34 | 到底可不可以使用join？

生产中，关于join语句的使用的问题，一般会集中在以下俩个

1. DBA不让用join，使用join有什么问题
2. 如果有大小两个表做join,应该用哪个表做驱动表

根据join执行流程，回答上面两个问题

创建表的初始化语句，

CREATE TABLE `t2` (

`id` int(11) NOT NULL,

`a` int(11) DEFAULT NULL,

`b` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `a` (`a`)

) ENGINE=InnoDB;

drop procedure idata;

delimiter ;;

create procedure idata()

begin

declare i int;

set i=1;

while(i<=1000)do

insert into t2 values(i, i, i);

set i=i+1;

end while;

end;;

delimiter ;

call idata();

create table t1 like t2;

insert into t1 (select \* from t2 where id<=100)

两个表都有一个主键索引id和一个索引a,字段b上无索引，存储过程idata()往表t2里插入1000行数据，在表t1里插入的100行数据。

**Index Nested-Loop Join**

下面的语句

select \* from t1 straight\_join t2 on (t1.a=t2.a);

如果直接使用join语句，MySQL优化器可能会选表t1或t2作为驱动表，这样会影响分析SQL 语句的执行过程，所以为了便于分析执行过程中的性能问题，用straight\_join让MySQL使用固定的连接方式执行查询，这样优化器只会按照指定的方式去join。在这个语句里，t1是驱动表，t2是被驱动表

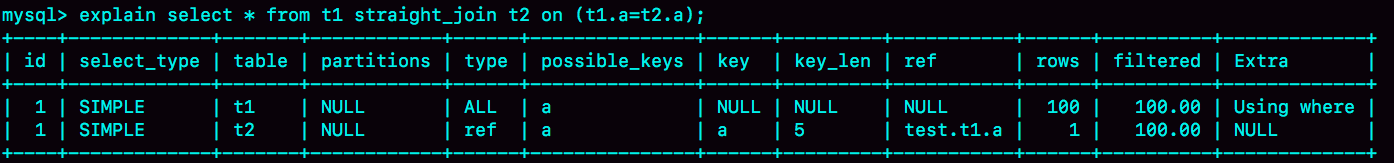


图 1 使用索引字段 join的explain 结果

在这条语句里，被驱动表t2的字段a上有索引，join过程中用上了这个索引，因此语句的执行流程是这样的

1. 从表t1读入一行数据R
2. 从数据行R中，取出字段a到表t2里去查找
3. 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
4. 重复执行步骤1到3,直到表t1的末尾循环结束

这个过程是先遍历表t1，然后根据从表t1中取出的每行数据中的a值，去表t2中查找满足条件的记录，并且用上了被驱动表的索引，所以称之为**“Index Nested-Loop Join”**，简称“**NLJ**”,对应的流程图如下

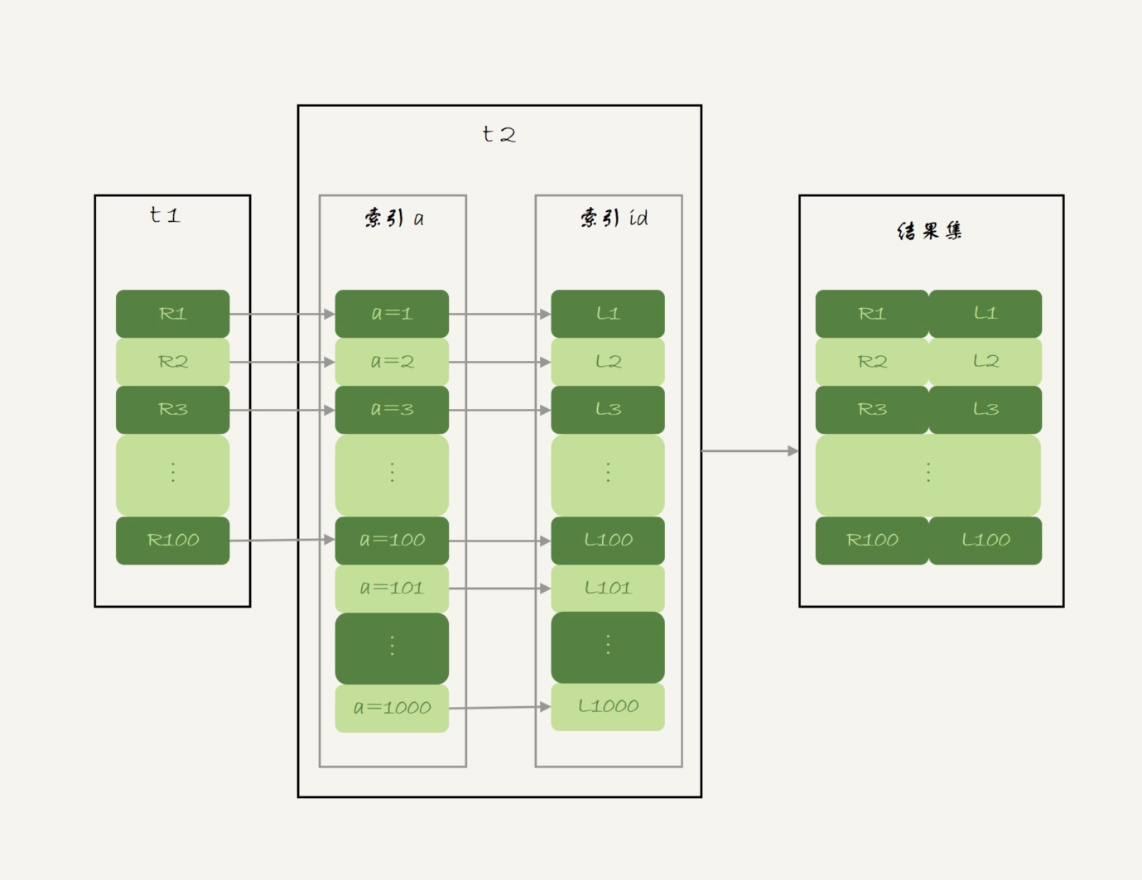


图 2 Index-Nested-Loop Join 算法的流程图

1. 对驱动表t1做了全表扫描，这个过程需要扫描100行
2. 而对于每一行R，根据字段a去表t2查找，走的是树的搜索过程，由于构造数据都是一一对应的，因此每次的搜索过程都只扫描一行，也是总共扫描100行
3. 所以，整个执行流程，总扫描行数200行

先看第一个问题：**能不能使用Join?**

假设不是用join,就只能使用单表查询，用单表查询怎么实现

1. 执行select \* from t1,查出表t1的所有数据，这里有100行
2. 循环遍历这100行数据
   1. 从每一行R取出字段a的值$R.a
   2. 执行select \* from t2 where a=$R.a
   3. 把返回的结果和R构成结果集的一行

在这个查询过程中，也是扫描了200行，但是总共执行了101条语句，比直接join多了100交互，除此之外，客户端还要自己拼接SQL语句和结果

显然，这么做还不如直接join好

再看第二个问题：**怎么选择驱动表？**

在这个join语句执行过程中，驱动表是走全表扫描，而被驱动表是走树搜索

假设被驱动表的行数是M，每次在被驱动表查一行数据，要先搜索索引a,在搜索主键索引。每次搜索一棵树近似复杂度是以2为底的M的对数，即为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，然后对每一行，到被驱动表上匹配一次

因此这个过程，近似复杂度N+N\*2\*log2M

显然，N对扫描行数的影响更大，因此应该让小表来做驱动表

如果没觉的这个影响有那么明显，可以这么理解，N扩大1000倍的话，扫描行数救护扩大1000倍，而M扩大1000倍，扫描行数扩大不到10倍

到这里小结一下，通过上面的分析得出两个结论

1. 使用join语句，性能比拆成多个单表执行SQL语句的性能要好
2. 如果使用join语句的话，需要让小表做驱动表

这个结论前提是：“可以使用被驱动表的索引”

接下来看看，被驱动表用不上索引的情况

语句改成这样

**select \* from t1 straight\_join t2 on (t1.a=t2.b);**

由于表t2字段b上没有索引，因此再用图2的执行流程时，每次到t2去匹配的时候，就要做一次全表扫描

先设想这个问题，继续使用图2的算法，是不是可以得到正确的结果呢？如果只看结果的话，这个算法是正确的，而且这个算法也有一个名字，叫做“Simple Nested-loop Join”.

这样算来，这个SQL请求就要扫描t2表多达100次，总共扫描100\* 1000=10万行

这还只是两个小表，如果t1和t2都是10万行的表（当然了，这也还是属于小表的范围），就要扫描100亿行，这个算法看上去太“笨重”了。

MySQL没有使用这个Simple Nested-Loop Join 算法，而是用一个叫“Block Nested-Loop Join”算法，简称BNL.

**Block Nested-Loop Join**

此时，被驱动表上没有可用的索引，算法的流程是这样的

1. 把表t1的数据读入线程内存join\_buffer中，由于这个语句中写的是select \* ,因此把整个表t1放入了内存
2. 扫描表t2，把表t2中的每一行取出来，跟join\_buffer中的数据对对比，满足join条件的，作为结果集的一部分返回

流程图如下

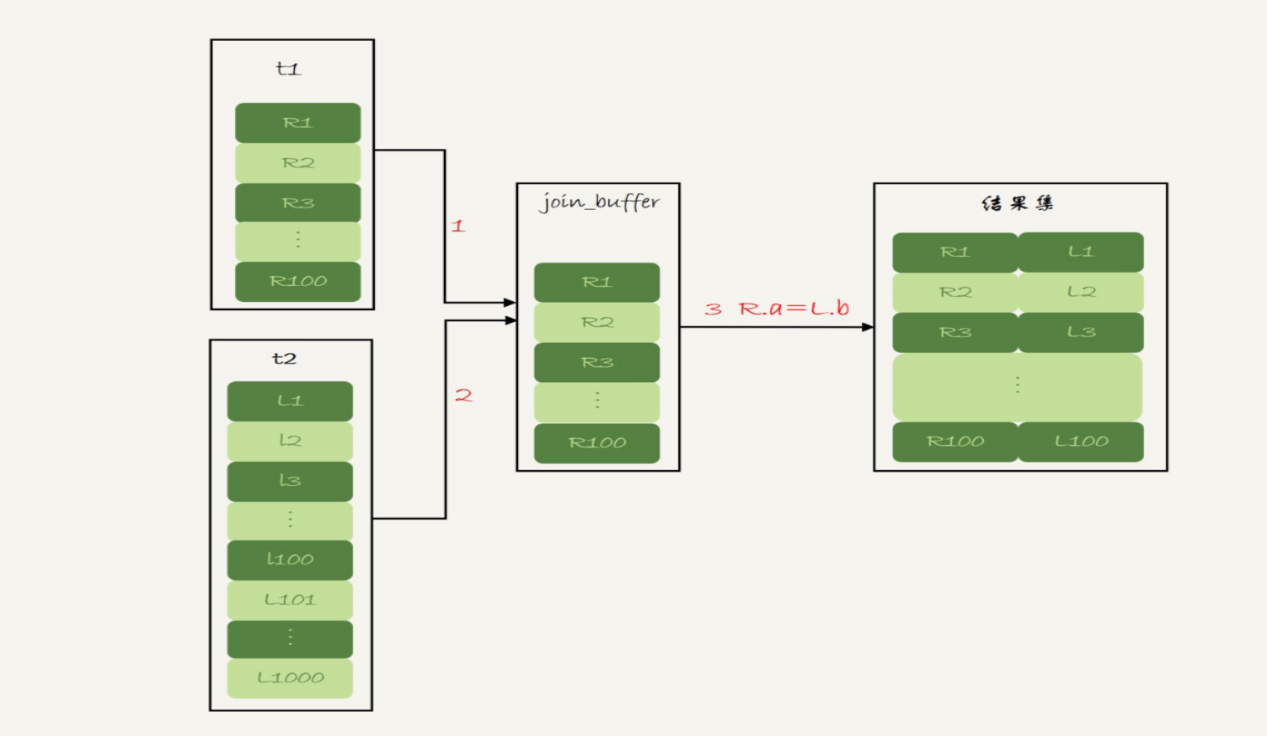


图3 Block Nested-Loop Join算法的执行流程

对应的，这条SQL语句的explain结果如下所示

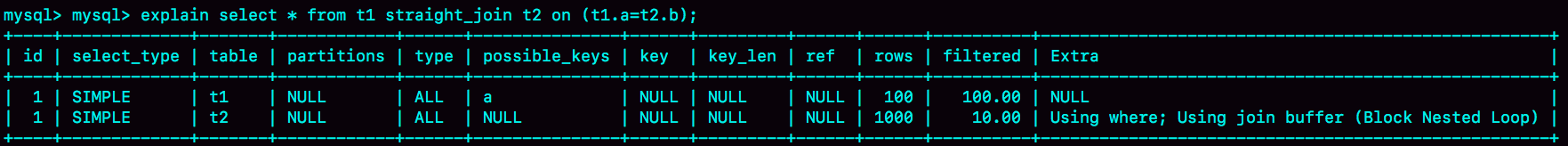


图 4 不使用索引字段join的explain 结果

这个过程中,对表t1和t2都做了一次全表扫描，因此扫描的行数是1100，由于join\_buffer是以无序的数组的方式组织的，因此对表t2中的每一行，都要做100次判断，总共需要在内存中做的判断是:100\*1000=10万次

如果使用Simple Nested-Loop Joi算法进行查询，扫描行数也是10万行，因此，从复杂度上来讲，这两个算法是一样的，但是Block Nested-Loop Join的算法10万次判断是内存操作，速度上快很多，性能也更好。

接下来，在这种情况下，应该选择那个表做驱动表

假设小表的行数是N,大表的行数是M，那么在这个算法里

1. 两个表都做一次全表扫描，所以总的扫描行数是M+N
2. 内存中的判断次数是M\*N

调换这两个算式中的M和N没差别，此时选择大表还是小表做驱动表，执行耗时是一样的。

这个例子里，t1才100行，要是表t1时一个大表，join\_buffer放不下怎么办

Join\_buffer的大小由参数join\_buffer\_size设定，默认是256K,**如果放不下表t1的所有数据话，策略很简单，就是分段放，**把join\_buffer\_size改成1200，再执行

select \* from t1 straight\_join t2 on (t1.a=t2.b);

执行过程就变成了

1. 扫描表t1,顺序读取数据行放入join\_buffer中，放完第88行joiin\_buffer满了，继续第2步
2. 扫描表t2，把t2中的每一行取出来，跟join\_buffer中的数据做对比，满足join条件的，作为结果集的一部分返回
3. 清空join\_buffer
4. 继续扫描表t1,顺序读取最后的12行数据放入join\_buffer中，继续执行步骤2。

执行流程如图

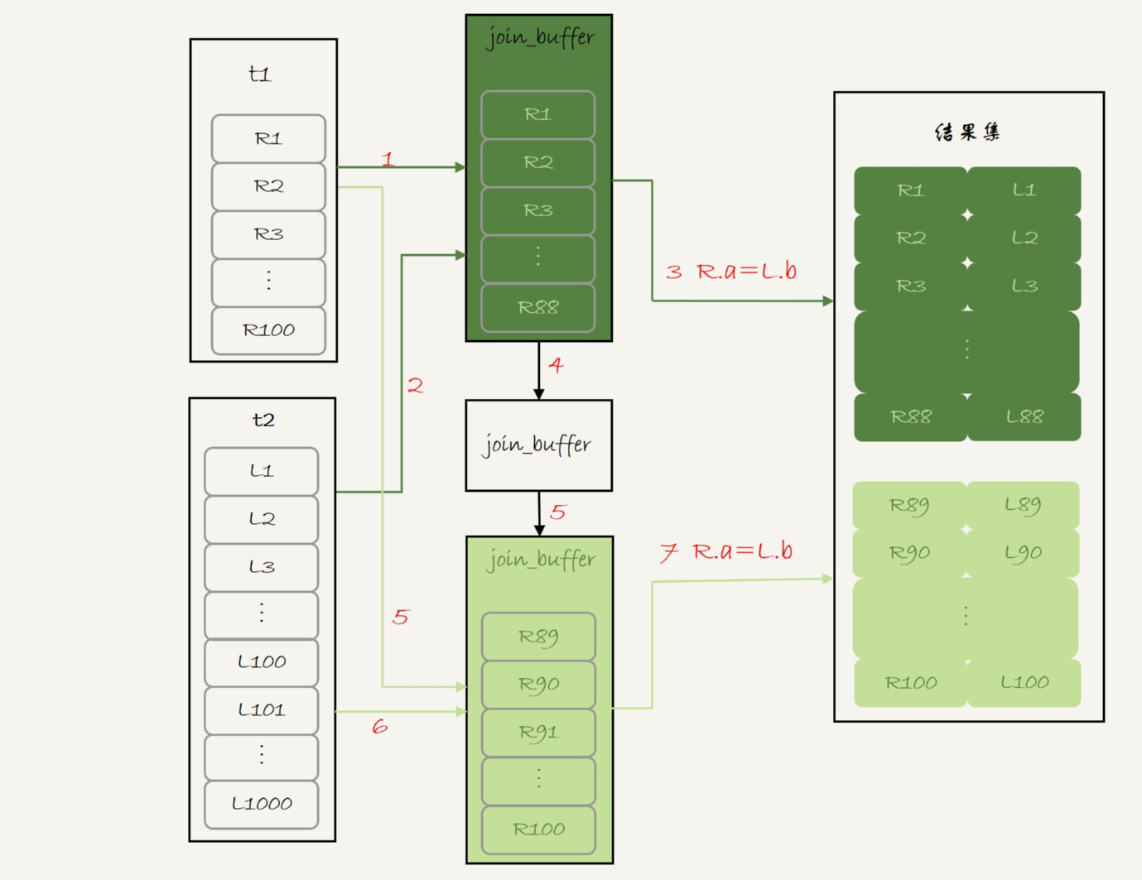


图 5 Block Nested-Loop Join --两端

图中步骤4和5,表示清空join\_buffer再复用

这个流程体现出这个算法名字中“Block”的由来，表示“分块去join”

此时，由于表t1被分成了两次放入join\_buffer中，导致表t2会被扫描两次，虽然分成两次放入join\_buffer，但是判断等值条件的次数还是不变的，依然是（88+12）\*1000=10万次

在这种情况下驱动表的选择问题

假设，驱动表的数据行是N,需要分K段才能完成算法流程，被驱动表的数据行数据是M,这里的K不是常数，N越大K就越大，因此把K表示λ\*N,显然λ的取值范围是（0,1）

所以，在这个算法的执行过程中

1. 扫描行数是N+λ\*N\*M
2. 内存判断N\*M次

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

N越大，分段数K越大，那么，N固定的时候，什么参数会影响K的大小呢？（也就是λ的大小）答案是join\_buffer\_size,join\_buffer\_size越大，一次可放入的行越多，分成的段数也就越少，对驱动表的全表扫描次数就越少。如果join语句很慢，就把join\_buffer\_size改大。

第一个问题：能不能用join语句？

1. 如果可以使用Index Nested-Loop Join算法，也就是可以用上被驱动表的索引，其实是没有问题的
2. 如果使用Block Nested-Loop Join算法，扫描行数就会过多，尤其是在大表上的join操作，这样可能要扫描被驱动表很多次，会占用大量的系统资源，所以这种join尽量不要用

**所以在判断要不要使用join语句时，就看explain结果里面，Extra字段里有没有出现Block Nested Loop字样**

第二个问题：如果使用join语句，应该选择大表做驱动表还是小表做驱动表？

1. 如果是Index Nested-Loop Join算法，应该选择小表做驱动表
2. 如果是Block Nested-Loop Join算法，

·在join\_buffer\_size足够大的时候，是一样的

·在join\_buffer\_size不够大的时候（这种情况更常见），应该选择小表做驱动表

所以，结论是：总是应该选择使用小表做驱动表

**什么叫做“小表”**

如果在where条件上加t2.id<=50这个限定条件，再来看这两条语句

select \* from t1 straight\_join t2 on (t1.b=t2.b) where t2.id<=50;

select \* from t2 straight\_join t1 on (t1.b=t2.b) where t2.id<=50;

为了让两条语句的被驱动表动用不上索引，所以join字段使用了没有索引的字段b

但如果用第二个语句的话，join\_buffer只需放入t2的50行，显然是更好的，所以这里t2的前50行是相对小表，也就是“小表”。

再来看另一组例子

select t1.b,t2.\* from t1 straight\_join t2 on (t1.b=t2.b) where t2.id<=100;

select t1.b,t2.\* from t2 straight\_join t1 on (t1.b=t2.b) where t2.id<=100;

这个例子，表t1和t2都只有100行参加join,但是，这两条语句每次查询放入join\_buffer中的数据是不一样的。

·表t1值查字段b,因此如果把t1放到join\_buffer中，则join\_buffer中只需要放入b的值

·表t2需要查所有的字段，如果把t2放到join\_buffer中，就需要放入三个字段，id,a和b

应该选择t1作为驱动表，也就是这个例子里，“只需要一列join的表t1是那个相对的小表”

**在决定那个表做驱动表的时候，应该是两个表按照各自的条件过滤，过滤完成之后，计算参与join的个字段的总数据量，总数据量小的表，“就是小表”，应该作为驱动表。**

**思考题**

如果被驱动表是一个大表，并且是一个冷数据表，除了查询过程中会导致IO压力大以外，对MySQL服务还有什么更严重的影响吗？