mysql视图触发器存储过程

0.课程内容

1. 视图
2. 触发器
3. 存储过程

数据表：

结构：

```
mysql> use mysql_php;
Database changed
mysql> desc user;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| id     | int(11)      | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255) | YES  |     | NULL    |                |
| age    | int(11)      | YES  |     | NULL    |                |
| sex    | varchar(255) | YES  |     | NULL    |                |
| status | varchar(255) | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
5 rows in set (0.05 sec)

mysql>
```

数据：

```
mysql> select * from `user`;
+----+----------+------+------+--------+
| id | name     | age  | sex  | status |
+----+----------+------+------+--------+
|  1 | 小红     |   11 | 女   | 1      |
|  2 | 小张     |   11 | 男   | 0      |
|  5 | 小哈     |   20 | 男   | 1      |
|  6 | 小星     |   20 | 男   | 1      |
|  7 | 小星     |   20 | 男   | 1      |
|  8 | 小cara   |   20 | 男   | 1      |
| 10 | 小shine  |   21 | 男   | 1      |
+----+----------+------+------+--------+
7 rows in set (0.01 sec)

mysql>
```

1. 视图

1.1 概念

1. 视图是 MySQL 在 5.0.1 版本中加入的功能。它可以理解为一个虚表。
2. 之所以被称为虚表，是因为它只是存储了一个结构，并不存储真实的数据。行和列的数据来自定义视图的查询中使用的表，并且是在使用视图时动态生成的，只保存了sql逻辑，不保存查询结果。
3. 视图并不是真的优化

1.2 创建一个视图

```
--创建视图的语法：
create view 视图名称 as select语句;

--创建一个视图
create view user_view as select id,name,age,sex from user;
```

```
Database changed
mysql> create view user_view as select id,name,age,sex from user;
Query OK, 0 rows affected (0.11 sec)

mysql>
```

1.3 查看视图

查看视图的方式和表一样可以通过select来查看，desc查看视图结构

```
1 desc user_view;
```

信息 | 结果1 | 概况 | 状态

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		0	
name	varchar(255)	YES		(Null)	
age	int(11)	YES		(Null)	
sex	varchar(255)	YES		(Null)	

```
--查看视图结构
desc user_view;
show create view user_view;
```

```
mysql> show create view user_view \G;
*************************** 1. row ***************************
                View: user_view
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW `user_view` AS sel
ect `user`.`id` AS `id`,`user`.`name` AS `name`,`user`.`age` AS `age`,`user`.`sex` AS `sex` from `user`
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)

ERROR:
No query specified

mysql>
```

```
--查看视图内容
select * from user_view;
```

```
mysql> select * from user_view;
+----+----------+------+------+
| id | name     | age  | sex  |
+----+----------+------+------+
|  1 | 小红     |   11 | 女   |
|  2 | 小张     |   11 | 男   |
|  5 | 小哈     |   20 | 男   |
|  6 | 小星     |   20 | 男   |
|  7 | 小星     |   20 | 男   |
|  8 | 小cara   |   20 | 男   |
| 10 | 小shine  |   21 | 男   |
+----+----------+------+------+
7 rows in set (0.00 sec)

mysql>
```

> **注意：视图查询的数据实则来自于源数据中的内容。而它本质只是存储了一个结构，并不是存储真实的数据**

> 比如：PHP当中封装了一个查询的方法

```
<?php
class db
{

  function select()
  {
    return "sql:select id,name,age,sex from `user`";
  }
}
?>
```

1.4 视图的优点

1. 第一个显著优点就是它简化了操作。此时我们完全不用关心视图是怎么处理数据的，我们只需要知道如何使用这个结果集即可，视图相当于一个中间层。
2. 第二个显著优点就是它更加安全。比如我们可以让用户有权去访问某个视图，但是不能访问原表，这样就可以起到保护原表中某些数据的作用。
3. 我们之后会接触到管理权限，权限是无法细致到某一个列的，通过视图，则很容易实现。
4. 第三个显著优点就是降低耦合。假如我们以后要修改原表的结构，那么我们可以通过修改视图的定义即可，而不用修改应用程序，对访问者是不会造成影响的，一般来说，这样代价会更小。

1.5 视图的缺点

1. 性能：从数据库视图查询数据可能会很慢，特别是如果视图是基于其他视图创建的。
2. 表依赖关系：将根据数据库的基础表创建一个视图。每当更改与其相关联的表的结构时，都必须更改视图。

1.6 视图的UID

表的数据是可以修改与更新的，但是在视图就不一定了。

```
--修改视图字段的值会直接修改到源数据表
update `user_view` set name='小哈' where id=1;
select * from `user`;

--删除视图
drop view user_view;
```

```
1 update `user_view` set name='小哈' where id=1;
2 select * from `user`;
```

信息 | 结果1 | 概况 | 状态

id	name	age	sex	status
1	小哈	11	女	1
2	小张	11	男	0
5	小哈	20	男	1
6	小星	20	男	1
7	小星	20	男	1
8	小cara	20	男	1
10	小shine	21	男	1

以下是视图不可更新的情况：

1. 包含聚合函数、distinct、group by、having、union、union all。
2. 常量视图。
3. select 包含子查询。
4. 包含连接操作。
5. from 一个不能更新的视图。
6. where 子句的子查询引用了 from 子句中的表。

创建一个视图：

```
create view user_view_2 as select id,name,age,sex,status from `user` where status=0;
select * from `user_view_2`;
```

```
1 create view user_view_2 as select id,name,age,sex,status from `user` where status=0;
2 select * from `user_view_2`;
```

信息 结果1 概况 状态

id	name	age	sex	status
2	小张	11	男	0

现在给视图添加一条status=1的数据

```
insert into `user_view_2`(id,name,age,sex,status)values(11,'小天天',22,'男',1);
select * from `user_view_2`;
select * from `user`;
```

```
1 insert into `user_view_2`(id,name,age,sex,status)values(11,'小天天',22,'男',1);
2 select * from `user_view_2`;
3 select * from `user`;
```

信息 结果1 结果2 概况 状态

id	name	age	sex	status
1	小哈	11	女	1
2	小张	11	男	0
5	小哈	20	男	1
6	小星	20	男	1
7	小星	20	男	1
8	小cara	20	男	1
10	小shine	21	男	1
11	小天天	22	男	1

可以发现即使不满足视图的条件也可以插入数据到表里面

> **with check option**

1. 对于上面的表 t2，我们想：是否可以创建一个视图，它只允许修改满足本视图要求的数据，而对于不满足本视图要求的数据操作， 统统拒绝呢？
2. 答案是肯定的。那就需要 with check option 了，不过该修饰符还有更加深一步的权限机制。
3. 首先我们还是利用上一步的 t2，我们创建一个视图 v3，它的创建

```
drop view `user_view_2`;
create view user_view_3 as select id,name,age,sex,status from `user` where status=0 with check option;
insert into `user_view_3`(id,name,age,sex,status)values(12,'小灰灰',22,'男',1);
select * from `user_view_3`;
```

```
1 create view user_view_3 as select id,name,age,sex,status from `user` where status=0 with check option;
2 insert into `user_view_3`(id,name,age,sex,status)values(12,'小灰灰',22,'男',1);
```

信息 概况 状态

```
[SQL]insert into `user_view_3`(id,name,age,sex,status)values(12,'小灰灰',22,'男',1);
[Err] 1369 - CHECK OPTION failed 'mysql_php.user_view_3'
```

错误信息

[Err] 1369 - CHECK OPTION failed 'mysql_php.user_view_3'

> 这里可以理解为 with check option 的作用就是多了一个 check 的功能，即检查的功能，也就是说插入的数据必须满足该视图的条件，才允许被操作。

1.7 视图的作用&好处

1. 提高了重用性，就像一个函数。如果要频繁获取user的name和goods的name。就应该使用以下sql语言。示例：

```
select a.name as username, b.name as goodsname from user as a, goods as b, ug as c where a.id=c.userid and c.goodsid=b.id;
```

但有了视图就不一样了，创建视图other。示例

```
    create view other as select a.name as username, b.name as goodsname from user as a, goods as b, ug as c where a.id=c.userid and
c.goodsid=b.id;
```

创建好视图后，就可以这样获取user的name和goods的name。示例：

```
    select * from other;
```

以上sql语句，就能获取user的name和goods的name了。

2. 对数据库重构，却不影响程序的运行。假如因为某种需求，需要将user拆房表usera和表userb，该两张表的结构如下： 测试表:usera有id，name，age字段 测试表:userb有id，name，sex字段 这时如果php端使用sql语句：select * from user;那就会提示该表不存在，这时该如何解决呢。解决方案：创建视图。以下sql语句创建视图：

```
    create view user as select a.name,a.age,b.sex from usera as a, userb as b where a.name=b.name;
```

> 以上假设name都是唯一的。此时php端使用sql语句：select * from user;就不会报错什么的。这就实现了更改数据库结构，不更改脚本程序的功能了。

3. 提高了安全性能。可以对不同的用户，设定不同的视图。例如：某用户只能获取user表的name和age数据，不能获取sex数据。则可以这样创建视图。示例如下：

```
    create view other as select a.name, a.age from user as a;
```

> 这样的话，使用sql语句：select * from other; 最多就只能获取name和age的数据，其他的数据就获取不了了。

4. 让数据更加清晰。想要什么样的数据，就创建什么样的视图。

2. 触发器

2.1 概念

1. 触发器（trigger）是MySQL提供给程序员和数据分析员来保证数据完整性的一种方法。
2. 它是与表事件相关的特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件来触发，比如当对一个表进行操作（insert，delete，update）时就会激活它执行。

2.2 创建触发器

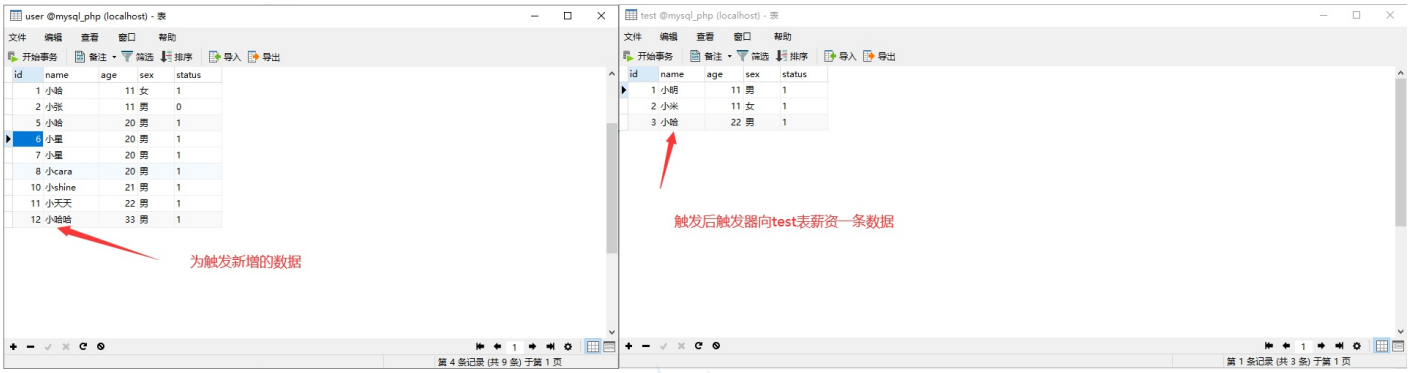
1. 创建触发器的语法：

```
--触发器创建语法
create trigger [触发器名称] trigger_time trigger_event
on [表名] for each row
begin
  trigger_stmt
end;
```

2. trigger_time：指的是触发器的触发时间，可以是 before 或者是 after。其中 before 表示在检查约束前触发，而 after 表示在检查约束后触发。
3. trigger_event： 是触发器的触发事件，可以是 insert、update、delete。
4. trigger_stmt：触发事件执行的语句
5. 对同一个表相同触发时间、相同触发事件的情况下，只能定义一个触发器。
6. 其中的 for each row 表示是行级触发的，也就是以行为单位。也有数据库支持语句级触发。但是目前对于 MySQL 来说 for each row 几乎是必须只能这么写。
7. 定义一个触发器

```
--创建一个触发器，用于监视`user`表，只要`user`表新增数据，`test`表一样增加一条数据
create trigger user_trigger after insert
on `user`
for each row
insert into `test` (id,name,age,sex,status)values(3,'小哈',22,'男',1);
--向`user`表添加一条id=12的数据
insert into `user` (id,name,age,sex,status)values(12,'小哈哈',33,'男',1);
```

```
--查询`test`表与`user`表的内容
select * from `test`;
select * from `user`;
```

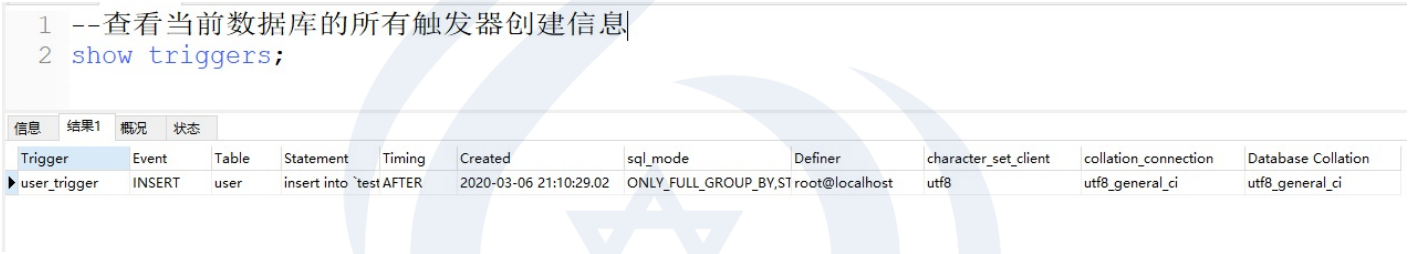


> 这时触发器的作用就已经体现出来了，在我们向 `user` 中插入数据的时候， `test` 中的数据也会自动随着变化.

2.3 查看触发器

如果我们要查看已经创建的触发器，可以选定某个数据库

```
--查看选定数据库的触发器
show triggers \G;
```



```
1 --查看当前数据库的所有触发器创建信息
2 show triggers;
```

信息 | 结果1 | 概况 | 状态

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
user_trigger	INSERT	user	insert into `test	AFTER	2020-03-06 21:10:29.02	ONLY_FULL_GROUP_BY,ST	root@localhost	utf8	utf8_general_ci	utf8_general_ci

2.4 使用的注意事项&产生的问题以及好处

注意事项

1. MySQL触发器能基于行触发，MySQL触发器始终时基于表中的一条记录触发，而不是一组SQL语句。因此，如果需要变动整个数据集而数据集数据量又较大时，触发器效果会非常低。
2. 每一个表的一个事件只能定义一个触发器，例如：不能在AFTER INSERT上定义一个以上的触发器。

产生的问题

由于MySQL触发器基于行触发的特性，因此对于批量操作并不适合使用触发器，如：汇总表、缓存表等。触发器使用不当，可能会导致以下问题：

1. 一个MySQL触发器可能会关联到另外一张表或几张表的操作。因此，会导致数据库服务器负荷也会相应的增加一倍或几倍，如果出现因为触发器问题导致的性能问题，会很难定位问题位置和原因。
2. 在基于锁的操作中，触发器可能会导致锁等待或死锁。触发器执行失败，原来执行的SQL语名也会执行失败。而因为触发器导致的失败结果和失败原因，往往很难排查。

由于MySQL触发器的种种问题，要求我们在创建触发器就应该充分考虑。避免使用不合适的触发器，并能对所有触发器有足够的了解，以便问题的定位和排查。

触发器的好处

对MySQL触发器有足够的认识和了解后，MySQL触发器会给我们带来极大的便利。当实现一些系统约束时，或在实现系统维护及针对操作数据的更新时，使用触发器都非常方便。在之胶我们介绍了MySQL触发器不适合做的一些工作，但MySQL触发器在以下一些应用场景中，会非常实用：

1. 基于行数据变更的日志记录。如：在用户订单系统中，我们可以基于用户订单数据状态的改变，使用触发器构建用户订单日志表数据。
2. 基于行数据变更的关系数据的更新。如：用户订单改变至付款或相关状态时，我们可以基于用户订单数据状态的改变，使用触发器改变用户会付款或相应状态信息。
3. 基于行数据变更的数据汇总。如：用户订单成交或失败，我们可以基于用户订单数据状态的改变，使用触发器构建用户总成交量或失败量汇总数据。

3. 存储过程

3.1 概念

1. MySQL 从 5.0 开始支持存储过程。
2. 存储过程和函数可以理解为一段 SQL 语句的集合，它们被事先编译好并且存储在数据库中。
3. 在 Pascal 语言中，是有"过程"和"函数"的区分的，过程可以理解为没有返回值的函数。不过在 C 家族语言中，则没有过程这个概念，统一为函数。

3.2 创建存储过程

```
--存储过程创建语法
create procedure 存储过程名(参数列表)
begin
存储过程体
end
```

当我们调用一个存储过程的时候，可以使用如下方式：

```
--调用存储过程的方式
call 存储过程名(参数列表)
```

下面创建一个简单的存储过程：

```
--创建一个存储过程，用于创建t1表
create procedure user_procedure()
begin
       create table t1(id int,name varchar(25));
end
--调用存储过程user_procedure
call user_procedure();

--创建一个能向t1表新增数据的存储过程
create procedure user_insert_procedure(in x int,in y varchar(25))--in 表示输入
begin
       insert into t1 values(x,y);
end
--调用存储过程user_insert_procedure
call user_insert_procedure(1,'sixstar');
```

```
--创建一个存储过程，用于创建t1表
create procedure user_procedure()
begin
  create table t1(id int,name varchar(25));
end
--调用存储过程user_procedure
call user_procedure();

--创建一个能向t1表新增数据的存储过程
create procedure user_insert_procedure(in x int,in y varchar(25))
begin
  insert into t1 values(x,y);
end
--调用存储过程user_insert_procedure
call user_insert_procedure(1,'sixstar');
--查询t1表的数据
select * from `t1`;
```

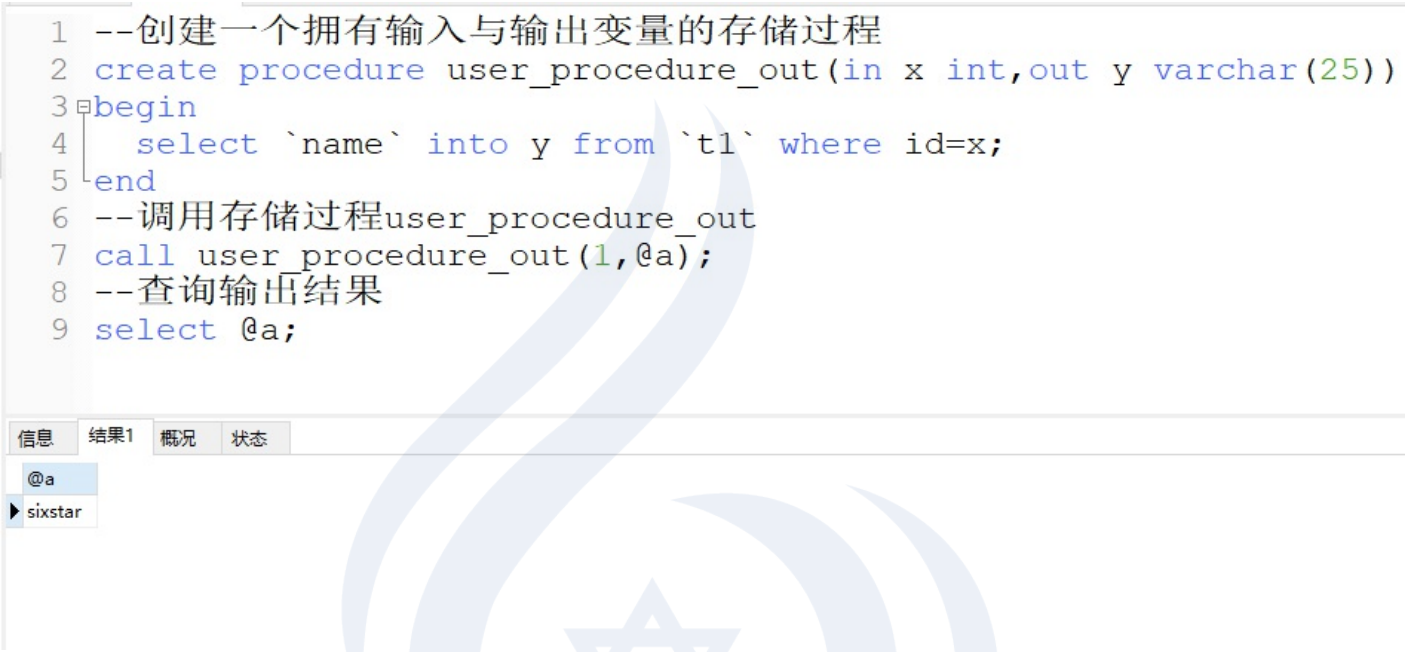
信息 | 结果1 | 概况 | 状态

id	name
1	sixstar

> 参数类型

1. 从上面的过程中我们了解到存储过程有参数类型这种说法，它的类型可以取值有三个：in、out、inout。
2. 其中它们的意义如下： (1)in 表示只是用来输入。 (2)out 表示只是用来输出。 (3)inout 可以用来输入，也可以用作输出。

```
--创建一个拥有输入与输出变量的存储过程
create procedure user_procedure_out(in x int,out y varchar(25))
begin
       select `name` into y from `t1` where id=x;
end
--调用存储过程user_procedure_out
call user_procedure_out(1,@a);
--查询输出结果
select @a;
```



删除存储过程

```
--删除存储过程语法
drop procedure 存储过程名

drop procedure user_procedure;
```

3.3 存储过程的理解

1. 调用存储过程与直接执行 SQL 语句的效果是相同的，但是存储过程的一个好处是处理逻辑都封装在数据库端。
2. 当我们调用存储过程的时候，我们不需要了解其中的处理逻辑，一旦处理逻辑发生变化，只需要修改存储过程即可，对调用它的程序完全无影响。
3. 调用存储过程和函数可以简化应用开发人员的很多工作，减少数据在数据库和应用服务器之间的传输，可以提高数据处理的效率。

3.4 变量

1.存储过程中是可以使用变量的，我们可以通过 declare 来定义一个局部变量，该变量的作用域只是 begin....end 块中。 2.变量的定义必须写在符合语句的开头，并且在任何其他语句的前面。我们可以一次声明多个相同类型的变量，我们还可以使用default 来赋予默认值。 3.定义一个变量的语法为：

```
declare 变量名 1 [,变量名 2...] 变量类型 [default 默认值]
```

4.上面的变量类型就是 MySQL 支持的类型，而变量名的取值规则也是一个老生常谈的话题了，就不赘述了。 5.变量可以直接赋值，还可以通过查询赋值。 6.直接赋值就是使用 set 来进行赋值，它的语法为：

```
set 变量名 1 = 表达式 1 [,变量名 2=表达式 2...]
```

7.也可以通过查询来将结果赋值给变量，它需要要求查询返回的结果只有一行，语法范例：

```
select 列名列表 into 变量列表 from 表名 其他语句;
```

8.定义一个存储过程，练习变量使用

```
create procedure user_procedure_declare(in x int,out y varchar(25))
begin
        declare s varchar(25);
        select `name` into s from `t1` where id=x;
        set y=s;
end
call user_procedure_declare(1,@a);
select @a;
```

3.5 存储过程的数据类型

存储过程中的数据类型

1. 数值类型：Int,float,double,decimal
2. 日期类型：timestamp,date,year
3. 字符串：char,varchar,text

timestamp: 是使用最多的数据类型-》十位数的时间戳 text：一旦用到text类型的时候就可以考虑分表； 如果部分表的话，该字段的查询不会直接放在一起查询，因为多个字段查询中其中如果有text字段的话，就容易遇到慢查询 所以通常的话，如果需要这个值的时候会根据id单独拿这个text字段

3.6 流程控制

```
if 的语法格式为：
if 条件表达式 then 语句
          [elseif 条件表达式 then 语句] ....
          [else 语句]
end if

case 的语法格式
首先是第一种写法：
case 表达式
      when 值 then 语句
      when 值 then 语句
      ...
      [else 语句]
end case
然后是第二种写法：
case
      when 表达式 then 语句
      when 表达式 then 语句
      ....
      [else 语句]
end case

loop 循环 语法格式为：
[标号:] loop
        循环语句
end loop [标号]

while
while a>100
do
  循环语句
End while

Repeat //游标
    SQL语句1 UNTIL
    条件表达式
END Repeat;

Loop
  SQL语句
  所有的条件判断和跳出需要自己实现
End loop

leave 语句用来从标注的流程构造中退出，它通常和 begin...end 或循环一起使用 leave 标号;

声明语句结束符，可以自定义：
DELIMITER [符合]
```

```
delimiter $$ $$
```

3.7 游标

1. 游标也有的资料上称为光标。
2. 我们可以在存储过程中使用游标来对结果集进行循环的处理。
3. 游标的使用步骤基本分为：声明、打开、取值、关闭。

> 语法

```
DECLARE test_cursor CURSOR FOR 结果集; //声明游标
OPEN test_cursor; //打开游标
CLOSE test_cursor; //关闭游标
DECLARE CONTINUE HANDLER FOR NOT FOUND //结果集查询不到数据自动跳出
```

> 总结

1. 游标的声明的语法：declare 游标名称 cursor for 查询语句;
2. 打开光标的语法：open 游标名称;
3. 获取游标数据：fetch 游标名称 into 变量名 1 [,变量名 2 ....]
4. 关闭游标的语法：close 游标名称;
5. 游标的基本使用须知：对某个表按照循环的处理，判断循环结束的条件是捕获 not found 的条件，当 fetch 光标找不到下一条记录的时候，就会关闭光标然后退出过程。
6. 可能有过 Pascal 编程经验的朋友们都会知道，声明的顺序也是很重要的，在 SQL 中，我们使用 declare 定义的顺序是：变量、条件、游标、应用程序。

操作 查询出来的数据会放置于临时表中，然后再通过游标去读取数据。

3.8 存储过程的优点

1. 第一点优势就是执行速度快。因为我们的每个 SQL 语句都需要经过编译，然后再运行，但是存储过程都是直接编译好了之后，直接运行即可。
2. 第二点优势就是减少网络流量。我们传输一个存储过程比我们传输大量的 SQL 语句的开销要小得多。
3. 第三点优势就是提高系统安全性。因为存储过程可以使用权限控制，而且参数化的存储过程可以有效地防止 SQL 注入攻击。保证了其安全性。
4. 第四点优势就是耦合性降低。当我们的表结构发生了调整或变动之后，我们可以修改相应的存储过程，我们的应用程序在一定程度上需要改动的地方就较小了。
5. 第五点优势就是重用性强。因为我们写好一个存储过程之后，再次调用它只需要一个名称即可，也就是"一次编写，随处调用",而且使用存储过程也可以让程序的模块化加强。

3.9 存储过程的缺点

1. 第一个缺点就是移植性差。因为存储过程是和数据库绑定的，如果我们要更换数据库之类的操作，可能很多地方需要改动。 2.第二个缺点就是修改不方便。因为对于存储过程而言，我们并不能特别有效的调试，它的一些 bug 可能发现的更晚一些，增加了应用的危险性。 3.第三个缺点就是优势不明显和赘余功能。对于小型 web 应用来说，如果我们使用语句缓存，发现编译 SQL 的开销并不大，但是使用存储过程却需要检查权限一类的开销，这些赘余功能也会在一定程度上拖累性能。

3.10 小结

1. 存储过程和函数的优势是可以将数据的处理放在数据库服务器上进行，避免将大量的结果集传输给客户端，减少了数据的传输，因此也减少了宽带和服务器的压力。
2. 但是在数据库服务器上进行大量的运算也会占用服务器的 CPU，造成数据库服务器的压力。
3. 一般来说是不建议在存储过程中进行大量的复杂的运算的，它们不是数据库服务器的强项，我们应该把这些操作让应用服务器去处理。