# 第11章 视图、触发器、存储过程

# 视图

## 什么是视图

视图(view)：从一个或几个基本表中根据用户需要而做成一个虚表

1：视图是虚表,它在存储时只存储视图的定义,而没有存储对应的数据

2：视图只在刚刚打开的一瞬间,通过定义从基表中搜集数据,并展现给用户

## 视图与查询的区别

视图和查询都是用由sql语句组成,这是他们相同的地方,但是视图和查询有着本质区别：

它们的区别在于：

1：存储上的区别：视图存储为数据库设计的一部分,而查询则不是.

2：更新限制的要求不一样

要注意：因为视图来自于表,所以通过视图可以间接对表进行更新,我们也可以通过update语句对表进行更新,但是对视图和查询更新限制是不同的,以下我们会知道虽然通过视图可以间接更新表但是有很多限制.

3：排序结果：通过sql语句,可以对一个表进行排序,而视图则不行。比如：创建一个含有order by子句的视图,看一下可以成功吗？

## 视图的优点

为什么有了表还要引入视图呢？这是因为视图具有以下几个优点：

1：能分割数据,简化观点。可以通过select和where来定义视图,从而可以分割数据基表中某些对于用户不关心的数据,使用户把注意力集中到所关心的数据列.进一步简化浏览数据工作。

2：为数据提供一定的逻辑独立性。 如果为某一个基表定义一个视图,即使以后基本表的内容的发生改变了也不会影响“视图定义”所得到的数据。

3：提供自动的安全保护功能。 视图能像基本表一样授予或撤消访问许可权。

4：视图可以间接对表进行更新,因此视图的更新就是表的更新。

## 视图的创建

CREATE VIEW 语句的基本语法如下所示：

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

和普通的 SQL SELECT 查询一样，你可以在上面的 SELECT 语句中包含多个数据表。

实现如下查询结果：

-- 班级名称 学生数量

SELECT b.name AS '班级名称',COUNT(s.id) AS '数量'

FROM student AS s INNER JOIN banji AS b ON s.id=b.id

GROUP BY s.banji\_id;

显示班级名称对应的学生数量，这个查询会经常使用到，我们就可以把这个经常使用到的查询做成一个视图：

CREATE VIEW view\_banji\_count(banji\_name,banji\_count) AS SELECT b.name AS '班级名称',COUNT(s.id) AS '数量'

FROM student AS s INNER JOIN banji AS b ON s.id=b.id

GROUP BY s.banji\_id;

视图可以看成临时存储的数据所构成的一张表，不是一个真实的表，其实他的本质就是select语句的结果集。

但是我们使用的时候可以当成一张表来使用。

SELECT \* FROM view\_banji\_count;

SELECT \* FROM view\_banji\_count WHERE banji\_count>1;

## 视图更新

您可以使用下面的语法来更新视图：

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS

SELECT column\_name(s)

FROM table\_name

WHERE condition

## 视图删除

您可以通过 DROP VIEW 命令来删除视图。

SQL DROP VIEW Syntax

DROP VIEW view\_name

# 触发器

## 触发器定义

进行数据库应用软件的开发时，我们有时会碰到表中的某些数据改变，希望同时引起其他相关数据改变的需求，利用触发器就能满足这样的需求。它能在表中的某些特定数据变化时自动完成某些查询。运用触发器不仅可以简化程序，而且可以增加程序的灵活性。

触发器是一类特殊的事务 ,可以监视某种数据操作(insert/update/delete),并触发相关操作(insert/update/delete)。

## 触发器应用场合

1.当向一张表中添加或删除记录时，需要在相关表中进行同步操作。

比如，当一个订单产生时,订单所购的商品的库存量相应减少。

2.当表上某列数据的值与其他表中的数据有联系时。

比如，当某客户进行欠款消费，

可以在生成订单时通过设计触发器判断该客户的累计欠款是否超出了最大限度。

3.当需要对某张表进行跟踪时。

比如，当有新订单产生时，需要及时通知相关人员进行处理，

此时可以在订单表上设计添加触发器加以实现

## 案例

商品表：goods

订单表：order

当下一个订单的时候，对应的商品要相应减少（买几个商品就少几个库存）

创建触发器的语法

Create trigger triggerName

After/before insert/update/delete on 表名

For each row #这句话是固定的

Begin

Sql语句; # 一句或多句,insert/update/delete范围内

End;

CREATE TABLE goods(

gid INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(20),

num SMALLINT

);

CREATE TABLE orders(

oid INT PRIMARY KEY AUTO\_INCREMENT,

gid INT COMMENT '买的哪件商品',

much SMALLINT

);

INSERT INTO goods VALUES

(1,'cat',34),

(2,'dog',65),

(3,'pig',21);

手动实现这个效果

INSERT INTO orders(gid,much) VALUES(1,2);-- 买了两只猫

UPDATE goods SET num=num-2 WHERE gid=1;

使用触发器完成

DELIMITER $

CREATE TRIGGER insert\_order\_update\_goods\_num\_trigger

AFTER

INSERT

ON orders

FOR EACH ROW

BEGIN

UPDATE goods SET num=num-2 WHERE gid=1;

END$

DELIMITER ;

触发器里after 和before的区别：

After 是先完成数据的增，删，改再触发，

触发的语句晚于监视的增，删，改，无法影响前面的增删改动作。

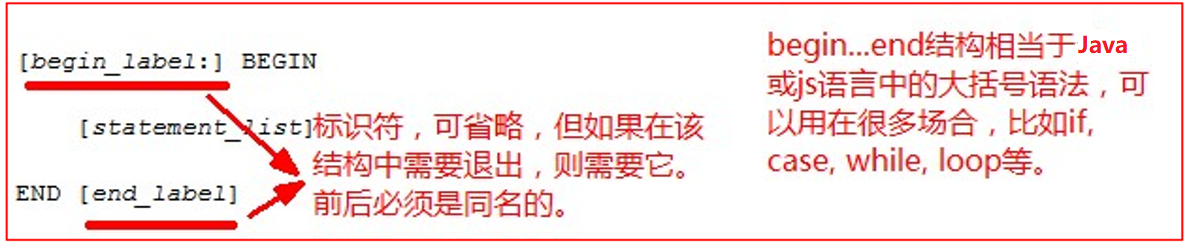
Before是先完成触发，再增删改，

触发的语句先于监视的增，删，改发生，我们有机会判断,修改即将发生的操作。

# 过程

## 基本语法形式

语句块模式：



在mysql编程中，begin....end;基本代替了原来编程语句中的{...}语法。

但又有所区别：

一个bigin...end;块，可以给定一个“标识符”，并且可以使用leave语句来“退出”该语句块。

## 基本流程控制语句

### IF语句

IF语句，是指如果满足某种条件，根据判断的结果为TRUE，或者FALSE执行相应的语句

语法格式

IF expr\_condition THEN statement\_list

[ELSEIF expr\_condition THEN statement\_list]

[ELSE statement\_list]

END IF

IF实现了一个基本的条件构造

参数说明

Expr\_condition，表示判断条件

Statement\_list，表示SQL语句列表，它可以包括一个或多个语句

如果，expr\_condition求值为TRUE，相应的SQL语句列表就会被执行，如果，没有expr\_condition匹配，则ELSE子句李的语句列表被执行

### case语句：（类似java中的switch）

第一种格式

CASE case\_expr

WHEN when\_value THEN statement\_list

[WHEN when\_value THEN statement\_list]……

[ELSE statement\_list]

END CASE

参数说明

Case\_expr，表示条件判断的表达式，决定了哪一个WHEN自己会被执行

When\_value，表示表达式可能的值，如果，某个when\_value表达式与case\_expr表达式结果相同，则执行对应THEN关键字后的statement中的语句

Statement\_list，表示不同when\_value值的执行语句

使用CASE流程控制语句，第一种格式，判断val值等于1、等于2，或者两者都不等于

CASE val

WHEN 1 THEN SELECT ‘val is 1’;

WHEN 2 THEN SELECT ‘val is 2’;

ELSE SELECT ‘val is not 1 or 2’;

END CASE;

当val值为1时，输出字符串val is 1

当val值为2时，输出字符串val is 2

否则，输出字符串val is not 1 or 2

第二种格式

CASE

WHEN expr\_condition THEN statement\_list

[WHEN expr\_condition THEN statement\_list]

[ELSE statement\_list]

END CASE;

注意，这里存储过程中的CASE语句，与控制流程函数中的SQL CASE表达式中的CASE是不同的

存储过程中，CASE语句不能有ELSE NULL子句，并且，用END CASE代替END来终止

### while循环语句

创建一个带条件判断的循环过程

与REPEAT不同的是，WHILE在语句执行时，先对指定的条件进行判断，如果为真，则执行循环内的语句，否则退出循环

语法格式

[while\_lable:] WHILE expr\_condition DO

Statement\_list

END WHILE [while\_lable]

参数说明

While\_lable，为WHILE语句的标注名称

Expr\_condition，为进行判断的表达式，如果表达式为真，WHILE语句内的语句，或语句群就被执行，直至expr\_condition为假，退出循环

使用WHILE语句，进行循环操作

DECLARE i INT DEFAULT 0;

WHILE i<10 DO

SET i=i+1;

END WHILE;

### mysql中的变量

变量它有一定的使用场景限制（其实包括前面的流程控制语句也这样）：

在通常的执行各种create，insert，delete，update，select等语句的“环境”中，并不使用常规变量，也不能使用流程控制语句。

其实，这些语句只能在编程环境中用——通常的增删改查，可以称为命令环境。

编程环境：存储函数中，存储过程中，触发器中。

普通变量的声明语法：

declare 变量名 变量类型 [default 初始值]

变量的赋值语法：

set 变量名 = 变量值

mysql还有一种变量形式，可以称为“会话变量”

会话变量的形式为： @变量名；

会话变量无需单独声明，而是直接赋值就可以，类似php

会话变量的赋值： set 变量名 = 值；

会话变量可以在普通命令行环境中或编程环境中使用。

# 存储函数、存储过程

## 存储函数

### 定义形式



### 使用

跟系统函数的调用形式一样，直接使用名字，并带括号，以及根据需要带实参。

比如：

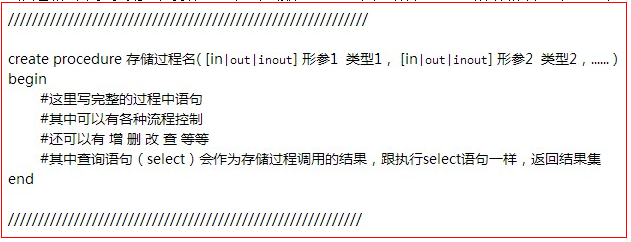
select 3, now(), f1(); //f1为自定义的存储函数。

select 3, now(), f2(3, ‘abc’, @v1); //带3个实参，其中@v1是一个前面赋值了的变量。

## 存储过程

存储过程就是一个没有返回值数据值的“函数”。

虽然存储过程不返回单个数据值，但可以“返回”结果集——这里的返回起始就是select语句的正常结果表现。



其中，

in，out，inout用于修饰一个形参的“数据流向”：

in：只作为外部输入数据，调用环境传入存储过程内部，默认值。

out：只作为输出数据，即存储过程传出到调用环境。

inout：具有双重。

存储过程调用形式：

call 存储过程名（实参1，实参2，.....）；

### 定义形式

DELIMITER $

CREATE PROCEDURE p1()

BEGIN

SELECT 2+3 FROM DUAL;

END$

DELIMITER ;

调用：

CALL p1();

### 使用

存储过程是可以编程的，意味着可以使用便利，表达式，控制结构来完成复杂的功能。

在存储过程中，用declare声明变量。

格式：declare 变量名 变量类型[default 默认值]

DELIMITER $

CREATE PROCEDURE p2()

BEGIN

DECLARE age INT DEFAULT 18;

DECLARE height INT DEFAULT 180;

SELECT CONCAT('年龄:',age,'，身高：',height) FROM DUAL;

END$

DELIMITER ;

调用：

CALL p2();

-- in:是传值过去，out：是输出型变量，传递的是变量。

DELIMITER $

CREATE PROCEDURE p3(IN n INT, OUT total INT)

BEGIN

DECLARE i INT DEFAULT 0;

SET total := 0;

WHILE i < n DO

SET i := i + 1;

SET total := total + i;

END WHILE;

END$

DELIMITER ;

CALL p3(100, @sumary);

SELECT @sumary;

+---------+

| @sumary |

+---------+

| 5050 |

+---------+

# 思途心得

1. 对于视图、触发器、存储过程要重点理解他们使用的场景，也就是他们的作用，这样对于学习视图、触发器、存储过程才能有一个更深的认识。
2. 视图是临时存储的数据所构成的一张表，不是一个真实的表，其实他的本质就是select语句的结果集，但是我们使用的时候可以当成一张表来使用。
3. 存储过程是通过给定的语法格式编写自定义数据库API，包含一系列sql语句的集合，完成一个复杂的功能，这个存储过程可以反复使用，执行效率也很高。
4. 触发器是一个特殊的存储过程，它的执行不是由程序调用，也不是手工启动，而是由事件触发。触发器经常用于加强数据的完整性约束和业务规则等。

# 作业

数据库sales

1. 客户表（Customer）

|  |  |  |  |
| --- | --- | --- | --- |
| 客户编号(CusNo) | 姓名(CusName) | 地址(Address) | 电话(Tel) |
| C001 | 杨婷 | 北京 | 010-5328953 |
| C002 | 李和平 | 上海 | 021-62359651 |
| C003 | 叶新 | 成都 | 024-3222781 |
| C004 | 冯辰诚 | 上海 | 021-87235965 |

1. 产品表(Product)

|  |  |  |  |
| --- | --- | --- | --- |
| 产品编号(ProNo) | 品名(ProName) | 单价(price) | 库存数量(Stocks) |
| P0001 | 液晶电视 | 5600.00 | 800 |
| P0002 | 空调 | 2390.00 | 460 |
| P0003 | 洗衣机 | 3700.00 | 600 |
| P0004 | 电热水器 | 890.00 | 120 |

1. 销售表(ProOut)

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 销售日期(SaleDate) | 客户编号(CusNo) | 产品编号(ProNo) | 销售数量(Quantity) |
| 2007-10-27 | C001 | P0001 | 3 |
| 2007-11-06 | C004 | P0003 | 40 |
| 2007-12-27 | C001 | P0003 | 5 |
| 2008-3-15 | C002 | P0002 | 12 |
| 2008-05-02 | C003 | P0002 | 21 |
| 2008-05-02 | C003 | P0001 | 9 |
| 2008-09-21 | C004 | P0001 | 30 |
| 2008-11-21 | C004 | P0001 | 73 |

1. 创建一自定义函数sumMoney，要求能够利用该函数计算出销售金额，并进行测试，利用该函数计算出每种产品(ProNo)的销售金额。
2. 创建视图viewPro，要求显示每种产品的销售量和销售金额。
3. 创建存储过程p\_Pro, 要求能够根据指定的客户编号，统计客户购买每种产品的产品号、数量。
4. 创建一个触发器t\_Stocks，要求当插入销售表(ProOut)的销售记录时，根据销售数量(Quantity)的变化，能更新产品表(Product)中相应的库存数量(Stocks)。