

看懂Oracle执行计划

www.cnblogs.com 2016-11-18 09:38

最近一直在跟Oracle打交道，从最初的一脸懵逼到现在的略有所知，也来总结一下自己最近所学，不定时更新ing...

一：什么是Oracle执行计划？

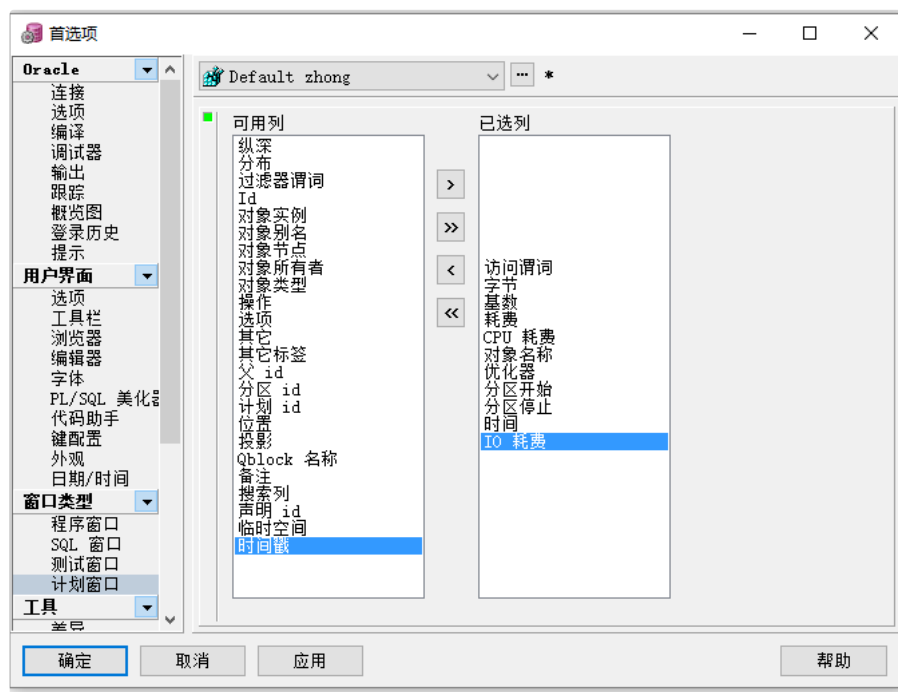
执行计划是一条查询语句在Oracle中的执行过程或访问路径的描述

二：怎样查看Oracle执行计划？

因为我一直用的PLSQL远程连接的公司数据库，所以这里以PLSQL为例：

①：配置执行计划需要显示的项：

工具 —> 首选项 —> 窗口类型 —> 计划窗口 —> 根据需要配置要显示在执行计划中的列



执行计划配置

执行计划的常用列字段解释：

基数 (Rows)：Oracle估计的当前操作的返回结果集行数

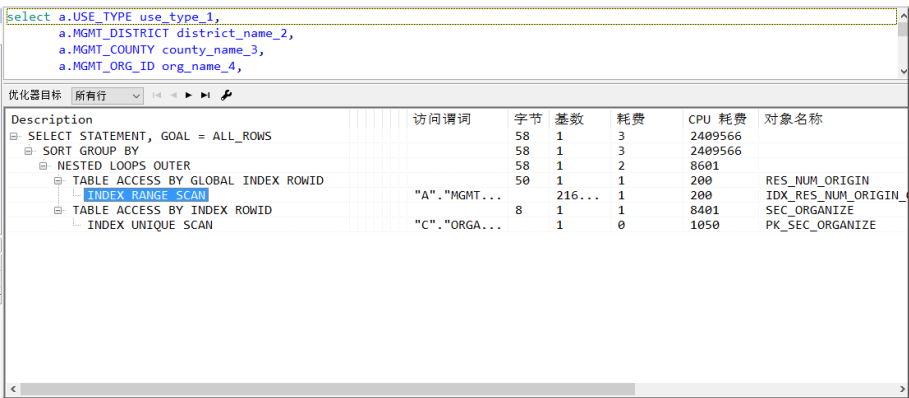
字节 (Bytes) : 执行该步骤后返回的字节数

耗费 (COST)、CPU耗费 : Oracle估计的该步骤的执行成本 , 用于说明SQL执行的代价 , 理论上越小越好 (该值可能与实际有出入)

时间 (Time) : Oracle估计的当前操作所需的时间

② : 打开执行计划 :

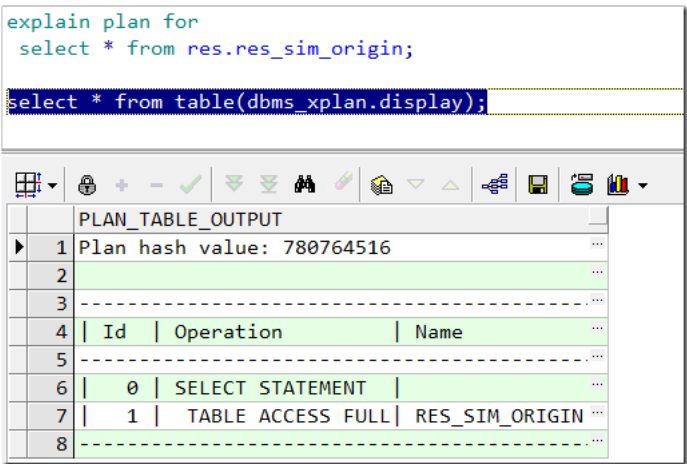
在SQL窗口执行完一条select语句后按 **F5** 即可查看刚刚执行的这条查询语句的执行计划



Description	访问谓词	字节	基数	耗费	CPU 耗费	对象名称
SELECT STATEMENT, GOAL = ALL_ROWS		58	1	3	2409566	
SORT GROUP BY		58	1	3	2409566	
NESTED LOOPS OUTER		58	1	2	8601	
TABLE ACCESS BY GLOBAL INDEX ROWID		50	1	1	200	RES_NUM_ORIGIN
INDEX RANGE SCAN	"A"."MGMT..."	8	216...	1	200	IDX_RES_NUM_ORIGIN
TABLE ACCESS BY INDEX ROWID		8	1	1	8401	SEC_ORGANIZE
INDEX UNIQUE SCAN	"C"."ORGA..."	1	0	1050		PK_SEC_ORGANIZE

执行计划查看

注 : 在PLSQL中使用SQL命令查看执行计划的话 , 某些SQL*PLUS命令PLSQL无法支持 , 比如SET AUTOTRACE ON



```
explain plan for
select * from res.res_sim_origin;

select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT			
1	Plan hash value: 780764516		
2	...		
3	-----		
4	Id	Operation	Name
5	-----		
6	0	SELECT STATEMENT	...
7	1	TABLE ACCESS FULL	RES_SIM_ORIGIN
8	-----		

执行计划sql查看

三 : 看懂Oracle执行计划

优化器目标 所有行					
Description	下一个操作	访问谓词	字节	基数	
SELECT STATEMENT, GOAL = ALL_ROWS			58	1	
SORT GROUP BY			58	1	
NESTED LOOPS OUTER			58	1	
TABLE ACCESS BY GLOBAL INDEX ROWID			50	1	
INDEX RANGE SCAN		"A"."MGMT_ORG_ID"=...	1	1	
TABLE ACCESS BY INDEX ROWID			8	1	
INDEX UNIQUE SCAN		"C"."ORGANIZE_ID"(...	1	1	

看懂执行计划

①：执行顺序：

根据Operation缩进来判断，缩进最多的最先执行；（缩进相同时，最上面的最先执行）

例：上图中 INDEX RANGE SCAN 和 INDEX UNIQUE SCAN 两个动作缩进最多，最上面的 INDEX RANGE SCAN 先执行；

同一级如果某个动作没有子ID就最先执行

同一级的动作执行时遵循最上最右先执行的原则

例：上图中 TABLE ACCESS BY GLOBAL INDEX ROWID 和 TABLE ACCESS BY INDEX ROWID 两个动作缩进都在同一级，则位于上面的 TABLE ACCESS BY GLOBAL INDEX ROWID 这个动作先执行；这个动作又包含一个子动作 INDEX RANGE SCAN，则位于右边的子动作 INDEX RANGE SCAN 先执行；

图示中的SQL执行顺序即为：

INDEX RANGE SCAN —> TABLE ACCESS BY GLOBAL INDEX ROWID —> INDEX UNIQUE SCAN —> TABLE ACCESS BY INDEX ROWID —> NESTED LOOPS OUTER —> SORT GROUP BY —> SELECT STATEMENT, GOAL = ALL_ROWS

（注：PLSQL提供了查看执行顺序的功能按钮(上图中的红框部分)）

②：对图中动作的一些说明：

1. 上图中 TABLE ACCESS BY ... 即描述的是该动作执行时表访问（或者说Oracle访问数据）的方式；

表访问的几种方式：（非全部）

（1）TABLE ACCESS FULL（全表扫描）：

Oracle会读取表中所有的行，并检查每一行是否满足SQL语句中的Where 限制条件；

全表扫描时可以使用多块读（即一次I/O读取多块数据块）操作，提升吞吐量；

使用建议：数据量太大的表不建议使用全表扫描，除非本身需要取出的数据较多，占到表数据总量的 5% ~ 10% 或以上

（2）TABLE ACCESS BY ROWID（通过ROWID的表存取）：

先说一下什么是ROWID？

rownumber

rowid是在伴随该行出现的值，在其生命周期内不会消失和改变，且会物理存储rowid，此处描述应该指的是rownumber

ROWID是由Oracle自动加在表中每行最后的一列伪列，既然是伪列，就说明表中并不会物理存储ROWID的值；

rownumber

你可以像使用其它列一样使用它，只是不能对该列的值进行增、删、改操作；

一旦一行数据插入后，则其对应的ROWID在该行的生命周期内是唯一的，即使发生行迁移，该行的ROWID值也不变。

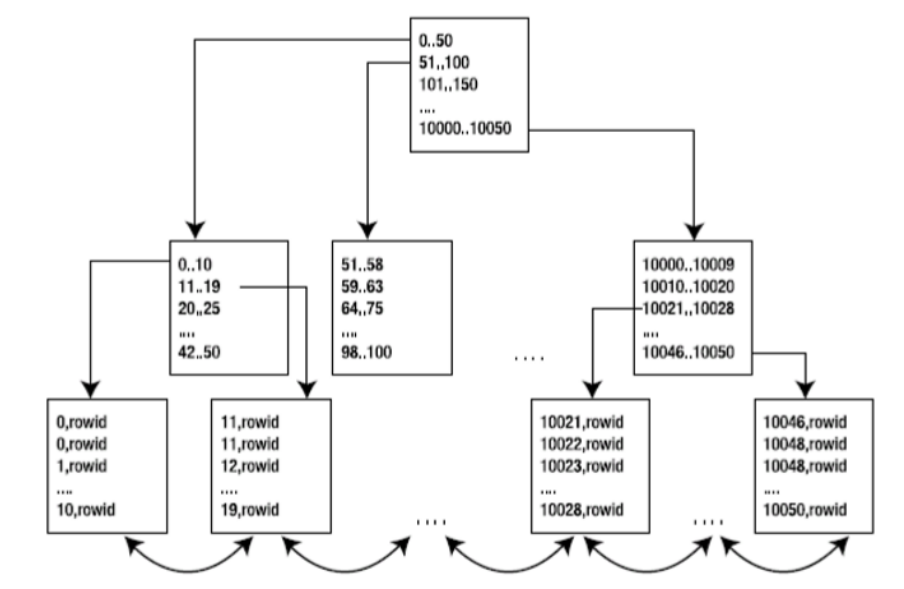
让我们再回到 TABLE ACCESS BY ROWID 来：

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可以快速定位到目标数据上，这也是Oracle中存取单行数据最快的方法；

（3）TABLE ACCESS BY INDEX SCAN（索引扫描）：

在索引块中，既存储每个索引的键值，也存储具有该键值的行的ROWID。

一个数字列上建索引后该索引可能的概念结构如下图：



所以索引扫描其实分为两步：

I：扫描索引得到对应的ROWID

II：通过ROWID定位到具体的行读取数据

-----索引扫描延伸-----

索引扫描又分五种：

a) INDEX UNIQUE SCAN (索引唯一扫描)：

针对唯一性索引 (UNIQUE INDEX) 的扫描，每次至多只返回一条记录；

表中某字段存在 UNIQUE、PRIMARY KEY 约束时，Oracle常实现唯一性扫描；

b) INDEX RANGE SCAN (索引范围扫描)：

使用一个索引存取多行数据；

发生索引范围扫描的三种情况：

c) INDEX FULL SCAN (索引全扫描)：

进行全索引扫描时，查询出的数据都必须从索引中可以直接得到（注意全索引扫描只有在CBO模式下才有效）

----- 延伸阅读：Oracle优化器简述 -----

Oracle中的优化器是SQL分析和执行的优化工具，它负责生成、制定SQL的执行计划。

Oracle的优化器有两种：

RBO：**rule-based-optimization**

RBO有严格的使用规则，只要按照这套规则去写SQL语句，无论数据表中的内容怎样，也不会影响到你的执行计划；

换句话说，RBO对数据“不敏感”，它要求SQL编写人员必须要了解各项细则；

RBO一直沿用至ORACLE 9i，从ORACLE 10g开始，RBO已经彻底被抛弃。

CBO：**cost-based-optimization**

CBO是一种比RBO更加合理、可靠的优化器，在ORACLE 10g中完全取代RBO；

CBO通过计算各种可能的执行计划的“代价”，即COST，从中选用COST最低的执行方案作为实际运行方案；

它依赖数据库对象的统计信息，统计信息的准确与否会影响CBO做出最优的选择，也就是对数据“敏感”。

d) INDEX FAST FULL SCAN (索引快速扫描)：

扫描索引中的所有数据块，与INDEX FULL SCAN类似，但是一个显著的区别是它不对查询出的数据进行排序（即数据不是以排序顺序被返回）

e) INDEX SKIP SCAN (索引跳跃扫描) :

Oracle 9i后提供, 有时候复合索引的前导列 (索引包含的第一列) 没有在查询语句中出现, Oracle也会使用该复合索引, 这时候就使用的INDEX SKIP SCAN;

什么时候会触发 INDEX SKIP SCAN 呢?

前提条件: 表有一个复合索引, 且在查询时有除了前导列 (索引中第一列) 外的其他列作为条件, 并且优化器模式为CBO时

当Oracle发现前导列的唯一值个数很少时, 会将每个唯一值都作为常规扫描的入口, 在此基础上做一次查找, 最后合并这些查询;

例如:

假设表emp有ename (雇员名称)、job (职位名)、sex (性别) 三个字段, 并且建立了如 create index idx_emp on emp (sex, ename, job) 的复合索引;

因为性别只有 '男' 和 '女' 两个值, 所以为了提高索引的利用率, Oracle可将这个复合索引拆成 ('男', ename, job), ('女', ename, job) 这两个复合索引;

当查询 select * from emp where job = 'Programmer' 时, 该查询发出后:

Oracle先进入sex为'男'的入口, 这时候使用到了 ('男', ename, job) 这条复合索引, 查找 job = 'Programmer' 的条目;

再进入sex为'女'的入口, 这时候使用到了 ('女', ename, job) 这条复合索引, 查找 job = 'Programmer' 的条目;

最后合并查询到的来自两个入口的结果集。

2. 上图中的 NESTED LOOPS ... 描述的是表连接方式;

JOIN 关键字用于将两张表作连接，一次只能连接两张表，JOIN 操作的各步骤一般是串行的（在读取做连接的两张表的数据时可以并行读取）；

表（row source）之间的连接顺序对于查询效率有很大的影响，对首先存取的表（驱动表）先应用某些限制条件（Where过滤条件）以得到一个较小的row source，可以使得连接效率提高。

-----延伸阅读：驱动表（Driving Table）与匹配表（Probed Table）-----

驱动表（Driving Table）：

表连接时首先存取的表，又称外层表（Outer Table），这个概念用于 NESTED LOOPS（嵌套循环）与 HASH JOIN（哈希连接）中；

如果驱动表返回较多的行数据，则对所有的后续操作有负面影响，故一般选择小表（应用Where限制条件后返回较少行数的表）作为驱动表。

匹配表（Probed Table）：

又称为内层表（Inner Table），从驱动表获取一行具体数据后，会到该表中寻找符合连接条件的行。故该表一般为大表（应用Where限制条件后返回较多行数的表）。

表连接的几种方式：

注：这里将首先存取的表称作 row source 1，将之后参与连接的表称作 row source 2；

（1）SORT MERGE JOIN（排序-合并连接）：

假设有查询：select a.name, b.name from table_A a join table_B b on (a.id = b.id)

内部连接过程：

a) 生成 row source 1 需要的数据，按照连接操作关联列（如示例中的 a.id）对这些数据进行排序

b) 生成 row source 2 需要的数据，按照与 a) 中对应的连接操作关联列（b.id）对数据进行排序

c) 两边已排序的行放在一起执行合并操作（对两边的数据集进行扫描并判断是否连接）

延伸：

如果示例中的连接操作关联列 a.id，b.id 之前就已经被排过序了的话，连接速度便可大大提高，因为排序是很费时间和资源的操作，尤其对于有大量数据的表。

故可以考虑在 a.id，b.id 上建立索引让其能预先排好序。不过遗憾的是，由于返回的结果集中包括所有字段，所以通常的执行计划中，即使连接列存在索引，也不会进入到执行计划中，除非进行一些特定列处理（如仅仅只查询有索引的列等）。

排序-合并连接的表无驱动顺序，谁在前面都可以；

排序-合并连接适用的连接条件有：< <= = > >=，不适用的连接条件有：<> like

（2）NESTED LOOPS（嵌套循环）：

内部连接过程：

a) 取出 row source 1 的 row 1（第一行数据），遍历 row source 2 的所有行并检查是否有匹配的，取出匹配的行放入结果集中

b) 取出 row source 1 的 row 2（第二行数据），遍历 row source 2 的所有行并检查是否有匹配的，取出匹配的行放入结果集中

c)

若 row source 1（即驱动表）中返回了 N 行数据，则 row source 2 也相应的会被全表遍历 N 次。

因为 row source 1 的每一行都会去匹配 row source 2 的所有行，所以当 row source 1 返回的行数尽可能少并且能高效访问 row source 2 (如建立适当的索引) 时，效率较高。

延伸：

嵌套循环的表有驱动顺序，注意选择合适的驱动表。

嵌套循环连接有一个其他连接方式没有的好处是：可以先返回已经连接的行，而不必等所有的连接操作处理完才返回数据，这样可以实现快速响应。

应尽可能使用限制条件 (Where过滤条件) 使驱动表 (row source 1) 返回的行数尽可能少，同时在匹配表 (row source 2) 的连接操作关联列上建立唯一索引 (UNIQUE INDEX) 或是选择性较好的非唯一索引，此时嵌套循环连接的执行效率会变得很高。若驱动表返回的行数较多，即使匹配表连接操作关联列上存在索引，连接效率也不会很高。

(3) HASH JOIN (哈希连接)：

哈希连接只适用于等值连接 (即连接条件为 =)

HASH JOIN对两个表做连接时并不一定是都进行全表扫描，其并不限制表访问方式；

内部连接过程简述：

a) 取出 row source 1 (驱动表，在HASH JOIN中又称为Build Table) 的数据集，然后将其构建成内存中的一个 Hash Table (Hash函数的Hash KEY就是连接操作关联列)，创建Hash位图 (bitmap)

b) 取出 row source 2 (匹配表) 的数据集，对其中的每一条数据的连接操作关联列使用相同的Hash函数并找到对应的 a) 里的数据在 Hash Table 中的位置，在该位置上检查能否找到匹配的数据

-----延伸阅读：Hash Table相关-----

来自Wiki的解释：

In computing, a hash table (hash map) is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

散列 (hash) 技术：在记录的存储位置和记录具有的关键字key之间建立一个对应关系 f ，使得输入key后，可以得到对应的存储位置 $f(key)$ ，这个对应关系 f 就是散列 (哈希) 函数；

采用散列技术将记录存储在一块连续的存储空间中，这块连续的存储空间就是散列表 (哈希表)；

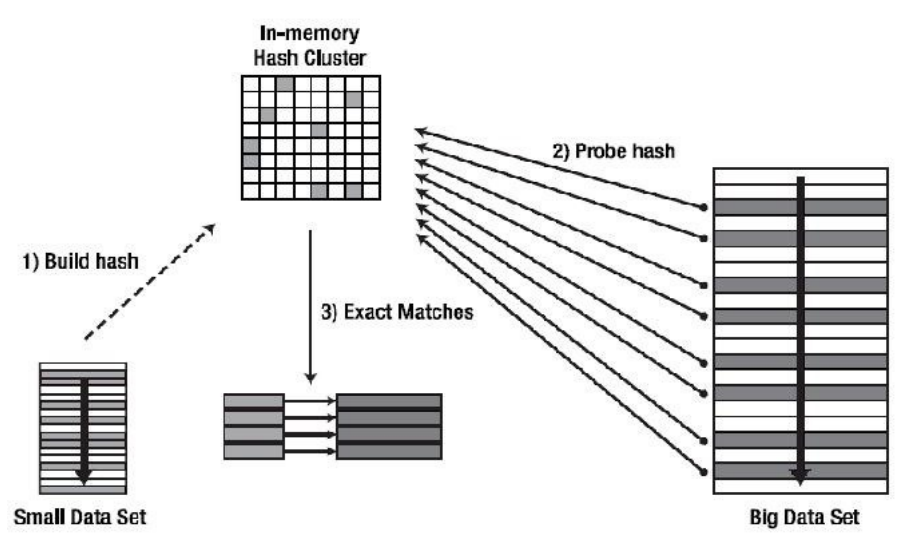
不同的key经同一散列函数散列后得到的散列值理论上应该不同，但是实际中有可能相同，相同时即是发生了散列 (哈希) 冲突，解决散列冲突的办法有很多，比如HashMap中就是用链地址法来解决哈希冲突；

哈希表是一种面向查找的数据结构，在输入给定值后查找给定值对应的记录在表中的位置以获取特定记录这个过程的速度很快。

HASH JOIN的三种模式：

1) OPTIMAL HASH JOIN：

OPTIMAL 模式是从驱动表 (也称Build Table) 上获取的结果集比较小，可以根据结果集构建的整个Hash Table都建立在用户可以使用的内存区域里。



连接过程简述：

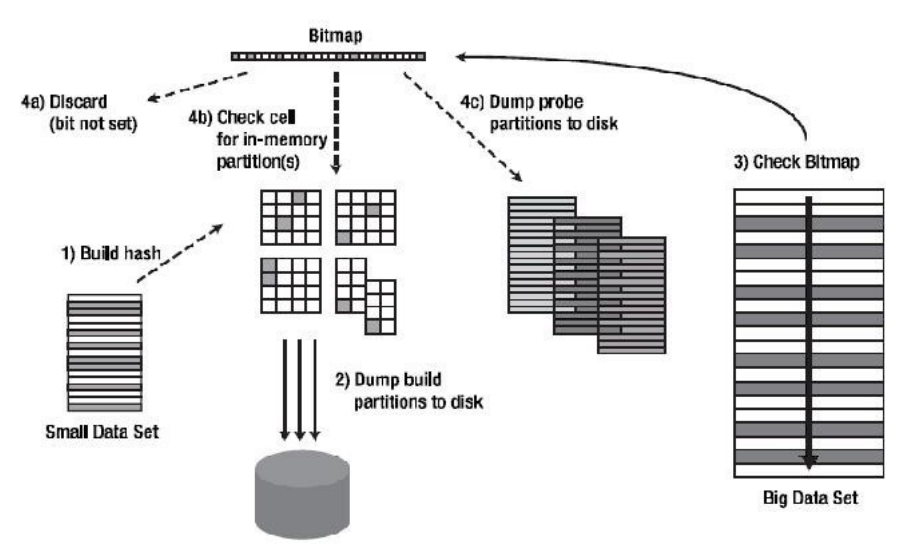
I：首先对Build Table内各行数据的连接操作关联列使用Hash函数，把Build Table的结果集构建成为内存中的Hash Table。如图所示，可以把Hash Table看作内存中的一块大的方形区域，里面有很多的小格子，Build Table里的数据就分散分布在这些小格子中，而这些小格子就是Hash Bucket（见上面Wiki的定义）。

II：开始读取匹配表（Probed Table）的数据，对其中每行数据的连接操作关联列都使用同上的Hash函数，定位Build Table里使用Hash函数后具有相同值数据所在的Hash Bucket。

III：定位到具体的Hash Bucket后，先检查Bucket里是否有数据，没有的话就马上丢掉匹配表（Probed Table）的这一行。如果里面有数据，则继续检查里面的数据（驱动表的数据）是否和匹配表的数据相匹配。

2): ONEPASS HASH JOIN：

从驱动表（也称Build Table）上获取的结果集较大，无法将根据结果集构建的Hash Table全部放入内存中时，会使用 ONEPASS 模式。



连接过程简述：

I：对Build Table内各行数据的连接操作关联列使用Hash函数，根据Build Table的结果集构建Hash Table后，由于内存无法放下所有的Hash Table内容，将导致有的Hash Bucket放在内存里，有的Hash Bucket放在磁盘上，无论放在内存里还是磁盘里，Oracle都使用一个Bitmap结构来反映这些Hash Bucket的状态（包括其位置和是否有数据）。

II：读取匹配表数据并对每行的连接操作关联列使用同上的Hash函数，定位Bitmap上Build Table里使用Hash函数后具有相同值数据所在的Bucket。如果该Bucket为空，则丢弃匹配表的这条数据。如果不为空，则需要看该Bucket是在内存里还是在磁盘上。

如果在内存中，就直接访问这个Bucket并检查其中的数据是否匹配，有匹配的话就返回这条查询结果。

如果在磁盘上，就先把这条待匹配数据放到一边，将其先暂存在内存里，等以后积累了一定量的这样的待匹配数据后，再批量的把这些数据写入到磁盘上（上图中的 Dump probe partitions to disk）。

III：当把匹配表完整的扫描了一遍后，可能已经返回了一部分匹配的数据了。接下来还有Hash Table中一部分在磁盘上的Hash Bucket数据以及匹配表中部分被写入到磁盘上的待匹配数据未处理，现在Oracle会把磁盘上的这两部分数据重新匹配一次，然后返回最终的查询结果。

3): MULTIPASS HASH JOIN：

当内存特别小或者相对而言Hash Table的数据特别大时，会使用MULTIPASS 模式。MULTIPASS会多次读取磁盘数据，应尽量避免使用该模式。

3. 上图中的 ... OUTER 描述的是表连接类型；

表连接的两种类型：

示例数据说明：

现有A、B两表，A表信息如下：

B表信息如下：

下面的例子都用A、B两表来演示。

（1）INNER JOIN（内连接）：

只返回两表中相匹配的记录。

INNER JOIN 又分为两种：

等值连接用的最多，下面以等值连接举例：

内连接的两种写法：

I : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a inner join B b on (a.id = b.id)

II : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a join B b on (a.id = b.id)

连接时只返回满足连接条件 (a.id = b.id) 的记录：

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美

(2) OUTER JOIN (外连接)：

OUTER JOIN 分为三种：

a) LEFT JOIN (左连接)：

返回的结果不仅包含符合连接条件的记录，还包含左边表中的全部记录。
(若返回的左表中某行记录在右表中没有匹配项，则右表中的返回列均为空值)

两种写法：

I : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a left outer join B b on (a.id = b.id)

II : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a left join B b on (a.id = b.id)

返回结果：

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美
4	小刘		
1	小明		

b) **RIGHT JOIN (右连接)** :

返回的结果不仅包含符合连接条件的记录，还包含右边表中的全部记录。
(若返回的右表中某行记录在左表中没有匹配项，则左表中的返回列均为空值)

两种写法 :

I : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a right outer join B b on (a.id = b.id)

II : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a right join B b on (a.id = b.id)

返回结果 :

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美
		6	小丽
		5	小琳

c) **FULL JOIN (全连接)** :

返回左右两表的全部记录。(左右两边不匹配的项都以空值代替)

两种写法 :

I : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a full outer join B b on (a.id = b.id)

II : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a full join B b on (a.id = b.id)

返回结果 :

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美
		5	小琳
		6	小丽
4	小刘		
1	小明		

-----延伸阅读：(+) 操作符-----

(+) 操作符是Oracle特有的表示法，用来表示外连接（只能表示 左外、右外连接），需要配合Where语句使用。

特别注意：(+) 操作符在左表的连接条件上表示右连接，在右表的连接条件上表示左连接。

如：

I : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a, B b where a.id = b.id(+)

查询结果：

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美
4	小刘		
1	小明		

右边()

实际与左连接 select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a left join B b on (a.id = b.id) 效果等价

II : select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a, B b where a.id(+) = b.id

查询结果：

A_ID	A_NAME	B_ID	B_NAME
2	小李	2	小新
3	小红	3	小美
		6	小丽
		5	小琳

左边()

实际与右连接 select a.id A_ID, a.name A_NAME, b.id B_ID, b.name B_NAME
from A a right join B b on (a.id = b.id) 效果等价

补充：

自连接（通过给一个表赋两个不同的别名让其与自身内连或外连接）