Informatique du Tronc commun

Chapitre n°1 Les bases de données

Introduction

Les bases de données permettent de stocker, organiser et analyser des données. On les utilise lorsque la gestion des informations stockées sous la forme d'un unique tableau devient trop fastidieuse.

La première partie de ce cours permettra de comprendre l'organisation des données dans une base de données.
La deuxième partie de ce cours concernera le langage SQL qui permet d'interroger une base de données relationnelle.

Objectifs :

— Acquérir le vocabulaire et les notions en lien avec le modèle relationnel.

— Aborder le modèle entités-associations, ses types d'associations, établir les liens avec les notions de clé primaire et clé étrangère.

— Connaître la structure et les mots clés d'une requête en langage SQL.

— Comprendre la notion de jointure et savoir l'utiliser.

— Connaître les fonctions d'agrégation.

— Être capable d'écrire des requêtes imbriquées.

Plan du cours

I Organisation des données 2

I Organisation d'une base de données 2
I.1 Intérêt des bases de données . . . 2
I.2 Base de données relationnelle . . . 2

II Modèle entités-Associations 3
II.1 Entités . . . 3
II.2 Associations . . . 3
II.3 Cardinalité . . . 4

III Application aux BDR 6
III.1 Vocabulaire . . . 6
III.2 Type de données . . . 7
III.3 Lien entre E/A et modèle relationnel . . . 7
III.3.a) Clé étrangère . . . 7
III.3.b) Association `1-*` . . . 7
III.3.c) Association `*-*` . . . 8
III.3.d) Association `1-1` . . . 9

II Le langage SQL 9

I Quelques règles sur l'écriture des requêtes 10

II Requêtes sur une seule table 11
II.1 Projection : `SELECT ... FROM ...` . . . 11
II.2 Sélection : `SELECT ... FROM ... WHERE` . . . 11
II.3 Organisation des résultats . . . 12
II.3.a) Suppression des doublons : `DISTINCT` 12
II.3.b) Calculs . . . 12
II.3.c) Renommage : `AS` . . . 12
II.3.d) Tri : `ORDER BY` . . . 13
II.4 Limiter avec `LIMIT` et `OFFSET` . . . 14
II.5 (Hors Programme) `LIKE` . . . 14

III Agrégation 15
III.1 Fonctions d'agrégation . . . 15
III.2 Groupement : Clause `GROUP BY` . . . 15
III.3 Filtrage des agrégats avec `HAVING` . . . 16

IV Requêtes sur plusieurs tables : Jointure 16
IV.1 Jointure . . . 16
IV.2 Auto-jointure . . . 17

V Opérateurs assemblistes 18
V.1 `UNION` . . . 18
V.2 `INTERSECT` . . . 18
V.3 `EXCEPT` . . . 18
V.4 Produit cartésien . . . 19

VI Requêtes imbriquées 19

Première partie

Organisation des données

Introduction

L'informatique s'est révélée d'une très grande efficacité pour manier des fichiers immenses ce qui a permis aux entreprises, aux administrations, une gestion rapide et efficace.

Pour ce faire, il a fallu concevoir des algorithmes très astucieux et d'une grande fiabilité.

Ceux qui trient les données sont fondamentaux car ils permettent l'accessibilité d'information dans les délais très brefs. Aussi, s'est-on attelé à concevoir ceux qui nécessitent le moins d'opérations. La scientifique américaine Betty Holberton en a été une des pionnières.

Le mathématicien et informaticien américain Edgar Franck Codd (1923-2003) a conçu une base mathématique pour traiter de la manière la plus efficiente les bases de données, appelée modèle relationnel. Elle consiste en une modélisation des relations entre les données et en la manière de les ordonner.

I Organisation d'une base de données

I.1 Intérêt des bases de données

Prenons un exemple, en tant que grand lecteur votre bibliothèque personnelle est fournie en livres, BD, en tous genres. Afin de vous y retrouver un peu, vous souhaitez stocker les informations concernant les livres et les auteurs, afin de pouvoir effectuer des recherches, des ajouts, ... facilement.

Votre première idée est de stocker toutes les informations dans un tableur, avec par exemple, les informations suivantes :

titre	édition	date	pages	langue	nom	prénom	naiss	décès	nationalité
Dans les bois éternels	J'ai lu	2006	478	Français	Vargas	Fred	1957		Française
Coule la scène	J'ai lu	2002		Français	Vargas	Fred	1957		Française
L'homme à l'envers	J'ai lu	1999		Français	Vargas	Fred	1957		Française
Pars vite et reviens tard	J'ai lu	2001		Français	Vargas	Fred	1957		Française
Sous les vents de Neptune	J'ai lu	2004		Français	Vargas	Fred	1957		Française
Un lieu incertain	J'ai lu	2008		Français	Vargas	Fred	1957		Française
L'armée furieuse	J'ai lu	2011		Français	Vargas	Fred	1957		Française
Temps glaciaire	J'ai lu	2015		Français	Vargas	Fred	1957		Française
Quand sort la recluse	Flammarion	2017		Français	Vargas	Fred	1957		Française
L'humanité en péril	J'ai lu	2020	349	Français	Vargas	Fred	1957		Française
Impact	Laffond	2020		Français	Norek	Olivier	1975		Française
Entre deux mondes	Pocket	2017		Français	Norek	Olivier	1975		Française
Surface	Pocket	2019		Français	Norek	Olivier	1975		Française
Le Lagon noir	Points	2017	384	Français	Indridason	Arnaldur	1961		Islandaise
Les roses de la nuit	Points	2020	284	Français	Indridason	Arnaldur	1961		Islandaise
Passages des ombres	Points	2018	365	Français	Indridason	Arnaldur	1961		Islandaise
La femme de l'ombre	Points	2018	383	Français	Indridason	Arnaldur	1961		Islandaise
Dans l'ombre	Points	2018	391	Français	Indridason	Arnaldur	1961		Islandaise
Les fantômes de Ryjkavik	Points	2021	364	Français	Indridason	Arnaldur	1961		Islandaise

Les informations concernant les auteurs pour lesquels vous avez plusieurs livres apparaissent un grand nombre de fois. Que ce passe-t-il si cet auteur change de nom ? de nationalité ? décède ? Vous êtes obligés de reprendre toutes les lignes le concernant pour corriger les informations.

De plus cela prend de la place en mémoire inutilement.

On souhaite donc organiser les données de sorte à économiser de la place mémoire (suppression des redondances), qu'elle soit facile maintenir et qu'elle permette des modifications sûres (c'est-à-dire sans perdre de données essentielles).

I.2 Base de données relationnelle

Définitions

- En informatique, une **base de données** (BD) est un **ensemble de données** qui ont été stockées sur un support informatique et qui ont été **organisées et structurées** de manière à ce qu'on puisse facilement consulter et modifier leur contenu.
- Une **base de données relationnelle (BDR)** est organisée en tables, encore appelées relations. Celles-ci peuvent être représentées graphiquement sous la forme de tableaux.

Pour concevoir une base de données une **modélisation** est au préalable nécessaire, après avoir analysé l'ensemble des informations à stocker et les utilisations que l'on voudra en faire. Cette modélisation va faire des liens entre les données, les contrôler (sécurité, validité), les traiter à l'aide d'un programme informatique, les rendre accessibles à un grand nombre d'utilisateurs, gérer des volumes de données de très grande taille. C'est pour cela qu'une BDR contient la plupart du temps **plusieurs tables qui sont reliées entre elles par des associations**.

Une bonne description théorique de cette organisation est le **modèle entité-association** développé dans les années 1970.

II Modèle entités-Associations

Le modèle Entité-Association a été conçu aux Etats-Unis en 1976 par le Taïwanais Peter Chen. Il est aujourd'hui à la base de la plupart des méthodes de modélisation des données.

Il est utile pour la modélisation, et consiste à décrire les informations représentant les données de manière simple. Il permet ensuite un passage facile au modèle relationnel.

II.1 Entités

Définitions : Entité

- Une **entité** est un élément physique, concret, ou abstrait. C'est un objet possédant des propriétés, appelées **attributs** servant à le décrire.
- L'ensemble des objets qui ont des caractéristiques communes, les attributs, est un **type d'entité** ou une **classe d'entité**.
- L'**identifiant** est un attribut particulier qui possède une valeur différente pour chaque entité et permet de distinguer deux entités.

  Chaque classe d'entité possède au moins un attribut identifiant.

On représente un type d'entité avec son nom et les noms des différents attributs dans un tableau.

Exemple 1.
Par exemple, considérons un type d'entité `Auteur` avec les attributs `ida`, `nom`, `prénom`, `naissance`, `décès`, `nationalité`. L'attribut `ida` est l'identifiant que l'on peut souligner.

Auteur
ida
Nom
Prénom
naissance
décès
nationalité

On peut considérer un deuxième type d'entité : le type `Livre`. L'attribut `idl` est l'identifiant que l'on peut souligner.

Livre
idl
titre
édition
date
pages
langue

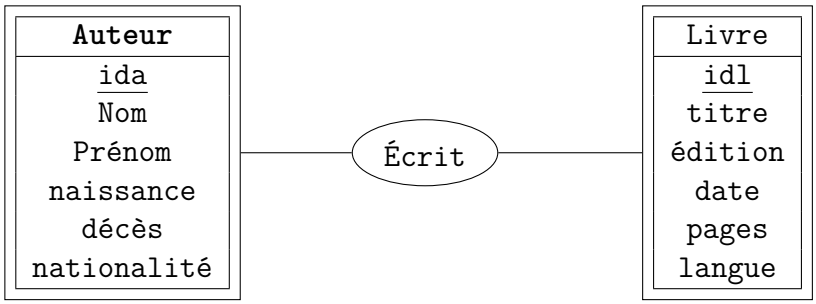
II.2 Associations

Définitions : Association

- Des liens ou relations existent entre les entités. Une **classe de relation** entre deux types d'entités contient toutes les relations du même type entre les entités appartenant à ces deux types.
- La relation ou le lien entre entités s'appelle une **association**.
- On parle donc de modèle **entité-association**.

Exemple 2. Modèle entité-association

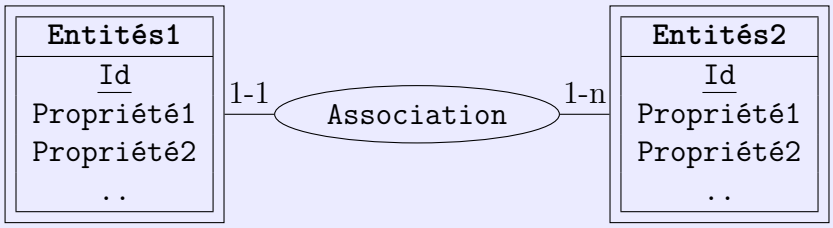
Le type d'association est décrit par un titre comme `Écrit` puisqu'un auteur écrit un livre ou `A été écrit` puisqu'un livre a été écrit par un auteur.



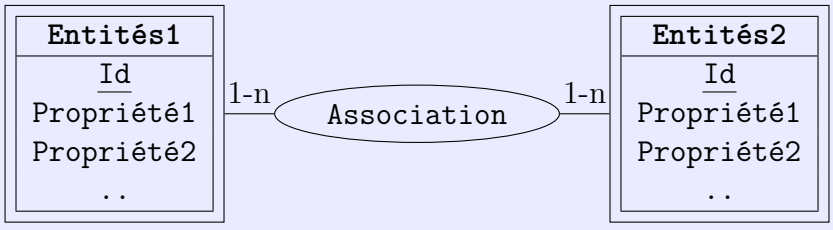
II.3 Cardinalité

Définitions : Cardinalité

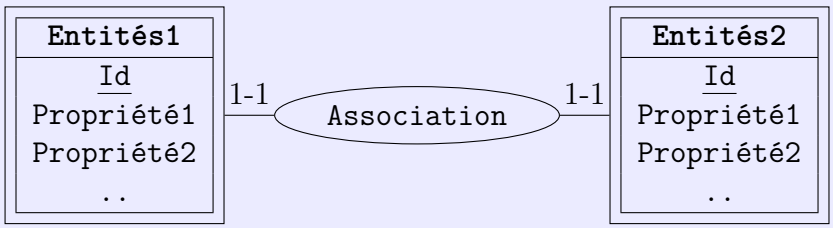
- Le lien entre une entité et une association est caractérisé par la **cardinalité**. Celle-ci est un couple minimum-maximum qui indique le nombre minimum de fois et le nombre maximum de fois qu'une entité peut participer à l'association.
- On distingue trois cardinalités entre une entité et une association :
  - Le couple est `1-1` si chaque entité participe exactement une fois à la relation.
  - Le couple est `1-n` si chaque entité participe au moins une fois à la relation.
  - Le couple est `0-n` si certaines entités ne participent pas à la relation.
- On peut présenter le modèle entité-association en parlant d'association `1-*`, `1-1` ou `*-*` :
  - **Association `1-*`** :



  - **Association `*-*`** :

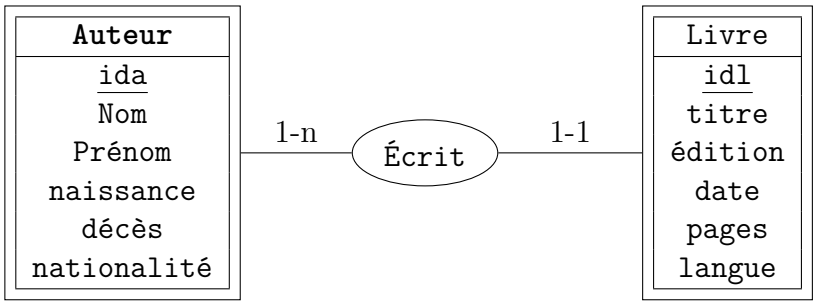


  - **Association `1-1`** :



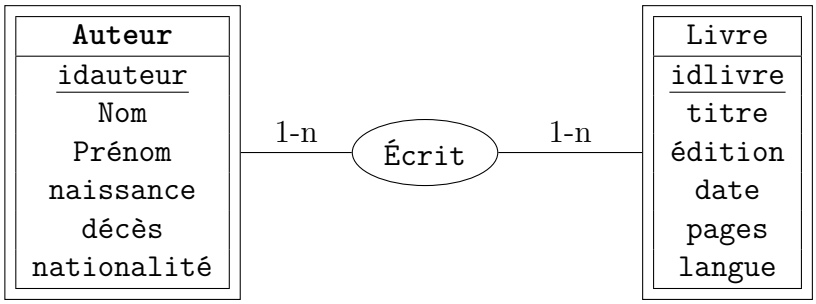
Exemple 3. Chaque auteur peut avoir écrit plusieurs livres, donc la cardinalité est 1-n entre `Auteur` et `Écrit`. Prenons, dans un premier temps le cas où chaque livre n'a été écrit que par un unique

auteur, la cardinalité est donc 1-1 entre `Livre` et `Écrit`.



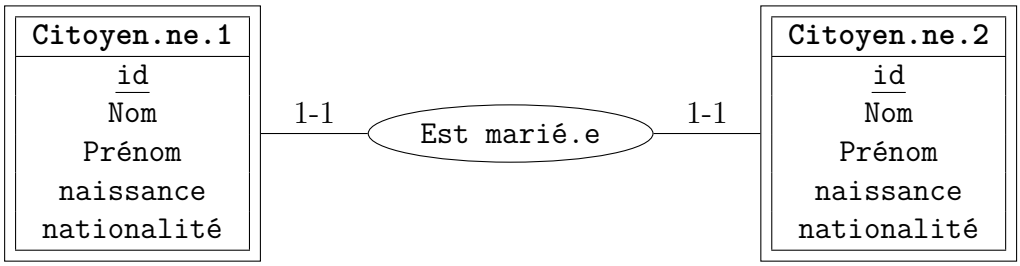
L'association est `1-*` entre `Livre` et `Auteur` : un auteur peut avoir écrit plusieurs livres mais un livre n'a été écrit que par un seul auteur.

Exemple 4. Dans votre bibliothèque certains livres ont été écrits par plusieurs auteurs. Chaque auteur a écrit au moins un livre, donc la cardinalité de l'association entre `Auteur` et `Écrit` est `1-n`. Un livre peut avoir été écrit par plusieurs auteurs, donc l'association entre `Livre` et `Écrit` est aussi `1-n`.



Il s'agit d'une association `*-*` entre `Auteur` et `Livre`.

Exemple 5. Prenons le cas d'une mairie qui souhaite gérer les mariages entre ses administré.e.s. Chaque administré.e ne peut être marié.e qu'à un.e autre administré.e.



Le.la citoyen.ne.1 peut être marié.e à un.e unique citoyen.ne.2, donc la cardinalité de l'association entre `Citoyen.ne.1` et `Est marié.e` est `1-1`. Le.la citoyen.ne.2 peut être marié.e à un.e unique citoyen.ne.1, donc la cardinalité de l'association entre `Citoyen.ne.2` et `Est marié.e` est `1-1`. Il s'agit d'une association `1-1` entre `Citoyen.ne.1` et `Citoyen.ne.2`.

III Application aux bases de données relationnelles

III.1 Vocabulaire

Définitions

relation (ou table) : est ce qu'on appelle tableau. Une base de données est composée d'autant de relations qu'il y a de tableaux.

attribut ou colonne : le nom d'une colonne d'une table.
Au sein d'une même table, il ne peut pas y avoir deux attributs ayant le même nom.

enregistrement ou lignes : c'est une ligne d'un tableau. Chaque ligne est unique.

domaine : c'est l'ensemble des valeurs que peut prendre un attribut. C'est le créateur de la base de données qui définit les domaines des différents attributs.

schéma de tables : précise le nom de la relation (=tableau) ainsi que la liste des attributs (=colonnes) avec leurs domaines. $S = \{A_1 : dom(A_1), A_2 : dom(A_2), ...\}$.

Définitions : Clé primaire

La **clé primaire** d'une table est l'ensemble minimal d'attributs qui permet d'identifier un tuple unique de la relation.

La clé primaire correspond à l'identifiant défini pour les types d'entité dans le modèle entité-association.

Exemple 6. Bibliothèque personnelle

— Dans le modèle relationnel, les deux types d'entités `Auteur` et `Livre` deviennent des tables `Auteur` et `Livre` avec leurs attributs respectifs et les identifiants `ida` et `idl` deviennent les clés primaires des deux tables.
Dans la relation `livres`, les attributs sont : `idl, titre, édition, date, pages, langue, idauteur`. Le domaine de l'attribut `pages` est l'ensemble des entiers pouvant être codé sur le nombre de bits alloués. Le domaine de l'attribut `langue` est l'ensemble des langues défini au préalable. Le domaine de l'attribut `idauteur` de la table des `livres` est l'ensemble des valeurs prises par l'attribut `ida` de la table des `auteurs`.
Le schéma de table `Livres` est : `Livres (idl, titre, édition, date, pages, langue, idauteur)`, que l'on peut représenter

Livres
idl
titre
édition
date
pages
langue
idauteur

— Le schéma de table `Auteurs` est : `Livres (ida, nom, prénom, naissance, décès, nationalité)`

III.2 Type de données

Les données présentes dans une base de données peuvent être de formats différents. Le créateur de la base de données définit pour chaque attribut le **type de données**. **Toutes les valeurs contenues dans une colonne sont de même type.**

- Les types numériques se décomposent en **type entier** ou en **type flottant**.
- Les textes seront de type **chaîne de caractères**.
- Concernant les **dates**, il existe différentes manières de les représenter. On se limitera à la représentation avec une **chaine de caractères** dans le format `aaaa-mm-jj` ou avec un **type numérique** (entier) dans le format `aaaammjj` qui permettent d'effectuer des comparaisons en utilisant l'**ordre lexicographique**.

> **Exemple 7.** Schéma relationnel de la table : `Auteurs (ida : INTEGER , nom : TEXT , prénom : TEXT , naissance : INTEGER , décès : INTEGER , nationalité : TEXT)`

III.3 Lien entre le modèle Entités-Associations et le modèle relationnel

Dans une base de données relationnelles, les associations (définies dans le modèle E/A) n'existent pas en tant que telles. On l'implémente en créant un attribut particulier qu'on appelle **clé étrangère** placée soit dans la table source soit dans la table but soit dans les deux, cela dépendra du type d'associations.

III.3.a) Clé étrangère

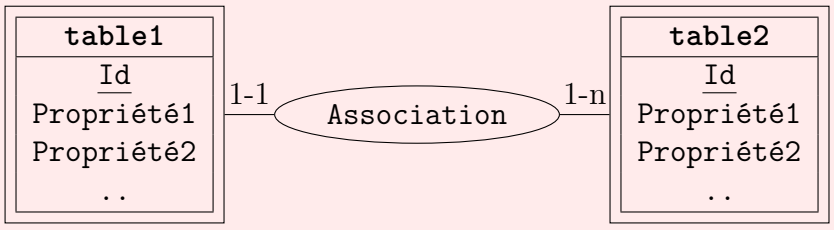
Définition : clé étrangère

Une **clé étrangère** de la table `T` est un attribut de cette table `T` qui est une clé primaire d'une autre table `T'`. Les valeurs permises prises par la clé étrangère sont uniquement celles prises par la clé primaire d'une autre relation.

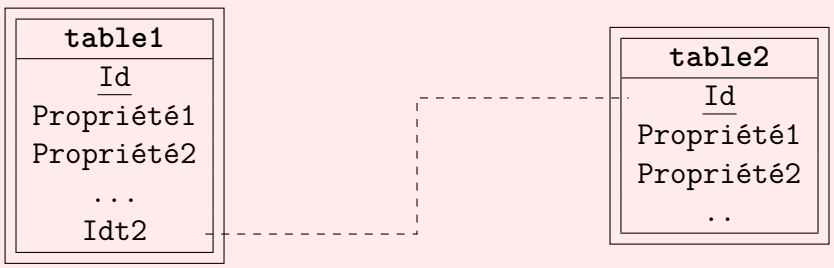
III.3.b) Association 1-*

À retenir

On considère l'association suivante pour laquelle les entités de la `table1` participent une seule fois à la relation, tandis que des entités de la `table2` participent plusieurs fois à la relation.

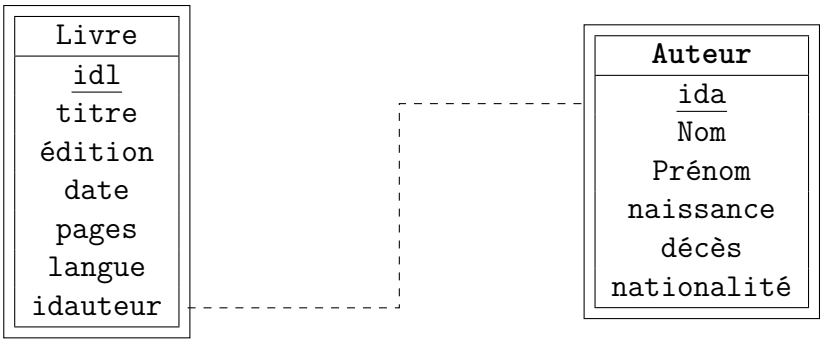


Une **association `1-*`** entre deux types d'entités `table1` et `table2` donne **deux tables** dans le modèle relationnel, les identifiants devenant les clés primaires de chacune des deux tables.
Cette association `1-*` conduit à ajouter un attribut à la table `table1` et à poser sur cet attribut une contrainte de **clé étrangère** en relation avec la clé primaire de la table `table2`.



Exemple 8. On reprend le cas où un livre a été écrit par un unique auteur, et un auteur peut avoir écrit plusieurs livres.

L'association `1-*` conduit à ajouter un attribut à la table `Livre`, `idauteur` et à poser sur cet attribut une contrainte de clé étrangère en relation avec la clé primaire `ida` de la table `Auteur`.

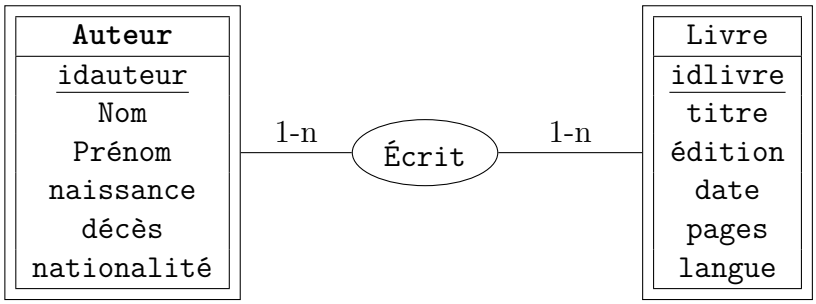


III.3.c) Association *-*

À retenir

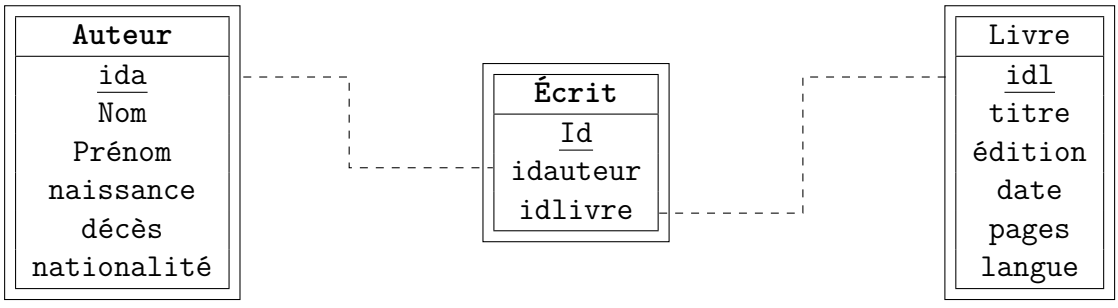
Une **association `*-*`** entre deux classes d'entité `table1` et `table2` est décomposée en deux associations `1-*`, **en ajoutant une classe d'entités**. Ainsi, la traduction de ce modèle entités-associations en un modèle relationnel nécessite **trois tables**.
La troisième table ajoutée contient deux clés étrangères chacune en relation avec la clé primaire de chaque table `table1` et `table2`.

Exemple 9. Dans votre bibliothèque certains livres ont été écrits par plusieurs auteurs



Il s'agit d'une association `*-*` entre `Auteur` et `Livre`.

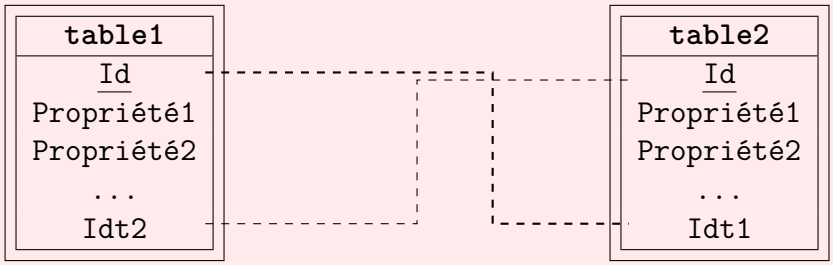
La traduction dans le modèle relationnel nécessite d'ajouter une table, `Écrit`, qui contient deux clés étrangères : `idauteur` en relation avec la clé primaire `ida` de la table `Auteur`, et `idlivre` en relation avec la clé primaire `idl` de la table `Livre`.



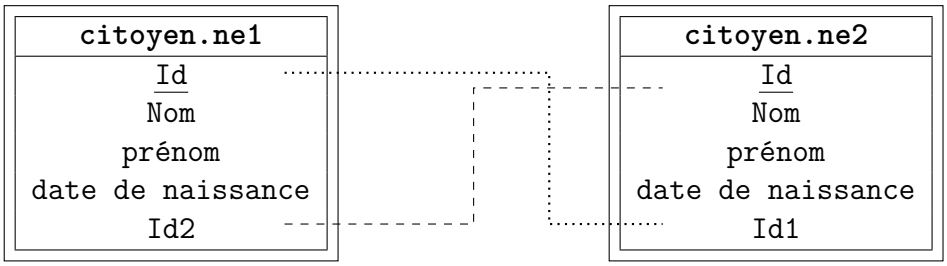
III.3.d) Association 1-1

À retenir

Une association `1-1` entre deux types d'entité `table1` et `table2` signifie que chaque entité de `table1` peut être en relation avec une et unique entité de `table2`.
Une association `1-1` entre deux entités est traduite par deux tables, et dans chaque table par la présence d'une clé étrangère en lien avec la clé primaire de l'autre table.



Exemple 10. Prenons par exemple le cas d'une mairie qui souhaite gérer les mariages entre ses administré.e.s. Chaque administré.e ne peut être marié.e qu'à un.e autre administré.e



Deuxième partie

Le langage SQL

Introduction

Un peu d'histoire

Les premières bases de données hiérarchiques apparaissent au début des années 1960 et Charles Bachmann conçoit en 1965 la première architecture d'un système de gestion de bases de données (SGBD). En 1968 Pick crée le premier système d'exploitation incluant un SGBD.

Dans les années 1970 Edgard Codd est l'initiateur de l'approche relationnelle. Sur une idée d'Edgard Codd, l'informaticien Donald Chamberlain développe une base de données relationnelle. Ceci a donné naissance dans les années 1970 au SQL, un langage spécifique pour interroger les bases de données, plusieurs fois amélioré et commercialisé dès 1979. À l'origine, on trouve le langage SEQUEL (Structured English Query Language).

Son efficacité modifie la plupart des activités comme les échanges boursiers, la gestion des salaires et impacte également notre vie courante comme l'utilisation d'une carte bancaire ou d'un smartphone.

Langage SQL

En 1986, intervient la première normalisation par l'ANSI avec l'appellation SQL-86 adoptée l'année suivante par l'ISO (Organisation internationale de standardisation).

Le SQL, pour Structured Query Language (langage de requêtes structurées) permet aux utilisateurs de communiquer avec une base de données. C'est **un langage déclaratif qui décrit ce que l'on souhaite obtenir à l'aide d'une requête**. La manière de parvenir au résultat n'est pas précisée, c'est le SGBD qui détermine la meilleure façon, en terme de complexité algorithmique, d'obtenir le résultat.

La suite du cours va être consacrée à la description des requêtes qui peuvent être effectuées sur une base de données.

BDD utilisée pour les exemples

Nous utiliserons dans la suite du cours la base de données `world` suivantes contenant un certain nombre de données sur les pays du monde

- `city (Id, Name, CountryCode, Population)`
  - Clé primaire : `Id`, de type entier, code de la ville.
  - `Name` : type chaîne de caractère.
  - `CountryCode` : type chaine de caractère, indique le code du pays où se situe la ville clé étrangère en relation avec la clé primaire `Code` de la table `country`.
  - `Population` : type entier.

  Une ligne de la table `city` est par exemple : (1, Kabul,AFG, 1780000).
- `country (Code,Name,Continent,SurfaceArea,IndepYear,Population,LifeExpectancy,GNP,Capital)`
  - Clé primaire : `Code`, de type chaine de caractère, code du pays.
  - `Name` : type chaîne de caractère.
  - `Continent` : type chaîne de caractères.
  - `SurfaceArea` : type flottant.
  - `IndepYear` : type entier, donne l'année d'indépendance.
  - `Population` : type entier.
  - `LifeExpectancy` : type flottant, renseigne sur l'espérance de vie.
  - `GNP` : type flottant, c'est le produit national brut.
  - `Capital` : type entier, est une clé étrangère en référence avec la clé `Id` de la table `city`.

  Une ligne de la table `country` est par exemple : (AFG, Afghanistan, Asia, 652090.00,1919,22720000,45.9,5976.00,1
- `countrylanguage (CountryCode, Language, IsOfficial, Percentage)` contient la liste des langues.
  - `CountryCode` : type chaine de caractères, indique le code du pays `Code` de la table `country` où cette langue est parlée.
  - `Language` : clé primaire, type chaine de caractères, c'est le nom de la langue.
  - `IsOfficial` : type chaine de caractères informe si la langue est officielle ('T') ou non ('F') dans le pays `CountryCode`.
  - `Percentage` : type flottant, renseigne le pourcentage de la population de `CountryCode` qui parle cette langue.

  Une ligne de la table `countrylanguage` est par exemple : (AFG,Pashto,T,52.4).

I Quelques règles sur l'écriture des requêtes

Les instructions en SQL ont une syntaxe assez simple et précise.

- **Le langage SQL n'est pas sensible à la casse** : un mot comme `SELECT` peut être écrit `SeLecT` sans provoquer d'erreur, de même pour le nom d'une table `table1` qui peut être écrit `TablE1` sans problème.
- Cependant **pour la lisibilité**, l'habitude est d'écrire **tous les mots du langage en lettres capitales** et **les noms de tables ou de colonnes en minuscules**.
- **Les espaces, les retours à la ligne et l'indentation n'ont aucune valeur syntaxique mais participent à la clarté de la requête.**
- **Seule l'écriture des valeurs doit respecter des règles** : un flottant s'écrit comme en Python avec un point pour séparateur, et une chaîne de caractères s'écrit entre des guillemets ou des apostrophes et doit respecter la casse `'une chaine'` est différente de `'Une Chaine'`.

II Requêtes sur une seule table

Dans cette partie, nous allons effectuer des demandes d'informations contenues dans une unique table.

II.1 Projection : SELECT ... FROM ...

À retenir : Projection

Les requêtes commencent toutes par l'instruction `SELECT`, suivie des noms des attributs souhaités. Ensuite, viennent les *clauses* (ou conditions), comme la clause `FROM` qui permet d'indiquer à partir de quelles tables sont extraites les informations.

- La requête la plus simple consiste à demander le contenu d'une colonne appartenant à une table :

```
SELECT attribut
FROM table
```

On obtient pour résultat un tableau à une colonne nommée `attribut`.

- On peut également obtenir le contenu de plusieurs colonnes en séparant les noms par des virgules :

```
SELECT attribut1, attribut2
FROM table
```

On obtient pour résultat un tableau à deux colonnes nommées `attribut1` et `attribut2`.

- On peut également sélectionner toutes les colonnes en utilisant le symbole `*`.

```
SELECT *
FROM table
```

II.2 Sélection : SELECT ... FROM ... WHERE

À retenir : Sélection

On peut ensuite ajouter des conditions que les résultats doivent vérifier avec la clause `WHERE`.

```
SELECT attributs séparés par une virgule (ou *)
FROM table
WHERE conditions
```

La condition (expression booléenne) qui suit le mot `WHERE` est formée :

- de noms de colonnes,
- d'opérateurs mathématiques : `+`, `-`, `*` et `/` ;
- d'opérateurs de comparaison : `=`, `<>` (pour $\neq$), `>`, `>=`, `<`, `<=` ;
- d'opérateurs logiques : `AND`, `OR` et `NOT`.

Exercice de cours A

Q1. Écrire la requête permettant d'obtenir la liste des pays.

Q2. Écrire la requête permettant d'obtenir la superficie et la population de la France.

Q3. Écrire la requête permettant d'obtenir les pays d'Asie ayant plus d'un milliard d'habitants.

II.3 Organisation des résultats

II.3.a) Suppression des doublons : DISTINCT

Dans les requêtes précédentes, il peut arriver que la requête affiche des doublons, on peut alors utiliser la clause `DISTINCT` qui impose une occurrence au plus de chaque entité.

À retenir

Pour ne sélectionner que des entrées distinctes, il faut ajouter `DISTINCT` juste après `SELECT` :

```
SELECT DISTINCT attribut
FROM table
WHERE conditions
```

Exercice de cours B

Q1. Écrire la requête permettant d'obtenir les langues parlées à travers le monde sans doublon.

Q2. Écrire la requête permettant d'obtenir la liste des continents.

II.3.b) Calculs

On peut obtenir des données résultant d'une ou plusieurs autres données et éventuellement d'un calcul. Ces nouvelles données sont intégrées dans un nouveau champ créé pour l'occasion. Les opérateurs utilisables sont : `+`, `-`, `*` et `/`

Exercice de cours C

Q1. Écrire la requête permettant d'obtenir la densité de de population dans chaque pays.

Q2. Écrire la requête permettant d'obtenir les pays avec leurs continents et le PNB par habitant où l'espérance de vie est supérieure ou égale à 80 ans.

II.3.c) Renommage : AS

Le renommage consiste à renommer un attribut ou une relation dans un but purement esthétique (pour ajouter une unité à l'attribut par ex.), ou nécessaire pour utiliser deux tables dont certains attributs portent le même nom alors qu'ils ont une signification différente (cf §IV.1 sur les jointures). Cela s'effectue grâce à l'opérateur `AS`.

À retenir

Pour renommer un attribut, cela se fait au début de l'instruction `SELECT`.

```
SELECT attribut1 AS A1
FROM table
```

Pour renommer une table, cela se fait juste après le nom de la table.

```
SELECT attribut
FROM table AS T
```

Exercice de cours D

Écrire la requête permettant d'obtenir la densité de population, renommée en `densité` exprimée en habitants par km^2.

II.3.d) Tri : ORDER BY

À retenir

Les résultats renvoyés par une commande `SELECT` ne sont pas classés, et sont présentés dans n'importe quel ordre. Le tri est une opération qui consiste à classer les enregistrements en fonction d'un ou de plusieurs critères.

- Pour classer les résultats (par ordre alphabétique, par ordre croissant), il faut utiliser `ORDER BY` placé à la fin de la requête, et suivi de l'attribut utilisé pour classer les résultats.

  Par défaut, les résultats sont classés dans l'ordre croissant.

```
SELECT attribut1
FROM table
WHERE conditions
ORDER BY attribut2
```

- Pour les classer par ordre décroissant il faut ajouter `DESC` après l'attribut en question :

```
SELECT attribut1
FROM table
WHERE conditions
ORDER BY attribut2 DESC
```

- On peut classer selon plusieurs critères :

```
SELECT attribut1
FROM table
WHERE conditions
ORDER BY attribut2 ASC, attribut3 DESC
```

  Dans ce cas, les enregistrements sont classés selon le premier critère. En cas d'égalité, ils sont classés selon le deuxième critère, ...

Exercice de cours E

Q1. Écrire la requête permettant d'obtenir la liste des noms des pays classés par ordre alphabétique.

Q2. Écrire la requête permettant d'obtenir les pays classés par année d'indépendance croissante puis par population décroissante.

II.4 Limiter avec LIMIT et OFFSET

Il est possible de limiter le nombre de lignes affichées, et à partir de laquelle. **Attention, les lignes sont numérotées à partir de 0.**

À retenir

- Pour limiter le nombre de lignes affichées à l'écran, on utilise `LIMIT n` qui affiche les `n` premières lignes.

```
SELECT attribut
FROM table
WHERE conditions
LIMIT n
```

  On obtient les lignes de 0 à `n-1` incluse.

- Pour afficher `n` lignes mais à partir d'une certaine ligne `m`, on utilise `OFFSET`, en complément de `LIMIT`.

```
SELECT attribut
FROM table
WHERE conditions
LIMIT n
OFFSET m
```

  On obtient `n` lignes de `m` à `m+n-1`.

  L'offset commence à 0, donc `OFFSET 0` signifie à partir de la première ligne.

L'ordre des données dans une table étant souvent quelconque (ordre d'enregistrement), `LIMIT` sera souvent utilisé avec la clause `ORDER BY` qui aura classé les données dans un ordre choisi par l'utilisateur.

Exercice de cours F

Q1. Écrire la requête permettant d'obtenir le pays d'Afrique le plus peuplé.

Q2. Écrire la requête permettant d'obtenir le deuxième pays d'Afrique le plus peuplé.

II.5 (Hors Programme) LIKE

Cette clause est HORS PROGRAMME, mais elle des questions portaient dessus dans l'épreuve de CCINP 2024... La syntaxe était rappelée, mais l'avoir déjà rencontrée sera moins déboussolant pour vous.

L'opérateur `LIKE` est utilisé pour comparer des chaînes de caractères dans la clause `WHERE` des requêtes SQL. Ce mot-clé permet d'effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Le caractère _ (underscore) représente n'importe quel caractère, mais un seul caractère uniquement alors que le caractère pourcentage % peut être remplacé par un nombre quelconque (et possiblement nul) de caractères.

Par exemple parmi une recherche dans les communes de France, `LIKE '_ff%f%'` ne renvoie que Offendorf alors que remplacer le `_` par un `%` (`LIKE '%ff%f%'`) renvoie Pfaffenhoffen et Staffelfelden en plus de Offendorf.

Exemple 11. • Écrire la requête permettant d'obtenir les pays commençant par un Z.

```
SELECT Name FROM country WHERE Name LIKE 'Z%'
```

• Écrire la requête permettant d'obtenir les pays ayant un z ni en première ni en dernière position.

```
SELECT Name FROM country WHERE Name LIKE '%z%'
```

• Écrire la requête permettant d'obtenir les pays ayant un r en deuxième position.

```
SELECT Name FROM country WHERE Name LIKE '_r%'
```

III Agrégation

III.1 Fonctions d'agrégation

Les fonctions précédentes portaient sur des n-uplets et permettaient de rechercher les éléments d'une table tels que telle condition était vérifiée. Le langage SQL permet d'exprimer des conditions portant sur des ensembles de n-uplets. Pour cela, on utilise des fonctions d'agrégations qui portent sur des sous-ensembles de données : c'est le sens de l'expression « données agrégées », c'est-à-dire vues dans leur ensemble à travers leurs nombres, leurs moyennes, minima, maxima, ...

À retenir : Fonctions d'agrégation

Toutes les fonctions d'agrégation s'utilisent à l'intérieur d'un `SELECT`.

- `MIN(attribut)` et `MAX(attribut)` : renvoie le minimum ou le maximum des éléments d'une colonne.
- `AVG(attribut)` : donne la moyenne des éléments (flottants ou entiers) d'une colonne.
- `SUM(attribut)` : fait la somme des éléments d'une colonne qui doivent être obligatoirement des flottants ou des entiers
- `COUNT(attribut)` : compte le nombre d'enregistrements. Le résultat de l'opération est affiché dans un nouveau champ.

  Si on veut compter les valeurs différentes prises par `attribut` il faut utiliser `DISTINCT` :

  `COUNT(DISTINCT attribut)`

Exemple 12. Premier exemple

Requête permettant d'obtenir le nombre de continents.

```
SELECT COUNT(DISTINCT continent)
FROM country
```

Exercice de cours G À vous de jouer !

Q1. Écrire la requête permettant d'obtenir la population mondiale.

Q2. Écrire la requête permettant d'obtenir la superficie moyenne des pays.

III.2 Groupement : Clause `GROUP BY`

On peut vouloir obtenir la population de chaque continent. Pour cela, il faut regrouper les pays par continent, puis calculer la somme des populations des pays de ces groupes. Le regroupement se fait à l'aide de la clause `GROUP BY`. Après ce regroupement, on utilisera les fonctions d'agrégations vues précédemment sur chaque groupe défini avec la clause `GROUP BY`.

À retenir : Clause `GROUP BY`

La clause `GROUP BY` permet de préciser les attributs qui sont utilisés pour les groupements. On peut alors appliquer une fonction d'agrégation sur chacun des groupes obtenus.

```
SELECT attribut1, attribut2, COUNT(*)
FROM table
GROUP BY attribut1
```

Les groupes sont formés de toutes les lignes qui ont la même valeur pour `attribut1`. On compte alors le nombre d'enregistrements dans la `table` pour chaque valeur de l'`attribut1`.

Exercice de cours H À vous de jouer !

Q1. Écrire la requête permettant d'obtenir le nombre de langues parlées dans chaque pays.

Q2. Écrire la requête permettant d'obtenir la population de chaque continent.

III.3 Filtrage des agrégats avec HAVING

La clause `WHERE` permet d'imposer une condition sur un attribut AVANT un groupement, et ne peut pas être utilisée après groupement ni sur une fonction d'agrégation. Pour cela on utilise la clause `HAVING`, qui est similaire à la clause `WHERE` pour filtrer les résultats, mais `HAVING` filtre APRÈS les groupements.

À retenir : Clause `HAVING`

La clause `HAVING` permet d'exprimer les conditions portant sur une fonction d'agrégation. Elle prend donc, après une clause `GROUP BY` la place qui serait celle de `WHERE` après `SELECT ... FROM ...` .

```
SELECT attribut1, attribut2
FROM table1
GROUP BY attribut2
HAVING condition
```

Exemple 13. Premiers exemples

Requête permettant d'obtenir la liste des codes des pays ayant au moins trois langues officielles

```
SELECT CountryCode
FROM countrylanguage
WHERE IsOfficial='T'
GROUP BY Language
HAVING COUNT(Language)>=3
```

Exercice de cours I À vous de jouer !

Q1. Écrire la requête permettant d'obtenir la liste des continents ayant moins de 15 pays.

Q2. Écrire la requête permettant d'obtenir la liste des continents ayant plus de un milliard d'habitants.

IV Requêtes sur plusieurs tables : Jointure

IV.1 Jointure

Jusqu'à présent nous avons effectué des requêtes sur une unique table. Pour connaître la capitale d'un pays, il est nécessaire de faire appel aux deux tables `city` et `country`. Pour cela, on effectue une **jointure** qui permet de joindre deux relations en une seule **suivant un critère donné.**

À retenir : Jointure

La jointure s'effectue en utilisant l'opérateur `JOIN` en précisant la condition de jointure après le `ON`.
`T1 JOIN T2 ON attribut1=attribut2` fait la jointure entre la table `T1` et la table `T2` selon la condition que la valeur de `attribut1` est égale à celle de `attribut2`.

Si deux attributs de deux tables différentes ont le même nom (ce qui est tout à fait possible), il y aura **ambiguïté pour la jointure**, il faudra alors **préciser devant l'attribut, la table** dont il fait partie : `table1.attribut`.

```
SELECT table1.attribut3,table1.attribut4
FROM table1
JOIN table2 ON  table1.attribut1=table2.attribut2
WHERE conditions
```

`attribut1` de `table1` et `attribut2` de `table2` font correspondre les deux tables. **La condition de jointure utilise les clés : la clé étrangère d'une des tables en relation avec la clé primaire de l'autre table.**

On peut avoir besoin d'avoir accès à trois (ou plus) tables, on fait alors plusieurs jointures :

```
SELECT table1.attribut3,table1.attribut4
FROM table1
JOIN table2
JOIN table3
ON table1.attribut1=table2.attribut2  AND table3.attribut3=table2.
   attribut4
WHERE conditions
```

L'ordre des tables avec `JOIN` et des critères de jointures avec `ON` n'a aucune importance.

REMARQUES

Le renommage permet de raccourcir et simplifier l'écriture d'une requête. On renomme les tables et on utilise le nouveau nom devant les champs dans la requête.

```
SELECT t1.attribut1,t2.attribut2
FROM table1 AS t1 JOIN table2 AS t2
ON  t1.id=t2.id
WHERE conditions
```

Exemple 14. Premier exemple
Requête permettant d'obtenir les langues parlées en France.

```
SELECT Language
FROM country
JOIN countrylanguage
ON Code=CountryCode
WHERE Name='France'
```

Exercice de cours J À vous de jouer !

Q1. Écrire la requête permettant d'obtenir la capitale du Portugal.

Q2. Écrire la requête permettant d'obtenir la langue officielle parlée au Vietnam.

Q3. Écrire la requête permettant d'obtenir la liste des langues officielles parlées dans la capitale Suisse (Berne, notée Bern dans la base de données).

IV.2 Auto-jointure

À retenir : Auto-jointure

L'auto-jointure est le fait de faire une jointure entre une table et elle-même. Cela nécessite de renommer la table sur laquelle on fait l'auto-jointure avec deux noms différents. Cela permet de trouver des couples de données ayant une valeur commune.

```
SELECT T1.attribut1,T2.attribut2
FROM table AS T1
JOIN table AS T2 ON T1.attribut3=T2.attribut3
WHERE conditions
```

Exercice de cours K À vous de jouer !

Écrire la requête permettant d'obtenir la liste des couples de pays ayant la même espérance de vie.

V Opérateurs assemblistes

V.1 UNION

À retenir

L'union est une opération assembliste, notée $R \cup S$, que l'on peut réaliser sur des relations à condition qu'elles aient la même structure, c'est-à-dire le même nombre d'attributs. Il s'agit de récupérer les lignes qui sont **dans l'une ou l'autre ou les deux relations** qui nous intéressent. On utilise pour cela l'opérateur `UNION`.

```
SELECT attribut1, attribut2 FROM table1 WHERE condition1
UNION
SELECT attribut1, attribut2 FROM table2 WHERE condition2
```

V.2 INTERSECT

À retenir

L'intersection est une opération assembliste, notée $R \cap S$, que l'on peut réaliser sur des relations à condition qu'elles aient la même structure, c'est-à-dire le même nombre d'attributs. Il s'agit de récupérer les lignes qui sont **dans les deux relations** qui nous intéressent.On utilise pour cela l'opérateur `INTERSECT`.

```
SELECT attribut1, attribut2 FROM table1 WHERE condition1
INTERSECT
SELECT attribut1, attribut2 FROM table2 WHERE condition2
```

Les opérateurs `OR` et `AND` peuvent également traduire une intersection et une union.

V.3 EXCEPT

À retenir

On peut réaliser la différence, notée $R - S$, de deux relations de même structure c'est-à-dire de même nombre d'attributs pour obtenir une relation contenant les lignes qui ne sont que dans le premier ensemble (R) et pas dans le second (S). On utilise pour cela l'opérateur `EXCEPT`.

```
SELECT attribut1, attribut2
FROM table1
WHERE condition1
EXCEPT
SELECT attribut1, attribut2
FROM table2
WHERE condition2
```

Exercice de cours L

Q1. Écrire la requête permettant d'obtenir la liste des noms des villes qui sont aussi des noms de pays.

Q2. Écrire la requête permettant d'obtenir la liste des villes d'Afrique qui ne sont pas des capitales avec le pays.

Q3. Écrire la requête permettant d'obtenir le pays d'Afrique qui a le plus grande superficie et celui d'Asie qui a la plus petite.

V.4 Produit cartésien

La produit cartésien est une opération ensembliste (comme union, intersection) entre deux tables, que l'on note `table1`×`table2`. En SQL, il s'obtient en écrivant les noms des deux tables séparés par une virgule.

```
SELECT attributs FROM table1, table2 WHERE conditions
```

Exemple 15. Par exemple, `country`×`city` s'obtient avec la requête :

```
SELECT * FROM country, city
```

Qui renvoie autant de lignes que le nombre de villes multiplié par le nombre de pays. Dans le tableau obtenu, un grand nombre de lignes n'a aucun sens. Pour obtenir un résultat ayant du sens, il faut ajouter une condition en utilisant les clés primaire et étrangère.

```
SELECT *
FROM country , city
WHERE Code=CountryCode
```

VI Requêtes imbriquées

À retenir : Requêtes imbriquées

On peut effectuer plusieurs requêtes successives à l'aide des mots clés `IN`, `NOT IN`, et des opérations `<`, `=`, ...

```
SELECT attribut1,attribut2,...
FROM table1
WHERE attribut1 IN (SELECT attribut1
                    FROM table1
                    WHERE conditions)
```

Les parenthèses sont nécessaires pour une requête correcte.

Exercice de cours M À vous de jouer !

Q1. Écrire la requête permettant d'obtenir les noms des villes qui sont aussi des noms de pays sans utiliser `INTERSECT`.

Q2. Écrire la requête permettant d'obtenir le pays qui a la plus petite superficie, sans utiliser `LIMIT`.

Bilan

À retenir : Format général d'une requête

L'ordre d'une requête est capital :

```
SELECT expressions (attributs, séparés par des virgules)
FROM Tables (séparées par JOIN)
ON conditions de jointure (si jointure avec JOIN)
WHERE conditions
GROUP BY attributs
HAVING conditions
ORDER BY attributs
LIMIT
OFFSET
```

REMARQUES

Pour information, l'ordre d'exécution d'une requête est le suivant, il est à distinguer de l'ordre d'écriture d'une requête :

1. Choix des tables
   - `FROM` : où trouver les informations
   - `JOIN` : si plusieurs tables
2. Choix des lignes
   - `ON` : sélection des lignes à traiter si jointure avec `JOIN`
   - `WHERE` : sélection des lignes à traiter selon une condition
   - `GROUP BY` : pour effectuer des regroupements sur les lignes à traiter
   - `HAVING` : pour ajouter des conditions sur les groupes obtenus
3. Choix des colonnes : `SELECT`
4. Traitement des colonnes
   - `MIN`, `MAX`, ..., `+`, `-`, ... : traiter les colonnes avec des fonctions ou des opérateurs ;
   - `DISTINCT` : supprimer des doublons ;
   - `ORDER BY` : ordonner les résultats ;
   - `LIMIT OFFSET` : ne conserver que certaines lignes de la réponse.

Il peut être intéressant d'avoir cet ordre d'exécution en tête notamment pour la gestion des renommages. En effet, si on renomme une table dans la clause `FROM`, il est disponible tout de suite et pour l'ensemble de l'instruction. Par contre, si on renomme dans l'instruction `SELECT`, le renommage n'est pas disponible pour le `WHERE` par exemple (puisque exécuté avant).