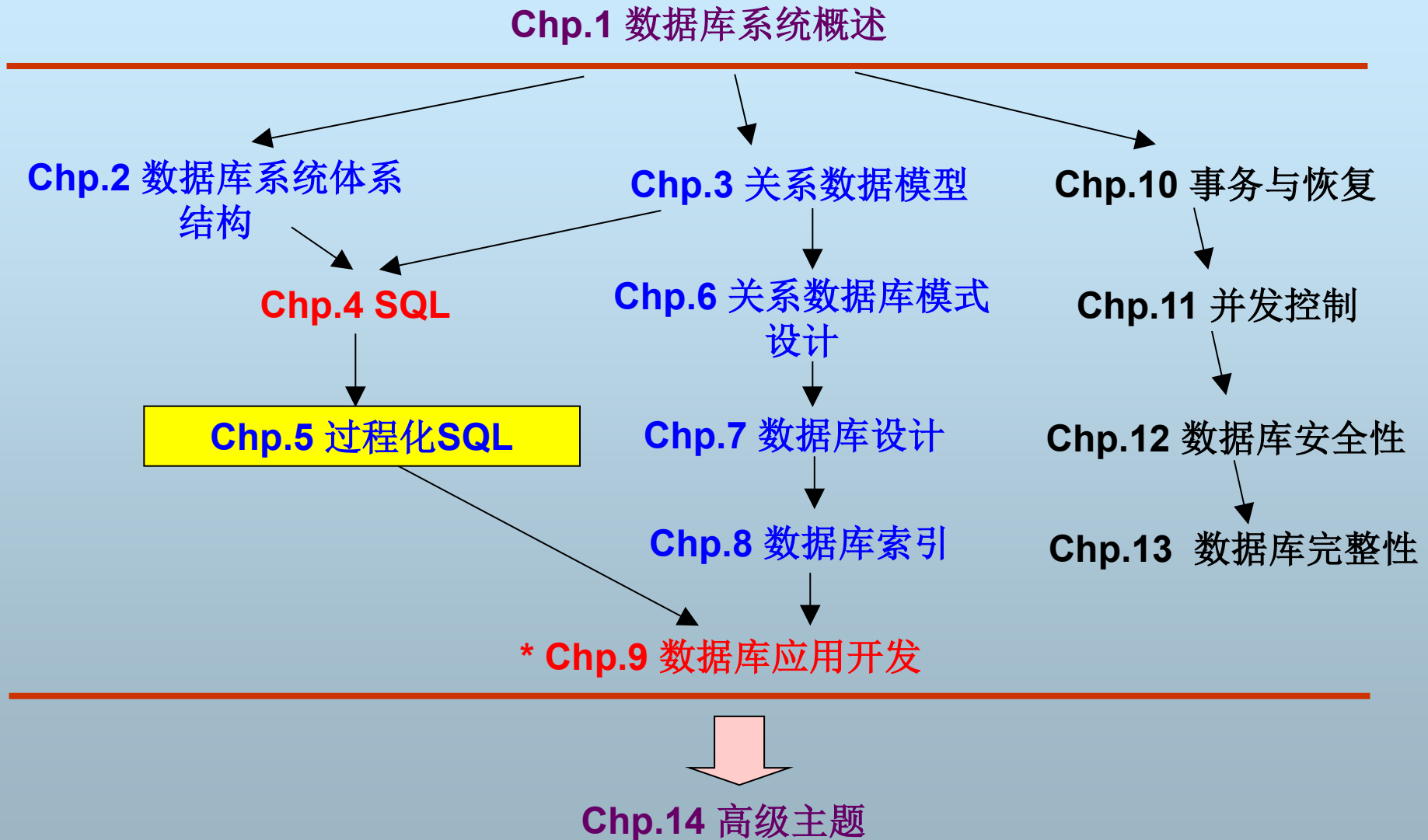


第5章 过程化SQL

课程知识结构



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）

五、游标

- 游标概念
- 游标操作
- 操纵游标的当前行

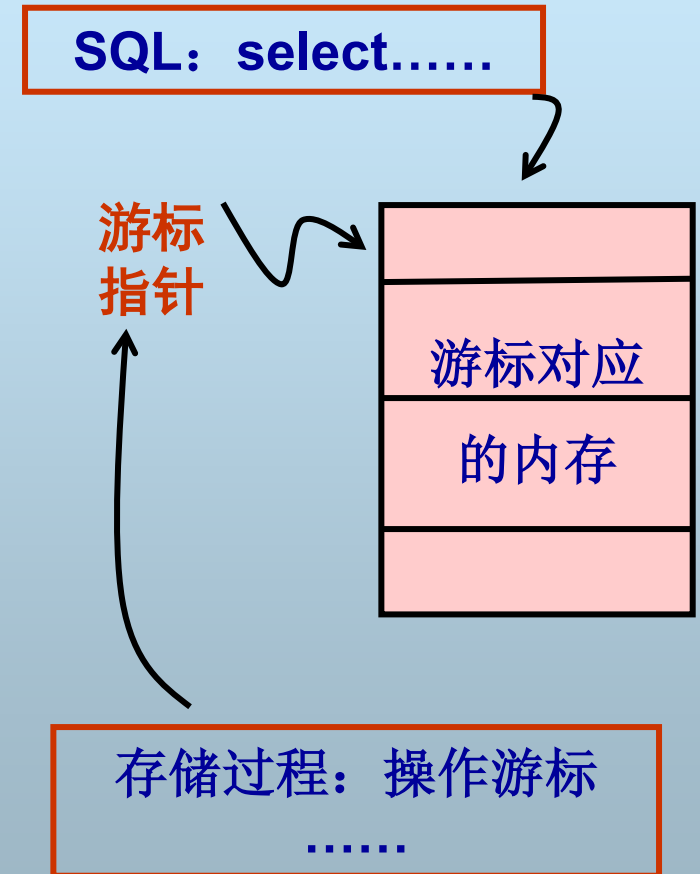
1、游标概念

■ 动机

- 过程化SQL程序中的变量每次只能存储单个记录；而SQL是描述性语言，每次可能返回多行记录。问题：
- 过程化SQL如何支持多行记录的操作？

■ 解决方法：游标


- 游标是客户机或数据库服务器上开辟的一块内存，用于存放SQL返回的结果
- 游标可以协调过程化SQL与SQL之间的数据处理矛盾
- 过程化SQL程序（存储过程/函数）可以通过游标来存取SQL返回的结果



2、游标操作

- 声明一个游标
- 打开游标
- 读取游标中的记录
- 关闭游标

一般的操
作顺序



(1) 声明游标

■ Declare

Cursor <名称> For <Select语句>

● 对比: Oracle PL/SQL

◆ **Declare Cursor <名称> IS <Select语句>**

■ 声明中的SQL语句在声明时并不执行，只是给出了游标对应的数据定义

--声明一个游标，用于存放所有学生记录

DECLARE

Cursor cs_stu For select * from student;

(2) 打开游标

■ Open <游标名>

- 打开游标时，**SELECT**语句被执行，其结果放入了游标中

```
--声明一个游标，用于存放所有学生记录
BEGIN
    DECLARE
        Cursor cs_stu For select * from student;
    Open cs_stu;
    ...
END;
```


(3) 读取游标中的记录

■ Fetch <游标名> Into <变量表>

- 打开游标后，游标指向了第一条记录
- **Fetch**后指向下一条记录
- 若要读取游标中的数据，一般需使用一个循环

--返回所有CS学生记录

BEGIN

Declare state INT default 0;

Declare s1, s2 VARCHAR(50);

Declare Cursor cs_stu For select sno, sname from student where dept='cs';

Declare continue Handler for NOT FOUND set state=1;

Open cs_stu;

Repeat

Fetch cs_stu Into s1,s2;

Until state=1

End Repeat;

.....

END

(4) 关闭游标

■ Close <游标名>

--返回所有CS学生记录

BEGIN

Declare state INT default 0;

Declare s1, s2 VARCHAR(50);

Declare **Cursor cs_stu** For select sno, sname from student where dept='cs';

Declare continue Handler for NOT FOUND set state=1;

Open cs_stu;

Repeat

Fetch cs_stu Into s1,s2;

Until state=1

End Repeat;

Close cs_stu;

END

1

2

3

4

(5) 游标示例

```
mysql80 test 运行 停止 解释
1  -- 计算给定学生的不及格学分
2  Delimiter //
3  DROP PROCEDURE IF EXISTS cursor_test;
4  CREATE PROCEDURE cursor_test ( IN sn VARCHAR (50), OUT total INT )
5  BEGIN
6  DECLARE state INT DEFAULT 0;
7  DECLARE sn1 VARCHAR(50);
8  DECLARE cred INT;
9  DECLARE
10 ct CURSOR FOR
11 (SELECT sc.sno, credit FROM sc, course WHERE course.cno=sc.cno AND score<60);
12 DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1;
13 SET total = 0;
14 OPEN ct;
15 REPEAT
16     FETCH ct INTO sn1, cred;
17     IF state = 0 THEN
18         IF sn1=sn THEN
19             SET total = total + cred;
20         END IF;
21     END IF;
22     UNTIL state = 1
23 END REPEAT;
24 CLOSE ct;
25 END //
26 Delimiter;
```

```
mysql80 test
1 set @sno='s5';
2 call cursor_test(@sno,@total);
3 select @sno,@total;
```

信息	结果 1	剖析	状态
@sno	@total		
▶ s5	5		

六、存储过程和函数

■ 存储过程

- 存储在数据库中的过程，可以随时运行，也可以被SQL或外部程序调用

■ 函数

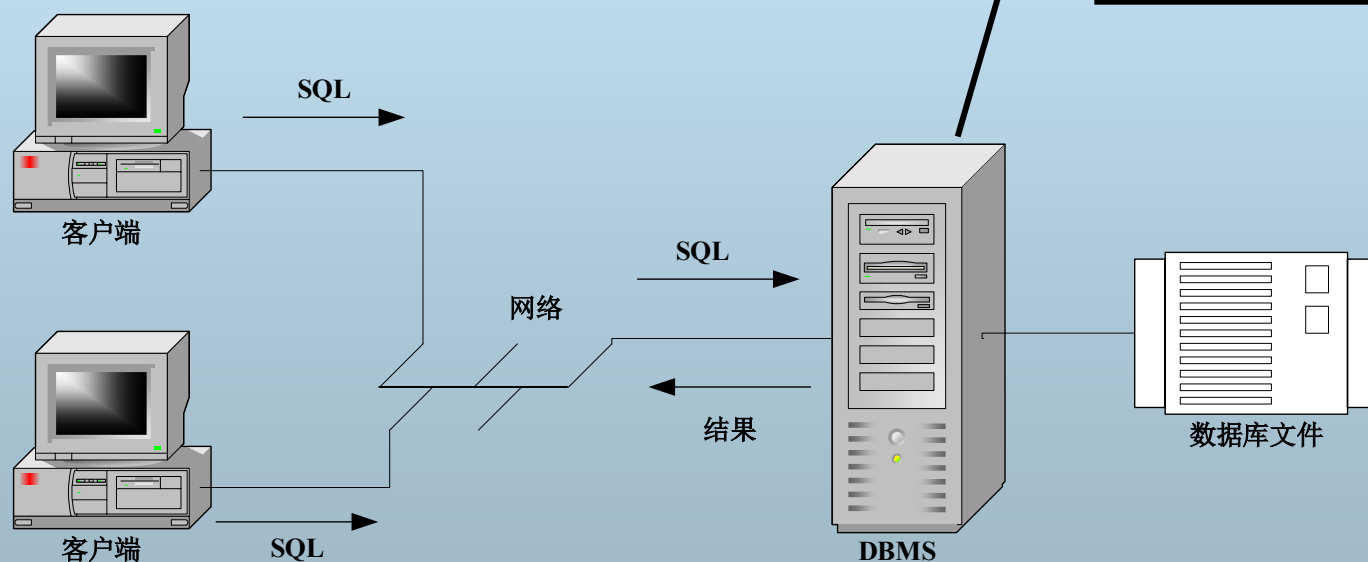
- 具有返回值的存储过程

1、存储过程概念

Client/Server计算模式：

客户端将业务处理任务交给服务器上的存储过程完成

存储过程位于数据库服务器端



2、存储过程定义

■ **Create Procedure <名称>(参数表)**

BEGIN

<变量定义>

过程化SQL代码

<异常处理>

END;

3、参数定义

- **[IN | OUT | INOUT] 参数名 数据类型**
 - 例 **IN name varchar(50), OUT result int**
- **IN参数**
 - 输入参数，在程序中不能修改
 - 如果不指定参数类型，默认为**IN**
- **OUT参数**
 - 输出参数，在程序中只能对其赋值
- **INOUT**
 - 既可作为**IN**参数使用，也可作为**OUT**参数使用

4、查看存储过程

■ Show Create Procedure <存储过程名>

```
mysql> use test;
Database changed
mysql> show create procedure cursor_test;
```

Procedure	sql_mode	Create Procedure	character_set_client	collation_client
cursor_test	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION	CREATE DEFINER='root'@'localhost' PROCEDURE `cursor_test`(IN sn VARCHAR (50), OUT total INT) BEGIN DECLARE state INT DEFAULT 0; DECLARE sn1 VARCHAR(50); DECLARE cred INT; DECLARE ct CURSOR FOR (SELECT sc.sno, credit FROM sc, course WHERE course.cno = sc.cno AND score < 60); DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1; SET total = 0; OPEN ct; REPEAT FETCH ct INTO sn1, cred; IF state = 0 THEN IF sn1=sn THEN SET total = total + cred; END IF; END IF; UNTIL state = 1 END REPEAT; CLOSE ct; END	utf8mb4	utf8mb4_0900_ai_ci

```
1 row in set (0.00 sec)

mysql>
```


5、删除存储过程

- **Drop Procedure** <存储过程名>

6、函数

- 具有返回值的存储过程
- **Create Function** <名称>(参数表)

RETURNS <类型>

[**Deterministic** | **Reads SQL data** | **No SQL** | **MODIFIES SQL DATA**]

BEGIN

<变量定义>

过程化SQL代码

RETURN <变量>;

<异常处理>

END;

函数类型:

Deterministic: 同一参数值的输出结果确定

Reads SQL data: 函数内部读数据库

No SQL: 函数内部不读数据库

MODIFIES SQL DATA: 表示函数会更新数据库

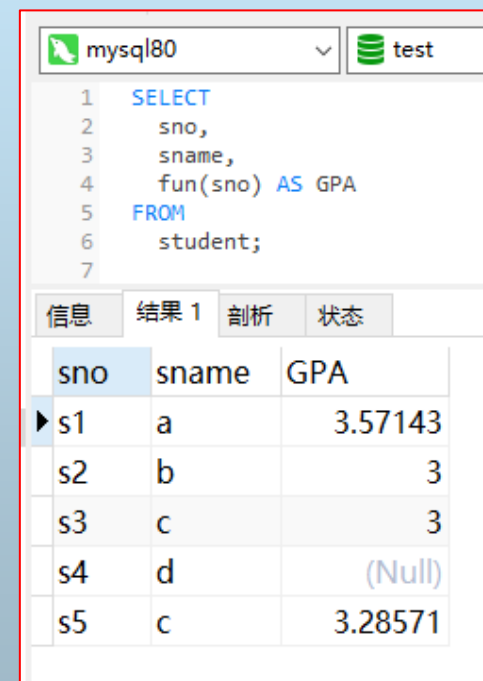
若设置了**binlog**信任函数创建者则无需指定类型

SET GLOBAL log_bin_trust_function_creators = TRUE;

7、函数例子：计算GPA

```
1  -- 计算给定学生的GPA
2  Delimiter //
3  DROP FUNCTION IF EXISTS fun;
4  CREATE FUNCTION fun(sn VARCHAR(50))
5  RETURNS FLOAT
6  READS SQL DATA
7  BEGIN
8      DECLARE state INT DEFAULT 0; -- cursor结束标记
9      DECLARE grade,cred,total_c,total_g FLOAT DEFAULT 0;
10     DECLARE sn1 VARCHAR(50);
11     DECLARE c_count INT;
12     DECLARE t, gpa FLOAT DEFAULT 0;
13     DECLARE
14         ct CURSOR FOR
15         (SELECT score,credit FROM sc,course c WHERE sc.cno=c.cno AND sno=sn AND score IS NOT NULL);
16     DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1;
17     OPEN ct;
18     REPEAT
19         FETCH ct INTO grade,cred; -- 每一门课程的成绩和学分
20         IF state = 0 THEN
21             CASE
22                 WHEN grade>=95 THEN SET t=4.3;
23                 WHEN grade>=90 AND grade<95 THEN SET t=4.0;
24                 WHEN grade>=85 AND grade<90 THEN SET t=3.7;
25                 WHEN grade>=82 AND grade<85 THEN SET t=3.3;
26                 ELSE SET t=3;
27             END CASE;
28             SET total_g=total_g + t*cred; -- 计算总的学分*绩点
29             SET total_c=total_c + cred; -- 计算总的学分
30         END IF;
31         UNTIL state = 1
32     END REPEAT;
33     CLOSE ct;
34     SET gpa=total_g/total_c;
35     RETURN gpa;
36 END //
37 Delimiter;
```

$$GPA = \frac{\sum \text{课程学分} * \text{课程学分绩点}}{\sum \text{课程学分}}$$



The screenshot shows a MySQL 8.0 test window. The SQL query is: `SELECT sno, sname, fun(sno) AS GPA FROM student;`. The results are displayed in a table with columns `sno`, `sname`, and `GPA`. The data rows are: `s1` (a, 3.57143), `s2` (b, 3), `s3` (c, 3), `s4` (d, (Null)), and `s5` (c, 3.28571).

sno	sname	GPA
s1	a	3.57143
s2	b	3
s3	c	3
s4	d	(Null)
s5	c	3.28571

总结：存储过程/函数的主要作用

1. 增强了SQL的功能和灵活性，可以完成复杂的判断和运算。
2. 可增强数据库的安全性。通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全。
3. 可增强数据库的完整性。
4. 在运行存储过程前，数据库已对其进行语法和句法分析，并给出了优化执行方案。由于执行SQL语句的大部分工作已经完成，所以存储过程能以较快的速度执行。
5. 可以降低网络的通信量。
6. 使体现企业规则的运算程序放入数据库服务器中，以便集中控制。

缺点：编写、调试和使用较复杂

七、触发器 (Trigger)

- 触发器的概念
- 触发器的种类
- 触发器的创建
- **old**和**new**系统变量

1、触发器的概念

- 与特定表关联的存储过程。当在该表上执行**DML**操作时，可以自动触发该存储过程执行相应的操作
 - 触发操作：**Update、Insert、Delete**
 - 通过触发器可以定制数据库对应用程序的反应
 - 一个触发器只能属于一个表，一个表可有多个触发器

2、触发器概念示例

- **Student (sno, sname, age, status)**
- **Sc(sno, cno, score)**
- 规定当学生有3门课不及格时，将该学生的**status**标记为‘不合格’
- 通过**SC**上的触发器实现： **当在SC中插入或更新记录时，自动检查是否有学生满足不合格条件**

Sno	Sname	age	status
01	aaa	22	合格
02	bbb	21	合格

Sno	Cno	Score
01	c1	55
01	c2	50
02	c1	80
01	c3	55

插入该记录后**01**学生的
status自动改为‘不合格’

3、触发器的种类

按执行先后

- **先触发器（Before Trigger）**：在DML语句执行之前触发
- **后触发器（After Trigger）**：在DML语句执行之后触发
- **替代触发器（Instead Trigger）**：用触发器代码替代DML执行

按执行方式

- **行级触发器**：对由触发的DML语句所导致变更的每一行触发一次（一个DML语句可能触发多次）
- **语句级触发器**：一个DML语句只触发一次

特殊的触发器

- **DDL触发器**：当执行DDL语句时触发
- **DB事件触发器**：当系统STARTUP、SHUTDOWN、LOGON、LOGOFF等事件发生时触发

3、触发器的种类

	MySQL	Oracle	MS SQL Server
先触发器	√	√	x
后触发器	√	√	√
替代触发器	x	√	√
行级触发器	√	√	x
语句级触发器	x	√	√
DDL触发器	x	√	√
DB事件触发器	x	√	√（仅支持LOGON）

4、触发器的创建

■ Create Trigger <名称>

[Before | After | Delete | Insert | Update]

ON <表名>

For Each Row

BEGIN

<过程化SQL程序>

END;

定义触发事件

先触发器还是后触发器

定义为行级触发器

● 注意：

- ◆ 没有参数。因为触发器是自动执行的，不能向它传参数
- ◆ 一个触发器只能定义一个触发事件。如果要触发多个事件，则只能定义多个触发器
【Oracle允许一个触发器触发多个事件】

5、系统变量old和new

- 对于行级触发器，系统变量old和new存储每一行的更新前值（old）和更新后值（new）
- 可以在触发器程序中需要时访问它们

操作 变量	Insert	Update	Delete
old的值	空	原记录	删除的记录
new的值	新记录	新记录	空

6、触发器例子：自动更新学生状态

```
mysql80 test 运行 停止
1 -- 当插入新的选课记录时更新学生的status
2 Delimiter //
3 DROP TRIGGER IF EXISTS updateStatus;
4 CREATE TRIGGER updateStatus AFTER INSERT ON sc FOR EACH ROW
5 BEGIN
6     DECLARE c_count INT;
7     SELECT count(cno) FROM sc
8         WHERE sno = new.sno AND score < 60 INTO c_count;
9     IF c_count >= 3 THEN
10         UPDATE student SET STATUS = '不合格'
11         WHERE sno = new.sno;
12     END IF;
13 END //
14 Delimiter;
```

Student

sno	sname	age	status
s1	a	21	合格
s2	b	22	合格
s3	c	23	合格
s4	d	24	合格
s5	c	22	合格

SC

sno	cno	score
s1	c1	90
s1	c2	90
s1	c3	80
s2	c1	70
s2	c2	80
s2	c3	60
s3	c3	60
s5	c1	50
s5	c3	40

当s5插入了一条新的不及格记录后student表中的status已自动更新

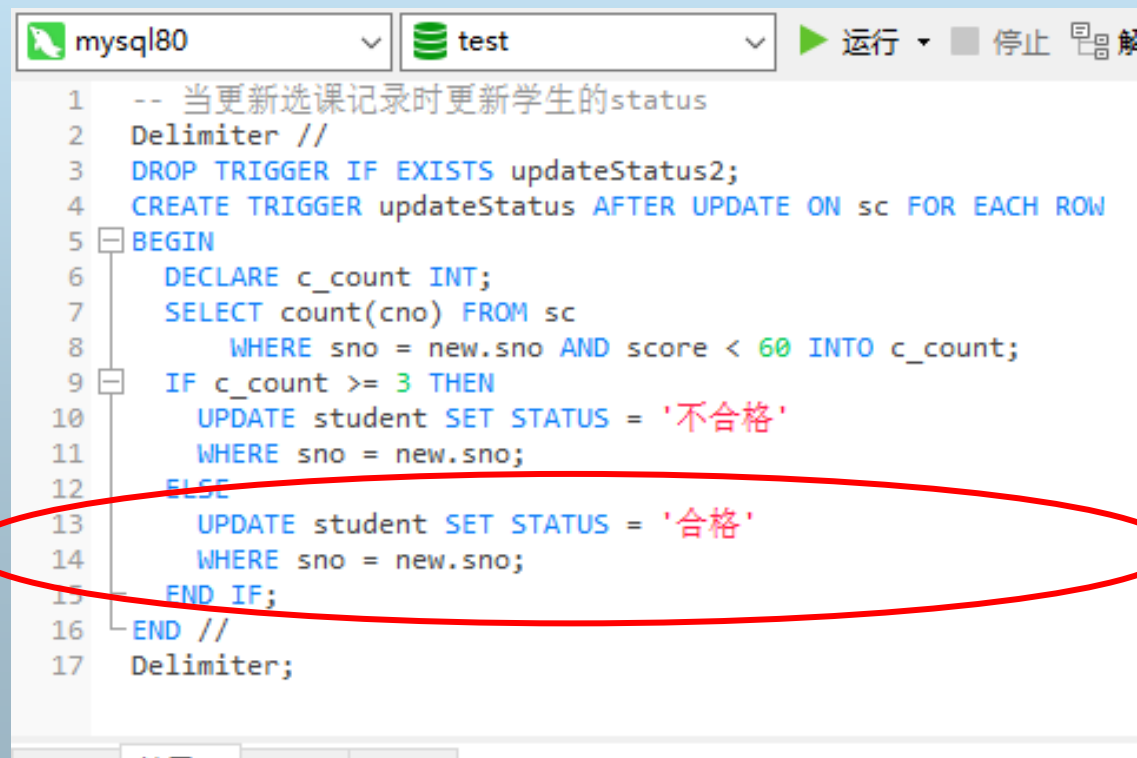


```
mysql80 test 运行
1 INSERT INTO sc VALUES ( 's5', 'c2', 55 );
2 SELECT * FROM student;
```

信息	结果 1	剖析	状态	
	sno	sname	age	status
	s1	a	21	合格
	s2	b	22	合格
	s3	c	23	合格
	s4	d	24	合格
	s5	c	22	不合格

6、触发器例子：自动更新学生状态

- 考虑学生补考情况，增加一个**After Update**触发器
- 如果学校允许销掉不及格的选课？——**Delete**触发器



```
mysql80 test 运行 停止 帮助
1  -- 当更新选课记录时更新学生的status
2 Delimiter //
3  DROP TRIGGER IF EXISTS updateStatus2;
4  CREATE TRIGGER updateStatus AFTER UPDATE ON sc FOR EACH ROW
5  BEGIN
6      DECLARE c_count INT;
7      SELECT count(cno) FROM sc
8          WHERE sno = new.sno AND score < 60 INTO c_count;
9      IF c_count >= 3 THEN
10         UPDATE student SET STATUS = '不合格'
11         WHERE sno = new.sno;
12     ELSE
13         UPDATE student SET STATUS = '合格'
14         WHERE sno = new.sno;
15     END IF;
16 END //
17 Delimiter;
```

7、查看触发器

- **Show triggers:** 显示当前数据库中的所有触发器
- **Show create trigger <触发器名称>:** 显示特定的触发器

```
mysql> show triggers;
+-----+-----+-----+-----+-----+-----+
| Trigger | Event | Table | Statement | Definer | character_set_client |
+-----+-----+-----+-----+-----+-----+
| updateStatus | INSERT | sc | Begin
    declare c_count int;
    select count(cno) from sc where sno=new.sno and score<60 into c_count;
    if c_count>=3 then
        update student set status='不合格' where sno=new.sno;
    end if;
end | AFTER | 2020-03-22 21:15:15.28 | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION | root@localhost | utf8mb4
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql> show create trigger updateStatus;
+-----+-----+-----+-----+-----+-----+
| Trigger | sql_mode | SQL Original Statement |
+-----+-----+-----+-----+-----+-----+
| updateStatus | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION | CREATE DEFINER='root'@'localhost' TRIGGER updateStatus AFTER INSERT ON sc FOR EACH ROW
    declare c_count int;
    select count(cno) from sc where sno=new.sno and score<60 into c_count;
    if c_count>=3 then
        update student set status='不合格' where sno=new.sno;
    end if;
end | utf8mb4 | utf8mb4_0900_ai_ci | utf8mb4_0900_ai_ci | 2020-03-22 21:15:15.28
+-----+-----+-----+-----+-----+-----+
1 row in set (0.01 sec)
```

总结：触发器的主要作用

1. **强化约束**：触发器能够实现复杂的约束。
2. **跟踪变化**：触发器可以侦测数据库内的操作，可以用来实施审计，以及不允许数据库中未经许可的更新和变化。
3. **级联运行**：触发器可以侦测数据库内操作，并自动地级联影响整个数据库的各项内容。

缺点：影响性能；潜在的运行错误风险



谨慎使用！

本章小结

- 过程化SQL与SQL
- 过程化SQL程序要素
- 游标
- 事务编程
- 存储过程和函数
- 触发器