# MySQL索引

## 索引是万能的吗？

索引就是帮助数据库管理系统高效获取数据的数据结构。

索引不是万能的，在有些情况下使用索引反而会让效率变低。

索引的价值是帮我们从海量数据中找到想要的数据，如果数据量少，那么是否使用索引对结果的影响并不大。

在数据表中的数据行数比较少的情况下，比如不到 1000 行，是不需要创建索引的。另外，当数据重复度大，比如高于 10% 的时候，也不需要对这个字段使用索引。

我之前讲到过，如果是性别这个字段，就不需要对它创建索引。这是为什么呢？如果你想要在 100 万行数据中查找其中的 50 万行（比如性别为男的数据），一旦创建了索引，你需要先访问 50 万次索引，然后再访问 50 万次数据表，这样加起来的开销比不使用索引可能还要大。

## 索引的种类有哪些？

**从功能逻辑上说，索引主要有 4 种，分别是普通索引、唯一索引、主键索引和全文索引。**

普通索引是基础的索引，没有任何约束，主要用于提高查询效率。

唯一索引就是在普通索引的基础上增加了数据唯一性的约束，在一张数据表里可以有多个唯一索引。

主键索引在唯一索引的基础上增加了不为空的约束，也就是 NOT NULL+UNIQUE，一张表里最多只有一个主键索引。

全文索引用的不多，MySQL 自带的全文索引只支持英文。我们通常可以采用专门的全文搜索引擎，比如 ES(ElasticSearch) 和 Solr。

其实前三种索引（普通索引、唯一索引和主键索引）都是一类索引，只不过对数据的约束性逐渐提升。在一张数据表中只能有一个主键索引，这是由主键索引的物理实现方式决定的，因为数据存储在文件中只能按照一种顺序进行存储。但可以有多个普通索引或者多个唯一索引。

**按照物理实现方式，索引可以分为 2 种：聚集索引和非聚集索引。我们也把非聚集索引称为二级索引或者辅助索引。**

聚集索引可以按照主键来排序存储数据

聚集索引指表中数据行按索引的排序方式进行存储，对查找行很有效。只有当表包含聚集索引时，表内的数据行才会按找索引列的值在磁盘上进行物理排序和存储。每一个表只能有一个聚集索引，因为数据行本身只能按一个顺序存储。

聚集索引与非聚集索引的原理不同，在使用上也有一些区别：

* 聚集索引的叶子节点存储的就是我们的数据记录，非聚集索引的叶子节点存储的是数据位置。非聚集索引不会影响数据表的物理存储顺序。
* 一个表只能有一个聚集索引，因为只能有一种排序存储的方式，但可以有多个非聚集索引，也就是多个索引目录提供数据检索。
* 使用聚集索引的时候，数据的查询效率高，但如果对数据进行插入，删除，更新等操作，效率会比非聚集索引低。

对 WHERE 子句的字段建立索引，可以大幅提升查询效率。 采用聚集索引进行数据查询，比使用非聚集索引的查询效率略高。如果查询次数比较多，还是尽量使用主键索引进行数据查询。

**除了业务逻辑和物理实现方式，索引还可以按照字段个数进行划分，分成单一索引和联合索引。**

这里需要说明的是联合索引存在最左匹配原则，也就是按照最左优先的方式进行索引的匹配。

使用索引可以帮助我们从海量的数据中快速定位想要查找的数据，不过索引也存在一些不足，比如占用存储空间、降低数据库写操作的性能等，如果有多个索引还会增加索引选择的时间。当我们使用索引时，需要平衡索引的利（提升查询效率）和弊（维护索引所需的代价）。

在实际工作中，我们还需要基于需求和数据本身的分布情况来确定是否使用索引，尽管索引不是万能的，但数据量大的时候不使用索引是不可想象的，毕竟索引的本质，是帮助我们提升数据检索的效率。

## 如何评价索引的数据结构设计好坏

数据库服务器有两种存储介质，分别为硬盘和内存。内存属于临时存储，容量有限，而且当发生意外时（比如断电或者发生故障重启）会造成数据丢失；硬盘相当于永久存储介质，这也是为什么我们需要把数据保存到硬盘上。

虽然内存的读取速度很快，但我们还是需要将索引存放到硬盘上，这样的话，当我们在硬盘上进行查询时，也就产生了硬盘的 I/O 操作。相比于内存的存取来说，硬盘的 I/O 存取消耗的时间要高很多。我们通过索引来查找某行数据的时候，需要计算产生的磁盘 I/O 次数，当磁盘 I/O 次数越多，所消耗的时间也就越大。如果我们能让索引的数据结构尽量减少硬盘的 I/O 操作，所消耗的时间也就越小。

## 什么是B树

如果用二叉树作为索引的实现结构，会让树变得很高，增加硬盘的 I/O 次数，影响数据查询的时间。因此一个节点就不能只有 2 个子节点，而应该允许有 M 个子节点 (M>2)。

B 树的出现就是为了解决这个问题，B 树的英文是 Balance Tree，也就是平衡的多路搜索树，它的高度远小于平衡二叉树的高度。在文件系统和数据库系统中的索引结构经常采用 B 树来实现。

你能看出来在 B 树的搜索过程中，我们比较的次数并不少，但如果把数据读取出来然后在内存中进行比较，这个时间就是可以忽略不计的。而读取磁盘块本身需要进行 I/O 操作，消耗的时间比在内存中进行比较所需要的时间要多，是数据查找用时的重要因素，B 树相比于平衡二叉树来说磁盘 I/O 操作要少，在数据查询中比平衡二叉树效率要高。

## 什么是B+树

整个过程一共进行了 3 次 I/O 操作，看起来 B+ 树和 B 树的查询过程差不多，但是 B+ 树和 B 树有个根本的差异在于，B+ 树的中间节点并不直接存储数据。这样的好处都有什么呢？

首先，B+ 树查询效率更稳定。因为 B+ 树每次只有访问到叶子节点才能找到对应的数据，而在 B 树中，非叶子节点也会存储数据，这样就会造成查询效率不稳定的情况，有时候访问到了非叶子节点就可以找到关键字，而有时需要访问到叶子节点才能找到关键字。

其次，B+ 树的查询效率更高，这是因为通常 B+ 树比 B 树更矮胖（阶数更大，深度更低），查询所需要的磁盘 I/O 也会更少。同样的磁盘页大小，B+ 树可以存储更多的节点关键字。

不仅是对单个关键字的查询上，在查询范围上，B+ 树的效率也比 B 树高。这是因为所有关键字都出现在 B+ 树的叶子节点中，并通过有序链表进行了链接。而在 B 树中则需要通过中序遍历才能完成查询范围的查找，效率要低很多。

磁盘的 I/O 操作次数对索引的使用效率至关重要。虽然传统的二叉树数据结构查找数据的效率高，但很容易增加磁盘 I/O 操作的次数，影响索引使用的效率。因此在构造索引的时候，我们更倾向于采用“矮胖”的数据结构。

B 树和 B+ 树都可以作为索引的数据结构，在 MySQL 中采用的是 B+ 树，B+ 树在查询性能上更稳定，在磁盘页大小相同的情况下，树的构造更加矮胖，所需要进行的磁盘 I/O 次数更少，更适合进行关键字的范围查询。

## Hash 索引与 B+ 树索引的区别

* Hash 索引不能进行范围查询，而 B+ 树可以。这是因为 Hash 索引指向的数据是无序的，而 B+ 树的叶子节点是个有序的链表。
* Hash 索引不支持联合索引的最左侧原则（即联合索引的部分索引无法使用），而 B+ 树可以。对于联合索引来说，Hash 索引在计算 Hash 值的时候是将索引键合并后再一起计算 Hash 值，所以不会针对每个索引单独计算 Hash 值。因此如果用到联合索引的一个或者几个索引时，联合索引无法被利用。
* Hash 索引不支持 ORDER BY 排序，因为 Hash 索引指向的数据是无序的，因此无法起到排序优化的作用，而 B+ 树索引数据是有序的，可以起到对该字段 ORDER BY 排序优化的作用。同理，我们也无法用 Hash 索引进行模糊查询，而 B+ 树使用 LIKE 进行模糊查询的时候，LIKE 后面前模糊查询（比如 % 开头）的话就可以起到优化作用。

## 创建索引有哪些规律？

1. 字段的数值有唯一性的限制，比如用户名
2. 频繁作为 WHERE 查询条件的字段，尤其在数据表大的情况下
3. 需要经常 GROUP BY 和 ORDER BY 的列

实际上多个单列索引在多条件查询时只会生效一个索引（MySQL 会选择其中一个限制最严格的作为索引），所以在多条件联合查询的时候最好创建联合索引。

1. UPDATE、DELETE 的 WHERE 条件列，一般也需要创建索引
2. DISTINCT 字段需要创建索引
3. 做多表 JOIN 连接操作时，创建索引需要注意以下的原则

## 什么时候不需要创建索引

WHERE 条件（包括 GROUP BY、ORDER BY）里用不到的字段不需要创建索引，索引的价值是快速定位，如果起不到定位的字段通常是不需要创建索引的。

第二种情况是，如果表记录太少，比如少于 1000 个，那么是不需要创建索引的。

第三种情况是，字段中如果有大量重复数据，也不用创建索引，比如性别字段。

最后一种情况是，频繁更新的字段不一定要创建索引。

## 什么情况下索引失效

1. 如果索引进行了表达式计算，则会失效
2. 如果对索引使用函数，也会造成失效
3. 在 WHERE 子句中，如果在 OR 前的条件列进行了索引，而在 OR 后的条件列没有进行索引，那么索引会失效。
4. 当我们使用 LIKE 进行模糊查询的时候，前面不能是 %
5. 索引列尽量设置为 NOT NULL 约束。
6. 我们在使用联合索引的时候要注意最左原则

实际工作中，查询的需求多种多样，创建的索引也会越来越多。这时还需要注意，我们要尽可能扩展索引，而不是新建索引，因为索引数量过多需要维护的成本也会变大，导致写效率变低。同时，我们还需要定期查询使用率低的索引，对于从未使用过的索引可以进行删除，这样才能让索引在 SQL 查询中发挥最大价值。