

一、触发器

与表有关的数据对象，在满足某种条件的时候，被动执行的SQL语句。

1、触发器的特性

1. 有begin、end的结构体（多条sql语句）
2. 需要指定触发的条件：INSERT，UPDATE，DELETE
3. 有指定的触发时间：BEFORE，AFTER

2、触发器的创建

- 单条业务逻辑的触发器创建

```
1  /*
2  CREATE TRIGGER 触发器名称 BEFORE|AFTER INSERT|UPDATE|DELETE ON 表名
3  FOR EACH ROW 业务逻辑
4  */
5  #当b_user表中插入数据后，b_log表中也插入一条数据
6  CREATE TRIGGER trigger_insert AFTER INSERT ON b_user
7  FOR EACH ROW INSERT INTO b_log(字段) VALUES('插入数据')
```

- 多条业务逻辑的触发器

```
1  /*
2  DELIMITER $
3  CREATE TRIGGER 触发器名称 BEFORE|AFTER INSERT|UPDATE|DELETE ON 表名
4  FOR EACH ROW
5  BEGIN
6  INSERT...;
7  UPDATE...;
8  END;$
9  */
10 #在b_user表中插入数据前，b_log表中插入2条数据
11 DELIMITER $
12 CREATE TRIGGER trigger_insert_before BEFORE INSERT ON b_user
13 FOR EACH ROW
14 BEGIN
15 INSERT INTO b_log(comments,name) values('insert1' , NEW.name);
16 INSERT INTO b_log(comments,name) values('insert2' , NEW.name) ;
17 END;$
```

总结

- BEFORE|AFTER INSERT用于获取将要插入的数据
- BEFORE|AFTER UPDATE|DELETE用于获取已经修改或删除的数据

3、删除触发器

```
1 DROP TRIGGER 触发器名称
```

二、存储过程

1、变量

1.1 系统变量

由mysql数据库管理系统提供的，变量名称固定，可以修改和查看值，分为**全局变量**和**会话变量**

全局变量：当mysql服务没有重启时，我们可以查看和修改的变量

会话变量：和MySQL连接形成的会话，生命周期在整个会话过程中

全局变量用global修饰，会话变量用session修饰，通常session可以省略

- 查看系统变量

```
1 SHOW GLOBAL variables; -- 查看全局变量
2 SHOW SESSION variables; -- 查看会话变量
3 SHOW variables; -- 查看会话变量
4 SHOW GLOBAL variables like '%dir%'; -- 模糊查询环境变量
5 SELECT @@datadir; -- 查看全局系统变量
6 SELECT @@session_track_transaction_info;
```

- 修改系统变量

```
1 SHOW GLOBAL variables like 'autocommit'; -- 全局系统变量中为自动提交事务
2 SET GLOBAL autocommit=0; -- 将全局的自动提交的事务改为手动提交
3 SHOW SESSION variables link 'autocommit'; -- 查看会话变量中自动提交事务
4 SET SESSION autocommit=0; -- 将会话变量中自动提交的事务改为手动提交
5 SET @@session.autocommit=1;
6 SET @@global.autocommit=1;
```

全局变量在修改后，在不同的会话中都会立即生效，但是在重新启动mysql服务后，全局变量会恢复为默认值，如果想让全局变量依旧有效，需要去修改.ini文件（MySQL配置文件）

会话变量在修改后只对当前会话有效。一般在开发过程中修改会话变量。如：字符编码格式等可以在ini文件中进行设置

1.2 用户变量

MySQL允许用户自定义变量，分为用户变量和局部变量

- 用户变量

作用域：当前会话有效

```

1  #设置方式1，先去声明并初始化用户变量，赋值操作既可以使用=进行赋值，也可以使用:=进行赋值
2  SET @变量名=值；
3  SET @变量名:=值；
4  SELECT @变量名:=值；
5  #设置方式2
6  SELECT 字段 into @变量名 FROM 表名；

```

- 局部变量

作用域：在begin end的结构体中，声明必须是begin end结构体的第一句

```

1  #声明方式，必须在begin后面从第一行开始
2  DECLARE 变量名 类型；
3  DECLARE 变量名 类型 DEFAULT 值；
4
5  #局部变量的赋值
6  SET 变量名:=值；
7  SELECT @变量名:=值；
8  SELECT 字段 into 变量名 FROM 表名；

```

2、存储过程的创建

存储过程是一组已经预先编译好的sql语句的集合，理解为批处理语句（增加流程控制语句），一般在复杂逻辑中才会使用存储过程

- 存储过程的优点
 - 提供了代码的可用性
 - 简化了数据库操作，将业务逻辑的细节隐藏在存储过程中
 - 减少了编译次数，减少了网络IO的次数，从而提高操作效率
- 存储过程的创建

```

1  /*
2  DELIMITER $
3  CREATE PROCEDURE 存储过程的名称(参数列表)
4  BEGIN
5  局部变量的定义
6  多条sql语句
7  流程控制语句
8  END;$
9  */

```

如果存储过程中只有一条SQL语句可以省略BEGIN END

参数列表

| 参数模式 | 形参名称 | 参数类型 |
|-------|----------|-----------------------------|
| IN | username | mysql数据库中的数据类型（数值型，字符型，日期型） |
| OUT | pwd | mysql数据库中的数据类型（数值型，字符型，日期型） |
| INOUT | xxx | mysql数据库中的数据类型（数值型，字符型，日期型） |

IN：声明该参数是一个输入性参数（类似于java中的形参）

OUT: 声明该参数为一个输出型参数（类似于java中的返回值），在一个存储过程中可以定义多个out类型的参数

INOUT: 声明该参数可以为输入型参数，也可以为输出型参数

- 存储过程调用

```
1 CALL 存储过程的名称(实参列表)
2 -- 实参列表中包含由输出类型的参数
```

- 存储过程演示

- 无参的存储过程

```
1 #用于向b_user表中插入2条数据
2 DELIMITER $
3 CREATE PROCEDURE pro_insert()
4 BEGIN
5 INSERT INTO b_user(name,sex) VALUES('1','1');
6 INSERT INTO b_user(name,sex) VALUES('2','2');
7 END;$
8
9 CALL pro_insert();
```

- 带有IN模式参数的存储过程

```
1 #用于向b_user插入2条数据，性别由客户输入
2 DELIMITER $
3 CREATE PROCEDURE pro_insert2(IN sex CHAR(1))
4 BEGIN
5 INSERT INTO b_user(name,sex) VALUES('1',sex);
6 INSERT INTO b_user(name,sex) VALUES('2',sex);
7 END;$
8
9 CALL pro_insert2('男');
```

- 多个带有IN参数的存储过程

```
1 #用于向b_user插入2条数据，用户名和密码由客户输入
2 DELIMITER $
3 CREATE PROCEDURE pro_insert3(IN name VARCHAR(10),IN sex
  VARCHAR(20))
4 BEGIN
5 INSERT INTO b_user(name,sex) VALUES(name,sex);
6 INSERT INTO b_user(name,sex) VALUES(name,sex);
7 END;$
8
9 CALL pro_insert2('uname','男');
```

- 带IN, OUT参数的存储过程

```
1 #判断用户登录，如果用户名和密码输入正确登录成功，否则登录失败
2 #根据输入的用户名和密码作为条件去b_user表中查询，如果查询总行数==1，则认为
  登录成功，让result返回登录成功，否则登录失败
3 DELIMITER $
```

```

4 CREATE PROCEDURE pro_login(IN name VARCHAR(20),IN pwd
  VARCHAR(20),OUT result VARCHAR(20))
5 BEGIN
6 DECLARE total INT DEFAULT 0;-- 用于存放查询总行数
7 select count(*) from b_user u where u.name=name and u.pwd=pwd;-
  -- 将查询结果赋值给total局部变量
8 SET result:=IF(total=1,'登录成功','登录失败');
9 END;$
10 #存储过程如何执行
11 -- 解决判断，使用自定义变量
12 SET @result:='';
13 CALL pro_login('李四','123',@result);
14 select @result;

```

■ 删除存储过程

```

1 DROP PROCEDURE 存储过程名称

```

■ 查看存储过程

```

1 SHOW CREATE PROCEDURE 存储过程名称;

```

■ 修改存储过程

```

1 DROP
2 CREATE

```

2.1 流程控制语句

选择结构

- IF函数
 - 功能：三目运算
 - 语法：IF(逻辑表达式, 表达式1, 表达式2)
- IF结构
 - 功能：实现多路选择
 - 注意：只能用在BEGIN END结构体中

```

1 /*
2 IF 逻辑表达式 THEN 语句1;
3 ELSEIF 逻辑表达式2 THEN 语句2;
4 ...
5 ELSE 语句n;
6 END IF;
7 */

```

- CASE结构
 - 等值选择

```

1 CASE 字段|变量|表达式
2 WHEN 值 THEN 值|语句
3 WHEN 值 THEN 值
4 ...
5 ELSE 值
6 END

```

- 不等值选择

```

1 CASE
2 WHEN 逻辑表达式 THEN 语句1
3 ...
4 ELSE 语句n
5 END

```

循环结构

- WHILE

```

1 /*
2 WHILE 逻辑表达式 DO
3 循环体
4 END WHILE;
5 */
6 #需求: 创建存储过程, 输入一个值, 返回1到该值的和
7 #分析: 一个输入参数, 一个返回值, 在结构体中, 从1循环到输入的值, 求和
8 DELIMITER //
9 CREATE PROCEDURE pro_sum(IN input INT,OUT total INT)
10 BEGIN
11 DECLARE i INT DEFAULT 1;
12 DECLARE sum_ INT DEFAULT 0;
13 WHILE i<=input do
14 SET sum_=sum_+i;
15 SET i=i+1;
16 END WHILE;
17 SET totle:=sum_;
18 END;//
19
20 SET @result=0;
21 CALL por_sun(10,@result);
22 SELECT @result;

```

- LOOP

```

1 #Loopnaem是定义的循环名称, 为了跳出循环时指定跳出的循环
2 loopname:LOOP;
3 IF 逻辑表达式 THEN
4 LEAVE loopname; -- 跳出当前指定的循环, 类似于java中的break
5 END IF;
6 END LOOP;
7
8 DELIMITER //
9 CREATE PROCEDURE pro_sum_loop(IN input INT,OUT total INT)
10 BEGIN
11 DECLARE i INT DEFAULT 1;

```

```

12 DECLARE sum_ INT DEFAULT 0;
13 a:LOOP;
14 SET sum_:=sum_+i;
15 SET i:=i+1;
16 IF i>input THEN
17 LEAVE a;
18 END IF;
19 END LOOP;
20 SET total:=sum_;
21 END;//
22
23 SET @result=0;
24 CALL por_sum_loop(10,@result);
25 SELECT @result;

```

- REPEAT

```

1 REPEAT
2 循环体
3 UNTIL 逻辑表达式 -- 当满足逻辑表达式，跳出循环
4 END REPEAT;
5
6 DELIMITER //
7 CREATE PROCEDURE pro_sum_loop(IN input INT,OUT total INT)
8 BEGIN
9 DECLARE i INT DEFAULT 1;
10 DECLARE sum_ INT DEFAULT 0;
11 REPEAT
12 SET sum_:=sum_+i;
13 SET i:=i+1
14 UNTIL i>input
15 END REPEAT;
16 SET total:=sum_;
17 END;//
18
19 SET @result=0;
20 CALL por_sum_loop(10,@result);
21 SELECT @result;

```

三、存储函数

函数也是一组预先编译好的sql的集合，基本和存储过程相似

函数和存储过程的区别

1. 存储过程可以有0个，1个或多个返回值，适用于insert、update、delete操作
2. 函数只能有一个返回值，适用于在处理数据以后，返回一个已知的结果

1、创建函数

```

1 CREATE FUNCTION 函数名称(参数列表) RETURNS 返回类型 BINLOG参数
2 BEGIN
3 函数体
4 END

```

参数列表：参数名称 参数类型

BINLOG参数

- NO SQL：函数体中没有sql语句，也不会改参数
- READS SQL DATE：函数体中存在sql语句，但是整个数据是只读的，不会修改数据
- MODIFIES SQL DATE：函数体中存在SQL语句，并且会修改数据
- CONTAINS SQL：函数体中包含有SQL语句

函数体：在函数体汇总必须包含return语句，将return放在函数体最后一行执行

```
1  #写一个函数，用于求两数之和
2  DELIMITER //
3  CREATE FUNCTION sum_(input1 INT,input2 INT) RETURNS INT NO SQL
4  BEGIN
5      return input1+input2;
6  END; //
```

2、使用函数

```
1  SELECT 函数名(参数列表);
```

3、查看函数

```
1  SHOW CREATE FUNCTION 函数名;
```

4、删除函数

```
1  DROP FUNCTION 函数名;
```

###

二、定时任务

一、查看定时策略是否开启

```
1  show variables like '%event_sche%';
```

开启定时策略：

```
1  set global event_scheduler=1;
```


三、创建定时任务

```
1 create event run_event
2 on schedule every 1 minute
3 on completion preserve disable
4 do call test_procedure ();
```

- 1、create event day_event：是创建名为run_event的事件
- 2、创建周期定时的规则，意思是每分钟执行一次
- 3、on completion preserve disable是表示创建后并不开始生效。
- 4、do call test_procedure ()是该event(事件)的操作内容

四、定时任务操作

- 1、查看定期任务

```
1 SELECT event_name,event_definition,interval_value,interval_field,status
2 FROM information_schema.EVENTS;
```

- 2、开启或关闭定时任务

```
1 alter event run_event on completion preserve enable;--开启定时任务
2 alter event run_event on completion preserve disable;--关闭定时任务
```

五、定时规则

- 1、周期执行-关键字 EVERY

单位有：second、minute、hour、day、week(周)、quarter(季度)、month、year

```
1 on schedule every 1 week --每周执行1次
```

- 2、在具体某个时间执行-关键字 AT

```
1 on schedule at current_timestamp()+interval 5 day --5天后执行
2 on schedule at '2019-01-01 00:00:00' --在2019年1月1日，0点整执行
```

- 3、在某个时间段执行-关键字 STARTS ENDS

```
1 on schedule every 1 day starts current_timestamp()+interval 5 day ends
  current_timestamp()+interval 1 month --5天后开始每天都执行执行到下个月底
2 on schedule every 1 day ends current_timestamp()+interval 5 day --从现在起每天
  执行，执行5天
```