关于 MySQL 索引的好处，如果正确合理设计并且使用索引的 MySQL 是一辆兰博基尼的话，那么没有设计和使用索引的 MySQL 就是一个人力三轮车。对于没有索引的表，单表查询可能几十万数据就是瓶颈，而通常大型网站单日就可能会产生几十万甚至几百万的数据，没有索引查询会变的非常缓慢。还是以 WordPress 来说，其多个数据表都会对经常被查询的字段添加索引，比如 wp_comments 表中针对 5 个字段设计了 BTREE 索引。

一个简单的对比测试

以我去年测试的数据作为一个简单示例，20 多条数据源随机生成 200 万条数据，平均每条数据源都重复大概 10 万次，表结构比较简单，仅包含一个自增 ID，一个 char 类型，一个 text 类型和一个 int 类型，单表 2G 大小，使用 MyIASM 引擎。开始测试未添加任何索引。

执行下面的 SQL 语句：

```
mysql> SELECT id,FROM_UNIXTIME(time) FROM article WHERE a.title='测试标题'
```

查询需要的时间非常恐怖的，如果加上联合查询和其他一些约束条件，数据库会疯狂的消耗内存，并且会影响前端程序的执行。这时给 title 字段添加一个 BTREE 索引：

```
mysql> ALTER TABLE article ADD INDEX index_article_title ON title(200);
```

再次执行上述查询语句，其对比非常明显：

MySQL 索引的概念

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。更通俗的说，数据库索引好比是一本书前面的目录，能加快数据库的查询速度。上述 SQL 语句，在没有索引的情况下，数据库会遍历全部 200 条数据后选择符合条件的；而有了相应的索引之后，数据库会直接在索引中查找符合条件的选项。如果我们把 SQL 语句换成“SELECT * FROM article WHERE id=2000000”，那么你是希望数据库按照顺序读取完200万行数据以后给你结果还是直接在索引中定位呢？上面的两个图片鲜明的用时对比已经给出了答案（注：**一般数据库默认都会为主键生成索引**）。

索引分为**聚簇索引和非聚簇索引**两种，**聚簇索引是按照数据存放的物理位置为顺序的**，而非聚簇索引就不一样了；**聚簇索引能提高多行检索的速度，而非聚簇索引对于单行的检索很快**。

MySQL 索引的类型

1. 普通索引

这是最基本的索引，它没有任何限制，比如上文中为 title 字段创建的索引就是一个普通索引，MyIASM 中默认的 BTREE 类型的索引，也是我们大多数情况下用到的索引。

- 直接创建索引

```
CREATE INDEX index_name ON table(column(length));
```

- 修改表结构的方式添加索引

```
ALTER TABLE table_name ADD INDEX index_name ON (column(length));
```

- 创建表的时候同时创建索引

```
CREATE TABLE `table` (
    `id` int(11) NOT NULL AUTO_INCREMENT ,
    `title` char(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL ,
    `content` text CHARACTER SET utf8 COLLATE utf8_general_ci NULL ,
    `time` int(10) NULL DEFAULT NULL ,
    PRIMARY KEY (`id`),
    INDEX index_name (title(length)));
```

- 删除索引

```
DROP INDEX index_name ON table;
```

2. 唯一索引

与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似。

--创建唯一索引

```
CREATE UNIQUE INDEX indexName ON table(column(length))
```

--修改表结构

```
ALTER TABLE table_name ADD UNIQUE indexName ON(column(length))
```

- 创建表的时候同时创建索引

```
CREATE TABLE `table` (
    `id` int(11) NOT NULL AUTO_INCREMENT ,
    `title` char(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL ,
    `content` text CHARACTER SET utf8 COLLATE utf8_general_ci NULL ,
    `time` int(10) NULL DEFAULT NULL ,
    PRIMARY KEY (`id`),
    UNIQUE indexName (title(length)));
```

3. 全文索引（FULLTEXT）

MySQL 从 3.23.23 版开始支持全文索引和全文检索，FULLTEXT 索引仅可用于 MyISAM 表；他们可以从 CHAR、VARCHAR 或 TEXT 列中作为 CREATE TABLE 语句的一部分被创建，或是随后使用 ALTER TABLE 或 CREATE INDEX 被添加。////对于较大的数据集，将你的资料输入一个没有 FULLTEXT 索引的表中，然后创建索引，其速度比把资料输入现有 FULLTEXT 索引的速度更为快。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。

--创建表的适合添加全文索引

```
CREATE TABLE `table` (
    `id` int(11) NOT NULL AUTO_INCREMENT ,
    `title` char(255) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL ,
    `content` text CHARACTER SET utf8 COLLATE utf8_general_ci NULL ,
    `time` int(10) NULL DEFAULT NULL ,
    PRIMARY KEY (`id`),
```

```
    FULLTEXT (content));
```

–修改表结构添加全文索引

```
ALTER TABLE article ADD FULLTEXT index_content(content)
```

–直接创建索引

```
CREATE FULLTEXT INDEX index_content ON article(content)
```

4. 单列索引、多列索引

多个单列索引与单个多列索引的查询效果不同，因为执行查询时，MySQL 只能使用一个索引，会从多个索引中选择一个限制最为严格的索引。

5. 组合索引（最左前缀）

平时用的 SQL 查询语句一般都有比较多的限制条件,所以为了进一步榨取 MySQL 的效率，就要考虑建立组合索引。例如上表中针对 title 和 time 建立一个组合索引:ALTER TABLE article ADD INDEX index_titme_time (title(50),time(10))。建立这样的组合索引，其实是相当于分别建立了下面两组组合索引：

- title,time

- title

为什么没有 time 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。并不是只要包含这两列的查询都会用到该组合索引，如下面的几个 SQL 所示：

1 –使用到上面的索引

```
SELECT * FROM article WHREE title='测试
2
' AND time=1234567890;
```

```
SELECT * FROM article WHREE utitle='测试
3
';
```

4 –不使用上面的索引

```
  SELECT   * FROM   article
5
  WHREE time=1234567890;
```

MySQL 索引的优化

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点：虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行 INSERT、UPDATE 和 DELETE。因为更新表时，MySQL 不仅要保存数据，还要保存一下索引文件。建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的 MySQL 有大数据量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。下面是一些总结以及收藏的 MySQL 索引的注意事项和优化方法。

1. 何时使用聚集索引或非聚集索引？

动作描述	使用聚集索引	使用非聚集索引
列经常被分组排序	使用	使用
返回某范围内的数据	使用	不使用
一个或极少不同值	不使用	不使用
小数目的不同值	使用	不使用
大数目的不同值	不使用	使用
频繁更新的列	不使用	使用
外键列	使用	使用
主键列	使用	使用
频繁修改索引列	不使用	使用

事实上，我们可以通过前面聚集索引和非聚集索引的定义的例子来理解上表。如：返回某范围内的数据一项。比如您的某个表有一个时间列，恰好您把聚合索引建立在了该列，这时您查询 2004 年 1 月 1 日至 2004 年 10 月 1 日之间的全部数据时，这个速度就将是很快的，因为您的这本字典正文是按日期进行排序的，聚类索引只需要找到要检索的所有数据中的开头和结尾数据即可；而不像非聚集索引，必须先查到目录中查到每一项数据对应的页码，然后再根据页码查到具体内容。其实这个具体用法我还不是很理解，只能等待后期的项目开发中慢慢学学了。

2. 索引不会包含有 NULL 值的列

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。

3. 使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

4. 索引列排序

MySQL查询只使用一个索引,因此如果where子句中已经使用了索引的话,那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

5. like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

6. 不要在列上进行运算

例如：select * from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select * from users where adddate<’2007-01-01′。关于这一点可以围观：一个单引号引发的MYSQL性能损失。

最后总结一下，MySQL只对一下操作符才使用索引：<,<=,=,>,>=,between,in,以及某些时候的like(不以通配符%或_开头的情形)。而理论上每张表里面最多可创建16个索引，不过除非是数据量真的很多，否则过多的使用索引也不是那么好玩的，比如我刚才针对text类型的字段创建索引的时候，系统差点就卡死了。