第11章、连接查询的原理

标签： MySQL 是怎样运行的

搞数据库一个避不开的概念就是`Join`，翻译成中文就是`连接`。相信很多小伙伴在初学连接的时候有些一脸懵逼，理解了连接的语义之后又可能不明白各个表中的记录到底是怎么连起来的，以至于在使用的时候常常陷入下边两种误区：

- 误区一：业务至上，管他三七二十一，再复杂的查询也用在一个连接语句中搞定。
- 误区二：敬而远之，上次 DBA 那给报过来的慢查询就是因为使用了连接导致的，以后再也不敢用了。

所以本章就来扒一扒连接的原理。考虑到一部分小伙伴可能忘了连接是个啥或者压根儿就不知道，为了节省他们百度或者看其他书的宝贵时间以及为了我的书凑字数，我们先来介绍一下 MySQL 中支持的一些连接语法。

连接简介

连接的本质

为了故事的顺利发展，我们先建立两个简单的表并给它们填充一点数据：

```
mysql> CREATE TABLE t1 (m1 int, n1 char(1));
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE t2 (m2 int, n2 char(1));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO t1 VALUES(1, 'a'), (2, 'b'), (3, 'c');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO t2 VALUES(2, 'b'), (3, 'c'), (4, 'd');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

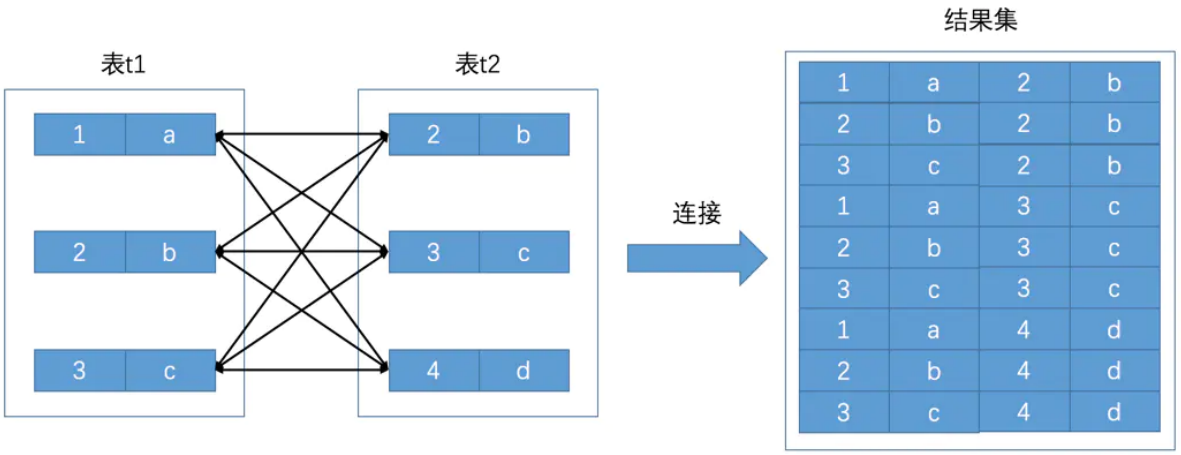
我们成功建立了`t1`、`t2`两个表，这两个表都有两个列，一个是`INT`类型的，一个是`CHAR(1)`类型的，填充好数据的两个表长这样：

```
mysql> SELECT * FROM t1;
+------+------+
| m1   | n1   |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | c    |
+------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+------+
```

```
| m2   | n2   |
+------+------+
|    2 | b    |
|    3 | c    |
|    4 | d    |
+------+------+
3 rows in set (0.00 sec)
```

连接的本质就是把各个连接表中的记录都取出来依次匹配的组合加入结果集并返回给用户。所以我们把t1和t2两个表连接起来的过程如下图所示：



这个过程看起来就是把t1表的记录和t2的记录连起来组成新的更大的记录，所以这个查询过程称之为连接查询。连接查询的结果集中包含一个表中的每一条记录与另一个表中的每一条记录相互匹配的组合，像这样的结果集就可以称之为笛卡尔积。因为表t1中有3条记录，表t2中也有3条记录，所以这两个表连接之后的笛卡尔积就有3×3=9行记录。在MySQL中，连接查询的语法也很随意，只要在FROM语句后边跟多个表名就好了，比如我们把t1表和t2表连接起来的查询语句可以写成这样：

```
mysql> SELECT * FROM t1, t2;
+------+------+------+------+
| m1   | n1   | m2   | n2   |
+------+------+------+------+
|    1 | a    |    2 | b    |
|    2 | b    |    2 | b    |
|    3 | c    |    2 | b    |
|    1 | a    |    3 | c    |
|    2 | b    |    3 | c    |
|    3 | c    |    3 | c    |
|    1 | a    |    4 | d    |
|    2 | b    |    4 | d    |
|    3 | c    |    4 | d    |
+------+------+------+------+
9 rows in set (0.00 sec)
```

连接过程简介

如果我们乐意，我们可以连接任意数量张表，但是如果没有任何限制条件的话，这些表连接起来产生的笛卡尔积可能是

非常巨大的。比方说3个100行记录的表连接起来产生的笛卡尔积就有100×100×100=1000000行数据！所以在连接的时候过滤掉特定记录组合是有必要的，在连接查询中的过滤条件可以分成两种：

- 涉及单表的条件

  这种只涉及单表的过滤条件我们之前都提到过一万遍了，我们之前也一直称为搜索条件，比如t1.m1 > 1是只针对t1表的过滤条件，t2.n2 < 'd'是只针对t2表的过滤条件。

- 涉及两表的条件

  这种过滤条件我们之前没见过，比如t1.m1 = t2.m2、t1.n1 > t2.n2等，这些条件中涉及到了两个表，我们稍后会仔细分析这种过滤条件是如何使用的哈。

下边我们就要看一下携带过滤条件的连接查询的大致执行过程了，比方说下边这个查询语句：

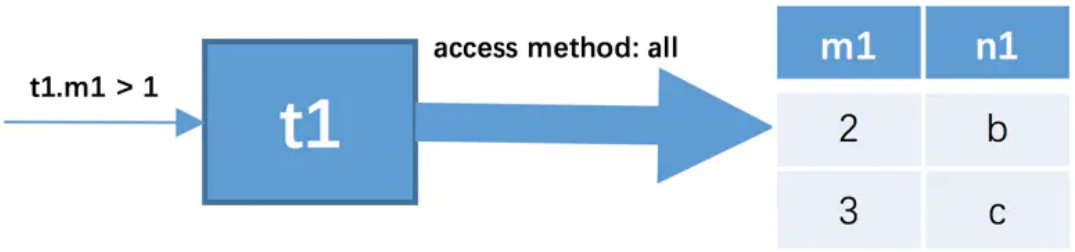
```
SELECT * FROM t1, t2 WHERE t1.m1 > 1 AND t1.m1 = t2.m2 AND t2.n2 < 'd';
```

在这个查询中我们指明了这三个过滤条件：

- t1.m1 > 1
- t1.m1 = t2.m2
- t2.n2 < 'd'

那么这个连接查询的大致执行过程如下：

1. 首先确定第一个需要查询的表，这个表称之为驱动表。怎样在单表中执行查询语句我们在前一章都唠叨过了，只需要选取代价最小的那种访问方法去执行单表查询语句就好了（就是说从const、ref、ref_or_null、range、index、all这些执行方法中选取代价最小的去执行查询）。此处假设使用t1作为驱动表，那么就需要到t1表中找满足t1.m1 > 1的记录，因为表中的数据太少，我们也没在表上建立二级索引，所以此处查询t1表的访问方法就设定为all吧，也就是采用全表扫描的方式执行单表查询。关于如何提升连接查询的性能我们之后再说，现在先把基本概念捋清楚哈。所以查询过程就如下图所示：

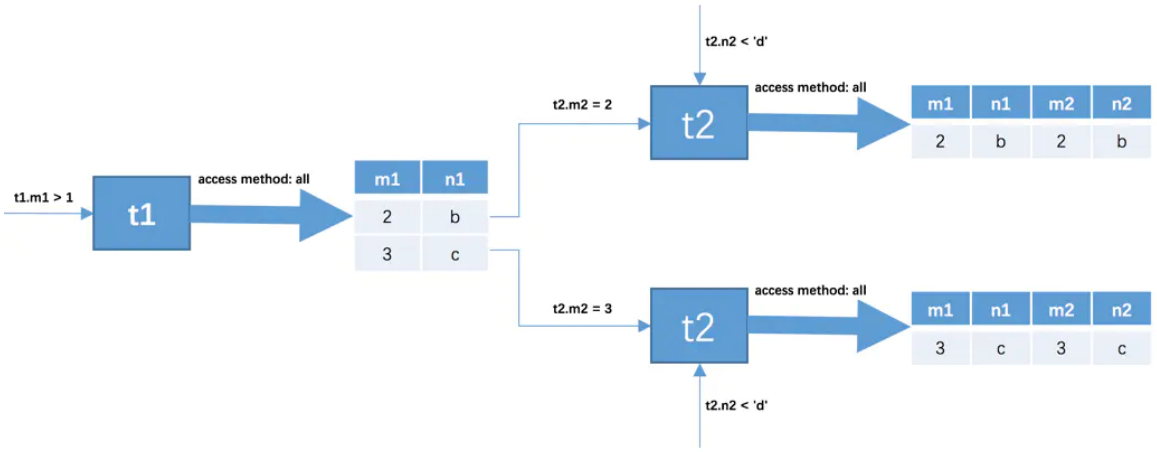


   我们可以看到，t1表中符合t1.m1 > 1的记录有两条。

2. 针对上一步骤中从驱动表产生的结果集中的每一条记录，分别需要到t2表中查找匹配的记录，所谓匹配的记录，指的是符合过滤条件的记录。因为是根据t1表中的记录去找t2表中的记录，所以t2表也可以被称之为被驱动表。上一步骤从驱动表中得到了2条记录，所以需要查询2次t2表。此时涉及两个表的列的过滤条件t1.m1 = t2.m2就派上用场了：

   - 当t1.m1 = 2时，过滤条件t1.m1 = t2.m2就相当于t2.m2 = 2，所以此时t2表相当于有了t2.m2 = 2、t2.n2 < 'd'这两个过滤条件，然后到t2表中执行单表查询。
   - 当t1.m1 = 3时，过滤条件t1.m1 = t2.m2就相当于t2.m2 = 3，所以此时t2表相当于有了t2.m2 = 3、t2.n2 < 'd'这两个过滤条件，然后到t2表中执行单表查询。

   所以整个连接查询的执行过程就如下图所示：



也就是说整个连接查询最后的结果只有两条符合过滤条件的记录：

```
+------+------+------+------+
| m1   | n1   | m2   | n2   |
+------+------+------+------+
|    2 | b    |    2 | b    |
|    3 | c    |    3 | c    |
+------+------+------+------+
```

从上边两个步骤可以看出来，我们上边唠叨的这个两表连接查询共需要查询1次`t1`表，2次`t2`表。当然这是在特定的过滤条件下的结果，如果我们把`t1.m1 > 1`这个条件去掉，那么从`t1`表中查出的记录就有3条，就需要查询3次`t2`表了。也就是说在两表连接查询中，驱动表只需要访问一次，被驱动表可能被访问多次。

内连接和外连接

为了大家更好理解后边内容，我们先创建两个有现实意义的表，

```
CREATE TABLE student (
    number INT NOT NULL AUTO_INCREMENT COMMENT '学号',
    name VARCHAR(5) COMMENT '姓名',
    major VARCHAR(30) COMMENT '专业',
    PRIMARY KEY (number)
) Engine=InnoDB CHARSET=utf8 COMMENT '学生信息表';

CREATE TABLE score (
    number INT COMMENT '学号',
    subject VARCHAR(30) COMMENT '科目',
    score TINYINT COMMENT '成绩',
    PRIMARY KEY (number, subject)
) Engine=InnoDB CHARSET=utf8 COMMENT '学生成绩表';
```

我们新建了一个学生信息表，一个学生成绩表，然后我们向上述两个表中插入一些数据，为节省篇幅，具体插入过程就不唠叨了，插入后两表中的数据如下：

```
mysql> SELECT * FROM student;
+----------+-----------+--------------------------+
| number   | name      | major                    |
+----------+-----------+--------------------------+
```

```
| 20180101 | 杜子腾     | 软件学院                 |
| 20180102 | 范统       | 计算机科学与工程         |
| 20180103 | 史珍香     | 计算机科学与工程         |
+----------+-----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM score;
+----------+-----------------------------+-------+
| number   | subject                     | score |
+----------+-----------------------------+-------+
| 20180101 | 母猪的产后护理              |    78 |
| 20180101 | 论萨达姆的战争准备          |    88 |
| 20180102 | 论萨达姆的战争准备          |    98 |
| 20180102 | 母猪的产后护理              |   100 |
+----------+-----------------------------+-------+
4 rows in set (0.00 sec)
```

现在我们想把每个学生的考试成绩都查询出来就需要进行两表连接了（因为score中没有姓名信息，所以不能单纯只查询score表）。连接过程就是从student表中取出记录，在score表中查找number相同的成绩记录，所以过滤条件就是student.number = socre.number，整个查询语句就是这样：

```
mysql> SELECT * FROM student, score WHERE student.number = score.number;
+----------+-----------+--------------------------+----------+-----------------------------+-------+
| number   | name      | major                    | number   | subject                     | score |
+----------+-----------+--------------------------+----------+-----------------------------+-------+
| 20180101 | 杜子腾    | 软件学院                 | 20180101 | 母猪的产后护理              |    78 |
| 20180101 | 杜子腾    | 软件学院                 | 20180101 | 论萨达姆的战争准备          |    88 |
| 20180102 | 范统      | 计算机科学与工程         | 20180102 | 论萨达姆的战争准备          |    98 |
| 20180102 | 范统      | 计算机科学与工程         | 20180102 | 母猪的产后护理              |   100 |
+----------+-----------+--------------------------+----------+-----------------------------+-------+
4 rows in set (0.00 sec)
```

字段有点多哦，我们少查询几个字段：

```
mysql> SELECT s1.number, s1.name, s2.subject, s2.score FROM student AS s1, score AS s2 WHERE s1.number = s2.number;
+----------+-----------+-----------------------------+-------+
| number   | name      | subject                     | score |
+----------+-----------+-----------------------------+-------+
| 20180101 | 杜子腾    | 母猪的产后护理              |    78 |
| 20180101 | 杜子腾    | 论萨达姆的战争准备          |    88 |
| 20180102 | 范统      | 论萨达姆的战争准备          |    98 |
| 20180102 | 范统      | 母猪的产后护理              |   100 |
+----------+-----------+-----------------------------+-------+
4 rows in set (0.00 sec)
```

从上述查询结果中我们可以看到，各个同学对应的各科成绩就都被查出来了，可是有个问题，史珍香同学，也就是学号为20180103的同学因为某些原因没有参加考试，所以在score表中没有对应的成绩记录。那如果老师想查看所有同学的考试成绩，即使是缺考的同学也应该展示出来，但是到目前为止我们介绍的连接查询是无法完成这样的需求的。我们稍微思考一下这个需求，其本质是想：驱动表中的记录即使在被驱动表中没有匹配的记录，也仍然需要加入到结果集。为了解决这个问题，就有了内连接和外连接的概念：

- 对于内连接的两个表，驱动表中的记录在被驱动表中找不到匹配的记录，该记录不会加入到最后的结果集，我们上边提到的连接都是所谓的内连接。
- 对于外连接的两个表，驱动表中的记录即使在被驱动表中没有匹配的记录，也仍然需要加入到结果集。

  在MySQL中，根据选取驱动表的不同，外连接仍然可以细分为2种：

  - 左外连接

    选取左侧的表为驱动表。

  - 右外连接

    选取右侧的表为驱动表。

可是这样仍然存在问题，即使对于外连接来说，有时候我们也并不想把驱动表的全部记录都加入到最后的结果集。这就犯难了，有时候匹配失败要加入结果集，有时候又不要加入结果集，这咋办，有点儿愁啊。。。嘻，把过滤条件分为两种不就解决了这个问题了么，所以放在不同地方的过滤条件是有不同语义的：

- WHERE子句中的过滤条件

  WHERE子句中的过滤条件就是我们平时见的那种，不论是内连接还是外连接，凡是不符合WHERE子句中的过滤条件的记录都不会被加入最后的结果集。

- ON子句中的过滤条件

  对于外连接的驱动表的记录来说，如果无法在被驱动表中找到匹配ON子句中的过滤条件的记录，那么该记录仍然会被加入到结果集中，对应的被驱动表记录的各个字段使用NULL值填充。

  需要注意的是，这个ON子句是专门为外连接驱动表中的记录在被驱动表找不到匹配记录时应不应该把该记录加入结果集这个场景下提出的，所以如果把ON子句放到内连接中，MySQL会把它和WHERE子句一样对待，也就是说：内连接中的WHERE子句和ON子句是等价的。

一般情况下，我们都把只涉及单表的过滤条件放到WHERE子句中，把涉及两表的过滤条件都放到ON子句中，我们也一般把放到ON子句中的过滤条件也称之为连接条件。

> 小贴士： 左外连接和右外连接简称左连接和右连接，所以下边提到的左外连接和右外连接中的`外`字都用括号扩起来，以表示这个字儿可有可无。

左（外）连接的语法

左（外）连接的语法还是挺简单的，比如我们要把t1表和t2表进行左外连接查询可以这么写：

```
SELECT * FROM t1 LEFT [OUTER] JOIN t2 ON 连接条件 [WHERE 普通过滤条件];
```

其中中括号里的OUTER单词是可以省略的。对于LEFT JOIN类型的连接来说，我们把放在左边的表称之为外表或者驱动表，右边的表称之为内表或者被驱动表。所以上述例子中t1就是外表或者驱动表，t2就是内表或者被驱动表。需要注意的是，对于左（外）连接和右（外）连接来说，必须使用ON子句来指出连接条件。了解了左（外）连接的基本语法之后，再次回到我们上边那个现实问题中来，看看怎样写查询语句才能把所有的学生的成绩信息都查询出来，即使是缺考的考生也应该被放到结果集中：

```
mysql> SELECT s1.number, s1.name, s2.subject, s2.score FROM student AS s1 LEFT JOIN score AS s2 ON s1.number = s2.number;
+----------+-----------+----------------------------+-------+
| number   | name      | subject                    | score |
+----------+-----------+----------------------------+-------+
| 20180101 | 杜子腾    | 母猪的产后护理             |    78 |
| 20180101 | 杜子腾    | 论萨达姆的战争准备         |    88 |
| 20180102 | 范统      | 论萨达姆的战争准备         |    98 |
| 20180102 | 范统      | 母猪的产后护理             |   100 |
```

```
| 20180103 | 史珍香     | NULL                         |  NULL |
+----------+-----------+------------------------------+-------+
5 rows in set (0.04 sec)
```

从结果集中可以看出来，虽然史珍香并没有对应的成绩记录，但是由于采用的是连接类型为左（外）连接，所以仍然把她放到了结果集中，只不过在对应的成绩记录的各列使用NULL值填充而已。

右（外）连接的语法

右（外）连接和左（外）连接的原理是一样一样的，语法也只是把LEFT换成RIGHT而已：

```
SELECT * FROM t1 RIGHT [OUTER] JOIN t2 ON 连接条件 [WHERE 普通过滤条件];
```

只不过驱动表是右边的表，被驱动表是左边的表，具体就不唠叨了。

内连接的语法

内连接和外连接的根本区别就是在驱动表中的记录不符合ON子句中的连接条件时不会把该记录加入到最后的结果集，我们最开始唠叨的那些连接查询的类型都是内连接。不过之前仅仅提到了一种最简单的内连接语法，就是直接把需要连接的多个表都放到FROM子句后边。其实针对内连接，MySQL提供了好多不同的语法，我们以t1和t2表为例瞅瞅：

```
SELECT * FROM t1 [INNER | CROSS] JOIN t2 [ON 连接条件] [WHERE 普通过滤条件];
```

也就是说在MySQL中，下边这几种内连接的写法都是等价的：

- SELECT * FROM t1 JOIN t2;
- SELECT * FROM t1 INNER JOIN t2;
- SELECT * FROM t1 CROSS JOIN t2;

上边的这些写法和直接把需要连接的表名放到FROM语句之后，用逗号,分隔开的写法是等价的：

```
SELECT * FROM t1, t2;
```

现在我们虽然介绍了很多种内连接的书写方式，不过熟悉一种就好了，这里我们推荐INNER JOIN的形式书写内连接（因为INNER JOIN语义很明确嘛，可以和LEFT JOIN和RIGHT JOIN很轻松的区分开）。这里需要注意的是，由于在内连接中ON子句和WHERE子句是等价的，所以内连接中不要求强制写明ON子句。

我们前边说过，连接的本质就是把各个连接表中的记录都取出来依次匹配的组合加入结果集并返回给用户。不论哪个表作为驱动表，两表连接产生的笛卡尔积肯定是一样的。而对于内连接来说，由于凡是不符合ON子句或WHERE子句中的条件的记录都会被过滤掉，其实也就相当于从两表连接的笛卡尔积中把不符合过滤条件的记录给踢出去，所以对于内连接来说，驱动表和被驱动表是可以互换的，并不会影响最后的查询结果。但是对于外连接来说，由于驱动表中的记录即使在被驱动表中找不到符合ON子句条件的记录时也要将其加入到结果集，所以此时驱动表和被驱动表的关系就很重要了，也就是说左外连接和右外连接的驱动表和被驱动表不能轻易互换。

小结

上边说了很多，给大家的感觉不是很直观，我们直接把表t1和t2的三种连接方式写在一起，这样大家理解起来就很easy了：

```
mysql> SELECT * FROM t1 INNER JOIN t2 ON t1.m1 = t2.m2;
+------+------+------+------+
| m1   | n1   | m2   | n2   |
+------+------+------+------+
|    2 | b    |    2 | b    |
|    3 | c    |    3 | c    |
```

```
+------+------+------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM t1 LEFT JOIN t2 ON t1.m1 = t2.m2;
+------+------+------+------+
| m1   | n1   | m2   | n2   |
+------+------+------+------+
|    2 | b    |    2 | b    |
|    3 | c    |    3 | c    |
|    1 | a    | NULL | NULL |
+------+------+------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM t1 RIGHT JOIN t2 ON t1.m1 = t2.m2;
+------+------+------+------+
| m1   | n1   | m2   | n2   |
+------+------+------+------+
|    2 | b    |    2 | b    |
|    3 | c    |    3 | c    |
| NULL | NULL |    4 | d    |
+------+------+------+------+
3 rows in set (0.00 sec)
```

连接的原理

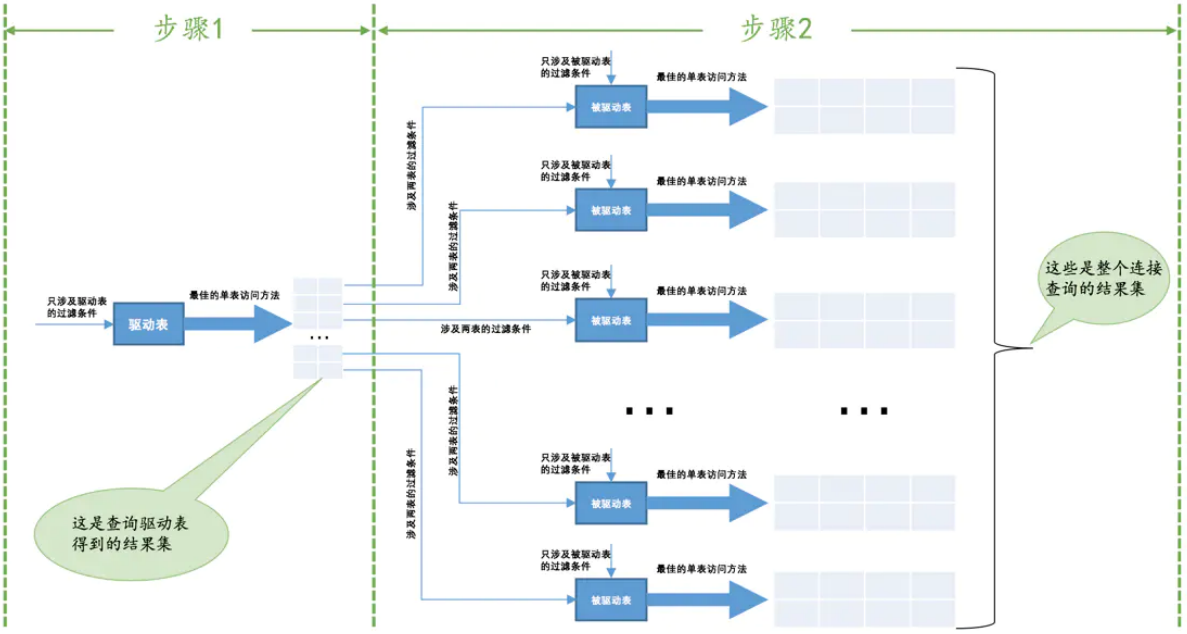
上边贼啰嗦的介绍都只是为了唤醒大家对连接、内连接、外连接这些概念的记忆，这些基本概念是为了真正进入本章主题做的铺垫。真正的重点是MySQL采用了什么样的算法来进行表与表之间的连接，了解了这个之后，大家才能明白为啥有的连接查询运行的快如闪电，有的却慢如蜗牛。

嵌套循环连接（Nested-Loop Join）

我们前边说过，对于两表连接来说，驱动表只会被访问一遍，但被驱动表却要被访问到好多遍，具体访问几遍取决于对驱动表执行单表查询后的结果集中的记录条数。对于内连接来说，选取哪个表为驱动表都没关系，而外连接的驱动表是固定的，也就是说左（外）连接的驱动表就是左边的那个表，右（外）连接的驱动表就是右边的那个表。我们上边已经大致介绍过t1表和t2表执行内连接查询的大致过程，我们温习一下：

- 步骤1：选取驱动表，使用与驱动表相关的过滤条件，选取代价最低的单表访问方法来执行对驱动表的单表查询。
- 步骤2：对上一步骤中查询驱动表得到的结果集中每一条记录，都分别到被驱动表中查找匹配的记录。

通用的两表连接过程如下图所示：



如果有3个表进行连接的话，那么步骤2中得到的结果集就像是新的驱动表，然后第三个表就成为了被驱动表，重复上边过程，也就是步骤2中得到的结果集中的每一条记录都需要到t3表中找一找有没有匹配的记录，用伪代码表示一下这个过程就是这样：

```
for each row in t1 {    #此处表示遍历满足对t1单表查询结果集中的每一条记录

    for each row in t2 {    #此处表示对于某条t1表的记录来说，遍历满足对t2单表查询结果集中的每一条记录

        for each row in t3 {    #此处表示对于某条t1和t2表的记录组合来说，对t3表进行单表查询
            if row satisfies join conditions, send to client
        }
    }
}
```

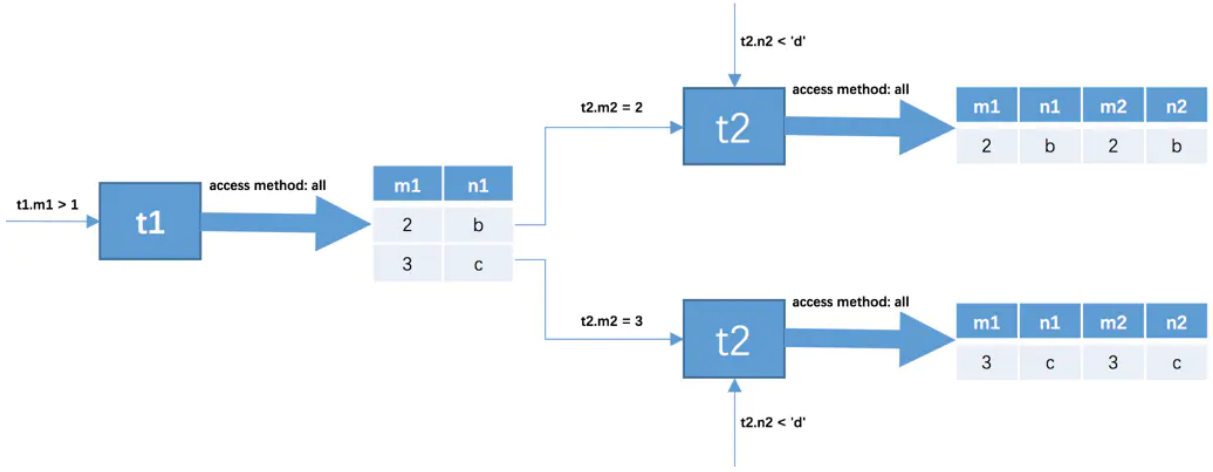
这个过程就像是一个嵌套的循环，所以这种驱动表只访问一次，但被驱动表却可能被多次访问，访问次数取决于对驱动表执行单表查询后的结果集中的记录条数的连接执行方式称之为嵌套循环连接（Nested-Loop Join），这是最简单，也是最笨拙的一种连接查询算法。

使用索引加快连接速度

我们知道在嵌套循环连接的步骤2中可能需要访问多次被驱动表，如果访问被驱动表的方式都是全表扫描的话，妈呀，那得要扫描好多次呀～～～ 但是别忘了，查询t2表其实就相当于一次单表扫描，我们可以利用索引来加快查询速度哦。回顾一下最开始介绍的t1表和t2表进行内连接的例子：

```
SELECT * FROM t1, t2 WHERE t1.m1 > 1 AND t1.m1 = t2.m2 AND t2.n2 < 'd';
```

我们使用的其实是嵌套循环连接算法执行的连接查询，再把上边那个查询执行过程表拉下来给大家看一下：



查询驱动表t1后的结果集中有两条记录，嵌套循环连接算法需要对被驱动表查询2次：

- 当t1.m1 = 2时，去查询一遍t2表，对t2表的查询语句相当于：

  ```
  SELECT * FROM t2 WHERE t2.m2 = 2 AND t2.n2 < 'd';
  ```

- 当t1.m1 = 3时，再去查询一遍t2表，此时对t2表的查询语句相当于：

  ```
  SELECT * FROM t2 WHERE t2.m2 = 3 AND t2.n2 < 'd';
  ```

可以看到，原来的t1.m1 = t2.m2这个涉及两个表的过滤条件在针对t2表做查询时关于t1表的条件就已经确定了，所以我们只需要单单优化对t2表的查询了，上述两个对t2表的查询语句中利用到的列是m2和n2列，我们可以：

- 在m2列上建立索引，因为对m2列的条件是等值查找，比如t2.m2 = 2、t2.m2 = 3等，所以可能使用到ref的访问方法，假设使用ref的访问方法去执行对t2表的查询的话，需要回表之后再判断t2.n2 < d这个条件是否成立。

  这里有一个比较特殊的情况，就是假设m2列是t2表的主键或者唯一二级索引列，那么使用t2.m2 = 常数值这样的条件从t2表中查找记录的过程的代价就是常数级别的。我们知道在单表中使用主键值或者唯一二级索引列的值进行等值查找的方式称之为const，而设计MySQL的大叔把在连接查询中对被驱动表使用主键值或者唯一二级索引列的值进行等值查找的查询执行方式称之为：eq_ref。

- 在n2列上建立索引，涉及到的条件是t2.n2 < 'd'，可能用到range的访问方法，假设使用range的访问方法对t2表的查询的话，需要回表之后再判断在m2列上的条件是否成立。

假设m2和n2列上都存在索引的话，那么就需要从这两个里边儿挑一个代价更低的去执行对t2表的查询。当然，建立了索引不一定使用索引，只有在二级索引 + 回表的代价比全表扫描的代价更低时才会使用索引。

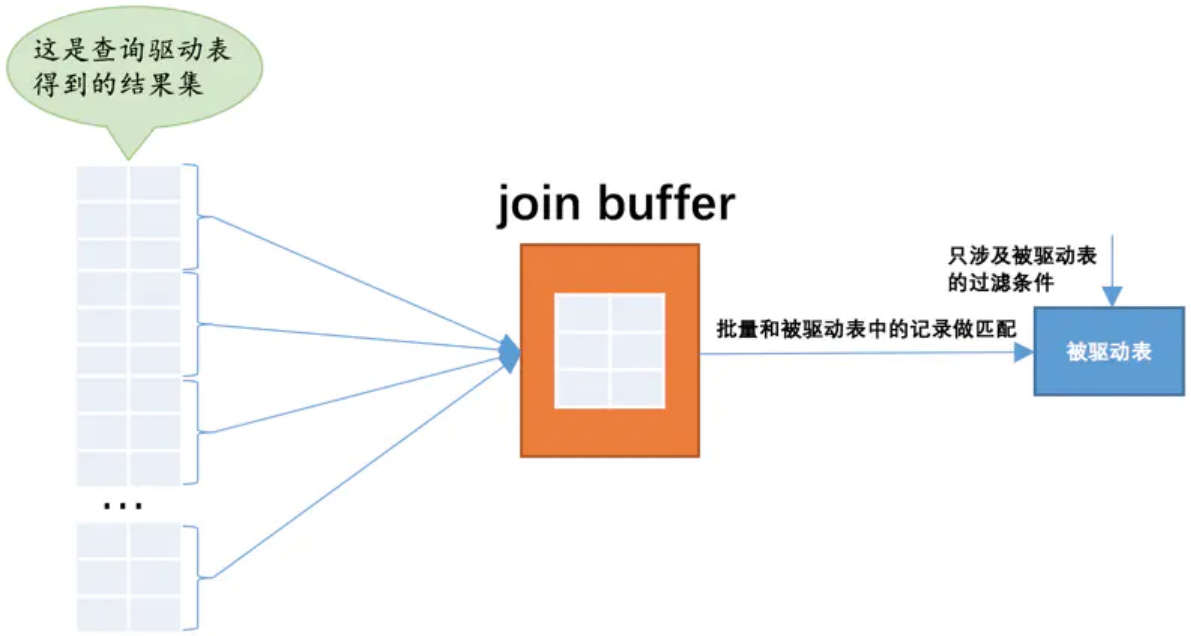
另外，有时候连接查询的查询列表和过滤条件中可能只涉及被驱动表的部分列，而这些列都是某个索引的一部分，这种情况下即使不能使用eq_ref、ref、ref_or_null或者range这些访问方法执行对被驱动表的查询的话，也可以使用索引扫描，也就是index的访问方法来查询被驱动表。所以我们建议在真实工作中最好不要使用*作为查询列表，最好把真实用到的列作为查询列表。

基于块的嵌套循环连接（Block Nested-Loop Join）

扫描一个表的过程其实是先把这个表从磁盘上加载到内存中，然后从内存中比较匹配条件是否满足。现实生活中的表可不像t1、t2这种只有3条记录，成千上万条记录都是少的，几百万、几千万甚至几亿条记录的表到处都是。内存里可能并不能完全存放的下表中所有的记录，所以在扫描表前边记录的时候后边的记录可能还在磁盘上，等扫描到后边记录的时候可能内存不足，所以需要把前边的记录从内存中释放掉。我们前边又说过，采用嵌套循环连接算法的两表连接过程中，被驱动表可是要被访问好多次的，如果这个被驱动表中的数据特别多而且不能使用索引进行访问，那就相当于要从磁盘上读好几次这个表，这个I/O代价就非常大了，所以我们得想办法：尽量减少访问被驱动表的次数。

当被驱动表中的数据非常多时，每次访问被驱动表，被驱动表的记录会被加载到内存中，在内存中的每一条记录只会和

驱动表结果集的一条记录做匹配，之后就会被从内存中清除掉。然后再从驱动表结果集中拿出另一条记录，再一次把被驱动表的记录加载到内存中一遍，周而复始，驱动表结果集中有多少条记录，就得把被驱动表从磁盘上加载到内存中多少次。所以我们可不可以在把被驱动表的记录加载到内存的时候，一次性和多条驱动表中的记录做匹配，这样就可以大大减少重复从磁盘上加载被驱动表的代价了。所以设计MySQL的大叔提出了一个join buffer的概念，join buffer就是执行连接查询前申请的一块固定大小的内存，先把若干条驱动表结果集中的记录装在这个join buffer中，然后开始扫描被驱动表，每一条被驱动表的记录一次性和join buffer中的多条驱动表记录做匹配，因为匹配的过程都是在内存中完成的，所以这样可以显著减少被驱动表的I/O代价。使用join buffer的过程如下图所示：



最好的情况是join buffer足够大，能容纳驱动表结果集中的所有记录，这样只需要访问一次被驱动表就可以完成连接操作了。设计MySQL的大叔把这种加入了join buffer的嵌套循环连接算法称之为基于块的嵌套连接（Block Nested-Loop Join）算法。

这个join buffer的大小是可以通过启动参数或者系统变量join_buffer_size进行配置，默认大小为262144字节（也就是256KB），最小可以设置为128字节。当然，对于优化被驱动表的查询来说，最好是为被驱动表加上效率高的索引，如果实在不能使用索引，并且自己的机器的内存也比较大可以尝试调大join_buffer_size的值来对连接查询进行优化。

另外需要注意的是，驱动表的记录并不是所有列都会被放到join buffer中，只有查询列表中的列和过滤条件中的列才会被放到join buffer中，所以再次提醒我们，最好不要把*作为查询列表，只需要把我们关心的列放到查询列表就好了，这样还可以在join buffer中放置更多的记录呢哈。