TP n°32 - Bases de données - Requêtes SQL

Vocabulaire à connaître : diagramme sagittal, requête, clause, relation (= table), enregistrement (=ligne), attribut (=colonne), schéma d'une relation, type et domaine d'un attribut, clé primaire, clé étrangère, 4 types d'associations (1-1, 1-*, *-1, *-*), jointure (produit cartésien + filtrage), opérations ensemblistes, fonctions d'agrégation.

Requêtes basiques de sélection d'enregistrements :

```
SELECT [DISTINCT] att_1 [AS nom_affichage1], att_2...   -- ATTRIBUTS DE CHAQUE ENREGISTREMENT RESULTAT,
                                                        -- QUE L'ON SOUHAITE AFFICHER.
                                                        --  * SI ON LES VEUT TOUS
                                                        -- AS, optionnel, permet de choisir le nom de la colonne associé à
                                                        -- cet attribut lors de l'affichage du résultat
FROM tabl [alias]                -- TABLE DANS LAQUELLE DOIVENT ETRE RECHERCHES LES ENREGISTREMENTS
           -- alias nom court facultatif pour désigner la table. alias.champ pour désigner un attribut de cette table
[WHERE condition]                   -- FILTRAGE/PROJECTION DES ENREGISTREMENTS SOUS CONDITION(S)
                                        -- OPERATEURS POUR LES CONDITIONS
                                            -- OPERATEURS D'EXISTENCE IS NULL, IS NOT NULL
                                            -- OPERATEURS DE COMPARAISON = <>, >=, <=, >, <
                                            --      avec relations d'ordres usuelles (lexicographiques sur les chaines)
                                            -- OPERATEURS LOGIQUES AND OR NOT POUR COMBINER DES CONDITIONS
[ORDER BY att_k ASC | DESC]       -- AFFICHAGE TRIE DES ENREGISTREMENTS RESULTATS SELON L'ATTRIBUT att_k, ascendant ou descendant
[LIMIT nombre_max_enreg ]         -- LIMITER LE NOMBRE D'ENREGISTREMENT RESULTATS AFFICHES
[OFFSET k];                       -- DEMARRER L'AFFICHAGE A PARTIR DE L'ENREGISTREMENT RESULTAT k
```

`DISTINCT` : clause facultative. Enlève les enregistrements (lignes) résultats identiques, s'il y en a.

Jointure de deux tables (produit cartésien + filtrage par une condition) :

```
SELECT att1, att_2...
FROM tabl_1 [alias1]
                                    -- DANS LA TABLE 1
JOIN tabl_2 [alias2]                -- PRODUIT CARTESIEN tabl_1×tabl_2
ON condition_jointure;              -- FILTRAGE DU PRODUIT CARTESIEN PAR UNE CONDITION DE JOINTURE
WHERE condition                     -- FILTRAGE/PROJECTION AVEC CONDITIONS SUR LES ENREGISTREMENTS SELECTIONNES

-------------------------
SELECT att1, att_2...
FROM tabl_1 [alias1]
                                    -- DANS LA TABLE 1
LEFT JOIN tabl_2 [alias2]           -- PRODUIT CARTESIEN tabl_1×tabl_2.
                                    -- LEFT JOIN: le filtrage garde l'enregistrement dans le cas où l'attribut
                                    -- servant pour le filtrage dans la table 2 est NULL
ON condition_jointure;              -- FILTRAGE DU PRODUIT CARTESIEN PAR UNE CONDITION DE JOINTURE
WHERE condition                     -- FILTRAGE/PROJECTION AVEC CONDITIONS SUR LES ENREGISTREMENTS SELECTIONNES

-------------------------
SELECT att1, att_2...
FROM tabl_1 [alias1]                -- DANS LA TABLE 1
NATURAL JOIN tabl_2 [alias2]        -- PRODUIT CARTESIEN tabl_1×tabl_2 + FILTRAGE PAR JOINTURE NATURELLE
                                    -- cette opération se base sur la présence d'attributs de même nom
                                    -- dans les deux tables
                                    -- seuls les enregistrement ayant la même valeur dans les 2 tables
                                    -- pour ces attributs homonymes sont selectionnés
WHERE condition                     -- FILTRAGE/PROJECTION AVEC CONDITIONS SUR LES ENREGISTREMENTS SELECTIONNES
```

`LEFT JOIN` : l'enregistrement est conservé même si le champ de la table *jointée* utilisé pour la condition de jointure n'existe pas.
`NATURAL JOIN` : la jointure se fait en ne conservant que les enregistrements du produit cartésien pour lesquels les champs qui ont le même nom dans les deux tables sont égaux.

Fonctions d'agrégation, constitution des agrégats sur lesquelles elles sont calculées avec `GROUP BY`, filtrage des enregistrements avec conditions sur les groupes avec `HAVING` :

```
SELECT AGR1(att_1),AGR2(att_2), att_3... -- ATTRIBUTS OU RESULTATS D'AGREGAT DE CHAQUE ENREGISTREMENT RESULTAT,
                                        -- QUE L'ON SOUHAITE AFFICHER.
                                        -- FONCTIONS D'AGREGAT: AVG, MAX, MIN, COUNT, SUM
FROM tabl [alias]                      -- TABLE DANS LAQUELLE SONT RECHERCHES LES ENREGISTREMENTS RESULTATS
[WHERE condition]                      -- FILTRAGE/PROJECTION AVEC CONDITIONS SUR LES ENREGISTREMENTS SELECTIONNES
GROUP BY att_k                         -- CRITERE DE REGROUPEMENT DES ENREGISTREMENT POUR LE CALCUL DES AGREGATS
HAVING condition                       -- FILTRAGE AVEC CONDITION SUR LES GROUPES D'ENREGISTREMENTS SELECTIONNES
[ORDER BY attribut_k ASC | DESC];      -- AFFICHAGE TRIE DES ENREGISTREMENTS RESULTATS
```

Opérations ensemblistes sur deux tables ayant le même schéma.

```
SELECT *
FROM tabl_1
UNION|INTERSECT|EXCEPT -- OPERATIONS ENSEMBLISTES. LES TABLES tabl_1 et tabl_2 DOIVENT AVOIR LE MEME SCHEMA
SELECT *
FROM tabl_2
```

On peut bien sûr utiliser conjointement ces différentes clauses, par exemple utiliser des fonctions d'agrégation dans des requêtes avec jointure, puis trier les affichages. On peut également imbriquer les requêtes SQL, par exemple pour utiliser le résultat d'une agrégation dans une condition de jointure...

Pour ce TP, nous utiliserons l'interface web SQL disponible à l'adresse suivante :

https://tchou.github.io/sqlsb/

Enregistrer régulièrement l'ensemble de votre historique de requête dans un fichier en utilisant le bouton `SQL` en haut de l'interface. Vous pouvez également faire des copier-coller de l'éditeur Web dans un fichier sur votre machine pour être certains de conserver une trace de votre travail.

Exercice 1 (Base de données du territoire administratif français - Partie 1).

On s'intéresse dans cet exercice à la base de données (BDD) `bd_villes.sqlite` mise à disposition sur Moodle.

1. Charger cette base de données dans l'interface. Analyser les différentes tables présentes et leurs schémas, puis dessiner un **diagramme sagittal** semblable à celui proposé dans le cours pour la base de données de cinéma. Sous chaque **entité**, vous préciserez son schéma (nom, type, et domaine des attributs). Vous soulignerez en trait plein les clés primaires, en pointillés les attributs correspondant à des clés étrangères. Vous représenterez les **associations** entre les entités en précisant bien la nature de ces associations (1-1, 1-*, *-1).
2. Écrire une requête qui renvoie la table de toutes les communes avec pour chacune, sa population, l'identificateur de son département, l'identificateur de sa région.
3. Donner la table des villes de plus de 100 000 habitants en précisant leur population et leur région.
4. Trier cette table par ordre croissant, puis décroissant de la population.
5. **Avec ou sans jointure ?**
   a. Identifier le code région du Limousin dans la table `regions`.
   Écrire une requête qui renvoie le nombre de communes de la région Limousin, sans jointure. Quel est le principal inconvénient pratique de cette requête ?
   b. Écrire une requête qui renvoie le même résultat en utilisant une jointure.
   c. Que se passe-t-il si on omet la clause `ON` dans la requête précédente ?
   d. Qu'observe-t-on si la clause de sélection est `SELECT *` ?
6. **Jointure sans doublons**
   a. Que fait la requête suivante ?
   ```
   SELECT C.nom_commune, D.nom_departement
   FROM communes C
   JOIN departements D
   ON C.id_departement = D.id_departement
   ```
   b. Écrire une requête qui renvoie la table de toutes les communes avec pour chacune, sa population, le nom de son département, le nom de sa région.
   c. Que fait la requête suivante ?
   ```
   SELECT COUNT(communes.nom_commune)
   FROM communes
   NATURAL JOIN regions
   WHERE regions.nom_region = 'Limousin'
   ```
   d. Qu'observe-t-on si la clause de sélection est `SELECT *` ?
   e. Modifier cette requête pour qu'elle renvoie les noms des communes de la région Aquitaine et les populations associées. Afficher le résultat par populations décroissantes puis croissantes.

Exercice 2 (Base de données du territoire administratif français - Partie 2 - Fonctions d'agrégations).

1. Écrire une requête qui renvoie la population moyenne des communes du Limousin. Modifier cette requête pour qu'elle affiche la population totale du Limousin.
2. Que fait la requête suivante ?
```
SELECT id_region, AVG(population), SUM(population)
FROM communes
GROUP BY id_region
```
Que se passe-t-il si on supprime la clause `GROUP BY` ? Quel inconvénient majeur cette requête présente-t-elle ?
3. Écrire une requête qui renvoie le nom de toutes les régions, leurs populations moyennes et leurs populations totales.
4. Écrire une requête qui renvoie le nom et le nombre de communes de toutes les régions possédant plus de 2000 communes. Quelle différence faites vous entre les clauses `HAVING` et `WHERE` ?
5. Écrire une requête qui, parmi les communes du Limousin, renvoie, par ordre croissant de population, celles dont la population est inférieure à la population moyenne de cette région. Le résultat doit également afficher la population de chaque commune. On pourra **imbriquer des requêtes SQL**.
6. Quelle proportion de communes du Limousin satisfont le critère précédent ? Pour vous aider dans les calculs, la fonction `ROUND(x,y)` retourne la valeur approchée de `x` arrondie à `y` chiffres après la virgule. La fonction `CAST(x AS real)` transforme `x` en un flottant.
7. Analyser la requête suivante. Quel est l'effet des clauses `AS` ?
```
SELECT nom_commune AS 'Commune', population AS 'Population',
(ROUND(100*CAST(population AS real)/
(SELECT SUM(population)
FROM communes NATURAL JOIN regions
WHERE nom_region = 'Limousin'),2)) AS 'Pourcentage'
FROM communes NATURAL JOIN regions
WHERE nom_region ='Limousin'
ORDER BY population DESC
```

Exercice 3 (Base de données géographique mondiale).

On s'intéresse dans cet exercice à la base de données (BDD) `bd_mondial.sqlite` mise à disposition sur Moodle.

1. Charger la base de données `bd_mondial.sqlite`. Analyser le contenu des schémas des différentes tables. Pouvez-vous deviner à quoi sert le mot clé `CONSTRAINT` ?
2. **Requêtes basiques**
   - **a.** Écrire une requête qui renvoie la table des pays dont la population excède 60 millions d'habitants.
   - **b.** Écrire une requête qui renvoie la même table triée par ordre alphabétique.
   - **c.** Écrire une requête qui renvoie la même table triée par ordre décroissant de population.
   - **d.** Écrire une requête qui renvoie le nom des dix plus petits pays en superficie.
   - **e.** Écrire une requête qui renvoie le nom des dix suivants.
3. **Jointures**
   - **a.** Observez la relation `encompasses`. Quelle information contient-elle ? A quelle(s) autres tables est-elle reliée ? De quelle manière ?
   - **b.** Écrire une requête qui renvoie le nom des pays qui sont à cheval sur plusieurs continents.
   - **c.** Écrire une requête qui renvoie les pays du continent américain qui comptent moins de 10 habitants par km^2.
   - **d.** Déterminer les capitales européennes situées à une latitude supérieure à 60°.
4. **Fonctions d'agrégations**
   - **a.** Quelles sont les dix langues parlées dans le plus de pays différents ?
   - **b.** Quelles sont les cinq langues les plus parlées au monde, présentes dans la base ? Préciser pour chacune d'elles le nombre de personnes qui la parlent.
   - **c.** Dans quel pays partiellement francophone la langue française est-elle la plus minoritaire ?