Les procédures stockées et les déclencheurs en MariaDB

Tables utilisées dans les exemples

Pour les exemples suivants, nous allons à nouveau utiliser les tables de la base de données "librairie", mais avec quelques ajouts:

```
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,
    -- AJOUT
    nombre_exemplaires_disponibles INT UNSIGNED NOT NULL DEFAULT 0,
    FOREIGN KEY (id_editeur) REFERENCES editeur(id),
    FOREIGN KEY (code_langue) REFERENCES langue(code)
);

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),
    FOREIGN KEY (isbn_livre) REFERENCES livre(isbn)
);

-- AJOUT
```

```sql
CREATE TABLE facture (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    date_heure DATETIME NOT NULL DEFAULT NOW()
);

-- AJOUT
CREATE TABLE facture_livre (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    id_facture INT UNSIGNED NOT NULL,
    isbn_livre CHAR(13) NOT NULL,
    prix_vente DECIMAL(5, 2) NOT NULL,
    FOREIGN KEY (id_facture) REFERENCES facture(id),
    FOREIGN KEY (isbn_livre) REFERENCES livre(isbn)
);
```

Les procédures stockées

- Une procédure stockée est une séquence de requêtes SQL préalablement écrite et enregistrée dans la base de données.
- Elle peut prendre des données en entrée et produire des données en sortie.
- Elle permet aussi d'utiliser des éléments de programmation structurée, tels que des variables, des structures conditionnelles (`IF - ELSE IF - ELSE`) et des boucles. Ces possibilités peuvent varier d'un SGBD à l'autre.
- Les procédures stockées ont plusieurs utilités, par exemple:
    - Limiter la répétition de code;
    - Réutiliser une requête complexe sans devoir la réécrire à chaque fois;
    - Contrôler l'accès aux données.

Créer et appeler une procédure stockée

Voici un exemple de création d'une procédure simple:

```sql
DELIMITER //

CREATE PROCEDURE recuperer_livres_editeur (
    IN nom_editeur VARCHAR(50)
)
BEGIN
    SELECT * FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur
    WHERE e.nom = nom_editeur;
END;
//

DELIMITER ;
```

Décortiquons chaque ligne de cette requête:

1. `DELIMITER //` : indique à MariaDB qu'on veut temporairement utiliser les caractères `//` au lieu d'un point-virgule (`;`) pour indiquer la fin d'une requête. Cela est nécessaire, car on utilisera le point-virgule à l'intérieur de la requête `CREATE PROCEDURE` et on ne veut pas que ce soit interprété comme la fin de celle-ci.
2. `CREATE PROCEDURE recuperer_livres_editeurs (` : on indique qu'on veut créer une procédure nommée `recuperer_livres_editeurs`, puis on ouvre une parenthèse pour définir la liste des paramètres. On aurait aussi pu utiliser `CREATE OR REPLACE PROCEDURE`.
3. `IN nom_editeur VARCHAR(50)` : on crée un paramètre d'entrée (`IN`) nommé `nom_editeur`. Ce paramètre prendra une donnée de type `VARCHAR(50)`.
4. `)` : fin de la liste des paramètres (nous avons un seul paramètre pour cette procédure).
5. `BEGIN` : indique le début du corps de la procédure.
6. On a ensuite une requête `SELECT`. Remarquons que cette requête utilise le paramètre `nom_editeur` reçu en entrée, et se termine par un `;` comme n'importe quelle requête.
7. `END;` : indique la fin du corps de la procédure.
8. `//` : indique la fin de la requête `CREATE PROCEDURE`, puisqu'on a temporairement remplacé le caractère de fin de requête par `//`.
9. `DELIMITER ;` : redéfinit le point-virgule comme caractère de fin de requête.

La requête suivante permet ensuite d'appeler la procédure stockée en lui passant le nom de l'éditeur voulu:

```
CALL recuperer_livres_editeur('Pearson');
```

Cet appel de procédure aura pour effet de récupérer la liste de l'éditeur "Pearson", puisque c'est ce que fait la requête dans le corps de la procédure.

Une procédure peut aussi retourner des données dans des paramètres de sortie. Voici un exemple:

```
DELIMITER //

CREATE OR REPLACE PROCEDURE recuperer_nombre_livres_editeur (
    IN nom_editeur VARCHAR(50),
    OUT nombre_livres INT
)
 BEGIN
     SELECT COUNT(*) INTO nombre_livres FROM livre l
        JOIN editeur e
            ON e.id = l.id_editeur
        WHERE e.nom = nom_editeur;
 END;
//

DELIMITER ;
```

Dans cet exemple, la procédure prend un deuxième paramètre, nommé `nombre_livres`, qui lui est de type `OUT` pour indiquer qu'il servira à stocker une donnée retournée par la procédure. C'est la clause `INTO`

`nombre_livres` dans la requête `SELECT` qui donne une valeur de sortie à ce paramètre.

Voici comment appeler cette procédure est récupérer la valeur de sortie:

```
SET @nb_livres = 0;

CALL recuperer_nombre_livres_editeur('Pearson', @nb_livres);

SELECT @nb_livres;
```

La commande `SET @nb_livres = 0;` permet de déclarer la variable qui sera utilisée pour stocker la valeur de sortie. On doit mettre un `@` au début du nom de la variable puisqu'il s'agit d'une variable utilisateur et non d'une variable système.

Voici maintenant un exemple de procédure plus complexe, qui utilise plusieurs requêtes, de même que des structures conditionnelles:

```
DELIMITER //

CREATE OR REPLACE PROCEDURE ajouter_livre (
    IN param_isbn CHAR(13),
    IN param_titre VARCHAR(50),
    IN param_nom_editeur VARCHAR(50),
    IN param_date_parution DATE,
    IN param_description TEXT,
    IN param_code_langue CHAR(2),
    IN param_prix DECIMAL(5, 2),
    IN param_nom_auteur VARCHAR(50),
    IN param_prenom_auteur VARCHAR(50)
)
BEGIN
  DECLARE var_id_editeur, var_id_auteur INT;

  SELECT id INTO var_id_editeur FROM editeur WHERE nom = param_nom_editeur;

  IF var_id_editeur IS NULL THEN
    INSERT INTO editeur(nom) VALUES(param_nom_editeur);
    SELECT LAST_INSERT_ID() INTO var_id_editeur;
  END IF;

  INSERT INTO livre(
    isbn,
    titre,
    id_editeur,
    date_parution,
    description,
    code_langue,
    prix
  )
  VALUES (
```

```
    param_isbn,
    param_titre,
    var_id_editeur,
    param_date_parution,
    param_description,
    param_code_langue,
    param_prix
  );

  SELECT id INTO var_id_auteur
    FROM auteur
    WHERE nom = param_nom_auteur AND prenom = param_prenom_auteur;

  IF var_id_auteur IS NULL THEN
    INSERT INTO auteur(nom, prenom)
        VALUES(param_nom_auteur, param_prenom_auteur);
    SELECT LAST_INSERT_ID() INTO var_id_auteur;
  END IF;

  INSERT INTO auteur_livre(isbn_livre, id_auteur)
    VALUES (param_isbn, var_id_auteur);
END;
//

DELIMITER ;
```

Cette procédure prend en paramètres les informations d'un livre à insérer, incluant le nom de l'éditeur, et les noms et prénoms de l'auteur du livre. Elle effectue ensuite les opérations suivantes:

- Récupérer l'ID de l'éditeur correspondant au nom reçu en paramètre;
- Si cet éditeur n'existe pas, le créer;
- Créer le nouveau livre;
- Récupérer l'ID de l'auteur correspondant au nom et prénom reçus en paramètres;
- Si cet auteur n'existe pas, le créer;
- Crée une entrée dans la table `auteur_livre` pour lier le livre au bon auteur.

Cette fois-ci, nous avons utilisé l'instruction `DECLARE` pour déclarer les variables internes à la procédure. Contrairement à `SET`, le mot-clé `DECLARE` est seulement disponible à l'intérieur d'une procédure stockée. Par ailleurs, il n'est pas obligatoire pour une variable déclarée de cette façon d'avoir un `@` au début de son nom.

Voici un exemple d'appel de cette procédure:

```
CALL ajouter_livre(
    '9782922145441',
    'Aliss',
    'ALIRE',
    '2000-11-16',
    CONCAT(
        'L''histoire d''Aliss, une adolescente rebelle ',
        'qui découvre un Montréal obscur et fantastique, ',
        'hanté par des créatures surnaturelles.'
```

```
    ),
    'fr',
    16.95,
    'Sénécal',
    'Patrick'
);
```

Les déclencheurs

Un déclencheur (*trigger*) permet d'exécuter une ou des requêtes automatiquement en réponse à une autre requête. Un déclencheur est exécuté lorsqu'un événement spécifique survient (soit un `INSERT`, un `UPDATE` ou un `DELETE`) sur une table spécifique, et peut être exécutée soit avant (`BEFORE`) ou après (`AFTER`) l'événement.

Voici par exemple un déclencheur `AFTER INSERT` sur la table `facture_livre`, qui décrémente le nombre d'exemplaires disponibles d'un livre lorsque celui-ci est ajouté à une facture:

```
DELIMITER //

CREATE TRIGGER maj_exemplaires_disponibles
AFTER INSERT ON facture_livre
FOR EACH ROW
BEGIN
    UPDATE livre
        SET nombre_exemplaires_disponibles = nombre_exemplaires_disponibles - 1
        WHERE isbn = NEW.isbn_livre;
END; //

DELIMITER ;
```

Dans cet exemple, le mot-clé `NEW` permet d'accéder aux champs de la requête d'insertion. Voici un exemple de code qui provoque l'exécution de ce déclencheur:

```
-- Créer une nouvelle facture
INSERT INTO facture VALUES();

-- Ajouter un livre à cette facture
-- Le déclencheur s'exécutera suite à l'insertion
INSERT INTO facture_livre(id_facture, isbn_livre, prix_vente)
    VALUES((SELECT LAST_INSERT_ID()), '0747532699', 16.95);
```

Voici maintenant un déclencheur `BEFORE INSERT`, toujours sur la table `facture_livre`, qui renseigne automatiquement le champ `prix_vente` lorsque celui-ci a la valeur `NULL` dans la requête d'insertion:

```
DELIMITER //

CREATE TRIGGER inserer_prix_vente
```

```
BEFORE INSERT ON facture_livre
FOR EACH ROW
BEGIN
    IF NEW.prix_vente IS NULL THEN
        SET NEW.prix_vente = (
            SELECT prix FROM livre WHERE isbn = NEW.isbn_livre
        );
    END IF;
END; //

DELIMITER ;
```

Voici une requête d'insertion qui sera modifiée par ce déclencheur:

```
INSERT INTO facture_livre(id_facture, isbn_livre, prix_vente)
    VALUES((SELECT LAST_INSERT_ID()), '0747532699', NULL);
```

Il est à noter qu'on doit absolument spécifier la valeur `NULL` pour le champ `prix_vente` afin que le déclencheur fonctionne comme prévu. On ne peut pas simplement omettre ce champ dans la requête.

Les déclencheurs de type `BEFORE` sont justement utiles lorsqu'on veut modifier la valeur d'un des champs d'un `INSERT` ou d'un `UPDATE` avant que celui-ci soit exécuté. Ces déclencheurs peuvent aussi être utilisés pour prévenir l'exécution d'une requête. Voici par exemple une requête qui empêche l'ajout d'un livre à une facture si celui-ci n'a aucun exemplaire disponible:

```
DELIMITER //

CREATE TRIGGER valider_livre_disponible
BEFORE INSERT ON facture_livre
FOR EACH ROW
BEGIN
    IF (
        SELECT nombre_exemplaires_disponibles
            FROM livre
            WHERE isbn = NEW.isbn_livre
    ) = 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Ce livre n''a aucun exemplaire disponible.';
    END IF;
END;//

DELIMITER ;
```

L'instruction `SIGNAL SQLSTATE '45000'` permet de provoquer volontairement une erreur lors de l'exécution du déclencheur. Puisqu'il s'agit d'un déclencheur de type `BEFORE`, l'erreur empêche l'exécution de la requête `INSERT`.

On peut aussi créer des déclencheurs sur les requêtes `UPDATE` et `DELETE`. Dans ce cas, le mot-clé `OLD` permet d'accéder aux anciennes valeurs. On pourrait par exemple utiliser de tels déclencheurs pour conserver un historique de toutes les modifications apportées à une table. Voici du code qui met en place une nouvelle table `historique_livre` et qui ajoute des déclencheurs pour ajouter une nouvelle ligne à cette table chaque fois qu'un livre est modifié ou supprimé:

```
CREATE TABLE historique_livre (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_requete CHAR(6) NOT NULL,
    moment_requete DATETIME DEFAULT NOW(),
    ancien_isbn CHAR(13),
    ancien_titre VARCHAR(50),
    ancien_id_editeur INT,
    ancienne_date_parution DATE,
    ancienne_description TEXT,
    ancien_code_langue CHAR(2),
    ancien_prix DECIMAL(5,2) UNSIGNED,
    ancien_nombre_exemplaires_disponibles INT UNSIGNED,
    nouveau_isbn CHAR(13),
    nouveau_titre VARCHAR(50),
    nouveau_id_editeur INT,
    nouvelle_date_parution DATE,
    nouvelle_description TEXT,
    nouveau_code_langue CHAR(2),
    nouveau_prix DECIMAL(5,2) UNSIGNED,
    nouveau_nombre_exemplaires_disponibles INT UNSIGNED,
    FOREIGN KEY (ancien_id_editeur) REFERENCES editeur(id),
    FOREIGN KEY (ancien_code_langue) REFERENCES langue(code),
    FOREIGN KEY (nouveau_id_editeur) REFERENCES editeur(id),
    FOREIGN KEY (nouveau_code_langue) REFERENCES langue(code)
);

DELIMITER //

CREATE TRIGGER livre_after_update
AFTER UPDATE ON livre
FOR EACH ROW
BEGIN
    INSERT INTO historique_livre (
        type_requete,
        ancien_isbn,
        ancien_titre,
        ancien_id_editeur,
        ancienne_date_parution,
        ancienne_description,
        ancien_code_langue,
        ancien_prix,
        ancien_nombre_exemplaires_disponibles,
        nouveau_isbn,
        nouveau_titre,
        nouveau_id_editeur,
        nouvelle_date_parution,
```

```
        nouvelle_description,
        nouveau_code_langue,
        nouveau_prix,
        nouveau_nombre_exemplaires_disponibles
    ) VALUES (
        'UPDATE',
        OLD.isbn,
        OLD.titre,
        OLD.id_editeur,
        OLD.date_parution,
        OLD.description,
        OLD.code_langue,
        OLD.prix,
        OLD.nombre_exemplaires_disponibles,
        NEW.isbn,
        NEW.titre,
        NEW.id_editeur,
        NEW.date_parution,
        NEW.description,
        NEW.code_langue,
        NEW.prix,
        NEW.nombre_exemplaires_disponibles
    );
END; //

CREATE TRIGGER livre_after_delete
AFTER DELETE ON livre
FOR EACH ROW
BEGIN
    INSERT INTO historique_livre (
        type_requete,
        ancien_isbn,
        ancien_titre,
        ancien_id_editeur,
        ancienne_date_parution,
        ancienne_description,
        ancien_code_langue,
        ancien_prix,
        ancien_nombre_exemplaires_disponibles
    ) VALUES (
        'DELETE',
        OLD.isbn,
        OLD.titre,
        OLD.id_editeur,
        OLD.date_parution,
        OLD.description,
        OLD.code_langue,
        OLD.prix,
        OLD.nombre_exemplaires_disponibles
    );
END; //

DELIMITER ;
```

Références

- Documentation officielle de MariaDB:
    - `CREATE PROCEDURE`
    - `SELECT INTO`
    - `Expressions de programmation structurée`
    - `Déclencheurs`
- Tutoriel de DigitalOcean sur les déclencheurs en MySQL
- Tutoriel sur MySQL de W3Schools