#### 1. SQL语句执行步骤

查询语句书写顺序：select – from- where- group by- having- order by-limit

查询语句执行顺序：from - where -group by - having - select - order by-limit

#### 2. SQL语句的组成

[DDL](https://www.baidu.com/s?wd=DDL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvmvnsnW03nHTLPyuBuWD0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtYPHbdrH6LPj6)：数据库模式定义语言，关键字：create  
[DML](https://www.baidu.com/s?wd=DML&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvmvnsnW03nHTLPyuBuWD0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtYPHbdrH6LPj6)：数据操纵语言，关键字：Insert、delete、update  
[DCL](https://www.baidu.com/s?wd=DCL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YvmvnsnW03nHTLPyuBuWD0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtYPHbdrH6LPj6)：数据库控制语言 ，关键字：grant、remove  
DQL：数据库查询语言，关键字：select

#### 3. union, union all , intersect, minus

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*；

intersect，对两个结果集进行交集操作，不包括重复行，同时进行默认规则的排序；

minus，对两个结果集进行差操作，不包括重复行，同时进行默认规则的排序。

可以在最后一个结果集中指定Order by子句改变排序方式。

#### 4. where and having and on

**where**在有ON条件的SELECT语句中是过滤中间表的约束条件。在没有ON的单表查询中，是限制物理表或者中间查询结果返回记录的约束。在两表或多表连接中是限制连接形成最终中间表的返回结果的约束。

**having**是对查询结果中得到的列发挥作用，筛选数据(也就是说从查询出的结果集再次进行筛选

**ON**条件：是过滤两个链接表笛卡尔积形成中间表的约束条件。

区别和联系：

（1）执行顺序：on、where、having这三个都可以加条件的子句中，on是最先执行，where次之，having最后。

（2）

#### 5. 内连接，外连接，交叉联接

（1）内连接：内联接使用比较运算符根据每个表共有的列的值匹配两个表中的行

（2）外连接：左向外联接、右向外联接或完整外部联接

（3）交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。

（4）LEFT JOIN或LEFT OUTER JOIN：

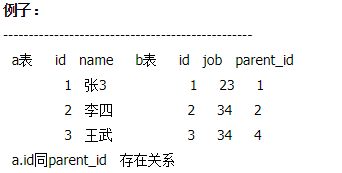
左向外联接的结果集包括 LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。

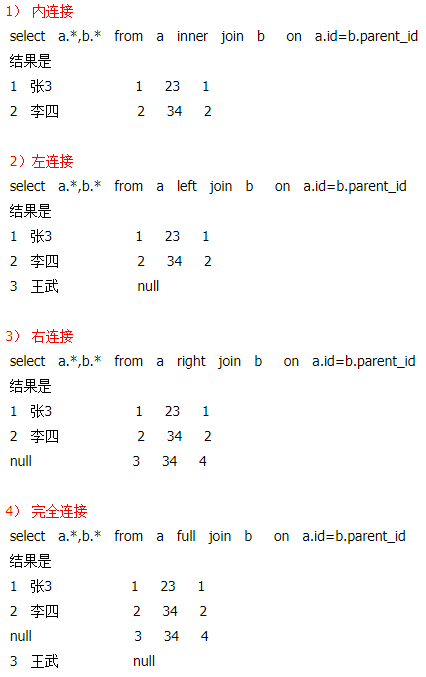
（5）RIGHT JOIN或RIGHT OUTER JOIN

右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。

（6）FULL JOIN 或 FULL OUTER JOIN

完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。





**交叉连接**：有两种，显式的和隐式的，不带ON子句，返回的是两表的乘积，也叫笛卡尔积。

**语句1：隐式的交叉连接，没有CROSS JOIN。**SELECT O.ID, O.ORDER\_NUMBER, C.ID, C.NAME  
FROM ORDERS O , CUSTOMERS C  
WHERE O.ID=1;

**语句2：显式的交叉连接，使用CROSS JOIN。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,  
C.NAME  
FROM ORDERS O CROSS JOIN CUSTOMERS C  
WHERE O.ID=1;

**内连接**（INNER JOIN）：有两种，显式的和隐式的，返回连接表中符合连接条件和查询条件的数据行。（所谓的链接表就是数据库在做查询形成的中间表）。

**语句3：隐式的内连接，没有INNER JOIN，形成的中间表为两个表的笛卡尔积。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,C.NAME  
FROM CUSTOMERS C,ORDERS O  
WHERE C.ID=O.CUSTOMER\_ID;

**语句4：显示的内连接，一般称为内连接，有INNER JOIN，形成的中间表为两个表经过ON条件过滤后的笛卡尔积。**SELECT O.ID,O.ORDER\_NUMBER,C.ID,C.NAME  
FROM CUSTOMERS C INNER JOIN ORDERS O ON C.ID=O.CUSTOMER\_ID;

#### 6. SQL查询的基本原理

第一、**单表查询**：根据WHERE条件过滤表中的记录，形成中间表（这个中间表对用户是不可见的）；然后根据SELECT的选择列选择相应的列进行返回最终结果。

第二、**两表连接查询**：对两表求积（笛卡尔积）并用ON条件和连接连接类型进行过滤形成中间表；然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。

第三、**多表连接查询**：先对第一个和第二个表按照两表连接做查询，然后用查询结果和第三个表做连接查询，以此类推，直到所有的表都连接上为止，最终形成一个中间的结果表，然后根据WHERE条件过滤中间表的记录，并根据SELECT指定的列返回查询结果。

#### 7. MySQL优化的八种方式

**1、选取最适用的字段属性**

MySQL可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

例如，在定义邮政编码这个字段时，如果将其设置为CHAR(255),显然给数据库增加了不必要的空间，甚至使用VARCHAR这种类型也是多余的，因为**CHAR(6)就可以很好的完成**任务了。同样的，如果可以的话，我们应该使用MEDIUMINT而不是BIGIN来定义整型字段。

另外一个提高效率的方法是在可能的情况下，应该**尽量把字段设置为NOTNULL**，这样在将来执行查询的时候，数据库不用去比较NULL值。

对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，**ENUM类型被当作数值型数据来处理**，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

**2、使用连接（JOIN）来代替子查询(Sub-Queries)**

MySQL从4.1开始支持SQL的子查询。这个技术可以使用SELECT语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户ID取出来，然后将结果传递给主查询，如下所示：

**DELETE FROM customer info**

**WHERE Customer ID NOT in (SELECTCustomerIDFROMsalesinfo)**

使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的SQL操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接（JOIN）..替代。例如，假设我们要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

**SELECT \* FROM customer info**

**WHERE Custome rID NOT in (SELECT Customer ID FROM salesinfo)**

如果使用连接（JOIN）..来完成这个查询工作，速度将会快很多。尤其是当salesinfo表中对CustomerID建有索引的话，性能将会更好，查询如下：

**SELECT \* FROM customerinfo**

**LEFT JOIN salesinfo ON customerinfo.CustomerID = salesinfo.CustomerID**

**WHERE salesinfo.CustomerID IS NULL**

连接（JOIN）..之所以更有效率一些，是因为MySQL不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

**3、使用联合(UNION)来代替手动创建的临时表**

MySQL从4.0的版本开始支持union查询，它可以把需要使用临时表的两条或更多的select查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用union来创建查询的时候，我们只需要用UNION作为关键字把多个select语句连接起来就可以了，要注意的是所有select语句中的字段数目要想同。下面的例子就演示了一个使用UNION的查询。

**SELECT Name, Phone FROM client UNION**

**SELECT Name, BirthDate FROM author UNION**

**SELECT Name, Supplier FROM product**

**4、事务**

尽管我们可以使用子查询（Sub-Queries）、连接（JOIN）和联合（UNION）来创建各种各样的查询，但不是所有的数据库操作都可以只用一条或少数几条SQL语句就可以完成的。更多的时候是需要用到一系列的语句来完成某种工作。但是在这种情况下，当这个语句块中的某一条语句运行出错的时候，整个语句块的操作就会变得不确定起来。设想一下，要把某个数据同时插入两个相关联的表中，可能会出现这样的情况：第一个表中成功更新后，数据库突然出现意外状况，造成第二个表中的操作没有完成，这样，就会造成数据的不完整，甚至会破坏数据库中的数据。要避免这种情况，就应该使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句话说，就是可以保持数据库中数据的一致性和完整性。事物以BEGIN关键字开始，COMMIT关键字结束。在这之间的一条SQL操作失败，那么，ROLLBACK命令就可以把数据库恢复到BEGIN开始之前的状态。

**BEGIN;**

**INSERT INTO salesinfo SET CustomerID=14;**

**UPDATE inventory SET Quantity = 11 WHERE item = 'book';**

**COMMIT;**

事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

**5、锁定表**

尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有成千上万的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。

其实，有些情况下我们可以通过锁定表的方法来获得更好的性能。下面的例子就用锁定表的方法来完成前面一个例子中事务的功能。

**LOCKTABLE**

**inventory WRITE SELECT Quantity FROM inventory WHERE Item = 'book';**

**UPDATE inventory SET Quantity = 11 WHERE Item = 'book';**

**UNLOCKTABLES**

这里，我们用一个select语句取出初始数据，通过一些计算，用update语句将新值更新到表中。包含有WRITE关键字的LOCKTABLE语句可以保证在UNLOCKTABLES命令被执行之前，不会有其它的访问来对inventory进行插入、更新或者删除的操作。

**6、使用外键**

锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。

例如，外键可以保证每一条销售记录都指向某一个存在的客户。在这里，外键可以把customerinfo表中的CustomerID映射到salesinfo表中CustomerID，任何一条没有合法CustomerID的记录都不会被更新或插入到salesinfo中。

**CREATE TABLE customerinfo(**

**CustomerID INT NOT NULL,PRIMARYKEY(CustomerID)**

**)TYPE=INNODB;**

**CREATE TABLE salesinfo(**

**SalesID INT NOT NULL,**

**CustomerID INT NOT NULL,**

**PRIMARYKEY(CustomerID,SalesID),**

**FOREIGNKEY(CustomerID) REFERENCES customerinfo(CustomerID) ON DELETE CASCADE**

**)TYPE=INNODB;**

注意例子中的参数“ONDELETECASCADE”。该参数保证当customerinfo表中的一条客户记录被删除的时候，salesinfo表中所有与该客户相关的记录也会被自动删除。如果要在MySQL中使用外键，一定要记住在创建表的时候将表的类型定义为事务安全表InnoDB类型。该类型不是MySQL表的默认类型。定义的方法是在CREATETABLE语句中加上TYPE=INNODB。如例中所示。

**7、使用索引**

索引是提高数据库性能的常用方法，它可以令数据库服务器以比没有索引快得多的速度检索特定的行，尤其是在查询语句当中包含有MAX(),MIN()和ORDERBY这些命令的时候，性能提高更为明显。

**那该对哪些字段建立索引呢？**

一般说来，索引应建立在那些将用于JOIN,WHERE判断和ORDERBY排序的字段上。尽量不要对数据库中某个含有大量重复的值的字段建立索引。对于一个ENUM类型的字段来说，出现大量重复值是很有可能的情况

例如customerinfo中的“province”..字段，在这样的字段上建立索引将不会有什么帮助；相反，还有可能降低数据库的性能。我们在创建表的时候可以同时创建合适的索引，也可以使用ALTERTABLE或CREATEINDEX在以后创建索引。此外，MySQL从版本3.23.23开始支持全文索引和搜索。全文索引在MySQL中是一个FULLTEXT类型索引，但仅能用于MyISAM类型的表。对于一个大的数据库，将数据装载到一个没有FULLTEXT索引的表中，然后再使用ALTERTABLE或CREATEINDEX创建索引，将是非常快的。但如果将数据装载到一个已经有FULLTEXT索引的表中，执行过程将会非常慢。

（1）.添加PRIMARY KEY（主键索引）

ALTER TABLE `table\_name` ADD PRIMARY KEY ( `column` )   
（2）.添加UNIQUE(唯一索引)

ALTER TABLE `table\_name` ADD UNIQUE ( `column`)

（3）.添加INDEX(普通索引)

ALTER TABLE `table\_name` ADD INDEX index\_name ( `column` )

（4）.添加FULLTEXT(全文索引)

ALTER TABLE `table\_name` ADD FULLTEXT ( `column`)

（5）.添加多列索引

ALTER TABLE `table\_name` ADD INDEX index\_name(`column1`, `column2`, `column3` )

8、优化的查询语句

绝大多数情况下，使用索引可以提高查询的速度，但如果SQL语句使用不恰当的话，索引将无法发挥它应有的作用。

下面是应该注意的几个方面。

**首先**，最好是在相同类型的字段间进行比较的操作。

在MySQL3.23版之前，这甚至是一个必须的条件。例如不能将一个建有索引的INT字段和BIGINT字段进行比较；但是作为特殊的情况，在CHAR类型的字段和VARCHAR类型字段的字段大小相同的时候，可以将它们进行比较。

**其次**，在建有索引的字段上尽量不要使用函数进行操作。

例如，在一个DATE类型的字段上使用YEAE()函数时，将会使索引不能发挥应有的作用。所以，下面的两个查询虽然返回的结果一样，但后者要比前者快得多。

**第三**，在搜索字符型字段时，我们有时会使用LIKE关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

例如下面的查询将会比较表中的每一条记录。

**SELECT \* FROM books**

**WHERE name like "MySQL%"**

但是如果换用下面的查询，返回的结果一样，但速度就要快上很多：

**SELECT \* FROM books**

**WHERE name＞="MySQL" and name＜"MySQM"**

**最后**，应该注意避免在查询中让MySQL进行自动类型转换，因为转换过程也会使索引变得不起作用。

#### 8.事务

如果一个数据库声称支持事务的操作，那么该数据库必须要具备以下四个特性：

**⑴ 原子性**（Atomicity）

　　原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，这和前面两篇博客介绍事务的功能是一样的概念，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

**⑵ 一致性**（Consistency）

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

**⑶ 隔离性**（Isolation）

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

　　关于事务的隔离性数据库提供了多种隔离级别，稍后会介绍到。

**⑷ 持久性**（Durability）

　　持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

　　例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

　　以上介绍完事务的四大特性(简称ACID)，现在重点来说明下事务的隔离性，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

update account set money=money+100 where name=’B’; (此时A通知B)

update account set money=money - 100 where name=’A’;

　　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。



　　以上四种隔离级别最高的是Serializable级别，最低的是Read uncommitted级别，当然级别越高，执行效率就越低。像Serializable这样的级别，就是以锁表的方式(类似于Java多线程中的锁)使得其他的线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况。在MySQL数据库中默认的隔离级别为Repeatable read (可重复读)。

　　在MySQL数据库中，支持上面四种隔离级别，默认的为Repeatable read (可重复读)；而在Oracle数据库中，只支持Serializable (串行化)级别和Read committed (读已提交)这两种级别，其中默认的为Read committed级别。

#### 9.索引

**（1）普通索引**

这是最基本的MySQL数据库索引，它没有任何限制。它有以下几种创建方式：

创建索引

**CREATE INDEX indexName ON mytable(username(length));**

如果是CHAR，VARCHAR类型，length可以小于字段实际长度；如果是BLOB和TEXT类型，必须指定 length，下同。

修改表结构

**ALTER mytable ADD INDEX [indexName] ON (username(length))**

创建表的时候直接指定

**CREATE TABLE mytable(**

**ID INT NOT NULL,**

**username VARCHAR(16) NOT NULL,**

**INDEX [indexName] (username(length))**

**);**

删除索引的语法：

**DROP INDEX [indexName] ON mytable;**

**（2）唯一索引**

它与前面的普通索引类似，不同的就是：MySQL数据库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。它有以下几种创建方式：

创建索引

**CREATE UNIQUE INDEX indexName ON mytable(username(length))**

修改表结构

**ALTER mytable ADD UNIQUE [indexName] ON (username(length))**

创建表的时候直接指定

**CREATE TABLE mytable(**

**ID INT NOT NULL,**

**username VARCHAR(16) NOT NULL,**

**UNIQUE [indexName] (username(length))**

**);**

**（3）主键索引**

它是一种特殊的唯一索引，不允许有空值。一般是在建表的时候同时创建主键索引：

**CREATE TABLE mytable(**

**ID INT NOT NULL,**

**username VARCHAR(16) NOT NULL,**

**PRIMARY KEY(ID)**

**);**

当然也可以用 ALTER 命令。记住：一个表只能有一个主键。

**（4）组合索引**

为了形象地对比单列索引和组合索引，为表添加多个字段：

**CREATE TABLE mytable(**

**ID INT NOT NULL,**

**username VARCHAR(16) NOT NULL,**

**city VARCHAR(50) NOT NULL,**

**age INT NOT NULL**

**);**

为了进一步榨取MySQL的效率，就要考虑建立组合索引。就是将 name, city, age建到一个索引里：

**ALTER TABLE mytable ADD INDEX name\_city\_age (name(10),city,age);**

建表时，usernname长度为 16，这里用 10。这是因为一般情况下名字的长度不会超过10，这样会加速索引查询速度，还会减少索引文件的大小，提高INSERT的更新速度。

如果分别在 usernname，city，age上建立单列索引，让该表有3个单列索引，查询时和上述的组合索引效率也会大不一样，远远低于我们的组合索引。虽然此时有了三个索引，但MySQL只能用到其中的那个它认为似乎是最有效率的单列索引。

建立这样的组合索引，其实是相当于分别建立了下面三组组合MySQL数据库索引：

usernname,city,age usernname,city usernname 为什么没有 city，age这样的组合索引呢？这是因为MySQL组合索引“最左前缀”的结果。简单的理解就是只从最左面的开始组合。并不是只要包含这三列的查询都会用到该组合索引，下面的几个SQL就会用到这个组合MySQL数据库索引：

SELECT \* FROM mytable WHREE username="admin" AND city="郑州"

SELECT \* FROM mytable WHREE username="admin"

而下面几个则不会用到：

SELECT \* FROM mytable WHREE age=20 AND city="郑州"

SELECT \* FROM mytable WHREE city="郑州"

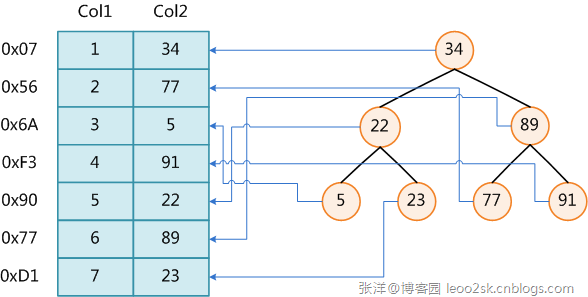
#### 10. 数据库索引的实现原理

说白了，索引问题就是一个查找问题。。。

**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。

为表设置索引要付出代价的：一是增加了数据库的存储空间，二是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。



上图展示了一种可能的索引方式。左边是数据表，一共有两列七条记录，最左边的是数据记录的物理地址（注意逻辑上相邻的记录在磁盘上也并不是一定物理相邻的）。为了加快Col2的查找，可以维护一个右边所示的二叉查找树，每个节点分别包含索引键值和一个指向对应数据记录物理地址的指针，这样就可以运用二叉查找在O(log2n)的复杂度内获取到相应数据。

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。 

也许会有人要问：增加索引有如此多的优点，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，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。

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。

**一般来说，应该在这些列上创建索引：**

（1）在经常需要搜索的列上，可以加快搜索的速度；

（2）在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

（3）在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；

（4）在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

（5）在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

（6）在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。  
 同样，对于有些列不应该创建索引。

**一般来说，不应该创建索引的的这些列具有下列特点：**

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，**修改性能和检索性能是互相矛盾的**。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

根据数据库的功能，可以在数据库设计器中创建三种索引：唯一索引、主键索引和聚集索引。

**唯一索引**

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在employee表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。

**主键索引**

数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。

在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

**聚集索引**

在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

局部性原理与磁盘预读

由于存储介质的特性，磁盘本身存取就比主存慢很多，再加上机械运动耗费，磁盘的存取速度往往是主存的几百分分之一，因此为了提高效率，要尽量减少磁盘I/O。为了达到这个目的，磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定长度的数据放入内存。这样做的理论依据是计算机科学中著名的局部性原理：当一个数据被用到时，其附近的数据也通常会马上被使用。程序运行期间所需要的数据通常比较集中。

由于磁盘顺序读取的效率很高（不需要寻道时间，只需很少的旋转时间），因此对于具有局部性的程序来说，预读可以提高I/O效率。

预读的长度一般为页（page）的整倍数。页是计算机管理存储器的逻辑块，硬件及操作系统往往将主存和磁盘存储区分割为连续的大小相等的块，每个存储块称为一页（在许多操作系统中，页得大小通常为4k），主存和磁盘以页为单位交换数据。当程序要读取的数据不在主存中时，会触发一个缺页异常，此时系统会向磁盘发出读盘信号，磁盘会找到数据的起始位置并向后连续读取一页或几页载入内存中，然后异常返回，程序继续运行。

**B-/+Tree索引的性能分析**

到这里终于可以分析B-/+Tree索引的性能了。

上文说过一般使用磁盘I/O次数评价索引结构的优劣。先从B-Tree分析，根据B-Tree的定义，可知检索一次最多需要访问h个节点。数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：

每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个node只需一次I/O。

B-Tree中一次检索最多需要h-1次I/O（根节点常驻内存），渐进复杂度为O(h)=O(logdN)。一般实际应用中，出度d是非常大的数字，通常超过100，因此h非常小（通常不超过3）。

而红黑树这种结构，h明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性，所以红黑树的I/O渐进复杂度也为O(h)，效率明显比B-Tree差很多。