Sql Trace 与 执行计划术语解读

作者: yunfang.shang

创建日期: 2006-07-22

最近更新: 2006-11-26

控制号: MD070

版本: 1

审批:

全富汉得 _____

_____

拷贝号 ____

文档控制

变更记录

日期	作者	版本	变更
22-Jul-06	Yunfang.shang	1.0	Create

审阅

姓名	职位

分发

拷贝号	姓名	地点
1		
1		
1		
1		

目录

第一章：执行计划

本章对 Oracle 和 优化器进行了简要的介绍，对执行计划中的术语进行了详细的解释

Oracle 的优化器

Oracle 在执行一个 SQL 之前,首先要分析一下语句的执行计划,然后再按执行计划去执行。分析语句的执行计划的工作是由优化器(Optimizer)来完成的。不同的情况,一条 SQL 可能有多种执行计划,但在某一时点,一定只有一种执行计划是最优的,花费时间是最少的。相信你一定会用 Pl/sql Developer、Toad 等工具去看一个语句的执行计划,不过你可能对 Rule、Choose、First rows、All rows 这几项有疑问,因为我当初也是这样的,那时我也疑惑为什么选了以上的不同的项,执行计划就变了?

1、优化器的优化方式

Oracle 的优化器共有两种的优化方式,即基于规则的优化方式(Rule-Based Optimization,简称为 RBO)和基于代价的优化方式(Cost-Based Optimization,简称为 CBO)。

A、RBO 方式：优化器在分析 SQL 语句时,所遵循的是 Oracle 内部预定的一些规则。比如我们常见的,当一个 where 子句中的一列有索引时去走索引。

B、CBO 方式：依词义可知,它是看语句的代价(Cost)了,这里的代价主要指 Cpu 和内存。优化器在判断是否用这种方式时,主要参照的是表及索引的统计信息。统计信息给出表的大小 、有少行、每行的长度等信息。这些统计信息起初在库内是没有的,是你在做 analyze 后才出现的,很多的时侯过期统计信息会令优化器做出一个错误的执行计划,因些我们应及时更新这些信息。在 Oracle8 及以后的版本,Oracle 列推荐用 CBO 的方式。

我们要明了,不一定走索引就是优的 ,比如一个表只有两行数据,一次 IO 就可以完成全表的检索,而此时走索引时则需要两次 IO,这时对这个表做全表扫描(full table scan)是最好的。

2、优化器的优化模式(Optermizer Mode)

优化模式包括 Rule,Choose,First rows,All rows 这四种方式,也就是我们以上所提及的。如下我解释一下：

Rule:不用多说,即走基于规则的方式。

Choolse:这是我们应观注的,默认的情况下 Oracle 用的便是这种方式。指的是当一个表或或索引有统计信息,则走 CBO 的方式,如果表或索引没统计信息,表又不是特别的小,而且相应的列有索引时,那么就走索引,走 RBO 的方式。

First Rows:它与 Choose 方式是类似的,所不同的是当一个表有统计信息时,它将是以最快的方式返回查询的最先的几行,从总体上减少了响应时间。

All Rows:也就是我们所说的 Cost 的方式,当一个表有统计信息时,它将以最快的方式返回表的所有的行,从总体上提高查询的吞吐量。没有统计信息则走基于规则的方式。

3、如何设定选用哪种优化模式

a、Instance 级别

我们可以通过在 init<SID>.ora 文件中设定 OPTIMIZER_MODE=RULE、OPTIMIZER_MODE=CHOOSE、OPTIMIZER_MODE=FIRST_ROWS、OPTIMIZER_MODE=ALL_ROWS 去选用 3 所提的四种方式,如果你没设定 OPTIMIZER_MODE 参数则默认用的是 Choose 这种方式。

B、Sessions 级别

通过 SQL> ALTER SESSION SET OPTIMIZER_MODE=<Mode>;来设定。

C、语句级别

这些需要用到 Hint,比如:

SQL> SELECT /*+ RULE */ a.userid, b.name, b.depart_name FROM tf_f_yhda a, tf_f_depart b WHERE a.userid=b.userid;

4、为什么有时一个表的某个字段明明有索引,当观察一些语的执行计划确不走索引呢？如何解决呢 ？

A、不走索引大体有以下几个原因

♀你在 Instance 级别所用的是 all_rows 的方式

♀你的表的统计信息让 Oracle 认为在 CBO 方式下不走索引更合理(最可能的原因)

♀你的表很小,上文提到过的,Oracle 的优化器认为不值得走索引。

B、解决方法

♀可以修改 init<SID>.ora 中的 OPTIMIZER_MODE 这个参数,把它改为 Rule 或 Choose,重起数据库。也可以使用 4 中所提的 Hint.

♀删除统计信息

SQL>analyze table table_name delete statistics;

♀表小不走索引是对的,不用调的。

5、其它相关

A、如何看一个表或索引是否是统计信息

SQL>SELECT * FROM user_tables WHERE table_name=<table_name> AND num_rows is not null;

SQL>SELECT * FROM user_indexes WHERE table_name=<table_name> AND num_rows is not null;

备注：user_tables 是个视图，自动过滤 Schema. 比如 要查询 GL_BALANCES 是否有统计信息，就应该以′GL′ 登录来执行上面的语句。

b、如果我们用 CBO 的方式,我们应及时去更新表和索引的统计信息,以免生形不切合实的执行计划。

SQL> ANALYZE TABLE table_name COMPUTE STATISTICS;
SQL> ANALYZE INDEX index_name ESTIMATE STATISTICS;

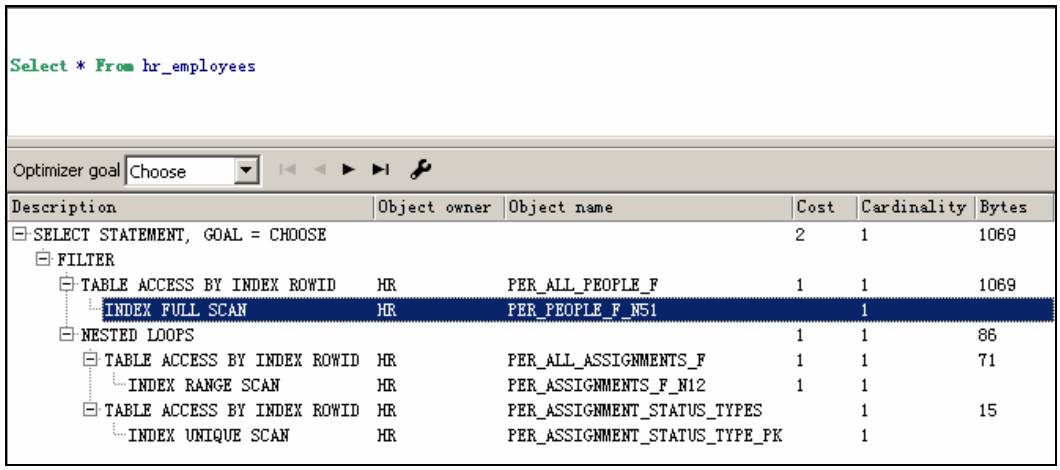
上面两句话只能在特定的 Schema 中执行，比如 GL_BALANCES 表，要以用户名'GL'登录.

一般我们使用 DBMS_STATS.GATHER_TABLE_STATS(<SCHEMA>,<TABLE>) 来收集一个表及其索引的统计信息。 比如:
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS('GL','GL_BALANCES');

执行计划解读

Oracle 用来运行一个语句的步骤就叫做执行计划（execution plan），执行计划包含了语句所涉及的每个表的访问路径和连接顺序。

我们一般是用 PL/SQL Developer 来查看某条语句的执行计划。在 Sql window 中，输入要执行的 sql 语句，然后按 F5 可以得到执行计划。如下图：



Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			2	1	1069
FILTER					
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_PEOPLE_F	1	1	1069
INDEX FULL SCAN	HR	PER_PEOPLE_F_N51		1	
NESTED LOOPS			1	1	86
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_ASSIGNMENTS_F	1	1	71
INDEX RANGE SCAN	HR	PER_ASSIGNMENTS_F_N12	1	1	
TABLE ACCESS BY INDEX ROWID	HR	PER_ASSIGNMENT_STATUS_TYPES		1	15
INDEX UNIQUE SCAN	HR	PER_ASSIGNMENT_STATUS_TYPE_PK		1	

如何解读执行计划？

你需要按照从里到外,从上到下的次序解读分析的结果. EXPLAIN PLAN 分析的结果是用缩进的格式排列的, 最内部的操作将被最先解读, 如果两个操作处于同一层中,带有最小操作号的将被首先执行. 在 PL/SQL developer 中，执行步骤可以通过 ◀ 和 ▶ 来查看上一步和下一步，如上图。

NESTED LOOP 是少数不按照上述规则处理的操作, 正确的执行路径是检查对 NESTED LOOP 提供数据的操作,其中操作号最小的将被最先处理.

下面我们对执行计划中的常见术语进行解释：

Full Table Scans（全表扫描）：

执行计划中的“**Full Table Scans**” 是指全表扫描。全表扫描的工作机理是这样的：首先，**Oracle** 的 **I/O** 是针对数据块的,通常一个数据块中存储着多条记录，被请求的记录要么聚集在少数几个块中，要么分散在大量的数据块中。而 **oracle** 对某个表进行全表扫描时，究竟应该读哪些数据块 是根据 全表扫描范围的标记**-HWM(High Water Mark)** 进行的。

全表扫描将读取 **HWM** 之下的所有数据块，访问表中的所有行，每一行都要经 **WHERE** 子句判断是否满足检索条件。当 **Oracle** 执行全表扫描时会按顺序读取每个块且只读一次，因此如果能够一次读取多个数据块，可以提高扫描效率，初始化参数

DB_FILE_MULTIBLOCK_READ_COUNT 用来设置在一次 **I/O** 中可以读取数据块的最大数量。

当一个表被大量删除记录之后，**HWM** 下面的大量数据块是空的，此时若对此表进行全表扫描，**Oracle** 仍然会读到 **HWM** 位置，会对全表扫描的性能产生极坏的影响。

1、无可用索引

如下面例子：

SELECT last_name, first_name FROM employees WHERE UPPER(last_name)='TOM'

last_name 字段有索引，但在查询中使用了函数，因此该查询不会使用索引。如果想让这个查询走索引，则需要建立函数索引 **create index ind_upper_lastname on last_name (upper(last_name))**。特别要注意的是隐式转换，比如 **colx** 字段是 **varchar2** 型但存放数字：**where colx=123456**，这时会发生隐式转换 **TO_NUMBER(colx)**，此时 **colx** 上的索引也会失效。

2、大量数据

如果优化器认为查询将会访问表中绝大多数的数据块，此时就算索引是可用的也会使用全表扫描。

3、小表

如果一个表 **HWM** 之下的数据块比 **DB_FILE_MULTIBLOCK_READ_COUNT** 要少，只需要一次 **I/O** 就能扫完，则使用全表扫描要比使用索引的成本低，此时会使用全表扫描。 如果有这样小表访问频率又高，通常把它固定在内存中为好 **alter table table_name storage(buffer_pool keep)**。

4、并行

如果在表一级设置了较高的并行度，如 **alter table table_name parallel(degree 10)**，通常会使 **CBO** 错误的选择全表扫描。通常不建议在表级的设置并行。并行查询通常可以提高全表扫描的性能，建议在语句级用 **HINTS** 来实现并行，如**/*+full(table_name) parallel(table_name degree)*/**。

5、全表扫描 **hints**

如果想强制优化器使用全表扫描可以用提示 **FULL**。

*小知识：关于 **HWM** 可以用下图解释：*

*首先要理解一个概念，即 **oracle** 数据块（**Block**）的组织方式：任何一个需要占用磁盘空间的数据库对象（比如表，索引等），从磁盘空间角度上看都体现为一个段（**Segment**），一个段由多个扩展（**Extent**）组成, **oracle** 的 **Extent** 是逻辑上的存储单位。一个 **Extent** 包含 **N** 个连续的 **BLOCK**． **N** 的数量取决于建表时指定的 **Next ExtentS** 的大小。比如 **Next Extents** 是 **64K** .而一个 **BLOCK** 是 **8K**， 则一个 **Extend** 包含 8 个连续的 **BLOCK.**一个 **Extend** 一旦分配给某个 **Table** 就不能再分配给其他 **Table.***

*每个段（**Segment**）的第一个扩展（**Extent**）的第一个数据块（**Block**）是被 **oracle** 系统保留的，不能用于存储用户数据。这个块称之为段头（**Segment header**）*

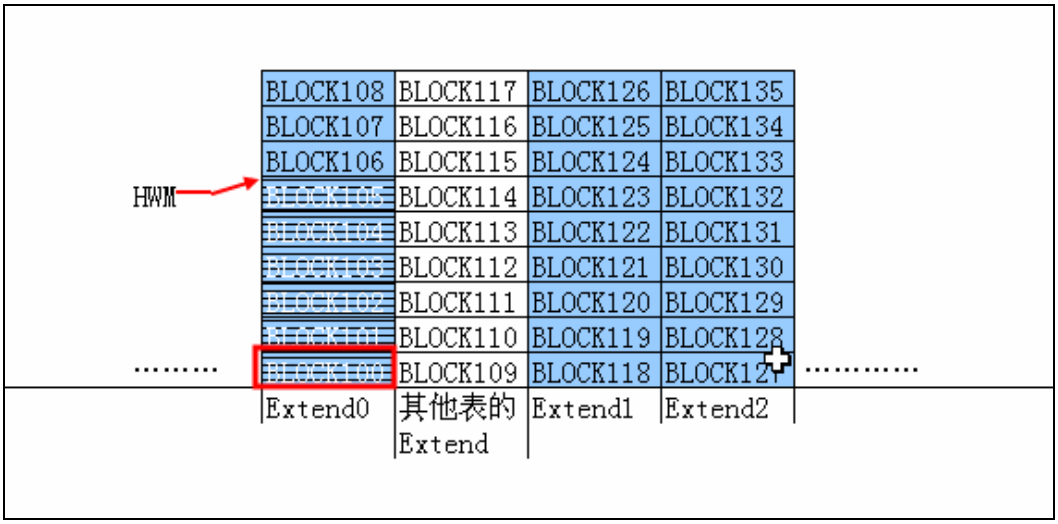
*段头（**Segment header** ） 中包含如下信息：*

***1**、段中的扩展信息表（**Extents Table**）*

***2**、剩余空间列表描述信息（**free list description**）*

***3**、高水位标记（**High water Mark (HWM)**）*

一个 ***Extend*** *中已经使用了哪些* ***BLOCK,*** *还有哪些没有使用？ 这是靠* ***High Water Mark*** *来标记的。* ***High water Mark (HWM)*** *顾名思义，非常形象，他把一个* ***Segment*** *看作一个桶。而把存储于其中的数据比作水。水平面就形成了一个高水位标记。如下图。*



在上图中， ***BLOCK100*** *就是段头。里面记录了* ***HWM*** *是* ***BLOCK105,*** *这就是说这个段中的* ***BLOCK105*** *以上的* ***BLOCK*** *（* ***BLOCK106-108,118-126,127-135*** *） 都是空余的。而在* ***BLOCK105*** *以下的* ***BLOCK(BLOKC100-105)*** *都是有数据的。*

当一个表被执行 ***Truncate*** *操作后，* ***HWM*** *标记会被置* ***0****。指向这个段的起点块位置（应该是段头块之外的第一个块）。但是* ***Delete*** *操作不会改变* ***HWM*** *的位置。*

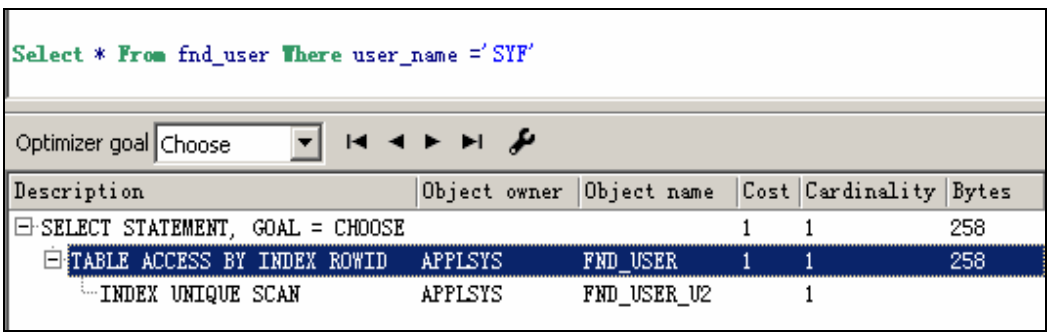
TABLE ACCESS BY INDEX ROWID (ROWID 扫描)

Rowid 就是一个记录在数据块中的位置，由于指定了记录在数据库中的精确位置，因此 **rowid** 是检索单条记录的最快方式。如果通过 **rowid** 来访问表，**Oracle** 首先需要获得被检索记录的 **rowid**，**Oracle** 可以在 **WHERE** 子句中得到 **rowid**，但更多的是通过扫描索引来获得，然后 **Oracle** 基于 **rowid** 来定位被检索的每条记录。

优化器何时使用 **Rowid** 扫描？
并不是每个索引扫描都伴随着 **rowid** 的访问，如果索引中包含了被访问的所有字段，则不再需要通过 **rowid** 来访问表。

比如解读下图中的一个执行计划就是：
第一步，根据唯一性索引 **FND_USER_U2** 检索条件 **Where User_name ='SYF'.** 找到后得到 **Rowid.**

第二步，根据 **Rowid** 找到这条记录的位置，返回这条记录所有字段。



注意：

Rowid 是 **Oracle** 表示数据存储的内部方法，它可能会由于版本的改变而改变。行迁移和行链接会导致 **rowid** 变化，**exp/imp** 也会使 **rowid** 变化。

小知识：关于 ***Rowid***

首先我们看一个 ***Rowid*** *的例子：*

```
SQL>select ROWID from gl_balances where rownum<2;
```

结果： ***AAALNvAAYAAAFHMAAA***

可以看到 ***Rowid*** *是个* ***18*** *位长度的字符窜值。实际上* ***Rowid*** *不是字符串值，这是一种特殊的数据类型。是六十四进制编码的* ***18*** *个字符显示的数值。而且这* ***18*** *个字符值还有一定的编码规则，就* ***oracle 8i*** *而言，其编码规则如下：*

数据对象编号	文件编号	块编号	行编号
OOOOOO	FFF	BBBBBB	RRR

那么什么是六十四位编码呢？就是说这 ***18*** *个字符中的每个字符允许使用如下范围中的字符：* ***A-Z,a-z,+,/*** *并且这些字符都对应着如下数值：*

A-Z <==> 0 - 25 (26)
a-z <==> 26 - 51 (26)
0-9 <==> 52 - 61 (10)
+/ <==> 62 - 63 (2)

明白了上述道理，我们可以来计算例子中的 ***Rowid*** *的数据对象编号的数值。*

AAALNvAAYAAAFHMAAA *的数据对象编号应该是前六位字符：* ***AAALNv*** *其对应的十进制数值是：*

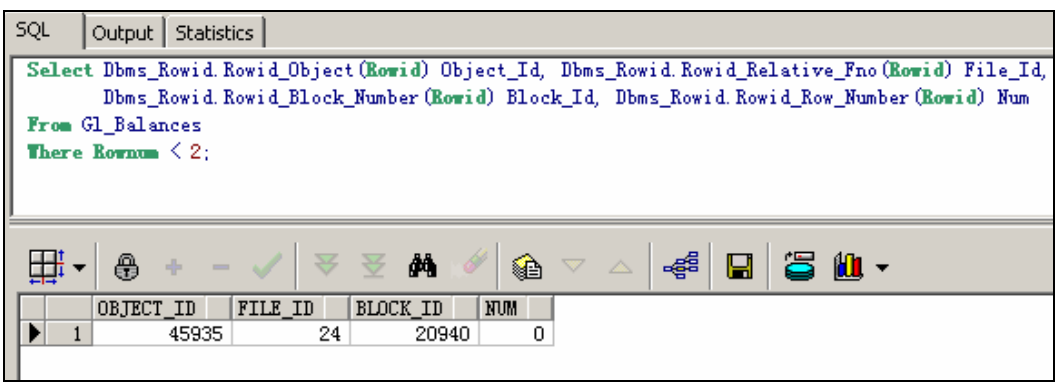
v*(64^0) = 21
N*(64^1) = 14*64=896
L*(64^2) = 12*4096=49152
A*(64^3) =0
A*(64^4) =0
A*(64^5) =0

所以该 ***Rowid*** *的数据对象编号的十进制数值就是* ***:21+896+49152+0+0+0=50069***

当然，手工进行这种转换计算是比较麻烦的， ***oracle*** *提供了一个* ***Package*** *来进行这种转换。这个* ***Package*** *是* ***dbms_rowid*** *。这个* ***package*** *中的四个函数可以以* ***rowid*** *作为参数分别返回十进制的数据对象编号，文件编号，块编号，行编号。比如，下面的例子就是返回刚才例子中的* ***Rowid*** *中包含的这些编号信息的：*

```
select dbms_rowid.rowid_object(rowid)  object_id,
dbms_rowid.rowid_relative_fno(rowid) file_id,dbms_rowid.rowid_block_number(rowid)
block_id ,dbms_rowid.rowid_row_number(rowid)  num from gl_balances where rownum<2;
```

结果如下

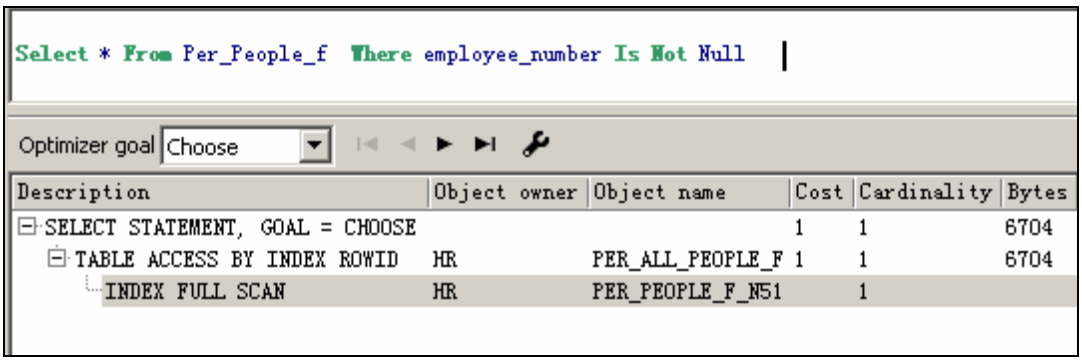


*正因为 **Rowid** 中包还了这么多信息，也有人用如下 **Sql** 语句查看一个表中已经使用了哪些块：*

```
Select distinct substr(rowid,1,15)    From gl_balances    ;
```

INDEX FULL SCAN(索引全扫描)

全索引扫描就是对整个索引进行一次逐条扫描，只需要一次 I/O。进行全索引扫描时因为有些查询条件必须对整个索引进行一次逐条扫描。 比如 在 per_people_f 的 Employee_number 字段上有个索引 PER_PEOPLE_F_N51 . 而下图的查询语句中的查询条件是 Where Employee_number is not null . 这种非空的查询条件只能逐条扫描拉。



INDEX UNIQUE SCAN(索引唯一扫描)

这种扫描通常发生在对一个主键字段或含有唯一约束的字段指定相等条件时，只有单行记录被访问。

INDEX RANGE SCAN(索引范围扫描)

索引范围扫描通常发生在对一个索引字段指定范围条件时，有多行记录被访问。是检索数据的常用方式，返回的数据返照索引字段升序排列，字段值相同的则按照 rowid 升序排列。如果在语句中指定了 order by 字句，而且排序字段是索引字段时 Oracle 将忽略 order by 子句。

INDEX XXX SCAN DESCENDING（索引降序范围扫描）

这里的 XXX 可能是 FULL,RANGE,UNIQUE.
如果在 **order by** 中指定了索引是降序排列的，或者使用了 **index_desc** 提示，**Oracle** 可能会使用索引降序范围扫描。 如下图：

```
select /*+index_desc(t PER_PEOPLE_F_N51)*/t.employee_num,t.*  from  hr_employees t where t.employee_num > 1
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			2	1	1069
FILTER					
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_PEOPLE_F	1	1	1069
INDEX FULL SCAN DESCENDING	HR	PER_PEOPLE_F_N51		1	
NESTED LOOPS			1	1	86
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_ASSIGNMENTS_F	1	1	71
INDEX RANGE SCAN	HR	PER_ASSIGNMENTS_F_N12	1	1	
TABLE ACCESS BY INDEX ROWID	HR	PER_ASSIGNMENT_STATUS_TYPES		1	15
INDEX UNIQUE SCAN	HR	PER_ASSIGNMENT_STATUS_TYPE_PK		1	

INDEX SKIP SCAN（索引跳跃式扫描）

索引跳跃式扫描发生在复合索引中，它在逻辑上将索引分离为较小的子索引，当复合索引的某一个字段不在查询中指定时，它将被跳过，从而提高索引扫描的效率。可以使用 index_ss 提示强制使用跳跃扫描。 如下图 , 索引 PER_PEOPLE_F_PK 是表 Per_all_people_f 上面的一个组合索引。

General | Columns | Keys | Checks | Indexes | Privileges

Owner	Name	Type	Columns	Compress
HR	PER_PEOPLE_F_FK1	Normal	BUSINESS_GROUP_ID	
HR	PER_PEOPLE_F_FK2	Normal	PERSON_TYPE_ID	
HR	PER_PEOPLE_F_N1	Normal	UPPER(FULL_NAME)	
HR	PER_PEOPLE_F_N50	Normal	LAST_NAME	
HR	PER_PEOPLE_F_N51	Normal	EMPLOYEE_NUMBER	
HR	PER_PEOPLE_F_N52	Normal	APPLICANT_NUMBER	
HR	PER_PEOPLE_F_N53	Normal	NATIONAL_IDENTIFIER	
HR	PER_PEOPLE_F_N54	Normal	FULL_NAME	
HR	PER_PEOPLE_F_N55	Normal	PARTY_ID	
HR	PER_PEOPLE_F_N56	Normal	NPW_NUMBER	
HR	PER_PEOPLE_F_PK	Unique	PERSON_ID, EFFECTIVE_START_DATE, EFFECTIVE_END_DATE	

```
Select /*+index_ss(Per_all_People_f per_people_f_pk)*/* From Per_all_People_f
where person_id  =61
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			1	1	6736
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_PEOPLE_F	1	1	6736
INDEX SKIP SCAN	HR	PER_PEOPLE_F_PK		1	

INDEX FAST FULL SCAN (索引快速全扫描)

索引快速全扫描只访问索引本身，而不去访问表，因此只有查询涉及的字段都包含在索引中时才会使用快速全索引扫描。满足条件后可以使用 index_ffs 提示来强制使用快速全索引扫描，快速全索引扫描只适用于 CBO。

举例如下图：

```
Select /*+index_ffs(Per_all_People_f per_people_f_pk)*/
 Person_Id, Effective_Start_Date, Effective_End_Date
From Per_All_People_f
Where Person_Id = 61
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			2	1	31
INDEX FAST FULL SCAN	HR	PER_PEOPLE_F_PK	2	1	31

值得一提的是,有时使用/*+index_ffs(<table_name>,<index_name>)*/ 其执行成本还不如不使用来得低。如下图：

```
Select  Person_Id, Effective_Start_Date, Effective_End_Date
From Per_All_People_f
Where Person_Id = 61
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			1	1	20
INDEX RANGE SCAN	HR	PER_PEOPLE_F_PK		1	20

不使用/*+index_ffs(<table_name>,<index_name>)*/时，其执行成本是 1，而使用时，其执行成本是 2。

快速全索引扫描并不能消除排序操作，因为索引键中的数据没有被排序。不同于全索引扫描，快速全索引扫描是通过多块读取的方式来读取整个索引的，并可以设置并行方式。

FILTER

当 Where 语句中有 In （Sql 子查询）, Exists （Sql 子查询）等条件时，执行计划中会有 FILTER 操作。 如下图：

```
Select  a.* From tst_a a Where Exists (Select 'X' From tst_b b Where a.Id = b.Id )
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE					
FILTER					
TABLE ACCESS FULL	APPS	TST_A			
TABLE ACCESS FULL	APPS	TST_B			

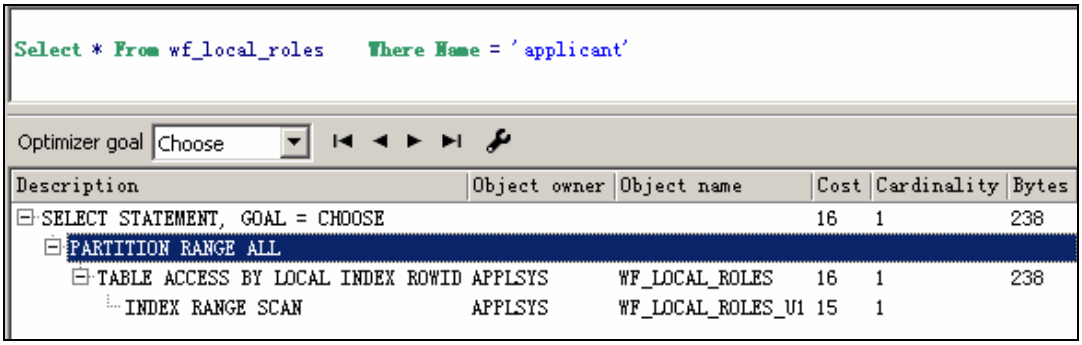
```
Select  a.* From tst_a a Where Exists (Select 'X' From tst_b b Where a.Id = b.Id )
```

PARTITION RANGE ALL

如果表是分区表，则对这个表查询的 Sql 语句的执行计划可能会出现 PATITION RANGE ALL .

比如 EBS 中 **wf_local_roles** 是个分区表。

```
Select * From wf_local_roles    Where   And orig_system_id =2
```



小知识：分区表相关的执行计划解读

网友飞茂对分区表的查询解说可谓比较形象：

普通表呢像一个小学生用的新华字典，分区表呢像一套博士辞海盒，在同一个盒子里面(表名）有若干本辞海分册（每一册就是一个分区了）。

如果说检索一张普通表就像查新华字典，检索分区表就像查辞海盒了。具体而言呢，又有这么几种方式：

***1).** 你知道你查的内容只会出现在某些分册里面，于是你很快的从辞海盒子里面取出你要的那个册子，不加思索的翻到索引页，根据索引页的指示，你飞快的翻到你的目标页面。取一本本册子呢就叫**partition range [inlist] iterator**，找索引页当然就是**index range scan**。如果你不找索引页，准备翻完整本书的找，那就是**full table scan** 了。如果你只找一本册子的，那**partition range iterator** 也就不必了。*

***2).** 你不知道你要查的内容在那本册子里？ 那你只好辛苦一点，翻阅所有册子了。这时，你做的动作就叫**partition range all.** 而对于每本册子言，也许你会找索引页（**index scan**），也许你想翻遍全册（**full table scan**）。*

3).** 也许你发现一册册的打开索引页找内容太繁重了，你突然想起来对你的辞海做个整改。于是你把每册的索引页全都拆了下来，专门装订成一册。每次你想利用索引页找东西时，你就打开这个索引册。从索引册，你就可以找到你要内容在哪一册哪个地方。这就是**global index scan.** 相对于**1,2**，就叫**local index scan.

4).** 你有儿子吗？有一天，你想培训儿子的能力，于是你就找来你儿子给你翻册子，找资料。可是你儿子非得和老子一起找才肯帮你。于是你们父子俩就开始一起检索起辞海来，你查某些册子，他查另一些册子。这就叫**partition scan.

NESTED LOOP(嵌套连接)

从执行计划的角度上看，表与表连接方法共有三种，嵌套循环是其中的一种，是执行计划中看到的最常见的一种连接。在嵌套循环中，内表被外表驱动，外表返回的每一行都要在内表中检索找到与它匹配的行。

嵌套循环在小表驱动大表，并且返回结果小的情况下是最快的一种连接方式。对于嵌套循环来说，整个查询返回的结果集不能太大（大于 1 万不适合），要把返回子集较小表的作为外表（CBO 默认外表是驱动表），而且在内表的连接字段上一定要有索引。

可以用 ORDERED 提示来改变 CBO 默认的驱动表，使用 USE_NL(table_name1 table_name2)强制 CBO 执行嵌套循环连接。

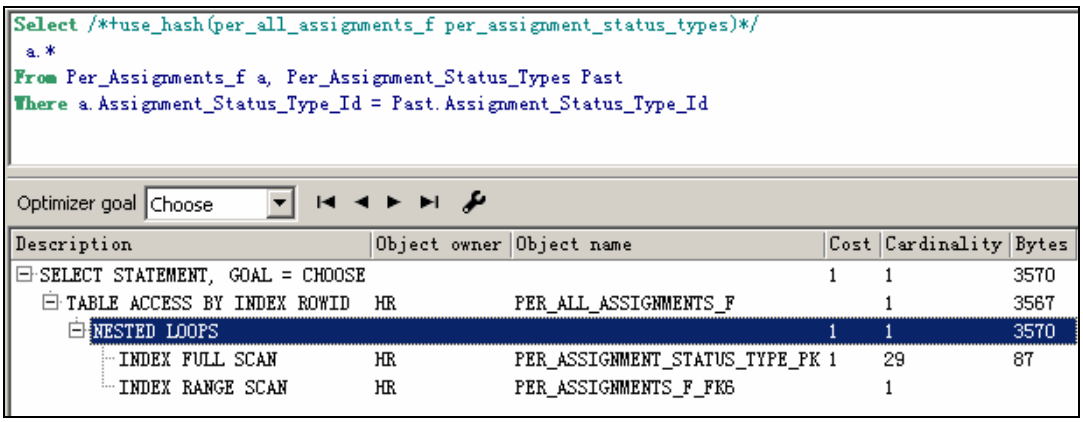
HASH JOIN(散列连接)

从执行计划的角度上看，表与表连接方法共有三种，散列连接是其中的一种。散列连接，又称哈希连接，是 CBO 做大数据集连接时的常用方式，优化器使用两个表中较小的表（或数据源）利用连接键在内存中建立散列表，然后扫描较大的表并探测散列表，找出与散列表匹配的行。

这种方式适用于较小的表完全可以放于内存中的情况，这样总成本就是访问两个表的成本之和。但是在表很大的情况下并不能完全放入内存，这时优化器会将它分割成若干不同的分区，不能放入内存的部分就把该分区写入磁盘的临时段，此时要有较大的临时段从而尽量提高 I/O 的性能。

可以用 USE_HASH(table_name1 table_name2)提示来强制使用散列连接。如果使用散列连接 HASH_AREA_SIZE 初始化参数必须足够的大，如果是 9i，Oracle 建议使用 SQL 工作区自动管理，设置 WORKAREA_SIZE_POLICY 为 AUTO，然后调整 PGA_AGGREGATE_TARGET 即可。

备注：在 Sql 语句中使用 Hint 时，如果对表使用了别名，则 Hint 中也要使用别名才能起作用。否则，即使使用 USE_HASH(table_name1 table_name2)，在执行计划中看，还是没有强制使用 Hash join ， 如下图：

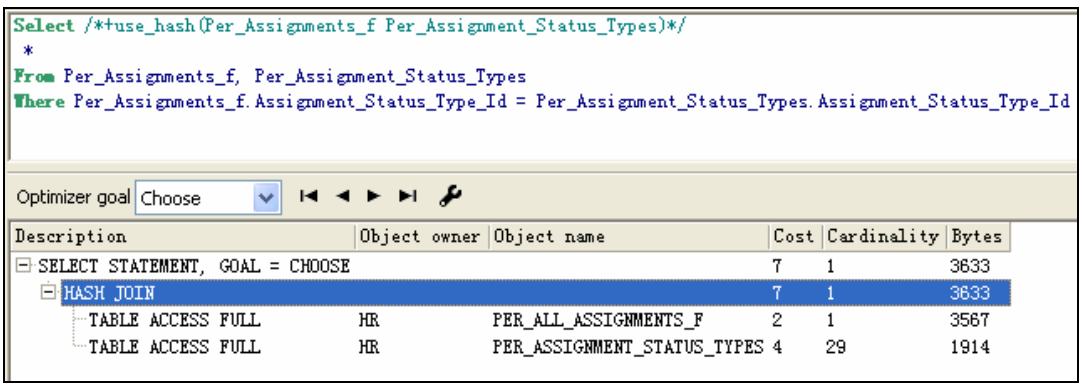


```
Select /*+use_hash(per_all_assignments_f per_assignment_status_types)*/
 a.*
From Per_Assignments_f a, Per_Assignment_Status_Types Past
Where a.Assignment_Status_Type_Id = Past.Assignment_Status_Type_Id
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			1	1	3570
TABLE ACCESS BY INDEX ROWID	HR	PER_ALL_ASSIGNMENTS_F		1	3567
NESTED LOOPS			1	1	3570
INDEX FULL SCAN	HR	PER_ASSIGNMENT_STATUS_TYPE_PK	1	29	87
INDEX RANGE SCAN	HR	PER_ASSIGNMENTS_F_FK6		1	

但是，如果把 Hint 中的表名换成别名就可以了。如下图：



```
Select /*+use_hash(Per_Assignments_f Per_Assignment_Status_Types)*/
 *
From Per_Assignments_f, Per_Assignment_Status_Types
Where Per_Assignments_f.Assignment_Status_Type_Id = Per_Assignment_Status_Types.Assignment_Status_Type_Id
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			7	1	3633
HASH JOIN			7	1	3633
TABLE ACCESS FULL	HR	PER_ALL_ASSIGNMENTS_F	2	1	3567
TABLE ACCESS FULL	HR	PER_ASSIGNMENT_STATUS_TYPES	4	29	1914

```
Select /*+use_hash(a past)*/
 a.*
From Per_Assignments_f a, Per_Assignment_Status_Types Past
Where a.Assignment_Status_Type_Id = Past.Assignment_Status_Type_Id
```

*小知识：哈希表**(Hash Table)***

哈希表又称散列表，其定义是这样的：根据设定的哈希函数 ***H(key)****和所选中的处理冲突的方法，将一组关键字映象到一个有限的、地址连续的地址集**(**区间**)**上，并以关键字在地址集中的“象”作为相应记录在表中的存储位置，这种表被称为哈希表。*

定义听起来比较难以理解，简单地说，哈希表是基于哈希函数建立的一种查找表。这种表最大的特点是表中的每条记录的关键字字段与这条记录在表中的位置有某种确定的函数关系。因此根据关键字查找记录不需要对整个表作查找扫描。而是可以由函数直接计算出这条记录的位置。所以查询速度是极快的。

例如下面这张表就是以 ***Stu_num*** *为关键字，以* ***f(Stu_num)=To_Number(Substr(Stu_num，4))*** *为哈希函数 的哈希表* ***(Student)*** *：*

Stu_num	Name	Age	Sex
STU1	张三	21	男
STU2	李四	22	男
STU3	王五	23	男
STU4	王六	24	男
STU5	阿扁	25	男
STU6	商云方	26	男

很显然，如果要执行如下 ***Sql*** *语句：* ***select * from student where Stu_num =′STU5′，*** *则不需要要进行检索，因为根据函数* ***f(′STU5′) =5*** *就直接得到这条记录的位置了。*

MERGE JOIN & SORT JOIN (排序合并)

从执行计划的角度上看，表与表连接方法共有三种，排序合并是其中的一种。一般我们称排序合并为 SORT MERGE，在执行计划中表现为两个表分别作 Sort Join 然后再在一起做个 Merge Jion。

sort merge join 的操作通常分三步：对连接的每个表做 table access full；对 table access full 的结果进行排序；进行 merge join 对排序结果进行合并。sort merge join 性能开销几乎都在前两步。一般是在没有索引的情况下，9i 开始已经很少出现了，因为其排序成本高，大多为 hash join 替代了。

通常情况下 hash join 的效果都比 sort merge join 要好，然而如果行源已经被排过序，在执行 sort merge join 时不需要再排序了，这时 sort merge join 的性能会优于 hash join。

在全表扫描比索引范围扫描再通过 rowid 进行表访问更可取的情况下，sort merge join 会比 nested loops 性能更佳。

```
select  /*+USE_MERGE(ias i)*/
        ias.item_type,    ias.item_key,i.user_key
from    wf_item_activity_statuses ias,  wf_items i
where   ias.activity_status     = 'ERROR'
and     i.item_type             = ias.item_type
and     i.item_key              = ias.item_key
order by ias.begin_date, ias.execution_time
```

Optimizer goal: Choose

Description	Object owner	Object name	Cost	Cardinality	Bytes
SELECT STATEMENT, GOAL = CHOOSE			11	1	406
SORT ORDER BY			11	1	406
MERGE JOIN			7	1	406
TABLE ACCESS BY INDEX ROWID	APPLSYS	WF_ITEMS	1	1	250
INDEX FULL SCAN	APPLSYS	WF_ITEMS_PK		1	
SORT JOIN			6	1	156
TABLE ACCESS BY INDEX ROWID	APPLSYS	WF_ITEM_ACTIVITY_STATUSES	1	1	156
INDEX RANGE SCAN	APPLSYS	WF_ITEM_ACTIVITY_STATUSES_N		1	

```
select  /*+USE_MERGE(ias i)*/
        ias.item_type,    ias.item_key,i.user_key
from    wf_item_activity_statuses ias,  wf_items i
where   ias.activity_status     = 'ERROR'
and     i.item_type             = ias.item_type
and     i.item_key              = ias.item_key
order by ias.begin_date, ias.execution_time
```

小知识：归并排序

归并排序是一种排序方法，把一组需要排序元素分成两组，先分别排序，然后再归并。这种排序方法又称为二路归并排序。比如，考虑 8 个元素，值分别为[1 0，4，6，3，8，2，5，7]。分成两组分别排序，则[1 0 , 4 , 6 , 3]和[8 , 2 , 5 , 7]将被分别独立地排序。结果分别为[3 , 4 , 6 , 1 0]和[2 , 5 , 7 , 8]。从两个序列的头部开始归并这两个已排序的序列。元素 2 比 3 更小，被移到结果序列；3 与 5 进行比较，3 被移入结果序列；4 与 5 比较，4 被放入结果序列；5 和 6 比较。。。依次类推。当这两个排好序的序列被归并后，即可得所需要的排序序列。[2,3,4,5,6,7,8,10].

归并排序为什么能成为一种执行计划中的两张表的连接方法呢？因为两张表通过某个字段连接。那么只有这两张表先分别按照连接字段排序，然后再按照连接字段归并，并把两个表连接字段交集以外的记录去除。即可生成两张表连接的结果了。

小知识：三种连接方式的对比

Note:207522.1

***Merge Jion** 是一种组操作，在所有行被处理完之前，它不会返回任何行记录给下一个操作。**Nested Loop** 和 **Hash Join** 是行操作，因此会很快将一批记录返回给下一操作。*

第二章： Sql Trace

本章讲述如何进行 Sql trace。

SQL Trace 基础知识

Oracle 的 SQL trace 工具 在数据库端 收集正在执行的 SQL 的性能状态数据并记录到一个跟踪文件(.trc 文件)中. 这个跟踪文件提供了许多有用的信息,例如解析次数.执行次数,CPU 使用时间等.这些数据将可以用来优化你的系统.

跟踪文件根据 Sql 执行的先后循序记录每条 Sql 的性能状态数据。其扩展名是.Trc。

.trc 文件为每条 Sql 语句产生如下统计信息：

1） Parse(解析)、Excute(执行)、Fetch(提取) 的次数

应该说 Parse(解析)、Excute(执行)、Fetch(提取) 是 Oracle 要执行一条 Sql 语句要可能要进行的三种动作。下面对这三种动作做具体说明：

Parse(**解析**) ：就是把 Sql 语句转化成一个执行计划的过程。当然这里面还包括一系列检查工作：比如当前用户是否有权访问这些表，这些表及表中的列是否存在等。
一条 Sql 语句提交给 Oracle 后，如果原来曾经解析过，解析过的结果会存放在缓存区。但是缓存区的数据不会永久保留，因为缓存区是要被循环使用的，所以一旦缓存中的解析结果丢失，那么再次提交这条语句的时候，Oracle 就又要解析一次了。所以 Parse 次数根据具体的运行环境可能是 0，1 或其他。

Excute(**执行**)：就是 Oracle 对 Sql 语句的实际的执行。对于 UPDATE、INSERT、DELETE 语句来说，就是更改数据。而对于 SELECT 语句来说就是把被选中的行标识出来。一条 Sql 语句在执行过程中有可能进行一次或多次 Excute 动作。每次 Excute 动作也会影响一条或多条记录。

Fetch(**提取**)：就是提取一个查询返回的行。只有 SELECT 语句会执行 Fetch 动作。一条 Sql 语句在执行过程中有可能进行一次或多次 Fetch 动作。每次 Fetch 动作也会提取一条或多条记录。

2） 每种动作耗费的 Cpu 时间和总时间(包括 Cpu 时间和 IO 时间的总时间)

3） 物理读取数量和逻辑读取数量
物理读取数量指从磁盘上的数据文件中读取的数据块数量，数据块的大小是由数据库的配置文件 InitProd.ora 中的参数 db_block_size 决定的，一般情况下，Oracle EBS 要求大小为 8K。逻辑读取数量指从内存中读取的缓冲数量。
这个缓冲大小是多大？

4） Misses in library cache 这是什么意思？

SQL trace 的作用

一般来说，做 Sql Trace 的作用有两个：
1） 做 Sql 语句的执行性能分析
2） 对于一些无法查看源代码的程序，做 sql Trace 可以看看这些程序究竟执行了哪些 sql. 这一点经常在查问题时用得着。

SQL trace 的方式

根据 Sql Trace 的结果的粗细程度不同，可以将 Trace 分为常规 Trace 和 其他详细 Trace．你可以根据不同的需要，选择不同的 Trace 方式。

不同的 Trace 方式说明如下：

Trace 方式	说明
Regular Trace	常规的 trace． 这种 Trace 的结果中，你看到的 Sql 语句中的所有变量都被“Bind Variable”所取代。 并且”Bind Variable” 的具体值也不会显示出来。所以适合于性能分析。 可能并不适合查找问题。
Trace with Binds	带变量的 Trace. 这种 trace 的结果中，会包含”Bind Variable”的值. 这对于分析问题很有用 。
Trace with Waits	
Trace with Binds and Waits	

*小知识：什么是“**Bind Variables**”*

Note: 296559.1

*一个“**Bind Variable**” 就是 **Trace** 文件中一个可替代的值。为了提高性能,**Oracle** 会重用一些 **Sql**．这些 **Sql** 中会为变量使用一些可替代值。 在运行此 **Sql** 前，**oracle** 会把*“Bind Variable” 替换成一个实际的值。

*比如下图是一个 **Trace** 文件的片段，其中有一段的 **Sql**：“**SELECT P.SPID FROM V$SESSION S, V$PROCESS P WHERE S.AUDSID = :B1 AND S.PADDR=P.ADDR**”，这里 **B1** 就是一个“可替代值” 它是用来代表变量 **p_session_id** 的。 在运行 **Sql** 前。 **B1** 会被替代为真正的值。在这个例子中，我们说 **B1** 是一个 **Bind Variable.***

*采用“ **Trace with Binds**”的方式进行 **Trace**，可以让 **Trace** 的结果文件中包含 **Bind Variable** 的值。包含 **Bind Variable** 值的 **Trace** 文件和常规的 **Trace** 文件相比，对每条 **Sql** 的 **Trace** 内容要多一段**[BINDS]**，如下图中的兰色字体部分。*

*在这一段的最后有 value=147698 ,这里的 **147698** 就是用来替换 **B1** 的具体值*

```
=====================
PARSING IN CURSOR #40 len=83 dep=2 uid=44 oct=3 lid=44 tim=15938550871
hv=3934942953 ad='6bc8c478'
SELECT P.SPID FROM V$SESSION S, V$PROCESS P WHERE S.AUDSID = :B1
AND S.PADDR=P.ADDR
END OF STMT
PARSE #40:c=0,e=93,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=15938550864
BINDS #40:
 bind 0: dty=2 mxl=22(21) mal=00 scl=00 pre=00 oacflg=13 oacfl2=1 size=24
offset=0
  bfp=0c03eb48 bln=22 avl=04 flg=05
  value=147698
EXEC #40:c=0,e=6889,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=15938566272
FETCH #40:c=0,e=733,p=0,cr=0,cu=0,mis=0,r=1,dep=2,og=4,tim=15938568970
FETCH #40:c=0,e=206,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=15938571082
```

Tkprof 工具

.trc 文件一般比较长，也不易阅读。经过 Tkprof 整理后就比较容易阅读了。（一般对 Trace 文件进行性能分析时，都使用 Tkprof 整理一下。但是对于排查问题来说，就不能用 Tkprof 整理了，因为整理后，Sql 执行的顺序就乱了，而且有些信息，比如“Bind Variable”的值 也丢失了）

Tkprof 命令是 Oracle 客户端的一个可执行文件，比如 在 11.5.10 安装完后，在 prodora\8.0.6\BIN 目录下 有个 TKPROF80.EXE .

TKPROF 的命令行格式是：

Tkprof tricefile outputfile [Sys=] [Explain=] [Sort=] [Table=] [Print=] [Insert=]

说明：

Tkprof ：即命令行的命令，有的版本是 tkprof80 .

Tricefile : 即.trc 文件的文件名。

Outputfile : 即 tkprof 的输出文件的文件名。

打中括号的六个选项都是可选的。如果需要可以加这些参数，这些参数的意义如下：

[Sys=] ：Tkprof 默认会列出所有的 sql 语句，包括 Sys 用户执行的语句。如果添加参数 Sys=no 则 Tkprof 就不会列出 Sys 用户执行的语句。

[Explain=] : Tkprof 默认不会列出每条 Sql 语句的执行计划，如果添加参数 [Explain=user_name/password] 则 Tkporf 就会列出每条 sql 语句的执行计划。这里 User_name/password 是你运行程序时连接数据库用的用户名和密码。比如 oracle EBS 中我们一般用 Apps/Apps’s password .

[Sort] : 如果指定此选项，则可以让 Tkprof 的结果按照一定的要求进行排序。排序的选项有很多，比如[Sort=prscnt] 表示需要按照 sql 解析的次数排序。[Sort=Exerow] 表示需要按照 Sql 执行过程中 被处理的行数来排序。。。 (更多的选项及意义可以在命令方式下运行 Tkprof 看其帮助)

[Table=] [Print=] [Insert=] 这三个选项不常用

小知识： ***Recursive Calls***

Recursive Calls : ***Recursive*** *英文的字面意思是递归。但是* ***Oracle*** *文档提及的* ***Recursive calls*** *的意思并非递归调用。可以理解为衍生的系统调用。*

有时候，为了执行用户发出的 ***Sql*** *语句，系统不得不首先调用其他语句来做一些事情。比如有时需要往某张表里插入一条记录，但是这张表所在的表空间已经满了，则系统不得不调用一条表空间扩展语句来动态的扩展这个表空间。然后在执行插入记录的操作。这种情况下，对表空间扩展语句的调用就是所谓的* ***Recursive Calls*** *了。一些常见的* ***Recursive Calls*** *包括数据字典高速缓存太小而不得不从磁盘读取。或者一些表、索引、回滚段以及临时段设置不合理而导致频繁的系统调用。*

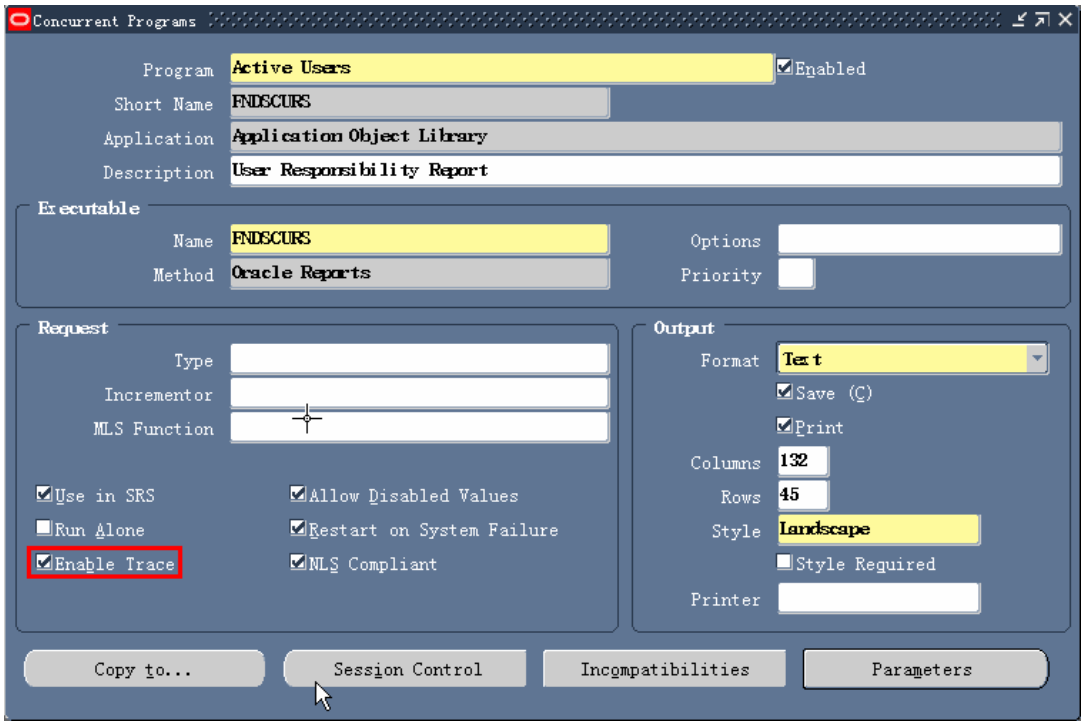
并发程序 Trace

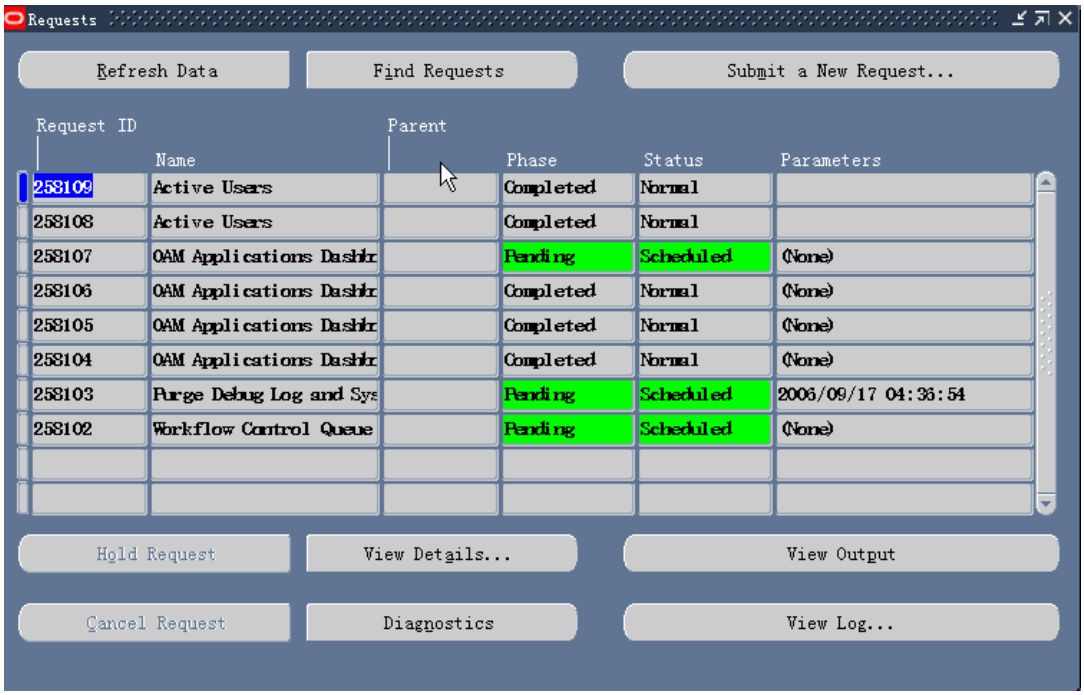
并发程序的 Trace ,需要在并发程序定义界面选中”Trace” 选项 ， 然后运行此并发程序， 系统会在数据库服务器的 Udump 目录下生成这次并发程序运行的 Trace 文件 。 举例如下：

常规 Trace

所谓常规 Trace 是相对于 “Trace With Binds” 而言的。常规 Trace 的结果文件中，不包含”Bind Variable”的值。

比如我们要对报表”Active User” 进行 Trace ,则 首先我们在并发程序定义的地方把 Enable Trace 选中，然后运行此报表。



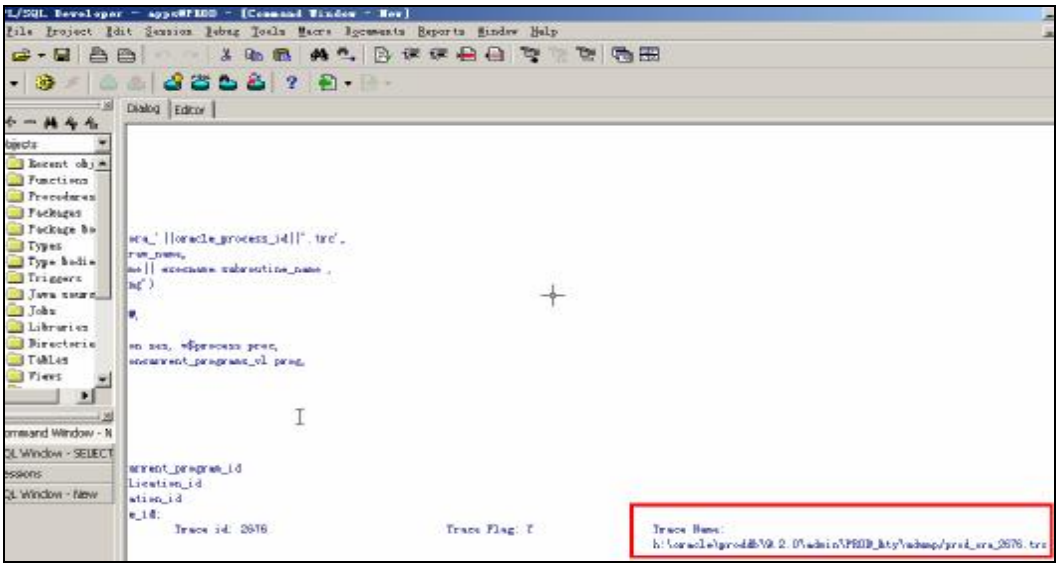


记住此次运行的并发程序的 Request id ,这里是 258109

在 Sql plus 或者 plsql developer 的 Command Window 中运行如下脚本(并发请求 Trace 文件名生成脚本)获得这次并发请求的 Trace 文件名：

```
prompt
accept request prompt 'Please enter the concurrent request id for the
appropriate concurrent program:'
prompt

column traceid format a8
column tracename format a80
column user_concurrent_program_name format a40
column execname format a15
column enable_trace format a12
set lines 80
set pages 22
set head off

SELECT 'Request id: '||request_id ,
'Trace id: '||oracle_Process_id,
'Trace Flag: '||req.enable_trace,
'Trace Name:
'||dest.value||'/'||lower(dbnm.value)||'_ora_'||oracle_process_id||'.trc',
'Prog. Name: '||prog.user_concurrent_program_name,
'File Name: '||execname.execution_file_name|| execname.subroutine_name ,
'Status : '||decode(phase_code,'R','Running')
||'-'||decode(status_code,'R','Normal'),
'SID Serial: '||ses.sid||','|| ses.serial#,
'Module : '||ses.module
from fnd_concurrent_requests req, v$session ses, v$process proc,
v$parameter dest, v$parameter dbnm, fnd_concurrent_programs_vl prog,
fnd_executables execname
where req.request_id = &request
and req.oracle_process_id=proc.spid(+)
and proc.addr = ses.paddr(+)
and dest.name='user_dump_dest'
and dbnm.name='db_name'
and req.concurrent_program_id = prog.concurrent_program_id
and req.program_application_id = prog.application_id
and prog.application_id = execname.application_id
and prog.executable_id=execname.executable_id;
```

运行后可得到包含全路径的 Trace 文件名：
h:\oracle\proddb\9.2.0\admin\PROD_hty\udump/prod_ora_2676.trc 如下图：

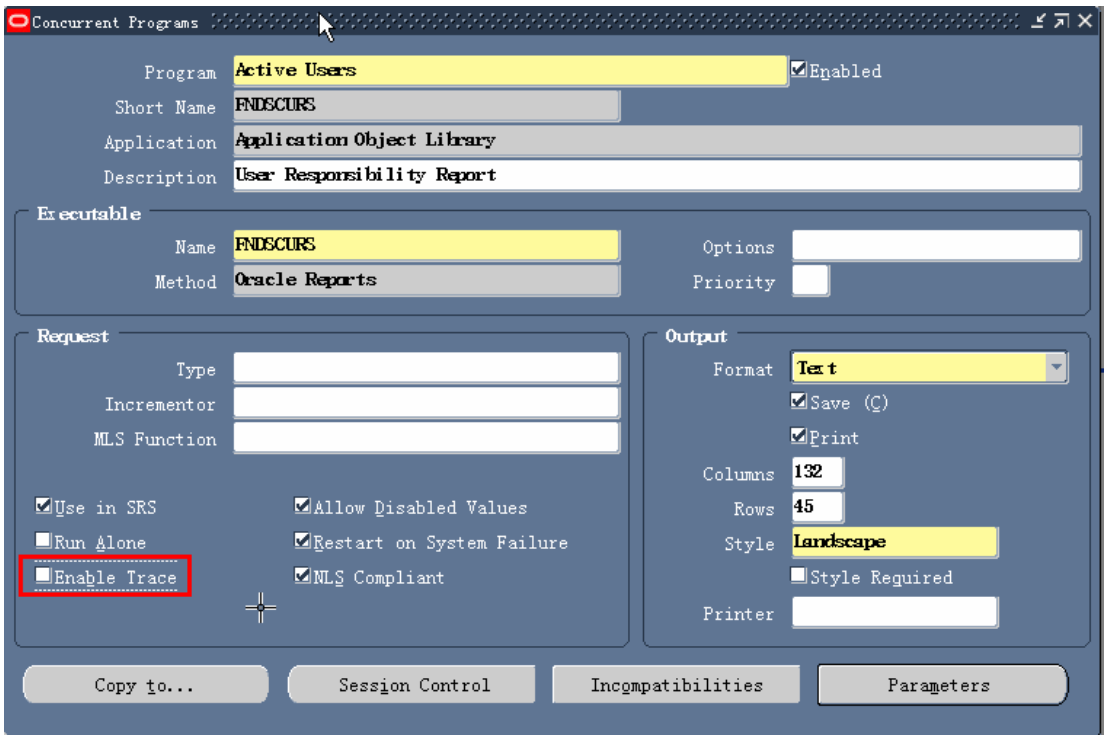


用户 Session 级的 Trace With Binds （使用 “设置系统配置文件选项”的方法）

对并发程序采用”Trace With Binds”的方式进行 Trace .需要多几个步骤，下面举例说明：

还是上文的例子，我们要对 Active Users 这个并发程序，以”Trace With Binds”的方式进行 Trace。

1、在并发程序定义的地方，确保 “Enable Trace” 没有打勾。



2、在运行此并发程序的时候不要启用 "Help -> Diagnostics -> Trace" 中的当前 Session 的 Trace 功能。

3、设置系统配置文件选项： "Initialization SQL Statement - Custom" 的 USER 级的值为：

```
begin fnd_ctl.fnd_sess_ctl('','TRUE','TRUE','LOG','ALTER SESSION SET EVENTS="10046 TRACE
NAME CONTEXT FOREVER, LEVEL 4" TRACEFILE_IDENTIFIER="<tar/bug#>"'); end;
```

注意：a）这句话需要确保是单行格式。

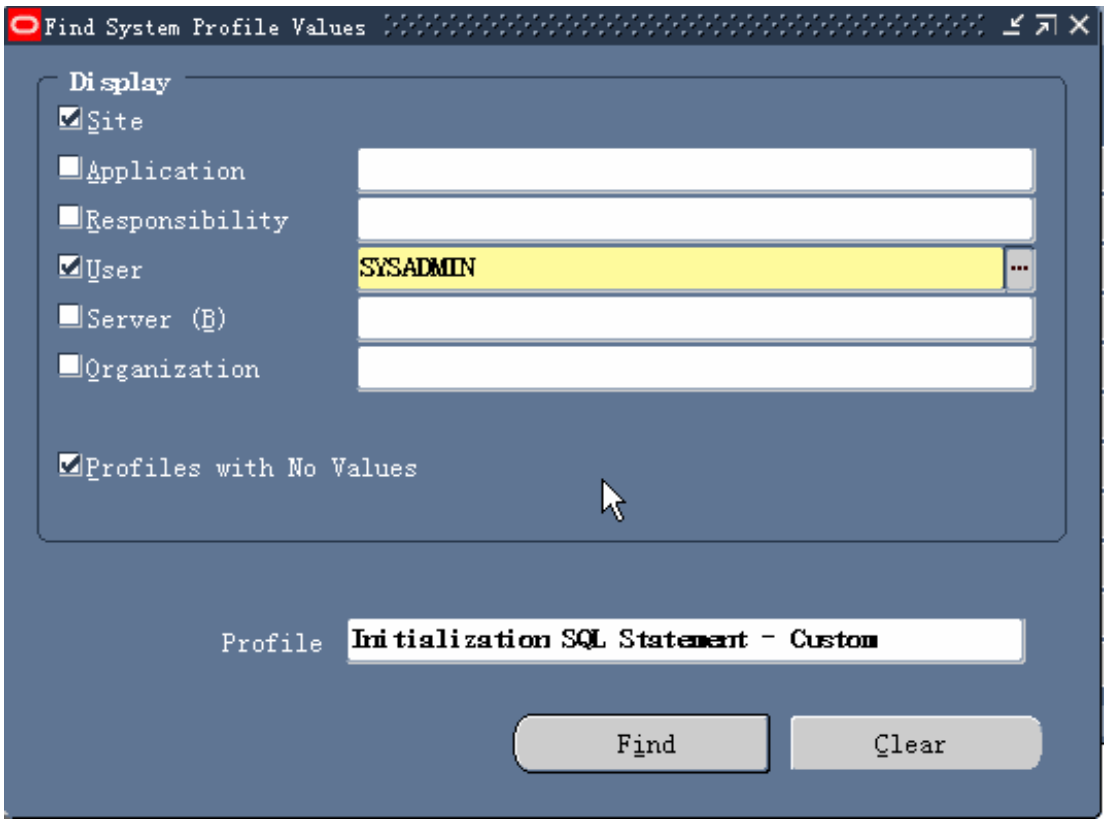
b) LEVEL 4 可被改为 LEVEL 8 或者 LEVEL 12,其意义分别如下：

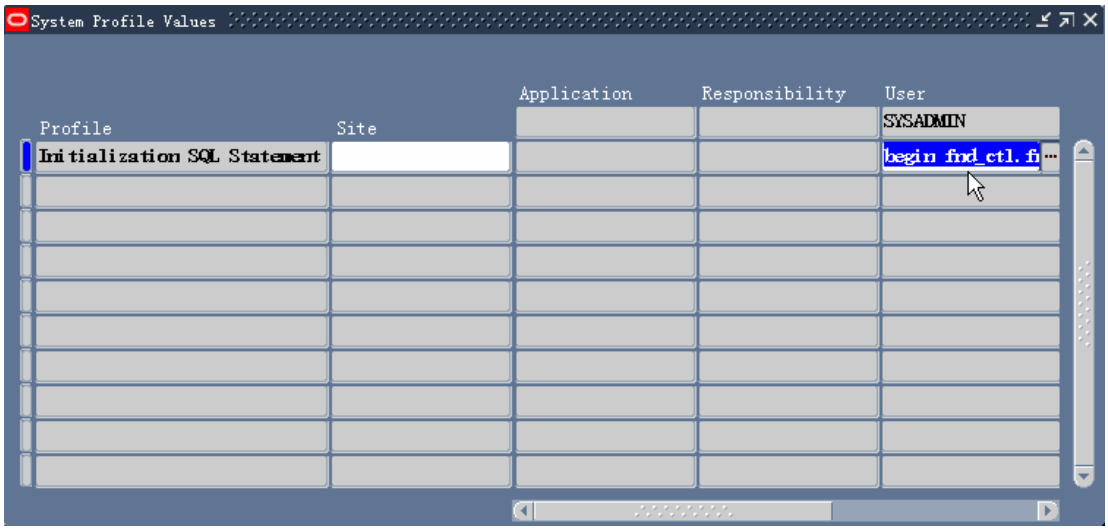
Use LEVEL 4 ： 以”Trace with Binds” 的方式进行 Trace

Use LEVEL 8 ： 以”Trace with Waits” 的方式进行 Trace

Use LEVEL 12 ： 以”Trace with Binds and Waits” 的方式进行 Trace

c) <tar/bug#> 需要被替换成一个数值，这个数值应该代表你指定的 Tar 或者 Bug 号，比如 123432.





4、退出 EBS ,然后重新登陆，以刚才设置预置文件时指定的用户登陆。

5、登陆后，不要做其他操作，直接去提交要 Trace 的并发程序。并发程序完成后把该用户的预置文件 "Initialization SQL Statement - Custom" 的值再恢复到设置前。然后退出 EBS.

6、生成的 Trace 文件在 User_dump_dest 目录下。User_dump_dest 的具体路径可以通过如下 Sql 得到：

```
select value from v$parameter where name = 'user_dump_dest';
```

7、生成的 Trace 文件名中会包含你指定的<tar/bug#>字符串。在 User_dump_dest 目录下可以使用如下命令来查找：

```
ls *<tar/bug#>*
```

如果是 Windows 平台，可以使用如下命令来查找：

```
Dir  *<tar/bug#>*.*
```

不过我们发现，其实在 User_dump_dest 目录下，我们还是发现例如若干个包含<tar/bug#> 的文件(本例中，<tar/bug#> =123432) 。

名称	大小	类型	修改日期
prod_ora_1532_123432.trc	968 KB	TRC File	2006-9-24 16:39
prod_ora_2732_123432.trc	39 KB	TRC File	2006-9-24 16:37
prod_ora_2732.trc	2 KB	TRC File	2006-9-24 16:37
prod_ora_1600_123432.trc	299 KB	TRC File	2006-9-24 16:37
prod_ora_1600.trc	2 KB	TRC File	2006-9-24 16:37
prod_ora_1532.trc	2 KB	TRC File	2006-9-24 16:36
prod_ora_3268_123432.trc	175 KB	TRC File	2006-9-24 16:36
prod_ora_3432_123432.trc	45 KB	TRC File	2006-9-24 16:36
prod_ora_3432.trc	2 KB	TRC File	2006-9-24 16:36
prod_ora_3268.trc	2 KB	TRC File	2006-9-24 16:36

那么为什么会有多个包含 123432 的文件呢？ 估计是这个预置文件被设置后，这个用户(本例中是 Sysadmin) 的每个 Session 都生成了一个 Trace 文件。这些 Trace 文件的命名规则是 <sid>_ORA_<process_id>_<tar/bug#>.trc .

如何找到我们要的那个并发程序的 Trace 文件呢？这可以使用[并发请求 Trace 文件名生成脚本] 根据 reques_id 获得 一个文件名，这个文件名中包含这个并发请求的 process_id . 根据 Process_id ,我们可以得出 prod_ora_2732_123432.trc 就是这个并发程序的 Trace 文件。

数据库级的 Trace With Binds （使用 bde_session_event_10046.sql 的方法）

上述更改系统配置文件选项的方法，可以让某个特定用户在运行并发程序后产生 Trace With Binds 的 Trace 文件。 如果要让所有用户运行此并发程序都有 Trace With Binds 信息，可以使用另一种方法：直接设置数据库系统级的 event_10046。方法步骤如下：

1、 以 Apps 登陆 Sqlplus

2、 运行 bde_session_event_10046.sql

3、 看到提示“Click ENTER to TURN ON EVENT 10046 for ALL new sessions”后按回车，启动系统级的 10046 事件。

4、 回到 ERP 操作，运行并发程序

5、 运行完成并发程序后。再回到 Sql Plus , 再次按回车　停止 10046 事件。

6、 查看产生的 Trace 文件。可以看到文件中包含　Binds 信息。

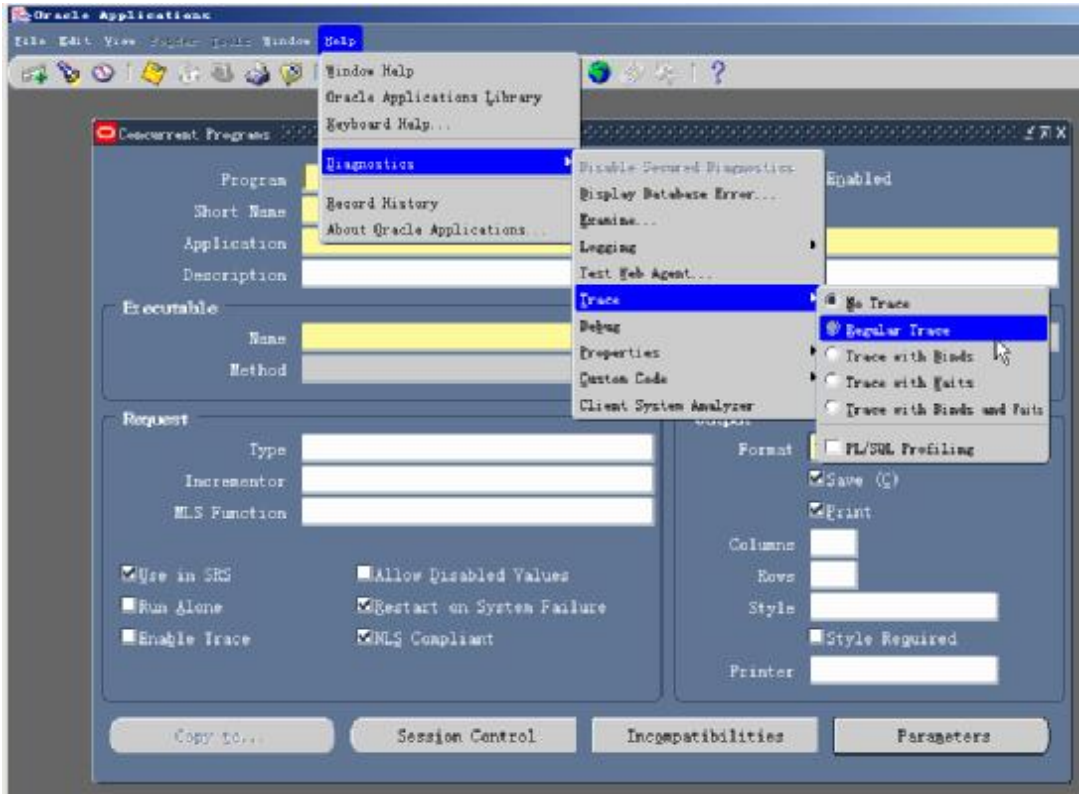
bde_session_event_10046.sql

```
/*************------- Instructions ------------
 1. Copy this whole Note into a text file as bde_session_event_10046.sql
 2. Read instructions embedded in PROMPT commands below.
 3. Execute from SQL*Plus using APPS account:
    # sqlplus apps/apps@vis11i
    SQL> START bde_session_event_10046.sql;    **************/
var v_mdfs varchar2(30);
var v_ts   varchar2(30);
BEGIN
   SELECT Value  INTO :v_mdfs  FROM v$parameter WHERE name = 'max_dump_file_size';
   SELECT Value  INTO :v_ts    FROM v$parameter WHERE name = 'timed_statistics';
END;
/
SET term on;
PAUSE  Click ENTER to TURN ON EVENT 10046 for ALL new sessions

SELECT TO_CHAR(sysdate,'DD-MON-YY HH24:MI:SS') "Start" FROM dual;
ALTER SYSTEM SET max_dump_file_size = unlimited;
ALTER SYSTEM SET timed_statistics = true;
ALTER SYSTEM SET EVENTS '10046 trace name context forever, level 12';

PAUSE  Click ENTER to TURN OFF EVENT 10046 for ALL new sessions

ALTER SYSTEM SET EVENTS '10046 trace name context off';
DECLARE
   v_sql        VARCHAR2(1000);
BEGIN
   v_sql:='ALTER SYSTEM SET timed_statistics = '||:v_ts;
   EXECUTE IMMEDIATE v_sql;
   v_sql:='ALTER SYSTEM SET max_dump_file_size = '||:v_mdfs;
   EXECUTE IMMEDIATE v_sql;
END;
/
PROMPT
PROMPT Find your raw SQL Trace with WAITS and BIND VARIABLES under the
PROMPT UDUMP directory.  There may be several other traces as well.
PROMPT
SELECT TO_CHAR(sysdate,'DD-MON-YY HH24:MI:SS') "End" FROM dual;
```

Form Trace

对 Form 进行 Trace 比较简单。只需要打开这个 Form,在执行任何动作前，从 help 菜单中启用 trace 即可。

Nav：Help > Diagnostics > Trace > Trace with binds

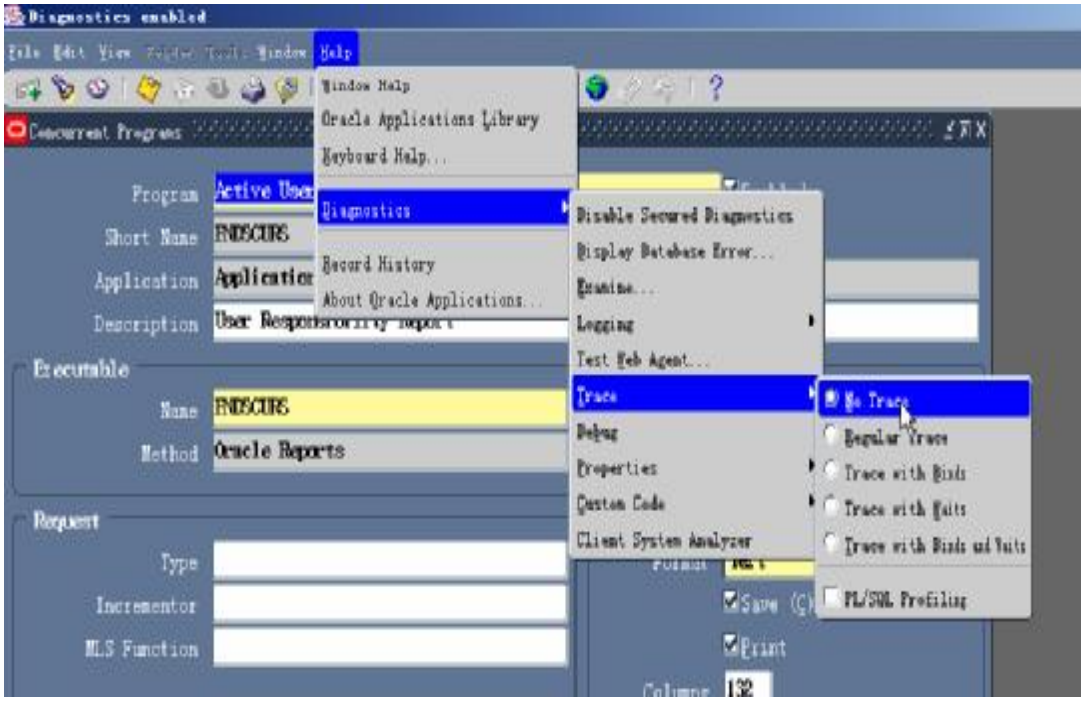


这里的 Trace 有若干种选项,各种选项的意义，我们在前面 Sql Trace 的基础知识中已经讲过了。这里不再赘述。

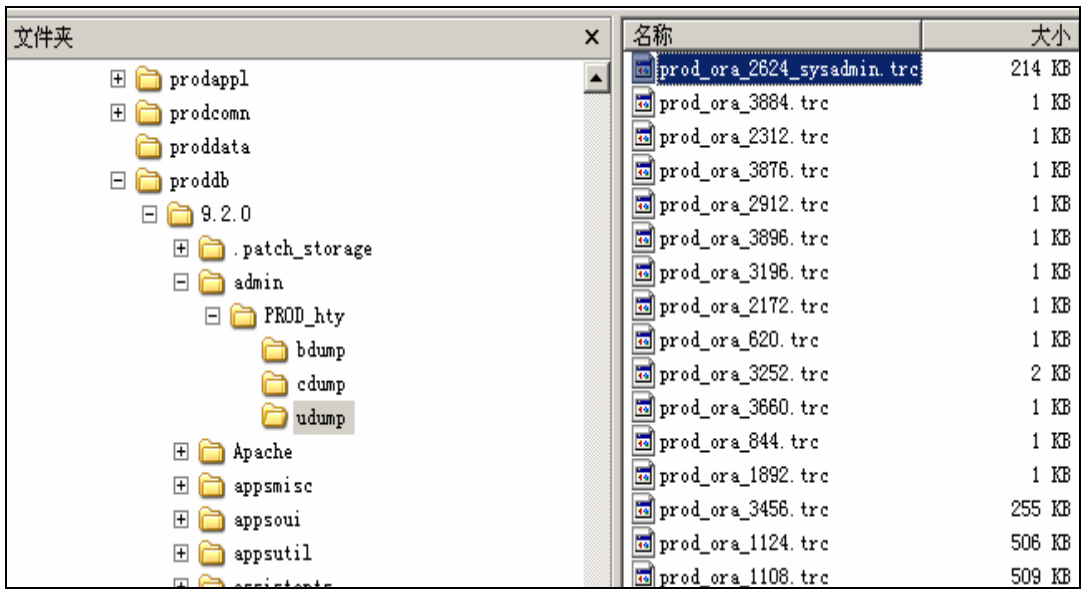
Form 的 Trace 比较简单，选择一种 Trace 方式以后,系统会提示你 Trace 文件的目录和文件名。如下图。

当执行了需要跟踪的操作后，需要停止 Trace ，只需要选择：

Nav：Help > Diagnostics > Trace > No Trace 即可



然后可以按照指定的路径找到这个 Trace 文件，如下图：



Web 应用的 Trace

使用上面提到的《用户 Session 级的 Trace With Binds》和 《数据库级的 Trace With Binds》 都可以用于 Web 应用的 Trace .

参考文献

Metlink Note 41634.1,

已解决问题和未解决问题

未解决问题

1、InitProd.ora 中有个参数：
timed_statistics = true 指让数据库支持时间统计信息，并且说：″This information is used by many options, including SQL_TRACE, Oracle Trace,statspack and Oracle Enterprise Manager.″
这里 SQL_TRACE 我们已经知道了。那么 Oracle Trace 指的是什么？

已解决问题