Langage SQL

Afin de pouvoir manipuler le contenu d'une base de données relationnelle, on utilise un langage appelé SQL (Structured Query Language). Dans ce chapitre, on utilisera SQL OnLine IDE, un système de gestion de base de données entièrement en ligne : https://sqliteonline.com/.

I./ Création des tables :

On reprend l'exemple du chapitre précédent avec les relations « FILMS » et « REALISATEURS » :

Relation « FILMS »

id	titre	id_réalisateur	année	note critique
1	Terminator	1	1974	****
2	Titanic	1	1997	***
3	Avatar	1	2009	****
4	Avatar 2	1	2022	***
5	Avatar 3	1	2024	?
6	E.T.	2	1982	**
7	Jurassic Park	2	1993	****
8	Docteur Folamour	3	1964	**
9	2001	3	1968	****
10	Eyes Wide Shut	3	1999	*
11	Memento	4	2000	**
12	Interstellar	4	2014	****
13	Je vous salue, Marie	5	1985	*
14	Nouvelle vague	5	1990	***

Relation « REALISATEURS »

id	réalisateur	année naissance réalisateur	pays réalisateur
1	Cameron	1954	Canada
2	Spielberg	1946	USA
3	Kubrick	1928	USA
4	Nolan	1970	UK/USA
5	Godard	1930	France/Suisse

On rappelle que les attributs « id » des deux tables sont des clefs primaires pour chacune de ces tables, et que « id_realisateu » est une clef étrangère de la table « FILMS », faisant référence à la clef primaire « id » de la table « REALISATEURS ».

Pour créer ces deux tables, les instructions SQL nécessaires sont :

```
CREATE TABLE FILMS (id INT, titre TEXT, id_realisateur INT, annee INT, note_critique TEXT, PRIMARY KEY(id), FOREIGN KEY(id_realisateur) REFERENCES REALISATEURS(id));

CREATE TABLE REALISATEURS(id INT, realisateur TEXT, annee_naissance INT, pays TEXT, PRIMARY KEY(id));
```

Plusieurs remarques :

- La syntaxe est **CREATE TABLE** nom_de_la_relation(nom1 type1, nom2, type2, …)
- **PRIMARY KEY**(attribut) permet de définir « attribut » comme la clef primaire de la relation.
- **FOREIGN KEY** permet de définir une clef étrangère.
- SQL n'est pas sensible à la casse (sauf bien sûr pour le nom des tables et le nom des attributs), mais l'habitude est d'écrire les mots clefs SQL en majuscule.
- Le point-virgule à la fin de la ligne n'est pas nécessaire lorsqu'il y a une seule commande SQL, mais il permet de séparer deux commandes différentes.

1./ Aller sur le site https://sqliteonline.com/ et vérifier que vous êtes bien sur l'onglet « SQLite ».

2./ Effacer le contenu de la fenêtre « SQLite » (qui doit normalement contenir « SELECT * FROM demo; »), et recopier à la place le texte correspondant aux deux commandes de créations des tables.

3./ Cliquer sur le bouton « Run ».

Que remarque-t-on ?

II./ Insertion de données dans les tables :

La commande SQL permettant d'insérer un enregistrement dans une table est **INSERT INTO … VALUES …**

Par exemple, pour créer la première ligne de la table « FILMS », tapez dans la fenêtre SQLite la commande suivante, puis cliquez sur « Run » :

```
INSERT INTO FILMS VALUES (1, 'Terminator', 1, 1974, '****');
```

(Note : il y a une erreur volontaire au niveau de la date. Veuillez bien taper cette date incorrecte, nous verrons plus tard comment la corriger).

Afin de vérifier que cela a bien fonctionné, tapez maintenant **SELECT * FROM FILMS;** et vérifiez le résultat. Nous verrons plus loin la syntaxe de la commande **SELECT**.

On peut insérer plusieurs lignes, en les séparant par des virgules. Tapez par exemple :

```
INSERT INTO FILMS VALUES (2, 'Titanic', 1, 1997, '***'), (3, 'Avatar', 1, 2009, '****');
```

On pourrait finir de remplir la table « FILMS », puis la table « REALISATEURS » de cette façon. Pour aller un peu plus vite, on va utiliser un script SQL tout fait.

4./ Dans SQL Online IDE, faire « File », puis « Open SQL », et choisir le fichier « Cours_SQL.txt ». Le lancer en cliquant sur « Run ».

5./ Vérifier que les deux tables « FILMS » et « REALISATEURS » correspondent bien à ce qui a été décrit dans le cours précédent.

6./ Examiner la structure du script « Cours_SQL.txt ». On peut voir que les retours à la ligne ne sont pas pris en compte par le moteur SQL, et que seuls les points-virgules permettent de séparer deux commandes SQL. On peut donc utiliser les retours à la ligne pour rendre le code plus lisible par un humain. On peut aussi voir qu'il y a quelques ajouts par rapport à ce que nous avons vu. A quoi servent ces ajouts, notamment le « **IF NOT EXISTS** » après « **CREATE TABLE** », ainsi que les deux commandes « **DELETE** » ?

III./ Modification des données : mise à jour et suppression :

On peut modifier un enregistrement de la base de données grâce à la commande **UPDATE**. Dans notre exemple on voudrait modifier la date de sortie du film « Terminator », qui est de 1984 et non de 1974.

7./ Taper la commande suivante, puis cliquer sur « Run » :

```
UPDATE FILMS SET annee = 1984 WHERE id = 1;
```

La clause WHERE permet de désigner l'enregistrement que l'on veut modifier. Ici on choisit l'enregistrement dont l'attribut « id » est égal à 1. Mais on aurait aussi pu faire :

```
UPDATE FILMS SET annee = 1984 WHERE nom = "Terminator";
```

8./ Aurait-on pu utiliser un autre attribut pour faire cette modification ? Pourquoi ?

On peut supprimer un (ou plusieurs) enregistrement(s) grâce à la commande **DELETE**.

9./ Taper maintenant la commande suivante :

```
DELETE FROM FILMS WHERE id_realisateur = 1 ;
```

Que remarque-t-on ?

10./ Rafraichir la page, puis recharger le fichier « Cours_SQL.txt » pour reconstruire la base de données.

Quelle commande faut-il taper pour supprimer tous les films dont la note est égale à quatre étoiles ?

11./ Même question si on veut supprimer tous les films dont la note est égale à une ou deux étoiles. Indication : les opérateurs logiques **OR** et **AND** sont utilisables en SQL.

Si on veut supprimer tous les enregistrements d'une table, on peut utiliser la commande :

```
DELETE FROM Nom_de_la_relation;
```

12./ Supprimer tous les enregistrements des deux tables « FILMS » et « REALISATEURS », puis lancer les deux commandes suivantes :

```
SELECT * FROM FILMS;
```

```
SELECT * FROM REALISATEURS;
```

Attention : la commande **DELETE** ne fait qu'effacer les enregistrements d'une relation, elle ne détruit pas cette relation. Si on veut supprimer totalement une table, il faut utiliser la commande **DROP TABLE**.

IV./ Interrogation de la base de données :

L'interrogation de la base de données se fait grâce à la commande **SELECT**.

Si on veut afficher tous les enregistrements d'une table, la syntaxe est :

```
SELECT * FROM Nom_de_la_relation;
```

Si on veut afficher seulement certains attributs, la syntaxe est :

```
SELECT attribut_1, attribut_2 FROM Nom_de_la_relation;
```

13./ Afficher le nom et l'année de tous les films de la relation « FILMS ».

De la même façon qu'avec la commande **DELETE**, on peut rajouter une condition à l'aide de la clause **WHERE**.

14./ Afficher le nom de tous les films ayant une note de trois étoiles.

15./ Même question avec tous les films ayant une note de deux ou de trois étoiles.

16./ Afficher le nom et le pays de tous les réalisateurs nés après 1940.

Dans le dernier exemple, on s'aperçoit que les noms des réalisateurs sont présentés dans l'ordre dans lequel ils ont été insérés dans la table. Mais on peut trier les données présentées, grâce à la clause **ORDER**.

17./ Taper la commande suivante :

```
SELECT titre FROM FILMS ORDER BY titre;
```

On peut utiliser le mot-clef **DESC** si on veut trier par ordre décroissant.

18./ Taper la commande suivante :

```
SELECT titre FROM FILMS ORDER BY titre DESC;
```

19./ Afficher le titre et l'année de sortie de tous les films de la base, triés par ordre de date de sortie croissante.

V./ Combinaison de plusieurs tables : le produit cartésien et la jointure :

Une première manière de combiner deux tables est le produit cartésien :

20./ Taper **SELECT * FROM FILMS, REALISATEURS;**. Le résultat obtenu est le produit cartésien de la table « FILMS » et de la table « REALISATEURS » : la première ligne de la table « FILMS » va être combinée avec chacune des lignes de la table « REALISATEURS », puis la deuxième ligne de la table « FILMS » va être combinée avec chacune des lignes de la table « REALISATEURS », et ainsi de suite.

21./ Combien la table obtenue après requête a-t-elle de lignes ? Était-ce prévisible ?

22./ Taper **`SELECT * FROM REALISATEURS, FILMS;`**. Le résultat obtenu est-il le même que lors de la requête précédente ? Le produit cartésien de deux tables est-il commutatif ?

La deuxième manière de combiner deux tables est la jointure. Il existe plusieurs types de jointures en SQL, mais seule **`INNER JOIN`** est au programme.

23./ Taper la commande SQL suivante :

```
SELECT * FROM FILMS INNER JOIN REALISATEURS
ON REALISATEURS.id = FILMS.id_realisateur;
```

On remarque que pour chaque ligne, l'attribut « id » de la table « REALISATEURS » correspond à l'attribut « id_realisateur » de la table « FILMS ». Une fois les deux tables combinées, on peut faire des requêtes plus complexes.

24./ Taper par exemple la commande suivante :

```
SELECT REALISATEURS.realisateur, REALISATEURS.annee_naissance
FROM FILMS INNER JOIN REALISATEURS ON REALISATEURS.id = FILMS.id_realisateur
WHERE FILMS.annee > 2000
```

Cette commande permet d'afficher le nom et l'année de naissance de tous les réalisateurs ayant réalisé un film dont l'année de sortie est strictement supérieure à 2000.

On remarque que le nom de l'un des réalisateurs apparait plusieurs fois. Pour éviter cela, on peut utiliser le mot-clef **`DISTINCT`**.

25./ Taper la commande suivante :

```
SELECT DISTINCT REALISATEURS.realisateur, REALISATEURS.annee_naissance
FROM FILMS INNER JOIN REALISATEURS ON REALISATEURS.id = FILMS.id_realisateur
WHERE FILMS.annee > 2000
```

26./ Afficher la liste des films et de leurs notes critiques, des films réalisés par des personnes nées avant 1960 qui ne soient pas originaires du Canada. On présentera les résultats dans l'ordre alphabétique des films.

27./ Même question en ne comptant que les films sortis après 1985.

VI./ Quelques fonctions d'agrégation :

Les fonctions d'agrégations sont des fonctions calculant des statistiques sur les tables.

28./ Taper la commande suivante : **`SELECT COUNT(*) FROM FILMS`**. Essayer ensuite la commande : **`SELECT COUNT(*) FROM REALISATEURS`**. Que remarque-t-on ?

29./ En déduire la commande SQL permettant de donner le nombre de films ayant une note critique de trois étoiles.

D'autres fonctions d'agrégation existent, comme **`AVG`** (pour le calcul de moyenne), **`MAX`** et **`MIN`** (pour obtenir respectivement le maximum ou le minimum d'une colonne), **`SUM`** (pour réaliser des sommes), …

Vous trouverez plus d'informations sur les fonctions d'agrégation sur le lien suivant : https://sql.sh/fonctions/agregation/