链滴

Mysql 调优 (二) 执行计划 索引入门

作者：longteng95
原文链接：https://ld246.com/article/1615293697082
来源网站：链滴

执行计划

通过explain 看对应sql语句中的问题，是否走索引（查询类型），是否全表查询，预估行数

explain select * from user;

id 执行顺序

表示当前sql执行的子句或操作表的顺序

1.id一样顺序执行。

2.id不同，值越大优先级越高。

3.同和不同，大的先，相同顺序执行。

select_type 查询类型

分辨查询的类型，是普通查询还是联合查询还是子查询

如用了子查询(sub query)，最外层会标记为：primary

```sql
--sample:简单的查询，不包含子查询和union
explain select * from emp;

--primary:查询中若包含任何复杂的子查询，最外层查询则被标记为Primary
explain select staname,ename supname from (select ename staname,mgr from emp) t
join emp on t.mgr=emp.empno ;

--union:若第二个select出现在union之后，则被标记为union
explain select * from emp where deptno = 10 union select * from emp where sal
>2000;

--dependent union:跟union类似，此处的depentent表示union或union all联合而成的结果会受外部表
影响
explain select * from emp e where e.empno  in ( select empno from emp where
deptno = 10 union select empno from emp where sal >2000)

--union result:从union表获取结果的select
explain select * from emp where deptno = 10 union select * from emp where sal
>2000;

--subquery:在select或者where列表中包含子查询
explain select * from emp where sal > (select avg(sal) from emp) ;

--dependent subquery:subquery的子查询要受到外部表查询的影响
explain select * from emp e where e.deptno in (select distinct deptno from dept);

--DERIVED: from子句中出现的子查询，也叫做派生类，
explain select staname,ename supname from (select ename staname,mgr from emp) t
join emp on t.mgr=emp.empno ;

--UNCACHEABLE SUBQUERY：表示使用子查询的结果不能被缓存
 explain select * from emp where empno = (select empno from emp where
deptno=@@sort_buffer_size);

--uncacheable union:表示union的查询结果不能被缓存：sql语句未验证
```

table sql

table 使用的表

对应行正在访问哪一个表，表名或别名，可能是临时表或union合并结果集。

1、如果是具体的表名，则表明从实际的物理表中获取数据，当然也可以是表的别名

2、表名是derivedN的形式，表示使用了id为N的查询产生的衍生表

3、当有union result的时候，表名是union n1,n2等的形式，n1,n2表示参与union的id

type * 访问类型

访问类型，表示当前是以何种方式去访问我们的数据，最容易想的是全表扫描，直接暴力的遍历一张去寻找需要数据。

All：全表扫描

index:全索引扫描

range:范围查询

const:只有一行匹配

访问类型非常多，效率才好到坏依次是：

system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL

```
--all:全表扫描，一般情况下出现这样的sql语句而且数据量比较大的话那么就需要进行优化。
explain select * from emp;

--index：全索引扫描这个比all的效率要好，主要有两种情况，一种是当前的查询时覆盖索引，即我们需要的数据在索引中就可以索取，或者是使用了索引进行排序，这样就避免数据的重排序
explain  select empno from emp;

--range：表示利用索引查询的时候限制了范围，在指定范围内进行查询，这样避免了index的全索引扫描，适用的操作符： =, <>, >, >=, <, <=, IS NULL, BETWEEN, LIKE, or IN()
explain select * from emp where empno between 7000 and 7500;

--index_subquery：利用索引来关联子查询，不再扫描全表
explain select * from emp where emp.job in (select job from t_job);

--unique_subquery:该连接类型类似与index_subquery,使用的是唯一索引
 explain select * from emp e where e.deptno in (select distinct deptno from
dept);

--index_merge：在查询过程中需要多个索引组合使用，没有模拟出来

--ref_or_null：对于某个字段即需要关联条件，也需要null值的情况下，查询优化器会选择这种访问方式
explain select * from emp e where  e.mgr is null or e.mgr=7369;

--ref：使用了非唯一性索引进行数据的查找
 create index idx_3 on emp(deptno);
 explain select * from emp e,dept d where e.deptno =d.deptno;

--eq_ref ：使用唯一性索引进行数据查找
explain select * from emp,emp2 where emp.empno = emp2.empno;

--const：这个表至多有一个匹配行，
explain select * from emp where empno = 7369;

--system：表只有一行记录（等于系统表），这是const类型的特例，平时不会出现
```

possible_keys 可能索引

显示可能应用在这张表的索引，一个或多个，查询设计到的字段上若存在索引，则该索引将被列出，实际查询不一定使用。

key 使用索引

实际使用的索引，如果为null,则没有用索引。查询中若用了覆盖索引，则该索引和查询的select字段叠。

key_len 索引长度

表示索引中使用的字节数，可通过key_len计算查询中使用的索引长度。在不损失精度的情况下长度短越好。

ref 索引列

显示索引的哪一列被使用了，如果可能，是一个常数

rows * 预估行数

根据表的统计信息及索引使用情况，估算出找出所需记录需要读取的行数。

直接反应了sql找了多少数据，越少越好。

extra 额外信息

额外信息

```
--using filesort:说明mysql无法利用索引进行排序，只能利用排序算法进行排序，会消耗额外的位置
explain select * from emp order by sal;

--using temporary:建立临时表来保存中间结果，查询完成之后把临时表删除
explain select ename,count(*) from emp where deptno = 10 group by ename;

--using index:这个表示当前的查询时覆盖索引的，直接从索引中读取数据，而不用访问数据表。如果同时出现using where 表名索引被用来执行索引键值的查找，如果没有，表面索引被用来读取数据，而不是真的查找
explain select deptno,count(*) from emp group by deptno limit 10;

--using where:使用where进行条件过滤
explain select * from t_user where id = 1;

--using join buffer:使用连接缓存，情况没有模拟出来

--impossible where: where语句的结果总是false
explain select * from emp where empno = 7469;
```

通过索引进行优化

索引结构来源

1.二叉查找树

问题 ： 数据倾斜

2.AVL树

原文链接：Mysql 调优 (二) 执行计划 索引入门

问题：插入新数据可能造成旋转，插入删除效率低（最短子树和最长子树，高度不超过1）

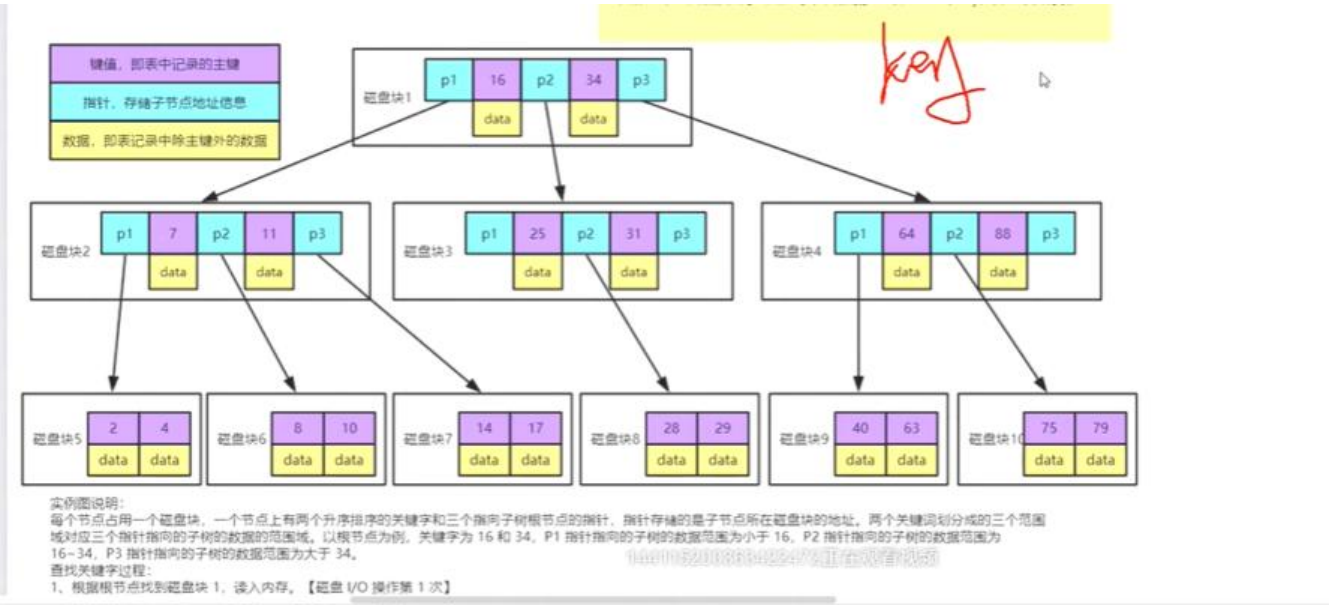
3.红黑树

最长子树不超过最短子树两倍即可，通过旋转和变色，提升插入效率。AVL树变种，损失部分查询性，满足插入性能的提升。

问题：有且**仅有两个分支**，节点过生造成**I/O次数过多。**

4.B树

(M阶，代表一个树节点最多有多少查询路径。M=2二叉树)

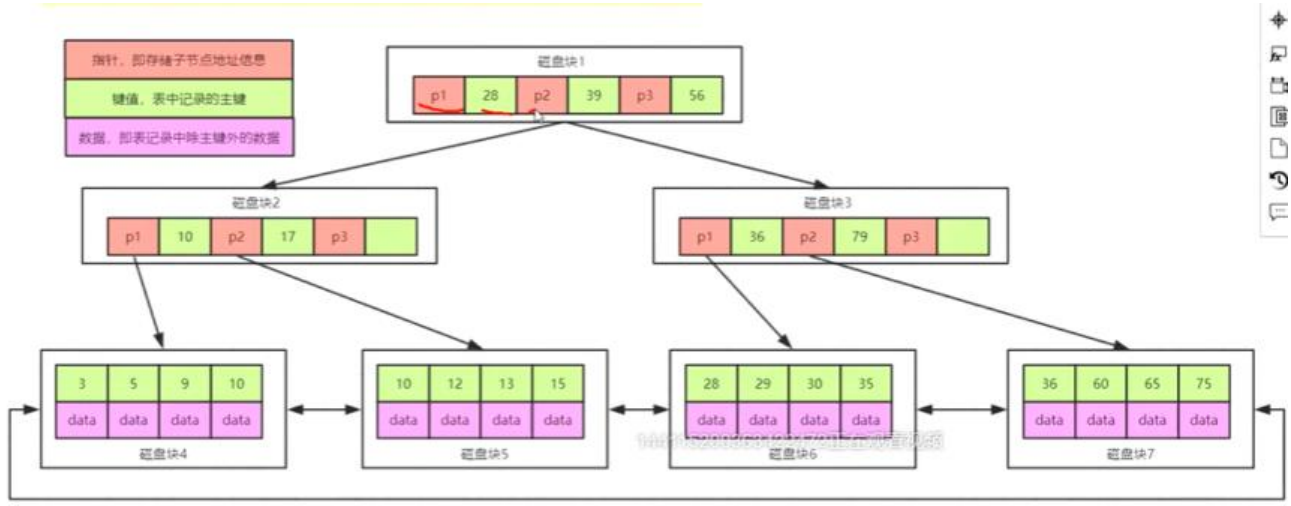


1. 更多的分支（多叉树）
2. 所有节点关键字按递增次序排序，并遵循左小，右大。
3. 子节点数：给叶子节点的子节点数》1，且<=M,M>2

4.所有叶子节点均在同一层，叶子节点包含关键字和关键字记录的指针外，也有指向器子节点指针。

问题：**非叶子节点也含有数据**，占用空间大。

5.B+树

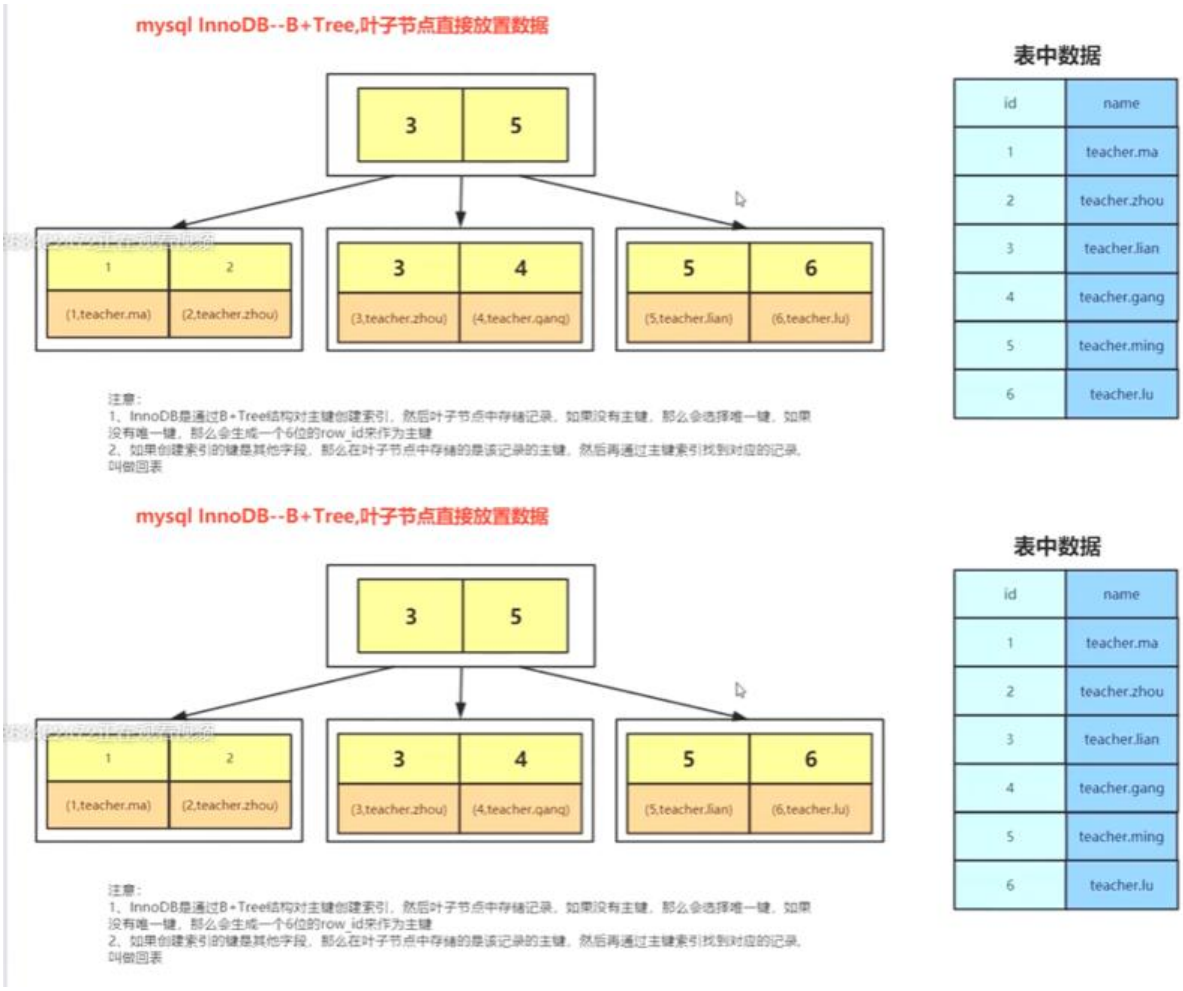


注意：在B+Tree上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构，因此可以对 B+Tree 进行两种查找运算：一种是对于主键的范围查找和分页查找，另一种是从根节点开始，进行随机查找。

1.在B树基础上坐了优化，非叶子节点不存数据信息，可存更多键值。

InnoDB--B+Tree

叶子节点直接放置---整行数据

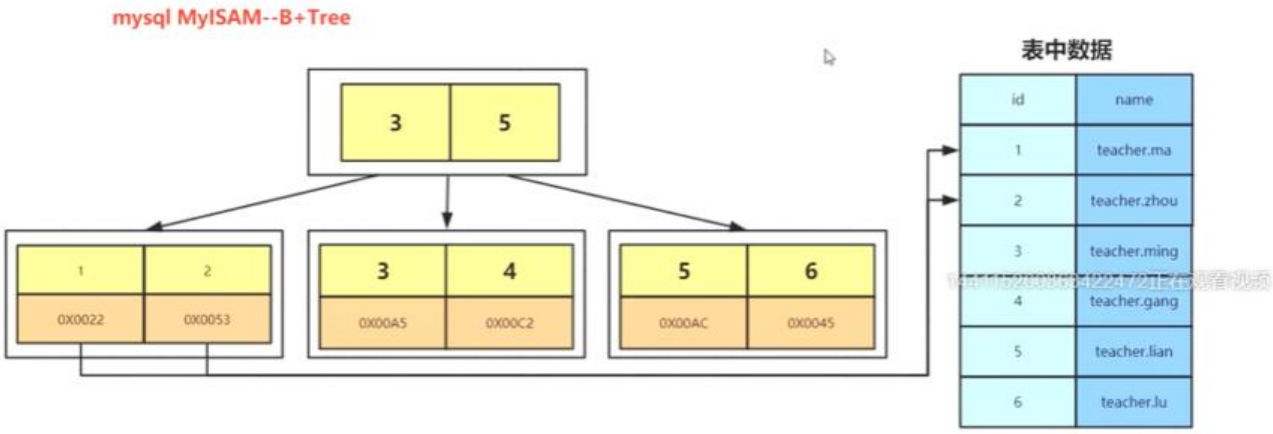


mysql InnoDB--B+Tree,叶子节点直接放置数据
3 5
1 2
(1,teacher.ma) (2,teacher.zhou)
3 4
(3,teacher.zhou) (4,teacher.qang)
5 6
(5,teacher.lian) (6,teacher.lu)
注意：
1、InnoDB是通过B+Tree结构对主键创建索引，然后叶子节点中存储记录，如果没有主键，那么会选择唯一键，如果没有唯一键，那么会生成一个6位的row_id来作为主键
2、如果创建索引的键是其他字段，那么在叶子节点中存储的是该记录的主键，然后再通过主键索引找到对应的记录，叫做回表
表中数据
id name
1 teacher.ma
2 teacher.zhou
3 teacher.lian
4 teacher.gang
5 teacher.ming
6 teacher.lu

MyISAM--B+Tree

原文链接：Mysql 调优 (二) 执行计划 索引入门

叶子节点中，存放实际数据所在文件地址（offset, seek）



索引优点

1.大大减少了服务器需要扫描的数据量

2.帮助服务器避免排序和零时表

3.随机IO变成顺序IO

索引用处

1.快速查找匹配Where子句的行

2.从consideretion中消除行，如果可以多个索引间进行选择，mysql通常会使用找到**最少行的索引。**

3.如果表具有多列索引，则优化器可以使用索引的任何最左前缀来查行。

4.当有表连接的时候，从其他表检索行数据。

5.查找特定索引列的min或max值

6.如果排序或分组时在可用索引的最左前缀上完成，再对表进行排序和分组

7.在某些情况下，可以优化查询以检索值而无需查询数据行。

索引分类

主键索引

数据库建立索引是：唯一且非空

唯一索引

普通索引

全文索引

原文链接：Mysql 调优 (二) 执行计划 索引入门

组合索引

技术名词

回表

使用普通索引进行查询时：

1.在普通索引对应的B+Tree上找，返回该列对应的主键索引。

2.查主键索引对应的B+Tree，找到对应的数据

普通索引存：当前列对应数据及主键

需要两次IO

覆盖索引

用普通索引查询对应列主键

1.1.在普通索引对应的B+Tree上找，直接返回该列的主键。

没有回表过程

最左比配

组合索引，按最左字段匹配

索引下推

组合索引：name,age

查： select xxx from aaa where name='xxx' and age= 'xxx'

老版本时：先将name符合条件的都取出来，在server层过滤age.

高版本用索引下推：在从存储引擎获取数据时，取name值时直接把age值过滤掉，无需在放到server。 减少server层到存储引擎层IO.

谓词下推

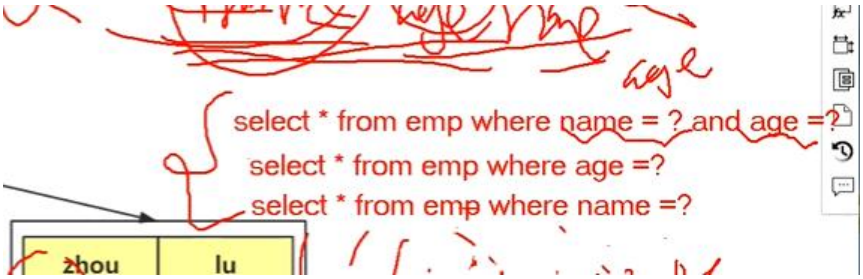
select t1.name ,t2.name from t1 join t2 on t1.id=t2.id

方式一：将t1表与t2表数据按照id关联，如果每张10列，则一共20列。从20列中取出name.

方式二：按表把t1.name,t1.id 及t2.name,t2.id取出来，再按id关联。

方式二效率更高。

案例



索引设计方案

1.建立name,age的组合索引

2.建立age的单独索引（age占用空间比name少）

单独建立索引，效率不一定高，设计**索引合并**，由优化器来完成，效率不一定高。

Mysql查询效率快吗?

快

表现的慢的原因：

1.IO问题

2.并发请求，会有n多个缓存，内存不够，则内存需频繁替换。

查询缓存效率低，命中率低。8已经删除