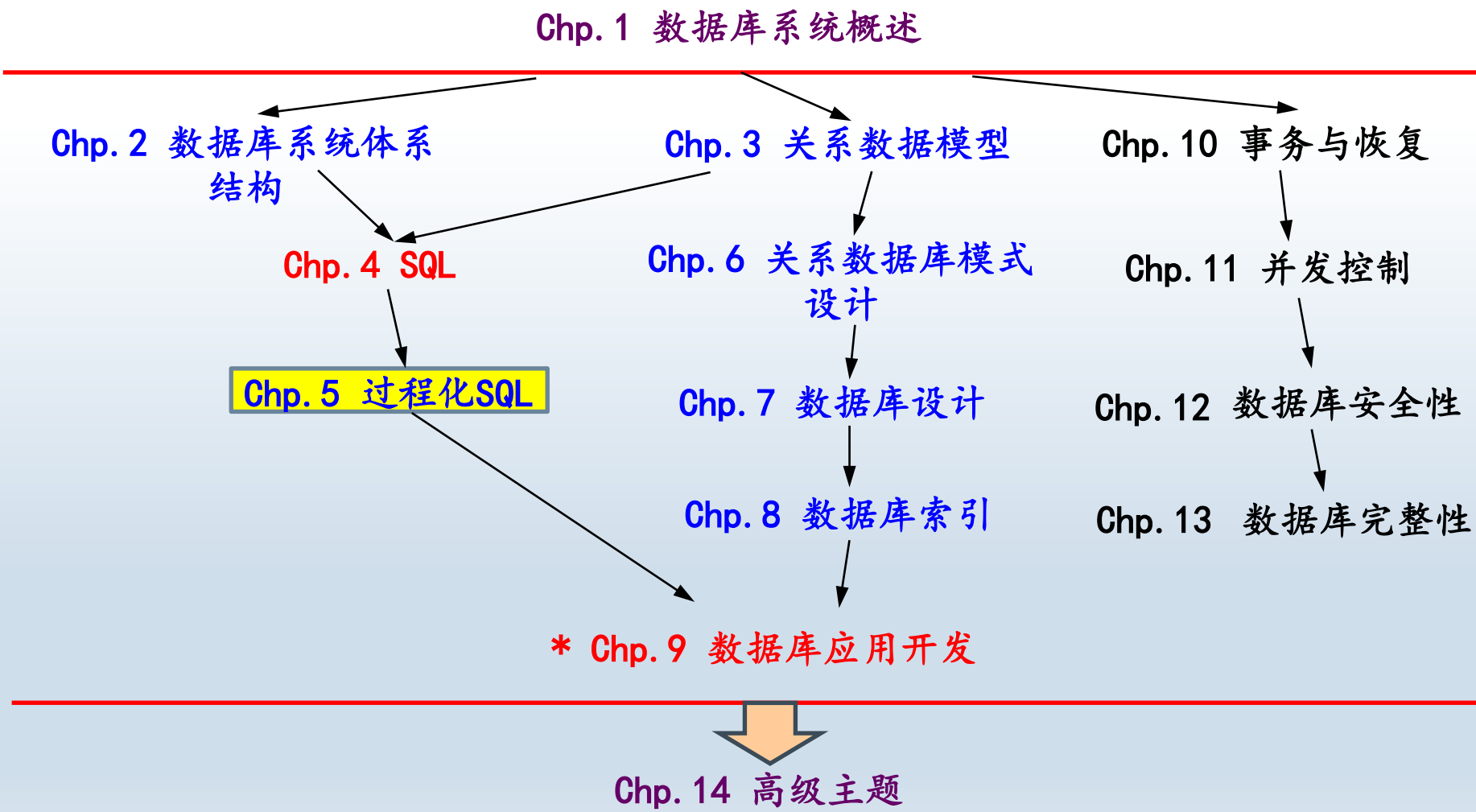
2024年春季学期

第5章 过程化SQL

课程知识结构


Chp.1 数据库系统概述
Chp.2 数据库系统体系结构
Chp.3 关系数据模型
Chp.10 事务与恢复
Chp.4 SQL
Chp.6 关系数据库模式设计
Chp.11 并发控制
Chp.5 过程化SQL
Chp.7 数据库设计
Chp.12 数据库安全性
Chp.8 数据库索引
Chp.13 数据库完整性
* Chp.9 数据库应用开发
Chp.14 高级主题

本章主要内容

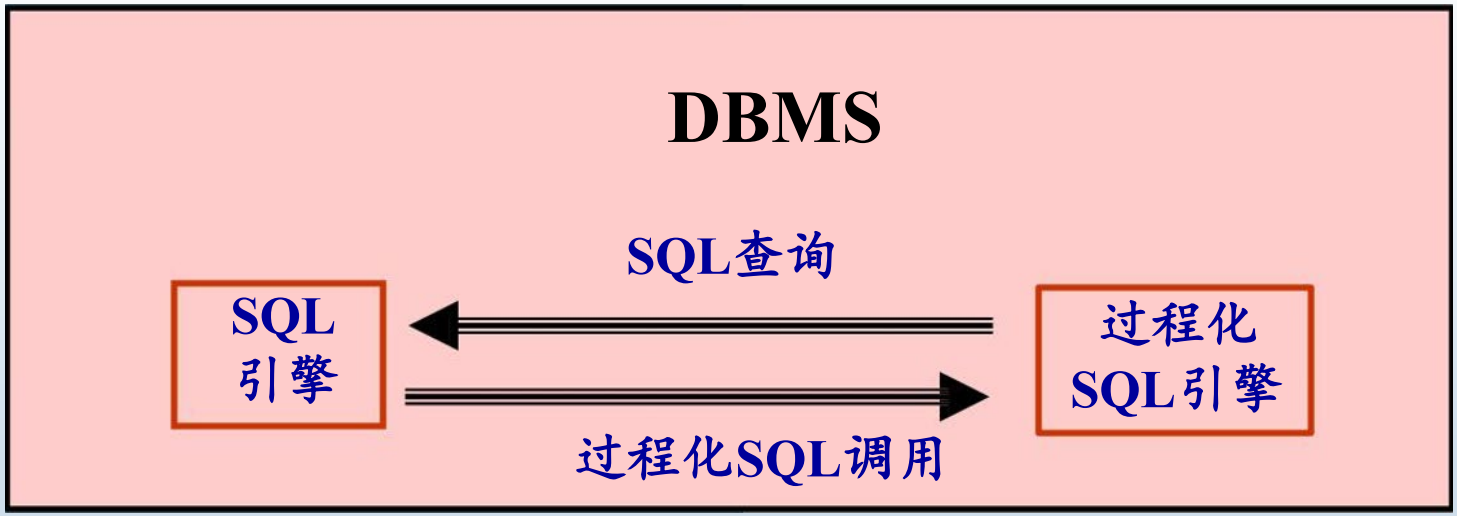
- 过程化SQL vs. SQL
- 过程化SQL编程
- 事务编程
- 游标（Cursor）
- 存储过程（Stored Procedure）
- 触发器（Trigger）

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
- 商用DBMS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

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

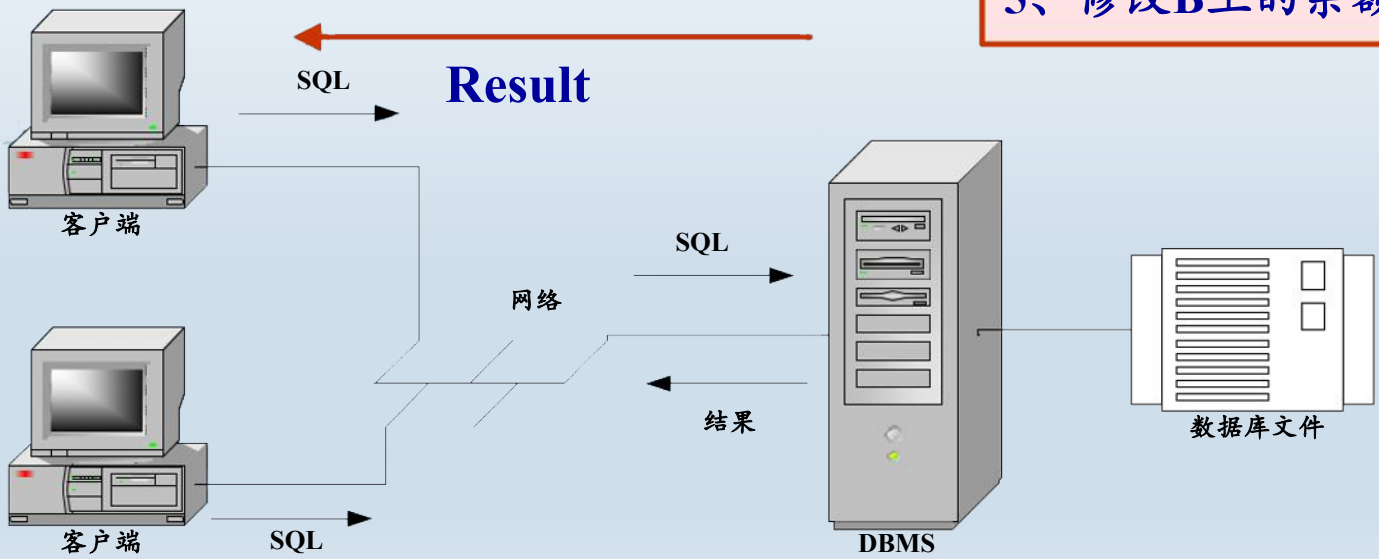
3、查询账号A上的余额

4、修改A上的余额

5、修改B上的余额

SQL

Result



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

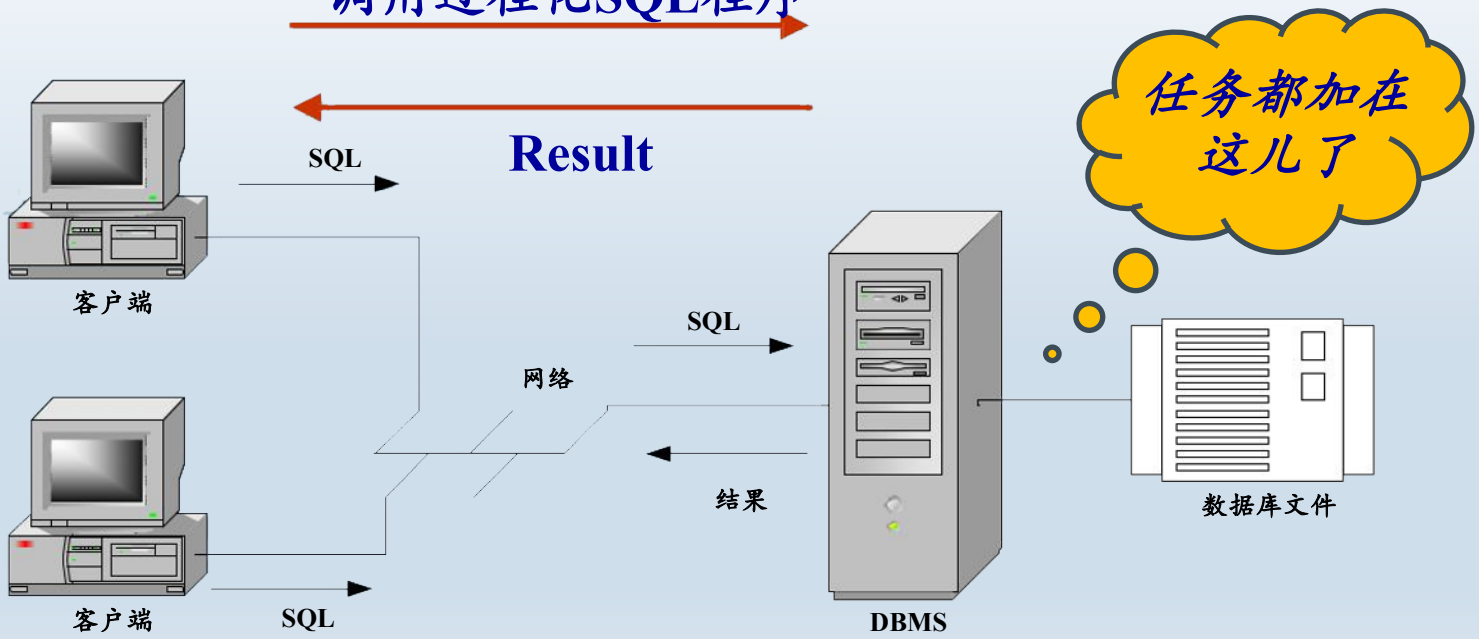
更少网络传输

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

Result



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

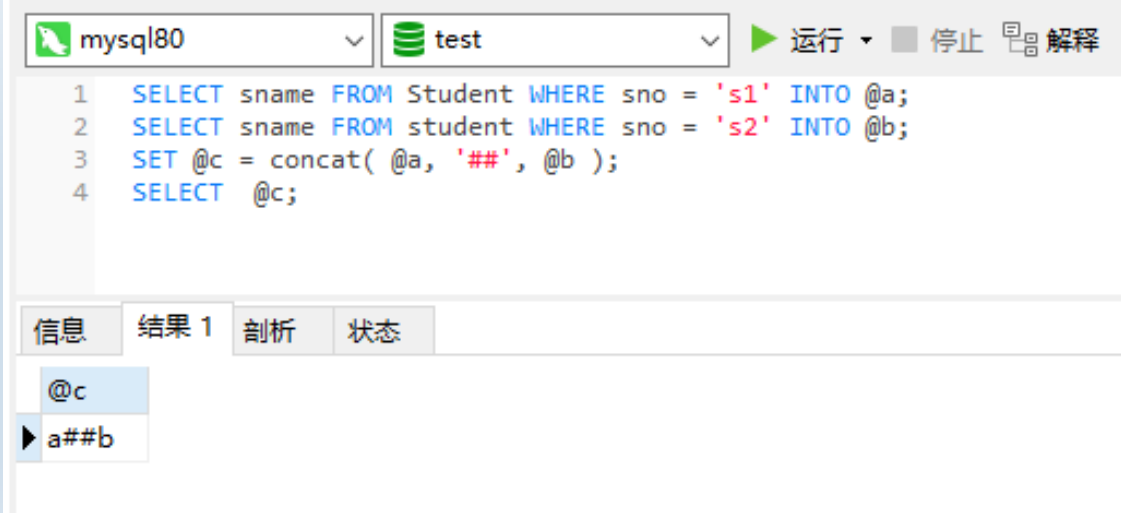
- 在程序中使用过程化SQL
  - 可以完成一些SQL不能完成的复杂计算，并且封装处理逻辑
  - 客户机计算任务少
  - 服务器计算任务加重
  - 网络传输少

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	√	√



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
  - 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
  - 会话变量
    - 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会话变量
  - 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：会话变量

```
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;
```

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； -- 会话变量不需要定义
  - 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; -- 会话变量
    - 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;

X

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 DELIMTER ;
```

- 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！

- 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
```

```
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 ;
```

mysql80 | test

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

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 ;
```

mysql80 | test | 运行

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

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>
    - Undo：回退 -----目前不支持
    - Continue：继续执行下一条语句
    - Exit：直接退出
  - <异常类型>
    - SQLSTATE值,或者是 SQLWARNING，NOT FOUND 或 SQLEXCEPTION 此类SQLSTATE值的简写
    - MySQL error code
    - 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

```
DECLARE condition_name CONDITION FOR condition_value;
```

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

学生不存在，state返回1

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 SQLEXEPTION 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 ;
```

四、事务编程

- 事务（transaction）
  - 不可分的DML操作序列
  - 例如，银行转账
    - Update A=A-100
    - Update B=B+100
- 事务的性质
  - ACID：Atomicity(原子性)，Consistency(一致性)，Isolation(隔离性)，Durability(持久性)

四、事务编程

- 当在过程化SQL中需要对多个数据进行更新，并且具有事务特性时，需要将它们做成事务进行处理，从而保证更新时的数据一致性
- 事务编程语句
  - **Start transaction：开始事务。从此往后的所有DML操作都属于一个事务**
  - **Commit：提交事务。事务所做的修改全部生效，写入持久存储介质**
  - **Rollback：回滚事务。事务所做的修改全部取消，数据库回退到事务开始之前的状态**

MySQL	Oracle	MS SQL Server	ANSI SQL
Start transaction	——	Begin transaction	Begin transaction
Commit	Commit work	Commit transaction	Commit transaction
Rollback	Rollback work	Rollback transaction	Rollback transaction

四、事务编程

id	name	balance
1	a	800
2	b	2100

转账：

```
1
2  delimiter //
3  CREATE PROCEDURE transfer(IN id_from INT, IN id_to INT, IN amount INT, OUT state INT)
4  BEGIN
5    DECLARE s INT DEFAULT 0;
6    DECLARE a INT;
7    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET s = 1;
8    START TRANSACTION;
9    SELECT count(*) FROM account WHERE id = id_from or id=id_to INTO a;
10   IF  a < 2 THEN -- 至少有一个账户不存在
11       SET s = 2;
12   END IF;
13
14   SELECT balance FROM account WHERE id = id_from INTO a;
15   IF  a < amount THEN  -- 余额不足
16       SET s = 3;
17   END IF;
18   UPDATE account SET balance = balance - amount WHERE id = id_from;
19   UPDATE account SET balance = balance + amount WHERE id = id_to;
20   IF s = 0 THEN
21       SET state = 0;
22       COMMIT;
23   ELSE
24       SET state = -1000;
25       ROLLBACK;
26   END IF;
27 END //
28 delimiter;
```

mysql80 test

```
1  call transfer(1,3,100,@state);
2  select @state;
```

信息 结果 1 剖析 状态

@state
-1000

mysql80 test

```
1  call transfer(1,2,100,@state);
2  select @state;
```

信息 结果 1 剖析 状态

@state
0

id	name	balance
1	a	900
2	b	2000

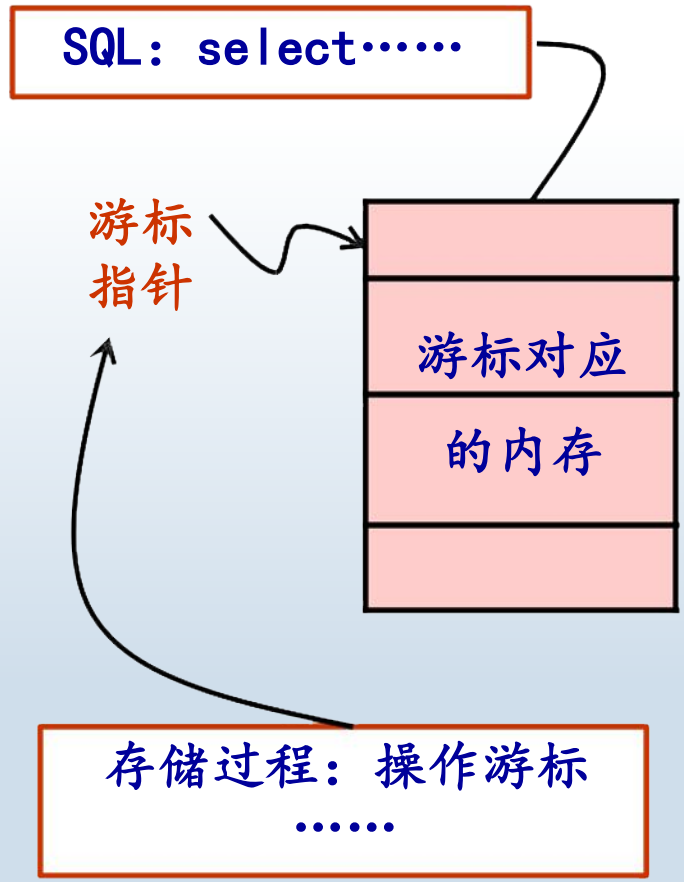
五、游标

- 游标概念
- 游标操作
- 游标属性
- 使用游标FOR循环
- 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';   1
   Declare continue Handler for NOT FOUND set state=1;
   Open cs_stu;   2
   Repeat
       Fetch cs_stu Into s1, s2;
                                                                               3
      Until state=1
   End Repeat;
   Close cs_stu;   4
END
```

（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

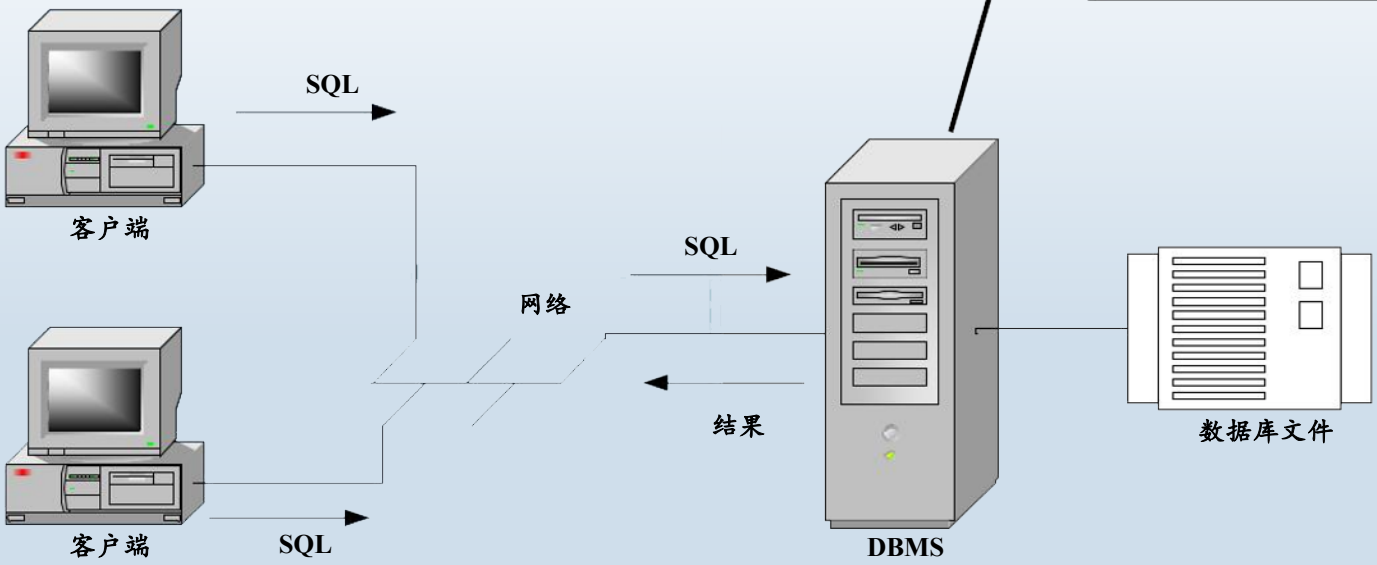
- 存储过程
  - 存储在数据库中的过程，可以随时运行，也可以被SQL或外部程序调用
- 函数
  - 具有返回值的存储过程

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 | coll
+-------------+-----------------------------------------+---------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------+----------------------+----
| 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   | 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]

BEGIN

<变量定义>

过程化SQL代码

RETURN <变量>;

<异常处理>

END;

函数类型：

Deterministic：确定性函数

Reads SQL data：函数内部读数据库

No SQL：函数内部不读数据库

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

```
-- 计算给定学生的GPA
Delimiter //
DROP FUNCTION IF EXISTS fun;
CREATE FUNCTION fun(sn VARCHAR(50))
RETURNS FLOAT
READS SQL DATA
BEGIN
  DECLARE state INT DEFAULT 0; -- cursor结束标记
  DECLARE grade,cred,total_c,total_g FLOAT DEFAULT 0;
  DECLARE sn1 VARCHAR(50);
  DECLARE c_count INT;
  DECLARE t, gpa FLOAT DEFAULT 0;
  DECLARE
    ct CURSOR FOR
    (SELECT score,credit FROM sc,course c WHERE sc.cno=c.cno AND sno=sn AND score IS NOT NULL);
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET state = 1;
  OPEN ct;
  REPEAT
    FETCH ct INTO grade,cred; -- 每一门课程的成绩和学分
    IF state = 0 THEN
      CASE
        WHEN grade>=95 THEN SET t=4.3;
        WHEN grade>=90 AND grade<95 THEN SET t=4.0;
        WHEN grade>=85 AND grade<90 THEN SET t=3.7;
        WHEN grade>=82 AND grade<85 THEN SET t=3.3;
        ELSE SET t=3;
      END CASE;
      SET total_g=total_g + t*cred; -- 计算总的学分*绩点
      SET total_c=total_c + cred; -- 计算总的学分
    END IF;
    UNTIL state = 1
  END REPEAT;
  CLOSE ct;
  SET gpa=total_g/total_c;
  RETURN gpa;
END //
Delimiter;
```

mysql80 | test

```
SELECT
  sno,
  sname,
  fun(sno) AS GPA
FROM
  student;
```

信息 | 结果 1 | 剖析 | 状态

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

- 增强了SQL语言的功能和灵活性，可以完成复杂的判断和运算。
- 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。
- 可增强数据库的完整性。
- 在运行存储过程前，数据库已对其进行了语法和句法分析，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。
- 可以降低网络的通信量。
- 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文件的反应
  - 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]

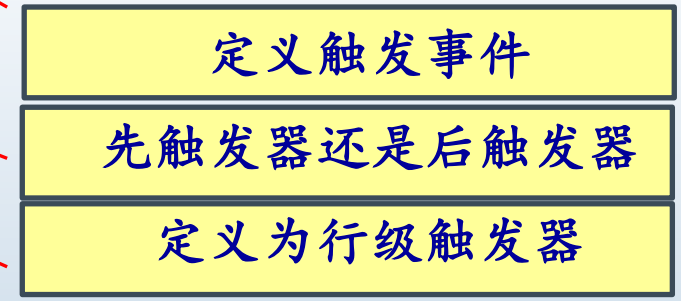
  ON <表 名>

  For Each Row

  BEGIN

  <过程化SQL程序>

  END;



- 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

```
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
 | Created                | sql_mode                                             | Definer        | charact
+--------------+--------+-------+---------------------------------------------------------------------
-+------------------------+------------------------------------------------------+----------------+--------
| 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@localho
+--------------+--------+-------+---------------------------------------------------------------------
-+------------------------+------------------------------------------------------+----------------+--------
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` T
        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

- 强化约束：触发器能够实现复杂的约束。
- 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。
- 级联运行：触发器可以侦测数据库内的操作，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

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