Join 本质【已总结】

CSDN：https://blog.csdn.net/qq_31482599/article/details/108802532

Mysql 官网

> MySQL executes joins between tables using a nested-loop algorithm or variations on it.

MySQL使用嵌套循环算法或它的变体来执行表之间的连接。

A 表 Join B 表，会把A表里面的每一个字段和B表里面的每一个字段做一个匹配

- Nested-Loop Join Algorithm

  > A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

  假设三个表t1、t2和t3之间的连接使用以下连接类型执行:

  ```
  Table   Join Type
  t1      range
  t2      ref
  t3      ALL
  ```

  如果使用简单的NLJ算法，连接处理如下:

  ```
  for each row in t1 matching range {
    for each row in t2 matching reference key {
      for each row in t3 {
        if row satisfies join conditions, send to client
      }
    }
  }
  ```

因为NLJ算法一次只将一行从外部循环传递到内部循环，所以它通常读取在内部循环中处理的表多次。

- Block Nested-Loop Join Algorithm

https://dev.mysql.com/doc/refman/5.7/en/nested-loop-joins.html#block-nested-loop-join-algorithm

A join B 的时候 Mysql 内部会进行一个优化，在读数据的时候，不一定先读A, 再读B

但是我们可以指定让它 先读A 再读B 通过一个关键字

最好 小表 join 大表

这个要求非驱动表（匹配表t2） 上有索引，可以通过索引来减少比较，加速查询

在查询时，驱动表t1 会根据关联字段的索引进行查找，当在索引上找到符合的值，再回表进行查询，也就是只有当匹配到索引以后才会进行回表查询

如果非驱动表 t2 的关联键是主键的话，性能会非常高，如果不是主键，要进行多次回表查询，先关联索引，然后根据耳机索引的主键ID进行回表操作，性能上比索引是主键的要慢

Mysql 的 Nested Loop Join 和 Hash Join

Nested Loop Join

官网说明

注：如下引用内容均摘抄与Mysql官网

> MySQL executes joins between tables using a nested-loop algorithm or variations on it.

MySQL使用嵌套循环算法或它的变体来执行表之间的连接。

Nested-Loop Join Algorithm

Block Nested-Loop Join Algorithm

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

Nested-Loop Join Algorithm

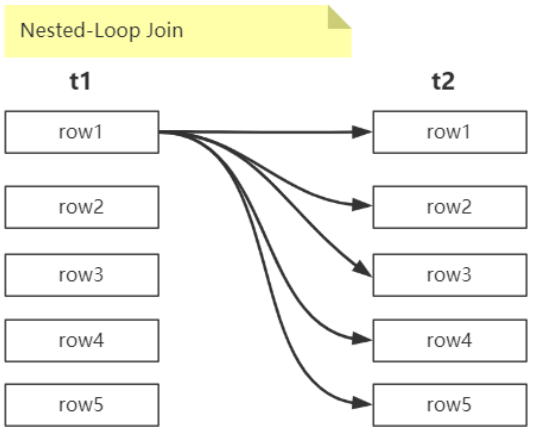
> A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

一个简单的嵌套循环联接(NLJ)算法一次从循环中的第一个表读取一行，将每一行传递到一个嵌套循环，该循环处理联接中的下一个表。只要还有需要连接的表，这个过程就会重复多次。

如果使用简单的NLJ算法，连接处理如下:

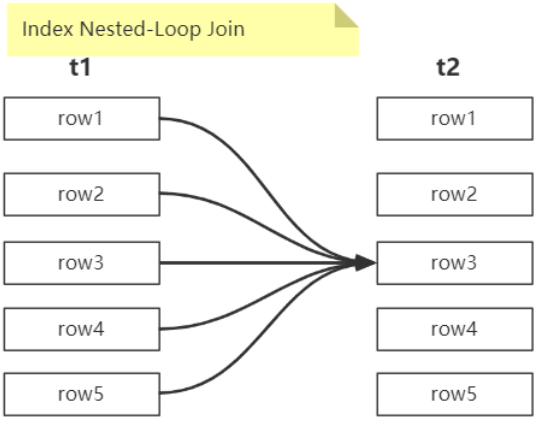
```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

因为NLJ算法一次只将一行从外部循环传递到内部循环，所以它通常读取在内部循环中处理的表多次。



t1 join t2
t1分别取出每一个记录去从t2 中匹配t2中的每一个列，然后再合并数据，这样，如果数据量过大（尤其是t1），对数据库的开销会很大，所以最好是小表join大表

t1 join t2 的时候，mysql会内部进行一个优化(内连接，因为外连接已经指定了驱动表)，在读取数据的时候，不一定是先读取t1，在读取t2。在这种情况下如果要指定先读哪一个表中的数据，可以使用关键字 STRAIGHT_JOIN，这样就强制了左表join右表的顺序
select * frm t1 STRAIGHT_JOIN t2 on t1.id = t2.id



这个要求非驱动表（匹配表t2） 上有索引，可以通过索引来减少比较，加速查询

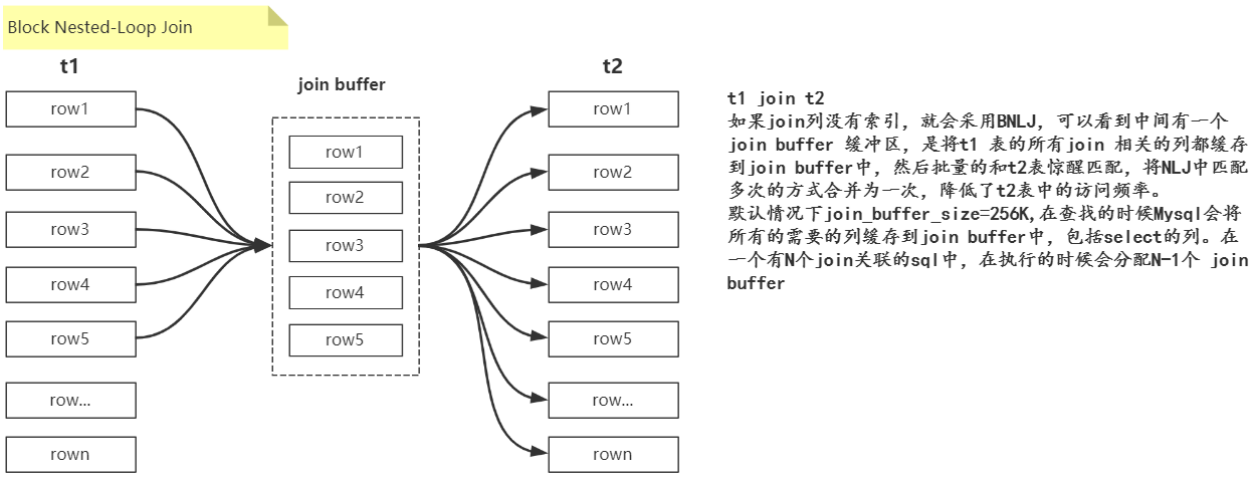
在查询时，驱动表t1 会根据关联字段的索引进行查找，当在索引上找到符合的值，再回表进行查询，也就是只有当匹配到索引以后才会进行回表查询

如果非驱动表 t2 的关联键是主键的话，性能会非常高，如果不是主键，要进行多次回表查询，先关联索引，然后根据耳机索引的主键ID进行回表操作，性能上比索引是主键的要慢

Block Nested-Loop Join Algorithm

> A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. This reduces by an order of magnitude the number of times the inner table must be read.

块嵌套循环(BNL)连接算法使用缓冲在外部循环中读取的行，以减少必须读取内部循环中的表的次数。例如，如果将10行读入一个缓冲区并将该缓冲区传递给下一个内部循环，则可以将内部循环中读取的每一行与缓冲区中的所有10行进行比较。这可以将必须读取内部表的次数减少一个数量级。



如果使用简单的B-NLJ算法，连接处理如下:

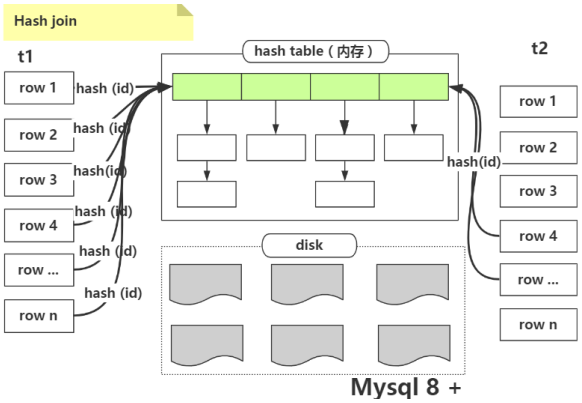
```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions, send to client
        }
      }
      empty join buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
```

分析

process on

Hash Join



散列连接通常比以前MySQL版本中使用的块嵌套循环算法更快，也更适合使用散列连接。从MySQL 8.0.20开始，删除了对块嵌套循环的支持，服务器在以前使用块嵌套循环的地方使用散列连接。
默认情况下，MySQL 8.0.18及以后版本尽可能使用散列连接。可以使用一个BNL和NO_BNL优化器提示来控制是否使用散列连接。
MySQL 8.0.18支持hash_join=on或hash_join=off作为optimizer_switch服务器系统变量以及优化器提示hash_join或NO_HASH_JOIN的设置的一部分。在MySQL 8.0.19及以后版本中，这些不再有任何效果
哈希连接的内存使用可以使用join_buffer_size系统变量来控制;散列连接不能使用超过此数量的内存。当散列连接所需的内存超过可用内存时，MySQL通过使用磁盘上的文件来处理。如果发生这种情况，您应该知道，如果散列连接无法装入内存，并且创建的文件超过open_files_limit设置的文件，则连接可能不会成功。为避免此类问题，请作出以下任何一项更改:
增加join_buffer_size，这样散列连接就不会溢出到磁盘。
增加open_files_limit。
从MySQL 8.0.18开始，散列连接的连接缓冲区是递增分配的;因此，您可以将join_buffer_size设置得更高，而无需使用分配大量RAM的小查询，但是外部连接会分配整个缓冲区。在MySQL 8.0.20及以后版本中，散列连接也用于外连接，因此这不再是一个问题。

select * from t1 join t2 on t1.id = t2.id
hash join 在进行一次查询的时候，会先选择一个表中的join字段进行hash 运算——hash(t1.id)，将该字段全量数据进行hash后，会存放到hash table中，但是如果表数据太多，内存是放不下全量数据的，所以mysql对这种情况做了额外的处理，会将数据写入文件块中。内存中数据匹配完成之后，会将文件块中的数据加载到内存，然后重复以上的过程。

如果join需要的内存超过了join_buffer_size，就会将外表分成若干段，每一段进行hash，然后同内标的hash做比较，但是这样会存在一个问题，假设外表分成了 N 片，这样每一片 hash 完之后，都要同内表中的所有数据进行匹配，这样就会扫描内标N次

所以：在Mysql8.0中，如果join需要的内层超过了限制，首先利用hash算将外表进行分区，并产生临时分片写到磁盘上；然后对于内表使用同样的hash算法进行分区。由于使用分片hash函数相同，那么key相同必然在同一个分片编号中。接下来，再对外表和内表中相同分片编号的数据进行In-Memory Join （CHJ）的过程，所有分片的CHJ做完，整个join过程就结束了。这种算法的代价是，对外表和内表分别进行了两次读IO，一次写IO。相对于之之前需要N次扫描内表IO，现在的处理方式更好。

> Beginning with MySQL 8.0.18, MySQL employs a hash join for any query for which each join has an equi-join condition and uses no indexes

从MySQL 8.0.18开始，MySQL对任何查询都具有相等连接条件且不使用索引的查询使用哈希连接

```
如：SELECT * FROM t1 JOIN t2 ON t1.c1=t2.c1;

EXPLAIN 查询执行计划后可以到Extra中看到  Extra: Using where; Using join buffer (hash join)
```

> A hash join is usually faster than and is intended to be used in such cases instead of the block nested loop algorithm employed in previous versions of MySQL. Beginning with MySQL 8.0.20, support for block nested loop is removed, and the server employs a hash join wherever a block nested loop would have been used previously.

散列连接通常比以前MySQL版本中使用的块嵌套循环算法更快，也更适合使用散列连接。从MySQL 8.0.20开始，删除了对块嵌套循环的支持，服务器在以前使用块嵌套循环的地方使用散列连接。

> By default, MySQL 8.0.18 and later employs hash joins whenever possible. It is possible to control whether hash joins are employed using one of the BNL and NO_BNL optimizer hints. (MySQL 8.0.18 supported hash_join=on or hash_join=off as part of the setting for the optimizer_switch server system variable as well as the optimizer hints HASH_JOIN or NO_HASH_JOIN. In MySQL 8.0.19 and later, these no longer have any effect.)
> Memory usage by hash joins can be controlled using the join_buffer_size system variable; a hash join cannot use more memory than this amount. When the memory required for a hash join exceeds the amount available, MySQL handles this by using files on disk. If this happens, you should be aware that the join may not succeed if a hash join cannot fit into memory and it creates more files than set for open_files_limit. To avoid such problems, make either of the following changes:
> Increase join_buffer_size so that the hash join does not spill over to disk.
> Increase open_files_limit.
> Beginning with MySQL 8.0.18, join buffers for hash joins are

> allocated incrementally; thus, you can set join_buffer_size higher without small queries allocating very large amounts of RAM, but outer joins allocate the entire buffer. In MySQL 8.0.20 and later, hash joins are used for outer joins as well, so this is no longer an issue.

默认情况下，MySQL 8.0.18及以后版本尽可能使用散列连接。可以使用一个BNL和NO_BNL优化器提示来控制是否使用散列连接。

(MySQL 8.0.18支持hash_join=on或hash_join=off作为optimizer_switch服务器系统变量以及优化器提示hash_join或NO_HASH_JOIN的设置的一部分。在MySQL 8.0.19及以后版本中，这些不再有任何效果。)

哈希连接的内存使用可以使用join_buffer_size系统变量来控制;散列连接不能使用超过此数量的内存。当散列连接所需的内存超过可用内存时，MySQL通过使用磁盘上的文件来处理。如果发生这种情况，您应该知道，如果散列连接无法装入内存，并且创建的文件超过open_files_limit设置的文件，则连接可能不会成功。为避免此类问题，请作出以下任何一项更改:

增加join_buffer_size，这样散列连接就不会溢出到磁盘。

增加open_files_limit。

从MySQL 8.0.18开始，散列连接的连接缓冲区是递增分配的;因此，您可以将join_buffer_size设置得更高，而无需使用分配大量RAM的小查询，但是外部连接会分配整个缓冲区。在MySQL 8.0.20及以后版本中，散列连接也用于外连接，因此这不再是一个问题。

分析

https://zhuanlan.zhihu.com/p/94065716

select * from t1 join t2 on t1.id = t2.id

hash join 在进行一次查询的时候，会先选择一个表中的join字段进行hash 运算——hash(t1.id)，将该字段全量数据进行hash后，会存放到hash table中，但是如果表数据太多，内存是放不下全量数据的，所以mysql对这种情况做了额外的处理，会将数据写入文件块中。

参考：https://www.cnblogs.com/cchust/p/11961851.html