## mysql\_unbuffered\_query()

mysql\_query()函数查询的方式是将结果全部查询出来，缓存到内存当中，但当结果集非常庞大时，就会超内存。

mysql\_unbuffered\_query()不会缓存结果集，而是查询出来数据后立马对结果集进行操作，边查询边返回。不会有内存的限制；

但是，使用mysql\_unbufferde\_query()时不能使用mysql\_num\_rows(),且向MySQL发送一条新的SQL查询之前，必须提取掉所有未缓存的 SQL 查询所产生的结果行。

PDO:

try{  
 $pdo = new PDO('mysql:host=localhost;dbname=test','root','root',array(PDO::*ATTR\_ERRMODE*=>PDO::*ERRMODE\_EXCEPTION*)) ;  
 $pdo->query('set names utf8');  
 $pdo->setAttribute(PDO::*MYSQL\_ATTR\_USE\_BUFFERED\_QUERY*,false);  
 $statement = $pdo->query("select goods\_name from goods limit 20");  
 echo '内存大小：'.*memory\_get\_usage*().'<br>';  
  
 while($row = $statement->fetch(PDO::*FETCH\_ASSOC*)){  
 *print\_r*($row);  
 echo "<br>";  
 echo '内存大小；'.*memory\_get\_usage*().'<br>';  
 }  
}catch(Exception $e){  
 echo $e->getMessage();  
}

//注意：dbname=后面不能有空格，否则读取不到数据库

//memory\_get\_usage():返回php脚本所占的内存



内存占用率很低。

## sql语句查询表的字段名

$sql = “desc users”;// desc 表名

$res = mysql\_query($sql);

$row = mysql\_fetch\_assoc($res)

echo $row[‘Field’];//Field必须大写，得到字段名

## 特殊的sql语句

IF：

例：update `group` set `group\_count`=if(`group\_count`>1,`group\_count`-1,0) where `id`=1;

解释：if($parm1,$parm2,$parm3)

$parm1是条件，若$parm1为真，则执行$parm2，假则执行$parm3.

条件里用‘=’表示相等，没有==一说

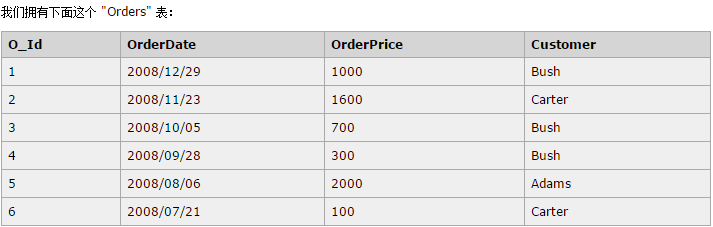
IFNULL(exp1,exp2)

若exp1为null，返回exp2的值；否则还是返回exp1。

Order by：order by 多个字段，每个字段后面都有排序方式，默认ASC

例如：select table a order by a.time1 ,a.time2 desc,a.time3 asc

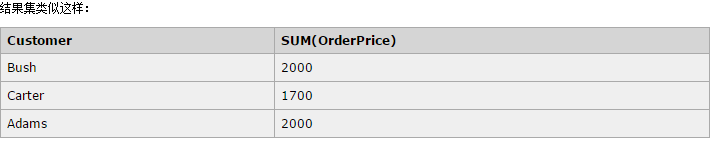
Group by:



现在，我们希望查找每个客户的总金额（总订单）。

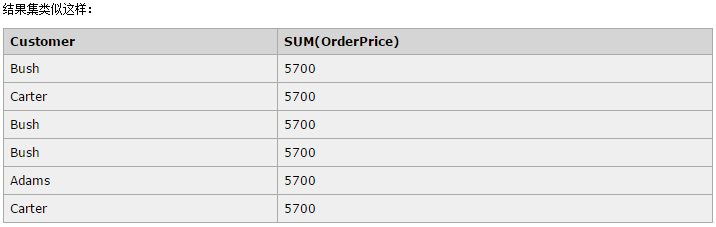
我们使用下列 SQL 语句：

SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer



当省略 group by时：

SELECT Customer,SUM(OrderPrice) FROM Orders



show table status from 数据库 like 表名；

包含信息：

array (

'Name' => 'message',

'Engine' => 'InnoDB',

'Version' => '10',

'Row\_format' => 'Compact',

'Rows' => '6355',

'Avg\_row\_length' => '41',

'Data\_length' => '262144',

'Max\_data\_length' => '0',

'Index\_length' => '0',

'Data\_free' => '7340032',

'Auto\_increment' => '6384',

'Create\_time' => '2017-05-12 09:20:34',

'Update\_time' => NULL,

'Check\_time' => NULL,

'Collation' => 'utf8\_general\_ci',

'Checksum' => NULL,

'Create\_options' => '',

'Comment' => '',

)

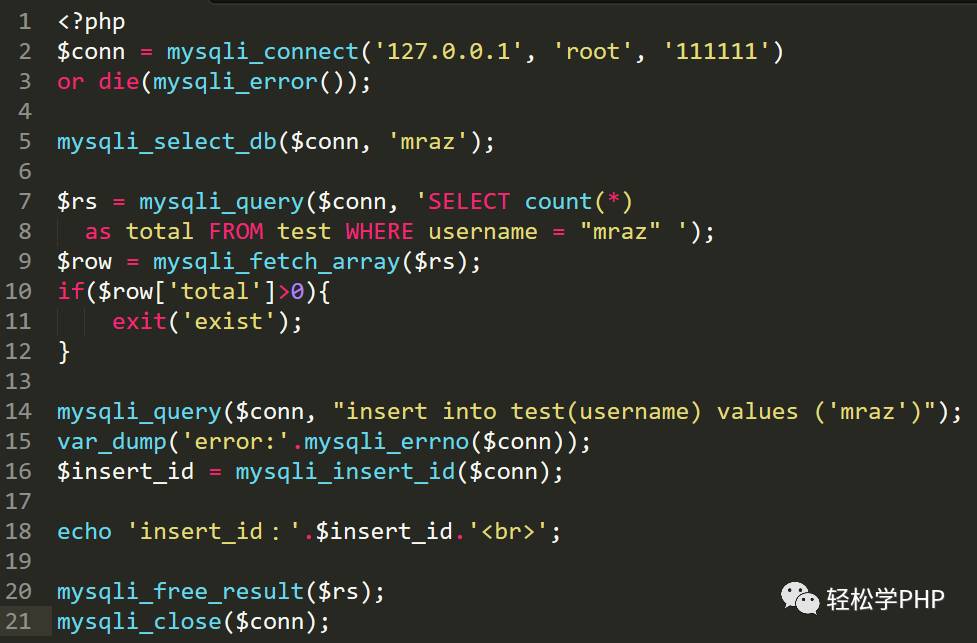
## SQL语句反引号``和单引号’’的作用

反引号`` ：**为了区分MYSQL的保留字与普通字符而引入的符号。**

举个例子：SELECT `select` FROM `test` WHERE select='字段值'

单引号’’： **字段值是字符或字符串，则要加引号**

## 高并发请求，解决插入多条重复记录



少量请求不会有问题，但是一旦高并发请求时，会插入多条username为mraz的记录；

解决：

利用Mysql的 FOR UPDATE语句和事务的隔离性，且FOR UPDATE仅适用于innoDB，必须在事物中才生效；



## Mysql锁机制

以MyISAM引擎为例：

有两种模式的表级锁：表共享读锁，表独占写锁。

对MyISAM表的读操作，不会阻塞其他用户对同一表的读请求，但会阻塞对同一表的写请求；

对MyISAM表的写操作，则会阻塞其他用户对同一表的读和写操作；

加锁：

MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，在执行更新操作（UPDATE、DELETE、INSERT等）前，会自动给涉及的表加写锁，这个过程并不需要用户干预

行级锁以InnoDB引擎为例：

InnoDB的行级锁定同样分为两种类型，共享锁和排他锁，而在锁定机制的实现过程中为了让行级锁定和表级锁定共存，InnoDB也同样使用了意向锁（表级锁定）的概念，也就有了意向共享锁和意向排他锁这两种。

共享锁、排它锁是行级锁定；

意向共享锁、意向排他锁是表级锁定



意向锁（表级锁定）是InnoDB自动加的，不需用户干预。对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁；事务可以通过以下语句显示给记录集加共享锁或排他锁。

共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE

排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE

//for update只能用于innodb，且只能在事务中使用；

由于InnoDB预设是Row-Level Lock，所以只有「明确」的指定主键，MySQL才会执行Row lock (只锁住被选取的资料例) ，否则MySQL将会执行Table Lock (将整个资料表单给锁住)。

InnoDB行锁是通过给索引上的索引项加锁来实现的，只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁

原理：假如一个表里还没任何锁，第一个人去给它加了行锁（排它锁），首先要先把表锁起来，给表加意向排它锁（innodb自己实现的），第二个人任何锁都获取不到了，因为整个表都被锁了。

若第一个人

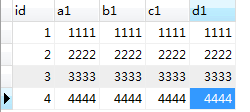
## 乐观锁

给数据加个version字段，读取数据时吧version读出来，更新数据时，要判断version跟读出来的version是否相同，相同则执行增删改操作，并且把version+1；若version跟读时的不同，则事务回滚。

## Mysql存在物理外键的表如何清空

|  |
| --- |
| 1. **SET** FOREIGN\_KEY\_CHECKS = 0; 2. **TRUNCATE** **TABLE** TABLE\_E; 3. **SET** FOREIGN\_KEY\_CHECKS = 1; |

## 将一个数据表的数据导出到另个表，且字段名不相同

Test1: 

Test2: 

|  |
| --- |
| $sql = "insert into yd\_test2 (`a2`,`b2`,`c2`,`d2`) select `a1`,`b1`,`c1`,`d1` from yd\_test1"; |

将test1中的a1,b1,c1,d1字段，对应test2表中a2,b2,c2,d2分别导入，使用一下sql语句：

## Sql语句 in(1,9,5,7,2,4,8)搜索结果如何按in里面的排序

Sql语句 in()查询到的结果，默认是按in里面数字升序排列的，若想要实现结果也是按in里的顺序

可以：

Select \* from goods where id in (2,5,7,1,4,3) order by field(id, 2,5,7,1,4,3)

## distinct()

1. 位置

单独用必须放在字段开头，否则报错

与其他函数配合时，没位置限制

2）用法

a.

在count计算不重复的记录的时候

SELECT COUNT( DISTINCT player\_id ) FROM task;

b.

在需要返回不重复的某个字段时；

SELECT DISTINCT player\_id FROM task;

c.

当distinct后有多个字段

SELECT DISTINCT player\_id, task\_id FROM task;

指的是查询player\_id和task\_id组合起来不同的记录

## concat()

语法：concat（列1，列2，列3）

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Age | gender |
| 1 | A | 1 | 0 |
| 2 | B | 2 | 1 |
| 3 | C | 1 | 0 |

Select id,concat(name,age,gender) as info from user

结果：

|  |  |
| --- | --- |
| Id | Info |
| 1 | A10 |
| 2 | B21 |
| 3 | C10 |

默认情况下，不做说明时，连接体之间没有连接符

使用concat\_ws(separator,列1，列2…)//第一个参数是分隔符

Select id,concat\_ws(‘-’,name,age,gender) as info from user

结果：

|  |  |
| --- | --- |
| Id | Info |
| 1 | A-1-0 |
| 2 | B-2-1 |
| 3 | C-1-0 |

举例：select concat()

## group\_concat()

mysql> select \* from aa;

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

| pid| name |

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

|1 | 10|

|1 | 20|

|1 | 30|

|2 | 20|

|3 | 200 |

|3 | 500 |

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

现在讲pid为1的分为一组，讲他们的name值用“，”拼接起来：

Select pid ,group\_concat(name) from aa group by pid

完整语法：**group\_concat([DISTINCT] 要连接的字段 [Order BY 排序字段 ASC/DESC] [Separator '分隔符'])**

## union和union all

(SELECT column\_name FROM table1)

UNION

(SELECT column\_name FROM table2)

Union:将所有查询结果集合并，去重，

Union all:将所有查询结果集合并，不去重；

合并的几个结果，必须有相同的列数，列的顺序必须相同

应用：整站搜索问题，将不同表搜出来的信息合并。

## 子查询

* 1. 子查询只有一条记录

select … from 表1 where 列1 >(子查询)

* 1. 子查询返回多条记录

SELECT \* FROM student WHERE student\_id **IN** (SELECT student\_id FROM absence);

注意：这里子查询里的搜索列需要跟where语句的搜索条件列一致。

只要父查询里的数据行与子查询里的数据行任何一行匹配，就返回。

**NOT IN**用法类似；

## 触发器

触发器是一类特殊的事务，可以监视某种数据操作（增、删、改），并触发相关的操作（增、删、改）

触发器的四要素：

监视对象，触发时间，监视动作，触发事件

delimiter $ 将结束符换成$,免得跟触发sql冲突

|  |
| --- |
| create trigger t1  after  insert  on ord  for each row  begin  update goods set num = num-new.gnum where gid=new.gid;  end$ |

|  |
| --- |
| create trigger t2  after  delete  on ord  for each row  begin  update goods set num = num+old.gnum where gid = old.gid;  end$ |

新增时，触发的sql中用new代表新数据，删除时，用old代表删除的数据；

查看已有trigger：show triggers

删除trigger：drop trigger 触发器名

更新：

|  |
| --- |
| create trigger t3  after  update  on ord  for each row  begin  update goods set num = num +old.gnum-new.gnum where gid=new.gid;  end$ |

|  |
| --- |
| create trigger t5  before  insert  on ord  for each row  begin  declare  rnum int; //定义一个变量  select num into rnum from goods where gid=new.gid;//先查出库存，赋给变量  if new.gnum > rnum then  set new.gnum = rnum;  end if;  update goods set num=num-new.gnum where gid=new.gid;  end$ |

此处的触发事件必须要在insert之前，因为插入之后就起不到作用了。

关于new和old的使用：

insert时，新插入的为new；

new

old

无

new

old

无

delete时，原来已存在的数据为old；

update时，数据是从old到new

for each row :行级触发器

当你一次改变多行数据时，会对应触发多次触发器；

在oracle中，若不写这句，无论一次改变多少行，最终都只会触发一次触发器。但是mysql中只能用for each row；

## 存储过程

封装一些sql语句

创建：

create procedure p1()

begin

select gid from goods where gid=1;

end$

查看已有的存储过程：

show procedure status;

调用存储过程：

call p1()

在存储过程中用declare来声明变量：

格式：declare 变量名 变量类型 [default 默认值]

|  |
| --- |
| create procedure p2()  begin  declare age int default 18;  declare gender char default "female";  select concat('年龄是',age,'性别是',gender);  end$ |

存储过程中运算：

set 变量 = expression；

|  |
| --- |
| create procedure p3()  begin  declare age int default 18;  set age=age+20;  select concat(‘20年后年龄是‘,age);  end$ |

if/else控制结构：

if condition then

else

end if； 分号不要忘

|  |
| --- |
| create procedure p4()  begin  declare age int default 18;  if age=18 then  select '已成年';  else  select 'sjdif';  end if;  end$ |

给存储过程传参：

语法：[in/out/inout] 参数名 参数类型

|  |
| --- |
| create procedure p5(width int,height int)  begin  select concat('你的面积是：',width \* height) as area;  if width>height then  select '你很胖';  elseif width<height then  select '你很瘦';  else  select '你刚刚好';  end if;  end$ |

while/do 流程

|  |
| --- |
| create procedure p6()  begin  declare total int default 0;  declare i int default 1;  while i<=100 do  set total: = total+i; //sql中比较也是用=，所以赋值时最好用:=,以示区分；  set i: = i+1;  end while;  select total;  end$ |

参数的in指需要传入的参数，out指输出的变量。

|  |
| --- |
| create procedure p8(in num int, out sum int)  begin  declare i int default 1;  set sum := 0;  while i<=num do  set sum :=sum+i;  set i :=i+1;  end while;  end$ |

call p8(10,@sum) //调用时第二个参数是可以自定义的，前面加@。

inout类型：

|  |
| --- |
| create procedure p9(inout age int)  begin  set age := age+20;  end$ |

调用时：

|  |
| --- |
| set @age ;=18$ 先定义一个变量  call p9(@age)$ 将变量传入  select @age$ 输出显示 |

case流程 ：（类似于PHP的switch）

|  |
| --- |
| create procedure p10(in age int)  begin  declare descri char(10);  case age  when 10 then set descri :='你十岁了';  when 20 then set descri :='你二十岁了';  when 30 then set descri :='你三十岁了';  else set descri :='你太老了';  end case;  select descri;  end$ |

repeat流程：

|  |
| --- |
| create procedure p11(in n int)  begin  set @x :=0;  set @total :=0;  repeat  set @total :=@x+@total ;  set @x:=@x+1;  until @x>n end repeat;  select @total;  end$ |

cursor游标：逐行读取结果集，内部有指针，每读取一行指针后移一位

定义游标：declare 游标名 cursor for select …

打开游标： open 游标名

游标取值：fetch 游标名 into var1,var2,var3…

关闭游标：close 游标名

//所有的declare必须在其他语句前面，否则出错；

|  |
| --- |
| create procedure p13()  begin  declare row\_gid int;  declare row\_good\_name varchar(20);  declare row\_num int ;  declare i int default 0;  declare con int; 总行数  declare getgoods cursor for select gid,good\_name,num from goods;  select count(\*) into con from goods;  open getgoods;  repeat  fetch getgoods into row\_gid,row\_good\_name,row\_num;  select row\_gid,row\_good\_name,row\_num;  set i :=i+1;  until i>=con end repeat;  close getgoods;  end$ |