

星型转换

——Oracle 优化器

中国 Oracle 用户组

作者：刘相兵 (Maclean Liu)

<http://www.acoug.org>

版本	发布时间
1.0	2011/1/30

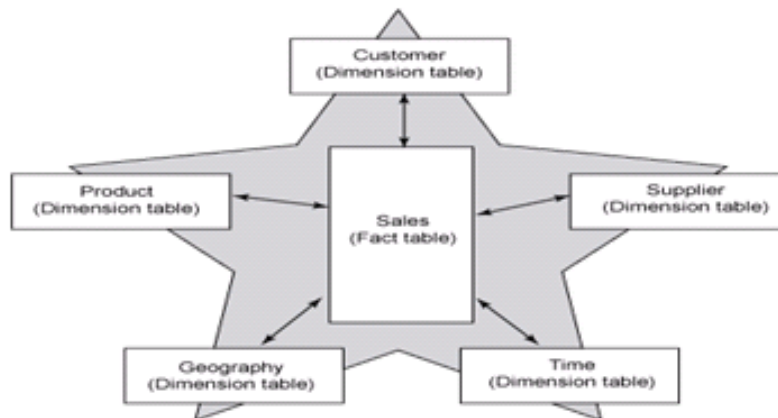
目录

1	星型转换	- 3 -
2	省略重复连接	- 11 -
3	临时表转换	- 12 -
4	如何启用星型查询	- 16 -
5	总结	- 16 -
6	作者介绍	- 17 -

摘要: 星型转换可以尽量避免直接去扫描星型模式中的事实表，从而达到减少物理读、提高效率的目的。本文通过一个例子说明了星型转换的优势，除此之外还介绍了如何启用星型查询，省略重复连接，临时表转换等。

1 星型转换

Oracle 8i 中引入了星型转换(star transformation)的优化器新特性以便更有效地处理星型查询。星型查询语句多用于基于星型模型设计的数据仓库应用中。星型模型的称谓源于该种模型以图形化表现时看起来形似一颗海星。这颗星的中央会由一个或多个事实表(fact tables)组成，而各个触角上则分布着多个维度表(dimension tables)，如下图：



星型转换的基本思路是尽量避免直接去扫描星型模式中的事实表，因为这些事实表总会因为存有大量数据而十分庞大，对这些表的全表扫描会引起大量物理读并且效率低下。在典型的星型查询中，事实表总是会与多个与之相比小得多的维度表发生连接(join)操作。典型的事实表针对每一个维度表会存在一个外键 (foreign key)，除去这些键值(key)外还会存在一些度量字段譬如销售额(sales amount)。与之对应的键值(key)在维度表上扮演主键的角色。而事实表与维度表间的连接操作一般都会发生在事实表上的外键和与之对应的维度表的主键间。同时这类查询总是会在维度表的其他列上存在限制十分严格的过滤谓词。充分结合这些维度表上的过滤谓词可以有效减少需要从事实表上访问的数据集合。这也就是星型转换(star transformation)的根本目的，仅访问事实上相关的、过滤后精简的数据集合。

Oracle 在 Sample Schema 示例模式中就存有星型模型的 Schema，譬如 SH：

```
SQL> select * from v$version;
```

```
BANNER
```

```
-----  
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
```

```
PL/SQL Release 11.2.0.1.0 - Production
```

```
CORE 11.2.0.1.0 Production
```

```
TNS for 32-bit Windows: Version 11.2.0.1.0 - Production
```

```
NLSRTL Version 11.2.0.1.0 - Production
```

```
SQL> select * from global_name;
```

```
GLOBAL_NAME
```

```
-----  
www.oracledatabase12g.com
```

```
SQL> conn maclean/maclean
```

```
Connected.
```

```
SQL> select table_name, comments
```

```
2   from dba_tab_comments
```

```
3   where owner = 'SH'
```

```
4   and table_name in ('SALES', 'CUSTOMERS', 'CHANNELS', 'TIMES');
```

```
TABLE_NAME
```

```
COMMENTS
```

```
-----  
CHANNELS
```

```
small dimension table
```

```
CUSTOMERS
```

```
dimension table
```

```
SALES
```

```
facts table, without a primary key; all rows are uniquely identified by the comb
```

```
TIMES
```

```
Time dimension table to support multiple hierarchies and materialized views
```

可以从以上各表的注释(comment)中看到, SALES 表是 SH 模式下一个没有主键的事实表, 而 CHANNELS、CUSTOMERS、TIMES 三个小表充当维度表的角色。我们试着构建以下星型查询语句, 该查询用以检索出从 1999 年 12 月至 2000 年 2 月间 Florida 州所有城市直销形式的每月销售额。

```
SQL> col name for a35
```

```
SQL> col description for a45
```

```
SQL> col value for a8
```

```
SQL> select name,value,description from v$system_parameter where name='star_transformation_enabled';
```

```
NAME
```

```
VALUE
```

```
DESCRIPTION
```

```
-----  
star_transformation_enabled
```

```
FALSE
```

```
enable the use of star transformation
```

```
/* 初始化参数 star_transformation_enabled 用以控制如何启用星型转换,
```

默认为 FALSE，该参数可以动态修改*/

```
SELECT c.cust_city,
       t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sh.sales s, sh.times t, sh.customers c, sh.channels ch
WHERE s.time_id = t.time_id
      AND s.cust_id = c.cust_id
      AND s.channel_id = ch.channel_id
      AND c.cust_state_province = 'FL'
      AND ch.channel_desc = 'Direct Sales'
      AND t.calendar_quarter_desc IN ('2000-01', '2000-02', '1999-12')
GROUP BY c.cust_city, t.calendar_quarter_desc;
```

```
SQL> select * from table(dbms_xplan.display_cursor(format => 'IOSTATS'));
```

PLAN_TABLE_OUTPUT

SQL_ID ddjm7k72b8p2a, child number 1

```
SELECT /*+ gather_plan_statistics */ c.cust_city,
       t.calendar_quarter_desc,          SUM(s.amount_sold) sales_amount FROM
sh.sales s, sh.times t, sh.customers c, sh.channels ch WHERE s.time_id
= t.time_id AND s.cust_id = c.cust_id AND s.channel_id =
ch.channel_id AND c.cust_state_province = 'FL' AND
ch.channel_desc = 'Direct Sales' AND t.calendar_quarter_desc IN
('2000-01', '2000-02', '1999-12') GROUP BY c.cust_city, t.calendar_quarter_desc
```

Plan hash value: 382868716

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1		24	00:00:00.62	1735	1726

1	HASH GROUP BY		1	24	24	00:00:00.62	1735	1726
* 2	HASH JOIN		1	1580	6015	00:00:00.42	1735	1726
* 3	TABLE ACCESS FULL	CUSTOMERS	1	2438	2438	00:00:01.73	1459	1455
* 4	HASH JOIN		1	4575	74631	00:00:00.18	276	271
5	PART JOIN FILTER CREATE	:BF0000	1	227	182	00:00:00.04	59	60
6	MERGE JOIN CARTESIAN		1	227	182	00:00:00.04	59	60
* 7	TABLE ACCESS FULL	CHANNELS	1	1	1	00:00:00.01	3	6
8	BUFFER SORT		1	227	182	00:00:00.02	56	54
* 9	TABLE ACCESS FULL	TIMES	1	227	182	00:00:00.02	56	54
10	PARTITION RANGE JOIN-FILTER		1	117K	117K	00:00:00.09	217	211
11	TABLE ACCESS FULL	SALES	2	117K	117K	00:00:00.07	217	211

可以看到在以上不使用星型转换的执行计划中对事实表 **SALES** 执行了全表扫描,这是我们不希望发生的。因为 **SALES** 表中每一行记录都对应于一笔销售记录,因此其可能包含数百万行记录。但实际上这其中仅有极小部分是我们在查询中指定的季度在佛罗里达州直销的纪录。若我们启用星型转换,执行计划是否有所改善?

```
SQL> alter session set star_transformation_enabled=temp_disable;
```

```
Session altered.
```

```
SQL> alter session set events '10053 trace name context forever,level 1';
```

```
Session altered.
```

在我们的理想当中星型变化会将原查询语句转换成如下形式:

```
SELECT c.cust_city,
       t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount
FROM sh.sales s, sh.times t, sh.customers c
WHERE s.time_id = t.time_id
      AND s.cust_id = c.cust_id
      AND c.cust_state_province = 'FL'
      AND t.calendar_quarter_desc IN ('2000-01', '2000-02', '1999-12')
      AND s.time_id IN
      (SELECT time_id
       FROM sh.times
       WHERE calendar_quarter_desc IN ('2000-01', '2000-02', '1999-12'))
      AND s.cust_id IN
```

```
(SELECT cust_id FROM sh.customers WHERE cust_state_province = 'FL')  
  
AND s.channel_id IN  
  
(SELECT channel_id  
  
FROM sh.channels  
  
WHERE channel_desc = 'Direct Sales')  
  
GROUP BY c.cust_city, t.calendar_quarter_desc;
```

/* 以添加 AND..IN 的形式明确了利用组合过滤谓词来减少需要处理的数据集 */

通过 10053 优化 trace 我们可以了解 Oracle 优化器是如何真正产生这部分过度谓词的:

```
FPD: Considering simple filter push in query block SEL$C3AF6D21 (#1)  
"S"."CHANNEL_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "CH"."CHANNEL_ID" FROM "SH"."CHANNELS" "CH")  
AND "S"."CUST_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "C"."CUST_ID" FROM "SH"."CUSTOMERS" "C") AND  
"S"."TIME_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "T"."TIME_ID"  
  
FPD: Considering simple filter push in query block SEL$ACF30367 (#4)  
"T"."CALENDAR_QUARTER_DESC"=' 2000-01' OR "T"."CALENDAR_QUARTER_DESC"=' 2000-02' OR  
"T"."CALENDAR_QUARTER_DESC"=' 1999-12'  
  
try to generate transitive predicate from check constraints for query block SEL$ACF30367 (#4)  
finally: "T"."CALENDAR_QUARTER_DESC"=' 2000-01' OR "T"."CALENDAR_QUARTER_DESC"=' 2000-02' OR  
"T"."CALENDAR_QUARTER_DESC"=' 1999-12'  
  
FPD: Considering simple filter push in query block SEL$F6045C7B (#3)  
"C"."CUST_STATE_PROVINCE"=' FL'  
  
try to generate transitive predicate from check constraints for query block SEL$F6045C7B (#3)  
finally: "C"."CUST_STATE_PROVINCE"=' FL'  
  
FPD: Considering simple filter push in query block SEL$6EE793B7 (#2)  
"CH"."CHANNEL_DESC"=' Direct Sales'  
  
try to generate transitive predicate from check constraints for query block SEL$6EE793B7 (#2)  
finally: "CH"."CHANNEL_DESC"=' Direct Sales'  
  
try to generate transitive predicate from check constraints for query block SEL$C3AF6D21 (#1)  
finally: "S"."CHANNEL_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "CH"."CHANNEL_ID" FROM "SH"."CHANNELS" "CH")  
AND "S"."CUST_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "C"."CUST_ID" FROM "SH"."CUSTOMERS" "C")  
AND "S"."TIME_ID"=ANY (SELECT /*+ SEMIJOIN_DRIVER */ "T"."TIME_ID
```

Final query after transformations:***** UNPARSED QUERY IS *****

最终转换后的查询语句:

```
SELECT "C"."CUST_CITY" "CUST_CITY",
       "T"."CALENDAR_QUARTER_DESC" "CALENDAR_QUARTER_DESC",
       SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S", "SH"."TIMES" "T", "SH"."CUSTOMERS" "C"
WHERE "S"."CHANNEL_ID" = ANY (SELECT /*+ SEMIJOIN_DRIVER */
                              "CH"."CHANNEL_ID" "ITEM_1"
                              FROM "SH"."CHANNELS" "CH"
                              WHERE "CH"."CHANNEL_DESC" = 'Direct Sales')
AND "S"."CUST_ID" = ANY (SELECT /*+ SEMIJOIN_DRIVER */
                          "C"."CUST_ID" "ITEM_1"
                          FROM "SH"."CUSTOMERS" "C"
                          WHERE "C"."CUST_STATE_PROVINCE" = 'FL')
AND "S"."TIME_ID" = ANY
(SELECT /*+ SEMIJOIN_DRIVER */
     "T"."TIME_ID" "ITEM_1"
FROM "SH"."TIMES" "T"
     WHERE "T"."CALENDAR_QUARTER_DESC" = '2000-01'
     OR "T"."CALENDAR_QUARTER_DESC" = '2000-02'
     OR "T"."CALENDAR_QUARTER_DESC" = '1999-12')
AND "S"."TIME_ID" = "T"."TIME_ID"
AND "S"."CUST_ID" = "C"."CUST_ID"
AND "C"."CUST_STATE_PROVINCE" = 'FL'
AND ("T"."CALENDAR_QUARTER_DESC" = '2000-01' OR
     "T"."CALENDAR_QUARTER_DESC" = '2000-02' OR
     "T"."CALENDAR_QUARTER_DESC" = '1999-12')
GROUP BY "C"."CUST_CITY", "T"."CALENDAR_QUARTER_DESC"

/* 要比我们想想的复杂一些，子查询将 IN 语句化解了，
   并且以 AND...ANY 的形式追加了过度谓词条件
*/
```

```

-----+
| Id | Operation                               | Name           | Rows | Bytes | Cost | Time      | Pstart |
Pstop |
-----+-----+
| 0  | SELECT STATEMENT                       |                |      |      | 1710 |          |        |
|
| 1  | HASH GROUP BY                          |                | 1254 | 77K   | 1710 | 00:00:21 |        |
|
| 2  | HASH JOIN                               |                | 1254 | 77K   | 1283 | 00:00:16 |        |
|
| 3  | HASH JOIN                               |                | 1254 | 45K   | 877  | 00:00:11 |        |
|
| 4  | TABLE ACCESS FULL                     | TIMES          | 227  | 3632  | 18   | 00:00:01 |        |
|
| 5  | PARTITION RANGE SUBQUERY              |                | 1254 | 26K   | 858  | 00:00:11 |        |
KEY(SUBQUERY) | KEY(SUBQUERY) |
| 6  | TABLE ACCESS BY LOCAL INDEX ROWID    | SALES          | 1254 | 26K   | 858  | 00:00:11 |        |
KEY(SUBQUERY) | KEY(SUBQUERY) |
| 7  | BITMAP CONVERSION TO ROWIDS           |                |      |      |      |          |        |
|
| 8  | BITMAP AND                             |                |      |      |      |          |        |
|
| 9  | BITMAP MERGE                           |                |      |      |      |          |        |
|
| 10 | BITMAP KEY ITERATION                   |                |      |      |      |          |        |
| 11 | BUFFER SORT                            |                |      |      |      |          |        |
|
| 12 | TABLE ACCESS FULL                     | CHANNELS       | 1    | 13    | 3    | 00:00:01 |        |
|
| 13 | BITMAP INDEX RANGE SCAN                | SALES_CHANNEL_BIX |      |      |      |          |        |
KEY(SUBQUERY) | KEY(SUBQUERY) |
| 14 | BITMAP MERGE                           |                |      |      |      |          |        |
|

```

15	BITMAP KEY ITERATION							
16	BUFFER SORT							
17	TABLE ACCESS FULL	TIMES	227	3632	18	00:00:01		
18	BITMAP INDEX RANGE SCAN	SALES_TIME_BIX						
KEY(SUBQUERY)		KEY(SUBQUERY)						
19	BITMAP MERGE							
20	BITMAP KEY ITERATION							
21	BUFFER SORT							
22	TABLE ACCESS FULL	CUSTOMERS	2438	38K	406	00:00:05		
23	BITMAP INDEX RANGE SCAN	SALES_CUST_BIX						
KEY(SUBQUERY)		KEY(SUBQUERY)						
24	TABLE ACCESS FULL	CUSTOMERS	2438	62K	406	00:00:05		

-----+

Predicate Information:

```

2 - access("S"."CUST_ID"="C"."CUST_ID")
3 - access("S"."TIME_ID"="T"."TIME_ID")
4 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-12' OR "T"."CALENDAR_QUARTER_DESC"='2000-01' OR
"T"."CALENDAR_QUARTER_DESC"='2000-02'))
12 - filter("CH"."CHANNEL_DESC"='Direct Sales')
13 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
17 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-12' OR "T"."CALENDAR_QUARTER_DESC"='2000-01' OR
"T"."CALENDAR_QUARTER_DESC"='2000-02'))
18 - access("S"."TIME_ID"="T"."TIME_ID")
22 - filter("C"."CUST_STATE_PROVINCE"='FL')

```

```
23 - access("S"."CUST_ID"="C"."CUST_ID")
24 - filter("C"."CUST_STATE_PROVINCE"='FL')
```

从以上演示中可以看到，星型转换添加了必要的对应于维度表约束的子查询谓词。这些子查询谓词又被称为位图半连接谓词(bitmap semi-join predicates，见 SEMIJOIN_DRIVER hint)。通过迭代来自于子查询的键值，再通过位图(bitmap)的 AND、OR 操作(这些位图可以源于位图索引 bitmap index，但也可以取自普通的 B*tree 索引)，我们可以做到仅仅访问事实表上的查询相关记录。理想状况下维度表上的过滤谓词可以帮我们过滤掉大量的数据，这样就可以使执行计划效率大大提升。当我们获取到事实表上的相关行后，这部分结果集可能仍需要同维度表使用原谓词重复连接(join back)。某些情况下，重复连接可以被省略，之后我们会提到。

如上演示中列出了星型转换后的查询语句的执行计划。这里可以看到 Oracle 是使用“TABLE ACCESS BY LOCAL INDEX ROWID”形式访问 SALES 事实表的，而非全表扫描。这里我们仅关心 7-23 行的执行计划，服务进程分别在(12,17,22)行从维度表中取得各维度表的相关键值(key value)，同时对部分结果集执行了 BUFFER SORT 操作;在(13,18,23)行的‘bitmap index range scan’操作中服务进程从事事实表的三个对应于维度表外键的位图索引上 (SALES_CHANNEL_BIX,SALES_TIME_BIX,SALES_CUST_BIX)获取了最原始的位图。位图上的每一个 bit 都对应于事实表上的一行记录。若从子查询中获取的键值(key values)与事实表上的值一致则 bit 置为 1，否则为 0。举例而言位图 bitmap:[1][0][1][1][0][0][0]..[0](之后都为零)表示事实表上仅有第一、三、四行匹配于由子查询提供的键值。我们假设以上位图是由 times 表子查询提供的众多键值中的一个(如‘2000-01’)的对应于事实表的位图表达式。

接着在执行计划的(10,15,20)行上的‘bitmap key iteration’操作会迭代每一个由子查询提供的键值并获取相应的位图。我们假设 times 表子查询提供的另外 2 个键值‘2000-02’和‘1999-12’分别对应的位图为 [0][0][0][0][0][1]..[0]和 [0][0][0][0][1][0]...[0]即每键值都只有一行符合。

毫无疑问 ITERATION 迭代操作会为我们生成众多位图，接下来需要对这些不同键值对应的位图进行位图合并操作(BITMAP MERGE，相当于对位图做 OR 操作)，可以看到在上例执行计划中为(9,14,19)行；以我们假设的 times 表子查询位图合并而言，会生产一个简单的位图 [1][0][1][1][1][1][0][0]...[0]，这个位图对应事实表上的第一、三、四、五、六行，是对‘2000-01’，‘2000-02’，‘1999-12’三个键值对应位图的合并。

在获得最后位图前我们还需要对来自于三个子查询的位图进一步处理，因为原始查询语句中各约束条件是 AND 与的形式，因此我们还要对这些已合并的位图 执行 AND 与操作，如执行计划中的第八行“BITMAP AND”，因为是 AND 与操作所以这步又会过滤掉大量记录。我们假设最终获得的位图是 [1][0][1][0]...[0]，即仅有第一、三行。

通过最终 bitmap 位图 Oracle 可以极高效地生成事实表的 ROWID，此步骤表现为第七行的“BITMAP CONVERSION TO ROWIDS”，我们使用这些 ROWID 来访问事实表取得少量的“绝对”相关记录。以我们的假设而言最终位图仅有 2 位为 1，只需要用这 2 行的 ROWID 从事事实表上直接 fetch2 条记录即可，从而避免了低效的全表扫描。

2 省略重复连接

因为子查询及位图树只是通过维度表上的过滤条件为事实表过滤掉大量的数据，所以从事事实表上获取的相关数据仍可能需要重复一次和维度表的连接。省略重复连接的前提是维度表上所有的谓词都是半连接谓词子查询的一部分，And 由子查询检索到的列均唯一(unique) And 维度表的列不被 select 或 group by 涉及。在上例中无需对 CHANNELS 表再次连接的理由是没有 select(或 group by)CHANNEL 表上的列，且 channel_id 列是唯一的。

3 临时表转换

若在已知星型转换中重复连接维度表无法被省略的话，Oracle 可以将对维度表的子查询结果集存储到内存中的全局临时表(global temporary table)上以避免重复扫描维度表。此外，因为将子查询的结果集物化了，故而若使用并行查询则每个并行子进程(slave)可以直接从物化结果集的临时表中获得数据，而不需要反复执行子查询。

试看以下示例，了解 Oracle 是如何利用物化临时表避免反复连接的：

```
SQL> alter session set star_transformation_enabled=true;
Session altered.

SQL> alter session set events '10053 trace name context forever,level 1';
Session altered.

SELECT "T1"."C1" "CUST_CITY",
       "T"."CALENDAR_QUARTER_DESC" "CALENDAR_QUARTER_DESC",
       SUM("S"."AMOUNT_SOLD") "SALES_AMOUNT"
FROM "SH"."SALES" "S",
     "SH"."TIMES" "T",
     "SYS"."SYS_TEMP_OFD9D660E_1DF5D6" "T1"
WHERE "S"."CUST_ID" = ANY (SELECT /*+ SEMIJOIN_DRIVER CACHE_TEMP_TABLE ("T1") */
                          "T1"."C0" "C0"
                          FROM "SYS"."SYS_TEMP_OFD9D660E_1DF5D6" "T1")
AND "S"."CHANNEL_ID" = ANY
(SELECT /*+ SEMIJOIN_DRIVER */
     "CH"."CHANNEL_ID" "ITEM_1"
     FROM "SH"."CHANNELS" "CH"
     WHERE "CH"."CHANNEL_DESC" = 'Direct Sales')
AND "S"."TIME_ID" = ANY
(SELECT /*+ SEMIJOIN_DRIVER */
     "T"."TIME_ID" "ITEM_1"
     FROM "SH"."TIMES" "T"
     WHERE "T"."CALENDAR_QUARTER_DESC" = '2000-01'
           OR "T"."CALENDAR_QUARTER_DESC" = '2000-02'
           OR "T"."CALENDAR_QUARTER_DESC" = '1999-12')
```

```
AND "S"."TIME_ID" = "T"."TIME_ID"
AND "S"."CUST_ID" = "T1"."C0"
AND ("T"."CALENDAR_QUARTER_DESC" = '2000-01' OR
      "T"."CALENDAR_QUARTER_DESC" = '2000-02' OR
      "T"."CALENDAR_QUARTER_DESC" = '1999-12')
GROUP BY "T1"."C1", "T"."CALENDAR_QUARTER_DESC"
```

以上为启用临时表后的星型转换后的查询语句，相应的执行计划如下：

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT				911	
1	TEMP TABLE TRANSFORMATION					
2	LOAD AS SELECT					
3	TABLE ACCESS FULL	CUSTOMERS	2438	62K	406	00:00:05
4	HASH GROUP BY		1254	64K	506	00:00:07
5	HASH JOIN		1254	64K	479	00:00:06
6	HASH JOIN		1254	45K	475	00:00:06
7	TABLE ACCESS FULL	TIMES	227	3632	18	00:00:01
8	PARTITION RANGE SUBQUERY		1254	26K	456	00:00:06
9	TABLE ACCESS BY LOCAL INDEX ROWID	SALES	1254	26K	456	00:00:06
10	BITMAP CONVERSION TO ROWIDS					

11	BITMAP AND						
12	BITMAP MERGE						
13	BITMAP KEY ITERATION						
14	BUFFER SORT						
15	TABLE ACCESS FULL	CHANNELS	1	13	3	00:00:01	
16	BITMAP INDEX RANGE SCAN	SALES_CHANNEL_BIX					
	KEY(SUBQUERY)	KEY(SUBQUERY)					
17	BITMAP MERGE						
18	BITMAP KEY ITERATION						
19	BUFFER SORT						
20	TABLE ACCESS FULL	TIMES	227	3632	18	00:00:01	
21	BITMAP INDEX RANGE SCAN	SALES_TIME_BIX					
	KEY(SUBQUERY)	KEY(SUBQUERY)					
22	BITMAP MERGE						
23	BITMAP KEY ITERATION						
24	BUFFER SORT						
25	TABLE ACCESS FULL	SYS_TEMP_0FD9D660E_1DF5D6	2438	12K	4	00:00:01	
26	BITMAP INDEX RANGE SCAN	SALES_CUST_BIX					
	KEY(SUBQUERY)	KEY(SUBQUERY)					
27	TABLE ACCESS FULL	SYS_TEMP_0FD9D660E_1DF5D6	2438	36K	4	00:00:01	

Predicate Information:

```

3 - filter("C"."CUST_STATE_PROVINCE"=' FL' )
5 - access("S"."CUST_ID"="C0")
6 - access("S"."TIME_ID"="T"."TIME_ID")
7 - filter(("T"."CALENDAR_QUARTER_DESC"=' 1999-12' OR "T"."CALENDAR_QUARTER_DESC"=' 2000-01' OR
"T"."CALENDAR_QUARTER_DESC"=' 2000-02' ))
15 - filter("CH"."CHANNEL_DESC"=' Direct Sales' )
16 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
20 - filter(("T"."CALENDAR_QUARTER_DESC"=' 1999-12' OR "T"."CALENDAR_QUARTER_DESC"=' 2000-01' OR
"T"."CALENDAR_QUARTER_DESC"=' 2000-02' ))
21 - access("S"."TIME_ID"="T"."TIME_ID")
26 - access("S"."CUST_ID"="C0")

```

从以上 trace 中可以看到系统命名的临时表 SYS_TEMP_0FD9D660E_1DF5D6 缓存 CUSTOMERS 表, 之后原先 CUSTOMERS 表被 SYS_TEMP_0FD9D660E_1DF5D6 所取代, 原 CUSTOMERS 表上的 cust_id 和 cust_city 列均被替换为别名为 T1 的临时表的 C0 和 C1 列。实际上该临时表也仅需要这 2 列即可满足计划的需求, 所以该临时表以如下查询语句填充:

```

ST: Subquery text:***** UNPARSED QUERY IS *****
SELECT "C"."CUST_ID" "ITEM_1", "C"."CUST_CITY" "ITEM_2" FROM "SH"."CUSTOMERS" "C" WHERE
"C"."CUST_STATE_PROVINCE"=' FL'
Copy query block qb# -1 () : SELECT /*+ CACHE_TEMP_TABLE(T1) */ "C0" FROM "SYS"."SYS_TEMP_0FD9D660E_1DF5D6" T1
ST: Subquery (temp table) text:***** UNPARSED QUERY IS *****
SELECT /*+ CACHE_TEMP_TABLE ("T1") */ "T1"."C0" "C0" FROM "SYS"."SYS_TEMP_0FD9D660E_1DF5D6" "T1"
Copy query block qb# -1 () : SELECT /*+ CACHE_TEMP_TABLE(T1) */ "C0", "C1" FROM "SYS"."SYS_TEMP_0FD9D660E_1DF5D6"
T1
ST: Join back qbc text:***** UNPARSED QUERY IS *****
SELECT /*+ CACHE_TEMP_TABLE ("T1") */ "T1"."C0" "C0", "T1"."C1" "C1" FROM "SYS"."SYS_TEMP_0FD9D660E_1DF5D6" "T1"

```

可以从以上执行计划中看到第一、二、三行的“TEMP TABLE TRANSFORMATION LOAD AS SELECT TABLE ACCESS FULL CUSTOMERS”看到 Oracle 是如何将子查询物化为临时表的。在第 25 行, Oracle 直接以该临时表替代了子查询来构建我们所需要的位图。到第 27 行 Oracle 直接利用该临时表来重复连接, 避免再次扫描 customers 表。因为我们在构建临时表时已经使用谓词条件(如上面的红字语句), 故而我们无需对临时表再次过滤。

4 如何启用星型查询

星型转换由初始化参数 `star_transformation_enabled` 控制，该参数可以有三种选项：

- ✚ TRUE: Oracle 优化器自动识别语句中的事实表和约束维度表并进行星型转换。这一切优化尝试都在 CBO 的藩篱内，优化器需要确定转换后的执行计划成本要低于不转换的执行计划；同时优化器还会尝试利用物化的临时表，如果那样真的好的话。
- ✚ False: 优化器不会考虑星型转换。
- ✚ TEMP_DISABLE: 当一个维度表超过 100 个块时，如果简单地设置 `star_transformation_enabled` 为 TRUE 来启用星型变换，那么会话会创建一个内存中的全局临时表(global temporary table)来保存已过滤的维度数据，这在过去会造成很多问题；这里说的 100 个块其实是隐式参数 `_temp_tran_block_threshold`(number of blocks for a dimension before we temp transform)的默认值，此外隐式参数 `_temp_tran_cache`(determines if temp table is created with cache option, 默认为 TRUE)决定了这类临时表是否被缓存住；为了避免创建全局临时表可能带来的问题，就可以用到 TEMP_DISABLE 这个禁用临时表的选项，让优化器不再考虑使用物化的临时表。

默认该参数为 False，若要问这是为什么？因为星型转换适用的场景是数据仓库环境中具有星型模型的模式，而且需要事实表的各个连接列上均有良好的索引时才能发挥其优势。如果能确定以上因素，那么我们可以放心的使用星型转换了，把 `star_transformation_enabled` 改为 true 或 temp_disable 吧！

5 总结

星型转换可以有效改善大的事实表与多个具有良好选择率的维度表间连接的查询。星型转换有效避免了全表扫描的性能窘境。它只 fetch 那些事实表上的“绝对”相关行。同时星型转换是基于 CBO 优化器的，Oracle 能很好地认清使用该种转换是否有利。一旦维度表上的过滤无法有效减少需要从事实上处理的数据集和时，那么可能全表扫描相对而言更为恰当。

以上我们力图通过一些简单的查询和执行计划来诠释星型转换的基本理念，但现实生产环境中实际的查询语句可能要复杂的多；举例而言如果查询涉及星型模型中的多个事实表的话，那么其复杂度就大幅提高了；如何正确构建事实表上的索引，收集相关列上的柱状图信息，在 Oracle 优化器无法正确判断的情况下循循善诱，都是大型数据仓库环境中 DBA 所面临的难题。

6 作者介绍



刘相兵，网名:Maclean Liu

Oracle 10g OCM

曾供职于某通信制造业巨头从事数据库维护工作。

目前供职于国内某 Oracle 第三方增值服务公司，负责 Oracle 高级技术的咨询顾问工作。

5 年以上从事维护数据库，对大型数据库的调优和诊断具有丰富的经验。

2011 年 1 月有幸成为 ACOUG 一员。

个人博客：<http://www.oracle-database12g.com>

Email&Gtalk：liu.maclean@gmail.com

QQ: 47079569