第 3 章 数据库编程

标准 SQL 是非过程化的查询语言，具有操作统一、面向集合、功能丰富、使用简单等多项优点。和程序设计语言相比，高度非过程化的优点同时也造成了 SQL 语言的一个弱点——缺少流程控制能力，难以实现应用业务中的逻辑控制。SQL 编程技术可以有效克服 SQL 语言在实现复杂应用方面的不足，提高应用系统和 RDBMS 间的互操作性。

3.1 MySQL 编程基础

为了提高代码的重用性及可维护性，经常需要将频繁使用的业务逻辑封装成存储程序。和其他数据库管理系统一样，MySQL 也提供了用于编写结构化程序的数据类型、常量、变量、运算符和表达式等，掌握这些内容是 MySQL 程序设计的基础。

3.1.1 常量与变量

在程序运行过程中，程序本身不能改变其值的数据称为常量。相应地，在程序运行过程中可以改变其值的数据称为变量。

1. 常量

在 SQL 程序设计过程中，常量的格式取决于其数据类型，常用的常量包括字符串常量、数值常量、日期和时间常量、布尔值常量和 NULL 值。

1) 字符串常量

字符串常量指用单引号或双引号括起来的字符序列。在 MySQL 中推荐使用单引号。

【例 3-1】 查询表 emp 中 ename 值为 SCOTT 的雇员的信息。

```
SELECT * FROM emp
 WHERE ename = 'SCOTT';
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

请读者考虑，为什么下面的 SQL 命令没有查到结果记录？

```
SELECT * FROM emp
 WHERE 'ename' = 'SCOTT';
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno

2）数值常量

数值常量可以分为整数常量和小数常量。

【例 3-2】 将表 emp 中 SCOTT 雇员的 comm 值改为 1250(要求用科学记数法表示)。

```
UPDATE emp SET COMM = 1.25E + 3 WHERE ename = 'SCOTT';

SELECT * FROM emp WHERE ename = 'SCOTT';
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	1250.00	20

3）日期和时间常量

日期和时间常量使用特定格式的字符日期值表示,用单引号括起来。例如'2023/01/01'、'2023-01-01 10:30:20'。

【例 3-3】 查询表 emp 中 1981 年以后雇员的 ename 和 hiredate 信息。

```
SELECT ename,hiredate FROM emp
 WHERE hiredate>'1981/12/31';
```

ename	hiredate
SCOTT	1987-04-19
ADAMS	1987-05-23
MILLER	1982-01-23

4）布尔值常量

布尔值常量只有 true 和 false 两个值,SQL 命令的运行结果用 1 代表 true,用 0 代表 false。

【例 3-4】 查询表 emp 中所有雇员的 ename 和 sal 是否大于或等于 2000 的判断结果。

```
SELECT ename,sal>2000 FROM emp;
```

ename	sal>2000
SMITH	0
ALLEN	0
WARD	0
JONES	1
MARTIN	0
BLAKE	1

5）NULL 值

NULL 值适用于各种字段类型,通常表示“不确定的值”。NULL 值参与的运算,结果仍为 NULL 值。

【例 3-5】 将表 emp 中雇员 SCOTT 的 comm 列值改为 NULL 值,然后在 NULL 值的基础上加 1250 元,请考虑最终 comm 列值是什么?

```
UPDATE emp SET comm = NULL WHERE ename = 'SCOTT';

UPDATE emp SET comm = comm + 1250 WHERE ename = 'SCOTT';

SELECT * FROM emp WHERE ename = 'SCOTT';
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

2. 变量

变量用于临时存放数据,变量中的数据随程序的运行而变化,变量有名字和数据类型两个属性。在 MySQL 系统中存在两种变量,一种是系统定义和维护的全局变量,通常在名称前面加@@符号;另一种是用户定义的用来存放中间结果的局部变量,通常在名称前面加@符号。

1) 局部变量

局部变量的作用范围限制在程序内部,它可以作为计数器来计算循环次数,或控制循环执行的次数;另外,利用局部变量还可以保存数据值,供控制流语句测试及保存由存储过程返回的数据值等。

(1) 局部变量的定义与赋值:使用 SET 语句定义局部变量,并为其赋值。SET 语句的语法格式如下:

```
SET @局部变量名 = 表达式 1[,@局部变量名 = 表达式 2,…];
```

注意:SET 语句可以同时定义多个变量,中间用逗号隔开。

(2) 局部变量的显示:使用 SELECT 语句显示局部变量。SELECT 语句的语法格式如下:

```
SELECT @局部变量名[,@局部变量名,…];
```

【例 3-6】 查询表 emp 中雇员 SMITH 的 sal 值赋给变量 salary,并显示其值。

```
SET @salary = (SELECT sal FROM emp WHERE ename = 'SMITH');
SELECT @salary;
```

@salary
800.00

【例 3-7】 查询表 emp 中雇员 SMITH 的 job 和 hiredate 值赋给变量 job_v、hiredate_v,并显示两个变量的值。

```
SELECT job,hiredate INTO @job_v,@hiredate_v
  FROM emp WHERE ename = 'SMITH';
SELECT @job_v,@hiredate_v;
```

@job_v	@hiredate_v
CLERK	1980-12-17

【例 3-8】 根据 name 变量所给的值查询指定员工的信息。

```
SET @name = 'SCOTT';
SELECT * FROM emp WHERE ename = @name;
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

2) 全局变量

全局变量是 MySQL 系统提供并赋值的变量。用户不能定义全局变量,只能使用。常用的系统全局变量及其说明如表 3-1 所示。

表 3-1　MySQL 系统全局变量及其说明

全局变量名称	说　明
@@back_log	返回 MySQL 主要连接请求的数量
@@basedir	返回 MySQL 安装基准目录
@@license	返回服务器的许可类型
@@port	返回服务器侦听 TCP/IP 连接所用的端口
@@storage_engine	返回存储引擎
@@version	返回服务器版本号

【例 3-9】　查看 MySQL 的版本信息。

```
SELECT @@version;
```

@@version
8.0.27

3.1.2　常用系统函数

函数是一组编译好的 SQL 语句，可以没有参数或有多个参数，并且定义了一系列的操作，返回一个数值、数值集合，或执行一些操作。函数能够重复执行一些操作，从而避免了用户不断重写代码。

MySQL 提供了丰富的系统函数，包括字符串函数、数学函数、日期和时间函数、系统信息函数等，方便用户对数据的查询和修改，同时用户也可以创建自定义的存储函数。

1. 字符串函数

1）计算字符串字符数的函数和字符串长度的函数

- CHAR_LENGTH(str)：返回字符串 str 所包含的字符个数。
- LENGTH(str)：返回字符串的字节长度。一个汉字是 3 字节，一个数字或字母是 1 字节。

【例 3-10】　计算字符串字符数和字符串长度的示例。

```
SELECT CHAR_LENGTH('CHINA'),LENGTH('CHINA');
```

CHAR_LENGTH('CHINA')	LENGTH('CHINA')
5	5

```
SELECT CHAR_LENGTH('中国') 字符数,LENGTH('中国') 字符串长度;
```

字符数	字符串长度
2	6

2）合并字符串函数

CONCAT(s1,s2,…)返回连接参数产生的字符串，如果任何一个参数为 NULL，则返回值为 NULL。

【例 3-11】　合并字符串示例。

```
SELECT CONCAT('MySQL 版本:',@@version) 版本信息;
```

版本信息
MySQL版本: 8.0.27

3) 字符串大小写转换函数

- LOWER(str)：将字符串 str 中的字母字符全部转换成小写字母。
- UPPER(str)：将字符串 str 中的字母字符全部转换成大写字母。

【例 3-12】 字符串大小写转换示例。

```
SET @name = 'sCOtt';

SELECT * FROM emp WHERE UPPER(ename) = UPPER(@name);
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

4) 删除空格函数

- LTRIM(str)：返回删除前导空格的字符串 str。
- RTRIM(str)：返回删除尾部空格的字符串 str。
- TRIM(str)：返回删除两侧空格的字符串 str。

注意：这 3 个函数只删除字符串前端和后端的空格，不删除字符串中间的空格。

【例 3-13】 删除空格示例。

```
SET @name = ' SCOtt ';

SELECT * FROM emp WHERE UPPER(ename) = TRIM(UPPER(@name));
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

注意：用户在前台“登录界面”输入用户名时可能无意地加上了前后空格，那么在后台数据库表中查找该用户时则需要将前后空格删除。

5) 取子串函数

SUBSTRING(str,start,length)返回字符串 str 从 start 开始长度为 length 的子串。

【例 3-14】 返回表 emp 中 ename 值以 S 开头的雇员的信息。

```
SELECT * FROM emp WHERE SUBSTRING(ename,1,1) = 'S';
```

EMPNO	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980-12-17	800.00	NULL	20
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	NULL	20

命令也可写成：

```
SELECT * FROM emp WHERE ename LIKE 'S%';
```

2. 数学函数

- ABS(x)：返回 x 的绝对值。
- PI()：返回圆周率 π 的值。
- SQRT()：返回非负数的二次方根。
- MOD(m,n)：返回 m 被 n 除后的余数。
- ROUND(x,y)：把 x 四舍五入到 y 指定的精度返回。如果 y 为负数，则将保留 x 值到小数点左边 y 位。

【例 3-15】 数学函数示例。

```
SELECT SQRT(ROUND(ABS(-4.01*4.01),0)),MOD(-10,3),MOD(10,-3)
```

SQRT(ROUND(ABS(-4.01*4.01),0))	MOD(-10,3)	MOD(10,-3)
4	-1	1

3. 日期和时间函数

1）获取当前系统的日期及取日期的年、月、日的函数

- CURDATE()：返回当前系统日期，格式为'YYYY-MM-DD'。
- YEAR(d)、MONTH(d)、DAY(d)：分别返回日期或日期时间 d 的年、月、日的值。

【例 3-16】 日期和时间函数示例。

```
SELECT CURDATE(),YEAR(CURDATE()),MONTH(CURDATE()),DAY(CURDATE());
```

CURDATE()	YEAR(CURDATE())	MONTH(CURDATE())	DAY(CURDATE())
2018-09-21	2018	9	21

【例 3-17】 查询表 emp 中员工 SMITH 的工作年限。

```
SELECT ename 姓名,YEAR(CURDATE()) - YEAR(hiredate) 工作年限 FROM emp
 WHERE ename = 'SMITH';
```

姓名	工作年限
SMITH	38

2）获取当前系统的日期时间的函数

CURRENT_TIMESTAMP()、LOCALTIME()、NOW()和 SYSDATE()4 个函数作用相同，均返回当前系统的日期时间，格式为'YYYY-MM-DD HH：MM：SS'。

【例 3-18】 获取当前系统的日期时间示例。

```
SELECT CURRENT_TIMESTAMP(),LOCALTIME(),NOW(),SYSDATE();
```

CURRENT_TIMESTAMP()	LOCALTIME()	NOW()	SYSDATE()
2018-09-21 10:14:04	2018-09-21 10:14:04	2018-09-21 10:14:04	2018-09-21 10:14:04

【例 3-19】 例 3-17 也可用如下方式实现。

```
SELECT ename 姓名,YEAR(SYSDATE()) - YEAR(hiredate) 工作年限 FROM emp
 WHERE ename = 'SMITH';
```

4. 系统信息函数

- USER()：返回当前登录的用户名。
- DATABASE()：返回当前所使用数据库的名。
- VERSION()：返回 MySQL 服务器版本号。

【例 3-20】 系统信息函数示例。

```
SELECT CONCAT('MySQL 版本号:',VERSION(),';用户:',USER()) AS 登录信息;
```

登录信息
MySQL版本号：8.0.27;用户：root@localhost

5. 条件控制函数

1) IF()函数

其格式为 IF(条件表达式,v1,v2),如果条件表达式是真,则函数返回 v1 值,否则返回 v2 值。

【例 3-21】 查询表 emp 中的前 5 条记录,显示 ename 和 comm 字段的值,当 comm 字段的值为 NULL 时显示值为 0,否则显示当前字段的值。

```
SELECT ename,IF(comm IS NULL,0,comm) 奖金
 FROM emp LIMIT 5; ##LIMIT 5 为显示前 5 条记录
```

ename	奖金
SMITH	0
ALLEN	300.00
WARD	500.00
JONES	0
MARTIN	1400.00

2) CASE()函数

```
CASE 表达式
 WHEN v1 THEN r1
 WHEN v2 THEN r2
 …
 [ELSE rn]
END
```

如果表达式的值等于某个 vn,则返回对应位置 THEN 后面的结果。如果与所有值都不相等,则返回 ELSE 后面的 rn。

【例 3-22】 查询 SMITH 所在部门的名称。

```
SELECT ename 姓名,
  CASE deptno
   WHEN 10 THEN 'ACCOUNTING'
   WHEN 20 THEN 'RESEARCH'
   WHEN 30 THEN 'SALES'
   WHEN 40 THEN 'OPERATIONS'
  END 部门名称
 FROM emp
 WHERE ename = 'SMITH';
```

姓名	部门名称
SMITH	RESEARCH

本例也可用如下方法实现:

```
SELECT ename 姓名,dname 部门名称 FROM dept,emp
 WHERE dept.deptno = emp.deptno AND ename = 'SMITH';
```

请读者考虑,在部门数量较少的情况下,上面两个命令哪个执行速度快?

6. 数据类型转换函数

CAST(x AS 新类型)和 CONVERT(x 新类型)两个函数作用相同,都是将一种类型的值转换为另一种类型的值。

【例 3-23】 数据类型转换示例。

```
SELECT ename,sal INTO @name,@salary FROM emp
    WHERE ename = 'SMITH';
SELECT CONCAT(@name,'的工资是',CAST(@salary AS CHAR(7))) 信息;
```

信息
SMITH的工资是800.00

3.2 程序控制流语句

与所有的程序设计语言一样，MySQL 提供了用于编写过程化代码的语法结构，可进行顺序、分支、循环、存储过程、存储函数、触发器等程序设计，编写结构化的模块代码，并放置到数据库服务器上。

3.2.1 语句块、注释和重置命令结束标记

在编写程序时，完成的功能往往用一组 SQL 语句实现，需要使用 BEGIN…END 将这些语句组合起来形成一个逻辑单元。

例如，对于存储过程中的源代码，为了方便编程人员开发或调试、帮助用户理解程序员的意图，可对一些语句加入注释进行说明。这些注释在程序编译和执行时被忽略，只起到说明的作用。

1. 语句块

BEGIN…END 用于定义 SQL 语句块，其语法格式如下：

```
BEGIN
  SQL 语句 | SQL 语句块
END
```

说明：

(1) BEGIN…END 语句块包含了该程序块的所有处理操作，允许语句块嵌套。

(2) 在 MySQL 中单独使用 BEGIN…END 语句块没有任何意义，只有将其封装到存储过程、存储函数、触发器等存储程序内部才有意义。

2. 注释

在源代码中加入注释便于用户对程序的更好理解，有两种声明注释的方法，即单行注释和多行注释。

1) 单行注释

使用“#”符号作为单行语句的注释符，写在需要注释的行或语句的后面。

【例 3-24】 单行注释示例。

```
#取两个数的最大值
SET @x = 5,@y = 6;                    #定义两个变量并赋值
SELECT IF(@x>@y,@x,@y) 最大值;
```

最大值
6

2) 多行注释

使用/ * 和 * /括起来可以连续书写多行注释语句。

【例 3-25】 多行注释示例。

```
/ * 在使用 MySQL 执行 UPDATE 的时候,如果不是用主键当 WHERE 语句,会报错,而主键用于 WHERE 语句
中则正常。因为 MySQL 运行在 SAFE-UPDATES 模式下,该模式会导致非主键条件下无法执行 UPDATE 或
者 DELETE 命令,执行命令 SET SQL_SAFE_UPDATES = 0 修改数据库模式 * /
SET SQL_SAFE_UPDATES = 0;
UPDATE dept_c SET deptno = 50 WHERE deptno = 10;
```

3. 重置命令结束标记

在 MySQL 中,服务器处理的语句是以分号为结束标记的。但在创建存储函数、存储过程时,在函数体或存储过程体中可以包含多个 SQL 语句,每个 SQL 语句都是以分号结尾,而服务器处理程序时遇到第 1 个分号则结束程序的执行,这时就需要使用 DELIMITER 语句将 MySQL 语句的结束标记修改为其他符号。

DELIMITER 语句的语法格式如下:

```
DELIMITER 符号
```

说明:

(1) 符号可以是一些特殊符号,例如两个“#”、两个“@”、两个“$”、两个“%”等。但避免使用反斜杠字符“/”,因为它是 MySQL 的转义字符。

(2) 恢复使用分号作为结束标记,执行“DELIMITER;”即可。

【例 3-26】 重置命令结束标记示例。

```
DELIMITER @@
SELECT * FROM emp@@

DELIMITER;
SELECT * FROM emp;
```

3.2.2 存储函数

用户在编写程序的过程中,不仅可以调用系统函数,也可以根据应用程序的需要创建存储函数。

1. 存储函数的创建

创建存储函数,需要使用 CREATE FUNCTION 语句,其语法格式如下:

```
CREATE FUNCTION 函数名([参数名 参数数据类型[,…]])
RETURNS 函数返回值的数据类型
BEGIN
  函数体;
  RETURN 语句;
END
```

2. 调用存储函数

对于新创建的存储函数，调用方法与调用系统函数相同，其语法格式如下：

```
SELECT 函数名([参数值[,…]]);
```

【例 3-27】 创建存储函数 name_fn()，根据所给的部门编号值 deptno，函数返回该部门的部门名称 dname。

```
SET GLOBAL log_bin_trust_function_creators = 1;        ##设置信任存储程序的创建者

DELIMITER @@
CREATE FUNCTION name_fn(dno DECIMAL(2))
RETURNS VARCHAR(14)
BEGIN
  RETURN(SELECT dname FROM dept
    WHERE deptno = dno);
END@@

DELIMITER;
SELECT name_fn(20);
```

name_fn(20)
RESEARCH

3. 删除存储函数

当不再需要某个存储函数时，可用 DROP FUNCTION 语句删除，其语法格式如下：

```
DROP FUNCTION 函数名;
```

注意：*函数名后面不要加括号。*

【例 3-28】 删除例 3-27 创建的存储函数 name_fn()。

```
DROP FUNCTION name_fn;
```

3.2.3 条件判断语句

MySQL 与其他编程语言一样，也具有条件判断语句。条件判断语句的主要作用是根据条件的变化选择执行不同的代码。常用的条件判断语句有 IF 语句和 CASE 语句。

1. 程序中变量的使用

局部变量可以在程序中声明并使用，这些变量的作用范围是 BEGIN…END 语句块。

1) 声明变量

在存储程序(例如存储函数、存储过程、触发器等)中需要使用 DECLARE 语句声明局部变量，其语法格式如下：

```
DECLARE 局部变量名[,局部变量名,…] 数据类型 [DEFAULT 默认值];
```

说明：

(1) DECLARE 声明的局部变量，变量名前不能加@。

(2) DEFAULT 子句提供了一个默认值,如果没有给默认值,局部变量的初始值默认为 NULL。

2) 为变量赋值

在声明变量后,可用 SET 命令为变量赋值,其语法格式如下:

```
SET 局部变量名 = 表达式 1[,局部变量名 = 表达式 2,…];
```

【例 3-29】 创建求任意两个数之和的存储函数 sum_fn()。

```
DELIMITER @@
CREATE FUNCTION sum_fn(a DECIMAL(5,2),b DECIMAL(5,2))
RETURNS DECIMAL
 BEGIN
  DECLARE x,y DECIMAL(5,2);                ##声明两个整型变量,注意变量名前没有@
  SET x = a,y = b;                         ##给两个整型变量赋值,注意变量名前没有@
  RETURN x + y;
 END@@

DELIMITER ;
SELECT sum_fn(7,3);
```

sum_fn(7,3)
10

2. IF 语句

在 MySQL 中为了控制程序的执行方向,引进了 IF 语句。IF 语句主要有以下两种形式。

1) 形式一

```
IF <条件> THEN
  SQL 语句块 1;
[ELSE
  SQL 语句块 2; ]
END IF;
```

ELSE 短语用方括号括起来,和其他语言一样,表示它为可选项。

【例 3-30】 创建函数 max_fn(),判断整型变量 a 和 b 的大小。

```
DELIMITER @@
CREATE FUNCTION max_fn(a int,b int)
 RETURNS INT
 BEGIN
  IF a > b THEN
    RETURN a;
  ELSE
    RETURN b;
  END IF;
 END@@

DELIMITER;
SELECT CONCAT('最大值:',CONVERT(max_fn(7,8),CHAR(3))) ;
```

CONCAT('最大值:',CONVERT(max_fn(7,8),CHAR(3)))
最大值:8

2）形式二

IF…END IF 语句一次只能判断一个条件，而 IF…ELSEIF…END IF 语句可以判定两个以上的判断条件。该语句的语法形式如下：

```
IF <条件 1> THEN
  SQL 语句块 1;
ELSEIF <条件 2> THEN
  SQL 语句块 2;
…
ELSE
  SQL 语句块 n;
END IF;
```

【例 3-31】 创建判断某一年是否为闰年的函数 leap_year()。

闰年的判断条件为年值能被 4 整除但不能被 100 整除；或者能被 400 整除。

```
DELIMITER @@
CREATE FUNCTION leap_year(year_date INT)
RETURNS VARCHAR(20)
BEGIN
DECLARE leap BOOLEAN;
IF MOD(year_date,4)<> 0 THEN
    SET leap = FALSE;
  ELSEIF MOD(year_date,100)<> 0 THEN
    SET leap = true;
  ELSEIF MOD(year_date,400)<> 0 THEN
    SET leap = FALSE;
  ELSE
    SET leap = TRUE;
  END IF;
  IF leap THEN
    RETURN (CONCAT(CONVERT(year_date,CHAR(4)),'年是闰年'));
  ELSE
    RETURN(CONCAT(CONVERT(year_date,CHAR(4)),'年是平年'));
  END IF;
END@@
DELIMITER;
SELECT leap_year(2012);
```

leap_year(2012)
2012年是闰年

3. CASE 语句

CASE 语句的作用和 IF…ELSEIF…END IF 语句相同，都可以实现多项选择。但 CASE 语句是一种更简洁的表示法，并且相对于 IF 结构表示法而言消除了一些重复。CASE 语句共有两种形式。

1）形式一

第一种形式是获取一个选择器的值，系统根据其值查找与此相匹配的 WHEN 常量，当找到一个匹配时，就执行与该 WHEN 常量相关的 THEN 子句。如果没有与选择器相匹配的 WHEN 常量，那么就执行 ELSE 子句。该语句的语法形式如下：

```
CASE <表达式>
   WHEN <表达式值 1> THEN SQL 语句块 1;
   WHEN <表达式值 2> THEN SQL 语句块 2;
   …
   WHEN <表达式值 n> THEN SQL 语句块 n;
   [ ELSE SQL 语句块 n + 1; ]
END;
```

【例 3-32】 判断显示 emp 表中前 3 条记录的姓名和职务。

```
SELECT ename 姓名,CASE job
  WHEN 'SALESMAN' THEN '销售员'
  WHEN 'CLERK' THEN '管理员'
  ELSE '经理'
END AS 职务
FROM emp LIMIT 3;
```

姓名	职务
SMITH	管理员
ALLEN	销售员
WARD	销售员

2) 形式二

第二种形式为不使用选择器,而是判断每个 WHEN 子句中的条件。该语句的语法形式如下:

```
CASE
  WHEN <条件 1> THEN SQL 语句块 1;
  WHEN <条件 2> THEN SQL 语句块 2;
  …
  WHEN <条件 n> THEN SQL 语句块 n;
  ELSE SQL 语句块 n + 1;
END;
```

【例 3-33】 判断显示 emp 表中前 3 条记录的姓名、基本工资和工资等级。

```
SELECT ename,sal,CASE
   WHEN sal BETWEEN 700 AND 1200 THEN '一级'
   WHEN sal BETWEEN 1201 AND 1400 THEN '二级'
   WHEN sal BETWEEN 1401 AND 2000 THEN '三级'
   WHEN sal BETWEEN 2001 AND 3000 THEN '四级'
   ELSE '五级'
  END 工资等级
  FROM emp LIMIT 3;
```

ename	sal	工资等级
SMITH	800.00	一级
ALLEN	1600.00	三级
WARD	1250.00	二级

3.2.4 循环语句

循环语句和条件语句一样,都能控制程序的执行流程,它允许重复执行一个语句或一组语句。MySQL 支持 3 种类型的循环,即 LOOP 循环、WHILE 循环和 REPEAT 循环。

1. LOOP 循环

LOOP 循环为无条件循环，这种类型的循环如果没有指定 LEAVE 语句，循环将一直运行，成为死循环。LEAVE 语句通常与条件语句结合，当条件表达式为真时，结束循环。该语句的语法形式如下：

```
标签:LOOP
  SQL 语句块;
  IF <条件表达式> THEN
    LEAVE 标签;
  END IF;
END LOOP;
```

【例 3-34】 LOOP 循环语句示例。创建存储函数 sum_fn()，返回 1～n 的和。

```
DELIMITER @@
CREATE FUNCTION sum_fn(n int)
RETURNS INT
BEGIN
 DECLARE s,i INT;
 SET s = 0,i = 1;
 loop_label: LOOP                    ##指明 LOOP 循环标签 loop_label
   SET s = s + i;
   SET i = i + 1;
   IF i > n THEN
     LEAVE loop_label;               ##通过标签结束 LOOP 循环
   END IF;
 END LOOP;
 RETURN s;
END@@

DELIMITER;
SELECT sum_fn(5);
```

sum_fn(5)
15

2. WHILE 循环

WHILE 循环在每次执行循环时，都将判断循环条件，如果它为 TRUE，那么循环将继续执行；如果条件为 FALSE，则循环将会停止执行。该语句的语法形式如下：

```
WHILE <条件表达式> DO
    SQL 语句块;
END WHILE;
```

【例 3-35】 WHILE 循环语句示例。创建存储函数 sum_fn()，返回 1～n 的和。

```
DELIMITER @@
```

```
CREATE FUNCTION sum_fn(n int)
RETURNS INT
BEGIN
 DECLARE s,i INT;
 SET s = 0,i = 1;
 WHILE i <= n DO
  SET s = s + i;
  SET i = i + 1;
 END WHILE;
 RETURN s;
END@@

DELIMITER;
SELECT sum_fn(5);
```

3. REPEAT 循环

在使用 REPEAT 循环语句时,首先执行其内部的循环语句块,在语句块的一次执行结束时判断条件表达式是否为真,如果为真,结束循环,否则重复执行其内部语句块。该语句的语法形式如下:

```
REPEAT
 SQL 语句块;
 UNTIL <条件表达式>
END REPEAT;
```

【例 3-36】 REPEAT 循环语句示例。创建存储函数 sum_fn(),返回 1～n 的和。

```
DELIMITER @@
CREATE FUNCTION sum_fn(n int)
RETURNS INT
BEGIN
 DECLARE s,i INT;
 SET s = 0,i = 1;
 REPEAT
  SET s = s + i;
  SET i = i + 1;
  UNTIL i > n
 END REPEAT;
 RETURN s;
END@@

DELIMITER;
SELECT sum_fn(5);
```

3.3 存 储 过 程

简单地说,存储过程就是一条或者多条 SQL 语句的集合,利用这些 SQL 语句完成一个或者多个逻辑功能。

存储过程可以被赋予参数，存储在数据库中，可以被用户调用，也可以被 Java 或者 C＃等编程语言调用。由于存储过程是已经编译好的代码，所以在调用的时候不必再次进行编译，从而提高了程序的运行效率。

3.3.1 创建存储过程

创建存储过程需要使用 CREATE PROCEDURE 语句，创建存储过程的语法形式如下：

```
CREATE PROCEDURE 存储过程名()
BEGIN
  过程体;
END
```

【例 3-37】 创建存储过程 emp_p，在 emp 表中查询职工编号为 7369 的员工的姓名和工作。

```
SET GLOBAL log_bin_trust_function_creators = 1;

DELIMITER @@

CREATE PROCEDURE emp_p()
BEGIN
 SELECT ename,job FROM emp
    WHERE empno = 7369;
END@@
```

3.3.2 调用存储过程

一旦创建了存储过程，之后就可以任意调用该存储过程。

用户可以使用 CALL 语句直接调用存储过程。CALL 语句的语法形式如下：

```
CALL 存储过程名();
```

【例 3-38】 调用执行例 3-37 创建的存储过程。

```
DELIMITER;
CALL emp_p();
```

ename	job
SMITH	CLERK

3.3.3 存储过程的参数

在创建存储过程时需要考虑存储过程的灵活应用，以便重新使用它们。通过使用“参数”可以使程序单元变得灵活。参数是一种向程序单元输入、输出数据的机制。存储过程可以接收和返回 0 到多个参数。MySQL 有 3 种参数模式，即 IN、OUT 和 INOUT。

创建带参数的存储过程的语法格式如下：

```
CREATE PROCEDURE 存储过程名(
  [ IN | OUT | INOUT]参数 1 数据类型,
  [ IN | OUT | INOUT]参数 2 数据类型,…
```

```
)
BEGIN
  过程体;
END
```

1. IN 参数

IN 参数为输入参数,该参数值由调用者传入,并且只能够被存储过程读取。

【例 3-39】 创建一个向 dept 表中插入新记录的存储过程 dept_p1。

```
DELIMITER @@
 CREATE PROCEDURE dept_p1(
  IN p_deptno DECIMAL(2,0),
  IN p_dname VARCHAR(14),
  IN p_loc VARCHAR(13)
 )
 BEGIN
  INSERT INTO dept
    VALUES(p_deptno,p_dname,p_loc);
 END@@

DELIMITER;
CALL dept_p(50,'HR','CHINA');
SELECT * FROM dept WHERE deptno = 50;
```

deptno	dname	loc
50	HR	CHINA

2. OUT 参数

OUT 参数为输出参数,该类型的参数值由存储过程写入。OUT 类型的参数适用于存储过程向调用者返回多条信息的情况。

【例 3-40】 创建存储过程 dept_p2,该过程根据提供的部门编号返回部门的名称和地址。

```
DELIMITER @@
CREATE PROCEDURE dept_p2(
  IN i_no DECIMAL(2,0),
  OUT o_name VARCHAR(14),
  OUT o_loc VARCHAR(13)
 )
 BEGIN
  SELECT dname,loc INTO o_name,o_loc FROM dept
     WHERE deptno = i_no;
 END@@

DELIMITER;
CALL dept_p2(10,@v_dname,@v_loc);
SELECT @v_dname,@v_loc;
```

@v_dname	@v_loc
ACCOUNTINT	NEW YORK

3. INOUT 参数

IN 参数可以接收一个值，但是不能在过程中修改这个值；对于 OUT 参数而言，它在调用过程时为空，在过程的执行中将为这个参数指定一个值，并在执行结束后返回；INOUT 类型的参数同时具有 IN 参数和 OUT 参数的特性，在过程中可以读取和写入该类型参数。

【例 3-41】 使用 INOUT 参数实现两个数的交换。

```
DELIMITER @@
 CREATE PROCEDURE swap(
  INOUT p_num1 int,
  INOUT p_num2 int
  )
  BEGIN
   DECLARE var_temp int;
   SET var_temp = p_num1;
   SET p_num1 = p_num2;
   SET p_num2 = var_temp;
  END@@

DELIMITER;
SET @v_num1 = 1;
SET @v_num2 = 2;
CALL swap(@v_num1,@v_num2);
SELECT @v_num1,@v_num2;
```

@v_num1	@v_num2
2	1

3.3.4 删除存储过程

删除存储过程是指删除数据库中已经存在的存储过程，在 MySQL 中使用 DROP PROCEDURE 语句来删除存储过程。DROP PROCEDURE 语句的语法形式如下：

```
DROP PROCEDURE 存储过程名;
```

【例 3-42】 删除已创建的存储过程 emp_p。

```
DROP PROCEDURE emp_p;
```

3.4 游　　标

当通过 SELECT 语句查询时，返回的结果是一个由多行记录组成的集合，而程序设计语言并不能处理以集合形式返回的数据，为此 SQL 提供了游标机制。游标充当指针的作用，使应用程序设计语言一次只能处理查询结果中的一行。

在 MySQL 中，为了处理由 SELECT 语句返回的一组记录，可以在存储程序中声明和处理游标。

3.4.1 游标的定义和使用

游标是在存储程序中使用包含 SELECT 语句声明的游标。如果需要处理从数据库中检索的一组记录,则可以使用显式游标。使用游标处理数据需要 4 个步骤,即声明游标、打开游标、提取数据和关闭游标。

1. 声明游标

在存储程序中,声明游标与声明变量一样,都需要使用 DECLARE 语句。其语法形式如下:

```
DECLARE 游标名 CURSOR
 FOR SELECT 语句;
```

说明:

(1) 声明游标的作用是得到一个 SELECT 查询结果集,在该结果集中包含了应用程序中要处理的数据,从而为用户提供逐行处理的途径。

(2) SELECT 语句是对表或视图的查询语句,可以带 WHERE 条件、ORDER BY 或 GROUP BY 等子句,但不能使用 INTO 子句。

2. 打开游标

打开游标使用 OPEN 语句,其语法形式如下:

```
OPEN 游标名;
```

游标必须先声明后打开。在打开游标时,SELECT 语句的查询结果被传送到了游标工作区,以供用户读取。

3. 提取数据

在游标打开后,使用 FETCH 语句将游标工作区中的数据读取到变量中,其语法形式如下:

```
FETCH 游标名 INTO 变量名 1[,变量名 2…];
```

在成功打开游标后,游标指针指向结果集的第 1 行之前,而 FETCH 语句将使游标指针指向下一行。因此,第一次执行 FETCH 语句时,将检索第 1 行中的数据保存到变量中。随后每执行一次 FETCH 语句,该指针将移动到结果集的下一行。用户可以在循环中使用 FETCH 语句,这样每一次循环都会从表中读取一行数据,然后进行相同的逻辑处理。

4. 关闭游标

在游标使用完之后,需要用 CLOSE 语句关闭,其语法形式如下:

```
CLOSE 游标名;
```

游标一旦被关闭,游标占用的资源就会被释放,用户不能再从结果集中检索数据。如果想重新检索,必须重新打开游标才可以。

【例 3-43】 创建存储过程 emp_p,用游标提取 emp 表中 7788 雇员的姓名和职务。

```
DELIMITER @@
CREATE PROCEDURE emp_p()
 BEGIN
  DECLARE v_ename VARCHAR(14);                ##定义存放姓名值的变量
  DECLARE v_job VARCHAR(13);                  ##定义存放工作值的变量
  DECLARE emp_cursor CURSOR                   ##声明游标
    FOR SELECT ename,job FROM emp
        WHERE empno = 7788;
  OPEN emp_cursor;                            ##打开游标
  FETCH emp_cursor INTO v_ename,v_job;        ##提取游标数据到变量
  CLOSE emp_cursor;                           ##关闭游标
  SELECT v_ename,v_job;
 END@@

DELIMITER;
CALL emp_p();
```

v_ename	v_job
SCOTT	ANALYST

【例 3-44】 创建存储过程 emp_p1,用游标显示工资最高的前 3 名雇员的姓名和工资。

```
DELIMITER @@
CREATE PROCEDURE emp_p1()
 BEGIN
  DECLARE v_ename VARCHAR(14);
  DECLARE v_sal DECIMAL(7,2);
  DECLARE i INT;
  DECLARE mycursor CURSOR
   FOR SELECT ename,sal FROM emp
         ORDER BY sal DESC
          LIMIT 3;
  SET i = 1;
  CREATE TABLE result(
   ename VARCHAR(14),
   sal DECIMAL(7,2)
  );
  OPEN mycursor;
  WHILE i <= 3 DO
   FETCH mycursor INTO v_ename,v_sal;
   INSERT INTO result VALUES(v_ename,v_sal);
   SET i = i + 1;
  END WHILE;
  CLOSE mycursor;
  SELECT * FROM result;
 END@@

DELIMITER;
CALL emp_p1();
```

ename	sal
KING	5000.00
FORD	3000.00
SCOTT	3000.00

3.4.2 异常处理

在存储程序中处理 SQL 语句可能导致一条错误消息,并且 MySQL 立即停止对存储过程的处理。例如,向一个表中插入新的记录而主键值已经存在,这条 INSERT 语句会导致一个出错消息。当存储过程发生错误时,数据库开发人员并不希望 MySQL 自动终止存储过程的执行,而是通过 MySQL 的错误处理机制帮助数据库开发人员控制程序流程。

存储程序中的异常处理通过 DECLARE HANDLER 语句实现,其语法形式如下。

```
DECLARE 错误处理类型 HANDLER FOR 错误触发条件 自定义错误处理程序;
```

说明:

(1) 在一般情况下,异常处理语句置于存储程序(存储过程或存储函数)中才有意义。

(2) 异常处理语句必须放在所有变量及游标定义之后,所有 MySQL 表达式之前。

(3) 错误处理类型。错误处理类型只有 CONTINUE 和 EXIT 两种,CONTINUE 表示错误发生后 MySQL 立即执行自定义错误处理程序,然后忽略该错误继续执行其他 MySQL 语句; EXIT 表示错误发生后 MySQL 立即执行自定义错误处理程序,然后立刻停止其他 MySQL 语句的执行。

(4) 错误触发条件。错误触发条件定义了自定义错误处理程序运行的时机。错误触发条件的形式如下。

```
SQLSTATE 'ANSI 标准错误代码'
 |MySQL 错误代码
 |SQLWARNING
 |NOT FOUND
 |SQLEXCEPTION
```

- 错误触发条件支持标准的 SQLSTATE 定义,也支持 MySQL 的错误代码。
- SQLWARNING 表示对所有以 01 开头的 SQLSTATE 代码的速记。
- NOT FOUND 表示对所有以 02 开头的 SQLSTATE 代码的速记。
- SQLEXCEPTION 表示对所有没有被 SQLWARNING 或 NOT FOUND 捕获的 SQLSTATE 代码的速记。

(5) 自定义错误处理程序。错误发生后,MySQL 会立即执行自定义错误处理程序中的 MySQL 语句。

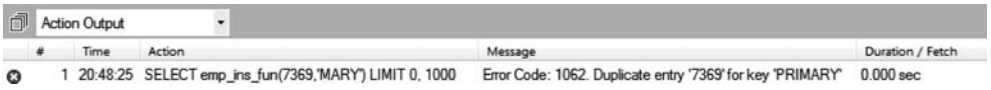
【例 3-45】 创建存储函数 emp_ins_fun(),向 emp 表中插入一条记录,empno 和 ename 字段的值为 7396、'MARY',已知雇员编号 7369 已存在于 emp 表中,违反了主键约束。

```
DELIMITER @@
CREATE FUNCTION emp_ins_fun(no DECIMAL(4,0),name VARCHAR(14))
  RETURNS VARCHAR(20)
  BEGIN
```

```
    INSERT INTO emp(empno,ename)
      VALUES(no,name);
    RETURN '插入成功';
  END@@

DELIMITER;
SELECT emp_ins_fun(7369,'MARY');
```

执行后的出错信息如下：



下面存储函数的创建加入了错误处理机制，数据库开发人员控制程序的运行流程，解决MySQL自动终止存储程序执行的问题。

```
DELIMITER @@
CREATE FUNCTION emp_ins_fun(no DECIMAL(4,0),name VARCHAR(14))
  RETURNS VARCHAR(20)
  BEGIN
   DECLARE EXIT HANDLER FOR SQLSTATE '23000'
     RETURN '违反主键约束!';
   INSERT INTO emp(empno,ename)
     VALUES(no,name);
   RETURN '插入成功';
  END@@

DELIMITER;
SELECT emp_ins_fun(7369,'MARY');
```

emp_ins_fun(7369,'MARY')
违反主键约束!

```
DELIMITER;
SELECT emp_ins_fun(7000,'MARY');
```

emp_ins_fun(7000,'MARY')
插入成功

注意：错误处理语句 DECLARE EXIT HANDLER FOR SQLSTATE '23000'可以替换成 DECLARE EXIT HANDLER FOR 1062，因为 MySQL 错误代码 1062 对应于 ANSI 标准错误代码 23000。

【例 3-46】 创建存储过程 emp_up_pro，使用游标更新 emp_c 表（与 emp 表相同）中的 comm 值。

```
DELIMITER @@
CREATE PROCEDURE emp_up_pro()
 BEGIN
  DECLARE v_empno DECIMAL(4,0);
  DECLARE v_sal DECIMAL(7,2);
  DECLARE v_comm DECIMAL(7,2);
  DECLARE flag BOOLEAN DEFAULT TRUE;
```

```
  DECLARE comm_cur CURSOR
   FOR SELECT empno,sal FROM emp_c;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
   SET flag = FALSE;
  OPEN comm_cur;
  WHILE flag DO
   FETCH comm_cur INTO v_empno,v_sal;
   IF v_sal < 500 THEN SET v_comm = v_sal * 0.25;
   ELSEIF v_sal < 1000 THEN SET v_comm = v_sal * 0.2;
   ELSEIF v_sal < 3000 THEN SET v_comm = v_sal * 0.15;
   ELSE SET v_comm = v_sal * 0.12;
   END IF;
   UPDATE emp_c SET comm = v_comm
    WHERE empno = v_empno;
  END WHILE;
  CLOSE comm_cur;
 END@@

DELIMITER;
SET SQL_SAFE_UPDATES = 0;
CALL emp_up_pro();
SELECT * FROM emp_c;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980-12-17	800.00	160.00	20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600.00	240.00	30
7521	WARD	SALESMAN	7698	1981-02-22	1250.00	187.50	30
7566	JONES	MANAGER	7839	1981-04-02	2975.00	446.25	20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250.00	187.50	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	427.50	30
7782	CLARK	MANAGER	7839	1981-06-09	2450.00	367.50	10
7788	SCOTT	ANALYST	7566	1987-04-19	3000.00	360.00	20

3.5 嵌入式 SQL**

SQL 是一种双重式语言,它既是一种用于查询和更新的交互式数据库语言,又是一种应用程序进行数据库访问时所采取的编程式数据库语言。SQL 语言在这两种方式中的大部分语法是相同的。在编写访问数据库的程序时,必须从普通的编程语言开始(例如 C 语言),再把 SQL 加入程序中。所以,嵌入式 SQL(ESQL)就是将 SQL 语句嵌入程序设计语言中,被嵌入的程序设计语言(例如 C、C++、Java)称为宿主语言,简称主语言。本节以 C 语言作为主语言。

3.5.1 SQL 与宿主语言接口

对于嵌入式 SQL 语句,一般采用预编译方法处理,即由 RDBMS 的预处理程序对源程序进行扫描,识别出 ESQL 语句,把它们转换成主语言调用语句,以使主语言编译程序能识别它们,最后由主语言的编译程序将整个源程序编译成目标码。

可见,将 SQL 嵌入主语言使用时应当注意如下问题。

1. 区分主语言语句与 SQL 语句

在嵌入式 SQL 中，为了能够快速区分 SQL 语句与主语言语句，所有 SQL 语句都必须加前缀。当主语言为 C 语言时，语法形式如下：

```
EXEC SQL SQL 语句;
```

2. 嵌入式 SQL 语句与主语言的通信

在主语言中嵌入 SQL 语句进行混合编程，其主要目的是发挥 SQL 语言和主语言各自的优势。SQL 语句是面向集合的描述性、非过程化语言，负责与数据库的数据交换；主语言是过程化的、与运行环境有关的语言，主要负责用户界面及控制程序流程，而程序流程与程序语句所处的变量环境有关。在程序执行过程中，主语言需要和 SQL 语句进行信息交换，其间的通信过程如下。

(1) SQL 语句将执行状态信息传递给主语言。主语言得到该状态信息后，可根据此状态信息来控制程序流程，以控制后面的 SQL 语句或主语言语句的执行。向主语言传递 SQL 执行状态信息，主要用 SQL 通信区(SQL Communication Area，SQLCA)实现。

(2) 主语言需要提供一些变量参数给 SQL 语句。该方法是在主语言中定义主变量(Host Variable)，在 SQL 语句中使用主变量，将参数值传递给 SQL 语句。

(3) 将 SQL 语句查询数据库的结果返回给主语言做进一步处理。如果 SQL 语句向主语言返回的是一条数据库记录，可使用主变量；若返回值为多条记录的集合，则使用游标。

3.5.2 SQL 通信区

在 SQL 语句执行后，系统要反馈给应用程序若干信息，主要包括描述系统当前工作状态和运行环境的各种参数，这些信息将被送到 SQL 通信区——SQLCA 中。主语言的应用程序从 SQLCA 中取出这些状态信息，据此决定后面语句的执行。

SQLCA 是一个数据结构，在程序的主语言中用 EXEC SQL INCLUDE SQLCA 加以定义。在 SQLCA 中有一个系统变量 SQLCODE，用来存放每次执行 SQL 语句后返回的代码。

应用程序在每执行一条 SQL 语句后均测试一下 SQLCODE 的值，以了解该 SQL 语句的执行情况并做相应处理。如果 SQLCODE 等于预定义的常量 SUCCESS，则表示 SQL 语句成功，否则在 SQLCODE 中存放错误代码。程序员可以根据错误代码查找问题。

因此，系统变量 SQLCODE 用于向主语言提供 SQL 语句执行的状态。例如，在执行更新语句 UPDATE 后，SQLCA 用如下状态之一返回其执行结果。

(1) 执行成功(SQLCA＝SUCCESS)，并有更新的行数。

(2) 违反完整性约束，更新操作被拒绝执行。

(3) 没有满足更新条件的行，一行也没有更新。

(4) 由于其他原因，执行出错。

3.5.3 主变量的定义与使用

在嵌入式 SQL 语句中可以使用主语言的程序变量来输入或输出数据。在 SQL 语句中使用的主语言程序变量简称为主变量。主变量根据其作用不同，分为输入主变量和输出主

变量。

在 SELECT INTO 和 FETCH 语句之后的主变量称为“输出主变量”,这是因为从数据库传递列数据到应用程序。除了 SELECT INTO 和 FETCH 语句以外的其他 SQL 语句中的主变量,称为“输入主变量”,这是因为从应用程序向数据库输入值,例如 INSERT、UPDATE 等语句。

1. 主变量的定义

在使用主变量之前,必须在 SQL 语句 BEGIN DECLARE SECTION 与 END DECLARE SECTION 之间进行声明。在声明之后,主变量可以在 SQL 语句中任何一个能够使用表达式的地方出现,为了与数据库对象名(例如表名、视图名、列名等)区别,应在 SQL 语句中的主变量名前加冒号(:)。

在使用主变量时应注意以下几个内容。

(1) 主变量在使用前,必须在嵌入 SQL 语句的说明部分明确定义。

(2) 主变量在定义时,所用的数据类型应为主语言提供的数据类型,而不是 SQL 的数据类型。同时要注意主变量的大小写。

(3) 在 SQL 语句中使用主变量时,必须在主变量前加一个冒号(:),在不含 SQL 语句的主语言语句中,则不需要在主变量前加冒号。

(4) 主变量不能是 SQL 命令的关键字,例如 SELECT 等。

(5) 在一条 SQL 语句中,主变量只能使用一次。

【例 3-47】 主变量定义示例。

```
EXEC SQL BEGIN DECLARE SECTION;     /* 说明主变量 */
 char msno[4],mcno[3],givensno[5];
 int mgrade;
 char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

在上面这个例子中说明了 5 个主变量,其中,SQLSTATE 是一个特殊的主变量,起着解释 SQL 语句执行状况的作用。当 SQL 语句执行成功时,系统自动给 SQLSTATE 赋上全零值,否则为非全零(“02000”)。因此,在执行一条 SQL 语句后,可以根据 SQLSTATE 的值转向不同的分支,以控制程序的流向。

2. 在 SELECT 语句中使用主变量

在嵌入式 SQL 中,如果查询结果为单记录,则 SELECT 语句需要用 INTO 子句指定查询结果的存放地点——主变量。

【例 3-48】 在 SELECT 查询中使用主变量示例。根据主变量 givensno 值查询成绩表 grade 中学生的学号、课号和分数。

```
EXEC SQL SELECT 学号,课号,分数
     INTO msno,mcno,mgrade
     FROM grade
     WHERE 学号 = :givensno;
```

3. 在 INSERT 语句中使用主变量

在 INSERT 语句的 VALUES 子句中,可以使用主变量指定插入的值。

【例 3-49】 在 INSERT 语句中使用主变量示例。某学生选修了一门课程，将其信息插入成绩表 grade 中，假设学号、课号、分数已分别赋给主变量 hsno、hcno、hgrade。

```
EXEC SQL INSERT INTO grade(学号,课号,分数)
     VALUES(:hsno,:hcno,:hgrade);
```

4. 在 UPDATE 语句中使用主变量

在 UPDATE 语句的 SET 子句和 WHERE 子句中，均可以使用主变量。

【例 3-50】 在 UPDATE 语句中使用主变量示例。更新 grade 表中指定学生指定课程的分数。

```
EXEC SQL UPDATE grade
     SET 分数 = :mgrade
     WHERE 学号 = :msno AND 课号 = :mcno;
```

5. 在 DELETE 语句中使用主变量

在 DELETE 语句的 WHERE 子句中，可以使用主变量指定删除条件。

【例 3-51】 在 DELETE 语句中使用主变量示例。删除 grade 表中指定学生的信息。

```
EXEC SQL DELETE FROM grade
     WHERE 学号 = :msno;
```

3.5.4 嵌入式 SQL 中游标的定义与使用

用嵌入式 SQL 语句查询数据分成两类情况，一类是单行结果，一类是多行结果。对于单行结果，可以使用 SELECT INTO 语句；对于多行结果，则必须使用游标来完成。游标是一个与 SELECT 语句相关联的符号名，它使用户可逐行访问游标返回的结果集。

游标的使用与前面的介绍相同，包括声明游标、打开游标、提取数据和关闭游标这 4 步。

1. 声明游标

在嵌入式 SQL 中，用 DECLARE 语句定义游标的一般形式如下：

```
EXEC SQL DECLARE 游标名 CURSOR
  FOR SELECT 语句;
```

2. 打开游标

在嵌入式 SQL 中，也是用 OPEN 语句打开游标。OPEN 语句的一般形式如下：

```
EXEC SQL OPEN 游标名;
```

3. 提取数据

提取数据是指从缓冲区中将当前记录取出来，送到主变量供主语言进一步处理，同时移动游标指针。

在嵌入式 SQL 中，FETCH 语句的一般形式如下：

```
EXEC SQL FETCH FROM 游标名 INTO 主变量[,主变量,…];
```

其中,主变量必须与游标中 SELECT 语句的列表达式一一对应。FETCH 语句通常用在一个循环结构中,通过循环执行 FETCH 语句逐条取出结果集中的行进行处理。

4. 关闭游标

在使用游标结束后,应关闭游标,以释放游标占用的缓冲区及其他资源。用 CLOSE 语句关闭游标,CLOSE 语句的一般形式如下:

```
EXEC SQL CLOSE 游标名;
```

游标被关闭后,就不再与原来查询所返回的结果相联系,而被关闭的游标可以再次被打开,以返回新的查询结果。

5. 程序实例

为了能够更好地理解有关概念,下面给出一个简单的 ESQL 编程实例。

【例 3-52】 在 C 语言中嵌入 SQL 的查询,检索某学生的学习成绩,其学号由主变量 givensno 给出,结果放在主变量 msno、mcno、mgrade 中。如果成绩不及格,则删除该记录;如果成绩为 60～69 分,则将成绩修改为 70 分,并显示学生的成绩信息。

```
#DEFINE NO_MORE_TUPLES !(strcmp(SQLSTATE,"02000"))
void sel()
{ EXEC SQL BEGIN DECLARE SECTION;
   char msno[4],mcno[3],givensno[5];
   int mgrade;
   char SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;
  EXEC SQL DECLARE gradex CURSOR FOR
   SELECT 学号,课号,分数 FROM grade
    WHERE 学号 = :givensno;
  EXEC SQL OPEN gradex;
  while(1){
     EXEC SQL FETCH FROM gradex
       INTO :msno,:mcno,:mgrade;
     if(NO_MORE_TUPLES) break;
     if(mgrade<60)
       EXEC SQL DELETE FROM grade
         WHERE CURRENT OF gradex;
     else{
       if(mgrade<70){
         EXEC SQL UPDATE grade SET 分数 = 70
           WHERE CURRENT OF gradex;
         mgrade = 70;
       }
    }
    printf("%s,%s,%d",msno,mcno,mgrade);
  }
}
```

3.5.5 动态 SQL 语句

前面提到的嵌入式 SQL 语句必须在源程序中完全确定,然后再由预处理程序预处理和

由主语言编译程序编译。在实际应用中,源程序往往不能包括用户的所有操作。用户对数据库的操作有时在系统运行时才能提出来,这时要用到嵌入式 SQL 的动态技术才能实现。

动态 SQL 技术主要有以下两个 SQL 语句。

1) 动态 SQL 预备语句

```
EXEC SQL PREPARE 动态 SQL 语句名 FROM 共享变量或字符串;
```

这里共享变量或字符串的值应是一个完整的 SQL 语句。这个语句可以在程序运行时由用户输入组合起来。此时,这个语句并不执行。

2) 动态 SQL 执行语句

```
EXEC SQL EXECUTE 动态 SQL 语句名;
```

动态 SQL 语句在使用时,还可以有以下两点改进。

(1) 当预备语句中组合而成的 SQL 语句只需执行一次时,预备语句和执行语句可合并成一个语句:

```
EXEC SQL EXECUTE IMMEDIATE 共享变量或字符串;
```

(2) 当预备语句中组合而成的 SQL 语句的条件值尚缺时,可以在执行语句中用 USING 短语补上:

```
EXEC SQL EXECUTE 动态 SQL 语句名 USING 共享变量;
```

【例 3-53】 下面两个 C 语言程序段说明了动态 SQL 语句的使用技术。

程序段一:

```
EXEC SQL BEGIN DECLARE SECTION;
  char * query;
EXEC SQL END DECLARE SECTION;
scanf("%s",query);                    /* 从键盘输入一个 SQL 语句 */
EXEC SQL PREPARE que FROM :query;
EXEC SQL EXECUTE que;
```

这个程序段表示从键盘输入一个 SQL 语句到字符数组中。字符指针 query 指向字符串的第 1 个字符。

如果执行语句只做一次,那么程序段最后两个语句可合并成一个语句:

```
EXEC SQL EXECUTE IMMEDIATE :query;
```

程序段二:

```
char * query = "DELETE FROM grade WHERE 学号 = ?";
EXEC SQL PREPARE dynprog FROM :query;
char sno[5] = "1001";
EXEC SQL EXECUTE dynprog USING :sno;
```

这里第 1 个 char 语句表示要执行的一个 SQL 语句,但有一个值(学生学号)还不能确

定,因此用“?”表示;第 2 个语句是动态 SQL 预备语句;第 3 个语句(char 语句)表示取到学生的学号值;第 4 个语句是动态 SQL 执行语句,“?”值到共享变量 sno 中取。

3.6 小　　结

本章介绍了 MySQL 编程的基础知识,包括变量的声明和使用、编程中常用的系统函数等,还介绍了以下内容。

存储程序中的 IF 语句可执行简单条件判断、二重分支判断和多重分支判断。在使用 IF 语句时,注意 END IF 是两个词,而 ELSEIF 是一个词。CASE 语句执行多重分支判断。LOOP 语句、WHILE 语句和 REPEAT 语句执行循环控制操作的方法。

在使用 SELECT 语句查询数据库时,查询返回的数据存放在结果集中。用户在得到结果集后,需要逐行、逐列地获取其中存储的数据,从而在应用程序中使用这些值,游标机制可完成此类操作。

当存储程序运行错误时,可以使用异常处理的方法来处理发生的错误。

存储过程是指用于执行特定操作的 SQL 语句的集合,在需要时可以直接调用,从而提高代码的重用性和共享性。

SQL 的用户可以是终端用户,也可以是应用程序。嵌入式 SQL 是将 SQL 作为一种数据子语言嵌入高级语言,利用高级语言和其他专门软件来弥补 SQL 语句在实现复杂应用方面的不足。动态 SQL 允许在执行一个应用程序时根据不同的情况动态地定义和执行某些 SQL 语句。动态 SQL 可实现应用中的灵活性。

习　题　3

一、选择题

1. 下列用来取绝对值的函数是(　　)。

A. MAX　　B. REPLACE　　C. ABS　　D. ABC

2. 下列用来返回当前登录名的函数是(　　)。

A. USER　　B. SHOW USER

C. SESSION_USER　　D. SHOW USERS

3. 下面创建自定义函数语法的是(　　)。

A. CREATE TABLE　　B. CREATE VIEW

C. CREATE FUNCTION　　D. 以上都不是

4. 下面删除自定义函数语法的是(　　)。

A. DROP TABLE　　B. DROP VIEW

C. DROP FUNCTION　　D. 以上都不是

5. 下面对存储过程的描述正确的是(　　)。

A. 存储过程创建好就不能够修改了

B. 修改存储过程相当于重新创建一个存储过程

C. 存储过程在数据库中只能应用一次

D. 以上都正确

6. 下面对存储过程的描述不正确的是(　　)。

A. 在存储过程中可以定义变量

B. 修改存储过程相当于重新创建一个存储过程

C. 存储过程不调用就可以直接使用

D. 以上都是错误的

7. 对于结构控制语句,下面为循环语句的是(　　)。

A. IF　　B. CASE　　C. WHICH　　D. LOOP

8. 下列关键字用来在 IF 语句中检查多个条件的是(　　)。

A. ELSE IF　　B. ELSEIF

C. ELSIF　　D. ELSIFS

9. 下列有关嵌入式 SQL 的叙述,不正确的是(　　)。

A. 主语言是指 C 一类高级程序设计语言

B. 主语言是指 SQL 语言

C. 在程序中要区分 SQL 语句和主语言语句

D. SQL 有交互式和嵌入式两种使用方式

10. 嵌入式 SQL 在实现时,采用预处理方式的目的是(　　)。

A. 把 SQL 语句和主语言语句区分开来

B. 为 SQL 语句加前缀标识和结束标识

C. 识别出 SQL 语句,并处理成函数调用形式

D. 把 SQL 语句编译成二进制码

11. 允许在嵌入的 SQL 语句中引用主语言的程序变量,在引用时(　　)。

A. 直接引用

B. 这些变量前必须加符号“*”

C. 这些变量前必须加符号“:”

D. 这些变量前必须加符号“&”

12. 如果嵌入的 SELECT 语句的查询结果肯定是单元组,那么嵌入时(　　)。

A. 肯定不涉及游标机制

B. 必须使用游标机制

C. 是否使用游标由应用程序员决定

D. 是否使用游标与 DBMS 有关

二、简答题

1. MySQL 存储过程和函数有什么区别?

2. 存储过程中的代码可以改变吗?

3. 在存储过程中可以调用其他存储过程吗?