JOIN原理

Join连接算法

MySQL 8.0 支持两种不同的 JOIN 连接算法，分别为 Nested Loop Join 以及 Hash Join。粗略地来说，Nested Loop Join 在查询数据量较小时使用，通常用于 OLTP；Hash Join 则在数据量较大时使用，通常适用于 OLAP

Nested Loop Join

Index Nested Loop Join

可以使用被驱动表中索引的时候，使用Index Nested Loop Join算法

```
select * from t1 straight_join t2 on (t1.a=t2.a);
```

- 从表t1中读入一行R;
- 从数据行R中，取出字段a到表t2中去查找
- 取出表t2中满足条件的行，跟R组成一行，作为结果集的一部分
- 重复1-3，直到表t1的末尾循环结束

Join语句执行的过程中，驱动表是走全表扫描，被驱动表是走树搜索

时间复杂度

- 假设被驱动表的行数是M。每次被驱动表查一行数据，要先搜索索引a，再搜索主键索引。每次搜索一颗树的相似复杂度是以2为底的M的对数，记为log2M，所以在被驱动表上查一行的时间复杂度是 2*log2M。
- 假设驱动表的行数是 N，执行过程就要扫描驱动表 N 行，然后对于每一行，到被驱动表上匹配一次。
- 所以时间复杂度为：O(N + N*2*log2M)

N 对扫描行数的影响更大，因此应该让小表来做驱动表。

Block Nested Loop Join

没法使用被驱动表中索引的时候，使用Block Nested Loop Join算法

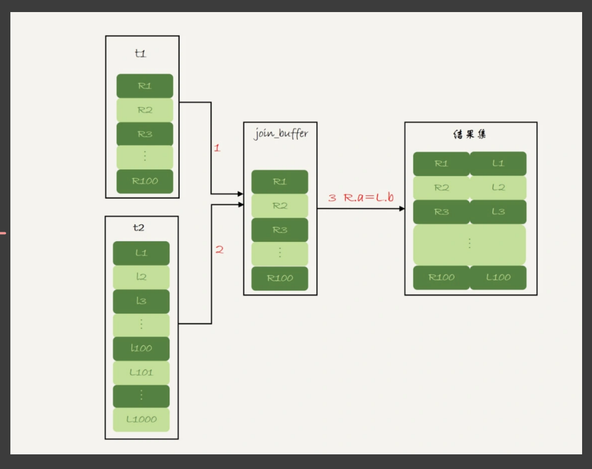
```
select * from t1 straight_join t2 on (t1.a=t2.b);
```

join buffer的大小是由参数 join_buffer_size 设定的，默认值是 256k。
join buffer放不下的时候，就采用分段放的策略

join buffer内存充足的时候

执行流程

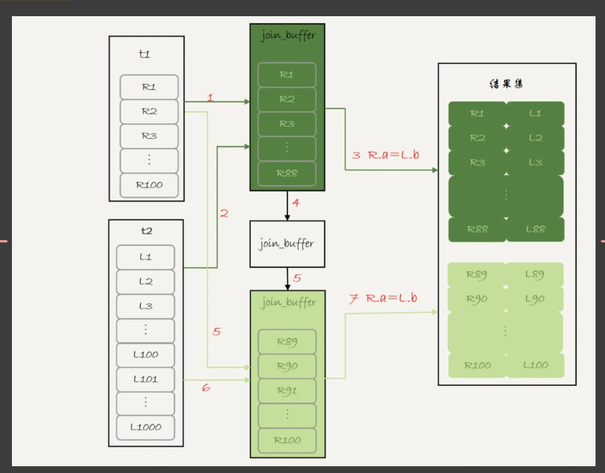
- 把表t1的数据读入线程内存join buffer中，由于这个语句是select *，因此把整个表t1放入内存
- 扫描表t2，把表t2中每一行取出来，跟join buffer中的数据对比，满足join条件的，作为结果集的一部分返回



时间复杂度

- 假设小表的行数是N，大表行数是M。
- 两个表都要做一次全表扫描，所以总扫描的行数是M+N
- 内存中的判断次数数M*N
- 那么近似的时间复杂度为：O(M*N)

join buffer内存不充足的时候



这个流程才体现出了这个算法名字中"Block"的由来，表示"分块去 join"。

假设，驱动表的数据行数是 N，需要分 K 段才能完成算法流程，被驱动表的数据行数是 M。

这里的 K 不是常数，N 越大 K 就会越大，因此把 K 表示为λ*N，显然λ的取值范围是 (0,1)。

时间复杂度

- 假设，驱动表的数据行数是 N，需要分 K 段才能完成算法流程，被驱动表的数据行数是 M。
- 这里的 K 不是常数，N 越大 K 就会越大，因此把 K 表示为λ*N，显然λ的取值范围是 (0,1)。
- 扫描行数是 N+λ*N*M;
- 内存判断 N*M 次。
- 那么近似的时间复杂度为：O(M*N)

内存判断次数是不受选择哪个表作为驱动表影响的。而考虑到扫描行数，在 M 和 N 大小确定的情况下，N 小一些，整个算式的结果会更小。

所以结论是，应该让小表当驱动表。

BNL 算法对系统的影响

- 可能会多次扫描被驱动表，占用磁盘 IO 资源
- 判断 join 条件需要执行 M*N 次对比（M、N 分别是两张表的行数），如果是大表就会占用非常多的 CPU 资源
- 可能会导致 Buffer Pool 的热数据被淘汰，影响内存命中率

NLJ算法的优化

Multi-Range Read

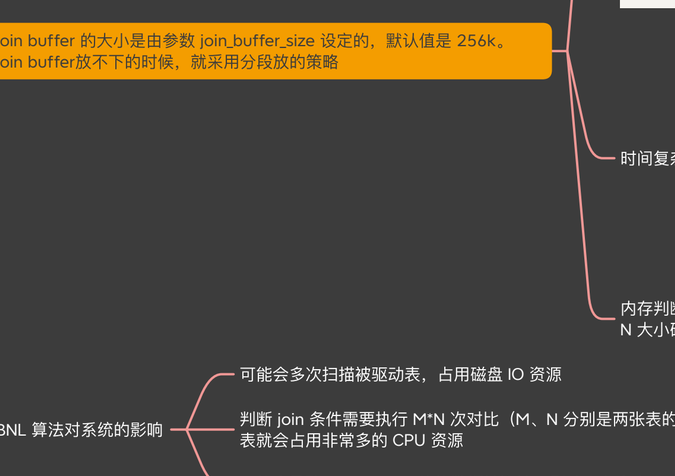
当我们使用二级索引查询完整行记录时，在查询完二级索引的 B+Tree 之后，还需要回到聚簇索引中取出完整行记录，这个过程称之为回表

那么假如二级索引中有多行满足查询要求，是进行一次回表还是多次回表呢？答案是进行多次回表。因为二级索引连续并不代表聚簇索引连续，所以没有办法在一次回表中取出所有的数据

MySQL 5.6 引入了 Multi Range Read 优化，也就是将二级索引查询得到的主键索引进行排序，然后再去聚簇索引中取出完整的行记录，以优化物理 I/O

MRR的核心是将索引的回表变成批量的顺序读

Batched Key Access(BKA)算法



本质上就是批量的从驱动表中取出一批数据，而将其暂存到 join buffer 中，以进行排序，然后再批量去 被驱动表中进行数据查询

NLJ + MRR

Hash Join

在前面的 Nested Loop Join 我们提到了如果被驱动表中没有相关的索引的话，那么其时间复杂度为 O(M*N)，假设驱动表 R 过滤后剩余的数据为 2000 行，被驱动表 S 共有 20000 行，那么一共需扫描 2000 * 20000 = 400 万行，代价非常昂贵

为了解决被驱动表没有索引的情况，MySQL 额外引入了 Hash Join 算法。Hash Join 的原理非常简单，直接将驱动表 R 中满足条件的行记录导入内存并创建一张哈希表，然后遍历被驱动表 S 一次，逐一在哈希表中判断当前记录是否满足条件

Hash Join 和 Nested Loop Join 一样，都会选择小表作为驱动表，并创建 hashmap。如果驱动表超过内存限制的话，那么 MySQL 会将结果转存至硬盘。因此使用 Hash Join 时不管是使用 R 作为驱动表，还是使用 S 作为驱动表，其时间复杂度均为 O(M+N)，那么使用小表作为驱动表将节省更多的内存空间