

【SPM】 oracle 如何固定执行计划

1.1 BLOG 文档结构图



- └─ 【SPM】 oracle 如何固定执行计划
 - └─ 1.1 BLOG 文档结构图
 - └─ 1.2 前言部分
 - └─ 1.2.1 导读和注意事项
 - └─ 1.2.2 相关参考文章链接
 - └─ 1.2.3 本文简介
 - └─ 第 2 章固定执行计划的三种方法演示
 - └─ 2.1 outline
 - └─ 2.1.1 基础知识
 - └─ 2.1.2 outline 使用演示
 - └─ 2.2 SQL Profile
 - └─ 2.2.1 基础知识
 - └─ 2.2.2 SQL Profile 使用演示
 - └─ 2.2.2.1 SQL Profile 使用示例--手工创建 SQL ...
 - └─ 一、使用 coe_xfr_sql_profile.sql 脚本生成 ...
 - └─ 2.2.2.2 SQL Profile 使用示例--使用 STA 来生 ...
 - └─ 2.3 SPM(SQL Plan Management)
 - └─ 2.3.1 基础知识
 - └─ 2.3.2 删除 Plans 和 Baselines
 - └─ 2.3.3 SPM 使用演示
 - └─ 2.4 总结

1.2 前言部分

1.2.1 导读和注意事项

各位技术爱好者，看完本文后，你可以掌握如下的技能，也可以学到一些其它你所不知道的知识，~O(∩_∩)O~：

① 固定执行计划的常用方法：outline、SQL Profile、SPM (重点)

② coe_xfr_sql_profile.sql 脚本的使用

Tips：

① 若文章代码格式有错乱，推荐使用 QQ、搜狗或 360 浏览器，也可以下载 pdf 格式的文档来查看，pdf 文档下载地址：<http://yunpan.cn/cdEQedhCs2kFz>（提取码：ed9b）

② 本篇 BLOG 中命令的输出部分需要特别关注的地方我都用灰色背景和粉红色字体来表示，比如下边的例子中，thread 1 的最大归档日志号为 33，thread 2 的最大归档日志号为 43 是需要特别关注的地方；而命令一般使用黄色背景和红色字体标注；对代码或代码输出部分的注释一般采用蓝色字体表示。

```
List of Archived Logs in backup set 11
Thrd Seq      Low SCN      Low Time     Next SCN     Next Time
-----
1      32           1621589      2015-05-29 11:09:52 1625242      2015-05-29 11:15:48
1      33           1625242      2015-05-29 11:15:48 1625293      2015-05-29 11:15:58
2      42           1613951      2015-05-29 10:41:18 1625245      2015-05-29 11:15:49
2      43           1625245      2015-05-29 11:15:49 1625253      2015-05-29 11:15:53
```

```
[ZHLHRDB1:root]:/>ls -lsvg -o  
T_XDESK_APP1_vg  
rootvg  
[ZHLHRDB1:root]:/>  
00:27:22 SQL> alter tablespace idxtbs read write;
```

====> 2097152*512/1024/1024/1024=1G

本文如有错误或不完善的地方请大家多多指正，ITPUB 留言或 QQ 皆可，您的批评指正正是我写作的最大动力。

1.2.2 相关参考文章链接

11.2.0.2 的 SPM 的一个 bug：<http://blog.itpub.net/26736162/viewspace-1248506/>

在 10g/11g 中如何查看 SQL Profiles 信息：<http://blog.itpub.net/26736162/viewspace-2106743/>

【OUTLINE】使用 Oracle Outline 技术暂时锁定 SQL 的执行计划：<http://blog.itpub.net/26736162/viewspace-2102180/>

1.2.3 本文简介

本文介绍了 oracle 在固定执行计划的过程中常使用的 3 种方法 , outline, SQL Profile 和 SPM, 其中 SQL Profile 和 SPM 是重点需要掌握的内容。

第 2 章 固定执行计划的三种方法介绍

2.1 outline

2.1.1 outline 基础知识

在实际项目中，通常在开发环境下一些 SQL 执行没有任何问题，而到了生产环境或生产环境的数据量发生较大的变量时，其 SQL 的执行效率会异常的慢。此时如果更改 SQL，则可能需要重新修改源程序以及重新编译程序。如果觉得修改源程序的成本比较大，则可以使用 OUTLINE 在不改变原应用程序的情况下更改特定 SQL 的执行计划。

OUTLINE 的原理是将调好的 SQL 的执行计划 (一系列的 HINT) 存贮起来，然后该执行计划所对应的 SQL 用目前系统那个效率低下的 SQL 来替代之。从而使得系统每

次执行该 SQL 时，都会使用已存贮的执行计划来执行。因此可以在不改变已有系统 SQL 的情况下达到改变其执行计划的目的。

OUTLINE 方式也是通过存贮 HINT 的方式来达到执行计划的稳定与改变。

当发现低效 SQL 之后，可以使用 hint 优化他，对于 SQL 代码可以修改的情况，直接修改 SQL 代码加上 hint 即可，但是对于 SQL 代码不可修改的情况，Oracle 提供了 outLine 功能来为 SQL 修改 hint，以致执行计划变更！

Outline 机制：

Outline 保存了 SQL 的 hint 在 outline 的表中。当执行 SQL 时，Oracle 会使用 outline 中的 hint 来为 SQL 生成执行计划。

使用 Outline 的步骤：

- (1) 生成新 SQL 和老 SQL 的 2 个 Outline
- (2) 交换两个 SQL 的提示信息
- (3) ON LOGON 触发器设定 session 的 CATEGORY (自定义类别)

SQL 命令行为：SQL> alter session set use_stored_outlines=special;

2.1.2 outline 使用演示

测试过程如下：

```
SYS@test> create user lhr identified by lhr;
User created.
SYS@test> grant dba to lhr;
```

Grant succeeded.

```
SYS@test> grant create any outline,alter any outline,DROP ANY OUTLINE to lhr;
```

Grant succeeded.

```
SYS@test> grant all on OL$HINTS to lhr;
```

Grant succeeded.

```
SYS@test> conn lhr/lhr
```

Connected.

```
LHR@test> select * from v$version;
```

BANNER

```
-----  
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production  
PL/SQL Release 11.2.0.4.0 - Production  
CORE      11.2.0.4.0      Production  
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production  
NLSRTL Version 11.2.0.4.0 - Production
```

```
LHR@test> create table TB_LHR_20160518 as select * from dba_tables;
```

Table created.

```
LHR@test> create index idx_TB_LHR_20160518 on TB_LHR_20160518(TABLE_NAME);
```

Index created.

```
LHR@test> SET AUTOTRACE ON;
```

```
LHR@test> select owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';
```

no rows selected

Execution Plan

Plan hash value: 2186742855

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	34	1 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	TB_LHR_20160518	1	34	1 (0)	00:00:01
* 2	INDEX RANGE SCAN	IDX_TB_LHR_20160518	1		1 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("TABLE_NAME"='TB_LHR_20160518')

Note

- dynamic sampling used for this statement (level=2)

Statistics

```

11 recursive calls
0 db block gets
72 consistent gets
8 physical reads
0 redo size
333 bytes sent via SQL*Net to client
508 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
0 rows processed

```

LHR@test> select /*+full(TB_LHR_20160518)*/ owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';

no rows selected

Execution Plan

Plan hash value: 1750418716

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	34	31 (0)	00:00:01
* 1	TABLE ACCESS FULL	TB_LHR_20160518	1	34	31 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("TABLE_NAME"='TB_LHR_20160518')

Note

- dynamic sampling used for this statement (level=2)

Statistics

```

-----
      7 recursive calls
      0 db block gets
     170 consistent gets
      0 physical reads
      0 redo size
     333 bytes sent via SQL*Net to client
     508 bytes received via SQL*Net from client
      1 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
      0 rows processed

```

LHR@test> set autotrace off;

LHR@test> create or replace outline TB_LHR_20160518_1 on select owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';

Outline created.

LHR@test> create or replace outline TB_LHR_20160518_2 on select /*+full(TB_LHR_20160518)*/ owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';

Outline created.

LHR@test> select name,USED,sql_text from dba_outlines where name like '%TB_LHR_20160518%';

NAME	USED	SQL_TEXT
TB_LHR_20160518_1	UNUSED	select owner from TB_LHR_20160518 where table_name='TB_LHR_20160518'
TB_LHR_20160518_2	UNUSED	select /*+full(TB_LHR_20160518)*/ owner from TB_LHR_20160518 where table_name='T

LHR@test> select name,HINT from dba_outline_hints where JOIN_POS=1 and name like '%TB_LHR_20160518%';

NAME	HINT
TB_LHR_20160518_1	INDEX_RS_ASC(@"SEL\$1" "TB_LHR_20160518"@"SEL\$1" ("TB_LHR_20160518"."TABLE_NAME"))
TB_LHR_20160518_2	FULL(@"SEL\$1" "TB_LHR_20160518"@"SEL\$1")

LHR@test> UPDATE OUTLN.OL\$ SET OL_NAME=DECODE(OL_NAME,'TB_LHR_20160518_2','TB_LHR_20160518_1','TB_LHR_20160518_1','TB_LHR_20160518_2') WHERE OL_NAME IN ('TB_LHR_20160518_1','TB_LHR_20160518_2');

2 rows updated.

LHR@test> commit;

Commit complete.


```
LHR@test> select name,USED,sql_text from dba_outlines where name like '%TB_LHR_20160518%';
```

NAME	USED	SQL_TEXT
TB_LHR_20160518_1	UNUSED	select /*+full(TB_LHR_20160518)*/ owner from TB_LHR_20160518 where table_name='T
TB_LHR_20160518_2	UNUSED	select owner from TB_LHR_20160518 where table_name='TB_LHR_20160518'

```
LHR@test> select name,HINT from dba_outline_hints where JOIN_POS=1 and name like '%TB_LHR_20160518%';
```

NAME	HINT
TB_LHR_20160518_1	INDEX_RS_ASC(@"SEL\$1" "TB_LHR_20160518"@"SEL\$1" ("TB_LHR_20160518"."TABLE_NAME"))
TB_LHR_20160518_2	FULL(@"SEL\$1" "TB_LHR_20160518"@"SEL\$1")

```
LHR@test> SET AUTOTRACE ON;
```

```
LHR@test> alter system set use_stored_outlines=true;
```

System altered.

```
LHR@test> select owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';
```

no rows selected

Execution Plan

Plan hash value: 1750418716

	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
	0	SELECT STATEMENT		89	3026	31 (0)	00:00:01
*	1	TABLE ACCESS FULL	TB_LHR_20160518	89	3026	31 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("TABLE_NAME"='TB_LHR_20160518')

Note

- outline "TB_LHR_20160518_2" used for this statement

Statistics

```

34 recursive calls
147 db block gets
125 consistent gets
0 physical reads
624 redo size
333 bytes sent via SQL*Net to client
508 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
2 sorts (memory)
0 sorts (disk)
0 rows processed

```

```
LHR@test> select /*+full(TB_LHR_20160518)*/ owner from TB_LHR_20160518 where table_name='TB_LHR_20160518';
```

```
no rows selected
```

Execution Plan

```
Plan hash value: 2186742855
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		89	3026	6 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	TB_LHR_20160518	89	3026	6 (0)	00:00:01
* 2	INDEX RANGE SCAN	IDX_TB_LHR_20160518	36		1 (0)	00:00:01

```
Predicate Information (identified by operation id):
```

```
2 - access("TABLE_NAME"='TB_LHR_20160518')
```

```
Note
```

```
- outline "TB_LHR_20160518_1" used for this statement
```

Statistics

```

34 recursive calls
147 db block gets
24 consistent gets
0 physical reads
584 redo size
333 bytes sent via SQL*Net to client
508 bytes received via SQL*Net from client

```

```
1 SQL*Net roundtrips to/from client
2 sorts (memory)
0 sorts (disk)
0 rows processed
```

LHR@test>

2.2 SQL Profile

2.2.1 SQL Profile 基础知识

在 oracle 11g 的后续版本中，`use_stored_outlines` 这个参数已经不存在了。意味着我们不能像以前的版本中使用 `create outline` 的方式来为一个 sql 创建 hint，然后使用 `store outline` 来固定执行计划这种方式了。

SQL Profile 就是为某一 SQL 语句提供除了系统统计信息、对象（表和索引等）统计信息之外的其他信息，比如运行环境、额外的更准确的统计信息，以帮助优化器为 SQL 语句选择更适合的执行计划。SQL Profiles 可以说是 Outlines 的进化。Outlines 能够实现的功能 SQL Profiles 也完全能够实现，而 SQL Profiles 具有 Outlines 不具备的优化，最重要的有二点：

- ① SQL Profiles 更容易生成、更改和控制。
- ② SQL Profiles 在对 SQL 语句的支持上做得更好，也就是适用范围更广。

使用 SQL Profiles 两个目的：

(一) 锁定或者说是稳定执行计划。

(二) 在不能修改应用中的 SQL 的情况下使 SQL 语句按指定的执行计划运行。

10g 之前有 outlines, 10g 之后 sql profile 作为新特性之一出现。如果针对非绑定变量的 sql, outlines 则力不从心。sql profile 最大的优点是在不修改 sql 语句和会话执行环境的情况下去优化 sql 的执行效率, 适合无法在应用程序中修改 sql 时。

SQL Profile 对以下类型语句有效：

SELECT 语句；

UPDATE 语句；

INSERT 语句（仅当使用 SELECT 子句时有效）；

DELETE 语句；

CREATE 语句（仅当使用 SELECT 子句时有效）；

MERGE 语句（仅当作 UPDATE 和 INSERT 操作时有效）。

另外，使用 SQL Profile 还必须有 CREATE ANY SQL PROFILE、DROP ANY SQL PROFILE 和 ALTER ANY SQL PROFILE 等系统权限。

2.2.2 SQL Profile 使用演示

有 2 种生成 SQL Profile 的方法，手动和采用 STA 来生成。

2.2.2.1 SQL Profile 使用示例--手工创建 SQL Profile

创建测试表，根据 DBA_OBJECTS 创建，OBJECT_ID 上有索引

```
LHR@dlhr> select * from v$version;

BANNER
-----
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
PL/SQL Release 11.2.0.4.0 - Production
CORE      11.2.0.4.0      Production
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
NLSRTL Version 11.2.0.4.0 - Production

LHR@dlhr> Create table TB_LHR_20160525 as select * from dba_objects;

Table created.

LHR@dlhr> create index IND_TB_LHR_ID on TB_LHR_20160525(object_id);

Index created.
```

查看 SQL 默认执行计划,走了索引，通过指定 outline 可以获取到系统为我们生成的 hint

```
LHR@dlhr> explain plan for select * from TB_LHR_20160525 where object_id= :a;

Explained.

LHR@dlhr> select * from table(dbms_xplan.display(null,null,'outline'));

PLAN_TABLE_OUTPUT
-----
Plan hash value: 4254050152
```

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		886	179K	7	(0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	TB_LHR_20160525	886	179K	7	(0)	00:00:01
* 2	INDEX RANGE SCAN	IND_TB_LHR_ID	354		1	(0)	00:00:01

Outline Data

```

-----

/*+
  BEGIN_OUTLINE_DATA
  INDEX_RS_ASC(@"SEL$1" "TB_LHR_20160525"@"SEL$1" ("TB_LHR_20160525"."OBJECT_ID"))
  OUTLINE_LEAF(@"SEL$1")
  ALL_ROWS
  DB_VERSION('11.2.0.4')
  OPTIMIZER_FEATURES_ENABLE('11.2.0.4')
  IGNORE_OPTIM_EMBEDDED_HINTS
  END_OUTLINE_DATA
*/

```

Predicate Information (identified by operation id):

```

-----

  2 - access("OBJECT_ID">=TO_NUMBER(:A))

```

Note

```

-----

  - dynamic sampling used for this statement (level=2)

```

32 rows selected.

如果我们想让它走全表扫描，首先获取全表扫描 HINT

```
LHR@dlhr> explain plan for select /*+ full(TB_LHR_20160525) */ from TB_LHR_20160525 where object_id= :a;
```

Explained.

```
LHR@dlhr> select * from table(dbms_xplan.display(null,null,'outline'));
```

PLAN_TABLE_OUTPUT

Plan hash value: 345881005

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		886	179K	352	(2)	00:00:05
* 1	TABLE ACCESS FULL	TB_LHR_20160525	886	179K	352	(2)	00:00:05

Outline Data

```

-----

/*+
  BEGIN_OUTLINE_DATA
  FULL(@"SEL$1" "TB_LHR_20160525"@"SEL$1")
  OUTLINE_LEAF(@"SEL$1")
  ALL_ROWS
  DB_VERSION('11.2.0.4')
  OPTIMIZER_FEATURES_ENABLE('11.2.0.4')
  IGNORE_OPTIM_EMBEDDED_HINTS
  END_OUTLINE_DATA
*/

```

Predicate Information (identified by operation id):

```

-----

1 - filter("OBJECT_ID"=TO_NUMBER(:A))

```

Note

```

-----
- dynamic sampling used for this statement (level=2)

```

31 rows selected.

可以看到全表扫描的 hint 已经为我们生成了，我们选取必要的 hint 就 OK 了，其他的可以不要，使用 sql profile

```

LHR@dlhr> declare
2      v_hints sys.sqlprof_attr;
3  begin
4      v_hints := sys.sqlprof_attr('FULL(@"SEL$1" "TB_LHR_20160525"@"SEL$1")'); -----从上面 Outline Data 部分获取到的 HINT
5      dbms_sqltune.import_sql_profile('select * from TB_LHR_20160525 where object_id= :a', -----SQL 语句部分
6                                     v_hints,
7                                     'TB_LHR_20160525', -----PROFILE 的名字
8                                     force_match => true);
9  end;
10 /

```

PL/SQL procedure successfully completed.

查看是否生效，已经生效了：

```
LHR@dlhr> explain plan for select * from TB_LHR_20160525 where object_id= :a;
```

Explained.

```
LHR@dlhr> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 345881005

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		886	179K	352 (2)	00:00:05
* 1	TABLE ACCESS FULL	TB_LHR_20160525	886	179K	352 (2)	00:00:05

Predicate Information (identified by operation id):

```
1 - filter("OBJECT_ID"=TO_NUMBER(:A))
```

Note

```
- dynamic sampling used for this statement (level=2)
- SQL profile "TB_LHR_20160525" used for this statement
```

18 rows selected.

```
LHR@dlhr> SELECT b.name,d.sql_text, extractvalue(value(h),'.') as hints
2 FROM dba_sql_profiles d,SYS.SQLOBJ$DATA A,
3 SYS.SQLOBJ$ B,
4 TABLE(XMLSEQUENCE(EXTRACT(XMLTYPE(A.COMP_DATA),
5 '/outline_data/hint')))) h
6 where a.signature = b.signature
7 and a.category = b.category
8 and a.obj_type = b.obj_type
9 and a.plan_id = b.plan_id
10 and a.signature=d.signature
11 and D.name = 'TB_LHR_20160525';
```

NAME	SQL_TEXT	HINTS
TB_LHR_20160525	select * from TB_LHR_20160525 where object_id= :a	FULL(@"SEL\$1" "TB_LHR_20160525"@"SEL\$1")

```
LHR@dlhr>
```


一、使用 coe_xfr_sql_profile.sql 脚本生成 sqlprof_attr 数据

最麻烦的 sqlprof_attr('FULL(t1@SEL\$1)') 是这里的格式如何写. 在 mos 上的文章 [note 215187.1](#) 中的 sqlt.zip 的目录 utl 中提供了脚本

coe_xfr_sql_profile.sql 可以生成这些信息.

1. 建立测试表和数据

```
SYS@dlhr> select * from v$version;

BANNER
-----
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
PL/SQL Release 11.2.0.4.0 - Production
CORE      11.2.0.4.0      Production
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
NLSRTL Version 11.2.0.4.0 - Production


LHR@dlhr> create table scott.test as select * from dba_objects;

Table created.

LHR@dlhr> create index scott.idx_test_01 on scott.test(object_id);

Index created.

LHR@dlhr> exec dbms_stats.gather_table_stats('scott','test',cascade=>true);

PL/SQL procedure successfully completed.

LHR@dlhr> update scott.test set object_id=10 where object_id>10;

LHR@dlhr> commit;

Commit complete.
```

```
LHR@dlhr> select OBJECT_ID ,count(1) from scott.test group by OBJECT_ID;
```

OBJECT_ID	COUNT(1)
6	1
7	1
5	1
8	1
3	1
2	1
10	87076
4	1
9	1

9 rows selected.

2. 执行查询语句

--执行原有的查询语句,查看执行计划发现走索引,实际上这时表中大部分行的 object_id 都已经被更新为 10,所以走索引是不合理的.

```
LHR@dlhr>
LHR@dlhr> set autot traceonly explain stat
LHR@dlhr>
LHR@dlhr> select * from scott.test where object_id=10;
```

87076 rows selected.

Execution Plan

Plan hash value: 3384190782

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	98	2 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	TEST	1	98	2 (0)	00:00:01
* 2	INDEX RANGE SCAN	IDX_TEST_01	1		1 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("OBJECT_ID"=10)

Statistics

```

0 recursive calls
0 db block gets
13060 consistent gets
0 physical reads
0 redo size
9855485 bytes sent via SQL*Net to client
64375 bytes received via SQL*Net from client
5807 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
87076 rows processed

```

LHR@dlhr> select /*+ full(test)*/ from scott.test where object_id=10;

87076 rows selected.

Execution Plan

Plan hash value: 217508114

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		1	98	351	(2)	00:00:05
* 1	TABLE ACCESS FULL	TEST	1	98	351	(2)	00:00:05

Predicate Information (identified by operation id):

1 - filter("OBJECT_ID"=10)

Statistics

```

1 recursive calls
0 db block gets
6973 consistent gets

```

```
0 physical reads
0 redo size
4159482 bytes sent via SQL*Net to client
64375 bytes received via SQL*Net from client
5807 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
87076 rows processed
```

3. 查询上面两个语句的 sql_id, plan_hash_value

```
LHR@dlhr> set autot off
LHR@dlhr>
LHR@dlhr> col sql_text format a100
LHR@dlhr> select sql_text, sql_id, plan_hash_value from v$sql
2 where sql_text like 'select * from scott.test where object_id=10%';
```

SQL_TEXT	SQL_ID	PLAN_HASH_VALUE
select * from scott.test where object_id=10	cpk9jsg2qt52r	3384190782

```
LHR@dlhr> select sql_text, sql_id, plan_hash_value from v$sql
2 where sql_text like 'select /*+ full(test)*/ from scott.test where object_id=10%';
```

SQL_TEXT	SQL_ID	PLAN_HASH_VALUE
select /*+ full(test)*/ from scott.test where object_id=10	06c2mucgn6t5g	217508114

4. 把 coe_xfr_sql_profile.sql 放在 \$ORACLE_HOME/rdbms/admin 下, 或者放在 /tmp 下都可以。



coe_xfr_sql_profile.sql

5.对上面的两个 sql 产生 outline data 的 sql.

```
[ZHLHRSPMDB2:oracle]:/oracle>cd /tmp
[ZHLHRSPMDB2:oracle]:/tmp>
[ZHLHRSPMDB2:oracle]:/tmp>
[ZHLHRSPMDB2:oracle]:/tmp>
[ZHLHRSPMDB2:oracle]:/tmp>
[ZHLHRSPMDB2:oracle]:/tmp>sqlplus / as sysdba
```

SQL*Plus: Release 11.2.0.4.0 Production on Thu May 26 09:15:14 2016

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:

Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options

```
SYS@dlhr> @$ORACLE_HOME/rdbms/admin/coe_xfr_sql_profile.sql cpk9jsg2qt52r 3384190782
```

Parameter 1:
SQL_ID (required)

PLAN_HASH_VALUE	AVG_ET_SECS
3384190782	.046

Parameter 2:
PLAN_HASH_VALUE (required)

Values passed to coe_xfr_sql_profile:
~~~~~

SQL\_ID : "cpk9jsg2qt52r"  
PLAN\_HASH\_VALUE: "3384190782"

```
SQL>BEGIN
 2 IF :sql_text IS NULL THEN
 3 RAISE_APPLICATION_ERROR(-20100, 'SQL_TEXT for SQL_ID &&sql_id. was not found in memory (gv$sqltext_with_newlines) or AWR (dba_hist_sqltext).');
 4 END IF;
 5 END;
 6 /
SQL>SET TERM OFF;
SQL>BEGIN
```

```
2 IF :other_xml IS NULL THEN
3     RAISE_APPLICATION_ERROR(-20101, 'PLAN for SQL_ID &sql_id. and PHV &plan_hash_value. was not found in memory (gv$sql_plan) or AWR (dba_hist_sql_plan).');
4 END IF;
5 END;
6 /
SQL>SET TERM OFF;
```

Execute coe\_xfr\_sql\_profile\_cpk9jsg2qt52r\_3384190782.sql  
on TARGET system in order to create a custom SQL Profile  
with plan 3384190782 linked to adjusted sql\_text.

COE\_XFR\_SQL\_PROFILE completed.

```
SQL>@$ORACLE_HOME/rdbms/admin/coe_xfr_sql_profile.sql 06c2mucgn6t5g 217508114
```

Parameter 1:  
SQL\_ID (required)

| PLAN_HASH_VALUE | AVG_ET_SECS |
|-----------------|-------------|
| 217508114       | .113        |

Parameter 2:  
PLAN\_HASH\_VALUE (required)

Values passed to coe\_xfr\_sql\_profile:  
~~~~~

SQL_ID : "06c2mucgn6t5g"
PLAN_HASH_VALUE: "217508114"

```
SQL>BEGIN
2 IF :sql_text IS NULL THEN
3     RAISE_APPLICATION_ERROR(-20100, 'SQL_TEXT for SQL_ID &sql_id. was not found in memory (gv$sqltext_with_newlines) or AWR (dba_hist_sqltext).');
4 END IF;
5 END;
6 /
SQL>SET TERM OFF;
SQL>BEGIN
2 IF :other_xml IS NULL THEN
3     RAISE_APPLICATION_ERROR(-20101, 'PLAN for SQL_ID &sql_id. and PHV &plan_hash_value. was not found in memory (gv$sql_plan) or AWR (dba_hist_sql_plan).');
4 END IF;
5 END;
6 /
SQL>SET TERM OFF;
```

```
Execute coe_xfr_sql_profile_06c2mucgn6t5g_217508114.sql  
on TARGET system in order to create a custom SQL Profile  
with plan 217508114 linked to adjusted sql_text.
```

```
COE_XFR_SQL_PROFILE completed.
```

6. 替换文件 coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql 中的 SYS.SQLPROF_ATTR 部分, 把它更改为

coe_xfr_sql_profile_06c2mucgn6t5g_217508114.sql 中产生的 SYS.SQLPROF_ATTR 部分, 其中:

coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql 的 SYS.SQLPROF_ATTR

```
h := SYS.SQLPROF_ATTR(  
q'[BEGIN_OUTLINE_DATA]',  
q'[IGNORE_OPTIM_EMBEDDED_HINTS]',  
q'[OPTIMIZER_FEATURES_ENABLE('11.2.0.4')]',  
q'[DB_VERSION('11.2.0.4')]',  
q'[ALL_ROWS]',  
q'[OUTLINE_LEAF(@"SEL$1")]',  
q'[INDEX_RS_ASC(@"SEL$1" "TEST"@"SEL$1" ("TEST"."OBJECT_ID"))]',  
q'[END_OUTLINE_DATA]');
```

----coe_xfr_sql_profile_06c2mucgn6t5g_217508114.sql 的 SYS.SQLPROF_ATTR

```
h := SYS.SQLPROF_ATTR(  
q'[BEGIN_OUTLINE_DATA]',  
q'[IGNORE_OPTIM_EMBEDDED_HINTS]',  
q'[OPTIMIZER_FEATURES_ENABLE('11.2.0.4')]',  
q'[DB_VERSION('11.2.0.4')]',  
q'[ALL_ROWS]',  
q'[OUTLINE_LEAF(@"SEL$1")]',  
q'[FULL(@"SEL$1" "TEST"@"SEL$1")]',  
q'[END_OUTLINE_DATA]');
```

生成的文件在当前目录:



coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql



coe_xfr_sql_profile_06c2mucgn6t5g_217508114.sql

7. 执行替换过 SYS.SQLPROF_ATTR 的 SQL, coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql

SQL> @/tmp/coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql

```
SQL>@coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql
SQL>REM
SQL>REM $Header: 215187.1 coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql 11.4.4.4 2016/05/26 carlos.sierra $
SQL>REM
SQL>REM Copyright (c) 2000-2012, Oracle Corporation. All rights reserved.
SQL>REM
SQL>REM AUTHOR
SQL>REM   carlos.sierra@oracle.com
SQL>REM
SQL>REM SCRIPT
SQL>REM   coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql
SQL>REM
SQL>REM DESCRIPTION
SQL>REM   This script is generated by coe_xfr_sql_profile.sql
SQL>REM   It contains the SQL*Plus commands to create a custom
SQL>REM   SQL Profile for SQL_ID cpk9jsg2qt52r based on plan hash
SQL>REM   value 3384190782.
SQL>REM   The custom SQL Profile to be created by this script
SQL>REM   will affect plans for SQL commands with signature
SQL>REM   matching the one for SQL Text below.
SQL>REM   Review SQL Text and adjust accordingly.
SQL>REM
SQL>REM PARAMETERS
SQL>REM   None.
SQL>REM
SQL>REM EXAMPLE
SQL>REM   SQL> START coe_xfr_sql_profile_cpk9jsg2qt52r_3384190782.sql;
SQL>REM
SQL>REM NOTES
SQL>REM   1. Should be run as SYSTEM or SYSDBA.
```



```
SQL>REM 2. User must have CREATE ANY SQL PROFILE privilege.
SQL>REM 3. SOURCE and TARGET systems can be the same or similar.
SQL>REM 4. To drop this custom SQL Profile after it has been created:
SQL>REM EXEC DBMS_SQLTUNE.DROP_SQL_PROFILE('coe_cpk9jsg2qt52r_3384190782');
SQL>REM 5. Be aware that using DBMS_SQLTUNE requires a license
SQL>REM for the Oracle Tuning Pack.
SQL>REM 6. If you modified a SQL putting Hints in order to produce a desired
SQL>REM Plan, you can remove the artificial Hints from SQL Text pieces below.
SQL>REM By doing so you can create a custom SQL Profile for the original
SQL>REM SQL but with the Plan captured from the modified SQL (with Hints).
SQL>REM
SQL>WHenever SQLERROR EXIT SQL.SQLCODE;
SQL>REM
SQL>VAR signature NUMBER;
SQL>VAR signaturef NUMBER;
SQL>REM
SQL>DECLARE
  2  sql_txt CLOB;
  3  h SYS.SQLPROF_ATTR;
  4  PROCEDURE wa (p_line IN VARCHAR2) IS
  5  BEGIN
  6  DBMS_LOB.WRITEAPPEND(sql_txt, LENGTH(p_line), p_line);
  7  END wa;
  8  BEGIN
  9  DBMS_LOB.CREATETEMPORARY(sql_txt, TRUE);
10  DBMS_LOB.OPEN(sql_txt, DBMS_LOB.LOB_READWRITE);
11  -- SQL Text pieces below do not have to be of same length.
12  -- So if you edit SQL Text (i.e. removing temporary Hints),
13  -- there is no need to edit or re-align unmodified pieces.
14  wa(q'[select * from scott.test where object_id=10]');
15  DBMS_LOB.CLOSE(sql_txt);
16  h := SYS.SQLPROF_ATTR(
17  q'[BEGIN_OUTLINE_DATA]',
18  q'[IGNORE_OPTIM_EMBEDDED_HINTS]',
19  q'[OPTIMIZER_FEATURES_ENABLE('11.2.0.4')] ',
20  q'[DB_VERSION('11.2.0.4')] ',
21  q'[ALL_ROWS]',
22  q'[OUTLINE_LEAF(@"SEL$1")]',
23  q'[FULL(@"SEL$1" "TEST"@"SEL$1")]',
24  q'[END_OUTLINE_DATA]');
25  :signature := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE(sql_txt);
26  :signaturef := DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE(sql_txt, TRUE);
27  DBMS_SQLTUNE.IMPORT_SQL_PROFILE (
28  sql_text => sql_txt,
29  profile => h,
30  name => 'coe_cpk9jsg2qt52r_3384190782',
31  description => 'coe_cpk9jsg2qt52r_3384190782 '||:signature||' '||:signaturef||'',
32  category => 'DEFAULT',
```

```
33 validate      => TRUE,
34 replace        => TRUE,
35 force_match => FALSE /* TRUE:FORCE (match even when different literals in SQL). FALSE:EXACT (similar to CURSOR_SHARING) */ );
36 DBMS_LOB.FREETEMPORARY(sql_txt);
37 END;
38 /
```

PL/SQL procedure successfully completed.

SQL>WHENEVER SQLERROR CONTINUE

SQL>SET ECHO OFF;

SIGNATURE

10910590721604799112

SIGNATUREF

15966118871002195466

... manual custom SQL Profile has been created

COE_XFR_SQL_PROFILE_cpk9jsg2qt52r_3384190782 completed

8.查看产生的 sql profile,此时原语句在不加 hint 的情况下也走全表扫了

```
select * from dba_sql_profiles;
```

```
SYS@dlhr> col sql_text for a50
```

```
SYS@dlhr> col hints for a50
```

```
SYS@dlhr> SELECT b.name,to_char(d.sql_text) sql_text, extractvalue(value(h),'.') as hints
```

```
2 FROM dba_sql_profiles d,SYS.SQLOBJ$DATA A,
```

```
3 SYS.SQLOBJ$ B,
```

```
4 TABLE (XMLSEQUENCE (EXTRACT (XMLTYPE (A.COMP_DATA),
```

```
5 '/outline_data/hint')))) h
```

```
6 where a.signature = b.signature
```

```
7 and a.category = b.category
```

```
8 and a.obj_type = b.obj_type
```

```
9 and a.plan_id = b.plan_id
```

```
10 and a.signature=d.signature
```

```
11 and D.name = 'coe_cpk9jsg2qt52r_3384190782';
```

NAME	SQL_TEXT	HINTS
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	BEGIN_OUTLINE_DATA
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	IGNORE_OPTIM_EMBEDDED_HINTS
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	OPTIMIZER_FEATURES_ENABLE('11.2.0.4')
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	DB_VERSION('11.2.0.4')
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	ALL_ROWS
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	OUTLINE_LEAF(@"SEL\$1")
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	FULL(@"SEL\$1" "TEST"@"SEL\$1")
coe_cpk9jsg2qt52r_3384190782	select * from scott.test where object_id=10	END_OUTLINE_DATA

8 rows selected.

SYS@dlhr>

9.验证 SQL Profile 是否生效

```
SYS@dlhr> set autot traceonly explain stat
SYS@dlhr> select * from scott.test where object_id=10;
```

87076 rows selected.

Execution Plan

Plan hash value: 217508114

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	98	351 (2)	00:00:05
* 1	TABLE ACCESS FULL	TEST	1	98	351 (2)	00:00:05

Predicate Information (identified by operation id):

1 - filter("OBJECT_ID"=10)

Note

- SQL profile "coe_cpk9jsg2qt52r_3384190782" used for this statement

Statistics

```
-----
      0 recursive calls
      0 db block gets
    6973 consistent gets
      0 physical reads
      0 redo size
  4159482 bytes sent via SQL*Net to client
    64375 bytes received via SQL*Net from client
    5807 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
    87076 rows processed
```

注意:

- 1.这个测试只是为了演示通过 `coe_xfr_sql_profile.sql` 实现手动加 hint 的方法,实际上面的语句问题的处理最佳的方法应该是重新收集 `scott.test` 的统计信息才对.
- 2.当一条 sql 既有 sql profile 又有 stored outline 时,优化器优先选择 stored outline.
- 3.force_match 参数,TRUE:FORCE (match even when different literals in SQL),FALSE:EXACT (similar to CURSOR_SHARING).
- 4.通过 sql profile 手动加 hint 的方法很简单,而为 sql 添加最合理的 hint 才是关键.
- 5.测试完后,可以通过 `exec dbms_sqltune.drop_sql_profile(name =>'coe_cpk9jsg2qt52r_3384190782');` 删除这个 sql profile.
- 6.执行 `coe_xfr_sql_profile.sql` 脚本的时候用户需要对当前目录有生成文件的权限,最好当前目录是 /tmp

2.2.2.2 SQL Profile 使用示例--使用 STA 来生成 SQL Profile

利用 STA 对语句进行优化后，STA 会对语句进行分析，采用最优的优化策略，并给出优化后的查询计划。你可以按照 STA 给出的建议重写语句。但是，有些情况下，你可能无法重写语句（比如在生产环境中，你的语句又在一个包中）。这个时候就可以利用 sql profile，将优化策略存储在 profile 中，Oracle 在构建这条语句的查询计划时，就不会使用已有相关统计数据，而使用 profile 的策略，生成新的查询计划。

一、 第一步：给用户赋权限

```
[ZHLHRSPMDB2:oracle]:/oracle>sqlplus / as sysdba

SQL*Plus: Release 11.2.0.4.0 Production on Wed May 25 16:47:29 2016

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options

SYS@dlhr>
SYS@dlhr>
SYS@dlhr>
SYS@dlhr> GRANT CREATE ANY SQL PROFILE TO LHR;

Grant succeeded.

SYS@dlhr> GRANT DROP ANY SQL PROFILE TO LHR;

Grant succeeded.

SYS@dlhr> GRANT ALTER ANY SQL PROFILE TO LHR;

Grant succeeded.

SYS@dlhr> conn lhr/lhr
```

```
Connected.
LHR@dlhr>
LHR@dlhr> select * from v$version;

BANNER
-----
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
PL/SQL Release 11.2.0.4.0 - Production
CORE      11.2.0.4.0      Production
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
NLSRTL Version 11.2.0.4.0 - Production

LHR@dlhr> create table lhr.TB_LHR_20160525_01 as select * from dba_objects;

Table created.

LHR@dlhr> create index lhr.TB_LHR_20160525_01_idx on TB_LHR_20160525_01(object_id);

Index created.

LHR@dlhr> exec dbms_stats.gather_table_stats('lhr','TB_LHR_20160525_01',cascade=>true,degree=>4);

PL/SQL procedure successfully completed.

LHR@dlhr> set autot on
LHR@dlhr> select /*+no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100 ;
```

COUNT(*)

1

Execution Plan

Plan hash value: 3612989399

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		1	5	351	(2)	00:00:05
1	SORT AGGREGATE		1	5			
* 2	TABLE ACCESS FULL	TB_LHR_20160525_01	1	5	351	(2)	00:00:05

Predicate Information (identified by operation id):

2 - filter("OBJECT_ID"=100)

Statistics

```
-----
      1 recursive calls
      0 db block gets
    1249 consistent gets
      0 physical reads
      0 redo size
     526 bytes sent via SQL*Net to client
     520 bytes received via SQL*Net from client
        2 SQL*Net roundtrips to/from client
        0 sorts (memory)
        0 sorts (disk)
        1 rows processed
```

LHR@dlhr> set autot off

LHR@dlhr> SELECT v.SQL_ID, v.SQL_TEXT FROM v\$sql v WHERE v.SQL_TEXT like '%no_index(TB_LHR_20160525_01%' and v.SQL_TEXT not like '%v\$sql%' ;

SQL_ID

SQL_TEXT

7jt1btjkcczb8

select /*no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100

7suktf0w95cry

EXPLAIN PLAN SET STATEMENT_ID='PLUS150249' FOR select /*no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100

二、 第二步：创建、执行优化任务

LHR@dlhr> DECLARE

2 my_task_name VARCHAR2(30);

3 my_sqltext CLOB;

4 BEGIN

5 my_sqltext := 'select /*no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100';

6 my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK(

7 sql_text => my_sqltext,

8 user_name => 'LHR',

9 scope => 'COMPREHENSIVE',

10 time_limit => 60,

11 task_name => 'sql_profile_test',

12 description => 'Task to tune a query on a specified table');

13 DBMS_SQLTUNE.EXECUTE_TUNING_TASK(task_name => 'sql_profile_test');

```
14 END;  
15 /
```

PL/SQL procedure successfully completed.

或者也可以使用 `sqlid` 来生成优化任务，如下：

```
LHR@dlhr> DECLARE  
2   a_tuning_task VARCHAR2(30);  
3 BEGIN  
4   a_tuning_task := dbms_sqltune.create_tuning_task(sql_id => '7jt1btjkcczb8',  
5                                                    task_name => 'sql_profile_test_SQLID');  
6   dbms_sqltune.execute_tuning_task(a_tuning_task);  
7 END;  
8 /
```

PL/SQL procedure successfully completed.

三、 第三步：查看优化建议

```
LHR@dlhr> set autot off  
LHR@dlhr> set long 10000  
LHR@dlhr> set longchunksize 1000  
LHR@dlhr> set linesize 100  
LHR@dlhr> SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK( 'sql_profile_test') from DUAL;
```

DBMS_SQLTUNE.REPORT_TUNING_TASK('SQL_PROFILE_TEST')

GENERAL INFORMATION SECTION

```
Tuning Task Name   : sql_profile_test  
Tuning Task Owner  : LHR  
Workload Type     : Single SQL Statement  
Scope             : COMPREHENSIVE  
Time Limit(seconds): 60  
Completion Status  : COMPLETED  
Started at        : 05/25/2016 16:58:31  
Completed at      : 05/25/2016 16:58:32
```

Schema Name: LHR

SQL ID : 9kzm8scz6t92z

SQL Text : select /*+no_index(TB_LHR_20160525_01


```
TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01
where object_id = 100
```

FINDINGS SECTION (1 finding)

1- SQL Profile Finding (see explain plans section below)

A potentially better execution plan was found for this statement.

Recommendation (estimated benefit: 99.83%)

- Consider accepting the recommended SQL profile.

```
execute dbms_sqltune.accept_sql_profile(task_name => 'sql_profile_test',
task_owner => 'LHR', replace => TRUE);
```

Validation results

The SQL profile was tested by executing both its plan and the original plan and measuring their respective execution statistics. A plan may have been only partially executed if the other could be run to completion in less time.

	Original Plan	With SQL Profile	% Improved
	-----	-----	-----
Completion Status:	COMPLETE	COMPLETE	
Elapsed Time (s):	.006278	.00004	99.36 %
CPU Time (s):	.003397	.000021	99.38 %
User I/O Time (s):	0	0	
Buffer Gets:	1249	2	99.83 %
Physical Read Requests:	0	0	
Physical Write Requests:	0	0	

DBMS_SQLTUNE.REPORT_TUNING_TASK('SQL_PROFILE_TEST')

Physical Read Bytes:	0	0
Physical Write Bytes:	0	0
Rows Processed:	1	1
Fetches:	1	1
Executions:	1	1

Notes

1. Statistics for the original plan were averaged over 10 executions.
 2. Statistics for the SQL profile plan were averaged over 10 executions.
-

EXPLAIN PLANS SECTION

1- Original With Adjusted Cost

Plan hash value: 3612989399

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	5	351 (2)	00:00:05
1	SORT AGGREGATE		1	5		
* 2	TABLE ACCESS FULL	TB_LHR_20160525_01	1	5	351 (2)	00:00:05

Predicate Information (identified by operation id):

2 - filter("OBJECT_ID">=100)

2- Using SQL Profile

Plan hash value: 661515879

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	5	1 (0)	00:00:01
1	SORT AGGREGATE		1	5		
* 2	INDEX RANGE SCAN	TB_LHR_20160525_01_IDX	1	5	1 (0)	00:00:01

Predicate Information (identified by operation id):

DBMS_SQLTUNE.REPORT_TUNING_TASK('SQL_PROFILE_TEST')

2 - access("OBJECT_ID">=100)

这里可以看到，在优化建议中给出了新的查询计划。现在，我们决定接受这个建议，并且不重写语句。

四、 第四步：接受 profile

```
LHR@dlhr> set autot on
LHR@dlhr> select /*+no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100 ;
```

```
COUNT(*)
```

```
1
```

```
Execution Plan
```

```
Plan hash value: 3612989399
```

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		1	5	351	(2)	00:00:05
1	SORT AGGREGATE		1	5			
* 2	TABLE ACCESS FULL	TB_LHR_20160525_01	1	5	351	(2)	00:00:05

```
Predicate Information (identified by operation id):
```

```
2 - filter("OBJECT_ID"=100)
```

```
Statistics
```

```
0 recursive calls
0 db block gets
1249 consistent gets
0 physical reads
0 redo size
526 bytes sent via SQL*Net to client
520 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
```

```
LHR@dlhr> execute dbms_sqltune.accept_sql_profile(task_name =>'sql_profile_test_SQLID', task_owner => 'LHR', replace => TRUE);
```

```
PL/SQL procedure successfully completed.
```

```
LHR@dlhr> set autot off
LHR@dlhr> SELECT e.task_name, b.name, d.sql_text, extractvalue(value(h), '.') as hints
2      FROM dba_sql_profiles d,
3           dba_advisor_tasks e,
4           SYS.SQLOBJ$DATA A,
5           SYS.SQLOBJ$ B,
6           TABLE(XMLSEQUENCE(EXTRACT(XMLTYPE(A.COMP_DATA),
7           '/outline_data/hint')))) h
8  where a.signature = b.signature
9         and a.category = b.category
10        and a.obj_type = b.obj_type
11        and a.plan_id = b.plan_id
12        and a.signature = d.signature
13        and d.task_id=e.task_id
14        and d.name = 'SYS_SQLPROF_0154e728ad3f0000'
15      ;
```

```
TASK_NAME          NAME
-----
SQL_TEXT
-----
HINTS
-----

sql_profile_test    SYS_SQLPROF_0154e728ad3f0000
select /*no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01
where object_id = 100
OPTIMIZER_FEATURES_ENABLE(default)

sql_profile_test    SYS_SQLPROF_0154e728ad3f0000
select /*no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01
where object_id = 100
IGNORE_OPTIM_EMBEDDED_HINTS
```

在这里用了包 DBMS_SQLTUNE 的另一个函数：ACCEPT_SQL_PROFILE。其中，参数 task_name 即我们创建的优化建议任务的名称，name 是 profile 的名字，可

以是任意合法名称。此外这个函数还有其他一些函数，下面是这个函数的原型：

```
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
  task_name      IN   VARCHAR2,
  object_id      IN   NUMBER      := NULL,
  name           IN   VARCHAR2 := NULL,
  description    IN   VARCHAR2 := NULL,
  category       IN   VARCHAR2 := NULL;
```

```

task_owner      IN VARCHAR2      := NULL,
replace         IN BOOLEAN       := FALSE,
force_match     IN BOOLEAN       := FALSE)
RETURN VARCHAR2;
```

Description 是 profile 的描述信息；task_owner 是优化建议任务的所有者；replace 为 TRUE 时，如果这个 profile 已经存在，就代替它；force_match 为 TRUE 时，表示与语句强制匹配，即强制使用绑定变量，和系统参数 cursor_sharing 设置为 FORCE 时类似，为 FALSE 时，与 cursor_sharing 设置为 EXACT 时类似，即完全匹配。

这里要特别提到的是 category 这个参数，你可以通过设置这个参数，制定特定会话使用这个 profile。在 10g 中，每个会话都有一个新参数 SQTUNE_CATEGORY，他的默认值是 DEFAULT。而我们在调用这个函数时，如果没有指定这个参数，那它的值也是 DEFAULT，而如果我们给这个 profile 指定了一个其它的 CATEGORY 值，如 FOR_TUNING，那么只有会话参 SQTUNE_CATEGORY 也为 FOR_TUNING 时，才会使用这个 profile。为什么说这个参数很有用呢？试想一个这样的环境：你在一个生产系统上利用 STA 调优一条语句，STA 已经给出了优化建议，但是你又不敢贸然实施它给出的建议（毕竟它只是机器嘛，不能完全信任），你就可以创建一个有特殊 CATEGORY 的 profile，然后在你自己的会话中制定 SQTUNE_CATEGORY 为这个特殊的 CATEGORY，那就既可以看优化建议的实际效果又不影响生产环境。

此外可以通过视图 DBA_SQL_PROFILES 来查看已经创建的 profile。

五、 第五步：查看 profile 的效果

```

LHR@dlhr> set autot on
LHR@dlhr> select /*+no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/count(*) from lhr.TB_LHR_20160525_01 where object_id = 100 ;
```

```

COUNT(*)
```

```

-----
1
```

```

Execution Plan
```

Plan hash value: 661515879

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	5	1 (0)	00:00:01
1	SORT AGGREGATE		1	5		
* 2	INDEX RANGE SCAN	TB_LHR_20160525_01_IDX	1	5	1 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("OBJECT_ID">=100)

Note

- SQL profile "SYS_SQLPROF_0154e728ad3f0000" used for this statement

Statistics

```

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

```

从 NOTE 部分可以看到，语句采用了 profile 中的数据，创建了新的查询计划。并且在查询计划中还有一些附加信息，表明这个语句是采用

了 'SYS_SQLPROF_0154e728ad3f0000' 这个 profile，而不是根据对象上面的统计数据来生成的查询计划。

但上述方法主要是依赖 sql tuning advisor，如果它无法生成你想要的执行计划。你还可以通过手动的方式，通过 sql profile 把 hint 加进去。复杂的 SQL 的

hint 可以采用脚本 coe_xfr_sql_profile.sql 来产生原语句的 outline data 和加 hint 语句的 outline data，然后替换对应的 SYS.SQLPROF_ATTR，最后执

行生成的 sql 就可以了。

使用 PLSQL DEVELOPER 11 查看执行计划，如下图，新版本的好处：

```
select /*+no_index(TB_LHR_20160525_01 TB_LHR_20160525_01_idx)*/
count(*)
  from lhr.TB_LHR_20160525_01
 where object id = 100
```

Optimizer goal: All rows

Tree HTML Text XML

Plan Hash Value : 661515879

Id	Operation	Name	Rows	Bytes	Cost	Time
0	SELECT STATEMENT		1	5	1	00:00:01
1	SORT AGGREGATE		1	5		
* 2	INDEX RANGE SCAN	TB_LHR_20160525_01_IDX	1	5	1	00:00:01

Predicate Information (identified by operation id):

- 2 - access("OBJECT_ID"=100)

Note

- SQL profile "SYS_SQLPROF_0154e728ad3f0000" used for this statement

2.3 SPM(SQL Plan Management)

2.3.1 SPM 基础知识

SQL 语句的 SQL 执行计划发生更改时，可能存在性能风险。

SQL 计划发生更改的原因有很多，如优化程序版本、优化程序统计信息、优化程序参数、方案定义、系统设计和 SQL 概要文件创建等。

在以前版本的 Oracle DB 中引入了各种计划控制技术（如存储的大纲(storedoutline(9i))和 SQL 概要文件等(SQLprofile(10g))），用于解决计划更改导致的性能回归。但是，这些技术都是需要手动干预的被动式进程。

SQL 计划管理是一种随 Oracle Database 11g 引入的新功能，通过维护所谓的“SQL 计划基线(SQL plan baseline(11g))”来使系统能够自动控制 SQL 计划演变。启用此功能后，只要证明新生成的 SQL 计划与 SQL 计划基线相集成不会导致性能回归，就可以进行此项集成。因此，在执行某个 SQL 语句时，只能使用对应的 SQL 计划基线中包括的计划。可以使用 SQL 优化集自动加载或植入 SQL 计划基线。

SQL 计划管理功能的主要优点是系统性能稳定，不会出现计划回归。此外，该功能还可以节省 DBA 的许多时间，这些时间通常花费在确定和分析 SQL 性能回归以及寻找可行的解决方案上。Oracle11g 中，Oracle 提供 dbms_spm 包来管理 SQL Plan，SPM 是一个预防机制，它记录并评估 sql 的执行计划，将已知的高效的 sql 执行计划建立为 SQL Plan Baselines，SQL Plan Baseline 的功能是保持 SQL 的性能而不必关注系统的改变。

在 SQL Plan BaseLines 捕获阶段，Oracle 记录 SQL 的执行计划并检测该执行计划是否已经改变，如果 SQL 改变后的执行计划是安全的，则 SQL 就使用新的执行计划，因此，Oracle 维护单个 SQL 执行计划的历史信息，Oracle 维护的 SQL 执行计划的历史仅仅针对重复执行的 SQL，SQL Plan Baseline 可以手工 load，也

可以设置为自动捕获。

加载 SQL 计划基线的方式有两种：

(1) 即时捕获,自动捕获(Automatic Plan Capture)：

使用自动计划捕获，方法是：将初始化参数 `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` 设置为 `TRUE`。默认情况下，该参数设置为 `FALSE`。将该参数设置为 `TRUE` 将打开自动标识可重复 SQL 语句，以及自动为此类语句创建计划历史记录的功能。如果要激活自动的 SQL Plan Capture，则需要设置 `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES`，该参数默认为 `False`，如果设置为 `True`，则表示自动捕获 SQL Plan，则系统会自动创建并维护 SQL Plan History，SQL Plan History 包括优化器关注的：比如 an execution plan, SQL text, outline, bind variables, and compilation environment。

(2) 成批加载 (Manual Plan Loading)：

使用 `DBMS_SPM` 程序包；该程序包支持手动管理 SQL 计划基线。使用此程序包，可以将 SQL 计划从游标高速缓存或现有的 SQL 优化集(STS) 直接加载到 SQL 计划基线中。对于要从 STS 加载到 SQL 计划基线的 SQL 语句，需要将其 SQL 计划存储在 STS 中。使用 `DBMS_SPM` 可以将基线计划的状态从已接受更改为未接受（以及从未接受更改为已接受），还可以从登台表导出基线计划，然后使用导出的基线计划将 SQL 计划基线加载到其它数据库中。

也可以手动装载一个存在的 SQL Plan 作为 SQL Plan Baseline，手动装载的 SQL Plan 并不校验它的性能：

```
--从 SQL Tuning Set 中装载：
```

```
DECLARE
```

```
my_plans pls_integer;
```

```
BEGIN
```

```
my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(sqlset_name => 'tset1');  
END;  
/  
  
--从 Cursor Cache 中装载  
  
DECLARE my_plans pls_integer;  
BEGIN  
    my_plans := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(sql_id => '7qqnad1j615m7');  
END;  
/
```

在 SQL 计划基线演化阶段，Oracle DB 会按常规方式评估新计划的性能，并将性能较好的计划集成到 SQL 计划基线中。

优化程序为 SQL 语句找到新的计划时，会将该计划作为未接受的计划添加到计划历史记录中。然后，相对于 SQL 计划基线的性能，验证该计划的性能。如果经验证某个未接受的计划不会导致性能回归（手动或自动），则该计划会被更改为已接受计划，并集成到 SQL 计划基线中。成功验证未接受计划的过程包括：对此计划的性能和从 SQL 计划基线中选择一个计划的性能进行比较，确保其性能更佳。

演化 SQL 计划基线的方式有两种：

(1) 使用 DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE 函数。该函数将返回一个报表，显示是否已将一些现有的历史记录计划移到了计划基线中。也可以在历史记录中指定要测试的特定计划。

(2) 运行 SQL 优化指导：通过使用 SQL 优化指导手动或自动优化 SQL 语句，演化 SQL 计划基线。SQL 优化指导发现已优化的计划，并确认其性能优于从相应的 SQL 计划基线中选择的计划的性能时，就会生成一个建议案以接受 SQL 概要文件。接受了该 SQL 概要文件后，会将已优化的计划添加到相应的 SQL 计划基线中。

在 SQL Plan Baselines 的演变阶段，Oracle 评估新的 Plan 的性能并将性能较好的 Plan 存放 SQL Plan Baselines 中，可以使用 dbms_spm package 的过程 EVOLVE_SQL_PLAN_BASELINE 将新的 SQL Plan 存入已经存在的 SQL Plan Baselines 中 新的 Plan 将会作为已经 Accept Plan 加入到 SQL Plan Baselines 中。

```
SET SERVEROUTPUT ON
SET LONG 10000
DECLARE report clob;
BEGIN report := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (sql_handle => 'SYS_SQL_593bc74fca8e6738');
DBMS_OUTPUT.PUT_LINE(report);
END;
/
```

如果将计划添加到计划历史记录中，则该计划将与一些重要的属性关联：

(1) SIGNATURE、SQL_HANDLE、SQL_TEXT 和 PLAN_NAME 是搜索操作的重要标识符。

(2) 使用 ORIGIN 可以确定计划是自动捕获的 (AUTO-CAPTURE)、手动演化的 (MANUALLOAD)、通过 SQL 优化指导自动演化的 (MANUAL-SQLTUNE) 还是通过自动 SQL 优化自动演化的 (AUTO-SQLTUNE)。

(3) **ENABLED** 和 **ACCEPTED**：ENABLED 属性表示计划已启用，可供优化程序使用。如果未设置 **ENABLED**，则系统将不考虑此计划。ACCEPTED 属性表示用户在将计划更改为 ACCEPTED 时计划已经过验证为有效计划（系统自动进行的或用户手动进行的）。如果将某个计划更改为 ACCEPTED，则仅当使用

DBMS_SPM.ALTER_SQL_PLAN_BASELINE()更改其状态时，该计划才是非 ACCEPTED 的。可以通过删除 ENABLED 设置暂时禁用 ACCEPTED 计划。**计划必须为**

ENABLED 和 **ACCEPTED**，优化程序才会考虑使用它。

(4) **FIXED** 表示优化程序仅考虑标记为 **FIXED** 的计划，而不考虑其它计划。例如，如果有 10 个基线计划，其中的三个计划被标记为 **FIXED**，则优化程序将仅使用这三个计划中的最佳计划，而忽略其它所有计划。如果某个 SQL 计划基线至少包含一个已启用的已修复计划，则该 SQL 计划基线就是 **FIXED** 的。如果在修复的 SQL 计划基线中添加了新计划，则在手动将这些新计划声明为 **FIXED** 之前，无法使用这些新计划。

可以使用 **DBA_SQL_PLAN_BASELINES** 视图查看每个计划的属性。然后，可以使用 **DBMS_SPM.ALTER_SQL_PLAN_BASELINE** 函数更改其中的某些属性。也可以使用 **DBMS_SPM.DROP_SQL_PLAN_BASELINE** 函数删除计划或整个计划历史记录。

注：**DBA_SQL_PLAN_BASELINES** 视图包含了一些附加属性；使用这些属性可以确定各个计划的上次使用时间，以及是否应自动清除某个计划。

如果使用的是自动计划捕获，则第一次将某个 SQL 语句标识为可重复时，其最佳成本计划将被添加到对应的 SQL 计划基线中。然后，该计划将用于执行相应的语句。

如果某个 SQL 语句存在计划基线，并且初始化参数 **OPTIMIZER_USE_SQL_PLAN_BASELINES** 被设置为 **TRUE**（默认值），则优化程序将使用比较计划选择策略。

每次编译 SQL 语句时，优化程序都会先使用传统的基于成本的搜索方法建立一个最佳成本计划，然后尝试在 SQL 计划基线中找到一个匹配的计划。如果找到了匹配的计划，则优化程序将照常继续运行。如果未找到匹配的计划，则优化程序会先将新计划添加到计划历史记录中，然后计算 SQL 计划基线中各个已接受的计划的成本，并选择成本最低的那个计划。使用随各个已接受的计划存储的大纲复制这些已接受的计划。因此，对于 SQL 语句来说，拥有一个 SQL 计划基线的好处就是：优化程序始终选择该 SQL 计划基线中的一个已接受的计划。

通过 SQL 计划管理，优化程序可以生成最佳成本计划，也可以生成基线计划。此信息将被转储在有关解释计划的 **plan_table** 的 **other_xml** 列中。

此外，还可以使用新的 `dbms_xplain.display_sql_plan_baseline` 函数，显示某个计划基线中给定 `sql_handle` 的一个或多个执行计划。如果还指定了 `plan_name`，则将显示相应的执行计划。

注：为了保留向后兼容性，如果用户会话的某个 SQL 语句的存储大纲对是活动的，则将使用此存储大纲编译该语句。此外，即使为会话启用了自动计划捕获，也不将优化程序使用存储大纲生成的计划存储在 SMB 中。

虽然存储大纲没有任何显式迁移过程，但可使用 `DBMS_SPM` 程序包中的 `LOAD_PLAN_FROM_CURSOR_CACHE` 过程或 `LOAD_PLAN_FROM_SQLSET` 过程将其迁移到 SQL 计划基线。迁移完成时，应禁用或删除原始的存储大纲。

在 SQL Plan 选择阶段，SQL 每一次编译，优化器使用基于成本的方式，建立一下 best-cost 的执行计划，然后去匹配 SQL Plan Baselines 中的 SQL Plan，如果找到了匹配的 SQL Plan，则会使用这个执行计划，如果没有找到匹配的 SQL Plan，优化器就会去 SQL Plan History 中去搜索成本最低的 SQL Plan，如果优化器在 SQL Plan History 中找不到任务匹配的 SQL Plan，则该 SQL Plan 被作为一个 Non-Accept Plan 被存入 SQL Plan History，新的 SQL Plan 直到它被验证不会引起一下性能问题才会被使用。

SPM 相关的数据字典：

```
SELECT * FROM dba_sql_plan_baselines;  
SELECT * FROM dba_sqlset_plans;  
SELECT * FROM dba_advisor_sqlplans;
```

2.3.2 删除 Plans 和 Baselines

DROP_SQL_PLAN_BASELINE 函数可以从 baselines 中 drop 某个执行的执行计划，如果不执行 plan name，那么会 drop 所有的 plan。即 drop 了 baseline。

Parameter	Description
sql_handle	SQL statement handle. It identifies plans associated with a SQL statement that are to be dropped. If NULL then plan_name must be specified.
plan_name	Plan name. It identifies a specific plan. Default NULL means to drop all plans associated with the SQL statement identified by sql_handle.

--删除某个 SQL 的 baseline

```
SET SERVEROUTPUT ON
DECLARE
  l_plans_dropped PLS_INTEGER;
BEGIN
  l_plans_dropped := DBMS_SPM.drop_sql_plan_baseline (
    sql_handle => 'SQL_7b76323ad90440b9',
    plan_name  => NULL);
  DBMS_OUTPUT.put_line(l_plans_dropped);
END;
/
```

--删除所有 baseline

```
declare

  v_plan_num PLS_INTEGER;
```

```
begin

    for cur in (SELECT * FROM dba_sql_plan_baselines) loop
        begin
            v_plan_num := dbms_spm.drop_sql_plan_baseline(sql_handle => cur.sql_handle);
        exception
            when others then
                null;
        end;
    end loop;

end;
/
```

2.3.3 SPM 使用演示

--取消自动捕获，也可以不取消自动捕捉：

```
show parameter baselines
ALTER SYSTEM SET OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=FALSE;
```

```
[ZHLHRSPMDB2:oracle]:/oracle>ORACLE_SID=d1hr
[ZHLHRSPMDB2:oracle]:/oracle>sqlplus / as sysdba
```

SQL*Plus: Release 11.2.0.4.0 Production on Thu May 26 15:47:55 2016

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production

With the Partitioning, Real Application Clusters, OLAP, Data Mining
and Real Application Testing options

```
SYS@dlhr> conn lhr/lhr
```

```
Connected.
```

```
LHR@dlhr>
```

```
LHR@dlhr>
```

```
LHR@dlhr>
```

```
LHR@dlhr> select * from v$version;
```

```
BANNER
```

```
-----  
Oracle Database 11g Enterprise Edition Release 11.2.0.4.0 - 64bit Production
```

```
PL/SQL Release 11.2.0.4.0 - Production
```

```
CORE 11.2.0.4.0 Production
```

```
TNS for IBM/AIX RISC System/6000: Version 11.2.0.4.0 - Production
```

```
NLSRTL Version 11.2.0.4.0 - Production
```

```
LHR@dlhr> show parameter baselines
```

```
NAME                                TYPE        VALUE
```

```
-----
```

```
optimizer_capture_sql_plan_baselines boolean      TRUE
```

```
optimizer_use_sql_plan_baselines    boolean      TRUE
```

```
LHR@dlhr> ALTER SYSTEM SET OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=FALSE;
```

```
System altered.
```

--创建表并插入数据：

```
CREATE TABLE tb_spm_test_lhr (  
  id          NUMBER,  
  description VARCHAR2(50)  
);
```

```
DECLARE
```



```
TYPE t_tab IS TABLE OF tb_spm_test_lhr%ROWTYPE;
l_tab t_tab := t_TAB();
BEGIN
  FOR i IN 1 .. 10000 LOOP
    l_tab.extend;
    l_tab(l_tab.last).id := i;
    l_tab(l_tab.last).description := 'Description for ' || i;
  END LOOP;

  FORALL i IN l_tab.first .. l_tab.last
    INSERT INTO tb_spm_test_lhr VALUES l_tab(i);

  COMMIT;
END;
/

EXEC DBMS_STATS.gather_table_stats(USER, 'tb_spm_test_lhr', cascade=>TRUE);

set autot trace
SELECT description FROM   tb_spm_test_lhr WHERE id = 100;
```

```
LHR@dlhr> CREATE TABLE tb_spm_test_lhr (
  2  id          NUMBER,
  3  description VARCHAR2(50)
  4 );
```

Table created.

```
LHR@dlhr>
LHR@dlhr> DECLARE
  2  TYPE t_tab IS TABLE OF tb_spm_test_lhr%ROWTYPE;
  3  l_tab t_tab := t_TAB();
  4  BEGIN
```

```

5  FOR i IN 1 .. 10000 LOOP
6      l_tab.extend;
7      l_tab(l_tab.last).id := i;
8      l_tab(l_tab.last).description := 'Description for ' || i;
9  END LOOP;
10
11  FORALL i IN l_tab.first .. l_tab.last
12      INSERT INTO tb_spm_test_lhr VALUES l_tab(i);
13
14  COMMIT;
15  END;
16  /

```

PL/SQL procedure successfully completed.

LHR@dlhr> set autot trace

LHR@dlhr> **SELECT description FROM tb_spm_test_lhr WHERE id = 100;**

Execution Plan

Plan hash value: 2196561629

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	40	13 (0)	00:00:01
* 1	TABLE ACCESS FULL	TB_SPM_TEST_LHR	1	40	13 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("ID"=100)

Note

- dynamic sampling used for this statement (level=2)

Statistics

```

4 recursive calls
0 db block gets
94 consistent gets
0 physical reads
0 redo size

```

```
546 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
  2 SQL*Net roundtrips to/from client
  0 sorts (memory)
  0 sorts (disk)
  1 rows processed
```

-----获取刚才查询的 SQL_ID:

```
set autot off
col SQL_TEXT format a100
select distinct a.SQL_ID,a.SQL_TEXT from v$sql a
WHERE a.SQL_TEXT like '%SELECT description FROM  tb_spm_test_lhr WHERE  id = 100%'
and a.SQL_TEXT not like '%v$sql%'
AND    sql_text NOT LIKE '%EXPLAIN%';
```

```
LHR@dlhr> set autot off
LHR@dlhr> col SQL_TEXT format a100
LHR@dlhr> select distinct a.SQL_ID,a.SQL_TEXT from v$sql a
  2 WHERE a.SQL_TEXT like '%SELECT description FROM  tb_spm_test_lhr WHERE  id = 100%'
  3 and a.SQL_TEXT not like '%v$sql%'
  4 AND    sql_text NOT LIKE '%EXPLAIN%';
```

SQL_ID	SQL_TEXT
garkwg3yy2ram	SELECT description FROM tb_spm_test_lhr WHERE id = 100

----使用 SQL_ID 从 cursor cache 中手工捕获执行计划：

```
SET SERVEROUTPUT ON
DECLARE
  l_plans_loaded  PLS_INTEGER;
BEGIN
  l_plans_loaded := DBMS_SPM.load_plans_from_cursor_cache(
```

```
    sql_id => '&sql_id');  
    DBMS_OUTPUT.put_line('Plans Loaded: ' || l_plans_loaded);  
END;  
/
```

-- --使用 DBA_SQL_PLAN_BASELINES 视图查看 SPM 信息：

```
col sql_handle for a35  
col plan_name for a35  
set lin 300  
SELECT SQL_HANDLE,plan_name,origin,enabled,accepted,fixed  
FROM    dba_sql_plan_baselines  
WHERE   sql_text LIKE '%tb_spm_test_lhr%'  
AND     sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

--刷新 Share Pool , 使下次 SQL 执行时必须进行硬解析：

```
ALTER SYSTEM FLUSH SHARED_POOL;
```

```
LHR@dlhr> SET SERVEROUTPUT ON
```

```
LHR@dlhr> DECLARE
```

```
2   l_plans_loaded PLS_INTEGER;
```

```
3 BEGIN
```

```
4   l_plans_loaded := DBMS_SPM.load_plans_from_cursor_cache(
```

```
5     sql_id => '&sql_id');
```

```
6   DBMS_OUTPUT.put_line('Plans Loaded: ' || l_plans_loaded);
```

```
7 END;
```

```
8 /
```

```
Enter value for sql_id: garkwg3yy2ram
```

```
old 5:   sql_id => '&sql_id';
```

```
new 5:   sql_id => 'garkwg3yy2ram');
```

```
Plans Loaded: 1
```

```
PL/SQL procedure successfully completed.
```

```
LHR@dlhr> col sql_handle for a35
```

```
LHR@dlhr> col plan_name for a35
```

```
LHR@dlhr> set lin 300
```

```
LHR@dlhr> SELECT sql_handle, plan_name,enabled, accepted
```

```

2 FROM   dba_sql_plan_baselines
3 WHERE  sql_text LIKE '%tb_spm_test_lhr%'
4 AND    sql_text NOT LIKE '%dba_sql_plan_baselines%';

```

SQL_HANDLE	PLAN_NAME	ENA	ACC
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvws3184920d2	YES	YES

```
LHR@dlhr> ALTER SYSTEM FLUSH SHARED_POOL;
```

System altered.

```
LHR@dlhr> set autot trace
```

```
SELECT description FROM   tb_spm_test_lhr WHERE   id = 100;
```

```
LHR@dlhr>
```

Execution Plan

Plan hash value: 2196561629

Id	Operation	Name	Rows	Bytes	Cost	(%CPU)	Time
0	SELECT STATEMENT		1	40	13	(0)	00:00:01
* 1	TABLE ACCESS FULL	TB_SPM_TEST_LHR	1	40	13	(0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("ID"=100)

Note

- dynamic sampling used for this statement (level=2)

- SQL plan baseline "SQL_PLAN_4y6fmtxbvws3184920d2" used for this statement

Statistics

```

555 recursive calls
16  db block gets
667 consistent gets
0   physical reads
3056 redo size
546 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2   SQL*Net roundtrips to/from client

```

```
32  sorts (memory)
0   sorts (disk)
1   rows processed
```

---创建索引，收集统计信息，并查询相同的 SQL:

```
CREATE INDEX spm_test_tab_idx ON tb_spm_test_lhr(id);
EXEC DBMS_STATS.gather_table_stats(USER,'tb_spm_test_lhr', cascade=>TRUE);

set autot trace
SELECT description FROM   tb_spm_test_lhr WHERE  id = 100;
```

```
LHR@dlhr> CREATE INDEX spm_test_tab_idx ON tb_spm_test_lhr(id);
```

Index created.

```
LHR@dlhr> EXEC DBMS_STATS.gather_table_stats(USER,'tb_spm_test_lhr', cascade=>TRUE);
```

PL/SQL procedure successfully completed.

```
LHR@dlhr>
```

```
LHR@dlhr>
```

```
LHR@dlhr>
```

```
LHR@dlhr> set autot trace
```

```
LHR@dlhr> SELECT description FROM   tb_spm_test_lhr WHERE  id = 100;
```

Execution Plan

Plan hash value: 2196561629

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	25	13 (0)	00:00:01
* 1	TABLE ACCESS FULL	TB_SPM_TEST_LHR	1	25	13 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("ID"=100)

Note

- SQL plan baseline "SQL_PLAN_4y6fmtxbvws3184920d2" used for this statement

Statistics

```
640 recursive calls
39 db block gets
493 consistent gets
2 physical reads
12268 redo size
546 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
10 sorts (memory)
0 sorts (disk)
1 rows processed
```

--这里我们创建了索引，但是这里还是走的全表扫描，这里使用索引明显才是最优的方案。

--查看 SPM 视图：

```
set autot off
col sql_handle for a35
col plan_name for a35
set lin 300
SELECT SQL_HANDLE,plan_name,origin,enabled,accepted,fixed
FROM dba_sql_plan_baselines
WHERE sql_text LIKE '%tb_spm_test_lhr%'
AND sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

```
LHR@dlhr> set autot off
LHR@dlhr> col sql_handle for a35
LHR@dlhr> col plan_name for a35
LHR@dlhr> set lin 300
LHR@dlhr> SELECT sql_handle, plan_name,enabled, accepted
2 FROM dba_sql_plan_baselines
3 WHERE sql_text LIKE '%tb_spm_test_lhr%'
4 AND sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

SQL_HANDLE	PLAN_NAME	ENA	ACC
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws3184920d2	YES	YES
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws38b725570	YES	NO

--通过 baselines 查询的结果，可以看到我们的 SQL 产生了 2 条执行计划。但是我们认为最优的执行计划并没有被标记为 ACCEPT，所以没有使用。

下边我们演化执行计划：演化就是将 cost 低的执行计划标记为 accept

```
LHR@dlhr> SET LONG 10000
LHR@dlhr> SELECT DBMS_SPM.evolve_sql_plan_baseline(sql_handle => '&sql_handle') FROM dual;
Enter value for sql_handle: SQL_4f19d3cf57be7303
old 1: SELECT DBMS_SPM.evolve_sql_plan_baseline(sql_handle => '&sql_handle') FROM dual
new 1: SELECT DBMS_SPM.evolve_sql_plan_baseline(sql_handle => 'SQL_4f19d3cf57be7303') FROM dual
```

```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(SQL_HANDLE=>'SQL_4F19D3CF57BE7303')
```

Evolve SQL Plan Baseline Report

Inputs:

```
SQL_HANDLE = SQL_4f19d3cf57be7303
PLAN_NAME  =
TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
VERIFY     = YES
COMMIT     = YES
```

```
Plan: SQL_PLAN_4y6fmtxbvwws38b725570
```

```
Plan was verified: Time used .018 seconds.
```


Plan passed performance criterion: 15 times better than baseline plan.
Plan was changed to an accepted plan.

	Baseline Plan	Test Plan	Stats Ratio
	-----	-----	-----
Execution Status:	COMPLETE	COMPLETE	
Rows Processed:	1	1	
Elapsed Time(ms):	.308	.025	12.32
CPU Time(ms):	.164	.015	10.93
Buffer Gets:	45	3	15
Physical Read Requests:	0	0	
Physical Write Requests:	0	0	
Physical Read Bytes:	0	0	
Physical Write Bytes:	0	0	
Executions:	1	1	

Report Summary

Number of plans verified: 1
Number of plans accepted: 1

--再次查看 DBA_SQL_PLAN_BASELINES 视图：

```
set autot off
col sql_handle for a35
col plan_name for a35
set lin 300
SELECT SQL_HANDLE,plan_name,origin,enabled,accepted,fixed
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE '%tb_spm_test_lhr%'
AND    sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

```
LHR@dlhr> set autot off
LHR@dlhr> col sql_handle for a35
LHR@dlhr> col plan_name for a35
LHR@dlhr> set lin 300
LHR@dlhr> SELECT sql_handle, plan_name,enabled, accepted
2 FROM   dba_sql_plan_baselines
```

```

3 WHERE sql_text LIKE '%tb_spm_test_lhr%'
4 AND   sql_text NOT LIKE '%dba_sql_plan_baselines%';

```

SQL_HANDLE	PLAN_NAME	ENA	ACC
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws3184920d2	YES	YES
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws38b725570	YES	YES

--再次执行 SQL:

```
set autot trace
```

```
SELECT description FROM   tb_spm_test_lhr WHERE   id = 100;
```

```
LHR@dlhr> set autot trace
```

```
LHR@dlhr> SELECT description FROM   tb_spm_test_lhr WHERE   id = 100;
```

Execution Plan

Plan hash value: 2587945646

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	25	2 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	TB_SPM_TEST_LHR	1	25	2 (0)	00:00:01
* 2	INDEX RANGE SCAN	SPM_TEST_TAB_IDX	1		1 (0)	00:00:01

Predicate Information (identified by operation id):

2 - access("ID"=100)

Note

- SQL plan baseline "SQL_PLAN_4y6fmtxbvwws38b725570" used for this statement

Statistics

```

13 recursive calls
14 db block gets

```

```
18 consistent gets
0 physical reads
3048 redo size
553 bytes sent via SQL*Net to client
519 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
```

--这次正确的使用了索引。 因为只有标记为 `ENABLE` 和 `ACCEPT` 的 `plan` 才可以被使用。

下面示例将我们的第一个走全表扫描的执行计划标记为 `fixed`。 标记为 `fixed` 的执行计划会被优先使用。`FIXED` 表示优化程序仅考虑标记为 `FIXED` 的计划，而不考虑其它计划。例如，如果有 10 个基线计划，其中的三个计划被标记为 `FIXED`，则优化程序将仅使用这三个计划中的最佳计划，而忽略其它所有计划。如果某个 SQL 计划基线至少包含一个已启用的已修复计划，则该 SQL 计划基线就是 `FIXED` 的。如果在修复的 SQL 计划基线中添加了新计划，则在手动将这些新计划声明为 `FIXED` 之前，无法使用这些新计划。

```
set autot off
select * from table(dbms_xplan.display_sql_plan_baseline (sql_handle => '&sql_handle', format => 'basic'));

SET SERVEROUTPUT ON
DECLARE
l_plans_altered PLS_INTEGER;
BEGIN
l_plans_altered := DBMS_SPM.alter_sql_plan_baseline(
    sql_handle      => '&sql_handle',
    plan_name       => '&plan_name',
    attribute_name  => 'fixed',
    attribute_value => 'YES');
```

```
DBMS_OUTPUT.put_line('Plans Altered: ' || l_plans_altered);  
END;  
/
```

```
LHR@dlhr> SET SERVEROUTPUT ON
```

```
LHR@dlhr> DECLARE
```

```
2  l_plans_altered PLS_INTEGER;  
3  BEGIN  
4  l_plans_altered := DBMS_SPM.alter_sql_plan_baseline(  
5    sql_handle      => '&sql_handle',  
6    plan_name       => '&plan_name',  
7    attribute_name  => 'fixed',  
8    attribute_value => 'YES');  
9  
10 DBMS_OUTPUT.put_line('Plans Altered: ' || l_plans_altered);  
11 END;  
12 /
```

```
Enter value for sql_handle: SQL_4f19d3cf57be7303
```

```
old 5: sql_handle => '&sql_handle',
```

```
new 5: sql_handle => 'SQL_4f19d3cf57be7303',
```

```
Enter value for plan_name: SQL_PLAN_4y6fmtxbvwws3184920d2
```

```
old 6: plan_name => '&plan_name',
```

```
new 6: plan_name => 'SQL_PLAN_4y6fmtxbvwws3184920d2',
```

```
Plans Altered: 1
```

```
PL/SQL procedure successfully completed.
```

--验证:

```
set autot off
```

```
col sql_handle for a35
```

```
col plan_name for a35
```

```
set lin 300
```

```
SELECT SQL_HANDLE,plan_name,origin,enabled,accepted,fixed
```

```
FROM dba_sql_plan_baselines
```

```
WHERE sql_text LIKE '%tb_spm_test_lhr%'
```

```
AND sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

```
LHR@dlhr> set autot off
LHR@dlhr> select * from table(dbms_xplan.display_sql_plan_baseline (sql_handle => '&sql_handle', format => 'basic'));
Enter value for sql_handle: SQL_4f19d3cf57be7303
old 1: select * from table(dbms_xplan.display_sql_plan_baseline (sql_handle => '&sql_handle', format => 'basic'))
new 1: select * from table(dbms_xplan.display_sql_plan_baseline (sql_handle => 'SQL_4f19d3cf57be7303', format => 'basic'))
```

PLAN_TABLE_OUTPUT

```
-----
SQL handle: SQL_4f19d3cf57be7303
SQL text: SELECT description FROM   tb_spm_test_lhr WHERE   id = 100
-----
```

```
-----
Plan name: SQL_PLAN_4y6fmtxbvws3184920d2      Plan id: 407445714
Enabled: YES      Fixed: YES      Accepted: YES      Origin: MANUAL-LOAD
-----
```

Plan hash value: 2196561629

```
-----
| Id | Operation          | Name                |
-----
|  0 | SELECT STATEMENT   |                     |
|  1 | TABLE ACCESS FULL| TB_SPM_TEST_LHR    |
-----
```

```
-----
Plan name: SQL_PLAN_4y6fmtxbvws38b725570      Plan id: 2339526000
Enabled: YES      Fixed: NO      Accepted: YES      Origin: AUTO-CAPTURE
-----
```

Plan hash value: 2587945646

```
-----
| Id | Operation                | Name                |
-----
|  0 | SELECT STATEMENT         |                     |
|  1 | TABLE ACCESS BY INDEX ROWID| TB_SPM_TEST_LHR    |
|  2 | INDEX RANGE SCAN         | SPM_TEST_TAB_IDX   |
-----
```

34 rows selected.

```
LHR@dlhr> set autot off
```

```
LHR@dlhr> col sql_handle for a35
LHR@dlhr> col plan_name for a35
LHR@dlhr> set lin 300
LHR@dlhr> SELECT SQL_HANDLE, plan_name, origin, enabled, accepted, fixed
  2 FROM    dba_sql_plan_baselines
  3 WHERE    sql_text LIKE '%tb_spm_test_lhr%'
  4 AND      sql_text NOT LIKE '%dba_sql_plan_baselines%';
```

SQL_HANDLE	PLAN_NAME	ORIGIN	ENA	ACC	FIX
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws3184920d2	MANUAL-LOAD	YES	YES	YES
SQL_4f19d3cf57be7303	SQL_PLAN_4y6fmtxbvwws38b725570	AUTO-CAPTURE	YES	YES	NO

--再次查看我们之前的 SQL:

```
set autot trace
SELECT description FROM    tb_spm_test_lhr WHERE    id = 100;
```

```
LHR@dlhr> set autot trace
LHR@dlhr> SELECT description FROM    tb_spm_test_lhr WHERE    id = 100;
```

Execution Plan

Plan hash value: 2196561629

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	25	13 (0)	00:00:01
* 1	TABLE ACCESS FULL	TB_SPM_TEST_LHR	1	25	13 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("ID"=100)

Note

- SQL plan baseline "SQL_PLAN_4y6fmtxbvwws3184920d2" used for this statement

Statistics

```
-----
      6 recursive calls
      8 db block gets
     46 consistent gets
      0 physical reads
      0 redo size
    546 bytes sent via SQL*Net to client
    519 bytes received via SQL*Net from client
      2 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
      1 rows processed
```

--这里已经走了全表扫描，根据前边的示例，我们知道这里走索引会更优，但因为我们将走全表扫描的执行计划设置为 `fixed`，所以优先使用这个执行计划。

2.4 总结

1、`coe_xfr_sql_profile.sql` 脚本需要从 MOS 下载，小麦苗已经下载放在了云盘，大家可以去下载，地址你懂的

2、`outline` 是 9i 的内容，`SQL Profile` 是 10g 的新特性，`SPM` 是 11g 的新特性

About Me

本文作者：小麦苗，只专注于数据库的技术，更注重技术的运用

ITPUB BLOG：<http://blog.itpub.net/26736162>

本文地址：<http://blog.itpub.net/26736162/viewspace-2107604/>

本文 pdf 版：<http://yunpan.cn/cdEQedhCs2kFz>（提取码：ed9b）

QQ：642808185 若加 QQ 请注明您所正在读的文章标题

于 2016-05-18 10:00~ 2016-05-26 19:00 在中行完成

【版权所有，文章允许转载，但须以链接方式注明源地址，否则追究法律责任】

.....

