



MySQL 5.0 存儲過程

MySQL 5.0 新特性系列 第一部分



MySQL 技術白皮書

Peter Gultzan

March, 2005 翻

譯：陳朋奕 西安電
子科技大學

2005-5-6

(聲明：屬於個人翻譯，不涉及任何商業目的，支援國內MySQL發展，請轉載時注明出處，謝謝)



Table of Contents

目錄(目錄不做翻譯了，因為基本都是專有名詞)

Introduction	3
A Definition and an Example	3
Why Stored Procedures	4
Why MySQL Statements are Legal in a Procedure Body.....	8
Characteristics Clauses	10
Parameters.....	13
The New SQL Statements	15
Scope	16
Loops.....	21
Error Handling.....	29
Cursors	35
Security	41
Functions	43
Metadata.....	44
Details	48
Style.....	52
Tips when writing long routines.....	63
Bugs	64
Feature Requests	65
Resources	65
Conclusion.....	66
About MySQL	66

Introduction

本書是為需要瞭解5.0版本新特性的MySQL老用戶而寫的。簡單的來說是介紹了“存儲過程、觸發器、視圖、資訊架構視圖”，這是介紹MySQL 5.0新特性叢書的第一集。希望這 本書能像內行專家那樣與您進行對話，用簡單的問題、例子讓你學到需要的知識。為了達到這樣的目的，我會從每一個細節開始慢慢的為大家建立概念，最後會給大家展示較大的實用用例，在學習之前也許大家會認為這個用例很難，但是只要跟著課程去學，相信很快就能掌握。

*Conventions and Styles*約定和編程風格

每次我想要演示實際代碼時，我會對mysql用戶端的螢幕就出現的代碼進行調整，將字體改成Courier，使他們看起來與普通文本不一樣。在這裏舉個例子：

```
mysql> DROP FUNCTION f;
Query OK, 0 rows affected (0.00 sec)
```

如果實例比較大，則需要在某些行和段落間加注釋，同時我會用將“<--”符號放在頁面的右邊以表示強調。例如：

```
mysql> CREATE PROCEDURE p ()
-> BEGIN
-> /* This procedure does nothing */           <--
-> END;
Query OK, 0 rows affected (0.00 sec)
```

有時候我會將例子中的"mysql>"和"->"這些系統顯示去掉，你可以直接將代碼複製到mysql用戶端程式中（如果你現在所讀的不是電子版的，可以在mysql.com網站下載相關腳本）

所以的例子都已經在Suse 9.2 Linux、Mysql 5.0.3公共版上測試通過。在您閱讀本書的時候，Mysql已經有更高的版本，同時能支持更多OS了，包括Windows，Sparc，HP-UX。因此這裏的例子將能正常的運行在您的電腦上。但如果運行仍然出現故障，可以諮詢你認識的資深Mysql用戶，以得到長久的支持和幫助。

A Definition and an Example 定義及實例

存儲過程是一種存儲在書庫庫中的程式（就像正規語言裏的副程式一樣），準確的來說，MySQL支援的“routines（常式）”有兩種：一是我們說的存儲過程，二是在其他SQL語句中 可以返回值的函數（使用起來和Mysql預裝載的函數一樣，如pi()）。我在本書裏面會更經常使用 存儲過程，因為這是我們過去的習慣，相信大家也會接受。

一個存儲過程包括名字，參數列表，以及可以包括很多SQL語句的SQL語句集。
在這裏對局部變數，異常處理，迴圈控制和IF條件句有新的語法定義。
下面是一個包括存儲過程的實例聲明：
(譯注：為了方便閱讀，此後的程式不添任何中文注釋)

```
CREATE PROCEDURE procedure1          /* name 存儲過程名*/
(IN parameter1 INTEGER)              /* parameters 參數*/
BEGIN                                /* start of block 語句塊頭*/
  DECLARE variable1 CHAR(10);        /* variables變數聲明 */
  IF parameter1 = 17 THEN             /* start of IF IF條件開始*/
    SET variable1 = 'birds';         /* assignment 賦值*/
  ELSE
    SET variable1 = 'beasts';        /* assignment 賦值*/
  END IF;                             /* end of IF IF結束*/
  INSERT INTO table1 VALUES (variable1); /* statement SQL語句*/
END                                  /* end of block 語句塊結束*/
```

下面我將會介紹你可以利用存儲過程做的工作的所有細節。同時我們將介紹新的資料庫對象——觸發器，因為觸發器和存儲過程的關聯是必然的。

Why Stored Procedures 為什麼要用存儲過程

由於存儲過程對於MySQL來說是新的功能，很自然的在使用時你需要更加注意。畢竟，在此之前沒有任何人使用過，也沒有很多大量的有經驗的用戶來帶你走他們走過的路。然而你應該開始考慮把現有程式（可能在伺服器應用程式中，用戶自定義函數（UDF）中，或是腳本中）轉移到存儲過程中來。這樣做不需要原因，你不得不去做。

存儲過程是已經被認證的技術！雖然在Mysql中它是新的，但是相同功能的函數在其他DBMS中早已存在，而它們的語法往往是相同的。因此你可以從其他人那裏獲得這些概念，也有很多你可以諮詢或者雇用的經驗用戶，還有許多第三方的文檔可供你閱讀。

存儲過程會使系統運行更快！雖然我們暫時不能在Mysql上證明這個優勢，用戶得到的體驗也不一樣。我們可以說的就是Mysql伺服器在緩存機制上做了改進，就像Prepared statements（預處理語句）所做的那樣。由於沒有編譯器，因此SQL存儲過程不會像外部語言（如C）編寫的程式運行起來那麼快。但是提升速度的主要方法卻在於能否降低網路資訊流量。如果你需要處理的是需要檢查、迴圈、多語句但沒有用戶交互的重複性任務，你就可以使用保存在伺服器上的存儲過程來完成。這樣在執行任務的每一步時，伺服器與用戶端之間就沒那麼多的資訊來往了。

存儲過程是可複用的組件！想像一下如果你改變了主機的語言，這對存儲過程不會產生影響，因為它是資料庫邏輯而不是應用程式。存儲過程是可以移植的！當你用SQL編寫存儲過程時，你就知道它可以運行在Mysql支援的任何平臺上，不需要你額外添加運行環境包，也不需要為程式在作業系統中執行設置許可，或者為你的不同型號的電腦配置不同的包。這就是與Java、C或PHP等外部語言相比使用SQL語句的優勢。不過，使用外部語言常式的好處還是很好的選擇，它們只是沒有以上的優點而已。



存儲過程將被保存！如果你編寫好了一個程式，例如顯示銀行事物處理中的支票撤銷，那想要瞭解支票的人就可以找到你的程式。它會以源代碼的形式保存在資料庫中。這將使資料和處理資料的進程有意義的關聯這可能跟你在課上聽到的規劃論中說的一樣。

存儲過程可以移植！Mysql完全支持SQL 2003標準。某些資料庫（如DB2、Mimer）同樣支援。但也有部分不支援的，如Oracle、SQL Server不支援。我們將會給予足夠幫助和工具，使為其他DBMS編寫的代碼能更容易轉移到Mysql上。

Setting up with MySQL 5.0 設置並開始MySQL 5.0服務

通過mysql_fix_privilege_tables或者~/mysql-5.0/scripts/mysql_install_db來開始MySQL服務 作為

我們練習的準備工作的一部分，我假定MySQL 5.0已經安裝。如果沒有資料庫管理員為你安裝好資料庫以及其他軟體，你就需要自己去安裝了。不過你很容易忘掉一件事，那就是你需要有一個名為mysql.proc的表。

在安裝了最新版本後，你必須運行mysql_fix_privilege_tables或者mysql_install_db（只需要運行其中一個就夠了）——不然存儲過程將不能工作。我同時啓用在root身份後運行一個非正式的SQL腳本，如下：

```
mysql>source/home/pgulutzan/mysql-5.0/scripts/mysql_prepare_privilege_tables_for_5.sql
```

Starting the MySQL Client 啓動MySQL用戶端

這是我啓動mysql用戶端的方式。你也許會使用其他方式，如果你使用的是二進位版本或者是Windows系統的電腦，你可能會在其他子目錄下運行以下程式：

```
pgulutzan@mysqlcom:~> /usr/local/mysql/bin/mysql --user=root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.3-alpha-debug
```

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

在演示中，我將會展示以root身份登陸後的mysql用戶端返回的結果，這樣意味著我有極大的特權。

Check for the Correct Version 核對版本

爲了確認使用的MySQL的版本是正確的，我們要查詢版本。我有兩種方法確認我使用的是5.0版本：

```
SHOW VARIABLES LIKE 'version';
or
SELECT VERSION();
```

例如：

```
mysql> SHOW VARIABLES LIKE 'version';
```

```
+-----+-----+
| Variable_name | Value          |
+-----+-----+
| version       | 5.0.3-alpha-debug |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT VERSION();
```

```
+-----+
| VERSION()      |
+-----+
| 5.0.3-alpha-debug |
+-----+
1 row in set (0.00 sec)
```

當看見數字'5.0.x'後就可以確認存儲過程能夠在這個用戶端上正常工作。

The Sample "Database" 示例資料庫

現在要做的第一件事是創建一個新的資料庫然後設定為默認資料庫實現這個步驟的SQL語句如下：

```
CREATE DATABASE db5;
USE db5;
```

例如：

```
mysql> CREATE DATABASE db5;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> USE db5;
Database changed
```

在這裏要避免使用有重要資料的實際的資料庫然後我們創建一個簡單的工作表。實現這個步驟的SQL語句如下：

```
mysql> CREATE DATABASE db5;
Query OK, 1 row affected (0.01 sec)
```

```
mysql> USE db5;
Database changed
```

```
mysql> CREATE TABLE t (s1 INT);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO t VALUES (5);
Query OK, 1 row affected (0.00 sec)
```

你會發現我只在表中插入了一列。這樣做的原因是我要保持表的簡單，因為在這裏並不需要展示查詢資料的技巧，而是教授存儲過程，不需要使用大的資料表，因為它本身已經夠複雜了。

這就是示例資料庫，我們將從這個名字為t的只包含一列的表開始

Pick a Delimiter 選擇分隔符號

現在我們需要一個分隔符號，實現這個步驟的SQL語句如下：

```
DELIMITER //
```

例如：

```
mysql> DELIMITER //
```

分隔符號是你通知mysql用戶端你已經完成輸入一個SQL語句的字元或字串符號。一直以來我們都使用分號“;”，但在存儲過程中，這會產生不少問題，因為存儲過程中有許多語句，所以每一個都需要一個分號因此你需要選擇一個不太可能出現在你的語句或程式中的字串作為分隔符號。我曾用過雙斜杠“//”，也有人用豎線“|”。我曾見過在DB2程式中使用“@”符號的，但我不喜歡這樣。你可以根據自己的喜好來選擇，但是在這個課程中為了更容易理解，你最好選擇跟我一樣。如果以後要恢復使用“;”（分號）作為分隔符號，輸入下面語句就可以了：

```
"DELIMITER ;//".
```

CREATE PROCEDURE Example 創建程式實例

```
CREATE PROCEDURE p1 () SELECT * FROM t; //
```

也許這是你使用Mysql創建的第一個存儲過程。假如是這樣的話，最好在你的日記中記下這個重要的里程碑。

```
CREATE PROCEDURE p1 () SELECT * FROM t; // <--
```

SQL語句存儲過程的第一部分是 “CREATE PROCEDURE”：

```
CREATE PROCEDURE p1 () SELECT * FROM t; // <--
```

第二部分是過程名，上面新存儲過程的名字是p1。

Digression: Legal Identifiers 題外話：合法識別字的問題

存儲過程名對大小寫不敏感，因此‘P1’和‘p1’是同一個名字，在同一個資料庫中你將不能給兩個存儲過程取相同的名字，因為這樣將會導致重載。某些DBMS允許重載（Oracle支持），但是MySQL不支持（譯者話：希望以後會支持吧。）。你可以採取

“資料庫名.存儲過程名”這樣的折中方法，如“db5.p1”。存儲過程名可以分開，它可以包括空白字元，其長度限制為64個字元，但注意不要使用MySQL內建函數的名字，如果這樣做了，在調用時將會出現下面的情況：

```
mysql> CALL pi();
Error 1064 (42000): You have a syntax error.
mysql> CALL pi ();
Error 1305 (42000): PROCEDURE does not exist.
```

在上面的第一個例子裏，我調用的是一個名字叫pi的函數，但你必須在調用的函數名後加上空格，就像第二個例子那樣。

```
CREATE PROCEDURE p1 () SELECT * FROM t; // <--
```

其中“（）”是“參數列表”。

CREATE PROCEDURE 語句的第三部分是參數列表。通常需要在括弧內添加參數。例子中的存儲過程沒有參數，因此參數列表是空的——所以我只需要鍵入空括弧，然而這是必須的。

```
CREATE PROCEDURE p1 () SELECT * FROM t; // <--
```

"SELECT * FROM t;"是存儲過程的主體。然後到了語句的最後一個部分了，它是存儲過程的主

體，是一般的SQL語句。過程體中語句

"SELECT * FROM t;"包含一個分號，如果後面有語句結束符號（//）時可以不寫這個分號。如果你還記得我把這部分叫做程式的主體將會是件好事，因為（body）這個詞是大家使用的技術上的術語。通常我們不會將SELECT語句用在存儲過程中，這裏只是為了演示。所以使用這樣的語句，能在調用時更好的看出程式是否正常工作。

Why MySQL Statements are Legal in a Procedure Body

什麼MySQL語句在存儲過程體中是合法的？

什麼樣的SQL語句在Mysql存儲過程中才是合法的呢？你可以創建一個包含INSERT, UPDATE, DELETE, SELECT, DROP, CREATE, REPLACE等等的語句。你唯一需要記住的是如果代碼中包含MySQL擴充功能，那麼代碼將不能移植。在標準SQL語句中：任何資料庫定義語言都是合法的，如：

```
CREATE PROCEDURE p () DELETE FROM t; //
```

SET、COMMIT以及ROLLBACK也是合法的，如：

```
CREATE PROCEDURE p () SET @x = 5; // MySQL的附加功
```

能：任何資料操作語言的語句都將合法。

```
CREATE PROCEDURE p () DROP TABLE t; //
```

MySQL擴充功能：直接的SELECT也是合法的：

```
CREATE PROCEDURE p () SELECT 'a'; //
```

順便提一下，我將存儲過程中包括DDL語句的功能稱為MySQL附加功能的原因是在SQL標準中把這個定義為非核心的，即可選元件。

在過程體中有一個約束，就是不能有對常式或表操作的資料庫操作語句。例如下面的例子就是非法的：

```
CREATE PROCEDURE p1 ()  
CREATE PROCEDURE p2 () DELETE FROM t; //
```

下面這些對MySQL 5.0來說全新的語句，在過程體中是非法的：

```
CREATE PROCEDURE, ALTER PROCEDURE, DROP PROCEDURE, CREATE FUNCTION,  
DROP FUNCTION, CREATE TRIGGER, DROP TRIGGER. 不過你可以使用  
"CREATE PROCEDURE db5.p1 () DROP DATABASE db5//", 但是類似"USE database"語句  
也是非法的，因為MySQL假定默認資料庫就是過程的工作場所。
```

Call the Procedure 調用存儲過程

1. 現在我們就可以調用一個存儲過程了，你所需要輸入的全部就是**CALL**和你過程名以及一個括弧再一次強調，括弧是必須的當你調用例子裏面的**p1**過程時，結果是螢幕返回了**t**表的內容

```
mysql> CALL p1() //  
+-----+  
| s1 |  
+-----+  
| 5 |  
+-----+  
1 row in set (0.03 sec)  
Query OK, 0 rows affected (0.03 sec)
```

因為過程中的語句是"SELECT * FROM t;"

2. 其他實現方式

```
mysql> CALL p1() //
```

和下面語句的執行效果一樣：

```
mysql> SELECT * FROM t; //
```

所以，你調用**p1**過程就相當於你執行了下面語句：

```
"SELECT * FROM t;"
```

好了，主要的知識點“創建和調用過程方法”已經清楚了。我希望你能對自己說這相當簡單。但是很快我們就有一系列的練習，每次都加一條子句，或者改變已經存在的子句。那樣在寫複雜部件前我們將會有許多可用的子句。

Characteristics Clauses 特徵子句

1.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL                <--
NOT DETERMINISTIC          <--
SQL SECURITY DEFINER        <--
COMMENT 'A Procedure'      <--
SELECT CURRENT_DATE, RAND() FROM t //
```

這裏我給出的是一些能反映存儲過程特性的子句。子句內容在括弧之後，主體之前。這些子句都是可選的，他們有什麼作用呢？

2.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL                <--
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t //
```

很好，這個LANGUAGE SQL子句是沒有作用的。僅僅是爲了說明下面過程的主體使用SQL語言編寫。這條是系統默認的，但你在這裏聲明是有用的，因爲某些DBMS（IBM的DB2）需要它，如果你關注DB2的相容問題最好還是用上。此外，今後可能會出現除SQL外的其他語言支援的存儲過程。

3.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL
NOT DETERMINISTIC          <--
SQL SECURITY DEFINER
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t //
```

下一個子句，NOT DETERMINISTIC，是傳遞給系統的資訊。這裏一個確定過程的定義就是那些每次輸入一樣輸出也一樣的程式。在這個案例中，既然主體中含有SELECT語句，那返回肯定是未知的因此我們稱其NOT DETERMINISTIC。但是MySQL內置的優化程式不會注意這個，至少在現在不注意。

4.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER          <--
COMMENT 'A Procedure'
SELECT CURRENT_DATE, RAND() FROM t //
```

下一個子句是SQL SECURITY，可以定義為SQL SECURITY DEFINER或SQL SECURITY INVOKER。這就進入了許可權控制的領域了，當然我們在後面將會有測試許可權的例子。

SQL SECURITY DEFINER意味著在調用時檢查創建過程用戶的許可權（另一個選項是SQL SECURITY INVOKER）。

現在而言，使用SQL SECURITY DEFINER指令告訴MySQL伺服器檢查創建過程的用戶就可以了，當過程已經被調用，就不檢查執行調用過程的用戶了。而另一個選項（INVOKER）則是告訴伺服器在這一步仍然要檢查調用者的許可權。

5.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT 'A Procedure'          <--
SELECT CURRENT_DATE, RAND() FROM t //
```

COMMENT 'A procedure' 是一個可選的注釋說明。最後，注釋子句會跟程序定義存儲在一

起。這個沒有固定的標準，我在文中會指出沒有固定標準的語句，不過幸運的是這些在我們標準的SQL中很少。

6.

```
CREATE PROCEDURE p2 ()
LANGUAGE SQL
NOT DETERMINISTIC
SQL SECURITY DEFINER
COMMENT "
SELECT CURRENT_DATE, RAND() FROM t //
```

上面過程跟下面語句是等效的：

```
CREATE PROCEDURE p2 ()
SELECT CURRENT_DATE, RAND() FROM t //
```

特徵子句也有預設值，如果省略了就相當於：

```
LANGUAGE SQL NOT DETERMINISTIC SQL SECURITY DEFINER COMMENT "
```

一些題外話

: 調用p20//的結果

```
mysql> call p2() //
+-----+-----+
| CURRENT_DATE | RAND() |
+-----+-----+
| 2004-11-09   | 0.7822275075896 |
+-----+-----+
1 row in set (0.26 sec)
Query OK, 0 rows affected (0.26 sec)
```

當調用過程p2時，一個SELECT語句被執行返回我們期望獲得的亂數。

: 不會改變的sql_mode

```
mysql> set sql_mode='ansi' //
mysql> create procedure p3(select'a' || 'b' //
mysql> set sql_mode="" //
mysql> call p3() //
+-----+
| 'a' || 'b' |
+-----+
| ab          |
+-----+
```

MySQL在過程創建時會自動保持運行環境。例如：我們需要使用兩條豎線來連接字串但是 這只有在sql mode為ansi的時候才合法。如果我們將sql mode改為non-ansi，不用擔心，它仍然能工作，只要它第一次使用時能正常工作。

Exercise 練習

Question 問題

如果你不介意練習一下的話，試試能否不看後面的答案就能處理這些請求。

創建一個過程，顯示`Hello world`。用大約5秒時間去思考這個問題，既然你已經學到了這裏，這個應該很簡單。當你思考問題的時候，我們再隨機選擇一些剛才講過的東西復習：DETERMINISTIC（確定性）子句是反映輸出和輸入依賴特性的子句……調用過程使用 CALL 過程名（參數列表）方式。好了，我猜時間也到了。

Answer 答案

好的，答案就是在過程體中包含"SELECT 'Hello, world'"語句

```
mysql> CREATE PROCEDURE p4 () SELECT 'Hello, world' //  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CALL p4()//  
+-----+  
| Hello, world |  
+-----+  
| Hello, world |  
+-----+  
1 row in set (0.00 sec)  
Query OK, 0 rows affected (0.00 sec)
```

Parameters 參數

讓我們更進一步的研究怎麼在存儲過程中定義參數

1. CREATE PROCEDURE p5
() ...
2. CREATE PROCEDURE p5
([IN] name data-type) ...
3. CREATE PROCEDURE p5
(OUT name data-type) ...
4. CREATE PROCEDURE p5
(INOUT name data-type) ...

回憶一下前面講過的參數列表必須在存儲過程名後的括弧中。上面的第一個例子中的參數列表是空的，第二個例子中有一個輸入參數。這裏的詞IN可選，因為默認參數為IN（input）。第三個例子中有一個輸出參數，第四個例子中有一個參數，既能作為輸入也可以作為輸出。

IN example 輸入的例子

```
mysql> CREATE PROCEDURE p5(p INT) SET @x = p //
Query OK, 0 rows affected (0.00 sec)
mysql> CALL p5(12345)//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x//
+-----+
| @x   |
+-----+
| 12345 |
+-----+
1 row in set (0.00 sec)
```

這個IN的例子演示的是有輸入參數的過程。在過程體中我將會話變數x設定為參數p的值。然後調用過程，將12345傳入參數p。選擇顯示會話變數@x，證明我們已經將參數值12345傳入。

OUT example 輸出的例子

```
mysql> CREATE PROCEDURE p6 (OUT p INT)
-> SET p = -5 //
mysql> CALL p6(@y)//
mysql> SELECT @y//
+-----+
| @y   |
+-----+
| -5   |
+-----+
```

這是另一個例子。這次的p是輸出參數，然後在程序呼叫中將p的值傳入會話變數@y中。在過程體中，我們給參數賦值-5，在調用後我們可以看出，OUT是告訴DBMS值是從過程中傳出的。同樣我們可以用語句"SET @y = -5;"來達到同樣的效果

Compound Statements 複合語句

現在我們展開的詳細分析一下過程體：

```
CREATE PROCEDURE p7 ()
BEGIN
  SET @a = 5;
  SET @b = 5;
  INSERT INTO t VALUES (@a);
  SELECT s1 * @a FROM t WHERE s1 >= @b;
END; //      /* I won't CALL this. 這個語句將不會被調用*/
```

完成過程體的構造就是BEGIN/END塊。這個BEGIN/END語句塊和Pascal語言中的BEGIN/END是基本相同的，和C語言的框架是很相似的。我們可以使用塊去封裝多條語句。在這個例子中，我們使用了多條設定會話變數的語句，然後完成了一些insert和select語句。如果你的過程體中有許多條語句，那麼你就需要BEGIN/END塊了。BEGIN/END塊也被稱為複合語句，在這裏你可以進行變數定義和流程控制。

The New SQL Statements 新SQL語句

Variables 變數

在複合語句中聲明變數的指令是DECLARE。

(1) **Example with two DECLARE statements** 兩個DECLARE語句的例子

```
CREATE PROCEDURE p8 ()
BEGIN
  DECLARE a INT;
  DECLARE b INT;
  SET a = 5;
  SET b = 5;
  INSERT INTO t VALUES (a);
  SELECT s1 * a FROM t WHERE s1 >= b;
END; //      /* I won't CALL this */
```

在過程中定義的變數並不是真正的定義，你只是在BEGIN/END塊內定義了而已（譯注：也就是形參）。注意這些變數和會話變數不一樣，不能使用修飾符@。你必須清楚的在BEGIN/END塊中聲明變數和它們的類型。變數一旦聲明，你就能在任何能使用會話變量、文字、列名的地方使用。

(2) **Example with no DEFAULT clause and SET statement** 沒有默認子句和設定語句的例子

```
CREATE PROCEDURE p9 ()
BEGIN
  DECLARE a INT /* there is no DEFAULT clause */;
  DECLARE b INT /* there is no DEFAULT clause */;
  SET a = 5; /* there is a SET statement */
  SET b = 5; /* there is a SET statement */
  INSERT INTO t VALUES (a);
  SELECT s1 * a FROM t WHERE s1 >= b;
END; //      /* I won't CALL this */
```

有很多初始化變數的方法。如果沒有默認的子句，那麼變數的初始值為NULL。你可以在任何時候使用SET語句給變數賦值。

(3) Example with DEFAULT clause 含有DEFAULT子句的例子

```
CREATE PROCEDURE p10 ()
BEGIN
  DECLARE a, b INT DEFAULT 5;
  INSERT INTO t VALUES (a);
  SELECT s1 * a FROM t WHERE s1 >= b;
END; //
```

我們在這裏做了一些改變，但是結果還是一樣的。在這裏使用了DEFAULT子句來設定初始值，這就不需要把DECLARE和SET語句的實現分開了。

(4) Example of CALL 調用的例子

```
mysql> CALL p10() //
+-----+
| s1 * a |
+-----+
|    25 |
|    25 |
+-----+
2 rows in set (0.00 sec)
Query OK, 0 rows affected (0.00 sec)
```

結果顯示了過程能正常工作

(5) Scope 作用域

```
CREATE PROCEDURE p11 ()
BEGIN
  DECLARE x1 CHAR(5) DEFAULT 'outer';
  BEGIN
    DECLARE x1 CHAR(5) DEFAULT 'inner';
    SELECT x1;
  END;
  SELECT x1;
END; //
```

現在我們來討論一下作用域的問題。例子中有嵌套的BEGIN/END塊，當然這是合法的。同時包含兩個變數，名字都是x1，這樣也是合法的。內部的變數在其作用域內享有更高的優先權。當執行到END語句時，內部變數消失，此時已經在其作用域外，變數不再可見了，因此在存儲過程外再也找不到這個聲明了的變數，但是你可以通過OUT參數或者將其值指派給會話變數來保存其值

調用作用域例子的過程：

```
mysql> CALL p11()//
```

```
+-----+
| x1   |
+-----+
| inner|
+-----+
+-----+
| x1   |
+-----+
| outer|
+-----+
```

我們看到的結果時第一個SELECT語句檢索到最內層的變數，第二個檢索到第二層的變數

Conditions and IF-THEN-ELSE 條件式和IF-THEN-ELSE

1. 現在我們可以寫一些包含條件式的例子：

```
CREATE PROCEDURE p12 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  IF variable1 = 0 THEN
    INSERT INTO t VALUES (17);
  END IF;
  IF parameter1 = 0 THEN
    UPDATE t SET s1 = s1 + 1;
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END; //
```

這裏是一個包含IF語句的過程。裏面有兩個IF語句，一個是IF 語句 END IF，另一個是IF 語句 ELSE 語句 END IF。我們可以在這裏使用複雜的過程，但我會儘量使其簡單讓你能更容易弄清楚。

2.

```
CALL p12 (0) //
```

我們調用這個過程，傳入值為0，這樣parameter1的值將為0。

3.

```
CREATE PROCEDURE p12 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;           <--
  IF variable1 = 0 THEN
    INSERT INTO t VALUES (17);
  END IF;
  IF parameter1 = 0 THEN
    UPDATE t SET s1 = s1 + 1;
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END; //
```

這裏變數variable1被賦值為parameter1加1的值，所以執行後變數variable1為1。

4.

```
CREATE PROCEDURE p12 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  IF variable1 = 0 THEN                     <--
    INSERT INTO t VALUES (17);
  END IF;
  IF parameter1 = 0 THEN
    UPDATE t SET s1 = s1 + 1;
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END; //
```

因為變數variable1值為1，因此條件"if variable1 = 0"為假，IF……END IF被跳過，沒有被執行。

5.

```
CREATE PROCEDURE p12 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  IF variable1 = 0 THEN
    INSERT INTO t VALUES (17);
  END IF;
  IF parameter1 = 0 THEN                   <--
    UPDATE t SET s1 = s1 + 1;
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END; //
```

到第二個IF條件，判斷結果為真，於是中間語句被執行了

6.

```
CREATE PROCEDURE p12 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  IF variable1 = 0 THEN
    INSERT INTO t VALUES (17);
  END IF;
  IF parameter1 = 0 THEN
    UPDATE t SET s1 = s1 + 1;          <--
  ELSE
    UPDATE t SET s1 = s1 + 2;
  END IF;
END;
```

因為參數parameter1值等於0，UPDATE語句被執行。如果parameter1值為NULL，則下一條UPDATE語句將被執行。現在表t中有兩行，他們都包含值5，所以如果我們調用p12，兩行的值會變成6。

7.

```
mysql> CALL p12(0)//
Query OK, 2 rows affected (0.28 sec)
```

```
mysql> SELECT * FROM t//
```

```
+-----+
| s1 |
+-----+
|  6 |
|  6 |
+-----+
```

```
2 rows in set (0.01 sec)
```

結果也是我們所期望的那樣。

CASE 指令

1.

```
CREATE PROCEDURE p13 (IN parameter1 INT)
BEGIN
  DECLARE variable1 INT;
  SET variable1 = parameter1 + 1;
  CASE variable1
    WHEN 0 THEN INSERT INTO t VALUES (17);
    WHEN 1 THEN INSERT INTO t VALUES (18);
    ELSE INSERT INTO t VALUES (19);
  END CASE;
END;
```

如果需要進行更多條件真假的判斷我們可以使用CASE語句。CASE語句使用和IF一樣簡單。我們可以參考上面的例子：

2.

```
mysql> CALL p13(1)//  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM t//
```

```
+-----+  
| s1 |  
+-----+  
|  6 |  
|  6 |  
| 19 |  
+-----+
```

3 rows in set (0.00 sec) 執行過程後，傳入值1，如上面例子，

值19被插入到表t中。 **Question** 問題

問題: CALL p13(NULL) //的作用是什麼？

另一個：這個CALL語句做了那些動作？

你可以通過執行後觀察SELECT做了什麼，也可以根據代碼判斷，在5秒內做出。

Answer 答案

```
mysql> CALL p13(NULL)//  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM t//
```

```
+-----+  
| s1 |  
+-----+  
|  6 |  
|  6 |  
| 19 |  
| 19 |  
+-----+
```

4 rows in set (0.00 sec)

答案是當你調用p13時，MySQL插入了另一條包含數值19的記錄。原因是變數variable1的值為NULL，CASE語句的ELSE部分就被執行了。希望這對大家有意義。如果你回答不出來，沒有問題，我們可以繼續向下走。

Loops 迴圈語句

```
WHILE ... END WHILE
LOOP ... END LOOP
REPEAT ... END REPEAT
GOTO
```

下面我們將會創建一些迴圈。我們有三種標準的迴圈方式：**WHILE**迴圈，**LOOP**迴圈以及**REPEAT**迴圈。還有一種非標準的迴圈方式：**GO TO**（譯者語：最好不要用吧，用了就使流程混亂）。

WHILE ... END WHILE

```
CREATE PROCEDURE p14 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  WHILE v < 5 DO
    INSERT INTO t VALUES (v);
    SET v = v + 1;
  END WHILE;
END; //
```

這是**WHILE**迴圈的方式。我很喜歡這種方式，它跟**IF**語句相似，因此不需要掌握很多新的語法。這裏的**INSERT**和**SET**語句在**WHILE**和**END WHILE**之間，當變數**v**大於5的時候迴圈將會退出。使用"**SET v = 0;**"語句使爲了防止一個常見的錯誤，如果沒有初始化，默認變數值爲**NULL**，而 **NULL**和任何值操作結果都爲**NULL**。

WHILE ... END WHILE example

```
mysql> CALL p14();//
Query OK, 1 row affected (0.00 sec)
```

以上就是調用過程**p14**的結果

不用關注系統返回是"**one row affected**" 還是 "**five rows affected**"，因爲這裏的計數只對最後一個**INSERT**動作進行計數。

WHILE ... END WHILE example: CALL

```
mysql> select * from t; //
+-----+
| s1 |
+-----+
....
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
+-----+
9 rows in set (0.00 sec)
```

調用後可以看到程式向資料庫中插入了5行。

REPEAT ... END REPEAT

```
CREATE PROCEDURE p15 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  REPEAT
    INSERT INTO t VALUES (v);
    SET v = v + 1;
  UNTIL v >= 5
  END REPEAT;
END; //
```

這是一個REPEAT迴圈的例子，功能和前面WHILE迴圈一樣。區別在於它在執行後檢查結果，而WHILE則是執行前檢查。（譯者語：可能等同於DO WHILE吧）

REPEAT ... END REPEAT: look at the UNTIL : UNTIL的作用

```
CREATE PROCEDURE p15 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  REPEAT
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    UNTIL v >= 5
  END REPEAT;
END; //
```

注意到UNTIL語句後面沒有分號，在這裏可以不寫分號，當然你加上額外的分號更好。

REPEAT ... END REPEAT: calling : 調用

```
mysql> CALL p15()//
Query OK, 1 row affected (0.00 sec)

mysql> SELECT COUNT(*) FROM t//
+-----+
| COUNT(*) |
+-----+
|      14 |
+-----+
1 row in set (0.00 sec)
```

我們可以看到調用p15過程後又插入了5行記錄

LOOP ... END LOOP

```
CREATE PROCEDURE p16 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN
      LEAVE loop_label;
    END IF;
  END LOOP;
END; //
```

以上是LOOP迴圈的例子。LOOP迴圈不需要初始條件，這點和WHILE迴圈相似，同時它又和REPEAT迴圈一樣也不需要結束條件。

LOOP ... END LOOP: with IF and LEAVE 包含IF和LEAVE的LOOP迴圈

```
CREATE PROCEDURE p16 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN                                <--
      LEAVE loop_label;
    END IF;
  END LOOP;
END; //
```

在迴圈內部加入IF語句，在IF語句中包含LEAVE語句。這裏LEAVE語句的意義是離開循環。LEAVE的語法是LEAVE加迴圈語句標號，關於迴圈語句的標號問題我會在後面進一步講解。

LOOP ... END LOOP: calling : 調用

```
mysql> CALL p16();//
Query OK, 1 row affected (0.00 sec)

mysql> SELECT COUNT(*) FROM t//
+-----+
| COUNT(*) |
+-----+
|      19 |
+-----+
1 row in set (0.00 sec)
```

調用過程p16後，結果是另5行被插入表t中。

Labels 標號

```
CREATE PROCEDURE p17 ()
label_1: BEGIN
  label_2: WHILE 0 = 1 DO LEAVE label_2; END
WHILE;
  label_3: REPEAT LEAVE label_3; UNTIL 0 =0
END REPEAT;
  label_4: LOOP LEAVE label_4; END LOOP;
END; //
```

最後一個迴圈例子中我使用了語句標號。現在這裏有一個包含4個語句標號的過程的例子。我們可以在BEGIN、WHILE、REPEAT或者LOOP語句前使用語句標號，語句標號只能在合法的語句前面使用。因此"LEAVE label_3"意味著離開語句標號名定義為label_3的語句或複合語句。

End Labels 標號結束符

```
CREATE PROCEDURE p18 ()
label_1: BEGIN
  label_2: WHILE 0 = 1 DO LEAVE label_2; END
WHILE label_2;
  label_3: REPEAT LEAVE label_3; UNTIL 0 =0
END REPEAT label_3 ;
  label_4: LOOP LEAVE label_4; END LOOP
label_4 ;
END label_1 ; //
```

你也可以在語句結束時使用語句標號，和在開頭時使用一樣。這些標號結束符並不是十分有用。它們是可選的。如果你需要，他們必須和開始定義的標號名字一樣當然為了有良好的編程習慣，方便他人閱讀，最好還是使用標號結束符。

LEAVE and Labels 跳出和標號

```
CREATE PROCEDURE p19 (parameter1 CHAR)
label_1: BEGIN
  label_2: BEGIN
    label_3: BEGIN
      IF parameter1 IS NOT NULL THEN
        IF parameter1 = 'a' THEN
          LEAVE label_1;
        ELSE BEGIN
          IF parameter1 = 'b' THEN
            LEAVE label_2;
          ELSE
            LEAVE label_3;
          END IF;
        END;
      END IF;
    END IF;
  END;
END;
```

LEAVE語句使程式跳出複雜的複合語句。

ITERATE 迭代

如果目標是ITERATE（迭代）語句的話，就必須用到LEAVE語句

```
CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP
    IF v = 3 THEN
      SET v = v + 1;
      ITERATE loop_label;
    END IF;
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN
      LEAVE loop_label;
    END IF;
  END LOOP;
END;
```

ITERATE（迭代）語句和LEAVE語句一樣也是在迴圈內部的迴圈引用，它有點像C語言中的“Continue”，同樣它可以出現在複合語句中，引用複合語句標號，ITERATE（迭代）意思是重新開始複合語句。

那我們啟動並觀察下面這個迴圈，這是個需要迭代過程的迴圈：

ITERATE: Walking through the loop 深入迴圈

```

CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP                                <--
    IF v = 3 THEN
      SET v = v + 1;
      ITERATE loop_label;
    END IF;
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN
      LEAVE loop_label;
    END IF;
  END LOOP;
END; //

```

讓這個已經定義了標號的迴圈運行起來。

ITERATE: Walking through the loop

```

CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP                                <--
    IF v = 3 THEN
      SET v = v + 1;
      ITERATE loop_label;
    END IF;
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN
      LEAVE loop_label;
    END IF;
  END LOOP;
END; //

```

v的值變成3，然後我們把它增加到4。

ITERATE: walking through the loop

```

CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
  loop_label: LOOP
    IF v = 3 THEN
      SET v = v + 1;
      ITERATE loop_label;                                <--
    END IF;
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN

```

```
    LEAVE loop_label;
  END IF;
END LOOP;
END; //
```

然後開始ITERATE（迭代）過程。

ITERATE: walking through the loop

```
CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
loop_label: LOOP                                <--
  IF v = 3 THEN
    SET v = v + 1;
    ITERATE loop_label;
  END IF;
  INSERT INTO t VALUES (v);
  SET v = v + 1;
  IF v >= 5 THEN
    LEAVE loop_label;
  END IF;
END LOOP;
END; //
```

這裏的ITERATE（迭代）讓迴圈又回到了迴圈的頭部。

ITERATE: walking through the loop

```
CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
loop_label: LOOP
  IF v = 3 THEN
    SET v = v + 1;
    ITERATE loop_label;
  END IF;
  INSERT INTO t VALUES (v);
  SET v = v + 1;
  IF v >= 5 THEN
    LEAVE loop_label;                                <--
  END IF;
END LOOP;
END; //
```

當v的值變為5時，程式將執行LEAVE語句

ITERATE: walking through the loop

```
CREATE PROCEDURE p20 ()
BEGIN
  DECLARE v INT;
  SET v = 0;
loop_label: LOOP
  IF v = 3 THEN
    SET v = v + 1;
```

```
        ITERATE loop_label;
    END IF;
    INSERT INTO t VALUES (v);
    SET v = v + 1;
    IF v >= 5 THEN
        LEAVE loop_label;
    END IF;
    END LOOP;
END; //                                     <-- LEAVE的結果
```

就是跳出迴圈，使運行指令到達複合語句的最後一步。

GOTO

```
CREATE PROCEDURE p...
BEGIN
...
LABEL label_name;
...
GOTO label_name;
...
END;
```

MySQL的存儲過程中可以使用GOTO語句。雖然這不是標準SQL語句，而且在這裏建立標號的方法也和慣例中的不一樣。由於爲了和其他DBMS相容，這個語句會慢慢被淘汰，所以我們在MySQL參考手冊中沒有提及。

Grand combination 大組合

```
CREATE PROCEDURE p21
(IN parameter_1 INT, OUT parameter_2 INT)
LANGUAGE SQL DETERMINISTIC SQL SECURITY INVOKER
BEGIN
    DECLARE v INT;
    label goto_label; start_label: LOOP
        IF v = v THEN LEAVE start_label;
        ELSE ITERATE start_label;
        END IF;
    END LOOP start_label;
    REPEAT
        WHILE 1 = 0 DO BEGIN END;
    END WHILE;
    UNTIL v = v END REPEAT;
    GOTO goto_label;
END; //
```

上面例子中的語句包含了我們之前講的所有語法，包括參數列表，特性參數，BEGIN/END塊複合語句，變數聲明，IF，WHILE，LOOP，REPEAT，LEAVE，ITERATE，GOTO。這是一個荒謬的存儲過程，我不會運行它，因爲裏面有無盡的迴圈。但是裏面的語法卻十分合法。

這些是新的流程控制和變數聲明語句。下面我們將要接觸更多新的東西。

Error Handling 異常處理

後面幾頁的資訊摘要

Sample Problem 問題樣例

Handlers 異常處理器

Conditions 條件

好了，我們現在要講的是異常處理

1. Sample Problem: Log Of Failures 問題樣例：故障記錄

當INSERT失敗時，我希望能將其記錄在日誌檔中我們用來展示出錯處理的問題樣例是很普通的。我希望得到錯誤的記錄。當INSERT失敗時，我想在另一個檔中記下這些錯誤的資訊，例如出錯時間，出錯原因等。我對插入特別感興趣的原因是它將違反外鍵關聯的約束

2. Sample Problem: Log Of Failures (2)

```
mysql> CREATE TABLE t2
  s1 INT, PRIMARY KEY (s1))
  engine=innodb;//
mysql> CREATE TABLE t3 (s1 INT, KEY (s1),
  FOREIGN KEY (s1) REFERENCES t2 (s1))
  engine=innodb;//
mysql> INSERT INTO t3 VALUES (5);//
...
ERROR 1216 (23000): Cannot add or update a child row: a foreign key
constraint fails （這裏顯示的是系統的出錯資訊）
```

我開始要創建一個主鍵表，以及一個外鍵表。我們使用的是InnoDB，因此外鍵關聯檢查是打開的。然後當我向外鍵表中插入非主鍵表中的值時，動作將會失敗。當然這種條件下可以很快找到錯誤號1216。

3. Sample Problem: Log Of Failures

```
CREATE TABLE error_log (error_message
CHAR(80))//
```

下一步就是建立一個在做插入動作出錯時存儲錯誤的表。

4. Sample Problem: Log Of Errors

```
CREATE PROCEDURE p22 (parameter1 INT)
BEGIN
  DECLARE EXIT HANDLER FOR 1216
  INSERT INTO error_log VALUES
  (CONCAT('Time: ',current_date,
  ', Foreign Key Reference Failure For
  Value = ',parameter1));
  INSERT INTO t3 VALUES (parameter1);
END;//
```

上面就是我們的程式。這裏的第一個語句DECLARE EXIT HANDLER是用來處理異常的。意思是如果錯誤1215發生了，這個程式將會在錯誤記錄表中插入一行。EXIT意思是當動作成功提交後退出這個複合語句。

5. Sample Problem: Log Of Errors

```
CALL p22 (5) //
```

調用這個存儲過程會失敗，這很正常，因為5值並沒有在主鍵表中出現。但是沒有錯誤資訊返回因為出錯處理已經包含在過程中了。t3表中沒有增加任何東西，但是error_log表中記錄下了一些資訊，這就告訴我們 `INSERT into table t3` 動作失敗。

DECLARE HANDLER syntax 聲明異常處理的語法

```
DECLARE  
{ EXIT | CONTINUE }  
HANDLER FOR  
{ error-number | { SQLSTATE error-string } | condition }  
SQL statement
```

上面就是錯誤處理的用法，也就是一段當程式出錯後自動觸發的代碼。MySQL允許兩種處理器，一種是EXIT處理，我們剛才所用的就是這種。另一種就是我們將要演示的，CONTINUE處理，它跟EXIT處理類似，不同在於它執行後，原主程序仍然繼續運行，那麼這個複合語句就沒有出口了。

1. **DECLARE CONTINUE HANDLER example** CONTINUE處理例子

```
CREATE TABLE t4 (s1 int,primary key(s1));/  
CREATE PROCEDURE p23 ()  
BEGIN  
  DECLARE CONTINUE HANDLER  
  FOR   SQLSTATE '23000' SET @x2 = 1;  
  SET @x = 1;  
  INSERT INTO t4 VALUES (1);  
  SET @x = 2;  
  INSERT INTO t4 VALUES (1);  
  SET @x = 3;  
  END; //
```

這是MySQL參考手冊上的CONTINUE處理的例子，這個例子十分好，所以我把它拷貝到這裏。通過這個例子我們可以看出CONTINUE處理是如何工作的。

2. **DECLARE CONTINUE HANDLER** 聲明CONTINUE異常處理

```
CREATE TABLE t4 (s1 int,primary key(s1));/  
CREATE PROCEDURE p23 ()  
BEGIN  
  DECLARE CONTINUE HANDLER  
  FOR   SQLSTATE '23000' SET @x2 = 1;          <--  
  SET @x = 1;  
  INSERT INTO t4 VALUES (1);  
  SET @x = 2;  
  INSERT INTO t4 VALUES (1);  
  SET @x = 3;  
  END; //
```

這次我將為SQLSTATE值定義一個處理程式。還記得前面我們使用的MySQL錯誤代碼1216嗎？事實上這裏的23000SQLSTATE是更常用的，當外鍵約束出錯或主鍵約束出錯就被調用了。

3. DECLARE CONTINUE HANDLER

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;                                <--
  INSERT INTO t4 VALUES (1);
  SET @x = 2;
  INSERT INTO t4 VALUES (1);
  SET @x = 3;
  END;//

```

這個存儲過程的第一個執行的語句是"SET @x = 1"。

4. DECLARE CONTINUE HANDLER example

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;
  INSERT INTO t4 VALUES (1);
  SET @x = 2;
  INSERT INTO t4 VALUES (1);                <--
  SET @x = 3;
  END;//

```

運行後值1被插入到主鍵表中。

5. DECLARE CONTINUE HANDLER

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;
  INSERT INTO t4 VALUES (1);
  SET @x = 2;                                <--
  INSERT INTO t4 VALUES (1);
  SET @x = 3;
  END;//

```

然後@x的值變為2。

6. DECLARE CONTINUE HANDLER example

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;

```

```

INSERT INTO t4 VALUES (1);
SET @x = 2;
INSERT INTO t4 VALUES (1);           <--
SET @x = 3;
END;//

```

然後程式嘗試再次往主鍵表中插入數值，但失敗了，因為主鍵有唯一性限制。

7. DECLARE CONTINUE HANDLER example

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;           <--
  SET @x = 1;
  INSERT INTO t4 VALUES (1);
  SET @x = 2;
  INSERT INTO t4 VALUES (1);
  SET @x = 3;
END;//

```

由於插入失敗，錯誤處理程式被觸發，開始進行錯誤處理。下一個執行的語句是錯誤處理的語句，@x2被設為2。

8. DECLARE CONTINUE HANDLER example

```

CREATE TABLE t4 (s1 int,primary key(s1));//
CREATE PROCEDURE p23 ()
BEGIN
  DECLARE CONTINUE HANDLER
  FOR   SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;
  INSERT INTO t4 VALUES (1);
  SET @x = 2;
  INSERT INTO t4 VALUES (1);
  SET @x = 3;                               <--
END;//

```

到這裏並沒有結束，因為這是CONTINUE異常處理。所以執行返回到失敗的插入語句之後，繼續執行將@x設定為3動作。

9. DECLARE CONTINUE HANDLER example

```

mysql> CALL p23();//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @x, @x2//
+-----+-----+
| @x  | @x2  |
+-----+-----+
| 3   | 1    |
+-----+-----+
1 row in set (0.00 sec)

```

運行過程後我們觀察@x的值，很確定的可以知道是3，觀察@x2的值，為1。從這裏可以判斷程式運行無誤，完全按照我們的思路進行。大家可以花點時間去調整錯誤處理器，讓檢查放在語句段的首部，而不是放在可能出現錯誤的地方，雖然那樣看起來程式很紊亂，跳來跳去的感覺。但是這樣的代碼很安全也很清楚。

1. DECLARE CONDITION

```
CREATE PROCEDURE p24 ()
BEGIN
  DECLARE `Constraint Violation`
  CONDITION FOR SQLSTATE '23000';
  DECLARE EXIT HANDLER FOR
  `Constraint Violation` ROLLBACK;
  START TRANSACTION;
  INSERT INTO t2 VALUES (1);
  INSERT INTO t2 VALUES (1);
  COMMIT;
END; //
```

這是另外一個錯誤處理的例子，在前面的基礎上修改的。事實上你可給SQLSTATE或者錯誤代碼其他的名字，你就可以在處理中使用自己定義的名字了。下面看看它是怎麼實現的：我把表t2定義為InnoDB表，所以對這個表的插入操作都會ROLLBACK（回滾）ROLLBACK（回滾事務）也是恰好會發生的。因為對主鍵插入兩個同樣的值會導致SQLSTATE 23000 錯誤發生，這裏SQLSTATE 23000是約束錯誤。

2. DECLARE CONDITION 聲明條件

```
CREATE PROCEDURE p24 ()
BEGIN
  DECLARE `Constraint Violation`
  CONDITION FOR SQLSTATE '23000';
  DECLARE EXIT HANDLER FOR
  `Constraint Violation` ROLLBACK;
  START TRANSACTION;
  INSERT INTO t2 VALUES (1);
  INSERT INTO t2 VALUES (1);
  COMMIT;
END; //
```

這個約束錯誤會導致ROLLBACK（回滾事務）和SQLSTATE 23000錯誤發生。

3. DECLARE CONDITION

```
mysql> CALL p24();//
Query OK, 0 rows affected (0.28 sec)

mysql> SELECT * FROM t2//
Empty set (0.00 sec)
```

我們調用這個存儲過程看看結果是什麼，從上面結果我們看到表t2沒有插入任何記錄。全部事務都回滾了。這正是我們想要的。

4. DECLARE CONDITION

```
mysql> CREATE PROCEDURE p9 ()
-> BEGIN
-> DECLARE EXIT HANDLER FOR NOT FOUND BEGIN END;
-> DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN END;
-> DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
-> END; //
Query OK, 0 rows affected (0.00 sec)
```

這裏是三個預聲明的條件： NOT FOUND (找不到行), SQLEXCEPTION (錯誤),

SQLWARNING (警告或注釋)。因為它們是預聲明的，因此不需要聲明條件就可以使用。不過如果你去做這樣的聲明："DECLARE SQLEXCEPTION CONDITION ..."，你將會得到錯誤資訊提示。

Cursors 游標

游標實現功能摘要：

```
DECLARE cursor-name CURSOR FOR SELECT ...;
OPEN cursor-name;
FETCH cursor-name INTO variable [, variable];
CLOSE cursor-name;
```

現在我們開始著眼游標了。雖然我們的存儲過程中的游標語法還並沒有完整的實現，但是已經可以完成基本的事務如聲明游標，打開游標，從游標裏讀取，關閉游標。

1. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
  UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

我們看一下包含游標的存儲過程的新例子。

2. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
  UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

這個過程開始聲明了三個變數。附帶說一下，順序是十分重要的。首先要進行變數聲明，然後聲明條件，隨後聲明游標，再後面才是聲明錯誤處理器。如果你沒有按順序聲明，系統會提示錯誤資訊。

3. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;          <--
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
    UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

程式第二步聲明了游標cur_1，如果你使用過嵌入式SQL的話，就知道這和嵌入式SQL差不多。

4. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND          <--
    SET b = 1;                                     <--
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
    UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

最後進行的是錯誤處理器的聲明。這個CONTINUE處理沒有引用SQL錯誤代碼和SQLSTATE值。它使用的是NOT FOUND系統返回值，這和SQLSTATE 02000是一樣的。

5. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET b = 1;
  OPEN cur_1;                                     <--
  REPEAT
    FETCH cur_1 INTO a;
    UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

過程第一個可執行的語句是OPEN cur_1，它與SELECT s1 FROM t語句是關聯的，過程將執行SELECT s1 FROM t，返回一個結果集。

6. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;                                <--
  UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

這裏第一個FETCH語句會獲得一行從SELECT產生的結果集中檢索出來的值，然而表t中有多行，因此這個語句會被執行多次，當然這是因為語句在迴圈塊內。

7. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;                                          <--
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
  UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

最後當MySQL的FETCH沒有獲得行時，CONTINUE處理被觸發，將變數b賦值為1。

8. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
  UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;                                       <--
  SET return_val = a;
END;
```

到了這一步 UNTIL b=1 條件就為真，迴圈結束。在這裏我們可以自己編寫代碼關閉游標，也可以由系統執行，系統會在複合語句結束時自動關閉游標，但是最好不要太依賴系統的自動關閉行爲（譯注：這可能跟Java的Gc一樣，不可信）。

9. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
    UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;          <--
END;
```

這個常式中我們爲輸出參數指派了一個局部變數，這樣在過程結束後的結果仍能使用。

10. Cursor Example

```
CREATE PROCEDURE p25 (OUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR SELECT s1 FROM t;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
  SET b = 1;
  OPEN cur_1;
  REPEAT
    FETCH cur_1 INTO a;
    UNTIL b = 1
  END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
END;
```

```
mysql> CALL p25(@return_val)//
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @return_val//
+-----+
| @return_val |
+-----+
| 5           |
+-----+
1 row in set (0.00 sec)
```

上面是程序呼叫後的結果。可以看到return_val參數獲得了數值5，因爲這是表t的最後一行。由此可以知道游標工作正常，出錯處理也工作正常。

Cursor Characteristics 游標的特性

摘要：

READ ONLY 唯讀屬性
NOT SCROLLABLE 順序讀取
ASENSITIVE 敏感

在5.0版的MySQL中，你只可以從游標中取值，不能對其進行更新。因為游標是（READ ONLY）唯讀的。你可以這樣做：

```
FETCH cursor1 INTO variable1;  
UPDATE t1 SET column1 = 'value1' WHERE CURRENT OF cursor1;
```

游標也是不可以滾動的，只允許逐一讀取下一行，不能在結果集中前進或後退。下面代碼就是錯誤的：

```
FETCH PRIOR cursor1 INTO variable1;  
FETCH ABSOLUTE 55 cursor1 INTO variable1;
```

同時也不允許在已打開游標進行操作的表上執行updates事務，因為游標是（ASENSITIVE）敏感的。因為如果你不阻止update事務，那就不知道結果會變成什麼。如果你使用的是InnoDB而不是MyISAM存儲引擎的話，結果也會不一樣。

Security 安全措施

摘要

Privileges (1) CREATE ROUTINE
Privileges (2) EXECUTE
Privileges (3) GRANT SHOW ROUTINE?
Privileges (4) INVOKERS AND DEFINERS

這裏我們要討論一些關於特權和安全相關的問題。但因為在MySQL安全措施的功能並沒有完全，所以我們不會對其進行過多討論。

2. Privileges CREATE ROUTINE

```
GRANT CREATE ROUTINE  
ON database-name . *  
TO user(s)  
[WITH GRANT OPTION];  
現在用root就可以了
```

在這裏要介紹的特權是CREATE ROUTINE，它不僅同其他特權一樣可以創建存儲過程和函數，還可以創建視圖和表。Root用戶擁有這種特權，同時還有ALTER ROUTINE特權。

2. Privileges EXECUTE

```
GRANT EXECUTE ON p TO peter  
[WITH GRANT OPTION];
```

上面的特權是決定你是否可以使用或執行存儲過程的特權，過程創建者默認擁有這個特權。

3. Privileges SHOW ROUTINE?

```
GRANT SHOW ROUTINE ON db6.* TO joey  
[WITH GRANT OPTION];
```

因為我們已經有控制視圖的特權了：GRANT SHOW VIEW。所以在這個基礎上，為了保證相容，日後可能會添加GRANT SHOW ROUTINE特權。這樣做是不太符合標準的，在寫本書的時候，

MySQL還沒實現這個功能。

4. *Privileges Invokers and Definers* 特權調用者和定義者

```
CREATE PROCEDURE p26 ()
SQL SECURITY INVOKER
SELECT COUNT(*) FROM t //
CREATE PROCEDURE p27 ()
SQL SECURITY DEFINER
SELECT COUNT(*) FROM t //
GRANT INSERT ON db5.* TO peter; //
```

現在我們測試一下SQL SECURITY子句吧。Security是我們前面提到的程式特性的一部分。你是root用戶，將插入權賦給了peter。然後使用peter登陸進行新的工作，我們看看peter可以怎麼使用 存儲過程，注意：peter沒有對表t的select權力，只有root用戶有。

5. *Privileges Invokers and Definers*

```
/* Logged on with current_user = peter */ 使用帳戶peter登陸
```

```
mysql> CALL p26();
ERROR 1142 (42000): select command denied to user
'peter'@'localhost' for table 't'
```

```
mysql> CALL p27();
+-----+
| COUNT(*) |
+-----+
|      1 |
+-----+
1 row in set (0.00 sec)
```

當peter嘗試調用含有調用保密措施的過程p26時會失敗。那是因為peter沒有對表的select的權力。但是當petre調用含有定義保密措施的過程時就能成功。原因是root有select權力，Peter有root的 權力，因此過程可以執行。

Functions 函數

Summary: 摘要

CREATE FUNCTION

Limitations of functions 函數的限制

我們已經很清楚可以在存儲過程中使用的元素了。下面我要講的是前面沒有提到的函數。

CREATE FUNCTION 創建函數

```
CREATE FUNCTION factorial (n DECIMAL(3,0))
  RETURNS DECIMAL(20,0)
  DETERMINISTIC
  BEGIN
  DECLARE factorial DECIMAL(20,0) DEFAULT 1;
  DECLARE counter DECIMAL(3,0);
  SET counter = n;
  factorial_loop: REPEAT
    SET factorial = factorial * counter;
    SET counter = counter - 1;
  UNTIL counter = 1
  END REPEAT;
  RETURN factorial;
END //
```

(代碼來源："Understanding SQL's stored procedures"，這裏只是作為例子引用) 函數跟過程很相似，唯一需要指出的語法上的不同就是創建函數後必須有RETURN語句返回函數指定的類型值。

這個例子來自Jim Melton的大作，他是SQL standard committee的成員，"Understanding SQL's stored procedures"的作者。原例在書的223頁。我決定使用這個例子是因為它的規範性。

2. Examples

```
INSERT INTO t VALUES (factorial(pi)) //
SELECT s1, factorial (s1) FROM t //
UPDATE t SET s1 = factorial(s1)
WHERE factorial(s1) < 5 //
```

上面就是我們需要的函數，把它放到SQL語句中跟其他函數看起來是一樣的。如果能很好的處理，函數將是美妙的，就像樂曲中的小調一樣。不過，它們也有缺陷，那就是不能在函數中訪問表，這使它們不如存儲過程強大。

3. Limitations 限制

Illegal: 非法聲明：

```
ALTER 'CACHE INDEX' CALL COMMIT CREATE DELETE
DROP 'FLUSH PRIVILEGES' GRANT INSERT KILL
LOCK OPTIMIZE REPAIR REPLACE REVOKE
ROLLBACK SAVEPOINT 'SELECT FROM table'
'SET system variable' 'SET TRANSACTION'
SHOW 'START TRANSACTION' TRUNCATE UPDATE
```

不能訪問表的限制削弱了函數的功能，因此你不能夠進行資料操作、資料描述、特權轉化或是事務控制。但我們的工作主要是靠這些，也許以後會支援這些特性吧。

4. Limitations

合法聲明：

```
'BEGIN END' DECLARE IF ITERATE LOOP
REPEAT RETURN 'SET declared variable'
WHILE
```

利用函數你能做的全部就是設置變數，然後在控制流語句中使用它們。實際上這個功能很強大，但是離人們想要的卻還很遠。

Metadata元數據

摘要：

```
SHOW CREATE PROCEDURE / SHOW CREATE FUNCTION
SHOW PROCEDURE STATUS / SHOW FUNCTION STATUS
SELECT from mysql.proc
SELECT from information_schema
```

到這裏我們已經創建了很多過程了，它們也都保存在MySQL資料庫中。我們如果要查看MySQL實際上保存了什麼資訊，有四種實現方法，兩種使用SHOW語句，兩種使用SELECT語句。

1. Show

```
mysql> show create procedure p6//
+-----+-----+-----+
| Procedure | sql_mode | Create Procedure |
+-----+-----+-----+
| p6       |          | CREATE PROCEDURE |
|          |          | `db5`.`p6` (out p |
|          |          | int) set p = -5  |
+-----+-----+-----+
1 row in set (0.00 sec)
```

第一種獲得元資料資訊的方法是執行SHOW CREATE PROCEDURE。這同SHOW CREATE TABLE以及其他類似MySQL語句一樣。它並不返回你創建過程時設定的返回值，但在大部分情況下已經夠用了。

2. Show

```
mysql> SHOW PROCEDURE STATUS LIKE 'p6'//
+-----+-----+-----+-----+
| Db  | Name | Type      | Definer      | ...
+-----+-----+-----+-----+
| db5 | p6  | PROCEDURE | root@localhost | ...
+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

第二種獲得metadata資訊的方法是執行SHOW PROCEDURE STATUS。這種方法可以返回更多資訊的細節。

3. *SELECT from mysql.proc*

```
SELECT * FROM mysql.proc WHERE name = 'p6'//
```

```
+-----+-----+-----+-----+
| db  | name | type      | specific_name | ...
+-----+-----+-----+-----+
| db5  | p6 | PROCEDURE | p6              | ...
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

第三種方法是執行SELECT語句，它能提供最多的資訊。

4. *SELECT from information_schema*: 我最喜歡的方式。

第四種方法是"SELECT ... FROM information_schema. ..."。我特別傾向使用"ANSI/ISO標準"的方式去完成工作。我相信這是種好實現方式，因為其他方式可能會出現錯誤。當然有不同MySQL用戶會堅持不同的觀點，也有不同理由，認真的持懷疑態度的看看這些理由。

- 1.其他DBMS，例如SQL Server 2000，使用information_schema。只有MySQL有SHOW方式。
- 2.我們訪問mysql.proc的特權是沒有保障的，因為我們有訪問information_schema視圖的特權，每個用戶都有內隱的對information_schema資料庫的SELECT特權。
- 3.SELECT功能很多，可以計算運算式，分組，排序，產生可以獲取資訊的結果集。而這些功能SHOW沒有。

現在瞭解我喜歡它的原因了吧，那下面我們舉幾個簡單的例子來演示一下。

首先我會使用SELECT information_schema來顯示information_schema常式中有哪些列。

```
mysql> SELECT TABLE_NAME, COLUMN_NAME, COLUMN_TYPE FROM
INFORMATION_SCHEMA.COLUMNS
```

```
-> WHERE TABLE_NAME = 'ROUTINES';//
```

```
+-----+-----+-----+-----+
| TABLE_NAME | COLUMN_NAME      | COLUMN_TYPE |
+-----+-----+-----+-----+
| ROUTINES    | SPECIFIC_NAME    | varchar(64) |
| ROUTINES    | ROUTINE_CATALOG  | longtext    |
| ROUTINES    | ROUTINE_SCHEMA   | varchar(64) |
| ROUTINES    | ROUTINE_NAME     | varchar(64) |
| ROUTINES    | ROUTINE_TYPE     | varchar(9)   |
| ROUTINES    | DTD_IDENTIFIER   | varchar(64) |
| ROUTINES    | ROUTINE_BODY     | varchar(8)   |
| ROUTINES    | ROUTINE_DEFINITION | longtext    |
| ROUTINES    | EXTERNAL_NAME    | varchar(64) |
| ROUTINES    | EXTERNAL_LANGUAGE | varchar(64) |
| ROUTINES    | PARAMETER_STYLE  | varchar(8)   |
| ROUTINES    | IS_DETERMINISTIC | varchar(3)   |
| ROUTINES    | SQL_DATA_ACCESS  | varchar(64) |
| ROUTINES    | SQL_PATH         | varchar(64) |
| ROUTINES    | SECURITY_TYPE     | varchar(7)   |
| ROUTINES    | CREATED          | varbinary(19) |
| ROUTINES    | LAST_ALTERED     | varbinary(19) |
| ROUTINES    | SQL_MODE         | longtext    |
| ROUTINES    | ROUTINE_COMMENT  | varchar(64) |
| ROUTINES    | DEFINER          | varchar(77) |
+-----+-----+-----+-----+
20 rows in set (0.01 sec)
```

漂亮吧？當我們想要查看information_schema視圖時，我們從information_schema中select資訊，



就跟從TABLES和COLUMNS獲取一樣。獲取的是元資料的資料元素。這裏我們看到的是我在資料庫db6中定義的存儲過程。

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.ROUTINES
-> WHERE ROUTINE_SCHEMA = 'db6';//
+-----+
| COUNT(*) |
+-----+
|      28 |
+-----+
1 row in set (0.02 sec)
```

現在進一步看看我們第一個創建的過程p1，我們重新格式mysql用戶端輸出視窗。

```
| SPECIFIC_NAME | ROUTINE_CATALOG | ROUTINE_SCHEMA
+-----+-----+-----+
| p19          | NULL           | p19
| ROUTINE_NAME  | ROUTINE_TYPE   | DTD_IDENTIFIER
+-----+-----+-----+
| p19          | PROCEDURE      | NULL
| ROUTINE_BODY | ROUTINE_DEFINITION | EXTERNAL_NAME
+-----+-----+-----+
| SQL          | select * from t | NULL
| EXTERNAL_LANGUAGE | PARAMETER_STYLE | IS_DETERMINISTIC
+-----+-----+-----+
| NULL         | SQL            | NO
| SQL_DATA_ACCESS | SQL_PATH      | SECURITY_TYPE | CREATED
+-----+-----+-----+
| CONTAINS SQL  | NULL          | DEFINER       | 2004-12-19
| CREATED       | LAST_ALTERED  | SQL_MODE
+-----+-----+-----+
| 2004-12-19 15:00:26 | 2004-12-19 15:00:26 |
| ROUTINE_COMMENT | DEFINER
+-----+-----+
|                | root@localhost
```

Access control for the ROUTINE_DEFINITION column

ROUTINE_DEFINITION列的訪問控制

在INFORMATION_SCHEMA中的ROUTINE_DEFINITION列是由過程或函數組成過程體獲得的。這裏可能會有敏感資訊，因此只對過程創建者可見。

CURRENT_USER <> INFORMATION_SCHEMA.ROUTINES.DEFINER : 如果對它使用SELECT的用戶不是創建它的用戶，那麼mysql將返回NULL值，而不是ROUTINE_DEFINITION列。這個檢查功能在作此書時還沒實現。

Additional clause in SHOW PROCEDURE STATUS顯示過程狀態子句

SHOW PROCEDURE STATUS中的輔助子句

既然我已經列出INFORMATION_SCHEMA.ROUTINES中的列，就可以回去解釋SHOW PROCEDURE STATUS的新細節，語法是：

```
SHOW PROCEDURE STATUS [WHERE condition];
```

語句中的條件判斷和SELECT語句的一樣：如果為真，則在輸出中返回行。但這裏有個需要



特別注意的部分：在WHERE子句中你必須使用INFORMATION_SCHEMA列的名字，結果顯示的是SHOW PROCEDURE STATUS欄位的名字。例如：

```
mysql> SHOW PROCEDURE STATUS WHERE Db = 'p';
ERROR 1054 (42S22): Unknown column 'Db' in 'where clause'

mysql> SHOW PROCEDURE STATUS WHERE ROUTINE_NAME = 'p';
+-----+-----+-----+
|Db  |Name |Type  |Definer  |...
+-----+-----+-----+
|db11|p    |PROCEDURE|root@localhost|...
+-----+-----+-----+
1 row in set (0.00 sec)
```

知道了這點，我們也許就能在MySQL認證考試中獲得高分，但是實際中從來不會用到。

Details 細節 後面

幾頁的摘要：

ALTER and DROP

Oracle / SQL Server / DB2 / ANSI comparison

Style

Bugs

Feature Requests

Resources

最後的環節中我們介紹了很多細節，但不會根據它們應該受到注意的程度去學習，這沒關係，因為在試錯中你會發現這些。

ALTER and DROP

```
ALTER PROCEDURE p6 COMMENT 'Unfinished' //
```

```
DROP PROCEDURE p6 //
```

Oracle Comparison 與Oracle的比較

Summary: 摘要

- Oracle允許在打開後再聲明。
MySQL必須在開始的時候聲明。

Oracle允許"CURSOR cursorname IS"這樣的聲明。
MySQL必須使用"DECLARE cursorname CURSOR"聲明。

Oracle不強制需要'()'。
MySQL必須有'()'。

Oracle允許在函數中訪問表元素。
MySQL不允許在函數中訪問表元素。

Oracle支持"packages"。
MySQL不支持"packages"。

如果你使用過Oracle的PL/SQL存儲過程的話，你會發現Oracle和MySQL的存儲過程有很多不同（譯者語：PL/SQL的功能比MySQL的存儲過程強大，也許這就是區別，哈哈）。我剛才指出的只是一些常見的區別。你也可以去<http://www.ispirer.com>網站閱讀更多的資料，這是一家提供將Oracle存儲過程轉換成MySQL存儲過程的產品的廠家。我並沒有說這家公司的產品很好的意思，因為我沒有用過。我只是感興趣已經對有人做出將其他DBMS的存儲過程遷移到MySQL上來的程式。

Tips for migration..資料遷移的技巧

把"a:=b"類似的賦值語句改成"SET a=b"。

將過程中的RETURN語句改為"LEAVE label_at_start"，這裏的label_at_start是你最初為存儲過程設定的標記。如：

[在Oracle存儲過程中]

```
CREATE PROCEDURE ... RETURN; ...
```

[在MySQL存儲過程中]

```
CREATE PROCEDURE () label_at_start: BEGIN ... LEAVE label_at_start; END
```

這一步驟僅僅在過程中需要，因為MySQL函數支援RETURN。

Side-By-Side 平行比較

Oracle	MySQL
CREATE PROCEDURE	CREATE PROCEDURE
sp_name	sp_name
AS	BEGIN
variable1 INTEGER	DECLARE variable1 INTEGER;
variable1 := 55	SET variable1 = 55;
END	END

與SQL Server的對比

摘要：

SQL Server參數名字必須以'@'開頭。

MySQL參數名是常規識別字。

SQL Server可以同時進行多個聲明，如："DECLARE v1 [data type], v2 [data type]"。

MySQL只允許每次聲明一個，如："DECLARE v1 [data type]; DECLARE v2 [data type]"。

SQL Server存儲過程體中沒有BEGIN ... END。

MySQL必須有BEGIN ... END語句。

SQL Server不需要以';'號結束語句。

MySQL必須使用';'號作為語句結束標誌，除了最後一條語句外。

SQL Server可以進行"SET NOCOUNT"設置和"IF @@ROWCOUNT"判斷，

MySQL沒有這些，但可以使用FOUND_ROWS()進行判斷。

SQL Server中使用"WHILE ... BEGIN"語句。

MySQL使用"WHILE ... DO"語句。

SQL Server允許使用"SELECT"進行指派。

MySQL只允許SET進行指派。

SQL Server允許在函數中訪問表。

MySQL不允許在函數中訪問表

Microsoft SQL Server的區別特別多，所以講Microsoft或Sybase的程式轉換成MySQL程式將會是個冗長的過程，而且區別都是在語法定義上的，所以轉換需要更多特別的技巧。

Some migration tips ... 一些遷移技巧

如果SQL Server中有名為@xxx的變數，你必須將其轉換，因為@在MySQL中並不代表過程變數，

而是總體變數，不要僅僅改成xxx，那樣會使其含義不明確。因為在資料庫的某個表中可能有一列的列名叫xxx，所以最好把@首碼改成你的自定義字元，如將@xxx改成var_xxx。

3. Side by Side 平行對比

SQL Server	MySQL
CREATE PROCEDURE	CREATE PROCEDURE
sp_procedure1	sp_procedure1
AS	()
DECLARE @x VARCHAR(100)	BEGIN
EXECUTE sp_procedure2 @x	DECLARE v__x VARCHAR(100);
DECLARE c CURSOR FOR	CALL sp_procedure2(v__x);
DECLARE c CURSOR FOR	SELECT * FROM t;
SELECT * FROM t	END
END	

與DB2的對比

DB2允許PATH（路徑）語句。

MySQL不允許PATH（路徑）語句。

DB2允許SIGNAL（信令）語句。

MySQL不允許SIGNAL（信令）語句。

DB2允許常式名的重載。

MySQL不允許對常式名的重載。

DB2有"label_x: ... GOTO label_x"語法。

MySQL有非正式的"label label_x; ... GOTO label_x"語法。

DB2允許函數訪問表。

MySQL不允許函數訪問表。

DB2存儲過程基本和MySQL一致，唯一的不同在於MySQL還沒有引進DB2的一些語句，還有就是DB2允許重載，因此DB2可以有兩個名字一樣的常式，通過常式的參數或返回類型來決定執行哪個，所以DB2存儲過程可以向下與MySQL的相容。

Some migration tips ... 一些遷移技巧 這裏的遷移基本不需要任何技巧。MySQL缺少SIGNAL語句，我們會在其他地方討論臨時工作區的問題。而對DB2的GOTO語句，我們直接用MySQL的GOTO代替就可以了。PATH（確定 DBMS尋找常式的資料庫目錄）問題只需要在常式名前加上首碼就可以避免了。關於函數訪問表的問題，我建議大家用OUT參數的存儲過程來代替就行了。

Side by Side 平行對比

DB2	MySQL
CREATE PROCEDURE	CREATE PROCEDURE
sp_name	sp_name
(parameter1 INTEGER)	(parameter1 INTEGER)
LANGUAGE SQL	LANGUAGE SQL
BEGIN	BEGIN
DECLARE v INTEGER;	DECLARE v INTEGER;
IF parameter1 >=5 THEN	IF parameter1 >=5 THEN
CALL p26();	CALL p26();
SET v = 2;	SET v = 2;
END IF;	END IF;
INSERT INTO t VALUES (v);	INSERT INTO t VALUES (v);
END @	END //



Standard Comparison 與SQL標準的比較

摘要：

Standard SQL requires: 標準SQL的要求
[跟DB2中的一樣]

MySQL的目標是支援以下兩個標準SQL特性：

Feature P001 "Stored Modules" (特性 P001 “存儲模式”)

Feature P002 "Computational completeness" (特性 P002 “計算完整性”)

DB2和MySQL相似的原因是兩者都支援標準SQL中的存儲過程。因此MySQL和DB2的區別就像我們背離ANSI/ISO標準語法那樣。但是我們比Oracle或SQL Server更標準。

Style 編程風格

```
CREATE PROCEDURE p ()
BEGIN
  /* Use comments! */
  UPDATE t SET s1 = 5;
  CREATE TRIGGER t2_ai ...
END;
```

在這個例子中，我們使用了一種編程風格來寫這個存儲過程。關鍵字要大寫。在命名約定中，我認為在手冊中表名最好為“t”什麼的，列名為“s”什麼的。實際上作為DBA，我們可以參考文章“SQL Naming Conventions”（來自<http://dbazine.com/gulutzan5.shtml>）。

注釋和C語言中的一樣：在BEGIN後縮進

（一般是一個TAB字元）。

在END前回縮（使END語句與BEGIN前的語句齊平）事實上我不喜歡這個細節。我更喜歡下面這樣：

```
CREATE PROCEDURE p ()
BEGIN
  /* Use comments! */
  UPDATE t SET s1 = 5;
  CREATE TRIGGER t2_ai ...
END;                                <--
```

就是當在END後回縮，而不是之前。在這裏我並不是要大家都跟隨我的風格，而是希望大家都有自己的風格，然後能在編寫存儲過程中堅持下去。這會使你的代碼更好閱讀和維護。

下面是一些函數和過程的例子，大家可以作為參考學習：

Stored Procedure Example: tables_concat() 字元串連接的函數

這是把所有表名連接到一個單一字串的函數，可以和MySQL內建的group_concat()函數對比一下。

```
CREATE PROCEDURE tables_concat
(OUT parameter1 VARCHAR(1000))
BEGIN
  DECLARE variable2 CHAR(100);
  DECLARE c CURSOR FOR
  SELECT table_name FROM information_schema.tables;
  DECLARE EXIT HANDLER FOR NOT FOUND BEGIN END; /* 1 */
  SET sql_mode='ansi'; /* 2 */
  SET parameter1 = "";
  OPEN c;
  LOOP
    FETCH c INTO variable2; /* 3 */
    SET parameter1 = parameter1 || variable2 || ',';
  END LOOP;
  CLOSE c;
END;
```

/* 1 */: 這裏的“BEGIN END”語句沒有任何作用，就像其他DBMS中的NULL語句。

/* 2 */: 將sql_mode設置為'ansi'以便“||”能正常連接，在退出存儲過程後sql_mode仍為'ansi'。

/* 3 */: 另一種跳出迴圈LOOP的方法：聲明EXIT出錯處理，當FETCH沒有返回行時。

以下就是我們調用這個過程的結果：

```
mysql> CALL tables_concat(@x);
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT @x;
+-----+
| SCHEMATA.TABLES.COLUMNS.CHARACTER_SETS.COLLATIONS.C
| OLLATION_CHARACTER_SET_APPLICABILITY.ROUTINES.STATIST
| ICS.VIEWS.USER_PRIVILEGES.SCHEMA_PRIVILEGES.TABLE_PRI
| VILEGES.COLUMN_PRIVILEGES.TABLE_CONSTRAINTS.KEY_COLUM
| N_USAGE.TABLE_NAMES.columns_priv.db.fn.func.help_cate
| gory.help_keyword.help_relation.help_topic.host.proc.
| tables_priv.time_zone.time_zone_leap_second.time_zone
+-----+
1 row in set (0.00 sec)
```

下面過程的目的是獲得整型包含行的數量的結果集，類似其他DBMS中的ROWNUM()。我們需要一個用戶變數來保存在每次調用rno()後的結果，就命名為@rno吧。

```
CREATE FUNCTION rno ()
RETURNS INT
BEGIN
    SET @rno = @rno + 1;
    RETURN @rno;
END;
```

通過rno()方法的SELECT我們獲得了行數。下面是調用程式的結果：

```
mysql> SET @rno = 0;//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT rno(),s1,s2 FROM t;//
+-----+-----+-----+
| rno() | s1  | s2  |
+-----+-----+-----+
| 1    | 1  | a   |
| 2    | 2  | b   |
| 3    | 3  | c   |
| 4    | 4  | d   |
| 5    | 5  | e   |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

在SELECT中將@rno置零的技巧是使用了WHERE的求值功能，而這個特性在今後的MySQL中可能丟失。

```
CREATE FUNCTION rno_reset ()
RETURNS INTEGER
BEGIN
    SET @rno = 0;
    RETURN 1;
END;
SELECT rno(),s1,s2 FROM t WHERE rno_reset()=1;//
```

Function Example: running_total()

這個累加的函數建立在rno()基礎上。不同在於我們要在每次調用時中傳值到參數中。

```
CREATE FUNCTION running_total (IN adder INT)
RETURNS INT
BEGIN
SET @running_total = @running_total + adder;
RETURN @running_total;
END;
```

下面是調用函數後的結果：

```
mysql> SET @running_total = 0;//
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT s1,running_total(s1),s2 FROM t ORDER BY s1;//
+-----+-----+-----+
| s1 | running_total(s1) | s2 |
+-----+-----+-----+
| 1 | 1 | a |
| 2 | 3 | b |
| 3 | 6 | c |
| 4 | 10 | d |
| 5 | 15 | e |
+-----+-----+-----+
5 rows in set (0.01 sec)
```

running_total()函數在ORDER BY完成後調用，但這樣寫既不標準也不能被移植。

Procedure Example: MyISAM "Foreign Key" insertion (MyISAM外鍵插入)

MyISAM存儲引擎不支援外鍵，但是你可以將這個邏輯加入存儲過程引擎進行檢查。如下例：

```
CREATE PROCEDURE fk_insert (p_fk INT, p_animal VARCHAR(10))
BEGIN
DECLARE v INT;
BEGIN
DECLARE EXIT HANDLER FOR SQLEXCEPTION, NOT FOUND
SET v = 0;
IF p_fk IS NOT NULL THEN
SELECT 1 INTO v FROM tpk WHERE cpk = p_fk LIMIT 1;
INSERT INTO tfk VALUES (p_fk, p_animal);
ELSE
SET v = 1;
END IF; END;
IF v <> 1 THEN
DROP TABLE `The insertion failed`;
END IF;
END;
```

注意：SQLEXCEPTION或NOT FOUND條件都會導致v變成0，如果這些條件為假，則v會變成1，因為SELECT會給v賦值1，而EXIT HANDLER沒有運行。下面看看運行結果：

```
mysql> CREATE TABLE tpk (cpk INT PRIMARY KEY);//
Query OK, 0 rows affected (0.01 sec)
mysql> CREATE TABLE tfk (cfk INT, animal VARCHAR(10));//
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tpk VALUES (1),(7),(10);//
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CALL fk_insert(1,'wombat');//
Query OK, 1 row affected (0.02 sec)

mysql> CALL fk_insert(NULL,'wallaby');//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL fk_insert(17,'wendigo');//
ERROR 1051 (42S02): Unknown table 'The insertion failed'
```

Procedure Example: Error Propagation 錯誤傳遞

設想如果過程1調用過程2，過程2調用了過程3，過程3中的錯誤就會傳遞到過程1。如果沒有異常處理器捕獲異常，那異常就會傳遞，導致過程2出錯，最後導致過程1出錯，最終異常傳遞到了調用者（MySQL用戶端實例）這種特性使得標準SQL中存在SIGNAL語句來使異常強制發生，其他DBMS中也有類似措施（RAISEERROR）。MySQL還不支援SIGNAL，直到支援此特性的新版本推出，大家就可以用SIGNAL來進行處理了。大家也可以用下面的異常處理方式。

```
CREATE PROCEDURE procedure1 ()
BEGIN
  CALL procedure2();
  SET @x = 1;
END;
CREATE PROCEDURE procedure2 ()
BEGIN
  CALL procedure3();
  SET @x = 2;
END;
CREATE PROCEDURE procedure3 ()
BEGIN
  DROP TABLE error.`error #7815`;
  SET @x = 3;
END;
```

調用過程1後結果如下：

```
mysql> CALL procedure1();//
ERROR 1051 (42S02): Unknown table 'error #7815'
```

@x並沒有改變，因為沒有一條"SET @x = ..."語句成功運行，而使用DROP可以產生一些可供診斷的錯誤資訊。不過，在這裏我們最好祈禱沒有名字叫做`error`的表存在。

Procedure Example: Library 庫

對庫的應用有詳細的規格說明。我們希望擁有許可權的用戶都能調用過程，所以我們這樣設置：

```
GRANT ALL ON database-name.* TO user-name;
```

如果要其他用戶只有訪問使用過程的許可權，沒有問題，只需要定義SQL SECURITY DEFINER特性就可以了，而這個選項是默認的，但最好顯式的聲明出來。

下面是一個向資料庫中添加書本的過程，這裏必須測試書的id是否確定，書名是否為空。例子是對MySQL不支援的CHECK限制功能的替代。

```
CREATE PROCEDURE add_book
(p_book_id INT, p_book_title VARCHAR(100))
SQL SECURITY DEFINER
BEGIN
  IF p_book_id < 0 OR p_book_title="" THEN
    SELECT 'Warning: Bad parameters';
  END IF;
  INSERT INTO books VALUES (p_book_id, p_book_title);
END;
```

我們需要一個添加買主的過程，過程必須檢查是否有超過一個的買主，如有則給出警告。這個功能可以在一個子查詢中完成，如下：

```
IF (SELECT COUNT(*) FROM table-name) > 2) THEN ... END IF;
```

不過，在寫作此書時子查詢功能有漏洞，於是我們用"SELECT COUNT(*) INTO variable-name"代替。

```
CREATE PROCEDURE add_patron
(p_patron_id INT, p_patron_name VARCHAR(100))
SQL SECURITY DEFINER
BEGIN
  DECLARE v INT DEFAULT 0;
  SELECT COUNT(*) FROM patrons INTO v;
  IF v > 2 THEN
    SELECT 'warning: already there are ',v,'patrons!';
  END IF;
  INSERT INTO patrons VALUES (p_patron_id,p_patron_name);
END;
```

下面我們需要書本付帳的過程。在事務處理過程中我們希望顯示已經擁有本書的買主，以及這些買主擁有的書，這些資訊可以通過對游標CURSOR的Fetch來獲得，我們會使用兩種不同的方法來測試是否fetch資料已經完畢，第一種是檢查變數在fetch動作後是否是NULL，第二種是通過NOT FOUND錯誤處理捕獲fetch的失敗動作。如果兩個游標在不同的BEGIN/END塊中程式看起來會顯得整潔，但我們要在主BEGIN/END塊中聲明這些變數，這樣做才能使它們的作用域覆蓋整個過程。

```
CREATE PROCEDURE checkout (p_patron_id INT, p_book_id INT)
SQL SECURITY DEFINER
BEGIN
  DECLARE v_patron_id, v_book_id INT;
  DECLARE no_more BOOLEAN default FALSE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more=TRUE;
  BEGIN
    DECLARE c1 CURSOR FOR SELECT patron_id
    FROM transactions WHERE book_id = p_book_id;
```

```
OPEN c1;
SET v_patron_id=NULL;
FETCH c1 INTO v_patron_id;
IF v_patron_id IS NOT NULL THEN
  SELECT 'Book is already out to this patron:',
  v_patron_id;
END IF;
CLOSE c1;
END; BEGIN
DECLARE c2 CURSOR FOR SELECT book_id
FROM transactions WHERE patron_id = p_patron_id;
OPEN c2;
book_loop: LOOP
  FETCH c2 INTO v_book_id;
  IF no_more THEN
    LEAVE book_loop;
  END IF;
  SELECT 'Patron already has this book:', v_book_id;
END LOOP;
END;
INSERT INTO transactions VALUES (p_patron_id, p_book_id);
END;
```

Procedure Example: Hierarchy (I) 分層次

hierarchy()過程實現的是其他DBMS中CONNECT BY部分功能。我們擁有一個Persons表，表中的後代通過person_id列與祖先相連。我們通過參數start_with傳遞列表的第一個人。然後按順序顯示祖先和後代。這個過程（實際是兩個）代碼在後面兩頁，相信大家學了那麼多，閱讀得仔細點也可以自己跟上的，不過開始我還是會給一些說明性的注釋。hierarchy()過程接受person的名字作為輸入參數，作用有點像初始化動作，不過重要的是它建立了個臨時表，用來存儲查找到的結果行的資料。然後它調用了hierarchy2()過程，此時hierarchy2()過程開始進行迴圈，不停的調用自身。如果父親只有1或0個兒子，這一步就不需要了，但事實上不會如此，所以要查詢到每個分支，因此要對樹進行不斷的迴圈查詢，直到最後節點才返回分支點進行另一個分支的查詢。我們這裏還在每條SQL語句執行後使用了錯誤處理set a flag (error)，如果語句失敗，程式使用"SELECT 'string'"輸出診斷資訊，然後離開出錯的過程。

同時在這裏還使用了複合語句的嵌套（就是在BEGIN END塊中放置BEGIN END塊），所以我們才可以為關聯特定語句的變數和出錯處理進行聲明。不過記住在外層複合語句中的聲明在內層語句中仍有效，除非你使用了重載，還有就是在內層複合語句完畢後，內層的聲明就失效。

下一頁是hierarchy()過程的創建代碼，後頁則是hierarchy2()過程的，在此之後的是成功調用hierarchy()過程的結果。

```
CREATE PROCEDURE hierarchy (start_with CHAR(10))
proc:
BEGIN
  DECLARE temporary_table_exists BOOLEAN;
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;
    DROP TABLE IF EXISTS Temporary_Table;
  END;
  BEGIN
    DECLARE v_person_id, v_father_id INT;
    DECLARE v_person_name CHAR(20);
    DECLARE done, error BOOLEAN DEFAULT FALSE;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SET error = TRUE;
    CREATE TEMPORARY TABLE Temporary_Table
    (person_id INT, person_name CHAR(20), father_id INT);
    IF error THEN
      SELECT 'CREATE TEMPORARY failed'; LEAVE proc; END IF;
    SET temporary_table_exists=TRUE;
    SELECT person_id, person_name
    INTO v_person_id, v_person_name FROM Persons
    WHERE person_name = start_with limit 1;
    IF error THEN
      SELECT 'First SELECT failed'; LEAVE proc; END IF;
    IF v_person_id IS NOT NULL THEN
      INSERT INTO Temporary_Table VALUES
      (v_person_id, v_person_name, v_father_id);
      IF error THEN
        SELECT 'First INSERT failed'; LEAVE proc; END IF;
      CALL hierarchy2(v_person_id);
      IF error THEN
        SELECT 'First CALL hierarchy2() failed'; END IF;
      END IF;
      SELECT person_id, person_name, father_id
      FROM Temporary_Table;
      IF error THEN
        SELECT 'Temporary SELECT failed'; LEAVE proc; END IF;
    END;
    IF temporary_table_exists THEN
      DROP TEMPORARY TABLE Temporary_Table;
    END IF;
  END;
```

```
CREATE PROCEDURE hierarchy2 (start_with INT)
proc:
BEGIN
  DECLARE v_person_id INT, v_father_id INT;
  DECLARE v_person_name CHAR(20);
  DECLARE done, error BOOLEAN DEFAULT FALSE;
  DECLARE c CURSOR FOR
  SELECT person_id, person_name, father_id
  FROM Persons WHERE father_id = start_with;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  SET error = TRUE;
  OPEN c;
  IF error THEN
  SELECT 'OPEN failed'; LEAVE proc; END IF;
  REPEAT
  SET v_person_id=NULL;
  FETCH c INTO v_person_id, v_person_name, v_father_id;
  IF error THEN
  SELECT 'FETCH failed'; LEAVE proc; END IF;
  IF done=FALSE THEN
  INSERT INTO Temporary_Table VALUES
  (v_person_id, v_person_name, v_father_id);
  IF error THEN
  SELECT 'INSERT in hierarchy2() failed'; END IF;
  CALL hierarchy2(v_person_id);
  IF error THEN
  SELECT 'Recursive CALL hierarchy2() failed'; END IF;
  END IF;
  UNTIL done = TRUE
  END REPEAT;
  CLOSE c;
  IF error THEN
  SELECT 'CLOSE failed'; END IF;
END;
```

下面是調用hierarchy()後的結果：

```
mysql> CREATE TABLE Persons (person_id INT, person_name CHAR(20), father_id INT);//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO Persons VALUES (1,'Grandpa',NULL);//
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Persons VALUES (2,'Pa-1',1),(3,'Pa-2',1);//
Query OK, 2 rows affected (0.00 sec)
Records: 2   Duplicates: 0   Warnings: 0
```

```
mysql> INSERT INTO Persons VALUES (4,'Grandson-1',2),(5,'Grandson-2',2);//
Query OK, 2 rows affected (0.00 sec)
Records: 2   Duplicates: 0   Warnings: 0
```

```
mysql> call hierarchy('Grandpa')//
+-----+-----+-----+
| person_id | person_name | father_id |
+-----+-----+-----+
|         1 | Grandpa     |        NULL |
|         2 | Pa-1        |          1 |
|         4 | Grandson-1  |          2 |
|         5 | Grandson-2  |          2 |
|         3 | Pa-2        |          1 |
+-----+-----+-----+
5 rows in set (0.01 sec)
```

```
Query OK, 0 rows affected (0.01 sec)
```

查詢從祖先一直查到孫子，然後又從兒子節點開始查詢，使用的是“深度優先”演算法。也許hierarchy()過程過於複雜，會有些我沒考慮到的漏洞。迴圈太深的話也許會出錯，試想如果有100代，則當有人是自己的爺爺時（這裏是直譯的，不過大家也能想到的，就是其中樹的節點互相纏繞，形成一個環），就會進入無限迴圈。不過我在這裏只是為了說明MySQL也可通過對存儲過程的迴圈實現CONNECT BY功能。

Tips when writing long routines 書寫長常式時的技巧

MySQL沒有存儲過程調試器，因此輸出的錯誤資訊大多可能沒用。下面是在創建一個20行的過程時進行異常處理的方法：

打開文本編輯器，將語句拷貝到編輯器視窗，使用copy和paste功能在文本編輯器和MySQL視窗間切換很方便。

如果是語法錯誤，可以每次去掉一句進行排錯。這比你光用眼睛看程式碼效率高多了，因為看的時候似乎一切都沒錯。

如果是運行時錯誤，則在每條可執行的語句後加入"SELECT n;"（n可以是0、1、2來標示運行到了哪條語句），調用後你就可以追中控制流來進行診斷了。

如果是資料錯誤，則可以通過添加"DECLARE CONTINUE HANDLER FOR ... BEGIN END;"來跳過，因為往往測試時資料庫的資料不存在，因此跳過這個使我們可以繼續進行。



Bugs 漏洞或缺陷

MySQL仍然是alpha版（譯者語：軟體中alpha版意思即內測，beta則是外測，翻譯時已出beta版），其存儲過程中仍有許多嚴重漏洞，我計算的2004-10-27和2004-12-14的漏洞如下：

Bug Count On October 27 2004:

Category 種類	Count
Causes Operating System To Reboot (導致系統重啓)	1
Crashes or Hangs (導致崩潰或掛起)	14
Returns 'Packets out of order' (返回'Packets out of order')	5
Fails	21
--	--
	41

Bug Count On December 14 2004:

Category	Count
Causes Operating System To Reboot	1
Crashes or Hangs	23
Returns 'Packets out of order'	6
Fails	31
--	--
	61

你可以通過訪問MySQL漏洞發佈網站來查詢存儲過程和觸發器的錯誤，下面是步驟：

1. 打開網址<http://bugs.mysql.com>
2. 點擊"Advanced Search"
3. 在"Find bugs with any of the words"框中輸入"procedure* trigger*".
4. 在下拉清單中默認為返回10個錯誤，然後選擇"All"
5. 點擊"Boolean mode"
6. 最後是點擊"Search"



Feature Requests 將會出現的特性

摘要：

SIGNAL

PATH

Accessing Tables in Functions or Triggers 在函數或觸發器中訪問表

BEGIN ATOMIC ... END

Encrypted storage 加密存儲

Timeout 超時

Debugger 調試器

External Languages 外部語言特性

DROP ... CASCADE|RESTRICT

下面是我們急切需要添加的功能：我們函數最需要的特性是擁有DB2和ANSI/ISO標準的強大功能，同時還需要超時功能使運行時間過長的迴圈或動作停止。

Resources 您可以獲得的資源

(I) MySQL的網站

(II) MySQL可供下載的資源

(III) 教程和參考手冊

在書的最後我將告訴讀者獲得更多資訊的方法和地方。

MySQL網站

MySQL參考手冊存儲過程章節：

http://www.mysql.com/doc/en/Stored_Procedures.html

“MySQL存儲過程”最新文章：

<http://www.mysql.com/newsletter/2004-01/a0000000297.html>

MySQL用戶會議的幻燈片：

mysql.mirror.anlx.net/Downloads/Presentations/MySQL-User-Conference-2003/MySQL-Stored-Procedures.pdf

這些都是你可以從網站獲取的資源。

MySQL下載

```
> cd ~/mysql-5.0/mysql-test/t
> dir sp*
10025 2004-10-23 05:19 sp-error.test
3974 2004-10-23 05:19 sp-security.test
754 2004-08-07 08:51 sp-threads.test
45344 2004-10-24 06:09 sp.test
> cd ~/mysql-5.0/Docs
> dir sp*
41909 2004-08-05 09:19 sp-imp-spec.txt
4948 2004-08-05 09:19 sp-implemented.txt
```

如果需要更多的資訊，你可以下載MySQL 5.0源代碼包。當然我不是讓讀者去讀代碼本身，而是去閱讀mysql-test目錄裏的測試腳本，在Docs目錄裏有文檔資料和新聞。（上面有操作過程）



Books 教程和參考手冊

Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM (書名, 不翻譯了)

作者Author: Jim Melton

發行Publisher: Morgan Kaufmann

Only available second-hand.

這是一本關於標準SQL存儲過程的參考書，MySQL也儘量符合SQL標準，因此作為MySQL用戶的您最好能有一本作為參考。

Conclusion結束語

到了最後我就不再去復述些什麼概念了，相信大家可以記住全部的。如果您喜歡這本書，您可以在我們的網站找到更多"MySQL 5.0 New Features"系列書籍，下一本書將會是關於“觸發器”和“視圖”。

感謝您的關注，如果您對這本書有什麼評論歡迎來MySQL論壇：<http://forums.mysql.com/>。

About MySQL (這部分就不翻譯了，估計大家也不看，呵呵)

MySQL AB develops and supports a family of high performance, affordable database servers and tools. The company's flagship product is MySQL, the world's most popular open source database, with more than six million active installations. Many of the world's largest organizations, including Yahoo!, Sabre Holdings, The Associated Press, Suzuki and NASA are realizing significant cost savings by using MySQL to power high-volume Web sites, business-critical enterprise applications and packaged software.

With headquarters in Sweden and the United States "C and operations around the world "C MySQL AB supports both open source values and corporate customers' needs in a profitable, sustainable business. For more information about MySQL, please visit www.mysql.com.

譯者的話：這是第一次做這樣的嘗試，所以挑選了這本比較入門的書來翻譯，目的是對國內MySQL用戶的支持，裏面可能會有不少的錯誤，希望大家可以原諒裏面的錯誤，但是我保證，錯誤只會是文法上的，絕對不會影響大家的操作。謝謝各位支援，轉載時請保留作者資訊和聲明。這是Peter Gulutzan先生大作——《MySQL 5.0新特性》系列的第一冊，以後我會抽空翻譯此系列的其他資料，當然大家如果需要什麼資料也可以給我寫信，如果時間充足我會給您回復翻譯，希望得到各位支持和鼓勵，謝謝大家的閱讀和寶貴意見。

作者：陳朋奕

畢業院校：西安電子科技大學 愛好：Java，

JSP、Oracle PL/SQL，MySQL 郵箱：

chenpengyi_007@163.com

地址：西安電子科技大學96# 113 (可能很快就走了……)