Chapitre 1

Base de données

I. Introduction

Chacun d'entre nous utilise quotidiennement, mais sans souvent en avoir conscience, des bases de données. Ces dernières regroupent des informations relatives à un domaine précis, souvent ordonnées sous une forme très structurée. Les exemples sont innombrables : gestion des stocks (magasins en ligne, bibliothèques, . . .), gestion des réservations (trains, avions, spectacles . . .), gestion du personnel d'une entreprise, création de listes diverses (vos playlists musicales préférées par exemple), etc. Prenons l'exemple du site de réservation de la SNCF. Les clients peuvent réaliser un certain nombre de tâches, parmi lesquelles :

- la consultation des trains répondants à certains critères (date, trajet, places disponibles, etc) ;
- la réservation d'une place dans le train choisi ;
- l'annulation d'une réservation.

Le personnel de la SNCF, outre les opérations précédentes, peut en outre :

- modifier la composition ou les horaires des trains ;
- ajouter des trains supplémentaires ;
- supprimer des trains.

À l'exception de la consultation, les autres opérations sont complexes car elles doivent gérer le principe de simultanéité qui régit les bases de données : plusieurs clients peuvent chercher à réserver en même temps des places dans le même train, et il ne s'agit pas d'attribuer la même place à plusieurs personnes différentes. Il en va de même de la suppression d'un train, qui doit tenir compte des personnes ayant déjà réservé dans celui-ci. C'est pourquoi nous nous limiterons, dans ce cours d'introduction aux bases de données, à la simple consultation des données.

II. Structure d'une base de données

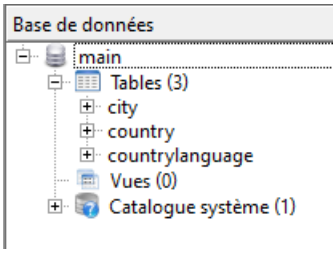
L'accès à une base de données se fait usuellement par l'intermédiaire d'une application qui traduit les demandes de l'utilisateur (en général via une interface graphique) dans un langage dédié à la communication avec une base de données. Ce langage, le SQL (Structured Query Language) est devenu un standard disponible sur presque tous les systèmes de gestion des bases de données (SGBD).

Le SQL comporte des instructions relatives :

- à l'interrogation de bases de données ;
- à la création et la modification des données ;

- au contrôle d'accès des données.

Comme indiqué dans le préambule, nous nous intéresserons uniquement aux instructions de la première catégorie.
Nous allons utiliser dans toute la suite du cours la base de données `world.db`. Cette base de donnée contient un certain nombre d'informations géographiques et politiques mondiales (en 2001).
Commençons par observer son contenu :



Une base de données est un ensemble de **tables** (ou de **relations**, les deux termes étant synonymes dans ce contexte) que l'on peut représenter sous forme de tableaux bi-dimensionnels. Dans l'exemple qui nous intéresse, notre base de données est composée de trois tables :

1. `city`, qui contient des informations relatives à certaines villes du monde ;
2. `country`, qui contient des informations relatives aux pays ;
3. `countrylanguage`, qui contient des informations relatives aux langues parlées dans le monde.

Examinons le contenu de la première, ou plutôt un extrait de son contenu :

```
SELECT * FROM city LIMIT 10;
```

	ID	Name	CountryCode	District	Population
1	1	Kabul	AFG	Kabol	1780000
2	2	Qandahar	AFG	Qandahar	237500
3	3	Herat	AFG	Herat	186800
4	4	Mazar-e-Sharif	AFG	Balkh	127800
5	5	Amsterdam	NLD	Noord-Holland	731200
6	6	Rotterdam	NLD	Zuid-Holland	593321
7	7	Haag	NLD	Zuid-Holland	440900
8	8	Utrecht	NLD	Utrecht	234323
9	9	Eindhoven	NLD	Noord-Brabant	201843
10	10	Tilburg	NLD	Noord-Brabant	193238

Nous pouvons constater qu'une table est un tableau bi-dimensionnel : les en-têtes des différentes colonnes sont appelés des **attributs**, les lignes des **enregistrements**. Ainsi, la table `city` possède 5 attributs : `ID`, `Name` (le nom d'une ville), `CountryCode` (le code du pays dans lequel se trouve cette ville), `District` (sa région d'appartenance) et `Population` (son nombre d'habitants).
Précisons maintenant les caractéristiques de chacun de ces attributs :

```
PRAGMA table_info(city)
```

	cid	name	type	notnull	dflt_value	pk
1	0	ID	INTEGER	1	*NULL*	1
2	1	Name	TEXT	1	''	0
3	2	CountryCode	TEXT	1	''	0
4	3	District	TEXT	1	''	0
5	4	Population	INTEGER	1	0	0

Nous constatons qu'un attribut est un objet typé (dans ce contexte, le type d'un attribut est son domaine). `ID` et `Population` sont des entiers, `Name`, `CountryCode` et `District` des chaîne de caractères. Ces caractéristiques sont fixées lors de la création d'une table.
La colonne suivante indique que tous les attributs doivent posséder une valeur.
Enfin, la troisième colonne apporte une information importante : chaque enregistrement d'une table doit pouvoir être identifié de manière unique ; c'est le rôle de la **clef primaire**, constituée d'un ou plusieurs attributs. Ici, la clef primaire est constituée de l'unique attribut `ID`. Il ne peut donc y avoir deux enregistrements ayant le même ID.

Regardons maintenant le contenu de la table country :

	cid	name	type	notnull	dflt_value	pk
1	0	Code	TEXT	1	*NULL*	1
2	1	Name	TEXT	1	*NULL*	0
3	2	Continent	TEXT	1	*NULL*	0
4	3	Region	TEXT	1	*NULL*	0
5	4	SurfaceArea	REAL	1	*NULL*	0
6	5	IndepYear	INTEGER	0	*NULL*	0
7	6	Population	INTEGER	1	*NULL*	0
8	7	LifeExpectancy	REAL	0	*NULL*	0
9	8	GNP	REAL	0	*NULL*	0
10	9	GNPOld	REAL	0	*NULL*	0
11	10	LocalName	TEXT	1	*NULL*	0
12	11	GovernmentForm	TEXT	1	*NULL*	0
13	12	HeadOfState	TEXT	0	*NULL*	0
14	13	Capital	INTEGER	0	*NULL*	0
15	14	Code2	TEXT	1	*NULL*	0

Ici, la clef primaire est l'attribut `Code`, qui est une chaîne caractères. On constate en outre que certaines données ne sont pas obligatoires : les attributs `IndepYear`, `LifeExpectancy`, `GNP`, `HeadOfState` et `Capital` peuvent être absents, auquel cas la valeur qui leur est attribuée est NULL.
Voyons un extrait de cette table, par exemple l'enregistrement qui concerne la France :

```
SELECT * FROM country WHERE Name = 'France';
```

	Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld	LocalName	GovernmentForm	HeadOfState	Capital	Code2
1	FRA	France	Europe	Western Europe	551500.0	843	59225700	78.8	1424285.0	1392448.0	France	Republic	Jacques Chirac	2974	FR

Deux attributs sont remarquables : l'attribut `Code` (clef primaire de la table) correspond à l'attribut `CountryCode` de la table `city`. Cette remarque est importante car c'est elle qui nous permettra de chercher des informations croisées dans ces deux tables. L'attribut `Capital` est plus intriguant : pourquoi la capitale de la France est-elle désignée par un entier ?
Il se trouve que cet attribut correspond à l'attribut `ID` de la table `city`, comme on peut s'en convaincre en consultant l'enregistrement dont l'attribut `ID` vaut 2974.

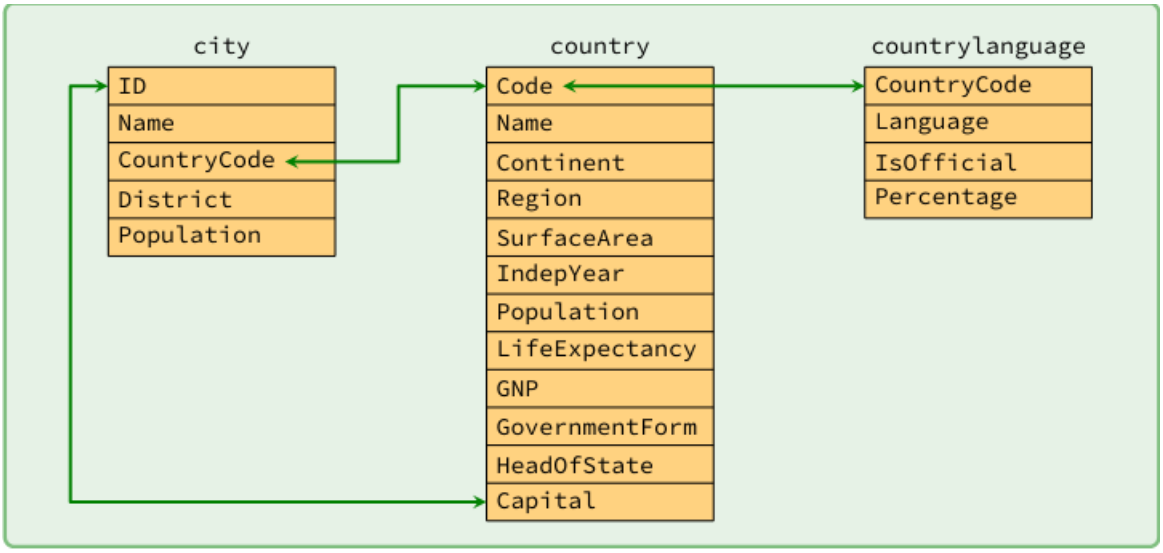
```
SELECT * FROM City WHERE Id = '2974'
```

Regardons enfin le contenu de la troisième et dernière table :

	cid	name	type	notnull	dflt_value	pk
1	0	CountryCode	TEXT	1	*NULL*	1
2	1	Language	TEXT	1	*NULL*	2
3	2	IsOfficial	INTEGER	1	0	0
4	3	Percentage	REAL	1	*NULL*	0

Cette table décrit les langues utilisées dans les différents pays en précisant le pourcentage de locuteurs et si cette langue est officielle ou non. Une même langue peut être parlée dans différents pays, et dans un même pays plusieurs langues peuvent être pratiquées. Ainsi, aucun des attributs ne peut prétendre être une clef primaire, c'est pourquoi nous avons ici un couple d'attributs en guise de clef primaire : le couple (CountryCode, Language).

Pour pouvoir efficacement interroger une base de données, il importe de connaître avec précision le contenu de chacune des tables, et les attributs qui vont nous permettre de les relier entre elles. Cet ensemble d'informations est appelé le schéma de la base de donnée.



III. Requêtes SQL

Nous allons maintenant nous intéresser à la syntaxe des requêtes qui vont nous permettre d'interroger la base de données. Cette syntaxe est proche de la langue anglaise ; en général, la lecture d'une requête permet d'en comprendre le sens. Elle n'en reste pas moins assez rigide dans sa structure, et les mots clefs que nous allons utiliser doivent être rangés dans un ordre bien précis :

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ... LIMIT ... OFFSET ...
```

Seuls les deux premiers (`SELECT` et `FROM`) sont indispensables, les autres sont optionnels, mais s'ils sont présents ils doivent être placés dans cet ordre.
Notons enfin que ces noms de commandes sont insensibles à la casse. Autrement dit, vous pouvez les écrire indifféremment en majuscules ou en minuscule.

1. Sélection des attributs et des enregistrements

- On sélectionne un ou plusieurs attributs d'une table par la syntaxe :

```
SELECT a1, a2, ... an FROM table
SELECT DISTINCT a1, a2, ... an FROM table
```

Le mot-clef `DISTINCT` permet d'éviter l'apparition de doublons parmi les résultats obtenus.

> **Remarque**
> Pour obtenir tous les attributs de `table` :
> ```
> SELECT * FROM table
> ```

> **Exemple 1.1**
> Pour déterminer la liste des différentes langues présentes dans la base :
> ```
> SELECT DISTINCT Language FROM CountryLanguage
> ```

- On sélectionne les enregistrements d'une table qui satisfont une expression logique par la syntaxe :

```
SELECT * FROM table WHERE expression_logique
SELECT a1, ..., an FROM table WHERE expression_logique
```

> **Exemple 1.2**
> Nous pouvons visualiser les villes françaises et leurs populations respectives en écrivant :
> ```
> SELECT Name, Population FROM City WHERE CountryCode = 'FRA'
> ```

- Le renommage permet la modification du nom d'un attribut d'une relation. Renommer l'attribut a en l'attribut b dans la relation R peut être effectué à l'aide du mot clé `AS`.

```
SELECT a AS b FROM table
```

- **Filtrage des résultats** :
  À la toute fin d'une requête il est possible de trier les résultats et de n'en renvoyer qu'une partie. Ces opérations se notent :
  - `ORDER BY a ASC / DESC` pour trier suivant l'attribut `a` par ordre croissant/décroissant ;
  - `LIMIT n` pour limiter la sortie à `n` enregistrements ;
  - `OFFSET n` pour débuter à partir du n-ième enregistrement.

> **Exemple 1.3**
> Par exemple, les cinq villes mondiales les plus peuplées sont :
> ```
> SELECT Name FROM City ORDER BY Population DESC LIMIT 5
> ```
> Et les 5 suivantes sont
> ```
> SELECT Name FROM City ORDER BY Population DESC LIMIT 5 OFFSET 5
> ```

Exercice 1.4

1. Écrire une requête SQL donnant le nom des dix pays asiatiques les plus grands.
2. Écrire une requête SQL donnant le nom et la date d'indépendance des dix pays les plus anciens.
3. Écrire une requête SQL donnant la liste des langues non officielles pratiquées en France, par ordre décroissant d'importance.
4. Écrire une requête SQL donnant le nom des cinq pays européens les moins densément peuplés.

2. Jointures et sous-requêtes

La jointure est une opération qui porte sur deux relations R_1 et R_2 et retourne une relation qui comporte les enregistrements combinés de R_1 et de R_2 qui satisfont une contrainte logique E.
En SQL on réalise une jointure par la requête :

```
SELECT * FROM table1 JOIN table2 ON expression_logique
```

Remarque

Deux tables reliées par une jointure peuvent posséder des attributs de même nom mais n'ayant rien à voir. C'est le cas par exemple de l'attribut `Name` présent dans les deux tables `city` et `country`. Pour les distinguer lors d'une jointure il est nécessaire de faire précéder le nom de l'attribut par le nom de la table, séparé par un point.
Par exemple, lors d'une jointure entre les deux tables de notre base de données exemple, le nom des villes est désigné par `city.name` et celui des pays par `country.Name`.
Notons que le mot-clef `AS` permet de renommer les tables ainsi que les attributs afin d'éviter d'écrire des noms à rallonge.

Exemple 1.5

Les deux tables `City` et `Country` possèdent deux jointures naturelles :

- on peut identifier les attributs `city.countryCode` et `country.Code`. Dans ce cas, la table résultant de la jointure possédera autant d'entrées que la table `City` et permettra de relier à chaque ville de la base les informations du pays auquel elle appartient ;
- on peut identifier les attributs `city.ID` et `country.Capital`. Dans ce cas, la table résultant de la jointure possédera autant d'entrées que la table `country` et à chaque pays sera attaché les informations relatives à sa capitale. En revanche, les villes qui ne sont pas des capitales ne seront pas présentes dans cette jointure.

Par exemple, si on souhaite connaître la capitale de l'Ouzbekistan et son nombre d'habitants on écrira :

```
SELECT City.Name, City.Population FROM City JOIN Country ON
City.ID = Country.Capital WHERE Country.Name = 'Uzbekistan'
SELECT City.Name, City.Population FROM Country JOIN City ON
City.ID = Country.Capital WHERE Country.Name = 'Uzbekistan'
```

Exercice 1.6

Écrire une requête SQL

1. donnant la liste des pays ayant adopté le Français parmi leurs langues officielles.
2. donnant les cinq villes les plus peuplées d'Europe.
3. donnant les cinq villes les plus peuplées d'Europe parmi celles qui ne sont pas des capitales.
4. donnant les capitales des pays dans lesquels l'allemand est langue officielle

Sous-requêtes

Revenons au problème consistant à déterminer la capitale de L'Ouzbekistan. Nous l'avons résolu à l'aide d'une jointure, mais il aurait aussi été possible de procéder à deux requêtes successives : une première requête pour déterminer l'identifiant de la capitale de l'Ouzbekistan, une seconde pour connaître le nom de cette ville.
Il est possible d'imbriquer la première au sein de la seconde pour n'en faire qu'une seule ; on parle alors de sous-requête. Celle-ci doit impérativement être délimitée par des parenthèses, et peut être située :

- à la place d'une table après le `FROM` ;
- au sein d'une expression logique après le `WHERE`

Par exemple, la détermination de la capitale de l'Ouzbekistan peut être obtenue par la requête

```
SELECT city.Name FROM city WHERE ID = (SELECT Capital FROM country WHERE
name = 'Uzbekistan');
```

Exercice 1.7

1. Donner la liste des pays dont l'espérance de vie est supérieure ou égale à celle de la France.
2. Donner le pourcentage de pays dont le PNB est supérieur ou égal à la moyenne

Notons enfin que lorsqu'une sous-requête prend la place d'une table, il est impératif de renommer les attributs de la sous-requête pour qu'ils soient utilisés dans la requête principale

3. Fonctions d'agrégation

SQL possède un certain nombre de fonctions statistiques qui par défaut s'appliquent à l'ensemble des enregistrements sélectionnés par la clause du `WHERE` :

- `COUNT()` : nombre d'enregistrements
- `MAX()` : valeur maximale d'un attribut
- `MIN()` : valeur minimale d'un attribut
- `SUM()` : somme des enregistrements
- `AVG()` : moyenne des enregistrements

Exemple 1.8

Par exemple, pour obtenir la population de l'Europe on écrit :

```
SELECT SUM(Population) FROM Country WHERE Continent = 'Europe'
```

Exercice 1.9

Déterminer la ville la moins peuplée de cette base de données

Mais il est aussi possible de regrouper les enregistrements d'une table par agrégation à l'aide du mot-clef `GROUP BY`. Ce regroupement permet d'appliquer la fonction statistique à chacun des

groupes : le résultat de la requête est l'ensemble des valeurs prises par la fonction statistique sur chacun des regroupements.

```
SELECT f(a1) FROM table WHERE expression_logique GROUP BY a2
```

Cette requête sélectionne les enregistrements de la table qui vérifient l'expression logique, les regroupe selon la valeur de l'attribut a2, puis applique la fonction f sur l'attribut a1 à chacun des groupes obtenus.
Notons enfin que le mot-clef `HAVING` permet d'imposer des conditions sur les groupes à qui on applique la fonction d'agrégation. La syntaxe générale de la requête prend alors la forme suivante :

```
SELECT f(a1) FROM table WHERE e1 GROUP BY a2 HAVING e2
```

Cette requête sélectionne les enregistrements de la table qui vérifient l'expression logique e1, les regroupe selon la valeur de l'attribut a2, puis applique la fonction f sur l'attribut a1 à chacun des groupes vérifiant l'expression logique e2.

Exemple 1.10

Pour obtenir les populations de chacun des continents on écrira :

```
SELECT continent, SUM(Population) FROM Country GROUP BY Continent
```

Pour ordonner cette liste suivant la population, il faut renommer l'attribut calculant la population totale de chaque continent :

```
SELECT Continent, SUM(Population) AS Pop FROM country GROUP BY Continent
ORDER BY Pop DESC;
```

Enfin, si on veut se restreindre aux continents qui comportent plus de 40 pays, on écrira :

```
SELECT Continent, SUM(Population), COUNT() FROM country GROUP BY Continent HAVING COUNT(*)
```

Exercice 1.11

1. Donner la liste des districts des États-Unis dont la population totale est supérieure à 3 000 000.
2. Déterminer les cinq langues les plus parlées au monde.
3. Déterminer, pour chaque continent, le pays le plus peuplé.

4. Opérations ensemblistes

On peut procéder à des opérations ensemblistes entre deux tables (en général le résultat de requêtes) à condition que celles-ci aient une même structure. Ces opérations sont :

- `UNION` pour calculer la réunion $A \cup B$ de deux requêtes
- `INTERSECT` pour calculer l'intersection $A \cap B$ de deux requêtes
- `EXCEPT` pour calculer la différence $A - B = A \setminus (A \cap B)$ de deux requêtes

Ces opérations présentent surtout un intérêt lorsque les deux requêtes sont issues de deux tables différentes.
Plusieurs SGBD n'implémentent d'ailleurs que l'union, les deux autres opérations ensemblistes pouvant être aisément remplacées par des requêtes équivalentes :

- `SELECT a FROM table1 INTERSECT SELECT b FROM table2` est équivalent à :

```
SELECT a FROM table1 WHERE a IN (SELECT b FROM table2)
```

- `SELECT a FROM table1 EXCEPTS SELECT b FROM table2` est équivalent à :

```
SELECT a FROM table1 WHERE a NOT IN (SELECT b FROM table2)
```

Il est enfin possible de réaliser le produit cartésien de deux tables (ou plus) en suivant la syntaxe :

```
SELECT A1 , A2 FROM table1, table2
```

IV. Exercice

On souhaite utiliser une base de données pour stocker les résultats obtenus par une communauté de joueurs en ligne. On suppose qu'on dispose d'une base de données comportant deux tables : `Joueurs(id_j, nom, pays)` et `Parties(id_p, date, duree, score, id_joueur)` où :

- `id_j` de type entier, est la clef primaire de la table `Joueurs`;
- `nom` est une chaîne de caractères donnant le nom du joueur;
- `pays` est une chaîne de caractères donnant le pays du joueur;

- `id_p` de type entier, est la clef primaire de la table `Parties`;
- `date` est la date (AAAAMMJJ) de la partie;
- `duree` de type entier, est la durée en secondes de la partie;
- `score` de type entier, est le nombre de points marqués au cours de la partie;
- `id_joueur` est un entier qui identifie le joueur de la partie.

1. Rédiger une requête SQL qui renvoie la date, la durée et le score de toutes les parties jouées par Alice, listées par ordre chronologique.
2. Alice vient de réaliser un score de n points. Rédiger une requête SQL qui renvoie la position qu'aura le score n dans le classement des parties par ordre de score (on suppose que la partie que vient de jouer Alice n'est pas encore insérée dans la base de données). En cas d'ex aequo pour le score n le rang sera le même pour tous les joueurs ayant ce score.
3. Rédiger une requête SQL qui renvoie le record de France pour ce jeu.
4. Rédiger une requête SQL qui renvoie le rang d'Alice, c'est-à-dire sa position dans le classement des joueurs par ordre de leur meilleur score.