一次分析的全过程

作者介绍：

盖国强，曾任 ITPUB MS 版版主，现任 Oracle 数据库管理版版主。曾任职于某国家大型企业，服务于烟草行业，开发过基于 Oracle 数据库的大型 ERP 系统，属国家信息产业部重点工程。同时负责 Oracle 数据库管理及优化，并为多家烟草企业提供 Oracle 数据库管理、优化及技术支持。目前任职于北京某电信增值业务系统提供商企业，负责数据库业务。管理全国 30 多个省点数据库平台。 实践经验丰富，长于数据库诊断、优化与 SQL 调整。希望与大家共同学习提高 Oracle 技术水平。

Mail: eygle@itpub.net

以下是一次 SQL 优化分析的全过程，曾经在 itpub 上发过相关的帖子，现在整理了一下，添加了详细的说明，希望对大家有些帮助。

环境说明:
数据库版本:Oracle8.1.7.4
平台:Hp-Ux11i

以下是从程序员处获得的一段代码，开发人员抱怨这段代码执行缓慢，我执行该段代码获得执行计划，分析如下：

这是该段代码的执行计划及统计数据：

```
SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
 10 "SP_TRANS"."PAY_MODE",
 11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
 12 0 as PAY_THIS
 13 FROM "SP_ITEM",
 14 "SP_TRANS_SUB",
 15 "SP_TRANS"
 16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
 17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
 18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
 19 /

8 rows selected.
```

```
Elapsed: 00: 00: 00.51

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE
   1 0 NESTED LOOPS
   2 1 NESTED LOOPS
   3 2 TABLE ACCESS (FULL) OF 'SP_TRANS'
   4 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB'
   5 4 INDEX (RANGE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE)
   6 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM'
   7 6 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)

Statistics
----------------------------------------------------------
          0 recursive calls
          4 db block gets
        323 consistent gets
          0 physical reads
          0 redo size
       1809 bytes sent via SQL*Net to client
        425 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          0 sorts (memory)
          0 sorts (disk)
          8 rows processed
```

该段代码用以按供货商查询 2003 年开始的处理单据（当时的数据量很少），查询时间大约是 0.51 秒。

此前这几个表都没有分析过，数据库选择的是 RBO 优化器。

为了加快代码的执行，analyze 相关表：

```
SQL> analyze table sp_trans_sub compute statistics;

Table analyzed.

Elapsed: 00: 00: 30.64
SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
```

```
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_ITEM",
14 "SP_TRANS_SUB",
15 "SP_TRANS"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

8 rows selected.

Elapsed: 00: 00: 06.49

Execution Plan
----------------------------------------------------------
   0    SELECT    STATEMENT    Optimizer=CHOOSE    (Cost=18577    Card=126726520
Bytes=30034185240)
   1 0 MERGE JOIN (Cost=18577 Card=126726520 Bytes=30034185240)
   2 1 SORT (JOIN) (Cost=14722 Card=310300 Bytes=20790100)
   3 2 HASH JOIN (Cost=358 Card=310300 Bytes=20790100)
   4 3 TABLE ACCESS (FULL) OF 'SP_TRANS' (Cost=43 Card=229  Bytes=8473)
   5 3 TABLE ACCESS (FULL) OF 'SP_TRANS_SUB' (Cost=158 Card =135502 Bytes=4065060)
   6 1 SORT (JOIN) (Cost=3855 Card=40840 Bytes=6942800)
   7 6 TABLE ACCESS (FULL) OF 'SP_ITEM' (Cost=77 Card=40840 Bytes=6942800)

Statistics
----------------------------------------------------------
        150 recursive calls
         89 db block gets
       1837 consistent gets
        755 physical reads
         60 redo size
       1732 bytes sent via SQL*Net to client
        425 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          4 sorts (memory)
```

```
          1 sorts (disk)
          8 rows processed

SQL>

SQL> analyze table sp_trans compute statistics;

Table analyzed.

Elapsed: 00: 00: 13.00
SQL>
SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_ITEM",
14 "SP_TRANS_SUB",
15 "SP_TRANS"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

8 rows selected.

Elapsed: 00: 00: 01.62

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=1453 Card=447198 Bytes=101066748)
   1 0 NESTED LOOPS (Cost=1453 Card=447198 Bytes=101066748)
   2 1 HASH JOIN (Cost=358 Card=1095 Bytes=61320)
   3 2 TABLE ACCESS (FULL) OF 'SP_TRANS' (Cost=43 Card=273 Bytes=7098)
   4 2 TABLE ACCESS (FULL) OF 'SP_TRANS_SUB' (Cost=158 Card=135502 Bytes=4065060)
   5 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM' (Cost=1 Card= 40840 Bytes=6942800)
   6 5 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)
```

```
Statistics
----------------------------------------------------------
          0 recursive calls
          8 db block gets
       1344 consistent gets
          0 physical reads
          0 redo size
       1824 bytes sent via SQL*Net to client
        425 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          3 sorts (memory)
          0 sorts (disk)
          8 rows processed

SQL>
SQL> analyze table sp_item compute statistics
  2 /

Table analyzed.

Elapsed: 00: 00: 11.67
SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_ITEM",
14 "SP_TRANS_SUB",
15 "SP_TRANS"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

8 rows selected.
```

```
Elapsed: 00: 00: 01.43

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=687 Card=1362 Bytes=128028)
   1 0 HASH JOIN (Cost=687 Card=1362 Bytes=128028)
   2 1 HASH JOIN (Cost=358 Card=1362 Bytes=76272)
   3 2 TABLE ACCESS (FULL) OF 'SP_TRANS' (Cost=43 Card=273 Bytes=7098)
   4 2 TABLE ACCESS (FULL) OF 'SP_TRANS_SUB' (Cost=158 Card=135502 Bytes=4065060)
   5 1 TABLE ACCESS (FULL) OF 'SP_ITEM' (Cost=77 Card=29547 Bytes=1122786)

Statistics
----------------------------------------------------------
          0 recursive calls
         12 db block gets
       1820 consistent gets
          0 physical reads
          0 redo size
       1732 bytes sent via SQL*Net to client
        425 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          3 sorts (memory)
          0 sorts (disk)
          8 rows processed
```

当分析了所有表以后，我们发现，最后 Oracle 给我选择了三个全表扫描。而且，从 db block gets 、consistent gets 全都“提高”了。

然后我在 VENDOR_CODE 列上建了一个索引，试图通过索引加快代码的执行(在实际应用中，通过添加适当的索引来加快 SQL 执行是常用的方法，尤其是在 RBO 模式下，但是添加索引可能改变很多相关 SQL 的执行计划，是否获得确切的性能提高，需要进行相应的测试来验证，Oracle9i 提供索引监视的功能，你可以通过 ALTER INDEX .. MONITORING USAGE 语句来收集索引的使用信息,以确认索引是否被正常使用):

```
SQL> create index idx_vendor on sp_trans(vendor_code);

索引已创建。

已用时间: 00: 00: 02.03
SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
```

```
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
 10 "SP_TRANS"."PAY_MODE",
 11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
 12 0 as PAY_THIS
 13 FROM "SP_ITEM",
 14 "SP_TRANS_SUB",
 15 "SP_TRANS"
 16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
 17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
 18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

已选择 8 行。

已用时间: 00: 00: 01.42

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=646 Card=1362 Bytes= 128028)
   1 0 HASH JOIN (Cost=646 Card=1362 Bytes=128028)
   2 1 HASH JOIN (Cost=317 Card=1362 Bytes=76272)
   3 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS' (Cost=2 Card=273 Bytes=7098)
   4 3 INDEX (RANGE SCAN) OF 'IDX_VENDOR' (NON-UNIQUE) (Cost=1 Card=273)
   5 2 TABLE ACCESS (FULL) OF 'SP_TRANS_SUB' (Cost=158 Card=135502 Bytes=4065060)
   6 1 TABLE ACCESS (FULL) OF 'SP_ITEM' (Cost=77 Card=29547 Bytes=1122786)

Statistics
----------------------------------------------------------
          0 recursive calls
          8 db block gets
       1546 consistent gets
          1 physical reads
          0 redo size
       1732 bytes sent via SQL*Net to client
        425 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          3 sorts (memory)
          0 sorts (disk)
          8 rows processed
```

```
SQL>
```

现在数据库使用了这个索引，执行计划稍好了一点，可是效果仍然不让人满意。

然后我删后来创建的索引，添加 Rule 提示，这是我们得到了最初的执行计划

```
SQL> SELECT /*+ rule */ "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_ITEM",
14 "SP_TRANS_SUB",
15 "SP_TRANS"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

已选择 8 行。
已用时间: 00: 00: 00.71

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=HINT: RULE
   1 0 NESTED LOOPS
   2 1 NESTED LOOPS
   3 2 TABLE ACCESS (FULL) OF 'SP_TRANS'
   4 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB'
   5 4 INDEX (RANGE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE)
   6 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM'
   7 6 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)

Statistics
----------------------------------------------------------
          0 recursive calls
          4 db block gets
```

```
        323 consistent gets
          0 physical reads
          0 redo size
       1809 bytes sent via SQL*Net to client
        426 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          0 sorts (memory)
          0 sorts (disk)
          8 rows processed

SQL>
```

然后我们再次创建这个索引，得出的时间大大缩短(为存在性能问题的数据库查询添加必要的索引，是 DBA 解决问题的重要手段之一):

```
SQL> create index idx_vendor on sp_trans(vendor_code);

索引已创建。

已用时间: 00: 00: 02.43
SQL> SELECT /*+ rule */ "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_ITEM",
14 "SP_TRANS_SUB",
15 "SP_TRANS"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

已选择 8 行。

已用时间: 00: 00: 00.31
```

```
Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=HINT: RULE
   1 0 NESTED LOOPS
   2 1 NESTED LOOPS
   3 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS'
   4 3 INDEX (RANGE SCAN) OF 'IDX_VENDOR' (NON-UNIQUE)
   5 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB'
   6 5 INDEX (RANGE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE)
   7 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM'
   8 7 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)

Statistics
----------------------------------------------------------
          0 recursive calls
          0 db block gets
         49 consistent gets
          1 physical reads
          0 redo size
       1809 bytes sent via SQL*Net to client
        426 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          0 sorts (memory)
          0 sorts (disk)
          8 rows processed

SQL>
```

然而这不是最快的，这是 RBO

我们修改 optimizer_index_cost_adj 这个初始化参数，然后再来查看该 SQL 的执行计划 。

```
SQL> alter session set optimizer_index_cost_adj=30
  2 /

会话已更改。
已用时间: 00: 00: 00.20

SQL> SELECT "SP_TRANS"."TRANS_NO",
  2 "SP_TRANS_SUB"."ITEM_CODE",
  3 "SP_ITEM"."ITEM_NAME",
  4 "SP_ITEM"."CHART_ID",
```

```
  5 "SP_ITEM"."SPECIFICATION",
  6 "SP_TRANS_SUB"."COUNTRY",
  7 "SP_TRANS_SUB"."QTY",
  8 "SP_TRANS_SUB"."PRICE",
  9 "SP_TRANS"."VENDOR_CODE",
10 "SP_TRANS"."PAY_MODE",
11 NVL("SP_TRANS_SUB"."PAY_QTY",0),
12 0 as PAY_THIS
13 FROM "SP_TRANS",
14 "SP_ITEM",
15 "SP_TRANS_SUB"
16 WHERE ( "SP_TRANS_SUB"."TRANS_NO" = "SP_TRANS"."TRANS_NO" ) and
17 ( "SP_ITEM"."ITEM_CODE" = "SP_TRANS_SUB"."ITEM_CODE" ) and
18 ( ( "SP_TRANS"."VENDOR_CODE" = '20011021023') )
19 /

已选择 8 行。

已用时间: 00: 00: 00.11

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=658 Card=1095 Bytes= 102930)
   1 0 NESTED LOOPS (Cost=658 Card=1095 Bytes=102930)
   2 1 NESTED LOOPS (Cost=329 Card=1095 Bytes=61320)
   3 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS' (Cost=1 Card=273 Bytes=7098)
   4 3 INDEX (RANGE SCAN) OF 'IDX_VENDOR' (NON-UNIQUE) (Cost=1 Card=273)
   5 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB' (Cost= 2 Card=135502
Bytes=4065060)
   6 5 INDEX (RANGE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE) (Cost=3 Card=135502)
   7 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM' (Cost=1 Card=29547 Bytes=1122786)
   8 7 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)

Statistics
----------------------------------------------------------
          0 recursive calls
          0 db block gets
         49 consistent gets
          0 physical reads
          0 redo size
       1809 bytes sent via SQL*Net to client
        426 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          0 sorts (memory)
```

```
          0 sorts (disk)
          8 rows processed

SQL>
```

我们来看看以下两个参数对于 CBO 的巨大影响：

OPTIMIZER_INDEX_CACHING

这个初始化参数代表一个百分比,取值范围在 0 到 99 之间.

缺省值是 0，代表当 CBO 使用索引访问数据时，在内存中发现数据的比率是 0%，这意味着通过索引访问数据将需要产生物理读取，代价昂贵。如果使用缺省设置，Oracle 评估成本的时候，很多时候就会错误的选择全表扫描。

OPTIMIZER_INDEX_COST_ADJ

这个初始化参数代表一个百分比,取值范围在 1 到 10000 之间.

该参数表示索引扫描和全表扫描成本的表较。缺省值 100 表示索引扫描成本等于全表扫描。

这些参数对于 CBO 的执行具有重大影响，其缺省值对于数据库来说通常需要调整。

一般来说对于 OPTIMIZER_INDEX_CACHING 可以设置为 90 左右

对于大多数 OLTP 系统，OPTIMIZER_INDEX_COST_ADJ 可以设置在 10 到 50 之间。对于数据仓库和 DSS 系统，可能不能简单的把 OPTIMIZER_INDEX_COST_ADJ 设置为 50，通常我们需要反复调整取得一个合理值.

```
SQL> drop index idx_vendor;

索引已丢弃。

已用时间: 00: 00: 00.61

SQL> /

已选择 8 行。

已用时间: 00: 00: 00.11

Execution Plan
----------------------------------------------------------
   0 SELECT STATEMENT Optimizer=CHOOSE (Cost=700 Card=1095 Bytes= 102930)
   1 0 NESTED LOOPS (Cost=700 Card=1095 Bytes=102930)
   2 1 NESTED LOOPS (Cost=371 Card=1095 Bytes=61320)
   3 2 TABLE ACCESS (FULL) OF 'SP_TRANS' (Cost=43 Card=273 Bytes=7098)
   4 2 TABLE ACCESS (BY INDEX ROWID) OF 'SP_TRANS_SUB' (Cost= 2 Card=135502
Bytes=4065060)
   5 4 INDEX (RANGE SCAN) OF 'PK_SP_TRANS_SUB' (UNIQUE) (Cost=3 Card=135502)
   6 1 TABLE ACCESS (BY INDEX ROWID) OF 'SP_ITEM' (Cost=1 Card= 29547 Bytes=1122786)
```

```
   7 6 INDEX (UNIQUE SCAN) OF 'PK_SP_ITEM' (UNIQUE)

Statistics
----------------------------------------------------------
          0 recursive calls
          4 db block gets
        323 consistent gets
          0 physical reads
          0 redo size
       1809 bytes sent via SQL*Net to client
        426 bytes received via SQL*Net from client
          2 SQL*Net roundtrips to/from client
          0 sorts (memory)
          0 sorts (disk)
          8 rows processed

SQL>
```

相关文档：

以下文档都是相当好的阅读材料，有兴趣的可以仔细阅读：

关于 optimizer_index_cost_adj 等影响 CBO 的参数及设置：
http://www.evdbt.com/SearchIntelligenceCBO.doc

关于成本的计算等，请参考以下文章：

http://www.centrexcc.com/A%20Look%20under%20the%20Hood%20of%20CBO%20-%20the%2010053%20Event.pdf
http://www.centrexcc.com/A%20Look%20under%20the%20Hood%20of%20CBO%20-%20the%2010053%20Event.ppt
关于 CBO,请参考以下文档：
http://metalink.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=35934.1
http://www.itpub.net/showthread.php?threadid=88905
关于执行计划的设置，请参考：
http://osi.oracle.com/~tkyte/article1/autotrace.html