

# Introduction au SQL

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Les bases du standard SQL et la sélection de données</b>	<b>3</b>
A. Les bases du standard SQL et la sélection de données .....	3
<b>III. Utiliser SQL pour insérer, modifier et supprimer des données</b>	<b>11</b>
A. Utiliser SQL pour insérer, modifier et supprimer des données .....	11
<b>IV. Introduction aux utilisations des fonctions SQL</b>	<b>14</b>
A. Introduction aux utilisations des fonctions SQL .....	14
<b>V. Essentiel</b>	<b>16</b>
<b>VI. Auto-évaluation</b>	<b>17</b>
A. Exercice .....	17
B. Test .....	18
<b>Solutions des exercices</b>	<b>19</b>

## I. Contexte

**Durée :** 1 h

**Environnement de travail :** navigateur web

**Prérequis :** aucun

### Contexte

SQL, ou Structured Query Language, est un langage standardisé utilisé pour interroger et manipuler des bases de données relationnelles. Il a été développé pour permettre aux utilisateurs de définir des relations entre les données, de les manipuler et de gérer les accès à celles-ci. SQL est basé sur le modèle relationnel des bases de données, introduit par Edgar Codd en 1970. Ce modèle organise les données en tables composées de colonnes définies par des attributs.

En tant que norme internationale depuis 1987, SQL est en constante évolution. Il s'agit d'un langage de requête déclaratif, ce qui signifie que les utilisateurs décrivent ce qu'ils veulent obtenir plutôt que comment l'obtenir. Les requêtes SQL sont généralement écrites sous forme de langage naturel, ce qui facilite leur compréhension et leur utilisation.

Voyons ensemble comment SQL peut être utilisé pour interagir avec les bases de données relationnelles.

## II. Les bases du standard SQL et la sélection de données

### A. Les bases du standard SQL et la sélection de données

#### Exemple SQL

Supposons que vous disposiez d'un système de gestion de base de données relationnelle, ou SGBDR, dans laquelle vous conservez des données sur des animaux, et que vous souhaitiez récupérer tous les animaux appartenant à la famille des requins.

Vous allez envoyer une requête SQL à votre base du type :

```
1 SELECT * FROM animaux WHERE famille='requin';
```

Cette requête indique les données que vous souhaitez obtenir. La base de données va vous répondre en vous renvoyant les données demandées.

Dans cet exemple, nous allons récupérer toutes les données de la table « *animaux* » où la valeur du champ « *famille* » est égale à « *requin* ».

Par ailleurs, la requête SQL ne dit pas au SGBDR comment récupérer ces informations et c'est là tout l'intérêt et la puissance des langages déclaratifs. Le SGBDR va donc répondre à votre requête de la manière la plus efficace possible sans que vous ayez à vous soucier de son fonctionnement interne.

### SGBDR

En effet, un SGBDR est un système de fichiers et de programmes complexe et sophistiqué nécessitant beaucoup de travail, de recherche et d'investissement.

Il gère énormément de choses (les données stockées, la mémoire, les accès concurrents à la base, les contraintes imposées sur les données, etc.) et réalise beaucoup d'optimisations en interne. L'intention de SQL est donc de masquer les détails d'implémentation compliqués de la base de données, et de fournir une interface uniforme et indépendante de toute implémentation. L'utilisateur d'un SGBDR, ou client, peut se concentrer sur l'écriture de requêtes SQL répondant à ses besoins, comme « *retourne-moi tous les animaux appartenant à la famille des requins* » sans se demander où sont stockées les données sur la machine, sous quel format, comment les lire, etc.

Par exemple, la manière de stocker les données physiquement est le problème du SGBDR, pas du standard SQL. Le standard SQL définit comment formuler la demande de données, le SGBDR doit se débrouiller pour y répondre.

Il existe de nombreuses implémentations du standard SQL, parmi les plus connues nous pouvons citer Db2 d'IBM, MySQL et Oracle d'Oracle Corporation, MariaDB, PostgreSQL, SQLite ou encore SQL Server de Windows. Chaque SGBDR a ses avantages et ses inconvénients, implémente de manière plus ou moins avancée le standard SQL et propose aussi ses propres fonctionnalités.

Oracle est par exemple le SGBDR le plus avancé et le plus performant au monde, mais c'est un SGBDR propriétaire, dont la licence est onéreuse. PostgreSQL est quant à lui le SGBDR open-source le plus avancé dans l'implémentation du standard SQL. MySQL est devenu populaire dans la stack LAMP (Linux, Apache, MySQL, PHP) au début des années 2000 pour la mise en place de sites web avec des logiciels libres. C'est le SGBDR utilisé par WordPress par exemple.

Comme SQL est un standard, la requête SQL citée précédemment vous permettra de récupérer les animaux de la famille des requins auprès de votre base, que votre base soit SQLite, SQL Server, Oracle ou MySQL.

## Fondamental Du modèle relationnel au SQL

Le modèle relationnel est fondé mathématiquement sur l'algèbre relationnelle d'Edgar Codd. Dans l'algèbre de Codd, les données sont organisées en relations (d'où le nom du modèle). Une relation est définie par un ou plusieurs attributs valués et par une clef. Par exemple, la relation `Employé` : `_matricule_`, `nom`, `prénom`, `date de naissance` a pour clef l'attribut `_matricule_` et est définie par quatre attributs. Chaque information rangée dans la base doit donc être définie par quatre valeurs. Par exemple « (1, Codd, Edgard, 23 septembre 1923) » est un ensemble de valeurs valides de la relation `Employé`. On parle aussi de tuple. La valeur de la clef (ici `_matricule_`) doit être différente pour chaque tuple de la relation, afin de pouvoir les identifier de manière unique.

En SQL, une relation est représentée sous forme de table et un attribut, sous forme de colonne. Un tuple correspondra à une ligne ou enregistrement de cette table. Ainsi, on pourra représenter l'exemple précédent sous la forme de la table `Employé` suivante :

<code>_matricule_</code>	<code>nom</code>	<code>prénom</code>	<code>date_de_naissance</code>
1	Codd	Edgard	23 septembre 1923
2	Turing	Alan	23 juin 1912

La relation `Employé` est représentée par la table `Employé` définie par les colonnes `_matricule_`, `nom`, `prenom` et `date_de_naissance`. (1, 'Codd', 'Edgard', '23/09/1923') est une *ligne* de la table `Employé`. La colonne `_matricule_` sert de clef, que l'on appelle **clef primaire** en SQL : sa valeur est différente pour chaque ligne afin de l'identifier de manière unique et sans ambiguïté. Dorénavant, nous représenterons les relations sous forme de tables avec des colonnes et des lignes.

## Pourquoi le modèle relationnel s'appelle-t-il ainsi ?

Le modèle relationnel est basé sur le concept de relations. Les relations sont souvent confondues avec les associations, les liens entre relations (qui sortent du cadre de ce module), qui procurent toute la puissance au modèle relationnel. Le mot relation est à comprendre au sens de « *relater* », « *rapporter les faits* » et non comme une liaison entre plusieurs entités, comme une relation amicale ou amoureuse. Une relation est donc une structure pouvant contenir des données, des faits, « *à rapporter* » à l'utilisateur de la base.

## Opération de sélection (SELECT) et syntaxe d'une requête SQL

L'algèbre relationnelle définit un ensemble d'opérations fondamentales possibles sur les relations. Parmi ces opérations se trouvent les opérations de restriction et de projection que l'on peut effectuer en SQL avec l'instruction `SELECT`. Cette opération permet de ne conserver que certaines colonnes et certaines lignes de la table ayant des

caractéristiques décrites par le biais d'un prédicat, c'est-à-dire remplissant une ou plusieurs conditions qui peuvent être évaluées à « vrai » ou « faux ». Notre requête d'illustration « *SELECT \* FROM animaux WHERE famille='requin';* » est un exemple d'opération de restriction : elle ne sélectionne que les lignes de la table « *animaux* » appartenant à la famille « *requin* ».

Pour rendre toutes ces notions concrètes, nous nous appuyons sur une table « *Book* » (donc une relation) contenant les enregistrements suivants :

id	title	author_name	nationality	publication_year
1	L'île au trésor	Stevenson	Royaume-Uni	1883
2	Les Hauts de Hurlevent	Brontë	Royaume-Uni	1847
3	Madame Bovary	Flaubert	France	1869
4	Moby Dick	Melville	États-Unis	1851
5	À la recherche du temps perdu	Proust	France	1927
6	Demande à la poussière	Fante	États-Unis	1939
7	Les raisins de la colère	Steinbeck	États-Unis	1939
8	Don Quichotte	Cervantès	Espagnol	1605
9	Cent ans de solitude	Marquez	Colombie	1967
10	Hamlet	Shakespeare	Royaume-Uni	1603
11	Bilbo le Hobbit	Tolkien	Royaume-Uni	1937
12	Les Robots	Asimov	États-Unis	1950

La requête *SELECT* va nous permettre d'introduire la syntaxe d'une requête SQL, c'est-à-dire les règles à suivre pour former une requête compréhensible par un SGBDR, peu importe celui que vous utilisez. Si la syntaxe n'est pas correcte, le système qui gère la base de données vous renverra une erreur ou pire, réalisera des opérations non désirées sur la base. Prenez garde toutefois, chaque SGBDR peut avoir des éléments mineurs de syntaxe différents, jouant avec quelques libertés laissées par le standard. Lorsque vous travaillez en SQL, effectuez toujours vos recherches en indiquant quel SGBDR vous utilisez.

Une requête SQL est une chaîne de caractères qui s'écrit dans un langage proche du langage naturel (de la langue anglaise) et qui est transmise au système de base de données. Ce dernier va l'analyser en suivant un ensemble de règles propre à son implémentation, et retournera une réponse sous forme de table.

La requête **SELECT** s'écrit sous la forme suivante :

```
1 SELECT col_name [, col_name]... FROM table;
```

Dans cette convention d'écriture, toute expression placée entre crochets (« *[]* ») est optionnelle et peut être omise. Le symbole « ... » indique la répétition de l'expression qui le précède.

Par convention, on indique les mots réservés du langage SQL en majuscules (*SELECT*, *FROM*, etc.). Bien que SQL soit insensible à la casse, cette convention permet de faciliter la lecture et la compréhension d'une requête. Dans cette notation, *col\_name* indique qu'il faut sélectionner au moins une colonne de la relation, et *[, col\_name]...* indique que l'on peut en sélectionner plusieurs si on le souhaite (deux, trois, quatre, etc.). C'est la partie projection

de la requête. Le mot clé `FROM` indique la table dont on souhaite extraire les données. Chaque requête SQL se termine par un point-virgule (`;`) pour indiquer au SGBDR la fin de la requête. Une requête peut s'écrire sur plusieurs lignes et n'est pas sensible aux espaces ou tabulations. Elle doit seulement se terminer par un point-virgule.

Par exemple, si nous voulons récupérer « *Tous les titres de livres publiés* » nous pouvons écrire :

```
1 SELECT title FROM Book ;
```

Littéralement cette requête peut se lire « *sélectionner toutes les valeurs de la colonne **title** de la table **Book*** ». Le système nous répondra en retournant la table suivante :

title
L'île au trésor
Les Hauts de Hurlevent
Madame Bovary
Moby Dick
À la recherche du temps perdu
Demande à la poussière
Les raisins de la colère
Don Quichotte
Cent ans de solitude
Hamlet
Bilbo le Hobbit
Les Robots

Le SGBDR nous renvoie une table avec une seule colonne (la colonne **title** demandée) et toutes les lignes. Si nous voulons récupérer « *Tous les titres de livres publiés et le nom de leur auteur* », nous pouvons écrire :

```
1 SELECT title, author_name FROM Book ;
```

Le SGBDR nous renverra la table suivante :

title	author_name
L'île au trésor	Stevenson
Les Hauts de Hurlevent	Brontë
Madame Bovary	Flaubert
Moby Dick	Melville
À la recherche du temps perdu	Proust
Demande à la poussière	Fante
Les raisins de la colère	Steinbeck

title	author_name
Don Quichotte	Cervantès
Cent ans de solitude	Marquez
Hamlet	Shakespeare
Bilbo le Hobbit	Tolkien
Les Robots	Asimov

Notez que les colonnes de la table retournées sont précisément celles indiquées dans la requête `SELECT`. Si nous voulons récupérer « *toutes les informations sur les livres* », nous pouvons écrire une requête sélectionnant toutes les colonnes :

```
1 SELECT id, title, author_name, nationality, publication_year FROM Book ;
```

SQL nous fournit comme raccourci l'astérisque (\*) signifiant « *toutes les colonnes* ». Ainsi, la dernière requête SQL peut s'écrire plus simplement `SELECT * FROM Book ;`. Le SGBDR nous renverra alors la table dans son intégralité (toutes les lignes et toutes les colonnes).

#### Complément Formater les requêtes SQL pour améliorer leur lisibilité

Lorsque les requêtes deviennent complexes, elles peuvent être longues et difficiles à lire. Comme tout code source, les requêtes SQL sont faites pour être exécutées par des machines, mais elles sont écrites pour être lues par des humains. Pour améliorer leur lisibilité, en plus d'écrire les mots-clés SQL en majuscules, il est d'usage de l'écrire sur plusieurs lignes, avec une indentation cohérente, et de lui adjoindre un commentaire décrivant ce qu'elle fait. Par exemple :

```
1 -- Récupère tous les titres de livres
2 SELECT title
3 FROM     Book ;
```

#### Attention Ne sélectionnez que les informations dont vous avez réellement besoin !

L'usage de l'astérisque (\*) est commode pour sélectionner toutes les colonnes.

```
1 SELECT * FROM Book
```

C'est un très bon outil pour explorer et inspecter une base de données ou pour s'exercer en SQL. Cependant, en conditions réelles, il n'est pas recommandé de l'utiliser sauf si vous avez une bonne raison de le faire. Vous risquez en effet de sélectionner trop d'informations inutiles, ce qui a un coût en termes de performances.

En situation réelle, il n'est pas rare de manipuler une base qui contient plusieurs centaines de milliers de lignes voire des millions. Sélectionner deux ou dix sur un tel volume de données peut avoir des conséquences notables sur la performance de la requête.

De plus, d'un point de vue applicatif, si vous sélectionnez plus d'information que nécessaire, cela indique que vous allez devoir allouer de la mémoire à un volume de données inutile et trier vous-même ces données dans votre code. Il vaut mieux laisser cette tâche au SGBDR qui est optimisé pour ce genre d'opérations.

Pour finir, toujours du point de vue applicatif, formuler une requête sélectionnant toutes les colonnes avec l'astérisque ne rend pas le code appelant clair sur ses intentions. Cela signifie qu'il faudra lire le code pour mieux comprendre ce qui cherche à être fait et sur quelles données le code travaille réellement. Une requête **SELECT** bien formulée ne doit remonter que les informations indispensables.

### Complément L'algèbre relationnelle est une fermeture

L'algèbre relationnelle est une fermeture : chaque opération sur une relation retourne une relation. Pour comprendre cette propriété, considérez l'addition : l'addition de deux entiers **2 + 3** retourne un entier (**5**) sur lequel il est possible d'appliquer à nouveau l'addition. C'est identique en algèbre relationnelle et donc en SQL : une opération de sélection sur une table, avec une requête SELECT, renvoie une nouvelle table sur laquelle il est possible d'effectuer de nouvelles opérations. C'est une propriété importante de l'algèbre relationnelle pour bien comprendre les réponses aux requêtes SQL et les exploiter au mieux.

### Restriction avec la clause WHERE

À présent, imaginons que nous souhaitons récupérer uniquement les livres écrits par des auteurs de nationalité américaine. Comment nous y prendre ? Pour cela, SQL fournit la clause WHERE qui indique une ou plusieurs conditions que des lignes doivent satisfaire pour être sélectionnées.

La clause WHERE est une clause optionnelle de la requête SELECT. La syntaxe complète d'une requête de sélection avec la clause WHERE s'écrit comme suit :

```
1 SELECT col_name [, col_name]... FROM table [WHERE conditions] ;
```

Nous rappelons que dans cette syntaxe (utilisée par la plupart des documentations des implémentations de SQL), toute expression entre crochets ([]) est optionnelle.

Pour sélectionner uniquement les livres écrits par des auteurs de nationalité américaine, nous pouvons sélectionner les lignes où la nationalité est égale à « États-Unis », autrement dit nous allons restreindre notre sélection à une valeur prise par la colonne « *nationality* ». On ajoutera donc la clause WHERE `nationality='États-Unis'` qui peut être traduite littéralement par « où la colonne *nationality* a pour valeur « États-Unis » ». Pour chaque ligne, la valeur renseignée dans la colonne *nationality* sera comparée à la chaîne de caractère « États-Unis ». Si elles sont identiques, la ligne sera sélectionnée. L'expression `nationality='États-Unis'` forme ce qu'on appelle un prédicat. Cette expression est évaluée à vrai ou faux pour chaque ligne de la table. Seules les lignes pour lesquelles la condition est vérifiée (évaluée à vrai) seront sélectionnées. La requête complète s'écrit donc :

```
1 SELECT * FROM Book WHERE nationality='États-Unis' ;
```

Le SGBDR nous renverra comme réponse la table suivante :

id	title	author_name	nationality	publication_year
4	Moby Dick	Melville	États-Unis	1851
6	Demande à la poussière	Fante	États-Unis	1939
7	Les raisins de la colère	Steinbeck	États-Unis	1939
12	Les Robots	Asimov	États-Unis	1950

### Méthode SQL manipule différents types de données

Dans l'algèbre de Codd, tous les attributs doivent être évalués et définis sur un domaine. En informatique, et donc en SQL, il faut leur donner un type pour allouer de la mémoire de stockage et définir le domaine de validité. Par exemple, une colonne « *Nombre de places de concerts* » est logiquement définie par un type « *nombre entier* », on ne voudrait pas y autoriser l'insertion d'une chaîne de caractères. SQL définit de nombreux types de données. Il y a des données de type numérique (INTEGER, DECIMAL, etc.), de chaînes de caractères (CHAR, VARCHAR, TEXT, etc.), de date et d'heure (DATE, TIME, DATETIME, TIMESTAMP) et bien d'autres encore (JSON, SPATIAL, BLOB, etc.). Les chaînes de caractères se manipulent entre guillemets. Si l'on souhaite manipuler des chaînes de caractères



dans une clause WHERE par exemple, il ne faut pas oublier de placer la valeur entre guillemets (par exemple WHERE author\_name='Fante'). Une valeur numérique n'a pas besoin d'être placée entre guillemets (par exemple WHERE id=1).

#### Méthode Restriction multi-conditionnelle (AND et OR)

La clause WHERE permet d'ajouter autant de conditions que désiré. Par exemple, imaginons que l'on souhaite récupérer à présent tous les livres écrits par des auteurs de nationalité américaine du XIX<sup>e</sup> siècle (entre 1801 et 1900). Les lignes que nous cherchons doivent satisfaire à présent deux conditions : la nationalité de l'auteur doit être égale à « États-Unis » et la date de publication doit être comprise entre 1801 et 1900. Pour exprimer le « ET » booléen, SQL nous met à disposition l'opérateur logique AND. La clause WHERE peut donc s'écrire WHERE nationality='États-Unis' AND publication\_year >= 1801 AND publication\_year <= 1900. SQL nous permet également de placer chaque condition entre parenthèses. La requête complète s'écrit donc :

```
1 SELECT * FROM Book WHERE (nationality='États-Unis') AND (publication_year >= 1801 AND
   publication_year <= 1900) ;
```

La base nous renverra une table ne contenant que la ligne suivante :

id	title	author_name	nationality	publication_year
4	Moby Dick	Melville	États-Unis	1851

SQL fournit également l'opérateur logique OR pour indiquer le OU logique. Nous rappelons que l'expression (exp1 OR exp2) est évaluée à vrai si exp1 ou exp2 est vrai, sinon elle est évaluée à faux. Par exemple, si nous souhaitons récupérer tous les livres écrits par des auteurs de nationalité américaine ou britannique, nous pouvons le faire avec la clause WHERE suivante : WHERE nationality='États-Unis' OR nationality='Royaume-Uni' ;. La requête complète serait :

```
1 SELECT * FROM Book WHERE nationality='États-Unis' OR nationality='Royaume-Uni';
```

La base nous renverrait la table suivante :

id	title	author_name	nationality	publication_year
1	L'île au trésor	Stevenson	Royaume-Uni	1883
2	Les Hauts de Hurlevent	Brontë	Royaume-Uni	1847
4	Moby Dick	Melville	États-Unis	1851
6	Demande à la poussière	Fante	États-Unis	1939
7	Les raisins de la colère	Steinbeck	États-Unis	1939
10	Hamlet	Shakespeare	Royaume-Uni	1603
11	Bilbo le Hobbit	Tolkien	Royaume-Uni	1937
12	Les Robots	Asimov	États-Unis	1950

### Exemple Combiner AND et OR

Il est possible de combiner les opérateurs AND et OR pour effectuer des sélections plus avancées. Nous pouvons récupérer les livres écrits par des auteurs de nationalité américaine ou britannique publiés entre 1883 et 1939 avec la clause WHERE suivante : `WHERE (nationality='États-Unis' OR nationality='Royaume-Uni') AND (publication_year ≥ 1883 AND publication_year ≤ 1939);`. On place ici les conditions entre parenthèses pour faciliter la lecture et nous assurer de la bonne précedence des opérateurs.

### Complément L'opérateur BETWEEN

L'opérateur BETWEEN peut aussi être utilisé dans une requête SQL pour sélectionner directement un intervalle de données dans une requête utilisant la clause WHERE au lieu d'écrire deux conditions AND. C'est du sucre syntaxique et cela facilite la lecture de la requête. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. Nous pourrions réécrire l'exemple précédent comme suit : `WHERE nationality='États-Unis' AND publication_year BETWEEN 1801 AND 1900`. Attention cependant, car l'opérateur BETWEEN ne se comportera pas exactement de la même manière entre les différentes implémentations de SQL. Certains SGBDR vont inclure les valeurs qui définissent l'intervalle, d'autres non. C'est pour cela qu'il est toujours recommandé de consulter la documentation officielle de chaque SGBD avant d'utiliser les différents opérateurs.

### Méthode Trier les résultats (ORDER BY)

Jusqu'à présent nous avons récupéré nos résultats sans indiquer d'ordre particulier : ils venaient dans l'ordre dans lequel ils étaient au départ, au moment de la création de la base. Cela dit, nous aimerions pouvoir trier les livres par titre, par nom d'auteur ou encore par date de parution. Pour cela, SQL nous fournit l'opérateur ORDER BY.

Comme la clause WHERE, ORDER BY est une clause optionnelle de SELECT. La syntaxe complète d'une requête de sélection avec la clause ORDER BY s'écrit comme suit :

```
1 SELECT col_name [, col_name]... FROM table [ORDER BY col_name [ASC|DESC]] ;
```

ORDER BY col\_name [ASC|DESC] indique qu'il faut donner le nom de la colonne col\_name sur laquelle on souhaite classer les lignes. Par défaut, les résultats sont classés par ordre ascendant (du plus petit au plus grand), mais il est possible de classer par ordre descendant avec le suffixe DESC. La notation ASC|DESC indique que l'on peut utiliser ASC (ascendant) ou DESC (descendant), mais pas les deux en même temps.

Récupérons la liste des titres d'ouvrage, triés par ordre alphabétique. Pour cela, nous allons classer les résultats en utilisant la colonne title de manière ascendante (ordre alphabétique), soit ORDER BY title ou de manière explicite ORDER BY title ASC. La requête complète s'écrit :

```
1 SELECT title FROM Book ORDER BY title ASC ;
```

Si nous voulons à présent récupérer la liste des titres d'ouvrage, du plus récent au plus ancien, nous pouvons classer les livres sur la colonne publication\_year du plus grand au plus petit, soit ORDER BY publication\_year DESC. La requête complète s'écrit :

```
1 SELECT title, publication_year FROM Book ORDER BY publication_year DESC;
```

La base de données nous répond avec la table suivante :

title	publication_year
Cent ans de solitude	1967
Les Robots	1950
Demande à la poussière	1939
Les raisins de la colère	1939

title	publication_year
Bilbo le Hobbit	1937
À la recherche du temps perdu	1927
L'île au trésor	1883
Madame Bovary	1869
Moby Dick	1851
Les Hauts de Hurlevent	1847
Don Quichotte	1605
Hamlet	1603

**Attention** Il n'y a pas de notion d'ordre des lignes dans une table

Dans une table, l'ordre des lignes est arbitraire et ne porte aucune information. Autrement dit, il n'existe pas de notion d'ordre des lignes dans une table d'une base de données relationnelle. Si l'on souhaite trier les lignes suivant un certain ordre, il faut appliquer une opération de sélection avec la clause ORDER BY.

[cf. up-exemple-fil-rouge-data.sql]

### III. Utiliser SQL pour insérer, modifier et supprimer des données

#### A. Utiliser SQL pour insérer, modifier et supprimer des données

SQL est le langage pour interagir avec les bases de données relationnelles et il répond à la fois aux problématiques de création de la base de données (relations, colonnes, etc.), de manipulation des données comme nous venons de le voir avec l'instruction SELECT, mais également de mise à jour des données contenues dans la base : l'insertion, la modification ou la suppression de données. Dans cette partie nous allons étudier les instructions INSERT, UPDATE et DELETE permettant respectivement l'ajout, la modification et la suppression de lignes dans une table. C'est ce qu'on appelle la *Data Manipulation Language* ou DML.

**Méthode** Insérer des données dans une table avec INSERT

L'ajout de lignes dans une table utilise l'instruction INSERT et répond à la syntaxe suivante :

```
1 INSERT INTO table_name [<liste_colonnes>] VALUES <liste_valeurs>;
```

<liste\_colonne> et <liste\_valeurs> sont des listes SQL et s'écrivent entre parenthèses où chaque élément est séparé de l'autre par une virgule, par exemple « (1, 2, 3) ». table\_name indique dans quelle table insérer les données. <liste\_colonnes> est un argument optionnel (indiqué entre crochets), nous y reviendrons après. <liste\_valeurs> contient la liste des valeurs à insérer.

Ajoutons un livre à notre table Book. Nous allons ajouter l'ouvrage *Le Parfum* de Patrick Süskind, d'origine allemande, publié en 1985. Nous lui donnerons pour clef la valeur 13, étant donné que nous avons déjà 12 enregistrements dans notre table. Nous allons donc insérer la liste de ces valeurs dans la base, en prenant

garde de bien respecter l'ordre des colonnes spécifiées dans la définition de la table, soit ici id, title, author\_name, nationality, publication\_date. Notre liste de valeurs à insérer est donc (13, 'Le Parfum', 'Süsskind', 'Allemagne', 1985). Nous pouvons à présent écrire notre requête d'insertion :

```
1 INSERT INTO Book VALUES (13, 'Le Parfum', 'Süsskind', 'Allemagne', 1985) ;
```

Cette syntaxe, bien qu'elle soit simple, a malheureusement certains défauts :

- Il faut se souvenir des colonnes et de leur ordre.
- Elle oblige à renseigner une valeur pour chaque colonne. Si nous ne connaissons pas la date de publication par exemple, nous ne pouvons pas insérer nos données.
- Si la structure de la table change dans le futur, la requête d'insertion pourrait ne plus fonctionner.
- La lecture de la requête n'est pas aisée pour un être humain (vous dans le futur ou un collaborateur). En effet, vous supposez que la personne connaît la structure (les colonnes et leur ordre) de votre table, ce qui n'est pas forcément le cas.

C'est pourquoi il est recommandé d'utiliser le paramètre optionnel <liste\_colonne> qui sert à indiquer sous forme de liste les colonnes pour lesquelles vous allez fournir une donnée. Vous pouvez ainsi choisir de fournir une donnée pour chaque colonne ou seulement pour quelques-unes et également changer l'ordre des colonnes au besoin. Par exemple, la requête précédente d'insertion peut être réécrite comme :

```
1 INSERT INTO Book (id, title, author_name, nationality, publication_year) VALUES (13, 'Le Parfum', 'Süsskind', 'Allemagne', 1985) ;
```

On a indiqué ici le nom de chaque colonne, la requête est plus lisible. Et si demain une nouvelle colonne est ajoutée à la table, cette requête fonctionnera toujours. Cette requête est équivalente à celle-ci, où l'ordre des colonnes a été changé :

```
1 INSERT INTO Book (title, publication_year, nationality, author_name, id) VALUES ('Le Parfum', 1985, 'Allemagne', 'Süsskind', 13) ;
```

Enfin, supposons que nous souhaitions ajouter l'ouvrage suivant *Le nom de la rose*, mais que nous ne connaissons pas encore avec certitude ni la date de parution, ni le nom de l'auteur, ni sa nationalité. Si la base de données accepte une information incomplète de manière temporaire, nous pouvons quand même enregistrer ce que nous savons déjà en listant les colonnes pour lesquelles nous avons une valeur à renseigner. Nous pourrions donc écrire la requête d'insertion suivante :

```
1 INSERT INTO Book (title, id) VALUES ('Le nom de la rose', 14) ;
```

Notez ici que seules les colonnes title et id sont indiquées. Les colonnes pour lesquelles nous n'avons pas renseigné de valeurs se voient attribuer la valeur par défaut NULL qui indique l'« absence de valeurs ». Voici à quoi ressemblerait notre table Book (en ne considérant que les deux enregistrements réalisés dans cette section) :

id	title	author_name	nationality	publication_year
13	Le Parfum	Süsskind	Allemagne	1985
14	Le Nom de la rose	NULL	NULL	NULL

#### **Attention** Lister toujours les colonnes dans l'instruction INSERT

Comme nous l'avons vu, nous pouvons ignorer l'argument <liste\_colonne> qui permet de lister les colonnes pour lesquelles nous allons fournir des valeurs. La requête d'insertion est alors complètement implicite à la fois sur les colonnes et sur l'ordre des colonnes. Nous avons vu qu'une telle syntaxe présentait de nombreux défauts. C'est pourquoi, en pratique, prenez toujours soin de lister les colonnes dans l'instruction INSERT.

**La structure de la base de données décide si une valeur peut être NULL ou non**

Dans l'exemple précédent, nous avons inséré un jeu de données incomplet en supposant que la base accepterait. Certaines colonnes se retrouvent donc avec la valeur NULL pour l'enregistrement, valeur réservée indiquant l'absence de données. C'est un choix de conception lors de la création de la base et de la table. Il est en effet possible d'ajouter une contrainte NOT NULL sur une colonne obligeant à renseigner une valeur. Dans ce cas, la requête d'insertion est rejetée par le SGBDR.

**Méthode Mettre à jour des données dans une table avec UPDATE**

La modification des données d'une table, d'une ou plusieurs lignes, répond à la syntaxe suivante :

```
1 UPDATE table_name SET col_name=value [, col_name=value] ... [WHERE conditions] ;
```

Dans cette instruction, `table_name` indique la table sur laquelle on souhaite modifier des lignes, `col_name=value` indique la nouvelle valeur `value` que l'on veut assigner à la colonne `col_name`. On peut modifier la valeur de plusieurs colonnes à la fois en séparant chaque affectation par une virgule. Nous retrouvons également la clause optionnelle `WHERE` qui permet de restreindre la modification à une ou plusieurs lignes respectant une certaine condition. En effet, sans clause `WHERE`, `UPDATE` va affecter la nouvelle valeur pour toutes les lignes de la table !

Reprenons notre exemple précédent. Nous avons inséré une donnée incomplète concernant le livre *Le nom de la rose*. Il nous manquait l'année de publication et le nom de l'auteur. Après quelques recherches, nous savons à présent qu'il a été publié en 1982 et qu'il a été écrit par l'italien Umberto Eco. Mettons à jour cet enregistrement avec l'instruction `UPDATE` :

```
1 UPDATE Book SET author_name='Eco', nationality='Italie', publication_year=1982 WHERE id=14;
```

Cette requête a pour effet de mettre à jour les valeurs des colonnes `author_name`, `nationality` et `publication_year` avec les valeurs fournies uniquement pour la ligne qui satisfait la condition `id=14`. Vous noterez ici l'intérêt de disposer d'une clef dans une table : la colonne `id` a pour seul but d'identifier de manière unique chaque enregistrement. Nous pouvons donc nous en servir pour identifier une ligne sans ambiguïté à condition de connaître sa valeur pour la ligne visée.

Après vérification, il semblerait que *Le Nom de la rose* ait été publié en 1980 et non en 1982. Essayez d'écrire la requête pour effectuer cette modification. Voici une réponse possible :

```
1 UPDATE Book SET publication_year=1980 WHERE id=14;
```

**Attention En base de données, il n'y a pas de retour en arrière (facile) possible !**

Lorsque vous faites un `UPDATE` ou un `DELETE` (que nous verrons après), faites attention de bien définir la clause `WHERE` si vous ne voulez appliquer le traitement qu'à une ou quelques lignes. Sinon votre requête s'appliquera à toute la table ! Et vous n'avez pas de bouton « annuler la dernière opération » sur une base de données : si la requête est exécutée, il est déjà trop tard ! Toute requête exécutée affecte votre base de données de manière irréversible. Faites donc bien attention à ce que vous faites et à bien vous préparer en amont lorsque vous exécutez une requête sur une base de données. « Avec de grands pouvoirs viennent de grandes responsabilités. ».

**Méthode Supprimer des données dans une table avec DELETE**

La suppression de lignes d'une table se fait avec l'instruction **DELETE** et nécessite la syntaxe suivante :

```
1 DELETE FROM table_name [WHERE conditions]
```

Cette requête est plus simple : `table_name` indique sur quelle table nous souhaitons supprimer des lignes et la clause optionnelle `WHERE` permet, comme toujours, de restreindre l'opération à une ou plusieurs lignes respectant une certaine condition. Attention, sans clause `WHERE`, la requête `DELETE` supprime toutes les lignes de la table !

Supposons que l'on souhaite retirer tous les livres écrits par des auteurs de nationalité française de notre base de données. Voici la requête que nous devrions écrire :

```
1 DELETE FROM Book WHERE nationality='France' ;
```

Cette requête aura pour effet de supprimer (de la base de données) les deux ouvrages *À la recherche du temps perdu* et *Madame Bovary*.

## IV. Introduction aux utilisations des fonctions SQL

### A. Introduction aux utilisations des fonctions SQL

SQL est un standard riche et complet. Il fournit énormément d'instructions et d'opérateurs pour définir, manipuler et interroger les données. Parmi les outils qu'il met à votre disposition, vous trouverez tout un ensemble de fonctions : fonctions numériques (ABS, COS, SIN, SQRT, ROUND, etc.), fonctions sur la manipulation de dates (ADDDATE, DATEDIFF, CURRENT\_TIME, etc.) et des fonctions de groupe (SUM, COUNT, AVG, etc.) s'appliquant sur plusieurs lignes d'une table à la fois.

Ces fonctions sont très pratiques pour faire de l'analyse de données, comme calculer des statistiques ou des valeurs minimales ou maximales d'un jeu de données.

Enfin, SQL vous fournit un moyen de regrouper les lignes pour appliquer des traitements spécifiques par lots. C'est ce que nous allons voir dans cette dernière partie introduction au SQL.

#### Méthode La fonction de groupe COUNT pour compter les lignes

Combien d'ouvrages disposons-nous dans notre table ? Quel est l'ouvrage publié le plus ancien ? Le plus récent ? Il est possible de répondre très facilement à ce genre de questions en SQL grâce à l'opération de sélection avec l'instruction SELECT que nous avons vue, combinée aux fonctions de groupe mises à disposition par SQL.

Pour compter le nombre d'enregistrements, SQL fournit la fonction COUNT. Rappelez-vous comment récupérer toutes les lignes de la table Book :

```
1 SELECT * FROM Book ;
```

Pour retourner non pas toutes les lignes, mais le nombre de lignes, on peut utiliser la fonction COUNT comme suit :

```
1 SELECT COUNT(*) FROM Book ;
```

La fonction COUNT s'applique sur l'ensemble de la table. La requête dit « sélectionne le décompte total des lignes de la table Book ». Et la base de données nous renvoie la table suivante :

COUNT(*)
12

#### Résultat et alias de colonne

Rappelez-vous, une sélection (avec SELECT) renvoie toujours une table (le modèle relationnel est une fermeture), donc ici on récupère une table avec comme nom de colonne par défaut l'opération COUNT(\*) et comme ligne le résultat 12. Vous pouvez également donner un alias au nom de la colonne de la table résultat avec la clause AS, par exemple :

```
1 SELECT COUNT(*) AS 'Nombre total de livres' FROM Book ;
```

La base vous répondra avec la table :

Nombre total de livres
12

Donner des alias à des colonnes permet de donner un peu de contexte sur le résultat.

#### Méthode Les fonctions de groupe MAX et MIN

Pour extraire des valeurs maximales ou minimales dans une table, SQL met à disposition les fonctions de groupe MAX et MIN. Sa syntaxe est la suivante :

```
1 SELECT {MAX|MIN}(col_name) FROM table_name ;
```

On passe ici en argument de MAX ou MIN le nom de la colonne `col_name` que l'on souhaite interroger. SQL trouvera lui-même la valeur MAX ou MIN dans cette colonne. Par exemple, sélectionnons l'année de publication la plus ancienne renseignée dans la base (et donc ayant la plus petite année de publication) en interrogeant la colonne `publication_year` :

```
1 SELECT MIN(publication_year) FROM Book ;
```

La base nous répondra avec la relation suivante :

MIN(publication_year)
1603

De la même manière, pour trouver l'année de publication la plus récente :

```
1 SELECT MAX(publication_year) AS 'Date de publication la plus récente' FROM Book ;
```

#### Méthode Opérateur de regroupement GROUP BY

L'opérateur de regroupement GROUP BY peut paraître plus complexe au premier abord. Nous finirons ce module avec lui. GROUP BY permet de regrouper plusieurs résultats en sous-groupes et d'appliquer une fonction (un comptage, une moyenne, max, etc.) sur chacun des sous-groupes constitués. Par exemple, supposons que nous souhaitions compter le nombre d'auteurs par nationalité. Pour cela, nous pouvons utiliser la fonction COUNT que nous avons déjà vue. Mais COUNT s'applique à toute la table, or nous voulons l'appliquer pour chaque nationalité, ou chaque sous-groupe défini par une nationalité. On voudrait dire « *Regroupe les auteurs des livres par nationalité, puis compte le nombre d'ouvrages pour chaque nationalité* ». C'est exactement à ce type d'usage que sert GROUP BY.

Comme il s'agit d'une requête de sélection de données, GROUP BY sera adjoint à une requête SELECT que vous connaissez déjà. Voici sa syntaxe :

```
1 SELECT col_name, group_operation FROM table_name [WHERE conditions] [GROUP BY col_name] ;
```

Vous connaissez déjà cette définition, nous avons seulement ajouté la clause optionnelle GROUP BY `col_name`, où `col_name` indique sur quelle colonne effectuer le regroupement. Le SGBDR regroupe dans les mêmes sous-groupes tous les enregistrements ayant la même valeur pour cette colonne. Il est possible ensuite de sélectionner la colonne sur laquelle on effectue le regroupement et n'importe quel résultat d'une fonction de groupe `group_operation`, comme COUNT.

Pour écrire la requête « *compter le nombre d'auteurs par nationalité* », nous allons donc regrouper les auteurs par nationalité avec la clause `GROUP BY nationality`. Nous allons ensuite appliquer l'opération `COUNT (*)` sur chaque sous-groupe et retourner une table contenant les colonnes `nationality` et `COUNT(*)` que nous aliaserons « *Nombre d'auteurs* ». Voici ce que donne la requête :

```
1 -- Retourne le nombre d'auteur·es par nationalité
2 SELECT nationality, COUNT(*) AS "Nombre d'auteur·es"
3 FROM Book
4 GROUP BY nationality;
```

Voici la table que vous obtiendrez en retour :

nationality	Nombre d'auteur·s
Royaume-Uni	4
États-Unis	4
Espagne	1
Colombie	1
Italie	1
Allemagne	1

Sans que vous le voyiez, la base a réparti toutes les lignes en différents sous-groupes (« *Royaume-Uni* », « *Espagne* », etc.) et a appliqué la fonction de groupe **COUNT** à chacun d'entre eux. Elle retourne ensuite le résultat sous la forme d'une table contenant les valeurs des colonnes **nationality** et de la somme sur chaque sous-groupe.

## V. Essentiel

Dans ce module d'introduction au SQL, nous avons vu que le SQL était un langage standardisé permettant d'interroger et de travailler sur des bases de données relationnelles. Plusieurs groupes et entreprises ont implémenté le standard SQL depuis sa création, et il existe une grande variété de systèmes sur le marché aujourd'hui (MySQL, Oracle, PostgreSQL, SQLite, etc.). Ces implémentations sont des Systèmes de Gestion de Base de Données Relationnelle ou SGBDR, des programmes complexes et optimisés qui savent interpréter et traiter les requêtes SQL de manière optimisée. Tous ces SGBDR implémentent le standard de manière plus ou moins avancée, car SQL évolue toujours aujourd'hui.

Pour interagir avec une base de données relationnelle, il faut lui envoyer une requête SQL, qui est une chaîne de caractères bien formée se terminant par un point-virgule. Une requête SQL permet de modifier la base de données, de mettre à jour les données, de l'interroger, mais aussi d'en gérer les accès. C'est une interface commune à tous les SGBDR. SQL est un langage déclaratif puissant, proche du langage naturel, où l'on exprime ce que l'on veut faire ou obtenir et non comment le faire, comme on le ferait dans un langage procédural.

Nous avons vu quelques instructions SQL essentielles comme l'instruction `SELECT` permettant d'interroger une base de données pour en extraire facilement de nombreuses informations. SQL ne permet pas seulement d'extraire les données stockées en base, il permet surtout de fabriquer de nouvelles informations à partir des données stockées avec les outils de filtre, de tri et les fonctions mises à disposition. Nous avons vu également les instructions `INSERT`, `UPDATE` et `DELETE` permettant de créer, modifier ou supprimer des données dans la base. Enfin, nous avons vu quelques fonctions de groupe comme `COUNT` et l'opérateur de regroupement `GROUP BY` qui permettent d'extraire des informations sur l'ensemble des données.



Nous avons vu uniquement ici les fondamentaux du langage SQL : sa nature, sa syntaxe de base et sa logique. Vous approfondirez vos connaissances en SQL lors de modules dédiés aux bases de données relationnelles.

## VI. Auto-évaluation

### A. Exercice

Vous travaillez pour une entreprise qui livre et maintient des systèmes d'information pour des cinémas dans la région Bretagne. Il vous est demandé d'analyser la base de données d'un cinéma local. On vous a extrait et mis à disposition une table Film de cette base. Elle contient les colonnes et les données suivantes :

id	title	duration_in_minutes	release_year	projection_time_in_weeks	total_entries
1	A man	122	2023	1	300
2	L'île rouge	116	2023	2	1 200
3	La petite sirène	135	2023	4	6 320
4	Les Gardiens de la galaxie 3	150	2023	4	10 256
5	Supermario Bros le film	93	2023	6	11 398
6	Casablanca	117	1942	1	457
7	Mon voisin Totoro	86	1998	1	512
8	Monty Python : Sacré Graal !	91	1975	1	417

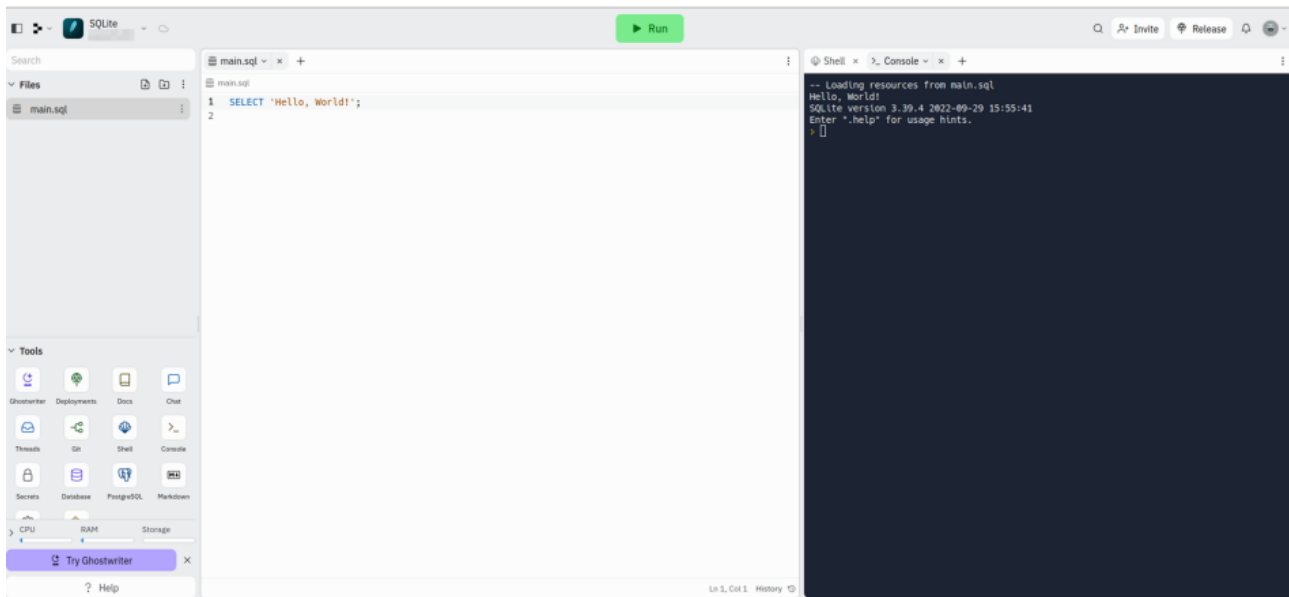
Pour réaliser cet exercice, rendez-vous sur le site web [replit.com](https://replit.com/) à l'adresse [replit](https://replit.com/)<sup>1</sup> et créez-vous un compte. Ouvrez ensuite une application **SQLite** pour accéder à un espace de travail en ligne prêt à l'emploi. Au centre, vous disposez d'un éditeur de texte où vous pouvez écrire et enregistrer vos requêtes SQL. Pour exécuter les requêtes présentes dans le fichier, cliquez sur le bouton Run ou pressez les touches Ctrl+Enter. À droite, une console vous permet d'observer la réponse du SGBDR SQLite.

Un script pour créer et initialiser la base de données vous est fourni. Copiez-collez le contenu du script et exécutez-le. Votre base et votre table **Film** sont prêtes.

[cf. cas-pratique-preparation-base.sql]

---

<sup>1</sup> <https://replit.com/>



### Question 1

[solution n°1 p.21]

Un film a été oublié. Écrivez une requête pour insérer le film *La beauté du geste*, réalisé en 2023, d'une durée d'1 h 40. Il a été projeté deux semaines et a réalisé 1513 entrées. Écrivez ensuite une requête SQL permettant d'afficher les titres et le nombre d'entrées des films sortis cette année (2023) en les classant par ordre de nombre d'entrées, en commençant par les films ayant attiré le plus de monde.

### Question 2

[solution n°2 p.21]

Le cinéma voudrait avoir des statistiques sur ses projections. Écrivez une requête SQL permettant de retourner le nombre total d'entrées réalisées, toutes projections confondues, le nombre moyen et maximum d'entrées par film et par semaine. Attention, il faut penser à diviser le nombre d'entrées par le nombre de semaines de projections avant de faire les calculs. Indice : pour calculer la somme sur une colonne, utilisez la fonction **SUM**. Pour calculer la moyenne sur une colonne, SQL vous fournit la fonction de groupe **AVG**.

## B. Test

### Exercice 1 : Quiz

[solution n°3 p.21]

#### Question 1

SQL est un Système de Gestion de Base de Données Relationnelle (SGBDR).

- ☐ Vrai
- ☐ Faux

#### Question 2

Dans un système de gestion de base de données relationnelle, une « *relation* » se traduit par :

- ☐ Une table
- ☐ Une liste
- ☐ Une base de données

#### Question 3

Soit la table Book suivante :

id	title	author_name	nationality	publication_year
1	L'île au trésor	Stevenson	Royaume-Uni	1883
2	Les Hauts de Hurlevent	Brontë	Royaume-Uni	1847
3	Madame Bovary	Flaubert	France	1869
4	Moby Dick	Melville	États-Unis	1851
5	À la recherche du temps perdu	Proust	France	1927
6	Demande à la poussière	Fante	États-Unis	1939

Parmi les requêtes SQL suivantes, laquelle permet d'obtenir cette table comme réponse le tableau suivant ?

id	title	publication_year
3	Madame Bovary	1869
6	Demande à la poussière	1939

- ☐ SELECT \* FROM Book ;
- ☐ SELECT id, title, publication\_year FROM Book WHERE id=3 OR id=6 ;
- ☐ SELECT \* FROM Book WHERE id=3 OR id=6 ;

#### Question 4

Parmi les requêtes SQL suivantes, laquelle permet de **modifier** une seule ligne d'une table :

- ☐ INSERT INTO Book (author\_name, id) VALUES ('Steinbeck', 14) ;
- ☐ UPDATE Book SET author\_name='Steinbeck' WHERE id=14;
- ☐ UPDATE Book SET author\_name='Steinbeck';

#### Question 5

Parmi les requêtes SQL suivantes, laquelle permet de supprimer l'enregistrement ayant pour **id** 1 dans la table **Book** :

- ☐ DELETE FROM Book WHERE id=1;
- ☐ DELETE FROM Book id=1 ;
- ☐ DELETE Book WHERE id=1 ;

## Solutions des exercices



**p. 18 Solution n°1**

Voici la réponse attendue :

```
1 -- Insère le film manquant
2 INSERT INTO
3 Film(
4 id,
5 title,
6 duration_in_minutes,
7 release_year,
8 projection_time_in_weeks,
9 total_entries
10 )
11 VALUES
12 (9, "La beauté du geste", 100, 2023, 2, 1513);
13
14 -- Retourne les films de l'année courante par ordre de nombre d'entrées en salle
15 SELECT
16 title,
17 total_entries
18 FROM
19 Film
20 WHERE
21 release_year = 2023
22 ORDER BY
23 total_entries DESC;
```

**p. 18 Solution n°2**

Voici la réponse attendue :


```
1 -- Retourne le nombre total d'entrées, le nombre moyen et maximum d'entrées par semaine et
   par film
2 SELECT
3 SUM(total_entries),
4 AVG(total_entries / projection_time_in_weeks),
5 MAX(total_entries / projection_time_in_weeks)
6 FROM Film;
```

**Exercice p. 18 Solution n°3****Question 1**

SQL est un Système de Gestion de Base de Données Relationnelle (SGBDR).

☐ Vrai

☒ Faux

 SQL est un standard. Il définit un langage pour interagir avec une base de données relationnelle informatique. Un SGBDR est une implémentation du standard SQL : c'est un programme (plus précisément un ensemble de programmes et de fichiers) qui doit être capable d'interpréter et d'exécuter les requêtes SQL et donc de gérer une base de données relationnelle de manière fiable et optimisée.

### Question 2

Dans un système de gestion de base de données relationnelle, une « *relation* » se traduit par :

- ☒ Une table
- ☐ Une liste
- ☐ Une base de données

**Q** Une relation est un objet mathématique défini par un ou plusieurs attributs valués. Il définit une collection valide de données, où chaque élément est appelé tuple de la relation. La traduction informatique du concept de relation est une table : la table définit la relation et porte son nom, les attributs sont représentés par des colonnes et les tuples par des lignes ou enregistrements. Une liste n'est pas un ensemble d'éléments définis par des attributs. Une base de données est un objet informatique plus complexe : elle est composée d'une ou plusieurs tables, de droits d'accès, de procédures, etc.

### Question 3

Soit la table Book suivante :

id	title	author_name	nationality	publication_year
1	L'île au trésor	Stevenson	Royaume-Uni	1883
2	Les Hauts de Hurlevent	Brontë	Royaume-Uni	1847
3	Madame Bovary	Flaubert	France	1869
4	Moby Dick	Melville	États-Unis	1851
5	À la recherche du temps perdu	Proust	France	1927
6	Demande à la poussière	Fante	États-Unis	1939

Parmi les requêtes SQL suivantes, laquelle permet d'obtenir cette table comme réponse le tableau suivant ?

id	title	publication_year
3	Madame Bovary	1869
6	Demande à la poussière	1939

- ☐ SELECT \* FROM Book ;
- ☒ SELECT id, title, publication\_year FROM Book WHERE id=3 OR id=6 ;
- ☐ SELECT \* FROM Book WHERE id=3 OR id=6 ;

**Q** La requête `SELECT id, title, publication_year FROM Book WHERE id=3 OR id=6 ;` est la bonne, car la réponse montre que l'on a demandé que les trois colonnes id, title et publication\_year et les deux lignes dont l'id est égale à 3 ou 6, ce que réalise la clause `WHERE id=3 OR id=6`.

**Question 4**

Parmi les requêtes SQL suivantes, laquelle permet de **modifier** une seule ligne d'une table :

- ☐ INSERT INTO Book (author\_name, id) VALUES ('Steinbeck', 14) ;
- ☒ UPDATE Book SET author\_name='Steinbeck' WHERE id=14;
- ☐ UPDATE Book SET author\_name='Steinbeck';

**Q** La première requête est une requête INSERT : elle permet d'ajouter une nouvelle ligne, non d'en modifier une. La dernière requête n'a pas de clause WHERE : elle va donc modifier toutes les lignes et remplacer toutes les valeurs de la colonne author\_name par « *Steinbeck* ». La bonne réponse est donc UPDATE Book SET author\_name='Steinbeck' WHERE id=14; qui effectue une mise à jour sur une seule ligne sélectionnée avec la clause WHERE.

**Question 5**

Parmi les requêtes SQL suivantes, laquelle permet de supprimer l'enregistrement ayant pour **id** 1 dans la table **Book** :

- ☒ DELETE FROM Book WHERE id=1;
- ☐ DELETE FROM Book id=1 ;
- ☐ DELETE Book WHERE id=1 ;

**Q** DELETE FROM Book WHERE id=1; est la bonne réponse. Les autres requêtes ont une erreur de syntaxe (mot FROM ou WHERE manquant).