# 1 Langage SQL

Dans les années 1980 est apparu le **langage SQL** (Structured Query Language) spécialement conçu pour faire des requêtes (*sélectionner, filtrer, mettre à jour...*) sur les SGBD.
SQL est un langage qui repose sur le **paradigme déclaratif** : on précise ce qui nous intéresse mais pas la manière dont le système doit procéder pour l'obtenir, ce n'est donc pas un langage de programmation avec de l'algorithmique.

# 2 Base de données d'exemple

- Langue (IdLangue : int, Langue : text)
- Auteur (IdAuteur : int, NomAuteur : text, PrenomAuteur : text, #IdLangue : int, AnneeNaissance : int)
- Theme (IdTheme : int, Intitule : text)
- Livre (IdLivre : int, Titre : text, #IdAuteur : int, AnneePubli : int)
- LivreTheme (#IdLivre : int, #IdTheme : int)

# 3 Requêtes d'interrogation : clause SELECT

## 3.1 Lire le contenu d'une relation

Pour lire des informations d'une base de données on écrit une **requête d'interrogation** avec la commande **SELECT**. Ces requêtes peuvent être devenir assez sophistiquées comme on va le verra plus loin. Pour le moment, nous nous contenterons de la forme la plus simple :

```
SELECT * FROM Auteur;
```

On voit ainsi apparaître tout le contenu de la relation `Auteur`.
Le caractère `*` est un joker qui indique d'afficher toutes les colonnes de la table.

**Attention :** Les requêtes SQL **se terminent toujours par un point-virgule**.
Elles peuvent s'écrire sur plusieurs lignes.
Elles sont non sensibles à la casse, mais c'est une bonne pratique que d'écrire toutes les commandes SQL en majuscules.

## 3.2 Projection et Sélection : clause WHERE

Une projection consiste à extraire certaines colonnes d'une table : en SQL, il suffit de préciser le nom des colonnes qu'on souhaite à la place du joker `*`.

Une sélection consiste à ne conserver que certaines lignes de la table : en SQL, la sélection se fait avec la clause **WHERE** :

Supposons que l'on veuille lister seulement noms et prénoms des auteurs nés avant 1900 :

```
SELECT NomAuteur, PrenomAuteur FROM Auteur WHERE AnneeNaissance < 1900;
```

Pour obtenir les auteurs prénommés Jules :

```
SELECT NomAuteur, PrenomAuteur FROM Auteur WHERE PrenomAuteur = "Jules";
```

Bien noter que les textes sont délimités par des guillemets !

*Attention* : essayez de remplacer "Jules" par "jules" et vous constaterez que la recherche est sensible aux majuscules.

## 3.3 Opérateurs logiques

Il est possible de croiser plusieurs critères de sélection à l'aide d'opérateurs booléens : **AND**, **OR** ou **NOT**.

Voici la liste des auteurs de langue française nés en 1900 ou plus tard :

```
SELECT NomAuteur, PrenomAuteur FROM Auteur
WHERE IdLangue = 2 AND AnneeNaissance >= 1900;
```

## 3.4 Fonctions d'agrégation

Il est possible d'appliquer certaines opérations qu'on appelle fonctions d'agrégation sur les enregistrements sélectionnés. On peut par exemple compter le nombres d'enregistrements sélectionnés (COUNT), ou calculer la moyenne des valeurs (AVG), ou encore rechercher la valeur maximum (MAX).

Exemple : Combien y a t-il d'auteurs nés entre 1900 et 1915 ?

```
SELECT COUNT(*) FROM Auteur WHERE AnneeNaissance > 1900 AND AnneeNaissance < 1915;
```

Exemple : Quelle est le nom de l'auteur et la date de naissance du plus ancien de la base ?

```
SELECT nomAuteur, MIN(AnneeNaissance) FROM Auteur;
```

## 3.5 Trier les réponses : ORDER BY

Il est possible de trier les sélections effectuées :

Listing des auteurs par ordre croissant d'année de naissance :

```
SELECT * FROM Auteur ORDER BY AnneeNaissance;
```

et par ordre décroissant, on ajoute **DESC** à la fin de la requête :

```
SELECT * FROM Auteur ORDER BY AnneeNaissance DESC;
```

### 3.6 Éviter les occurrences multiples : DISTINCT

Interrogeons la table Livres sur les années de publication, rangées par ordre croissant :

```
SELECT AnneePubli FROM Livre ORDER BY AnneePubli;
```

On constate la présence de quelques doublons. Pour éviter les redondances dans les résultats, on peut rajouter le mot-clef **DISTINCT** juste après le SELECT :

```
SELECT DISTINCT AnneePubli FROM Livre ORDER BY AnneePubli;
```

## 4 Requêtes portant sur plusieurs tables

Jusqu'à présent, les requêtes ne portaient que sur une seule table. Nous allons voir comment effectuer des requêtes pour croiser des données en provenance de plusieurs tables.

### 4.1 Jointure : JOIN ON

La **jointure** consiste à **croiser les données de plusieurs tables** pour les présenter sous forme d'un seul tableau. On va utiliser ce mécanisme pour afficher clairement la langue de l'auteur plutôt qu'un numéro qui n'est pas forcément très parlant. Nous utiliserons pour cela la clause **JOIN** ... **ON**.
Une jointure repose en général sur un attribut qui est une clef étrangère d'une relation correspondant à la primaire d'une autre relation.

```
SELECT NomAuteur, PrenomAuteur, Langue
FROM Auteur JOIN Langue ON Auteur.IdLangue = Langue.IdLangue;
```

*Remarque* : on retrouve la syntaxe de la notation de programmation objet avec le point qui sépare le nom d'une relation et le nom d'un attribut de cette relation.

### 4.2 Le cas des relations de n à n

Parfois il arrive que les données à collecter se trouvent dans plus que deux relations : c'est le cas des Thèmes pour les livres qui nécessitent l'analyse de 3 relations : Livre et Thème bien sûr, mais aussi la relation LivreTheme.

Observer et étudier la requête ci-dessous :
Le principe est d'**enchaîner deux jointures** JOIN ... ON en utilisant la table de Relation au milieu. En effet, la requête se lit de la gauche vers la droite et on ne peut faire de jointure que si on a une clef étrangère en commun, ce qui n'est par exemple pas le cas entre Livre et Thème.

```
SELECT Titre, Intitule FROM Livre
JOIN LivreTheme ON Livre.IdLivre = LivreTheme.IdLivre
JOIN Theme ON Theme.IdTheme = LivreTheme.IdTheme;
```

# 5 Requêtes d'insertion : clause INSERT INTO

Insérer des enregistrements se fait avec la syntaxe de format :
`INSERT INTO Table VALUES (données);`

Exemple :

```
INSERT INTO Langue VALUES (3, "Espagnol");
```

# 6 Mise à jour, effacement : clauses UPDATE et DELETE

Les requêtes de type **UPDATE** ou **DELETE** fonctionnent sur le même modèle que les requêtes d'interrogation SELECT.

**Attention !** On a vite fait d'effacer toutes ses données si on ne configure pas bien sa requête. Une bonne habitude à prendre est de **tester d'abord ses critères de sélection** à l'aide d'un SELECT **avant d'effectuer une quelconque modification d'enregistrement**, ou pire d'effacement.

## 6.1 Mise à jour : UPDATE

La syntaxe SQL des requêtes de type UPDATE est la suivante :

```
UPDATE Table
SET attribut1 = valeur1, attribut2 = valeur2, ...
WHERE critères;
```

Exemple : on désire modifier des informations concernant Jules Verne :

```
UPDATE Auteur
SET NomAuteur = "Ze Djoule", PrenomAuteur = "Juju"
WHERE IdAuteur = 10;
```

Remarque : on aura d'abord effectuer la requête suivante

```
SELECT NomAuteur FROM Auteur WHERE IdAuteur = 10;
```

en s'assurant qu'elle renvoie bien les infos concernant Jules Vernes, et pas un autre auteur...

## 6.2 Effacement : DELETE

La syntaxe SQL des requêtes de type DELETE est la suivante :

```
DELETE FROM Table WHERE critères;
```

Exemple : on supprime l'enregistrement "idiot" qu'on a créé :

```
DELETE FROM Auteur WHERE IdAuteur = 10;
```