链滴

Oracle 执行计划详解

作者：Eddie

原文链接：https://ld246.com/article/1492676265513

来源网站：链滴

本文源自TTT BLOG，原文地址：http://blog.chinaunix.net/u3/107265/showart_2192657.html

简介：

本文全面详细介绍[**oracle**](%2E.:;)执行计划的相关的概念，访问数据的存取方法，表之间的连接内容。

并有总结和概述，便于理解与记忆!

+++

目录

（4） 索引快速扫描（index fast full scan）

三、表之间的连接

1，排序 - - 合并连接（Sort Merge Join， SMJ）

2，嵌套循环（Nested Loops， NL）

3，哈希连接（Hash Join， HJ）

另外，笛卡儿乘积（Cartesian Product）

总结Oracle连接方法

Oracle执行计划总结概述

+++

一．相关的概念

**

**　　**Rowid的概念**：rowid是一个伪列，既然是伪列，那么这个列就不是用户定义，而是系统自己加上的。 对每个表都有一个rowid的伪列，但是表中并不物理存储ROWID列的值。不过你可以像使其它列那样使用它，但是不能删除改列，也不能对该列的值进行 修改、插入。一旦一行数据插入**数据库**则rowid在该行的生命周期内是唯一的，即即使该行产生行迁移，行的rowid也不会改变。

Recursive SQL概念：有时为了执行用户发出的一个sql语句，Oracle必须执行一些额外的语句我们将这些额外的语句称之为''recursive calls''或''recursive SQL statements''.如当一个DDL语句发后，ORACLE总是隐含的发出一些recursive SQL语句，来修改数据字典信息，以便用户可以成功的执该DDL语句。当需要的数据字典信息没有在共享内存中时，经常会发生Recursive calls，这些Recursie calls会将数据字典信息从硬盘读入内存中。用户不比关心这些recursive SQL语句的执行情况，在要的时候，ORACLE会自动的在内部执行这些语句。当然DML语句与SELECT都可能引起recursive SQL简单的说，我们可以将触发器视为recursive SQL.

Row Source（行源）：用在查询中，由上一操作返回的符合条件的行的集合，即可以是表的全行数据的集合；也可以是表的部分行数据的集合；也可以为对上2个row source进行连接操作（如joi连接）后得到的行数据集合。

Predicate（谓词）：一个查询中的WHERE限制条件

Driving Table（驱动表）：该表又称为外层表（OUTER TABLE）。这个概念用于嵌套与HASH接中。如果该row source返回较多的行数据，则对所有的后续操作有负面影响。注意此处虽然翻译为动表，但实际上翻译为驱动行源（driving row source）更为确切。一般说来，是应用查询的限制条后，返回较少行源的表作为驱动表，所以如果一个大表在WHERE条件有有限制条件（如等值限 制）则该大表作为驱动表也是合适的，所以并不是只有较小的表可以作为驱动表，正确说法应该为应用查的限制条件后，返回较少行源的表作为驱动表。在执行 计划中，应该为靠上的那个row source，后会给出具体说明。在我们后面的描述中，一般将该表称为连接操作的row source 1.

Probed Table（被探查表）：该表又称为内层表（INNER TABLE）。在我们从驱动表中得到具一行的数据后，在该表中寻找符合连接条件的行。所以该表应当为大表（实际上应该为返回较大row surce的表）且相应的列上应该有索引。在我们后面的描述中，一般将该表称为连接操作的row source .

组合索引（concatenated index）：由多个列构成的索引，如create index idx_emp on emp col1， col2， col3， ……），则我们称idx_emp索引为组合索引。在组合索引中有一个重要的概念引导列（leading column），在上面的例子中，col1列为引导列。当我们进行查询时可以使用“wher

col1 = ？ ”，也可以使用“where col1 = ？ and col2 = ？”，这样的限制条件都会使用索引，但“where col2 = ？ ”查询就不会使用该索引。所以限制条件中包含先导列时，该限制条件才会使用组合索引。

可选择性（selectivity）：比较一下列中唯一键的数量和表中的行数，就可以判断该列的可选择。 如果该列的“唯一键的数量/表中的行数”的比值越接近1，则该列的可选择性越高，该列就越适合建索引，同样索引的可选择性也越高。在可选择性高的列上进 行查询时，返回的数据就较少，比较适使用索引查询。

二．oracle访问数据的存取方法

**

**　　1） 全表扫描（Full Table Scans， FTS）**

为实现全表扫描，Oracle读取表中所有的行，并检查每一行是否满足语句的WHERE限制条件一多块读操作可以使一次I/O能读取多块数据块（db_block_multiblock_read_count参数设定），而不只读取一个数据块，这极大的减 少了I/O总次数，提高了系统的吞吐量，所以利用多块读的方法可以分高效地实现全表扫描，而且只有在全表扫描的情况下才能使用多块读操作。在这种访问模 式下，每数据块只被读一次。

使用FTS的前提条件：在较大的表上不建议使用全表扫描，除非取出数据的比较多，超过总量的5 —— 10%，或你想使用并行查询功能时。

使用全表扫描的例子：

SQL> explain plan for select * from dual;

Query Plan

SELECT STATEMENT[CHOOSE] Cost=

TABLE ACCESS FULL DUAL

2） 通过ROWID的表存取（Table Access by ROWID或rowid lookup）

行的ROWID指出了该行所在的数据文件、数据块以及行在该块中的位置，所以通过ROWID来存数据可以快速定位到目标数据上，是Oracle存取单行数据的最快方法。

这种存取方法不会用到多块读操作，一次I/O只能读取一个数据块。我们会经常在执行计划中看该存取方法，如通过索引查询数据。

使用ROWID存取的方法：

SQL> explain plan for select * from dept where rowid = ''AAAAyGAADAAAAATAAF'';

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID DEPT [ANALYZED]

3）索引扫描（Index Scan或index lookup）

我们先通过index查找到数据对应的rowid值（对于非唯一索引可能返回多个rowid值），然后根rowid直接从表中得到具体的数据，这 种查找方式称为索引扫描或索引查找（index lookup）。一个rwid唯一的表示一行数据，该行对应的数据块是通过一次i/o得到的，在此情况下该次i/o只会读取一个据库块。

在索引中，除了存储每个索引的值外，索引还存储具有此值的行对应的ROWID值。

索引扫描可以由2步组成：

（1） 扫描索引得到对应的rowid值。

（2） 通过找到的rowid从表中读出具体的数据。

每步都是单独的一次I/O，但是对于索引，由于经常使用，绝大多数都已经CACHE到内存中，所第1步的 I/O经常是逻辑I/O，即数据可以从内存中得到。但是对于第2步来说，如果表比较大，则其据不可能全在内存中，所以其I/O很有可能是物理I/O，这 是一个机械操作，相对逻辑I/O来说，是极费时间的。所以如果多大表进行索引扫描，取出的数据如果大于总量的5% —— 10%，使用索引扫描效率下降很多。如下列所示：

SQL> explain plan for select empno， ename from emp where empno=10;

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX UNIQUE SCAN EMP_I1

但是如果查询的数据能全在索引中找到，就可以避免进行第2步操作，避免了不必要的I/O，此时使通过索引扫描取出的数据比较多，效率还是很高的

SQL> explain plan for select empno from emp where empno=10;-- 只查询empno列值

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

INDEX UNIQUE SCAN EMP_I1

进一步讲，如果sql语句中对索引列进行排序，因为索引已经预先排序好了，所以在执行计划中需要再对索引列进行排序

SQL> explain plan for select empno, ename from emp

where empno > 7876 order by empno;

Query Plan

SELECT STATEMENT[CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX RANGE SCAN EMP_I1 [ANALYZED]

从这个例子中可以看到：因为索引是已经排序了的，所以将按照索引的顺序查询出符合条件的行因此避免了进一步排序操作。

根据索引的类型与where限制条件的不同，有4种类型的索引扫描：

索引唯一扫描（index unique scan）

索引范围扫描（index range scan）

索引全扫描（index full scan）

索引快速扫描（index fast full scan）

（1） 索引唯一扫描（index unique scan）

通过唯一索引查找一个数值经常返回单个ROWID.如果存在UNIQUE 或PRIMARY KEY 约束（它证了语句只存取单行）的话，Oracle经常实现唯一性扫描。

使用唯一性约束的例子：

SQL> explain plan for

select empno，ename from emp where empno=10；

Query Plan

SELECT STATEMENT [CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX UNIQUE SCAN EMP_I1

（2） 索引范围扫描（index range scan）

使用一个索引存取多行数据，在唯一索引上使用索引范围扫描的典型情况下是在谓词（where限条件）中使用了范围操作符（如>、<、<>、>=、<=、between）

使用索引范围扫描的例子：

SQL> explain plan for select empno，ename from emp

where empno > 7876 order by empno；

Query Plan

SELECT STATEMENT[CHOOSE] Cost=1

TABLE ACCESS BY ROWID EMP [ANALYZED]

INDEX RANGE SCAN EMP_I1 [ANALYZED]

在非唯一索引上，谓词col = 5可能返回多行数据，所以在非唯一索引上都使用索引范围扫描。

使用index rang scan的3种情况：

（a） 在唯一索引列上使用了range操作符（> < <> >= <= between）

（b） 在组合索引上，只使用部分列进行查询，导致查询出多行

（c） 对非唯一索引列上进行的任何查询。

（3） 索引全扫描（index full scan）

与全表扫描对应，也有相应的全索引扫描。而且此时查询出的数据都必须从索引中可以直接得到。

全索引扫描的例子：

An Index full scan will not perform. single block i/o''s and so it may prove to be inefficient.

e.g.

Index BE_IX is a concatenated index on big_emp （empno， ename）

SQL> explain plan for select empno， ename from big_emp order by empno，ename；

Query Plan

SELECT STATEMENT[CHOOSE] Cost=26

INDEX FULL SCAN BE_IX [ANALYZED]

（4） 索引快速扫描（index fast full scan）

扫描索引中的所有的数据块，与 index full scan很类似，但是一个显著的区别就是它不对查询出数据进行排序，即数据不是以排序顺序被返回。在这种存取方法中，可以使用多块读功能，也可以使并行读入，以便获得最大吞吐量与缩短执行时间。

索引快速扫描的例子：

BE_IX索引是一个多列索引： big_emp （empno，ename）

SQL> explain plan for select empno，ename from big_emp；

Query Plan

SELECT STATEMENT[CHOOSE] Cost=1

INDEX FAST FULL SCAN BE_IX [ANALYZED]

只选择多列索引的第2列：

SQL> explain plan for select ename from big_emp；

Query Plan

SELECT STATEMENT[CHOOSE] Cost=1

INDEX FAST FULL SCAN BE_IX [ANALYZED]

三、表之间的连接

**

** Join是一种试图将两个表结合在一起的谓词，一次只能连接2个表，表连接也可以被称为表关。在后面的叙 述中，我们将会使用“row source”来代替“表”，因为使用row source更严谨一些并且将参与连接的2个row source分别称为row source1和row source 2.Join过程的各个步骤经常是行操作，即使相关的row source可以被并行访问，即可以并行的读取做join连接的两个row source的据，但是在将表中符合限制条件的数据读入到内存形成row source后，join的其它步骤一般是串行的有多种方法可以将2个表连接起来，当然每种方法都有自己的优缺点，每种连接类型只有在特定的条下才会 发挥出其最大优势。

row source（表）之间的连接顺序对于查询的效率有非常大的影响。通过首先存取特定的表，即该表作为驱动表，这样可以先应用某些限制条件，从而得到一个 较小的row source，使连接的效率高，这也就是我们常说的要先执行限制条件的原因。一般是在将表读入内存时，应用where子句中对表的限制条件。

根据2个row source的连接条件的中操作符的不同，可以将连接分为等值连接（如WHERE A.COL = B.COL4）、非等值连接（WHERE A.COL3 > B.COL4）、外连接（WHERE A.COL3 = B.COL4（））。上面的各个连接的连接原理都基本一样，所以为了简单期间，下面以等值连接为例进行介绍。

在后面的介绍中，都以以下Sql为例进行说明：

SELECT A.COL1， B.COL2

FROM A， B

WHERE A.COL3 = B.COL4；

假设A表为Row Soruce1，则其对应的连接操作关联列为COL 3；

B表为Row Soruce2，则其对应的连接操作关联列为COL 4；

连接类型：

目前为止，无论连接操作符如何，典型的连接类型共有3种：

排序 - - 合并连接（Sort Merge Join （SMJ） ）

嵌套循环（Nested Loops （NL） ）

哈希连接（Hash Join）

另外，还有一种Cartesian product（笛卡尔积），一般情况下，尽量避免使用。

1，排序 - - 合并连接（Sort Merge Join， SMJ）

内部连接过程**：**

1） 首先生成row source1需要的数据，然后对这些数据按照连接操作关联列（如A.col3）进行序。

2） 随后生成row source2需要的数据，然后对这些数据按照与sort source1对应的连接操作关列（如B.col4）进行排序。

3） 最后两边已排序的行被放在一起执行合并操作，即将2个row source按照连接条件连接起来

下面是连接步骤的图形表示：

MERGE

/

SORTSORT

||

Row Source 1Row Source 2

如果row source已经在连接关联列上被排序，则该连接操作就不需要再进行sort操作，这样可以大提高这种连接操作的连接速度，因为排序是个极其费资源的操 作，特别是对于较大的表。预先排序row source包括已经被索引的列（如a.col3或b.col4上有索引）或row source已经在前面的步骤中被序了。尽管合并两个row source的过程是串行的，但是可以并行访问这两个row source（如并行读数据，并行排序）。

SMJ连接的例子：

```
SQL> explain plan for
select**/*+ ordered */**e.deptno， d.deptno
from emp e， dept d
where e.deptno = d.deptno
order by e.deptno， d.deptno;
Query Plan
-------------------------------------
SELECT STATEMENT [CHOOSE] Cost=17
MERGE JOIN
SORT JOIN
TABLE ACCESS FULL EMP [ANALYZED]
```

SORT JOIN

TABLE ACCESS FULL DEPT [ANALYZED]

排序是一个费时、费资源的操作，特别对于大表。基于这个原因，SMJ经常不是一个特别有效的接方法，但是如果2个row source都已经预先排序，则这种连接方法的效率也是蛮高的。

2，嵌套循环（Nested Loops， NL）

这个连接方法有驱动表（外部表）的概念。其实，该连接过程就是一个2层嵌套循环，所以外层环的次数越少越好，这也就是我们为什么将小表或返回较小 row source的表作为驱动表（用于外层环）的理论依据。但是这个理论只是一般指导原则，因为遵循这个理论并不能总保证使语句产生的I/次数最少。有时 不遵守这个理论依据，反而会获得更好的效率。如果使用这种方法，决定使用哪个表为驱动表很重要。有时如果驱动表选择不正确，将会导致语句的性能很差、很差。

内部连接过程：

Row source1的Row 1 —— Probe ->Row source 2

Row source1的Row 2 —— Probe ->Row source 2

Row source1的Row 3 —— Probe ->Row source 2

……。

Row source1的Row n —— Probe ->Row source 2

从内部连接过程来看，需要用row source1中的每一行，去匹配row source2中的所有行，所以时保持row source1尽可能的小与高效的访问row source2（一般通过索引实现）是影响这个连接效的关键问题。这只是理论指导原则，目的是使整个连接操作产生最少的物理I/O次数，而且如果遵守这个原则，一般也会使总的物理I/O数最少。但是如果不遵从这个指导原则，反而能用更少的物理I/O实连接操作，那尽管违反指导原则吧！因为最少的物理 I/O次数才是我们应该遵从的真正的指导原则，后面的具体案例分析中就给出这样的例子。

在上面的连接过程中，我们称Row source1为驱动表或外部表。Row Source2被称为被探查表或部表。

在NESTED LOOPS连接中，Oracle读取row source1中的每一行，然后在row sourc2中检查是有匹配的行，所有被匹配的行都被放到结果集中，然后处理row source1中的下一行。这个过程一直续，直到row source1中的所有行都被处理。这是从连接操作中可以得到第一个匹配行的最快的方法一，这种类型的连接可以用在需要快速响应的语句中，以响应速度为 主要目标。

如果driving row source（外部表）比较小，并且在inner row source（内部表）上有唯一索引或有高选择性非唯一索引时，使用这种方法可以得到较好的效率。NESTED LOOPS有其它连接方法有的的一个优点是：可以先返回已经连接的行，而不必等待所有的连接操作处理完才返回数据，这可实现快速的响应时间。

如果不使用并行操作，最好的驱动表是那些应用了where 限制条件后，可以返回较少行数据的的，所以大表也可能称为驱动表，关键看限制条件。对于并行查询，我们经常选择大表作为驱动表，因大表可以充分利用并 行功能。当然，有时对查询使用并行操作并不一定会比查询不使用并行操作效率，因为最后可能每个表只有很少的行符合限制条件，而且还要看你的硬件配置是否 可以支持并行（如否有多个CPU，多个硬盘控制器），所以要具体问题具体对待。

NL连接的例子：

```
SQL> explain plan for
select a.dname，b.sql
from dept a，emp b
where a.deptno = b.deptno;
Query Plan
```

SELECT STATEMENT [CHOOSE] Cost=5

NESTED LOOPS

TABLE ACCESS FULL DEPT [ANALYZED]

TABLE ACCESS FULL EMP [ANALYZED]

3，哈希连接（Hash Join， HJ)

这种连接是在oracle 7.3以后引入的，从理论上来说比NL与SMJ更高效，而且只用在CBO优化器。

较小的row source被用来构建hash table与bitmap，第2个row source被用来被hansed，并与一个row source生成的hash table进行匹配，以便进行进一步的连接。Bitmap被用来作为一种比较的查找方法，来检查在hash table中是否有匹配的行。特别的，当hash table比较大而不能全部容纳内存中时，这种查找方法更为有用。这种连接方法也有NL连接中所谓的驱动表的概念，被构建为hash able与bitmap的表为驱动表，当被构建的hash table与bitmap能被容纳在内存中时，这种连接方式效率极高。

HASH连接的例子：

SQL> explain plan for

select/*+ use_hash（emp） */empno

from emp， dept

where emp.deptno = dept.deptno;

Query Plan

SELECT STATEMENT[CHOOSE] Cost=3

HASH JOIN

TABLE ACCESS FULL DEPT

TABLE ACCESS FULL EMP

要使哈希连接有效，需要设置HASH_JOIN_ENABLED=TRUE，缺省情况下该参数为TRUE，另外不要忘了还要设置 hash_area_size参数，以使哈希连接高效运行，因为哈希连接会在该参数指定大小内存中运行，过小的参数会使哈希连接的性能比其他连接方式还 要低。

另外，笛卡儿乘积（Cartesian Product)

当两个row source做连接，但是它们之间没有关联条件时，就会在两个row source中做笛卡儿积，这通常由编写代码疏漏造成（即程序员忘了写关联条件）。笛卡尔乘积是一个表的每一行依次与一个表中的所有行匹配。在特殊情况下我们可以使用笛卡儿乘积，如在星形连接中，除此之外，我们尽量不使用笛卡儿乘积，否则，自己想结果是什么吧!

注意在下面的语句中，在2个表之间没有连接。

SQL> explain plan for

select emp.deptno，dept，deptno

from emp，dept

Query Plan

SLECT STATEMENT [CHOOSE] Cost=5

MERGE JOIN CARTESIAN

TABLE ACCESS FULL DEPT

SORT JOIN

TABLE ACCESS FULL EMP

CARTESIAN关键字指出了在2个表之间做笛卡尔乘积。假如表emp有n行，dept表有m行，笛卡乘积的结果就是得到n * m行结果。

最后，总结一下，在哪种情况下用哪种连接方法比较好：

排序 - - 合并连接（Sort Merge Join， SMJ）：

a） 对于非等值连接，这种连接方式的效率是比较高的。

b） 如果在关联的列上都有索引，效果更好。

c） 对于将2个较大的row source做连接，该连接方法比NL连接要好一些。

d） 但是如果sort merge返回的row source过大，则又会导致使用过多的rowid在表中查询数据，数据库性能下降，因为过多的I/O.

嵌套循环（Nested Loops， NL）：

a） 如果driving row source（外部表）比较小，并且在inner row source（内部表）上有唯一引，或有高选择性非唯一索引时，使用这种方法可以得到较好的效率。

b） NESTED LOOPS有其它连接方法没有的的一个优点是：可以先返回已经连接的行，而不必待所有的连接操作处理完才返回数据，这可以实现快速的响应时间。

哈希连接（Hash Join， HJ）：

a） 这种方法是在oracle7后来引入的，使用了比较先进的连接理论，一般来说，其效率应该好于它2种连接，但是这种连接只能用在CBO优化器中，而且需要设置合适的hash_area_size参数，才能得较好的性能。

b） 在2个较大的row source之间连接时会取得相对较好的效率，在一个row source较小时则能得更好的效率。

c） 只能用于等值连接中

+++

Oracle执行计划的概述

Oracle执行计划的相关概念：

Rowid：系统给oracle数据的每行附加的一个伪列，包含数据表名称，数据库id，存储数据库id及一个流水号等信息，rowid在行的生命周期内唯一。

Recursive sql：为了执行用户语句，系统附加执行的额外操作语句，譬如对数据字典的维护等。

Row source（行源）：oracle执行步骤过程中，由上一个操作返回的符合条件的行的集合。

Predicate（谓词）：where后的限制条件。

Driving table（驱动表）：又称为连接的外层表，主要用于嵌套与hash连接中。一般来说是将用限制条件后，返回较少行源的表作为驱动表。在后面的描述中，将driving table称为连接操作的row source 1。

Probed table（被探查表）：连接的内层表，在我们从driving table得到具体的一行数据后，在robed table中寻找符合条件的行，所以该表应该为较大的row source，并且对应连接条件的列上应有索引。在后面的描述中，一般将该表称为连接操作的row source 2.

Concatenated index（组合索引）：一个索引如果由多列构成，那么就称为组合索引，组合索的第一列为引导列，只有谓词中包含引导列时，索引才可用。

可选择性：表中某列的不同数值数量/表的总行数如果接近于1，则列的可选择性为高。

Oracle访问数据的存取方法：

**

** **Full table scans, FTS(全表扫描)**：通过设置db_block_multiblock_read_count可以设置一次O能读取的数据块个数，从而有效减少全表扫描时的IO总次数，也就是通过预读机制将将要访问的数块预先读入内存中。只有在全表扫描情况下才能使用多块读操作。

Table Access by rowed（通过rowid存取表，rowid lookup）：由于rowid中记录了行存储位置，所以这是oracle存取单行数据的最快方法。

Index scan（索引扫描index lookup）：在索引中，除了存储每个索引的值外，索引还存储具此值的行对应的rowid值，索引扫描分两步1，扫描索引得到rowid；2，通过 rowid读取具体数据。步都是单独的一次IO，所以如果数据经限制条件过滤后的总量大于原表总行数的5% - 10%,则使用索扫描效率下降很多。而如果结果数据能够全部在索引中找到，则可以避免第二步操作，从而加快检索度。

根据索引类型与where限制条件的不同，有4种类型的索引扫描：

Index unique scan（索引唯一扫描）：存在unique或者primary key的情况下，返回单个rowid据内容。

Index range scan（索引范围扫描）：1，在唯一索引上使用了range操作符（>,<,<>,>=,<=,between）；2，在组合索引上，只使用部分列进行查询；3，对非唯一索引上的列进行的查询。

Index full scan（索引全扫描）：需要查询的数据从索引中可以全部得到。

Index fast full scan（索引快速扫描）：与index full scan类似，但是这种方式下不对结果进行序。

目前为止，典型的连接类型有3种：

**

** **Sort merge join（SMJ排序 - 合并连接）**：首先生产driving table需要的数据，然后对这些据按照连接操作关联列进行排序；然后生产probed table需要的数据，然后对这些数据按照与driving able对应的连接操作列进行排序；最后两边已经排序的行被放在一起执行合并操作。排序是一个费时费资源的操作，特别对于大表。所以smj通常不是一个特别有效的连接方法，但是如果driving table和robed table都已经预先排序，则这种连接方法的效率也比较高。

Nested loops（NL嵌套循环）：连接过程就是将driving table和probed table进行一次嵌套循的过程。就是用driving table的每一行去匹配probed table 的所有行。Nested loops可以先返回已连接的行，而不必等待所有的连接操作处理完成才返回数据，这可以实现快速的响应时间。

Hash join（哈希连接）：较小的row source被用来构建hash table与bitmap，第二个row source用来被hashed，并与第一个row source生产的hash table进行匹配。以便进行进一步的连接。当被建的hash table与bitmap能被容纳在内存中时，这种连接方式的效率极高。但需要设置合适的hash_area_size参数且只能用于等值连接中。

另外，还有一种连接类型：**Cartesian product（笛卡尔积）**：表的每一行依次与另外一表的所行匹配，一般情况下，尽量避免使用。