# 数据库（表）设计三范式

## 第一范式（1NF）

要求数据达到原子性，数据不可再分

没有达到要求



修改后



## 第二范式（2NF）

使每一行数据具有唯一性，并消除数据之间的“部分依赖”，使一个表中的非主键字段，完全依赖于主键字段；

实验上，每个表都要设计主键，，通常int，自增长设计主键；

实际：有两方面要求

1. 每一行数据具有唯一性：只要给表设计主键，就可以保证唯一性、
2. 消除数据之间的“部分依赖”

## 什么叫依赖

依赖，就是在一个表中，其中某个字段的值B可以由另一个字段的值A来“决定”

字段B依赖于字段A

字段A决定于字段B

其意思是，如果根据字段a的某个值，一定可以找出一个确定的字段B的值，就是A决定B

对一个有主键的表，这种情况就成了：

确定了主键字段的值，则其他字段肯定都确定了

也可以这样说：

主键决定了其他字段

其他字段依赖于主键

## 什么叫部分依赖

如果某个字段，只依赖于“部分主键字段”，此时就称为“部分依赖”——发生此情况的前提一定是：主键字段有多个

## 什么叫做完全依赖

就是某个字段，是依赖于“主键的所有字段”。

——推论：如果一个表的主键只有一个字段，则此时必然是完全依赖

不良：

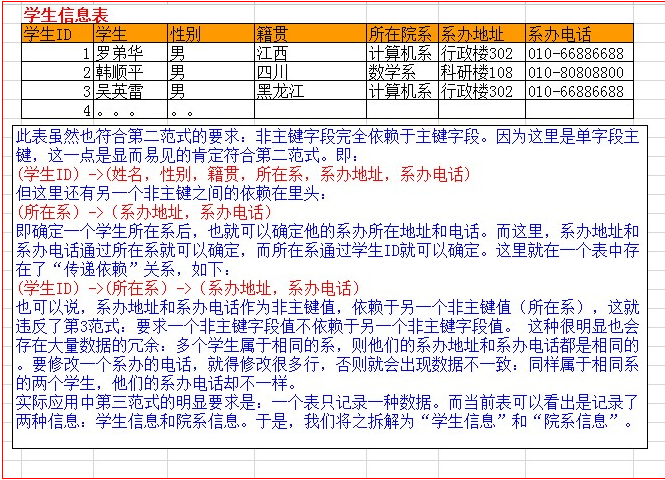


改进



## 第三范式（3NF）

使每个字段都独立地依赖于主键字段（独立性），而要消除其中部分非主键字段的内部依赖 ——这种内部依赖会构成“传递依赖”



改正

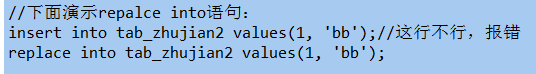


经验总结：通常在设计表的时候，基本主要遵循这样的一个原则，就可以满足前述3范式的要求

# 数据操作语句

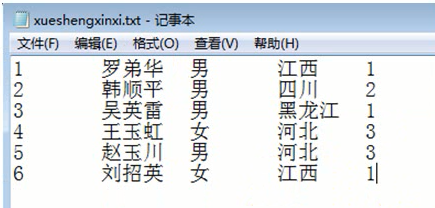
## 插入数据（增）

多种语法形式：

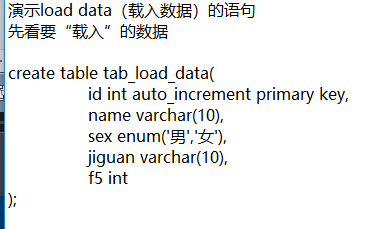
* 形式1：insert [into] 表名 [（字段名1，字段名2，....）] values (值表达式1，值表达式2，....), (.....), ....；
  + 这是最常用的插入语句，可以一次性插入多行数据，用逗号隔开；
  + 插入语句，要理解为每次插入数据都是以“行”为单位进行插入。
  + 字段名通常不推荐省略，其后续的“值列表”应该跟字段名列表“一一对应”；
  + 其中的值表达式，可以是一个“直接”，或“函数调用结果”，或变量值；其中，如果对应字段是字符或时间类型，则直接值应该使用单引号；
* 形式2：replace [into] 表名 [（字段名1，字段名2，....）] values (值表达式1，值表达式2，....), (.....), ....；
  + 其跟insert into几乎一样：唯一区别是，如果插入的数据的主键或唯一键“有重复”，则此时就会变成“修改该行数据”；
  + 
* 形式3：insert [into] 表名 [（字段名1，字段名2，....）] select 字段名1，字段名2，.... from 其他表名；
  + 将select语句查询的结果数据（可能多条），都插入到指定的表中。
  + 其中，也需要注意字段的对应：select出来的字段列表，要跟前面指定的字段列表“一一对应”
  + 
* 形式4：insert [into] 表名 set 字段名1=值表达式1，字段名2=值表达式2，....；

## Load data语法（载入数据）

用于载入如下图“结构整齐的纯文本数据”；



前提是，有一个这样的对应结构的现有的表



语法形式：

Load data infile “完整的数据文件路径” into table 表名

## 报错1290 待解决

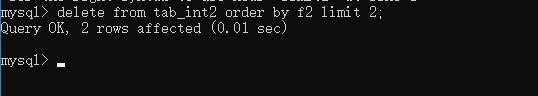
## 删除数据（删）

语句形式：

Delete from 表名 where order by排序 limit限定行数

说明：

1. 删除数据 以“行”为单位进行
2. 通常删除数据都需要带where；否则会删除所有数据
3. Where条件语法跟select中的语法一样
4. Order by 排序设定，用于指定这些数据的删除顺序
5. Limit限定用于设定删除多少行（按order设定的顺序）
6. 实际应用中，通常很少用到order和limit



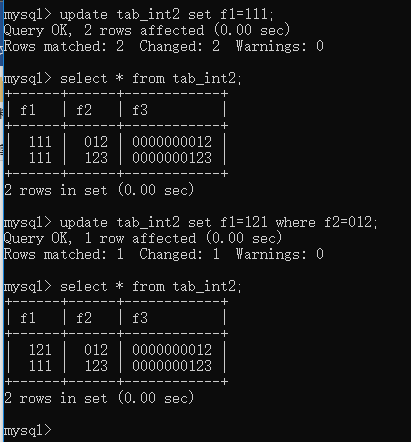
## 修改数据（改）

语法形式：

Update 表名 set 字段1=值1 字段2=值2 …… where order by排序 limit限定行数

说明：

1. 通常，update也需要添加where条件，否则修改所有数据
2. Where条件的语法跟select中的语法一样
3. Order by 用于设置修改的顺序，limit用于设定修改的行数，他们通常也结合使用；
4. 则通常情况下，很少用到order by和limit，则形式：
   1. Update 表名 set 字段1=值1 字段2=值2 …… where
5. 注意，其中的“数据值”，如果是string或datetime类型的“直接值”，就应该用单引号



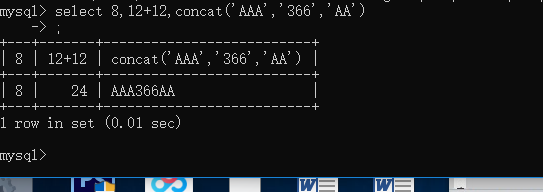
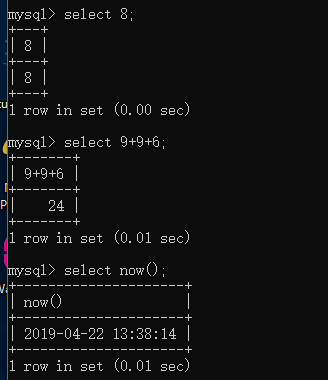
# 基本查询

## 基本语法形式

**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]；**

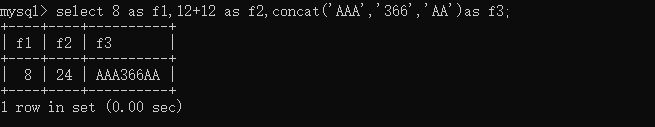
## 字段或表达式列表

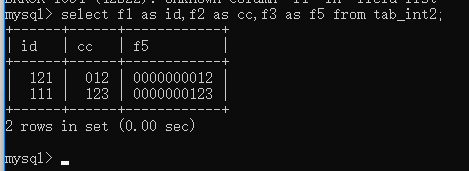
1. 字段，自然来源于“表”，则其必然依赖于from字句
2. 表达式是类似这样一个内容：8,8+3，now（）



其中now，concat是mysql系统函数，用于“连接”多个字符串

1. 每个“输出项”（字段或表达式结果），也给其字段设置别名 加上as 内容





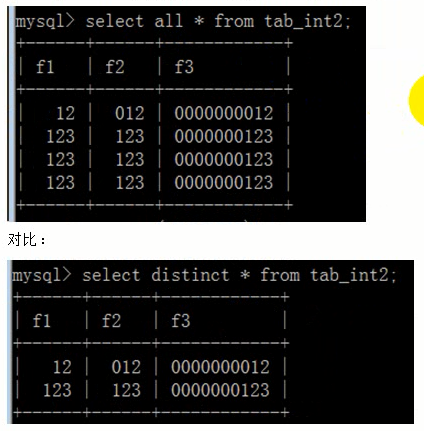
注意：实际上，表的字段并没有改变，而只是改变了结果集的字段名.

All和distinct

用于select出来的数据，是否消除“重复行”，可以不写，那就是默认值 distinct；

All 表示不消除，即所有都出来 默认值为all

Distinct：表示会消除；



# From字句

From子句表示select部分从中“取得”数据的数据源——其实就是表

通常，其后面就是表名；比如 from tab1

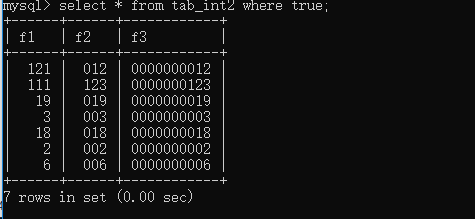
但，也可以是其他一些数据来源 from tab1 tab2（连接表）

Select输出部分，如果给定的是字段名，则其必然是来源于这里的“数据源”的字段

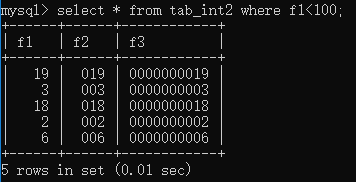
# Where字句

说明

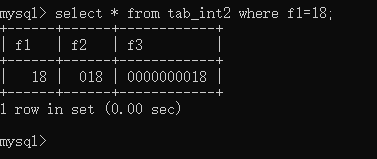
1. 对from子句中的“数据源”中的数据进行筛选的条件设定，筛选的机制是“一行一行进行判断”，其作用，几乎就跟各种语言中if语句作用一样
2. 则，可见，where子句依赖于from子句
3. Where子句中，通常都需要使用各种运算符；
   1. 算术运算符 + - \* /
   2. 比较运算符 > >= < <= =等于 <>不等于 ==等于 !=不等于
   3. 逻辑运算符 and or not



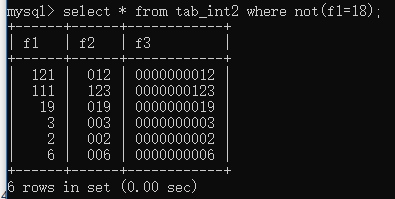
True代表全部取出



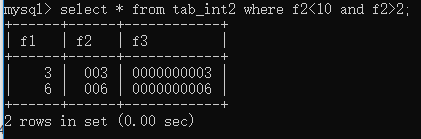
小于f1字段100的行



F1的值等于18



F1的值不等于18的值 取反（not（））



F2的值小于10，大于2

# 空值和布尔值的判断（is运算符）

有4种情况的使用：

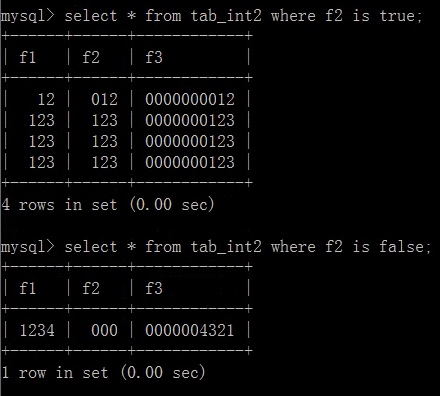
Xx is null；判断某个字段是“null”值——就是没有值

Xx is not null：判断某字段不是“null”值

Xx is true：判断某字段为“真”

Xx is false：判断某字段为“假” 0, 0.0， “” null

所谓布尔值，其实是tinyint（1）这个类型的一个“别名”，本质上，只是判断一个数字是否为0



# Between运算符：范围判断

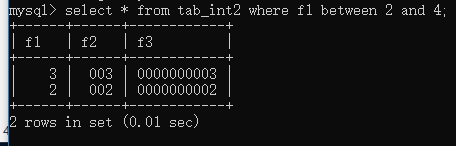
用于判断某个字段的数据是否在某个给定的范围——适用于数字类型

语法：

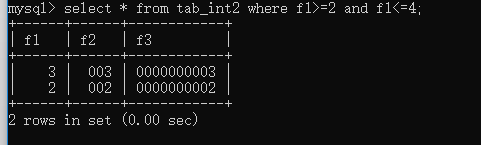
Xx between 值1 and 值2

含义：

Xx字段的值在给定“值1”和“值2”之间，其实相当于 值1 >= and <=值2



对比—select \* from tab\_int2 where f1>=2 and f1<=4;



In运算符，给定确定数据范围判断

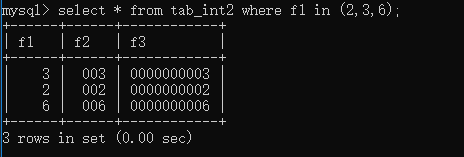
语法：

Xx in （值1，值2，值3……..）

含义：

表示字段xx的值为所列出的这些值中的一个，就算是满足了条件；这些值通常的零散

他罗列出的数据，如果有一定的规律，则其实可以使用逻辑运算符或between运算符来代替；



对比 select \* from tab\_int2 where f1=2,3,6;

# Like运算符 对字符串进行模糊查找

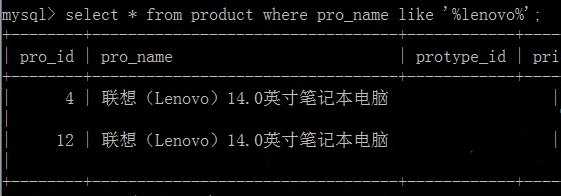
语法：

Xx like ‘要查找的内容’

含义：实现对字符串的某种特征信息的模糊查找，它其实依赖以下2个特殊的“符号”

% ：它代表“任何个数的任何字符”

* ：（下划线）它代表“一个任何字符”；



常见示例及含义：

Name= like %罗% 表示name中含“罗”这个字的所有数据行

Name=like 罗% 表示name中含“罗”开头的所有数据行

Name=like %罗 表示name中以“罗”结尾的所有数据行

Name=like 罗\_ 表示name中以“罗”开头并只有两个字符的所有数据行

Name=like \_罗表示name中以“罗”结尾并只有两个字符的所有数据行

比如 罗兰 C罗

一个新的问题：

如果我要找某个字段中含“%”或”\_”行 怎么办？

转义就OK \% \\_

例如：

Xx like %\%% 表示xx中含有百分号（%）这个字符的所有行

# Group by 子句：分组

形式：

Group by 字段 1【desc|asc】，字段字段2【desc|asc】…..

说明：

1. 分组是对“前述”已经找出的数据（即where已经筛选过了）进行某种指定标准（依据）的分组
2. 同时，该分组结果，可以同时指定其“排序筛选”；desc（倒序），asc（顺序）
3. 通常，分组就一个字段（依据），2个以上很少。

什么叫分组？

就是将多行数据，以某种标准（就是指定字段）来进行“分类”存放；

特别注意：

分组之后的结果，一定要理解为：只有一个一个组了

则结果是：在select 语句中输出（取出）部分，只应该出现“组”信息

Select 组信息1，组信息2 ….. from 数据源 group by 字段

应用中，分组之后，通常只有如下几种可用”组信息”了（即可出现在select）

1.分组依据本身的信息，其实就是该分组依据的字段名

2.每一组的数量信息：就是用count（\*）获得

3.原来数据中的“数值类型字段的聚合信息”；如下几个

a.最大值 max（字段名）

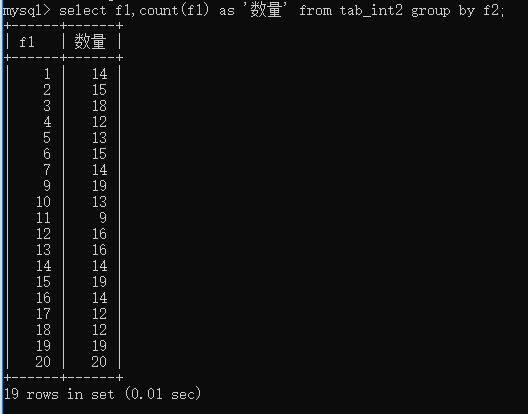
b.最小值 min（字段名）

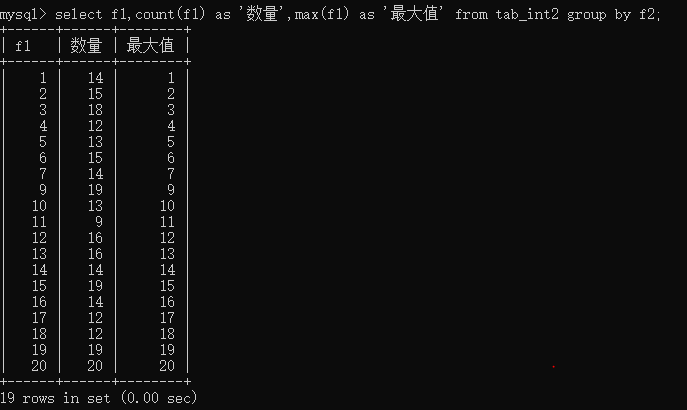
c.平均值 avg（字段名）

d.总和值 sum（字段名）

上述其实是4个系统内部函数；

Select 表名，count（\*） as 数量 from





结果含义明显，数据都出来了

# Having子句

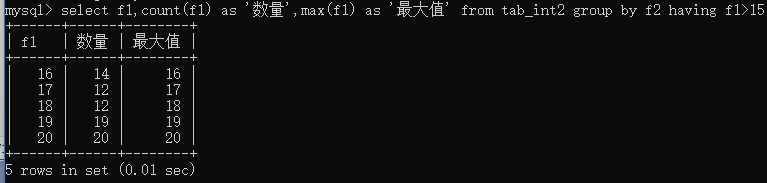
一句话：

Having的作用跟where完全一样，但其只是对“分组的结果数据”进行筛选，即：

Where对原始数据进行筛选；

Having对分组数据进行筛选

示例1：找出f1大于15的信息



示例2：找出最大值大于18的



可以用以下语句“恢复”（导入）数据

Mysql登录后

Source ‘备份数据文件的完整路径’

# Erder by 子句.

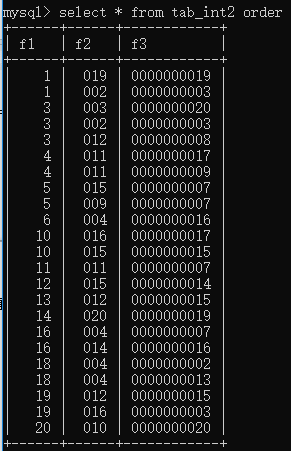
他用于将前面“取得·”的数据以设定的标准（字段）来进行排序以输出结果

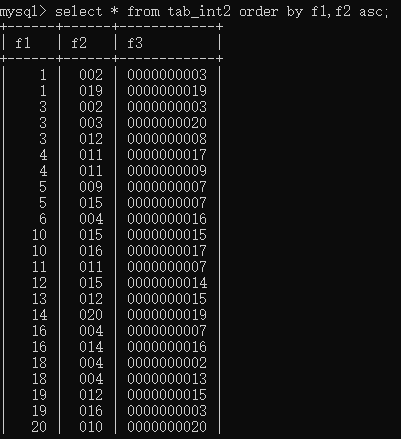
形式：

Order by 字段 asc|desc，字段2 asc|desc….

说明：

1. 对前面的结果数据以指定的一个或多个字段排序
2. 排序可以指定正序（asc，默认值）或倒序（desc）
3. 多个字段的排序，都是在前一个字段基础上，如果还有“相等值”，就继续以后续字段排序





# Limit子句

含义：

他用于将“前述取得的数据”，按指定的行取出来：从第几行开始取出1多少行；

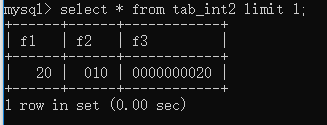
形式：

Limit 起始行数 要取出的行数

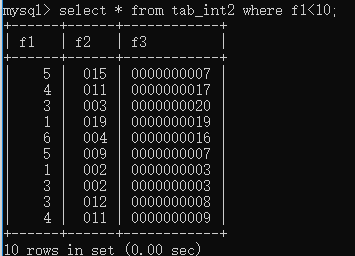
说明：

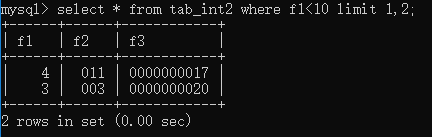
1. 起始行号都是从0开始算起
2. 起始行号跟数据中的任何一个字段（比如id）没有关系
3. 要取出的行数也是一个数字，自然应该大于0的
4. 有一个简略形式limit 行号 表示直接从0行开始取指定的行数

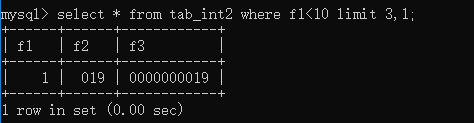
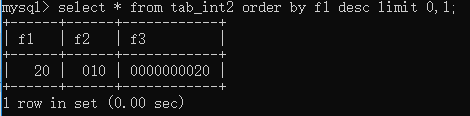
举例1；



举例2：取出1行到2行



.

对整个select语句的一些总结

1. 虽然在形式上，select的很多子句都是可以省略的，但他们的顺序（如果出现），就不能打乱的必须仍然按照给出的顺序写出来
2. Where子句依赖于from子句；即没有from就不能有Having子句
3. Having子句依赖于groupby子句，即没有groupby，就不能有having
4. Select中的“字段”也是依赖于from子句
5. 上述各子句的“内部执行过程”，基本上也是按该顺序进行的
   1. 即从from的数据源中获得“所有数据”，然后使用where对执行数据进行“筛选”，之后再使用groupby子句进行筛选出来的数据进行“分组”，接下来才可以使用having对这些分组的数据进行筛选，然后才可以使用order和limit子句，其大致如下：

