

第1章 索引介绍

1.介绍

- 1 索引相当于一本书的目录，可以优化查询。

2.索引查找算法

- 1 1 --> 100 盒子
- 2 谁最快猜到数字，礼品归谁。
- 3 我会给大家提示。
- 4 1. 遍历
- 5 2. 二分法 ---> 二叉树 ---> 红黑树 ---> Balance Ttree(平衡多叉树，简称为BTREE)

3.BTREE查找算法演变

- 1 1.B-TREE : 普通 BTREE
- 2 2.B+TREE : 叶子节点双向指针
- 3 3.B++TREE (B*TREE) : 枝节点的双向指针

B-TREE示意图：

根节点

1,9,49,73

枝节点

1,5

9,13

49,65

73,97

叶子节点

1,2,3,4

5,6,7,8

9,10,11,12

13,14,15,16

49,50,51,52

65,66,67,68

73,74,75,76

97,98,99,100

B+TREE示意图：



≥ 50
AND
 ≤ 73

根节点

1,9,49,73

枝节点

1,5

9,13

49,65

73,97

叶子节点

1,2,3,4

5,6,7,8

9,10,11,12

13,14,15,16

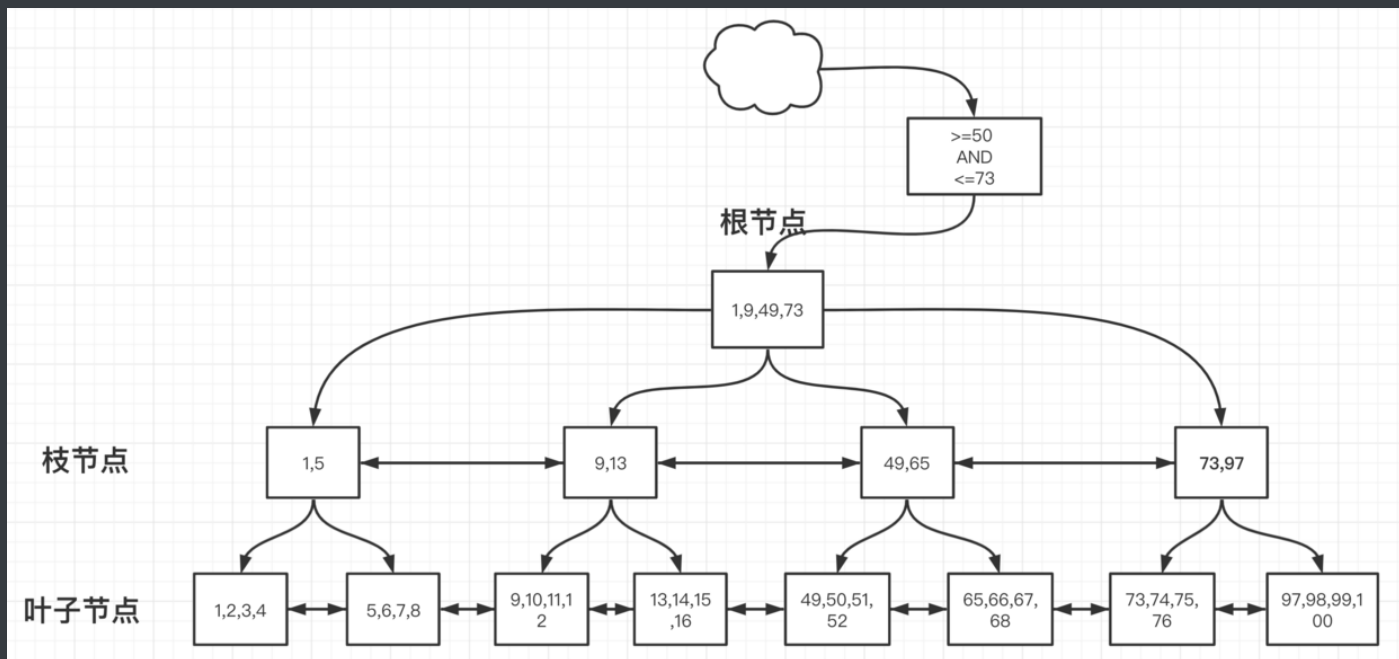
49,50,51,52

65,66,67,68

73,74,75,76

97,98,99,100

B++TREE示意图：



第2章 聚簇(区)索引

1.前提

1. 如果表中设置了主键（例如ID列），自动根据ID列生成索引树。
2. 如果没有设置主键，自动选择第一个唯一键的列作为聚簇索引
3. 自动生成隐藏的聚簇索引。

2.建议

- 1 在建表时，显示的创建主键，最好是数字自增列

3.功能

1. 录入数据时，按照聚簇索引组织存储数据，在磁盘上有序存储数据行。
2. 加速查询。基于ID作为条件的判断查询。

4.构建过程

1. 叶子节点：存储数据行时就是有序的，直接将数据行的page作为叶子节点（相邻的叶子结点，有双向指针）
2. 枝节点：提取叶子节点ID的范围+指针，构建枝节点（相邻枝节点，有双向指针）
3. 根节点：提取枝节点的ID的范围+指针，构建根节点

第3章 辅助索引

1.前提

- 1 需要人为创建辅助索引，将经常作为查询条件的列创建辅助索引，起到加速查询的效果。

2.功能

- 1 按照辅助索引列，作为查询条件时。
- 2 1. 查找辅助索引树，得到ID值
- 3 2. 拿着ID值回表（聚簇索引）查询

3.构建过程

- 1 1. 叶子节点：提取主键（ID）+辅助索引列，按照辅助索引列进行从小到大排序后，生成叶子节点。（相邻的叶子结点，有双向指针）
- 2 2. 枝节点：提取叶子节点辅助索引列的范围+指针，构建枝节点（相邻枝节点，有双向指针）
- 3 3. 根节点：提取枝节点的辅助索引列的范围+指针，构建根节点

第4章 索引考虑事项

1.回表是什么？ 回表会带来什么问题？ 怎么减少回表？

- 1 a. 按照辅助索引列，作为查询条件时，先查找辅助索引树，再到聚簇索引树查找数据行的过程。
- 2 b. IO量多、IO次数多、随机IO会增多
- 3
- 4 减少回表：
 - 5 1. 辅助索引能够完全覆盖查询结果，可以使用联合索引。
 - 6 2. 尽量让查询条件精细化，尽量使用唯一值多的列作为查询条件
 - 7 3. 优化器：MRR (Multi-Range-Read) ， 锦上添花的功能。
- 8 `mysql> select @@optimizer_switch;`
- 9 `mysql> set global optimizer_switch='mrr=on';`
- 10
- 11 功能：
 - 12 1. 辅助索引查找后得到ID值，进行自动排序
 - 13 2. 一次性回表，很有可能受到B+TREE中的双向指针的优化查找。

2.索引树高度的影响因素？ 如何解决？

- 1 a. 高度越低越好
- 2
- 3 b. 数据行越多，高度越高。
 - 4 1. 分区表。一个实例里管理。
 - 5 2. 按照数据特点，进行归档表。
 - 6 3. 分布式架构。针对海量数据、高并发业务主流方案。
 - 7 4. 在设计方面，满足三大范式。
 - 8
- 9 c. 主键规划：长度过长。
 - 10 1. 主键，尽量使用自增数字列。
 - 11
- 12 d. 列值长度越长，数据量大的话，会影响到高度。
 - 13 1. 使用前缀索引
 - 14 100字符 只取前10个字符，构建索引树。

15
16 e. 数据类型的选择。
17 选择合适的、简短的数据类型。
18 例如：
19 1. 存储人的年龄，使用 tinyint 和 char(3)哪个好一些
20 2. 存储人名，char(20)和varchar(20)的选择哪个好。
21 a. 站在数据插入性能角度思考，应该选：char
22 b. 从节省空间角度思考，应该选：varchar
23 c. 从索引树高度的角度思考，应该选：varchar
24 建议使用varchar类型存储变长列值。

第5章 索引应用

1.压测

```
1 source /root/t100w.sql
2 mysqlslap --defaults-file=/etc/my.cnf --concurrency=100 --iterations=1
  --create-schema='test' --query="select * from test.t100w where
  k2='780P'" engine=innodb --number-of-queries=2000 -uroot -p123 -verbose
3
4 --concurrency=100 : 模拟同时100会话连接
5 --create-schema='test' : 操作的库是谁
6 --query="select * from test.t100w where k2='780P'" : 做了什么操作
7 --number-of-queries=2000 : 一共做了多少次查询
8
9 Running for engine rbose
10 Average number of seconds to run all queries: 648.657 seconds
11 Minimum number of seconds to run all queries: 648.657 seconds
12 Maximum number of seconds to run all queries: 648.657 seconds
13 Number of clients running queries: 100
14 Average number of queries per client: 20
```

2.查询表的索引

查看索引：

```
1 desc t100w;  
2 show index from t100w;
```

索引类型：

```
1 -----  
2 Key  
3 -----  
4 PK      --> 主键（聚簇索引）  
5 MUL     --> 辅助索引  
6 UK      --> 唯一索引
```

3.创建索引

3.1 单列辅助索引

查询语句：

```
1 select * from test.t100w where k2='780P'
```

优化方法：

```
1 alter table 表名 add index 索引名(列名);  
2 alter table t100w add index idx_k2(k2);
```

3.2 创建联合索引

```
1 mysql> alter table t100w add index idx_k1_num(k1,num);
```

3.3 前缀索引创建

```
1 select count(distinct(left(name,5))) from city ;
2 select count(distinct name) from city ;
3 创建前缀索引
4 mysql> alter table city add index idx_n(name(5));
```

4.删除索引

```
1 alter table city drop index idx_n;
```

第6章 执行计划获取和分析

1.命令介绍

```
1 explain
2 desc
```

2.使用方法


```

1 mysql> desc select * from city where countrycode='CHN';
2 mysql> explain select * from city where countrycode='CHN';
3 +----+-----+-----+-----+-----+-----+-----+
   | id | select_type | table | partitions | type | possible_keys | key
   | key_len | ref | rows | filtered | Extra |
4 +----+-----+-----+-----+-----+-----+-----+
   | 1 | SIMPLE | city | NULL | ref | CountryCode |
   | CountryCode | 3 | const | 363 | 100.00 | NULL |
5 +----+-----+-----+-----+-----+-----+-----+
   | 1 | SIMPLE | city | NULL | ref | CountryCode |
   | CountryCode | 3 | const | 363 | 100.00 | NULL |
6 +----+-----+-----+-----+-----+-----+-----+
   | 1 | SIMPLE | city | NULL | ref | CountryCode |
   | CountryCode | 3 | const | 363 | 100.00 | NULL |
7 +----+-----+-----+-----+-----+-----+-----+
   | 1 | SIMPLE | city | NULL | ref | CountryCode |
   | CountryCode | 3 | const | 363 | 100.00 | NULL |

```

3.执行计划信息介绍

```

1 table      : 此次查询访问的表
2 type       : 索引查询的类型 (ALL、index、range、ref、eq_ref、
   const(system)、NULL)
3 possible_keys : 可能会应用的索引
4 key        : 最终选择的索引
5 key_len    : 索引覆盖长度，主要是用来判断联合索引应用长度。
6 rows       : 需要扫描的行数
7 Extra      : 额外信息

```

4.type信息详解

4.1 ALL 没有使用到索引

```

1 a. 查询条件没建立索引
2 mysql> desc select * from city where district='shandong';
3 b. 有索引不走
4 mysql> desc select * from city where countrycode!='CHN';
5 mysql> desc select * from city where countrycode not in ('CHN','USA');
6 mysql> desc select * from city where countrycode like '%CH%';

```

4.2 index 全索引扫描

```
1 mysql> desc select countrycode from city;
```

4.3 range 索引范围扫描

```
1 会受到： B+TREE额外优化，叶子节点双向指针
2 mysql> desc select * from city where id<10;
3 mysql> desc select * from city where countrycode like 'CH%';
4
5 以下两种查询，大几率受不到叶子节点双向指针优化。
6 mysql> desc select * from city where countrycode in ('CHN','USA');
7 mysql> desc select * from city where countrycode='CHN' or
  countrycode='USA';
8
9 建议： 如果查询列重复值少的话，我们建议改写为 union all
10 desc
11 select * from city where countrycode='CHN'
12 union all
13 select * from city where countrycode='USA';
```

4.4 ref 辅助索引等值查询

```
1 desc select * from city where countrycode='CHN';
```

4.5 eq_ref：多表连接查询中，非驱动表的连接条件是主键或唯一键时

```
1 mysql> desc select city.name,country.name
2 from city
3 left join country
4 on city.countrycode=country.code where city.population<100;
```

4.6 const(system): 主键或唯一键等值查询

```
1 mysql> select * from city where id=1;
```

4.7 NULL

```
1 mysql> desc select * from city where id=10000000000000000;
```

5.key_len信息详解

5.1 作用

- 1 用来判断联合索引应用的部分。
- 2
- 3 例如：
- 4 `idx(a,b,c)`
- 5 我们希望应用联合索引的部分越多越好

5.2 如何计算

```
1 key_len=a+b+c
2 列的key_len长度，按照每列的最大预留长度来做的计算。
3
4 create table t1 (
5 id int,
6 a int ,
7 b char(10),
8 c varchar(10))
9
10 最大存储预留长度（字节）：
11 -----
12 -----
13
14 数据类型：          占用字节量          有not null          没有Not Null
```

```

13 -----
14 -----
14 数字类型:
15 tinyint      : 1字节          1          1+1
16 int          : 4字节          4          4+1
17 bigint       : 8字节          8          8+1
18 -----
19 -----
19 字符串类型:
20 utf8:
21 char(10)      : 10*3字节 =30      30      30+1
22 varchar(10)   : 10*3+2字节=32     32      32+1
23 -----
24 -----
24 utf8mb4:
25 char(10)      : 10*4字节 =40      40      40+1
26 varchar(10)   : 10*4字节+2 =42    42      42+1
27 -----
28 -----
29 use test;
30 create table test (
31 id int not null primary key auto_increment,
32 a int not null ,                # 4
33 b int ,                        # 5
34 c char(10) not null ,          # 40
35 d varchar(10),                 # 43
36 e varchar(10) not null         # 42
37 )engine=innodb charset=utf8mb4;
38
39 alter table test add index idx(a,b,c,d,e);
40
41 5个列覆盖:
42 4+5+40+43+42=134
43
44 4个列覆盖:

```

45 4+5+40+43=92

46

47 3个列覆盖:

48 4+5+40=49

49

50 2个列覆盖:

51 4+5=9

52

53 应用1个列:

54 4

5.3 测试

```
1 mysql> desc select * from test where a=10 and b=10 and c='a' and d='a'
  and e='a';
2 mysql> desc select * from test where a=10 and b=10 and c='a' and d='a';
3 mysql> desc select * from test where a=10 and b=10 and c='a';
4 mysql> desc select * from test where a=10 and b=10;
```

5.4 联合索引应用细节

条件:

- 1 联合索引应用要满足最左原则
- 2 a. 建立联合索引时, 选择重复值最少的列作为最左列。
- 3 b. 使用联合索引时, 查询条件中, 必须包含最左列, 才有可能应用到联合索引。

联合索引不同覆盖场景:

```
1 mysql> alter table t100w add index idx(num,k1,k2);
2 num : 5
3 k1 : 9
4 k2 : 17
```

a.全部覆盖 (key_len:31)

```
1 mysql> desc select * from t100w where num=913759 and k1='ej' and k2='EFfg';
2 mysql> desc select * from t100w where k1='ej' and k2='EFfg' and num=913759 ;
3 mysql> desc select * from t100w where num=913759 and k1='ej' and k2 in('EFfg','abcd');
4 mysql> desc select * from t100w where num=913759 and k1='ej' and k2 like 'EF%';
```

说明:

```
1 a= and b= and c=
2 b= and c= and a=
```

b.部分覆盖 idx(a,b,c)

```
1 where a = and b =
2 where b = and a =
3 where a =
4 where a = and b> < >= <= in like between and and c=
5
6 例如:
7 mysql> desc select * from t100w where num=913759 and k1>'zz' and k2='EFfg';
8
9 总结:
10 如果联合索引中间出现了<>,between,like都会使得索引匹配截止于此。
11
12 如何优化?
13 (num,k1,k2) ----> (num,k2,k1)
14 mysql> desc select * from t100w where num=913759 and k2='EFfg' and k1>'zz';
```

c. 完全不覆盖 idx(a,b,c)

```
1 where b c
2 where b
3 where c
```

6.extra 额外的信息

```
1 using filesort ---> group by \ order by \distinct \ union all
2
3 mysql> desc select * from city where countrycode='CHN' order by
  population;
4
5 注意: where+order by一定要点联合索引
6
7 优化:
8 mysql> alter table city add index idx_1(CountryCode,population);
9 mysql> show index from city;
10 mysql> desc select * from world.city where countrycode='CHN' order by
    population;
```

7.应用场景

```
1 数据库慢:
2 a. 应急性的慢。
3     show full processlist; ----> 慢语句 ----> explain SQL ----> 优化索引、改
    写语句
4 b. 间歇性慢。
5     slowlog ----> 慢语句 ----> explain SQL ----> 优化索引、改写语句
```

第7章 建立索引的原则

1.说明

- 1 为了使索引的使用效率更高，在创建索引时，必须考虑在哪些字段上创建索引和创建什么类型的索引。

2.降低索引树高度

- 1 （必须的）建表时一定要要有主键，一般是个无关自增列数字列。

3.选择唯一性索引

- 1 1.唯一性索引的值是唯一的，可以更快速的通过该索引来确定某条记录。
- 2 2.例如，学生表中学号是具有唯一性的字段。为该字段建立唯一性索引可以很快的确定某个学生的信息。
- 3 3.如果使用姓名的话，可能存在同名现象，从而降低查询速度。

优化方案：

- 1 1.如果非得使用重复值较多的列作为查询条件(例如：男女)，可以将表逻辑拆分
- 2 2.可以将此列和其他的查询类，做联和索引
- 3 `select count(*) from world.city;`
- 4 `select count(distinct countrycode) from world.city;`
- 5 `select count(distinct countrycode,population) from world.city;`

4.尽量使用前缀来索引

- 1 如果索引字段的值很长，最好使用值的前缀来索引。

5.限制索引的数目

- 1 索引的数目不是越多越好。
- 2 可能会产生的问题：
 - 3 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。
 - 4 2.修改表时，对索引的重构和更新很麻烦。越多的索引，会使更新表变得很浪费时间。
 - 5 3.优化器的负担会很重，有可能会影响到优化器的选择。
- 6 4.percona-toolkit中有个工具，专门分析索引是否有用

6.删除不再使用或很少使用的索引(percona toolkit)

- 1 1.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。
- 2 2.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。

7.建索引原则总结

- 1 1.必须要有主键，如果没有可以做为为主键条件的列，创建无关列
- 2 2.经常做为where条件列 order by group by join on, distinct 的条件(业务:产品功能+用户行为)
- 3 3.最好使用唯一值多的列作为索引，如果索引列重复值较多，可以考虑使用联合索引
- 4 4.列值长度较长的索引列，我们建议使用前缀索引。
- 5 5.降低索引条目，一方面不要创建没用索引，不常使用的索引清理，percona toolkit(xxxxx)
- 6 6.索引维护要避开业务繁忙期，建议用pt-osc

第8章 不走索引的情况

1.没有查询条件或者查询条件没有建立索引

- 1 select * from city;
- 2 select * from city where 1=1;

2.查询结果集是原表中的大部分数据，应该是15-25%以上

```
1 100w num 有索引
2 desc select * from t100w where num>1; ----> 全表
3
4 查询的结果集，超过了总数行数25%，优化器觉得就没有必要走索引了。
5 MySQL的预读功能有关。
6
7 可以通过精确查找范围，达到优化的效果。
8 1000000
9 desc select * from t100w where num>50000 and num<60000;
```

3.索引本身失效，统计信息不真实（过旧）

```
1 索引有自我维护的能力。
2 对于表内容变化比较频繁的情况下，有可能会出现索引失效。
3 一般是删除重建
4
5 现象：
6 有一条select语句平常查询时很快，突然有一天很慢，会是什么原因
7 select? ---->索引失效，统计数据不真实
8 innodb_index_stats
9 innodb_table_stats
10
11 立即更新：
12 mysql> ANALYZE TABLE world.city;
```

4.查询条件使用函数在索引列上或者对索引列进行运算

```
1 错误的例子：select * from test where id-1=9;
2 正确的例子：select * from test where id=10;
3 算术运算
4 函数运算
5 子查询
```

5.隐式转换导致索引失效

```
1  这样会导致索引失效。 错误的例子：
2  mysql> CREATE TABLE `num` (
3      `id` int(11) NOT NULL AUTO_INCREMENT,
4      `name` char(10) NOT NULL,
5      `num` char(10) NOT NULL,
6      PRIMARY KEY (`id`),
7      KEY `inx` (`num`)
8  ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
9
10 mysql> desc num;
11 +-----+-----+-----+-----+-----+-----+
12 | Field | Type      | Null | Key | Default | Extra |
13 +-----+-----+-----+-----+-----+-----+
14 | id    | int(11)   | NO   | PRI | NULL    | auto_increment |
15 | name  | char(10)  | NO   |     | NULL    |                |
16 | num   | char(10)  | NO   | MUL | NULL    |                |
17 +-----+-----+-----+-----+-----+-----+
18
19 mysql> insert into num(name,num)
20 values
21 ('z3','123456'),
22 ('l4','123'),
23 ('w5','321');
24
25 mysql> ALTER TABLE num ADD INDEX inx(num);
26 mysql> SHOW INDEX FROM num;
27
28 mysql> DESC SELECT * FROM num WHERE num='123456';
29 mysql> DESC SELECT * FROM num WHERE num=123456;
```

6. <>,not in 不走索引 (辅助索引)

```
1 EXPLAIN SELECT * FROM teltab WHERE telnum <> '110';
2 EXPLAIN SELECT * FROM teltab WHERE telnum NOT IN ('110','119');
3
4 mysql> select * from tab where telnum <> '1555555';
5 +-----+-----+-----+
6 | id | name | telnum |
7 +-----+-----+-----+
8 | 1 | a | 1333333 |
9 +-----+-----+-----+
10 1 row in set (0.00 sec)
11
12 mysql> explain select * from tab where telnum <> '1555555';
13
14 单独的>,<,in 有可能走,也有可能不走,和结果集有关,尽量结合业务添加limit
15 or或in 可以修改成union all
16 EXPLAIN SELECT * FROM teltab WHERE telnum IN ('110','119');
17
18 改写成:
19 EXPLAIN SELECT * FROM teltab WHERE telnum='110'
20 UNION ALL
21 SELECT * FROM teltab WHERE telnum='119'
```

7.like "%_" 百分号在最前面不走

```
1 EXPLAIN SELECT * FROM teltab WHERE telnum LIKE '31%' 走range索引扫描
2 EXPLAIN SELECT * FROM teltab WHERE telnum LIKE '%110' 不走索引
3 %linux%类的搜索需求,可以使用elasticsearch 或者 mongodb 专门做搜索服务的数据库产品
```