# 子查询

## 什么是子查询

### 子查询概念

子查询：sub query

子查询是一种常用计算机语言SELECT-SQL语言中嵌套查询下层的程序模块。当一个查询是另一个查询的条件时，称之为子查询。

子查询：指在一条select语句中，嵌入了另外一条select语句，那么被嵌入的select语句称之为子查询语句。

### 主查询概念

主查询：主要的查询对象，第一条select语句，确定的用户所有获取的数据目标（数据源），以及要具体得到的字段信息。

### 子查询和主查询的关系

1、 子查询是嵌入到主查询中的；

2、 子查询的辅助主查询的：要么作为条件，要么作为数据源

3、 子查询其实可以独立存在：是一条完整的select语句

## 子查询分类

### 按功能分

标量子查询：子查询返回的结果是一个数据（一行一列）

列子查询：返回的结果是一列（一列多行）

行子查询：返回的结果是一行（一行多列）

表子查询：返回的结果是多行多列（多行多列）

Exists子查询：返回的结果1或者0（类似布尔操作）

### 按位置分

Where子查询：子查询出现的位置在where条件中

From子查询：子查询出现的位置在from数据源中（做数据源）

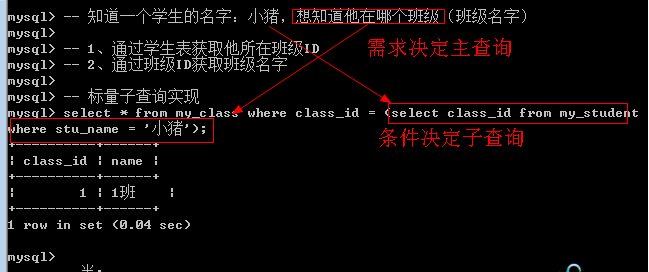
# 标量子查询

## 概念

标量子查询：子查询得到结果是一个数据（一行一列）

## 语法

基本语法：select \* from 数据源 where 条件判断 =/<> (select 字段名 from 数据源 where 条件判断); //子查询得到的结果只有一个值



# 列子查询

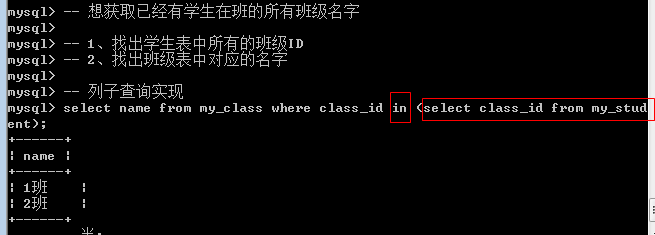
## 概念

列子查询：子查询得到的结果是一列数据（一列多行）

## 语法

基本语法：

主查询 where 条件 in (列子查询);



# 行子查询

## 概念

行子查询：子查询返回的结果是一行多列

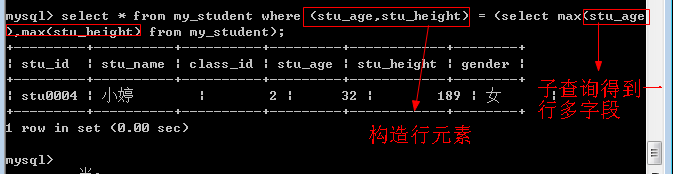
## 行元素

行元素：字段元素是指一个字段对应的值，行元素对应的就是多个字段：多个字段合起来作为一个元素参与运算，把这种情况称之为行元素。

## 语法

基本语法：

主查询 where 条件[（构造一个行元素）] = (行子查询);



## 总结

已经学过三个子查询：常见的三个子查询

标量子查询、列子查询和行子查询：都属于where子查询

# 表子查询

## 概念

表子查询：子查询返回的结果是多行多列，表子查询与行子查询非常相似，只是行子查询需要产生行元素，而表子查询没有。

行子查询是用于where条件判断：where子查询

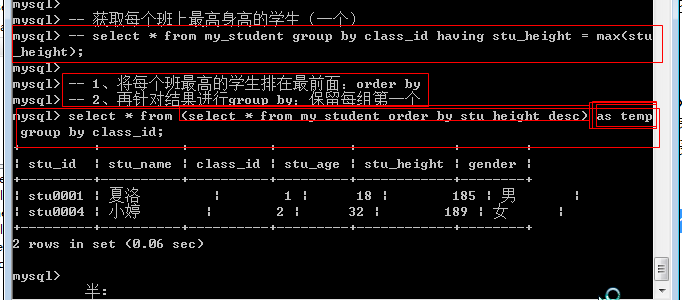
表子查询是用于from数据源：from子查询

## 语法

基本语法：

Select 字段表 from (表子查询) as 别名 [where] [group by] [having] [order by] [limit];

这里第一个查询语句，老师说是错误的。原因，先执行group by，每个class\_id只有一条最上面的记录了，然后再执行having



# Exists子查询

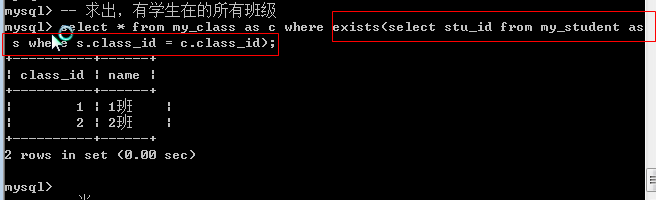
## 概念

Exists子查询：查询返回的结果只有0或者1，1代表成立，0代表不成立

## 语法

基本语法：where exists(查询语句); //exists就是根据查询得到的结果进行判断：如果结果存在，那么返回1，否则返回0

Where 1：永远为真



# 子查询中特定关键字的使用

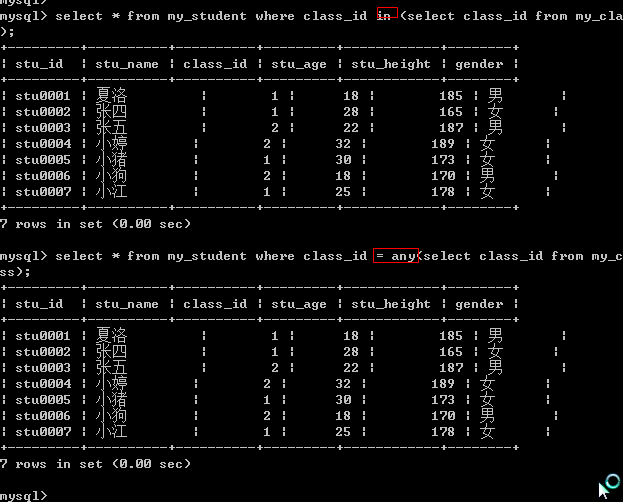
## In

主查询 where 条件 in (列子查询);

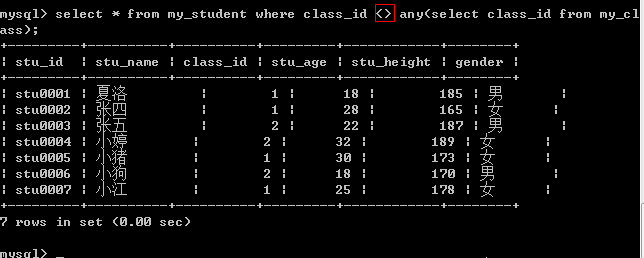
## Any

任意一个

= any(列子查询)：条件在查询结果中有任意一个匹配即可，等价于in



<>any(列子查询)：条件在查询结果中不等于任意一个



1 =any(1,2,3) ===== true 1等于1，2，3中的一个

1 <>any(1,2,3) ===== true 1不等于1，2，3中的一个

## Some

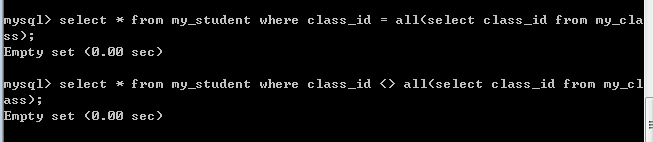
与any完全一样：在国外，some与any的正面含义一致，但是否定就大不相同：not any与not some

开发者为了让对应的使用者不要在语法上纠结：重新设计了some

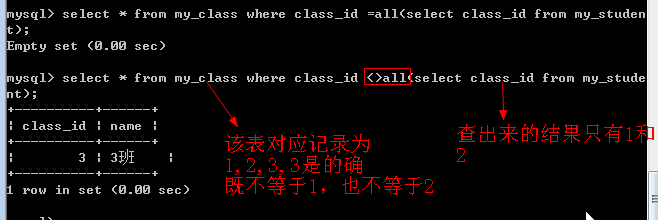
## All

= all(列子查询)：等于里面所有

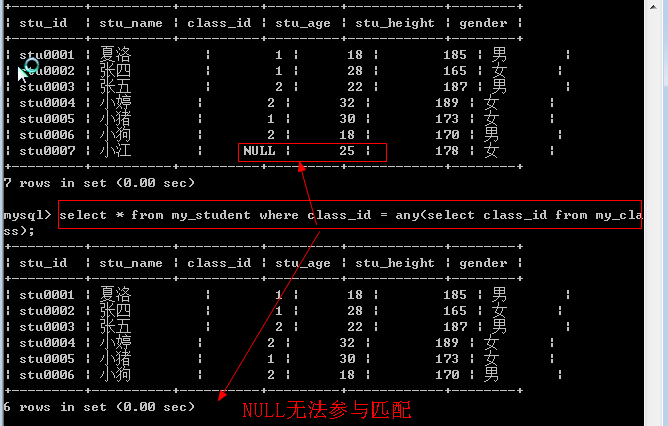
<>all(列子查询)：不等于其中所有



All数据展示



如果对应的匹配字段有NULL，那么不参与匹配



# 整库数据备份与还原

整库数据备份也叫SQL数据备份：备份的结果都是SQL指令

在Mysql中提供了一个专门用于备份SQL的客户端：mysqldump.exe



## 应用场景

SQL备份是一种mysql非常常见的备份与还原方式，SQL备份不只是备份数据，还备份对应的SQL指令（表结构）：即便是数据库遭到毁灭性的破坏（数据库被删），那么利用SQL备份依然可以实现数据还原。

SQL备份因为需要备份结构，因此产生的备份文件特别大，因此不适合特大型数据备份，也不适合数据变换频繁型数据库备份。

## 应用方案

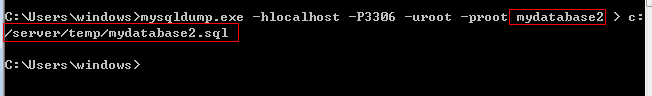
### SQL备份

SQL备份用到的是专门的备份客户端，因此还没与数据库服务器进行连接。

基本语法：mysqldump/mysqldump.exe -hPup 数据库名字 [表1 [表2…]] > 备份文件地址

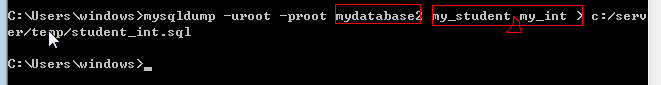
备份可以有三种形式：

1、 整库备份（只需要提供数据库名字）

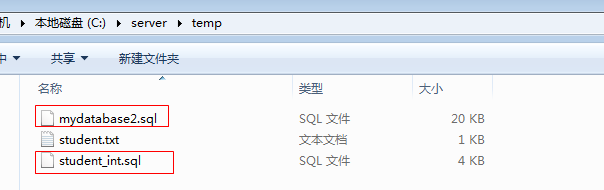


2、 单表备份：数据库后面跟一张表

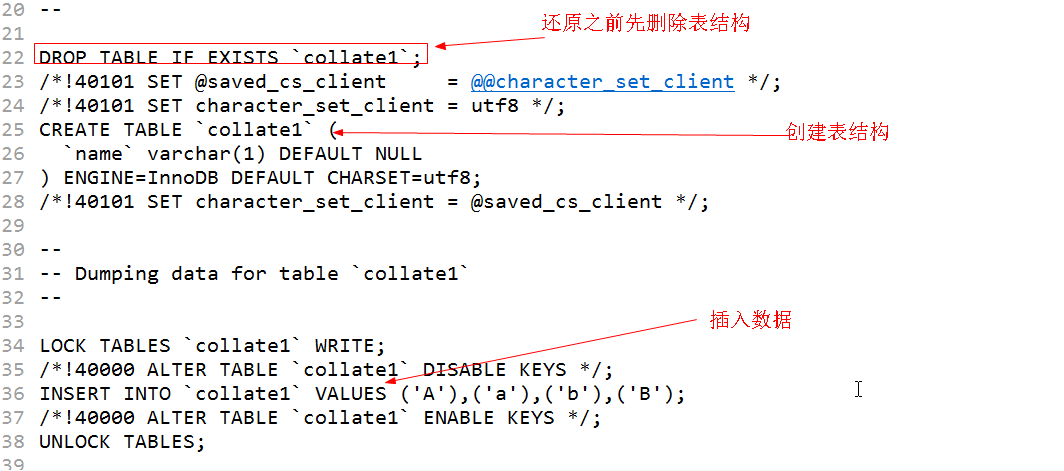
3、 多表备份：数据库后跟多张表（表与表之间空格）



查看备份的成果



查看备份文件中的具体内容



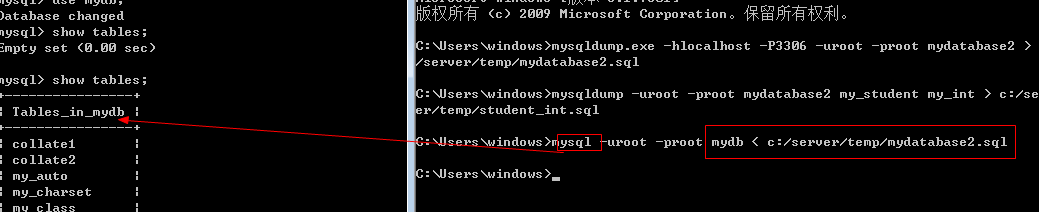
### 数据还原

Mysql提供了多种方式来实现：两种

Mysqldump备份的数据中没有关于数据库本身的操作，都是针对表级别的操作：当进行数据（SQL还原），必须指定数据库

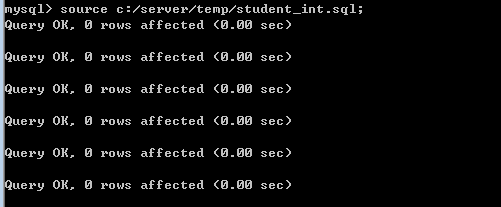
1、 利用mysql.exe客户端：没有登录之前，可以直接用该客户端进行数据还原

Mysql.exe –hPup 数据库 < 文件位置



2、 在SQL指令，提供了一种导入SQL指令的方式

Source SQL文件位置; //必须先进入到对应的数据库



3、 人为操作：打开备份文件，复制所有SQL指令，然后到mysql.exe客户端中去粘贴执行。（不推荐）

# 用户权限管理

用户权限管理：在不同的项目中给不同的角色（开发者）不同的操作权限，为了保证数据库数据的安全。

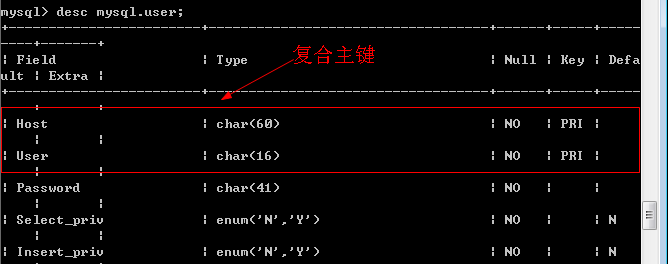
通常，一个用户的密码不会长期不变，所以需要经常性的变更数据库用户密码来确保用户本身安全（mysql客户端用户）

## 用户管理

Mysql需要客户端进行连接认证才能进行服务器操作：需要用户信息。Mysql中所有的用户信息都是保存在mysql数据库下的user表中。



默认的，在安装Mysql的时候，如果不选择创建匿名用户，那么意味着所有的用户只有一个：root超级用户



在mysql中，对用的用户管理中，是由对应的Host和User共同组成主键来区分用户。

User：代表用户的用户名

Host：代表本质是允许访问的客户端（IP或者主机地址）。如果host使用%代表所有的用户（客户端）都可以访问

### 创建用户

理论上讲可以采用两种方式创建用户：

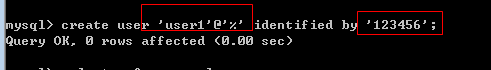
1、 直接使用root用户在mysql.user表中插入记录（不推荐）

2、 专门创建用户的SQL指令

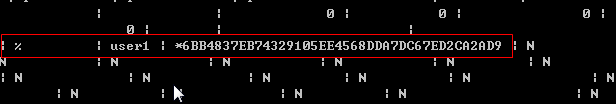
基本语法：create user 用户名 identified by ‘明文密码’;

用户：用户名@主机地址

主机地址：’’ / ‘%’



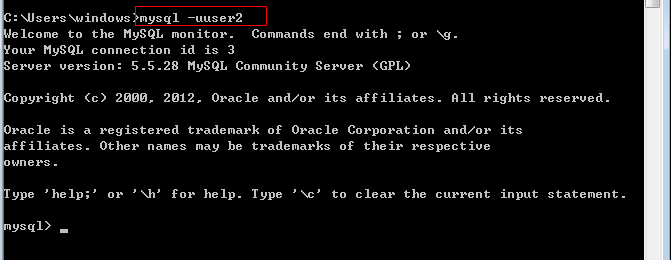
查看mysql.user表中是否存在新增的用户



简化版创建用户（谁都可以访问，不需要密码）不推荐



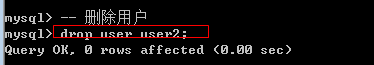
当用户创建完成之后，用户是否可以使用?



### 删除用户

注意：mysql中user是带着host本身的（具有唯一性）

基本语法：drop user 用户名@host;

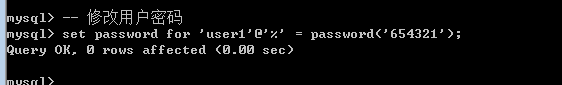


### 修改用户密码

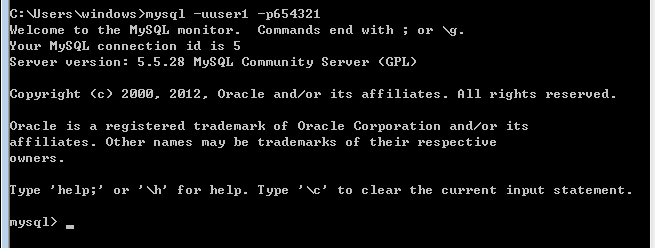
Mysql中提供了多种修改的方式：基本上都必须使用对应提供的一个系统函数：password()，需要靠该函数对密码进行加密处理。

1、 使用专门的修改密码的指令

基本语法：set password for 用户 = password(‘新的明文密码’);



修改后的数据测试



2、 使用更新语句update来修改表

基本语法：update mysql.user set password = password(‘新的明文密码’) where user = ‘’ and host= ‘’;

## 权限管理

在mysql中将权限管理分为三类：

1、 数据权限：增删改查（select\update\delete\insert）

2、 结构权限：结构操作（create\drop）

3、 管理权限：权限管理（create user\grant\revoke）：通常只给管理员如此权限

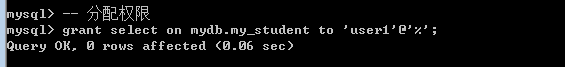
### 授予权限：grant

将权限分配给指定的用户

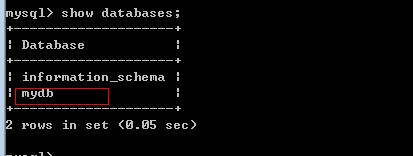
基本语法：grant 权限列表 on 数据库/\*.表名/\* to 用户;

权限列表：使用逗号分隔，但是可以使用all privileges代表全部权限

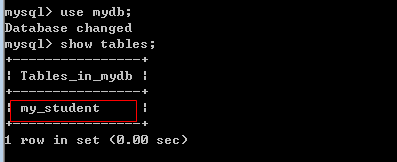
数据库.表名：可以是单表（数据库名字.表名），可以是具体某个数据库（数据库.\*），也可以整库（\*.\*）



用户被分配权限以后不需要退出就可以看到效果



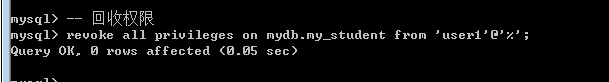
具体权限查看：单表权限只能看到数据库中的一张表



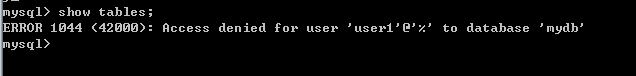
### 取消权限：revoke

权限回收：将权限从用户手中收回

基本语法：revoke 权限列表/all privileges on 数据库/\*.表/\* from 用户;



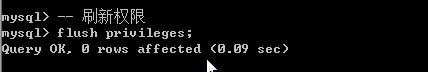
权限回收，同样不需要刷新，用户马上就会感受到



### 刷新权限：flush

Flush：刷新，将当前对用户的权限操作，进行一个刷新，将操作的具体内容同步到对应的表中。

基本语法：flush privileges;



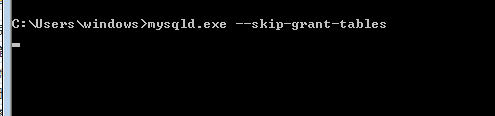
## 密码丢失的解决方案

如果忘记了root用户密码，就需要去找回或者重置root用户密码

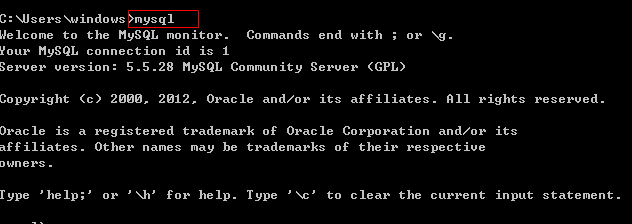
1、 停止服务



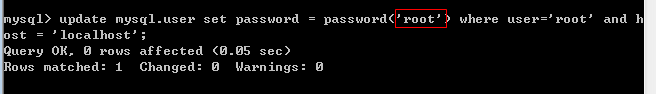
2、 重新启动服务：mysqld.exe –skip-grant-tables //启动服务器但是跳过权限



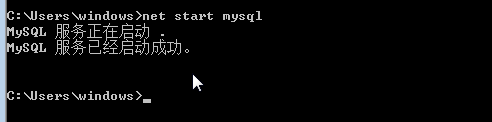
3、 当前启动的服务器没有权限概念：非常危险，任何客户端，不需要任何用户信息都可以直接登录，而且是root权限：新开客户端，使用mysql.exe登录即可



4、 修改root用户的密码：指定 用户名@host



5、 赶紧关闭服务器，重启服务



# 外键

## 外键概念

如果公共关键字在一个关系中是主关键字，那么这个公共关键字被称为另一个关系的外键。由此可见，外键表示了两个关系之间的相关联系。以另一个关系的外键作主关键字的表被称为主表，具有此外键的表被称为主表的从表。外键又称作外关键字。

外键：foreign key

一张表（A）中有一个字段，保存的值指向另外一张表（B）的主键

B：主表

A：从表

## 外键的操作

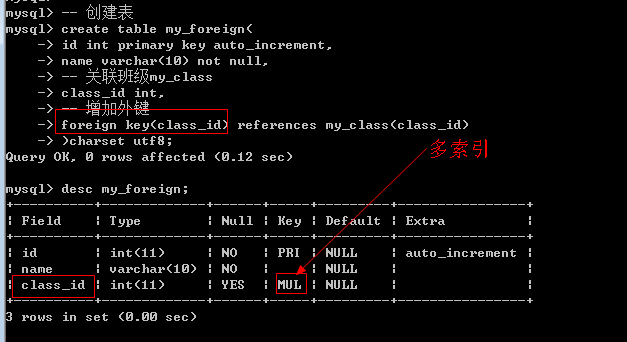
### 增加外键

Mysql中提供了两种方式增加外键

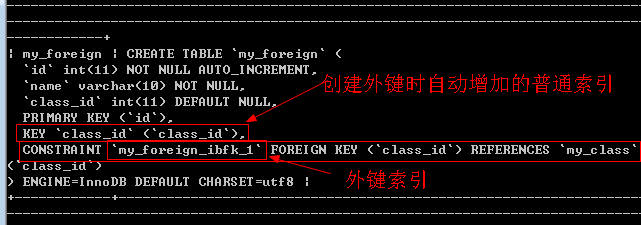
1、 方案1：在创建表的时候增加外键（类似主键）

基本语法：在字段之后增加一条语句

[constraint `外键名`] foreign key(外键字段) references 主表(主键);

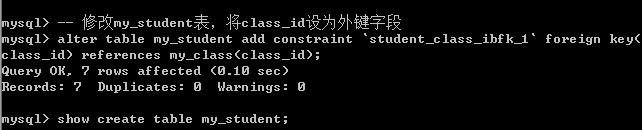


MUL：多索引，外键本身是一个索引，外键要求外键字段本身也是一种普通索引

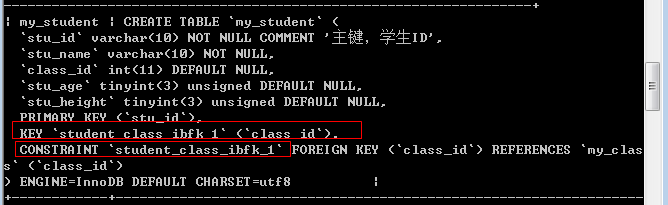


2、 方案2：在创建表后增加外键

Alter table 从表 add [constraint `外键名`] foreign key(外键字段) references 主表(主键);



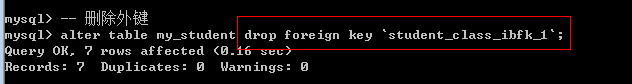
外键名字可以指定



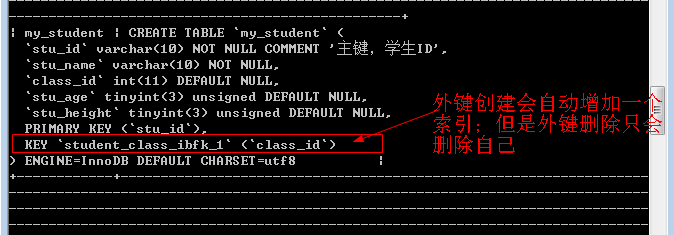
### 修改&删除外键

外键不允许修改，只能先删除后增加

基本语法：alter table 从表 drop foreign key 外键名字;



外键不能删除产生的普通索引，只会删除外键自己



如果想删除对应的索引：alter table 表名 drop index 索引名字;

### 外键基本要求

1、 外键字段需要保证与关联的主表的主键字段类型完全一致；

2、 基本属性也要相同

3、 如果是在表后增加外键，对数据还有一定的要求（从表数据与主表的关联关系）

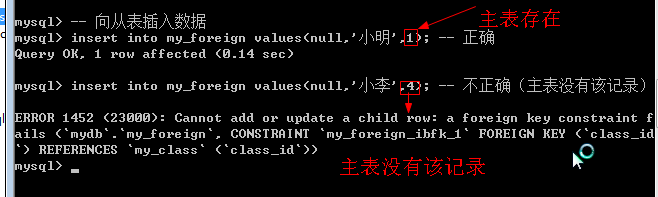
4、 外键只能使用innodb存储引擎：myisam不支持

## 外键约束

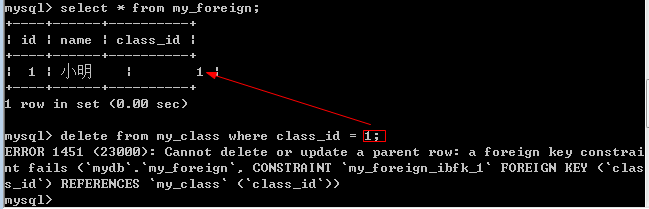
外键约束：通过建立外键关系之后，对主表和从表都会有一定的数据约束效率。

### 约束的基本概念

1、 当一个外键产生时：外键所在的表（从表）会受制于主表数据的存在从而导致数据不能进行某些不符合规范的操作（不能插入主表不存在的数据）；



2、 如果一张表被其他表外键引入，那么该表的数据操作就不能随意：必须保证从表数据的有效性（不能随便删除一个被从表引入的记录）



### 外键约束的概念

可以在创建外键的时候，对外键约束进行选择性的操作。

基本语法： add foreign key(外键字段) references 主表(主键) on 约束模式;

约束模式有三种：

1、 district：严格模式，默认的，不允许操作

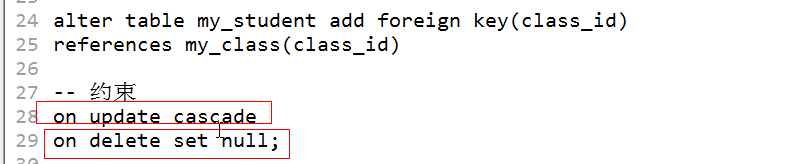
2、 cascade：级联模式，一起操作，主表变化，从表数据跟着变化

3、 set null：置空模式，主表变化（删除），从表对应记录设置为空：前提是从表中对应的外键字段允许为空

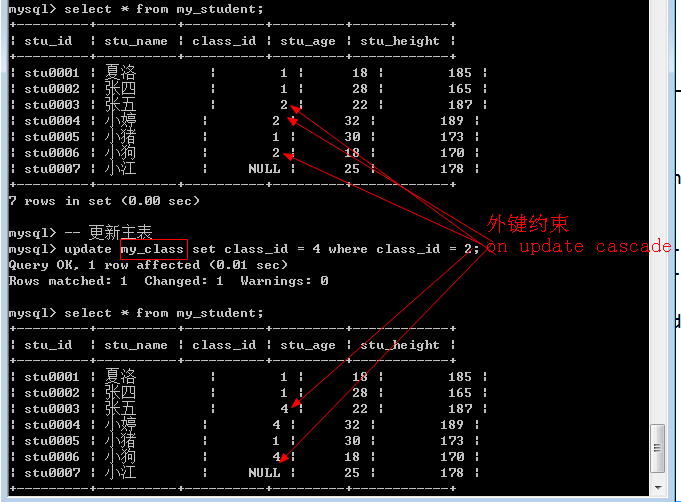
外键约束主要约束的对象是主表操作：从表就是不能插入主表不存在的数据

通常在进行约束时候的时候，需要指定操作：update和delete

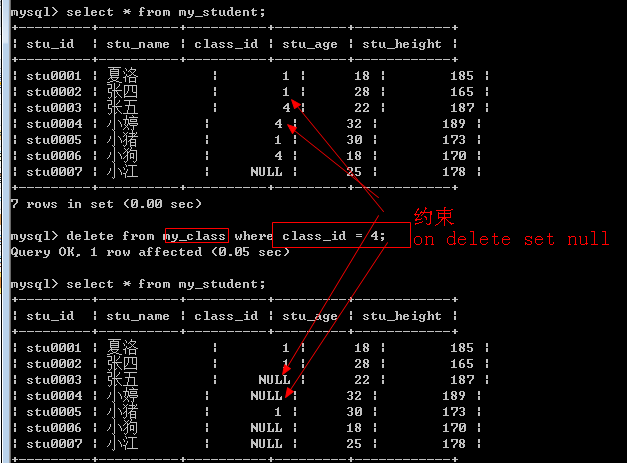
常用的约束模式： on update cascade, on delete set null，更新级联，删除置空



更新模式



删除模式



### 约束作用

保证数据的完整性：主表与从表的数据要一致

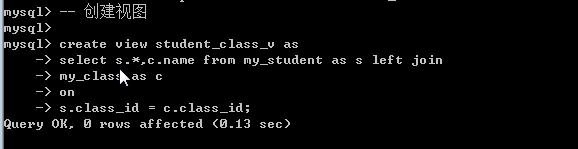
正是因为外键有非常强大的数据约束作用，而且可能导致数据在后台变化的不可控。导致程序在进行设计开发逻辑的时候，没有办法去很好的把握数据（业务），所以外键比较少使用。

# 视图基本操作

## 创建视图

视图的本质是SQL指令（select语句）

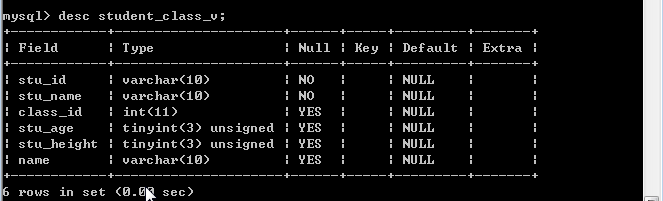
基本语法：create view 视图名字 as select指令; //可以是单表数据，也可以是连接查询，联合查询或者子查询

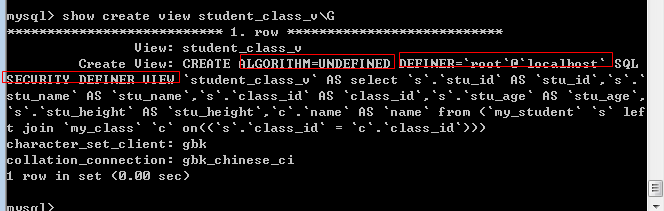


查看视图结构：视图本身是虚拟表，所以关于表的一些操作都适用于视图



Show tables/show create table[view]/desc 视图名字；

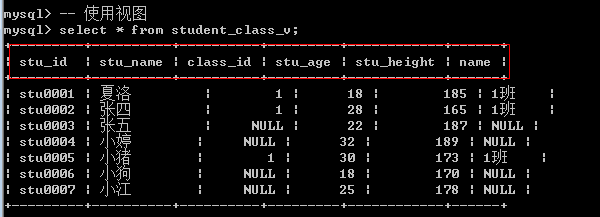




## 使用视图

视图是一张虚拟表：可以直接把视图当做“表”操作，但是视图本身没有数据，是临时执行select语句得到对应的结果。视图主要用于查询操作。

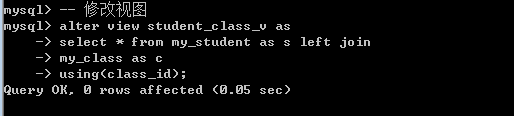
基本语法：select 字段列表 from 视图名字 [子句];



## 修改视图

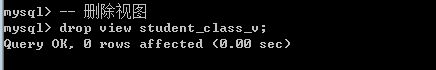
修改视图：本质是修改视图对应的查询语句

基本语法：alter view 视图名字 as 新select指令;



## 删除视图

基本语法：drop view 视图名字;



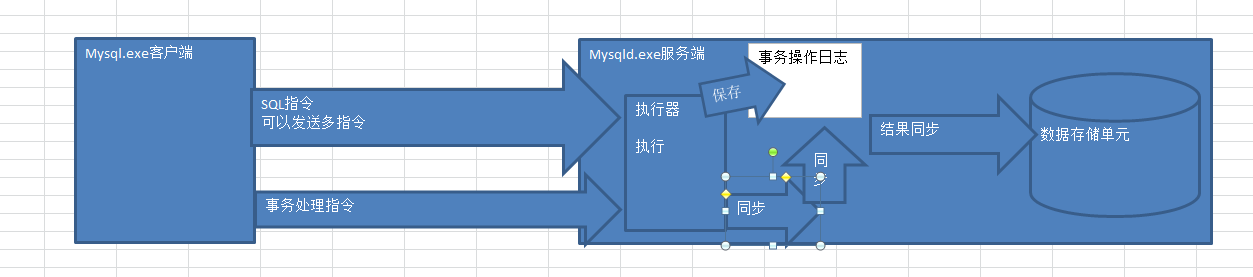
# 事务安全

## 事务概念

事务(Transaction)是访问并可能更新数据库中各种数据项的一个程序执行单元(unit)。事务通常由高级数据库操纵语言或编程语言书写的用户程序的执行所引起。事务由事务开始(begin transaction)和事务结束(end transaction)之间执行的全体操作组成。

## 事务基本原理

基本原理：Mysql允许将事务统一进行管理（存储引擎INNODB），将用户所做的操作，暂时保存起来，不直接放到数据表（更新），等到用于确认结果之后再进行操作。

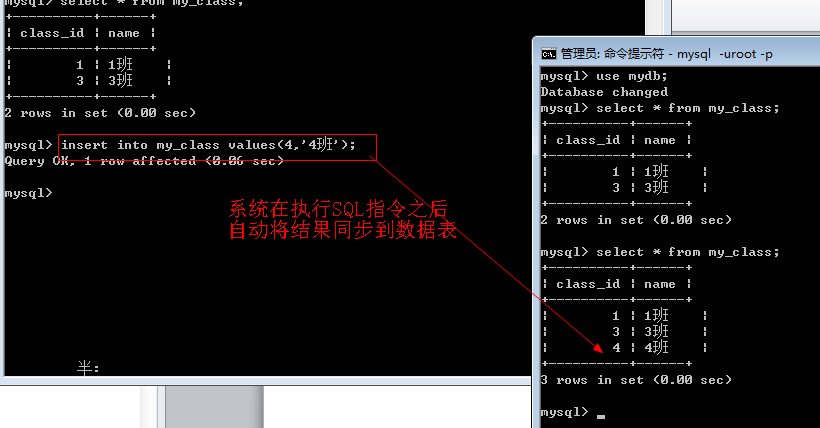


事务在mysql中通常是自动提交的，但是也可以使用手动事务。

# 自动事务

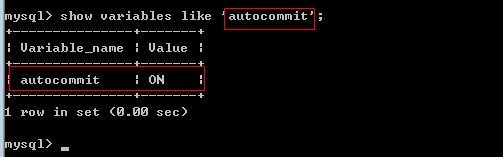
自动事务：autocommit，当客户端发送一条SQL指令（写操作：增删改）给服务器的时候，服务器在执行之后，不用等待用户反馈结果，会自动将结果同步到数据表。

证明：利用两个客户端，一个客户端执行SQL指令，另外一个客户端查看执行结果



自动事务：系统做了额外的步骤来帮助用户操作，系统是通过变量来控制的。Autocommit

Show variables like ‘autocommit%’;

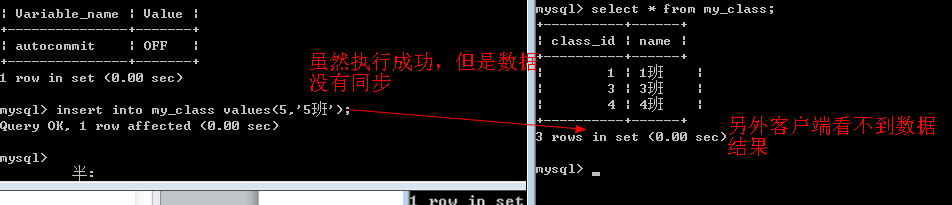


关闭自动事务：关闭之后系统就不在帮助用户提交结果了

Set autocommit = Off;



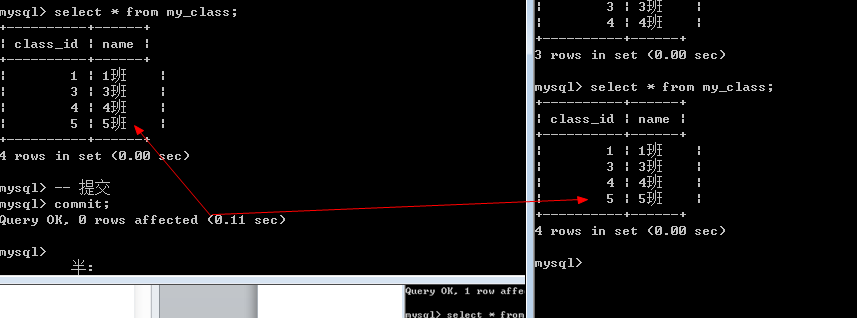
查看执行结果



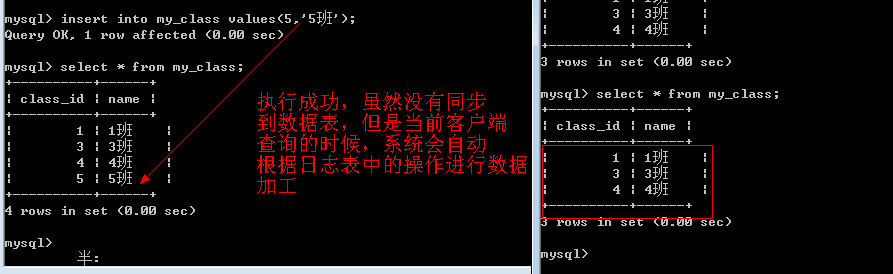
一旦自动事务关闭，那么需要用户提供是否同步的命令

Commit：提交（同步到数据表：事务也会被清空）

Rollback：回滚（清空之前的操作，不要了）



事务没有提交的对比查看：在执行事务端的客户端中，系统在进行数据查看的时候会利用事务日志中保存的结果对数据进行加工



通常，我们不会关闭自动事务：这样操作太麻烦。因此只会在需要使用事务处理的时候，才会进行操作（手动事务）

# 手动事务

手动事务：不管是开始还是过程还是结束都需要用户（程序员），手动的发送事务操作指令来实现。

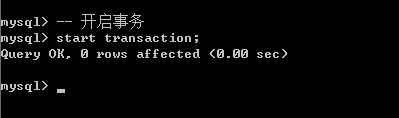
手动事务对应的命令：

1、 start transaction; //开启事务：从这条语句开始，后面的所有语句都不会直接写入到数据表（保存在事务日志中）

2、 事务处理：多个写指令构成

3、 事务提交：commit/rollback，到这个时候所有的事务才算结束

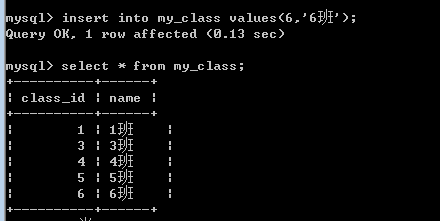
## 开启事务



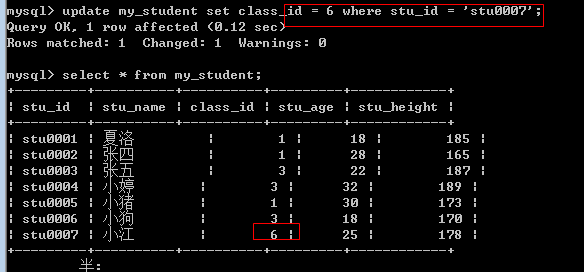
## 执行事务

将多个连续的但是是一个整体的SQL指令，逐一执行

1、 事务操作1：新增数据

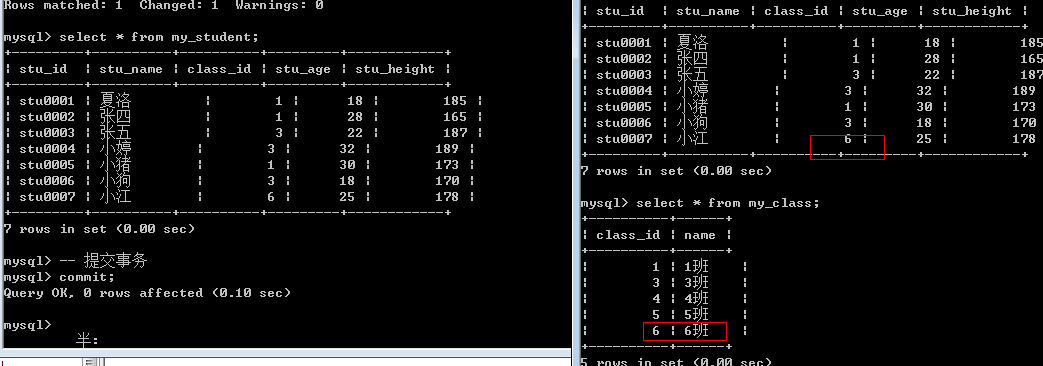


2、 事务操作2：更新数据



## 提交事务

确认提交：commit，数据写到数据表（清空）



回滚操作：rollback，所有数据无效并清空

## 回滚点

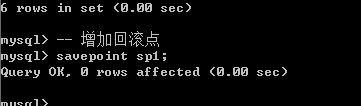
回滚点：savepoint，当有一系列事务操作时，而其中的步骤如果成功了，没有必要重新来过，可以在某个点（成功），设置一个记号（回滚点），然后如果后面有失败，那么可以回到这个记号位置。

增加回滚点：savepoint 回滚点名字; //字母数字和下划线构成

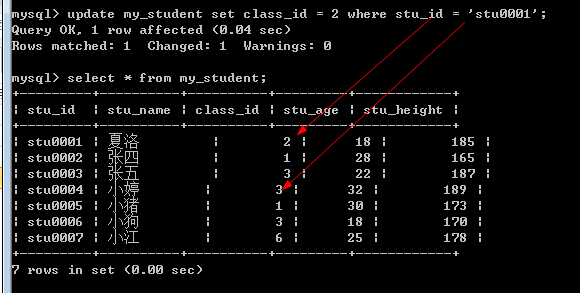
回到回滚点：rollback to 回滚点名字; //那个记号（回滚点）之后的所有操作没有了

注意：在一个事务处理中，如果有很多个步骤，那么可以设置多个回滚点。但是如果回到了前面的回滚点，后面的回滚点就失效；

1、 增加回滚点操作



2、 出现错误步骤



3、 回到正确点：回滚



# 事务特点

事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。

原子性（atomicity）。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。

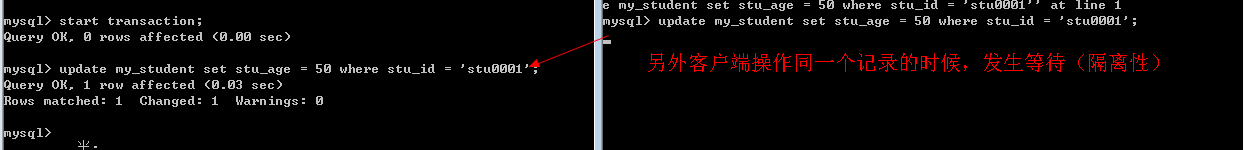
事务从start transaction起到提交事务（commit或者rollback），要么所有的操作都成功，要么就是所有的操作都失败；

一致性（consistency）。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。

数据表中的数据修改，要么是所有操作一次性修改，要么是根本不懂

隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

如果一个客户端在使用事务操作一个数据（可能是一行/整表）的时候，另外一个客户端不能对该数据进行操作



什么时候是行被隔离？什么时候是整表被隔离？

说明：如果条件中使用了索引（主键），那么系统是根据主键直接找到某条记录，这个时候与其他记录无关，那么只隔离一条记录；反之，如果说系统是通过全表检索（每一条记录都去检查：没有索引），被检索的所有数据都会被锁定（整表）

持久性（durability）。持久性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。