游标

查看存储程序

查看信息

语法

```
show procedure | function status [where expression];
```

样例

```
show procedure status;
show function status where name like '%product%';
```

查看源码

语法

```
show create procedure | function stored_procedure_name;
```

样例

```
show create procedure build_email_list;
show create function GetStudentSizeByDept2;
```

存储过程/函数基本语法

```
-- Procedures
create procedure proc_name([params])
begin
routine_body
end
-- Functions
create function func_name([params])
returns data_type -- 存储函数要求指定返回的类型
begin
routine_body
end
```

- params：`[in|out|inout]param_name data_type`一般默认是in。

- 只有in类型的参数可以在存储函数中使用。

变量

语法

```
declare x; -- 普通变量
declare @x; -- 会话变量
set x = 1; -- 赋值
```

流程控制语句

IF

```
IF if_expression THEN commands
[ELSEIF elseif_expression THEN commands]
[ELSE commands]
END IF;
```

CASE

```
CASE  case_expression
WHEN when_expression_1 THEN
    commands
WHEN when_expression_2 THEN
    commands
...
ELSE commands
END CASE;
```

Loop

```
-- While Loop
WHILE expression DO
      Statements
END WHILE;
-- Repeat Loop
REPEAT
     Statements;
UNTIL expression
END REPEAT;
```

触发器

触发器定义

触发器（Trigger）是用户定义在关系表上的一类由事件驱动的特殊过程。

- 触发器保存在数据库服务器中
- 任何用户对表的增、删、改操作均由服务器自动激活相应的触发器
- 触发器可以实施更为复杂的检查和操作，具有更精细和更强大的数据控制能力

ECA模型

- E指Event，即是否检测到insert、delete和update等事件的发生?
- C指Condition，即如果E发生，是否还满足额外的条件?
- A指Action，即如果E发生，并且C被满足，则对应的触发器会被调用。

触发器定义语法

```
create trigger trigger_name  <触发器名>
 { before | after }
 { insert | update | delete } ON <表名>
 for each row
 <trigger body>
```

对上述语法的一些解释：

- before、after：在事件发生前运行触发器内容还是在事件发生后运行触发器内容。
- for each row：触发器运行在那些受事件影响的行（记录）上。触发器分为row trigger和statement trigger，row trigger如前所述，statement trigger则会运行在所有行（记录）上。MySQL只支持row trigger，不支持statement trigger。

触发器删除语法

```
drop trigger [if exists] trigger_name
```

OLD和NEW

- NEW和OLD分别指引用事件之后的新值和事件之前的旧值，其中old是只读的。
- insert中只有new，delete中只有old，update两者都有。

举例

```
create trigger before_insert_sale
before
insert on Sale
for each row
begin
    set new.sale_date = curdate();
    if new.percent < 5 then
        set new.percent = 5;
    end if;
end
```

对这个样例的一些解释（这里只介绍主体，其余部分上面已经介绍过）：

- `set new.sale_date = curdate();`：将insert数据的sale_date字段的值修改为函数`curdate`返回的值。
- `if X then X end if`:判断insert数据的percent字段的值是否小于5，如果小于5，则将其改为5，否则不做处理。
- 由此可见，new是可读可写的。

如果触发器不是before，而是after，则上述修改不起作用，因为值已经写入数据库了。

触发器的应用

- 修改插入数据库的字段的值。
- 强制执行数据完整性约束。
- 维护数据一致性。

触发器的限制

触发器不能做到以下：

1. 修改DML正在使用的表，不使用NEW或OLD别名。
2. 使用SELECT而不使用INTO变量名。
3. 使用SHOW命令。
4. 使用ALTER VIEW。
5. 在存储程序中使用RETURN。
6. 使用显式或隐式开始或结束事务的语句，例如“开始事务”、“提交”或“回滚”。

Events事件

我们可以根据时间设置启动一个事件。创建事件的基本语法如下所示。

```
create event event_name
on schedule schedule_spec
do event_body;

# 以下是解释
schedule_spec: at timestamp [+ interval interval_spec] | every interval [starts timestamp [+ interval interval_spec] ] [ends timestamp [+ interval interval_spec] ]
interval_spec: quantity {year | quarter | month | day | hour | minute | week | second | year_month | day_hour | day_minute | day_second | hour_minute | hour_second | minute_second}
```

样例1

假设我们有一张表：`MySchedule(event_name, event_time, event_place)`。现在，我需要向表中插入一个3h后的会议。语法如下所示。

```
create event my_schedule
on schedule at current_timestamp + interval 3 hours
do insert into MySchedule values (‘faculty
 meeting’, ‘2013-11-14 16:30:00’, ‘G11’);
```

样例2

假设同样例1。现在的需求为：每天从MySchedule表中删除过期的活动。

```
create event my_schedule
on schedule every 1 day
do delete from MySchedule where event_time < current_timestamp;
```

参考资料

- SZU dzh的PPT