Le langage SQL

Sommaire

Sommaire

Sommaire

Introduction

- La mise œuvre pratique du modèle relationnel de données vu au chapitre précédent est réalisée par un logiciel, le Système de Gestion de Bases de Données Relationnel, ou SGBDR (en anglais DBMS ou RDBMS, pour Relational Database Management System).
- La grande majorité des SGBDR utilisent les requêtes du langage SQL pour permettre à l'utilisateur de communiquer avec le SGBDR et de manipuler les données (SQL : Structured Query Language ou langage de requêtes structuré).
- La connaissance approfondie de SQL dépasse largement le cadre de ce cours. On abordera ici seulement quelques notions de base.
- Remarque : chaque SGBD implémente un « dialecte » légèrement différent de la norme SQL , ce qui oblige parfois à quelques modifications.

- L'architecture sur laquelle repose le SGBDR est celle de client-serveur [1] :

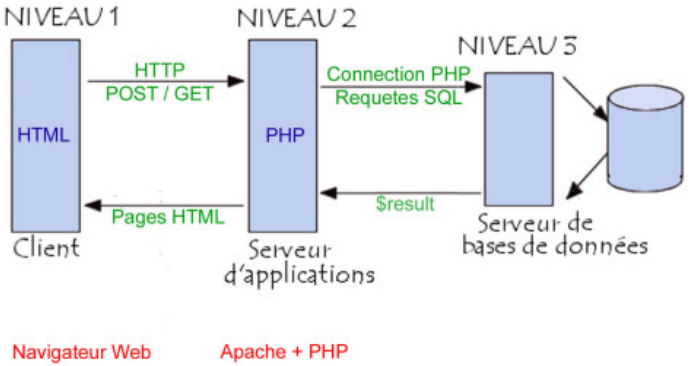


Figure 1 – l'architecture trois tiers

- Deux types de requêtes SQL existent :
  1. mises à jour : création/suppression de relations, ajout/modification/suppression d'entités dans les relations ;
  2. consultation de données : provenants d'une ou plusieurs relations, selon certains critères.
- Ici on choisit de découvrir les bases de SQL par l'interface de phpMyAdmin , qui permet à la fois d'interagir en ligne de commande et graphiquement (dans une page web).

1. Vous pouvez visionner cette vidéo sur le sujet du développement web, qui explique assez bien les trois tiers en jeu dans les applications web.

Sommaire

Vocabulaire et syntaxe du langage SQL

- Dans le vocabulaire de SQL et des BdD :
  - une relation est appelée une *table* ;
  - une entité est appelée une *ligne* ;
  - un attribut d'une entité est appelé une *colonne*.
- Quelques points de syntaxe SQL :
  - insensibilité à la casse (majuscule/minuscule). Bonne pratique : écrire les mots-clés SQL en majuscule ;
  - caractères espaces et indentation non significatifs ;
  - point virgule en fin de requête ;
  - les noms de table, d'attributs ne peuvent contenir d'espace. Bonne pratique : utiliser le caractère «underline» comme séparateur, exemple : `id_e` ;
  - ne pas utiliser des mots réservés comme nom de table ou de colonne, par exemple le mot `table`[2].

2. On peut contourner cela en plaçant le mot entre des guillemets obliques (backticks en anglais), comme ceci : `` `table` ``. Ceci-dit, les guillemets obliques sont optionnels dans la syntaxe SQL , donc autant les éviter. Par contre les guillemets droits sont obligatoires autour des valeurs de type (chaîne de) caractère(s).

Sommaire

Création de BdD et de tables

- Exemple :

```
CREATE DATABASE annuaire DEFAULT CHARACTER SET utf8
                         DEFAULT COLLATE utf8_general_ci;
```

- Ci-dessus :
  - `annuaire` est le nom de la BdD ;
  - `utf8` est l'encodage qui sera utilisé (l'ensemble des caractères disponibles) ;
  - `utf8_general_ci` représente l'assemblage [3], c'est à dire les règles qui seront utilisées pour comparer les chaînes de caractères (exemple : «é» est-il avant ou après «e», ou sont-ils considérés égaux ?).
  - Pour supprimer une BdD (créée par erreur) :

```
DROP DATABASE annuaire;
```

3. le suffixe `_ci` du nom de cet assemblage signifie «case insensitive», soit «insensible à la casse». Cela signifie entre autre que, pour cet assemblage, les majuscules et minuscules ont le même classement (et sont donc confondues).

Remarques :

- attention : une suppression est irréversible ;
- pour lancer la pile LAMP [4], c'est à dire démarrer localement un serveur web et un serveur PHP , taper dans un terminal :

  ```
  eleve@g10-nsi-2:~$ sudo /opt/lampp/lampp start
  ```

  puis taper le mot de passe du compte `eleve`. On peut alors accéder à la page d'accueil de LAMP à l'url : localhost et à celle de phpMyAdmin à l'url : localhost/phpmyadmin
- dans l'interface graphique de phpMyAdmin , très chargée, être attentif à ce qu'on fait (et éviter de cliquer n'importe où...), privilégier les commandes SQL ...dans l'onglet SQL même si tout peut être fait soit en ligne de commande, soit graphiquement.

4. Linux Apache MariaDB PHP/Perl/Python. Le langage Perl est en réalité de moins en moins utilisé pour le développement Web et 80% des sites utilisent désormais PHP.

Ex. 1 La finalité de cet exercice est d'importer la BdD `mediatheque` dans phpMyAdmin , de comprendre sa structure et de faire une première requête sur cette BdD.

Pour cela, après avoir lancé le serveur LAMP et téléchargé le fichier `mediatheque.sql` :

1. Ouvrir le fichier `mediatheque.sql`, lire les huit premières commandes SQL de ce fichier. Quel est leur rôle ? Énoncer toutes les contraintes d'intégrité associées aux tables créees.
2. Ouvrir un navigateur à l'url /localhost/phpmyadmin. Dans la page web de phpMyAdmin sélectionner l'onglet `Importer`. Dans celui-ci :
   - dans la zone `Fichier à importer`, cliquer sur le bouton `Parcourir`, puis en naviguant dans la fenêtre qui s'ouvre se déplacer jusqu'au répertoire où a été téléchargé le fichier `mediatheque.sql` et sélectionner celui-ci ;
   - laisser (dans cette zone de saisie et dans toutes les autres) tous les choix à leur valeur par défaut ;
   - en bas de la page (si besoin faire défiler pour le faire apparaître), cliquer sur le bouton `Importer`.

3. un message doit apparaître en haut de la page indiquant : `L'importation a réussi, 447 requêtes exécutées (mediatheque.sql)`.
   On vérifie, dans le panneau de gauche de phpMyAdmin qu'une nouvelle BdD a été créée, nommée `mediatheque` ;
4. en sélectionnant `mediatheque`, sa structure apparaît. Vérifier qu'elle est composée de 5 tables :
   - `auteur`
   - `auteur_de`
   - `emprunt`
   - `livre`
   - `usager`
5. Dans phpMyAdmin , sélectionner la table `auteur`, puis son onglet `structure`. Lire le nom des colonnes de cette table, comment repère-t-on celle qui est une clef primaire ? Mêmes questions pour la table `auteur_de`. Comment repère-t-on une clef étrangère ?
   Expliquer le rôle respectif des tables `auteur` et `auteur_de`.

6. Dans la partie basse de la fenêtre de phpMyAdmin , agrandir la zone intitulée `Console de requêtes SQL` . Si besoin effacer les requêtes précédentes, puis taper la requête

```
SELECT COUNT(*) AS total FROM livre;
```

l'exécuter en appuyant sur `Ctrl+Entrée`. Lire en haut la valeur `total` renvoyée. Que signifie cette valeur ?

On passe maintenant à la syntaxe SQL de création de tables :

```
CREATE TABLE usager (
        nom VARCHAR(60),
        prenom VARCHAR(60),
        adresse VARCHAR(300),
        cp VARCHAR(5),
        code_barre CHAR(15) PRIMARY KEY);
```

- Les types les plus utilisés de SQL :
  - numérique : table 1
  - texte : table 2
  - booléen : `TINYINT(1)` (1 est la largeur d'affichage)
  - date, durée, instant : table 3
- Remarques :
  - SQL gère uniquement les numériques décimaux ;
  - la gestion des dates est très complexe ;
  - les chaînes de caractères sont entourées de guillemets simples : `'ABC'`. Pour obtenir l'apostrophe, le doubler : `'l''informatique'`.

Table 1 – Types numériques en SQL .

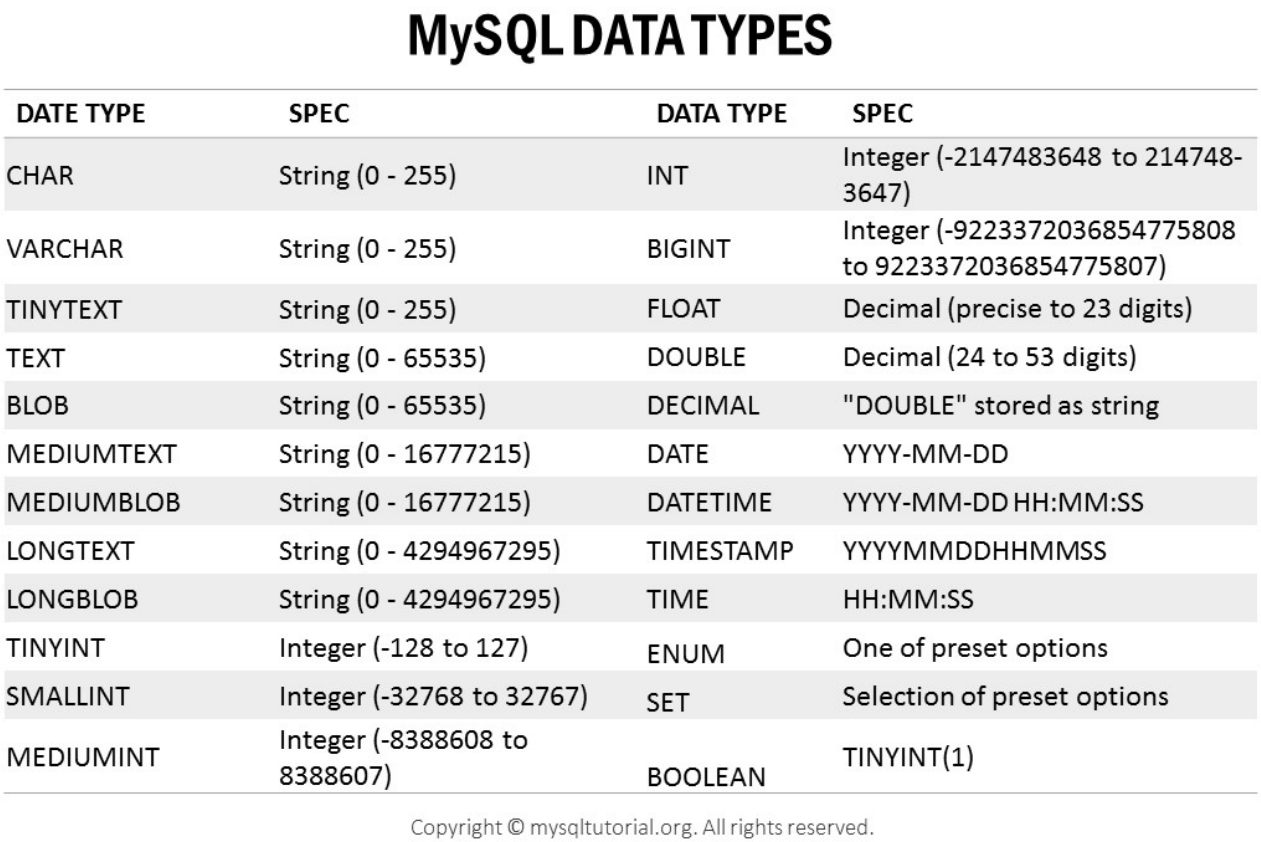
nom du type	exact/approché	description
`SMALLINT`	exact	entier 16 bits signé
`INTEGER`	exact	entier 32 bits signé
`INT`	exact	alias pour `INTEGER`
`BIGINT`	exact	entier 64 bits signé
`DECIMAL(t, f)`	exact	décimal signé de `t` chiffres dont `f` après la virgule
`REAL`	approché	flottant 32 bits
`DOUBLE PRECISION`	approché	flottant 64 bits

Table 2 – Types texte en SQL .

nom du type	description
`CHAR(n)`	Chaîne d'exactement n caractères. Les caractères manquants sont complétés par des espaces.
`VARCHAR(n)`	Chaîne d'au plus n caractères.
`TEXT`	Chaîne de taille quelconque.

Table 3 – Types date en SQL .

nom	description
`DATE`	une date au format `'AAAA-MM-JJ'`
`TIME`	une heure au format `'hh:mm:ss'`
`TIMESTAMP`	un instant (date et heure) au format `'AAAA-MM-JJ hh:mm:ss'`



MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

Figure 2 – Types les plus utilisés en SQL .

- Les contraintes d'entité (clef primaire) : l'attribut `code_barre` de la table `usager` précédemment créée est déclaré comme clef primaire.
- une clef primaire peut être composite (composée de plusieurs colonnes). Exemple :

```
CREATE TABLE point2D_rgb (
        x REAL,
        y REAL,
        r SMALLINT,
        g SMALLINT,
        b SMALLINT,
        PRIMARY KEY (x, y));
```

- Remarques :
  - MySQL n'impose pas qu'une table ait toujours une clef primaire (donc en théorie, il peut y avoir des doublons) mais en pratique on en crée toujours une ;
  - on peut aussi créer une colonne jouant le rôle de clef primaire, souvent appelé `id` (identificateur), de type entier et auto-incrémenté. Un exemple (artificiel) :

```
CREATE TABLE personne (
       id_pers INT AUTO_INCREMENT PRIMARY KEY,
       nom VARCHAR(60),
       prenom VARCHAR(60));
```

- Les contraintes de référence. Avec l'exemple de la BdD mediatheque :

```
CREATE TABLE livre (
       titre VARCHAR(300),
       editeur VARCHAR(90),
       annee INT,
       isbn CHAR(14) PRIMARY KEY);

CREATE TABLE emprunt (
       code_barre CHAR(15),
       isbn CHAR(14) PRIMARY KEY,
       retour DATE,
       FOREIGN KEY (code_barre) REFERENCES usager(code_barre),
       FOREIGN KEY (isbn) REFERENCES livre(isbn));
```

- Remarques :
  - on peut donner le même nom de colonne dans 2 tables différentes (par exemple pour une clef étrangère) ;
  - on peut créer des clefs étrangères composites.

- Remarques (suite) :
  - un bon tutoriel sur SQL : w3schools.com/sql/
  - on peut aussi spécifier qu'un attribut ne peut être indéfini avec la syntaxe `NOT NULL`, ou qu'il doit être unique, avec le mot clef `UNIQUE`[5].
- Les contraintes utilisateurs peuvent être spécifiées avec le mot clef `CHECK` suivi d'une formule booléenne. Exemple pour une BdD de gestion de produits en vente dans un commerce :

```
CREATE TABLE produit (
        id INT PRIMARY KEY,
        nom VARCHAR(100) NOT NULL,
        quantite INT NOT NULL,
        prix DECIMAL(10,2) NOT NULL
        CHECK (quantite >= 0 AND prix >= 0));
```

5. La différence entre une clef primaire et une contrainte `UNIQUE` est que la première ne peut être indéfinie, alors que la seconde peut l'être.

Sommaire

Insertion d'éléments dans une table, suppression d'une table

- Syntaxe de l'insertion, pour l'exemple de la table `auteur` de la BdD `mediatheque` :

```
INSERT INTO auteur VALUES (97, 'Ritchie', 'Dennis'),
                          (98, 'Voltaire');
```

- Remarque : les contraintes d'intégrité sont vérifiées au moment de l'insertion.
- Syntaxe de la suppression d'une table :

```
DROP TABLE emprunt;
```

- Remarque : si une table contient une clé primaire référencée par la clé étrangère d'une autre table, elle ne pourra être supprimée. La suppression des tables doit se faire dans un certain ordre.

Ex. 2 Regrouper les termes synonymes : colonne, entité, domaine, attribut, ligne, schéma, base de données, type, *column*, *row*.

Ex. 3 Reprendre la modélisation obtenue en correction de l'exercice qui consistait à créer une BdD correspondant à un annuaire téléphonique. Donner la commande MySQL permettant de créer la table avec un maximum de contraintes d'intégrité. Tester/valider votre solution sous phpMyAdmin .

Ex. 4 Reprendre la modélisation obtenue en correction de l'exercice sur la création d'une BdD pour un bulletin scolaire. Donner les commandes MySQL permettant de créer les tables avec un maximum de contraintes d'intégrité. Donner les ordre MySQL permettant de supprimer ces tables une fois qu'elle existent. Tester/valider vos réponses sous phpMyAdmin .

Ex. 5 Pour chacune des séquences d'ordre MySQL suivantes, dire quelle instruction provoque une erreur. On suppose que la BdD ne contient aucune table au début de chaque séquence.

1.
```
DROP TABLE acheteur;     /* éviter l'identificateur */
                         /* «client»: mot réservé   */
CREATE TABLE acheteur (id_a INT PRIMARY KEY,
        nom VARCHAR (100), prenom VARCHAR (100),
        points_fidelite INT NOT NULL,
        CHECK (points_fidelite >= 0));
```
2.
```
CREATE TABLE acheteur (id_a INT PRIMARY KEY,
        nom VARCHAR (100), prenom VARCHAR (100),
        points_fidelite INT NOT NULL,
        CHECK (points_fidelite >= 0));

CREATE TABLE commande (id_a INT, id_p INT,
        datec DATE  NOT NULL,
        FOREIGN KEY (id_a) REFERENCES acheteur(id_a),
        FOREIGN KEY (id_p) REFERENCES produit(id_p));

CREATE TABLE produit (id_p INT PRIMARY KEY,
                      nom VARCHAR (100),
                      prix DECIMAL(10,2));
```

3.
```
CREATE TABLE acheteur (id_a INT PRIMARY KEY,
                       nom VARCHAR (100),
                       prenom VARCHAR (100),
                       points_fidelite INT NOT NULL,
                       CHECK (points_fidelite >= 0));

CREATE TABLE produit (id_p INT PRIMARY KEY,
                      nom VARCHAR (100),
                      prix DECIMAL(10,2));

CREATE TABLE commande (id_a INT, nom VARCHAR (100),
        id_p INT,
        datec DATE  NOT NULL,
        FOREIGN KEY (id_a) REFERENCES acheteur(id_a),
        FOREIGN KEY (nom) REFERENCES produit(nom));
```

4.
```
CREATE TABLE acheteur (id_a INT PRIMARY KEY,
                       nom VARCHAR (100),
                       prenom VARCHAR (100),
                       points_fidelite INT NOT NULL,
                       CHECK (points_fidelite >= 0));

CREATE TABLE produit (id_p INT PRIMARY KEY,
                      nom VARCHAR (100),
                      prix DECIMAL(10,2));

CREATE TABLE commande (id_a INT, id_p INT,
        datec DATE  NOT NULL,
        FOREIGN KEY (id_a) REFERENCES acheteur(id_a),
        FOREIGN KEY (id_p) REFERENCES produit(id_p));

INSERT INTO commande VALUES (0, 0, '2020-03-02');
```

Ex. 6 On considère les deux tables suivantes :

```
CREATE TABLE joueur (jid INT PRIMARY KEY,
                     nom VARCHAR (100) NOT NULL);

CREATE TABLE partie (j1 INT, j2 INT,
                 score1 INT NOT NULL,
                 score2 INT NOT NULL,
                 FOREIGN KEY (j1) REFERENCES joueur (jid),
                 FOREIGN KEY (j2) REFERENCES joueur (jid),
                 CHECK (j1 <> j2));
```

Ces tables stockent des résultats de parties entre des joueurs. Lister toutes les contraintes d'intégrité et pour chacune donner des ordres SQL violant ces contraintes.

Ex. 7 Modifier les ordres de création de table de l'exercice précédent pour prendre en compte les modifications suivantes :

- La table `partie` contient en plus une colonne `jour` non `NULL`, indiquant la date à laquelle la partie a eu lieu ;
- les scores ne peuvent pas être négatifs ;
- deux joueurs ne peuvent pas jouer deux fois le même jour.

Sommaire

Consultation des données

- Il s'agit ici :
  1. de réussir à traduire dans le langage des requêtes SQL des questions qu'un utilisateur se pose sur les données (correspondant à des informations effectivement présentes dans la BdD) ;
  2. d'exécuter ces requêtes puis de récupérer les tables qu'elles renvoient, pour qu'elles soient éventuellement traitées ultérieurement.
- Sur l'exemple de la médiathèque, des questions possibles, à un instant donné, sont :
  - quels livres sont empruntés par un usager dont le code-barre (de la carte) est connu ;
  - un livre dont on a l'isbn est-il emprunté ?
  - quels sont les usagers en retard pour rendre leurs livres ?
  - quels sont les livres écrits par tel auteur qui sont empruntables ?
  - quel est le nombre total de livres empruntés ?
- Relire la structure des différentes tables de cette BdD, données page 27 et 28.

```
CREATE TABLE usager (nom VARCHAR(90) NOT NULL,
                     prenom VARCHAR(90) NOT NULL,
                     adresse VARCHAR(300) NOT NULL,
                     cp VARCHAR(5) NOT NULL,
                     ville VARCHAR(60) NOT NULL,
                     email VARCHAR(60) NOT NULL,
                     code_barre CHAR(15) PRIMARY KEY);

CREATE TABLE livre (titre VARCHAR(300) NOT NULL,
                    editeur VARCHAR(90) NOT NULL,
                    annee INT NOT NULL,
                    isbn CHAR(14) PRIMARY KEY);

CREATE TABLE auteur (a_id INT PRIMARY KEY,
                     nom VARCHAR(90) NOT NULL,
                     prenom VARCHAR(90) NOT NULL);
```

```
CREATE TABLE auteur_de (a_id INT, isbn CHAR(14),
        FOREIGN KEY (a_id) REFERENCES auteur(a_id),
        FOREIGN KEY (isbn) REFERENCES livre(isbn),
        PRIMARY KEY (a_id, isbn));

CREATE TABLE emprunt (code_barre CHAR(15),
        isbn CHAR(14) PRIMARY KEY,
        retour DATE NOT NULL,
        FOREIGN KEY (code_barre) REFERENCES usager(code_barre),
        FOREIGN KEY (isbn) REFERENCES livre(isbn));
```

- Pour obtenir la liste des titres des livres publiés après 1990 :

```
SELECT titre FROM livre WHERE annee >= 1900;
```

- La clause `WHERE` doit être suivie d'une expression booléenne. Pour récupérer les titres des livres publiés entre 1970 et 1990 par l'éditeur Dargaud :

```
SELECT titre FROM livre WHERE annee >= 1970 AND
                              annee <= 1990 AND
                              editeur = 'Dargaud';
```

- Possibilité d'utiliser un filtre avec `LIKE` :

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

- Ci-dessus, la chaîne

  `'%Astérix%'`

  qu'on appelle un motif, regroupe toutes les chaînes contenant la chaîne de caractères `'Astérix'` précédée et suivie de zéro ou plusieurs caractères (grâce au caractère '%' [6]).

6. Voir ce lien pour plus de détails.

- Possibilité de récupérer plusieurs informations (colonnes) de la table :

```
SELECT titre, isbn FROM livre WHERE annee >= 1900;
```

- Possibilité de récupérer toutes les colonnes avec * :

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

- Possibilité de renommer les informations récupérées avec **AS** :

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

- Possibilité d'appliquer une fonction d'*agrégation* à l'ensemble des valeurs d'une colonne récupérées qui sont alors résumées en une seule valeur :

  `COUNT` : renvoie le nombre de lignes trouvées
  `MIN` : renvoie la plus petite valeur trouvée
  `MAX` : renvoie la plus grande valeur trouvée
  `AVG` : renvoie la moyenne des valeurs (numériques) trouvées
  `SUM` : renvoie la somme des valeurs (numériques) trouvées

- Exemple : trouver le nombre total de livres dont le titre contient `'Astérix'` et le nommer `total` :

```
SELECT COUNT(titre) AS total FROM livre
                    WHERE titre LIKE '%Astérix%';
```

- La requête suivante équivaut à la précédente :

```
SELECT COUNT(*) AS total FROM livre
                WHERE titre LIKE '%Astérix%';
```

- Autres exemples :

```
SELECT MIN(annee) AS inf FROM livre;
SELECT MAX(annee) AS sup FROM livre;
```

- Possibilité d'ordonner les résultats dans un ordre ascendant ou descendant :

```
SELECT titre FROM livre WHERE annee >= 1990
                        ORDER BY titre ASC;
```

- Possibilité de retirer les doublons d'un attribut avec `DISTINCT` :

```
SELECT DISTINCT annee FROM livre WHERE annee >= 2010
                                 ORDER BY annee ASC;
```

- Si on donne le nom de deux colonnes, cela élimine les lignes où ces couples sont identiques. On peut alors continuer (éventuellement) à ordonner les résultats selon l'ordre induit sur ces couples.

```
SELECT DISTINCT annee, isbn FROM livre WHERE annee >= 2010
                ORDER BY annee, isbn ASC;
```

- Si on veut récupérer un jeu d'informations reliées entre elles mais réparties dans *deux* tables on dit qu'on réalise une *jointure* des tables. On doit pour cela donner le critère (de jointure) qui récupère seulement certaines lignes présentes dans la deuxième table.
- Exemple : on veut connaître toutes les informations sur les livres empruntés, à savoir `titre`, `editeur`, `annee` et `isbn` :

```
SELECT livre.* FROM livre
        JOIN emprunt ON emprunt.isbn = livre.isbn;
```

Remarque : la notation pointée ci-dessus permet de désigner toutes les colonnes de la table `livre`.

- Pour n'avoir que le titre et la date de retour des livres à rendre avant le 01 février 2021 :

```
SELECT livre.titre, emprunt.retour FROM emprunt
       JOIN livre ON emprunt.isbn = livre.isbn
       WHERE emprunt.retour < '2020-02-01';
```

Remarque : la requête précédente est équivalente à la suivante :

```
SELECT livre.titre, emprunt.retour FROM livre
       JOIN emprunt ON emprunt.isbn = livre.isbn
       WHERE emprunt.retour < '2020-02-01';
```

- Possibilité de faire des jointures sur plus de deux tables :

```
SELECT usager.nom, usager.prenom,
       livre.titre, emprunt.retour FROM emprunt
       JOIN livre ON emprunt.isbn =  livre.isbn
       JOIN usager ON usager.code_barre = emprunt.code_barre
       WHERE emprunt.retour < '2020-02-01';
```

- Possibilité de faire des alias pour écrire des requêtes plus concises :

```
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 u.code_barre = e.code_barre
       WHERE e.retour < '2020-02-01';
```

Remarque : les jointures que nous venons de décrire sont dites *internes* (on peut les écrire `INNER JOIN`). Il existe d'autres types de jointure, que nous de présentons pas ici. Elles permettent de récupérer une autre partie des informations concernées par les deux tables, par exemple les données communes aux deux tables *et* celles d'une des deux tables (`LEFT JOIN` ou `RIGHT JOIN`), ou au contraire toutes les informations des deux tables *sauf* celles qui leur sont communes (`FULL JOIN`).

Sommaire

Modification des données – Suppression de lignes

- Pour supprimer toutes les lignes d'une table vérifiant une condition, on utilise `DELETE` :

```
DELETE FROM emprunt WHERE code_barre = '9347012819311582';
```

- Attention : opération irréversible !
- Rappel : on ne peut supprimer des lignes qui contiennent des colonnes référencées ailleurs (clés étrangères).
- Si une suppression, qui peut porter sur plusieurs lignes, provoque une erreur (p.ex. pour un problème de clé), *aucune* suppression n'est réalisée : comportement «tout-ou-rien».
- Exemple :

```
DELETE FROM usager WHERE cp = '75001' OR cp = '75002';
```

- Si aucun usager du 75001 n'a emprunté de livre mais qu'au moins un du 75002 en a emprunté un, aucune suppression n'est faite.

Mise à jour de lignes

- Pour remplacer certaines colonnes des lignes d'une table vérifiant une certaine condition, on utilise `UPDATE` :

```
UPDATE usager SET email = 'spetit@hmail.com'
       WHERE code_barre = '934701281931582';
```

- Il est possible de manipuler ainsi directement des attributs, par exemple ajouter 30 jours à une date :

```
UPDATE emprunt SET retour = retour + INTERVAL 30 DAY
       WHERE retour >= '2020-04-01';
```

- Remarque : étant donné l'irréversibilité des modifications, on doit généralement faire des sauvegardes de secours. Exemple ;

```
CREATE TABLE usager_copie LIKE usager;
INSERT INTO usager_copie SELECT * FROM usager;
```

Sommaire

Requêtes imbriquées

- Ces requêtes permettent d'effectuer des recherches sur des tables temporaires sans avoir à créer celles-ci. Trois exemples :

```
SELECT titre FROM livre WHERE annee =
        (SELECT MIN(annee) FROM livre);

SELECT titre FROM livre WHERE annee =
        (SELECT annee FROM livre WHERE titre = 'Moby Dick');

SELECT titre FROM livre WHERE annee IN
        (SELECT annee FROM livre WHERE titre LIKE '%Astérix%');
```

Sommaire

Exercices

Ex. 8 On utilise la BdD `mediatheque`. Donner le code MySQL de chacune des requêtes ci-dessous. Les mots en `police fixe` donnent une indication sur les attributs et les tables à utiliser dans la requête.

1. Tous les `titres` de `livre`.
2. Tous les `noms` d'`usager`.
3. Tous les `noms` d'`usager` en retirant les doublons.
4. Les `titres` des livres publiés avant 1980.
5. Les `titres` des livres dont le titre contient la lettre «A».
6. Les `isbn` des livres à rendre pour le 01/01/2020.
7. Les `noms` d'`auteurs` triés par ordre alphabétique.
8. Les `noms` d'`usagers` vivant dans le 12^{e} ou 13^{e} arrondissement de Paris (codes postaux 75012 et 75013).

9. Les noms et adresses des usagers n'habitant pas dans une rue (la chaîne «Rue» ne doit pas apparaître dans l'adresse).
10. Les annees et titres des livres publiés lors d'une année bissextile. On rappelle que ce sont les années divisibles par 4, mais pas celles divisibles par 100 sauf si elles sont divisibles par 400.

Ex. 9 Soit la BdD mediatheque déjà utilisée. Donner le code SQL de chacune des requêtes ci-dessous. Les mots en police fixe donnent une indication sur les attributs et les tables à utiliser dans la requête.

1. Le titre des livres empruntés.
2. Le titre des livres empruntés à rendre avant le 31/03/2020.
3. Le nom et prenom de l'auteur du livre '1984'.
4. Le nom et le prenom des usagers ayant emprunté des livres, sans doublons (*i.e.* si un usager a emprunté plusieurs livres il ne doit apparaître qu'une fois dans la liste).
5. Même requête que précédemment, avec les noms et prénoms triés par ordre alphabétique.

6. Les titres des livres publiés strictement avant 'Dune'.
7. Les noms et prenoms des auteurs des livres trouvés à la question précédente.
8. Comme à la question précédente, en retirant les doublons.
9. Le nombre de résultats trouvés à la question précédente.

Ex. 10 Soir la BdD mediatheque. Formuler simplement en français les requêtes SQL suivantes :

1. `SELECT * FROM livre WHERE titre LIKE '%Robot%';`
2. `SELECT nom, prenom FROM usager WHERE ville = 'Guingamp';`
3.
```
SELECT u.nom, u.prenom
FROM usager AS u
JOIN emprunt AS e ON u.code_barre = e.code_barre
WHERE retour < '2020-04-02';
```

5.
```
SELECT l.titre
FROM livre AS l
WHERE l.isbn IN (SELECT isbn FROM livre
                                    WHERE annee > 1990);
```

6. Réécrire la requête précédente de façon à n'utiliser qu'une seule clause SELECT.

Ex. 11 Soir la BdD mediatheque. Calculer tous les couples d'auteurs distincts ayant collaboré sur un ouvrage et les renvoyer sous la forme (id_1, n_1, p_1, id_2, n_2, p_2, t), où les id_i sont les identificateurs de chaque auteur, n_i leurs noms, p_i leurs prénoms et t le titre sur lequel ils ont collaboré. Si, par exemple, trois auteurs ont collaboré sur un même livre, on souhaite avoir tous les couples d'auteurs sur trois lignes différentes. Rechercher une contrainte permettant de ne pas afficher deux fois la même paire d'auteurs (le couple (id_1, n_1, p_1, id_2, n_2, p_2, t) et le couple (id_2, n_2, p_2, id_1, n_1, p_1, t)).

Ex. 12 On considère les trois tables de la page 45, créées par les trois requêtes ci-dessous.

```
CREATE TABLE x (a INT PRIMARY KEY, b INT, CHECK (b >= 0));
CREATE TABLE y (c INT PRIMARY KEY, d INT, CHECK (d <= 30));
CREATE TABLE z (a INT, c INT, e INT,
                FOREIGN KEY (a) REFERENCES x(a),
                FOREIGN KEY (c) REFERENCES y(c),
                UNIQUE (a, c));
```

Pour chacune des requêtes MySQL suivantes, calculer son résultat à la main.

1. `SELECT * FROM x WHERE b > 3;`
2. `SELECT DISTINCT e FROM z WHERE e > 10 AND e < 50;`
3. `SELECT * FROM y WHERE c % 2 = 0 ORDER BY d ASC;`
4.
```
SELECT x.a, x.b FROM x
                JOIN z ON z.a = x.a
                WHERE z.e < 9;
```

5.
```
SELECT DISTINCT x.b, y.d FROM x
                JOIN z ON z.a = x.a
                JOIN y ON y.c = z.c;
```

Table 4 – La table x.

a	b
1	1
2	2
3	2
4	2
5	1
6	9
7	1

Table 5 – La table y.

c	d
9	9
10	10
11	9
12	20
13	30
14	9
15	1
16	10
17	10

Table 6 – La table z.

a	c	e
1	11	30
2	14	9
5	15	1
7	17	3
1	10	50
2	9	8
2	15	15
3	17	19
4	16	12
5	10	20
2	11	30
7	14	9
7	9	12

Ex. 13 On considère les trois tables de l'exercice précédent. Pour chacune des modifications ci-dessous, indiquer si elle réussit ou si elle échoue. Si elle réussit, indiquer comment la table est modifiée. Si elle échoue, expliquer pourquoi. Les questions sont indépendantes, c'est à dire chacune repart des tables de l'exercice précédent entre chaque question.

1. `UPDATE x SET b = b + a;`
2. `UPDATE x SET b = b - 2;`
3. `INSERT INTO z VALUES (1, 17, 1);`
4. `INSERT INTO z VALUES (1, 18, 1);`
5. `INSERT INTO z VALUES (1, 10, 1);`
6. `DELETE FROM y WHERE c >= 12 AND c <= 13;`
7. `DELETE FROM y WHERE c >= 12 AND c <= 14;`
8. `INSERT INTO y VALUES (40, 20);`
9. `INSERT INTO y VALUES (20, 40);`
10. `DELETE FROM z WHERE a % 2 = 0 OR c % 2 = 0 OR e % 2 = 0;`