# Chapitre 8 - Le langage SQL

### Objectifs :

- ▷ Identifier les composants d'une requête.
- ▷ Savoir utiliser des requêtes d'interrogation et de mise à jour d'une base de données.
- ▷ Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : `SELECT`, `FROM`, `WHERE`, `JOIN`.
- ▷ Construire des requêtes d'insertion et de mise à jour à l'aide de : `UPDATE`, `INSERT`, `DELETE`.

## 1 Introduction

Dans le chapitre précédent sur le modèle relationnel, nous avons découvert la structure mathématique des bases de données. Maintenant que nous avons vu comment celles-ci sont organisées, nous allons interagir avec elles.

Les SGBD jouent le rôle d'interface entre l'être humain et la base de données. Par l'intermédiaire de requêtes, l'utilisateur va **consulter** ou **mettre à jour** la base de données.

Le langage utilisé pour communiquer avec le SGBD est le **langage SQL**, (pour *Structured Query Langage* ou langage de requêtes structurées).

Les SGBD les plus utilisés sont basés sur le **modèle relationnel**. Parmi eux, on peut trouver :


La quasi-totalité de ces SGBD fonctionnent avec un modèle **client-serveur**.

Cette année, nous allons travailler principalement avec le langage *SQLite* qui peut lui s'utiliser directement sans démarrer un serveur : la base de données est entièrement représentée dans le logiciel utilisant *SQLite*. Sa simplicité d'utilisation en fera notre choix pour illustrer cette présentation du langage SQL.

Il existe de nombreuses façons d'écrire des requêtes SQL avec une base de données *Sqlite* :

- ▷ Avec un logiciel muni d'une interface graphique comme *DB Browser for SQLite*.
- ▷ Via un langage de programmation comme Python grâce au module `sqlite3`
- ▷ En console, en lançant : `sqlite3 nom_de_ma_bdd.db`.
- ▷ En ligne avec des sites comme *SQLite online* ou *SQLite Viewer*,

# 2 Le langage SQL

Les instructions SQL s'écrivent d'une manière qui ressemble à celle de phrases ordinaires en anglais. C'est un **langage déclaratif**, c'est-à-dire qu'il permet de décrire le résultat escompté, sans décrire la manière de l'obtenir.

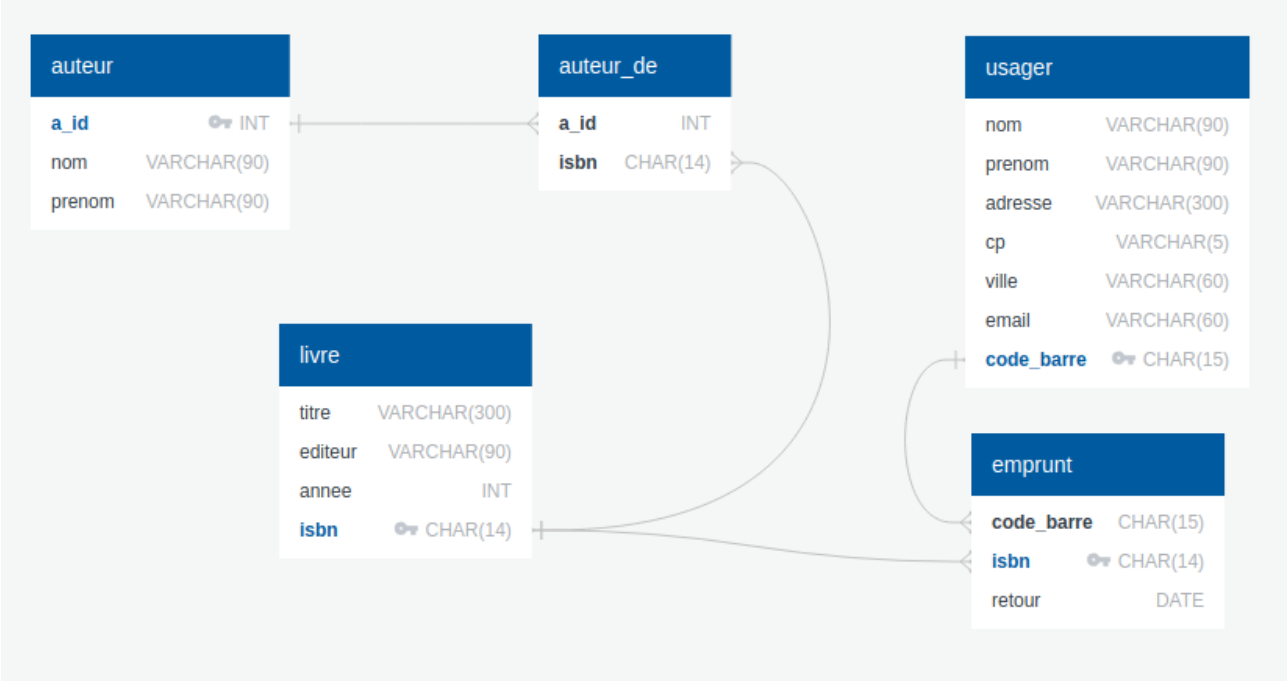
Le langage SQL n'est **pas sensible à la casse**, mais a l'habitude d'écrire les instructions en majuscules, on peut écrire les instructions sur plusieurs lignes avec ou sans indentation et chaque instruction doit être terminée par un **point-virgule**.

Nous verrons cette année les instructions de manipulation du contenu de la base de données qui commencent par les mots clés :

- `SELECT` : recherche de contenu ;
- `UPDATE` : modification,
- `INSERT` : ajout,
- `DELETE` : suppression.

Pour illustrer ce cours, nous étudierons les données situées dans la base de données `livres.db`.

Voici le diagramme relationnel de cette base :




— Les clés primaires sont en bleu (suivi d'une icone de clé)
— Les clés étrangères sont en noir et reliées à leur clé primaire.

# 3 Sélection de données

## 3.1 Comment interroger une base de données ?

### 3.1.1 Requête basique : SELECT ... FROM ...

**Exemple 0**

```
SELECT titre FROM livre ;
```

**Traduction :**
On veut tous les titres de la table `livre`.

**Remarques :**

- Les mots-clés SQL sont traditionnellement écrits en MAJUSCULES.
- Le ; signale la fin de l'instruction. Il peut donc être omis s'il n'y a pas d'instructions enchaînées (ce qui sera toujours notre cas).
- L'indentation n'est pas syntaxique (pas comme en Python). On peut donc faire des retours à la ligne et des indentations pour rendre le code plus lisible.

### 3.1.2 Requête filtrée : SELECT ... FROM ... WHERE ...

**Exemple 1**

```
SELECT titre FROM livre WHERE annee >= 1990;
```

**Traduction :**
On veut les titres de la table `livre` qui sont parus après (ou en ) 1990 ;

**Remarques :**

- Le mot-clé `WHERE` doit être suivi d'un booléen. Les opérateurs classiques `=` , `!=`, `>`, `>=`, `<`, `<=` peuvent être utilisés, mais aussi le mot-clé `IN`.

### 3.1.3 Requête avec plusieurs possibilités : WHERE ... IN ...

**Exemple 1bis**

```
SELECT titre FROM livre WHERE annee IN (1990, 1991, 1992);
```

**Traduction :**
On veut les titres de la table `livre` qui sont parus en 1990, 1991 ou 1992.

### 3.1.4 Requête avec booléens : AND - OR

**Exemple 2**

```
SELECT titre FROM livre WHERE   annee >= 1970 AND
                                annee <= 1980 AND
                                editeur = 'Dargaud';
```

**Traduction :**
On veut les titres de la table `livre` qui sont parus entre 1970 et 1980 chez l'éditeur Dargaud ;

### 3.1.5 Requête approchée : LIKE

**Exemple 3**

```
SELECT titre FROM livre WHERE titre LIKE '%Asterix%';
```

**Traduction :**
On veut les titres de la table `livre` dont le titre contient la chaîne de caractères `Astérix`.

**Remarque :**

- ▷ Le symbole % est un joker qui peut symboliser n'importe quelle chaîne de caractères.

### 3.1.6 Plusieurs colonnes

**Exemple 4**

```
SELECT titre, isbn FROM livre WHERE annee >= 1990;
```

**Traduction :**
On veut les titres et les ISBN de la table `livre` qui sont parus après 1990.

**Remarque :**

- ▷ Le symbole % est un joker qui peut symboliser n'importe quelle chaîne de caractères.

### 3.1.7 Toutes les colonnes : *

**Exemple 5**

```
SELECT * FROM livre WHERE annee >= 1990;
```

**Traduction :**
On veut les titres et les ISBN de la table `livre` qui sont parus après 1990.

**Remarque :**

- ▷ L'astérisque * est un joker (*wildcard* en anglais).

### 3.1.8 Renommer les colonnes : AS

**Exemple 6**

```
SELECT titre AS titre_du_livre FROM livre WHERE annee >= 1990;
```

**Traduction :**
Lors de l'affichage du résulats et dans la suite de la requête (important), la colonne `titre` est renommée `titre_du_livre`.

**Remarque :**

- ▷ L'alias `AS` sera souvent utilisé pour raccourcir un nom, notamment lors des jointures de plusieurs tables.

# 4 Opérations sur les données

## 4.1 Sélection avec agrégation

Les requêtes effectuées jusqu'ici ont juste sélectionné des données grâce à différents filtres : aucune action à partir de ces données n'a été effectuée.

Nous allons maintenant effectuer des **opérations** à partir des données sélectionnées.

### 4.1.1 Compter : COUNT

**Exemple 7**

```
SELECT COUNT(*) AS total FROM livre
WHERE titre LIKE "%Asterix%";
```

**Traduction :**
On veut compter le nombre d'enregistrements de la tables `livre` comportant le mot `"Astérix"`. Le résultat sera le seul élément d'une colonne nommée `total`.

### 4.1.2 Additionner : SUM

**Exemple 8**

```
SELECT SUM(annee) AS somme FROM livre
WHERE titre LIKE "%Asterix%";
```

**Traduction :**
On veut additionner les années des livres de la tables `livre` comportant le mot `Astérix`. Le résultat sera le seul élément d'une colonne nommée `somme`.

**Remarque :**

- ▷ Attention : dans notre cas précis, ce calcul n'a aucun sens car on additionne des années...

### 4.1.3 Faire une moyenne : AVG

**Exemple 9**

```
SELECT AVG(annee) AS moyenne FROM livre
WHERE titre LIKE "%Asterix%";
```

**Traduction :**
On veut calculer la moyenne des années de parution des livres de la table `livre` comportant le mot `Astérix`. Le résultat sera le seul élément d'une colonne nommée `moyenne`.

**Remarque :**

- ▷ Attention : là encore, ce calcul n'a aucun sens...

### 4.1.4 Trouver les extremums : MIN, MAX

**Exemple 10**

```
SELECT MIN(annee) AS minimum FROM livre
WHERE titre LIKE "%Asterix%";
```

**Traduction :**
On veut trouver la plus petite valeur de la colonne `annee` parmi les livres de la table `livre` comportant le mot `Astérix`. Le résultat sera le seul élément d'une colonne nommée `minimum`. Le fonctionnement est identique avec `MAX` pour la recherche du maximum.

### 4.1.5 Classer des valeurs : ORDER BY, ASC, DESC, LIMIT

**Exemple 11**

```
SELECT titre, annee FROM livre
WHERE titre LIKE "%Asterix%"
ORDER BY annee DESC;
```

**Traduction :**
On veut afficher tous les albums d'Astérix, et leur année de parution, classés par année décroissante.

**Remarques :**

- **Comportement par défaut** : Si le paramètre `ASC` ou `DESC` est omis, le classement se fait par ordre croissant (donc `ASC` est le paramètre par défaut).
- **Utilisation de `LIMIT`** : Le mot-clé `LIMIT` (suivi d'un nombre) permet de limiter le nombre de résultats affichés. Ainsi la requête

### 4.1.6 Suppression des doublons : DISTINCT

**Exemple 12**

```
SELECT DISTINCT editeur FROM livre;
```

**Traduction :**
On veut la liste de tous les éditeurs. Sans le mot-clé `DISTINCT`, beaucoup de doublons apparaîtraient.

# 5 Les jointures

Parfois, on a besoin aussi d'effectuer des requête complexes, comme par exemple des **recherches croisées** sur les tables. Pour cela, on va devoir faire des **jointures**.

**Définition**

En SQL, une **jointure** est une opération qui permet de combiner des données de deux tables (ou plus) en fonction d'une ou plusieurs **colonnes communes**.

Cela permet de relier des informations provenant de différentes tables pour effectuer des **requêtes complexes** et obtenir des résultats plus riches en informations.

Les jointures sont couramment utilisées pour **récupérer des données à partir de plusieurs sources** de données liées entre elles, en utilisant des critères de correspondance spécifiques.

## 5.1 Exemple

Par exemple, si on exécute la requête suivante sur le contenu de la table `emprunt` :

```
SELECT * FROM emprunt;
```

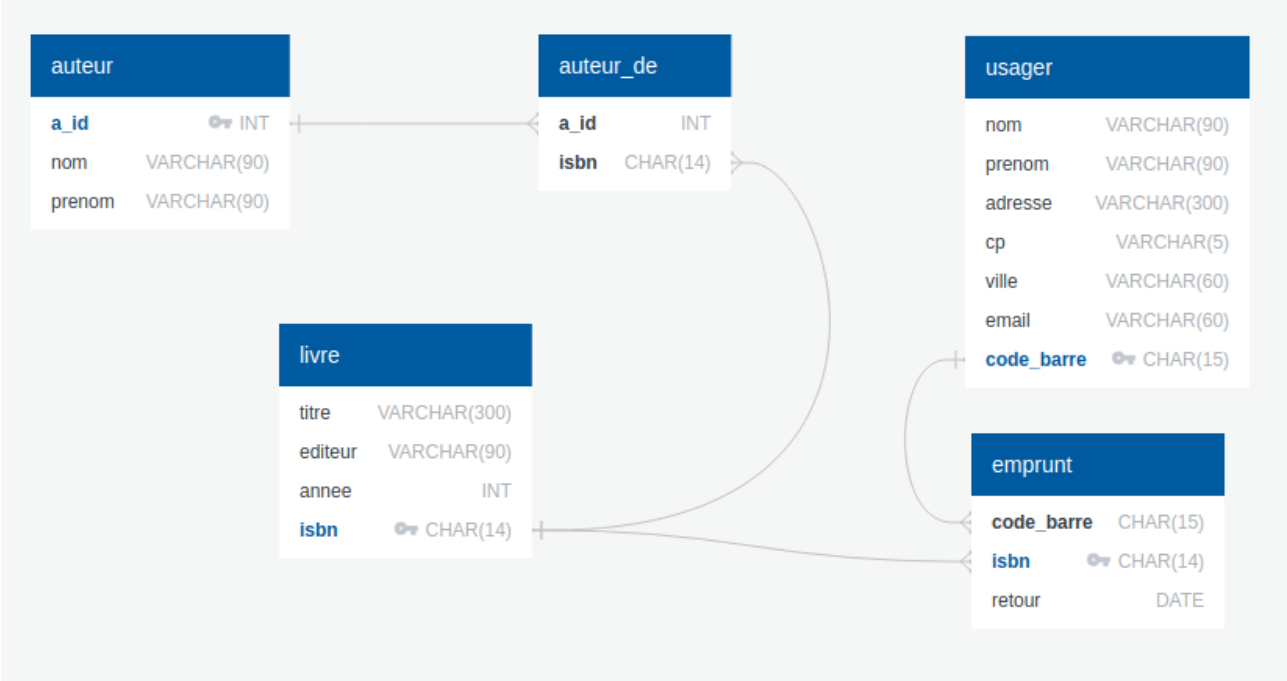
On obtient ceci :

| | code_barre | isbn | retour |
|---|---|---|---|
| 1 | 421921003090881 | 978-2081358881 | 2020-04-28 |
| 2 | 421921003090881 | 978-2207249123 | 2020-04-28 |
| 3 | 421921003090881 | 978-2824709420 | 2020-04-28 |
| 4 | 137332830764072 | 978-2352879183 | 2020-02-20 |

17 enregistrements ramenés en 0ms depuis : SELECT * FROM emprunt;

Le contenu est peu lisible... Qui a emprunté quel livre ? Qui est l'auteur ou le titre du livre emprunté ?

En effet, le schéma relationnel est le suivant :




Pour que la table `emprunt` soit lisible, il faudrait (dans un premier temps) que l'on affiche à la place de l'ISBN le titre de l'ouvrage. Or ce titre est disponible dans la table `livres`

On va donc procéder à une **jointure** de ces deux tables.

### 5.1.1 Jointure de 2 tables : JOIN

Exemple 13

```
SELECT livre.titre, emprunt.code_barre, emprunt.retour FROM emprunt
        JOIN livre ON emprunt.isbn = livre.isbn;
```

**Traduction :**
Comme plusieurs tables sont appelées, nous préfixons chaque colonne avec le nom de la table. Nous demandons ici l'affichage de la table `emprunt`, mais où on aura remplacé l'ISBN (peu lisible) par le titre du livre.

**Résultat :**

| | titre | code_barre | retour |
|---|---|---|---|
| 1 | Mrs Dalloway | 421921003090881 | 2020-04-28 |
| 2 | Fondation et Empire | 421921003090881 | 2020-04-28 |
| 3 | Le Journal d'un fou | 421921003090881 | 2020-04-28 |
| 4 | Guerre et Paix | 137332830764072 | 2020-02-20 |
| 5 | Les Voyages de Gulliver | 137332830764072 | 2020-02-20 |
| 6 | Gargantua et Pantagruel | 137332830764072 | 2020-02-20 |

```
17 enregistrements ramenés en 1ms depuis : SELECT livre.titre, emprunt.code_barre,
emprunt.retour FROM emprunt
        JOIN livre ON emprunt.isbn = livre.isbn;
```

**Remarques :**

L'expression

```
JOIN livre ON emprunt.isbn = livre.isbn
```

doit se comprendre comme ceci :

- ▷ on «invite» la table `livres` (dont on va afficher la colonne `titre`)
- ▷ la correspondance entre la table `livres` et la table `emprunt` doit se faire sur l'attribut ISBN, qui est la clé primaire de `livres` et une clé étrangère d'`emprunts`.

Il est donc très important de spécifier ce sur quoi les deux tables vont se retrouver (ici, l'ISBN)

### 5.1.2 Jointure de 3 tables : JOIN

Le résultat précédent a permis d'améliorer la visibilité de la table `emprunt`, mais il reste la colonne `code_barre` qui est peu lisible. Nous pouvons la remplacer par le titre du livre, en faisant une nouvelle jointure, en invitant maintenant les deux tables `livre` et `usager`.

**Exemple 14**

```
SELECT u.nom, u.prenom, l.titre, e.retour FROM emprunt AS e
    JOIN livre AS l ON e.isbn = l.isbn
    JOIN usager AS u ON e.code_barre = u.code_barre;
```

**Traduction :**
Il faut bien comprendre que la table principale qui nous intéresse ici est «`emprunts`», mais qu'on modifie les valeurs affichées en allant chercher des correspondances dans deux autres tables.
Notez ici que des alias sont donnés aux tables (par `AS`) afin de faciliter l'écriture.

**Résultat :**

| | nom | prenom | titre | retour |
|---|---|---|---|---|
| 1 | MOREAU | ALAIN | Mrs Dalloway | 2020-04-28 |
| 2 | MOREAU | ALAIN | Fondation et Empire | 2020-04-28 |
| 3 | MOREAU | ALAIN | Le Journal d'un fou | 2020-04-28 |
| 4 | DUBOIS | PHILIPPE | Guerre et Paix | 2020-02-20 |
| 5 | DUBOIS | PHILIPPE | Les Voyages de Gulliver | 2020-02-20 |

```
17 enregistrements ramenés en 4ms depuis : SELECT u.nom, u.prenom, l.titre, e.retour FROM
emprunt AS e
        JOIN livre AS l ON e.isbn = l.isbn
        JOIN usager AS u ON e.code_barre = u.code_barre;
```

# 6 Création et modification d'une base de données

L'objectif est maintenant de créer la table suivante :

| id | Nom | Maths | Anglais | NSI |
|---|---|---|---|---|
| 1 | Alice | 16 | 11 | 17 |
| 2 | Bob | 12 | 15 | 10 |
| 3 | Charles | 9 | 11 | 18 |

## 6.1 Création d'une table : CREATE TABLE

Exemple 15

```
CREATE TABLE Table_notes (
        Id INTEGER PRIMARY KEY,
        Nom TEXT,
        Maths INTEGER,
        Anglais INTEGER,
        NSI INTEGER
        );
```

**Remarques :**

- C'est l'utilisateur qui spécifie, éventuellement, quel attribut sera une **clé primaire**.
- Dans *DB Browser*, il faut avoir au préalable créé une nouvelle base de données.

## 6.2 Insertion de valeurs : INSERT INTO, VALUES

Exemple 16

```
INSERT INTO Table_notes VALUES  (1, 'Alice', 16, 11, 17),
        (2, 'Bob', 12, 15, 10),
        (3, 'Charles', 9, 11, 18);
```

**Traduction :** On ajoute 3 élèves dans la table `Tables_notes` avec leurs notes respectives.

**Remarques :**

Si on essayer d'insérer un quatrième enregistrement ayant le même `Id` qu'un autre élève.

Par exemple :

```
INSERT INTO Table_notes VALUES  (3, 'Denis', 18, 10, 12);
```

Alors, la contrainte de relation est violée : le SGBD «protège» la base de données en n'acceptant pas la proposition d'insertion.

La base de données **n'est pas modifiée**.

```
UNIQUE constraint failed: Table_notes.Id: INSERT INTO Table_notes VALUES  (3, 'Denis',
18, 10, 12);
```

## 6.3 Modification d'une valeur UPDATE, SET

Exemple 17

Pour modifier la note de Maths d'`Alice` :

```
UPDATE Table_notes SET Maths = 18 WHERE Nom = 'Alice';
```

## 6.4 Suppression d'un enregistrement : DELETE

Exemple 18

Pour supprimer totalement la ligne concernant `Charles` :

```
DELETE FROM Table_notes WHERE Nom = 'Charles';
```

**Remarque :**
Si une autre table contient par exemple l'attribut `Id` comme clé étrangère, et si l'`Id` de Charles fait partie de cette table, le SGBD refusera de supprimer cette ligne, afin de ne pas violer la contrainte de référence.

## 6.5 Suppression totale d'une table : DROP TABLE

Exemple 19

Pour supprimer totalement et défitivement la table :

```
DROP TABLE Table_notes;
```

**Remarque :** Si une autre table est reliée à `Table_notes` par une clé étrangère, la suppression sera bloquée par le SGBD.

# 7 Exercices

**Exercice 1 :** Base de données *Prix Nobel*

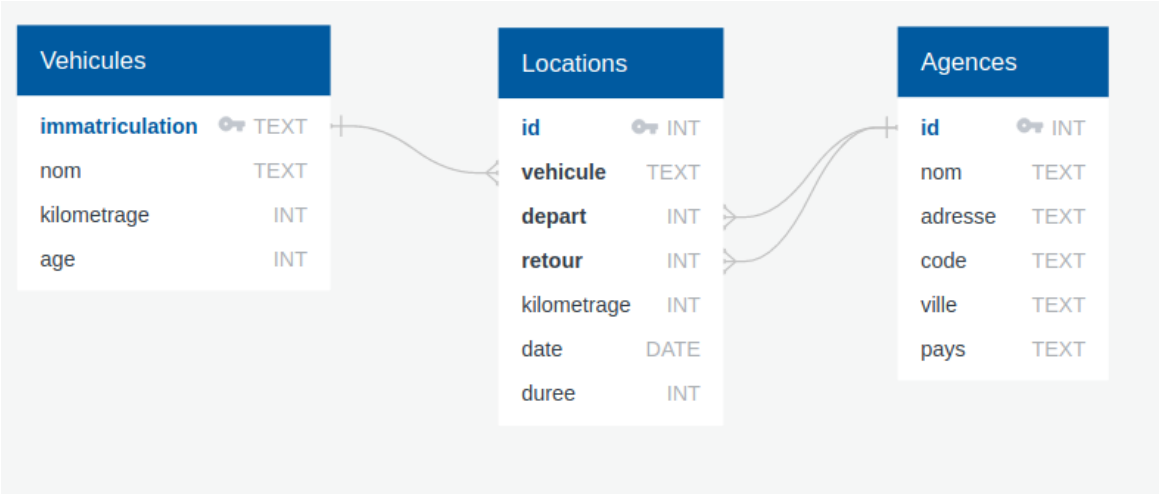
- ▷ Aller sur le site de la base de données des prix Nobel
- ▷ Répondre aux questions interactives pour réaliser les 15 requêtes demandées (en anglais).

**Exercice 2 :** Base de données *World*

- ▷ Aller sur le site de la base de données de la population mondiale
- ▷ Répondre uniquement aux 5 premières questions interactives (en anglais).

**Exercice 3 :** *Gestion d'un réseau d'agences de location de voitures*

La base de données `locations.db` contient les tables `Agences`, `Locations`, `Vehicules`.

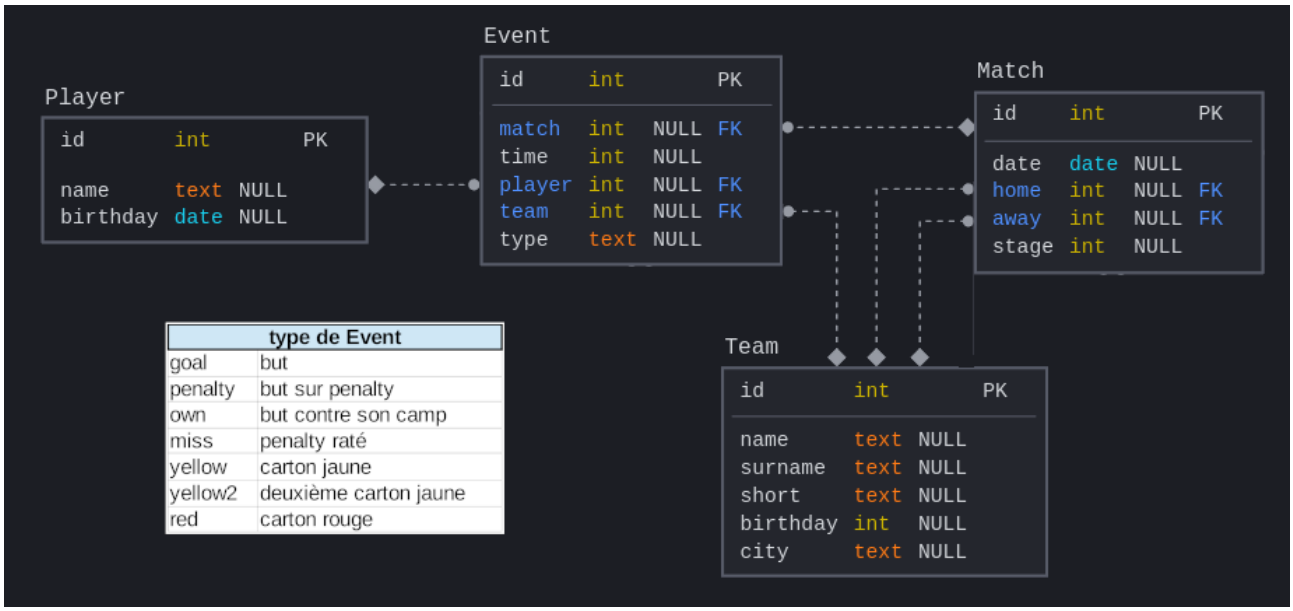



Ecrire les requêtes SQL permettant de répondre aux questions ci-dessous :

1. Visualiser toute la relation `Agences`.
2. Lister uniquement les noms des agences et de leur ville.
3. Lister les noms des agences de la ville de Lorient.
4. Lister les noms des agences du département du Morbihan (code postal `56***`) ainsi que les codes postaux en utilisant par exemple un `WHERE LIKE`.
5. Déterminer le nombre de voitures que vous possédez.
6. Déterminer l'âge minimum et maximum de vos véhicules.
7. Quels sont la marque et le modèle de votre dernière acquisition qui date de trois mois ?
8. Quel est le kilométrage maximum des véhicules ?
9. Quel est le kilométrage moyen des véhicules ?
10. Afficher toute la flotte de véhicules par ordre décroissant de kilométrage.
11. Visualiser toute la relation `Locations`.
12. Déterminer le nombre de locations effectuées avec changement d'agence.
13. Déterminer le nombre total de kilomètres effectués durant les locations.
14. Lister toutes les locations en y associant les caractéristiques du véhicule.
15. Afficher le nom et l'immatriculation du véhicule ainsi que la date de la location et le kilométrage réalisée pour chacune des locations.
16. Afficher une seule fois le nom et l'immatriculation des véhicules ayant déjà été loués.
17. Afficher les locations du véhicule immatriculé AB-224-BA en précisant le nom de l'agence de départ ainsi que la ville de départ dans l'ordre chronologique des locations.

**Exercice 4 :** *Championnat de France de football*

La base de données `soccer.db` contient les tables `Team`, `Match`, `Event`, `Player`.



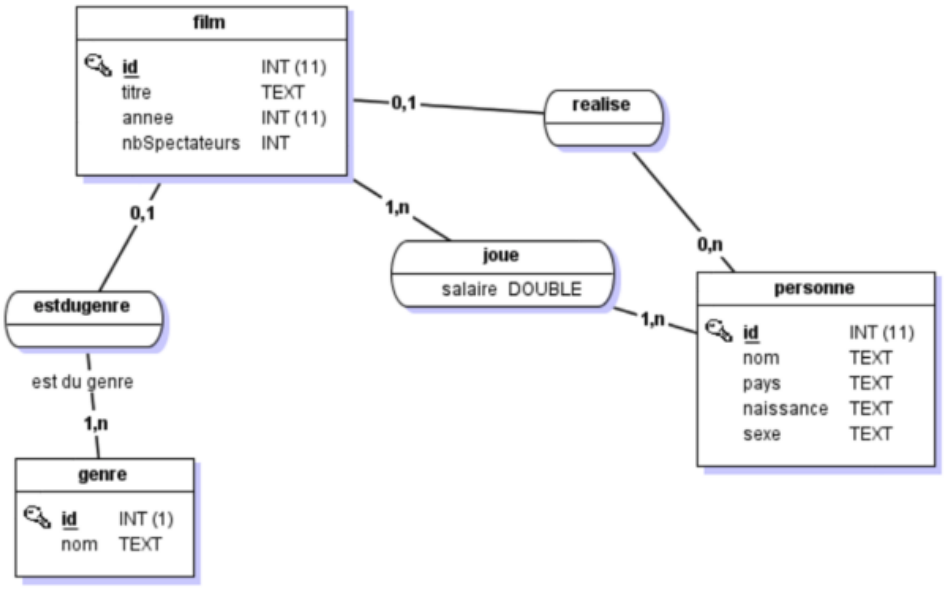

| type de Event | |
|---|---|
| goal | but |
| penalty | but sur penalty |
| own | but contre son camp |
| miss | penalty raté |
| yellow | carton jaune |
| yellow2 | deuxième carton jaune |
| red | carton rouge |

Ecrire les requêtes SQL permettant de répondre aux questions ci-dessous :

1. Combien d'équipes ont participé à ce championnat ?
2. Lister les noms des clubs ainsi que leur date de création dans l'ordre chronologique de leur création.
3. Combien de pénaltys ont été marqués ?
4. Combien de pénaltys ont été sifflés ?
5. Combien de cartons ont été distribués ?
6. Combien de buts ont été marqués ?
7. Afficher tous les renseignements sur les 10 cartons rouges obtenus le plus rapidement pendant un match.
8. Donner le nom du joueur qui a obtenu le carton rouge le plus rapidement.

**Exercice 5 :** *Films*

On dispose de la base de données `films.db` suivante :



Elle donne naissance aux tables du modèle relationnel suivant :

**Table film**

| name | type | notnull |
|---|---|---|
| id | int(11) | 1 |
| titre | TEXT | 0 |
| annee | int(11) | 1 |
| nbSpectateurs | int(11) | 0 |
| idRealisateur | int(11) | 0 |
| idGenre | int(11) | 0 |

**Table genre**

| name | type | notnull |
|---|---|---|
| id | int(11) | 1 |
| nom | text | 1 |

**Table joue**

| name | type | notnull |
|---|---|---|
| idActeur | int(11) | 1 |
| idFilm | int(11) | 1 |
| salaire | double | 0 |

**Table personne**

| name | type | notnull |
|---|---|---|
| id | int(11) | 1 |
| nom | text | 1 |
| pays | text | 0 |
| naissance | date | 0 |
| sexe | text | 0 |

Ecrire les requêtes qui permettent d'afficher :

1. Dix noms de personnes (réalisateurs ou acteurs)
2. La liste des titres des films commençant par la lettre `M` et se terminant par la lettre `r` ou `s`
3. La liste des titres des films sortis entre 2002 et 2004 (sur les 3 années)
4. La Liste des noms et dates de naissance des personnes connues de la base dont le prénom commence par '`Ro`' et de nationalité française (le pays vaut '`France`'), par ordre alphabétique
5. Les pays (une fois chacun) dont ont connaît au moins une personne de sexe féminin (donc une réalisatrice ou une actrice)
6. La liste des films sortis en 2006 en indiquant le titre, et le genre (en texte), par ordre alphabétique des titres
7. La liste des drames sortis strictement avant 1970 en donnant le titre et l'année de sortie
8. La liste alphabétique des acteurs du film `"Le convoyeur"`
9. La table contenant seulement l'année de sortie du plus ancien film, et l'année de sortie du film le plus récent
10. La moyenne du nombre de spectateurs par film
11. La liste des 10 films comptant le plus d'acteurs
12. La liste des films (avec leur année de sortie) sortis au moins 50 ans après le film le plus ancien de la base.

**Exercice 6 :** Base de données *Euro 2012*

- ▷ Aller sur le site de la base de données de l'Euro 2012
- ▷ Répondre aux questions interactives pour réaliser les 13 requêtes demandées (en anglais).

**Exercice 7 :** *Livres*

Afin de créer une nouvelle base de données de livres, suivre les instructions pas à pas du site suivant (activités 3.1 à 3.16).

**Exercice 8 :** *QCM de révisions*

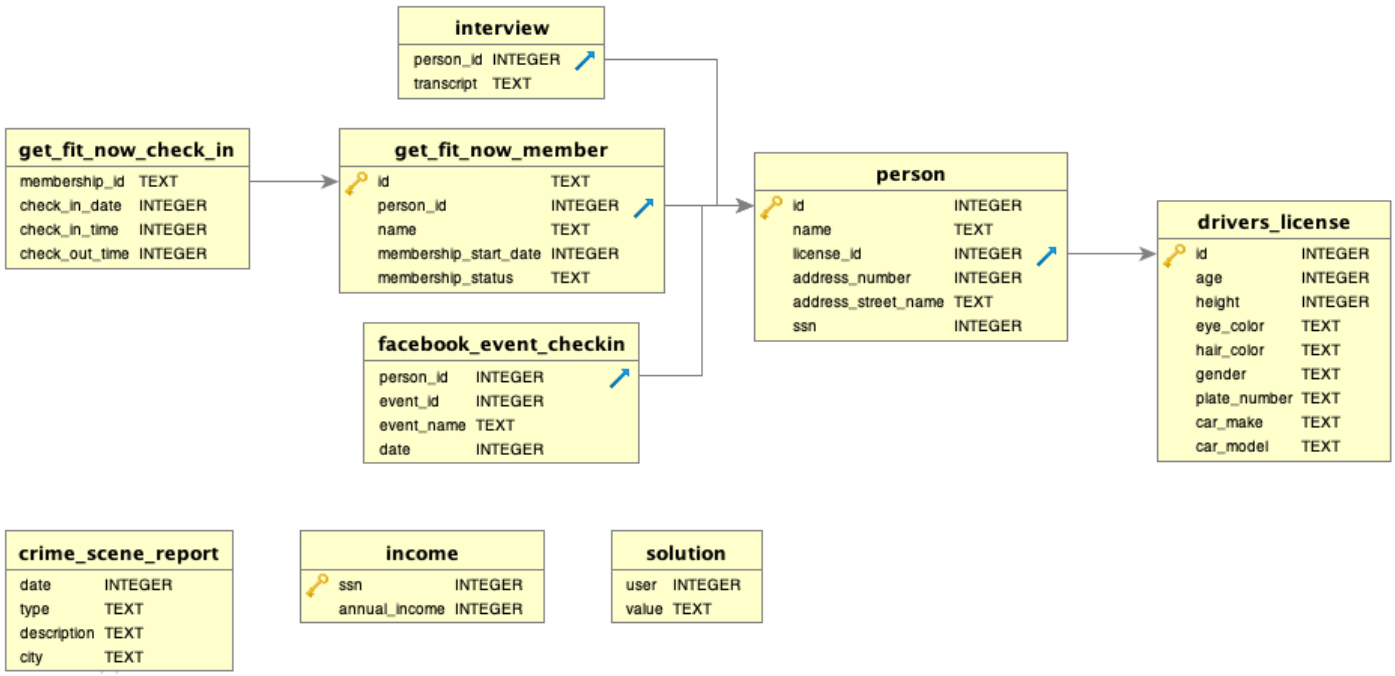
Répondre au QCM suivant sur les principales commandes SQL à connaître. (en anglais)

**Exercice 9 :** *SQL Murder Mystery*

Cet exercice en ligne est proposé le **Knight Lab** de l'université américaine *Northwerstern University*.


Le point de départ de l'histoire : un meurtre a été commis dans la ville de *SQL City* le 15 janvier 2018.

À partir de ce point de départ et d'une base de données dont le diagramme est donné ci-dessous, il s'agit de trouver le meurtrier.




Rendez-vous sur cette page, et bonne enquête à coups de requêtes !