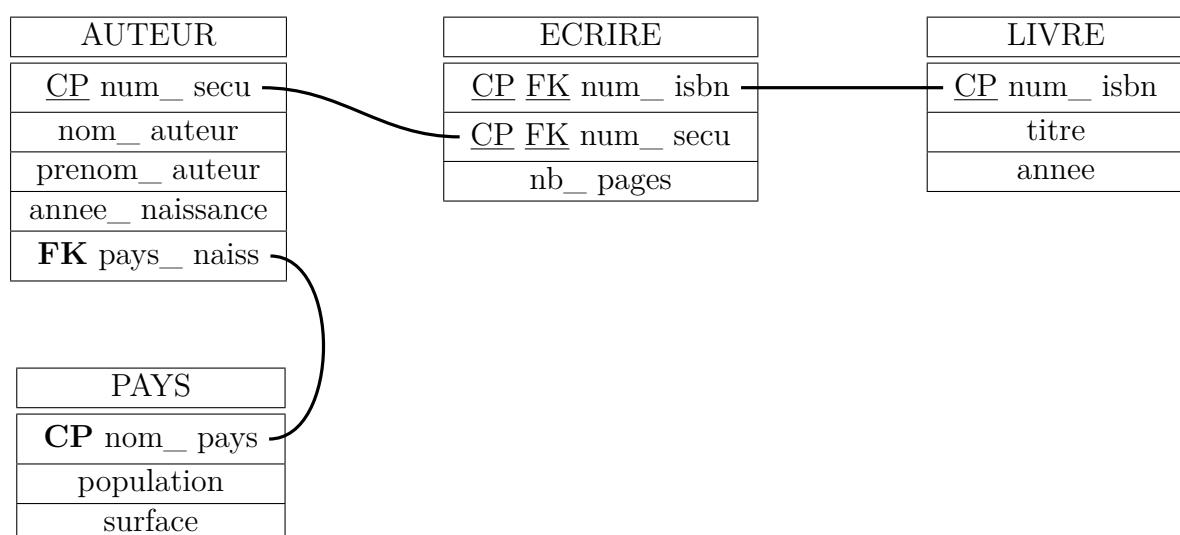
SQLite est une bibliothèque écrite en langage C qui propose un moteur de base de données relationnelle accessible par le langage SQL

Remarque

Noter qu'il existe d'autres systèmes de gestion de base de données relationnelle comme MySQL ou PostgreSQL. Dans tous les cas, le langage de requête utilisé est le SQL.

 Attention, on peut parfois noter quelques petites différences de syntaxe entre les différentes SGBD).

L'objectif ici est de créer le schéma relationnel suivant, d'y ajouter des enregistrements pour ensuite effectuer des requêtes !!!

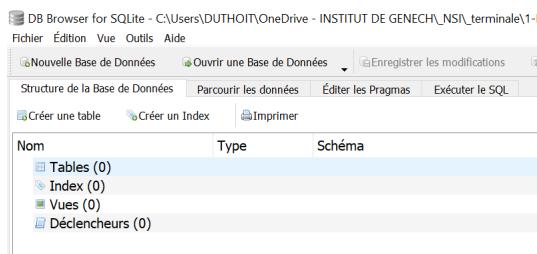


1.3.1 Creation d'une base de donnees

1. lancez le logiciel "DB Browser for SQLite"
2. Cliquez sur Nouvelle base de donnees.
3. Choisissez un nom pour cette base de donnees
4. Cliquez sur Annuler

☛ Un message confirme la creation de la table.

La base de donnees est cre e !



1.3.2 Creation, suppression et modification d'une relation

Creation d'une table

☛ Nous allons utiliser ici le mot clef **CREATE TABLE**

1. Cliquez sur l'onglet "Executer le SQL"
2. Recopiez le texte suivant dans la fentre "SQL 1"

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

3. Cliquez ensuite sur "Lecture" (petit triangle ou F5)

Nous avons pour chaque attribut prcise son domaine : id : entier (**INT**), titre : chane de caracteres (**TEXT**), nom_auteur : **TEXT** : ann_publi : **INT**

Remarque

Il faut noter la possibilit d'indiquer **NOT NULL** pour un attribut (sauf la cle primaire qui est forcement NOT NULL) ;

Cela signifie que l'on ne pourra pas cre re un enregistrement avec le marqueur **NULL** pour cet attribut.

⚠ NOT NULL ne signifie pas diffent de zero. Cela ne signifie pas non plus un champ vide.

☛ **NULL** est un marqueur utilis pour signifier une absence de valeur.

Création de la clef primaire

- ☛ Nous allons utiliser ici le mot clef **PRIMARY KEY**

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 LIVRE
(id INT, titre TEXT, nom_auteur TEXT, PRIMARY KEY (id));
```

Modification d'une table

- ☛ Nous allons utiliser ici le mot clef **ALTER TABLE** :

- Pour ajouter un attribut :

```
ALTER TABLE AUTEURS ADD email INT
```

- Pour modifier un attribut :

```
ALTER TABLE AUTEURS MODIFY email TEXT
```

- Pour changer le nom d'un attribut :

```
ALTER TABLE AUTEURS CHANGE email mail
```

- Pour supprimer un attribut :

```
ALTER TABLE AUTEUR DROP mail
```

Suppression d'une table

- ☛ Nous allons utiliser ici le mot clef **DROP TABLE**

```
DROP TABLE PAYS
```

 A utiliser avec beaucoup d'attention, car après les données sont perdues définitivement.

1.3.3 Ajout de données

Ajout de données

- ☛ Nous allons utiliser ici le mot clef **INSERT INTO** :

1. Cliquez sur "Exécuter le SQL"
2. Entrez le texte :

```
INSERT INTO LIVRES
(id, titre, nom_auteur, ann_publi)
VALUES
(1,"Les fleurs du mal", "Baudelaire", 1857),
(2,"L'étranger", "Camus", 1942),
(3,"Les misérables", "Hugo", 1862),
(4,"Les liaisons dangeureuses", "Laclos", 1782),
(5,"Le petit prince", "Saint-Exupéry", 1943),
(6,"Cyrano de Bergerac", "Rostand", 1897),
(7,"Les trois mousquetaires", "Dumas", 1844),
(8,"Le comte de Monte-Cristo", "Dumas", 1844),
(9,"La peste", "Camus", 1947),
(10,"Notre-Dame de Paris", "Hugo", 1831),
(11,"Orgueil et préjugés", "Austen", 1813),
(12,"Oliver Twist", "Dickens", 1839),
(13,"Le Portrait de Dorian Gray", "Wilde", 1890),
(14,"L'Ami retrouvé", "Uhlman", 1971),
(15,"Gatsby le Magnifique", "Fitzgerald", 1925);
```

Remarque

| Un message nous indique une nouvelle fois que la requête a été réalisée avec succès.

liste des enregistrements

	id	titre	nom_auteur	ann_publi
1	1	Les fleurs du mal	Baudelaire	1857
2	2	L'étranger	Camus	1942
3	3	Les misérables	Hugo	1862
4	4	Les liaisons dangeureuses	Laclos	1782
5	5	Le petit prince	Saint-Exupéry	1943
6	6	Cyrano de Bergerac	Rostand	1897
7	7	Les trois mousquetaires	Dumas	1844
8	8	Le comte de Monte-Cristo	Dumas	1844
9	9	La peste	Camus	1947
10	10	Notre-Dame de Paris	Hugo	1831
11	11	Orgueil et préjugés	Austen	1813
12	12	Oliver Twist	Dickens	1839
13	13	Le Portrait de Dorian Gray	Wilde	1890
14	14	L'Ami retrouvé	Uhlman	1971
15	15	Gatsby le Magnifique	Fitzgerald	1925

Suppression de données

☛ Nous allons utiliser ici le mot clef **DELETE FROM** :

DELETE permet de supprimer des lignes dans une relation. On peut utiliser la clause WHERE pour sélectionner les ligne à supprimer :

```
DELETE FROM CLIENTS
WHERE id_client <= 3
```

⚠ Il est préférable de sauvegarder la base de données avant de supprimer des lignes. De cette façon, un retour en arrière sera toujours possible !

⚠ (red) Si la clause WHERE n'est pas présente, toutes les lignes seront effacées !!

Modification de données

- ☛ Nous allons utiliser ici le mot clef **UPDATE** :

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Exemple :

```
UPDATE CLIENTS SET solde = 11724
WHERE prenom = 'John'
```

Il est possible également de réaliser ceci :

```
UPDATE CLIENTS SET nom = UPPER(nom), prix = prix * 1.1
```

1.3.4 Les requêtes

Liste des enregistrements

- ☛ Nous allons utiliser ici le mot clef **SELECT** :

```
SELECT id,titre, nom_auteur
FROM LIVRES
```

liste des enregistrements

The screenshot shows a MySQL command-line interface window titled "SQL 1". It contains the following text:

```
1  SELECT id, titre, nom_auteur
2  FROM LIVRES
```

Below the text, the results of the query are displayed in a table:

	id	titre	nom_auteur
1	1	Les fleurs du mal	Baudelaire
2	2	L'étranger	Camus
3	3	Les misérables	Hugo
4	4	Les liaisons dangereuses	Laclos
5	5	Le petit prince	Saint-Exupéry

Below the table, the following message is displayed:

```
L'exécution s'est terminée sans erreur.
Résultat : 15 enregistrements ramenés en 20ms
À la ligne 1 :
SELECT id, titre, nom_auteur
FROM LIVRES
```

Remarque

Il est possible de saisir :

```
SELECT *
FROM LIVRES
```

pour obtenir tous les attributs !

☛ * permet d'obtenir tous les attributs.

Remarque

L'utilisation de la commande SELECT en SQL peut potentiellement afficher des lignes en doubles.

☛ Pour éviter des redondances dans les résultats il faut simplement ajouter **DISTINCT** après le mot SELECT.

Filtres

- Nous allons utiliser ici le mot clef **WHERE**

liste des enregistrements

```
SQL 1
1  SELECT id, titre, nom_auteur
2  FROM LIVRES
3  WHERE nom_auteur = 'Camus' |
```

id	titre	nom_auteur
1	2 L'étranger	Camus
2	9 La peste	Camus

L'exécution s'est terminée sans erreur.
Résultat : 2 enregistrements ramenés en 21ms
À la ligne 1 :
SELECT id, titre, nom_auteur
FROM LIVRES
WHERE nom_auteur = 'Camus'

Remarque

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

Exercice 1.55

Écrire une requête qui permet d'obtenir la liste des livres qui ont été écrits après 1950 ou par Camus.

	id	titre	nom_auteur	ann_publi
1	2	L'étranger	Camus	1942
2	9	La peste	Camus	1947
3	14	L'Ami retrouvé	Uhlman	1971

Tri

Il est aussi possible de

- Nous allons utiliser ici le mot clef **ORDER BY**

rajouter la clause ORDER BY afin d'obtenir les résultats classés dans un ordre précis.

```
SELECT titre
FROM LIVRES
WHERE ann_publi >= 1900 ORDER BY nom_auteur
```

Exercice 1.56

Écrire une requête qui permet d'obtenir la liste "titre, nom_auteur", liste rangée par ordre chronologique de publication, pour les livres écrits après 1900.

Remarque

- Il est possible d'obtenir un classement en sens inverse à l'aide de la clause DESC

```
SELECT titre
FROM LIVRES
WHERE ann_publi >= 1900 ORDER BY nom_auteur DESC
```

Eviter les doublons

- ☛ Nous allons utiliser ici le mot clef **DISTINCT**
Si on effectue :

```
SELECT nom_auteur  
FROM LIVRES
```

On se retrouve avec des noms d'auteur en double.
Il suffit alors d'utiliser DISTINCT :

```
SELECT DISTINCT nom_auteur  
FROM LIVRES
```

1.3.5 Lier plusieurs tables d'une même base de données

Création des tables avec les différentes clés

☛ Nous allons utiliser ici le mot clef **FOREIGN KEY** notamment :

Créer une nouvelle base de données nommée "Exemple1".

Créer les tables suivantes :

AUTEUR	ECRIRE	LIVRE
<u>CP id_auteur</u>	<u>CP FK id_livre</u>	<u>CP id_livre</u>
nom_auteur	<u>CP FK id_auteur</u>	titre
prenom_auteur		annee
annee_naissance		
FK pays_naiss	nb_pages	

PAYS
<u>CP nom_pays</u>
population
surface

☛ Pour créer la clef étrangère sur l'attribut **pays_naiss** de la relation AUTEURS, on écrira :
FOREIGN KEY (pays_naiss) REFERENCES PAYS(nom_pays)

Ajout des données

On considère les livres suivants afin de compléter notre base de données :

"Les fleurs du mal", "L'étranger", "Les misérables", "Les liaisons dangereuses", "Le petit prince", "Cyrano de Bergerac", "Les trois mousquetaires", "Le comte de Monte-Cristo", "La peste", "Orgueil et préjugés", "Oliver Twist", "Le Portrait de Dorian Gray", "L'Ami retrouvé", "Gatsby le Magnifique", "Le Cercle littéraire des amateurs d'épluchures de patates".

Exercice 1.57

Compléter les instruction SQL suivantes :

INSERT INTO LIVRE

VALUES

(1,"Les fleurs du mal", 1857),
(2,"L'étranger", 1942),....

...

INSERT INTO AUTEUR

(id_auteur, nom_auteur, prenom_auteur, annee_naiss, pays_naiss)

VALUES

(1,'Baudelaire', 'Charles', 1821, 'France'), (2,'Camus', 'Albert', 1913, 'Algérie'),

...

INSERT INTO ECRIRE

(id_livre,id_auteur, nb_page)

VALUES

(1,1,100),

(2,2,100),
...
(on indiquera de façon arbitraire un nb non nul de pages)

```
INSERT INTO PAYS
(nom_pays, population, superficie)
VALUES
('France', 67, 643),
('Irlande', 5, 70),
On exprimera la population en million d'habitants et la superficie en milliers de km2
```

Les jointures

☛ Nous allons utiliser ici le mot clef **INNER JOIN** :

Observez bien cet exemple qui permet de joindre les différentes tables et d'afficher le nom, prénom et année de naissance des auteurs nés en France

```
SELECT nom_auteur, prenom_auteur, annee_naiss
FROM LIVRES
INNER JOIN ECRIRE ON ECRIRE.id_livre = LIVRES.id_livre
INNER JOIN AUTEUR ON ECRIRE.id_auteur = AUTEUR.id_auteur
INNER JOIN PAYS ON PAYS.nom_pays = AUTEUR.pays_naiss
WHERE pays_naiss = "France"
```

	nom_auteur	prenom_auteur	annee_naiss
1	Baudelaire	Charles	1821
2	Hugo	Victor	1802
3	Laclos	Choderlos de	1741
4	Saint-Exupéry	Antoine de	1900
5	Rostand	Edmond	1868
6	Dumas	Alexandre	1802
7	Dumas	Alexandre	1802
8	Hugo	Victor	1802

Remarque

On peut spécifier une contrainte de suppression et de mise à jour **sur la clé étrangère** avec respectivement **ON DELETE CASCADE** et **ON UPDATE CASCADE**.

Cela signifie que si on supprime des occurrences dans la relation vers laquelle "pointe" la clé étrangère, on supprime aussi les lignes en référence dans la relation où se trouve la clé étrangère.

Si on modifie la valeur de l'attribut "pointé" par la clé étrangère, on modifie su coup la ligne correspondante dans la relation où se trouve la clé étrangère.

La syntaxe est :

```
CREATE TABLE MA_TABLE_1
(id INT PRIMARY KEY,
att1 TXT
);
CREATE TABLE MA_TABLE_2
(id INT PRIMARY KEY,
att2 INT,
FOREIGN KEY(att2) REFERENCES MA_TABLE_1(id)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

1.3.6 Les fonctions d'agrégation

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur une colonne. Les principales fonctions sont :

sum : pour calculer la somme d'un attribut

min : pour récupérer la valeur minimale

max : pour récupérer la valeur maximale

avg : pour calculer la moyenne

count : pour compter le nombre d'enregistrements