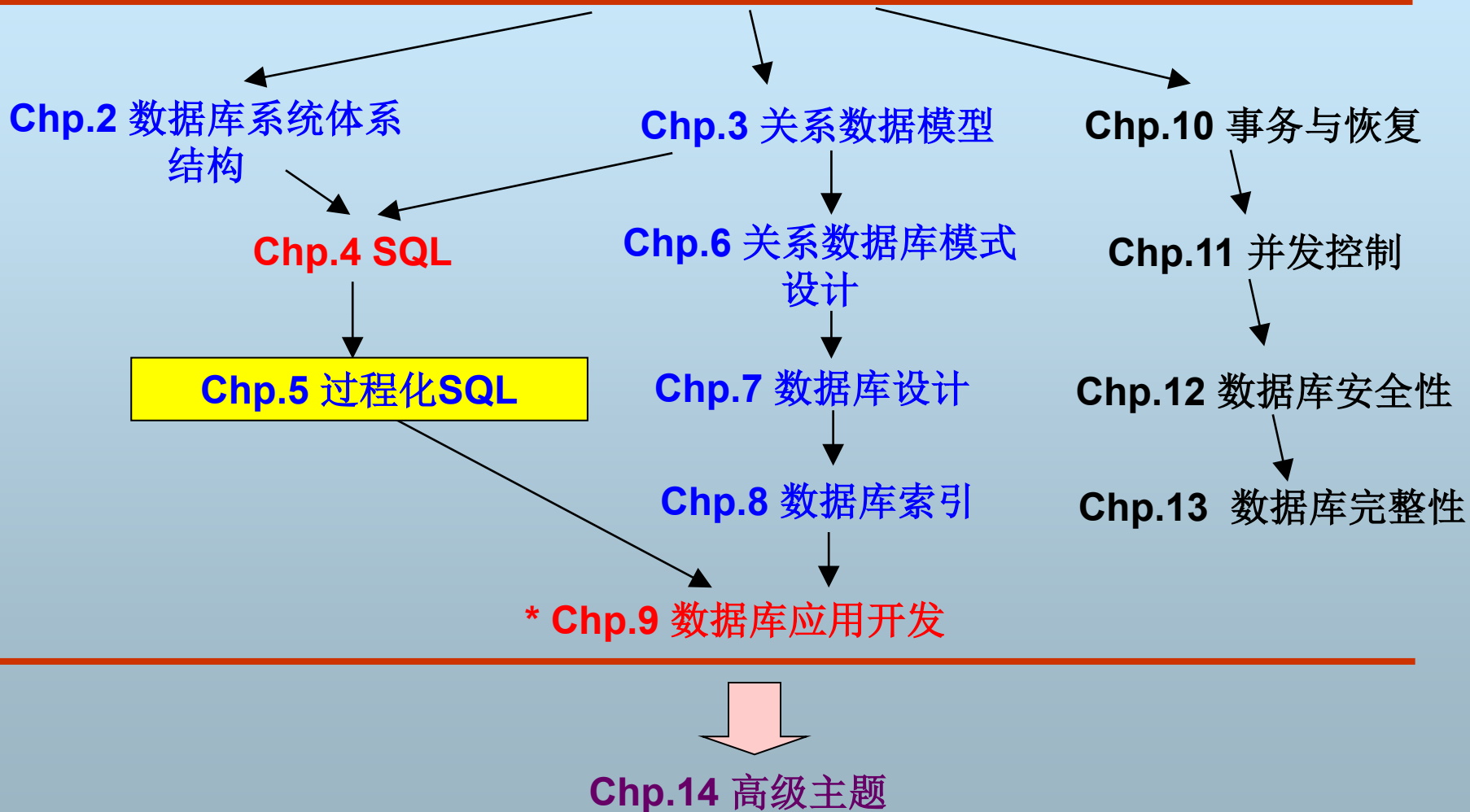


第5章 过程化SQL



课程知识结构

Chp.1 数据库系统概述



本章主要内容

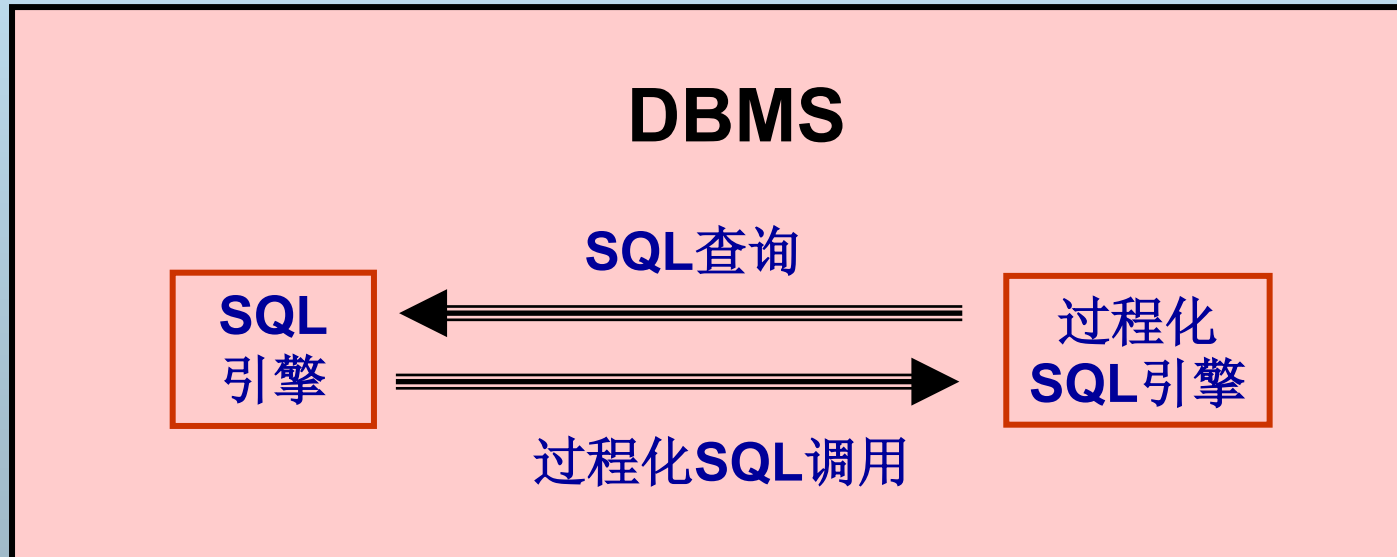
- 过程化SQL vs. SQL
- 过程化SQL编程
- 事务编程
- 游标 (Cursor)
- 存储过程 (Stored Procedure)
- 触发器 (Trigger)

一、过程化SQL与SQL

- **SQL是描述性语言，过程化SQL是对SQL的一个扩展，是一种过程化的程序设计语言**
 - **SQL本身并不能建立数据库应用程序**
 - **过程化SQL是包含SQL的一种过程性语言，它不仅支持SQL，还支持一些过程性语言特性**
- **其它商用DBMS一般也都提供类似的扩展**
 - **Oracle — PL/SQL**
 - **Microsoft/Sybase — Transact-SQL (T-SQL)**
 - **IBM DB2 — SQL PL**
 - **PostgreSQL — PL/pgSQL**

一、过程化SQL与SQL

- 二者均可以在DBMS中运行，可以相互调用



一、过程化SQL与SQL

从A账号转帐
100到B账号

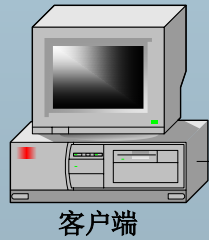
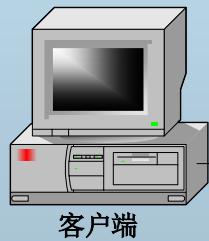
Total 5
客户端多次计算
多次网络传输

SQL

客户机

- 1、通过SQL查询账号A是否存在
- 2、通过SQL查询账号B是否存在
- 3、查询账号A上的余额
- 4、修改A上的余额
- 5、修改B上的余额

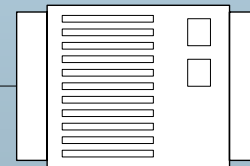
SQL Result



网络

SQL

结果



一、过程化SQL与SQL

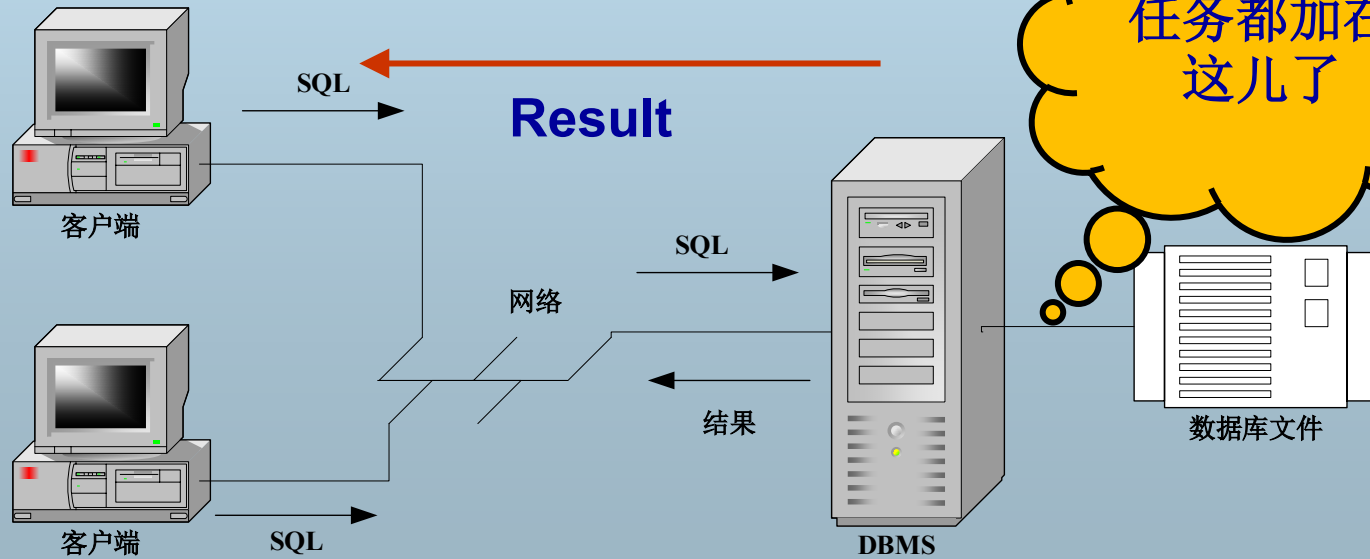
从A账号转帐
100到B账号

Total 1
客户端更少计算
更少网络传输

客户机
执行一个预先编写好并存储
在服务器上的过程化
SQL程序完成转帐

调用过程化SQL程序

任务都加在这儿了



一、过程化SQL与SQL

■ 在程序中使用SQL

- 客户机计算任务多
- 网络传输重
- 服务器计算任务少

胖客户机、瘦服务器

■ 在程序中使用过程化SQL

- 可以完成一些SQL不能完成的复杂计算，并且封装处理逻辑
- 客户机计算任务少
- 服务器计算任务加重
- 网络传输少

瘦客户机、胖服务器

二、过程化SQL的程序结构

■ 第1种结构

● 在一次会话中使用过程化SQL语句编程

- ◆ 赋值，输出，表达式计算，函数等等
- ◆ 不能使用流程控制语句（分支、循环）
- ◆ 不支持定义语句块Begin.....End
- ◆ 不允许定义局部变量

MySQL	Oracle	MS SQL Server
X	√	√
X	√	√
X	√	√

```
mysql80 test 运行 停止 解释
1 SELECT sname FROM Student WHERE sno = 's1' INTO @a;
2 SELECT sname FROM student WHERE sno = 's2' INTO @b;
3 SET @c = concat( @a, '##', @b );
4 SELECT @c;

信息 结果 1 剖析 状态
@c
a##b
```

二、过程化SQL的程序结构

■ 第2种结构

- 在存储过程和触发器中编程
 - ◆ 支持全部的过程化程序设计要素
 - ◆ 支持事务编程

MySQL	Oracle	MS SQL Server
√	√	√
√	√	√

Delimiter语句重定义程序结束标记

```
mysql80 test 运行 停止 解释
1
2 Delimiter //
3 CREATE PROCEDURE g4Score(IN sn VARCHAR ( 50 ), OUT sout FLOAT)
4 BEGIN
5     DECLARE
6         n FLOAT;
7     SELECT avg( score ) FROM SC WHERE sno=sn INTO n;
8     SET sout=n;
9 END //
10 Delimiter;
```

定义存储过程

```
mysql80 test 运行
1 -- Test stored procedure
2 call g4score('s1',@avg_score);
3 select @avg_score;

信息 结果 1 剖析 状态
@avg_score
86.66666412353516
```

调用存储过程

二、过程化SQL的程序结构

■ 过程化SQL对SQL的主要扩展

- *输入输出：
 - ◆ 输出：select, 输入：NA
- 程序块定义：Begin.....End
- 变量
- 流程控制
 - ◆ 顺序结构/分支结构/循环结构
- 出错处理
- 游标
- 过程：存储过程/函数、触发器

1、变量

■ MySQL支持三种类型的变量

● 局部变量

- ◆ 必须使用**DECLARE**定义：<变量名> <类型>
- ◆ 变量名使用常规定义，字母、数字、下划线
- ◆ 作用域为**Begin.....End**之间的程序块

● 用户变量

- ◆ **不需要预先定义**，变量名前须加一个“@”符号
- ◆ 作用域为当前会话（连接），所有存储过程和函数可共享用户变量

● 系统变量

- ◆ **MySQL**内部定义的变量，变量名前有“@@”符号
- ◆ 作用域为所有客户端连接，只能读取
- ◆ 一般用于在程序中判断系统当前的某个特定状态值
 - 例如：@@version
- ◆ 查看所有的系统变量：**show global variables**

1、变量

■ 例1：定义局部变量

● Begin

Declare sno, snp **INT DEFAULT 0;**

Declare name **varchar(10);**

.....

End

■ 例2：用户变量

```
1  
2 Delimiter //  
3 CREATE PROCEDURE g6Score(IN sn VARCHAR ( 50 ))  
4 BEGIN  
5     DECLARE  
6         n,n2 FLOAT;  
7     DECLARE n3 INT;  
8     SELECT avg( score ) FROM SC WHERE sno=sn INTO n;  
9     SET @sou=n;  
10 END //  
11 Delimiter;
```



The screenshot shows a MySQL command-line interface with the following content:

```
mysql80 test  
1 call g6score('s2');  
2 select @sou;
```

Below the command input, there are tabs for '信息', '结果 1', '剖析', and '状态'. The '结果 1' tab is active, showing the result of the query:

@sou
70

1、变量

■ 变量的赋值

● Set赋值 (MySQL和MS SQL Server, Oracle用 “:=”)

- ◆ Declare **status** int;
Set **status**=1; -- 局部变量须预先定义
- ◆ Set **@done**=1; -- 用户变量不需要定义

● Select Into <变量> (都支持)

- ◆ Select max(score) From SC into v1; -- 局部变量
- ◆ Select sname from student where sno='s1' Into **@name**; -- 用户变量
- ◆ Select v1 into @name;
- ◆ **SELECT max(score) , min(score) into n2, n3 FROM SC;**



SELECT max(score) into n2 , min(score) into n3 FROM SC;

2、分支控制语句

■ **IF <表达式> THEN**

<语句>

ELSEIF <表达式> THEN

<语句>

.....

ELSE

<语句>

END IF;

```
IF x=5 THEN
```

```
    SET x=5;
```

```
END IF;
```

注意分号！

2、分支控制语句

■ CASE *case_value*

WHEN *when_value* THEN *statement_list*

WHEN *when_value* THEN *statement_list*

...

ELSE *statement_list*

END CASE;

注意分号!

```
1 DELIMITER |
2 CREATE PROCEDURE p(IN e INT)
3 BEGIN
4     DECLARE v INT DEFAULT 1;
5     SELECT level INTO v FROM EMP WHERE eno=e;
6     CASE v
7         WHEN 2 THEN SET v=1;
8         WHEN 3 THEN SET v=2;
9         WHEN 1 THEN SET v=3;
10        ELSE
11            SET v=(v-2) mod 3;
12    END CASE;
13 END |
14 DELIMITER ;
```

■ CASE

WHEN *search_condition* THEN *statement_list*

WHEN *search_condition* THEN *statement_list*

...

ELSE *statement_list*

END CASE;

3、循环语句

■ MySQL

- WHILE循环
- REPEAT循环
- LOOP循环

■ Oracle

- WHILE循环
- FOR循环
- LOOP循环

■ MS SQL Server

- WHILE循环

(1) WHILE循环

■ **While** <循环控制条件> **Do**
 <语句>

.....

End While;

注意分号和
DO!

```
1  -- 计算1到i的偶数和
2  DELIMITER //
3  CREATE PROCEDURE even_sum(IN i INT, OUT sum INT)
4  BEGIN
5      DECLARE j INT DEFAULT 1;
6      SET sum=0;
7      WHILE j<=i DO
8          IF j%2=0 THEN
9              SET sum=sum+j;
10         END IF;
11         SET j=j+1;
12     END WHILE;
13 END //
14 DELIMITER ;
```

■ 对比Oracle

While <表达式> **Loop**
 <语句>
End Loop;

■ 对比MS SQL Server

While <表达式>
Begin
 <语句>
End



The screenshot shows a MySQL command window with the following content:

```
mysql80 test
1 call even_sum(100,@sum);
2 select @sum;
```

The output shows the result of the query:

信息	结果 1	剖析	状态
@sum	2550		

(2) REPEAT循环

Repeat

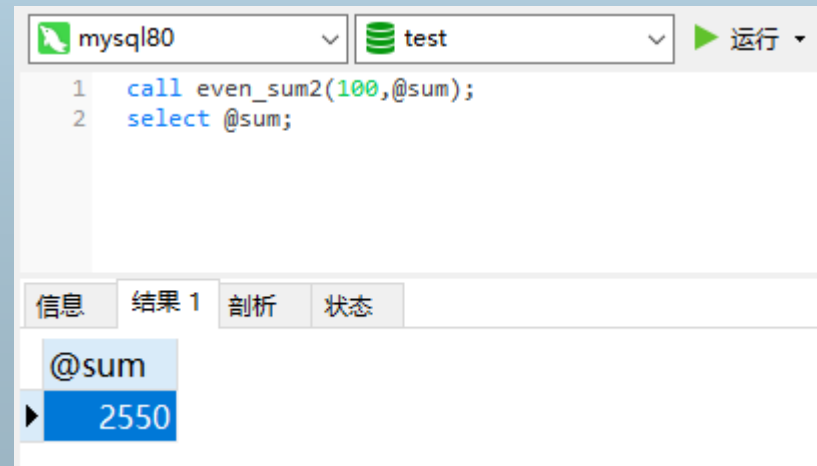
<语句>

Until <循环控制条件>

End Repeat;

注意Until后面没有分号!

```
1  -- 计算1到i的偶数和
2  DELIMITER //
3  CREATE PROCEDURE even_sum2(IN i INT, OUT sum INT)
4  BEGIN
5      DECLARE j INT DEFAULT 1;
6      SET sum=0;
7      REPEAT
8          IF j%2=0 THEN
9              SET sum=sum+j;
10         END IF;
11         SET j=j+1;
12         UNTIL j>i
13     END REPEAT;
14 END //
15 DELIMITER ;
```



The screenshot shows a MySQL command-line interface with the following content:

```
mysql80 test 运行
1 call even_sum2(100,@sum);
2 select @sum;
```

The output shows the value of the variable @sum:

信息	结果 1	剖析	状态
	@sum		
	2550		

(3) LOOP循环

- 无内部控制结构的循环结构，循环执行其中的<语句>
- 必须在循环体中显式地结束循环
- 使用**Leave**语句退出循环
- **label: Loop**
 <语句>
 IF <循环控制条件> **THEN**
 Leave label;
 END IF;
 End Loop label;
- 对比**Oracle**

Loop
 <语句>
 Exit When <循环控制条件>
 End Loop

```
1  -- 计算1到i的偶数和
2  DELIMITER //
3  CREATE PROCEDURE even_sum3(IN i INT, OUT sum INT)
4  BEGIN
5      DECLARE j INT DEFAULT 1;
6      SET sum=0;
7      iter: LOOP
8          IF j%2=0 THEN
9              SET sum=sum+j;
10         END IF;
11         SET j=j+1;
12         IF j>i THEN
13             LEAVE iter;
14         END IF;
15     END LOOP iter;
16 END //
17 DELIMITER ;
```

The screenshot shows a MySQL command-line interface with the following content:

```
mysql80 test 运行
1 call even_sum3(100,@sum);
2 select @sum;
```

The output shows the variable @sum with the value 2550.

信息	结果 1	剖析	状态
	@sum		
	2550		

三、处理异常

■ 存储过程内部执行时出错怎么办？

- 需要使用错误陷阱，捕捉程序运行中出现的错误或意外情况，并加以处理

■ 基本方法

- **Declare** <处理方式> **Handler For** <异常类型> <sql>

- ◆ <处理方式>

- **Continue**: 继续执行下一条语句
- **Exit**: 直接退出（很少用）
- **Undo**: 回退（目前不支持）

- ◆ <异常类型>

- **SQLSTATE**值
- **MySQL error code**
- **SQLWARNING, NOT FOUND**或**SQLException**, 是**SQLSTATE**值简写
- 与**MySQL**错误代码或**SQLSTATE**值相关联的命名条件。

- ◆ <sql>

- 当处理方式为**Continue**时执行的**sql**语句

1、异常类型

■ SQLSTATE

- 5个字符，正常执行时返回00开头的State
- 01开头SQLSTATE——SQLWARNING
- 02开头SQLSTATE——NOT FOUND，表示游标或SELECT语句没有返回值
- 其它的SQLSTATE——SQLEXCEPTION

■ MySQL Error Code: 4位数字

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02' SET @info='NO_SUCH_TABLE';  
  
DECLARE CONTINUE HANDLER FOR 1146 SET @info='NO_SUCH_TABLE';  
  
DECLARE CONTINUE HANDLER FOR SQLWARNING SET @status=1;  
  
DECLARE CONTINUE HANDLER FOR NOT FOUND SET @status=1;  
  
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET @status=1;
```

2、NOT FOUND例子

- 根据学号查询学生的年龄，如果学生不存在则会触发NOT FOUND异常。在程序中捕捉并返回

```
1  -- 返回给定学生的年龄
2  DELIMITER //
3  CREATE PROCEDURE error2 ( IN sn VARCHAR(50) , OUT c INT, OUT state INT)
4  BEGIN
5      DECLARE s INT DEFAULT 0;
6      DECLARE CONTINUE HANDLER FOR NOT FOUND SET s = 1;
7      SELECT age FROM student WHERE sno=sn INTO c;
8      IF s = 1 THEN
9          SET state = 1;
10     ELSE
11         SET state = 0;
12     END IF;
13 END //
14 DELIMITER;
```

学生存在
， state
返回0



mysql80 test

```
1 call error2('s1',@age,@state);
2 select @state,@age;
```

信息	结果 1	剖析	状态
	@state	@age	
	0	21	

学生不存在
， state
返回1



mysql80 test

```
1 call error2('s9',@age,@state);
2 select @state,@age;
```

信息	结果 1	剖析	状态
	@state	@age	
	1	(Null)	

3、SQLEXCEPTION例子

- 当插入记录时出现问题时（例如重复主键）返回错误码，并且取消操作

```
1
2 DELIMITER //
3 CREATE PROCEDURE error1 ( OUT state INT )
4 BEGIN
5     DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
6         SET @STATUS = 1;
7     START TRANSACTION;
8     INSERT INTO student VALUES( 's6', 'f', 19 );
9     INSERT INTO student VALUES( 's7', 'f', 19 );
10    IF @STATUS = 1 THEN
11        SET state = 1;
12        ROLLBACK;
13    ELSE
14        COMMIT;
15    END IF;
16 END //
17 DELIMITER;
```


4、一般的错误处理框架

```
1  -- 一般的错误处理框架，state用于返回错误码
2  DELIMITER //
3  CREATE PROCEDURE error_handler ( IN sn VARCHAR(50) , OUT state INT)
4  BEGIN
5      DECLARE s INT DEFAULT 0;
6      DECLARE CONTINUE HANDLER FOR 1146 SET s = 1; -- 特定错误的捕捉
7      DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02' SET s=2; -- 特定错误的捕捉
8      DECLARE CONTINUE HANDLER FOR NOT FOUND SET s = 3; -- 如果有查询语句，空集错误的捕捉
9      DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET s = 4; -- 其余未知错误的捕捉
10     -- 如果有数据更新，则开始事务
11     START TRANSACTION;
12     -- 执行DML语句
13     SELECT age FROM student WHERE sno=sn INTO c;
14     INSERT INTO ...;
15     UPDATE student ...;
16     IF <自定义异常> THEN -- 可以自定义异常，比如余额不足1000
17         SET s=5;
18     END IF;
19
20     -- 下面开始集中处理错误
21     IF s=0 THEN
22         SET state=0;
23         COMMIT;
24     ELSE
25         CASE s -- 根据s值进行错误处理，例如设置state值
26             WHEN 1 THEN
27             WHEN 2 THEN
28             WHEN 3 THEN
29             WHEN 4 THEN
30             ELSE
31
32         END CASE;
33         ROLLBACK; -- 取消所有操作
34     END IF;
35 END //
36 DELIMITER ;
```