C2 Langage SQL

Memento `SQL`

Memento - SQL

Ce document n'a pas la vocation d'être exhaustif. On pourra se référer à ce site afin de le compléter et avoir plus de détails.

Dans certains *dialectes* `SQL`, les points-virgules en fin de ligne sont indispensables même s'il n'y a qu'une seule instruction. Bien que n'utilisions ici que des instructions uniques, on écrit tout de même les points-virgules en fin de ligne.

1. Liens directs :

- Création d'une base de données
- Création de tables
- Afficher toute une table
- Effacer une base, une table
- Insertion de valeurs
- Suppression de valeurs
- Mise à jour de valeurs
- Sélections

2. Création d'une base de données

- La base n'existe pas :

  🗗 **Requête SQL**

  ```
  CREATE DATABASE ma_base;
  ```

- Au cas où la base existe et que l'on ne souhaite pas l'écraser :

  🗗 **Requête SQL**

  ```
  CREATE DATABASE IF NOT EXISTS ma_base;
  ```

3. Création de tables

- Cas de base :

Requête SQL

```
CREATE TABLE ma_table (
    attribut_1 INTEGER PRIMARY KEY AUTOINCREMENT,
    attribut_2 TEXT,
    attribut_3 VARCHAR(50)
);
```

Là encore on peut ajouter l'argument `IF NOT EXISTS`.

- Clé primaire multiple :

Requête SQL

```
CREATE TABLE ma_table (
    attribut_1 INTEGER,
    attribut_2 TEXT,
    attribut_3 VARCHAR(50),
    PRIMARY KEY (attribut_1, attribut_2)
);
```

- Clé étrangère :

Requête SQL

```
CREATE TABLE ma_table (
    attribut_1 INTEGER PRIMARY KEY AUTOINCREMENT,
    attribut_2 TEXT,
    attribut_3 VARCHAR(50),
    FOREIGN KEY (attribut_1) REFERENCES autre_table (attribut_de_reference)
);
```

4. Afficher toute une table

- On effectue une requête :

Requête SQL

```
SELECT *
FROM ma_table;
```

5. Effacer une base, une table

- Une base :

❐ **Requête SQL**

```
DROP DATABASE ma_base;
```

- Une table :

❐ **Requête SQL**

```
DROP TABLE ma_table;
```

6. Insertion de valeurs

- On renseigne touts les attributs dans l'ordre de leur création :

❐ **Requête SQL**

```
INSERT INTO ma_table
VALUES (valeur_1, valeur_2, valeur_3);
```

- On ne renseigne que les attributs cités dans l'ordre souhaité :

❐ **Requête SQL**

```
INSERT INTO ma_table (attribut_3, attribut_1)
VALUES (valeur_3, valeur_1);
```

- Insertion de plusieurs lignes :

❐ **Requête SQL**

```
INSERT INTO ma_table (attribut_3, attribut_1)
VALUES
(valeur_3_1, valeur_1_1),
(valeur_3_2, valeur_1_2),
(valeur_3_3, valeur_1_3);
```

7. Suppression de valeurs

- On supprime des entrées, des lignes, en précisant une condition (ici les entrées dont l'`attribut_1` vaut `8`) :

❐ **Requête SQL**

```
DELETE FROM ma_table
WHERE attribut_1 = 8;
```

8. Mise à jour de valeurs

- Modification de la valeur pour **toutes** les lignes d'une table :

  🗗 **Requête SQL**

  ```
  UPDATE ma_table
  SET attribut_1 = valeur_1;
  ```

- Modification de la valeur en posant une condition :

  🗗 **Requête SQL**

  ```
  UPDATE ma_table
  SET attribut_1 = valeur_1
  WHERE condition;
  ```

9. Sélections

9.1. Cas de base

- Tous les attributs :

  🗗 **Requête SQL**

  ```
  SELECT *
  FROM ma_table;
  ```

- Seulement certains attributs :

  🗗 **Requête SQL**

  ```
  SELECT attribut_1, attribut_3
  FROM ma_table;
  ```

- Avec une condition :

  🗗 **Requête SQL**

  ```
  SELECT attribut_1, attribut_3
  FROM ma_table
  WHERE attribut_2 = valeur_2;
  ```

  On peut utiliser les opérateurs `=` (attention, un seul symbole contrairement au `==` de Python), `<>` (plusieurs systèmes de gestion de BDD acceptent aussi `!=`), `>`, `<`, `>=`, `<=`, `AND`, `OR`.

- Les chaînes de caractères :

  L'usage veut que l'on délimite les chaînes de caractères par des guillemets simples : `'chaine'`. Les guillemets doubles sont réservés aux noms de tables, d'attributs : `"attribut_1" = 'chaine'`. Dans les faits, **cela ne change**

pas grand chose !

En cas de stricte égalité :

🗗 **Requête SQL**

```
SELECT attribut_1, attribut_3
FROM ma_table
WHERE attribut_2 = 'chat'
```

Si l'on cherche les chaînes débutant par `'chat'` (`chat`, `chatte`, `chaton`...):

🗗 **Requête SQL**

```
SELECT attribut_1, attribut_3
FROM ma_table
WHERE attribut_2 LIKE 'chat%'
```

Si l'on cherche les chaînes se terminant par `'chat'` (`achat`, `le chat`...):

🗗 **Requête SQL**

```
SELECT attribut_1, attribut_3
FROM ma_table
WHERE attribut_2 LIKE '%chat'
```

9.2. Fonctions d'agrégation

- Compter les lignes vérifiant une condition :

🗗 **Requête SQL**

```
SELECT COUNT(*)
FROM ma_table
WHERE condition;
```

- Regrouper toutes les lignes selon la valeur d'un attribut :

🗗 **Requête SQL**

```
SELECT COUNT(*)
FROM ma_table
WHERE condition
GROUP BY attribut_1;
```

- La maximum d'un attribut :

🗗 **Requête SQL**

```
SELECT MAX(attribut_1)
FROM ma_table
WHERE condition;
```

Il existe aussi la fonction `MIN`.

- La somme d'un attribut :

Requête SQL

```
SELECT SUM(attribut_1)
FROM ma_table
WHERE condition;
```

- La moyenne d'un attribut :

Requête SQL

```
SELECT AVG(attribut_1)
FROM ma_table
WHERE condition;
```

- Trier les données :

Requête SQL

```
SELECT attribut_1
FROM ma_table
WHERE condition
ORDER BY attribut_2 ASC;
```

`ASC` pour `ASCENDING` et l'ordre croissant. Utiliser `DESC` pour l'ordre décroissant.

- N'afficher que les 10 premiers résultats :

Requête SQL

```
SELECT attribut_1
FROM ma_table
WHERE condition
LIMIT 10;
```

9.3. Jointures

- Mettre en correspondance plusieurs tables grâce aux clés étrangères :

Requête SQL

```
SELECT table_1.attribut_1, table_2.attribut_3
FROM table_1
JOIN table_2 ON table_1.attribut_1 = table_2.attribut_de_reference
JOIN table_3 ON table_2.attribut_2 = table_3.attribut_de_reference;
```

On précise à quelles tables appartiennent les attributs afin de lever les ambiguïtés si deux tables ont des attributs portant le même nom.