**存储过程简介**

SQL语句需要先编译然后执行，而存储过程（Stored Procedure）是一组为了完成特定功能的SQL语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给定参数（如果该存储过程带有参数）来调用执行它。

存储过程是可编程的函数，在数据库中创建并保存，可以由SQL语句和控制结构组成。当想要在不同的应用程序或平台上执行相同的函数，或者封装特定功能时，存储过程是非常有用的。数据库中的存储过程可以看做是对编程中面向对象方法的模拟，它允许控制数据的访问方式。

存储过程的优点：

(1).**增强SQL语言的功能和灵活性**：存储过程可以用控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算。

(2).**标准组件式编程**：存储过程被创建后，可以在程序中被多次调用，而不必重新编写该存储过程的SQL语句。而且数据库专业人员可以随时对存储过程进行修改，对应用程序源代码毫无影响。

(3).**较快的执行速度**：如果某一操作包含大量的Transaction-SQL代码或分别被多次执行，那么存储过程要比批处理的执行速度快很多。因为存储过程是预编译的。在首次运行一个存储过程时查询，优化器对其进行分析优化，并且给出最终被存储在系统表中的执行计划。而批处理的Transaction-SQL语句在每次运行时都要进行编译和优化，速度相对要慢一些。

(4).减少网络流量：针对同一个数据库对象的操作（如查询、修改），如果这一操作所涉及的Transaction-SQL语句被组织进存储过程，那么当在客户计算机上调用该存储过程时，网络中传送的只是该调用语句，从而大大减少网络流量并降低了网络负载。

(5).作为一种安全机制来充分利用：通过对执行某一存储过程的权限进行限制，能够实现对相应的数据的访问权限的限制，避免了非授权用户对数据的访问，保证了数据的安全。

**MySQL的存储过程**

存储过程是数据库的一个重要的功能，MySQL 5.0以前并不支持存储过程，这使得MySQL在应用上大打折扣。好在MySQL 5.0开始支持存储过程，这样即可以大大提高数据库的处理速度，同时也可以提高数据库编程的灵活性。

**MySQL存储过程的创建**

**语法**

**CREATE PROCEDURE**  **过程名([[IN|OUT|INOUT] 参数名 数据类型[,[IN|OUT|INOUT] 参数名 数据类型…]]) [特性 ...] 过程体**

**DELIMITER //  
 CREATE PROCEDURE myproc(OUT s int)  
 BEGIN  
 SELECT COUNT(\*) INTO s FROM students;  
 END  
 //  
DELIMITER ;**

**分隔符**

MySQL默认以";"为分隔符，如果没有声明分割符，则编译器会把存储过程当成SQL语句进行处理，因此编译过程会报错，所以要事先用“DELIMITER //”声明当前段分隔符，让编译器把两个"//"之间的内容当做存储过程的代码，不会执行这些代码；“DELIMITER ;”的意为把分隔符还原。

**参数**

存储过程根据需要可能会有输入、输出、输入输出参数，如果有多个参数用","分割开。MySQL存储过程的参数用在存储过程的定义，共有三种参数类型,IN,OUT,INOUT:

 **IN**参数的值必须在调用存储过程时指定，在存储过程中修改该参数的值不能被返回，为默认值

 **OUT:**该值可在存储过程内部被改变，并可返回

 **INOUT:**调用时指定，并且可被改变和返回

**过程体**

过程体的开始与结束使用BEGIN与END进行标识。

**IN参数例子**

**DELIMITER //  
 CREATE PROCEDURE in\_param(IN p\_in int)  
 BEGIN  
 SELECT p\_in;  
 SET p\_in=2;  
 SELECT p\_in;  
 END;  
 //  
DELIMITER ;**

**#调用  
SET @p\_in=1;  
CALL in\_param(@p\_in);**

**SELECT @p\_in;**

**执行结果：**

以上可以看出，p\_in虽然在存储过程中被修改，但并不影响@p\_id的值

**OUT参数例子**

**#存储过程OUT参数  
DELIMITER //  
 CREATE PROCEDURE out\_param(OUT p\_out int)  
 BEGIN  
 SELECT p\_out;  
 SET p\_out=2;  
 SELECT p\_out;  
 END;  
 //  
DELIMITER ;  
#调用  
SET @p\_out=1;  
CALL out\_param(@p\_out);**

**SELECT @p\_out;**

**执行结果：**

**INOUT参数例子**

**#存储过程INOUT参数  
DELIMITER //  
 CREATE PROCEDURE inout\_param(INOUT p\_inout int)  
 BEGIN  
 SELECT p\_inout;  
 SET p\_inout=2;  
 SELECT p\_inout;  
 END;  
 //  
DELIMITER ;  
#调用  
SET @p\_inout=1;  
CALL inout\_param(@p\_inout) ;  
SELECT @p\_inout;**

**执行结果：**

**变量**

**语法：**DECLARE 变量名1[,变量名2...] 数据类型 [默认值];

数据类型为MySQL的数据类型：

数值类型

日期和时间类型

字符串类型

**变量赋值**

语法：SET 变量名 = 变量值 [,变量名= 变量值 ...]

**用户变量**

用户变量一般以@开头

注意：滥用用户变量会导致程序难以理解及管理

**#在MySQL客户端使用用户变量  
SELECT 'Hello World' into @x;  
SELECT @x;**

**SET @y='Goodbye Cruel World';  
SELECT @y;  
SET @z=1+2+3;  
SELECT @z;**

**执行结果：**

**#在存储过程中使用用户变量  
CREATE PROCEDURE GreetWorld() SELECT CONCAT(@greeting,' World');  
SET @greeting='Hello';  
CALL GreetWorld();**

**执行结果：**

**#在存储过程间传递全局范围的用户变量**

**CREATE PROCEDURE p1() SET @last\_proc='p1';  
CREATE PROCEDURE p2() SELECT CONCAT('Last procedure was ',@last\_proc);  
CALL p1();  
CALL p2();**

**执行结果：**

**注释**

MySQL存储过程可使用两种风格的注释：

 双杠：--，该风格一般用于单行注释

 C风格： 一般用于多行注释

**MySQL存储过程的调用**

用call和你过程名以及一个括号，括号里面根据需要，加入参数，参数包括输入参数、输出参数、输入输出参数。

**MySQL存储过程的查询**

**#查询存储过程  
SELECT name FROM mysql.proc WHERE db='数据库名';  
SELECT routine\_name FROM information\_schema.routines WHERE routine\_schema='数据库名';  
SHOW PROCEDURE STATUS WHERE db='数据库名';**

**#查看存储过程详细信息  
SHOW CREATE PROCEDURE 数据库.存储过程名;**

**MySQL存储过程的修改**

ALTER PROCEDURE 更改用CREATE PROCEDURE 建立的预先指定的存储过程，其不会影响相关存储过程或存储功能。

**ALTER {PROCEDURE | FUNCTION} sp\_name [characteristic ...]  
characteristic:  
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  
| SQL SECURITY { DEFINER | INVOKER }  
| COMMENT 'string'**

 sp\_name参数表示存储过程或函数的名称；

 characteristic参数指定存储函数的特性。

 CONTAINS SQL表示子程序包含SQL语句，但不包含读或写数据的语句；

 NO SQL表示子程序中不包含SQL语句；

 READS SQL DATA表示子程序中包含读数据的语句；

 MODIFIES SQL DATA表示子程序中包含写数据的语句。

 SQL SECURITY { DEFINER | INVOKER }指明谁有权限来执行，DEFINER表示只有定义者自己才能够执行；INVOKER表示调用者可以执行。

 COMMENT 'string'是注释信息。

实例：

**#将读写权限改为MODIFIES SQL DATA，并指明调用者可以执行。  
ALTER PROCEDURE num\_from\_employee  
 MODIFIES SQL DATA  
 SQL SECURITY INVOKER ;  
#将读写权限改为READS SQL DATA，并加上注释信息'FIND NAME'。  
ALTER PROCEDURE name\_from\_employee  
 READS SQL DATA  
 COMMENT 'FIND NAME' ;**

**MySQL存储过程的删除**

DROP PROCEDURE [过程1[,过程2…]]

从MySQL的表格中删除一个或多个存储过程。

**MySQL存储过程的控制语句**

**变量作用域**

内部变量在其作用域范围内享有更高的优先权，当执行到end时，内部变量消失，不再可见了，在存储  
过程外再也找不到这个内部变量，但是可以通过out参数或者将其值指派给会话变量来保存其值。

**#变量作用域  
DELIMITER //  
 CREATE PROCEDURE proc()  
 BEGIN  
 DECLARE x1 VARCHAR(5) DEFAULT 'outer';  
 BEGIN  
 DECLARE x1 VARCHAR(5) DEFAULT 'inner';  
 SELECT x1;  
 END;  
 SELECT x1;  
 END;  
 //  
DELIMITER ;  
#调用  
CALL proc();**

**执行结果：**

**条件语句**

IF-THEN-ELSE**语句**

**#条件语句IF-THEN-ELSE  
DROP PROCEDURE IF EXISTS proc3;  
DELIMITER //  
CREATE PROCEDURE proc3(IN parameter int)  
 BEGIN  
 DECLARE var int;  
 SET var=parameter+1;  
 IF var=0 THEN  
 INSERT INTO t VALUES (17);  
 END IF ;  
 IF parameter=0 THEN  
 UPDATE t SET s1=s1+1;  
 ELSE  
 UPDATE t SET s1=s1+2;  
 END IF ;  
 END ;  
 //  
DELIMITER ;**

**CASE-WHEN-THEN-ELSE语句**

**#CASE-WHEN-THEN-ELSE语句  
DELIMITER //  
 CREATE PROCEDURE proc4 (IN parameter INT)  
 BEGIN  
 DECLARE var INT;  
 SET var=parameter+1;  
 CASE var  
 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 ;  
 //  
DELIMITER ;**

**循环语句**

**WHILE-DO…END-WHILE  
DELIMITER //  
 CREATE PROCEDURE proc5()  
 BEGIN  
 DECLARE var INT;  
 SET var=0;  
 WHILE var<6 DO  
 INSERT INTO t VALUES (var);  
 SET var=var+1;  
 END WHILE ;  
 END;  
 //  
DELIMITER ;**

**REPEAT...END REPEAT**

此语句的特点是执行操作后检查结果  
**DELIMITER //  
 CREATE PROCEDURE proc6 ()  
 BEGIN  
 DECLARE v INT;  
 SET v=0;  
 REPEAT  
 INSERT INTO t VALUES(v);  
 SET v=v+1;  
 UNTIL v>=5  
 END REPEAT;  
 END;  
 //  
DELIMITER ;**

**LOOP...END LOOP**  
**DELIMITER //  
 CREATE PROCEDURE proc7 ()  
 BEGIN  
 DECLARE v INT;  
 SET v=0;  
 LOOP\_LABLE:LOOP  
 INSERT INTO t VALUES(v);  
 SET v=v+1;  
 IF v >=5 THEN  
 LEAVE LOOP\_LABLE;  
 END IF;  
 END LOOP;  
 END;  
 //  
DELIMITER ;**

**LABLES标号**

标号可以用在begin repeat while 或者loop 语句前，语句标号只能在合法的语句前面使用。可以跳出循环，使运行指令达到复合语句的最后一步。

**ITERATE迭代**

通过引用复合语句的标号,来从新开始复合语句

**#ITERATE  
DELIMITER //  
 CREATE PROCEDURE proc8()  
 BEGIN  
 DECLARE v INT;  
 SET v=0;  
 LOOP\_LABLE:LOOP  
 IF v=3 THEN  
 SET v=v+1;  
 ITERATE LOOP\_LABLE;  
 END IF;  
 INSERT INTO t VALUES(v);  
 SET v=v+1;  
 IF v>=5 THEN  
 LEAVE LOOP\_LABLE;  
 END IF;  
 END LOOP;  
 END;  
 //  
DELIMITER ;**

**MySQL存储过程的基本函数**

**字符串类**

CHARSET(str) //返回字串字符集  
CONCAT (string2 [,... ]) //连接字串  
INSTR (string ,substring ) //返回substring首次在string中出现的位置,不存在返回0  
LCASE (string2 ) //转换成小写  
LEFT (string2 ,length ) //从string2中的左边起取length个字符  
LENGTH (string ) //string长度  
LOAD\_FILE (file\_name ) //从文件读取内容  
LOCATE (substring , string [,start\_position ] ) 同INSTR,但可指定开始位置  
LPAD (string2 ,length ,pad ) //重复用pad加在string开头,直到字串长度为length  
LTRIM (string2 ) //去除前端空格  
REPEAT (string2 ,count ) //重复count次  
REPLACE (str ,search\_str ,replace\_str ) //在str中用replace\_str替换search\_str  
RPAD (string2 ,length ,pad) //在str后用pad补充,直到长度为length  
RTRIM (string2 ) //去除后端空格  
STRCMP (string1 ,string2 ) //逐字符比较两字串大小,  
SUBSTRING (str , position [,length ]) //从str的position开始,取length个字符,  
注：mysql中处理字符串时，默认第一个字符下标为1，即参数position必须大于等于1

**SELECT SUBSTRING('abcd',0,2);**

结果：

**SELECT SUBSTRING('abcd',1,2);**

结果：

TRIM([[BOTH|LEADING|TRAILING] [padding] FROM]string2) //去除指定位置的指定字符  
UCASE (string2 ) //转换成大写  
RIGHT(string2,length) //取string2最后length个字符  
SPACE(count) //生成count个空格

**数学类**

ABS (number2 ) //绝对值  
BIN (decimal\_number ) //十进制转二进制  
CEILING (number2 ) //向上取整  
CONV(number2,from\_base,to\_base) //进制转换  
FLOOR (number2 ) //向下取整  
FORMAT (number,decimal\_places ) //保留小数位数  
HEX (DecimalNumber ) //转十六进制  
注：HEX()中可传入字符串，则返回其ASC-11码，如HEX('DEF')返回4142143  
也可以传入十进制整数，返回其十六进制编码，如HEX(25)返回19  
LEAST (number , number2 [,..]) //求最小值  
MOD (numerator ,denominator ) //求余  
POWER (number ,power ) //求指数  
RAND([seed]) //随机数  
ROUND (number [,decimals ]) //四舍五入,decimals为小数位数] 注：返回类型并非均为整数，如：

**#默认变为整型值**

**SELECT ROUND(1.23);**

**SELECT ROUND(1.56);**

**#设定小数位数，返回浮点型数据  
SELECT ROUND(1.567,2);**

SIGN (number2 ) // 正数返回1，负数返回-1

**日期时间类**

ADDTIME (date2 ,time\_interval ) //将time\_interval加到date2  
CONVERT\_TZ (datetime2 ,fromTZ ,toTZ ) //转换时区  
CURRENT\_DATE ( ) //当前日期  
CURRENT\_TIME ( ) //当前时间  
CURRENT\_TIMESTAMP ( ) //当前时间戳  
DATE (datetime ) //返回datetime的日期部分  
DATE\_ADD (date2 , INTERVAL d\_value d\_type ) //在date2中加上日期或时间  
DATE\_FORMAT (datetime ,FormatCodes ) //使用formatcodes格式显示datetime  
DATE\_SUB (date2 , INTERVAL d\_value d\_type ) //在date2上减去一个时间  
DATEDIFF (date1 ,date2 ) //两个日期差  
DAY (date ) //返回日期的天  
DAYNAME (date ) //英文星期  
DAYOFWEEK (date ) //星期(1-7) ,1为星期天  
DAYOFYEAR (date ) //一年中的第几天  
EXTRACT (interval\_name FROM date ) //从date中提取日期的指定部分  
MAKEDATE (year ,day ) //给出年及年中的第几天,生成日期串  
MAKETIME (hour ,minute ,second ) //生成时间串  
MONTHNAME (date ) //英文月份名  
NOW ( ) //当前时间  
SEC\_TO\_TIME (seconds ) //秒数转成时间  
STR\_TO\_DATE (string ,format ) //字串转成时间,以format格式显示  
TIMEDIFF (datetime1 ,datetime2 ) //两个时间差  
TIME\_TO\_SEC (time ) //时间转秒数]  
WEEK (date\_time [,start\_of\_week ]) //第几周  
YEAR (datetime ) //年份  
DAYOFMONTH(datetime) //月的第几天  
HOUR(datetime) //小时  
LAST\_DAY(date) //date的月的最后日期  
MICROSECOND(datetime) //微秒  
MONTH(datetime) //月  
MINUTE(datetime) //分返回符号,正负或0  
SQRT(number2) //开平方