SQL : procédures stockées et triggers

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

SQL

Procédures stockées (stored procedures)

- Disponible depuis la version 5 de **MySQL**
- Ensemble d'instructions SQL portant un nom, qu'on peut l'utiliser pour l'appeler : `call nomProcedure()`
- Facilitant certains traitement sur une ou plusieurs tables (possibilité d'effectuer de tests, boucles...)
- Minimisant le trafic entre le client et le serveur de données

SQL

Pour créer une procédure stockée

```
CREATE PROCEDURE nomProcedure (les paramètres)

BEGIN
    -- traitements = les instructions SQL
END
```

SQL

Pour créer une procédure stockée

```
CREATE PROCEDURE nomProcedure (les paramètres)

BEGIN
    -- traitements = les instructions SQL
END
```

Remarques

- Chaque instruction SQL d'une procédure doit se terminer par `;` (délimiteur)
- Chaque requête SQL doit aussi se terminer par `;`
- Il faut changer le délimiteur avant le début de la procédure et le remettre à la fin

SQL

Pour créer une procédure stockée

```
DELIMITER |
CREATE PROCEDURE nomProcedure (les paramètres)

BEGIN
    -- traitements = les instructions SQL
END |
DELIMITER ;
```

SQL

Exemple

- Créons une procédure stockée qui augmente le salaire de la personne ayant le plus petit salaire de la table personne
- Le montant à ajouter au salaire est passé en paramètre

SQL

Exemple

- Créons une procédure stockée qui augmente le salaire de la personne ayant le plus petit salaire de la table personne
- Le montant à ajouter au salaire est passé en paramètre

Déclaration de la procédure

```
DELIMITER |
CREATE PROCEDURE  augmenterSalaireMin(somme int)
BEGIN
  DECLARE id int;
  SELECT num INTO id FROM personne WHERE salaire = (SELECT MIN(salaire)
      FROM personne);
  UPDATE personne SET salaire = salaire + somme WHERE num = id;
END |
DELIMITER ;
```

SQL

Explication

- `DECLARE` permet de déclarer une variable et `DEFAULT` de l'initialiser
- `INTO` permet d'indiquer le nom de la variable dans laquelle on va placer le contenu du `SELECT`

SQL

Explication

- `DECLARE` permet de déclarer une variable et `DEFAULT` de l'initialiser
- `INTO` permet d'indiquer le nom de la variable dans laquelle on va placer le contenu du `SELECT`

Remarque

`SELECT INTO` **permet de sélectionner seulement une seule ligne. Une erreur sera générée si la requête sélectionne plusieurs lignes. En cas de doute, pensez à ajouter** `LIMIT 1`**.**

SQL

Appel de la procédure (exécution)

```
CALL augmenterSalaireMin(50);
```

SQL

Appel de la procédure (exécution)

```
CALL augmenterSalaireMin(50);
```

Pour consulter le code de la procédure `augmenterSalaireMin`

```
SHOW CREATE PROCEDURE augmenterSalaireMin;
```

SQL

Appel de la procédure (exécution)

```
CALL augmenterSalaireMin(50);
```

Pour consulter le code de la procédure `augmenterSalaireMin`

```
SHOW CREATE PROCEDURE augmenterSalaireMin;
```

Pour supprimer la procédure `augmenterSalaireMin`

```
DROP PROCEDURE IF EXISTS augmenterSalaireMin;
```

SQL

Appel de la procédure (exécution)

```
CALL augmenterSalaireMin(50);
```

Pour consulter le code de la procédure `augmenterSalaireMin`

```
SHOW CREATE PROCEDURE augmenterSalaireMin;
```

Pour supprimer la procédure `augmenterSalaireMin`

```
DROP PROCEDURE IF EXISTS augmenterSalaireMin;
```

Remarque

On ne peut modifier une procédure avec MySQL. Il faut donc supprimer puis recréer.

SQL

Il est possible d'utiliser une structure de contrôle de type `if ... then ... else`

```
DELIMITER |
CREATE PROCEDURE  augmenterSalaireMin(somme int)
BEGIN
  DECLARE id INT;
  DECLARE smic INT;
  DECLARE min INT;
  SET smic = 1200;
  SELECT MIN(salaire) INTO min FROM personne;
  SELECT num INTO id FROM personne WHERE salaire = min LIMIT 1;
  IF min > smic THEN
    UPDATE personne SET salaire = salaire + somme WHERE num = id;
  ELSE
    UPDATE personne SET salaire = smic + somme WHERE num = id;
  END IF;
END |
DELIMITER ;
```

SQL

Il est possible d'utiliser une structure de contrôle de type `if ... then ... else`

```
DELIMITER |
CREATE PROCEDURE  augmenterSalaireMin(somme int)
BEGIN
  DECLARE id INT;
  DECLARE smic INT;
  DECLARE min INT;
  SET smic = 1200;
  SELECT MIN(salaire) INTO min FROM personne;
  SELECT num INTO id FROM personne WHERE salaire = min LIMIT 1;
  IF min > smic THEN
    UPDATE personne SET salaire = salaire + somme WHERE num = id;
  ELSE
    UPDATE personne SET salaire = smic + somme WHERE num = id;
  END IF;
END |
DELIMITER ;
```

Il existe aussi `ELSEIF` **pour enchaîner les tests.**

SQL

Autre structure de contrôle : `CASE ... WHEN ... THEN ... ELSE`

```
CASE nomVariable
  WHEN value1 THEN traitement1;
  WHEN value2 THEN traitement2;
  ...
  ELSE autreTraitement;
END CASE;
```

SQL

Autre structure de contrôle : `CASE ... WHEN ... THEN ... ELSE`

```
CASE nomVariable
  WHEN value1 THEN traitement1;
  WHEN value2 THEN traitement2;
  ...
  ELSE autreTraitement;
END CASE;
```

Exercice

En utilisant `CASE ... WHEN ... THEN ... ELSE`, écrire une **procédure stockée** qui permet d'augmenter

- de 200 euros le salaire de la personne qui habite à Marseille et qui a un véhicule si son salaire est égal au SMIC,
- de 100 euros sinon.

SQL

La boucle : `WHILE ... DO`

```
WHILE condition(s) DO
  -- traitements
END WHILE;
```

SQL

La boucle : `WHILE ... DO`

```
WHILE condition(s) DO
  -- traitements
END WHILE;
```

Exercice

En utilisant `WHILE ... DO`, écrire une **procédure stockée** qui

- prend deux paramètres : `n` et `somme`
- permet d'ajouter `n` fois `somme` au salaire de la personne qui habite à Marseille et qui a un véhicule

SQL

La boucle : `repeat ... until`

```
REPEAT
  -- traitements
UNTIL condition(s)
END REPEAT;
```

SQL

La boucle : `repeat ... until`

```
REPEAT
  -- traitements
UNTIL condition(s)
END REPEAT;
```

Exercice

Refaire l'exercice précédent avec `repeat ... until`

SQL

On peut aussi définir des libellés et utiliser `ITERATE ... LEAVE`

```
label_loop: boucle -- peut être WHILE, REPEAT ou autre
  -- traitements
  IF conditions THEN
    LEAVE label_loop;
  END IF;
  IF autres_conditions THEN
    ITERATE label_loop;
  END IF;
END LOOP;
```

SQL

On peut aussi définir des libellés et utiliser `ITERATE ... LEAVE`

```
label_loop: boucle -- peut être WHILE, REPEAT ou autre
  -- traitements
  IF conditions THEN
    LEAVE label_loop;
  END IF;
  IF autres_conditions THEN
    ITERATE label_loop;
  END IF;
END LOOP;
```

Explication

- `ITERATE` permet de relancer une itération en ignorant le reste du code (de la boucle)
- `LEAVE` permet de quitter la boucle en ignorant le reste du code (de la boucle)

SQL

La boucle `LOOP ... LEAVE`

```
label_loop: LOOP
  -- traitements
  IF condition THEN
    LEAVE label_loop;
  END IF;
END LOOP;
```

SQL

Déclencheurs (triggers)

- Un ensemble d'instructions SQL attaché à une table
- Exécuté avant ou après un évènement sur la table de type (insertion, modification ou suppression)

SQL

Déclencheurs (triggers)

- Un ensemble d'instructions SQL attaché à une table
- Exécuté avant ou après un évènement sur la table de type (insertion, modification ou suppression)

Remarque

- Pour une table donnée, un seul trigger par évènement et par moment
- Possibilité d'avoir un trigger `before insert` et un `after insert`
- Possibilité d'avoir un trigger `before insert` et un `before update`

SQL

`OLD` et `NEW`

- `OLD.colonne` désigne l'ancienne valeur de `colonne`
- `NEW.colonne` désigne la nouvelle valeur de `colonne`

SQL

Exemple

Avant chaque insertion d'un nouveau tuple dans la table `personne`, si une valeur pour la colonne `ville` n'a pas été renseignée, on attribue la valeur `Marseille` à la colonne `ville`.

SQL

Le trigger

```
DELIMITER |
CREATE TRIGGER setVilleMarseille BEFORE INSERT ON
   personne
FOR EACH ROW
BEGIN

   IF NEW.ville IS NULL THEN
     SET NEW.ville = 'Marseille';
   END IF;

END |
DELIMITER ;
```

SQL

Exercice

Écrire un trigger qui vérifie avant chaque augmentation de salaire si la différence entre l'ancien et nouveau salaire dépasse 200 euros. Si c'est le cas, on annule l'augmentation. Sinon, l'augmentation est acceptée.

SQL

Solution

```
DELIMITER |
CREATE TRIGGER  augmenterSalaire BEFORE UPDATE ON
   personne
FOR EACH ROW
BEGIN

   IF NEW.salaire >= OLD.salaire + 200 THEN
     SET NEW.salaire = OLD.salaire;
   END IF;

END |
DELIMITER ;
```