Part 8 索引的创建与设计原则

1. 索引的声明与使用

1.1 索引的分类

MySQL的索引包括普通索引、唯一性索引、全文索引、单列索引、多列索引和空间索引等。

- 从功能逻辑上
  - 普通索引
  - 唯一性索引
  - 主键索引
  - 全文索引：支持搜索，但浪费空间，是用磁盘空间换时间的技术。
- 从物理实现方式上
  - 聚簇索引
  - 非聚簇索引
- 从作用字段个数上
  - 单列索引
  - 联合索引
- 补
  - 空间索引
    - 使用`参数SPATIAL`可以设置索引为`空间索引`。
    - 空间索引只能建立在空间数据类型上，这样可以提高系统获取空间数据的效率。
    - MySQL中的空间数据类型包括GEONETRY、POINT、LINESTRING和POLYGON等。
    - 目前只有MyISAM存储引擎支持空间检索，而且索引的字段不能为空值。对于初学者来说，这类索引很少会用到。

小结：不同的存储引擎支持的索引类型也不一样

InnoDB： 支持 B-tree、Full-text 等索引，不支持 Hash索引；

MyISAM： 支持 B-tree、Full-text 等索引，不支持 Hash 索引；

Memory： 支持 B-tree、Hash 等索引，不支持 Full-text 索引；

NDB： 支持 Hash 索引，不支持 B-tree、Full-text 等索引；

Archive： 不支持 B-tree、Hash、Full-text 等索引；

1.2 创建索引

- 在创建表的定义语句`CREATE TABLE`中指定索引列
- 使用`ALTER TABLE`语句在存在的表上创建索引
- 使用`CREATE INDEX`语句在已存在的表上添加索引

1.2.1 创建表的时候创建索引

隐式创建：使用CREATE TABLE创建表时，除了可以定义列的数据类型外，还可以定义主键约束、外键约束或者唯一性约束，而不论创建哪种约束，在定义约束的同时相当于在指定列上创建了一个索引。

显式创建：

```
CREATE TABLE table_name [col_name data_type]
[UNIQUE | FULLTEXT | SPATIAL][INDEX |KEY][index_name] (col_name [length])
[ASC | DESC]
```

- UNIQUE、FULLTEXT和SPATIAL为可选参数，分别表示唯一索引、全文索引和空间索引；
- INDEX与KEY为同义词，两者的作用相同，用来指定创建索引；
- index_name指定索引的名称，为可选参数，如果不指定，那么MySQL默认col_name为索引名；
- col_name为需要创建索引的字段列，该列必须从数据表中定义的多个列中选择；
- length为可选参数，表示索引的长度，只有字符串类型的字段才能指定索引长度；
- ASC或DESC指定升序或者降序的索引值存储。

1. 创建普通索引

在book表中的year_publication字段上建立普通索引，SQL语句如下：

```
#显式的方式创建
#1创建普通的索引
CREATE TABLE book (
    book_id INT ,
    book_name VARCHAR (100) ,
    AUTHORS VARCHAR (100) ,
    info VARCHAR(100) ,
    COMMENT VARCHAR (100) ,
    year_publication YEAR,
    #声明索引
    INDEX idx_bname (book_name));

#通过命令查看索引
#方式1:
mysql> show create table book \G
*************************** 1. row ***************************
       Table: book
Create Table: CREATE TABLE `book` (
  `book_id` int(11) DEFAULT NULL,
  `book_name` varchar(100) DEFAULT NULL,
  `AUTHORS` varchar(100) DEFAULT NULL,
  `info` varchar(100) DEFAULT NULL,
  `COMMENT` varchar(100) DEFAULT NULL,
  `year_publication` year(4) DEFAULT NULL,
  KEY `idx_bname` (`book_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

# 方式2（好用，推荐）：
show index from book;
```

2. 创建唯一索引

```
# 创建唯一索引
CREATE TABLE book (
    book_id INT ,
    book_name VARCHAR (100) ,
    #声明索引
    UNIQUE INDEX uk_idx_bname (book_name));
```

3. 主键索引

创建主键索引：设定为主键后数据库会自动建立索引，innodb为聚簇索引

删除主键索引：

```
ALTER TABLE student drop PRIMARY KEY;
```

修改主键索引：必须先删除掉(drop)原索引，再新建(add)索引

4. 创建组合索引

```
# 创建唯一索引
CREATE TABLE book (
    book_id INT ,
    book_name VARCHAR (100) ,
    author VARCHAR (100) ,
    #声明索引
    INDEX union_key_ba (book_name,author))
;

show index from book;
```

5. 创建全文索引

```
CREATE TABLE fulltext_test (
  id int(11) NOT NULL AUTO_INCREMENT,
    content TEXT NOT NULL,
    tag VARCHAR(255),
    PRIMARY KEY (id),
    FULLTEXT KEY content_tag_fulltext(content, tag) WITH PARSER ngram
) ENGINE = InnoDB DEFAULT CHARSET=utf8mb4;
```

6. 创建空间索引

空间索引可以更高效地对数据类型为“geometry”或“geography”的列（空间数据列）执行特定操作。 空间索引创建中，要求空间类型的字段必须为非空。

```
CREATE TABLE index6(
    id INT,
    SPACE GEOMETRY NOT NULL,
    SPATIAL INDEX index6_sp(SPACE)
) ENGINE=MYISAM;
```

1.2.2 在已经存在的表上创建索引

- **使用ALTER TABLE语句创建索引**的基本语法如下：

```
ALTER TABLE table_name ADD [UNIQUE | FULLTEXT | SPATIAL] [INDEX | KEY]
[index_name] (col_name[length],...) [ASC | DESC]

# 举例
ALTER TABLE book ADD INDEX index_name(book_name);
ALTER TABLE book ADD UNIQUE uk_idx_bname(book_name);
ALTER TABLE book ADD UNIQUE mul_bid_na(book_name,author);
```

- **使用CREATE INDEX创建索引** CREATE INDEX语句可以在已经存在的表上添加索引，在MySQL中，CREATE INDEX被映射到一个ALTER TABLE语句上，基本语法结构为：

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
ON table_name (col_name[length],...) [ASC | DESC]

create 索引类型 索引名称 on 表名(字段);

# 举例
create index idx_cmt on book(comment);
create unique index idx_cmt on book(comment);
create index idx_cmt on book(comment,author);
```

1.3 删除索引

- **使用ALTER TABLE删除索引**的基本语法格式如下：

```
ALTER TABLE table_name DROP INDEX index_name;
```

- **使用DROP INDEX语句删除索引**的基本语法格式如下：

```
DROP INDEX index_name ON table_name;
```

注意：

- 在需要大量删除表数据，修改表数据时，可以考虑先删除索引。等修改完数据之后再插入
- AUTO_INCREMENT 约束字段的唯一索引不能被删除
- 删除表中的列时，如果要删除的列为索引的组成部分，则该列也会从索引中删除。如果组成索引的所有列都被删除，则整个索引将被删除。

2. MySQL 8. 0 索引新特性

2.1 支持降序索引

降序索引以降序存储键值。MySQL在8.0**版本之前创建的仍然是升序索引，使用时进行反向扫描，这大大降低了数据库的效率。**降序索引避免数据库使用额外的文件排序操作，从而提高性能。

> 注意：降序索引只对查询中特定的排序顺序有效，如果使用不当，反而查询效率更低。

2.2 隐藏索引

在MySQL 5.7版本及之前，只能通过显式的方式删除索引。若删除索引后出现错误，只能重新显式创建将索引加回来，这个过程成本很高，消耗资源。

从MySQL 8.x开始支持`隐藏索引（invisible indexes）`，只需要将待删除的索引设置为隐藏索引，使查询优化器不再使用这个索引（即使使用force index（强制使用索引），优化器也不会使用该索引）确认将索引设置为隐藏索引后系统不受任何响应，就可以彻底删除索引。`这种通过先将索引设置为隐藏索引，再删除索引的方式就是软删除`。

希望删除索引时可以先隐藏索引，验证删除索引后对数据库效率的影响。

> 注意：主键不能被设置为隐藏索引。当表中没有显式主键时，表中第一个唯一非空索引会成为隐式主键，也不能设置为隐藏索引。

索引默认是可见的，在使用CREATE TABLE，CREATE INDEX或者ALTERTABLE等语句时可以通过VISIBLE或者INVISIBLE关键词设置索引的可见性。

2.2.1 创建隐藏索引

1. 在MySQL中创建（CREATE TABLE）

隐藏索引通过SQL语句INVISIBLE来实现，其语句比普通索引多了一个关键字INVISIBLE，用来标记索引为不可见索引。其语法形式如下：

```
CREATE TABLE tablename(
    propname1 type1 [ CONSTRAINT1],propname2 type2[ CONSTRAINT2],
    ...
    propnamen typen,
    INDEX [indexname ](propname1 [ ( length)]) INVISIBLE
);
```

2. 在已经存在的表上创建（CREATE INDEX）

可以为已经存在的表设置隐藏索引，其语法形式如下：

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name ON table_name
(col_name[length] [ASC | DESC] ,...) [INVISIBLE|VISIBLE]
```

3. 通过ALTER TABLE语句创建

```
ALTER TABLE book2 ADD index idx_name(book_name) INVISIBLE;
```

2.2.2 切换索引可见状态

已存在的索引可通过如下语句切换可见状态：

```
ALTER TABLE book2 alter index idx_name visible; # 切换成非隐藏索引
ALTER TABLE book2 alter index idx_name invisible; # 切换成隐藏索引
```

如果将index_cname索引切换成可见状态，通过explain查看执行计划，发现优化器选择了idx_name索引。

> 注意：当索引被隐藏时，它的内容仍然是和正常索引一样实时更新的。如果一个索引需要长期被隐藏，那么可以将其删除，因为索引的存在会影响插入、更新和删除的性能。

通过设置隐藏索引的可见性可以查看索引对调优的帮助。

2.2.3 使隐藏索引对查询优化器可见

> 只是有个全局的地方设置可见性，没什么用

在MySQL 8.x版本中，为索引提供了一种新的测试方式，可以通过查询优化器的一个开关（use_invisible_indexes）来打开某个设置，使隐藏索引对查询优化器可见。如果use_invisible_indexes设置为off(默认)，优化器会忽略隐藏索引。如果设置为on，即使隐藏索引不可见，优化器在生成执行计划时仍会考虑使用隐藏索引。

（1）在MySQL命令行执行如下命令查看查询优化器的开关设置。

```
mysql> select @@optimizer_switch \G
```

在输出的结果信息中找到如下属性配置。

```
use_invisible_indexes=off
```

此属性配置值为off，说明隐藏索引默认对查询优化器不可见。

（2）使隐藏索引对查询优化器可见，需要在MySQL命令行执行如下命令：

```
mysql> set session optimizer_switch = "use_invisible_indexes=on" ;
```

SQL语句执行成功，再次查看查询优化器的开关设置。

此时，在输出结果中可以看到如下属性配置。

```
mysql> select @@optimizer_switch \G
```

use_invisible_indexes属性的值为on，说明此时隐藏索引对查询优化器可见。

3. 索引的设计原则

3.1 哪些情况适合创建索引

3.1.1 字段的数值有唯一性的限制

> 业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。（来源：Alibaba）
>
> 说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的。

3.1.2 频繁作为 WHERE 查询条件的字段

某个字段在SELECT语句的 WHERE 条件中经常被使用到，那么就需要给这个字段创建索引了。尤其是在数据量大的情况下，创建普通索引就可以大幅提升数据查询的效率。

3.1.3 经常 GROUP BY 和 ORDER BY 的列

索引就是让数据按照某种顺序进行存储或检索，因此当我们使用 GROUP BY 对数据进行分组查询，或者使用 ORDER BY 对数据进行排序的时候，就需要`对分组或者排序的字段进行索引`。如果待排序的列有多个，那么可以在这些列上建立组合索引。

3.1.4 UPDATE、DELETE 的 WHERE 条件列

对数据按照某个条件进行查询后再进行 UPDATE 或 DELETE 的操作，如果对 WHERE 字段创建了索引，就能大幅提升效率。原理是因为我们需要先根据 WHERE 条件列检索出来这条记录，然后再对它进行更新或删除。 如果进行更新的时候，更新的字段是`非索引字段`，提升的效率会更明显，这是因为非索引字段更新不需要对索引进行维护。

3.1.5 DISTINCT 字段需要创建索引

有时候我们需要对某个字段进行去重，使用 DISTINCT，那么对这个字段创建索引，也会提升查询效率。因为索引会对数据按照某种顺序进行排序，所以在去重的时候也会快很多。 因为紧挨着所以去重特别方便。

3.1.6 多表 JOIN 连接操作时，创建索引注意事项

- 首先，`连接表的数量尽量不要超过 3 张`，因为每增加一张表就相当于增加了一次嵌套的循环，数量级增长会非常快，严重影响查询的效率。
- 其次，对`WHERE 条件创建索引，`因为 WHERE 才是对数据条件的过滤。如果在数据量非常大的情况下，没有 WHERE 条件过滤是非常可怕的。
- 最后，`对用于连接的字段创建索引，`并且该字段在多张表中的`类型必须一致`。比如 course_id 在 student_info 表和 course 表中都为 int(11) 类型，而不能一个为 int 另一个为 varchar 类型。

3.1.7 使用列的类型小的创建索引

我们这里所说的`类型大小`指的就是该类型表示的数据范围的大小。我们在定义表结构的时候要显式的指定列的类型，以整数类型为例，有`TINYINT`、`MEDIUMINT`、`INT`、`BIGINT`等，它们占用的存储空间依次递增，能表示的整数范围当然也是依次递增。

如果我们想要对某个整数列建立索引的话，在表示的整数范围允许的情况下，尽量让索引列使用较小的类型，比如我们能使用`INT`就不要使用`BIGINT`，能使用`MEDIUMINT`就不要使用`INT`。

- 数据类型越小，在查询时进行的比较操作越快
- 数据类型越小，索引占用的存储空间就越少，在一个数据页内就可以放下更多的记录，从而减少磁盘I/O带来的性能损耗，也就意味着可以把更多的数据页缓存在内存中，从而加快读写效率。

这个建议对于表的`主键来说更加适用`，因为不仅是聚簇索引中会存储主键值，其他所有的二级索引的节点处都会存储一份记录的主键值，如果主键使用更小的数据类型，也就意味着节省更多的存储空间和更高效的I/O。

3.1.8 使用字符串前缀创建索引

假设我们的字符串很长，那存储一个字符串就需要占用很大的存储空间。在我们需要为这个字符串列建立索引时，那就意味着在对应的B+树中有这么两个问题:

- B+树索引中的记录需要把该列的完整字符串存储起来，更费时。而且字符串越长，`在索引中占用的存储空间越大。`
- 如果B+树索引中索引列存储的字符串很长，那在做字符串`比较时会占用更多的时间。`

我们可以通过截取字段的前面一部分内容建立索引，这个就叫`前缀索引`。这样在查找记录时虽然不能精确的定位到记录的位置，但是能定位到相应前缀所在的位置，然后根据前缀相同的记录的主键值回表查询完整的字符串值。既`节约空间`，又`减少了字符串的比较时间`，还大体能解决排序的问题。

> 问题是，截取多少呢？截取得多了，达不到节省索引存储空间的目的；截取得少了，重复内容太多，字段的散列度(选择性)会降低。 **怎么计算不同的长度的选择性呢**?
>
> 先看一下字段在全部数据中的选择度：
>
> ```
> 1 select count(distinct address) / count(*) from shop;
> ```
>
> 通过不同长度去计算，与全表的选择性对比：
>
> 公式：
>
> ```
> 1 count(distinct left(列名, 索引长度))/count(*)
> ```
>
> **引申另一个问题：索引列前缀对排序的影响**
>
> 因为二级索引中不包含完整的TEXT列信息，所以无法对选取的前缀相同，后边的字符不同的记录进行排序，也就是使用索引列前缀的方式`无法支持使用索引排序`，只能使用文件排序。

3.1.9 区分度高(散列性高)的列适合作为索引

`列的基数`指的是某一列中不重复数据的个数。**在记录行数一定的情况下，列的基数越大，该列中的值越分散;列的基数越小，该列中的值越集中。**这个列的基数指标非常重要，直接影响我们是否能有效的利用索引。最好为列的基数大的列建立索引，为基数太小列的建立索引效果可能不好。

可以使用公式`select count(distinct a)/count(*) from t1`计算区分度，越接近1越好，一般超过33%就算是比较高效的索引了。

拓展：联合索引把区分度高(散列性高)的列放在前面。

3.1.10 使用最频繁的列放到联合索引的左侧

这样也可以较少的建立一些索引。同时，由于"最左前缀原则"，可以增加联合索引的使用率。

> **最左前缀匹配原则：**在MySQL建立联合索引时会遵守最左前缀匹配原则，即最左优先，在检索数据时从联合索引的最左边开始匹配。

3.1.11 在多个字段都要创建索引的情况下，联合索引优于单值索引

3.2 限制索引的数目

在实际工作中，我们也需要注意平衡，索引的数目不是越多越好。我们需要限制每张表上的索引数量，建议单张表索引数量`不超过6个`。原因:

① 每个索引都需要占用`磁盘空间`，索引越多，需要的磁盘空间就越大。

② 索引会影响`INSERT`、`DELETE`、`UPDATE`等语句的性能，因为表中的数据更改的同时，索引也会进行调整和更新，会造成负担。

③优化器在选择如何优化查询时，会根据统一信息，对每一个可以用到的`索引来进行评估`，以生成出一个最好的执行计划，如果同时有很多个索引都可以用于查询，会增加MySQL优化器生成执行计划时间，降低查询性能。

3.3 哪些情况不适合创建索引

3.3.1 在where中使用不到的字段，不要设置索引

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。

3.3.2 数据量小的表最好不要使用索引

如果表记录太少，比如少于1000个，那么是不需要创建索引的。表记录太少，是否创建索引`对查询效率的影响并不大`。甚至说，查询花费的时间可能比遍历索引的时间还要短，索引可能不会产生优化效果。

3.3.3 有大量重复数据的列上不要建立索引

在条件表达式中经常用到的不同值较多的列上建立索引，但字段中如果有大量重复数据，也不用创建索引。比如在学生表的"`性别`"字段上只有“男”与“·女"两个不同值，因此无须建立索引。如果建立索引，不但不会提高查询效率，反而会`严重降低数据更新速度`。

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。

3.3.4 避免对经常更新的表创建过多的索引

第一层含义: 频繁`更新的字段`不一定要创建索引。因为更新数据的时候，也需要更新索引，如果索引太多，在更新索引的时候也会造成负担，从而影响效率。

第二层含义: 避免对`经常更新的表`创建过多的索引，并且索引中的列尽可能少。此时，虽然提高了查询速度，同时却会降低更新表的速度。

3.3.5 不建议用无序的值作为索引

无序数据可能导致多次中间插入，严重加剧了索引维护的成本。

3.3.6 删除不再使用或者很少使用的索引

表中的数据被大量更新，或者数据的使用方式被改变后，原有的一些索引可能不再需要。数据库管理员应当定期找出这些索引，将它们删除，从而减少索引对更新操作的影响。

3.3.7 不要定义冗余或重复的索引

① 冗余索引

比如不要对联合索引中第一项设立单独索引，这个冗余索引只会增加维护成本，不会对搜索有什么好处。

② 重复索引

比如不要对主键再次定义一个普通索引，因为其本身就有一个聚簇索引，避免重复。

3.5 小结

索引是一把`双刃剑`，可提高查询效率，但也会降低插入和更新的速度并占用磁盘空间。

选择索引的最终目的是为了使查询的速度变快，上面给出的原则是最基本的准则，但不能拘泥于上面的准则，在以后的学习和工作中进行不断的实践，根据应用的实际情况进行分析和判断，选择最合适的索引方式。