

# Activité les données structurées des bases de données

Quand nous avons une **grande quantité d'informations numériques** à gérer, il est nécessaire de les **organiser** et d'utiliser un **logiciel spécialisé**, le **Systèmes de Gestion de Bases de Données** pour les utiliser.

Actuellement la grande majorité des SGBD utilise le **modèle relationnel** qui consiste :

- à mettre ensemble des **données de même nature** en les rangeant dans des **tables** de données ;
- à définir des **relations** pour relier et croiser les données **entre elles**.
- à utiliser le **langage SQL** ((Structured Query Language) pour **interroger, modifier ou supprimer** des données.

Dans cette activité je vais **interroger** les données sur les villes française du site web <http://sio.lycees.nouvelle-aquitaine.pro/sql> en utilisant le langage de requêtes SQL pour exécuter des **requêtes SQL** sur la table **ville**.

Il existe plusieurs logiciels de gestion de base de données. **MySQL** (racheté par la société Oracle), continue à être une **base de données OpenSource** sous le nom de projet **MariaDB**.

## Les requêtes SQL

### La projection

La projection consiste à choisir les informations, en utilisant la clause **select** suivie les champs à afficher :

- Exemple :

```
SELECT *  
FROM ville ;
```

Exécuter une ou des requêtes SQL sur la base données:

```

1 SELECT *
2 FROM villes ;

```

j'écris la requête SQL

1

[ Délimiteur ; ]  Afficher à nouveau la requête après exécution  Conserver la boîte de requêtes  ROLLBACK à la fin

les champs de la table : les descripteurs

2

j'exécute le requête SQL

Exécuter

Le résultat de la requête

Options	dep	nom	cp	nb_hab_2010	nb_hab_1999	nb_hab_2012	dens	surf	long	lat	alt
1	Ozan		1190	618	469	500	93	6,6	4,91667	46,3833	1
1	Cormoranche-sur-Saône		1290	1058	903	1000	107	9,85	4,83333	46,2333	16
1	Plagne		1130	129	83	100	20	6,2	5,73333	46,1833	56
1	Tossiat		1250	1406	1111	1400	138	10,17	5,31667	46,1333	24
1	Pouillat		1250	88	58	100	14	6,23	5,43333	46,3333	33
1	Torcieu		1230	698	643	700	65	10,72	5,4	45,9167	29

3

Après la clause **select**, j'indique :

- les **colonnes** à afficher **séparées par des virgules** ,
- ou bien je mets le caractère \* pour visualiser le contenu de **toutes les colonnes**.

Puis j'indique avec la clause **from** la table dans laquelle se trouve les données.

**Question 1 :** Ecrire la requête SQL pour avoir le **nom**, le **code postal** et le **numéro de département** (dans cet ordre) des villes.

Résultat à obtenir :

nom	cp	dep
Ozan	1190	1
Cormoranche-sur-Saône	1290	1

...

- je peux renommer une colonne en définissant un alias avec le mot clé **as** et aussi faire des calculs ou utiliser des fonctions : `sql> select concat(nom, ' ', cp) as adresse from villes ;` **Résultat :**

adresse
Ozan 1190
Cormoranche-sur-Saône 1290

**Question 2 :** Ecrire la requête SQL qui indique pour chaque ville l'**augmentation** de la population entre 2010 et 2012. **Résultat à obtenir :**

ville	augmentation
Ozan	-118
Cormoranche-sur-Saône	-58
Plagne	-29
Tossiat	-6
Pouillat	12

...

## Eviter des résultats en double

Des requêtes peuvent renvoyer des **résultats identiques** et il est parfois utile d'éviter cela avec le mot clé **distinct**.

- Exemple connaître la liste des départements sans utiliser **distinct** :

```
SELECT dep AS Département
FROM villes ;
```

Résultat :

Département
1
1
1

... soit **36 700 lignes**.

- Connaître la liste des départements en utilisant **distinct** qui doit être placé une **seule fois** juste après le mot clé **select** :

```
SELECT DISTINCT dep AS Département
FROM villes ;
```

Résultat :

Département
1
2
3

... soit **102 lignes**.

**Question 3 :** Ecrire la requête SQL qui donne la liste des codes postaux. La requête SQL doit renvoyer uniquement **6 082 lignes**.

## Trier les résultats obtenus

Les requêtes SQL renvoient en général les données dans l'**ordre** où elles sont disponibles dans la base de données. Pour obtenir un ordre de **tri différent** on utilise les mots clés **order by** suivi des colonnes à trier en ascendant, par défaut (**asc**) ou en descendant (**desc**).

- Exemple connaître la liste villes par ordre alphabétique :

```
SELECT nom AS Ville
FROM villes
ORDER BY nom ASC ;
```

Résultat :

<b>Ville</b>
Aast
Abainville
Abancourt

... soit **36 700 lignes**.

**Question 4 :** Ecrire la requête SQL qui donne la liste des villes **selon le nom d'habitants** par ordre **décroissant** (indiquer la ville la plus peuplée premier)

### La sélection

J'utilise la **sélection** si je ne souhaite avoir des données qui réponde à une **condition** en utilisant la clause **where** :

- Exemple : avoir toutes les information de la ville de Panazol

```
SELECT *
FROM villes
WHERE nom = 'panazol';
```

**Résultat :** 1 ville(s) trouvée(s) !

dep	nom	cp	nbhab2010	nbhab1999	nbhab2012	dens	surf	longitude	latitude	altmin ^ altmax	
87	Panzol	87350	10392	9727	10100	518	20	1.3	45.8333	215	351

Voici les opérateurs utilisables :

### Les opérateurs de comparaison et logiques

Opérateur de comparaison	Description	Opérateurs logiques	Description
=	égal à	<b>and</b>	les deux conditions doivent être vérifiées simultanément
<	inférieur à	<b>or</b>	au moins une des deux conditions doit être vérifiée
>	supérieur à		
≤	inférieur ou égal		
≥	supérieur ou égal		
<>	différent de		

**Question 5 :** Ecrire la requête SQL qui donne la liste des villes qui ont **plus de 5000 habitants** en **2012**. La requête SQL doit renvoyer uniquement **2 007 lignes**.

### Les opérateur d'appartenance (ou non) d'un élément à un intervalle

<b>BETWEEN</b>	<b>appartenance</b>
<b>NOT BETWEEN</b>	<b>non appartenance</b>

\* Exemple : connaître les villes dont l'altitude maximale est comprise entre 200 et 300 m :

```
SELECT nom AS ville, alt_max
FROM villes
WHERE alt_max BETWEEN 200 AND 300;
```

**Résultat :**

ville	alt_max
Ozan	205
Cormoranche-sur-Saône	211
Replonges	207

...



**Question 6 :** Ecrire la requête SQL qui donne la liste des villes dont la superficie est comprise entre 100 et 300 kilomètres carré. La requête SQL doit renvoyer uniquement **142 lignes**.

### Les opérateur de comparaison de chaînes de caractères

LIKE	comparaison de chaînes (identiques)
NOT LIKE	chaîne différente

% permet de remplacer n caractères \_ permet de remplacer 1 caractère

- Exemple : connaître les villes dont le nom commence par Limoges : `select nom as ville from villes where nom like 'limoges%';` **Résultat :**

ville
Limoges-Fourches
Limoges

**Question 7 :** Ecrire la requête SQL qui donne la liste des villes dont le nom contient les caractères **paris**. La requête SQL doit renvoyer uniquement **10 lignes**.

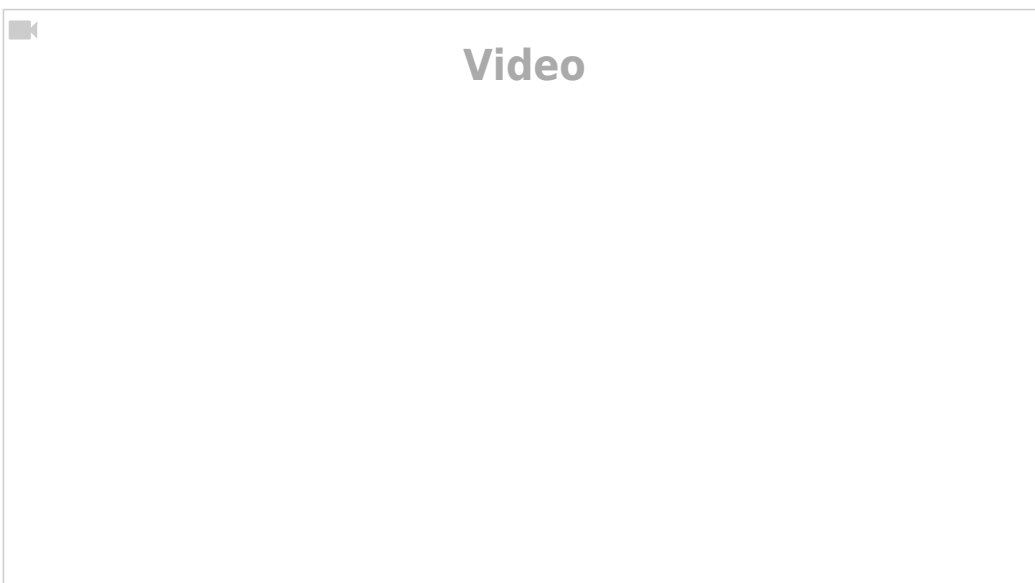
## Activité à faire à la maison

Je **consulte** la vidéo <https://pixees.fr/lhistoire-des-base-de-donnees-ou-presque/> pour répondre aux questions suivantes :

- Quel a été le **premier usage** des bases de données ?
- Quelles sont les **deux tables** utilisées pour gérer les données des expériences du professeur Tournesol ?
- Quel a été la **méthode** utilisée pour retrouver l'**unique expérience** qui a permis d'obtenir une grande rose à partir de deux graines de petite tailles ?

Une petite histoire des bases de données :

- <https://pixees.fr/lhistoire-des-base-de-donnees-ou-presque/>



Je continue ...

Je reviens à l'accueil SNT du thème [Les données structurées et leur traitement](#)

From:

/ - **Les cours du BTS SIO**

Permanent link:

</doku.php/snt/donnee/activite5?rev=1568279737>

Last update: **2019/09/12 11:15**

