5000 entreprises postulent pour vous
talent.io
JE REÇOIS DES OFFRES

Utilisation avancée des procédures stockées MySQL

Table des matières

I. Récapitulatif des bases
II. Que puis-je faire avec les procédures stockées et pourquoi ?
III. Déclarer et configurer des variables dans vos procédures stockées
IV. Donner des valeurs variables aux requêtes SELECT
V. Contrôler le flux de votre procédure stockée
V-A. La commande IF
V-B. La commande CASE
V-C. La boucle WHILE
V-D. La boucle LOOP
V-E. La boucle REPEAT
V-F. Les commandes LEAVE et ITERATE
VI. Conclusion
VII. Liens

Il y a quelque temps, j'ai écrit un article sur l'utilisation des procédures stockées dans MySQL et l'extension de MySQLi en PHP pour les exécuter. Je vais maintenant rapidement couvrir quelques techniques avancées que vous pouvez utiliser pour réduire le nombre de données transférées entre la base de données et votre application.

2 commentaires

Article lu 50016 fois.

Les deux auteurs

Luke Skelding (Auteur)

Joris Crozier (Traducteur)

L'article

Publié le 17 septembre 2009 - Mis à jour le 10 décembre 2019

Version PDF | Version hors-ligne

ePub, Azw et Mobi

Liens sociaux

Partager

I. Récapitulatif des bases

D'abord un récapitulatif des bases pour ceux qui n'auraient pas lu mon article original. Les procédures stockées sont des bouts de code SQL stockés sur le serveur de bases de données qui peuvent être appelés par leurs noms depuis les clients. Les procédures stockées peuvent traitées de multiples commandes SQL, prendre des paramètres et retourner des DataSets. Les procédures stockées sont présentes dans d'autres systèmes de bases de données depuis plusieurs années jusqu'à ce que récemment cette fonctionnalité soit accessible dans MySQL. Le format d'une requête SQL pour créer une procédure stockée ressemble à ceci :

Sélectionnez

```
DELIMITER $$

CREATE PROCEDURE `sdpExampleProcedure`
  (
      paramètre 1 VARCHAR(255),
      paramètre 2 INTEGER
  )
  BEGIN
    /*
    Requêtes SQL séparées par un point-virgule (;)
    Utilisant paramètre 1 et paramètre 2      */
  END$$

DELIMITER ;
```

Notez la commande **DELIMITER** au début et à la fin qui change la manière de séparer les requêtes de MySQL. Ensuite pour appeler la procédure, vous pouvez utiliser la commande suivante :

Sélectionnez

```
CALL `sdpExampleProcedure`('testdata', '11');
```

II. Que puis-je faire avec les procédures stockées et pourquoi ?

Les procédures stockées peuvent être utilisées pour grouper les commandes SQL et ajouter une logique dans vos états SQL, tout en réduisant la quantité de données transférées depuis le serveur de bases de données et votre application. Par exemple, en administrant des sessions dans une application PHP. Si vous voulez mettre à jour un enregistrement session quand une requête est faite pour une session existante, ou créer une nouvelle session en tout vous pourriez faire tout cela avec des contrôles dans une procédure stockée. Cela devrait normalement consister à utiliser une commande **SELECT** pour récupérer toutes les données voulues depuis le serveur de données en rapport avec l'id de session donné, ensuite en programmant votre code pour exécuter une commande **INSERT** ou **UPDATE** qui dépend du nombre de lignes retournées. Avec les procédures stockées vous pouvez inclure tout ceci dans une seule commande SQL simple qui retourne simplement vrai quand une session est créée, et faux lorsque une session est mise à jour.

III. Déclarer et configurer des variables dans vos procédures stockées

En addition des paramètres définis qui sont passés durant l'appel, il est possible de définir des variables sur lesquelles travailler durant l'exécution. Pour faire ça vous devez utiliser la commande **DECLARE**, faisant état du nom et du type de la variable. Dans l'exemple suivant, je définis trois variables appelées « session_count », « session_id » et « session_content », de type **INTEGER**, **VARCHAR** (de taille 32) et **TEXT** respectivement.

Sélectionnez

```
DECLARE session_count INTEGER;
DECLARE session_id VARCHAR(32);
DECLARE session_content TEXT;
```

Maintenant que vous avez déclaré vos variables, vous pouvez configurer leur valeur en utilisant la commande **SET**. Une fois que ces variables ont une valeur, vous pouvez les utiliser plus tard dans votre procédure.

Sélectionnez

```
SET session_count = 1;
SET session_id = '699d571326815e40b5e1ae99af04563c';
SET session_content = 'Du contenu de session ici';
```

Si vous connaissez la valeur de vos variables au moment de les déclarer, vous pouvez utiliser la commande **DEFAULT** pour les configurer. Ci-dessous j'ai condensé les six commandes en trois utilisant cette technique :

Sélectionnez

```
DECLARE session_count INTEGER DEFAULT 1;
DECLARE session_id VARCHAR(32) DEFAULT '699d571326815e40b5e1ae99af04563c';
DECLARE session_content TEXT DEFAULT 'Du contenu de session ici';
```

IV. Donner des valeurs variables aux requêtes SELECT

Une utilisation excellente des variables dans vos procédures est d'utiliser le résultat d'une commande **SELECT** pour configurer la valeur d'une variable existante. Dans d'autres systèmes de bases de données, ceci est vraiment simple, cependant dans MySQL bien que ce soit simple, la syntaxe est quelque peu inattendue.

Dans le but de transmettre une valeur depuis une requête **SELECT** vers une variable, il y a une ligne supplémentaire que nous devons ajouter à la requête **SELECT** juste avant la définition de la table dans laquelle les données seront extraites. La commande **INTO** met les données dans une variable définie. Voici un exemple pour illustrer son utilisation :

Sélectionnez

```
DECLARE session_count INTEGER;

SELECT COUNT(*)
INTO session_count
FROM `tblSessions`;
```

La variable session_count contient maintenant la valeur renvoyée par la requête **SELECT** qui compte le nombre de lignes dans la table des sessions. Il est aussi possible de mettre les valeurs dans de multiples variables comme dans l'exemple suivant :

Sélectionnez

```
DECLARE user VARCHAR(15);
DECLARE pass VARCHAR(32);

SELECT `strUserName`, `strPassword`
INTO user, pass
FROM `tblUsers`
LIMIT 2, 1;
```

V. Contrôler le flux de votre procédure stockée

Maintenant que nous avons configuré des variables et leur avons assigné des valeurs, nous pouvons contrôler le flux de notre procédure pour décider de ce que nous allons faire par la suite. MySQL supporte une flopée de contrôleurs de flux que vous allez utiliser comme vous pourrez le voir dans d'autres langages de programmation.

V-A. La commande IF

La commande **IF** est bien sûr le contrôle de flux le plus connu et est vraiment similaire dans MySQL comme partout ailleurs. La commande commence par **IF**, suivi d'une opération logique qui retourne vrai ou faux et se termine avec la première ligne qui finit par **THEN**. Les actions qui ont besoin de s'exécuter après la condition vraie peuvent ensuite être définies et une commande **ELSE** peut être définie au besoin. La commande IF se termine par une commande **END IF** ; (prenez note du point-virgule, source de beaucoup d'erreurs). La commande suivante vérifie si la variable « session_count » est supérieure à 0, si c'est le cas elle retourne vrai, sinon elle retourne faux.

Sélectionnez

```
IF session_count > 0 THEN
    SELECT 1;
ELSE
    SELECT 0;
END IF;
```

V-B. La commande CASE

MySQL inclut une implémentation de la commande switch case comme la plupart des langages de programmation usuels. Il ne peut y avoir aucun lien entre les cas comme il peut y en avoir en PHP, mais il y a un cas par défaut au cas où aucune condition ne correspond. Voici un switch case qui essaie d'attraper une variable nommée « catchme » basée sur ses conditions. Le cas par défaut doit être appelé, car « catchme » ne correspond à aucune des conditions définies dans le switch case.

Sélectionnez

```
DECLARE catchme INTEGER DEFAULT 10;

CASE catchme
   WHEN 2 THEN SET catchme = catchme + 2;
   WHEN 5 THEN SET catchme = catchme + 5;
   ELSE
      SET catchme = 50;
END CASE
```

V-C. La boucle WHILE

La boucle **WHILE**, outil commun hautement utilisé dans grand nombre de langages de programmation. MySQL n'y fait pas exception. La syntaxe est vraiment similaire aux autres la rendant facile à utiliser, mais voici un résumé. Les mots clés principaux que vous devez vous rappeler pour utiliser une boucle while dans MySQL sont **WHILE**, **DO** et **END WHILE**;. Quand vous définissez une boucle while vous pouvez lui donner un label qui peut être utilisé par plusieurs commandes pour contrôler la boucle. Le label vient avant la définition de la boucle while et après la fin de la boucle. L'exemple simple suivant crée une boucle WHILE appelée « iterwhile » qui s'itère tant que la variable « iter » est inférieure à 9.

Sélectionnez

```
DECLARE iter INTEGER DEFAULT 0;

iterwhile: WHILE iter < 9 DO
   SET iter = iter + 1;
END WHILE iterwhile;
```

V-D. La boucle LOOP

La commande **LOOP** ultra simple fait juste ça. Elle boucle indéfiniment jusqu'à ce que vous lui demandiez d'arrêter avec la commande **LEAVE**. Comme la boucle **WHILE**, elle peut être nommée grâce à un label dans sa définition afin d'être ciblée ensuite par des commandes pour la contrôler. Voici une commande **LOOP** appelée « iterloop » qui fait la même chose que l'exemple **WHILE** précédent, mais utilise une commande **IF** et une commande **LEAVE** pour s'arrêter.

Sélectionnez

```
DECLARE iter INTEGER DEFAULT 0;

iterloop: LOOP
   IF iter < 9 THEN
      SET iter = iter + 1;
   ELSE
      LEAVE iterloop;
   END IF;
END LOOP iterloop;
```

V-E. La boucle REPEAT

La commande **REPEAT** est essentiellement une boucle **WHILE** qui vérifie si les conditions correspondent à la fin de chaque itération contrairement au **WHILE** qui vérifie en début. Cela veut dire que quand vous utilisez la boucle **REPEAT** elle sera exécutée au moins une fois. Voici l'équivalent **REPEAT** de **WHILE** et **LOOP** vus précédemment :

Sélectionnez

```
DECLARE iter INTEGER DEFAULT 0;

iterrepeat: REPEAT
   SET iter = iter + 1;
UNTIL iter < 9
END REPEAT iterrepeat;
```

V-F. Les commandes LEAVE et ITERATE

Les commandes **LEAVE** et **ITERATE** sont similaires aux commandes **break** et **continue** utilisées dans d'autres langages de programmation. Elles peuvent être appliquées à n'importe quelle boucle **WHILE**, **LOOP** ou **REPEAT** qui sont actuellement actives en y faisant référence au label donné à la boucle. L'exemple suivant utilise les commandes **LEAVE** et **ITERATE** dans une **LOOP** appelée « testloop » :

Sélectionnez

```
DECLARE iter INTEGER DEFAULT 0;

testloop: LOOP
   IF iter = 10 THEN
      SET iter = iter + 12;
      ITERATE testloop;
   END IF;
   IF iter > 126 THEN
      LEAVE testloop;
   END IF;
   SET iter = iter + 1;
END LOOP testloop;
```

VI. Conclusion

Un mix de ces commandes peut vraiment ajouter de la puissance à vos codes SQL. Pas seulement, car elles peuvent réduire le trafic réseau, mais parce que vous pouvez réduire le nombre d'appels à votre base de données significativement, rendant les choses un peu plus organisées. Si vous commencez seulement à utiliser ces techniques et avez des difficultés avec les erreurs de syntaxe SQL, veillez à bien vérifier vos commandes individuellement avant de les mettre ensemble dans une procédure. Gardez un œil sur le point-virgule aussi, il peut causer une belle somme de problèmes quand il n'est pas utilisé au bon endroit donc assurez-vous de pouvoir différencier une commande d'une autre.

VII. Liens

Vous pouvez aussi aller voir mes autres traductions.

Vous avez aimé ce tutoriel ? Alors partagez-le en cliquant sur les boutons suivants : Partager

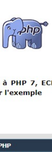
Copyright © 2009 Joris CROZIER. Aucune reproduction, même partielle, ne peut être faite de ce site ni de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

PHP 8 : voici un aperçu des nouveautés attendues dans la prochaine version majeure du langage, selon un développeur PHP

C++
C#
Visual Basic .NET
JavaScript
PHP

Popularité des langages de programmation : Objective-C en perte de vitesse d'après TIOBE, Python a dépassé Java pour la première fois en termes d'utilisation sur les projets open source GitHub

Quelles sont les choses appréciées par les utilisateurs finaux, mais qui ne le sont pas forcément des développeurs ?



Introduction à PHP 7, ECMASCRIPT 6 et VUE.JS par l'exemple

Contacter le responsable de la rubrique PHP

Nous contacter | Participez | Hébergement | Informations légales | Partenaire : Hébergement Web

© 2000-2020 - www.developpez.com