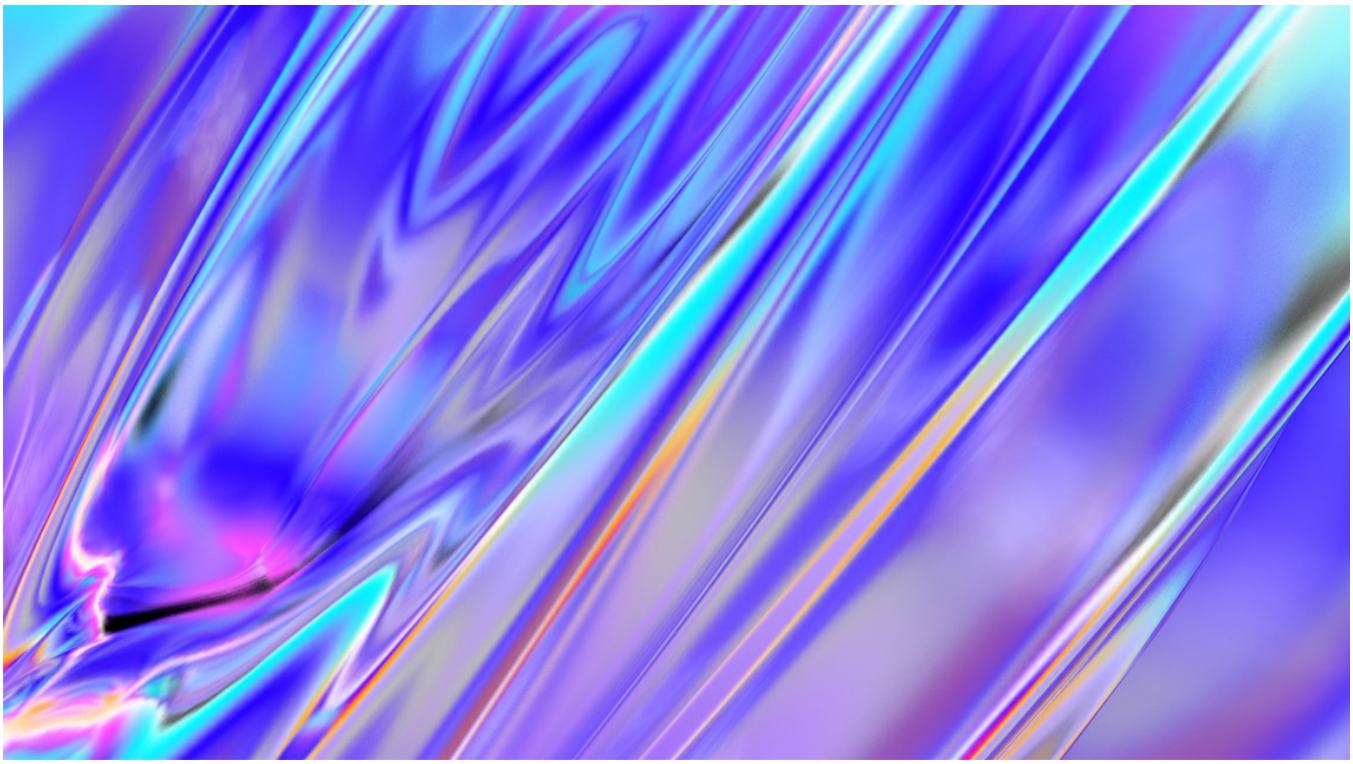
23丨索引的概览：用还是不用索引，这是一个问题

2019-08-02 陈旸



提起优化SQL，你可能会把它理解为优化索引。简单来说这也不算错，索引在SQL优化中占了很大的比重。索引用得好，可以将SQL查询的效率提升10倍甚至更多。但索引是万能的吗？既然索引可以提升效率，只要创建索引不就好了吗？实际上，在有些情况下，创建索引反而会降低效率。

今天我们就来讲一下索引，索引涉及到的内容比较多，今天先来对索引有个整体的认知。

1. 什么情况下创建索引，什么时候不需要索引？
2. 索引的种类有哪些？

索引的原理很好理解，在今天的内容里，我依然会通过SQL查询实验验证今天的内容，帮你进一步加深理解。

索引是万能的吗？

首先我们需要了解什么是索引（Index）。数据库中的索引，就好比一本书的目录，它可以帮我们快速进行特定值的定位与查找，从而加快数据查询的效率。

索引就是帮助数据库管理系统高效获取数据的数据结构。

如果我们不使用索引，就必须从第1条记录开始扫描，直到把所有的数据表都扫描完，才能找到想要的数据。既然如此，如果我们想要快速查找数据，就只需要创建更多的索引就好了呢？

其实**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。

当然，空口无凭，我们来做两个实验，更直观地了解索引。

实验1：数据行数少的情况下，索引效率如何

我在百度网盘上提供了数据表，heros_without_index.sql 和 heros_with_index.sql，提取码为 wxho。

在第一个数据表中，除了自增的id以外没有建立额外的索引。第二张数据表中，我对name字段建立了唯一索引。

heros数据表一共有69个英雄，数据量很少。当我们对name进行条件查询的时候，我们观察一下创建索引前后的效率。

```
SELECT id, name, hp_max, mp_max FROM heros_without_index WHERE name = '刘禅'
```

运行结果（1条数据，运行时间0.072s）：

id	name	hp_max	mp_max
10015	刘禅	8581	1694

我对name字段建立索引后，再进行查询：

```
SELECT id, name, hp_max, mp_max FROM heros_with_index WHERE name = '刘禅'
```

运行结果（1条数据，运行时间0.080s）：

id	name	hp_max	mp_max
10015	刘禅	8581	1694

你能看到运行结果相同，但是创建了name字段索引的效率比没有创建索引时效率更低。在数据量不大的情况下，索引就发挥不出作用了。

实验2：性别（男或女）字段真的不应该创建索引吗？

如果一个字段的取值少，比如性别这个字段，通常是不需要创建索引的。那么有没有特殊的情况呢？

下面我们来看一个例子，假设有一个女儿国，人口总数为100万人，男性只有10个人，也就是占总人口的10万分之1。

女儿国的人口数据表user_gender见百度网盘中的user_gender.sql。其中数据表中的user_gender字段取值为0或1，0代表女性，1代表男性。

如果我们要筛选出这个国家中的男性，可以使用：

```
SELECT * FROM user_gender WHERE user_gender = 1
```

运行结果（10条数据，运行时间0.696s）：

user_id	user_name	user_gender
110000	student_100000	1
210000	student_200000	1
.....	.....	.....
1010000	Student_1000000	1

你能看到在未创建索引的情况下，运行的效率并不高。如果我们针对user_gender字段创建索引呢？

```
SELECT * FROM user_gender WHERE user_gender = 1
```

同样是10条数据，运行结果相同，时间却缩短到了0.052s，大幅提升了查询的效率。

其实通过这两个实验你也能看出来，索引的价值是帮你快速定位。如果想要定位的数据有很多，那么索引就失去了它的使用价值，比如通常情况下的性别字段。不过有时候，我们还要考虑这个字段中的数值分布的情况，在实验2中，性别字段的数值分布非常特殊，男性的比例非常少。

我们不仅要看字段中的数值个数，还要根据数值的分布情况来考虑是否需要创建索引。

索引的种类有哪些？

虽然使用索引的本质目的是帮我们快速定位想要查找的数据，但实际上，索引有很多种类。

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，这样在查找行的时候非常有效。举个例子，如果是一本汉语字典，我们想要查找“数”这个字，直接在书中找汉语拼音的位置即可，也就是拼音“shu”。这样找到了索引的位置，在它后面就是我们想要找的数据行。

非聚集索引又是什么呢？

在数据库系统会有单独的存储空间存放非聚集索引，这些索引项是按照顺序存储的，但索引项指向的内容是随机存储的。也就是说系统会进行两次查找，第一次先找到索引，第二次找到索引对应的位置取出数据行。非聚集索引不会把索引指向的内容像聚集索引一样直接放到索引的后面，而是维护单独的索引表（只维护索引，不维护索引指向的数据），为数据检索提供方便。我们还以汉语字典为例，如果想要查找“数”字，那么按照部首查找的方式，先找到“数”字的偏旁部首，然后这个目录会告诉我们“数”字存放到第多少页，我们再去指定的页码找这个字。

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：

1. 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。
2. 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。
3. 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。

实验3：使用聚集索引和非聚集索引的查询效率

还是针对刚才的user_gender数据表，我们来看下使用聚集索引和非聚集索引的查询效率有什么区别。在user_gender表中，我设置了user_id为主键，也就是聚集索引的字段是user_id。这里我们查询下user_id=90001的用户信息：

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_id = 900001
```

运行结果（1条数据，运行时间0.043s）：

user_id	user_name	user_gender
900001	Student_890001	0

我们再直接对user_name字段进行条件查询，此时user_name字段没有创建索引：

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_name = 'student_890001'
```

运行结果（1条数据，运行时间0.961s）：

user_id	user_name	user_gender
900001	Student_890001	0

你能看出对没有建立索引的字段进行条件查询，查询效率明显降低了。

然后我们对user_name字段创建普通索引，进行SQL查询：

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_name = 'student_890001'
```

运行结果（1条数据，运行时间0.050s）：

user_id	user_name	user_gender
900001	Student_890001	0

通过对这3次SQL查询结果的对比，我们可以总结出以下两点内容：

1. 对WHERE子句的字段建立索引，可以大幅提升查询效率。
2. 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。

索引列为一列时为单一索引；多个列组合在一起创建的索引叫做联合索引。

创建联合索引时，我们需要注意创建时的顺序问题，因为联合索引(x, y, z)和(z, y, x)在使用的时候效率可能会存在差别。

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。比如刚才举例的(x, y, z)，如果查询条件是WHERE x=1 AND y=2 AND z=3，就可以匹配上联合索引；如果查询条件是 WHERE y=2，就无法匹配上联合索引。

实验4：联合索引的最左原则

还是针对user_gender数据表，我们把user_id和user_name字段设置为联合主键，然后看下SQL查询效率有什么区别。

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_id = 900001 AND user_name =
```

运行结果（1条数据，运行时间0.046s）：

user_id	user_name	user_gender
900001	Student_890001	0

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_id = 900001
```

运行结果（1条数据，运行时间0.046s）：

user_id	user_name	user_gender
900001	Student_890001	0

我们再来看下普通的条件查询是什么样子的：

```
SELECT user_id, user_name, user_gender FROM user_gender WHERE user_name = 'student_890001'
```

运行结果（1条数据，运行时间0.943s）：

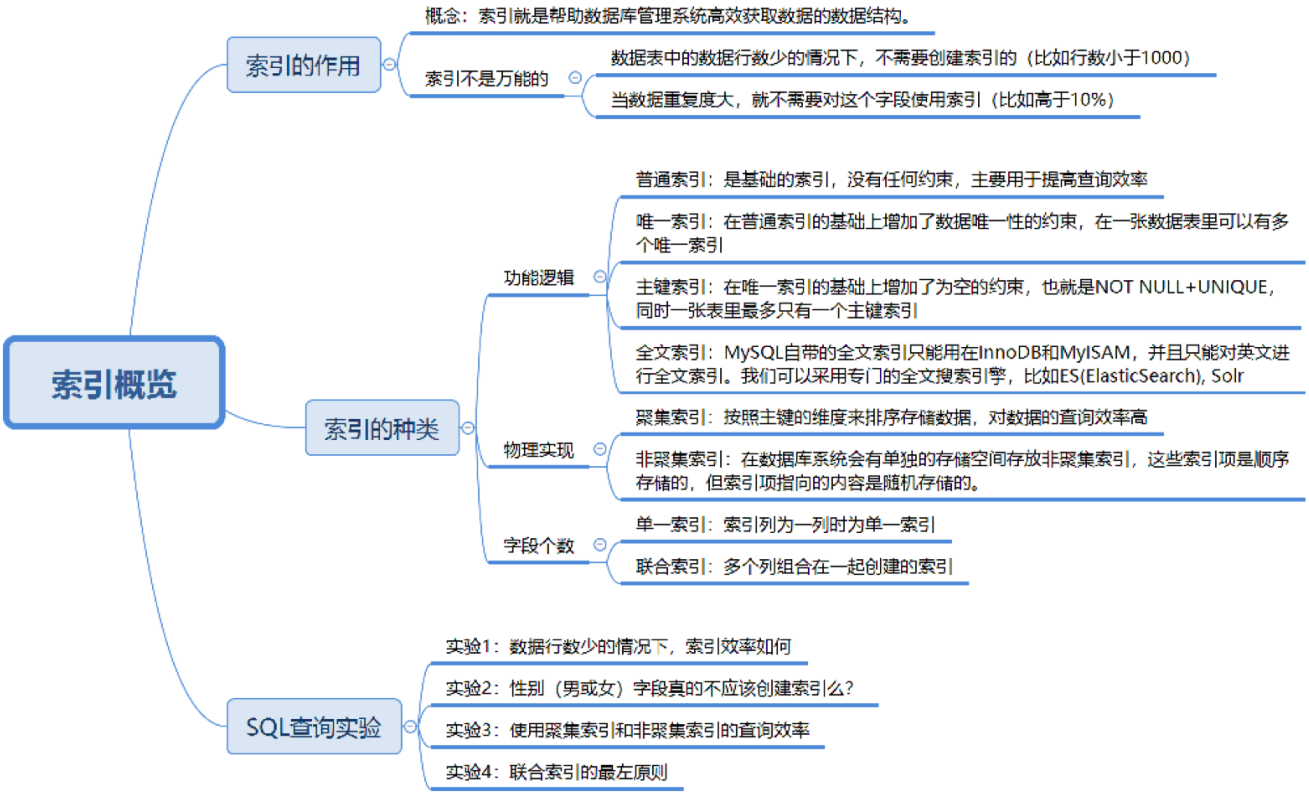
user_id	user_name	user_gender
900001	Student_890001	0

你能看到当我们使用了联合索引(user_id, user_name)的时候，在WHERE子句中对联合索引中的字段user_id和user_name进行条件查询，或者只对user_id进行查询，效率基本上是一样的。当我们对user_name进行条件查询时，效率就会降低很多，这是因为根据联合索引的最左原则，user_id在user_name的左侧，如果没有使用user_id，而是直接使用user_name进行条件查询，联合索引就会失效。

总结

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。



今天的内容到这里就结束了，给你留个问题。关于联合索引的最左原则指的是什么？在使用联合索引时，有哪些需要注意的地方呢？

欢迎你在评论区写下你的答案，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。