# MySQL数据库

## Linux环境下安装MySQL

Ubuntu系统中输入apt命令就可以自动化安装

sudo apt-get install mysql-server //服务端

sudo apt-get install mysql-client //客户端

sudo apt-get install libmysqlclient-dev //程序编译时链接的库

使用以下的语句判定是否安装成功

sudo netstat -tap | grep mysql

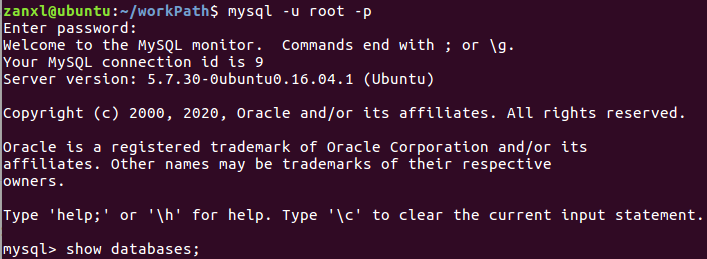


如上图所示就是安装成功的标志，listen

## 使用Ubuntu终端登录MySQL服务器

经过以上的安装步骤，MySQL服务器已经安装成功，并且也已经成功的启动，正处于listen状态，等待客户端的连接。此时可以通过以下的命令进行登录：

mysql -u root -p 然后输入在安装过程中设置的密码。这样就进入了MySQL的环境。



## windows客户端远程访问MySQL服务器

1.MySQL服务器授权远程访问命令

grant all privileges on \*.\* to 'root' @'%' identified by 'qwe123321';

2.刷新权限

flush privileges;

3.关闭防火墙

查看网络状态：service iptables status

关闭防火墙：service iptables stop

## MySQL基础概念

### 数据类型

#### 数值型

##### 严格数值数据类型

（范围可以根据该数据类型所占据的空间大小来计算所代表的数值）

（也就是使用2^bit的形式进行表述）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 大小 | 范围（有符号） | 范围（无符号） | 用途 |
| TINYINT | 1 byte | (-128，127) | (0，255) | 小整数值 |
| SMALLINT | 2 bytes | (-32768，32767) | (0，65535) | 大整数值 |
| MEDIUMINT | 3 bytes | (-2^23，2^23-1) | (0，2^24-1) | 大整数值 |
| INT/INTEGER | 4 bytes | (-2^31,2^31-1) | (0，2^32-1) | 大整数值 |
| BIGINT | 8 bytes | (-2^63,2^63-1) | (0，2^64-1) | 极大整数值 |

##### 近似数值数据类型

（范围可以根据该数据类型所占据的空间大小来计算所代表的数值）

（也就是使用2^bit的形式进行表述）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 大小 | 范围 | 用途 | 精确性 |
| FLOAT[(M,D)] | 4 bytes | M：最大位数  D：小数位数 | 单精度 浮点数值 | 低 |
| DOUBLE[(M,D)] | 8 bytes | M：最大位数  D：小数位数 | 双精度 浮点数值 | 低(比float高) |
| DECIMAL(M，D) | M+2 | M：最大位数(1-65)默认10  D：小数位数（0-30）默认0 | 小数值 | 高 |

### 字符串类型

#### char

固定长度存放，char(M),M代表宽度，即可以容纳的字符数，如果此时数据库声明的是UTF8类型，则可以存放8个中文，也就是以字符的形式进行存放的。

存放的结构，如果存放的数据小于M，则在数据后面使用空格补全，然后在读取的时候将空格删除。由于是定长的数据结构，所以在访问的时候速度较快。利用率可能达到100%。

#### varchar

变长存放，varchar(M),M代表宽度，即可以容纳的字符数。其存放的数据结构使用一块空间记录长度，设存放的数据字符数为N<=M-2,则此时占用的空间为N+2（2为字节数），所以其利用率并不能够达到100%。M范围0-65535，然后剪掉控制所使用的空间，其能够表述的最大2万+。

#### text

存放较大的文件数据，但是速度较慢。用于存储普通文本，不受行的最大长度的限制，同类还有：tinytext, mediuntext, longtext

### 日期时间类型

#### date

日期格式：0000-00-00，year-mouth-date，范围1000-01-01~9999-12-31

#### time

时间格式：00：00：00，hour : minute : second 范围-838:59:59~838:59:59

#### datetime

日期时间格式：0000-00-00 00:00:00 year-mouth-date hour :minute: second

范围：1000-01-01 00:00:00~9999-12-31 23:59:59

#### year

年的格式0000

如果使用两个数字的形式进行表示，如70-99，则表示1970-1999

如00-69，则表示2000-2069

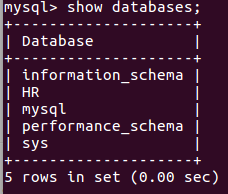
#### timestamp

时戳：就是从计算机元年1970年开始，到现在为止所经历的秒数。在编程的时候，使用int的方式存放精确到秒的时间，这样可以使用时间戳的方式方便时间的相关计算。

## 基本SQL操作

### 查看数据库信息

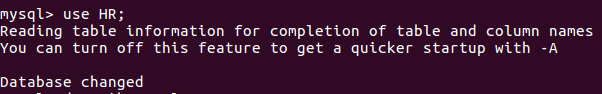
show databases;



上图所示就是当前MySQL数据库中数据库的样式，在这之后如果想要使用某一个数据库应该使用

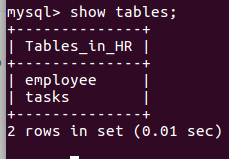
**use HR；#HR为所要使用的数据库的名称。**

这样的方式来使用当前的数据库。



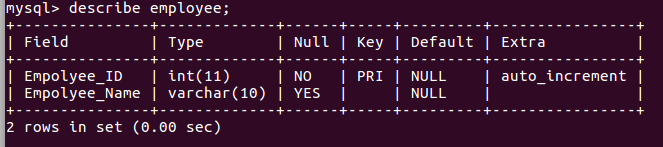
### 查看当前数据库中的表

**show tables;**



### 查看表的结构

**describe employee；**



### 创建数据库

create database name;



### 删除数据库

drop database name;



### 添加数据表

**create table employee(key,value)；**以这样的方式进行进行表的设计

示例：

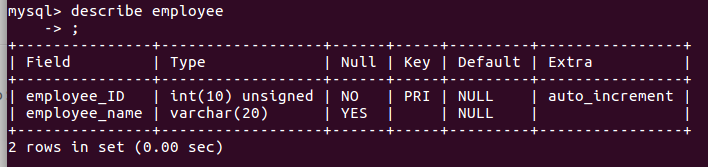
create table employee （

employee\_ID int auto\_increment,

employee\_name varchar(20) NULL,

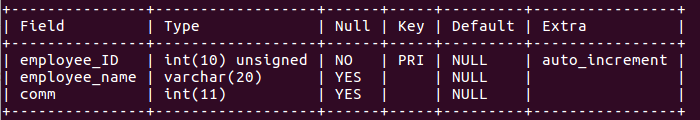
)engine = innodb,charset =utf8;（最好加上引擎以及字符集）

由上述创建的数据表的格式为：



### 在原有数据表的相关位置增加新列

**alter table employee add column comm int after employee\_name;**

****

使用after来指定新列插入的位置

### 插入数据

insert into employee values (1,"liming",5000);

推荐格式

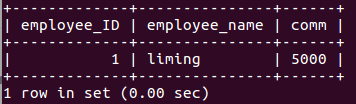
insert into tablename

(column1,column2,…)

values

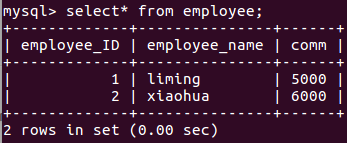
(value1,value2…);





同样也可以使用指定列的方式进行数据填充，如果某一列设定为not NULL 则必须为该列设定具体的值





### 更新某一行的数据

**update employee set comm=7000 where employee\_ID = 2;**

**推荐格式**

**update tablename**

**set**

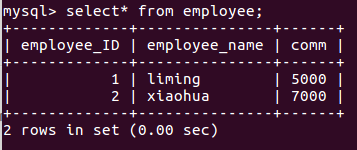
**column1 = value1**

**column2=value2**

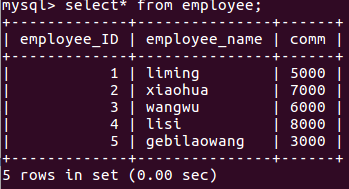
**…**

**where**

**---**



### 删除某一行元素



需要使用where语句指定某一特定需要删除的行

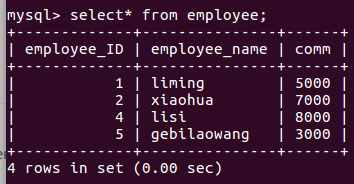
推荐格式

**delete from tablename**

**where**

**---**





## 查询

### SELECT

要将列名当作变量进行处理

select column\_name,column\_name FROM table\_name [WHERE Clause] [LIMIT N][ OFFSET M]

推荐格式：

**select column1，column2，---from tablename**

**where ---**

实现模糊查询，使用like，select\* from tablename where like “%阿萨大\_”

**%——匹配任意字符 \_ ——匹配单个字符**

查询语句中你可以使用一个或者多个表，表之间使用逗号(,)分割，并使用WHERE语句来设定查询条件。SELECT 命令可以读取一条或者多条记录。

你可以使用星号（\*）来代替其他字段，SELECT语句会返回表的所有字段数据

你可以使用 WHERE 语句来包含任何条件。

你可以使用 **LIMIT 属性来设定返回的记录数。**

你可以通过**OFFSET指定SELECT语句开始查询的数据偏移量**。默认情况下偏移量为0

使用\*号或者使用列来进行相应的查询，其效果有时候表现是一致的，但是我们建议使用列名的方式进行相应的访问，因为**\*号在查询机制的内部仍然需要进行转化**，转化为所有的列的选项之后，然后在项目的集合中进行查询，其效率相比较而言是低于直接使用列查询的。

### where子句

可以使用where子句来特定指出哪一列元素，然后将其与一个特定值作比较得到一个bool值，当bool值为true的时候选中该行，其作用类似于编程语言中的if语句。

你可以使用 AND 或者 OR 指定一个或多个条件。

WHERE 子句也可以运用于 SQL 的 DELETE 或者 UPDATE 命令。

WHERE 子句类似于程序语言中的 if 条件，根据 MySQL 表中的字段值来读取指定的数据。



### group by

group by 是以table中的某一列作为分组的标准，然后将符合这一标准的行数据化为一个分组，然后在此分组中进行相应的查询操作。

在group分组中可以使用统计函数有：

max()——求取分组中的最大值

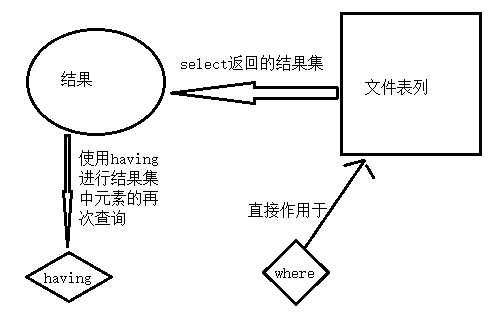
min()——求取分组中的最小值

avg()——求取分组中某一列的平均值

count()——求取分组中某一数据出现的次数

sum()——求取分组中的某一列的总和

### having

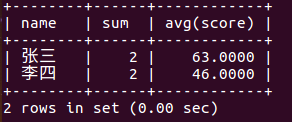




要求：只使用一次select查询

首先查出来每一个同学的平均分，然后在结果集合中约束条件

select name ,sum(score<60) sum,avg(score) from stu group by name having sum >=2;



### order by

默认情况下是升序排列的，如果想要进行降序排列，需要在order by 之后的字段名之后添加 desc进行降序排序，同样order by 支持多字段排序，也就是首先按照最前面的字段进行排序，如果前面的排序规则并不能够区分出谁在前谁在后，则按照第二个字段进行排序，一次往下，同样每一个字段默认是升序排列，如需降序排列需要直接指定desc。

### limit

用于限制输出结果的条目，limit M,N M为偏移量，也就是进行计数的起始点，然后N为限定的数目。M默认为0，必须指定N的值。

### 总结

上述五种子句按照where group having order limit这样的方式出现，可以出现其中的几项，如果同时出现了上述所有的选项，则必须按照相应的顺序进行书写。

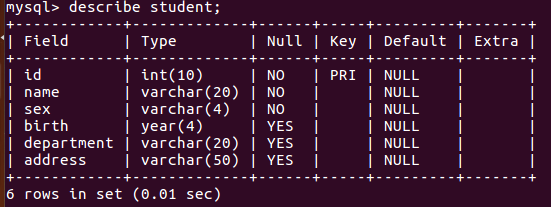
where表达式，就将该表达式放到表中，看是否为真，也就是相当于if语句，哪一行是成立的就将哪一行取出来。

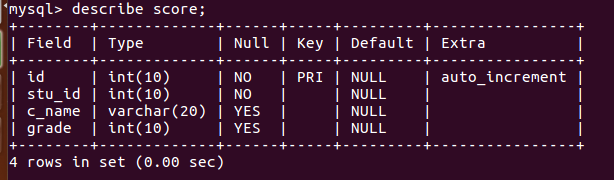
列名看作变量，变量就可以进行加减乘除等相关运算

select返回的结果集可以看作一个新的临时表进行再次查询。

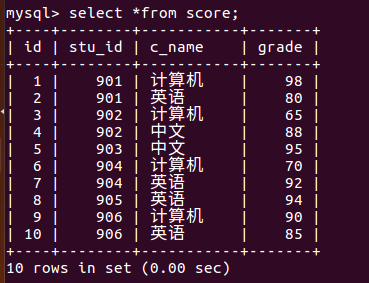
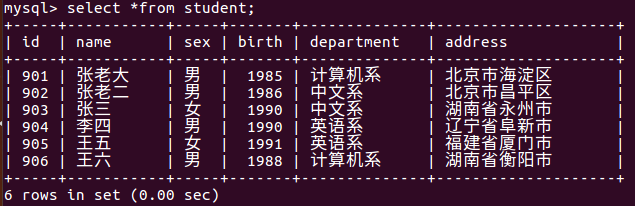
## 练习

### 表结构





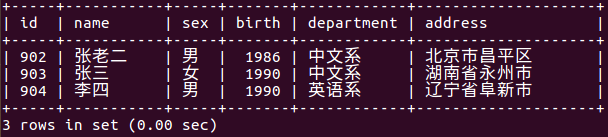
### 表数据



### 问题

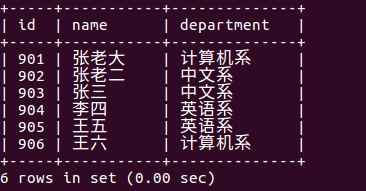
#### 问题1：查询student表的第2条到4条记录

* select \* from student s where s.id in (902,903,904);//这样是在知道表数据的情况下
* select \* from student limit 1,3;//这样的方式才是真正的获取表中的第2到第四条记录



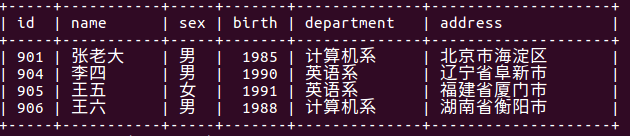
#### 问题2：.从student表查询所有学生的学号（id）、姓名（name）和院系（department）的信息

* select s.id,s.name,s.department from student s;
* select id,name,department from student;



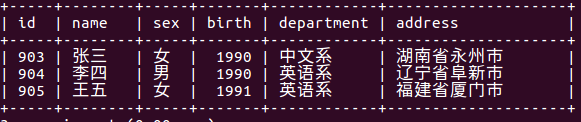
#### 问题3：从student表中查询计算机系和英语系的学生的信息

* select \*from student s where s.department = "计算机系" or s.department = "英语系";
* select \*from student where department in ("英语系","计算机系");



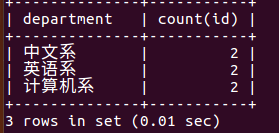
#### 问题4：从student表中查询年龄25~30岁的学生信息

* select \* from student where (2020-birth)>=25 and (2020-birth)<=30;
* select \* from student where (2020-birth) between 25 and 30;



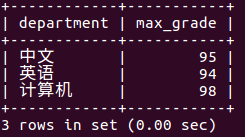
#### 问题5：从student表中查询每个院系有多少人

select department, count(id) from student group by department ;



#### 问题6：从score表中查询每个科目的最高分

select c\_name department, max(grade) max\_grade from score group by c\_name;

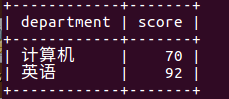


#### 问题7：查询李四的考试科目（c\_name）和考试成绩（grade）

select c\_name department,

-> grade score

-> from score where stu\_id = (select id from student where name = "李四");



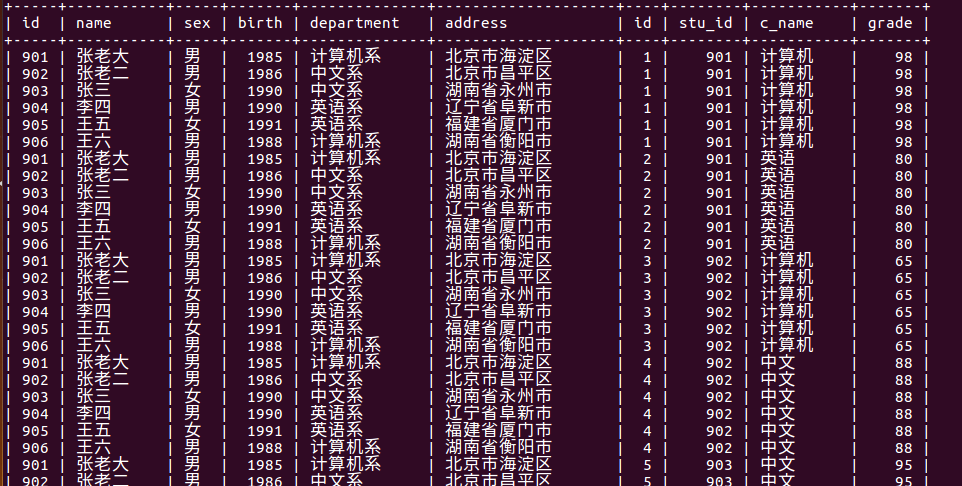
#### 问题8：用连接的方式查询所有学生的信息和考试信息

select \* from student, score;

直接使用这样的方式会获得两张数据表的笛卡尔集

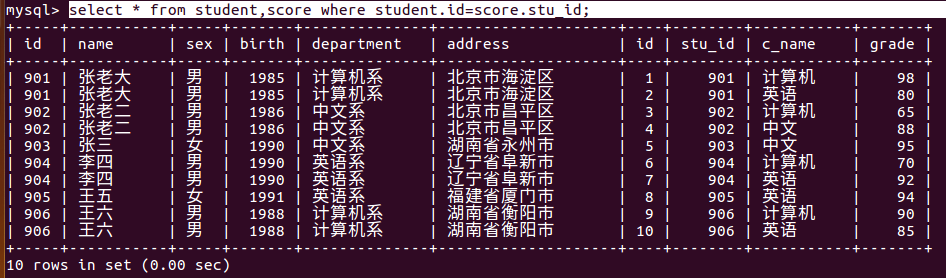
笛卡尔集的表述，其行数是两张表行数的相乘积，其列数是两张表的列数的相加和。

就是使用后面这一张表的每一行去遍历前面表的每一行，前面表的数据在前，然后后面表的数据直接附加到每一行数据之后。



但是在本题中，不能够直接得到笛卡尔集的形式，而是需要使用where语句对结果进行一定量的限定。

select \* from student,score where student.id=score.stu\_id;



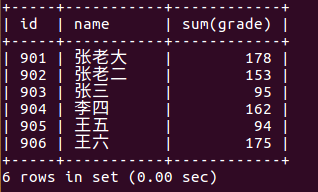
#### 问题9：计算每个学生的总成绩

select student.id, name,sum(grade)

from student , score

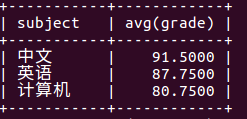
where student.id=score.stu\_id

group by id;



#### 问题10：计算每个考试科目的平均成绩

select c\_name subject,avg(grade) from score group by c\_name;



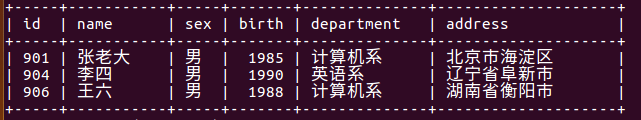
#### 问题11：查询计算机成绩低于95的学生信息

select \* from student where id in (select stu\_id from score where grade <95 and c\_name = "计算机");



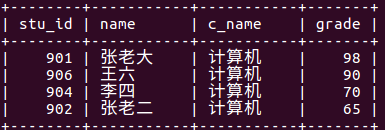
#### 问题12：查询同时参加计算机和英语考试的学生的信息

select \* from student where id in (select stu\_id from score where stu\_id in (select stu\_id from score where c\_name = "计算机") and c\_name = "英语");



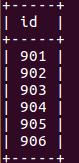
#### 问题13：将计算机考试成绩按从高到低进行排序

select stu\_id,name,c\_name, grade from score,student where student.id=score.stu\_id and c\_name = "计算机" order by grade desc;



#### 问题14：从student表和score表中查询出学生的学号，然后合并查询结果

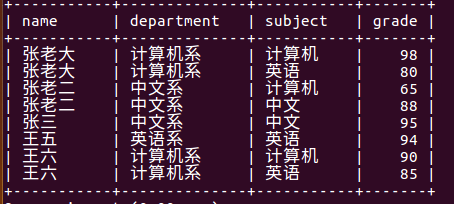
select id from student union select stu\_id from score;



#### 问题15：查询姓张或者姓王的同学的姓名、院系和考试科目及成绩

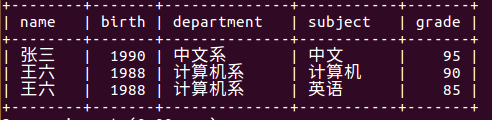
select name,department,c\_name subject,grade from student,score

-> where student.id=score.stu\_id and (name like "王%" or name like "张%");



#### 问题16：查询都是湖南的学生的姓名、年龄、院系和考试科目及成绩

select name, birth,department,c\_name subject,grade from student,score where student.id=score.stu\_id and address like "湖南%";



## 子查询

### where子查询

将内层中的子查询当作外层查询的比较条件

比如需要查询最新加入的商品（已知最新的商品其具有的ID是最大的）

则首先使用select语句查询出每一个栏目下的最大ID，然后将返回的ID集合当作判定条件，也就是外层select语句使用where　id　in　set的方式进行最终信息的输出。

一定要确保进行判断时，里面只有一列，这样可以进行in的比较，同样也可以只是一个元素，这样可以使用=进行判定。

**典型例题--查询最大商品，最贵商品**

### from子查询

也就是内层子查询返回的结果紧跟着from，也就是将上述返回的结果当作一个临时表进行再次查询，这样也就避免了进行临时便的创建。**--查询每一个栏目下的最大最贵商品**

### exist型子查询

把外层的查询拿到内层，看内层的查询是否成立。**--查询有商品的栏目**

**select \* from category where exists （select \* from goods where cat\_id = category.cat\_id）**

## 连接

### union

将两次查询所查询返回的结果**合并**起来。

两次查询的结果**必须列数一致**，并且**推荐**是每一列的数据类型也一致。

其可以来自于多张表，多次取出的列表名称可能不相同，但是显示的是首个sql语句返回的列表的名称。

union 语句使用中，如果两张表返回的行数据中具有**完全相同的两行**（也就是说这两行数据中每一列的数据都相同）这样将会执行合并的操作，也就是两行数据只会显示一行数据。也就是达到了去重的效果。

如果想要**保留两行数据的结果**，这样可以使用union all关键词。

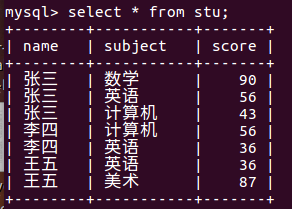
如果union连接的两个子句具有order by 或者limit 等子句需要使用小括号进行优先级选择。推荐在所有子句之后进行排序。

在子句中，order by 与limit进行配合可以生效。

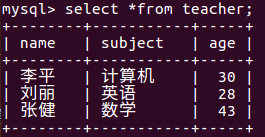
### left join

测试：

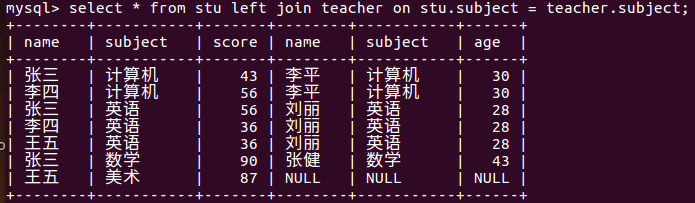
学生表：



教师表



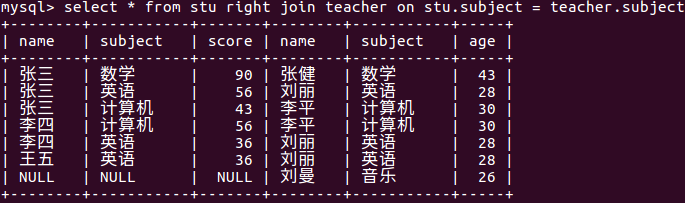
如果想要找到教授学生某一科目的老师，并将两张表连接起来，使用left join



left join

左边的表保持不变，然后使用右表中的数据进行匹配，如果匹配不到直接使用NULL进行代替。

### right join

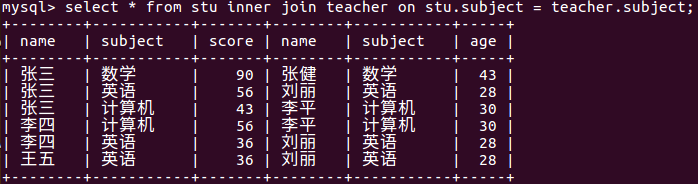


保持右表不变，然后使用坐标的数据对右表进行匹配，如果左表中的数据可以匹配，则选择出来，如果没有数据进行匹配，则使用NULL代替

**lift join 与right join 可以相互转换。**

### 内连接（inner join）

查询两表都具有的元素，也就是去除左右连接的NULL元素



### 练习题



## 视图

### 概念

视图是由查询结构形成的一种虚拟表。

为了解决一个查询频繁调用的问题，可以将上述查询所得到的结果定义为一个视图，

### 语法

创建：create view name as select ….

后面的select是select语句返回的结果集，后续对视图的操作是在此结果集上进行的。

删除：drop view name

修改：alter view name as select \*

当数据库中的数据发生变化时，其select后产生的视图会发生相应的变化。

对视图增删改时可能造成数据库的值改变，但是视图并不是总能增删改的。

能够增删改的视图时，视图能够与表的数据一一对应，并且在进行insert的时候必须要包括没有默认指的那一列。

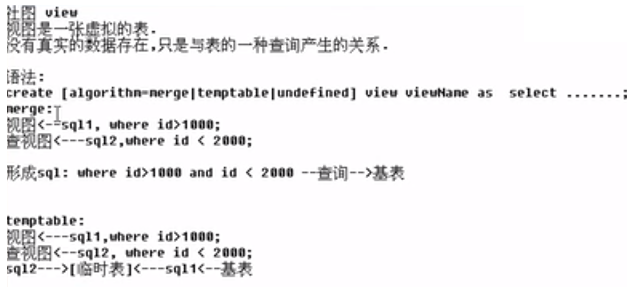
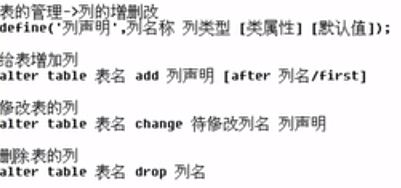
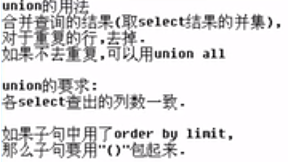
algorithm 中的三种类型：merge，在对视图进行select查询的时候，将查询中的语句与原有创建视图的select语句进行分析，然后进行合并，这样的话，select的时候所针对的表就是直接针对原有的创建视图的表。而temptable的形式就是将视图所显示的结果集存放在一个临时表中，然后后续对于视图的select的操作就会直接对于临时表进行操作，而不会将两个select语句进行分析合并，然后再去原表中进行查询。然后最后一种为undefined。这个时由系统进行选择。

### 优势

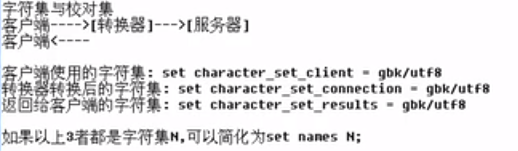
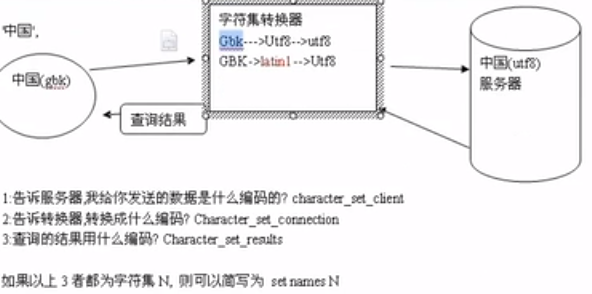
简化查询

可以更改表的权限，进行权限控制，隐藏相应的表的项目，但是给予相应视图的权限。

大数据分表时可以用到



字符集



## 触发器

触发器就是让程序监视一个事件的发生，然后当此事件发生之后，就会执行出发操作，自动的执行相关预定义的操作，每一次触发都会将相应的语句进行执行。

### 要素

监视地点，监视事件，触发时间，触发事件

### 创建触发器的语法

修改结束的标志（delimiter $）

create trigger triggerName

after/before insert/update/delete on tableName

for each row (这一句话是固定的)

begin

SQL语句集合：insert 。。。。；delete 。。。。； update 。。。。。；

end$

drop trigger triggerName;删除触发器

show triggers;查看触发器

### 创建引用列值的语法

对于insert而言，可以使用new来引用新插入的那一行，使用new.column的方式引用所在行的所在列

对于delete而言，可以使用old来引用最近删除的哪一行数据，并且使用old.column的方式引用所在行的所在列值。

对于update而言，可以使用old引用更新前的行数据，然后使用new来引用更新后的行数据。

### after与before的区别

after是指预定义的触发器触发的动作在相应监视的事件发生之后才会发生，这样不能够保证数据库中的数据是否满足监视事件中发生的事件，所以可能导致数量变为负数的问题。

before是指预定义的触发器触发的操作在监视事件发生之前发生，可以有效的判定监视的事件会不会对数据库中的数据产生不好的影响。

使用before的时候同样可以使用new，以及old的方式引用相关行的数据。

在触发器语句中，可以使用if end if 的语句块，用以判定监视的语句会不会对数据库中的数据造成不好的影响。

## 事务

### 事务的特性

原子性：同一个事务中的SQL操作要么全部执行，要么全部不执行。

隔离性：在事务提交之前，其他的事务并不能够看到此事务对表数据的修改操作。

一致性：事务在开始之前与结束之后，所有事务看到的表数据都是一致的。

持久性：事务一旦进行提交，则会永远生效，“弥补性事务”就是另一个事务了，并不是对原有事务的回退。

### 事务使用的过程

首先需要在创建数据库的过程中，声明采用的存储引擎为innodb（innodb：支持事务，更加安全，插入删除效率较低，行锁机制）

使用start transaction；开始事务

sql。。。。

sql。。。。

commit 提交

/

rollback 回滚

当一个事务使用commit或者rollback执行提交或者回滚的时候，事务将会自动的结束了。

同样需要进行注意，有些操作将会导致事务的隐式提交，比如由重新开启了一个事务等。

## 备份与恢复

使用系统自带的备份工具对数据进行备份：mysqldump

### 备份

1。导出某库下面的某表

mysqldump -u 用户名 -p 密码 库名 表1 表2 表3>地址/备份文件名称。

导出的是建表语句以及insert语句

2.导出库下面的所有表

mysqldump -u 用户名 -p 密码 库名 >地址/备份文件名称

3.以库为单位导出库

mysqldump -u 用户名 -p 密码 -B 库1 库2 库3>地址/备份文件名称

4.导出所有库

mysqldump -u 用户名 -p 密码 -A>地址/备份文件名称

### 恢复

1。登录到mysql命令行

对于库级别备份文件的恢复

source 备份文件地址

对于表级别备份文件的恢复

use 库名

source 备份文件地址

2.不登录到mysql命令行

对于库级别备份文件的恢复

mysql -u用户名 -p 密码 < 库级备份文件地址

对于表级别备份文件的恢复

mysql -u用户名 -p 密码 库名 <表级备份文件地址

## 索引

针对数据所建立的目录

可以加快查询的速度，降低数据库的IO成本，通过索引列对数据进行排序，降低数据排序的成本，降低CPU的消耗。

但是会增加增删改的响应时间，每一次对表进行更新，都需要对该数据的索引列进行相应的更新维护，索引列同样占用存储空间即增加了存储空间的使用情况

500万行数据（带索引），不要直接带着索引进行导入导出，而是首先将索引先撤销，然后将数据移动完毕之后，再整体加上索引。这样将会大大减少导入导出的时间。

### 索引的类型

普通索引（index）：加快查找数据的速度

唯一索引（unique）：唯一索引修饰的列不能够具有重复的值

主键索引（primary key）：同样也是不能够存在重复的值

全文索引（fulltext）：将文章中的每一个单词设置索引，但是会有停止词，也就是非常常见的词。

唯一索引与逐渐索引之间的区别：

虽然两者修饰的列都不能够出现重复的值，但是两者存在区别，主键索引必须是唯一索引，但是唯一索引不一定是主键索引，在一张表中，只能够有一个主键索引，但是可以由一个或者多个唯一索引。

### 命令

**查看表中的索引：**show index from tablename;

**建立索引：**

alter table tablename add index/unique/fulltext 【索引名】（列名）//如果没有指定索引名将会默认使用列名当作索引的名字。

alter table tablename add primary key （列名）//不需要加索引名，因为主键只有一个

**删除索引**

alter table tablename drop index 索引名；

删除主键的时候直接删除primary key就好。

全文索引的使用：match（全文索引名）against （“keyword”）

停止词：全文索引不针对非常频繁的词作索引，如my，the, this, you等等。

### 避免索引失效

1. 对于复合索引，执行全值匹配的原则，也就是为所有复合索引中的字段都指定select的条件。
2. 最左前缀法则，对于复合索引，查询的条件要从索引的最左前列开始，并且不跳过中间的列，只要有一列就相当于匹配，与顺序无关。
3. 范围查询右边的列不会走索引，例如> in between < !=
4. 索引列上进行计算，将会不走索引
5. 字符串不加单引号，导致索引失效（将会被MySQL优化器进行优化）
6. 尽量使用覆盖索引，减少select\* 的使用
7. 用or分割开的条件， 如果or前的条件中的列有索引，而后面的列中没有索引，那么涉及的索引都不会被用到。
8. 以%开头的立刻模糊查询，索引失效（通过覆盖索引进行解决）
9. 如果MySQL评估使用索引比全表更慢，不走索引，这是由于索引的区分度太低。
10. is null / is not null有时索引失效
11. in走索引，not in 不走索引
12. 尽量使用复合索引，而少使用单列索引。

## 存储过程以及存储函数

存储过程与函数是实现经过编译并存储在数据库的一段SQL语句的集合，调用存储过程以及函数可以简化应用开发人员的工作，减少数据在数据库与应用服务器之间的交互，对于提高数据传输效率是有好处的。

存储过程没有返回值，但是其可以通过in传入参数，通过out传出参数。

存储函数是一个由返回值的过程。

与触发器中的语句类似，因为需要多个sql语句一块书写，所以需要更改声明SQL语句的分割符，delimiter；

### 创建存储过程

delimiter $ #此时不能够加分号，如果加了分号将会是以$;结束

create procedure mypro ()

begin

select \*from city;

select \* from country;

end$

delimiter ;

**调用存储过程**： call mypro();

**删除存储过程**：drop procedure name；

### 语法

在存储过程中可以使用变量等相关编程的结构，以做到SQL编程。

**声明变量**：declare name varchar(20) #与创建数据库时的格式相同。

**赋值操作**：set name = “”；以及可以使用select 。。。into name；

**条件判断语句**：if 。。。 elseif 。。。 else 。。。 endif

**存储过程的参数**：in 传入参数 out 传出参数 inout 传入传出参数

eg.

create procedure pro\_test5(in height int , out description varchar(100))

call pro\_test5(168, @description)$

select @description$

@description : 这种变量要在变量名称前面加上“@”符号，叫做用户会话变量，代表整个会话过程他都是有作用的，这个类似于全局变量一样。

@@global.sort\_buffer\_size : 这种在变量前加上 "@@" 符号, 叫做 系统变量

**case 语句**：与when连用，使用when进行条件的符合判定

**循环语句**：while，repeat，loop

**while** search\_condition do

statement\_list

end while;

**REPEAT**

statement\_list

UNTIL search\_condition

END REPEAT;

[begin\_label:] **LOOP**

statement\_list

END LOOP [end\_label]

### 存储函数

CREATE FUNCTION function\_name([param type ... ])

RETURNS type

BEGIN

...

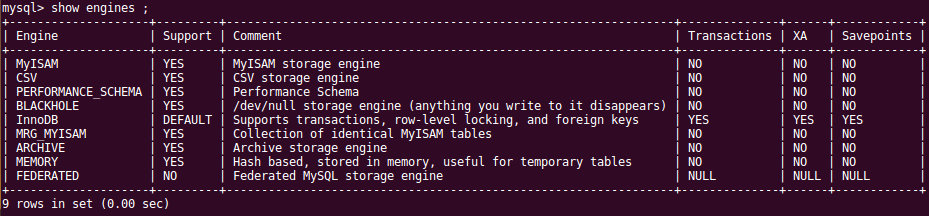
END;

## 存储引擎

存储引擎就是数据库实现存储数据，建立索引，更新查询数据等操作的方式。存储引擎时基于表结构的，而不是基于库的，所以存储引擎可以被认为是表结构的。

MySQL支持众多的存储引擎，其中默认的是innoDB存储引擎，以及MYISAM存储引擎。

查看MySQL所支持的存储引擎的语句为：show engines ;



### i**nnoDB存储引擎**

具有三大优势：支持事务，支持行级锁，支持外键。

外键：在创建外键的时候，要求父表必须有对应的索引，子表在创建外键的时候，也会自动的创建相应的索引。

CONSTRAINT `fk\_city\_country` FOREIGN KEY(country\_id) REFERENCES

country\_innodb(country\_id) ON DELETE RESTRICT ON UPDATE CASCADE

在创建索引的时候，可以指定在删除、更新父表时，对子表进行的相应操作，包括：

restrict/no action：限制在子表有关联记录的情况下，父表不能够进行更新。

cascade：表示符表在更新删除记录时，子表做出相对应的更新。

set NULL ：在父表进行更新或者删除的时候，将子表相应的记录置为NULL

### MyISAM存储引擎

**不支持事务，不支持外键，但是访问速度快**

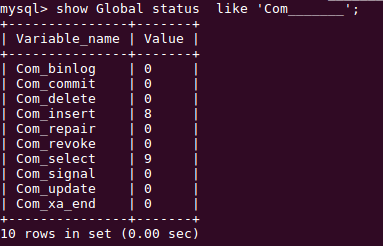
InnoDB : 是Mysql的默认存储引擎，用于事务处理应用程序，支持外键。如果应用对事务的完整性有比较高的要求，在并发条件下要求数据的一致性，数据操作除了插入和查询意外，还包含很多的更新、删除操作，那么InnoDB存储引擎是比较合适的选择。InnoDB存储引擎除了有效的降低由于删除和更新导致的锁定， 还可以确保事务的完整提交和回滚，对于类似于计费系统或者财务系统等对数据准确性要求比较高的系统，InnoDB是最合适的选择。

MyISAM ： 如果应用是以读操作和插入操作为主，只有很少的更新和删除操作，并且对事务的完整性、并发性要求不是很高，那么选择这个存储引擎是非常合适的。

## SQL语句优化

### 分析sql语句的调用频次

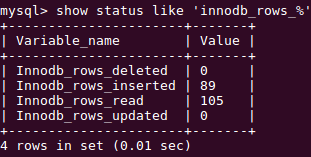
show status like ‘Com\_\_\_\_\_\_\_’;



Com\_insert:执行select的次数，每次加一

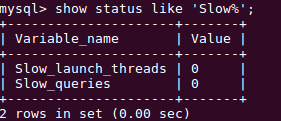
Com\_select:执行insert的次数，每次加一

Com\_。。。：执行相应操作的次数

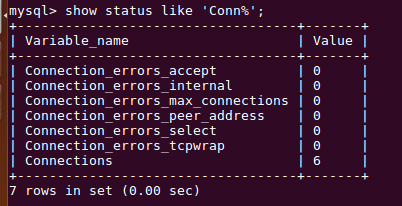


show status like ‘innodb\_rows\_%’;

执行相应的操作而从数据库返回的行数



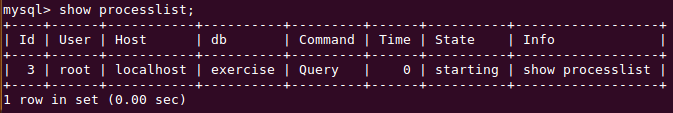
慢查询的个数：可以在配置文件中对慢查询的阈值进行自定义



尝试连接MySQL服务器的次数

### 实时查看MySQL进程列表

show processlist;



id:系统在用户连接时自动分配的ID

user:当前用户

host：IP地址+端口号

DB：当前使用的数据库名称

command：当前连接所执行的命令：休眠（sleep），查询（query），连接（connect）等

time：显示这个状态所持续的时间，单位为s

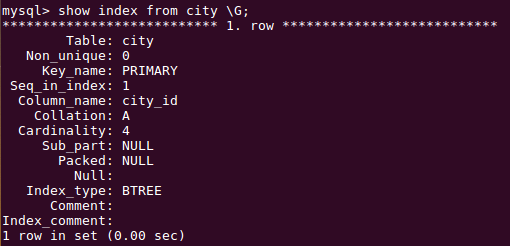
state：当前连接的sql语句的状态

info：用于判定sql语句是否存在问题

### explain分析执行计划

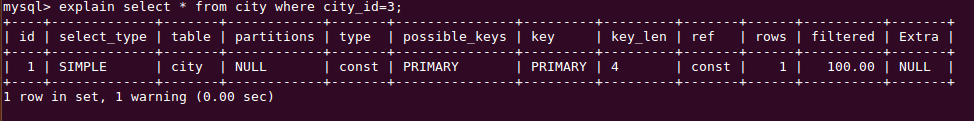
可以使用explain语句直接观察某一个sql语句是否走了索引进行相应结果的查询，同样可以根据其信息判定走的是哪一个索引，有哪一个设计的索引并没有起作用，从而根据信息的提示，完成对于sql语句的优化。

**查看某一个表的全部索引**show index from tablename

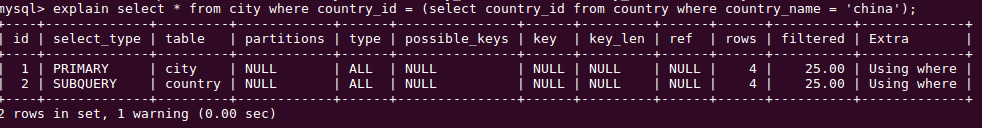


使用 explain语句来分析执行计划：（直接在sql语句的前面加上关键词explain）

explain select \* from city where city\_id = 1；



explain返回值各列详情解析：



**id字段**

select查询的序列号，是一组数字，表示的是查询中执行select子句或者是操作表的顺序。

id相同时表示加载表的顺序时从上到下的。

id不同时，值越大优先级越高，越先被执行。

id有相同有不同时，先执行值最大的，然后相同的时从上到下执行。

**select\_type字段**

表示select的类型。

simple ：简单查询，在查询中不包含子查询或者是union

primary：查询中若包含任何复杂的子查询，最外层查询标记为该标识。

subquery：在select或者where列表中包含了子查询

derived：from子查询中，被标记为衍生的sql语句执行后返回的结果将会被放置在临时表中。

union：若第二个select出现在union之后，则被标记为union，若union包含于from查询子句中，外层的select将会被标记为derived。

union result：从union表中获取结果的select。

**table字段：**

展示select这一行的数据是关于哪一个表的。

**type字段**

表示访问的类型。

NULL：MySQL不访问任何表索引，直接返回结果。

system：表中只有一行数据，这是const类型的特例。

const：表示通过索引一次就找到了

eq\_ref：类似于ref，区别在于使用的是唯一性索引，例如主键索引或者是unique索引。

ref：非唯一性索引扫描，返回匹配某个单独值得所有行。

range：之间所给定返回的行，使用一个索引来选择行，where之后出现between in <>:

all:遍历全表以找到匹配的行

index：index与all的区别为index类型只是遍历了索引树，通常比ALL快。ALL遍历的是数据文件。

NULL > system > const > eq\_ref > ref > fulltext > ref\_or\_null > index\_merge >unique\_subquery > index\_subquery > range > index > ALL

system > const > eq\_ref > ref > range > index > ALL

**possible\_keys字段**

显示可能应用在这张表上的索引，一个或者多个。

**key字段**

实际使用的索引如果为NULL，则没有使用索引。

**key\_len**

指的是索引使用的字节数，该值为索引字段的最大可能长度，并非使用长度，在不损失精确性的前提下，长度越短越好。

**rows字段**

查询所扫描的行数

**extra字段**

using filesort：说明MySQL会使用一个外部的索引排序，而不是按照表内的索引顺序进行读取，成为文件排序，效率低下。

using temporary：使用了临时表存放中间查询结果，常见于group by以及order by

using index：表示select使用了覆盖索引，避免访问文件的数据行。

### show profile分析sql

### trace分析优化器的执行过程

## MySQL的锁问题

### 锁的概念

锁是计算机协调多个进程或者线程并发访问某一资源的机制（可以避免争抢）。

以对数据操作的粒度进行划分

表级锁：操作时会锁定整张表，偏向MyISAM 存储引擎，开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。

行级锁：操作时会锁定当前操作行。偏向InnoDB 存储引擎，开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。（对于无索引的行加行锁，其将会自动变为表级锁）

以对数据操作的类型进行划分：

读锁（共享锁）：针对同一份数据，多个读操作可以同时进行而不会相互影响。

写锁（排他锁）：当前操作没有完成之前，将会阻塞其他读锁或者写锁。

读锁会阻塞写，但是不会阻塞读。而写锁，则既会阻塞读，又会阻塞写。

前面介绍了事务的特性，主要是原子性，一致性，隔离性，以及持久性。

事务：所有的操作要么全都完成，要么全部都不能完成。

### 并发事务带来的问题

**更新丢失**：当两个事务同时修改一行数据，第一个事务修改的数据可能被第二个事务的操作覆盖。例如事务A与事务B并发同时修改一行数据，事务A发现其中name字段发生了错误，然后进行修改，而此时事务B发现了年龄字段出现了错误，其对年龄字段进行修改，但是其并没有发现姓名字段的错误，还是维持原状，然后当事务A提交了之后，事务B进行提交，事务A发现自己更新的字段并没有生效，而又让事务B写了回去。

**脏读**：当一个事务正在访问数据，并对数据进行了修改，而这种数据的修改并没有提交到数据库中（可能进行了回滚），但是另一个事务也来访问数据，然后使用了未被提交的数据。

**不可重复读**：一个事务在读取数据后的某段时间，再次读取以前读过的数据，却发现和以前读过的数据不一致（由于在多次读的过程中，被另一个事务对数据进行了更新）。（解决方式：可以将正在读取的行数据加锁，也就是行级锁）

**幻读**：一个事务修改数据完成之后，再次读取相应的数据，但是此时另一个事务新插入了一行数据，此时这个事务发现自己的更改并没有生效，就像出现了幻觉一样。（解决方式：需要对当前正在处理的表进行加锁，也就是表级锁）

### 事务隔离级别

事务的隔离级别的提出是为了解决上述并发事务所带来的众多问题。在并发的事务中，更新丢失是所有的隔离级别都不允许的，也就是说两个事务同时修改一行数据，一个事务的操作被另一个事务的操作影响而覆盖或者是回滚，而导致其中一个事务的更新并没有生效，也就是出现了更新丢失，这样的方式是不被允许的，而这个问题的解决是通过加锁的方式进行。

MySQL的事务隔离的实现方式是加锁，加锁自然会带来性能的损失。

**读未提交**：读未提交模式是没有进行加锁的，也就是事务对于数据的更改会第一时间呈现在其他事务的面前，即使其事务还没有进行提交。由于读未提交模式并没有加锁，所以其速度是最快的，但是其安全性是最低的，连脏读都没有办法解决。

**读提交**：如果一个事务只能够读取到事务提交之后的数据，这样就可以解决脏读的问题，但是在事务B进行重复读的时候，事务A此时提交了自己对数据的修改，也就是现在并不能够解决重复度的问题，也就无法解决幻读的问题。

**可重复读**：可重复读意味着某一个事务在未提交之前对某一行的多次读取所得到的值是相同的，但是这样还是没有办法解决幻读的问题，也就是自己对一组数据进行了修改，但是另一个事务又重新插入了一个与之前相同的数据行，这样就像自己所作的修改没有生效的样子。

**串行化**：将事务的执行方式变成了顺序执行，相当于单线程，后一个事务的执行必须等待当前事务的结束。这样虽然能够解决脏读，不可重复读，幻读，以及更新丢失的问题，但是也大大降低了效率。

**更新丢失问题的解决**：使用悲观锁/乐观锁进行解决。

**悲观锁**：是假设更新丢失的情况一定会发生，持有的是悲观的态度。

使用数据库内部锁机制，进行数据库表的锁定。就是在A管理员修改数据时，A管理员就将数据锁定，此时B管理员无法进行修改、查询。避免两个事务同时修改，也就解决了丢失更新问题。而悲观锁使用写锁（排他锁）来进行实现。

**乐观锁**：也就是假设更新丢失的时间不会发生，不使用锁机制，而是使用一个特殊标记字段，在实际的开发中，一般使用时戳的方式进行，数据库中具有时戳这一列（将会自动记录最后一次执行更新操作的时间戳），在事务A进行数据的修改时，如果当前的时戳与数据库中时戳列相同，也就是说明在自己修改之前并没有其他的事物对数据进行修改，而如果不相同，则说明在自己修改之前，已经有了其他的事物对数据进行了一次修改，所以自己需要获取最新的数据信息，而不能够直接提交自身的修改。

### 建议

1. 尽可能让所有数据检索都能通过索引来完成，避免无索引行锁升级为表锁。
2. 合理设计索引，尽量缩小锁的范围
3. 尽可能减少索引条件，及索引范围，避免间隙锁
4. 尽量控制事务大小，减少锁定资源量和时间长度
5. 尽可使用低级别事务隔离（但是需要业务层面满足需求）

## MySQL编写顺序

SELECT DISTINCT select

<select list>

FROM from

<left\_table> <join\_type>

JOIN join

<right\_table> ON <join\_condition> on

WHERE where

<where\_condition>

GROUP BY group by

<group\_by\_list>

HAVING having

<having\_condition>

ORDER BY order by

<order\_by\_condition>

LIMIT limit

<limit\_params>

## MySQL的执行顺序

FROM <left\_table> from 先找到工作的表

ON <join\_condition> on 格式化工作的表用于后面的左右表之间的join

<join\_type> JOIN <right\_table> join 将最终的工作表进行显示

WHERE <where\_condition> where 限定表显示的条目

GROUP BY <group\_by\_list> group by将条件下的项目进行分组

HAVING <having\_condition> having 将剩余的结果进行

SELECT DISTINCT <select list>

ORDER BY <order\_by\_condition> order by 排序

LIMIT <limit\_params> limit 限定显示的条目数量