Notebook 2 - Bases de données - Plusieurs tables [1]

Cette fois-ci, on veut tirer des infos à partir de plusieurs tables ayant des éléments en commun mais disséminés. On va dans ce but utiliser une base de données des villes de France.

I - Description de la base utilisée et notion de schéma de tables

Cette base est constituée de trois tables `communes`, `regions` et `departements`, décrites ci-dessous. Par usage, les clefs primaires sont soulignées.

communes	
id	str
dep	str
nom	str
pop	int

departements	
id	str
reg	int
nom	str

regions	
id	int
nom	str

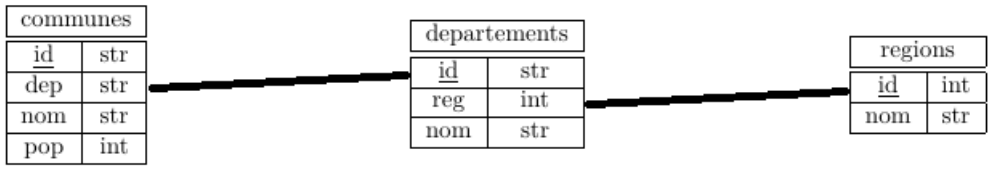
La relation `communes` possède une clef primaire `idt` permettant d'identifier de façon unique chaque ville de France, dont le nom est placé dans le champ `nom`. Le département est une chaîne de caractères placée dans le champ `dep`. Le champ `pop` contient la population de la ville.

1 Pourquoi le champ `nom` ne constitue-t-il pas une clef primaire ?

2 En affichant l'ensemble des valeurs possibles de `dep`, expliquer pourquoi le type utilisé est la chaîne de caractères et non le type entier.

La table `departements` comprend une clef primaire qui correspond au département, un champ `nom` qui est le nom du département et un champ `reg` qui est l'identification de la région d'appartenance du département (au moment de la création de cette base, il n'y avait pas encore eu la réforme des régions). Enfin, la table `regions` contient une clef primaire numérotant la région et correspondant à la même numérotation que celle du champ `reg` de la table `departements`, et un champ `nom` contenant... le nom !

Un **schéma de tables** permet de définir les différentes **tables** avec leurs **attributs** et les **types** associés ainsi que les **relations** qui permettent de relier les attributs des différentes tables ensembles. Voici par exemple le schéma de tables obtenu à partir de la base de données proposée.



Les droites permettent de voir quels sont les attributs qui coïncident entre les différentes tables. Ainsi, par exemple, le champ `dep` de la table `communes` est une chaîne de caractères permettant de connaître le département de la ville consiédérée, et cette chaîne a exactement la même signification que celle du champ `id` de la table `departements`. Il en est de même, comme déjà évoqué, pour le champ `reg` de la table `departements` et le champ `id` de la table `regions`. Réaliser une **jointure** entre les tables `communes` et `departements` en précisant que `dep` et `id` (pour `departements`) sont identiques, c'est en quelque sorte fusionner les deux tables afin de rendre possible de trouver, par exemple, le nom du département d'une ville, et non simplement son numéro. Finissons en disant que `dep`, par exemple, est une **clef étrangère** : ce n'est pas une clef primaire de sa table d'origine, mais elle fait le lien et s'identifie avec la clef primaire d'une autre table.

1. Code Capytale 3d01-635279

II - Jointure de tables : JOIN ... ON...

On vient d'introduire intuitivement la notion de jointure entre deux tables : on veut relier deux tables afin de tirer des informations de plusieurs d'entre elles. Pour réaliser la jointure de différentes tables, on utilise la syntaxe suivante :

SELECT exp **FROM** tab1 **JOIN** tab2 **ON** relation **WHERE** cond

La relation suivant le **ON** permet d'identifier les colonnes correspondant à la même « chose ». Par exemple, dans la base fournie, on peut lier les deux tables `communes` et `departements` par

FROM communes **JOIN** departements **ON** dep = departements.id

Ici, vous remarquerez que nous avons dû écrire quelque chose de bizarre : `departements.id`. En effet, le champ `id` est utilisé comme nom pour les deux tables. Écrire le nom de la table `departements` avant `id` permet de caractériser de façon univoque la colonne utilisée portant ce nom, donc ici la colonne `id` de la table `departements`. Ainsi, quand les noms des colonnes sont différents entre les tables, il n'y a pas d'ambiguïtés possibles lors de la jointure. En revanche, si plusieurs tables ont des colonnes de même nom, il faut les distinguer. À partir de cette remarque préalable, regardons la requête suivante

SELECT communes.nom, departements.nom **FROM** communes **JOIN** departements **ON** communes.dep = departements.id

Ici, on utilise cette syntaxe avec le nom de la table car dans chacune d'entre elles, il existe une colonne s'appelant `nom`. Toutefois, on peut noter que `communes.dep` est superflu, on aurait pu écrire simplement `dep` puisque ce nom n'apparaît que dans la table `communes`. La dernière chose à souligner est le caractère pénible de cette écriture, surtout quand les noms de tables utilisés sont longs... Il existe cependant un moyen de faire plus malin :

SELECT C.nom, D.nom **FROM** communes **AS** C **JOIN** departements **AS** D **ON** C.dep = D.id

Cette fois, la syntaxe permet de **joindre** les deux tables `communes` et `departements` grâce à l'utilisation d'un **alias**, c'est-à-dire d'un nouveau mot-clef **AS** permettant de **renommer des variables**. Notons une nouvelle fois que l'on aurait pu écrire tout simplement `dep` plutôt que `C.nom`. Pour finir, **et même si cela ne fonctionnerait pas**, voici la requête avec des parenthèses pour mieux comprendre la structure de celle-ci :

SELECT C.nom, D.nom **FROM** (communes **AS** C **JOIN** departements **AS** D) **ON** (C.dep = D.id)

La plupart du temps au concours, le point clef sera de trouver quelles sont les colonnes sur lesquelles on réalise la fusion, c'est-à-dire les colonnes « signifiant » la même chose. Seul l'humain peut le préciser !

Il existe deux types de **renommages** : le renommage d'une table, comme ci-dessus, et celui d'une projection (c'est-à-dire d'une donnée que l'on veut afficher, qui est donc soit un des champs de la table, soit une grandeur construite à partir de champs de la table). Dans le premier cas, le `AS` est utilisé dans le `FROM`, lors de la jointure, comme par exemple ci-dessus. Dans le second cas, on peut donner, en reprenant le premier TD, l'exemple suivant, qui parle de lui-même :

```
SELECT ab+bc+ac AS perimetre FROM triangles WHERE perimetre>500
```

Et s'il y a plus de deux tables ? Dans ce cas, plusieurs syntaxes fonctionnent, mais je vous recommande la suivante (si on relie trois tables `A`, `B` et `C` avec par exemple les colonnes `bambou` de `A` et `panda` de `B` qui coïncident d'une part, et les colonnes `babar` de `B` et `celeste` de `C` d'autre part). Une fois de plus, les parenthèses sont présentes pour mieux visualiser, mais il ne faut pas les mettre (en réalité, cela marchera sur certaines distributions de SQL et pas sur d'autres. Par exemple, cela ne marche pas sur Capytale...)

SELECT * **FROM** (A **JOIN** B **JOIN** C) **ON** (bambou=panda **AND** babar=celeste)

3 Déterminer la liste de toutes les communes avec, pour chacune, son département, sa région et sa population.

4 Donner la liste des communes de plus de 100 000 habitants, ainsi que leurs populations et leurs régions.

5 Donner le nombre de noms de villes de plus de 100 000 habitants. Quelle différence ? Et bien plusieurs villes peuvent avoir le même nom :-)... On utilisera à cette fin le nouveau mot-clef **DISTINCT**, qui s'utilise après **SELECT** et avant **FROM**, comme une fonction d'agrégation.

6 Trier les villes de la liste obtenue deux questions plus haut par ordre décroissant de population.

Notations alternatives :

- On peut effectuer des « fusions » sans utiliser de jointure, mais en bidouillant un peu. La première méthode consiste à utiliser la forme standard **SELECT... FROM... WHERE...** sur **deux** tables et préciser dans le **WHERE** la condition de fusion. En reprenant l'exemple plus haut, on aurait pu écrire :

  SELECT C.nom, D.nom **FROM** communes **AS** C, departements **AS** D **WHERE** C.dep = D.id

  Ce n'est pas une jointure à proprement parler puisque l'on n'utilise pas le mot-clef **JOIN**. Les deux requêtes sont équivalentes en efficacité, pour des raisons qui ne sont pas au programme. On peut donc choisir l'approche que l'on préfère, tout en faisant attention à l'énoncé qui, s'il demande une jointure, appelle alors la première.
- On peut aussi faire un `JOIN` mais sans mettre de `ON`, dans le cas où les tables liées ont une colonne de même nom : dans ce cas, la syntaxe SQL effectue la jointure « naturelle » entre les tables.

III - Autojointure - Difficile, vous pouvez le sauter en première lecture

Il est parfois utile de joindre une table avec elle-même, ne serait-ce que lorsque l'on a besoin de deux fois certaines informations. Dans ce cas, on utilisera obligatoirement des alias car on aura besoin de savoir quelle information vient de la première utilisation de la table et quelle info vient de la seconde. Rien de mieux qu'un exemple !

7 Quel est le but de la requête suivante ? On rappelle que l'opérateur <> signifie « différent de ».

SELECT C1.nom, C2.nom, C2.dep **FROM** communes **AS** C1 **JOIN** communes **AS** C2 **ON** C1.nom=C2.nom **WHERE** C1.id <> C2.id **AND** C1.dep=1

IV - GROUP BY et HAVING : révisions

8 Écrire une requête qui permet d'obtenir le numéro du département suivi de sa population. En déduire le département le plus peuplé de France (la requête doit afficher une seule ligne donnant le numéro et la population). Notez bien qu'ici, les informations demandées sont contenues dans la seule table `communes` : pas besoin de jointure.

9 Si on veut le nom du département précédent, il faut alors faire une jointure. Expliquer pourquoi et donner la requête permettant de répondre à la question précédente.

10 Proposer une requête affichant le nom et la population des départements uniquement si cette dernière est supérieure à 1,5 millions d'habitants.

V - Pour le plaisir

Les questions suivantes sont plus funs mais utilisent des fonctions SQL **qui ne sont pas au programme**. Mais après tout, pourquoi ne pas les connaître et, si vous êtes un jour bloqués, les utiliser pour répondre quelque chose plutôt que rien du tout ?

11 Donner la liste des communes (nom et population) dont le nom commence par PA, finit par IS, en utilisant éventuellement le mot-clef **LIKE** (quelques explications sur cette commance sur https ://sql.sh/cours/where/like, ou dans la fiche synthèse).

12 Déterminer les communes qui ont strictement plus de lettres dans leur nom que leur nombre d'habitants. Il vous faudra utiliser le mot-clef **LENGTH**.

VI - Pour conclure, syntaxe générale !

On peut retenir que la syntaxe générale d'une requête SQL est de la forme suivante, sachant qu'il existe bien sûr de nombreux autres mots-clefs qui ne sont pas au programme, et que, comme nous l'avons déjà dit, seuls **SELECT** et **FROM** sont strictement indispensables :

SELECT expressions **FROM** tables **JOIN**.... **ON**.... **WHERE** conditions
GROUP BY attributs **HAVING** conditions **ORDER BY** attributs **ASC/DESC LIMIT** xx **OFFSET** yy

VII - Bonus : un ancien DS machine !

La base de données utilisée est celle du TD !

1 Combien y-a-t-il de communes en France ?

2 Combien y-a-t-il de communes dans le département numéro 17 ?

3 Combien y-a-t-il de communes en tout dans l'ensemble Ain, Aisne, Allier ?

4 Donner la requête SQL permettant d'afficher le nom de la commune, celui de son département et de sa région et de sa population, lorsque celle-ci possède une population entre 15 000 et 20 000 habitants (au sens large).

5 Donner la population totale de la région Centre.

6 *Difficile :* Donner la région de France dont la population moyenne par commune est la plus petite. Ce n'est donc pas forcément la moins peuplée !

VIII - Un exemple de sujet d'écrit - Mines 2021

Dans la suite, vous devez répondre sans utiliser de logiciel et uniquement à la main. Rien de difficile à cela, puisque de toutes façons vous n'avez pas la base de données proposée :-). Il s'agit du sujet Mines Communes 2021. Je vous demanderai plusieurs façons de répondre pour vous entraîner, mais bien entendu, une seule suffirait au concours. Profitons-en pour expliquer comment bien présenter une requête SQL. La façon de faire élégante est la suivante : si la requête est simple et comporte peu d'éléments, on met tout sur une seule ligne. En revanche, si on a plusieurs éléments (par exemple une jointure), on met les clauses à raison d'une seule par ligne. Par exemple, avec la base de la partie II, si on veut les noms des communes de plus de 10 000 habitants et de leurs départements respectifs, ainsi que leurs populations triéss de façon croissante, on pourrait répondre ainsi :

```
SELECT C.nom, D.nom, pop
FROM communes AS C JOIN departements AS D ON dep = D.id
WHERE pop > 10 000
ORDER BY pop
```

Vous remarquerez immédiatement la lisibilité supérieure, la facilité pour quelqu'un qui vous lit de savoir ce que vous faites... C'est hautement appréciable ! Notez que cette façon de présenter n'est pas du tout une obligation. Mais c'est quand même beaucoup mieux. Allez, au boulot maintenant !

Lors de la préparation d'une randonnée, une accompagnatrice doit prendre en compte les exigences des participants. Elle dispose d'informations rassemblées dans deux tables d'une base de données :

- La table `Rando` décrit les randonnées possibles – la clef primaire entière `rid`, son nom, le niveau de difficulté du parcours (entier entre 1 et 5), le dénivelé en mètres, la durée moyenne en minutes :

rid	rnom	diff	deniv	duree
1	La belle des champs	1	20	30
2	Lac de Castellagne	4	650	150
3	Le tour du mont	2	200	120
4	Les crêtes de la mort	5	1200	360
5	Yukon Ho !	3	700	210
...	...	...	...	...

- La table `Participant` décrit les randonneurs – la clef primaire entière `pid`, le nom du randonneur, son année de naissance, le niveau de difficulté maximum de ses randonnées :

pid	pnom	ne	diff_max
1	Calvin	2014	2
2	Hobbes	2015	2
3	Susie	2014	2
4	Rosalyn	2001	4
...	...	...	...

1 Compter le nombre de participants nés entre 1999 et 2003 inclus. Deux solutions : « normale », ou bien sans le mot-clef `AND` mais avec `ABS`, s'il le faut (et si vous devinez ce que c'est comme mot-clef !).

2 Calculer la durée moyenne des randonnées pour chaque niveau de difficulté. Deux solutions : avec ou sans `GROUP BY`. *Indice :* dans le second cas, utiliser `UNION`.

3 Extraire le nom des participants pour lesquels la randonnée n°42 est trop difficile. On le fera avec une jointure puis uniquement avec des structures `SELECT FROM WHERE` mais avec une sous-requête.

4 Extraire les clés primaires des randonnées qui ont un ou des homonymes (nom identique et clé primaire distincte), sans redondance.