# 数据库命令及维护

我的LNAG=C.UTF-8,在本地可以输入终端,但是在mysql控制台却无法输入中文

bash-4.2# env

HOSTNAME=9c7d70c1dfd8

TERM=xterm

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

PWD=/

LANG=C.UTF-8

SHLVL=1

HOME=/root

\_=/usr/bin/env

我们先改变本地LNAG属性的值为en\_US.UTF-8

export LANG=en\_US.UTF-8

然后在链接mysql

mysql -uroot -p

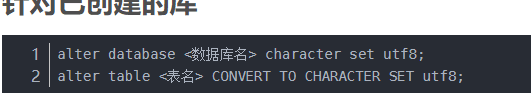
此时应该已经解决了无法输入中文的问题,如果出现在mysql控制台查看数据库中文乱码,此时则需要修改数据库的字符集

我们先使用show variables 命令查看字符集目前的属性值,如果出现database connection client字符集不是utf8的情况

Create table name (

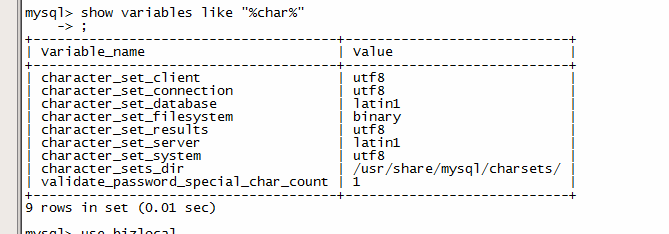
)default charset=utf8 表支持中文

Create database name2 charset utf8; 数据库支持中文



修改它们的属性值即可

Show variables like “%char%”;



可在/etc/my.cnf修改character\_set\_server=utf8，默认支持中文，不然默认创建数据库编码是latin1，不支持中文

MySQL5.7之后多了一个备份工具：mysqlpump。它是mysqldump的一个衍生，mysqldump就不多说明了，现在看看mysqlpump到底有了哪些提升，可以查看[官方文档](https://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html" \t "https://www.cnblogs.com/zhoujinyi/p/_blank)，这里针对如何使用做下说明。

mysqlpump和mysqldump一样，属于逻辑备份，备份以SQL形式的文本保存。逻辑备份相对物理备份的好处是不关心undo log的大小，直接备份数据即可。它最主要的特点是：

* **并行备份**数据库和数据库中的对象的，加快备份过程。
* 更好的控制数据库和数据库对象（表，存储过程，用户帐户）的备份。
* 备份用户账号作为帐户管理语句（CREATE USER，GRANT），而不是直接插入到MySQL的系统数据库。
* 备份出来直接生成压缩后的备份文件。
* 备份进度指示（估计值）。
* 重新加载（还原）备份文件，先建表后插入数据最后建立索引，减少了索引维护开销，加快了还原速度。
* 备份可以排除或则指定数据库。

show processlist 可以看到线程处理

ql和mysql执行顺序，发现内部机制是一样的。最大区别是在别名的引用上。 

#### 数据库备份还原：

备份全部数据库：

mysqldump --all-databases -u root -p密码 > /mysqldata/mysql$rq.sql 或者：

mysqldump -A -u root -p密码 > /mysqldata/mysql$rq.sql

备份单个数据库：mysqldump -u$db\_user -p$db\_passwd $db\_name >> /home/backup/$name.sql

备份单个表：mysqldump -u$db\_user -p$db\_passwd $db\_name $table\_name >> /home/backup/$name.sql

只备份数据库结构，不备份数据：

mysqldump -u$db\_user -p$db\_passwd -d $db\_name >> /home/backup/$name.sql

只备份数据库表结构，不备份数据：

mysqldump -u$db\_user -p$db\_passwd -d $db\_name $table\_name >> /home/backup/$name.sql

恢复数据库或者表：

mysql -u$db\_user -p$db\_passwd $db\_name < /home/backup/$name.sql

或者登陆mysql，创建库，用source /home/backup/$name.sql

从压缩文件直接恢复：#gzip < tm\_050519.sql.gz | mysql -u root -p tm

若是全备数据库：mysql -u$db\_user -p$db\_passwd < /home/backup/$name.sql，不需要写数据库名

**导入数据：**　　由于mysqldump导出的是完整的SQL语句，所以用mysql客户程序很容易就能把数据导入了：

复制代码代码如下:

#mysql　数据库名　<　文件名   
#source　/tmp/xxx.sql

一、导出

命令行下具体用法如下：

导出某个表的结构

mysqldump -u用戶名 -p密码 -d 数据库名 表名 > 脚本名;

导出整个数据库结构和数据

mysqldump -h localhost -uroot -p123456 database > dump.sql

导出单个数据表结构和数据

mysqldump -h localhost -uroot -p123456 database table > dump.sql

导出整个数据库结构（不包含数据）

mysqldump -h localhost -uroot -p123456 -d database > dump.sql

导出单个数据表结构（不包含数据）

mysqldump -h localhost -uroot -p123456 -d database table > dump.sql

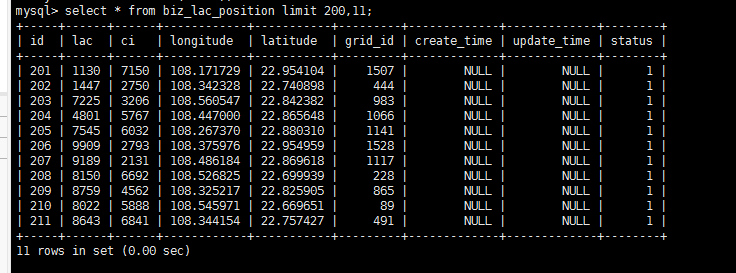
MYSQL分页：

因为数据量过大，如果一次性查询会耗用大量时间以及性能，因此对数据进行分页显示变得尤为重要

MySQL实现分页效果比较简单，只有一个limit关键字就可以解决。

示例：SELECT username,password FROM tb\_user WHERE id = 1 LIMIT 100,10;  
具体：select \* from tableName where 条件 limit 当前页码\*页面容量-1,页面容量

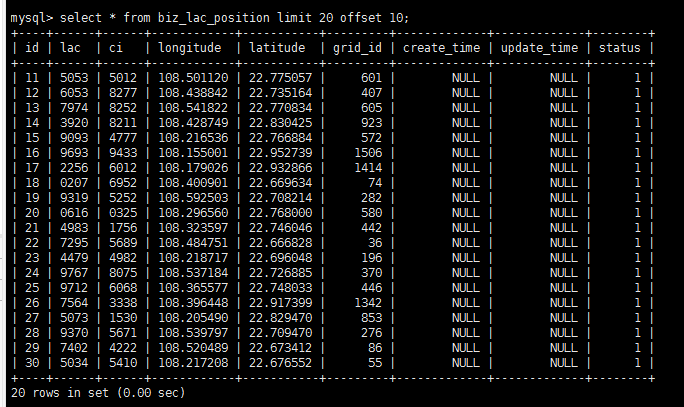
select \* from biz\_lac\_position limit 200,11; #选择id号200之后（不包括200），总共11条信息



select 查询列表from 表

limit (page-1)\*size,size; #假如size=10，page=11

select \* from biz\_lac\_position limit 20 offset 10 #从id为10开始，选择20行，相当于limit 10,20,-- limit用法如下  
-- 第一个参数是指要开始的地方，第二个参数是指每页显示多少条数据  
-- 第一页用0表示



一、sql执行顺序   
(1)from

1. on   
   (3) join   
   (4) where   
   (5)group by(开始使用select中的别名，后面的语句中都可以使用)  
   (6) 计算函数，avg,sum....   
   (7)having   
   (8) select   
   (9) distinct   
   (10) order by   
     
   从这个顺序中我们不难发现，所有的 查询语句都是从from开始执行的，在执行过程中，每个步骤都会为下一个步骤生成一个虚拟表，这个虚拟表将作为下一个执行步骤的输入。   
   第一步：首先对from子句中的前两个表执行一个笛卡尔乘积，此时生成虚拟表 vt1（选择相对小的表做基础表）   
   第二步：接下来便是应用on筛选器，on 中的逻辑表达式将应用到 vt1 中的各个行，筛选出满足on逻辑表达式的行，生成虚拟表 vt2   
   第三步：如果是outer join 那么这一步就将添加外部行，left outer jion 就把左表在第二步中过滤的添加进来，如果是right outer join 那么就将右表在第二步中过滤掉的行添加进来，这样生成虚拟表 vt3   
   第四步：如果 from 子句中的表数目多余两个表，那么就将vt3和第三个表连接从而计算笛卡尔乘积，生成虚拟表，该过程就是一个重复1-3的步骤，最终得到一个新的虚拟表 vt3。   
   第五步：应用where筛选器，对上一步生产的虚拟表引用where筛选器，生成虚拟表vt4，在这有个比较重要的细节不得不说一下，对于包含outer join子句的查询，就有一个让人感到困惑的问题，到底在on筛选器还是用where筛选器指定逻辑表达式呢？on和where的最大区别在于，如果在on应用逻辑表达式那么在第三步outer join中还可以把移除的行再次添加回来，而where的移除的最终的。举个简单的例子，有一个学生表（班级,姓名）和一个成绩表(姓名,成绩)，我现在需要返回一个x班级的全体同学的成绩，但是这个班级有几个学生缺考，也就是说在成绩表中没有记录。为了得到我们预期的结果我们就需要在on子句指定学生和成绩表的关系（学生.姓名=成绩.姓名）那么我们是否发现在执行第二步的时候，对于没有参加考试的学生记录就不会出现在vt2中，因为他们被on的逻辑表达式过滤掉了,但是我们用left outer join就可以把左表（学生）中没有参加考试的学生找回来，因为我们想返回的是x班级的所有学生，如果在on中应用学生.班级='x'的话，left outer join会把x班级的所有学生记录找回（感谢网友康钦谋\_\_康钦苗的指正），所以只能在where筛选器中应用学生.班级='x' 因为它的过滤是最终的。

第六步：group by 子句将中的唯一的值组合成为一组，得到虚拟表vt5。如果应用了group by，那么后面的所有步骤都只能得到的vt5的列或者是聚合函数（count、sum、avg等）。原因在于最终的结果集中只为每个组包含一行。这一点请牢记。   
第七步：应用cube或者rollup选项，为vt5生成超组，生成vt6.   
第八步：应用having筛选器，生成vt7。having筛选器是第一个也是为唯一一个应用到已分组数据的筛选器。   
第九步：处理select子句。将vt7中的在select中出现的列筛选出来。生成vt8.   
第十步：应用distinct子句，vt8中移除相同的行，生成vt9。事实上如果应用了group by子句那么distinct是多余的，原因同样在于，分组的时候是将列中唯一的值分成一组，同时只为每一组返回一行记录，那么所以的记录都将是不相同的。   
第十一步：应用order by子句。按照order\_by\_condition排序vt9，此时返回的一个游标，而不是虚拟表。sql是基于集合的理论的，集合不会预先对他的行排序，它只是成员的逻辑集合，成员的顺序是无关紧要的。对表进行排序的查询可以返回一个对象，这个对象包含特定的物理顺序的逻辑组织。这个对象就叫游标。正因为返回值是游标，那么使用order by 子句查询不能应用于表表达式。排序是很需要成本的，除非你必须要排序，否则最好不要指定order by，最后，在这一步中是第一个也是唯一一个可以使用select列表中别名的步骤。   
第十二步：应用top选项。此时才返回结果给请求者即用户。

# 用xtrabackup软件完全热备份

innobackupex --defaults-file=/etc/my.cnf --socket=/app/mysql/mysql.sock --user=root --password=123456 (--databases=zabbix) /app/innobackup（完全备份的路径）

放在后台运行：nohup innobackupex ... &

在使用innobackupex进行备份时，还可以使用--no-timestamp选项来阻止命令自动创建一个以时间命名的目录：如此一来，innobackupex命令将会创建一个BACKUP-DIR目录来存储备份数据。

测试端准备(prepare)一个完全备份:

innobackupex --apply-log /app/innobackup/2018-05-21\_15-02-53/

#关闭数据库：

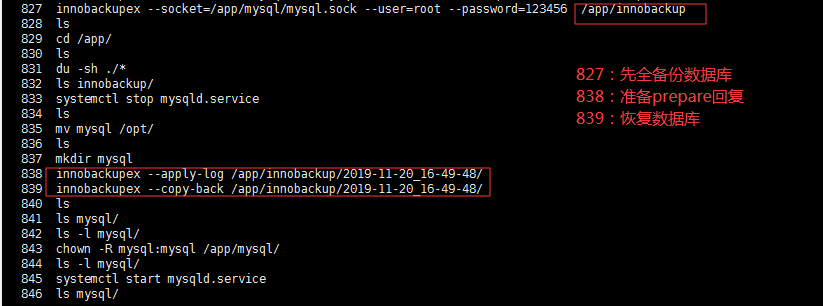
[root@test backup]# systemctl stop mysqld.service

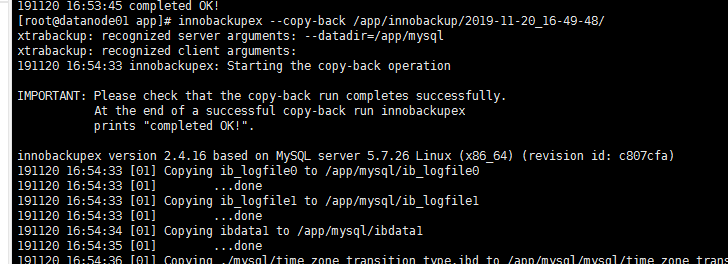
#恢复数据：

[root@test backup]# innobackupex --copy-back /app/innobackup/2018-05-21\_15-02-53/

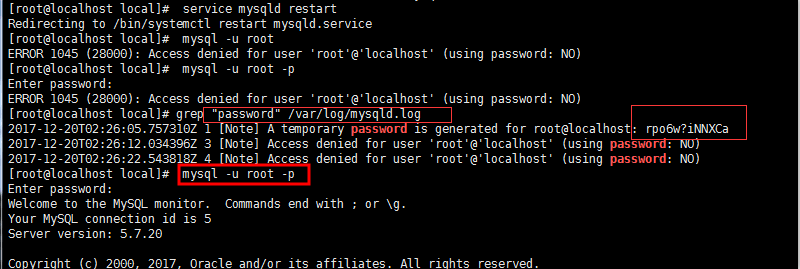
Original data directory /app/wislife is not empty!

#执行上面的命令需要保证数据库目录是空的，否则会失败，因为目录里有binlog日志，要么移走，要么就用手动复制需要的文件。所以说日志不要和数据放在一起！





初始密码：/var/log/mysqld.log



分析MySQL语句查询性能的方法除了使用 EXPLAIN 输出执行计划，还可以让MySQL记录下查询超过指定时间的语句，我们将超过指定时间的[SQL语句](https://baike.baidu.com/item/SQL%E8%AF%AD%E5%8F%A5/5714895" \t "https://baike.baidu.com/item/%E6%85%A2%E6%9F%A5%E8%AF%A2/_blank)查询称为“慢查询”。

**show processlist 命令**

SHOW PROCESSLIST显示哪些线程正在运行。您也可以使用**mysqladmin processlist**语句得到此信息。

**各列的含义和用途：**

ID列

一个标识，你要kill一个语句的时候很有用，用命令杀掉此查询 /\*/mysqladmin kill 进程号。

user列

显示单前用户，如果不是root，这个命令就只显示你权限范围内的sql语句。

host列

显示这个语句是从哪个ip的哪个端口上发出的。用于追踪出问题语句的用户。

db列

显示这个进程目前连接的是哪个数据库。

command列

显示当前连接的执行的命令，一般就是休眠（sleep），查询（query），连接（connect）。

time列

此这个状态持续的时间，单位是秒。

state列

显示使用当前连接的sql语句的状态，很重要的列，后续会有所有的状态的描述，请注意，state只是语句执行中的某一个状态，一个 sql语句，以查询为例，可能需要经过copying to tmp table，Sorting result，Sending data等状态才可以完成

info列

显示这个sql语句，因为长度有限，所以长的sql语句就显示不全，但是一个判断问题语句的重要依据。

**这个命令中最关键的就是state列，mysql列出的状态主要有以下几种：**

**Checking table**  
　正在检查数据表（这是自动的）。  
**Closing tables**  
　正在将表中修改的数据刷新到磁盘中，同时正在关闭已经用完的表。这是一个很快的操作，如果不是这样的话，就应该确认磁盘空间是否已经满了或者磁盘是否正处于重负中。  
**Connect Out**  
　复制从服务器正在连接主服务器。

**Copying to tmp table on disk**  
　由于临时结果集大于tmp\_table\_size，正在将临时表从内存存储转为磁盘存储以此节省内存。  
**Creating tmp table**  
　正在创建临时表以存放部分查询结果。  
**deleting from main table**  
　服务器正在执行多表删除中的第一部分，刚删除第一个表。  
**deleting from reference tables**  
　服务器正在执行多表删除中的第二部分，正在删除其他表的记录。

**Flushing tables**  
　正在执行FLUSH TABLES，等待其他线程关闭数据表。  
**Killed**  
　发送了一个kill请求给某线程，那么这个线程将会检查kill标志位，同时会放弃下一个kill请求。MySQL会在每次的主循环中检查kill标志位，不过有些情况下该线程可能会过一小段才能死掉。如果该线程程被其他线程锁住了，那么kill请求会在锁释放时马上生效。  
**Locked**  
　被其他查询锁住了。  
**Sending data**  
　正在处理SELECT查询的记录，同时正在把结果发送给客户端。

**Sorting for group**  
　正在为GROUP BY做排序。  
　Sorting for order  
　正在为ORDER BY做排序。  
**Opening tables**  
　这个过程应该会很快，除非受到其他因素的干扰。例如，在执ALTER TABLE或LOCK TABLE语句行完以前，数据表无法被其他线程打开。正尝试打开一个表。  
**Removing duplicates**  
　正在执行一个SELECT DISTINCT方式的查询，但是MySQL无法在前一个阶段优化掉那些重复的记录。因此，MySQL需要再次去掉重复的记录，然后再把结果发送给客户端。

**Reopen table**  
　获得了对一个表的锁，但是必须在表结构修改之后才能获得这个锁。已经释放锁，关闭数据表，正尝试重新打开数据表。  
**Repair by sorting**  
　修复指令正在排序以创建索引。  
**Repair with keycache**  
　修复指令正在利用索引缓存一个一个地创建新索引。它会比Repair by sorting慢些。  
**Searching rows for update**  
　正在讲符合条件的记录找出来以备更新。它必须在UPDATE要修改相关的记录之前就完成了。  
**Sleeping**  
　正在等待客户端发送新请求.

**System lock**  
　正在等待取得一个外部的系统锁。如果当前没有运行多个mysqld服务器同时请求同一个表，那么可以通过增加--skip-external-locking参数来禁止外部系统锁。  
**Upgrading lock**  
　INSERT DELAYED正在尝试取得一个锁表以插入新记录。  
**Updating**  
　正在搜索匹配的记录，并且修改它们。

**User Lock**  
　正在等待GET\_LOCK()。  
**Waiting for tables**  
　该线程得到通知，数据表结构已经被修改了，需要重新打开数据表以取得新的结构。然后，为了能的重新打开数据表，必须等到所有其他线程关闭这个表。以下几种情况下会产生这个通知：FLUSH TABLES tbl\_name, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE,或OPTIMIZE TABLE。  
**waiting for handler insert**  
　INSERT DELAYED已经处理完了所有待处理的插入操作，正在等待新的请求。  
　大部分状态对应很快的操作，只要有一个线程保持同一个状态好几秒钟，那么可能是有问题发生了，需要检查一下。  
　还有其他的状态没在上面中列出来，不过它们大部分只是在查看服务器是否有存在错误是才用得着。

**explain来了解SQL执行的状态**

explain显示了mysql如何使用索引来处理select语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。

**使用方法，在select语句前加上explain就可以了**：

例如：

explain select surname,first\_name form a,b where a.id=b.id

**EXPLAIN列的解释**

table

显示这一行的数据是关于哪张表的

type

这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为const、eq\_reg、ref、range、indexhe和ALL

possible\_keys

显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中选择一个合适的语句

key

实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引。这种情况下，可以在SELECT语句 中使用USE INDEX（indexname）来强制使用一个索引或者用IGNORE INDEX（indexname）来强制MYSQL忽略索引

key\_len

使用的索引的长度。在不损失精确性的情况下，长度越短越好

ref

显示索引的哪一列被使用了，如果可能的话，是一个常数

rows

MYSQL认为必须检查的用来返回请求数据的行数

Extra

关于MYSQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，意思MYSQL根本不能使用索引，结果是检索会很慢

**extra列返回的描述的意义**

Distinct

一旦MYSQL找到了与行相联合匹配的行，就不再搜索了

Not exists

MYSQL优化了LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行，就不再搜索了

Range checked for each Record（index map:#）

没有找到理想的索引，因此对于从前面表中来的每一个行组合，MYSQL检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一

Using filesort

看到这个的时候，查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行

Using index

列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候

Using temporary

看到这个的时候，查询需要优化了。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上

Where used

使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型ALL或index，这就会发生，或者是查询有问题不同连接类型的解释（按照效率高低的顺序排序）

const

表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行，这个值实际就是常数，因为MYSQL先读这个值然后把它当做常数来对待

eq\_ref

在连接中，MYSQL在查询时，从前面的表中，对每一个记录的联合都从表中读取一个记录，它在查询使用了索引为主键或惟一键的全部时使用

ref

这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如，利用最左边前缀）时发生。对于之前的表的每一个行联合，全部记录都将从表中读出。这个类型严重依赖于根据索引匹配的记录多少—越少越好

range

这个连接类型使用索引返回一个范围中的行，比如使用>或<查找东西时发生的情况

index

这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好，因为索引一般小于表数据）

ALL

这个连接类型对于前面的每一个记录联合进行完全扫描，这一般比较糟糕，应该尽量避免

**子查询及问题**

子查询的写法：

select userid from play\_fav where play\_id=(select id from play\_list where play\_name = '老男孩');

别名 可以不用使用AS 关键字，直接空格加别名就可以了；

子查询在性能上有一定劣势，不利于mysql性能优化器进行优化；

因为内层表和驱动表用户自己定死了，而联结的驱动表和内层表 性能优化器 会根据实际情况 来定；

子查询为什么不利于优化：

联结是嵌套循环查询实现；

如select \* from play\_list,play\_fav where play\_list.id = play\_fav.play\_id;

play\_list驱动表（where等号左边）；内层表play\_fav（where等号右边）;

遍历去东北play\_list.id，找到一个id后再去play\_fav.play\_id中找；依次循环下去；

内层表此时可以查询一次或者几次**索引**，便可以得到；

所以基本的优化就是将表量比较小的作为驱动表，这样减少了循环的次数；

**union**

作用：把不同表中相同的字段聚合在一个结果集中返回给用户

场景8：老板想看创建和收藏歌单的所有用户，查询play\_list和play\_fav两表中所有的userid;

mysql> select userid from play\_list

-> union

-> select userid from play\_fav;

默认的union 会对结果集进行去重处理，不想去重使用 union all；

（3）复制db2.a1的表结构和记录

# 这就是复制表的操作（既复制了表结构，又复制了记录）

mysql> create table b1 select \* from db2.a1;

Query OK, 2 rows affected (0.03 sec)

ps1：如果只要表结构，不要记录

#在db2数据库下新创建一个b2表，给一个where条件，条件要求不成立，条件为false，只拷贝表结构

mysql> create table b2 select \* from db2.a1 where 1>5;

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

ps2:还有一种做法，使用like(只拷贝表结构，不拷贝记录)

[IMG_256](https://www.cnblogs.com/MayDayTime/p/javascript:void(0);)

IMG_257

mysql> create table b3 like db2.a1;

Query OK, 0 rows affected (0.01 sec)

mysql> desc b3;

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

| Field | Type | Null | Key | Default | Extra |

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

| id | int(11) | YES | | NULL | |

| name | varchar(50) | YES | | NULL | |

| age | int(3) | YES | | NULL | |

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

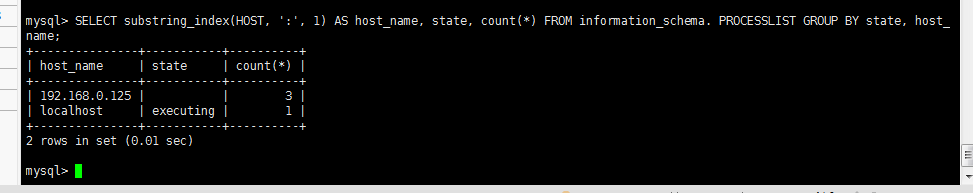
3 rows in set (0.02 sec)

mysql> select \* from db3.b3;

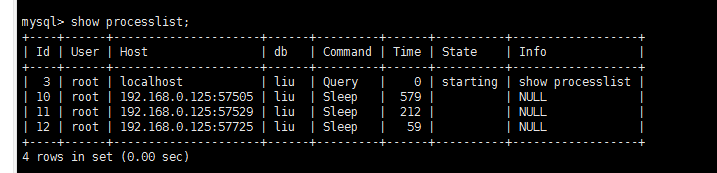
Empty set (0.00 sec)

查看当前有谁连接数据库，并统计连接数。

SELECT substring\_index(HOST, ':', 1) AS host\_name, state, count(\*) FROM information\_schema. PROCESSLIST GROUP BY state, host\_name;name;

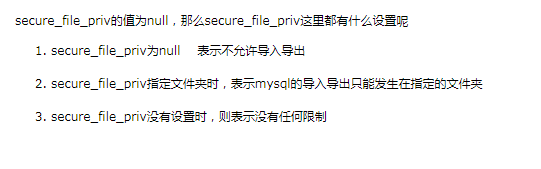


Show processlist; 也可查看连接数情况



导入文件的数据：

 1、在mysql中执行  show variables like '%secure%';  查看 secure-file-priv 当前的值是什么



可在/etc/my.cnf中设置 secure-file-priv= ，导入不受限制，再重启数据库

Systemctl restart mysqld.service

1. 导入csv文件，前提是该文件有mysql用户读的权限，若不能读，会报permission denied，格式

#导入数据中不包含中文，local要加，不然会报permission denied  
load data local infile '/home/wislife/test1.csv' --CSV文件存放路径

into table student--要将数据导入的表名

fields terminated by ',' optionally enclosed by '"' escaped by '"' --字段以逗号分隔，内容用””括起来的

lines terminated by '\r\n';

#导入数据中包含中文，local要加，不然会报permission denied  
  
load data local infile '/home/wislife/test1.csv' --CSV文件存放路径

into table test character set utf8 --要将数据导入的表名

fields terminated by ',' optionally enclosed by '"' escaped by '"'

lines terminated by '\r\n' --行间隔符

ignore 1 lines; --忽略第一行，有时候第一行是字段名，插入不了

Source /路径/sss.sql

**导出**

select \* from 表名 into outfile '导出路径\\test.csv' fields terminated by ',' optionally enclosed by '"' escaped by '"' lines terminated by '\n';

# 降CSV导入hive

将csv文件导入hive后出现了所有的字段只显示在新建的表的第一个字段中，后面的字段全是null.

出现这种的原因是hive以行分隔数据，需要修改为按逗号'  ,  ‘ 进行分隔读取，

具体操作如下，

首先在hive中创建自己的table,  并且设置以逗号分隔。

 create table IP(ip varchar(30), country varchar(30), province varchar(30), city varchar(30), district varchar(30), linetype varchar(30))

 row format delimited fields terminated by ',' ;

然后再输入导入的语句：

load data local inpath '/usr/testFile/result.csv' overwrite into table biao;

（load data local inpath '文件路径' overwrite into table 表名;） #该地址是本地服务器的地址，不用上传至hive集群，overwrite会覆盖数据，最好不要用overwrite，省略掉

然后最后查询

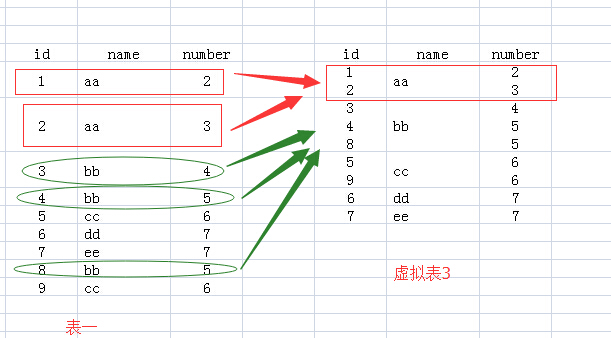
show tables;

select \* from ip limit 100;

1. sql执行顺序   
   (1)from
2. on  
   (3) join    
   (4) where   
   (5)group by(开始使用select中的别名，后面的语句中都可以使用)  
   (6) 计算函数，聚合函数，avg,sum，count，max，min....   
   (7)having   
   (8) select   
   (9) distinct   
   (10) order by

SELECT name FROM test

GROUP BY name



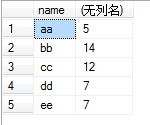
就要接下来针对虚拟表3执行选择语句了：

（1）如果执行select \*的话，那么返回的结果应该是虚拟表3，可是id和数中有的单元格里面的内容是多个值的，而关系数据库就是基于关系的，单元格中是不允许有多个值的，所以你看，执行select \*语句就报错了。

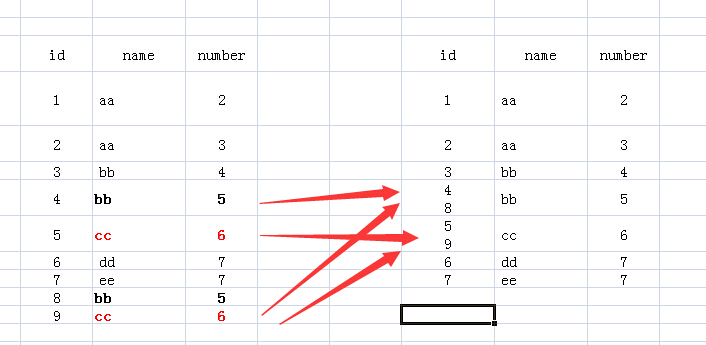
（2）我们再看名列，每个单元格只有一个数据，所以我们选择名称的话，就没有问题了。为什么名称列每个单元格只有一个值呢，因为我们就是用名列出来的。

（3）那么对于ID和号码里面的单元格有多个数据的情况怎么办呢？就是答案用聚合函数，聚合函数就用来输入多个数据，输出一个数据的。如COUT（ID），和（数），而每个聚合函数的输入就是每一个多数据的单元格。

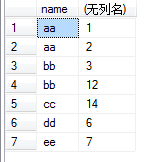
1. 例如我们执行选择名称，来自测试组的总和（数字）名称，那么总和就虚拟表3的数字列的每个单元格进行sum操作，例如对名称为aa的那一行的数字列执行综上所述操作，即2 + 3，返回图5，最后执行结果如下：



1. group by多个字段该怎么理解呢：如group by name，number，我们可以把名称和数字看成一个****整体字段****，以他们整体来进行



1. 接下来就可以配合选择和聚合函数进行操作了。如执行选择名称，来自测试组的总和（id）按名称，数字，结果如下图：



case when语句，用于计算条件列表并返回多个可能结果表达式之一。

CASE 具有两种格式：简单 CASE 函数将某个表达式与一组简单表达式进行比较以确定结果。CASE 搜索函数计算一组布尔表达式以确定结果。 两种格式都支持可选的 ELSE 参数。

简单函数

**CASE [col\_name] WHEN [value1] THEN [result1]…ELSE [default] END**

枚举这个字段所有可能的值\*

CASE input\_expression

WHEN when\_expression THEN

result\_expression [...n 个when] [

ELSE

else\_result\_expression

END

简单 CASE 函数：返回结果值介绍：

计算 input\_expression，然后按指定顺序对每个 WHEN 子句的 input\_expression = when\_expression 进行计算。

返回第一个取值为 TRUE 的 (input\_expression = when\_expression) 的 result\_expression。如果没有取值为 TRUE 的 input\_expression = when\_expression，则当指定 ELSE 子句时 SQL Server 将返回 else\_result\_expression；若没有指定 ELSE 子句，则返回 NULL 值。

SELECT

NAME '英雄',

CASE NAME

WHEN '德莱文' THEN

'斧子'

WHEN '德玛西亚-盖伦' THEN

'大宝剑'

WHEN '暗夜猎手-VN' THEN

'弩'

ELSE

'无'

END '装备' FROM

user\_info;



搜索函数

**CASE WHEN [expr] THEN [result1]…ELSE [default] END：**搜索函数可以写判断，并且搜索函数只会返回第一个符合条件的值，其他case被忽略

CASE

WHEN Boolean\_expression THEN

result\_expression [...n个when ] [...]

ELSE

else\_result\_expression

END

CASE 搜索函数：返回结果值介绍：

按指定顺序为每个 WHEN 子句的 Boolean\_expression 求值。返回第一个取值为 TRUE 的 Boolean\_expression 的 result\_expression。

如果没有取值为 TRUE 的 Boolean\_expression，则当指定 ELSE 子句时 SQL Server 将返回 else\_result\_expression；若没有指定 ELSE 子句，则返回 NULL 值。

# when 表达式中可以使用 and 连接条件SELECT

NAME '英雄',

age '年龄',

CASE

WHEN age < **18** THEN

'少年'

WHEN age < **30** THEN

'青年'

WHEN age >= **30**

AND age < **50** THEN

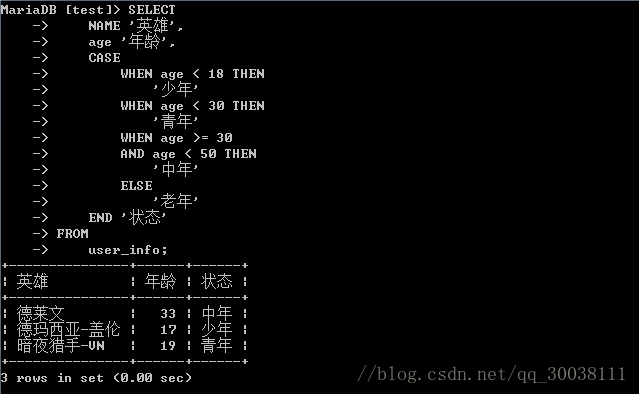
'中年'

ELSE

'老年'

END '状态' FROM

user\_info;



Mysql语句练习网址<https://blog.csdn.net/fashion2014/article/details/78826299/>

1. *查询"01"课程比"02"课程成绩高的学生的信息及课程分数*

*a,b,c可在后面定义，join是a与b表直接连接，on后面接条件，但结果取a.\*,b.score的字段，连起来后再用left join 左连接（以刚刚连接的表作为主表），而且取c表中课程为02或者空（没成绩）的字段，与b中的01课程比较大小。*

select a.\* ,b.s\_score as 01\_score,c.s\_score as 02\_score from

student a

join score b on a.s\_id=b.s\_id and b.c\_id='01'

left join score c on a.s\_id=c.s\_id and c.c\_id='02' or c.c\_id = NULL where b.s\_score>c.s\_score

--也可以这样写

select a.\*,b.s\_score as 01\_score,c.s\_score as 02\_score from student a,score b,score c

where a.s\_id=b.s\_id

and a.s\_id=c.s\_id

and b.c\_id='01'

and c.c\_id='02'

and b.s\_score>c.s\_score

*2、查询"01"课程比"02"课程成绩低的学生的信息及课程分数，跟第一个一样，只是left join 和 join对调了顺序，结果还是一样的*

select a.\* ,b.s\_score as 01\_score,c.s\_score as 02\_score from

student a left join score b on a.s\_id=b.s\_id and b.c\_id='01' or b.c\_id=NULL

join score c on a.s\_id=c.s\_id and c.c\_id='02' where b.s\_score<c.s\_score

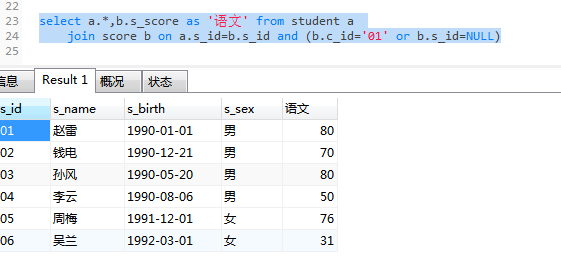
查看语数英成绩：

select a.\*,b.s\_score as '语文' from student a

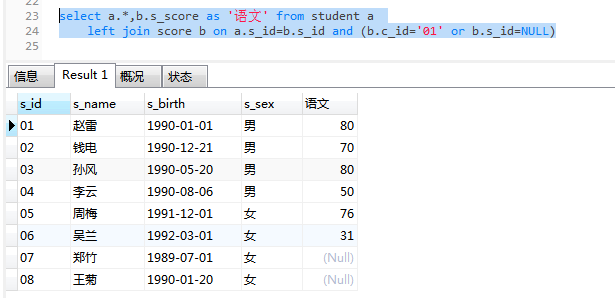
join score b on a.s\_id=b.s\_id and (b.c\_id='01' or b.c\_id=NULL) #查看语文成绩，这条命令不会显示成绩为null的同学，join显示非空的行，若需要显示null，则需要改为left join，左连接

等价于：

SELECT a.\*,b.s\_score as '语文' from student a,score b WHERE a.s\_id=b.s\_id and b.c\_id='01' or b.s\_id=NULL



修改为左连接后：



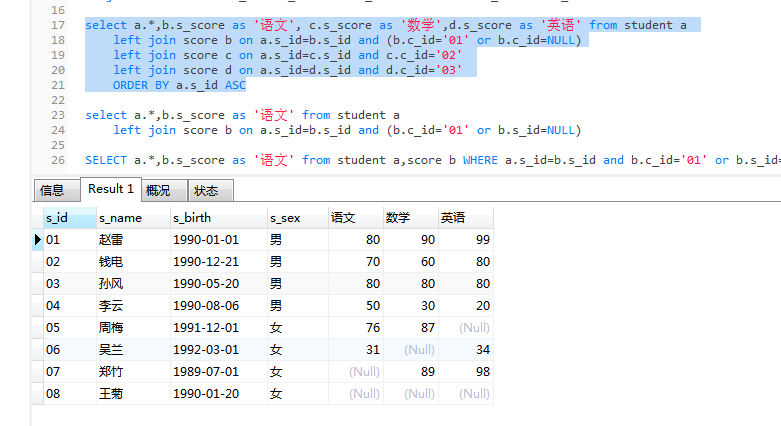
select a.\*,b.s\_score as '语文', c.s\_score as '数学',d.s\_score as '英语' from student a

left join score b on a.s\_id=b.s\_id and (b.c\_id='01' or b.c\_id=NULL)

left join score c on a.s\_id=c.s\_id and c.c\_id='02'

left join score d on a.s\_id=d.s\_id and d.c\_id='03'

ORDER BY a.s\_id ASC



*3、查询平均成绩大于等于60分的同学的学生编号和学生姓名和平均成绩*

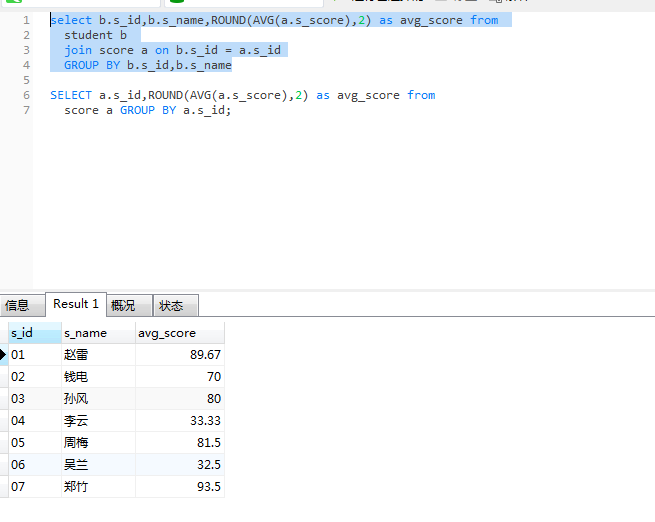
select b.s\_id,b.s\_name,ROUND(AVG(a.s\_score),2) as avg\_score from

student b

join score a on b.s\_id = a.s\_id

GROUP BY b.s\_id,b.s\_name HAVING avg\_score >=60;

From 取student表，把student和score表通过相等的s\_id（on条件）进行合并（join），再根据s\_id,s\_name（group by）算出s\_score列的成绩平均数（ avg），保留两位小数，结果起名avg\_score，取avg\_score>60（having）的s\_id,s\_name,avg\_score（select）



*4、查询平均成绩小于60分的同学的学生编号和学生姓名和平均成绩* *-- (包括有成绩的和无成绩的)*

select b.s\_id,b.s\_name,ROUND(AVG(a.s\_score),2) as avg\_score from

student b

left join score a on b.s\_id = a.s\_id

GROUP BY b.s\_id,b.s\_name HAVING avg\_score <60

union

select a.s\_id,a.s\_name,0 as avg\_score from

student a

where a.s\_id not in (

select distinct s\_id from score);

UNION用的比较多union all是直接连接，取到得是所有值，记录可能有重复 union 是取唯一值，记录没有重复   
1、UNION 的语法如下：  
 [SQL 语句 1]  
 UNION  
 [SQL 语句 2]  
  
2、UNION ALL 的语法如下：  
 [SQL 语句 1]  
 UNION ALL  
 [SQL 语句 2]  
  
效率：  
UNION和UNION ALL关键字都是将两个结果集合并为一个，但这两者从使用和效率上来说都有所不同。  
  
1、对重复结果的处理：UNION在进行表链接后会筛选掉重复的记录，Union All不会去除重复记录。  
  
2、对排序的处理：Union将会按照字段的顺序进行排序；UNION ALL只是简单的将两个结果合并后就返回。  
  
从效率上说，UNION ALL 要比UNION快很多，所以，如果可以确认合并的两个结果集中不包含重复数据且不需要排序时的话，那么就使用UNION ALL。

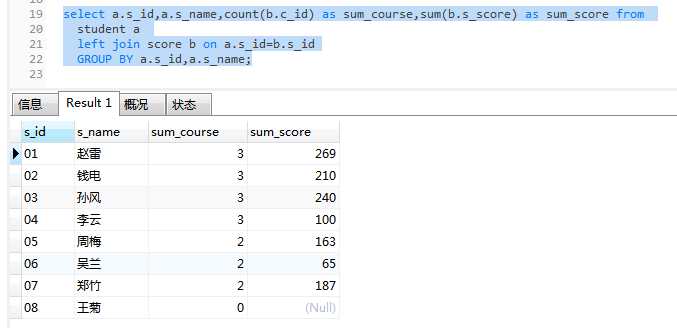
*5、查询所有同学的学生编号、学生姓名、选课总数、所有课程的总成绩*

select a.s\_id,a.s\_name,count(b.c\_id) as sum\_course,sum(b.s\_score) as sum\_score from

student a

left join score b on a.s\_id=b.s\_id

GROUP BY a.s\_id,a.s\_name;



1. *查询"李"姓老师的数量*

select count(t\_id) from teacher where t\_name like '李%';

7、查询学过"张三"老师授课的同学的信息

select a.\* from

student a

join score b on a.s\_id=b.s\_id where b.c\_id in(

select c\_id from course where t\_id =(

select t\_id from teacher where t\_name = '张三'));

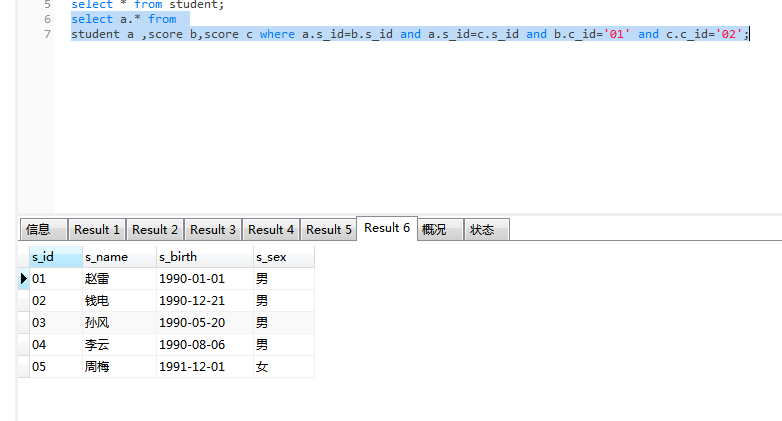
先在teacher表中找出张三老师的t\_id，再根据t\_id找出授课的c\_id，再根据c\_id找出学过的学生信息

9、查询学过编号为"01"并且也学过编号为"02"的课程的同学的信息

select a.\* from

student a,score b,score c

where a.s\_id = b.s\_id and a.s\_id = c.s\_id and b.c\_id='01' and c.c\_id='02';

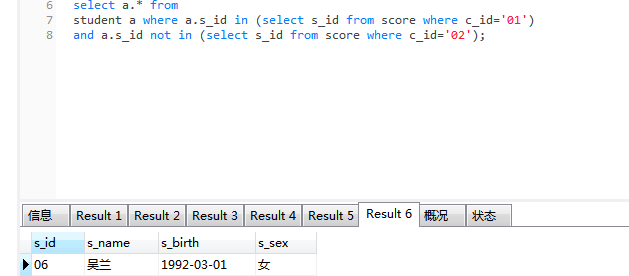


10、查询学过编号为"01"但是没有学过编号为"02"的课程的同学的信息

select a.\* from

student a

where a.s\_id in (select s\_id from score where c\_id='01' ) and a.s\_id not in(select s\_id from score where c\_id='02')



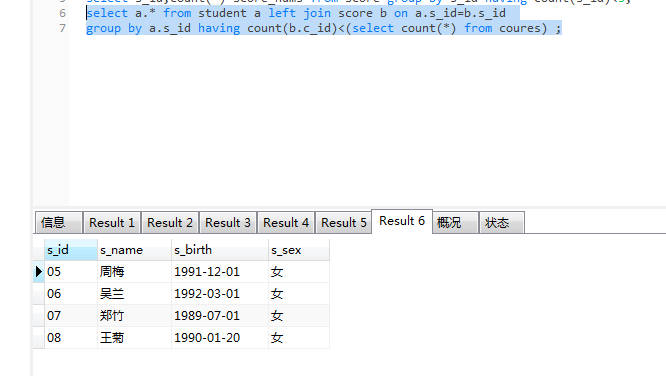
11、查询没有学全所有课程的同学的信息

--@wendiepei的写法

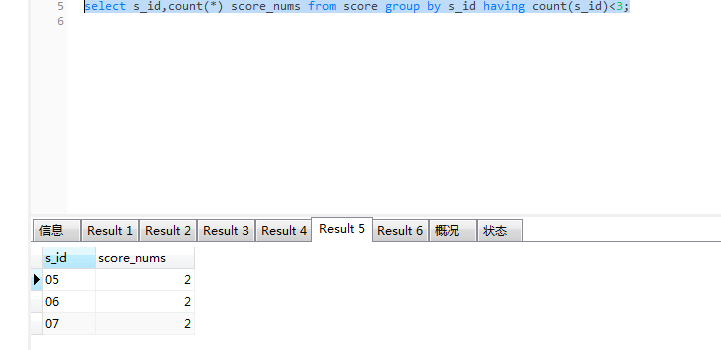
select s.\* from student s

left join Score s1 on s1.s\_id=s.s\_id

group by s.s\_id having count(s1.c\_id)<(select count(\*) from coures)



先选择student表（from），然后根据s\_id（on）左连接score表（left join），根据a.s\_id分组排序，最后having判断b.c\_id数量是否小于3（括号中的count）

这样操作是因为student有所有学生，score表中8号学生王菊一门课程的分数都没有，如果只从score表中查，会少记录

13、查询和"01"号的同学学习的课程完全相同的其他同学的信息

--@ouyang\_1993的写法

SELECT

Student.\*

FROM

Student

WHERE

s\_id IN (SELECT s\_id FROM Score GROUP BY s\_id HAVING COUNT(s\_id) = (

#下面的语句是找到'01'同学学习的课程数

SELECT COUNT(c\_id) FROM Score WHERE s\_id = '01'

)

)

AND s\_id NOT IN (

#下面的语句是找到学过‘01’同学没学过的课程，有哪些同学。并排除他们

SELECT s\_id FROM Score

WHERE c\_id IN(

#下面的语句是找到‘01’同学没学过的课程

SELECT DISTINCT c\_id FROM Score

WHERE c\_id NOT IN (

#下面的语句是找出‘01’同学学习的课程

SELECT c\_id FROM Score WHERE s\_id = '01'

)

) GROUP BY s\_id

) #下面的条件是排除01同学

AND s\_id NOT IN ('01')

14、查询没学过"张三"老师讲授的任一门课程的学生姓名

select a.s\_name from student a where a.s\_id not in (

select s\_id from score where c\_id =

(select c\_id from course where t\_id =(

select t\_id from teacher where t\_name = '张三')));

15、查询两门及其以上不及格课程的同学的学号，姓名及其平均成绩

select a.s\_id,a.s\_name,ROUND(AVG(b.s\_score)) from

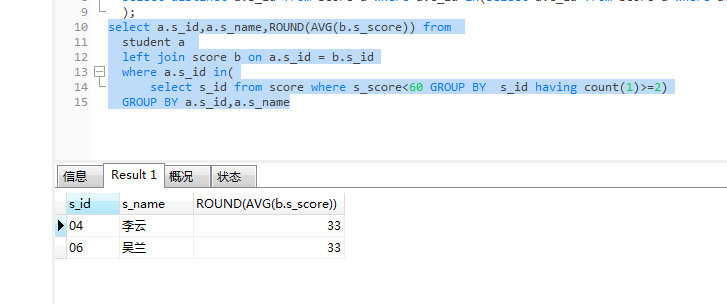
student a

left join score b on a.s\_id = b.s\_id

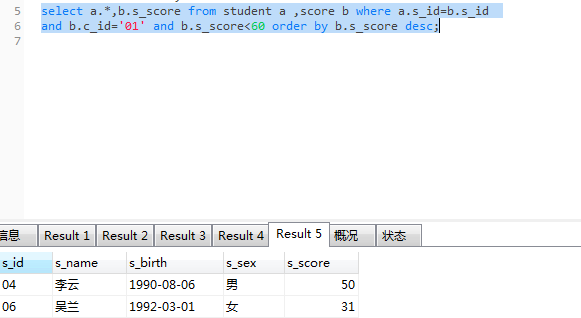
where a.s\_id in(

select s\_id from score where s\_score<60 GROUP BY s\_id having count(1)>=2)

GROUP BY a.s\_id,a.s\_name



*16、检索"01"课程分数小于60，按分数降序排列的学生信息*



17、按平均成绩从高到低显示所有学生的所有课程的成绩以及平均成绩

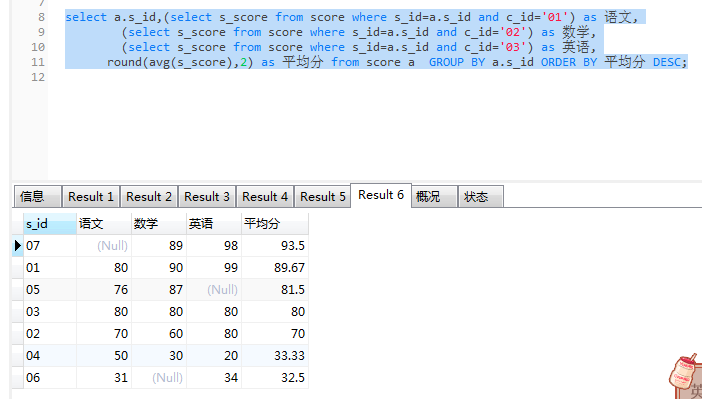
select a.s\_id,(select s\_score from score where s\_id=a.s\_id and c\_id='01') as 语文, --这里s\_id=a.s\_id是指找出前面的a.s\_id，然后找出c\_id=’01’的成绩

(select s\_score from score where s\_id=a.s\_id and c\_id='02') as 数学,

(select s\_score from score where s\_id=a.s\_id and c\_id='03') as 英语,

round(avg(s\_score),2) as 平均分 from score a GROUP BY a.s\_id ORDER BY 平均分 DESC;

先选择score表（from）作为a表，然后以a.s\_id分组（group by），之后（select）选择a.s\_id,a.s\_id的语文成绩，数学成绩，英语成绩，平均分，最后按降序排列（order by \* desc）



21、查询不同老师所教不同课程平均分从高到低显示

select a.t\_id,c.t\_name,a.c\_id,ROUND(avg(s\_score),2) as avg\_score from course a

left join score b on a.c\_id=b.c\_id

left join teacher c on a.t\_id=c.t\_id

GROUP BY a.c\_id,a.t\_id,c.t\_name ORDER BY avg\_score DESC;

