无语，我差点被面试官怼坏了，又给我问到MySQL索引

Java程序猿阿谷 关注
2 2020.05.18 20:42:09 字数 5,144 阅读 51,553

前一阵子，又跑出去搞了一场面试，心态算是崩了，关于MySQL索引的原理及使用被面试官怼的体无完肤，立志要总结一番，然后一直没有时间（其实是懒.....），准备好了吗？

image

一、MySQL中索引的语法

创建索引

在创建表的时候添加索引

```
CREATE TABLE mytable(
    ID INT NOT NULL,
    username VARCHAR(16) NOT NULL,
    INDEX [indexName] (username(length))
);
```

在创建表以后添加索引

```
ALTER TABLE my_table ADD [UNIQUE] INDEX index_name(column_name);
或者
CREATE INDEX index_name ON my_table(column_name);
```

注意：

1、索引需要占用**磁盘空间**，因此在创建索引时要考虑到磁盘空间是否足够

2、创建索引时需要**对表加锁**，因此实际操作中需要在业务空闲期间进行

根据索引查询

```
具体查询：
SELECT * FROM table_name WHERE column_1=column_2;(为column_1建立了索引)

或者模糊查询
SELECT * FROM table_name WHERE column_1 LIKE '%三'
SELECT * FROM table_name WHERE column_1 LIKE '三%'
```

```
SELECT * FROM table_name WHERE column_1 LIKE '%三%'

SELECT * FROM table_name WHERE column_1 LIKE '_好_'

如果要表示在字符串中既有A又有B，那么查询语句为：
SELECT * FROM table_name WHERE column_1 LIKE '%A%' AND column_1 LIKE '%B%';

SELECT * FROM table_name WHERE column_1 LIKE '[张李王]三';  //表示column_1中有匹配张三、李三、王三的
SELECT * FROM table_name WHERE column_1 LIKE '[^张李王]三';  //表示column_1中有匹配除了张三、李三、王

//在模糊查询中，%表示任意0个或多个字符；_表示任意单个字符（有且仅有），通常用来限制字符串长度;[]表示其中的

或者在全文索引中模糊查询
SELECT * FROM table_name WHERE MATCH(content) AGAINST('word1','word2',...);
```

删除索引

```
DROP INDEX my_index ON tablename；
或者
ALTER TABLE table_name DROP INDEX index_name;
```

查看表中的索引

```
SHOW INDEX FROM tablename
```

查看查询语句使用索引的情况

```
//explain 加查询语句
explain SELECT * FROM table_name WHERE column_1='123';
```

二、索引的优缺点

优势：可以快速检索，减少I/O次数，加快检索速度；根据索引分组和排序，可以加快分组和排序；

劣势：索引本身也是表，因此会占用存储空间，一般来说，索引表占用的空间的数据表的1.5倍；索引表的维护和创建需要时间成本，这个成本随着数据量增大而增大；构建索引会降低数据表的修改操作（删除，添加，修改）的效率，因为在修改数据表的同时还需要修改索引表；

三、索引的分类

常见的索引类型有：主键索引、唯一索引、普通索引、全文索引、组合索引

1、主键索引：即主索引，根据主键pk_clolum（length）建立索引，**不允许重复，不允许空值**；

```
ALTER TABLE 'table_name' ADD PRIMARY KEY pk_index('col')；
```

2、唯一索引：用来建立索引的列的值必须是**唯一的，允许空值**

```
ALTER TABLE 'table_name' ADD UNIQUE index_name('col')；
```

3、普通索引：用表中的普通列构建的索引，没有任何限制

```
ALTER TABLE 'table_name' ADD INDEX index_name('col')；
```

4、全文索引：用大文本对象的列构建的索引（下一部分会讲解）

```
ALTER TABLE 'table_name' ADD FULLTEXT INDEX ft_index('col');
```

5、组合索引：用多个列组合构建的索引，这多个列中的值不允许有空值

```
ALTER TABLE 'table_name' ADD INDEX index_name('col1','col2','col3');
```

*遵循"最左前缀"原则，把最常用作为检索或排序的列放在最左，依次递减，组合索引相当于建立了col1,col1col2,col1col2col3三个索引，而col2或者col3是不能使用索引的。

*在使用组合索引的时候可能因为列名长度过长而导致索引的key太大，导致效率降低，在允许的情况下，可以只取col1和col2的前几个字符作为索引

ALTER TABLE 'table_name' ADD INDEX index_name(col1(4),col2（3))；

表示使用col1的前4个字符和col2的前3个字符作为索引

四、索引的实现原理

MySQL支持诸多存储引擎，而各种存储引擎对索引的支持也各不相同，因此MySQL数据库支持多种索引类型，如BTree**索引，**B+Tree**索引，哈希索引，全文索引**等等，

1、哈希索引：

只有memory（内存）存储引擎支持哈希索引，哈希索引用索引列的值计算该值的hashCode，然后在hashCode相应的位置存执该值所在行数据的物理位置，因为使用散列算法，因此访问速度非常快，但是一个值只能对应一个hashCode，而且是散列的分布方式，因此哈希索引不支持范围查找和排序的功能。

2、全文索引：

FULLTEXT（全文）索引，仅可用于MyISAM和InnoDB，针对较大的数据，生成全文索引非常的消耗时间和空间。对于文本的大对象，或者较大的CHAR类型的数据，如果使用普通索引，那么匹配文本前几个字符还是可行的，但是想要匹配文本中间的几个单词，那么就要使用LIKE %word%来匹配，这样需要很长的时间来处理，响应时间会大大增加，这种情况，就可使用时FULLTEXT索引了，在生成FULLTEXT索引时，会为文本生成一份单词的清单，在索引时及根据这个单词的清单来索引。FULLTEXT可以在创建表的时候创建，也可以在需要的时候用ALTER或者CREATE INDEX来添加：

```
//创建表的时候添加FULLTEXT索引
CTREATE TABLE my_table(
    id INT(10) PRIMARY KEY,
    name VARCHAR(10) NOT NULL,
    my_text TEXT,
    FULLTEXT(my_text)
)ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

```
//创建表以后，在需要的时候添加FULLTEXT索引
ALTER TABLE my_table ADD FULLTEXT INDEX ft_index(column_name);
```

全文索引的查询也有自己特殊的语法，而不能使用LIKE %查询字符串%的模糊查询语法

```
SELECT * FROM table_name MATCH(ft_index) AGAINST('查询字符串');
```

注意：

*对于较大的数据集，把数据添加到一个没有FULLTEXT索引的表，然后添加FULLTEXT索引的速度比把数据添加到一个已经有FULLTEXT索引的表快。

*5.6版本前的MySQL自带的全文索引只能用于MyISAM存储引擎，如果是其它数据引擎，那么全文索引不会生效。5.6版本之后InnoDB存储引擎开始支持全文索引

*在MySQL中，全文索引支队英文有用，目前对中文还不支持。5.7版本之后通过使用ngram插件开始支持中文。

*在MySQL中，如果检索的字符串太短则无法检索得到预期的结果，检索的字符串长度至少为4字节，此外，如果检索的字符包括停止词，那么停止词会被忽略。

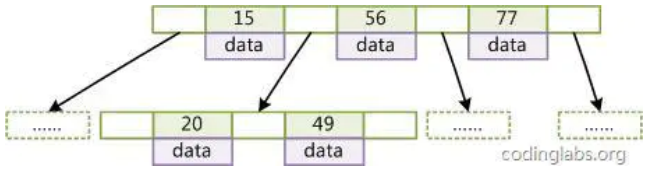
3、BTree索引和B+Tree索引

- BTree索引

BTree是平衡搜索多叉树，设树的度为2d（d>1），高度为h，那么BTree要满足以一下条件：

- 每个叶子结点的高度一样，等于h；
- 每个非叶子结点由n-1个key和n个指针point组成，其中d<=n<=2d,key和point相互间隔，结点两端一定是key；
- 叶子结点指针都为null；
- 非叶子结点的key都是[key,data]二元组，其中key表示作为索引的键，data为键值所在行的数据；

BTree的结构如下：



image

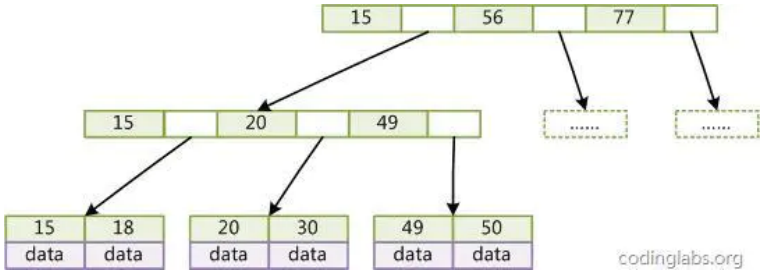
在BTree的机构下，就可以使用二分查找的查找方式，查找复杂度为h*log(n)，一般来说树的高度是很小的，一般为3左右，因此BTree是一个非常高效的查找结构。

- **B+Tree索引**

B+Tree是BTree的一个变种，设d为树的度数，h为树的高度，B+Tree和BTree的不同主要在于：

- B+Tree中的非叶子结点不存储数据，只存储键值；
- B+Tree的叶子结点没有指针，所有键值都会出现在叶子结点上，且key存储的键值对应data数据的物理地址；
- B+Tree的每个非叶子节点由n**个键值**key和n**个指针**point组成；

B+Tree的结构如下：



image

B+Tree对比BTree的优点：

1、磁盘读写代价更低

一般来说B+Tree比BTree更适合实现外存的索引结构，因为存储引擎的设计专家巧妙的利用了外存（磁盘）的存储结构，即磁盘的最小存储单位是扇区（sector），而操作系统的块（block）通常是整数倍的sector，操作系统以页（page）为单位管理内存，一页（page）通常默认为4K，数据库的页通常设置为操作系统页的整数倍，因此索引结构的节点被设计为一个页的大小，然后利用外存的"预读取"原则，每次读取的时候，把整个节点的数据读取到内存中，然后在内存中查找，已知内存的读取速度是外存读取I/O速度的几百倍，那么提升查找速度的关键就在于尽可能少的磁盘I/O，那么可以知道，每个节点中的key个数越多，那么树的高度越小，需要I/O的次数越少，因此一般来说B+Tree比BTree更快，因为B+Tree的非叶节点中不存储data，就可以存储更多的key。

2、查询速度更稳定

由于B+Tree非叶子节点不存储数据（data），因此所有的数据都要查询至叶子节点，而叶子节点的高度都是相同的，因此所有数据的查询速度都是一样的。

更多操作系统内容参考：

硬盘结构

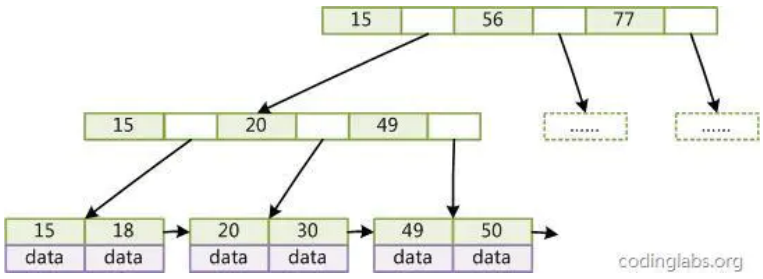
扇区、块、簇、页的区别

操作系统层优化（进阶，初学不用看）

- 带顺序索引的B+TREE

很多存储引擎在B+Tree的基础上进行了优化，添加了指向相邻叶节点的指针，形成了带有顺序访问指针的B+Tree，这样做是为了提高区间查找的效率，只要找到第一个值那么就可以顺序的查找后面的值。

B+Tree的结构如下：



image

聚簇索引和非聚簇索引

分析了MySQL的索引结构的实现原理，然后我们来看看具体的存储引擎怎么实现索引结构的，MySQL中最常见的两种存储引擎分别是MyISAM和InnoDB，分别实现了非聚簇索引和聚簇索引。

聚簇索引的解释是:聚簇索引的顺序就是数据的物理存储顺序

非聚簇索引的解释是:索引顺序与数据物理排列顺序无关

（这样说起来并不好理解，让人摸不着头脑，清继续看下文，并在插图下方对上述两句话有解释）

首先要介绍几个概念，在索引的分类中，我们可以按照索引的键是否为主键来分为“主索引”和“辅助索引”，使用主键键值建立的索引称为“主索引”，其它的称为“辅助索引”。因此主索引只能有一个，辅助索引可以有很多个。

MyISAM——**非聚簇索引**

- MyISAM存储引擎采用的是非聚簇索引，非聚簇索引的主索引和辅助索引几乎是一样的，只是主索引不允许重复，不允许空值，他们的叶子结点的key都存储指向键值对应的数据的物理地址。
- 非聚簇索引的数据表和索引表是分开存储的。
- 非聚簇索引中的数据是根据数据的插入顺序保存。因此非聚簇索引更适合单个数据的查询。插入顺序不受键值影响。
- 只有在MyISAM中才能使用FULLTEXT索引。(mysql5.6以后innoDB也支持全文索引)

最开始我一直不懂既然非聚簇索引的主索引和辅助索引指向相同的内容，为什么还要辅助索引这个东西呢，后来才明白索引不就是用来查询的吗，用在那些地方呢，不就是WHERE和ORDER BY语句后面吗，那么如果查询的条件不是主键怎么办呢，这个时候就需要辅助索引了。

InnoDB——**聚簇索引**

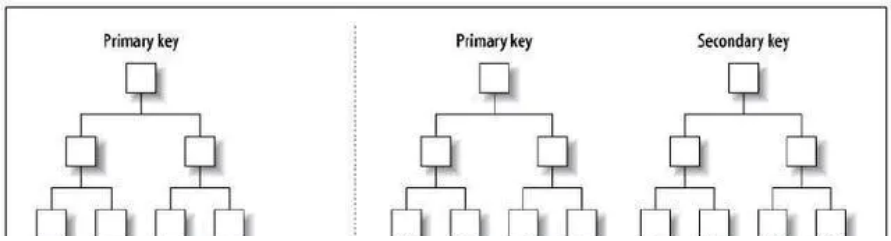
- 聚簇索引的主索引的叶子结点存储的是键值对应的数据本身，辅助索引的叶子结点存储的是键值对应的数据的主键键值。因此主键的值长度越小越好，类型越简单越好。
- 聚簇索引的数据和主键索引存储在一起。
- 聚簇索引的数据是根据主键的顺序保存。因此适合按主键索引的区间查找，可以有更少的磁盘I/O，加快查询速度。但是也是因为这个原因，聚簇索引的插入顺序最好按照主键单调的顺序插入，否则会频繁的引起页分裂，严重影响性能。
- 在InnoDB中，如果只需要查找索引的列，就尽量不要加入其它的列，这样会提高查询效率。

使用主索引的时候，更适合使用聚簇索引，因为聚簇索引只需要查找一次，而非聚簇索引在查到数据的地址后，还要进行一次I/O查找数据。

*因为聚簇辅助索引存储的是主键的键值，因此可以在数据行移动或者页分裂的时候降低成本，因为这时不用维护辅助索引。但是由于主索引存储的是数据本身，因此聚簇索引会占用更多的空间。

*聚簇索引在插入新数据的时候比非聚簇索引慢很多，因为插入新数据时需要检测主键是否重复，这需要遍历主索引的所有叶节点，而非聚簇索引的叶节点保存的是数据地址，占用空间少，因此分布集中，查询的时候I/O更少，但聚簇索引的主索引中存储的是数据本身，数据占用空间大，分布范围更大，可能占用好多的扇区，因此需要更多次I/O才能遍历完毕。

下图可以形象的说明聚簇索引和非聚簇索引的区别



image

从上图中可以看到聚簇索引的辅助索引的叶子节点的data存储的是主键的值，主索引的叶子节点的data存储的是数据本身，也就是说数据和索引存储在一起，并且索引查询到的地方就是数据（data）本身，那么索引的顺序和数据本身的顺序就是相同的；

而非聚簇索引的主索引和辅助索引的叶子节点的data都是存储的数据的物理地址，也就是说索引和数据并不是存储在一起的，数据的顺序和索引的顺序并没有任何关系，也就是索引顺序与数据物理排列顺序无关。

此外MyISAM和innoDB的区别总结如下：

MyISAM和innoDB引擎对比

	MyISAM	innoDB
索引类型	非聚簇	聚簇
支持事务	是	否
支持表锁	是	是
支持行锁	否	是（默认）
支持外键	否	是
支持全文索引	是	是（5.6以后支持）
适用操作类型	大量select下使用	大量insert、delete和update下使用

image

总结如下：

- InnoDB 支持事务，支持行级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；
- MyISAM 不支持事务，支持表级别锁定，支持 B-tree、Full-text 等索引，不支持 Hash 索引；

此外，Memory 不支持事务，支持表级别锁定，支持 B-tree、Hash 等索引，不支持 Full-text 索引；

五、索引的使用策略

什么时候要使用索引?

- 主键自动建立唯一索引；
- 经常作为查询条件在WHERE或者ORDER BY 语句中出现的列要建立索引；
- 作为排序的列要建立索引；
- 查询中与其他表关联的字段，外键关系建立索引
- 高并发条件下倾向组合索引；
- 用于聚合函数的列可以建立索引，例如使用了max(column_1)或者count(column_1)时的column_1就需要建立索引

什么时候不要使用索引?

- 经常增删改的列不要建立索引；
- 有大量重复的列不建立索引；
- 表记录太少不要建立索引。只有当数据库里已经有了足够多的测试数据时，它的性能测试结果才有实际参考价值。如果在测试数据库里只有几百条数据记录，它们往往在执行完第一条查询命令之后就被全部加载到内存里，这将使后续的查询命令都执行得非常快--不管有没有使用索引。只有当数据库里的记录超过了1000条、数据总量也超过了MySQL服务器上的内存总量时，数据库的性能测试结果才有意义。

索引失效的情况：

- 在组合索引中不能有列的值为NULL，如果有，那么这一列对组合索引就是无效的。
- 在一个SELECT语句中，索引只能使用一次，如果在WHERE中使用了，那么在ORDER BY中就不要用了。
- LIKE操作中，'%aaa%'不会使用索引，也就是索引会失效，但是'aaa%'可以使用索引。
- 在索引的列上使用表达式或者函数会使索引失效，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'。其它通配符同样，也就是说，在查询条件中使用正则表达式时，只有在搜索模板的第一个字符不是通配符的情况下才能使用索引。
- 在查询条件中使用不等于，包括<符号、>符号和！=会导致索引失效。特别的是如果对主键索引使用！=则不会使索引失效，如果对主键索引或者整数类型的索引使用<符号或者>符号不会使索引失效。（经erwkjrfhjwkdb同学提醒，不等于，包括<符号、>符号和！，如果占总记录的比例很小的话，也不会失效）
- 在查询条件中使用IS NULL或者IS NOT NULL会导致索引失效。
- 字符串不加单引号会导致索引失效。更准确的说是类型不一致会导致失效，比如字段email是字符串类型的，使用WHERE email=99999 则会导致失败，应该改为WHERE email='99999'。
- 在查询条件中使用OR连接多个条件会导致索引失效，除非OR链接的每个条件都加上索引，这时应该改为两次查询，然后用UNION ALL连接起来。
- 如果排序的字段使用了索引，那么select的字段也要是索引字段，否则索引失效。特别的是如果排序的是主键索引则select * 也不会导致索引失效。
- 尽量不要包括多列排序，如果一定要，最好为这队列构建组合索引；

六、索引的优化

1、最左前缀

索引的最左前缀和和B+Tree中的"最左前缀原理"有关，举例来说就是如果设置了组合索引<col1,col2,col3>那么以下3中情况可以使用索引：col1，<col1,col2>，<col1,col2,col3>，其它的列，比如<col2,col3>，<col1,col3>，col2，col3等等都是不能使用索引的。

根据最左前缀原则，我们一般把排序分组频率最高的列放在最左边，以此类推。

2、带索引的模糊查询优化

在上面已经提到，使用LIKE进行模糊查询的时候，'%aaa%'不会使用索引，也就是索引会失效。如果是这种情况，只能使用全文索引来进行优化（上文有讲到）。

3、为检索的条件构建全文索引，然后使用

```
SELECT * FROM tablename MATCH(index_colum) ANGAINST('word');
```

4、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。