TP BDD : Corrigé

Question 1. Les clés primaires sont `id` pour les deux tables. L'attribut `chef_lieu` de la table `depts` est une clé secondaire référençant la clé primaire de la table `communes`, et c'est l'inverse pour le `dep_id` de la table `communes`.

Question 2.

```
SELECT * FROM communes
SELECT * FROM depts
```

Question 3. *Sélection.*

```
SELECT * FROM communes WHERE dep_id=6
```

Question 4. *Sélection multiples.* Deux solutions :

```
SELECT * FROM communes WHERE dep_id=6 AND population > 10
SELECT * FROM communes WHERE dep_id = 6 INTERSECT SELECT * FROM communes WHERE population > 10
```

Question 5. *Ordonnancement.*

```
SELECT * FROM communes WHERE dep_id=6 AND population > 10 ORDER by nom
```

Question 6. *Ordonnancement bis.*

```
SELECT * FROM communes WHERE dep_id=6 AND population > 10 ORDER by population DESC
```

Question 7. *Limitation de l'affichage.*

```
SELECT * FROM communes WHERE dep_id=6 AND population > 10 ORDER by population DESC LIMIT 1
```

(Le résultat est le même sans condition sur la population, bien sûr).

Question 8. *Ignorer les premiers résultats.*

```
SELECT * FROM communes WHERE dep_id=6 ORDER by superficie DESC LIMIT 20 OFFSET 100
```

Question 9. *Utilisation de LIKE, hors programme.*

```
SELECT nom FROM communes WHERE nom LIKE "AB%"
```

Question 10. *Nombre.*

```
SELECT COUNT(*) FROM communes
SELECT COUNT(*) FROM depts
```

Question 11. *Moyenne.*

```
SELECT AVG(population) FROM communes WHERE dep_id=6
```

Question 12. *Somme.*

```
SELECT SUM(population)*1000 FROM communes WHERE dep_id=6
```

Question 13. *Min et Max.*

```
SELECT MIN(altitude), MAX(altitude) FROM communes
```

Question 14. *Utilisation de DISTINCT.*

```
SELECT COUNT(DISTINCT altitude) FROM communes
```

Question 15. *Regroupement.*

```
SELECT dep_id, SUM(population) FROM communes GROUP BY dep_id
```

Question 16. *Sélection après une agrégation.*

```
SELECT dep_id, SUM(population) s FROM communes GROUP BY dep_id HAVING s>500 ORDER BY s DESC
```

Question 17.

```
SELECT c.nom, d.nom FROM communes c JOIN depts d ON c.dep_id=d.id ORDER BY superficie DESC LIMIT 10
```

Question 18. Rajouter `DESC` après `population` pour la deuxième partie de la question.

```
SELECT c.nom FROM communes c JOIN depts d ON c.id = d.chef_lieu ORDER BY population LIMIT 1
```

Question 19.

```
SELECT  AVG(superficie) FROM communes c JOIN depts d ON c.dep_id = d.id WHERE d.code = "2A" OR
d.code = "2B"
```

Question 20.

```
SELECT d.nom, SUM(superficie) s FROM communes c JOIN depts d ON c.dep_id = d.id GROUP BY d.id
ORDER BY s DESC
```

Question 21.

```
SELECT nom, altitude FROM communes WHERE altitude = (SELECT MAX(altitude) FROM communes)
```

Question 22.

```
SELECT COUNT(*) FROM communes WHERE altitude >= (SELECT AVG(altitude) FROM communes)
```

Question 23.

```
SELECT COUNT(*) FROM communes WHERE altitude = (SELECT ROUND(AVG(altitude)) FROM communes)
```

Question 24.

```
SELECT AVG(0.01*s) FROM (SELECT SUM(superficie) s FROM communes GROUP BY dep_id)
```

Autre version :

```
SELECT (SELECT 0.01*SUM(superficie) FROM communes)/(SELECT COUNT(*)  FROM depts)
```

Question 25.

```
SELECT * FROM communes WHERE id=
  (SELECT d.chef_lieu FROM communes c JOIN depts d ON c.dep_id = d.id
  GROUP BY d.id ORDER BY SUM(population)
  LIMIT 1)
```

Version avec deux fois la table `communes`, jointe à `depts` des deux manières :

```
SELECT a.nom FROM communes a JOIN communes b JOIN depts d ON a.id = d.chef_lieu AND b.dep_id = d.id
GROUP BY d.id
ORDER BY SUM(b.population)
LIMIT 1
```

Question 26.

```
WITH a AS (SELECT * FROM communes WHERE dep_id=6)
  SELECT a.nom, b.nom, a.altitude FROM a JOIN a AS b ON a.altitude = b.altitude AND a.id < b.id
  ORDER BY a.altitude
```

Question 27.

```
SELECT COUNT(*) FROM
 (SELECT nom n, altitude a FROM COMMUNES WHERE dep_id = 6 AND
   EXISTS (SELECT * FROM communes WHERE dep_id = 6 AND nom <> n AND altitude = a))
```