TP 22
Langage SQL

Deuxième TP sur les bases de données, nous approfondissons le langage SQL avec comme support une base de données des aéroports et des régions du monde. Bienvenue à bord, veuillez attacher vos ceintures !

Démarrage

Le logiciel **DB Browser for SQLite** (*explorateur de base de données pour le langage SQLite*, qui est *une* version du SQL) permet d'ouvrir une base de données, visualiser le contenu (onglet « Parcourir les données ») et exécuter des requêtes SQL dessus (onglet « Exécuter le SQL »). Récupérer le fichier matériel, le décompresser, puis depuis le logiciel « Ouvrir une base de données en lecture seule » et aller ouvrir le fichier `airports.db`.
On peut écrire plusieurs requêtes SQL dans la zone de texte, en les terminant par un point-virgule `;` On peut alors demander à n'exécuter que la requête courante (icône ressemblant à un play-pause).
Contrairement au langage Python, les indentations et sauts de ligne n'ont pas d'importance et permettent seulement de rendre le code plus lisible. Les commentaires sont contenus entre les deux symboles : `/* commentaire */`.
La base de données contient trois tables : `airports` avec tous les aéroports, `countries` avec tous les pays, et `regions` avec toutes les régions de tous les pays.

Exercice 1. Explorer manuellement les tables. Quelles sont les clés primaires et étrangères ici ?

I Requêtes simples

On s'intéresse d'abord aux requêtes permettant d'aller chercher des informations **dans une seule table**. La syntaxe la plus simple, la plus courante, est :

```
SELECT attributs
FROM table
WHERE condition
```

où

- ***attributs*** désigne un attribut de la table (une colonne), ou plusieurs séparés par des virgules, que l'on veut sélectionner, ou le caractère `*` pour sélectionner tous les attributs d'un coup,
- ***table*** est le nom de la table dans laquelle on sélectionne,
- ***condition*** est une condition exprimée sur les attributs.

Les attributs ont un type de donnée : entier, flottant, chaine de caractère (entourées préférablement par des apostrophes simples `'texte'`). Il existe éventuellement d'autres types, permettant par exemple de stocker facilement une date et de comparer rapidement les données selon l'ordre chronologique... Il est important aussi d'avoir une valeur spéciale `NULL` indiquant l'absence de donnée.
La condition peut être formulée avec :

- Des comparaisons mathématiques comme d'habitude `>`, `<`, `>=`, `<=`, `=`, `<>` (mais `==` fonctionnera aussi pour l'égalité et `!=` aussi pour la différence),
- Les opérations arithmétiques habituelles `+`, `-`, `*`, `/`,
- On peut combiner les conditions avec les mots-clés `AND`, `OR`, `NOT`, et utiliser des parenthèses.

Exercice 2. Écrire des requêtes SQL pour :

1. Sélectionner le nom de tous les pays.
2. Sélectionner le nom de tous les pays d'Europe.
3. Sélectionner toutes les bases d'hydravion (`seaplane_base`) du monde.
4. Sélectionner toutes les bases d'hydravion en Europe.
5. Sélectionner le nom et les villes des aéroports en France qui sont gros ou moyens.

6. Sélectionner tous les aéroports qui sont en-dessous du niveau de la mer.

Remarque 1.
1. On peut toujours renommer un attribut ou une table avec le mot-clé `AS` :

```
SELECT attributs AS nouveau_nom
```

ou

```
FROM table AS nouveau_nom
```

2. Il se peut qu'après une requête, une entrée apparaisse plusieurs fois. Pour éliminer les doublons, on utilise le mot-clé `SELECT DISTINCT` à la place de `SELECT`.
3. On peut renvoyer le résultat classé selon un attribut, qui est soit un nombre (ordre usuel) soit une chaine de caractères (ordre lexicographique = ordre du dictionnaire) avec le mot-clé `ORDER BY`, auquel on peut aussi rajouter `ASC` (*ascending*, croissant) ou `DESC` (*descending*, décroissant) :

```
SELECT attributs
FROM table
WHERE condition
ORDER BY critère
```

ou à la fin

```
ORDER BY critère ASC
```

ou

```
ORDER BY critère DESC
```

Exercice 3. Écrire des requêtes SQL pour :

1. Afficher les aéroports français moyens ou gros, classés du nord au sud.
2. Afficher les villes des gros aéroports des États-Unis, classés d'ouest en est.
3. (a) Afficher les noms et villes des héliports français, situés à plus de 1000 m d'altitude, sachant que 1 ft = 0,3048 m (sans utiliser sa calculatrice avant).
   (b) ... classés par altitude.
   (c) ... pouvez-vous afficher directement l'altitude en mètres ?
4. Afficher les codes ISO des pays africains contenant un gros aéroport. Sans doublon.
5. (a) Afficher les régions allemandes et leur code ISO, classées par ordre alphabétique.
   (b) Afficher tous les villes de la région Rhénanie-du-Nord-Westphalie contenant un aéroport.

Remarque 2. Le saviez-vous ? La région Rhénanie-du-Nord-Westphalie est l'une des plus importantes d'Allemagne. La plus peuplée (17,9 million d'habitants), la plus importante économiquement, l'une des plus riches (avec la Bavière). Elle comprend de nombreuses villes très peuplées formant une énorme agglomération Rhin-Ruhr : Bonn, Cologne, Düsseldorf, Duisburg le long du Rhin, et Essen, Bochum, Dortmund, Wuppertal dans la vallée de la Ruhr. La capitale est Düsseldorf, où se trouve aussi un aéroport international, le troisième aéroport allemand après Frankfurt et Munich.

II Les fonctions d'agrégation

Ce sont les fonctions qui calculent un résultat à partir de **toute une colonne**. Celles utilisées sont les suivantes :
- `MIN()` : minimum,
- `MAX()` : maximum,
- `SUM()` : somme,
- `COUNT()` : nombre d'entrées,
- `AVG()` : moyenne (en anglais *average*)

Elles s'utilisent **dans** `SELECT`, souvent en leur donnant un nom avec `AS`.

Exemple 1. Si on veut connaitre le nombre total de pays, alors la bonne commande est :

```
SELECT COUNT(id) AS nombre
FROM countries
```

Bien sûr, on peut combiner avec un `WHERE`...
Le résultat est un nombre seul — donc une table avec un seul élément !

Exercice 4. Écrire des requêtes SQL pour :

1. Donner l'élévation moyenne de tous les aéroports aux Pays-Bas (*Netherland...*)
2. Même question pour la Suisse (*Confédération Helvétique...*)
3. Donner le nombre de gros aéroports en Europe.
4. Donner le nombre de régions allemandes.

Remarque 3.
1. Pour `COUNT`, si toutes les entrées ont tous leurs attributs bien remplis, on peut utiliser `COUNT(*)` car l'attribut choisi pour compter les entrées n'importe pas...
2. Il est donc important que les valeurs éventuellement manquantes soient `NULL`, ainsi ces fonctions peuvent ne pas en tenir compte. La moyenne n'est bien sûr pas la même si on ignore les valeurs manquantes que si on les complète par 0 ! Et `COUNT` compte les entrées non `NULL`.
3. Il se peut que l'on souhaite calculer les fonctions d'agrégation non pas sur *toute* la colonne, mais en regroupant entre elles les entrées selon certains critères (voir l'exemple juste après). On utilise alors le mot-clé `GROUP BY` *`attribut`*, qui se place après `WHERE` mais avant `ORDER BY`.
4. Après un `GROUP BY`, on peut vouloir ne garder que les entrées satisfaisant une certaine condition qui est elle-même le résultat d'une fonction d'agrégation. On utilise alors la commande `HAVING` *`condition`*, qui se place juste après le `GROUP BY`. Ce n'est pas tout à fait la même chose que `WHERE` car `WHERE` agit *avant* la fonction d'agrégation, pour sélectionner seulement ce qui nous intéresse et calculer la fonction dessus, alors que `HAVING` agit *après le calcul*.

Exemple 2. Pour ces deux derniers cas, la commande suivante permet de donner le nombre total de pays par continents :

```
SELECT COUNT(id) AS nombre, continent
FROM countries
GROUP BY continent
```

et si on veut garder seulement les continents avec plus de 20 pays :

```
SELECT COUNT(id) AS nombre, continent
FROM countries
GROUP BY continent
HAVING nombre >= 20
```

Exercice 5. Écrire des requêtes SQL pour :

1. Donner le nombre de régions par pays d'Europe.
2. Donner l'élévation moyenne par pays des aéroports gros ou moyens d'Amérique du Sud.
3. Donner les villes de France ayant au moins trois aéroports.
4. Donner les villes d'Europe ayant plusieurs aéroports gros ou moyens.

III Sélection dans plusieurs tables

Parfois, il faut recouper les informations qui sont contenues à travers plusieurs tables. C'est le concept de **jointure**. La semaine dernière il était exprimé avec la commande `FROM` suivi d'une liste de tables ; ce n'est pas la commande la plus claire (la jointure possède de nombreuses subtilités)... On préfèrera utiliser le mot-clé `JOIN` :

```
SELECT attributs
FROM table
JOIN autre_table
ON critère
```

à la suite de quoi on peut bien entendu rajouter les `WHERE`, `ORDER BY`, les agrégations et autres. Souvent le ***critère*** est simplement l'égalité entre un attribut de la première table et un attribut de la seconde.
Par exemple, chaque aéroport est attaché à une région, par un code appelé `iso_region` dans la table `airports`. Les régions se trouvent elles dans une autre table appelée `regions`, dont le code est simplement donné par l'attribut `code`. Pour obtenir des informations simultanément sur les aéroports et leur région, il faut utiliser une jointure. Observez le résultat de la commande suivante :

```
SELECT * FROM airports
JOIN regions ON airports.iso_region = regions.code
WHERE airports.iso_country='FR' AND airports.type='large_airport'
```

Ici apparait la syntaxe `table.attribut` permettant de bien distinguer les attributs appartenant à chaque table... On peut s'en passer en utilisant le renommage `AS`.
Le résultat de la commande précédente serait plus lisible si on sélectionnait seulement ce qui nous intéresse :

```
SELECT airports.name, airports.municipality, regions.name FROM airports
JOIN regions ON airports.iso_region = regions.code
WHERE airports.iso_country='FR' AND airports.type='large_airport'
```

Exercice 6. Écrire des requêtes SQL pour :

1. Donner la liste des gros aéroports européens et leur région.
2. Donner les gros aéroports d'Asie et leur pays, classés d'ouest en est.
3. Un voyageur a un billet pour SCL. Dans quelle ville et quel pays se rend-il ?
4. Un voyageur a un billet pour DPS. Dans quel pays et quelle région se rend-il ?
5. (a) Donner le classement des pays avec le plus de codes IATA.
   (b) ... en ne gardant que ceux avec plus de 100 codes.
6. Donner le classement des pays du monde dont l'altitude moyenne des aéroports est la plus élevée.

Annexe : quelques compléments

1. Nous nous sommes intéressé au minimum vital pour effectuer une requête SQL. Le langage permet également d'*insérer* des entrées dans une base de données (mot-clé `INSERT`), de créer des tables (`CREATE TABLE`), bref, de gérer en entier une base de données à travers le langage. Tout cela est bien entendu un monde à part entière, quand il s'agit de gérer la base de données d'un réseau social avec des tables d'utilisateurs, de publications, de vues et de likes...
2. Il existe divers logiciels qui interprètent le langage SQL, avec des variantes de dialecte. D'où quelques problèmes lorsqu'on veut l'utiliser sur des logiciels ou en ligne, et plusieurs syntaxes possibles pour les commandes. Mais ce genre de détail s'apprend lorsque l'on en a vraiment besoin !
3. En général le SQL tout seul est peu utile, car on ne va pas insérer les entrées de la table unes par unes avec des requêtes SQL... Il est utilisé conjointement avec d'autres langages ou d'autres programmes pour en récupérer les données, faire des calculs dessus ou les représenter graphiquement. Il y a des fonctions Python permettant d'ouvrir une base de données et de lancer des requêtes SQL dessus, puis récupérer les résultats dans des listes Python. Les pages internet utilisent aussi du SQL côté serveur pour aller chercher par exemple les informations de l'utilisateur dans leur propre base de données.
4. Il faut garder en tête que les bases de données peuvent contenir des millions d'entrées, ainsi chaque requête demande beaucoup de travail à l'ordinateur. Le but de jeu est toujours d'écrire le minimum possible de requêtes, et que chaque requête soit la plus efficace possible. Les fonctions d'agrégations fournies par le langage SQL sont aussi bien plus efficaces, quand on les utilise correctement, que de récupérer les données *puis* de faire les calculs dessus.
5. Il faut garder en tête qu'une table **n'est pas** un tableau avec ses colonnes fixées et un ensemble de lignes. D'une part le résultat de chaque requête `SELECT` **est** une table, et les requêtes peuvent s'emboîter entre elles. Et surtout, dans une base de données les tables peuvent se faire référence les unes aux autres, de façon très élaborée, un attribut d'une table contenant un numéro ID fait référence aux données d'une autre table

et ainsi de suite. La phase de conception de la base de données et de l'ensemble des tables est d'ailleurs un problème à part entière pour les ingénieurs (on parle d'architecture des bases de données). Il faut mieux penser la base de données en terme de **relations** : quelles sont les données à enregistrer et lesquelles on veut mettre en relation entre elles ? De ce point de vue le formalisme mathématiques est intéressant. Une entrée dans une table signifie simplement, que les données présentes sont en relation, et le langage SQL permet d'agir sur les relations.

Ressources :

— Cours de SQL à la fois bien complet et interactif (mais en anglais) : https://www.w3schools.com/sql/
— Exercices interactifs (en anglais) : https://sqlzoo.net/wiki/SQL_Tutorial
— Excellent cours, exercices interactifs : https://cpge-itc.github.io/bcpst2/6_sql/sql.html