# 数据表操作

## 修改数据表

表单修改分为两部分：修改表本身和修改字段

**修改表本身：**修改表名和表选现象

修改表名：rename 老表明 to 新表名

修改表选项：修改字符集，校对集和存储引擎

Alter table 表名 表选项 [=] 值;

**修改字段：**字段操作很多：新增，修改，重命名，删除。

新增字段： alter table 表名 add [column] 字段名 数据类型 [列属性] [位置]

位置：新增字段可以放在表中的任意位置：first | after ;

修改字段：通常修改字段属性或字段的数据类型。

Alter table 表名 modify 字段名 数据类型 [属性] [位置] ;

重命名字段：alter table 表名 旧字段 新字段 数据类型 [属性] [位置] ;

删除字段：alter table drop 字段名 ;

## 添加数据

## 修改数据

## 查看数据

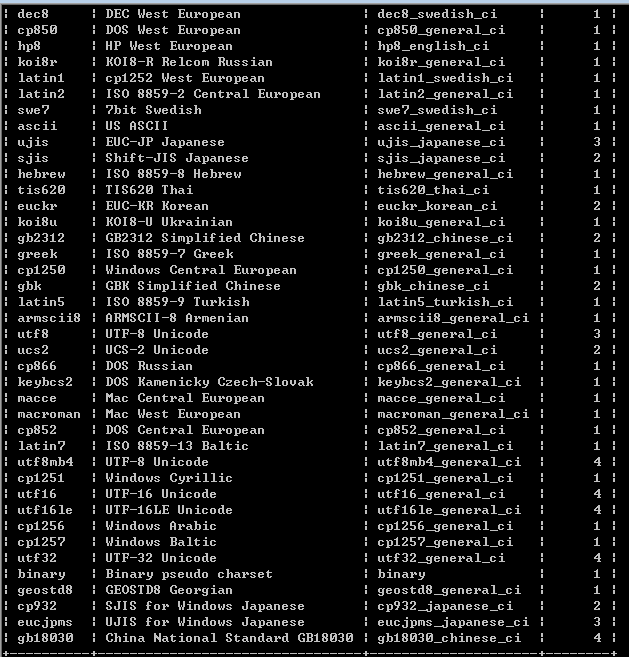
# 中文数据问题

中文数据问题本质还是字符集问题

所有的数据库服务器认为（表现）的一些特性都是通过服务器的变量进行保存：系统先读取自己的变量，看看应该如何表现。

//查看服务器到底识别哪些字符集

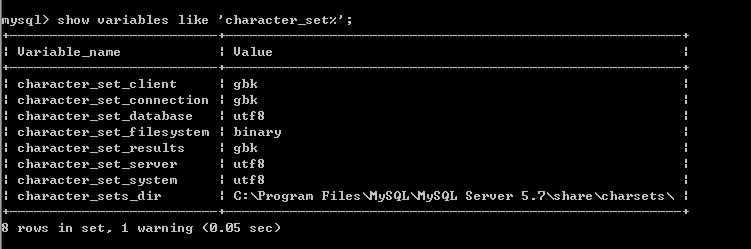
Show character set ;



基本上服务器是万能的，什么字符集都支持

//既然服务器支持这么多 的字符集，总有一种是服务器默认的跟客户端进行打交道的字符集

Show variables like ‘character\_set%’;



Character\_set\_client:服务器默认的客户端来的数据字符集

Character\_set\_connection:连接字符集

Character\_set\_database:当前数据库的字符集

Character\_set\_results:服务器默认的给外部数据的字符集

修改服务器默认的接受数据字符集：set character\_set\_client=新字符集

Set 变量=值 ：修改的只是会话级别（当前客户端，当前连接有效，关闭无效）

设置服务器对客户端的字符集认识：可以使用快捷方式：set names 设置字符集

# Unicode和UTF8区别

Unicode：是一种在计算机上使用的字符编码，它为每种语言中的每个字符设定了统一且唯一的二进制编码，以满足跨语言和跨平台进行文本转换和处理的要求，注意，Unicode是字符编码，不是字符集。

Unicode是基于通用字符集（Universal Character Set）的标准进行发展的，同时以书本的方式对外发表。

Unicode是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案，Unicode数字0～0x10FFFF来映射这些字符，最多可以容纳1114112个字符，或者说有1114112个码位。码位就是可以分配个字符的数字。

# 校对集问题

校对集：指对指定字符集下不同字符的比较规则，其特征如下：

1. 两个不同的字符集不能有相同的排序规则
2. 每个字符集有一个默认的排序规则。
3. 有一些常用的命名规则。

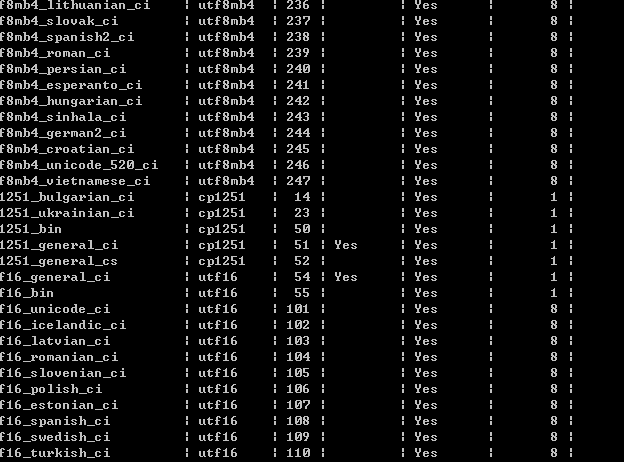
校对集有三种方式：

\_bin:binary,二进制比较，取出二进制位，一位一位的进行比较，区分大小写。

\_cs:case sensitive,大小写敏感，区分大小写。

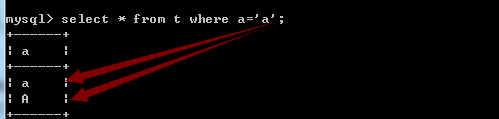
\_ci:case insensitive,大小写不敏感，不区分大小写

查看数据支持的所用的校对集：show collation



校对集应用：只有当数据进行比较的时候，校对集才生效。

校对集必须在没有数据之前声明好，如果有了数据，那么再进行数据集修改，那么修改无效。

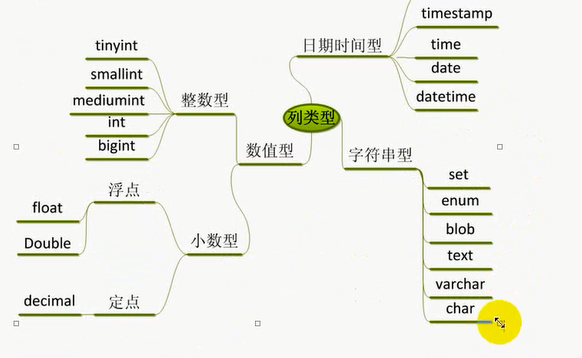


该

# 数据类型

所谓的数据类型：对数据进行统一的分类，从系统的角度出发为了能过统一的方式进行管理，更好的利用的有限的数据空间。

SQL中数据类型分为三类：数值类型，字符串类型和日期类型



显示宽度：没有特别的意义，只是默认的告诉用户可以显示的形式而已，实际上用户可以控制的，这种控制不会改变数据本身的大小。

显示宽度的意义：当数据不够显示宽度的时候，会自动让数据变成对应的显示宽度，通常需要搭配一个前导0来增加宽度，不改变值大小：zerofill（零填充）：零填充会导致数值自动变成无符号。

## 整数型



## 浮点型



## 时间类型



## 字符串类型

### 定长字符串

定长字符串：char 磁盘在定义结构的时候，就已经确定了最终数据存储长度

Char(L):L代表长度，可以存储的长度，单位为字符，最大长度值为255

### 变长字符串

变长字符串：varchar，在分配空间的时候，按照最大的空间分配，但实际用了多少是根据具体的数据来确定的。

Varchar(L):L表示字符长度，理论长度是65536个字符，但是会多出1到2个的字节来确定存储的实际长度。

例如：varchar(10):的确存储10个汉字，utf8情况下10\*3+1=31个字节。

### 文本字符串

如果存储的数据量很大，通常来说超过255个字符就会使用文本字符串

文本字符串根据存储的数据格式进行分类:text和blob

Text:存储文字

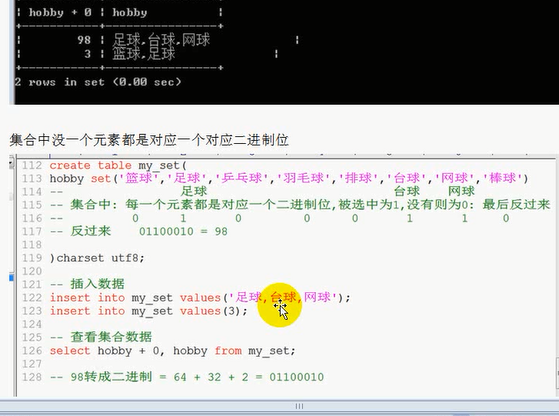
Blob:存储二进制数据（通常不用）

### 枚举字符串



枚举原理：枚举在进行数据规范时候（定义的时候），系统会自动建立一个数字与枚举元素的对应关系（关系放在日志中），然后在进行数据插入的时候，系统自动的将字符串转化中对应的数字存储，在进行数据查询的时候，系统自动的将数字转换成对应的字符串显示。

### 集合字符串



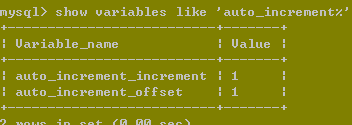
# 自动增长

自增长的特点：

1. 任何一个字段做自增长必须前提是本身是一个索引（key一栏有值）
2. 自增长字段必须是数字(整形)
3. 一张表中最多只有一个自增长

## 查看自增长变量

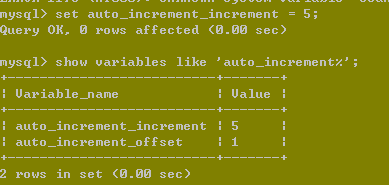
Show variables like ‘auto\_increment%’



## 修改自增长步长

修改是对整个数据的修改，而不是单张表（修改是回话级别）

Set auot\_increment\_increment = ?



## 删除自增长：自增长是字段的一个属性

Alter table 表名 modify 字段 属性；

# 唯一键

唯一键的本质与主键差不多：唯一键默认的允许自动为空，而且可以多个为空(空字段不参与唯一性比较)。

## 增加唯一键

### 在创建表的时候，在字段之后直接跟unique/unique key

### 在所有的字段之后增加unique key(字段列表):复合唯一键

### 在创建表之后添加唯一键

## 唯一键约束

唯一键与主键本质相同，唯一的区别就是唯一键默认允许为空，而且是多个为空。

## 更新唯一键&删除唯一键

更新唯一键：先删除后新增（唯一键可以有多个）

### 更新唯一键：

Alter table 表名 drop index 索引名字；--唯一键默认字段名为索引名

# 索引

索引：系统根据某种算法，将已有的数据(未来可能新增的数据)单独建议一个文件，文件能够实现快速上匹配数据，并且能够快速的找到对应表中的记录。

索引的意义：

1. 提高查询数据的效率。
2. 约束数据的有效性(唯一性等)

如果某个字段需要作为查询条件经常使用，那么可以使用索引；

如果某个字段需要进行数据的有效性约束，也可以使用索引（主键，唯一键）

Mysql中提供了多个索引

1. 主键索引：primary key
2. 唯一键索引：unique key
3. 全文索引：fulltext index
4. 普通索引：index

全文索引：针对文章内部的关键字进行索引。

## 创建索引

Alter table 表明 add index 索引名(表字段名or多个字段名)；

## 删除索引

drop index 索引名 on 表名；

## 查看索引

Show index from 表名；



Table：表的名称。  
Non\_unique：如果索引不能包括重复词，则为0。如果可以，则为1。  
 Key\_name：索引的名称。  
Seq\_in\_index：索引中的列序列号，从1开始。  
Column\_name：列名称。  
Collation：列以什么方式存储在索引中。在MySQL中，有值‘A’（升序）或NULL（无分类）。  
Cardinality： 索引中唯一值的数目的估计值。通过运行ANALYZE TABLE或myisamchk -a可以更新。基数根据被存储为整数的统计数据来计数，所以即使对于小型表，该值也没有必要是精确的。基数越大，当进行联合时，MySQL使用该索引的机会就越大。  
Sub\_part如果列只是被部分地编入索引，则为被编入索引的字符的数目。如果整列被编入索引，则为NULL。  
Packed　 指示关键字如何被压缩。如果没有被压缩，则为NULL。  
Null如果列含有NULL，则含有YES。如果没有，则该列含有NO。  
Index\_type　 用过的索引方法（BTREE, FULLTEXT, HASH, RTREE）。

# 关系

将实体与实体的关系，反应到最终数据库表的设计上来：讲关系分成三种：一对一，一对多，多对多。

## 一对一

一张表中数据一定只能与另外一张表的一条记录进行对应，反之亦然。

## 一对多

一张表中有一条记录可以对应另外一张表中的多条记录，但是反过来，另外一张表的一条记录只能对应第一张表的一条记录。

## 多对多

一张表中有一条记录可以对应另外一张表中的多条记录，同时，另外一张表的一条记录只能对应第一张表的多条记录。

# 范式

范式(normal format)：是一种离散数学中的知识，是为了解决一种数据的存储与优化的问题，保证数据的存储后，凡是能够通过关系寻找出来的数据，坚决不再重复存储，终极目标是为了减少数据的冗余。

范式：是一种分层结构的规范，分为六层，每一层都比上一层更加严格，若要满足下一层范式前提是满足上一层范式。

六层范式：1NF,2NF,3NF,…6NF. 1NF是最底层，要求最低，6NF是最高层，最严格。

## 第一范式(1NF)

在设计表存储数据时，如果表中设计的字段存储的数据，在取出来使用之前还需要额外的处理，那么说表的设计不满足第一范式，第一范式要求字段的数据具有原子性：不可再分。

## 第二范式(2NF)

第二范式：在数据表设计过程中，如何复合主键（多字段主键），且表中有字段并不是由整个主键来确定，而是依赖主键中的某个字段（主键的部分）:存在字段依赖主键的部分的问题，称之为部分依赖：第二范式就是解决表设计不允许出现部分依赖。

## 第三范式(3NF)

# 数据的高级操作

数据的操作：增删改查

## 新增数据

Insert into 表名[（字段名列表）] value（值列表）

数据的插入时候，假设主键对应的值已经存在了，插入一定会失败！

### 主键冲突

当主键存在冲突的时候(Duplicate key)可以选择性的进行处理：更新或替换

主键冲突：更新操作

Insert into 表名[(字段列表：包含主键)] value (值列表) on duplicate key update 字段=新值

主键冲突：替换操作

Replace into 表名 [(字段列表：包含主键)] values(值列表)

## 蠕虫复制

蠕虫复制：从已有的数据中去获取数据，然后将数据又进行新增操作：数据成倍的增加。

表创建高级操作：从已有的表创建新表(复制表结构)

Create table 表名 like 数据库.表名

蠕虫复制：先查出数据，然后将查处的数据新增一遍

Insert into 表名[(字段列表)] select 字段列表/\* from 数据表名 ；、

蠕虫复制意义：

1. 从已有的表中拷贝数据到新表中
2. 可以迅速的让表中的数据膨胀到一定的数量级，测试表的压力以效率。

## 更新数据

Update 表名 set 字段=值 where [条件]

高级新增语法

Update 表名 set 字段= 值【where条件】limit 条件。

## 删除数据

与更新类似，可以通过limit来限制数量

Delete from 表名 【where条件】limit 条件。

删除：如果表中有自增长，那么删除后自增长不会还原。

思路：数据的删除是不会改变表结构，只能删除表后重建表。

Truncate 表名

## 查询数据

基本语法：select 字段列表/\* from 表名

完整语法：select [select 选项] 字段列表[字段别名]/\* from 数据源 [where条件语句][group by字句][having字句][order by字句][limit字句]

**Select选项：**select对查出的结果的处理方式

All:默认的保留所有的结果

Distinct:去重，查出的数据，将重复给去除。

**字段别名：**当数据进行查询出来的时候，有时候名字并不一定满足需求(多表查询的时候，可能会有同名字段)需要对字段进行重命名：别名。

字段名 [as] 别名。

**数据源：**数据的来源，关系型数据库的来源都是数据表，本质上只有保证数据类似二维表，最终都可以作为数据源。

数据源分为多种：单表数据源，多表数据源，查询语句。

多表数据源 select \* from 表名1,表名2,….;

子查询：数据的来源是一条查询语句(查询语句的结果是二维表)

Select \* from（select 语句） as 表名

**Where子句:**用来判断数据，筛选数据 where子句返回结果：0或1

where原理：where是唯一一个直接从磁盘获取数据的时候就开始判断条件，从磁盘取出一条记录，开始进行where判断：判断的结果如果成立就保存到内存，如果失败直接放弃。

Where过滤时，有两种过滤不被允许：

1. 由于数据还没有分组，因此现在还不能在where过滤条件中使用 where\_condition=MIN(col)这类对类型统计。
2. 由于没有进行列的选取操作，因此在select中使用列的别名也是不允许的。

**Group by字句：**分组，根据某个字段进行分组(相同的放一组，不同的分到不同的组)

基本语法：group by 字段。

分组的意义：是为了统计数据(按组统计，按分组字段进行数据统计)

SQL提供了一系列的统计函数

Count():统计分组后的记录数，每一组有多少记录

Max():统计每组中最大是值。

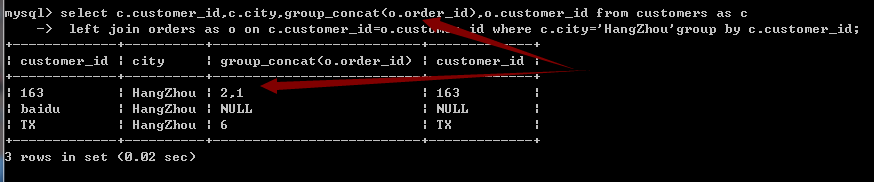
Avg()：统计平均值。】

Sum()：统计和

分组会自动排序：根据分组字段默认升序。

多字段分组：先根据一个字段进行分组，然后对分组后的结果再次按照其他字段进行分组

group\_concat(字段名) 可以对分组的结果中的某个字段进行字符连接。



回溯统计:with rollup:任何一个分组后都会有一个小组，最后都需要向上级分组进行汇报统计：根据当前分组的字段这就是回溯统计，回溯统计的时候会将分组字段为空。

多字段回溯：考虑第一层分组会有回溯，第二次分组要看第一次分组的组数，组数是多少，回溯就是多少，然后再加上第一层回溯即可。

**Having子句**：与where子句一样：进行条件判断的，having也是对分组进行过滤的筛选器。

Where是针对磁盘数据进行判断，进入内存之后，会进行分组操作，返会结果就需要having来处理。

1. Having几乎能做where所有的事情,where不能，比如对分组后的结果进行筛选。
2. Having可以使用别名，where不能

**Order by**：根据某个字段进行升序或降序排序，依赖校对集

基本语法：order by 字段名 [asc|desc] –---asc升序 ，desc降序

排序可以进行对字段排序：先根据某个字段进行排序，然后排序好的内部，再按照某个数据进行再次排序。

**Limit子句：**是一种限制结果 的语句，限制数量。

Limit有两种使用方式：

方案一：

1. 只用来限制长度（数据量）:limit数据量
2. 限制起始位置，限制数量：limit起始位置，长度。

方案二：主要用于来实现数据的分页，为用户节省时间，提交服务器的响应效率，减少资源的浪费。

**Distinct:**如果在查询中指定了Distinct子句，则会创建一张内存临时表（如果内存中存在不下就放到磁盘上）。临时表的列增加了一个唯一索引，以此去除重复数据。

# 连接查询

连接查询：将多张表（可以大于2）进行记录的连接（按照某个指定的条件进行数据拼接）最终结果：记录数可能变化，字段数一定会增加(至少两张表的合并)

连接查询的意义：在用户查看数据的时候，需要显示的数据来自多张表

连接查询：join 使用方式：左表 join 右表

左表：在join关键字左边的表

右表：在join关键字右边的表

### 连接查询的分类

SQL中将连接查询分成四类：内连接，外连接，自然连接和交叉连接

create table customers

(

customer\_id varchar(10) not null ,

city varchar(10) not null,

primary key(customer\_id)

);

insert into customers select '163','HangZhou';

insert into customers select '9you','ShangHai';

insert into customers select 'TX','HangZhou';

insert into customers select 'baidu','HangZhou';

create table orders

(

order\_id int not null auto\_increment ,

customer\_id varchar(10) ,

primary key(order\_id)

);

insert into orders select null,'163';

insert into orders select null,'163';

insert into orders select null,'9you';

insert into orders select null,'9you';

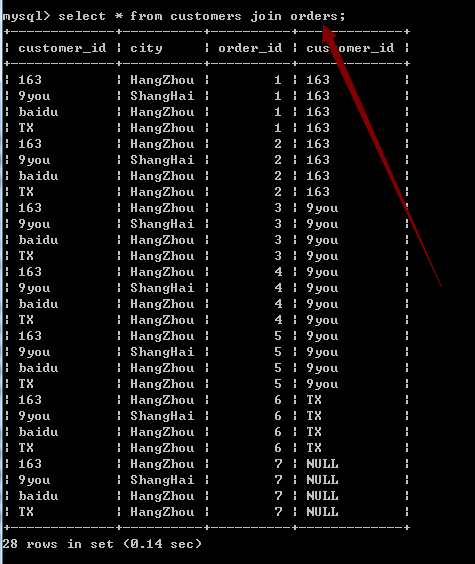
insert into orders select null,'9you';

insert into orders select null,'TX';

insert into orders select null,null;

#### 交叉连接

交叉连接：从一张表中循环取出每一条数据，每条记录都去另外一张表进行匹配：匹配一定保留，而连接本身字段就会增加(保留)，最终形成的结果叫做：笛卡尔积

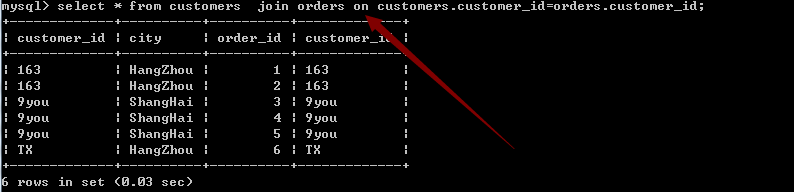


#### 内连接

内连接[inner]join:从左表中取出的每一条记录，去由表中与所有的记录进行匹配，匹配必须是某个条件在左表中与右表中相同，最终才会保留结果，否则不保留。

基本语法： 左表[inner] join 右表 on 左表.字段=右表.字段

on表示连接条件，条件字段就是代表相同的业务含义



内连接可以没有连接条件,没有on之后的内容，这个时候系统会保留所有的结果(笛卡尔积)

内连接可以使用where代替 on关键字(where没有on效率高)。

#### 外连接

外连接(outer join):以某张表为主，取出里面的所有的记录，然后每条记录与另外一张表进行连接，不管能不能匹配上条件，最终都会保留，能匹配，正确保留，不能匹配，其他表的字段都置空Null.

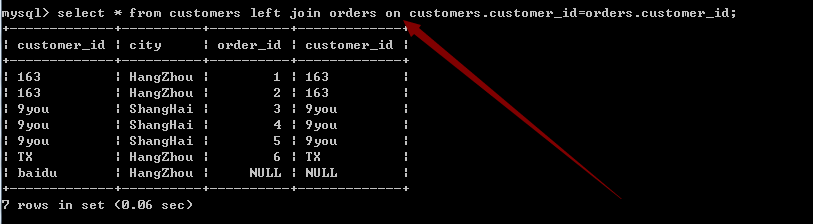
外连接分为两种：是以某张表为主，有主表

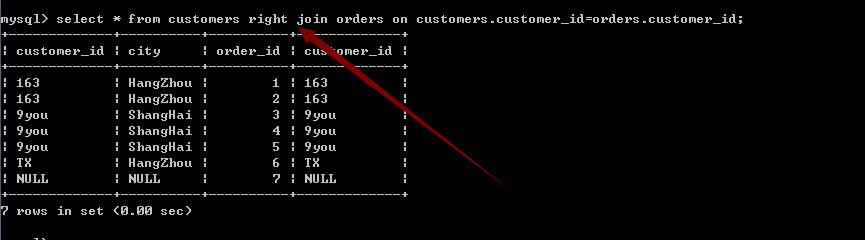
Left join:左外连接，以左表为主表

Right join:右外连接，以右表为主表。

基本语法：左表 left/right join 右表 on 左表.字段=右表.字段

虽然左连接和右连接有主表差异，但是显示的结果：左表的数据在左边，右表的数据在右边





#### 自然连接

自然连接(natural join)：就是自动匹配连接条件：系统以字段名字作为匹配模式(同名字段就作为条件，多个同名字段都最为条件)

自然连接：可以分为自然内连接和自然外连接

自然内连接：左表 natural join 右表

# 联合查询

联合查询：将多次查询(select语句)，在记录上进行拼接(字段不会增加)

基本语法：对条select语句构成：每一条select语句获取的字段数必须严格一致（但是字段类型无关）

Select语句1 Union[union选项] select 语句2….

Union选项：与select选项一样有两种：

All:保留所有(不管重复)

Distinct:去重（整个重复）默认的

意义：分为两种：

1.查询同一张表，但是需求不同，如：查询一个班级男生升高 升序，女生 降序

2.多表查询，多张表的结构是完全一样的，保存的数据（结构）也是一样的。

在联合查询中 order by 不能直接使用，需要对查询语句使用().如果要order by 生效必须使用limit，limit使用限制的最大值。

# 子查询

子查询：查询是在某个查询结果之上进行（一条select语句内容包含了另外一条select语句）

子查询的好处：

（1）子查询允许结构化查询，这样就可以把一个查询语句的每一个部分隔开

（2）子查询提供了另一种方法来执行有些需要复杂的JOIN和UNION来实现的操作

一个子查询会返回一个标量，一个行，一个列或一个表，这些子查询被称为标量，列，行和表子查询。

子查询的分类：按位置分类，按结果分类

按位置分类：子查询(select语句)在外部查询(select语句)出现的位置

From子查询：子查询在from字后

Where子查询：子查询在where条件中

Exists子查询：子查询出现在exists里面

按结果分类：根据子查询得到的数据进行分类(理论上任何一个查询得到的结果都可以理解为二维表)

标量子查询：子查询得到的结果是一行一列

列子查询：子查询得到的结果是一列多行。

行子查询：子查询得好的结果是多列一行(多行多列)

上面几个出现的位置都是在where之后。

表子查询：子查询得到的结果是多行多列(出现的位置是在from之后)

按照期望值是数量，可以将子查询分为标量子查询和多值子查询。

若按照对外部查询的依赖可分为独立子查询和相关子查询。

标量子查询和多值子查询可以是独立子查询。

## 标量子查询

## 列子查询

需求：查询所有在读班级的学生（班级表中存在的班级）

1. 确定数据源：学生表

Select \* from my\_student where c\_id in(？)

1. 确定有效的班级id:所有的班级id

Select id from my\_class ;

组合：select \* from my\_student where c\_id in (select id from my\_class);

列子查询返回的结果会比较，需要使用in作为条件匹配，其实在mysql中还有几个类似的条件：all ,some,any，in

**ANY：**any关键词必须与一个比较操作符一起使用，any关键词的意思是“对于子查询返回的列中的任意数值，如果比较结果为TRUE则返回TRUE”

Select s1 from t1 where s1 > any(select s1 from t2 );

**IN：**in是“=ANY”的别名

**SOME:**词语some是any的别名，使用some很少。

ALL:词语all必须与比较操作符一起使用，all的意思是“对于子查询返回的列中的所有值，如果比较结果为TRUE，则返回TRUE”

## 行子查询

行子查询：返回的结果可以是多行多列（一行多列），行子查询需要构建行元素:行元素是由多个字段构成。

需求：查询学生中，年龄最大和身高最高的学生。

1. 确定数据源

Select \* from my\_student where age=? And height=?

1. 确定最大年龄和最高身高

Select max(age),max(height) from my\_student;

-

行子查询组合：

Select \* from my\_student where

(age,height) = -- 行元素

(select max(age),max(height) from my\_student)

## 表子查询

表子查询：返回的结果是多行多列的二维表：子查询返回的结果当做二维表来使用。

需求：找出每一个班级最高的学生。

1. 确定数据源：现将学生按照身高进行降序排序

Select \* from my\_student order by height desc ;

1. 从每一个班级中选出第一个学生

Select \* from my\_student group by c\_id;

组合： select \* from (select \* from my\_student order by height desc) as student group by c\_id ;

表子查询：from子查询：得到的结果作为from的数据源。

## Exists子查询

Exists:是否存在的意思，exists子查询是用来判断某些条件是否满足（垮表），exists是接在where之后的，exists返回的结果只有1和0。

Exists是一个非常强大的谓词，它允许数据库高效地检查指定查询是否产生某些行，通常Exists的输入时一个子查询，并关联到外部查询，但这不是必须的。根据子查询是否返回行，该谓词返回TRUE或FALSE。与其他谓词和逻辑表达式不同的是，无论输入子查询是否返回行，Exists都不会返回UNKOWN。

# 视图

## 视图数据操作

### 新增数据

1. 多表视图是不能新增数据的
2. 可以向单表视图中新增数据，但是视图中包含的字段必须有基表中所有不能为空（或者没有默认值）字段

## 视图算法

视图算法：系统对视图以及外部查询视图的select 语句的一种解析方法。

视图算法定义为三种

Undefined:未定义（默认的），这不是一种实际使用是算法，是一种推卸责任的算法，告诉系统，视图没有定义算法，系统自己看着办。

Temptable:临时表算法，系统应先执行视图的select语句，后执行外部查询语句。

Merge:合并算法，系统应该将试图的select语句和外部查询语句进行合并，然后执行（效率高）。

算法指定：在创建视图的时候进行指定算法

Create algorithm=算法名 view 视图名 as select语句。

算法选择：如果试图的select语句中包括一个查询字句（五字句），而且很可能顺序比外部的查询语句都要靠后，一定是用算法temptable

# 事物

**事物的四个基本要求：**原子性（Atomicity），一致性（Consistency），隔离性（Isolation）和

持续性（Durablility）。

原子性：一个事物就是一个单元工作，当中可能包括数个步骤，这些步骤必须全部执行成功，若一个失败，则整个事物声明失败，事物中其他步骤必须撤销曾经执行过的动作，回到事物前的状态。

一致性：事物作用的数据集合在事物前后必须一致，若事物成功，真个数据集合都必须是事物操作后的状态，若事物失败，整个数据集合必须与开始事物前一样没有变更，不能发生整个数据集合部分变更，部分没有变更转态。

隔离性：在多人使用的情况下，每个用户可能进行自己的事物，事物与事物之间，必须互不干扰，用户不会意识到别的用户正在进行事物，就好像只有自己在进行操作一样。

持续性：事物一旦成功，所有的变更必须保存下来。

## 事物操作

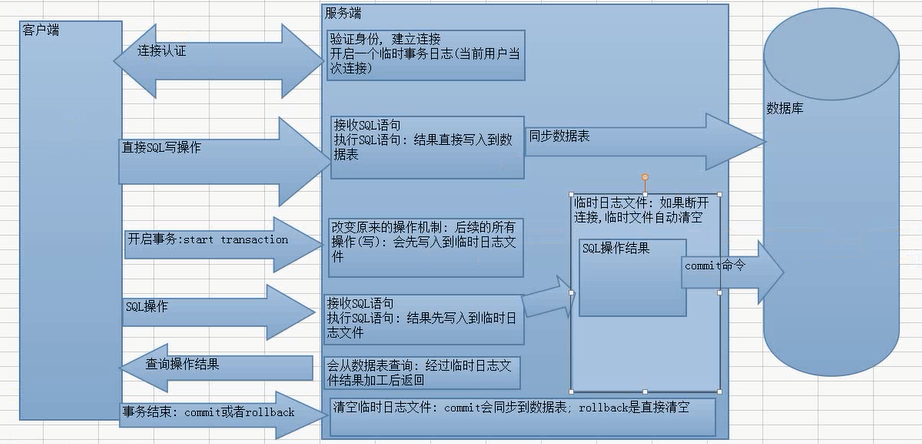
事物操作分为两种：自动事物（默认的），手动事物。

手动事物：操作流程

1. 开启事物：告诉系统一下所有的操作（写）不要直接写入到数据库中，先存放到事务日志中。Start transaction .
2. 进行实物操作。
3. 关闭事物：选择性的将日志文件中操作的结果保存到数据表中（同步）或者直接清空事物（原来操作全部清空）。
4. 提交事物：同步数据表（操作成功）commit
5. 回滚事物：直接清空日志表（）rollback

## 事物原理

事物操作原理：事物开启之后，所有的操作都会临时保存到事物日志中，事物日志只有得到commit命令才会同步到数据表，其他任何情况都会清空（rollback，断点，断开连接）

ck

## 回滚点

回滚点：在某个成功操作之后，后续的操作有可能成功有可能失败，但是不管成功还是失败，前面的操作已经成功，可以在当前成功的位置，设置一个点，可以供后续操作失败之后回到该位置，而不是返回所有的操作，这个点称之为回滚点。

设置回滚点语法：savepoint 回滚点名

回到回滚点语法：rollback to 回滚点名

# 变量

变量分为：系统变量和自定义变量

## 系统变量

系统定义好的变量，大部分的时候不需要使用系统变量：系统变量是用来控制服务器的表现。

查看系统变量：show variables;

查看具体变量：任何一个有数据返回的内容都是有select查看

Select @@变量名。

### 修改系统变量

修改系统变量分为：会话级别和全局级别

会话级别：临时修改，当前客户端连接有效。

Set 变量名 = 值 或者 set @@变量名=值

全局级别：一次修改，全局有效。

Set globel 变量名=值

## 自定义变量

系统为了区分系统变量，规定用户自定义变量必须使用@符号

Set @变量名 = 值 ；

Mysql允许从数据表中获取数据，然后赋值给变量：两种方法。

1. 边赋值边查看结果

Select @变量名 := 字段 from 数据源

1. 只赋值不查看结果：要求很严格，数据记录最多只允许获取一条数据：mysql不支持数组

Select 字段列表 from 数据源 where 条件into 变量列表

# 数据备份和还原

备份：将当前已有的数据或记录保存。

还原：将以保留的数据恢复到对应的表中。

数据的备份和还原有很多：数据表备份，单表数据备份，SQL备份，增量备份。

## 数据表备份

不需要通过SQL来备份，直接进入到数据库文件夹复制对应的表结果以及数据文件，以后还原的时候直接将备份的文件放入即可。

数据表备份有前提条件：根据不同的存储引擎有不同的要求。

Mysql的存储引擎：innodb和myisam等。

伏笔Innodb和Myisam数据存储方式：

Innodb:只有表结构,，数据全部存储到iddata1文件中

Myisam:表，数据和索引单独分开储存。

数据表备份适用于Myisam存储引擎，

## 单表备份

每次只能备份一张表，只能备份表数据（表结构不备份）

通常使用：将表中的数据导入到外部文件。

备份：从表中选出一部分数据保存到外部文件中（outfile）。

Select \* / 字段列表 into outfile ‘文件所在路径’ from 数据源

高级备份：自己制定字段和行的处理方式。

Select \* /字段列表 into outfile ‘文件所在路径’ fields字段处理 lines行处理 from 数据源。

Field字段处理：

Enclosed by :字段使用什么包裹，默认使用’’空字符串。

Terminated by :字段以什么结束，默认使用’\t’。

Escaped by ：特殊字符使用什么处理，默认使用’\\’反斜杠处理。

Lines行处理：

Starting by :每行以什么开始，默认使用’’空字符串。

Terminated by:每行以什么结束，默认使用’\r\n’

数据还原：将一个外部保存到的数据文件重新恢复到表中（数据库中表结构必须存在）。

Load data infile ‘文件路径’ into table 表名[(字段列表) fields 字段处理 lines 行处理]。

## SQL备份

备份的是SQL语句：系统会对表结构和数据进行处理，变成对应的sql语句，然后进行备份，还原的时候只要执行sql语句即可。

备份：mysql没有提供对应的语句，需要使用myql提供对应的软件：mysqldump.exe。

Mysqldump.exe是一种客户端，需要连接认证。

Mysqldump/mysqldum.exe –u –p 数据库名 [数据表名1，数据表名2…] > 外部文件路径（简易使用.sql结尾）。

还原：1.使用mysql.exe客户端还原

Mysql.exe/mysql –u –p 数据库名 > 文件路径。

1. 使用sql指令还原

Source 备份文件路径。

# 触发器

触发器（trigger）:事先为某张表绑定好一段代码，当表中的某些内容发生改变时，系统会自动触发代码执行。

触发类型：事件类型，触发时间，触发对象。

事件类型：增删改，三种类型：insert delete和update

触发时间：befor和after

触发对象：表中的每一条记录。

一张表最多只能有一种触发时间的一种类型：最多一张表有6个触发器

需求：有两张表，一张订单表，一张商品表，每生成一张订单是，库存就会自动减少。

## 创建触发器

在mysql高级结构中：没有大括号，都是使用对用的字符符号对应。

触发器的基本语法：

Delimiter 自定义符号:后续代码中只要碰到该符号才算结束

Create trigger 触发器名字 触发时间 事件类型 on 表名 for each row

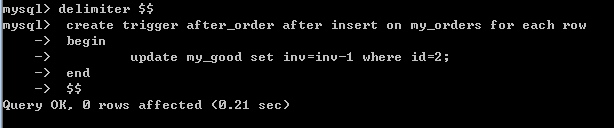
Begin

End

语句结束符：自定义符号

--将临时修改修正回来

Delimiter ;

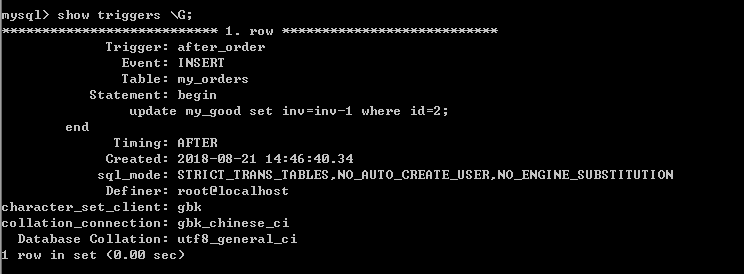


存在问题：

## 查看触发器

查看所有的触发器或者模糊匹配

Show triggers[like ‘pattern’]



查看触发器创建语句

Show create trigger 触发器名字；

所有的触发器都会保存到一张表中：information\_schema.trigger

## 修改触发器

触发器只能删除，不能修改，只能在删除后修改。

Drop trigger 触发起名

## 触发器记录

触发器记录：不管触发器是否出发了，只要当某种操作准备执行时，系统就会将当前要操作的记录的当前状态和即将执行之后新的状态给分别保留下来，供触发器使用，其中，要操作当前状态保存到old中，操作之后的可能形态保存给new。

Old:代表旧记录，new:代表新纪录。

删除的时候是没有new的，插入的时候是没有old。

# 存储过程

存储过程如同一门程序设计语言，同样包含了数据类型、流程控制、输入和输出和它自己的函数库。

基本语法：

create procedure sp\_name()  
begin  
.........  
end

调用存储器：call sp\_name()  
注意：存储过程名称后面必须加括号，哪怕该存储过程没有参数传递。

删除存储过程：drop procedure sp\_name。