Requêtes sur plusieurs tables

I Jointures

L'intérêt principal des bases de données est de pouvoir manipuler plusieurs tables en même temps et de pouvoir en croiser les résultats. On supposera disposer des trois tables suivantes :

nom	num_at	symbole	colonne	ligne	bloc
Hydrogène	1	H	1	1	s
Hélium	2	He	18	1	s
Lithium	3	Li	1	2	s
Béryllium	4	Be	2(IIA)	2	s

La table `tableau`

nom	pays	symbole	date
Henry Cavendish	Grande-Bretagne	H	1766
Jules Janssen	Grande-Bretagne	He	1895
Joseph Norman Lockyer	Grande-Bretagne	He	1895
Johan August Arfwedson	Suède	Li	1817

La table `découverte`

symbole	rempl	masse_at	temp_fus	temp_ébu
H	1	1.00794		
He	2	4.002602		-268.93
Li	2|1	6.94100	180.5	1342
Be	2|2	9.012182	1287	2471

La table `grandeurs`

1. Produit cartésien

Si une table `T1` est un sous ensemble du produit $E_1 \times \cdots \times E_n$ et `T2` est un sous ensemble de $F_1 \times \cdots \times F_p$, le produit cartésien `T1` $\times$ `T2` sera un sous-ensemble de $E_1 \times \ldots E_n \times F_1 \times \cdots \times F_p$.
En SQL, un tel produit cartésien est obtenu par `SELECT * FROM T1 JOIN T2`. Si par exemple on fait le produit cartésien des tables `tableau` et `grandeurs`, on obtient la table suivante :

nom	num_at	symbole	colonne	ligne	bloc	symbole	rempl	masse_at	temp_fus	temp_ébu
Hydrogène	1	H	1	1	s	H	1	1.00794		
Hydrogène	1	H	1	1	s	He	2	4.002602		-268.93
Hydrogène	1	H	1	1	s	Li	2|1	6.94100	180.5	1342
⋮										
Hélium	2	He	18	1	s	H	1	1.00794		
Hélium	2	He	18	1	s	He	2	4.002602		-268.93
⋮										
Lithium	3	Li	1	2	s	H	1	1.00794		
⋮										

Ce produit cartésien contient $n_1 \times n_2$ enregistrements si n_1 et n_2 sont les nombres de lignes de `T1` et `T2` et un grand nombre d'entre elles n'ont pas vraiment d'intérêt (la deuxième mixe des données de l'hydrogène à celles de l'Hélium). Il est possible sur une telle table de sélectionner seulement les enregistrements intéressants avec `WHERE` mais il est plus efficace de faire cette sélection (qui sert en fait à créer une correspondance entre les tables) au moment de créer ce produit cartésien : c'est ce que l'on appelle faire une jointure.

2. Jointure

La jointure de deux tables se fait par l'utilisation de la commande `table1 JOIN table2 ON condition` ; la `condition` permet de faire la liaison entre les deux tables.

`SELECT` *attributs* `FROM` *table_1* `JOIN` *table_2* `ON` *condition*

Exemple(s) :

(*I.1*) `SELECT * FROM tableau AS t JOIN grandeurs AS g ON t.symbole = g.symbole` va créer une table donnant tous les renseignements des éléments chimiques présents dans la table `tableau` et `grandeurs` (en les faisant correspondre par leur symbole). Comme l'attribut `symbole` est présent dans les deux tables, il faut préciser `tableau.symbole` et `grandeurs.symbole` (ou utiliser un alias avec `AS`, qui peut aussi être omis).
Cette jointure produit la table suivante :

nom	num_at	symbole	colonne	ligne	bloc	symbole	rempl	masse_at	temp_fus	temp_ébu
Hydrogène	1	H	1	1	s	H	1	1.00794		
Hélium	2	He	18	1	s	He	2	4.002602		-268.93
Lithium	3	Li	1	2	s	Li	2|1	6.94100	180.5	1342

(*I.2*) On peut ensuite faire une sélection (avec `WHERE`) sur cette nouvelle table, ou utiliser toutes les fonctions vues dans le premier cours :

```
SELECT DISTINCT pays FROM découverte d JOIN tableau t
ON d.symbole = t.symbole WHERE ligne = 2
```

renvoie les pays, sans répétition, où ont été découverts les éléments de la ligne 2. Comme `ligne` et `pays` ne sont pas des attributs des deux tables, le préfixe (avec ou sans alias) est inutile.

La condition de jointure peut être une condition composée avec les opérateurs `AND`, `OR` et `NOT` et porter sur plusieurs attributs.

On peut ensuite réaliser des jointures avec plus de deux tables :

`SELECT` *attributs* `FROM` *table_1* `JOIN` *table_2* ...`JOIN` *table_n* `ON` *conditions*

Remarque(s) :

(*I.3*) La première chose à faire quand on doit faire une jointure est de déterminer quels sont les tables nécessaires ; ce sont les tables qui font apparaître les attributs nécessaires (ceux que l'on veut récupérer et ceux sur lesquels portent les conditions de sélection).

Exemple(s) :

(*I.4*) Créer une table contenant la liste des symboles des éléments, leur numéro atomique et le remplissage de leurs couches : le symbol est présent dans toutes les tables, le numéro atomique seulement dans `tableau` et le remplissage dans `grandeurs`, il faut donc joindre ces deux tables :

```
SELECT t.symbole,numéro_atomique,remplissage FROM tableau t JOIN
    grandeurs g
ON t.symbole = g.symbole
```

(*I.5*) Créer une table contenant le nombre d'élément, le plus grand et le plus petit numéro atomique, la moyenne des masses atomiques de chaque ligne : les mêmes tables sont nécessaires mais il faut en plus faire un groupement par ligne (pour appliquer les fonctions d'agrégation sur chaque ligne) :

```
SELECT COUNT(*),MAX(numéro_atomique),MIN(numéro_atomique),
AVG(masse_atomique) FROM tableau t JOIN grandeurs g
ON t.symbole = g.symbole GROUP BY ligne
```

(I.6) Déterminer les noms des personnes qui ont découvert les éléments et ces éléments dont le numéro atomique est < 50 et qui ne sont pas solides à une température nulle : les trois tables sont nécessaires

```
SELECT d.nom,d.symbole FROM découverte d JOIN tableau t JOIN
    grandeurs g
ON d.symbole=g.symbole AND d.symbole = t.symbole
WHERE numéro_atomique < 50 AND temp_fusion < 0
```

Le préfixe `d` devant `nom` est nécessaire pour distinguer l'attribut `nom` de la table `découverte` (le nom de la personne qui a découvert l'élément) de l'attribut `nom` de la table `tableau` (le nom de l'élément). Même si la jointure s'est faite sur le symbole, le préfixe `d.symbole` est indispensable pour que la commande soit exécutable (même si `g.symbole` produirait le même effet par exemple).

3. Autojointure

Il est aussi possible de faire une jointure entre une table et elle même ; la syntaxe est la même mais les préfixes sont alors indispensables pour préciser dans quel exemplaire de la table doit être pris l'attribut correspondant.

Exemple(s) :

(I.7) Déterminer les couples de symboles (distincts) que l'on peut former avec les éléments de la $3^{\text{ème}}$ colonne : il faut ici pourvoir accéder deux fois à l'attribut symbole donc on va créer une table avec ces deux attributs, en les faisant correspondre (condition de jointure) par leur numéro de colonne

```
SELECT t1.symbole,t2.symbole FROM tableau t1 JOIN tableau t2
ON t1.colonne=t2.colonne
WHERE t1.colonne=3 AND t1.symbole!=t2.symbole
```

(I.8) Déterminer les symboles des éléments chimiques découverts par un chercheur de la même nationalité que celui qui a découvert le Radon : il faut dont déterminer le pays du chercheur qui a découvert le Radon (Marie Curie donc la France) et chercher tous les éléments découverts par des chercheurs français (mais on ne peut bien sûr pas diviser cette tache en 2). On va devoir constituer une table contenant deux attributs symbole, le premier pour repérer le Radon, le second pour trouver les autres éléments avec lequel il « correspond », la correspondance (condition de jointure) se faisant sur le pays.

```
SELECT DISTINCT d2.symbole FROM découverte d1 JOIN découverte d2
ON d1.pays=d2.pays WHERE d1.symbole='Ra'
```

Il est alors possible de faire ensuite une autre jointure avec d'autres tables (ou bien une « autojointure triple »)

4. Opérations ensemblistes

Le résultat d'un requête (utilisation de `SELECT`) crée en fait une nouvelle table que l'on peut ensuite réutiliser dans une autre requête.
Les commandes `UNION`, `INTERSECT` et `EXCEPT` permettent d'exécuter les opérations de réunion, d'intersection ou de différence sur les tables que l'on crée.

```
SELECT * FROM t1 UNION SELECT * FROM t2      # enregistrements présents dans t1 ou
    t2 (sans répétition)

SELECT * FROM t1 INTERSECT SELECT * FROM t2      # enregistrements présents dans t1
    et t2

SELECT * FROM t1 EXCEPT SELECT * FROM t2      # enregistrements présents dans t1
    mais pas dans t2
```

Pour pouvoir utiliser ces opération, les tables doivent avoir le même schéma relationnel (mêmes attributs)

Exemple(s) :

(I.9) `SELECT * FROM tableau WHERE ligne = 2 UNION SELECT * FROM tableau WHERE symbole < 'D'` renvoie la sous-table de `tableau` des éléments qui sont sur la deuxième ligne ou dont le symbole commence par une lettre avant D.

(I.10) `SELECT * FROM tableau WHERE ligne = 2 EXCEPT SELECT * FROM tableau WHERE colonne =1` renvoie la sous-table des éléments de la ligne 2 sauf celui de la colonne 1.

Remarque(s) :

(I.11) Il est possible d'utiliser le résultat d'une requête à l'intérieur d'une autre requête (sous-requête) : reprenons l'exemple I.8 :

- on peut commencer par déterminer le pays des chercheurs ayant découvert le Radon

```
SELECT pays FROM découverte WHERE symbole = 'Ra'
```

qui renvoie `'France'` (donc un seul pays)

- puis chercher les éléments découverts par des chercheurs français

```
SELECT DISTINCT symbole FROM découverte WHERE pays = 'France'
```

- mais on peut dans cette deuxième requête remplacer `'France'` par le résultat de la première requête :

```
SELECT DISTINCT d2.symbole FROM découverte AS d2
WHERE d2.pays = (SELECT d1.pays FROM découverte AS d1 WHERE
    d1.symbole = 'Ra')
```

Une telle sous-requête doit être placée entre parenthèses et peut être utilisée dans une autre requête à la place d'une table (après `FROM`) ou dans une condition logique (après `WHERE`).

Dans le cas où cette sous-requête renvoie plusieurs résultats, on peut tester l'appartenance à cet ensemble de résultats avec `IN` (à la place du `=`) mais cette commande ne figure pas au programme.

II Modèle entités-associations

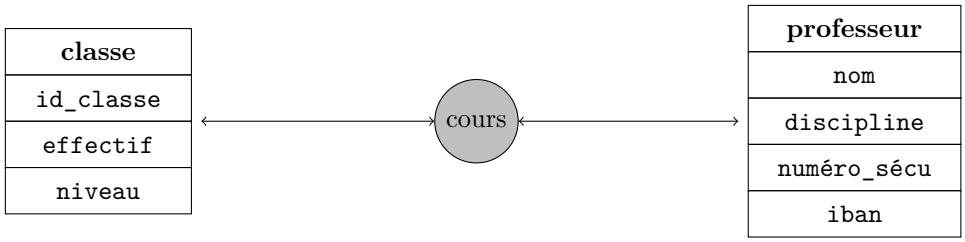
1. Vocabulaire

Une **entité** est un objet (concret ou abstrait) qui intervient dans le système que l'on cherche à modéliser par une base de données : une entité peut être une personne, un lieu, un objet physique, ... Elle est associée à un certain nombre de caractéristiques qui lui sont propres : une personne peut, par exemple, avoir comme caractéristiques sont nom, son prénom, sa date de naissance, sa profession,...

En général, quand on conçoit une base de données, on crée une table pour chaque type d'entités qui intervient, les attributs sont alors les caractéristiques des entités et chaque entité est alors un des enregistrement de la table.

On cherche ensuite, lorsqu'on utilise plusieurs tables à faire apparaître les relations entre les différentes entités de deux tables ou plus : ce sont les **associations**.

Si on cherche à concevoir une base de données pour représenter les classes du lycée Montaigne, on peut définir un premier type d'entités qui sont les classes et dont les caractéristiques sont leur identifiant (`PSI1` par exemple), leur effectif, le niveau (`terminale` ou `CPGE2` par exemple) ; on peut ensuite définir un autre type d'entités qui sont les professeurs dont les caractéristiques sont leur nom, leur discipline, leur numéro de sécurité sociale, leur coordonnées bancaires (IBAN, pour les payer !). Une association entre ces deux types d'entités est alors la participation à un cours : les classes suivent des cours dispensés par certains des professeurs. On peut alors schématiser la situation de la façon suivante :



2. Cardinalité d'une association

La cardinalité d'une association entre deux tables `T1` et `T2` dépend du nombre de fois où une entité peut participer à une association :

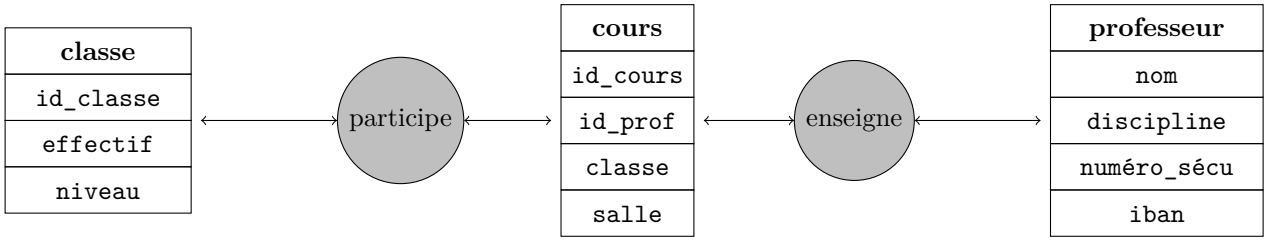
- on parle de cardinalité `1 - 1` si chaque entité (de chacune des deux tables) participe exactement une fois à la relation.
- on parle de cardinalité `1 - *` si chaque entité de la table `T1` participe une seule fois à la relation et si chaque entité de la table `T2` peut participer plusieurs fois à la relation ; on définit de manière symétrique la cardinalité `* - 1`.
- on parle de cardinalité `* - *` lorsque chaque entité de `T1` et chaque entité de `T2` peuvent participer plusieurs fois à la relation.

Dans l'exemple de la base de données sur le tableau périodique des éléments, entre la table `tableau` et la table `grandeurs`, on avait une association `1 - 1`. Ce genre d'association permet facilement de faire correspondre les deux tables : il suffit de les joindre en faisant coïncider leurs clés primaires (qui étaient les symboles des éléments chimiques dans cet exemple).

Dans l'exemple précédent, chaque classe participe à plusieurs cours et chaque professeur enseigne dans différentes classes : on a donc une association `* - *`. Ce genre d'association rend difficile la création de correspondances entre les tables. Comme on va le voir par la suite, une association `1 - *` peut facilement être traduite en SQL.

3. Séparation d'une association * - *

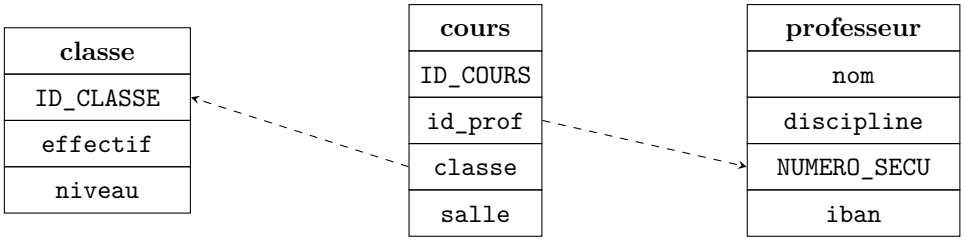
Comme une association `* - *` est peu pratique, on va la séparer en deux associations `1 - *` en créant une nouveau type d'entités (une nouvelle table).
On crée un nouveau type d'entités `cours`, pour lequel les caractéristiques sont un identifiant (`PSI1_maths`, `PSI1_anglais1`, `PSI2_ITC`, ...), `id_prof` qui représente l'enseignant de ce cours, la classe concernée par ce cours et la salle dans laquelle il se déroule.
Le schéma de la relation devient le suivant :



On a maintenant deux associations `1 - *` :
- chaque cours ne concerne qu'une seule classe (on simplifie la situation et donc les regroupements d'élèves de différentes classes comme pour des cours de langue sont considérés impossibles)
- chaque cours n'est réalisé que par un seul professeur.

4. Clés étrangères

Avec le schéma précédent, il est facile de faire correspondre les différentes entités :
- l'attribut `id_classe` de la table `classes` correspond à l'attribut `classe` de la table `cours`
- l'attribut `numéro_sécu` de la table `professeurs` correspond à l'attribut `id_prof` de la table cours



Dans le schéma précédent, les attributs en majuscule désigne les clés primaires de chacune des tables.
Il est alors possible de définir les attributs `classe` et `id_profs` de la table `cours` comme des **clés étrangères** de cette table : ils font référence respectivement aux attributs `ID_CLASSE` et `NUMERO_SECU`. La déclaration d'une clé étrangère (comme celle d'une clé primaire) se fait au moment de la création de la table ; il est alors indispensable que pour tous les enregistrements de la table `cours`, les attributs `classe` et `id_prof` aient des valeurs présentes dans les deux autres tables. Une clé étrangère est donc un attribut (ou un ensemble d'attributs) d'une table qui référence un attribut d'une autre table (pas forcément une clé primaire de l'autre table, même si c'est le cas le plus courant).