

Chap.1 : Bases de données

Table des matières

1	Description d'une base de données	2
1.1	Introduction	2
1.2	Structure d'une base de données	2
2	Requêtes SQL	6
2.1	Sélection des attributs et des enregistrements	7
2.2	Jointures	10
2.3	Sous-requêtes	12
2.4	Fonctions d'agrégation	13
2.5	Opérateurs ensemblistes	15
3	Application	16

On pourra se connecter à l'activité numéro 5d41-758151 sur la plateforme Capytale pour l'illustration des exemples.

La base de donnée utilisée tout au long de ce chapitre y est enregistrée et il est possible de formuler les requêtes sur cette dernière.

1 Description d'une base de données

1.1 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.

1.2 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 d'interrogation.

Dans la suite de ce cours j'utiliserai MySQL comme système de gestion pour interroger une base de données intitulée *world.sql*. Cette base de donnée contient un certain nombre d'informations géographiques et politiques mondiales (en 2001). Commençons par observer son contenu :

```

mysql> show tables;
+-----+
| Tables_in_world |
+-----+
| city              |
| country           |
| countrylanguage  |
+-----+
  
```

Une **base de données** est un ensemble de **relations** (ou de tables, 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 :

- *city*, qui contient des informations relatives à certaines villes du monde ;
- *country*, qui contient des informations relatives aux pays ;
- *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 qui est très important :

```

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

Nous pouvons constater qu'une relation 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 :

```
mysql> describe city;
```

Field	Type	Null	Key
ID	int	NO	PRI
Name	char(35)	NO	
CountryCode	char(3)	NO	
District	char(20)	NO	
Population	int	NO	

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* une chaîne d'au plus 35 caractères
- *CountryCode* une chaîne d'au plus 3 caractères
- *District* une chaîne d'au plus 20 caractères.

Ces caractéristiques sont fixées lors de la création d'une relation et influent sur les opérations que nous serons susceptibles de réaliser sur les valeurs de ces attributs :

- +, -, *, / pour les entiers et les nombres flottants,
- =, <, >, <=, >, < >=, pour les comparaisons (pour les chaînes de caractères, l'ordre lexicographique est utilisé).

La colonne suivante indique que tous les attributs doivent posséder une valeur.

La troisième colonne apporte une information importante : chaque enregistrement d'une relation doit pouvoir être identifié de manière unique ; c'est le rôle de la **clef primaire**, constitué 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 relation *country* :

```
mysql> describe country;
```

Field	Type	Null	Key
Code	char(3)	NO	PRI
Name	char(52)	NO	
Continent	enum('Asia','Europe','North America','Africa','Oceania','Antarctica','South America')	NO	
Region	char(26)	NO	
SurfaceArea	decimal(10,2)	NO	
IndepYear	smallint	YES	
Population	int	NO	
LifeExpectancy	decimal(3,1)	YES	
GNP	decimal(10,2)	YES	
GovernmentForm	char(45)	NO	
HeadOfState	char(60)	YES	
Capital	int	YES	

Ici, la clef primaire est l'attribut *Code*, qui est une chaîne de trois 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 relation, par exemple l'enregistrement qui concerne la France :

```
mysql> select * from country where Name = 'France';
```

Code	Name	Continent	Region	SurfaceArea	IndepYear	Population
FRA	France	Europe	Western Europe	551500.00	843	59225700

LifeExpectancy	GNP	GovernmentForm	HeadOfState	Capital
78.8	1424285.00	Republic	Jacques Chirac	2974

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 intrigant : 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 :

```
mysql> select * from city where ID = 2974;
```

ID	Name	CountryCode	District	Population
2974	Paris	FRA	Île-de-France	2125246

Regardons enfin le contenu de la troisième et dernière relation :

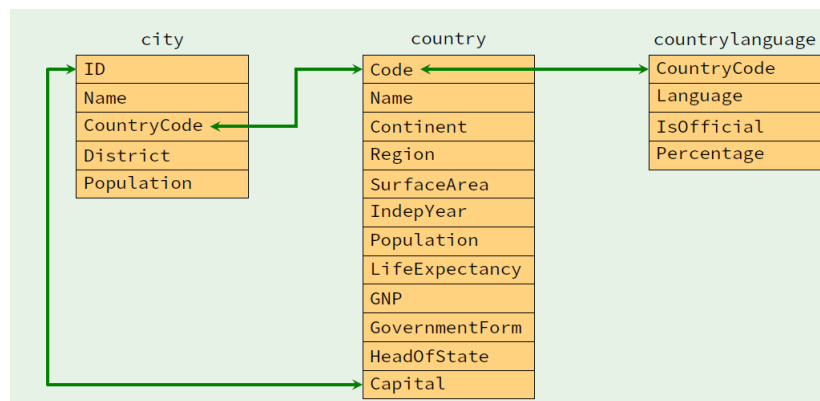
```
mysql> describe countryLanguage;
+-----+-----+-----+-----+
| Field      | Type          | Null | Key |
+-----+-----+-----+-----+
| CountryCode | char(3)       | NO   | PRI |
| Language    | char(30)     | NO   | PRI |
| IsOfficial  | enum('T','F') | NO   |     |
| Percentage  | decimal(4,1) | NO   |     |
+-----+-----+-----+-----+
```

Cette relation 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*).

Ainsi, 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** ; celui de notre exemple est représenté ci-dessous :



2 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 minuscules.

2.1 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.

En algèbre relationnelle, cette opération s'appelle une **projection** et se note :

$$\pi_{(A_1, \dots, A_n)}(R)$$

où A_1, \dots, A_n sont les attributs sélectionnés dans la relation R :

$$R : \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_1 & b_1 & c_3 \\ \hline \end{array} \text{ et } \pi_{(A,B)}(R) : \begin{array}{|c|c|} \hline A & B \\ \hline a_1 & b_1 \\ a_2 & b_2 \\ \hline \end{array}$$

Exemple 2.1. La liste des différentes langues présentes dans la base de donnée world s'obtient en écrivant :

```
mysql> SELECT DISTINCT Language FROM countrylanguage;
+-----+
| Language |
+-----+
| Dutch   |
| English |
| Papiamentó |
| Spanish |
+-----+
| Bemba   |
| Chewa   |
| Lozi    |
| Nsenga  |
+-----+
457 rows in set (0,00 sec)
```

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

```
SELECT * FROM table WHERE expression_logique
```

En algèbre relationnelle cette opération s'appelle une sélection et se note :

$$\sigma_E(R)$$

où E est l'expression logique et R une relation.

R :															
<table border="1" style="display: inline-table; border-collapse: collapse; text-align: center;"><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>a_1</td><td>b_1</td><td>c_1</td></tr><tr><td>a_2</td><td>b_2</td><td>c_2</td></tr><tr><td>a_3</td><td>b_3</td><td>c_3</td></tr><tr><td>a_4</td><td>b_4</td><td>c_4</td></tr></table>	A	B	C	a_1	b_1	c_1	a_2	b_2	c_2	a_3	b_3	c_3	a_4	b_4	c_4
A	B	C													
a_1	b_1	c_1													
a_2	b_2	c_2													
a_3	b_3	c_3													
a_4	b_4	c_4													

et $\sigma_E(R)$:

A	B	C
a_2	b_2	c_2
a_4	b_4	c_4

Ces deux opérations peuvent bien entendu être combinées pour extraire de la table une sélection d'attributs et d'enregistrements :

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

ce qui se note en algèbre relationnelle :

$$\pi_{(A_1, \dots, A_n)}(\sigma_E(R)).$$

Par exemple, nous pouvons visualiser les villes françaises et leurs populations respectives en écrivant :

```
mysql> SELECT Name, Population FROM city WHERE CountryCode = 'FRA';
+-----+-----+
| Name          | Population |
+-----+-----+
| Paris         | 2125246   |
| Marseille    | 798430    |
| Lyon          | 445452    |
| Toulouse     | 390350    |
+-----+-----+
| Argenteuil   | 93961     |
| Tourcoing    | 93540     |
| Montreuil    | 90674     |
+-----+-----+
40 rows in set (0,00 sec)
```

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 s'écrit :

$$\rho_{a \leftarrow b}(R)$$

en algèbre relationnelle et à l'aide du mot-clef AS en SQL :

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

La nécessité du renommage apparaîtra lorsque nous aborderons les sous-requêtes.

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 :


```
mysql> SELECT Continent, SUM(Population) FROM country
-> GROUP BY Continent;
```

Continent	SUM(Population)
North America	482993000
Asia	3705025700
Africa	784475000
Europe	730074600
South America	345780000
Oceania	30401150
Antarctica	0

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

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

Continent	Pop
Asia	3705025700
Africa	784475000
Europe	730074600
North America	482993000
South America	345780000
Oceania	30401150
Antarctica	0

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

```
mysql> SELECT Continent, SUM(Population), COUNT(*) FROM country
-> GROUP BY Continent HAVING COUNT(*) > 40;
```

Continent	SUM(Population)	COUNT(*)
Asia	3705025700	51
Africa	784475000	58
Europe	730074600	46

2.5 Opérateurs ensemblistes

On peut procéder à des opérations ensemblistes entre deux tables (en général le résultat de requêtes) à condition que celles-ci aient 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 $A \setminus (A \cap B)$ de deux requêtes.

Ces opérations présentent surtout un intérêt lorsque les deux requêtes sont issues de deux relations différentes.

MySQL (ainsi que d'autres SGBD) n'implémente 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 EXCEPT SELECT b FROM table2` est équivalent à :

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

Autrement dit, seule l'union présente un réel intérêt.

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

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

mais la création d'un produit cartésien doit être réalisé avec prudence, car le cardinal du résultat est égal au produit des cardinaux des tables qui le composent :

```
mysql> select count(*) from city
-> union select count(*) from country
-> union select count(*) from countrylanguage
-> union select count(*) from city, country, countrylanguage;
+-----+
| count(*) |
+-----+
|      4079 |
|       239 |
|       984 |
| 959282904 |
+-----+
```

3 Application

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*(*idj*, *nom*, *pays*) et *Parties*(*idp*, *date*, *duree*, *score*, *idjoueur*) où :

- *idj* 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 ;

