Bases de données

La *gestion de données* est l'une des tâches les plus courante en informatique. À haut niveau, la gestion de données recouvre un ensemble d'opérations variées, parmi lesquelles :

- l'acquisition des données (par exemple des capteurs météos, des informations de trafic ferroviaire, …) ;
- la validation et la mise en forme des données (par exemple, la normalisation de données issues de capteurs, la suppression de valeur impropres, …) ;
- le stockage et l'organisation des données sur un support physique (format des fichiers de stockage, organisation de ces derniers sur les disques durs, …) ;
- l'extraction et l'interrogation des données (pour des données météos, répondre à des questions comme « quelle température faisait-il le 2 janvier 2022 ?», « quel est le niveau de précipitations cumulé sur le mois de février 2022 ? ») ;
- le contrôle d'accès au données (garantir que seul le patient et son médecin ont accès au dossier médical du patient, et pas n'importe quel professionnel de santé).

Malgré la très gande variété des types de données, des domaines d'activité et types de traitements, il existe une approche systématique que l'on peut appliquer aussi bien à des données météorologiques qu'à des ensembles d'élèves et à leurs notes, ou aux produits d'un site de commmerce en ligne. Cette approche est celle du *modèle relationnel*. Nous présentons dans ce chapitre le modèle relationnel ainsi que son pendant pratique, le langage SQL permettant, entre autre, d'interroger des données relationnelles.

1 Le modèle relationnel

1.1 Relations

Le modèle relationnel a été introduit en 1969 par l'informaticien britanique Edgar Frank Codd. Pour illustrer les principales caractéristiques de ce modèle,nous prenons comme exemple celui d'un ensemble de films. Pour chaque film, nous voulons stocker comme information :

- son titre ;
- son année de sortie ;
- sa durée en minutes ;
- son genre (tel que « comédie », « drame », « documentaire », …) ;
- son pays de production ;
- la liste de son ou ses réalisateurs ;
- la liste de ses acteurs, avec le rôle de chaque acteur dans le film ;

Comme on le voit dans cette description, il y a différent niveau de données. Par exemple, un film est décrit par l'ensemble des propriété ci-dessus. À l'inverse, une année n'est qu'un entier (qui n'a aucune « sous-composante »). Dans le modèle relationnel, des objets complexes (comme ici nos films) sont appelés des *entités* ou des *enregistrements*. Le modèle relationnel permet de définir des ensembles d'entités appelé *relations*. Une relation est un ensemble de n-uplets, tous de même taille. Chaque composante d'un n-uplet est appelé *un attribut*. Les attributs sont caractérisés par leur *nom* et leur *domaine* (aussi appelé *type*). Le nom de la relation associé aux noms et domaines de tous les attributs est appelé le *schéma* de la relation. Le nombre d'attributs est appelé le *degré* ou l'*arité* de la relation. Le nombre d'éléments qu'elle contient appelé le *cardinal* de la relation.

$$\mathit{Film}(\mathit{titre} : \texttt{string},\ \mathit{année} : \texttt{int},\ \mathit{durée} : \texttt{int},\ \mathit{genre} : \texttt{string},\ \mathit{pays} : \texttt{string})$$

$$\begin{aligned}\mathit{Film} = \{&\\ &(\texttt{"La mélodie du bonheur"}, 1965, 174, \texttt{"musical"}, \texttt{"États-unis"}),\\ &(\texttt{"Crocodile Dundee 2"}, 1988, 112, \texttt{"aventure"}, \texttt{"Australie"}),\\ &(\texttt{"Le livre de la jungle"}, 1967, 78, \texttt{"animation"}, \texttt{"États-unis"}),\\ &(\texttt{"Casino Royale"}, 2006, 144, \texttt{"espionnage"}, \texttt{"Royaume-uni"}),\\ &(\texttt{"Léon"}, 1994, 110, \texttt{"drame"}, \texttt{"France"}),\\ &\vdots\\ \}&\end{aligned}$$

FIGURE 1 – La relation *Film* – Un schéma possible pour la relation *Film* ainsi que des valeurs pour les entités qu'elle contient.

Il est aussi courant d'utiliser une représentation tabulaire pour les relations :

Film

titre `string`	*année* `int`	*durée* `int`	*genre* `string`	*pays* `string`
`La mélodie du bonheur`	1965	174	`musical`	`États-unis`
`Crocodile Dundee 2`	1988	112	`aventure`	`Australie`
`Le livre de la jungle`	1967	78	`animation`	`États-unis`
`Casino Royale`	2006	144	`espionnage`	`Royaume-uni`
`Léon`	1994	110	`drame`	`France`
⋮				

Plus encore, il est aussi courant de parler de *table* plutôt que de relation, de *ligne* plutôt que d'entité et de *colonne* plutôt que d'attribut, la représentation tabulaire montrant naturellement la correspondance entre ces concepts.

Vocabulaire de bases de données

Bien qu'issu d'une autre branche de l'informatique les bases de données emploient des concepts présents dans d'autres domaine, en particulier les langages de programmation avec une même signification :

- les *enregistrements* sont les éléments d'une relation et donc des n-uplets dont les composantes sont nommées. C'est le même concept que celui des `struct` de C ou des types enregistrements d'OCaml.
- les domaines ou *types* des attributs représentent le même concept que les types de données des langages de programmation.

Le modèle relationnel tel que défini par Codd ne donne pas de liste précise des domaines des attributs. Il suppose juste l'existence d'un certain nombre de domaines *finis* dans lesquels les attributs prennent leurs valeurs et d'opérations prédéfinies sur ces domaines. Dans le cadre de ce chapitre, on supposera l'existence de trois domaines :

`int` : les entiers signés d'une taille fixe (non spécifiée)

`string` : les chaînes de caractères d'une taille maximale fixe (non spécifiée)

`float` : les nombres flottants d'une taille fixe (non spécifiée)

Nous détaillerons dans la suite les opérations définies sur ces domaines. Le modèle relationnel définit de plus une valeur spéciale, `NULL` valide pour tous les types, et qui représente une absence de valeur.
La relation de l'exemple 1 est une première étape dans notre modélisation relationnelle des films. Il manque cependant crucialement l'information des acteurs, de leur rôle dans chaque film et des réalisateurs. On pourrait être tenté de rajouter à notre relation *Film* des colonnes supplémentaires pour indiquer les acteurs et réalisateurs des films. Cette approche montre vite ses limites. En effet, le nombre d'acteurs d'un film étant variable (de même que le nombre de réalisateurs), on se retrouverait à rajouter des colonnes telles ques :

...	na_1 `string`	pa_1 `string`	ra_1 `string`	...	na_m `string`	pa_m `string`	ra_m `string`	nr_1 `string`	pr_1 `string`	...	nr_n `string`	pr_n `string`
⋮												

où les na_i, pa_i, ra_i sont les noms, prénoms et rôle des acteurs et les nr_j, pr_j sont les noms et prénoms des réalisateurs. Une telle modélisation est inélégante :

- il faut fixer *a priori* le nombre maximal de réalisateurs et d'acteurs d'un film ;
- si un film a plus de réalisateurs ou d'acteurs que cette limite, on ne peut les stocker ;
- si un film a moins d'acteurs ou de réalisateurs que la limite, il faut tout de même remplir toutes les colonnes, avec des valeurs par défaut.

Nous sommes ici dans la situation où l'on veut pouvoir *associer* un nombre arbitraire d'entités (par exemple des acteurs) à une entité donnée (un film). Avant de présenter la façon de modéliser de telles associations, nous allons modéliser des personnes (par le couple de leur nom et de leur prénom). Nous verrons ensuite comment associer des personnes à un film et comment indiquer qu'elles sont associée en qualité d'acteur ou de réalisateur.
Des personnes peuvent être modélisées par la table *Personne* ci-après :

Personne

nom string	*prénom* string
Plummer	Christophe
Andrews	Julie
Reno	Jean
Craig	Daniel
Green	Eva
Besson	Luc
⋮	

Cette modélisation pose cependant un problème que nous présentons maintenant.

1.2 Clé primaire

Notre but est de pouvoir associer des films à des personnes (en qualité d'acteur ou de réalisateur). Cependant un problème se pose avec notre table *Personne*. En effet, il est possible que deux personnes différentes aient le même nom et le même prénom. Cette situation est courante par exemple si on considère la liste des étudiants d'une université, des électeurs d'une grande ville, …Mais même sur un échantillon relativement réduit que représentent les acteurs, la situation se produit. En effet, l'ancien basketteur américain Michael Jordan (1963–) joue dans le film *Space Jam* (Warner Bros., 1996). Mais ce sportif est homonyme de l'acteur Michael Jordan (1987–) qui joue par exemple dans le film *Black Panther* (Walt Disney, 2018) [1]. Le nom et le prénom ne permettent donc pas d'identifier de façon unique une personne. Nous devons cependant différencier ces deux entités.
Le modèle relationnel définit la notion de *clé primaire*. Une clé primaire est un sous-ensemble des attributs d'une table qui identifie une ligne de façon unique. Pour la relation *Film*, l'attribut *titre* ne constitue pas une clé, car il est possible que plusieurs films portent le même titre. Par contre, on peut raisonnablement penser que le triplet $(titre, année, durée)$ identifie un film de façon unique (car il semble improbable que la même année, deux films différents ayant le même titre et la même durée à la minute près soient produits). Lorsque l'on donne le schéma d'une relation, la convention veut que l'on souligne l'ensemble des attributs constituant la clé primaire. On a donc pour la relation *Film* :

$$Film(\underline{titre : \texttt{string}, année : \texttt{int}, durée : \texttt{int}}, genre : \texttt{string}, pays : \texttt{string})$$

On ne peut cependant trouver un tel sous-ensemble d'attributs pour la relation *Personne*. C'est une violation du modèle relationnel qui impose que chaque entité (chaque ligne) d'une table soit identifiable de façon unique par une clé primaire. Une façon de faire dans ce cas, est d'ajouter à chaque entité un identifiant unique. Pour la relation *Personne* cela donne :

$$Personne(\underline{pid : \texttt{int}}, nom : \texttt{string}, prénom : \texttt{string})$$

Ici, nous avons artificiellement ajouté un attribut de type entier, unique qui identifie chaque ligne de la table. C'est le rôle du processus qui remplit la table d'associer un tel identifiant unique pour chaque entité. Il est considéré comme une bonne pratique d'utiliser un identifiant unique plutôt que de supposer que des données de la vie réelle vont être unique. Ainsi, on peut aussi redéfinir la relation *Film* comme ceci :

$$Film(\underline{fid : \texttt{int}}, titre : \texttt{string}, année : \texttt{int}, durée : \texttt{int}, genre : \texttt{string}, pays : \texttt{string})$$

Attention, une clé primaire *ne peut jamais valoir* `NULL`.
Nos entités étant maintenant identifiées de façon unique, nous pouvons nous attaquer au problème d'associer plusieurs entités entre elles. Nous allons pour cela procéder en deux étapes. En premier lieu, nous allons nous intéresser aux différentes façon d'associer des entités entre elles et présenter un formalisme synthétique permettant de représenter ces associations logiques. Dans un second temps, nous verrons comment implémenter concrètement ces associations avec les outils du modèle relationnel (des tables).

1.3 Entités et associations

Un système de gestion d'information se doit de pouvoir représenter des entités concrètes du domaine d'étude (films, personnes, livres, billets de trains, pays, …), mais doit aussi pouvoir représenter des aspects intangibles par exemple les liens logiques entre ces différentes entités. Dans notre exemple, le fait qu'une personne joue dans un film est un lien logique. Une façon systématique de présenter ces liens logiques est celle du *modèle entité-association* (en anglais *entity-relationship*, parfois improprement traduit en français comme « entité-relation »). Ce modèle a été introduit par l'informaticien Taïwanais Peter Chen (1947–) en 1976. Ce modèle, bien qu'utilisant des concepts proches du modèle relationnel présenté précédemment est distinct de ce dernier et s'intéresse simplement à décrire les entités d'un domaine d'étude et les liens logiques entre ces dernières. Comme dans le modèle relationnel, les entités sont des objets d'étude ayant une existence propre.

1. En pratique, le second se fait appeler Michael B. Jordan pour éviter la confusion avec le célèbre sportif.

Ces entités peuvent avoir un nombre fixe *attributs* (identiques aux attributs ou colonnes d'une table) qui les caractérisent et parmi ceux-ci, une clé primaire.
Les entités sont reliées entre elles par des associations. Les associations sont caractérisées par leur cardinalités :

1–1 : (*one to one*) une entité d'une relation A est reliée à au plus une entité d'une relation B ;

1–* : (aussi noté 1–N, *one to many*) une entité d'une relation A est relié à plusieurs entités d'une relation B ;

***–* :** (ausis noté N–N ou N–M, *many to many*) plusieurs entités d'une relation A sont reliées à plusieurs entité d'une relation B.

Nous allons maintenant complexifier notre exemple afin d'illustrer les différents types de cardinalités.

Association 1–1 Imaginons que l'on souhaite stocker dans notre base de données si un film a obtenu l'Oscar du meilleur film une année donnée. Si on considère, au niveau logique, un Oscar du meilleur film comme un objet d'étude à part entière, alors il y a une association 1–1 entre l'Oscar du meilleur film d'une année donnée et un film. En effet, pour une année donnée : l'Oscar n'est attribué qu'à *un film* et un film ne peut avoir (plus) *qu'un seul* Oscar du meilleur film (on ignore les autres types d'Oscars).

Association 1–* Considérons maintenant le pays de production d'un film. On pourrait considérer les pays comme un objet d'étude ayant une existence propre (par exemple si on souhaite aussi représenter le pays d'une personne dans notre base). Dans notre modèle, les films ont un seul pays de production (nous avons fait cette simplification) mais un même pays peut être associé à plusieurs films.

Association *–* Nous arrivons enfin aux exemples d'associations *many to many*. Si nous considérons les relations « réaliser un film » ou « jouer dans un film » ce sont deux associations *–*. En effet, une personne peut avoir réalisé plusieurs films et un films peut avoir été réalisé par plusieurs personnes. De même, une même personne peut avoir joué dans plusieurs films et plusieurs personnes ont pu jouer dans le même film. On remarque de plus que l'association « jouer dans un film » peut possède un attribut : le rôle que la personne a joué (par exemple : Mark Hamill joue le rôle de Luke Skywalker dans *La guerre des étoiles*).

Représentation graphique L'un des principaux attraits du modèle entité-association est de disposer d'un langage graphique simple permettant de représenter les principales caractéristiques d'une base de données. Dans ce modèle, les entités sont représentés par des rectangles. Les associations sont représentées par des losanges. Les propriétés sont représentées par des ellipses. La figure 2 donne le diagramme entité-association de notre base de données de films. Ce formalisme est très couramment utilisé dans les premières phases de conception d'une base de donnée, afin de lister les différents types d'objets d'études et les associations à représenter. Les cardinalités des associations sont indiquées du côté de l'entité correspondante. Le cas le plus intéressant est le cas asymétrique des associations 1–*. Dans la figure, le fait qu'un film n'ait qu'un seul pays de production mais qu'un pays puisse produire plusieurs films se voit dans l'annotation « 1 » du côté « Pays » de l'association « produit en » et « * » du côté « Film ».

1.4 Clé étrangère

Revenons maintenant à une modélisation relationnelle de nos données. Pour le cas d'une relation 1–1, la modélisation relationnelle semble toute trouvée. Par exemple, si l'on souhaite associer aux entités *Film* leur Oscar (s'ils en ont gagné un), on peut simplement étendre la table de la façon suivante :

$$\mathit{Film}(\underline{\mathit{fid} : \texttt{int}}, \mathit{titre} : \texttt{string}, \mathit{année} : \texttt{int}, \mathit{durée} : \texttt{int}, \mathit{genre} : \texttt{string}, \mathit{oscar} : \texttt{int})$$

Nous omettons volontairement le pays dans cette nouvelle modélisation de la table *Film*, ce dernier sera ajouté ultérieurement dans le cadre d'une relation 1–*. Comme on le voit, pour une relation 1–1, rien n'est plus simple que de mettre une nouvelle colonne du type approprié. On choisit ici un entier pour représenter l'année d'obtention de l'Oscar (qui est usuellement l'année suivant l'année de sortie du film, mais pas toujours). On pourra utiliser `NULL` pour indiquer que le film n'a pas remporté d'Oscar. Par exemple :

Film

fid int	*titre* string	*année* int	*durée* int	*genre* string	*oscar* int
42	La mélodie du bonheur	1965	174	musical	1966
38	Crocodile Dundee 2	1988	112	aventure	NULL
14	Le livre de la jungle	1967	78	animation	NULL
499	Casino Royale	2006	144	espionnage	NULL
302	Léon	1994	110	drame	NULL
771	The Artist	2011	100	comédie	2011
⋮					

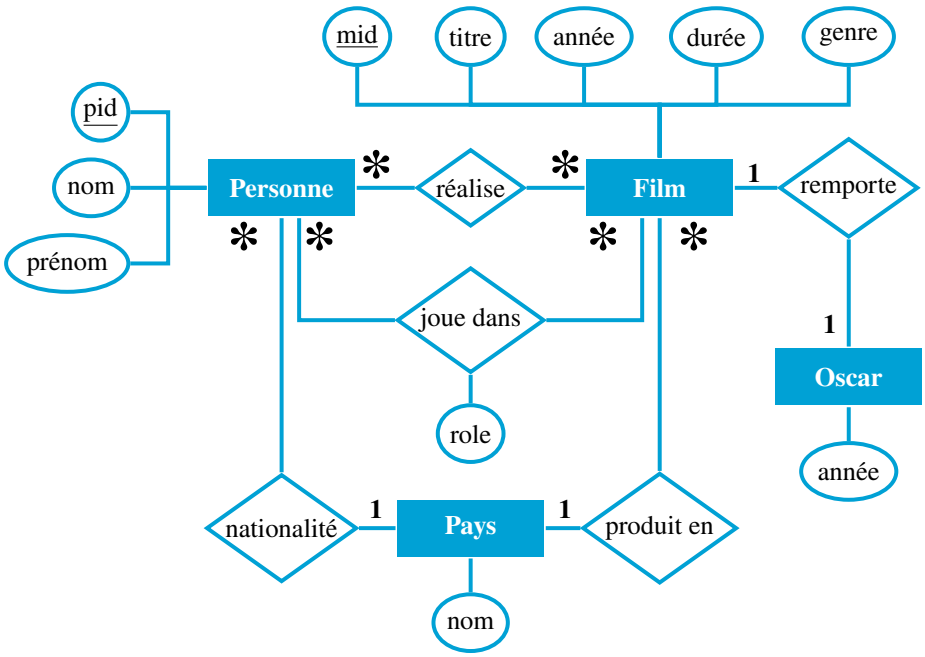


FIGURE 2 – Diagramme entité-association pour la base de données de films

Les identifiants des films (colonne *fid*) sont arbitraires, nous savons simplement qu'ils sont uniques. La situation est assez simple. Cependant, si l'entité à associer comporte plusieurs attributs, ont peut assez vite se retrouver avec des tables ayant énormément de colonnes. Qui plus est dans notre cas, la plupart du temps, ces colonnes contiendront `NULL`, car seul un film par an peut avoir l'Oscar du meilleur film. De plus les deux entités distinctes « Oscar » et « Film » de notre diagramme entité-association se retrouvent confonduent en une seule entité (au sens du modèle relationnel), dans une table *Film*.
Une solution alternative, plus flexible et plus élégante consiste à utiliser la notion de *clé étrangère*. Une clé étrangère, est un ensemble d'attributs d'une table qui forme une clé primaire dans une autre table. On utilise donc deux tables distinctes pour représenter les deux entités :

$$\textit{Film}(\underline{\textit{fid} : \texttt{int}}, \textit{titre} : \texttt{string}, \textit{année} : \texttt{int}, \textit{durée} : \texttt{int}, \textit{genre} : \texttt{string})$$
$$\textit{Oscar}(\underline{\textit{fid} : \texttt{int}}, \underline{\textit{annee} : \texttt{int}})$$

Dans les schémas ci-dessus, on a dénoté par un soulignement haché la colonne *fid* de la table *Oscar*. Cette dernière constitue une clé étrangère. Cette *contrainte* implique que la colonne *fid* de la table *Oscar* ne peut contenir *que* des entiers apparaissant comme clé primaire dans la table *Film*. En reprenant les valeurs de l'exemple précédant :

Film

fid int	*titre* string	*année* int	*durée* int	*genre* string
42	La mélodie du bonheur	1965	174	musical
38	Crocodile Dundee 2	1988	112	aventure
14	Le livre de la jungle	1967	78	animation
499	Casino Royale	2006	144	espionnage
302	Léon	1994	110	drame
771	The Artist	2011	100	comédie
⋮				

Oscar

fid int	*année* int
771	2011
42	1966
⋮	

Intuitivement, la colonne *fid* de la table *Oscar* peut être interprétée comme un « pointeur » vers l'unique ligne de la table *Film* ayant la même valeur dans la colonne *fid*. L'unicité est garantie par le fait que *fid* dans *Film* est une clé primaire. Cette modélisation relationnelle correspond bien à la représentation en diagramme entité-association.
L'utilisation d'une clé étrangère pour « pointer » entre deux tables peut naturellement s'étendre au cas des relations 1–*.

Si on considère maintenant le pays, on peut créer une table *Pays* comme ceci :

$$Pays(\underline{cid : \texttt{int}}, nom : \texttt{string})$$

Dans laquelle on pourrait retrouver les valeurs suivantes :

Pays

cid int	nom string
18	États-unis
19	France
25	Royaume-uni
33	Australie
⋮	

La table *Film* peut alors simplement modifiée de la sorte :

$$Film(\underline{fid : \texttt{int}}, titre : \texttt{string}, année : \texttt{int}, durée : \texttt{int}, genre : \texttt{string}, \underline{cid : \texttt{int}})$$

Film

fid int	titre string	année int	durée int	genre string	cid int
42	La mélodie du bonheur	1965	174	musical	18
38	Crocodile Dundee 2	1988	112	aventure	33
14	Le livre de la jungle	1967	78	animation	18
499	Casino Royale	2006	144	espionnage	25
302	Léon	1994	110	drame	19
771	The Artist	2011	100	comédie	19
⋮					

L'association 1–* apparaît alors clairement. La même ligne $(19, France)$ est associée à deux lignes dans la table *Film*.
Le cas des associations *–* est plus complexe. En effet, si on voulait appliquer la même technique il faudrait :

- soit pouvoir associer dans la table *Film*, pour un film donné, un nombre arbitraire de clés primaires de personnes (qui ont réalisé ce film) ;
- soit de façon symmétrique, pouvoir associer dans la table *Personne* un nombre arbitraire de clés primaires de films (que cette personne a réalisé).

Plus complexe encore, l'association « joue dans » devrait d'une façon ou d'une autre réussir à stocker en plus le nom du rôle joué par la personne dans un film.
La solution pour représenter des associations *–* est de passer par une table auxiliaire, appelée *table de liaison* (ou parfois table de jonction. Cette table de liaison matérialise *explicitement* l'association *-* que l'on souhaite représenter. Par exemple, dans le cadre des réalisateurs, il suffit de créer une table :

$$Réalisateur(\underline{fid : \texttt{int}}, \underline{\texttt{pid} : \texttt{int}})$$

dans laquelle *fid* est une clé étrangère vers *Film* et *pid* une clé étrangère vers *Personne*. Une ligne dans la table *Réalisateur* s'interprète donc naturellement comme : « la personne dont l'identifiant est *pid* réalise le film dont l'identifiant est *fid* ».

$$Film(\underline{fid : \texttt{int}}, titre : \texttt{string}, année : \texttt{int}, durée : \texttt{int}, genre : \texttt{string}, \underline{cid : \texttt{int}})$$

Film

fid int	titre string	année int	durée int	genre string	cid int
42	La mélodie du bonheur	1965	174	musical	18
38	Crocodile Dundee 2	1988	112	aventure	33
14	Le livre de la jungle	1967	78	animation	18
499	Casino Royale	2006	144	espionnage	25
302	Léon	1994	110	drame	19
771	The Artist	2011	100	comédie	19
⋮					

Personne

pid `int`	***nom*** `string`	***prénom*** `string`	***cid*** ***int***
17	Wise	Robert	18
18	Cornell	John	33
21	Reitherman	Wolfgang	18
42	Campbell	Martin	34
60	Besson	Luc	19
70	Hazanavicius	Michel	19
⋮			

Réalisateur

fid `int`	***pid*** `int`
42	17
38	18
771	70
14	21
499	42
302	60
⋮	

mes notes

2 Requêtes SQL

La création d'une BDR et sa manipulation requièrent l'usage d'un langage de requête. Normalisé depuis 1986, le *generic*, (Structured Query Language) est l'un de ces langages. Quatre rôles peuvent lui être attribués.

- C'est d'abord un *langage de manipulation de données* : recherche, ajout, modification, suppression de données d'une BDR.
- C'est un *langage de définition de données* : création et modification de l'organisation des données dans une BDR.
- C'est aussi un *langage de contrôle de transaction* : contrôle de l'évolution du contenu d'une BDR suite à des requêtes.
- Enfin, c'est un *langage de contrôle des données* en ce qu'il permet de limiter l'accès de certaines données à certains utilisateurs. Dans ce cas, l'aspect sécurité est mis en avant.

Assez peu d'instructions composent ce langage. Très expressives, elles permettent souvent une traduction aisée de requêtes humaines en requêtes informatiques. Dans la suite, nous abordons essentiellement l'aspect *langage de manipulation de données* de SQL.

2.1 Requêtes élémentaires

Les requêtes que nous formulons sont transmises à la BDR par l'intermédiaire de l'interface graphique de DB Browser for SQLite. L'onglet « Exécuter le SQL » propose une zone de saisie d'instructions SQL et une zone d'affichage des résultats d'exécution des requêtes.
L'exemple suivant présente une base de données des cafés parisiens à moins de 1 euro. Cette base ne comporte qu'une seule table, nommée `cafes`. Ses attributs, en tête de chaque colonne dans la zone d'affichage, sont : `rowid` (clé primaire), `maj` (date de mise à jour de l'information), `nom`, `adresse`, `arrondissement`, `comptoir` (prix du café au comptoir), `salle` (prix du café en salle), `terrasse` (prix du café en terrasse), `lat_lon` (latitude et longitude). Exprimons une requête qui sélectionne toutes les informations contenues dans la table en vue de leur manipulation ultérieure. En SQL, cette requête s'écrit :

```
SELECT *
FROM 'cafes-paris'
```

TABLE cafes | Recherche | Tout voir | Ajouter une nouvelle entrée | Duplicate | Éditer les entrées sélectionnées | Supprimer les entrées sélectionnées

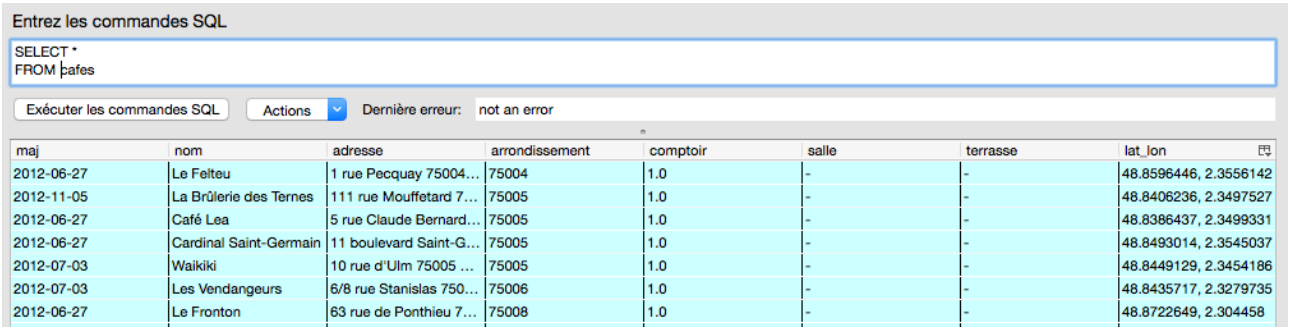
rowid	maj	nom	adresse	arrondissement	comptoir	salle	terrasse	lat_lon
1	2012-07-03	Chez Prune	36 rue Beaurepair…	75010	1.0	-	-	48.8715822, 2.36448
2	2012-07-03	La cantine de Zoé	136 rue du Faubo…	75010	1.0	-	-	48.8806562, 2.34995
3	2012-06-27	Le sully	13 rue du Faubour…	75010	1.0	-	-	48.8783988, 2.35703
4	2012-06-27	En attendant l'or	3 rue Faidherbe 7…	75011	1.0	-	-	48.8508323, 2.38408
5	2012-07-03	Extra old café	307 fg saint Antoin…	75011	1.0	-	-	48.8490041, 2.39264
6	2012-06-27	Bistrot Saint-Antoine	58 rue du Fbg Sai…	75012	1.0	-	-	48.8519278, 2.37323
7	2012-06-27	Le Killy Jen	28 bis boulevard …	75012	1.0	-	-	48.8465821, 2.38003
8	2012-09-10	L'Écir	59 Boulevard Sain…	75014	1.0	-	-	48.833524, 2.334252
9	2012-06-27	Au Vin Des Rues	21 rue Boulard 75…	75014	1.0	-	-	48.8331593, 2.32835
10	2012-10-22	Le café des amis	125 rue Blomet 75…	75015	1.0	1	-	48.8397592, 2.29690
11	2012-10-22	Le drapeau de la fi…	21 rue Copreaux 7…	75015	1.0	1	-	48.841628, 2.307005
12	2012-06-27	Le Parc Vaugirard	358 rue de Vaugir…	75015	1.0	-	-	48.8353891, 2.29242
13	2012-06-27	Café antoine	17 rue Jean de la …	75016	1.0	-	-	48.8516716, 2.27389
14	2012-10-22	Le BB (Bouchon d…	2 rue Lemercier 75…	75017	1.0	1	-	48.8889861, 2.31953
15	2012-11-05	Petits Freres des …	47 rue de Batignol…	75017	0.45	-	-	48.8856383, 2.31958
16	2012-10-22	Institut des Cultur…	19-23 rue Léon 75…	75018	1.0	1	-	48.8875966, 2.35347
17	2012-06-27	L'Olive	8 rue L'Olive 7501…	75018	1.0	-	-	48.8906792, 2.36126
18	2012-10-22	Populettes	86 bis rue Riquet …	75018	1.0	1	-	48.8900361, 2.36226
19	2012-06-27	Café Pistache	9 rue des petits ch…	75001	1.0	-	-	48.8662669, 2.33874
20	2012-11-05	Ragueneau	202 rue Saint Hon…	75001	1.0	-	-	48.8627103, 2.33770
21	2012-07-03	Au panini de la place	47 rue Belgrand 7…	75020	1.0	-	-	48.8646193, 2.40805
22	2012-06-27	La Cagnotte	13 Rue Jean-Bapti…	75020	1.0	-	-	48.8745379, 2.38775
23	2012-06-27	Melting Pot	3 rue de Lagny 75…	75020	1.0	-	-	48.8488784, 2.39996
24	2012-06-27	Au cerceau d'or	129 boulevard seb…	75002	1.0	-	-	48.8680642, 2.35333
25	2012-10-22	La Cordonnerie	142 Rue Saint-De…	75002	1.0	1	-	48.8626681, 2.34957
26	2012-09-10	Le Café frappé	95 rue Montmartre…	75002	1.0	-	-	48.8679433, 2.34360
27	2012-09-10	La Perle	78 rue vieille du te…	75003	1.0	-	-	48.8597859, 2.36056
28	2012-06-27	Chez Oscar	11/13 boulevard B…	75004	1.0	-	-	48.8549497, 2.36870
29	2012-09-10	Le Café Livres	10 rue Saint Marti…	75004	1.0	-	-	48.8577394, 2.34963
30	2012-07-03	L'avant comptoir	3 carrefour de l'O…	75006	1.0	-	-	48.8520876, 2.33867
31	2012-09-10	Cafe de grenelle	188 rue de Grenell…	75007	1.0	-	-	48.8576572, 2.30563
32	2012-06-27	Café Varenne	36 rue de Varenne…	75007	1.0	-	-	48.8541218, 2.32353

<< < 1 à 100 de 162 > >>

FIGURE 3 – Cafés de Paris à 1 euro

Les mots clés `SELECT ... FROM` réalisent l'interrogation de la table[2]. L'étoile placée après `SELECT` indique que les informations à sélectionner sont toutes celles contenues dans la table.
Le résultat est affiché dans la fenêtre destinée à cet effet.

2. Les quotes ' sont utilisées quand le nom d'un table ou d'un attribut comporte des caractères spéciaux, comme le tiret, ou des espaces.



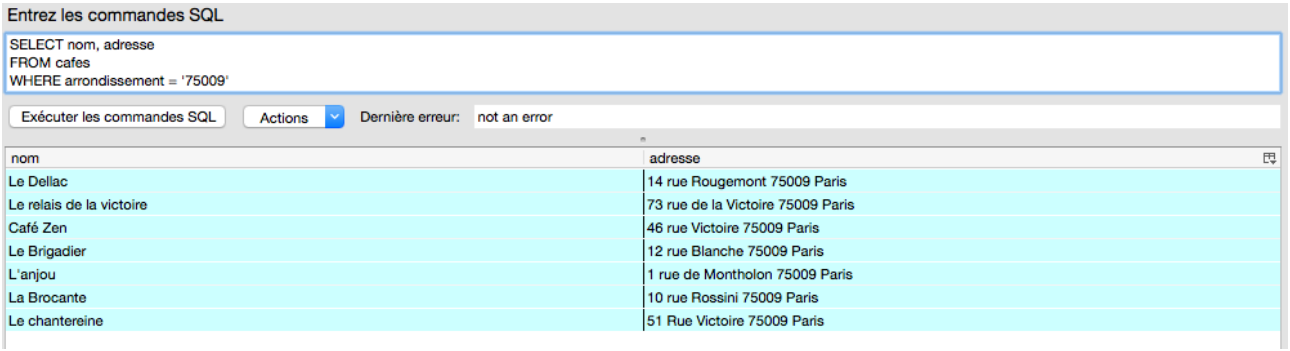
maj	nom	adresse	arrondissement	comptoir	salle	terrasse	lat_lon
2012-06-27	Le Felteu	1 rue Pecquay 75004...	75004	1.0	-	-	48.8596446, 2.3556142
2012-11-05	La Brûlerie des Ternes	111 rue Mouffetard 7...	75005	1.0	-	-	48.8406236, 2.3497527
2012-06-27	Café Lea	5 rue Claude Bernard...	75005	1.0	-	-	48.8386437, 2.3499331
2012-06-27	Cardinal Saint-Germain	11 boulevard Saint-G...	75005	1.0	-	-	48.8493014, 2.3545037
2012-07-03	Waikiki	10 rue d'Ulm 75005 ...	75005	1.0	-	-	48.8449129, 2.3454186
2012-07-03	Les Vendangeurs	6/8 rue Stanislas 750...	75006	1.0	-	-	48.8435717, 2.3279735
2012-06-27	Le Fronton	63 rue de Ponthieu 7...	75008	1.0	-	-	48.8722649, 2.304458

FIGURE 4 – Extrait des résultats affichés

Si à présent, nous souhaitons ne récupérer qu'une partie des informations contenues dans la table, comme les noms et les adresses des cafés du 9ème arrondissement, la requête devient :

```
SELECT nom, adresse
FROM cafes
WHERE arrondissement = '75009'
```

Le mot clé `WHERE` filtre les données qui répondent au critère de sélection. Par ailleurs, seuls les attributs `nom` et `adresse` sont à présent sélectionner. Cette requête renvoie donc les informations sous la forme suivante.



nom	adresse
Le Dellac	14 rue Rougemont 75009 Paris
Le relais de la victoire	73 rue de la Victoire 75009 Paris
Café Zen	46 rue Victoire 75009 Paris
Le Brigadier	12 rue Blanche 75009 Paris
L'anjou	1 rue de Montholon 75009 Paris
La Brocante	10 rue Rossini 75009 Paris
Le chantereine	51 Rue Victoire 75009 Paris

FIGURE 5 – Cafés du 9ème arrondissement

D'autres clauses permettent de formuler des requêtes plus élaborées. Les instructions données dans le tableau suivant permettant d'écrire quelques requêtes de base.

`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 6 – Requêtes de base SQL

Exercice 1

Qu'expriment les requêtes suivantes ?

```
SELECT nom, lat_lon
FROM 'cafes-paris'
WHERE arrondissement='75015'
ORDER BY nom
```

.....**réponse**.....

```
SELECT nom, lat_lon
FROM 'cafes-paris'
WHERE arrondissement='75001' OR arrondissement='75002'
ORDER BY nom
```

.....**réponse**.....

```
SELECT nom, adresse
FROM 'cafes-paris'
WHERE NOT(arrondissement='75001')
```

.....**réponse**.....

```
SELECT nom, adresse
FROM 'cafes-paris'
WHERE maj > '2013-01-01'
```

.....**réponse**.....

```
SELECT nom, adresse
FROM 'cafes-paris'
WHERE NOT(arrondissement < '75005') AND NOT(arrondissement > '75007')
ORDER BY arrondissement
```

.....**réponse**.....

2.2 Jointures

Jusqu'à présent, les données manipulées n'appartenaient qu'à une seule table. Cette situation présente peu d'intérêt et, surtout, ressemble beaucoup à ce qui est déjà fait par un tableur ! L'un des intérêts des BDR réside dans ce qu'elles comportent plusieurs tables. Des requêtes peuvent être formulées qui croisent des informations entre ces différentes tables. Les *jointures* assurent ce rôle.

Là encore, nous illustrons notre propos à l'aide de la BDR `bd_villes` (voir figure 7) qui comporte 3 tables :

- 1 table `communes` de toutes les communes de France,
- 1 table `departements` de tous les départements de France,
- 1 table `regions` de toutes les régions de France.

TABLE communes | Recherche | Tout voir | Ajouter une nouvelle entrée | Duplicate | Éditer les entrées sélectionnées | Supprimer les entrées sélectionnées

rowid	code_commune	id_region	id_departement	nom_commune	population
1	1001	82	1	L'Abergement-Clémenciat	784
2	1002	82	1	L'Abergement-de-Varey	221
3	1004	82	1	Ambérieu-en-Bugey	13835
4	1005	82	1	Ambérieux-en-Dombes	1616
5	1006	82	1	Ambléon	116
6	1007	82	1	Ambronay	2362
7	1008	82	1	Ambutrix	729
8	1009	82	1	Andert-et-Condon	340
9	1010	82	1	Anglefort	994
10	1011	82	1	Apremont	363
11	1012	82	1	Aranc	302
12	1013	82	1	Arandas	163
13	1014	82	1	Arbent	3476
14	1015	82	1	Arbignieu	480

(a) table `communes`

TABLE departements | Recherche | Tout voir | Ajouter une nouvelle entrée | Duplicate | Éditer les entrées sélectionnées | Supprimer les entrées sélectionnées

rowid	id_departement	id_region	nom_departement
1	971	1	Guadeloupe
2	972	2	Martinique
3	973	3	Guyane
4	974	4	La Réunion
5	976	6	Mayotte
6	75	11	Paris
7	95	11	Val-d'Oise
8	94	11	Val-de-Marne
9	93	11	Seine-Saint-Denis
10	92	11	Hauts-de-Seine
11	91	11	Essonne
12	78	11	Yvelines
13	77	11	Seine-et-Marne
14	51	21	Marne
15	10	21	Aube

(b) table `departements`

TABLE regions | Recherche | Tout voir | Ajouter une nouvelle entrée | Duplicate | Éditer les entrées sélectionnées | Supprimer les entrées sélectionnées

rowid	id_region	nom_region
1	1	Guadeloupe
2	2	Martinique
3	3	Guyane
4	4	La Réunion
5	6	Mayotte
6	11	Île-de-France
7	21	Champagne-Ardenne
8	22	Picardie
9	23	Haute-Normandie
10	24	Centre

(c) table `regions`

FIGURE 7 – Base de données `bd_villes` à trois tables

À présent, sélectionnons les noms et les populations des communes du département de la Gironde. Le numéro de département de la Gironde est 33. Une première façon de procéder est d'écrire la requête suivante.

```
SELECT nom_commune, population
FROM communes
WHERE id_departement = 33
ORDER BY nom_commune
```

Il serait cependant plus naturel de faire directement référence au département en invoquant son nom plutôt que son numéro. Cette information est présente dans la table `departements`. La requête doit donc aller chercher des informations dans les deux tables `communes` et `departements` sachant que seules les attributs `noms` et `pop` des communes de la Gironde doivent être extraits de la première table.
Un lien entre les deux tables est nécessaire pour réaliser cette extraction. Ce lien est établi par l'attribut `id` du numéro de département dans la table `departements` et par l'attribut `dep`, associé au département, de la table `communes`. La clause `JOIN ... ON` réalise l'opération de jointure entre les deux tables pour permettre l'extraction des seules données utiles.

```
SELECT communes.nom_commune, communes.population
FROM communes
JOIN departements
ON communes.id_departement = departements.id_departement
WHERE departements.nom_departement = 'Gironde'
ORDER BY communes.nom_commune
```

Pour cette requête, la syntaxe importe puisque les tables ont des attributs de même nom. Ainsi, `communes.nom` est l'attribut `nom` de la table `communes` alors que `departements.nom` est l'attribut `nom` de la table `departements`. Cette requête exploite donc les informations issues des deux tables `departements` et `communes`. En pratique, la jointure crée une table intermédiaire formée du produit cartésien des deux tables et applique ensuite la requête sur la nouvelle relation.
La notation précédente peut se mettre sous une forme équivalente en introduisant des alias pour les noms des tables. `c` est un alias pour la table `communes`. `d` est un alias pour la table `departements`.

```
SELECT c.nom_commune, c.population
FROM communes c
JOIN departements d
ON c.id_departement = d.id_departement
WHERE d.nom_departement = 'Gironde'
ORDER BY c.nom_commune
```

Exercice 2

Qu'exprime la requête suivante ?

```
SELECT c.nom_commune, d.nom_departement
FROM communes c
JOIN departements d
ON c.id_departement = d.id_departement
```

.....**réponse**.....

.....
b

D'autres syntaxes existent pour écrire des jointures. Certaines seront vues en travaux pratiques.

2.3 Fonctions d'agrégation

Les requêtes exprimées jusqu'à présent portaient sur l'ensemble des enregistrements d'une BDR. Parfois, il est commode de formuler des requêtes en regroupant des lignes suivant certains critères. Les *fonctions d'agrégation* assurent cette fonction de regroupement d'enregistrements. Parmi ces fonctions, on trouve des fonctions statistiques, des fonctions de concaténation, de décompte et de tris.
Les fonctions statistiques sont les suivantes.

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

Dans l'exemple suivant, la fonction `COUNT` calcule le nombre de lignes dans la table `communes`, c'est-à-dire le nombre de communes. Le résultat de la requête est le nombre total de communes de France.

```
SELECT COUNT(nom_commune)
FROM communes
```

Exercice 3

Que font les requêtes suivantes (la région 72 est la région Aquitaine) ?

```
SELECT COUNT(*)
FROM communes
WHERE id_region = '72'
```

.....**réponse**.....

```
SELECT SUM(population)
FROM communes
WHERE id_region = '72'
```

.....**réponse**.....

```
SELECT r.nom_region, SUM(c.population)
FROM communes c
JOIN regions r
ON c.id_region = r.id_region
GROUP BY c.id_region
ORDER BY r.nom_region
```

.....**réponse**.....

.....**mes notes**.....

3 Algèbre relationnelle

3.1 Pourquoi l'algèbre relationnelle ?

L'algorithmique permet de formuler un problème en termes informatiques indépendamment d'un langage de programmation. L'*algèbre relationnelle* définit un cadre formel dans lequel exprimer des requêtes sur 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.
Les notations adoptées pour désigner les opérations de l'algèbre relationnelle ne sont pas normalisées. Nous proposons ci-dessous des notations parmi les plus fréquemment rencontrées.

3.2 Opérations ensemblistes

Trois opérations ensemblistes de base peuvent être effectuées avec les relations.Ce sont les opérations d'*union*, d'*intersection* et de *différence*. Si R_1 et R_2 sont deux relations ayant les mêmes schémas relationnels :

- leur union $R_1 \cup R_2$ est une relation constituée d'enregistrements appartenant à l'une ou à l'autre des deux relations, sans répétition ;
- leur intersection $R_1 \cap R_2$ est une relation constituée d'enregistrements communs aux deux relations ;
- leur différence $R_1 - R_2$ est une relation constitué d'enregistrements de R_1 non présents dans R_2.

En SQL, ces opérations s'écrivent :

Union `table1` $\cup$ `table2`	Intersection `table1` $\cap$ `table2`	Différence `table1` $-$ `table2`

`SELECT * FROM table1`
`UNION`
`SELECT * FROM table2`
	`SELECT * FROM table1`
	`INTERSECT`
	`SELECT * FROM table2`
		`SELECT * FROM table1`
		`EXCEPT`
		`SELECT * FROM table2`

3.3 Opérations de l'algèbre relationnelle

3.3.1 Produit et division cartésiens

Nous avons déjà rencontré le *produit cartésien*. Ce dernier définit une relation R à partir de relations R_1 et R_2 de départ.

$$R = R_1 \times R_2$$

En SQL, il s'exprime simplement par la requête :

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

Seul, le produit cartésien ne produit pas nécessairement de résultat pertinent. Mais il constitue une opération de base pour d'autres opérations plus complexes. Ainsi, la *jointure* est un produit cartésien suivi d'une sélection.

$$R_1 \underset{E}{} R_2 = \sigma_E(R_1 \times R_2)$$

La *division cartésienne* est encore une opération qui produit une relation à partir de relations de départ. Partant des relations R_1 et R_2, on note :

$$R = R_1 \div R_2$$

Une telle opération n'a de sens que si R_2 est incluse dans R_1. De fait, elle est non commutative. La relation R obtenue contient tous les attributs de R_1 et aucun de ceux de R_2.
Cette opération n'est pas implémentée en SQL. Il est toutefois possible de la construire à l'aide d'instructions SQL bien que cette solution soit peu naturelle.

3.3.2 Projection

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

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

D'un point de vue pratique, la projection supprime des colonnes d'une relation.
En SQL, la projection se traduit par la requête :

```
SELECT DISTINCT A1, A2, ... FROM table
```

C'est la clause `DISTINCT` qui permet de ne retenir qu'une seule occurrence.
L'exemple suivant réalise la projection de la table des `communes` sur l'attribut `id_departement` et retourne une table ne comportant que les enregistrements associés à l'attribut, en éliminant les répétitions. Le résultat de cette requête est la liste des numéros des départements, sans doublons.

```
SELECT DISTINCT id_departement
FROM communes
```

Exercice 4

Que renvoie la requête précédente si on supprime la clause `DISTINCT` ?

.....**réponse**.....

.....

3.3.3 Sélection

La *sélection* (ou *restriction*) extrait les enregistrements d'une relation R qui satisfont une expression logique E. On la note :

$$\sigma_E(R)$$

D'un point de vue pratique, la sélection supprime des lignes d'une relation.
En SQL, la projection se traduit par la requête :

```
SELECT A1, A2, ... FROM table WHERE expression_logique
```

En fait, en SQL, c'est une double opération qui est réalisée par l'expression précédente : une sélection et une projection. Dans l'exemple suivant, la sélection $\sigma_{\texttt{pop > 250000}}(\texttt{communes})$ suivie de la projection $\pi_{\texttt{nom\_commune, population}}\left(\sigma_{\texttt{pop > 250000}}(\texttt{communes})\right)$ retourne la relation qui comporte tous les enregistrements des noms des communes dont les populations sont supérieures à 250 000.

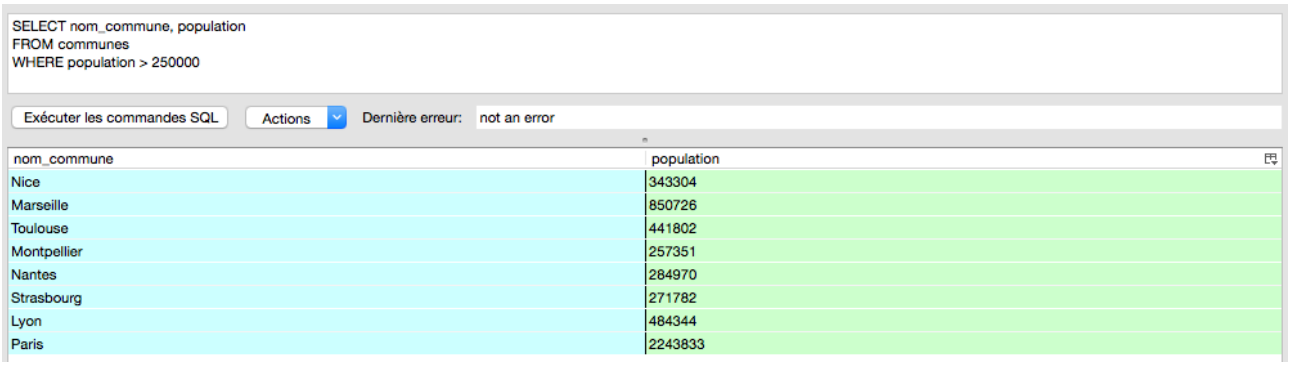


FIGURE 8 – Requête $\pi_{\texttt{nom\_commune, population}}\left(\sigma_{\texttt{pop > 250000}}(\texttt{communes})\right)$

3.3.4 Renommage

Le *renommage* permet la modification du nom d'un ou plusieurs attributs d'une relation. Les enregistrements de la relation obtenue après cette opération sont identiques à ceux de la relation de départ. Seul le schéma de la relation a changé. Ainsi, renommer l'attribut `A` en l'attribut `B` s'écrit :

$$\rho_{\texttt{A}\to\texttt{B}}(R)$$

En SQL, cette opération est réalisée par la requête :

```
ALTER TABLE table RENAME COLUMN ancien_nom_colonne TO nouveau_nom_colonne
```

Cette opération n'est malheureusement pas disponible en SQLITE. Seul le renommage d'une table ou l'ajout d'une colonne sont possibles.

3.3.5 Jointure

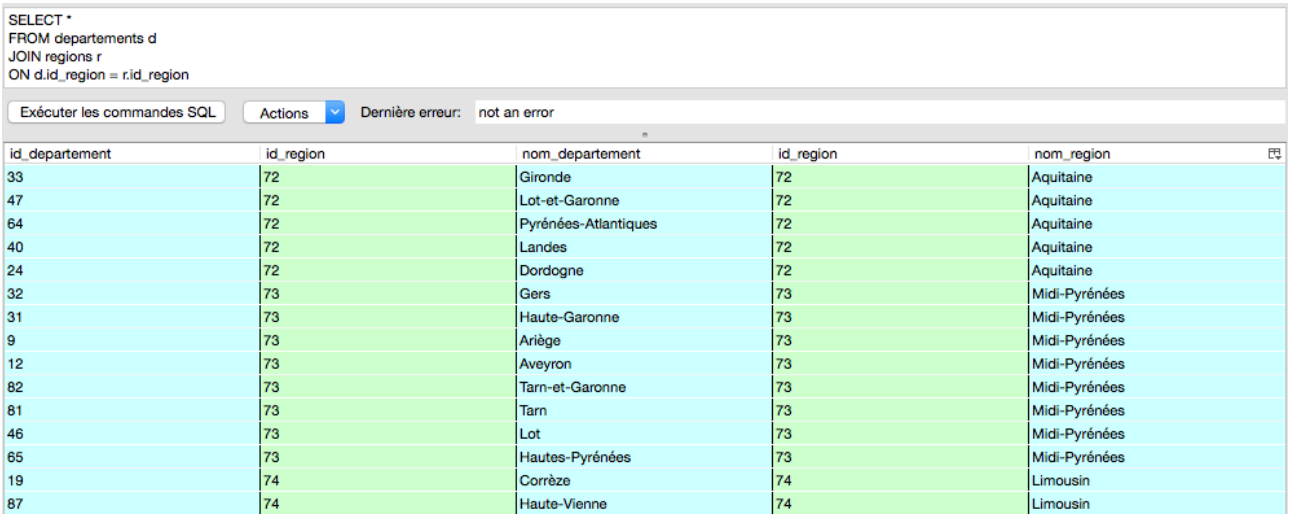
La *jointure* est une opération qui porte sur deux relations R_1 et R_2 et retourne une relation qui comporte les enregistrements des deux premières relations qui satisfont une contrainte logique E. On la note à l'aide du symbole .

$$R_1 \ _E \ R_2$$

Une première syntaxe SQL est la suivante :

```
SELECT <attributs> FROM table1 JOIN table2 ON expression_logique
```

Appliquée aux tables des régions et départements, l'exemple suivant réalise une jointure en identifiant tous les enregistrements pour lesquels `departements.region = region.id`.



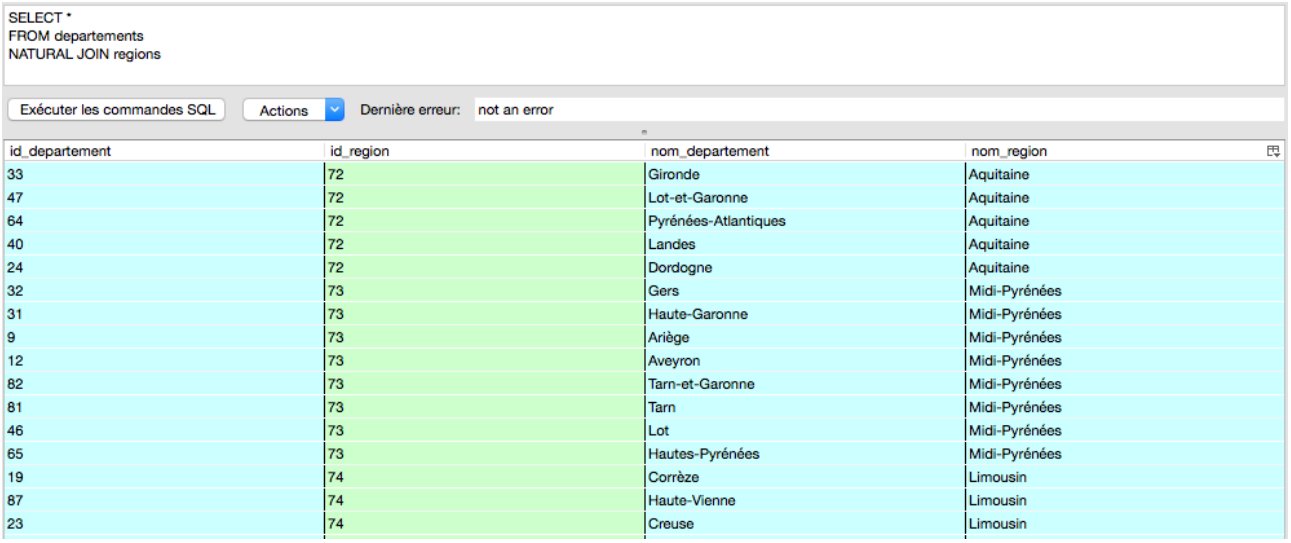
```
SELECT *
FROM departements d
JOIN regions r
ON d.id_region = r.id_region
```

Exécuter les commandes SQL | Actions | Dernière erreur: not an error

id_departement	id_region	nom_departement	id_region	nom_region
33	72	Gironde	72	Aquitaine
47	72	Lot-et-Garonne	72	Aquitaine
64	72	Pyrénées-Atlantiques	72	Aquitaine
40	72	Landes	72	Aquitaine
24	72	Dordogne	72	Aquitaine
32	73	Gers	73	Midi-Pyrénées
31	73	Haute-Garonne	73	Midi-Pyrénées
9	73	Ariège	73	Midi-Pyrénées
12	73	Aveyron	73	Midi-Pyrénées
82	73	Tarn-et-Garonne	73	Midi-Pyrénées
81	73	Tarn	73	Midi-Pyrénées
46	73	Lot	73	Midi-Pyrénées
65	73	Hautes-Pyrénées	73	Midi-Pyrénées
19	74	Corrèze	74	Limousin
87	74	Haute-Vienne	74	Limousin

FIGURE 9 – Jointure `JOIN ... ON`

La table obtenue présente un inconvénient. Les enregistrements associés à `d.id_region` et `r.id_region` ayant servi à établir le lien entre les tables sont dupliqués. Il serait intéressant de supprimer cette redondance. La clause `NATURAL JOIN ... ON` réalise cette opération.



```
SELECT *
FROM departements
NATURAL JOIN regions
```

Exécuter les commandes SQL | Actions | Dernière erreur: not an error

id_departement	id_region	nom_departement	nom_region
33	72	Gironde	Aquitaine
47	72	Lot-et-Garonne	Aquitaine
64	72	Pyrénées-Atlantiques	Aquitaine
40	72	Landes	Aquitaine
24	72	Dordogne	Aquitaine
32	73	Gers	Midi-Pyrénées
31	73	Haute-Garonne	Midi-Pyrénées
9	73	Ariège	Midi-Pyrénées
12	73	Aveyron	Midi-Pyrénées
82	73	Tarn-et-Garonne	Midi-Pyrénées
81	73	Tarn	Midi-Pyrénées
46	73	Lot	Midi-Pyrénées
65	73	Hautes-Pyrénées	Midi-Pyrénées
19	74	Corrèze	Limousin
87	74	Haute-Vienne	Limousin
23	74	Creuse	Limousin

FIGURE 10 – Jointure sans redondance `NATURAL JOIN ... ON`

3.3.6 Requêtes agrégats

Les traitements d'agrégation ne sont pas couverts par l'algèbre relationnelle. Toutefois, il est possible d'étendre l'algèbre pour représenter ces concepts. Nous utilisons la notation γ pour désigner les opérations associées à une requête d'agrégation.

SQL	Algèbre relationnelle
`SELECT MIN(*) FROM table`	$\gamma_{MIN(*)}(table)$
`SELECT MAX(*) FROM table`	$\gamma_{MAX(*)}(table)$
`SELECT AVG(*) FROM table`	$\gamma_{AVG(*)}(table)$
`SELECT COUNT(*) FROM table`	$\gamma_{COUNT(*)}(table)$
`SELECT SUM(*) FROM table`	$\gamma_{SUM(*)}(table)$
`SELECT * FROM table GROUP BY prop`	$\gamma_{GROUP\ BY(prop)}(table)$

Les TP donneront l'occasion d'appliquer ces notions.

mes notes