Chapitre 4

Bases de données relationnelles

Ce chapitre est un document de révision. Se référer à votre cours de première année pour un cours plus complet sur le sujet.

Introduction

Schématiquement, une *base de données relationnelle* (BDR) est un ensemble de *tables* contenant des données reliées entre elles par des *relations* ; on y extrait de l'information par le biais de *requêtes* exprimées dans un langage appelé SQL (*Structured Query Language*).

Python et SQL

Le système de gestion de bases de données relationnelles (SGBDR) que nous utiliserons préférentiellement s'appelle SQLite ; il présente l'avantage d'être présent dans la bibliothèque standard de Python. Cela signifie donc que vous pouvez écrire en Python une application contenant son propre SGBDR intégré à l'aide du module `sqlite3`. Vous trouverez en annexe un script Python rudimentaire mais suffisant pour pouvoir interagir avec une base de donnée enregistrée sur votre disque dur.

Algèbre relationnelle

SQL est le langage de prédilection pour interagir avec une BDR, mais ce l'est pas le seul. L'*algèbre relationnelle* fournit un cadre théorique indépendant du langage, proche de la théorie des ensembles. Les opérations sur les BDR y sont définies de manière formelle et permettent de les composer efficacement entre-elles. La formulation abstraite dans le cadre de l'algèbre relationnelle permet d'obtenir des requêtes non seulement correctes mais aussi et surtout efficaces.

1. Le langage SQL

Pour illustrer ce document nous allons utiliser une base de données qui se nomme `mondial.sq3` que vous pouvez récupérer sur le site de ce cours. À condition de bénéficier d'une liaison avec internet vous pouvez aussi vous rendre à l'adresse : `http://www.semwebtech.org/sqlfrontend/` ; vous y trouverez une interface graphique permettant d'interroger directement cette BDR en utilisant un simple navigateur.

1.1 Requêtes de base

Cette base de données contient de nombreuses tables de données géographiques. Parmi celles-ci on trouve une table nommée `country` qui possède six attributs :

```
Name    Code    Capital    Province    Area    Population
```

Les quatre premiers sont des chaînes de caractères [1], le cinquième un nombre flottant et le dernier un entier ; le code du pays est la *clé primaire* de la table : son unicité est garantie, ce qui permet d'identifier de manière unique un enregistrement de cette table.

Remarque. On pourrait trouver pertinent de choisir comme clé primaire le nom du pays, mais ce serait une mauvaise idée : il peut arriver qu'un pays change de dénomination après un changement de régime politique, voire que deux états distincts revendiquent le même nom ; en revanche il ne changera pas de code, ce qui sera précieux lorsqu'il faudra réaliser des jointures entre différentes tables.

1. La province désigne la région où se situe la capitale.

Name	Code	Capital	Province	Area	Population
Lebanon	RL	Beirut	Lebanon	10400.0	3776317
West Bank	WEST	None	None	5860.0	1427741
Japan	J	Tokyo	Tokyo	377835.0	125449703
South Korea	ROK	Seoul	South Korea	98480.0	45482291
Maldives	MV	Male	Maldives	300.0	270758
Oman	OM	Muscat	Oman	212460.0	2186548
Mexico	MEX	Mexico City	Distrito Federal	1972550.0	95772462
Canada	CDN	Ottawa	Ontario	9976140.0	28820671

FIGURE 1 – Un extrait de la table `country`.

Commençons par extraire de cette table le nom de tous les pays qu'elle contient (il y en a 241) :

```
SELECT name FROM country
```

Les mots-clés **SELECT** ... **FROM** réalisent l'interrogation de la table. Dans le cas de l'exemple ci-dessus on ne liste qu'un seul des attributs de la table, pour en avoir plusieurs on sépare les attributs par une virgule ; pour les avoir tous on les désigne par une étoile. Par exemple, les deux requêtes qui suivent donnent pour la première le nom de chacun des pays ainsi que leurs capitales, pour la seconde l'intégralité des données de la table :

```
SELECT name, capital FROM country
SELECT * FROM country
```

Le mot-clé **WHERE** filtre les données qui répondent à un critère de sélection. Par exemple, pour connaître le nom de la capitale du Botswana on écrira :

```
SELECT capital FROM country WHERE name = 'Botswana'
```

On peut observer figure 1 que West Bank (la Cisjordanie) n'a pas de capitale (reconnue internationalement). Pour connaître le nom de tous les territoires dans ce cas :

```
SELECT name FROM country WHERE capital is NULL
```

Différentes clauses permettent de formuler des requêtes plus élaborées ; la figure 2 rassemble les instructions les plus fréquentes.

SELECT *	sélection des colonnes
SELECT DISTINCT *	sélection sans doublon
FROM table	nom d'une ou plusieurs tables
WHERE condition	imposer une condition
GROUP BY expression	grouper les résultats
HAVING condition	condition sur un groupe
UNION | **INTERSECT** | **EXCEPT**	opérations ensemblistes sur les requêtes
ORDER BY expression	trier les résultats
LIMIT number	limiter à n enregistrements
OFFSET start	débuter à partir de n enregistrements

FIGURE 2 – Principales requêtes SQL.

Exercice 1 Rédiger une requête SQL pour obtenir :

1. la liste des pays dont la population excède 60 000 000 d'habitants ;
2. la même liste triée par ordre alphabétique ;
3. la même liste triée par ordre décroissant de population ;
4. le nom des dix plus petits pays ;
5. le nom des dix suivants.

1.2 Jointures

L'intérêt d'une base de données réside en particulier dans la possibilité de croiser des informations présentes dans plusieurs tables par l'intermédiaire d'une *jointure*. Dans la base de données qui nous occupe on trouve une table nommée `encompasses` qui possède trois attributs :

`Country` `Continent` `Percentage`

Le premier attribut est le code du pays, le deuxième le nom du continent et le dernier la portion du pays présente sur ce continent. La clé primaire de cette table est le couple (`Country`, `Continent`), et la valeur du troisième argument ne peut pas être nulle.
Cette seconde table possède un attribut en commun avec la première table : l'attribut `Country` de la table `encompasses` est identique à l'attribut `Code` de la table `country` et va nous permettre par son intermédiaire de croiser les informations de ces deux tables.
Par exemple, pour connaître la liste des pays dont une fraction au moins est en Europe on écrira la requête :

```
SELECT country.name
FROM country JOIN encompasses
ON country.code = encompasses.country
WHERE encompasses.continent = 'Europe'
```

Les mots-clés **JOIN** ... **ON** créent une table intermédiaire formée du produit cartésien des dexu tables et applique ensuite la requête sur la nouvelle relation.

Remarque. L'interrogation de plusieurs tables simultanément rend nécessaire le préfixage de l'attribut par le nom de la table pour le cas où certaines d'entres-elles auraient des noms d'attributs en commun. On peut alléger cette syntaxe à l'aide d'alias pour la rendre plus compacte. Par exemple, la requête précédente peut s'écrire plus succinctement :

```
SELECT c.name
FROM country c JOIN encompasses e
ON c.code = e.country
WHERE e.continent = 'Europe'
```

Exercice 2 Rédiger une requête SQL pour obtenir :

1. le nom des pays qui sont à cheval sur plusieurs continents ;
2. les pays du contient américain qui comptent moins de 10 habitants par km^2.

Dans la base de données figure une table nommée `city` qui possède les attributs suivants :

`Name` `Country` `Province` `Population` `Longitude` `Latitude`

(L'attribut `Country` est le code du pays.)

3. Déterminer les capitales européennes situées à une latitude supérieure à 60°.

1.3 Fonctions d'agrégation

Il est possible de regrouper les lignes d'une table suivant certains critère à l'aide de fonctions d'*agrégation* (on trouvera la liste des fonctions statistiques figure 3). Par exemple, pour obtenir le nombre de pays africains on écrira :

```
SELECT COUNT(*)
FROM country c JOIN encompasses e ON c.code = e.country
WHERE e.continent = 'Africa'
```

De même, pour connaître le nombre d'habitants sur le continent américain on demandera :

```
SELECT SUM(c.population)
FROM country c JOIN encompasses e ON c.code = e.country
WHERE e.continent = 'America'
```

COUNT( )	nombre d'enregistrements sur une table ou une colonne distincte
MAX( )	valeur maximale d'une colonne
MIN( )	valeur minimale d'une colonne
SUM( )	calcul de la somme sur un ensemble d'enregistrement
AVG( )	calcul de la moyenne sur un ensemble d'enregistrement

Figure 3 – Fonctions statistiques.

Exercice 3 La table `language` possède les attributs suivants :

`Country` `Name` `Percentage`

L'attribut `Country` est le code du pays, `Name` le nom d'une langue parlée dans celui-ci, et `Percentage` la proportion d'habitants dont c'est la langue maternelle.

1. Donner la liste ordonnée des dix langues parlées dans le plus de pays différents.
2. Quelles sont les cinq langues les plus parlées au monde (et présentes dans la base) ? On précisera pour chacune d'elles le nombre de personnes qui la parlent.
3. Dans quel pays partiellement francophone la langue française est-elle la plus minoritaire ?

2. Algèbre relationnelle

De même que l'algorithmique permet de formuler un problème informatique indépendamment d'un langage de programmation particulier, l'algèbre relationnelle définit un cadre formel dans lequel exprimer des requêtes et des relations. Elle s'appuie très largement sur la théorie des ensembles et propose un ensemble d'opérations formelles qui permettent de construire de nouvelles relations à partir de relations existantes.

2.1 Opérations ensemblistes

Trois opérations ensemblistes de base peuvent être effectuées avec les relations : les opérations d'union ($\cup$), d'intersection ($\cap$) et de différence ($-$).

SELECT * **FROM** t1 **UNION SELECT** * **FROM** t2	enregistrements présents dans l'une des deux tables t_1 ou t_2 (sans répétition)
SELECT * **FROM** t1 **INTERSECT SELECT** * **FROM** t2	enregistrements présents dans les deux tables t_1 et t_2
SELECT * **FROM** t1 **EXCEPT SELECT** * **FROM** t2	enregistrements présents dans t_1 mais pas dans t_2

Figure 4 – Les trois opérations ensemblistes de base en SQL.

Pour être réalisées, ces opérations doivent agir sur des relations ayant le même schéma relationnel :

R_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2

A	B	C
a_1	b_1	c_1
a_3	b_3	c_3

$R_1 \cup R_2$

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

$R_1 \cap R_2$

A	B	C
a_1	b_1	c_1

$R_1 - R_2$

A	B	C
a_2	b_2	c_2

2.2 Opérations spécifiques à l'algèbre relationnelle

Projection
La *projection* extrait une relation d'une relation donnée en supprimant des attributs de cette dernière. Si $A_1, A_2, \ldots, A_n$ sont des attributs d'une relation R, la projection de R suivant $A_1, A_2, \ldots, A_n$ est l'ensemble des enregistrements de R dont les attributs sont $A_1, A_2, \ldots, A_n$ et qui ne se répètent pas. On la note :

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

En SQL la projection se traduit par la requête :

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

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

Sélection
La *sélection* permet d'extraire les enregistrements d'une relation R qui satisfont une expression logique E. On la note :

$$\sigma_E(R)$$

En SQL la sélection se traduit par la requête :

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

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

Renommage
Le *renommage* permet la modification du nom d'un ou plusieurs attributs d'une relation. Renommer l'attribut a en l'attribut b dans la relation R s'écrit :

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

En SQL le renommage se traduit par la requête :

ALTER TABLE table **RENAME COLUMN** a **TO** b

Attention, le renommage n'est pas possible avec SQLite (seule le renommage d'une table est possible).

R

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

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

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

Jointure
La *jointure* est une opération qui porte sur deux relations R_1 et R_2 et retourne une relation qui comporte les enregistrements des deux premières relations qui satisfont une contrainte logique E. Cette nouvelle relation se note :

$$R_1 \bowtie_E R_2$$

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	E
a_1	b_1	c_1	e_1
a_2	b_2	c_2	e_2

Produit et division cartésienne
Le *produit cartésien* de deux relations R_1 et R_2 se note :

$$\boxed{R_1 \times R_2}$$

et se traduit en SQL par :

```
SELECT * FROM table1, table2
```

R_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2

D	E
d_1	e_1
d_2	e_2

$R_1 \times R_2$

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2

Effectuer le produit cartésien de deux tables de grandes tailles n'est pas une opération toujours pertinente, mais il constitue une opération de base pour définir d'autres opérations plus complexes. Ainsi, la jointure est un produit cartésien suivi d'une sélection :

$$R_1 \bowtie_E R_2 = \sigma_E(R_1 \times R_2).$$

Enfin, la *division cartésienne* est encore une opération qui produit une relation à partir de deux relations R_1 et R_2 vérifiant $R_2 \subset R_1$. La relation obtenue possède tous les attributs de R_1 que ne possède pas R_2 ; on la note :

$$\boxed{R_1 \div R_2}$$

et se caractérise par : $\forall x \in R_1 \div R_2$, $\forall y \in R_2$, $xy \in R_1$. Cette opération n'a pas d'équivalent en SQL.

R_1

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_2
a_2	b_2	c_3	d_3
a_3	b_3	c_1	d_1
a_3	b_3	c_2	d_2

R_2

C	D
c_1	d_1
c_2	d_2

$R_1 \div R_2$

A	B
a_1	b_1
a_3	b_3

Les amateurs prendront plaisir à prouver le théorème suivant :

Théorème. — *Si* $S_1 = \pi_{R_1 - R_2}(R_1)$ *et* $S_2 = \pi_{R_1 - R_2}(S_1 \times R_2 - R_1)$ *alors* $R_1 \div R_2 = S_1 - S_2$.

Exercice 4 Donner un sens aux expressions de l'algèbre relationnelle suivantes :

1. $\pi_{Name}(\sigma_{Latitude>66}(\mathsf{City}) \cap \sigma_{Population>10000}(\mathsf{City}))$;
2. $\pi_{Country.Name}(\sigma_{City.Latitude<0}(x) - \sigma_{City.Latitude>-23}(x))$ avec $x = \mathsf{Country} \bowtie_{country.code=city.country} \mathsf{City}$.

Exercice 5 Et maintenant que vous êtes fans d'algèbre relationnelle, prouver le théorème ci-dessus (vous n'espériez quand même pas y échapper ?)

Annexe : interaction avec une base de données en Python

Le module permettant d'intégrer un SGBDR à un environnement Python s'appelle `sqlite3` ; une fois ce dernier importé, on se connecte à une base de données par l'intermédiaire de la fonction `connect`, en précisant en paramètre un chemin d'accès vers la base de données. Par exemple, pour utiliser la base de données `mondial.sq3` (et en supposant que celle-ci est dans le répertoire courant) on écrira :

```
import sqlite3 as sql

base = sql.connect("mondial.sq3")
```

La méthode `cursor` appliquée à la connexion que nous venons de créer crée un intermédiaire entre l'interface et la BDR destiné à mémoriser temporairement les données en cours de traitement ainsi que les opérations que vous effectuez sur elles, avant leur transfert définitif vers la base de données. Son utilisation permet donc d'annuler si nécessaire plusieurs opérations qui se seraient révélées inadéquates sans que la base de données n'en soit affectée.

```
cur = base.cursor()
```

Une fois le curseur créé, la méthode `execute` permet de transmettre des requêtes rédigées en SQL sous forme de chaîne de caractères :

```
cur.execute("ici on écrit une requête en langage SQL")
```

Enfin, si des modifications ont été effectuées sur la BDR, il faut appliquer la méthode `commit` à la connexion créée pour qu'elles deviennent définitives. On peut ensuite refermer le curseur et la connexion :

```
base.commit()
cur.close()
base.close()
```

Voici enfin un script permettant un dialogue interactif avec la base de données `mondial.sq3` utilisée dans ce document :

```
import sqlite3

base = sqlite3.connect ("mondial.sq3")
cur = base.cursor ()

while True:
    requete = input("Veuillez entrer une requête SQL (ou <Enter> pour terminer) :")
    if requete == "":
        break
    try:
        cur.execute(requete)
    except:
        print('*** Requête SQL incorrecte ***')
    else:
        for enreg in cur:
            print(enreg)
    print()
choix = input("Confirmez-vous l'enregistrement de l'état actuel (o/n) ? ")
if choix == "o":
    base.commit()
cur.close()
base.close()
```