Les fonctions et les vues en MariaDB

Rappels

- Dans les derniers cours, nous avons vu les nombreuses possibilités que le langage SQL nous offre, en combinaison avec la commande `SELECT`, afin de pouvoir effectuer des requêtes qui retournent exactement les données dont nous avons besoin.
- Par exemple:
    - Les filtres (clause `WHERE`)
    - La clause de tri (`ORDER BY`)
    - La clause `LIMIT`
    - Les différents types de jointures
    - Le mot-clé `AS`
    - Les expressions conditionnelles (`CASE`)
    - Les opérations sur les ensembles (`UNION`, `INTERSECT`, `EXCEPT`)
    - Les sous-requêtes
    - Les jointures
- Nous avons aussi vu quelques fonctions disponibles sous MariaDB:
    - `CONCAT`
    - `NOW`
    - `DATEDIFF`

Tables utilisées dans les exemples

Pour les exemples suivants, nous allons à nouveau utiliser les tables de la base de données "librairie", créées par les requêtes suivantes:

```
CREATE TABLE editeur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL
);

CREATE TABLE langue (
    code CHAR(2) PRIMARY KEY,
    nom VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE livre (
    isbn CHAR(13) PRIMARY KEY,
    titre VARCHAR(50) NOT NULL,
    id_editeur INT,
    date_parution DATE NOT NULL,
```

```
    description TEXT,
    code_langue CHAR(2),
    prix DECIMAL(5,2) UNSIGNED,
        -- prix: Maximum de 5 chiffres dont 2 après la virgule
    FOREIGN KEY (id_editeur) REFERENCES editeur(id)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
    FOREIGN KEY (code_langue) REFERENCES langue(code)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

CREATE TABLE auteur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL
);

CREATE TABLE auteur_livre (
    id_auteur INT NOT NULL,
    isbn_livre CHAR(13) NOT NULL,
    PRIMARY KEY (id_auteur, isbn_livre),
    FOREIGN KEY (id_auteur) REFERENCES auteur(id)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
    FOREIGN KEY (isbn_livre) REFERENCES livre(isbn)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Les fonctions

- Le concept de fonction en informatique est semblable à celui qu'on retrouve en mathématiques
    - En mathématiques, une fonction fait correspondre une variable (`x`) à une autre (`y`)
    - Prenons par exemple la fonction `y = 2x`
        - Si `x` = 3, alors `y` = 6
        - Si `x` = 25, alors `y` = 50
        - Etc.
    - On peut donc dire qu'une fonction prend une **valeur d'entrée** (`x`) et produit une **valeur de sortie**
- En informatique, une fonction:
    - Peut prendre une ou des variables en entrée, qu'on appelle des **paramètres**
    - Peut produire (**retourner**) une valeur en sortie (un **résultat**)
    - Peut aussi modifier l'environnement dans lequel elle s'exécute (**effet de bord** ou ***side effect***)

        - Ex: afficher du texte à l'écran

- Les fonctions existent dans la vaste majorité des langages de programmation, et existent également en SQL.

- Nous en avons déjà vu les fonctions `CONCAT`, `DATEDIFF` et `NOW` disponibles sur MariaDB.

    - Qu'est-ce que `CONCAT` prend en paramètres, et que retourne-t-elle comme résultat?
    - Qu'est-ce que `DATEDIFF` prend en paramètres, et que retourne-t-elle comme résutlat?
    - La fonction `NOW` peut être appelée sans paramètre. Que retourne-t-elle comme résultat?
    - Est-ce que ces fonctions produisent des effets de bord?

Les fonctions intégrées dans MariaDB

- MariaDB propose plusieurs fonctions intégrées, c'est-à-dire qu'elles existent « de base » dans ce SGBD. Les fonctions que nous avons déjà vues en font partie.
- Les fonctions intégrées peuvent différer d'un SGBD à l'autre, mais on peut généralement trouver une équivalence à ce qu'on cherche.
- La documentation sur les fonctions intégrées dans MariaDB est disponible ici.

Voici quelques exemples de fonctions intégrées:

Fonctions sur les chaînes de caractères

- `CONCAT`
- `CONCAT_WS` pour concaténer avec un séparateur
- `CHAR_LENGTH` retourne le nombre de caractères dans une chaîne
- `LOWER` et `UPPER` pour convertir une chaîne en minuscules ou en majuscules

Fonctions numériques

- `CEILING` pour arrondir vers le haut
- `FLOOR` pour arrondir vers le bas
- `ROUND` pour arrondir au plus proche
- `ABS` pour obtenir la valeur absolue

Fonctions sur les dates et heures

- `NOW`
- `CURTIME`
- `DAYOFMONTH`
- `MONTH`
- `YEAR`
- `HOUR`
- `MINUTE`

Fonctions cryptographiques

- `AES_ENCRYPT`

- `AES_DECRYPT`

Autres

- `COALESCE` qui retourne la première valeur non-nulle dans une liste
- `CAST` pour convertir une valeur d'un type vers un autre

Les fonctions d'agrégation

- Une fonction d'agrégation retourne le résultat d'un calcul effectué sur un ensemble de données (par exemple, surt tous les résultats d'une requête)
- Il s'agit souvent d'une fonction statistique (ex: moyenne, maximum, somme, etc)
- Les fonctions d'agrégation les plus courantes sont les suivantes:
    - `COUNT` qui retourne le nombre d'éléments
    - `SUM` qui retourne la somme
    - `AVG` qui retourne la moyenne
    - `MIN` qui retourne le minimum
    - `MAX` qui retourne le maximum

Voici des exemples d'utilisation de ces fonctions:

```
-- Obtenir le nombre de livres dans la base de données
SELECT COUNT(*) FROM livre;

-- Obtenir le nombre de livres avec des titres uniques
SELECT COUNT(DISTINCT titre) FROM livre;

-- Obtenir la somme des prix de tous les livres
SELECT SUM(prix) FROM livre;

-- Obtenir la moyenne des prix de tous les livres
SELECT AVG(prix) FROM livre;

-- Obtenir le prix le plus bas
SELECT MIN(prix) FROM livre;

-- Obtenir le prix le plus haut
SELECT MAX(prix) FROM livre;

-- Obtenir le titre de livre le plus lointain dans l'ordre alphabétique
SELECT MAX(titre) FROM livre;

-- Obtenir le prix minimum, le prix maximum et le prix moyen
SELECT  MIN(prix) AS prix_min,
        MAX(prix) AS prix_max,
        AVG(prix) AS prix_moyen
    FROM livre;
```

```sql
-- Obtenir le nombre moyen de caractères dans les titres des livres
SELECT AVG(CHAR_LENGTH(titre)) FROM livre;

-- Obtenir la moyenne des prix des livres dont le titre commence par "Harry
Potter"
SELECT AVG(prix)
    FROM livre
    WHERE titre LIKE 'Harry Potter%';

-- Obtenir la moyenne des prix des livres écrits par J.K. Rowling
SELECT AVG(l.prix)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    WHERE a.nom = 'Rowling' AND prenom = 'J.K.';

-- Obtenir les titres des livres dont le prix est inférieur à la moyenne
SELECT titre
    FROM livre
    WHERE prix < (SELECT AVG(prix) FROM livre);
```

Les groupements (`GROUP BY`)

- La clause `GROUP BY` permet de regrouper les données selon la valeur d'une colonne avant d'exécuter une fonction d'agrégation.
- Par exemple, pour obtenir le nombre de livres par langue:

```sql
-- Nombre de livres par langue
SELECT code_langue, COUNT(*)
    FROM livre
    GROUP BY code_langue;

-- Si on veut afficher les noms des langues au lieu de leurs codes
SELECT lng.nom, COUNT(*)
    FROM livre l
    LEFT JOIN langue lng
        ON lng.code = l.code_langue
    GROUP BY lng.nom;
```

- Si la liste de sélection contient des colonnes qui ne sont pas des agrégats, il faut absolument ajouter une clause `GROUP BY`.
- Un groupement peut aussi être constitué de plusieurs colonnes. Par exemple, pour grouper les prix moyens des livres par les noms et prénoms des auteurs:

```sql
SELECT a.nom, a.prenom, AVG(l.prix)
    FROM livre l
```

```
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom, a.prenom;
```

- La clause `GROUP BY` a aussi pour effet d'ordonner les données de manière ascendante par les colonnes du groupement, comme le ferait un `ORDER BY`.
    - L'ordre des colonnes du `GROUP BY` a donc une importance, de la même façon que dans un `ORDER BY`.
- On peut ajouter des `ASC` ou `DESC` aux colonnes du `GROUP BY`, comme on le ferait dans un `ORDER BY`:

```
SELECT a.nom, a.prenom, AVG(l.prix)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom DESC, a.prenom ASC;
```

- Si on veut ordonner les données autrement, il faut ajouter une clause `ORDER BY`. Celle-ci doit obligatoirement se trouver APRÈS le `GROUP BY`. Par exemple, si on veut les ordonner seulement par prénom (sans prendre en compte le nom):

```
SELECT a.nom, a.prenom, AVG(l.prix)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom DESC, a.prenom DESC
    ORDER BY a.prenom;
```

- On peut même ordonner les données selon une colonne qui ne se trouve pas dans la liste de sélection. Par exemple, pour ordonner les données par date descendante de la date de parution des livres (c'est le livre le plus récent de chaque auteur qui déterminera l'ordre):

```
SELECT a.nom, a.prenom, AVG(l.prix)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
```

```
    GROUP BY a.nom DESC, a.prenom DESC
    ORDER BY l.date_parution DESC;
```

- On peut aussi ordonner par la valeur d'un agrégat. Par exemple:

```
SELECT a.nom, a.prenom, AVG(l.prix)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom, a.prenom
    ORDER BY AVG(prix), nom, prenom;
```

- Ou encore mieux dans ce cas-ci:

```
SELECT a.nom, a.prenom, AVG(l.prix) AS prix_moyen
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom, a.prenom
    ORDER BY prix_moyen, nom, prenom;
```

Les filtres sur les groupements (`HAVING`)

- Si on inclut une clause `WHERE` dans une requête qui comprant un `GROUP BY`, il faut absolument la placer AVANT le `GROUP BY`.
- Le filtre sera d'ailleurs appliqué avant l'agrégation et le groupement.
- Voici quelques exemples:

```
-- Obtenir la date de parution du livre le plus récent de chaque auteur dont le
nom commence par 'S'
SELECT a.nom, a.prenom, MAX(date_parution)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    WHERE a.nom LIKE 'S%'
    GROUP BY a.nom, a.prenom;

-- Obtenir le plus bas prix des livres parus depuis l'an 2000, par éditeur
SELECT e.nom, MIN(prix)
    FROM livre l
```

```
    JOIN editeur e
        ON e.id = l.id_editeur
    WHERE YEAR(l.date_parution) >= 2000
    GROUP BY e.nom;
```

On ne peut cependant pas inclure d'agrégat dans une clause `WHERE`. La requête suivante ne serait par exemple pas valide:

```
SELECT a.nom, a.prenom, COUNT(*)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    WHERE COUNT(*) > 1 -- Clause WHERE invalide
    GROUP BY a.nom, a.prenom;
```

Dans ce cas-ci, il faut plutôt utiliser la clause `HAVING`:

```
SELECT a.nom, a.prenom, COUNT(*)
    FROM livre l
    JOIN auteur_livre al
        ON al.isbn_livre = l.isbn
    JOIN auteur a
        ON a.id = al.id_auteur
    GROUP BY a.nom, a.prenom
    HAVING COUNT(*) > 1;
```

Les vues

- Une vue en SQL est une requête de type `SELECT` à laquelle on a donné un nom, et qu'on a enregistré dans la base de données pour pouvoir la rappeler ultérieurement.
- Une vue s'utilise comme une table, on peut donc faire des requêtes sur celle-ci.
- Les données d'une vue ne sont pas directement stockées dans la base de données, ce sont les données des tables utilisées par la vue qui sont stockées.
- Une vue peut être utile pour se simplifier la vie lorsqu'on doit souvent travailler sur des données provenant de requêtes complexes (par exemple, avec beaucoup de jointures ou des fonctions d'agrégation).
- L'exemple suivant crée une vue permettant d'obtenir les informations de tous les livres avec leurs noms d'éditeurs et d'auteurs:

```
CREATE VIEW vue_livres AS
    SELECT  l.isbn,
            l.titre,
            CONCAT_WS(' ', a.prenom, a.nom) AS auteur,
```

```
            e.nom AS editeur,
            l.date_parution,
            l.description,
            l.code_langue
        FROM livre l
        JOIN auteur_livre al
            ON al.isbn_livre = l.isbn
        JOIN auteur a
            ON a.id = al.id_auteur
        JOIN editeur e
            ON e.id = l.id_editeur;
```

- On peut aussi utiliser la commande `CREATE OR REPLACE` de façon à créer la vue si elle n'existe pas déjà, ou à la remplacer si elle existe déjà:

```
CREATE OR REPLACE VIEW vue_livres AS
    SELECT  l.isbn,
            l.titre,
            CONCAT_WS(' ', a.prenom, a.nom) AS auteur,
            e.nom AS editeur,
            l.date_parution,
            l.description,
            l.code_langue
        FROM livre l
        JOIN auteur_livre al
            ON al.isbn_livre = l.isbn
        JOIN auteur a
            ON a.id = al.id_auteur
        JOIN editeur e
            ON e.id = l.id_editeur;
```

- Une fois que la vue est créée, on peut exécuter des requêtes sur celle-ci:

```
-- Récupérer toutes les données de vue_livres
SELECT * FROM vue_livres;

-- Récupérer toutes les données de vue_livres et les ordonner par titre
SELECT * FROM vue_livres ORDER BY titre;

-- Récupérer toutes les données de vue_livres pour les livres dont l'auteur est
J.K. Rowling
SELECT * FROM vue_livres WHERE auteur = 'J.K. Rowling';

-- SELECT sur une vue avec une jointure sur une table
SELECT vl.*, lng.nom AS nom_langue
    FROM vue_livres vl
    JOIN langue lng
        ON lng.code = vl.code_langue;
```

- On peut supprimer une vue avec la commande `DROP VIEW`.
- Il est aussi possible de faire des `INSERT` et des `UPDATE` sur une vue, mais celle-ci doit respecter une série de conditions qui sont décrites ici.

Références

- Documentation officielle de MariaDB:
  - Sur les fonctions intégrées
  - Sur les requêtes `SELECT`
  - Sur la clause `GROUP BY`
  - Sur les vues
- Tutoriel sur MySQL de W3Schools