Chapitre 10
Bases de données relationnelles

Introduction

Au sens premier du terme, l'informatique est la science du traitement automatique de l'information ; à ce titre, la structuration des données en est un élément essentiel. Jusqu'à présent, nous avons essentiellement manipulé des tableaux, qui supposent l'existence d'un ordre permettant le classement de l'information. Mais un tel classement présente des limites : plusieurs critères peuvent être également pertinents, et ranger ces données dans un tableau exige d'en privilégier certains au détriments d'autres.
Notons que le problème de l'ordonnancement de l'information n'est pas nouveau et précède la naissance de l'informatique de plusieurs siècles : encore aujourd'hui les bibliothèques publiques utilisent un système de classification inventé au XIX^e siècle : la classification décimale de Dewey. Les documents sont répartis en 10 classes, chaque classe est divisée en 10 divisions, chaque division en 10 subdivisions, et ainsi de suite. Cette classification permet depuis bientôt 150 ans de ranger nos bibliothèques, mais n'en présente pas moins de nombreux défauts : l'information est hiérarchisée suivant des critères qui étaient pertinents au moment de l'élaboration de ce système mais qui ne le sont plus forcément aujourd'hui mais surtout, si elle facilite le travail du classificateur, elle ne contribue pas à faciliter la tâche du chercheur, à moins que ce dernier ne sache très précisément à quelle discipline rattacher l'objet de sa recherche. A contrario, les logiciels de gestion des livres numériques gèrent sans peine des milliers de références en autorisant des recherches multi-critères (et multi-bibliothèques) sans qu'il soit nécessaire pour l'utilisateur de connaître la structuration interne des données.
Ces outils informatiques utilisent tous des *bases de données relationnelles* (BDR) qui offrent un moyen d'organiser efficacement les données et de les manipuler grâce à des requêtes. Schématiquement, une base de données 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 spécifique.

Les principes et l'architecture d'une base de données

Un *système de gestion de bases de données* (SGBD) est un logiciel qui organise et gère les données de façon transparente pour l'utilisateur. Ce sont des logiciels dont la conception est bien trop complexe pour pouvoir être abordée dans ce cours ; nous nous contenterons d'interagir avec eux par l'intermédiaire de requêtes exprimées dans un langage devenu standard au fil des temps : le langage SQL (pour *Structured Query Language*). La majorité des SGBD comprend au moins un sous-ensemble des vocables SQL, agrémenté d'un certain nombre d'expressions qui leur sont spécifiques.

• Architecture trois-tiers

Sur un réseau informatique, des informations sont en permanence échangées entre deux machines, un logiciel assurant le traitement des informations sur chacune d'entre elles. On distingue le *logiciel client* installé sur la machine qui envoie des requêtes du *logiciel serveur* installé sur la machine qui traite les requêtes. Par extension, les machines sont également désignées par les noms de client et serveur. Ce mode de communication est appelé architecture à deux niveaux. C'est l'une des architectures client-serveur possibles.

L'architecture trois-tiers [1] est une architecture client-serveur qui ajoute un niveau supplémentaire dans l'environnement précédemment décrit. Un serveur de données transmet les informations à un serveur d'application qui, à son tour, transmet les informations traitées vers un client. Ce modèle d'architecture présente plusieurs avantages :

- meilleure prise en compte de l'hétérogénéité des plates-formes ;
- amélioration de la sécurité des données en supprimant le lien entre le client et les données ;
- meilleure répartition des tâches entre les différentes couches logicielles.

1. De l'anglais *tier*, qui signifie niveau.

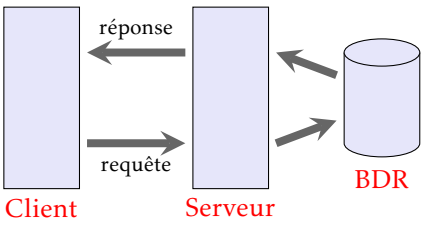

FIGURE 1 – Architecture à deux niveaux

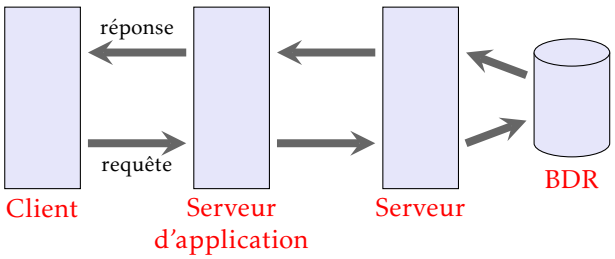

FIGURE 2 – Architecture à trois niveaux

Considérons à titre d'exemple la base de données MONDIAL que nous allons utiliser pour illustrer ce cours. Il s'agit d'une BDR qui compile un certain nombre de données géographiques et qui est gérée par l'université de Göttingen. Il est possible d'interagir avec elle en utilisant un formulaire que l'on trouve à l'adresse : `http://www.semwebtech.org/sqlfrontend/`

FIGURE 3 – L'interface client de la BDR MONDIAL.

Nous sommes en présence d'une architecture trois tiers : la première couche (le client) est représentée en HTML pour être exploitée par un navigateur web et sert d'interface entre l'homme et la machine. La seconde couche (le serveur d'application) est un serveur web qui reçoit des données textuelles de la part du client, les transmet sous la forme de requêtes SQL au serveur de la base puis actualise la page web du client pour y intégrer la réponse du serveur. Enfin, la troisième couche est un SGBD, ici ORACLE DATABASE, qui gère la base de données et répond aux sollicitations du serveur d'application.

• Python et sql

SQLite est un autre SGBD qui présente l'avantage d'être présent dans la bibliothèque standard de Python. Cela signifie que vous pouvez écrire en Python une application contenant son propre SGBD intégré à l'aide du module `sqlite3`. Dans ce cas, il ne s'agit plus à proprement parlé d'une interface client-serveur puisque les données sont intégralement stockées dans un fichier indépendant. Vous trouverez en annexe un script Python rudimentaire mais suffisant pour pouvoir interagir avec une base de donnée enregistrée sur votre disque dur.

Enfin, on notera qu'il existe quelques différences entre les dialectes SQL utilisés par Oracle et par SQLite. Ces différences seront indiquées dans ce document mais de toute façon, elles ne concernent que des notions hors programme.

Remarque. Durant ce cours on pourra remarquer des différences entre les réponses fournies par la base de données suivant que l'on interroge la base en ligne ou celle enregistrée sous la forme d'un fichier sur le disque dur. Ces différences s'expliquent bien entendu par le fait que la première est mise à jour régulièrement, contrairement à la seconde.

1. Le langage SQL

1.1 Relations

Nous l'avons dit, une base de données est un ensemble de *tables* [2] que l'on peut représenter sous la forme de tableaux bi-dimensionnels. Par exemple, la base de données mondial contient (entre autre) une table nommée `country` [3] qui possède six *attributs* :

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

Chaque attribut est un objet typé appelé ici *domaine*. Par exemple le domaine des quatre premiers attributs est une chaîne de caractères, le domaine du cinquième attribut 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 le client de la BDR connaît uniquement les attributs et leurs domaines respectifs (ce qu'on appelle le *schéma* de la relation) lorsqu'il interagit avec elle.

NAME	CODE	CAPITAL	PROVINCE	AREA	POPULATION
France	F	Paris	Ile de France	547030.	64933400
Spain	E	Madrid	Madrid	504750.	46815916
Austria	A	Wien	Wien	83850.	8499759
Czech Republic	CZ	Praha	Praha	78703.	10562214
Germany	D	Berlin	Berlin	356910.	80219695
Hungary	H	Budapest	Budapest	93030.	9937628
Italy	I	Roma	Lazio	301230.	59433744
Liechtenstein	FL	Vaduz	Liechtenstein	160.	36636

Figure 4 – Un extrait de la table `country`.

• Clé primaire

En général, une base de données contient plusieurs tables et l'on peut souhaiter croiser les données présentes dans plusieurs d'entre elles (nous verrons cela plus loin). Pour cela il est nécessaire de pouvoir identifier par une caractérisation unique chaque enregistrement d'une table ; c'est le rôle de la *clé primaire*. En général constituée d'un attribut (mais ce n'est pas une règle, certaines clé primaires peuvent être composées de plusieurs attributs), elle garantit que deux enregistrements distincts ont deux clés primaires distinctes.
Dans le cas de la table `country`, la clé primaire est l'attribut `Code` ; il est fréquent que parmi les attributs d'une table on trouve un identifiant, en général dénué de signification, dont le seul rôle est de jouer le rôle de clé primaire, comme c'est le cas ici.

2. Ou de *relations*, les deux termes étant synonymes dans ce contexte.
3. Notons que les enregistrements de cette table ne sont pas tous des pays mais plus généralement des entités politiques, ce qui explique par exemple que l'île de la Réunion soit une entrée distincte de la France.

1.2 Requêtes de base

Commençons par extraire de cette table le nom de tous les pays qu'elle contient :

```
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'
```

Il se peut que certains attributs d'un enregistrement soient manquants ; dans ce cas la valeur de cet attribut est **NULL**. Par exemple, dans la table `country` un territoire ne possède pas de capitale ; pour le connaître on produit la requête :

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

Différentes clauses permettent de formuler des requêtes plus élaborées ; la figure 5 rassemble les instructions les plus fréquentes.

SELECT *	sélection des colonnes
SELECT DISTINCT *	sélection sans doublon
FROM table	nom d'une table
WHERE condition	imposer une condition
GROUP BY expression	grouper les résultats
HAVING condition	condition sur un groupe
ORDER BY expression **ASC** / **DESC**	trier les résultats par ordre croissant / décroissant
LIMIT n	limiter à n enregistrements (SQLite)
OFFSET n	débuter à partir de n enregistrements (SQLite)
OFFSET n **ROWS**	débuter à partir de n enregistrements (Oracle)
FETCH FIRST n **ROWS ONLY**	limiter à n enregistrements (Oracle)
UNION | **INTERSECT** | **EXCEPT**	opérations ensemblistes sur les requêtes

Figure 5 – Principales requêtes SQL.

Exercice 1 Rédiger une requête SQL pour obtenir :

a) la liste des pays dont la population excède 60 000 000 d'habitants ;

b) la même liste triée par ordre alphabétique ;

c) la liste des pays et de leurs populations respectives, triée par ordre décroissant de population ;

d) le nom des dix pays ayant la plus petite superficie ;

e) le nom des dix suivants.

1.3 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 d'un enregistrement 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 deux 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'
```

Ainsi, réaliser cette jointure revient à créer une table (virtuelle) nommée :

country c **JOIN** encompasses e **ON** c.code = e.country

qui possède huit attributs :

c.Code = e.Country c.Name c.Capital c.Province c.Area c.Population e.Continent e.Percentage

NAME	CODE	CAPITAL	PROVINCE	AREA	POPULATION	COUNTRY	CONTINENT	PERCENTAGE
Bulgaria	BG	Sofia	Bulgaria	110910.	7284552	BG	Europe	100
Romania	RO	Bucuresti	Bucuresti	237500.	20121641	RO	Europe	100
Turkey	TR	Ankara	Ankara	780580.	75627384	TR	Asia	97
Turkey	TR	Ankara	Ankara	780580.	75627384	TR	Europe	3
Denmark	DK	Kobenhavn	Hovedstaden	43070.	5580516	DK	Europe	100

FIGURE 6 – Un extrait de la jointure entre les tables `country` et `encompasses`.

Exercice 2 Rédiger une requête SQL pour obtenir :

a) le nom des pays qui sont à cheval sur plusieurs continents ;

b) les pays du contient américain qui comptent moins de 10 habitants par km^2.

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

1.4 Fonctions d'agrégation

Il est possible de regrouper certains enregistrements d'une table par *agrégation* à l'aide du mot-clé **GROUP BY** pour ensuite leur appliquer une fonction (on trouvera figure 7 quelques exemples de fonctions statistiques disponibles). Par exemple, pour connaître le nombre de pays de chaque continent on écrira :

```
SELECT e.continent, COUNT(*)
FROM country c JOIN encompasses e ON c.code = e.country
GROUP BY e.continent
```

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 7 – Fonctions statistiques.

Enfin, on notera que c'est à l'aide du mot-clé **HAVING** que l'on peut imposer des conditions sur un groupe. Par exemple, pour obtenir la liste des continents dont la population totale dépasse le milliard d'habitants on écrira :

```
SELECT e.continent, SUM(c.population)
FROM country c JOIN encompasses e ON c.code = e.country
GROUP BY e.continent
HAVING SUM(c.population) > 1000000000
```

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 [4].

a) Donner la liste ordonnée des dix langues parlées dans le plus de pays différents.

b) Quelles sont les langues parlées dans exactement six pays ? Et de quels pays s'agit-t-il ?

c) Quelles sont les langues parlées par moins de 30 000 personnes dans le monde ?

d) Quelles sont les cinq langues les plus parlées en Afrique ? On précisera pour chacune d'elles le nombre de personnes qui la parlent.

1.5 Sous-requêtes

Notons pour finir qu'il est possible d'imbriquer une requête dans une clause **SELECT**, ou (le plus souvent) au sein d'un filtre **WHERE** ou **HAVING**, la sous-requête devant simplement être encadrée par une paire de parenthèses. Supposons par exemple que l'on veuille déterminer les pays dont la densité de population est supérieure à la moyenne. Il y a clairement deux calculs à effectuer : d'abord le calcul de la moyenne, puis la détermination des pays dont la densité est supérieure à cette moyenne. Ceci peut se traduire en une seule requête SQL :

```
SELECT name FROM country
WHERE population / area > (SELECT AVG(population / area) FROM country)
```

Exercice 4 Dans la BDR mondial se trouve une table `economy` qui possède les attributs suivants : `Country` (le code du pays), `GDP` (le PIB, en millions de dollars), `agriculture` (la part de l'agriculture dans le PIB, en pourcentage), `Service` (la part des services dans le PIB), `Industry` (la part de l'industrie dans le PIB), `Inflation` (le taux d'inflation) et `Unemployment` (le taux de chômage).

a) Déterminer les pays majoritairement agricoles dont le taux de chômage est inférieur à la moyenne mondiale.

b) Déterminer pour chaque continent le pays au taux d'inflation le plus faible parmi les pays majoritairement industriels.

4. Attention, ces données ne sont pas disponibles pour tous les pays.

2. 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.
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 8 – 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 :

$$\boxed{\sigma_E(R)}$$

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 :

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

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 :

$$\boxed{R_1 \bowtie_E R_2}$$

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	E
a_1	b_1	c_1	e_1
a_2	b_2	c_2	e_2

Produit et division cartésiens
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 5 Donner un sens aux expressions de l'algèbre relationnelle suivantes :

a) $\pi_{Name}(\sigma_{Latitude>66}(\mathsf{city}) \cap \sigma_{Population>10000}(\mathsf{city}))$;

b) $\pi_{country.Name}(\sigma_{city.Latitude<0}(x) - \sigma_{city.Latitude<-23}(x))$ avec $x = \mathsf{country} \bowtie_{country.code=city.country} \mathsf{city}$.

Exercice 6 Montrer que si X est un ensemble d'attributs de R et A un attribut élément de X alors

$$\sigma_{A=a}(\pi_X(R)) = \pi_X(\sigma_{A=a}(R)).$$

Est-ce toujours vérifié si $A \notin X$?

Exercice 7 Et maintenant que vous êtes fans d'algèbre relationnelle, prouver le théorème relatif à la division cartésienne.

Annexe : interaction avec une base de données en Python

Le module permettant d'intégrer un SGBD 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 soit 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 :

```
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 'stop' pour terminer) :")
    if requete == 'stop':
        break
    try:
        cur.execute(requete)
    except:
        print('*** Requête SQL incorrecte ***')
    else:
        for enreg in cur:
            print(enreg)
    print()
cur.close()
base.close()
```