Le SQL pour les MGTMN : résumé

Ce document reprend les éléments essentiels du cours présent sur le site.
Les captures d'écran qui s'y trouvent sont tronquées par manque de place.
Vous avez également à votre disposition :

- un tutoriel du logiciel DB Browser for SQLite ;
- un memento du langage SQL.

I. Création ou modification d'une base de données

Elle peut se faire avec un logiciel comme DB Browser for SQLite, à l'aide d'une interface graphique, ou avec des requêtes SQL (hors programme).

Une table est un tableau de la base de données.
Un champ est une colonne d'une table.

Les noms des tables et des colonnes ne sont pas quelconques :

- ils peuvent comporter des lettres, des chiffres, des symboles comme #, $, etc. ;
- ils ne peuvent pas commencer par un chiffre ni comporter d'espace (utiliser _ dans ce cas) ;
- les majuscules ou minuscules sont équivalentes (ce n'est pas le cas des valeurs, par exemple "Paris" ne revient pas à "paris") ;
- ils ne doivent pas être des mots du langage SQL comme SELECT, FROM, etc.

Enfin, les données entrées peuvent être de différents types :

- INTEGER : entier ;
- TEXT : un texte (une *chaîne de caractères*) ;
- BLOB : données brutes au format binaire ;
- REAL : nombres à virgules ;
- NUMERIC : texte contenant un nombre (exemple : "59") ou nombre (exemple : 59).

II. Le langage SQL

C'est un langage permettant de créer, modifier ou interroger une base de données à l'aide de requêtes.
Nous ne verrons ici que les interrogations (extraction de données à partir de table(s)), qui commencent toujours par le mot clé SELECT.
Le résultat d'une telle requête est une table.

Ouvrez le logiciel DB Browser for SQLite.
Les requêtes seront saisies dans l'onglet « Exécuter le SQL ». Vous pouvez couper les requêtes en plusieurs lignes pour mieux vous y retrouver.

Si vous n'avez qu'une requête à exécuter, utilisez F5.
Si vous avez plusieurs requêtes à exécuter successivement, vous pouvez soit créer plusieurs onglets, soit finir les requêtes par ; et utiliser Maj F5.

III. Extraction sur une seule table

Ouvrez dans DB Browser la base de données communes.
Observez la structure de cette base de données, le nombre de tables et le nom des champs ainsi que les types de données qui y sont présentes.

1. Extraction de lignes sans condition

La syntaxe est : `SELECT une_col,une_autre_col,... FROM une_table`.

Testez les requêtes suivantes. Observez bien le nombre d'enregistrements (de lignes) renvoyés.

- `SELECT code_dept FROM communes`
  affiche les valeurs présentes dans la colonne nom_comm de la table communes

	code_dept
1	29
2	30
3	30
4	24
5	88
6	30
7	12
8	34

- `SELECT nom_comm,code_dept FROM communes`
  affiche les valeurs présentes dans deux colonnes

	nom_comm	code_dept
1	SAINT-VOUGAY	29
2	ROUSSON	30
3	SAINT-FELIX-DE-PALLIERES	30
4	CENDRIEUX	24
5	TAINTRUX	88
6	LES MAGES	30
7	FONDAMENTE	12
8	LES RIVES	34
9	FLORAC	48
10	DAX	40

- `SELECT DISTINCT code_dept FROM communes`
  affiche les valeurs sans doublons (ici 96 départements)

	code_dept
1	29
2	30
3	24
4	88
5	12

- `SELECT * FROM communes`
  affiche la table complète (36613 enregistrements)

- `SELECT nom_comm, code_dept`
  `FROM communes`
  `ORDER By code_dept`
  les lignes seront triées par code de département croissant

	nom_comm	code_dept
1	BEAUPONT	01
2	PREVESSIN-MOENS	01
3	AMBERIEUX-EN-DOMBES	01
4	SAINT-LAURENT-SUR-SAONE	01
5	CHANOZ-CHATENAY	01
6	CURTAFOND	01
7	SAINT-ANDRE-SUR-VIEUX-JONC	01

- `SELECT nom_comm, code_dept`
  `FROM communes`
  `ORDER By nom_comm DESC`
  les lignes seront triées par nom de commune décroissant (ordre alphabétique)

	nom_comm	code_dept
1	ZUYTPEENE	59
2	ZUYDCOOTE	59
3	ZUTKERQUE	62
4	ZUDAUSQUES	62
5	ZUANI	2B
6	ZOZA	2A
7	ZOUFFTGEN	57
8	ZOUAFQUES	62

2. Extraction de lignes avec condition(s)

Il faut utiliser le mot clé WHERE suivi d'une condition.

- ```
  SELECT nom_comm FROM communes
  WHERE code_dept = 59
  ```
  affiche les noms des communes du Nord

	nom_comm
1	SAINT-HILAIRE-SUR-HELPE
2	FLERS-EN-ESCREBIEUX
3	WALLERS
4	MARCQ-EN-BAROEUL
5	ANZIN
6	MERVILLE
7	VILLENEUVE-D'ASCQ
8	LIESSIES
9	ECCLES

- ```
  SELECT nom_comm FROM communes
  WHERE population >= 200
  ```

	nom_comm
1	NANTES
2	MONTPELLIER
3	TOULOUSE
4	LILLE
5	BORDEAUX
6	RENNES
7	STRASBOURG
8	NICE
9	PARIS-15E-ARRONDISSEMENT
10	PARIS-18E-ARRONDISSEMENT

- ```
  SELECT nom_comm FROM communes
  WHERE nom_dept IN ("COTES-D'ARMOR",
        "FINISTERE","ILLE-ET-VILAINE","MORBIHAN")
  ```
  affiche les noms des communes de la région Bretagne

	nom_comm
1	SAINT-VOUGAY
2	PLOERMEL
3	CLEGUEREC
4	SAINT-THURIEN
5	TREMOREL
6	MAURE-DE-BRETAGNE
7	MESPAUL
8	LA CHAPELLE-JANSON
9	PLEMET

- ```
  SELECT nom_comm FROM communes
  WHERE nom_dept NOT IN ("COTES-D'ARMOR",
        "FINISTERE","ILLE-ET-VILAINE","MORBIHAN")
  ```
  affiche les noms des communes qui ne sont pas en Bretagne

- ```
  SELECT nom_comm,population FROM communes
  WHERE nom_comm LIKE "L%"
  ORDER BY code_dept
  ```
  affiche les noms des communes qui commencent par L
  et leur population, ces communes sont rangées par département

	nom_comm	population
1	LA CHAPELLE-DU-CHATELARD	0.3
2	LE GRAND-ABERGEMENT	0.1
3	LA BURBANCHE	0.1
4	LAGNIEU	6.7
5	LAIZ	1.1
6	LES NEYROLLES	0.7
7	LESCHEROUX	0.7
8	L'ABERGEMENT-CLEMENCIAT	0.8
9	LOYETTES	2.5

Plusieurs conditions peuvent être utilisées en utilisant les mots clés AND, OR ou NOT.
Attention dans ce cas à utiliser des parenthèses pour les conditions compliquées.

- ```
  SELECT * FROM communes
  WHERE code_dept = 59
        OR population > 100
  ```
  affiche les informations des communes du Nord
  ou de celles ayant plus de 100 (mille ?) habitants
  (renvoie aussi les communes vérifiant les deux conditions)

	id	code_comm	insee_com	nom_comm	statu
1	49	534	59534	SAINT-HILAIRE-SUR-HELPE	Commune si
2	419	109	44109	NANTES	Préfecture d
3	506	224	68224	MULHOUSE	Sous-préfect
4	668	112	75112	PARIS-12E-ARRONDISSEMENT	Chef-lieu car
5	711	234	59234	FLERS-EN-ESCREBIEUX	Commune si
6	713	632	59632	WALLERS	Commune si
7	800	378	59378	MARCQ-EN-BAROEUL	Chef-lieu car

- ```
  SELECT nom_comm,code_dept FROM communes
  WHERE code_dept = 59 OR
        (code_dept = 56 AND population > 5)
  ORDER BY code_dept
  ```
  affiche les communes qui sont dans le Nord ou
  celles du Morbihan ayant plus de 5 000 habitants (680 réponses)

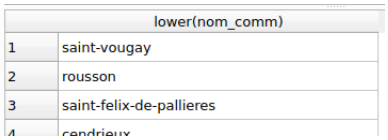
- ```
  SELECT nom_comm,code_dept FROM communes
  WHERE (code_dept = 59 OR code_dept = 56)
       AND population > 5
  ORDER BY code_dept
  ```
  affiche les communes qui sont dans le Nord ou dans le Morbihan
  et qui ont plus de 5 000 habitants (136 réponses)

- ```
  SELECT nom_comm,code_dept FROM communes
  WHERE NOT (code_dept = 59 OR code_dept = 56)
  ```
  affiche les communes qui ne sont pas dans le Nord ou dans le Morbihan

3. Modifications de l'affichage des résultats

Les fonctions upper(colonne) et lower(colonne) mettent en majuscules ou en minuscules.

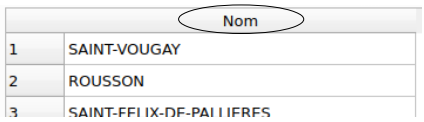
- ```
  SELECT lower(nom_comm) FROM communes
  ```
  affiche les noms de communes en miniscule



	lower(nom_comm)
1	saint-vougay
2	rousson
3	saint-felix-de-pallieres
4	cendrieux

Les alias modifient les noms des colonnes affichées.

- ```
  SELECT nom_comm AS Nom FROM communes
  ```
  le AS peut être omis :
- ```
  SELECT nom_comm Nom FROM communes
  ```



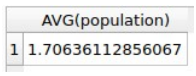
	Nom
1	SAINT-VOUGAY
2	ROUSSON
3	SAINT-FELIX-DE-PALLIERES

Les alias permettent aussi de réduire la taille des requêtes (voir IV).

4. Fonctions statistiques ou agrégatives

Ces fonctions ne peuvent être utilisées que dans une clause SELECT ou dans une clause HAVING. Elles renvoient un tableau d'une seule ligne.
Ces fonctions sont MAX(colonne), MIN(colonne), SUM(colonne), AVG(colonne), STD(colonne) et COUNT(colonne) (maximum, minimum, somme, moyenne, écart type, comptage).

- ```
  SELECT AVG(population) FROM communes
  WHERE code_dept = 59
  ```
  affiche la population moyenne des communes du Nord



	AVG(population)
1	1.70636112856067

- ```
  SELECT nom_comm,MAX(superficie) FROM communes
  ```
  affiche le nom et la superficie de la commune la plus étendue

- ```
  SELECT COUNT(*) FROM communes
  WHERE code_dept = 56
  ```
  affiche le nombre de communes du Morbihan (le nombre de lignes vérifiant la condition)
  ```

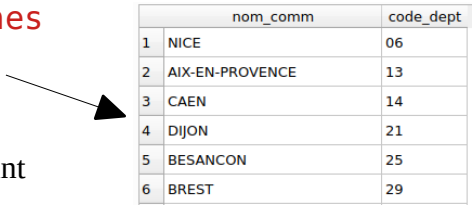
5. Extraction de données sur des groupes

La clause GROUP BY permet de regrouper les résultats d'une requête. Attention : pour chaque groupe, une seule ligne est affichée !

- ```
  SELECT nom_comm,code_dept FROM communes
  GROUP BY code_dept
  ```
  groupe les communes par département
  et renvoie une ligne pour chaque groupe (donc 96 lignes)

	nom_comm	code_dept
1	BEAUPONT	01
2	CHOUY	02
3	SAINT-POURCAIN-SUR-BESBRE	03
4	MANE	04
5	CHATEAUROUX-LES-ALPES	05

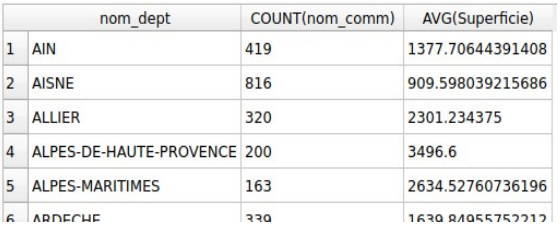
- ```
  SELECT nom_comm,code_dept FROM communes
  WHERE population > 100
  GROUP BY code_dept
  ```
  ne garde que les villes de plus de 100 000 habitants
  puis regroupe les résultats par département en n'affichant
  qu'une ville par département (35 résultats)



	nom_comm	code_dept
1	NICE	06
2	AIX-EN-PROVENCE	13
3	CAEN	14
4	DIJON	21
5	BESANCON	25
6	BREST	29

⚠ La clause GROUP BY peut donner des résultats peu pertinents si elle est mal utilisée (dans le premier exemple du 5., seule la commune de Beaupont est affichée pour le département de l'Ain). Il faut que les résultats demandés n'aient qu'une valeur par groupe. Par exemple :

- ```
  SELECT nom_dept,COUNT(nom_comm),
         AVG(Superficie)
  FROM communes
  GROUP BY nom_dept
  ```

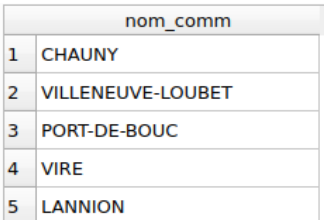


	nom_dept	COUNT(nom_comm)	AVG(Superficie)
1	AIN	419	1377.70644391408
2	AISNE	816	909.598039215686
3	ALLIER	320	2301.234375
4	ALPES-DE-HAUTE-PROVENCE	200	3496.6
5	ALPES-MARITIMES	163	2634.52760736196
6	ARDECHE	339	1639.84955752212

Ici, des groupes sont créés pour chaque nom de département donc, pour chaque groupe, chacun des termes `nom_dept`, `COUNT(nom_comm)` et `AVG(Superficie)` n'a qu'une valeur (le nom du département, le nombre de communes qui s'y trouvent et la superficie moyenne de ces communes).

La clause HAVING permet de ne conserver que les groupes vérifiant une certaine condition (à ne pas confondre avec WHERE qui sélectionne des lignes, pas des groupes).

- ```
  SELECT nom_comm FROM communes
  WHERE population > 10
  GROUP BY code_dept
  HAVING COUNT(*) > 5
  ```
  ne conserve que les groupes contenant plus de cinq résultats
  (donc les départements ayant plus de 5 villes de plus
  de 10000 habitants)



	nom_comm
1	CHAUNY
2	VILLENEUVE-LOUBET
3	PORT-DE-BOUC
4	VIRE
5	LANNION

IV. Extraction sur plusieurs tables

Téléchargez la base artiste.sqlite et ouvrez-là dans DB Browser for SQLite.

1. Produit cartésien

Dans l'onglet « Parcourir les données », jetez un œil sur les noms des artistes et sur les noms des œuvres ainsi que sur le libellé des champs contenant ces données.

Testez maintenant cette requête :

- `SELECT * FROM artiste,oeuvre`

Que pensez-vous des résultats affichés ?

⚠ Le produit cartésien donne des résultats dont la plupart ne sont pas utiles, il peut de plus fortement ralentir un serveur (si les deux tables interrogées font respectivement 1000 et 5000 lignes alors leur produit fera 5 000 000 lignes) !

2. Noms de colonnes ambigus

Testez maintenant cette requête :

- `SELECT idO FROM avoir,oeuvre`

Le logiciel voit une ambiguïté : il y a une colonne idO dans chacune des tables avoir et oeuvre : il ne sait pas laquelle choisir !

Pour préciser que nous voulons, par exemple, les valeurs de la colonne idO contenues dans la table avoir il faut *préfixer* le nom de colonne par le nom de la table :

- `SELECT avoir.idO FROM avoir,oeuvre`

Ceci n'est pas nécessaire quand il n'existe pas deux colonnes ayant le même nom dans la base de données mais c'est quand même recommandé quand on interroge plusieurs tables.

3. Jointures

Pour interroger plusieurs tables, il faut utiliser les mots clés JOIN … ON …

Cette commande :

- `SELECT * FROM artiste JOIN oeuvre`

fait encore un produit cartésien des deux tables. Pour supprimer les lignes non pertinentes, il faut donner la relation existant entre une colonne de la première et une colonne de la seconde après ON :

- `SELECT artiste.nom,par.idO FROM artiste JOIN par ON artiste.idA = par.idA`
  permet de récupérer les noms des artistes et les identifiants d'œuvres qu'ils ont faîtes

	Nom	idO
1	Courbet	0
2	Leonard de Vinci	1
3	Manet	2
4	Carpeaux	3
5	Carpeaux	4

- ```
  SELECT artiste.nom,oeuvre.titre
  FROM artiste
  JOIN par ON artiste.idA = par.idA
  JOIN oeuvre ON oeuvre.idO = par.idO
  ```
  permet de récupérer les noms des artistes et les titres d'œuvres qu'ils ont faîtes (la table par sert ici de passerelle entre les artistes et les œuvres)

	Nom	Titre
1	Courbet	L'origine du monde
2	Leonard de Vinci	La joconde
3	Manet	Le dejeuner sur l'herbe
4	Carpeaux	La danse
5	Carpeaux	Femme nue couchee sur le ventre

La syntaxe générale d'une jointure est :

```
SELECT table1.col1, table2.col2, ... FROM table1
JOIN table2 ON table1.une_col = table2.col_correspondante
[JOIN table3 ON …]
[JOIN …]
[WHERE condition]
[GROUP BY une_autre_col]
[HAVING autre_condition]
[ORDER BY encore_une_autre_col]
```

Pensez au schéma relationnel pour trouver les relations entre les tables (les clés étrangères).

- ```
  SELECT oeuvre.titre
  FROM artiste
  JOIN par ON artiste.idA = par.idA
  JOIN oeuvre ON oeuvre.idO = par.idO
  WHERE artiste.nom = "Veronese"
  ```
  donne les œuvres de Véronese

	Titre
1	Les noces de Cana
2	Le repas chez Levi

Voici la même requête mais en utilisant des alias pour les noms de tables et de colonne :

- ```
  SELECT O.titre AS "Oeuvres de Véronese"
  FROM artiste AS A
  JOIN par AS P ON A.idA = P.idA
  JOIN oeuvre AS O ON O.idO = P.idO
  WHERE A.nom = "Veronese"
  ```

	Oeuvres de Véronese
1	Les noces de Cana
2	Le repas chez Levi

Remarque : il existe une syntaxe sans JOIN et ON, qui consiste à faire un produit cartésien puis à ne garder que les lignes pertinentes. En dehors du fait qu'il faut que le logiciel soit optimisé pour éviter de vraiment faire un produit cartésien, cette syntaxe n'est plus conseillée depuis 1992.
  ```