LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Stored Procedure / Function

LBS 4

25.09.2020

.

Inhalt

Einleitung

Stored Procedures dienen zum Zusammenfassen von wiederkehrenden Befehlen. Diese werden in der Datenbank abgelegt und können jederzeit aufgerufen werden. Somit kann die Funktionalität des DBMS erweitert werden und ermöglicht einen schnellen Zugriff auf die Prozedur.

Stored Procedures / Functions sind also kleine Programme, die von extern aufgerufen werden. Jeder große Datenbankhersteller implementiert seine eigenen Sprachen zum Ablegen der Funktionen. Während Oracle PL/SQL verwendet, kommt bei Microsoft Transact SQL zum Einsatz. MySQL verwendet den 2003 Standard, der von der ISO definiert wurde.

Die Datenbank als Speicherort bietet hier eigene Vorteile. Wenn Sie in heterogenen Umgebungen arbeiten, erhält man plattformunabhängigen Code, der von jedem Client ausgeführt werden kann. Weiters werden die Befehle nicht immer wieder über das Netzwerk versendet, somit wird Bandbreite gespart. Prozeduren erhöhen auch die Datenintegrität, weil sichergestellt ist, dass immer derselbe Code ausgeführt wird. Als Nachteil kann man den erhöhten Ressourcenverbrauch sehen, der bei einer großen Anzahl von parallelen Zugriffen an einer anderen Stelle fehlen könnte.

Der Delimiter

Mit dem Delimiter werden normalerweise die Befehle begrenzt. Als Standard wird das Semikolon ; verwendet. Allerdings kann es vorkommen, dass das Semikolon auch in der Prozedur verwendet werden muss und somit weiß die Datenbank nicht, wann die Prozedur zu Ende ist. Darum sollte man vor dem Erstellen den Delimiter neu setzen und am Ende der Prozedur wieder zurücksetzen.

```
DELIMITER $$
CREATE PROCEDURE prozedurname()
  BEGIN
    ...
  END$$
DELIMITER ;
```

Unterschied zwischen einer Prozedur und einer Funktion

Prozeduren können mehrere INPUT sowie OUTPUT Parameter verwenden, während eine Funktion nur einen Rückgabewert liefert. Funktionen können in einer SQL-Anweisung verwendet werden. Prozeduren müssen das Schlüsselwort CALL verwenden.

Elemente einer Prozedur

Prozeduren können mittels Telnet/SSH Konsole oder der SQL-Konsole eingegeben werden. Als Ergebnis werden ein oder mehrere Datensätze über den OUT-Parameter zurückgegeben.

Jede Prozedur muss nach nachstehendem Muster aufgebaut werden.

```
CREATE PROCEDURE prozedurname()
BEGIN
…..
END;
```

Der Aufruf erfolgt mit dem Schlüsselwort CALL prozedurname()

```
CALL prozedurname();
```

Prozeduren können auch Parameter enthalten, die beim Aufruf übergeben werden müssen. Parameter können als IN, OUT oder INOUT angegeben werden.

IN: ist der Standard und wird beim Aufruf übergeben. IN ist „schreibgeschützt“ und kann nicht verändert werden

OUT: in diese Variable werden Daten geschrieben, welche in der Prozedur gesetzt werden.

INOUT: kann als Eingabe- und Ausgabevariable verwendet werden.

```
DELIMITER //
CREATE PROCEDURE calcNumber(IN myID INT, OUT numberOf INT)
BEGIN
SELECT COUNT(*) INTO numberOf FROM liefert
WHERE w_id = myID;
END//
DELIMITER ;
```

Ergebnisse können mit den Schlüsselwort INTO in die OUT-Parameter geschrieben werden.

Zum Deklarieren von Variablen wird das Schlüsselwort DECLARE verwendet. Es muss der Name sowie der Datentyp angegeben werden, zusätzlich kann der Schalter DEFAULT verwendet werden.

```
DECLARE maxNumber DEC(10,2) DEFAULT 0.0;
DECLARE x, y INT DEFAULT 0;
```

Zum Initialisieren wird das Schlüsselwort SET verwendet.

```
SET total = 10;
```

Die Gültigkeit der Variablen ist auf den Bereich BEGIN und END beschränkt.

Aufgerufen wird die Prozedur mit den geforderten Parametern.
Beim OUT-Parameter wird das @ vorgestellt.

```
CALL calcNumber(2, @numberOf);
```

Der Befehl SELECT gibt das Ergebnis zurück.

```
SELECT @numberOf
```

Wenn die Prozedur keine Parameter enthält und trotzdem eine Ausgabe erfolgen soll, muss das Select im Code erfolgen.

```
DELIMITER $$

CREATE PROCEDURE GeNumberOfOrders()
BEGIN
        DECLARE maxOrder INT DEFAULT 0;

    SELECT COUNT(*)
    INTO maxOrder
    FROM orders;

    SELECT maxOrder;
END$$

DELIMITER ;
```

Zum Ausführen der Prozedur wird der CALL Parameter verwendet.

```
CALL GetNumberOfOrders();
```

In der Ausgabe ist dann die Anzahl der Bestellungen zu sehen.

Erweiterte Prozeduren

Wie in jeder prozeduralen Programmiersprache können auch hier Entscheidungen und Schleifen verwendet werden. Zusätzlich kann auch die Ausnahmebehandlung mittels Exception durchgeführt werden.

Entscheidungen

IF-THEN-ELSE sowie SWITCH-CASE können im Block zwischen BEGIN und END verwendet werden. Die Struktur muss wie nachstehend aufgebaut sein.

```
IF Bedingung THEN
   statements;
END IF;
```

Das nachstehende Beispiel zeigt eine mögliche Verwendung.

```
DELIMITER $$

CREATE PROCEDURE OrderLevel(
               IN customerID INT,
               OUT level VARCHAR(20))
BEGIN
    DECLARE maxOrder INT DEFAULT 0;
    SELECT COUNT(*)
    INTO maxOrder
    FROM orders
    WHERE customer_id = customerID;

        IF maxOrder > 100 THEN
               SET level = 'GOLD-USER'

        ELSEIF maxOrder > 1000 THEN
               SET level = 'PLATIN-USER'
        ELSE
               SET level = 'NORMAL-USER'
        END IF
END$$

DELIMITER ;
```

Mit CALL wird die Prozedur ausgeführt und mit SELECT das Ergebnis ausgegeben.

```
CALL OrderLevel(12, @level);
SELECT @level;
```

Schleifen

Man kann in MySQL eine einfache LOOP-Schleife oder kopf- und fußgesteuerte-Schleifen verwenden. Diese können als einfache Konstrukte oder in verschachtelten Strukturen verwendet werden.

Schleifen werden bei MySQL zum Iterieren in Attributen, mehrmaliges Ausführen von CODE sowie zum Filtern in Ergebnissen aus SELECT-Statements verwendet.

LOOP Statement

Die einfachste Form ist der LOOP. Der Aufbau ist nachstehend angeführt.
Das [label:] ist optional und muss nicht verwendet werden.

```
[label:] LOOP
        statements
    END LOOP
[label];
```

Das Schlüsselwort LOOP initiiert die Schleife und END LOOP begrenzt den auszuführenden Code. Labels dienen zum Identifizieren in verschachtelten Schleifen und können auch als Kontrollelement verwendet werden.

LOOP-Statements müssen eine Abbruchbedingung enthalten, andernfalls gibt es eine Endlosschleife.

Das Schlüsselwort LEAVE verlässt die Schleife und verhindert so die Endlosausführung.

```
LEAVE label;
```

Hier wird der Name des Labels zum Beenden der Schleife verwendet.

Beispiel:

```
SET i=1;
countLoop: LOOP
    SET i=i+1;
    IF i=10 THEN
            LEAVE countLoop;
    END IF;
END LOOP countLoop;
```

Während mit LEAVE die Schleife verlassen wird, kann mit ITERATE die Schleife zu einem Zeitpunkt abgebrochen und dann wieder fortgesetzt werden. Das Verhalten ist mit CONTINUE wie bei C zu vergleichen.

Das nachstehende Beispiel erzeugt eine Prozedur, die nur ungerade Zahlen von 0 bis 10 in einer SELECT-Anweisung ausgibt.

```
-- DROP PROCEDURE IF EXISTS ShowOdd;
DELIMITER //
CREATE PROCEDURE ShowOdd()
BEGIN
DECLARE i INT;
SET i=0;
SET str = ''
countOdd: LOOP
    IF i>=10 THEN          /*letzte Zahl - Schleife verlassen*/
         LEAVE countOdd;
    ELSEIF MOD(i,2)=0 THEN /*gerade Zahl - Schleife wiederholen*/
                       SET i=i+1;
         ITERATE countOdd;
    END IF;
    SELECT CONCAT(i,': ungerade Zahl') AS 'ungerade Zahl';
        SET i=i+1;
END LOOP countOdd;
END//
delimiter ;

call ShowOdd()
```

Versuchen Sie mit Grundlage dieses Beispiels die Prozedur so umzuschreiben, dass alle ungeraden Zahlen als Ergebnis angezeigt werden.

	ungerade Zahlen
▶	1,3,5,7,9,

Weiters ist es möglich eine WHILE-Schleife die einer DO-WHILE oder eine LOOP … UNTIL Schleife zu verwenden.

```
    [label:] WHILE expression DO
        statements
    END WHILE [label]
```

Auch hier kann das Label zur Steuerung der Schleife verwendet werden.

Elemente einer Funktion

Funktionen dienen in einer Datenbank zum Berechnen von Attributen. Diese unterscheiden sich zum einem von Prozeduren, dass eine Funktion mindestens einen Rückgabewert besitzt. Der Aufbau ähnelt dem Aufbau einer Prozedur. Genauso wie bei der Prozedur soll am Beginn der Delimiter geändert werden.

```
CREATE FUNCTION calc()
  RETURNS ...
  BEGIN
    ...
  END;
```

Der Aufruf der Funktion wird mit einer SELECT-Anweisung durchgeführt.

```
SELECT * FROM calc();
```

Alle CODE-Elemente einer Prozedur können auch in einer Funktion verwendet werden!