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

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

- 多条业务逻辑的触发器

```
/*
DELIMITER $
CREATE TRIGGER 触发器名称 BEFORE|AFTER INSERT|UPDATE|DELETE ON 表名
FOR EACH ROW
BIGIN
INSERT...;
UPDATE...;
END;$
*/
#在b_user表中插入数据前，b_log表中插入2条数据
DELIMITER $
CREATE TRIGGER trigger_ insert_before BEFORE INSERT ON b_user
FOR EACH ROW
BEGIN
INSERT INTO b_log(comments,name) values('insert1' , NEW.name);
INSERT INTO b_log(comments,name) values('insert2' , NEW.name) ;
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

```
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

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

- 修改系统变量

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

```
#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
SET @变量名=值;
SET @变量名:=值;
SELECT @变量名:=值;
#设置方式2
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

```
#声明方式，必须在begin后面从第一行开始
DECLARE 变量名 类型;
DECLARE 变量名 类型 DEFAULT 值;

#局部变量的赋值
SET 变量名:=值;
SELECT @变量名:=值;
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

```
/*
DELIMITER $
CREATE PROCEDURE 存储过程的名称(参数列表)
BEGIN
局部变量的定义
多条sql语句
流程控制语句
END;$
*/
```

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姓参数（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

```
CALL 存储过程的名称(实参列表)
-- 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

```
#用于向b_user表中插入2条数据
DELIMITER $
CREATE PROCEDURE pro_insert()
BEGIN
INSERT INTO b_user(name,sex) VALUES('1','1');
INSERT INTO b_user(name,sex) VALUES('2','2');
END;$

CALL pro_insert();
```

  - 带有IN模式参数的存储过程

```
#用于向b_user插入2条数据，性别由客户输入
DELIMITER $
CREATE PROCEDURE pro_insert2(IN sex CHAR(1))
BEGIN
INSERT INTO b_user(name,sex) VALUES('1',sex);
INSERT INTO b_user(name,sex) VALUES('2',sex);
END;$

CALl pro_insert2('男');
```

  - 多个带有IN参数的存储过程

```
#用于向b_user插入2条数据，用户名和密码由客户输入
DELIMITER $
CREATE PROCEDURE pro_insert3(IN name VARCHAR(10),IN sex
VARCHAR(20))
BEGIN
INSERT INTO b_user(name,sex) VALUES(name,sex);
INSERT INTO b_user(name,sex) VALUES(name,sex);
END;$

CALL pro_insert2('uname','男');
```

  - 带IN，OUT参数的存储过程

```
#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
#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
DELIMITER $
```

```
CREATE PROCEDURE pro_login(IN name VARCHAR(20),IN pwd
VARCHAR(20),OUT result VARCHAR(20))
BEGIN
DECLARE total INT DEFAULT 0;-- 用于存放查询总行数
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
SET result:=IF(total=1,'登录成功','登录失败');
END;$
#存储过程如何执行
-- 解决判断，使用自定义变量
SET @result:='';
CAll pro_login('李四','123',@result);
select @result;
```

- 删除存储过程

```
DROP PROCEDURE 存储过程名称
```

- 查看存储过程

```
SHOW CREATE PROCEDURE 存储过程名称;
```

- 修改存储过程

```
DROP
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

```
/*
IF 逻辑表达式 THEN 语句1;
ELSEIF 逻辑表达式2 THEN 语句2;
...
ELSE 语句n;
END IF;
*/
```

- CASE结构
  - 等值选择

```
CASE 字段|变量|表达式
WHEN 值 THEN 值|语句
WHEN 值 THEN 值
...
ELSE 值
END
```

- 不等值选择

```
CASE
WHEN 逻辑表达式 THEN 语句1
...
ELSE 语句n
END
```

循环结构

- WHILE

```
/*
WHILE 逻辑表达式 DO
循环体
END WHILE;
*/
#需求：创建存储过程，输入一个值，返回1到该值的和
#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
DELIMITER //
CREATE PROCEDURE pro_sum(IN input INT,OUT total INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE sum_ INT DEFAULT 0;
WHILE i<=input do
SET sum_=sum_+i;
SET i=i+1;
END WHILE;
SET totle:=sum_;
END;//

SET @result=0;
CALL por_sun(10,@result);
SELECT @result;
```

- LOOP

```
#Loopnaem是定义的循环名称，为了跳出循环时指定跳出的循环
loopname:LOOP;
    IF 逻辑表达式 THEN
    LEAVE loopname; -- 跳出当前指定的循环，类似于java中的break
    END IF;
END LOOP;

DelIMITER //
CREATE PROCEDURE pro_sum_loop(IN input INT,OUT total INT)
BEGIN
DECLARE i INT DEFAULT 1;
```

```
DECLARE sum_ INT DEFAULT 0;
a:LOOP;
SET sum_:=sum_+i;
SET i:=i+1;
IF i>input THEN
LEAVE a;
END IF;
END LOOP;
SET total:=sum_;
END;//

SET @result=0;
CALL por_sum_loop(10,@result);
SELECT @result;
```

- REPEAT

```
REPEAT
循环体
UNTIL 逻辑表达式 -- 当满足逻辑表达式，跳出循环
END REPEAT;

DELIMITER //
CREATE PROCEDURE pro_sum_loop(IN input INT,OUT total INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE sum_ INT DEFAULT 0;
REPEAT
SET sum_:=sum_+i;
SET i:=i+1
UNTIL i>input
END REPEAT;
SET total:=sum_;
END;//

SET @result=0;
CALL por_sum_loop(10,@result);
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

```
CREATE FUNCTION 函数名称(参数列表) RETURNS 返回类型 BINLOG参数
BEGIN
函数体
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

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

2、使用函数

```
SELECT 函数名(参数列表);
```

3、查看函数

```
SHOW CREATE FUNCTION 函数名;
```

4、删除函数

```
DROP FUNCTION 函数名;
```

###

二、定时任务

一、查看定时策略是否开启

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

开启定时策略：

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

三、创建定时任务

```
create event run_event
on schedule every 1 minute
on completion preserve disable
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

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

2、开启或关闭定时任务

```
alter event run_event on completion preserve enable;//开启定时任务
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

```
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

```
on schedule at current_timestamp()+interval 5 day //5天后执行
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

```
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
on schedule every 1 day ends current_timestamp()+interval 5 day //从现在起每天
执行，执行5天
```