# 一. mysql 子查询

出现在其他语句内的 select 字句

1. 子查询嵌套在查询内部，且必须出现在圆括号内

2. 子查询可以包含多个关键字或条件：DISTINCT、GROUP BY、ORDER BY、LIMIT、函数等

3. 子查询的外层查询可以是：SELECT、INSERT、UPDATE、SET 或 DO

**使用比较运算符的子查询**

mysql> select round(avg(goods\_price),2) from tdb\_goods;

mysql> select \* from tdb\_goods where goods\_price > (select round(avg(goods\_price),2) from tdb\_goods);

子查询返回多个结果时可用 ANY、SOME、ALL 进行修饰



mysql> select \* from tdb\_goods where goods\_price > all (select goods\_price from tdb\_goods where goods\_cate='超级本');

**使用 [NOT]IN 的子查询**

= ANY 与 IN 等效，!= ALL 与 NOT IN 等效

mysql> select \* from tdb\_goods where goods\_price not in (select goods\_price from tdb\_goods where goods\_cate='超级本');

**使用 [NOT]EXISTS 的子查询**

如果子查询返回任何行，EXISTS 将返回 TRUE，否则 FALSE

使用 INSERT ... SELECT 插入

mysql> insert into tdb\_goods\_cates(cate\_name) select goods\_cate from tdb\_goods group by goods\_cate;

# 二. 多表更新

**INSERT + SELECT**

mysql> update tdb\_goods g inner join tdb\_goods\_cates gc on g.goods\_cate = gc.cate\_name set g.goods\_cate = gc.cate\_id;

**CREATE + SELECT**

mysql> create table tdb\_goods\_brands (

-> brand\_id smallint unsigned primary key auto\_increment,

-> brand\_name varchar(40) not null)

-> select brand\_name from tdb\_goods group by brand\_name;

mysql> update tdb\_goods g inner join tdb\_goods\_brands gb on g.brand\_name = gb.brand\_name set g.brand\_name = gb.brand\_id;

修改表结构使用事实外键（而不是物理外键）

mysql> alter table tdb\_goods change goods\_cate cate\_id smallint unsigned not null,

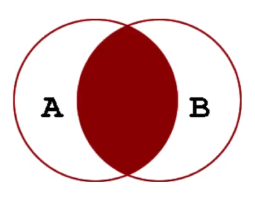
-> change brand\_name brand\_id smallint unsigned not null;

# 三. 连接

通常使用 ON 来设定连接条件（也可以用 WHERE），用 WHERE 对结果集进行过滤

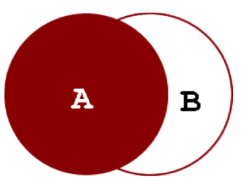
**1. INNER JOIN**（CROSS JOIN、JOIN）

显示左表及右表符合连接条件的记录



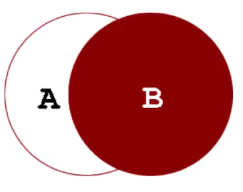
**2. LEFT** [OUTER] **JOIN**

显示左表的全部记录及右表符合连接条件的记录



**3. RIGHT** [OUTER] **JOIN**

显示右表的全部记录及左表符合连接条件的记录



mysql> select goods\_id,goods\_name,cate\_name,brand\_name,goods\_price from tdb\_goods as g

-> inner join tdb\_goods\_cates c on g.cate\_id = c.cate\_id

-> inner join tdb\_goods\_brands b on g.brand\_id = b.brand\_id

A left join B on .... where ...

1. 数据表 B 的结果集依赖数据表 A ( on ... 条件)

2. 数据表 A 的结果集根据左连接条件依赖所有除 B 表以外的数据表 ...???

3. 左外连接条件决定如何检索数据表 B

4. 如果数据表 A 的某一条记录符合 WHERE 条件，但在数据表 B 中不存在符合连接条件的记录，将额外生成一个从 B 表获取字段为 NULL 的记录

如果内连接查找的记录在数据表中不存在，并且 WHERE 字句尝试以下操作：col\_name IS NULL ，如果 col\_name 被定义为 NOT NULL，MySQL 将在找到符合连接条件的记录后停止搜索更多行。

**无限级分类表设计（自身连接）**

+--------------+-------------------+-----------------+

| type\_id | type\_name | parent\_id |

+--------------+-------------------+-----------------+

| 1 | 家用电器 | 0 |

| 2 | 电脑、办公 | 0 |

| 3 | 大家电 | 1 |

| 4 | 生活电器 | 1 |

| 5 | 平板电视 | 3 |

| 6 | 空调 | 3 |

| 7 | 电风扇 | 4 |

| 8 | 饮水机 | 4 |

| 9 | 电脑整机 | 2 |

| 10 | 电脑配件 | 2 |

| 11 | 笔记本 | 9 |

| 12 | 超级本 | 9 |

| 13 | 游戏本 | 9 |

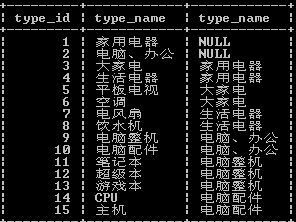
| 14 | CPU | 10 |

| 15 | 主机 | 10 |

+--------------+-------------------+-----------------+

mysql> select s.type\_id,s.type\_name,p.type\_name from tdb\_goods\_types s left join tdb\_goods\_types p

-> on s.parent\_id = p.type\_id;



**多表删除**

mysql> delete t1 from tdb\_goods t1 left join (select goods\_id,goods\_name from tdb\_goods group by goods\_name having count(goods\_name) >= 2) t2 on t1.goods\_name = t2.goods\_name where t1.goods\_id > t2.goods\_id;

# 四. 函数

1. **字符函数**



mysql> select concat('imooc',' ','MySql');

-> imooc MySql

mysql> select concat\_ws('-','a','b','c');

-> a-b-c

mysql> select format(1256.74,4);

-> 1256.7400

mysql> select lower('MYsql') low,upper('loword') up;

mysql> select upper(concat(left('mysql',2),'-',right('mysql',3)));

-> MY-SQL



mysql> select concat(ltrim(' \*mysql' ),rtrim(' - mysql\* '));

-> \*mysql - mysql\*

mysql> select trim(leading '?' from '??mysql ?');

-> mysql ? // leading、trailing、both用于删除起始、末尾、中间的指定字符

mysql> select replace('??my??sql?','?','');

-> mysql

mysql> select substring('mysql',1,2);

-> my //下标从 1 开始

mysql> select substring('mysql',-2);

-> ql //倒数第二个取到结尾

mysql> select 'tom%' like '%1%%' escape '1';

-> 1 // escape 表示紧跟在 1 后面的字符不当成通配符

**%** 代表任意字符（字符串），**\_** 代表任意一个字符

2. **数值运算符和函数**



mysql> select ceil(3.01);

-> 4 //向上取整，floor() 为向下取整

div 整除，mod 求余数

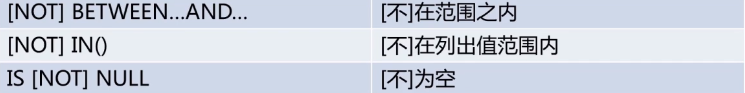
3 div 4 = 0，5 mod 3 = 2

power(3,3) = 27

round(3.652,2) = 3.65

truncate(125.89,1) = 125.8 // 直接把数值去掉，而不是四舍五入

3. **比较运算符与函数**



mysql> select 15 between 1 and 22;

-> 1 // true

mysql> select 10 not in(5,2,3,4,10);

-> 0 // false

mysql> select '' or 0 is null;

-> 0

mysql> select null is not null;

-> 0

4. **日期时间函数**



mysql> select date\_add('2014-03-12',interval 365 day);

-> 2015-03-12

mysql> select date\_add('2014-03-12',interval 5 week); // -5 则减少

-> 2014-04-16

mysql> select datediff('2013-3-12','2013-3-8');

-> 4

mysql> select date\_format('2012-3-14','%m/%d/%Y');

-> 03/14/2012

5. **信息函数**



mysql> select connection\_id();

-> 48 // 线程 id

mysql> insert into tdb\_goods\_cates values(default,'cate name'); // 新纪录 id 为 8

mysql> select distinct(last\_insert\_id()) from tdb\_goods\_cates; // 必须存在一个自增的主键

-> 8

6. **聚合函数**



mysql> select brand\_id,avg(goods\_price) avg from tdb\_goods group by brand\_id;

mysql> select max(goods\_price) from tdb\_goods;

mysql> select min(goods\_price) from tdb\_goods;

mysql> select sum(goods\_price) from tdb\_goods;

7. **加密函数**



# 五. 自定义函数

1. 用户自定义函数（user-defined function, UDF），是一种对 MySQL 扩展的途径，其用法与内置函数相同

2. 函数可以返回任意类型的值，同样可以接收这些类型的参数（参数最多为1024）

CREATE FUNCTION function\_name

RETURNS

{STRING | INTEGER | REAL | DECIMAL}

routine\_body

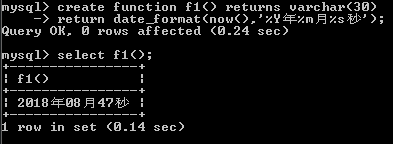
1. 函数体由合法 sql 语句构成

2. 函数体可以是简单的 select 或 insert

3. 函数体如果为复合结构则使用 begin...end

4. 复合结构可以包含声明，循环，控制结构

不带参数的函数

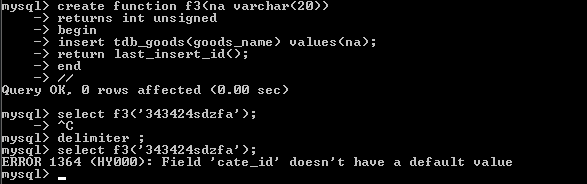


带参数的函数

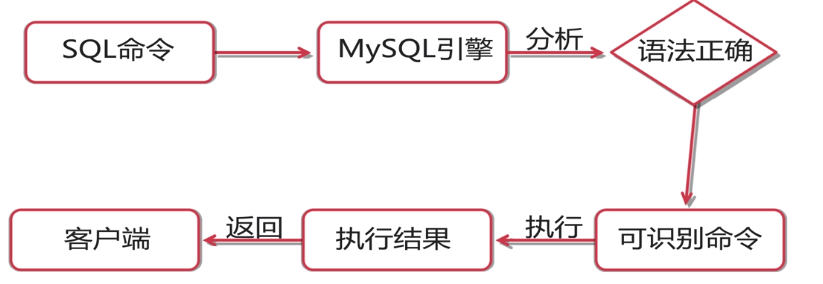


复合结构函数

delimiter // 创建复合结构函数前把终止符修改成 // ，避免与 ; 冲突



# 六. 存储过程

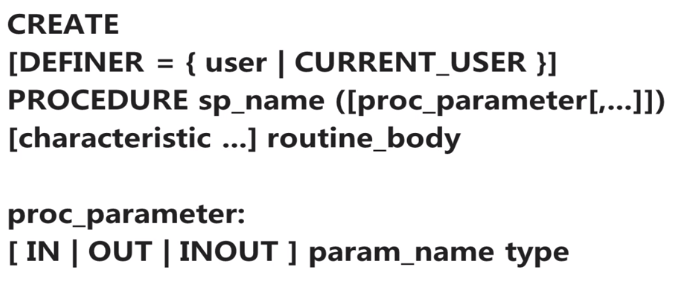


存储过程是 SQL 语句和控制语句的预编译集合，以一个名称存储并作为一个处理单元。

1. 增强 SQL 语句的功能和灵活性

2. 较快的执行速度（预编译：第一次调用时进行编译，之后无效语法分析和编译）

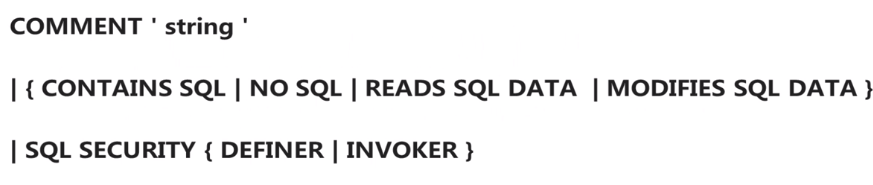
3. 减少网络流量

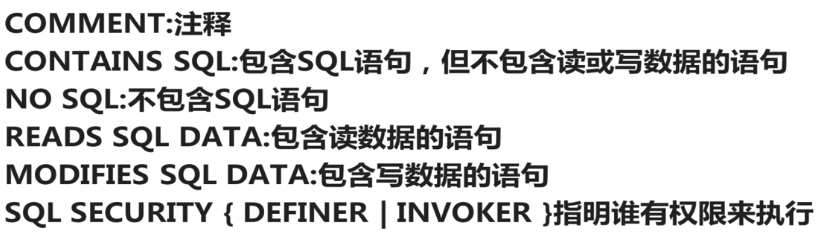


IN：该参数的值必须在调用存储过程时指定

OUT：表示该参数的值可以被存储过程改变，并且可以返回

INOUT：表示该参数在调用时指定，并且可以被改变和返回





**过程体**

1. 合法的 SQL 语句

2. 任意 SQL 语句（除去表的创建，数据库创建等）

3. 为复合结构用 BEGIN ... END

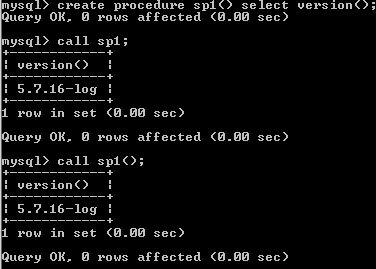
4. 复合结构可以声明变量，循环，控制结构

**调用存储过程**

CALL sp\_name([parameter[,...]])

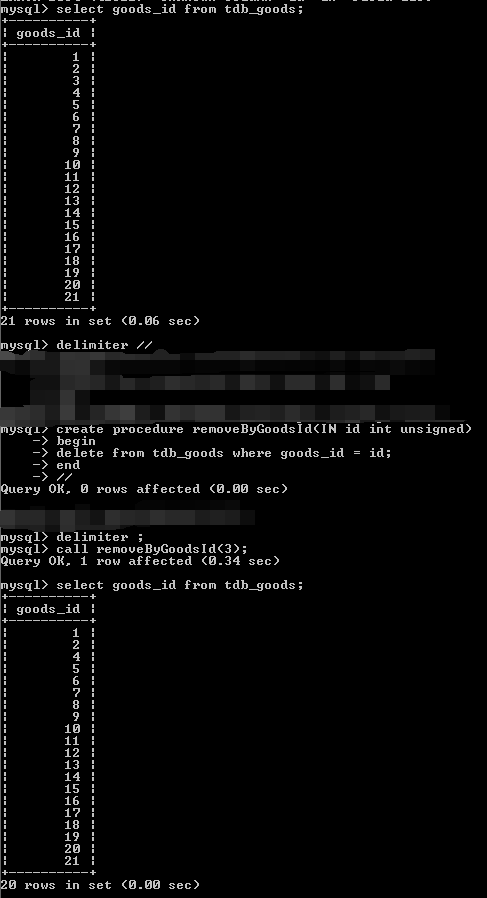
CALL sp\_name[()] 无参数小括号可省略

**无参数存储过程**

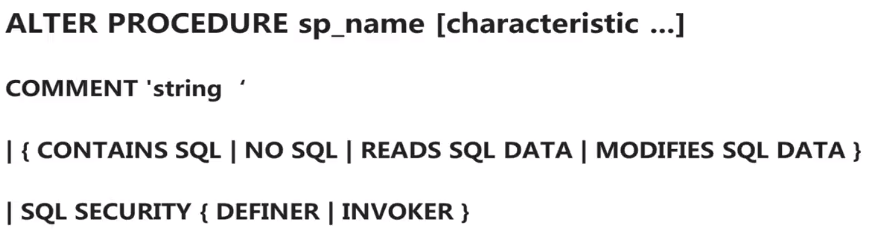


**IN 类型存储过程**

注意参数名不能与使用到的字段名相同



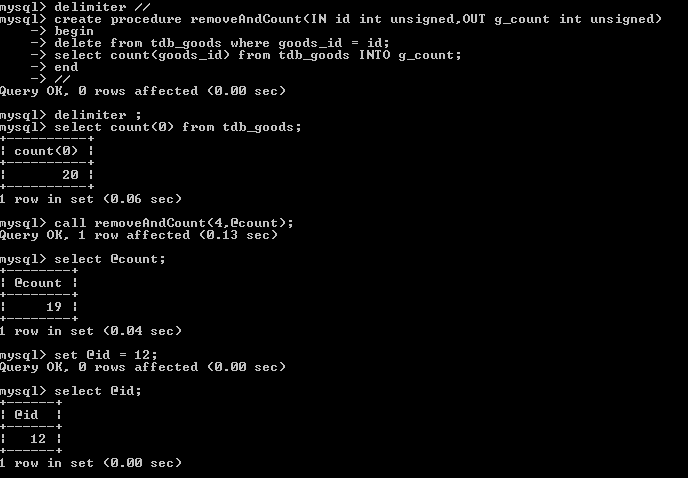
**修改存储过程**



**删除存储过程**

DROP PROCEDURE [IF EXISTS] sp\_name

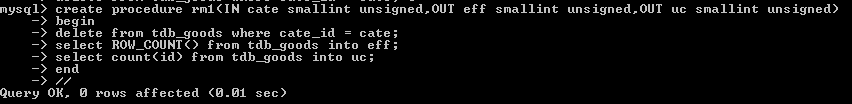
**IN 和 OUT 类型参数的存储过程**



begin .. end 中间用 declare 声明变量，属于局部变量，只在 begin ... end 之间有效，declare 必须位于第一行。

set @id 称为用户变量，只对当前用户的当前客户端有效。

**多个 OUT 参数存储过程**

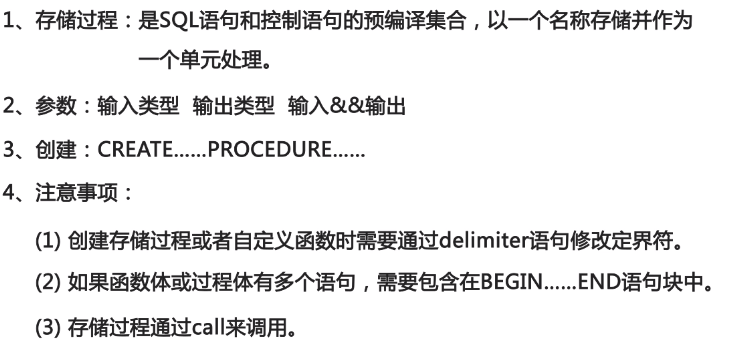


**存储过程与自定义函数的区别**

1. 存储过程实现的功能要复杂一些，而函数针对性更强

2. 存储过程可以返回多个值，函数只能返回一个值

3. 存储过程一般独立执行，而函数可以作为其他 SQL 语句的组成部分出现



# 七. MySQL 存储引擎

**MySQL 支持的存储引擎**

1. MyISAM

2. InnoDB

3. Memory

4.CSV

5. Archive

**并发控制**

当多个连接对记录进行修改时保证数据的一致性和完整性

1. 共享锁（读锁）：同一时间段内，多个用户可以安全的读取同一资源

2. 排他锁（写锁）：任何时候只能有一个用户写入资源，进行写锁时会阻塞其他的读锁或写锁操作

**锁粒度**

1. 表锁：开销最小

2. 行锁：开销最大

**事务（原子性、一致性、隔离性、持久性）**

保证数据库完整性

**外键和索引**

1. 外键：保证数据一致性

2. 索引：对数据表中一列或多列进行排序的结构，目录

各种存储引擎的特点

