

1.3

Le langage SQL

NSI TERMINALE - JB DUTHOIT

1.3.1 Introduction

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

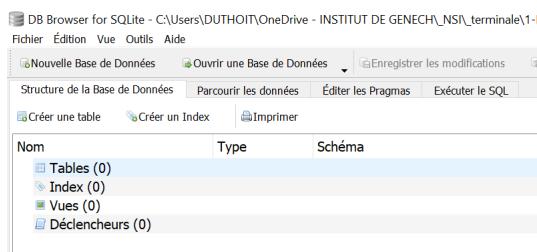
Noter qu'il existe d'autres systèmes de gestion de base de données relationnelle comme MySQL ou PostgreSQL. Dans tous les cas, le langage de requête utilisé est le SQL.

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

1.3.2 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 !



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

Ici, nous allons cre re la table LIVRES :

```
LIVRES(\underline{id INT},titre TEXT, ann_publi INT)
```

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, 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res (**TEXT**), ann_ publi : INT

Remarque

Il faut noter la possibilit d'indiquer **NOT NULL** pour un attribut (sauf la cl primaire qui est forcment 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rent de zro. Cela ne signifie pas non plus un champ vide.

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

Il existe de nombreux type de domaine, en voici quelques-uns :

type de données	syntaxe
Entier	INT, INTEGER ...
Chaîne de caractères	TEXT
	VARCHAR(20) (au maximum 20 caractères)
	CHAR(20) (chaine de 20 caractères, rempli de blancs si besoin)
Réel	REAL, FLOAT...

Remarque

- Type de données booléen :

SQLite ne dispose pas d'une classe de stockage booléenne distincte. Au lieu de cela, les valeurs booléennes sont stockées comme des entiers 0 (faux) et 1 (vrai).

- Date :

SQLite ne dispose pas d'une classe de stockage séparée pour stocker les dates et/ou les heures, mais SQLite est capable de stocker les dates et les heures sous forme de valeurs TEXT, REAL ou INTEGER.

Classe de stockage et format de date

- TEXT - Une date dans un format comme "AAAA-MM-JJ HH :MM :SS.SSS".
- REAL - Le nombre de jours depuis midi à Greenwich le 24 novembre 4714 avant J.-C.
- INTEGER - Le nombre de secondes depuis 1970-01-01 00 :00 :00 UTC

Vous pouvez choisir d'enregistrer les dates et les heures dans l'un de ces formats et convertir librement les formats à l'aide des fonctions intégrées de date et d'heure.

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 LIVRES
(id INT, titre TEXT, PRIMARY KEY (id));
```

ou bien :

```
CREATE TABLE LIVRES
(id INT PRIMARY KEY, titre TEXT, annee_publi TEXT);
```

Modification d'une table

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

- Pour ajouter un attribut :

```
ALTER TABLE LIVRES ADD nb_chap INT
```

- Pour modifier un attribut :

```
ALTER TABLE LIVRES MODIFY nb_chap REAL
```

- Pour changer le nom d'un attribut :

```
ALTER TABLE LIVRES CHANGE nb_chap nb_chapitres
```

- Pour supprimer un attribut :

```
ALTER TABLE AUTEUR DROP nb_chapitres
```

Suppression d'une table

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

```
DROP TABLE LIVRES
```

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

1.3.4 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, ann_publi)
VALUES
(1,"Les fleurs du mal", 1857),
(2,"L'étranger", 1942),
(3,"Les misérables", 1862),
(4,"Les liaisons dangeureuses", 1782),
(5,"Le petit prince", 1943),
(6,"Cyrano de Bergerac", 1897),
(7,"Les trois mousquetaires",1844),
(8,"Le comte de Monte-Cristo",1844),
(9,"La peste", 1947),
(10,"Notre-Dame de Paris", 1831),
(11,"Orgueil et préjugés",1813),
(12,"Oliver Twist", 1839),
(13,"Le Portrait de Dorian Gray", 1890),
(14,"L'Ami retrouvé", 1971),
(15,"Gatsby le Magnifique", 1925);
```

Remarque

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

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 LIVRES
WHERE ann_publi = 1971 ;
```

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

⚠ 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 LIVRES SET ann_publi = 1000
WHERE titre = "L'étranger" ;
```

Il est possible également de réaliser ceci :

```
UPDATE LIVRES SET titre = UPPER(titre)
```

1.3.5 Les requêtes

Liste des enregistrements

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

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

liste des enregistrements

	id	titre	annee_publi
1	1	Les fleurs du mal	1857
2	2	L'étranger	1942
3	3	Les miséables	1862
4	4	Les liaisons dangereuses	1782
5	5	Le petit prince	1943
6	6	Cyrano de Bergerac	1897
7	7	Les trois mousquetaires	1844
8	8	Le comte de Monte-Cristo	1844
9	9	La peste	1947
10	10	Notre-Dame de Paris	1831
11	11	Orgueil et préjugés	1813
12	12	Oliver Twist	1839
13	13	Le Portrait de Dorian Gray	1890
14	14	L'Ami retrouvé	1971
15	15	Gatsby le Magnifique	1925

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**

Remarque

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

Exercice 1.13

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

	id	titre	annee_publi
1	14	L'Ami retrouvé	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 titre
```

titre
1 Gatsby le Magnifique
2 L'étranger
3 La peste
4 Le petit prince
5 L'Ami retrouvé

Exercice 1.14

Écrire une requête qui permet d'obtenir la liste "titre, année_ publication", 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, ann_publi
FROM LIVRES
WHERE ann_publi >= 1900 ORDER BY ann_publi DESC
```

	titre	annee_publi
1	L'Ami retrouvé	1971
2	La peste	1947
3	Le petit prince	1943
4	L'étranger	1942
5	Gatsby le Magnifique	1925

Eviter les doublons

☛ Nous allons utiliser ici le mot clef **DISTINCT**

Si on effectue :

```
SELECT ann_publi  
FROM LIVRES
```

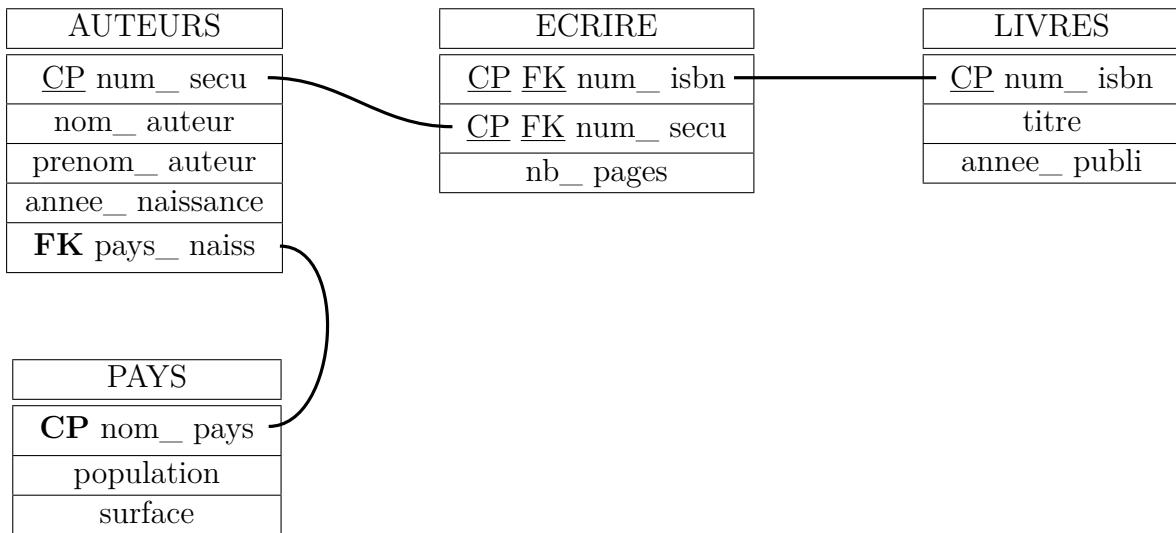
On se retrouve avec des doublons (pour 1844 par exemple).

Il suffit alors d'utiliser DISTINCT :

```
SELECT DISTINCT ann_publi  
FROM LIVRES
```

1.3.6 Lier plusieurs tables d'une même base de données

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



Création des tables avec les différentes clés

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

☛ Pour créer la clef étrangère sur l'attribut `pays_naiss` de la relation AUTEURS, on écrira :

```
CREATE TABLE AUTEURS
(id_auteur INT,
nom_auteur TEXT,
prenom_auteur TEXT,
annee_naissance INT,
pays_naissance TEXT,
PRIMARY KEY (id_auteur),
FOREIGN KEY (pays_naissance) REFERENCES PAYS(nom_pays);
```

☛ Faire de même avec la table PAYS et la table AUTEURS

Ajout des données

On ajoutera à notre relation LIVRES le livre "Le Cercle littéraire des amateurs d'épluchures de patates", qui a une particularité que vous allez vite trouver! :-)

Exercice 1.15

1. Compléter les instruction SQL suivantes afin d'ajouter le livre "Le Cercle littéraire des amateurs d'épluchures de patates" :

```
INSERT INTO LIVRES
VALUES
(...)
```

2. Ajouter les auteurs concernés :

```
INSERT INTO AUTEUR
(id_auteur, nom_auteur, prenom_auteur, annee_naissance, pays_naissance)
```

```

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

```

3. Entrer aussi les informations concernant la relation ECRIRE :

```

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)

4. Et enfin, on entrera les informations pour la relation PAYS :

```

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 km^2

Les jointures

- Nous allons utiliser ici le mot clef **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
JOIN ECRIRE ON ECRIRE.id_livre = LIVRES.id_livre
JOIN AUTEUR ON ECRIRE.id_auteur = AUTEUR. id_auteur
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

Exercice 1.16

Faire les requêtes qui permettent de répondre aux requêtes suivantes, sur l'ensemble des 4 tables précédentes : (Vous trouverez la réponse attendue issue de la requête)

- Quel est le pays de naissance d'un auteur qui a publié un livre en 1925

nom_pays	
1	USA

- Quel auteur à écrit "Notre-Dame de Paris" ?

nom_auteur	
1	Hugo

- Quels auteurs ont écrit "Le cercle littéraire des amateurs d'épluchures de patates"

nom_auteur	
1	Barrows
2	Shaffer

- Quels sont les livres écrits par "Camus" ?

titre	
1	L'étranger
2	La peste

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
);
```

1.3.7 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