# 数据库索引

索引是对数据库表中一或多个列的值进行排序的结构，是帮助MySQL高效获取数据的数据结构。索引是加快检索表中数据的方法。数据库的索引类似于书籍的索引。在书籍中，索引允许用户不必翻阅完整个书就能迅速地找到所需要的信息。在数据库中，索引也允许数据库程序迅速地找到表中的数据，而不必扫描整个数据库。

索引就一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。索引可以是唯一的，创建索引允许指定单个列或者是多个列。

**索引原则**  
1.索引越少越好

主要在修改数据时，第个索引都要进行更新，降低写速度。  
2.最窄的字段放在键的左边  
3.避免file sort排序，临时表和表扫描.

**索引的底层实现原理和优化：**

B+树，经过优化的B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

**目的：**

快速访问数据表中的特定信息，提高检索速度

创建唯一性索引，保证数据库表中每一行数据的唯一性。

加速表和表之间的连接

使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排序的时间

**优点：**

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

**缺点：**

**为什么不对表中的每一个列创建一个索引呢？**因为，增加索引也有许多不利的方面。

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，同时也增加了数据库的尺寸大小。除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

**分类：**

MySQL数据库几个基本的索引类型：普通索引、唯一索引、主键索引、全文索引

1、按照索引列值的唯一性，索引可分为[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)和非[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)；

非[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)：

create index 索引名 on 表名（列名） tablespace [表空间](https://www.baidu.com/s?wd=%E8%A1%A8%E7%A9%BA%E9%97%B4&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)名；

唯一索引：

建立主键或者[唯一约束](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%BA%A6%E6%9D%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)时会自动在对应的列上建立唯一索引；

2、索引列的个数：**单列索引**和**复合索引**；

**两者的共同点:**

1):要想利用索引,都要符合SARG标准.

2) :都是为了提高查询速度.

3):都需要额外的系统开销,磁盘空间.

**优缺点比较:**

1):索引所占用空间:单一列索引相对要小.

2):索引创建时间:单一列索引相对短.

3):索引对insert,update,delete的影响程序:单一列索引要相对低.

4):在多条件查询时,联合索引效率要高.

索引的使用范围:**单一列索引可以出现在where 条件中的任何位置,而联合索引需要按一定的顺序来写.**

3、按照索引列的物理组织方式。

## 普通索引

这是最基本的索引，它没有任何限制。创建方式：

代码如下:

create index idx\_name on user(  
name(20)  
);  
mysql支持前缀索引，一般姓名不会超过20个字符，所以我们这里建立索引的时候限定了长度20，这样可以节省索引文件大小

## 唯一索引

它与前面的普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一。创建方式：

代码如下:

CREATE UNIQUE INDEX idx\_email ON user(  
email  
);

## 主键索引

它是一种特殊的唯一索引，不允许有空值。在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字 PRIMARY KEY 来创建。一般是在建表的时候同时创建主键索引。

代码如下:

CREATE TABLE user(  
id int unsigned not null auto\_increment,  
name varchar(50) not null,  
email varchar(40) not null,  
primary key (id)  
);

主键分为复合主键和联合主键

### 复合主键

就是指你表的主键含有一个以上的字段组成 。  
例如；  
create table test   
(   
   name varchar(19),   
   id number,   
   value varchar(10),   
   primary key (id,name)   
)   
上面的id和name字段组合起来就是你test表的复合主键 （若其一为单索引字段时，左边的id才会有索引）它的出现是因为你的name字段可能会出现重名，所以要加上ID字段这样就可以保证你记录的唯一性。一般情况下，主键的字段长度和字段数目要越少越好

### 联合主键

顾名思义就是多个主键联合形成一个主键组合，体现在联合。(主键原则上是唯一的，别被唯一值所困扰。) 

## 唯一索引和主键索引的区别：

主键一定是唯一性索引，唯一性索引并不一定就是主键。

所谓主键就是能够唯一标识表中某一行的属性或属性组，一个表只能有一个主键，但可以有多个候选索引。因为主键可以唯一标识某一行记录，所以可以确保执行数据更新、删除的时候不会出现张冠李戴的错误。主键除了上述作用外，常常与外键构成参照完整性约束，防止出现数据不一致。数据库在设计时，主键起到了很重要的作用。

1. 主键可以保证记录的唯一和主键域非空，数据库管理系统对于主键自动生成唯一索引，所以主键也是一个特殊的索引。   
2. 一个表中可以有多个唯一性索引，但只能有一个主键。   
3. 主键列不允许空值，而唯一性索引列允许空值。   
4. 索引可以提高查询的速度。

## 索引和主键的区别：

其实主键和索引都是键，不过主键是逻辑键，索引是物理键，意思就是主键不实际存在，而索引实际存在在数据库中，主键一般都要建，主要是用来避免一张表中有相同的记录，索引一般可以不建，但如果需要对该表进行查询操作，则最好建，这样可以加快检索的速度。

## 主键、外键和索引的区别

主键、外键和索引的区别

定义：

 主键--唯一标识一条记录，不能有重复的，不允许为空

 外键--表的外键是另一表的主键, 外键可以有重复的, 可以是空值

 索引--该字段没有重复值，但可以有一个空值

作用：

 主键--用来保证数据完整性

 外键--用来和其他表建立联系用的

 索引--是提高查询排序的速度

个数：

 主键--主键只能有一个

 外键--一个表可以有多个外键

 索引--一个表可以有多个唯一索引

## 全文索引

MySQL支持全文索引和搜索功能。MySQL中的全文索引类型为FULLTEXT的索引。  FULLTEXT 索引仅可用于 MyISAM表；

代码如下:

CREATE TABLE articles (  
   id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,  
   title VARCHAR(200),  
   body TEXT,  
   FULLTEXT (title,body)  
    );

## 单列索引

**单一列索引的应用结论:**

1):只要条件列中出现索引列,无论在什么位置,都能利用索引查询.

## 联合索引

两个或更多个列上的索引被称作复合索引。复合索引的创建方法与创建单一索引的方法完全一样。用户可以在多个列上建立索引，利用索引中的附加列，可以缩小搜索的范围，但使用一个具有两列的索引，不同于使用两个单独的索引。复合索引在数据库操作期间所需的开销更小，可以代替多个单一索引。创建复合索引时，应该仔细考虑列的顺序。对索引中的**所有列**执行搜索或**仅对前几列**执行搜索时，复合索引非常有用；仅对后面的任意列执行搜索时，复合索引则没有用处。当有多个字段名都经常被查询的话用复合索引。当表的行数远远大于索引键的数目时，使用这种方式可以明显加快表的查询速度。

联合索引又叫复合索引。对于复合索引:Mysql从左到右的使用索引中的字段，一个查询可以只使用索引中的一部份，但只能是最左侧部分。例如索引是key index (a,b,c). 可以支持**a**|**a,b**|**a,b,c** 3种组合进行查找，但不支持 b,c进行查找。当最左侧字段是常量引用时，索引就十分有效。

比如有一条语句是这样的：select \* from users where area=’beijing’ and age=22;

如果我们是在area和age上**分别创建**单个索引的话，由于mysql查询每次只能使用**一个索引**，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在area、age两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age,salary)的复合索引，那么其实相当于创建了(area,age,salary)、(area,age)、(area)三个索引，这被称为最佳左前缀特性。  
因此我们在创建复合索引时应该将**最常用**作限制条件的列放在**最左边**，依次递减。

**联合索引使用结论:**

1):查询条件中出现联合索引第一列,或者全部,则能利用联合索引.

2):条件列中只要条件相连在一起,以本文例子来说就是:

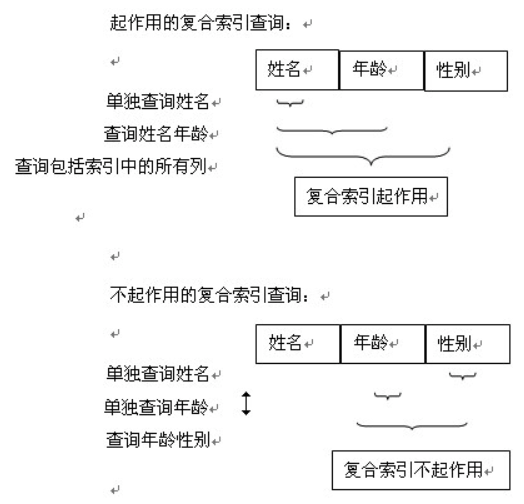
last\_name=’1′ and first\_name=’1′

与

first\_name=’1′ and last\_name=’1′

,无论前后,都会利用上联合索引.

3):查询条件中没有出现联合索引的第一列,而出现联合索引的第二列,或者第三列,都不会利用联合索引查询.



## 主键/唯一/联合索引的区别

索引是一种特殊的文件(InnoDB数据表上的索引是表空间的一个组成部分)，它们包含着对数据表里所有记录的引用指针。

**普通**索引(由关键字KEY或INDEX定义的索引)的唯一任务是加快对数据的访问速度。普通索引允许被索引的数据列包含重复的值。

如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字UNIQUE把它定义为一个**唯一**索引。也就是说，唯一索引可以保证数据记录的唯一性。

**主键**，是一种特殊的唯一索引，在一张表中只能定义一个主键索引，主键用于唯一标识一条记录，使用关键字PRIMARY KEY 来创建。

索引可以覆盖多个数据列，如像INDEX(columnA, columnB)索引，这就是**联合**索引。

索引可以极大的提高数据的查询速度，但是会降低插入、删除、更新表的速度，因为在执行这些写操作时，还要操作索引文件。

## 哪些列建索引

1、表的主键、外键必须有索引；  
2、数据量超过300的表应该有索引；  
3、经常与其他表进行连接的表，在连接字段上应该建立索引；  
4、经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；  
5、索引应该建在选择性高的字段上；  
6、索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；  
7、复合索引的建立需要进行仔细分析；尽量考虑用单字段索引代替：  
A、正确选择复合索引中的主列字段，一般是选择性较好的字段；  
B、复合索引的几个字段是否经常同时以AND方式出现在Where子句中？单字段查询是否极少甚至没有？如果是，则可以建立复合索引；否则考虑单字段索引；  
C、如果复合索引中包含的字段经常单独出现在Where子句中，则分解为多个单字段索引；  
D、如果复合索引所包含的字段超过3个，那么仔细考虑其必要性，考虑减少复合的字段；  
E、如果既有单字段索引，又有这几个字段上的复合索引，一般可以删除复合索引；  
8、频繁进行数据操作的表，不要建立太多的索引；  
9、删除无用的索引，避免对执行计划造成负面影响；

## 什么情况下应不建或少建索引

**表记录太少**

如果一个表只有5条记录，采用索引去访问记录的话，那首先需访问索引表，再通过索引表访问数据表，一般索引表与数据表不在同一个数据块，这种情况下ORACLE至少要往返读取数据块两次。而不用索引的情况下ORACLE会将所有的数据一次读出，处理速度显然会比用索引快。

如表zl\_sybm（使用部门）一般只有几条记录，除了主关键字外对任何一个字段建索引都不会产生性能优化，实际上如果对这个表进行了统计分析后ORACLE也不会用你建的索引，而是自动执行全表访问。如：

select \* from zl\_sybm where sydw\_bh=’5401’（对sydw\_bh建立索引不会产生性能优化）

**经常插入、删除、修改的表**

对一些经常处理的业务表应在查询允许的情况下尽量减少索引，如zl\_yhbm，gc\_dfss，gc\_dfys，gc\_fpdy等业务表。

**数据重复且分布平均的表字段**

假如一个表有10万行记录，有一个字段A只有T和F两种值，且每个值的分布概率大约为50%，那么对这种表A字段建索引一般不会提高数据库的查询速度。

**经常和主字段一块查询但主字段索引值比较多的表字段**

如gc\_dfss（电费实收）表经常按收费序号、户标识编号、抄表日期、电费发生年月、操作 标志来具体查询某一笔收款的情况，如果将所有的字段都建在一个索引里那将会增加数据的修改、插入、删除时间，从实际上分析一笔收款如果按收费序号索引就已 经将记录减少到只有几条，如果再按后面的几个字段索引查询将对性能不产生太大的影响。

## 索引查询不一定能提高查询的性能

通常,通过索引查询数据比全表扫描要快.但是我们也必须注意到它的代价.

索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时,索引本身也会被修改. 这意味着每条记录的INSERT,DELETE,UPDATE将为此多付出4,5 次的磁盘I/O. 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.使用索引查询不一定能提高查询性能,索引范围查询(INDEX RANGE SCAN)适用于两种情况:

基于一个范围的检索,一般查询返回结果集小于表中记录数的30%

基于非唯一性索引的检索

## 索引失效

**1、对单字段建了索引，where条件多字段。**

例：建了以下索引：

https://img-blog.csdn.net/20170110112743547?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGVoZXhpYW94aWE=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

查询语句：

[html] [view plain](https://blog.csdn.net/hehexiaoxia/article/details/54312130) [copy](https://blog.csdn.net/hehexiaoxia/article/details/54312130)

select \* from template t  where t.logicdb\_id = 4 and t.sync\_status = 1

**2、建立联合索引，where条件单字段。与上面情况正好相反。**

例：建了以下索引：

https://img-blog.csdn.net/20170110113012863?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGVoZXhpYW94aWE=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

查询语句：

[html] [view plain](https://blog.csdn.net/hehexiaoxia/article/details/54312130) [copy](https://blog.csdn.net/hehexiaoxia/article/details/54312130)

select \* from template t  where t.sync\_status = 4

**3、对索引列运算，运算包括（+、-、\*、/、！、<>、%、like'%\_'（%放在前面）、or、in、exist等），导致索引失效。**

**4、类型错误，如字段类型为varchar，where条件用number。**

例：template\_id字段是varchar类型。

错误写法：select \* from template t where t.template\_id = 1

正确写法：select \* from template t where t.template\_id = '1'  
  
**5、对索引应用内部函数，这种情况下应该建立基于函数的索引。**

例：

[html] [view plain](https://blog.csdn.net/hehexiaoxia/article/details/54312130) [copy](https://blog.csdn.net/hehexiaoxia/article/details/54312130)

select \* from template t  where ROUND(t.logicdb\_id) = 1

此时应该建ROUND(t.logicdb\_id)为索引。

**6、查询表的效率要比应用索引查询快的时候。**

**7、is null 索引失效；is not null Betree索引生效。导致的原因，个人认为应该是，mysql没有在null写进索引。还要看应用的数据库而定。**只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有**一列**含有**NULL值**，那么这一列对于此复合索引就是**无效**的。所以我们在数据库设计时不要让字段的默认值为NULL。

**8、OR语句前后没有同时使用索引**

**9、数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）**

# Oracle中的索引

在Oracle中的索引可以分为：B树索引、[位图索引](https://www.baidu.com/s?wd=%E4%BD%8D%E5%9B%BE%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)、[反向键索引](https://www.baidu.com/s?wd=%E5%8F%8D%E5%90%91%E9%94%AE%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)、基于函数的索引、簇索引、全局索引、局部索引等，下面逐一讲解：

## 一、B树索引：

最常用的索引，各[叶子节点](https://www.baidu.com/s?wd=%E5%8F%B6%E5%AD%90%E8%8A%82%E7%82%B9&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)中包括的数据有 索引列的值和 [数据表](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E8%A1%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)中对应行的ROWID，简单的说，在B树索引中，是通过在索引中保存 排过续的索引列值 与相对应记录的ROWID 来实现快速查询的目的。可以保证无论用户要搜索哪个分支的[叶子结点](https://www.baidu.com/s?wd=%E5%8F%B6%E5%AD%90%E7%BB%93%E7%82%B9&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)，都需要经过相同的索引层次，即都需要相同的I/O次数。

## 二、[位图索引](https://www.baidu.com/s?wd=%E4%BD%8D%E5%9B%BE%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)：

有些字段中使用B树索引的效率仍然不高，例如性别的字段中，只有“男、女”两个值，则即便使用了B树索引，在进行检索时也将返回接近一半的记录。

所以当字段的基数很低时，需要使用[位图索引](https://www.baidu.com/s?wd=%E4%BD%8D%E5%9B%BE%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)。(“低”的标准是取值数量 < 行数\*1%)索引中不再记录rowid和键值，而是将每个值作为一列，用0和1表示该行是否等于该键值(0表示否;1表示是)。其中位图索引的行顺序与原表的行顺序一致，可以在查询数据的过程中对应计算出行的原始物理位置。

## 三、[反向键索引](https://www.baidu.com/s?wd=%E5%8F%8D%E5%90%91%E9%94%AE%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PyNBnymvnjbkuAfYmycY0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En10vP1RznjDznWTLPWczPj6Y)

是一种特殊的B树索引，在存储构造中与B树索引完全相同，但是针对数值时，反向键索引会先反向每个键值的字节，然后对反向后的新数据进行索引。例如输入2008则转换为8002，这样当数值一次增加时，其反向键在大小中的分布仍然是比较平均的。

## 四、基于函数的索引：

## 五、全局索引和局部索引：

 这个索引貌似很复杂，其实很简单。总得来说一句话，就是无论怎么分区，都是为了方便管理。

    具体索引和表的关系有三种：

    1、局部分区索引：分区索引和分区表1对1

    2、全局分区索引：分区索引和分区表N对N

    3、全局非分区索引：非分区索引和分区表1对N

数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常使用B树及其变种B+树。在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

# 数据库引擎

MySQL数据库存储引擎的名称：MyISAM、InnoDB、BDB（BerkeleyDB）、Merge、Memory（Heap）、Example、Federated、Archive、CSV、Blackhole、MaxDB 等等十几个引擎。

Mysql中InnoDB和MyISAM的比较  
1）MyISAM：

MyIASM引擎，它是MySql的默认引擎，但不提供事务的支持，也不支持行级锁和外键。因此当执行Insert插入和Update更新语句时，即执行写操作的时候需要锁定这个表。所以会导致效率会降低。不过和Innodb不同的是，MyIASM引擎是保存了表的行数，于是当进行Select count(\*) from table语句时，可以直接的读取已经保存的值而不需要进行扫描全表。所以，如果表的**读**操作远远多于写操作时，并且不需要事务的支持的。可以将MyIASM作为数据库引擎的首先。

每个MyISAM在磁盘上存储成三个文件。第一个文件的名字以表的名字开始，扩展名指出文件类型。.frm文件存储表定义。数据文件的扩展名为.MYD (MYData)。

MyISAM表格可以被压缩，而且它们支持全文搜索。不支持事务，而且也不支持外键。如果事物回滚将造成不完全回滚，不具有原子性。在进行updata时进行表锁，并发量相对较小。如果执行大量的SELECT，MyISAM是更好的选择。

MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的，没有使用压缩从而会造成Innodb比MyISAM体积庞大不小

MyISAM缓存在内存的是索引，不是数据。而InnoDB缓存在内存的是数据，相对来说，服务器内存越大，InnoDB发挥的优势越大。

大批量的插入语句时（这里是INSERT语句）在MyIASM引擎中执行的比较的快，但是UPDATE语句在Innodb下执行的会比较的快，尤其是在并发量大的时候。

优点：查询数据相对较快，适合大量的select，可以全文索引。  
缺点：不支持事务，不支持外键，并发量较小，不适合大量update

2）InnoDB：

这种类型是事务安全的。.它与BDB类型具有相同的特性,它们还支持外键。InnoDB表格速度很快。具有比BDB还丰富的特性,因此如果需要一个事务安全的存储引擎，建议使用它。在update时表进行行锁，并发量相对较大。如果你的数据执行大量的INSERT或UPDATE，出于性能方面的考虑，应该使用InnoDB表。

Innodb引擎，Innodb引擎提供了对数据库ACID事务的支持。并且还提供了行级锁和外键的约束。它的设计的目标就是处理大数据容量的数据库系统。它本身实际上是基于Mysql后台的完整的系统。Mysql运行的时候，Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是，该引擎是不支持全文搜索的。同时，启动也比较的慢，它是不会保存表的行数的。当进行Select count(\*) from table指令的时候，需要进行扫描全表。所以当需要使用数据库的事务时，该引擎就是首选。由于锁的粒度小，写操作是不会锁定全表的。所以在并发度较高的场景下使用会提升效率的。

大容量的数据集时趋向于选择Innodb。因为它支持事务处理和故障的恢复。Innodb可以利用数据日志来进行数据的恢复。主键的查询在Innodb也是比较快的。  
优点：支持事务，支持外键，并发量较大，适合大量update  
缺点：查询数据相对较快，不适合大量的select  
对于支持事物的InnoDB类型的表，影响速度的主要原因是AUTOCOMMIT默认设置是打开的，而且程序没有显式调用BEGIN 开始事务，导致每插入一条都自动Commit，严重影响了速度。可以在执行sql前调用begin，多条sql形成一个事物（即使autocommit打开也可以），将大大提高性能。

两种引擎所使用的索引的数据结构是什么？

答案:都是B+树!

MyIASM引擎，B+树的数据结构中存储的内容实际上是实际数据的地址值。也就是说它的索引和实际数据是分开的，只不过使用索引指向了实际数据。这种索引的模式被称为非聚集索引。

Innodb引擎的索引的数据结构也是B+树，只不过数据结构中存储的都是实际的数据，这种索引有被称为聚集索引。

## MyISAM与InnoDB的区别

MyISAM类型不支持事务处理等高级处理，而InnoDB类型支持。MyISAM类型的表强调的是性能，其执行数度比InnoDB类型更快，但是不提供事务支持，而InnoDB提供事务支持已经外部键等高级数据库功能。

区别于其他数据库的最重要的特点就是其插件式的表存储引擎。切记：存储引擎是基于表的，而不是数据库。

**InnoDB与MyISAM的区别：**

InnoDB存储引擎: 主要面向OLTP(Online Transaction Processing，在线事务处理)方面的应用，是第一个完整支持ACID事务的存储引擎(BDB第一个支持事务的存储引擎，已经停止开发)。innodb是支持事务的存储引擎；合于插入和更新操作比较多的应用；设计合理的话是行锁（最大区别就在锁的级别上）；适合[大数据](http://lib.csdn.net/base/hadoop)，大并发。特点：行锁设计、支持外键,支持事务，支持并发，锁粒度是支持mvcc得行级锁；

MyISAM存储引擎: 是MySQL官方提供的存储引擎，主要面向OLAP(Online Analytical Processing,在线分析处理)方面的应用。MyISAM 是非事务的存储引擎；适合用于频繁查询的应用；表锁，不会出现死锁；适合小数据，小并发。特点：不支持事务，锁粒度是支持并发插入得表级锁，支持表所和全文索引。操作速度快，不能读写操作太频繁；

**以下是一些细节和具体实现的差别：**  
1.InnoDB不支持FULLTEXT类型的索引。  
2.InnoDB 中不保存表的具体行数，也就是说，执行select count(\*) fromtable时，InnoDB要扫描一遍整个表来计算有多少行，但是MyISAM只要简单的读出保存好的行数即可。注意的是，当count(\*)语句包含where条件时，两种表的操作是一样的。  
3.对于AUTO\_INCREMENT类型的字段，InnoDB中必须包含只有该字段的索引，但是在MyISAM表中，可以和其他字段一起建立联合索引。  
4.DELETE FROM table时，InnoDB不会重新建立表，而是一行一行的删除。  
5.LOAD TABLE FROMMASTER操作对InnoDB是不起作用的，解决方法是首先把InnoDB表改成MyISAM表，导入数据后再改成InnoDB表，但是对于使用的额外的InnoDB特性(例如外键)的表不适用。

另外，InnoDB表的行锁也不是绝对的，假如在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表，例如updatetable set num=1 where name like “a%”  
两种类型最主要的差别就是Innodb支持事务处理与外键和行级锁.而MyISAM不支持.所以MyISAM往往就容易被人认为只适合在小项目中使用。

**一般来说，MyISAM适合：**  
(1)做很多count 的计算；  
(2)插入不频繁，查询非常频繁；  
(3)没有事务。

**InnoDB适合：**  
(1)可靠性要求比较高，或者要求事务；  
(2)表更新和查询都相当的频繁，并且表锁定的机会比较大的情况指定数据引擎的创建。让所有的灵活性成为可能的开关是提供给ANSI SQL的MySQL扩展——TYPE参数。MySQL能够让你在表格这一层指定数据库引擎，所以它们有时候也指的是table formats。下面的示例代码表明了如何创建分别使用MyISAM、ISAM和HEAP引擎的表格。要注意，创建每个表格的代码是相同的，除了最后的 TYPE参数，这一参数用来指定数据引擎。

我作为使用MySQL的用户角度出发，Innodb和MyISAM都是比较喜欢的，但是从我目前运维的数据库平台要达到需求：99.9%的稳定性，方便的扩展性和高可用性来说的话，MyISAM绝对是我的首选。

**原因如下：**  
1、首先我目前平台上承载的大部分项目是读多写少的项目，而MyISAM的读性能是比Innodb强不少的。  
2、MyISAM的索引和数据是分开的，并且索引是有压缩的，内存使用率就对应提高了不少。能加载更多索引，而Innodb是索引和数据是紧密捆绑的，没有使用压缩从而会造成Innodb比MyISAM体积庞大不小。  
3、从平台角度来说，经常隔1，2个月就会发生应用开发人员不小心update一个表where写的范围不对，导致这个表没法正常用了，这个时候MyISAM的优越性就体现出来了，随便从当天拷贝的压缩包取出对应表的文件，随便放到一个数据库目录下，然后dump成sql再导回到主库，并把对应的binlog补上。如果是Innodb，恐怕不可能有这么快速度，别和我说让Innodb定期用导出xxx.sql机制备份，因为我平台上最小的一个数据库实例的数据量基本都是几十G大小。  
4、从我接触的应用逻辑来说，select count(\*) 和order by是最频繁的，大概能占了整个sql总语句的60%以上的操作，而这种操作Innodb其实也是会锁表的，很多人以为Innodb是行级锁，那个只是where对它主键是有效，非主键的都会锁全表的。  
5、还有就是经常有很多应用部门需要我给他们定期某些表的数据，MyISAM的话很方便，只要发给他们对应那表的frm.MYD,MYI的文件，让他们自己在对应版本的数据库启动就行，而Innodb就需要导出xxx.sql了，因为光给别人文件，受字典数据文件的影响，对方是无法使用的。  
6、如果和MyISAM比insert写操作的话，Innodb还达不到MyISAM的写性能，如果是针对基于索引的update操作，虽然MyISAM可能会逊色Innodb,但是那么高并发的写，从库能否追的上也是一个问题，还不如通过多实例分库分表架构来解决。  
7、如果是用MyISAM的话，merge引擎可以大大加快应用部门的开发速度，他们只要对这个merge表做一些selectcount(\*)操作，非常适合大项目总量约几亿的rows某一类型(如日志，调查统计)的业务表。  
当然Innodb也不是绝对不用，用事务的项目如模拟炒股项目，我就是用Innodb的，活跃用户20多万时候，也是很轻松应付了，因此我个人也是很喜欢Innodb的，只是如果从数据库平台应用出发，我还是会首MyISAM。  
另外，可能有人会说你MyISAM无法抗太多写操作，但是我可以通过架构来弥补，说个我现有用的数据库平台**容量：**主从数据总量在几百T以上，每天十多亿pv的动态页面，还有几个大项目是通过数据接口方式调用未算进pv总数，(其中包括一个大项目因为初期memcached没部署,导致单台数据库每天处理9千万的查询)。而我的整体数据库服务器平均负载都在0.5-1左右。

# delete/truncate/drop

**相同点：**

1.truncate和不带where子句的delete、以及drop都会删除表内全部行。

2.drop、truncate都是DDL语句(数据定义语言),执行后会自动提交。

**不同点：**

1.**删除：**truncate 和 delete 只删除数据不删除表的结构(定义)。但TRUNCATE TABLE比DELETE速度快，且使用的系统和事务日志资源少。DELETE语句每次删除一行，并在事务日志中为所删除的每行记录一项。TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。drop 语句将删除表的结构被依赖的约束(constrain)、触发器(trigger)、索引(index)；依赖于该表的存储过程/函数将保留,但是变为 invalid 状态。TRUNCATE TABLE删除表中的所有行，但表结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用DELETE。如果要删除表定义及其数据，请使用DROP TABLE语句。

2.**生效：** delete 语句是数据库操作语言(dml)，不会自动提交，这个操作会放到 rollback segement 中，事务提交之后才生效；如果有相应的 trigger，执行的时候将被触发。truncate、drop 是数据库定义语言(ddl)，执行后会被隐式提交，操作立即生效，原数据不放到 rollback segment 中，不能回滚，操作不触发 trigger。

3.**空间：**delete 语句不影响表所占用的 extent，高水线(high watermark)保持原位置不动。drop 语句将表所占用的空间全部释放。runcate 语句缺省情况下见空间释放到 minextents个 extent，除非使用reuse storage；truncate 会将高水线复位(回到最开始)。 TRUNCATE将重新设置高水平线和所有的索引。在对整个表和索引进行完全浏览时，经过TRUNCATE操作后的表比Delete操作后的表要快得多。在oracle里,使用delete删除数据以后，数据库的存储容量不会减少，而且使用delete删除某个表的数据以后，查询这张表的速度和删除之前一样，不会发生变化。

4.**速度**：一般来说: drop> truncate > delete

5.**安全性：**DELETE 可以回退 可以有条件的删，有ROLLBACK命令Delete将被撤销：DELETE FROM 表名WHERE 条件。TRUNCATE TABLE 无法回退 默认所有的表内容都删除 删除速度比delete快。TRUNCATE TABLE 表名。小心使用 drop 和 truncate，尤其没有备份的时候.否则哭都来不及。使用上,想删除部分数据行用 delete，注意带上where子句. 回滚段要足够大。想删除表,当然用 drop。想保留表而将所有数据删除，如果和事务无关，用truncate即可。如果和事务有关,或者想触发trigger,还是用delete。如果是整理表内部的碎片，可以用truncate跟上reuse stroage，再重新导入/插入数据。  
6.**适用范围：**TRUNCATE TABLE不能用于参与了索引视图的表。对于由FOREIGN KEY约束引用的表，不能使用TRUNCATE TABLE，而应使用不带WHERE子句的DELETE语句。由于TRUNCATE TABLE 不记录在日志中，所以它不能激活触发器。

**drop、delete与truncate分别在什么场景之下使用？**

不再需要一张表的时候，用drop

想删除部分数据行时候，用delete，并且带上where子句

保留表而删除所有数据的时候用truncate

# 什么是事务

事务：是一系列的数据库操作，是数据库应用的基本逻辑单位。事务就是被绑定在一起作为一个逻辑工作单元的SQL语句分组，如果任何一个语句操作失败那么整个操作就被失败，以后操作就会回滚到操作前状态，或者是上有个节点。为了确保要么执行，要么不执行，就可以使用事务。要将有组语句作为事务考虑，就需要通过ACID[测试](http://lib.csdn.net/base/softwaretest)，即原子性，一致性，隔离性和持久性。

事务（Transaction）是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

事务（transaction）是作为一个单元的一组有序的数据库操作。如果组中的所有操作都成功，则认为事务成功，即使只有一个操作失败，事务也不成功。如果所有操作完成，事务则提交，其修改将作用于所有其他数据库进程。如果一个操作失败，则事务将回滚，该事务所有操作的影响都将取消。ACID 四大特性,原子性、隔离性、一致性、持久性。

# ACID

ACID是指数据库管理系统中事务所具有的四个特性：原子性、一致性、隔离性、持续性

**原子性atomicity**

即不可分割性，事务要么全部被执行，要么就全部不被执行。

整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

**一致性consistency**

事务的执行使得数据库从一种正确状态转换成另一种正确状态。

在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。

**隔离性isolation**

在事务正确提交之前，不允许把该事务对数据的任何改变提供给任何其他事务

两个事务的执行是互不干扰的，一个事务不可能看到其他事务运行时，中间某一时刻的数据。

**持久性durability**

在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

事务正确提交后，其结果将永久保存在数据库中，即使在事务提交后有了其他故障，事务的处理结果也会得到保存。

# 事务并发会导致的问题

1） 脏读：如一个事务A正在更新数据，另一个事务B尝试读取同一个数据，B在更新期间读取了数据，但是事务A可能最后回滚了事务，导致事务B读取的数据不准确，视为脏读。  
2） 更新丢失：两个事务同时更新同一个数据，导致其中一个的更新丢失，如你修改账户，减少了100块，另一个事务也修改你的账户，减少100块，结果就是你最后发现账户少了200块。  
3） 不可重复读：如一个事务在操作期间两次读取同一批数据，结果发现数据不一样，说明另一个事务在此期间更新了数据，视为不可重复读。

4） 幻象读：如一个事务操作期间两次读取同一个表的数据，结果发现第二次读取的数据多了几条，说明在此期间有其他事务插入数据到该表，跟幻觉一样，视为幻象读。

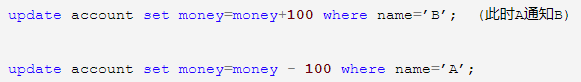
# 事务的隔离级别

当多个线程都开启事务操作数据库中的数据时，数据库系统要能进行隔离操作，以保证各个线程获取数据的准确性，在介绍数据库提供的各种隔离级别之前，我们先看看如果不考虑事务的隔离性，会发生的几种问题：

**1、脏读**

　　脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据。

　　当一个事务正在多次修改某个数据，而在这个事务中这多次的修改都还未提交，这时一个并发的事务来访问该数据，就会造成两个事务得到的数据不一致。例如：用户A向用户B转账100元，对应SQL命令如下



当只执行第一条SQL时，A通知B查看账户，B发现确实钱已到账（此时即发生了脏读），而之后无论第二条SQL是否执行，只要该事务不提交，则所有操作都将回滚，那么当B以后再次查看账户时就会发现钱其实并没有转。

**2、不可重复读**

　　不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了。

　　例如事务T1在读取某一数据，而事务T2立马修改了这个数据并且提交事务给数据库，事务T1再次读取该数据就得到了不同的结果，发送了不可重复读。

　　不可重复读和脏读的区别是，脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据。

　　在某些情况下，不可重复读并不是问题，比如我们多次查询某个数据当然以最后查询得到的结果为主。但在另一些情况下就有可能发生问题，例如对于同一个数据A和B依次查询就可能不同，A和B就可能打起来了……

**3、虚读(幻读)**

　　幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

　　幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）。

　　现在来看看MySQL数据库为我们提供的四种隔离级别：

　　① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

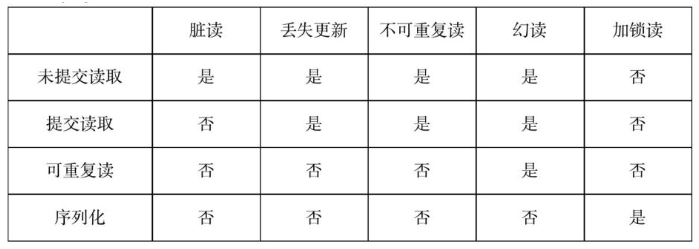
　　② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

　　③ Read committed (读已提交)：可避免脏读的发生。

　　④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

1）未提交读取：设置值为1，说明当一个事务修改数据的时候，另一个事务可以读取数据，这样会导致脏读、更新丢失、不可重复读和幻象读等问题，所以是最不安全的级别，不太常用。  
2）提交读取：设置值为2：说明当一个事务修改数据的时候，另一个事务必须等待修改完成，才能读取数据，这样可以防止脏读，但依然可能导致更新丢失、不可重复读和幻象读等问题。是比较常用的一种隔离级别。  
3）重复读：设置值为4：比提交读取安全一些，因为限制一个事务修改数据时，另一个事务不能读取该数据，同时，限制一个事务读取数据时，其他事务不能修改该数据，从而防止脏读、更新丢失和不可重复读的问题，但依然可能出现幻象读的问题。

4）序列化：设置值为8:，最安全的隔离级别，当一个事务在操作数据时，其他事务不能操作该数据，包括不能插入，从而预防各种并发问题。相当于一个事务用完该数据，另一个数据才能用。



# 如何设计一个高并发的系统

① 数据库的优化，包括合理的事务隔离级别、SQL语句优化、索引的优化

② 使用缓存，尽量减少数据库 IO

③ 分布式数据库、分布式缓存

④ 服务器的负载均衡

# ---------------------------------------

# [表，视图，存储过程，函数之间的区别](https://www.cnblogs.com/zwc1112/articles/8108758.html)

表 是真实存在的，它占内存空间  
视图 是虚拟表，不存储数据，存储的是sql，检索他的时候实际上是执行定义它的sql语句，不占任何内存  
存储过程 理解的简单一点就是“数据库中的程序”，可以在不需要外部程序（如C，java，vb等）的情况下，让数据库自己解决复杂的、用一般sql不能实现的功能，而视图则不然  
**表和视图的区别**视图不占实际空间，可以对任意的表进行叠加和剪裁，利用分区视图的功能，能加快表的I/O读取时间（需要2块以上硬盘）

**视图和存储过程的区别**  
视图只不过是存储在sqlserver上的select语句罢了，当对视图请求时，sqlserver会像执行一句普通的select语句那样的执行视图的select语句，它的性能并 不像人们想象得那么出色。   
而存储过程在编译后可以生成执行计划，这使得每次执行存储过程的时候效率将会更高，这是存储过程，另外台提交参数的时候，使用存储过程将会减少网络带宽流量，这是存储过程相对于普通的sql语句在性能上的最大的优势

**存储过程和函数的区别**  
函数：只能返回一个变量的限制。而存储过程可以返回多个。  
函数是可以嵌入在sql中使用的,可以在select中调用，而存储过程不行   
执行的本质都一样。   
函数限制比较多，比如不能用临时表，只能用表变量．还有一些函数都不可用等等．而存储过程的限制相对就比较少  
1. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。  
2. 对于存储过程来说可以返回参数，而函数只能返回值或者表对象。  
3. 存储过程一般是作为一个独立的部分来执行，而函数可以作为查询语句的一个部分来调用，由于函数可以返回一个表对象，因此它可以在查询语句 中位于FROM关键字的后面。  
4. 当存储过程和函数被执行的时候，SQL Manager会到PRocedure cache中去取相应的查询语句，如果在procedure cache里没有相应的查询语句，SQL Manager就会对存储过程和函数进行编译

# Union 和 union all区别

Union：对两个结果集进行并集操作，不包括重复行，同时进行默认规则的排序；

Union All：对两个结果集进行并集操作，包括重复行，不进行排序；

# 子查询和关联性查询的区别

连接查询是 通过主外键 让多个表数据对应 成一个表数据，而子查询是查到的数据 利用这个数据再查别的 如查张三的年级ID 再用这个年级ID去查对应的年级名

1，表关联的效率要高于子查询，因为子查询走的是笛卡尔积  
2，表关联可能有多条记录，子查询只有一条记录，如果需要唯一的列，最好走子查询。如果只需要返回 一个表的数据，建议用exists 或者in。  
3、如果要返回2个或多个表的数据，那么就用关联。如果外表数量少 内表数量很多 而且两者之间有个一一对应关系 那么用联合查询会快很多 用exists会很慢

in 是把外表和那表作hash join，而exists是对外表作loop，每次loop再对那表进行查询。  
这样的话，in适合内外表都很大的情况，exists适合外表结果集很小的情况。

一般来说 exists会比in 快，因为in要遍历所有的值 再判断是否符合 而exists是一边查一边判断，找到了就不查了 所以，一般会比in快

**子查询**

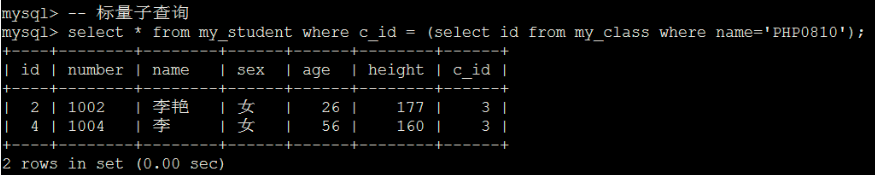
子查询：sub query,查询是在某个查询结果之上进行的(一条select 语句内部包含了另外一条select语句)

**子查询分类**  
子查询有两种分类方式：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)中出现的位置  
from 子查询：子查询跟在from之后  
where子查询：子查询出现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(理论上讲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：子查询得到的结果是多列一行(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)

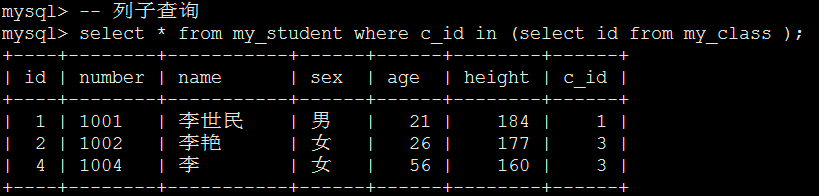
**标量子查询**  
需求：知道班级名字为PHP0710,想获取该班的所有学生

1.确定数据源：获取所有的学生  
select \* from my\_student where c\_id=?;  
2.获取班级ID:可以通过班级名字确定  
select id from my\_class where name='PHP0810'; -- id一定只有一个值(一行一列)  
标量子查询实现

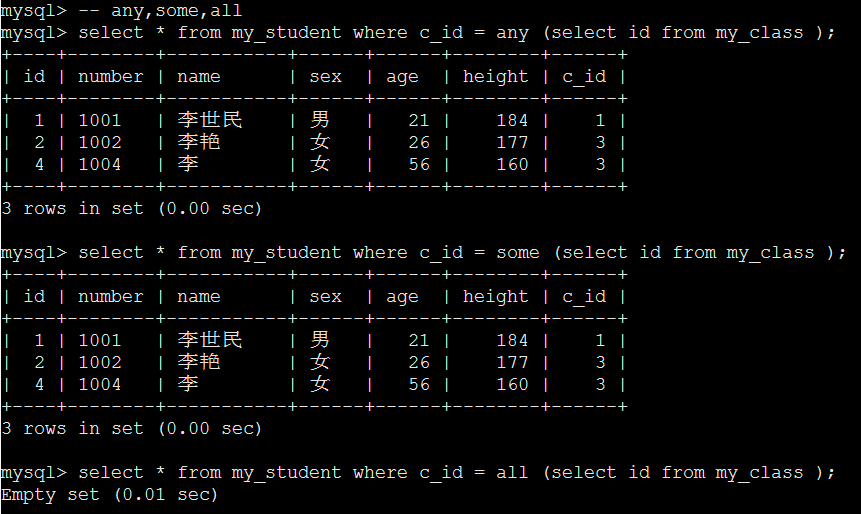


**列子查询**  
需求：查询所有在读班级的学生(班级中存在的班级)  
1. 确定数据源：学生  
select \* from my\_student where c\_id in(?);  
2.确定有效班级的 ID：所有班级id  
select id from my\_class;

列子查询实现

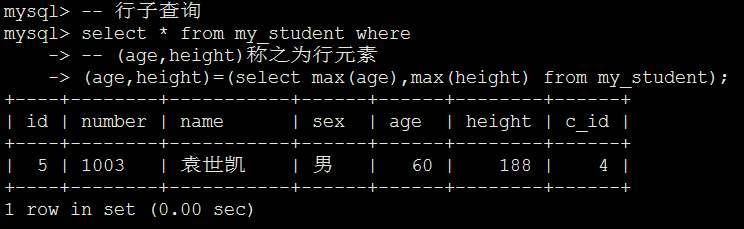


列子查询返回的结果会比较：一列多行，需要使用in 作为条件匹配：其实在mysql中还有几个类似的条件：all ,some ,any



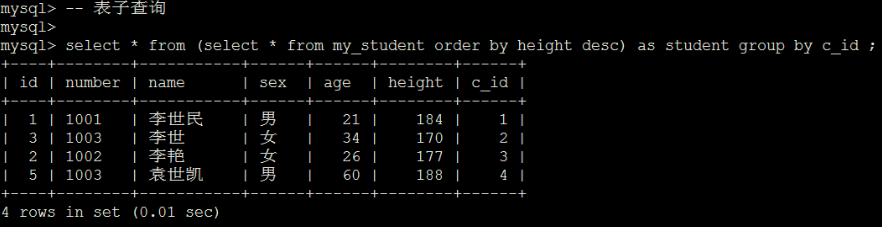
**行子查询**  
行子查询：返回的结果可以是多行多列(一行多列)  
需求：要求查询整个学生中，年龄最大且身高是最高的学生  
1.确定数据源  
select \* from my\_student where age=? and height =?;  
2.确定最大的年龄和最高的身高  
select max(age),max(height) from my\_student;

行子查询：需要构造行元素，

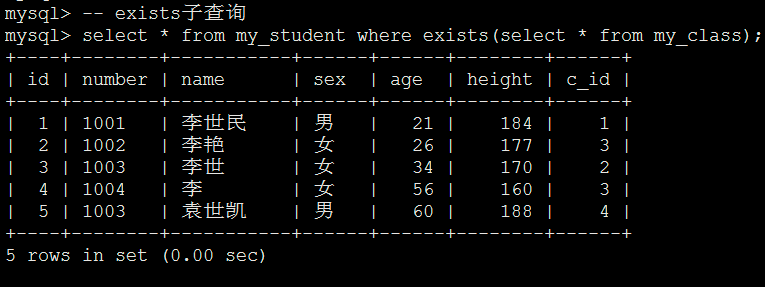


**表子查询**

表子查询：子查询返回的结果是多行多列的二维表：子查询返回的结果是当做二维表来使用  
需求：找出每个班最高的一个学生  
1.确定数据源，先将学生按照身高进行降序排序  
selelct \* from my\_student order by height desc;  
2.从每个班选出第一个学生  
selelct \* from my\_student group by c\_id; -- 每个班选出第一个学生  
表子查询：from子查询：得到的结果作为from的数据源

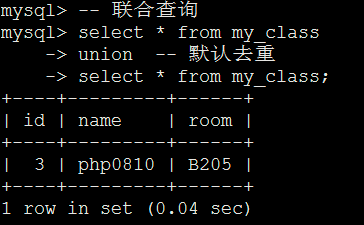


**Exists 子查询**  
Exists: 是否存在的意思，exists子查询就是用来判断某些条件是否满足(跨表),exists是接在where之后 exists返回的结果只有0&1  
需求：查询所有的学生：前提条件是班级存在  
1.确定数据源  
select \* from my\_student where ?;  
2.确定条件是否满足  
Exists(select \* from my\_class)； -- 是否成立  
exists子查询

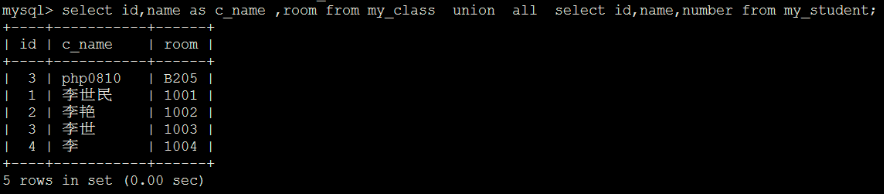


**联合查询**  
联合查询：将多次查询(多条select 语句)，在记录上进行拼接(字段不会增加)

**基本语法**  
多条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 选项一样有两个  
All:保留所有(不管重复)  
Distinct：去重(整个重复)：默认的

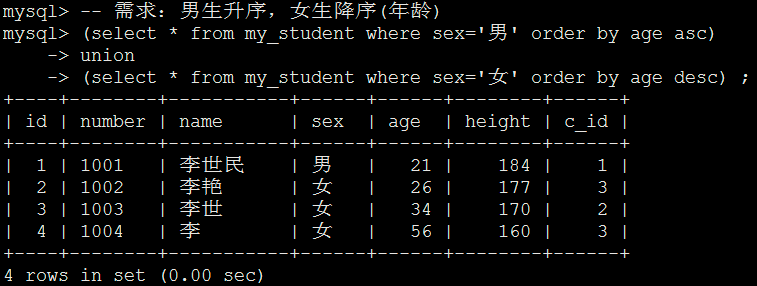


联合查询只要求字段一样，跟数据类型 无关

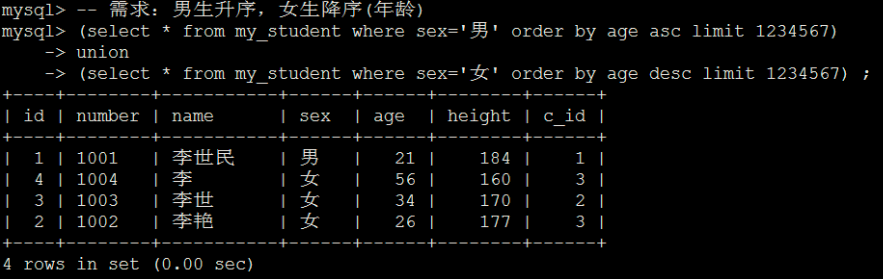


**联合查询意义**  
联合查询的意义分为两种：  
1.查询同一张表，但是需求不同：如查询学生信息，男生身高升序，女生身高降序。  
2.多表查询：多张表的结构是完全一样的，保存的数据(结构)也是一样的。

**order by 使用**  
在联合查询中：order by 不能直接使用，需要对查询语句使用括号才行



若要order by 生效：必须搭配limit：limit使用限定的最大数即可



# 数据库实现原理

* 底层和上层数据库组件概况
* 查询优化过程概况
* 事务和缓冲池管理概况

数据库是由多种互相交互的组件构成的。

核心组件：

* 进程管理器（process manager）：很多数据库具备一个需要妥善管理的进程/线程池。再者，为了实现纳秒级操作，一些现代数据库使用自己的线程而不是操作系统线程。
* 网络管理器（network manager）：网路I/O是个大问题，尤其是对于分布式数据库。所以一些数据库具备自己的网络管理器。
* 文件系统管理器（File system manager）：磁盘I/O是数据库的首要瓶颈。具备一个文件系统管理器来完美地处理OS文件系统甚至取代OS文件系统，是非常重要的。
* 内存管理器（memory manager）：为了避免磁盘I/O带来的性能损失，需要大量的内存。但是如果你要处理大容量内存你需要高效的内存管理器，尤其是你有很多查询同时使用内存的时候。
* 安全管理器（Security Manager）：用于对用户的验证和授权。
* 客户端管理器（Client manager）：用于管理客户端连接。  
  ……  
  工具：
* 备份管理器（Backup manager）：用于保存和恢复数据。
* 复原管理器（Recovery manager）：用于崩溃后重启数据库到一个一致状态。
* 监控管理器（Monitor manager）：用于记录数据库活动信息和提供监控数据库的工具。
* Administration管理器（Administration manager）：用于保存元数据（比如表的名称和结构），提供管理数据库、模式、表空间的工具。【译者注：好吧，我真的不知道Administration manager该翻译成什么，有知道的麻烦告知，不胜感激……】

……  
查询管理器：

* 查询解析器（Query parser）：用于检查查询是否合法
* 查询重写器（Query rewriter）：用于预优化查询
* 查询优化器（Query optimizer）：用于优化查询
* 查询执行器（Query executor）：用于编译和执行查询

数据管理器：

* 事务管理器（Transaction manager）：用于处理事务
* 缓存管理器（Cache manager）：数据被使用之前置于内存，或者数据写入磁盘之前置于内存
* 数据访问管理器（Data access manager）：访问磁盘中的数据

# 通配符

通配符主要以下几种：**%**、**\_**、**[]**、**[^]** 。

1. %：模糊匹配一个或多个字符。如，以ab开头

**[sql]** [view plain](https://blog.csdn.net/ludonqin/article/details/52506427) [copy](https://blog.csdn.net/ludonqin/article/details/52506427)

1. **SELECT** \* **FROM** (
2. **SELECT** 'abc\_2sd\_38u' 'a'
3. **UNION** ALL
4. **SELECT** 'bcf' 'a'
5. ) c
6. **WHERE** c.a LIKE 'ab%'

a  
-----------  
abc\_2sd\_38u

2. \_：占位符，匹配任意一个字符。如，以abc\_开头

**[sql]** [view plain](https://blog.csdn.net/ludonqin/article/details/52506427) [copy](https://blog.csdn.net/ludonqin/article/details/52506427)

1. **SELECT** \* **FROM** (
2. **SELECT** 'abc\_2sd\_38u' 'a'
3. **UNION** ALL
4. **SELECT** 'abcd\_5d\_3u' 'a'
5. **UNION** ALL
6. **SELECT** 'abc' 'a'
7. ) c
8. **WHERE** c.a LIKE 'abc[\_]%'

a  
-----------  
abc\_2sd\_38u

3. []：目标匹配字符：匹配单个字符，若写了多个，匹配任一。如，所有以1或者2开头的

**[sql]** [view plain](https://blog.csdn.net/ludonqin/article/details/52506427) [copy](https://blog.csdn.net/ludonqin/article/details/52506427)

1. **SELECT** \* **FROM** (
2. **SELECT** '1\_21' 'a'
3. **UNION** ALL
4. **SELECT** '121' 'a'
5. **UNION** ALL
6. **SELECT** '131' 'a'
7. **UNION** ALL
8. **SELECT** '142' 'a'
9. **UNION** ALL
10. **SELECT** '223' 'a'
11. **UNION** ALL
12. **SELECT** '324' 'a'
13. ) x
14. **WHERE** x.a LIKE '[12]%'

a  
-----------  
1\_21  
121  
131  
142  
223

4. [^]：目标匹配字符，[]相反。如，所有非1、2开头的

**[sql]** [view plain](https://blog.csdn.net/ludonqin/article/details/52506427) [copy](https://blog.csdn.net/ludonqin/article/details/52506427)

1. **SELECT** \* **FROM** (
2. **SELECT** '1\_21' 'a'
3. **UNION** ALL
4. **SELECT** '121' 'a'
5. **UNION** ALL
6. **SELECT** '131' 'a'
7. **UNION** ALL
8. **SELECT** '142' 'a'
9. **UNION** ALL
10. **SELECT** '223' 'a'
11. **UNION** ALL
12. **SELECT** '324' 'a'
13. ) x
14. **WHERE** x.a LIKE '[^12]%'

a  
-----------  
324

# Group by （找列）order by

当一个查询语句同时出现了where,group by,having,order by的时候，执行顺序和编写顺序   
使用count（列名）当某列出现null值的时候，count（\*）仍然会计算，但是count(列名)不会。   
二、数据分组(group by ):   
select 列a,聚合函数（聚合函数规范） from 表明 where 过滤条件 group by 列a   
group by 字句也和where条件语句结合在一起使用。当结合在一起时，where在前，group by 在后。即先对select xx from xx的记录集合用where进行筛选，然后再使用group by 对筛选后的结果进行分组。   
三、使用having字句对分组后的结果进行筛选，语法和where差不多:having 条件表达式

# Having 和where区别

1.having只能用在group by之后，对分组后的结果进行筛选(即使用having的前提条件是分组)。   
2.where肯定在group by 之前，即也在having之前。   
3.where后的条件表达式里不允许使用聚合函数，而having可以。

四、当一个查询语句同时出现了where,group by,having,order by的时候，执行顺序和编写顺序是：   
1.执行where xx对全表数据做筛选，返回第1个结果集。 2.针对第1个结果集使用group by分组，返回第2个结果集。   
3.针对第2个结果集中的每1组数据执行select xx，有几组就执行几次，返回第3个结果集。

4.针对第3个结集执行having xx进行筛选，返回第4个结果集。 5.针对第4个结果集排序。

———————————————————————————————————————————————-

Group By 和 Having, Where ,Order by这些关键字是按照如下顺序进行执行的：Where, Group By, Having, Order by。

———————————————————————————–

粗浅的理解的话，  
SELECT的语法顺序就是起执行顺序  
  
FROM   
WHERE （先过滤单表／视图／结果集，再JOIN）  
GROUP BY  
HAVING （WHERE过滤的是行，HAVING过滤的是组，所以在GROUP之后）  
ORDER BY

# 左连接右链接

左连接where只影向右表，右连接where只影响左表。  
  
Left Join  
  
select \* from tbl1 Left Join tbl2 where tbl1.ID = tbl2.ID  
  
左连接后的检索结果是显示tbl1的所有数据和tbl2中满足where 条件的数据。  
  
简言之 Left Join影响到的是右边的表  
  
Right Join  
  
select \* from tbl1 Right Join tbl2 where tbl1.ID = tbl2.ID  
  
检索结果是tbl2的所有数据和tbl1中满足where 条件的数据。  
  
简言之 Right Join影响到的是左边的表。

inner join  
  
select \* FROM tbl1 INNER JOIN tbl2 ON tbl1.ID = tbl2.ID  
  
功能和 select \* from tbl1,tbl2 where tbl1.id=tbl2.id相同。

左连接：左边有的，右边没有的为null

右连接：左边没有的，右边有的为null

内连接：显示左边右边共有的

select \* from table LIMIT 5; #返回前5行

# 什么是存储过程？有哪些优缺点？

存储过程是一些预编译的SQL语句。

更加直白的理解：存储过程可以说是一个记录集，它是由一些T-SQL语句组成的代码块，这些T-SQL语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

存储过程是一个预编译的代码块，执行效率比较高

一个存储过程替代大量T\_SQL语句 ，可以降低网络通信量，提高通信速率

可以一定程度上确保数据安全

如果你对存储过程不熟悉，建议阅读：[存储过程详解-博客园](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/knowledgesea/archive/2013/01/02/2841588.html)

# 5. 什么是锁？

  答：数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，对其加锁。加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。

基本锁类型：锁包括行级锁和表级锁

# 2. 锁的优化策略

① 读写分离

② 分段加锁

③ 减少锁持有的时间

④ 多个线程尽量以相同的顺序去获取资源

等等，这些都不是绝对原则，都要根据情况，比如不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。这部分跟面试官谈了很久

# 乐观锁和悲观锁

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。

乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

# 超键、候选键、主键、外键

超键：在关系中能唯一标识元组的属性集称为关系模式的超键。一个属性可以为作为一个超键，多个属性组合在一起也可以作为一个超键。超键包含候选键和主键。

候选键：是最小超键，即没有冗余元素的超键。

主键：数据库表中对储存数据对象予以唯一和完整标识的数据列或属性的组合。一个数据列只能有一个主键，且主键的取值不能缺失，即不能为空值（Null）。

外键：在一个表中存在的另一个表的主键称此表的外键。

# 视图的使用场景

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

只暴露部分字段给访问者，所以就建一个虚表，就是视图。

查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异

# 三个范式

第一范式（1NF）：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

第二范式（2NF）：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字。

第三范式（3NF）：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如 果存在"A → B → C"的决定关系，则C传递函数依赖于A。因此，满足第三范式的数据库表应该不存在如下依赖关系： 关键字段 → 非关键字段 x → 非关键字段y

答：第一范式：1NF是对属性的原子性约束，要求属性具有原子性，不可再分解；

第二范式：2NF是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。。

范式化设计优缺点:

优点:

可以尽量得减少数据冗余，使得更新快，体积小

缺点:对于查询需要多个表进行关联，减少写得效率增加读得效率，更难进行索引优化

反范式化:

优点:可以减少表得关联，可以更好得进行索引优化

缺点:数据冗余以及数据异常，数据得修改需要更多的成本

# 6. 实践中如何优化MySQL

我当时是按以下四条依次回答的，他们四条从效果上第一条影响最大，后面越来越小。

① SQL语句及索引的优化

② 数据库表结构的优化

③ 系统配置的优化

④ 硬件的优化

# 8. sql注入的主要特点

变种极多，攻击简单，危害极大

# 9. sql注入的主要危害

未经授权操作数据库的数据

恶意纂改网页

私自添加系统账号或者是数据库使用者账号

网页挂木马

# SQL语句的优化

order by要怎么处理

alter尽量将多次合并为一次

insert和delete也需要合并

等等

（1）Where子句中：where表之间的连接必须写在其他Where条件之前，那些可以过滤掉最大数量记录的条件必须写在Where子句的末尾.HAVING最后。

（2）用EXISTS替代IN、用NOT EXISTS替代NOT IN。

（3）避免在索引列上使用计算

（4）避免在索引列上使用IS NULL和IS NOT NULL

（5）对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

（6）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

（7）应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描

# 优化数据库的方法

[MySQL数据库优化的八大方式（经典必看）点击获取](http://www.jianshu.com/p/dac715a88b44)

· 选取最适用的字段属性，尽可能减少定义字段宽度，尽量把字段设置NOTNULL，例如'省份'、'性别'最好适用ENUM

· 使用连接(JOIN)来代替子查询

· 适用联合(UNION)来代替手动创建的临时表

· 事务处理

· 锁定表、优化事务处理

· 适用外键，优化锁定表

· 建立索引

· 优化查询语句

# 如何进行SQL优化

答：

（1）选择正确的存储引擎

以 MySQL为例，包括有两个存储引擎 **MyISAM** 和 **InnoDB**，每个引擎都有利有弊。

MyISAM 适合于一些需要大量查询的应用，但其对于有大量写操作并不是很好。甚至你只是需要update一个字段，整个表都会被锁起来，而别的进程，就算是读进程都无法操作直到读操作完成。另外，MyISAM 对于 SELECT COUNT(\*) 这类的计算是超快无比的。

InnoDB 的趋势会是一个非常复杂的存储引擎，对于一些小的应用，它会比 MyISAM 还慢。但是它支持“行锁” ，于是在写操作比较多的时候，会更优秀。并且，他还支持更多的高级应用，比如：事务。

（2）优化字段的数据类型

记住一个原则，越小的列会越快。如果一个表只会有几列罢了（比如说字典表，配置表），那么，我们就没有理由使用 INT 来做主键，使用 MEDIUMINT, SMALLINT 或是更小的 TINYINT 会更经济一些。如果你不需要记录时间，使用 DATE 要比 DATETIME 好得多。当然，你也需要留够足够的扩展空间。

（3）为搜索字段添加索引

索引并不一定就是给主键或是唯一的字段。如果在你的表中，有某个字段你总要会经常用来做搜索，那么最好是为其建立索引，除非你要搜索的字段是大的文本字段，那应该建立全文索引。

(4)避免使用Select \*从数据库里读出越多的数据，那么查询就会变得越慢。并且，如果你的数据库服务器和WEB服务器是两台独立的服务器的话，这还会增加网络传输的负载。即使你要查询数据表的所有字段，也尽量不要用\*通配符，善用内置提供的字段排除定义也许能给带来更多的便利。

(5)使用 ENUM 而不是 VARCHAR

ENUM 类型是非常快和紧凑的。在实际上，其保存的是 TINYINT，但其外表上显示为字符串。这样一来，用这个字段来做一些选项列表变得相当的完美。例如，性别、民族、部门和状态之类的这些字段的取值是有限而且固定的，那么，你应该使用 ENUM 而不是 VARCHAR。

(6)尽可能的使用 NOT NULL

除非你有一个很特别的原因去使用 NULL 值，你应该总是让你的字段保持 NOT NULL。 NULL其实需要额外的空间，并且，在你进行比较的时候，你的程序会更复杂。 当然，这里并不是说你就不能使用NULL了，现实情况是很复杂的，依然会有些情况下，你需要使用NULL值。

(7)固定长度的表会更快

如果表中的所有字段都是“固定长度”的，整个表会被认为是 “static” 或 “fixed-length”。 例如，表中没有如下类型的字段： VARCHAR，TEXT，BLOB。只要你包括了其中一个这些字段，那么这个表就不是“固定长度静态表”了，这样，MySQL 引擎会用另一种方法来处理。

固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

并且，固定长度的表也更容易被缓存和重建。不过，唯一的副作用是，固定长度的字段会浪费一些空间，因为定长的字段无论你用不用，他都是要分配那么多的空间。

# 21.了解XSS攻击吗？如何防止？

XSS是跨站脚本攻击，首先是利用跨站脚本漏洞以一个特权模式去执行攻击者构造的脚本，然后利用不安全的Activex控件执行恶意的行为。  
使用htmlspecialchars()函数对提交的内容进行过滤，使字符串里面的特殊符号实体化。

# 22.SQL注入漏洞产生的原因？如何防止？

SQL注入产生的原因：程序开发过程中不注意规范书写sql语句和对特殊字符进行过滤，导致客户端可以通过全局变量POST和GET提交一些sql语句正常执行。

防止SQL注入的方式：

开启配置文件中的magic\_quotes\_gpc 和 magic\_quotes\_runtime设置

执行sql语句时使用addslashes进行sql语句转换

Sql语句书写尽量不要省略双引号和单引号。

过滤掉sql语句中的一些关键词：update、insert、delete、select、 \* 。

提高数据库表和字段的命名技巧，对一些重要的字段根据程序的特点命名，取不易被猜到的。

Php配置文件中设置register\_globals为off,关闭全局变量注册

控制错误信息，不要在浏览器上输出错误信息，将错误信息写到日志文件中。

**负面影响：**  
创建索引和维护索引需要耗费时间，这个时间随着数据量的增加而增加；索引需要占用物理空间，不光是表需要占用数据空间，每个索引也需要占用物理空间；当对表进行增、删、改、的时候索引也要动态维护，这样就降低了数据的维护速度。

c)、为数据表建立索引的原则有哪些？

在最频繁使用的、用以缩小查询范围的字段上建立索引。

在频繁使用的、需要排序的字段上建立索引

d)、 什么情况下不宜建立索引？

对于查询中很少涉及的列或者重复值比较多的列，不宜建立索引。

对于一些特殊的数据类型，不宜建立索引，比如文本字段（text）等

# MySQL外连接、内连接与自连接的区别

**先说什么是交叉连接:** 交叉连接又叫笛卡尔积，它是指不使用任何条件，直接将一个表的所有记录和另一个表中的所有记录一一匹配。

**内连接** 则是只有条件的交叉连接，根据某个条件筛选出符合条件的记录，不符合条件的记录不会出现在结果集中，即内连接只连接匹配的行。  
**外连接** 其结果集中不仅包含符合连接条件的行，而且还会包括左表、右表或两个表中  
的所有数据行，这三种情况依次称之为左外连接，右外连接，和全外连接。

左外连接，也称左连接，左表为主表，左表中的所有记录都会出现在结果集中，对于那些在右表中并没有匹配的记录，仍然要显示，右边对应的那些字段值以NULL来填充。右外连接，也称右连接，右表为主表，右表中的所有记录都会出现在结果集中。左连接和右连接可以互换，MySQL目前还不支持全外连接。

# 33、Myql中的事务回滚机制概述

事务是用户定义的一个数据库操作序列，这些操作要么全做要么全不做，是一个不可分割的工作单位，事务回滚是指将该事务已经完成的对数据库的更新操作撤销。

要同时修改数据库中两个不同表时，如果它们不是一个事务的话，当第一个表修改完，可能第二个表修改过程中出现了异常而没能修改，此时就只有第二个表依旧是未修改之前的状态，而第一个表已经被修改完毕。而当你把它们设定为一个事务的时候，当第一个表修改完，第二表修改出现异常而没能修改，第一个表和第二个表都要回到未修改的状态，这就是所谓的事务回滚

# 2. SQL语言包括哪几部分？每部分都有哪些操作关键字？

答：SQL语言包括数据定义(DDL)、数据操纵(DML),数据控制(DCL)和数据查询（DQL）四个部分。

数据定义：Create Table,Alter Table,Drop Table, Craete/Drop Index等

数据操纵：Select ,insert,update,delete,

数据控制：grant,revoke

数据查询：select

# 3. 完整性约束包括哪些？

答：数据完整性(Data Integrity)是指数据的精确(Accuracy)和可靠性(Reliability)。

分为以下四类：

1) 实体完整性：规定表的每一行在表中是惟一的实体。

2) 域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。

3) 参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。

4) 用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

与表有关的约束：包括列约束(NOT NULL（非空约束）)和表约束(PRIMARY KEY、foreign key、check、UNIQUE) 。

# 6. 什么叫视图？游标是什么？

答：视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

  游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

# 7. 什么是存储过程？用什么来调用？

答：存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。可以用一个命令对象来调用存储过程。

# 10. 什么是基本表？什么是视图？

答：基本表是本身独立存在的表，在 SQL 中一个关系就对应一个表。  视图是从一个或几个基本表导出的表。视图本身不独立存储在数据库中，是一个虚表

# 11. 试述视图的优点？

答：(1) 视图能够简化用户的操作  (2) 视图使用户能以多种角度看待同一数据； (3) 视图为数据库提供了一定程度的逻辑独立性； (4) 视图能够对机密数据提供安全保护。

# 12. NULL是什么意思

答：NULL这个值表示UNKNOWN(未知):它不表示“”(空字符串)。对NULL这个值的任何比较都会生产一个NULL值。您不能把任何值与一个 NULL值进行比较，并在逻辑上希望获得一个答案。

使用IS  NULL来进行NULL判断

# 如何确保表格里的字段只接受特定范围里的值?

答：Check限制，它在数据库表格里被定义，用来限制输入该列的值。

触发器也可以被用来限制数据库表格里的字段能够接受的值，但是这种办法要求触发器在表格里被定义，这可能会在某些情况下影响到性能。

# 16. SQL语句中‘相关子查询’与‘非相关子查询’有什么区别？

答：子查询：嵌套在其他查询中的查询称之。

子查询又称内部，而包含子查询的语句称之外部查询（又称主查询）。

所有的子查询可以分为两类，即相关子查询和非相关子查询

（1）非相关子查询是独立于外部查询的子查询，子查询总共执行一次，执行完毕后将值传递给外部查询。

（2）相关子查询的执行依赖于外部查询的数据，外部查询执行一行，子查询就执行一次。

故非相关子查询比相关子查询效率高

# 17. char和varchar的区别？

答：是一种固定长度的类型，varchar则是一种可变长度的类型，它们的区别是：

char(M)类型的数据列里，每个值都占用M个字节，如果某个长度小于M，MySQL就会在它的右边用空格字符补足．（在检索操作中那些填补出来的空格字符将被去掉）在varchar(M)类型的数据列里，每个值只占用刚好够用的字节再加上一个用来记录其长度的字节（即总长度为L+1字节）．

varchar得适用场景:

字符串列得最大长度比平均长度大很多 2.字符串很少被更新，容易产生存储碎片 3.使用多字节字符集存储字符串

Char得场景:

    存储具有近似得长度（md5值,身份证，手机号）,长度比较短小得字符串（因为varchar需要额外空间记录字符串长度），更适合经常更新得字符串，更新时不会出现页分裂得情况，避免出现存储碎片，获得更好的io性能

# 19. 数据表类型有哪些

       答：MyISAM、InnoDB、HEAP、BOB,ARCHIVE,CSV等。

       MyISAM：成熟、稳定、易于管理，快速读取。一些功能不支持（事务等），表级锁。

       InnoDB：支持事务、外键等特性、数据行锁定。空间占用大，不支持全文索引等。

# 20. MySQL数据库作发布系统的存储，一天五万条以上的增量，预计运维三年,怎么优化？

a. 设计良好的数据库结构，允许部分数据冗余，尽量避免join查询，提高效率。

b. 选择合适的表字段数据类型和存储引擎，适当的添加索引。

c. mysql库主从读写分离。

d. 找规律分表，减少单表中的数据量提高查询速度。

e。添加缓存机制，比如memcached，apc等。

f. 不经常改动的页面，生成静态页面。

g. 书写高效率的SQL。比如 SELECT \* FROM TABEL 改为 SELECT field\_1, field\_2, field\_3 FROM TABLE.

# 21. 对于大流量的网站,您采用什么样的方法来解决各页面访问量统计问题？

答：a. 确认服务器是否能支撑当前访问量。

b. 优化数据库访问。

c. 禁止外部访问链接（盗链）, 比如图片盗链。

d. 控制文件下载。

e. 使用不同主机分流。

f. 使用浏览统计软件，了解访问量，有针对性的进行优化。

# 为表中得字段选择合适得数据类型（物理设计）

 字段类型优先级: 整形>date,time>enum,char>varchar>blob,text

 优先考虑数字类型，其次是日期或者二进制类型，最后是字符串类型，同级别得数据类型，应该优先选择占用空间小的数据类型

# 23:存储时期

Datatime:以 YYYY-MM-DD HH:MM:SS 格式存储时期时间，精确到秒，占用8个字节得存储空间，datatime类型与时区无关

Timestamp:以时间戳格式存储，占用4个字节，范围小1970-1-1到2038-1-19，显示依赖于所指定得时区，默认在第一个列行的数据修改时可以自动得修改timestamp列得值

Date:（生日）占用得字节数比使用字符串.datatime.int储存要少，使用date只需要3个字节，存储日期月份，还可以利用日期时间函数进行日期间得计算

Time:存储时间部分得数据

注意:不要使用字符串类型来存储日期时间数据（通常比字符串占用得储存空间小，在进行查找过滤可以利用日期得函数）

使用int存储日期时间不如使用timestamp类型

# 1）B树

B树中每个节点包含了键值和键值对于的数据对象存放地址指针，所以成功搜索一个对象可以不用到达树的叶节点。成功搜索包括节点内搜索和沿某一路径的搜索，成功搜索时间取决于关键码所在的层次以及节点内关键码的数量。

在B树中查找给定关键字的方法是：首先把根结点取来，在根结点所包含的关键字K1,…,kj查找给定的关键字（可用顺序查找或二分查找法），若找到等于给定值的关键字，则查找成功；否则，一定可以确定要查的关键字在某个Ki或Ki+1之间，于是取Pi所指的下一层索引节点块继续查找，直到找到，或指针Pi为空时查找失败。

# 2）B+树

B+树非叶节点中存放的关键码并不指示数据对象的地址指针，非也节点只是索引部分。所有的叶节点在同一层上，包含了全部关键码和相应数据对象的存放地址指针，且叶节点按关键码从小到大顺序链接。如果实际数据对象按加入的顺序存储而不是按关键码次数存储的话，叶节点的索引必须是稠密索引，若实际数据存储按关键码次序存放的话，叶节点索引时稀疏索引。

B+树有2个头指针，一个是树的根节点，一个是最小关键码的叶节点。

所以 B+树有两种搜索方法：

一种是按叶节点自己拉起的链表顺序搜索。

一种是从根节点开始搜索，和B树类似，不过如果非叶节点的关键码等于给定值，搜索并不停止，而是继续沿右指针，一直查到叶节点上的关键码。所以无论搜索是否成功，都将走完树的所有层。

B+ 树中，数据对象的插入和删除仅在叶节点上进行。

这两种处理索引的数据结构的不同之处：  
1）B树中同一键值不会出现多次，并且它有可能出现在叶结点，也有可能出现在非叶结点中。而B+树的键一定会出现在叶结点中，并且有可能在非叶结点中也有可能重复出现，以维持B+树的平衡。  
2）因为B树键位置不定，且在整个树结构中只出现一次，虽然可以节省存储空间，但使得在插入、删除操作复杂度明显增加。B+树相比来说是一种较好的折中。  
3）B树的查询效率与键在树中的位置有关，最大时间复杂度与B+树相同(在叶结点的时候)，最小时间复杂度为1(在根结点的时候)。而B+树的时候复杂度对某建成的树是固定的。

# 将学生表按成绩排序输出前10

初始化表

Student(S#,Sname,Sage,Ssex) 学生表

Course(C#,Cname,T#) 课程表

SC(S#,C#,score) 成绩表

Teacher(T#,Tname) 教师表

create table Student(S# varchar(20),Sname varchar(10),Sage int,Ssex varchar(2))

前面加一列序号：

if

exists(select table\_name from information\_schema.tables

where table\_name='Temp\_Table')

drop table Temp\_Table

go

select 排名=identity(int,1,1),\* INTO Temp\_Table from Student

go

select \* from Temp\_Table

go

drop database [ ] --删除空的没有名字的数据库

问题：查询各科成绩前十名的记录:(不考虑成绩并列情况)

SELECT t1.S# as 学生ID,t1.C# as 课程ID,Score as 分数

FROM SC t1

WHERE score IN (SELECT TOP 10 score

FROM SC

WHERE t1.C#= C#

ORDER BY score DESC

)

ORDER BY t1.C#;