数据库表连接主要有三种方式：hash join ,sort merge-join 和 nested loop

Mysql使用的是Nested-Loop Join。

Nested-Loop Join：就是通过驱动表的结果集作为循环基础数据，然后一条一条的通过该结果集中的数据作为过滤条件到下一个表中查询数据，然后合并结果。如果有多个join，则将前面的结果集作为循环数据，再一次作为循环条件到后一个表中查询数据。

没有索引的复杂度是O(N\*M)，这样的复杂度对于大数据集是非常劣势的，一般来讲会通过索引来提升性能。

**外层查询（驱动表）匹配行数决定内层查询返回数**

For each tuple r in R do

For each tuple s in S do

If r and s satisfy the join condition

Then output the tuple <r,s>

MySQL会自动选择结果集最小的表作为驱动表

STRAIGHT\_JOIN与join的区别是，驱动表不由mysql优化器选择，直接由左表做数据驱动表。

驱动表是对于满足查询条件的扫描然后与其他表关联，使用straight\_join的场景之一就是：

Sql1:

SELECT post.\* FROM post STRAIGHT\_JOIN post\_tag ON post.id = post\_tag.post\_id WHERE post.status = 1 AND post\_tag.tag\_id = 123 ORDER BY post.created DESC LIMIT 100;

| post | status\_created | 119340 | Using where |

| post\_tag | post\_id | 1 | Using where |

Sql2:

SELECT post.\* FROM post INNER JOIN post\_tag ON post.id = post\_tag.post\_id WHERE post.status = 1 AND post\_tag.tag\_id = 123 ORDER BY post.created DESC LIMIT 100;

| post\_tag | tag\_id | 71220 | Using where; Using filesort |

| post | PRIMARY | 1 | Using where |

**Order by的字段若不在驱动表只能使用filesort**

**join语句的优化**

1. 用小结果集驱动大结果集，尽量减少join语句中的Nested Loop的循环总次数；

2. 优先优化Nested Loop的内层循环，因为内层循环是循环中执行次数最多的，每次循环提升很小的性能都能在整个循环中提升很大的性能；

3. 对被驱动表的join字段上建立索引；

4. 当被驱动表的join字段上无法建立索引的时候，设置足够的Join Buffer Size。

Join 又分为 inner（default） right outer、left outer和STRAIGHT\_JOIN

Mysql的JOIN, CROSS JOIN, and INNER JOIN,nature join为相同含义，不会出现笛卡尔积(N\*M)的结果返回

simple Nested loop algotithm:

A simple nested-loop join (NLJ) algorithm reads rows from the **first table（驱动表）** 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.

有三表的join t1, t2, and t3按照下面的格式查询:

Table Join Type

t1 range

t2 ref

t3 ALL

If a simple NLJ algorithm is used, the join is processed like this:

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

}

}

}

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.

外层循环的结果加了一层缓存，减少内表被读的次数。

例：外层的10行记录被读入缓存，内层的每个循环都会与缓存的10行进行比较，减少内层循环的次数。

1.Join buffering can be used when the join is of type [ALL](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#jointype_all) or [index](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#jointype_index) (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or [range](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#jointype_range).（外层循环的all查询、range查询和index查询可以缓存）

2．A join buffer is never allocated for the first nonconstant table, even if it would be of type [ALL](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#jointype_all) or [index](https://dev.mysql.com/doc/refman/5.7/en/explain-output.html#jointype_index).（查询条件为“唯一键==constant的表”）**？**

3.Only columns of interest to a join are stored in its join buffer, not whole rows.（只缓存join需要的列）

4. One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.（一个query多个buffer）

上面那个查询加了buffer后：

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

}

}

}