Exemples de requêtes SQL

I - Questions sur la géographie française

On trouve sur le blog du site `sql.sh` plusieurs bases de données gratuites, dont j'ai extrait — sur une idée de mon (excellent) collègue Mathieu Roux — trois tables concernant les (anciennes) régions, les départements et les communes de France ; voici les attributs disponibles :

Table `communes`		Table `departements`	Table `regions`
`num_departement`	`surface`	`num_departement`	`num_region`
`nom`	`longitude`	`num_region`	`nom`
`canton`	`latitude`	`nom`	
`population_2010`	`zmin`		
`population_1999`	`zmax`		

Les noms sont suffisamment explicites ; précisons toutefois que `zmin`, `zmax` sont des altitudes en mètres et que `longitude`, `latitude` sont données en milligrades.

Ces tables sont placées dans une base de données SQLite nommée `geographie.sqlite`, disponible sur le site `PSIxClem` pour ceux qui voudraient faire des travaux pratiques... Pour cela on peut utiliser par exemple le logiciel `SQLiteman` ou le module complémentaire `SQLite Manager` de `Firefox`.

Les sections suivantes proposent un certain nombre de questions auxquelles une requête SQL convenable permet de répondre.

Sauf indication contraire, les questions de population se rapportent au recensement de 2010.

1) Requêtes sans jointure

1) Combien d'habitants en France (en supposant que chacun n'a été compté qu'une fois dans les différentes communes...) ?

2) Combien de communes en Loire-Atlantique ?

3) Quelles sont les dix plus petites communes de France, en superficie ?

4) Quelles sont les dix communes de France les plus peuplées ?

5) Quelles sont les douze communes de Loire-Atlantique les plus densément peuplées ?

6) Donner la liste des numéros de département, avec pour chaque numéro le nombre de communes du département. Afficher un titre explicite pour la colonne des nombres de communes.

7) Donner la liste des numéros de département, avec pour chaque numéro la population totale du département. Trier par population totale décroissante et limiter la liste aux départements ayant plus d'un million d'habitants.

8) Donner les communes ayant les six voyelles dans leur nom. *Noter que* SQL *ne distingue pas majuscules/minuscules.* Lister ces noms, triés par longueur croissante *(la fonction* `LENGTH` *donne la longueur d'une chaîne de caractères).* Y en a-t-il en Loire-Atlantique ? Dans le Maine-et-Loire ?

9) Quelles sont les quinze communes pour lesquelles l'écart entre les altitudes maximale et minimale est le plus grand ?

10) Quelles sont les dix communes de Loire-Atlantique où la population a le plus augmenté en valeurs absolues, entre 1999 et 2010 ?

11) Même question en valeurs relatives. **Attention !** *Pour* SQL*, un quotient d'entiers est le quotient (entier !) de la division euclidienne. Pour forcer un calcul en flottants, on peut multiplier par* `1.0` !

2) Requêtes avec jointures

1) Donner la liste des noms des départements des régions Bretagne et Pays de la Loire.

2) Donner la liste des noms de départements, avec pour chaque département le nombre de communes. Ordonner par population décroissante.

3) Donner la liste des noms des régions avec la densité de population de chaque région.

4) *Un exemple où le mot-clé* `DISTINCT` *est utile, comme dans le sujet du concours Centrale 2015...*

Donner **sans doublons** la liste des noms des départements contenant une commune dont le nom commence par "Petit".

3) Requêtes imbriquées

Le résultat d'un `SELECT` *est une table qui peut être réutilisée dans une autre requête. C'est ce qu'il fallait faire dans 2 questions sur 3 de la partie "bases de données" du sujet du concours Mines-Ponts 2015 et c'est ce qui était sans doute attendu dans le sujet Mines-Ponts 2016...*

1) *Lorsqu'un* `SELECT` *renvoie un simple scalaire (par exemple via une fonction d'agrégation), on peut réutiliser ladite requête* `SELECT...` *(entre parenthèses) comme ledit scalaire.*

Donner la liste des noms de communes dont la population excède 100 fois la population moyenne des différentes communes.

2) *Un peu comme aux Mines 2016...*

Donner la liste des noms de communes dont l'altitude minimale est la deuxième plus petite valeur parmi celles qui sont au moins égales à 940.

3) Donner sans doublons la liste des noms des régions contenant au moins un département dont le nom commence par "V". *On peut utiliser* `DISTINCT` *ou* `EXISTS`...

4) Donner sans doublons la liste des noms des régions ne contenant aucune commune dont le nom contient les six voyelles.

5) Donner la liste des noms des régions avec pour chacune le nom et la population de la commune la plus peuplée de la région.

II - Exemples de solutions

1) Requêtes sans jointure

1)
```
SELECT SUM(population_2010) FROM communes;
```

2)
```
SELECT COUNT(*) FROM communes WHERE num_departement=44;
```

3)
```
SELECT nom, num_departement, surface FROM communes ORDER BY surface LIMIT 10;
```

4)
```
SELECT nom, num_departement,population_2010
FROM communes ORDER BY population_2010 DESC LIMIT 10;
```

5)
```
SELECT nom, population_2010/surface AS densité
FROM communes WHERE num_departement=44 ORDER BY densité DESC LIMIT 12;
```

6)
```
SELECT num_departement, COUNT(*) AS nb_communes
FROM communes GROUP BY num_departement;
```

7)
```
SELECT num_departement, SUM(population_2010) AS population FROM communes
GROUP BY num_departement HAVING population>1000000 ORDER BY population DESC;
```

8)
```
SELECT nom FROM communes WHERE nom LIKE '%a%' AND nom LIKE '%e%'
AND nom LIKE '%i%' AND nom LIKE '%o%' AND nom LIKE '%u%' AND nom LIKE '%y%'
ORDER BY LENGTH(nom);
```
Pour voir par exemple s'il y en a en Loire-Atlantique, ajouter `AND num_departement=44` !

9)
```
SELECT nom, num_departement, zmax-zmin as écart FROM communes
ORDER BY écart DESC LIMIT 15;
```

10)
```
SELECT nom, population_2010-population_1999 AS var_abs
FROM communes WHERE num_departement=44 ORDER BY var_abs DESC LIMIT 10;
```

11)
```
SELECT nom, 1.*population_2010/population_1999-1 AS var_rel
FROM communes WHERE num_departement=44 ORDER BY var_rel DESC LIMIT 10;
```

2) Requêtes avec jointures

1)
```
SELECT departements.nom, regions.nom
FROM departements JOIN regions
ON departements.num_region = regions.num_region AND
  (regions.nom = 'Bretagne' OR regions.nom = 'Pays de la Loire');
```

2)
```
SELECT departements.nom, departements.num_departement, COUNT(*) as nb_communes,
SUM(population_2010) AS population FROM communes
JOIN departements ON communes.num_departement = departements.num_departement
GROUP BY communes.num_departement ORDER BY population DESC;
```

3)
```
SELECT regions.nom, SUM(population_2010)/SUM(surface) AS densité
FROM communes
JOIN departements ON communes.num_departement = departements.num_departement
JOIN regions ON departements.num_region = regions.num_region
GROUP BY regions.num_region ORDER BY densité DESC;
```

4)
```
SELECT DISTINCT departements.nom
FROM communes JOIN departements
ON communes.num_departement = departements.num_departement
WHERE communes.nom LIKE 'petit%';
```

3) Requêtes imbriquées

1)
```
SELECT nom, population_2010 AS population FROM communes
WHERE population > 100*(SELECT AVG(population_2010) FROM communes);
```

2)
```
SELECT nom, num_departement, zmin FROM communes
WHERE zmin = (SELECT MIN(zmin) FROM communes WHERE zmin >= 940
AND zmin != (SELECT MIN(zmin) FROM communes WHERE zmin >= 940));
```
On peut penser à utiliser un tri avec l'option `OFFSET` mais, d'une part ce n'est pas explicitement au programme, d'autre part cela ne fonctionne qu'en l'absence d'ex æquo...

3) **a)** Première version avec `DISTINCT` :
```
SELECT DISTINCT regions.nom
FROM departements JOIN regions
ON departements.num_region = regions.num_region
WHERE departements.nom LIKE 'V%';
```

b) Seconde version avec `EXISTS`, qui suppose d'imbriquer une deuxième requête :
```
SELECT nom FROM regions
WHERE EXISTS (SELECT * FROM departements
              WHERE departements.num_region = regions.num_region AND
                    departements.nom LIKE 'V%');
```

4) **a)** Première version avec `DISTINCT` :
```
SELECT DISTINCT regions.nom AS nr FROM regions
WHERE nr NOT IN (SELECT regions.nom FROM communes
JOIN departements ON communes.num_departement = departements.num_departement
JOIN regions ON departements.num_region = regions.num_region
WHERE communes.nom LIKE '%a%' AND communes.nom LIKE '%e%' AND
      communes.nom LIKE '%i%' AND communes.nom LIKE '%o%' AND
      communes.nom LIKE '%u%' AND communes.nom LIKE '%y%');
```

b) Seconde version avec `EXISTS` :
```
SELECT nom FROM regions
WHERE NOT EXISTS (SELECT * FROM communes
JOIN departements ON communes.num_departement = departements.num_departement
WHERE departements.num_region = regions.num_region AND
      communes.nom LIKE '%a%' AND communes.nom LIKE '%e%' AND
      communes.nom LIKE '%i%' AND communes.nom LIKE '%o%' AND
      communes.nom LIKE '%u%' AND communes.nom LIKE '%y%');
```

5)
```
SELECT nom_région, communes.nom, population_2010 AS population
FROM (SELECT regions.nom AS nom_région, regions.num_region AS nr,
             MAX(population_2010) AS pop_max
      FROM communes
      JOIN departements ON communes.num_departement = departements.num_departement
      JOIN regions ON regions.num_region = departements.num_region
      GROUP BY regions.num_region)
JOIN communes ON population_2010=pop_max
ORDER BY population DESC;
```