



数据库原理 与技术

浙江农林大学
ZHEJIANG A&F UNIVERSITY

数据库系统概论

An Introduction to Database System

第四章 MySQL

中国人民大学信息学院

注释符

-- 单行注释

#单行注释

/*

多行注释

*/



用户变量

- 作用域

针对当前会话（连接）有效

- 赋值

- 方式一：@num为变量名，value为值

SET @num=value

SET @num:=value

- 方式二：SELECT INTO

SELECT 字段 INTO @变量名 FROM 表名 WHERE

- 使用

SELECT @num

- 赋值并使用

SELECT @num1:=value1, @num2:=value2,...



局部变量

- 作用域

在其声明的begin...end块内

- 声明

declare 变量名 变量类型 [default...]

- 赋值

➤ 方式一：num为变量名，value为值

SET num=value

SET num:=value

➤ 方式二：SELECT INTO

SELECT 字段 INTO 变量名 FROM 表名 WHERE

- 使用

SELECT num

- 赋值并使用

SELECT num1: =value1, num2:=value2,...



存储过程和函数

一系列SQL语句的集合，封装到一起保存到数据库中

调用：

- 存储过程：CALL 名称和参数
- 存储函数：SELECT 名称和参数

区别：

- 存储过程：没有返回值，参数类型可以是IN、OUT、INOUT
- 存储函数：必须有返回值,参数类型只能是IN

存储过程和函数的优点

- 具有良好的封装性
- 应用程序与SQL逻辑分离
- 让SQL具备处理能力
- 减少网络交互
- 能够提高系统性能
- 降低数据出错的概率
- 保证数据的安全性



创建存储过程和函数

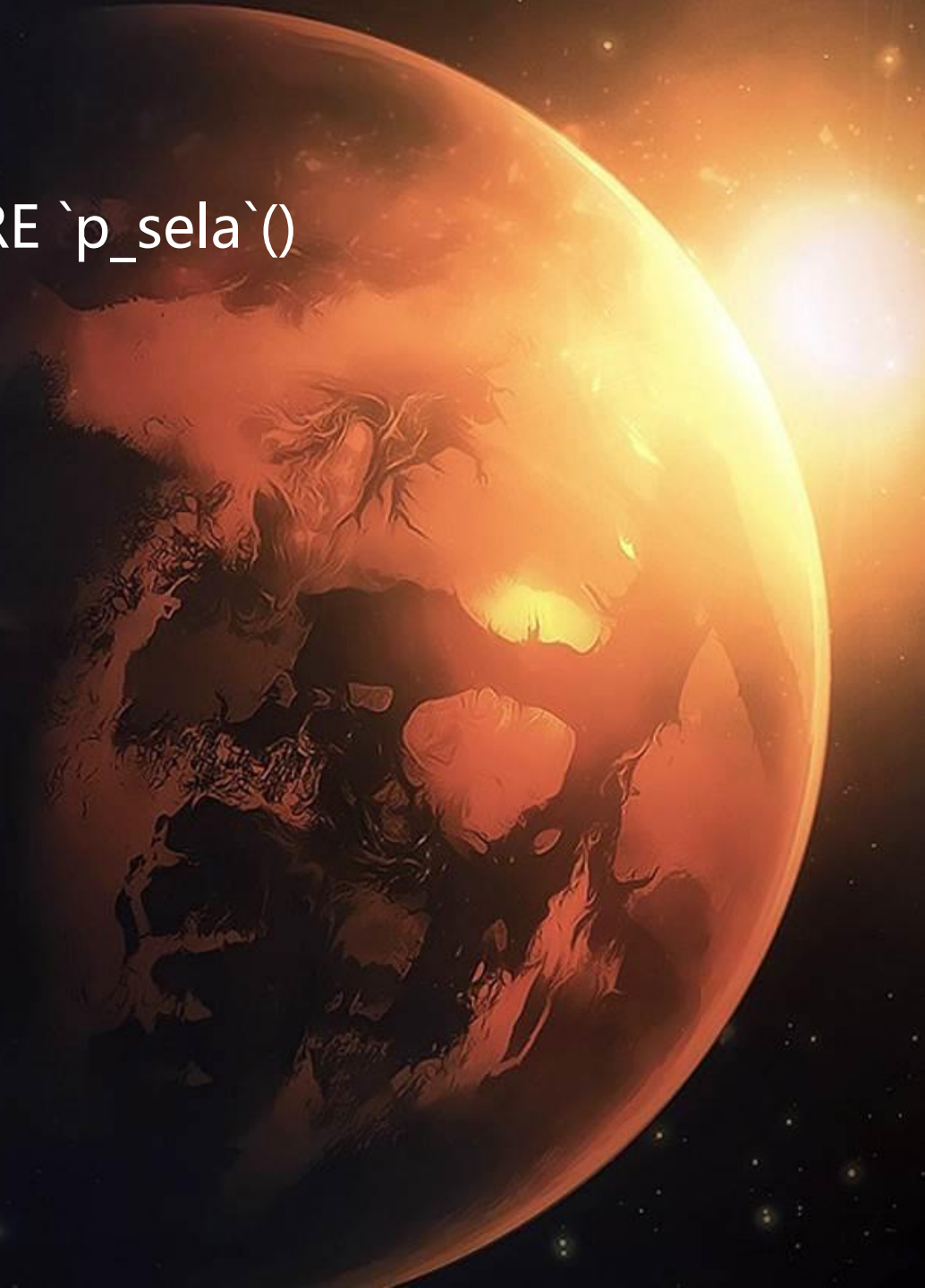
创建存储过程语法

```
CREATE PROCEDURE sp_name ([proc_parameter[,...]])  
    [characteristic ...] routine_body
```

- **proc_parameter**: 参数列表
[IN| OUT| INOUT] param_name type
- **routine_body**: SQL执行体, 使用**BEGIN...END**封装

简单示例

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `p_sela`()  
BEGIN  
    SELECT * FROM s;  
END
```



创建存储过程和函数

创建存储函数语法

```
CREATE FUNCTION func_name ([func_parameter[,...]])  
    RETURNS type  
    [characteristic ...] routine_body
```

- func_parameter: 参数列表,只能是IN类型
- RETURNS type: 创建函数时指定的返回数据类型
- routine_body : 函数的SQL执行体

简单示例

创建存储函数前执行

```
set global log_bin_trust_function_creators=1
```

```
CREATE DEFINER=`root`@`localhost` FUNCTION `f_sela`()
```

```
RETURNS varchar(30) CHARSET utf8mb4
```

```
    DETERMINISTIC
```

```
BEGIN
```

```
    RETURN (SELECT sname FROM s WHERE snum='s030101');
```

```
END
```

查看存储过程和函数

查看创建或定义信息语法

SHOW CREATE {PROCEDURE | FUNCTION} sp_name

例： SHOW CREATE PROCEDURE p_sela
SHOW CREATE FUNCTION f_sela

查看状态信息语法：

SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']

例： SHOW PROCEDURE STATUS LIKE '%sel%'
SHOW FUNCTION STATUS LIKE '%sel%'

调用存储过程和函数

调用存储过程语法:

CALL proc_name ([parameter[,...]])

例: CALL p_sela()

调用函数语法:

SELECT func_name ([parameter[,...]])

例: SELECT f_sela()



删除存储过程和函数

删除存储过程语法:

DROP PROCEDURE [IF EXISTS] proc_name

例: DROP PROCEDURE p_sela

删除调用函数语法:

DROP FUNCTION [IF EXISTS] func_name

例: DROP FUNCTION f_sela



在存储过程中使用变量

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `p_selectcountandscore`()  
BEGIN  
    DECLARE ccount INT DEFAULT 0;  
    DECLARE csum,cavg DECIMAL(10,2) DEFAULT 0.00;  
    SELECT COUNT(*) INTO ccount FROM sc;  
    SELECT SUM(score) INTO csum FROM sc;  
    SET cavg=csum/ccount;  
    SELECT ccount,csum,cavg;  
END
```

在函数中使用变量

```
CREATE DEFINER=`root`@`localhost` FUNCTION `f_tavg`()
RETURNS int DETERMINISTIC
BEGIN
    DECLARE tcount,tsum,tavg INT DEFAULT 0;
    SELECT COUNT(*) INTO tcount FROM t;
    SELECT SUM(tsalary) INTO tsum FROM t;
    SET tavg=tsum/tcount;
    RETURN tavg;
END
```



MySQL中控制流程的使用

使用IF语句控制流程的语法

```
IF search_condition THEN statement_list  
  [ELSEIF search_condition THEN statement_list] ...  
  [ELSE statement_list]  
END IF
```



CASE语句

使用CASE语句控制流程的语法：

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
或者
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```



LOOP

使用LOOP语句控制流程的语法：

```
[begin_label:] LOOP  
    statement_list  
END LOOP [end_label]
```



LEAVE

使用LEAVE语句控制流程的语法：

LEAVE label

其中，label表示被标注的流程标志。

从被标注的流程结果中退出



ITERATE

使用ITERATE语句控制流程的语法：

ITERATE label

其中，label表示被标注的流程标志。

跳过本次循环，而执行下次循环操作

只可出现在LOOP、REPEAT和WHILE语句内

REPEAT

使用REPEAT语句控制流程的语法：

```
[begin_label:] REPEAT  
    statement_list  
UNTIL search_condition  
END REPEAT [end_label]
```

- **begin_label**和**end_label**为循环的标志，可省略
- 当**search_condition**条件为true时，退出循环

WHILE

使用WHERE语句控制流程的语法：

```
[begin_label:] WHILE search_condition DO  
    statement_list  
END WHILE [end_label]
```

- **begin_label**和**end_label**为循环的标志，可省略
- 当**search_condition**条件为**true**时，继续执行循环体

例：带输入和输出参数的存储过程

创建一存储过程，输入参数为学号，输出参数为课程数，总成绩和平均成绩

```
CREATE DEFINER=`root` @`localhost` PROCEDURE `p_sc`(IN  
sid CHAR(12),OUT number INT,OUT avgscore INT,OUT sumscore  
INT)  
BEGIN  
  
SELECT  
COUNT(*),IFNULL(AVG(score),0),IFNULL(SUM(score),0)  
INTO number,avgscore,sumscore  
FROM sc  
WHERE snum=sid;  
END
```


例：带输入和输出参数的存储过程

创建一存储过程，输入参数为学号，输出参数为课程数，总成绩和平均成绩

```
CALL p_sc('s030101',@counts,@avgs,@sums);  
SELECT @counts,@avgs,@sums;
```

触发器 (Trigger)

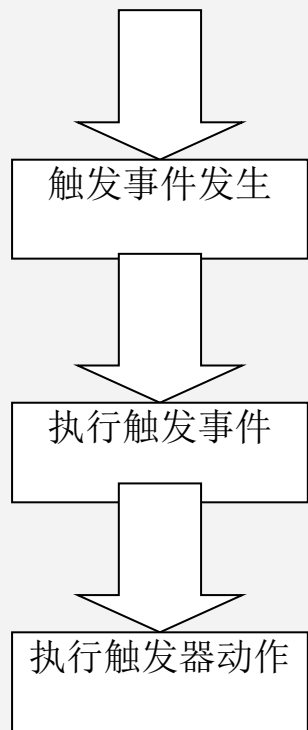
- ▶ **Trigger**
- ▶ 扳机
- ▶ 与枪支的操作原理类似，只有扣动扳机，子弹就会飞出。
- ▶ 触发事件： **INSERT、UPDATE、DELETE**
- ▶ 触发时间： **BEFORE、AFTER**

触发器 (Trigger)

- 触发器是数据库服务器中发生事件时自动执行的一种特殊的存储过程，为数据库提供了有效的监控和处理机制，确保了数据的完整性。
- 定义后，任何用户对表操作均由服务器自动激活相应的触发器，不能直接调用，更不允许设置参数和返回值。

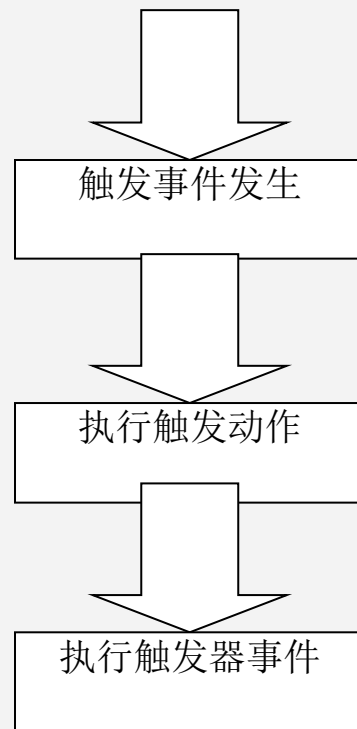
触发器结构

AFTER 条件



AFTER条件触发器执行过程示意图

BEFORE条件



BEFORE条件触发器执行过程示意图

触发器作用

- 安全性
- 审计：能够跟踪用户对数据库的操作
- 实现复杂的数据完整性规则
- 同步实时地复制表中的数据
- 主动计算数据值



触发器语法三要素

- 地点(table)
- 事件(insert/update/delete)
- 触发时间(after/before)



语法

```
create trigger triggerName  
after/before insert/update/delete on 表名  
for each row #这句话在mysql是固定的  
begin  
    sql语句;  
end
```

语法

在 **INSERT** 型触发器中:**NEW** 用来表示将要 (**BEFORE**) 或已经 (**AFTER**) 插入的新数据;

在 **UPDATE** 型触发器中:**OLD** 用来表示将要或已经被修改的原数据, **NEW** 用来表示将要或已经修改为的新数据;

在 **DELETE** 型触发器中:**OLD** 用来表示将要或已经被删除的原数据;

使用方法: **NEW.columnName** (**columnName** 为相应数据表某一列名)

另外, **OLD** 是只读的, 而 **NEW** 则可以在触发器中使用 **SET** 赋值, 这样不会再次触发触发器, 造成循环调用。

1. 创建简单的触发器

例：在学生表 S 上创建触发器，当新生入学时，触发器向客户端发出提示信息

```
CREATE TRIGGER tri_s
```

```
AFTER INSERT
```

```
ON s
```

```
FOR EACH ROW
```

```
BEGIN
```

```
SET @str=concat(new.sname,'农林大学欢迎你! ');
```

```
SIGNAL SQLSTATE '45000'
```

```
SET MESSAGE_TEXT= @Str;
```

```
END
```

2. 通过触发器实现业务规则

例： 在选课表SC上创建触发器，当向该表中插入记录时，若学生选课门数超过 5 门时，将插入记录自动删除。(限选5门)

3. 通过触发器实现业务规则

例： 在选课表SC上创建触发器，当向该表中插入记录时，学生成绩在[0,100]



4. 通过触发器实现业务规则

例：当操作者对sc表的score进行update操作时，触发器会在UPDATE 操作之后，将操作记录写入

`audit_log(logid,snum,cnum,old_score,new_score,updateby,updateon)`

`CREATE TABLE audit_log`

`(logid int primary key auto_increment,`

`snum char(12),`

`cnum char(12),`

`old_score smallint unsigned,`

`new_score smallint unsigned,`

`updateby varchar(50),`

`updateon datetime`

`)`

5. 通过触发器实现业务规则(级联删除)

创建一触发器,在删除学生记录的同时删除他所有的选课记录



6. 通过触发器实现业务规则

创建一触发器,在更新教师的职称时, 教授不能降级



思考

- ▶ Coll表, d表,t表,tc表, s表,sc表如何实现级联删除
- ▶ 创建一触发器tric1,在向C表插入课程记录时,先检查是否与该课程同名的课程已经存在,以避免课程的混淆



浙江农林大学
ZHEJIANG A&F UNIVERSITY

数据库原理 与技术

信息工程 刘丽娟