Chapitre 4

Bases de données relationnelles

1. Description d'une base de données

Une base de données est un ensemble de *tables*[1] que l'on peut représenter sous forme de tableaux bi-dimensionnels. La base qui va nous servir d'exemple comporte deux tables : une table nommée `departements` et une table nommée `communes`. Chacune de ces tables possède un certain nombre d'*attributs* décrits figure 1.

- Attributs de la table `departements` :
  - `id` (un entier) la clef primaire de l'enregistrement ;
  - `code` (une chaîne de caractères) le code administratif du département ;
  - `nom` (une chaîne de caractères) le nom du département ;
  - `chef_lieu` (un entier) l'identifiant du chef-lieu du département.
- Attributs de la table `communes` :
  - `id` (un entier) la clé primaire de l'enregistrement ;
  - `nom` (une chaîne de caractères) le nom de la commune ;
  - `superficie` (un flottant) la superficie de la commune, en km^2 ;
  - `altitude` (un entier) l'altitude de la commune, en mètres ;
  - `population` (un entier) la population de la commune ;
  - `departement` (un entier) l'identifiant du département où se trouve la commune.

Figure 1 – les attributs des tables `communes` et `departements`

Chaque attribut est un objet typé appelé ici *domaine*. Par exemple le domaine des attributs `nom` de chacune de ces deux tables est une chaîne de caractères, le domaine de l'attribut `superficie` est un nombre flottant et le domaine de l'attribut `population` un entier.

Les attributs désignent les éléments de chacune des colonnes du tableau qui représente la relation ; les lignes en forment les *enregistrements* : chacun d'eux est un n-uplet dont les éléments appartiennent à chaque colonne de la table. En général, une table contient un grand nombre d'enregistrements, et l'utilisateur connaît uniquement les attributs et leurs domaines respectifs (ce qu'on appelle le *schéma* de la relation) lorsqu'il interagit avec elle. Le mot-clef `NULL` est utilisé lorsqu'un des attributs d'un enregistrement n'est pas renseigné dans la base.

ID	CODE	NOM	CHEF_LIEU
79	"18"	"CHER"	6413
125	"2B"	"HAUTE-CORSE"	11315
216	"78"	"YVELINES"	32352

ID	NOM	SUPERFICIE	ALTITUDE	POPULATION	DEPARTEMENT
1356	"AINAY-LE-VIEIL"	1379,2	180	253	79
11315	"BASTIA"	1977,0	260	43553	125
28645	"VAUDREUILLE"	1146,3	332	352	32

Figure 2 – Un extrait des tables `departements` et `communes`.

1. Ou de *relations*, les deux termes étant synonymes dans ce contexte.

• **Clef primaire**
Chaque enregistrement d'une table doit pouvoir être identifié de manière unique : c'est le rôle de la *clef primaire*, qui est constituée d'un ou plusieurs attributs. Dans les deux tables qui nous servent d'exemple, la clef primaire est constituée de l'unique attribut `id`, dénué de toute autre signification.

2. Requêtes SQL

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 $\boxed{\pi_{(A_1,\dots,A_n)}(R)}$ où $A_1,\dots,A_n$ sont les attributs sélectionnés dans la relation R.

R

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_1	b_1	c_3

$\pi_{(A,B)}(R)$

A	B
a_1	b_1
a_2	b_2

- 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 $\boxed{\sigma_E(R)}$ où E est l'expression logique et R une relation.

R

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

$\sigma_E(R)$

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

- 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 $\boxed{\rho_{a\leftarrow b}(R)}$ 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, mais les mots-clefs qui réalisent ces opérations ne sont pas normalisés. En SQLite 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^e enregistrement.

Exercice 1.

a) Rédiger une requête SQL donnant le nom des dix communes les plus vastes parmi celles situées à plus de 1 000 mètres d'altitude.

b) Rédiger une requête SQL donnant le nom des cinq communes les moins densément peuplées.

2.2 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. Cette nouvelle relation se note $\boxed{R_1 \bowtie_E R_2}$.
En SQL on réalise une jointure par la requête :

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

R_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

R_2

D	E
a_1	e_1
a_2	e_2

$R_1 \bowtie_{A=D} R_2$

A	B	C	D	E
a_1	b_1	c_1	a_1	e_1
a_2	b_2	c_2	a_2	e_2

Seules les conditions d'égalité entre deux attributs figurent au programme.

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 `nom` présent dans les deux tables `communes` et `departements`. Pour les distinguer 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 départements est désigné par `departements.nom` et celui des communes par `communes.nom`.
Notons que le mot-clef `AS` permet de renommer les tables afin d'éviter d'écrire des noms trop longs.

Exemple. Les deux tables `departements` et `communes` possèdent deux jointures naturelles :

- on peut identifier les attributs `departements.id` et `communes.departement`. Dans ce cas, la table résultante de la jointure possédera autant d'entrées que la table `communes` (à condition que ces attributs ne prennent jamais la valeur `NULL`) ;
- on peut identifier les attributs `departements.chef_lieu` et `communes.id`. Dans ce cas, la table résultante de la jointure possédera autant d'entrées que la table `departements` et seules les communes qui sont des chefs-lieux y seront présentes.

Exercice 2.

a) Rédiger une requête SQL donnant le nom des départements dans lesquels se trouvent des communes situées à plus de 1 000 mètres d'altitude.

b) Rédiger une requête SQL donnant le nom du chef-lieu du département de l'Allier.

c) Rédiger une requête SQL donnant le département dans lequel se trouve la commune de plus de 10 000 habitants la plus haute de France.

d) Rédiger une requête SQL donnant le nom du département dont le chef-lieu est le moins peuplé de France.

• Sous-requêtes
Considérons le problème suivant : je souhaite connaître les chefs-lieux des départements dans lesquels se trouvent une commune nommée Sainte-Colombe (il y en a plusieurs). Il est facile de répondre à cette question en utilisant deux requêtes : la première pour déterminer les départements dans lesquels se trouvent une commune nommée ainsi, la seconde pour déterminer les chefs-lieux correspondants. Mais il est possible de répondre à cette question en une seule phrase SQL imbriquant ces deux requêtes, car le résultat d'une requête peut être intégré au sein d'une autre requête.

La sous-requête doit impérativement être délimitée par des parenthèses, et peut être située :

- en lieu et place d'une table après le **FROM** ;
- ou au sein d'une expression logique après le **WHERE**.

Le problème posé peut par exemple être résolu de la façon suivante :

```
SELECT c.nom FROM communes AS c JOIN departements AS d
                ON d.chef_lieu = c.id
WHERE d.id IN (SELECT d.id FROM communes AS c JOIN departements AS d
                            ON c.departement = d.id
                WHERE c.nom = "SAINTE-COLOMBE")
```

Noter 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.

2.3 Fonctions d'agrégation

SQL possède un certain nombre de fonctions statistiques (voir la liste figure 3) 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 d'un attribut
AVG()	moyenne d'un attribut

FIGURE 3 – Fonctions statistiques.

Par exemple, pour obtenir la superficie de la Gironde on écrirait :

```
SELECT SUM(communes.superficie) FROM communes JOIN departements
                                ON communes.departement = departements.id
WHERE departements.nom = "GIRONDE"
```

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.

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_3	c_3
a_3	b_4	c_4
a_3	b_5	c_5

group by A →

R

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_3	c_3
a_3	b_4	c_4
a_3	b_5	c_5

On utilise la syntaxe suivante (dans laquelle a_1 et a_2 sont des attributs et f une fonction d'agrégation) :

```
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 a_2, puis applique la fonction f sur l'attribut a_1 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 e_1, les regroupe selon la valeur de l'attribut a_2, puis applique la fonction f sur l'attribut a_1 à chacun des groupes vérifiant l'expression logique e_2.

Exercice 3.

a) Rédiger une requête SQL donnant la liste des départements accompagnés de leur superficies respectives.

b) Modifier cette requête pour la restreindre aux départements bretons. On rappelle que la Bretagne est constituée des départements de codes administratifs 22, 29, 35, 56[a].

c) Rédiger une requête SQL donnant la liste des départements dont la densité de population est inférieure à 25 habitants par km^2.

d) Rédiger une requête SQL calculant la superficie moyenne des départements français.

e) Rédiger une requête SQL donnant le nom du département le plus vaste de France sans utiliser **ORDER BY**.

a. Rajouter 44 si vous êtes breton.