昨日回顾 2

数据操作语言（ＤＭＬ） 4

数据插入（添加数据） 4

有3种形式 4

解释说明： 4

其他类似插入数据的语句： 4

删除数据 5

基本语法形式： 5

解释说明： 5

类似删除语句truncate： 5

修改数据 6

基本语法： 6

语法说明 6

一个问题：复制表 6

数据查询语言DQL 7

基本查询 7

语法形式 7

解释说明： 7

[all | distinct] 9

from子句 9

where子句 9

group by 分组子句 14

having子句 15

昨日回顾

[数据类型](#Toc27703)

[概览](#Toc12902)：分

数字类型

整数： tinyint, smallint, mediumint, int, bigint

小数:

字符串类型：

时间类型

[整数类型](#Toc17057)

[范围](#Toc5129)：各不一样

[通用设定形式](#Toc23097)： 类型名 [（显示长度）] [unsigned] [zerofill]

[小数类型](#Toc22116)

浮点小数：非精确小数，包括：float， double

定点小数：精确小数，decimal

[时间日期类型](#Toc15326)

date， time。 datetime， year，

timestamp：时间戳，是一个在insert或update的时候能自动获得时间的时间类型名

[字符串类型](#Toc27372)

[最基本最重要的2个：](#Toc14083)

varchar(长度）：

char（长度）：

[2个二进制文本：](#Toc8169)

binary：类似char，存储字符的二进制字节而已

varbinary：类似varchar，存储字符的二进制字节而已

[2个大文本类型：](#Toc13030)text（用于存储超长文本），blob（常用于存储图片这种二进制数据）

[2个有关“选项”的文本存储形式：](#Toc12631)

enum(‘aa’,’bb’,’cc’,’ddd’。。。)，实际对应的数字为：1,2,3,4,5,......最大6万多

set(‘aa’,’bb’,’cc’,’ddd’。。。)，实际对应的数字为：1,2,4,8，16,......最大64

[表定义语句](#Toc28284)

[创建表：](#Toc12691)

[基本形式](#Toc19689)：

create table [ if not exists ] 表名(字段列表，索引或约束列表)表选项列表

字段的形式：字段名 类型 [字段属性列表]

[字段属性设置](#Toc4233)

primary key:既是字段属性，也是约束，还是索引

unique: 既是字段属性，也是约束，还是索引

auto\_increment：

not null：既是字段属性，也是约束

default ‘默认值’：既是字段属性，也是约束

comment ‘注释’

[索引设置](#Toc31347)

普通索引：ｋｅｙ　（字段名１，字段名２，　．．．）

主键索引：ｐｒｉｍａｒｙ　ｋｅｙ　（字段名１，字段名２，　．．．）

唯一索引：ｕｎｉｑｕｅ　ｋｅｙ　（字段名１，字段名２，　．．．）

全文索引：ｆｕｌｌｔｅｘｔ（字段名１，字段名２，　．．．）

[约束设置](#Toc32068)：

主键约束，唯一约束，非空约束，默认值约束，外键约束

外键约束：foreign key (字段名１，字段名２...）references 表2（字段名１，字段名２...)

[表选项](#Toc22099)

engine = “存储引擎”：选择在性能和功能上的最佳结合点。

auto\_increment=起始数字

charset = ‘字符编码名’

comment=’注释’

[修改表：](#Toc32015)

[一般概述](#Toc12245)：可以创建的，都可以修改。

增删改字段，增删约束，增删索引，修改表名

[修改表的基本形式：](#Toc11396)alter table 表名 修改语句1，修改语句2，.....；

[删除表：](#Toc6561)drop table 表名

[表的其他操作：](#Toc578)

show tables； desc 表名； show create table 表名；

create table 表名2 like 表名1；

rename table 表名1 to 表名2；

[视图](#Toc27684)

[什么是视图：](#Toc26656)就是一条预先存储的select语句（并给定一个名字），以供后续“方便查询”

[视图创建形式：](#Toc6259)create view 视图名 [（字段名1，字段名2， ....）] as select 语句；

[视图的使用](#Toc26385)：select 。。。。 from 视图名；

[修改视图：](#Toc14116)

[删除视图：](#Toc17175)

[数据库（数据表）的设计思想介绍](#Toc23267)

[数据库设计3范式（3NF）：](#Toc4096)

[第一范式（１ＮＦ），原子性，数据不可再分](#Toc11480)

[第二范式（２ＮＦ），唯一性，消除部分依赖](#Toc19639)

[第三范式（3NF）：独立性，消除传递依赖](#Toc18710)

遵循一个大原则：一种数据存入一个表中（不要将多种数据混在一个表中）

数据操作语言（ＤＭＬ）

数据插入（添加数据）

有3种形式

形式1：

insert into 表名（字段名1，字段名2，....）values (值a1，值a2， .....)， (值b1，值b2， .....)，..... ；

形式2：

insert into 表名1（字段名1，字段名2，....）select 字段名1，字段名2，.... from 表名2；

形式3：

insert into 表名 set 字段名1=值1，字段名2=值2， ..... ；

解释说明：

1，形式1和形式2，可以一次插入多条数据；

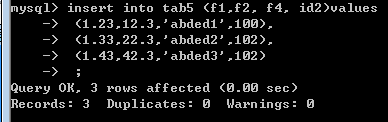
2，不管哪种形式，在“字段”和“值”之间，都有“一一对应”关系。

3，值的形式，通常是这样：数字直接写，字符串和时间加单引号，但如果是函数值，则不能加引号

4，观念问题：不管那种形式，都要理解为：插入数据的单位是“行”；

5，有的字段通常无需插入数据，此时不应该出现该字段名：auto\_increment， timestamp，

举例：

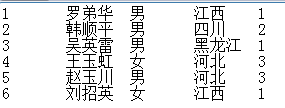


其他类似插入数据的语句：

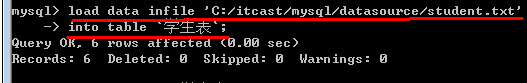
载入外部“形式整齐”的数据：

load data infile ‘文件完整名（含路径）’ into table 表名；

类这样的数据：



载入语法：



复制一个表的结构和数据：

create table 表名1 select \* from 表名2；

删除数据

基本语法形式：

**delete from 表名 [where条件] [order排序] [limit限定]**；

解释说明：

1，观念问题：删除也是应该以理解为“以行为单位”进行的。

2，删除语句中，where条件通常都要写上，因为如果不写，则就删除了所有数据，应用中极少如此。

3，order排序子句和limit限定子句，应用中通常不需要。

3.1 order排序子句用于设定删除数据的先后顺序。

3.2 limit限定子句用于限定在设定的顺序情况下删除指定的某些行。

类似删除语句truncate：

truncate [table] 表名；用于直接删除整个表（结构）并重新创建该表。

1，删除整个表（数据和结构都没有了）

2，重新创建该表（全新表）。

3，跟delete 语句不带where条件，有什么区别？

主要影响的是类似：auto\_increment这种类型的字段值：

truncate结果会重新计算，delete还能继续增长。

举例：



翻译：执行成功，0行受影响(0.00秒)

修改数据

基本语法：

**update 表名 set 字段名1=值表达式1，字段名2=值表达式2，....[where条件] [order排序] [limit限定]**；

语法说明

1， 观念问题：仍然要理解为更新是以“行”为单位进行的，虽然可以指定只更新其中的部分字段。

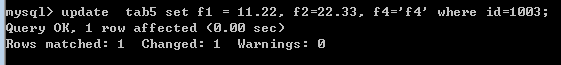
2， where条件子句，在应用中，同样几乎都必须有，否则很可能就失去意义，类似delete

3，order排序子句通常不需要，用于将要更新的数据指定更新的顺序。

4，limit限定子句通常不需要，用于将要更新的数据的指定顺序去更新部分（局部）数据，比如：前500行。

5，字段的值可以是表达式，或直接值，或函数，如果是直接值，同样遵循insert语句中的值的写法。

举例：



一个问题：复制表

复制表方法1：

create table tab2 like tab1; //复制结构了

insert into tab2 select \* from tab1; //复制数据

这种方法可以比较完整。

复制表方法2：

create table tab2 select \* from tab1; //同时复制结构和数据

这种方法可能会丢一些结构信息，比如：索引，约束，自增长属性

数据查询语言DQL

基本查询

语法形式

select [all | distinct] 字段或表达式列表 [from子句] [where子句] [group by子句] [having子句] [order by子句] [limit子句]；

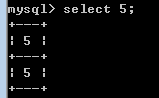
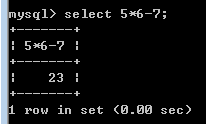
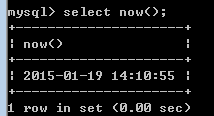
解释说明：

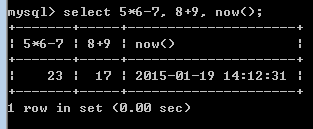
select语句，作用是从“数据源”中，找出（取出）一定的数据，并作为该语句的返回结果（数据集）

数据源：

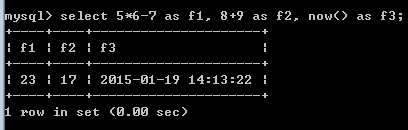
通常，数据源就是“表”。但：

也可以没有数据源，而是使用“直接数据”（或函数执行结果）。

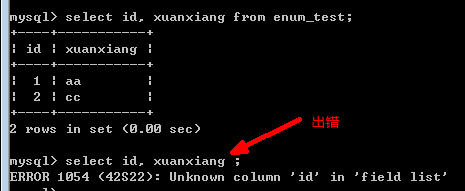
　　　　　



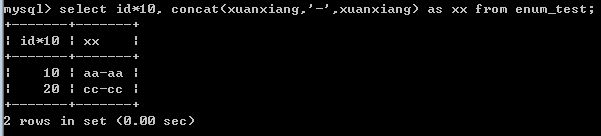
可以给其加上字段（别）名：



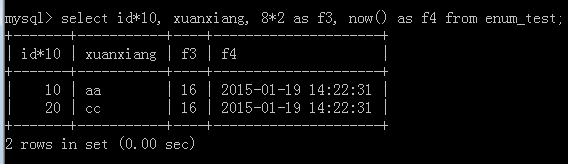
此语法，不可以没有ｆｒｏｍ子句而使用表的字段：



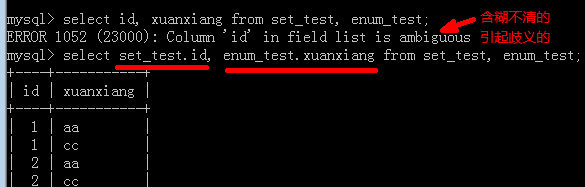
而如果有表（ｆｒｏｍ子句），则对字段同样可以进行“计算”：



还可以对表中的字段和“直接值”（或函数返回值）同时并列“取出”：



也可以使用“表名.字段名”的形式来指定某个表的某个字段（通常用于多表查询）：



[all | distinct]

用于设定所select出来的数据是否允许出现重复行（完全相同的数据行）

all：允许出现——默认不写就是All（允许的）。

distinct：不允许出现——就是所谓的“消除重复行”。

from子句

就是指定数据的来源，其实就是“表”，可以是一个表名，也可以是多个表——多表查询了。

where子句

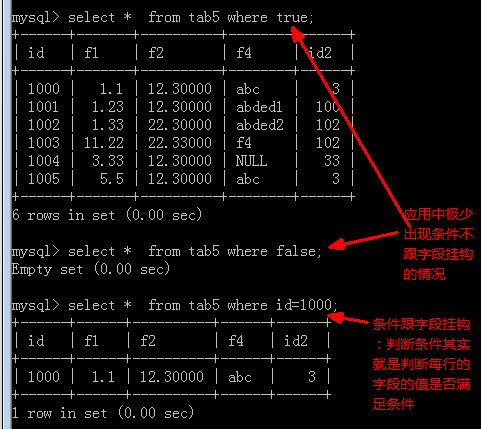
一个概念：where子句，相当于php或js中的if条件语句：其最终结果就是布尔值（true/false）

php：if($n % 4 == 0 && $n % 100 != 0 || $n % 400 == 0 ){}

则：

where true, where 1; where 1=1; 都表示true

where false, where 0; where 1<>1; 都表示false



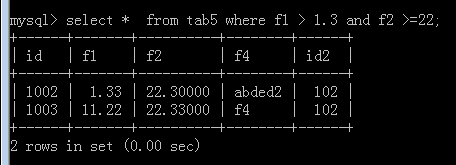
where中可用的运算符：

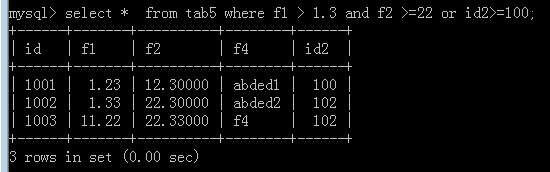
算术运算符： + - \* / %

比较运算符： > >= < <= =（等于） <>（不等于）

==（等于，mysql扩展），!=（不等于，mysql扩展）

逻辑运算符： and（与） or（或） not（非）





布尔值的判断方式：

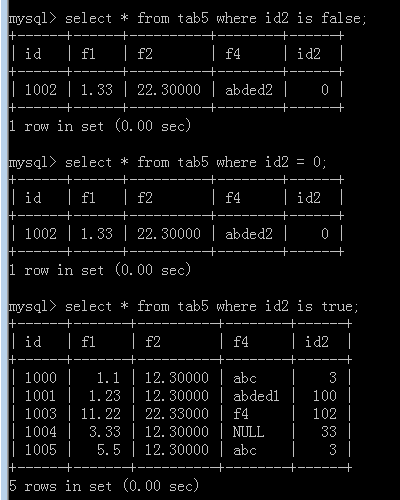
布尔值：本质上，布尔值只是一位整数的“别名”，0表示false，非0表示true。

判断为true： XX is true

判断为fale： XX is false

XX应该是一个字段名，且其类型应该是一个整数。

实际应用中，布尔值判断很少用，因为可以直接使用数学大小。

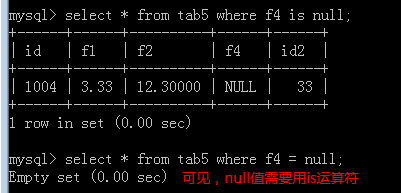


空值的判断方式：

判断为null： XX is null

判断为非空： XX is not null

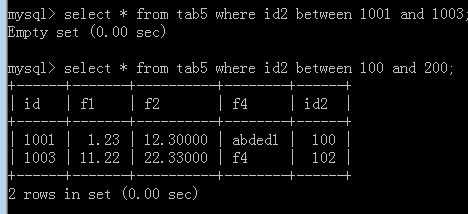
XX应该是一个字段名



between语法：

XX between 值1 and 值2；

含义：字段XX的值在值1和值2之间（含），相当于：XX >=值1 and XX<=值2；



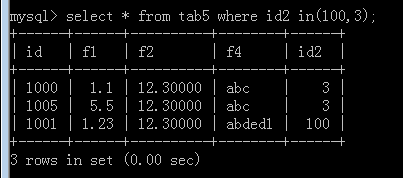
in语法：

XX in (值1，值2，.......)；

含义：XX等于其中所列出的任何一个值都算成立，相当于：

XX = 值1 or XX = 值2 or XX = 值2

注意：其中的值1通常是“直接值”，但也可以是后面要学习的“查询结果值”



ｌｉｋｅ语法（模糊查找）：

语法形式：　ＸＸ　like ‘要查找字符’;

说明：

1，like语法（模糊查找）用于对字符类型的字段进行字符匹配查找

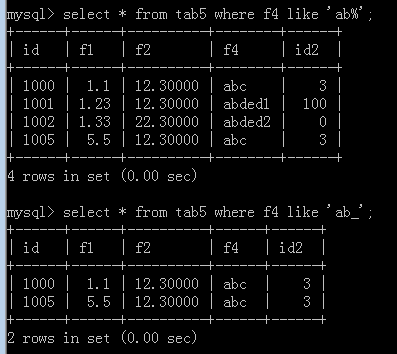
2，要查找的字符中，有2个特殊含义的字符：

2.1: % 其含义是：代表任意个数的任意字符

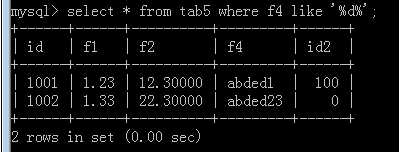
2.2: \_ 其含义是：代表1个的任意字符

2.3：这里的字符，都是指现实中可见的一个“符号”，而不是字节。

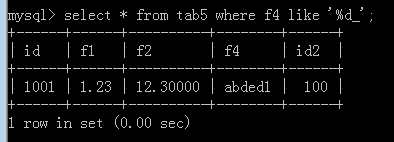
3，实际应用中的模糊查找，通常都是这样：like ‘%关键字%’；



最常见的模糊查询：



还可以进一步结合（配合）使用：



如果要查找的字符中包含“%”或“\_”，“’”，则只要对他们进行转义就可以：

like ‘%ab\%cd%’ //这里要找的是： 包含 ab%cd 字符的字符

like ‘\\_ab%’ //这里要找的是： \_ab开头的字符

like ‘%ab\’cd%’ //这里要找的是： 包含 ab’cd 字符的字符

where子句前面必须有from子句。虽然他们2者都可以省略，但有from可以没有where，而有where必须有from。

group by 分组子句

形式：

group by 字段1 排序方式1，字段2 排序方式2， .....

通常都只进行一个字段的分组。

含义：

什么叫分组？就是将数据以某个字段的值为“依据”，分到不同的“组别”里。

分组的结果通常：

1，数据结果只能是“组”——没有数据本身的个体

2，数据结果就可能“丢失”很多特性，比如没有性别，身高，姓名，等等。

3，实际上，结果中通常只剩下“组”作为整体的信息：

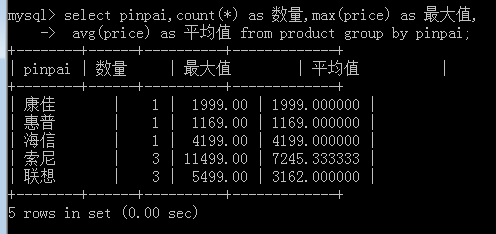
首先是该组的本身依据值，

另外，这几个可能的值：组内成员的个数，组内某些字段的最大值，最小值，平均值，总和值。

**其他字段，通常就不能用了**。

4，如果是2个字段或以上分组，则其实是相当于对前一分组的组内，再进行后一依据的分组。

上述说明的结果，其实是反映在ｓｅｌｅｃｔ语句中，就是select的“取出项”（输出项）就基本只剩下以上信息了，比如：



可见，在分组查询中，基本都依赖于一下几个函数（聚合函数，统计函数）：

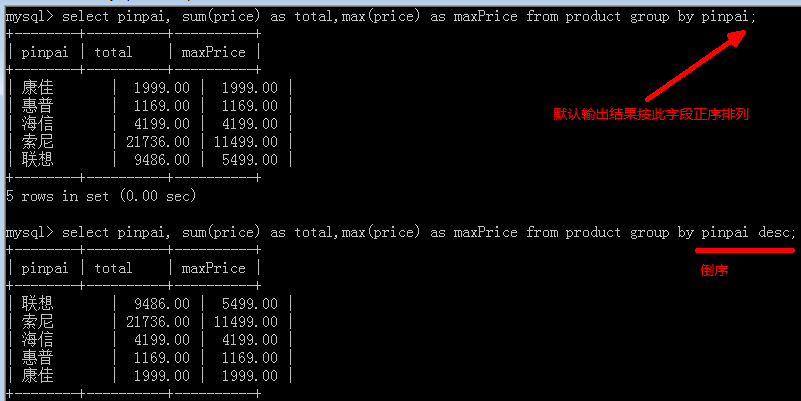
count(\*): 统计一组中的数量，通常用“\*”做参数

max(字段名)：获取该字段中在该组中的最大值。

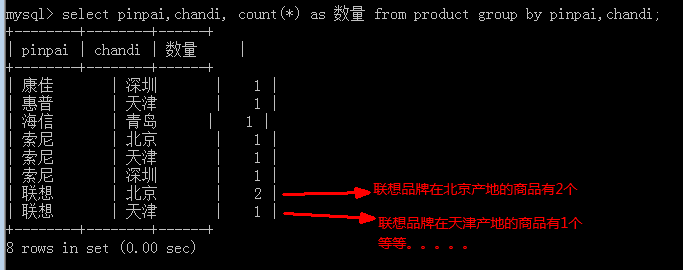
min(字段名)：获取该字段中在该组中的最小值。

sum(字段名)：获取该字段中在该组中的总和。

avg(字段名)：获取该字段中在该组中的平均值。



两级分组示例：



having子句

having子句其实概念跟where子句完全一样：

where是针对表的字段的值进行“条件判断”

having是只针对groupby之后的“组”数据进行条件判断，即

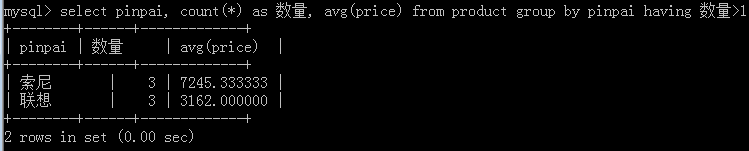
其不能使用：字段名>10

但可以使用：count(字段名)>10， 或 max(price) > 2000, 但如果字段是分组依据，也可以。

当然，通常也可以使用select中的有效的字段别名，比如：

select count(\*) as f1 , max(f1) as f2 from tab1 group by f3 having f1 > 5 and f2 < 1000;\

举例：



再使用逻辑运算符：

