SQL 执行计划与性能调优

SQL语句处理的基本过程

- 查询语句处理(select)
- DML语句处理(insert, update, delete)
- DDL 语句处理(create .. , drop .. , alter .. ,)
- 事务控制(commit, rollback)

DML语句的处理

- 举例

```
EXEC SQL UPDATE employees
                    SET salary = 1.10 * salary
WHERE department_id = :var_department_id;
```

- 每种类型的语句都需要如下阶段：
  - 第1步: Create a Cursor 创建游标
  - 第2步: Parse the Statement 分析语句
  - 第5步: Bind Any Variables 绑定变量
  - 第7步: Run the Statement 运行语句
  - 第9步: Close the Cursor 关闭游标

如果使用了并行功能，还会包含下面这个阶段：

- 第6步: Parallelize the Statement 并行执行语句

如果是查询语句，则需要以下几个额外的步骤，

- 第3步: Describe Results of a Query 描述查询的结果集
- 第4步: Define Output of a Query 定义查询的输出数据
- 第8步: Fetch Rows of a Query 取查询出来的行

- **第1步: 创建游标(Create a Cursor)**

  由程序接口调用创建一个游标(cursor)。任何SQL语句都会创建它，特别在运行DML语句时，都是自动创建游标的，不需要开发人员干预。多数应用中，游标的创建是自动的。

  说明：在预编译程序(pro*c)中游标的创建，可能是隐含的，也可能显式的创建。

- **第2步:分析语句(Parse the Statement)**

  在语法分析期间，SQL语句从用户进程传送到Oracle，SQL语句经语法分析后，SQL语句本身与分析的信息都被装入到共享SQL区。

语法分析执行下列操作

- 翻译SQL语句，验证它是合法的语句，即书写正确
- 实现数据字典的查找，以验证是否符合表和列的定义
- 在所要求的对象上获取语法分析锁，使得在语句的语法分析过程中不改变这些对象的定义
- 验证为存取所涉及的模式对象所需的权限是否满足
- 决定此语句最佳的执行计划
- 将它装入共享SQL区
- 对分布式查询语句，把语句的全部或部分路由到包含所涉及数据的远程节点

- 只有在共享池中不存在等价SQL语句的情况下，才对SQL语句作语法分析
- 虽然语法分析验证SQL语句的正确性，但语法分析只能识别在SQL语句执行之前所能发现的错误(如书写错误、权限不足等)。
- 有些错误通过语法分析是抓不到的。例如，在数据转换中的错误或在数据中的错(如企图在主键中插入重复的值)以及死锁等均是只有在语句执行阶段期间才能遇到和报告的错误或情况。

- **第3步: 描述查询结果(Describe Results of a Query)**

  描述阶段只有在查询结果的各个列是未知时才需要;

  例如当查询由用户交互地输入需要输出的列名。在这种情况要用描述阶段来决定查询结果的特征(数据类型，长度和名字)。

- **第4步: 定义查询的输出数据(Define Output of a Query)**

  在查询的定义阶段，指定与查询出的列值对应的接收变量的位置、大小和数据类型，这样我们通过接收变量就可以得到查询结果。如果必要的话，Oracle会自动实现数据类型的转换。这是将接收变量的类型与对应的列类型相比较决定的。

- **第5步: 绑定变量(Bind Any Variables)**

  此时，Oracle知道了SQL语句的意思，但仍没有足够的信息用于执行该语句。Oracle 需要得到在语句中列出的所有变量的值。

  在该例中，Oracle需要得到对department_id列进行限定的值。得到这个值的过程就叫绑定变量(binding variables)

- **第6步: 并行执行语句(Parallelize the Statement)**

ORACLE 可以在SELECTs, INSERTs, UPDATEs, MERGEs, DELETEs语句中执行相应并行查询操作，对于某些DDL操作，如创建索引、用子查询创建表、在分区表上的操作，也可以执行并行操作。

并行化可以导致多个服务器进程(oracle server processes)为同一个SQL语句工作，使该SQL语句可以快速完成，但是会耗费更多的资源，所以除非很有必要，否则不要使用并行查询。

Oracle优化器

- **基于规则的优化器 -- Rule Based (Heuristic) Optimization(简称RBO):**

  在ORACLE7之前，主要是使用基于规则的优化器。ORACLE在基于规则的优化器中采用启发式的方法(Heuristic Approach)或规则(Rules)来生成执行计划。

  如果一个查询的where条件(where clause)包含一个谓词(predicate，其实就是一个判断条件，如”=”, “>”, ”<”等)，而且该谓词上引用的列上有有效索引，那么优化器将使用索引访问这个表，而不考虑其它因素，如表中数据的多少、表中数据的易变性、索引的可选择性等。

RBO举例

select * from emp where deptno = 10;

如果是使用基于规则的优化器，而且deptno列上有有效的索引，则会通过deptno列上的索引来访问emp表。

1) emp表比较小，该表的数据只存放在几个数据块中。此时使用全表扫描比使用索引访问emp表反而要好。

因为表比较小，极有可能数据全在内存中，所以此时做全表扫描是最快的。而如果使用索引扫描，需要先从索引中找到符合条件记录的rowid，然后再一一根据这些rowid从emp中将数据取出来，在这种条件下，效率就会比全表扫描的效率要差一些

2) emp表比较大时，而且deptno = 10条件能查询出表中大部分的数据如(50%)。如该表共有4000万行数据，共放在有500000个数据块中，每个数据块为8k，则该表共有约4G，则这么多的数据不可能全放在内存中，绝大多数需要放在硬盘上。此时如果该查询通过索引查询，则是你梦魇的开始。

db_file_multiblock_read_count参数的值200。如果采用全表扫描，则需要500000/db_file_multiblock_read_count=500000/200=2500次I/O。

但是如果采用索引扫描，假设deptno列上的索引都已经cache到内存中，所以可以将访问索引的开销忽略不计。因为要读出4000万x 50% = 2000万数据，假设在读这2000万数据时，有99.9%的命中率，则还是需要20000次I/O,比上面的全表扫描需要的2500次多多了，所以在这种情况下，用索引扫描反而性能会差很多。在这样的情况下，用全表扫描的时间是固定的，但是用索引扫描的时间会随着选出数据的增多使查询时间相应的延长。

基于规则的优化器使用的执行路径与各个路径对应的等级

- ✓ RBO Path 1: Single Row by Rowid(等级最高)
- ✓ RBO Path 2: Single Row by Cluster Join
- ✓ RBO Path 3: Single Row by Hash Cluster Key with Unique or Primary Key
- ✓ RBO Path 4: Single Row by Unique or Primary Key
- ✓ RBO Path 5: Clustered Join
- ✓ RBO Path 6: Hash Cluster Key
- ✓ RBO Path 7: Indexed Cluster Key
- ✓ RBO Path 8: Composite Index
- ✓ RBO Path 9: Single-Column Indexes
- ✓ RBO Path 10: Bounded Range Search on Indexed (等级最低)

基于代价的优化器 -- Cost Based Optimization(简称CBO)

- Oracle把一个代价引擎(Cost Engine)集成到数据库内核中，用来估计每个执行计划需要的代价，该代价将每个执行计划所耗费的资源进行量化，从而CBO可以根据这个代价选择出最优的执行计划。
- 一个查询耗费的资源可以被分成3个基本组成部分：I/O代价、CPU代价、Network代价。I/O代价是将数据从磁盘读入内存所需的代价。

- 访问数据包括将数据文件中数据块的内容读入到SGA的数据高速缓存中，在一般情况下，该代价是处理一个查询所需要的最主要代价，所以我们在优化时，一个基本原则就是降低查询所产生的I/O总次数。
- CPU代价是处理在内存中数据所需要的代价，如一旦数据被读入内存，则我们在识别出我们需要的数据后，在这些数据上执行排序(sort)或连接(join)操作，这需要耗费CPU资源。

- 对于需要访问跨节点(即通常说的服务器)数据库上数据的查询来说，存在network代价，用来量化传输操作耗费的资源。查询远程表的查询或执行分布式连接的查询会在network代价方面花费比较大。

- 使用CBO时，需要有表和索引的统计数据(分析数据)作为基础数据。依据这些数据，CBO对各个执行计划计算出相对准确的代价，从而使CBO选择最佳的执行计划
- 定期的对表、索引进行分析是绝对必要的，这样才能使统计数据反映数据库中的真实情况。否则就会使CBO选择较差的执行计划，影响数据库的性能。分析操作不必做的太频繁
- ANALYZE命令DBMS_STATS存储包在10g后自动安装。

优化技术的选择

- 主要是由optimizer_mode初始化参数决定的。
- 该参数可能的取值为：
  - first_rows_[1 | 10 | 100 | 1000]
  - first_rows
  - all_rows
  - choose
  - Rule

- RULE为使用RBO优化器。
- CHOOSE则是根据实际情况，如果数据字典中包含被引用的表的统计数据，即引用的对象已经被分析，则就使用CBO优化器，否则为RBO优化器。
- ALL_ROWS为CBO优化器使用的第一种具体的优化方法，是以数据的吞吐量为主要目标，以便可以使用最少的资源完成语句。
-

- FIRST_ROWS为优化器使用的第二种具体的优化方法，是以数据的响应时间为主要目标，以便快速查询出开始的几行数据。
- FIRST_ROWS_[1 | 10 | 100 | 1000] 为优化器使用的第三种具体的优化方法，让优化器选择一个能够把响应时间减到最小的查询执行计划，以迅速产生查询结果的前 n 行。该参数为ORACLE 9I新引入的。

- 在oracle 10g中，CBO可选的运行模式有两种：First_ROWS(n) ALL_ROWS(10g中默认)
- 查看CBO模式的命令

Show parameter optimizer_mode

- 修改CBO模式的三种方法
- (1) SQL 语句
- Session级别：alter session set optimizer_mode=all_rows
- (2) 修改pfile参数
- optimizer_mode=RULE/CHOOSE/FIRST_ROWS/ALL_ROWS

- (3) 语句级别的hint(/*+…*/)
- Select /*+first_rows(10)*/ name from table

什么是优化？

- 优化是选择最有效的执行计划来执行SQL语句的过程，这是在处理任何数据的语句(SELECT,INSERT,UPDATE或DELETE)中的一个重要步骤。
- 对Oracle来说，执行这样的语句有许多不同的方法，譬如说，将随着以什么顺序访问哪些表或索引的不同而不同。所使用的执行计划可以决定语句能执行得有多快。Oracle中称之为优化器(Optimizer)的组件用来选择这种它认为最有效的执行计划。

执行计划的访问路径

- 在物理层，**oracle**读取数据，一次读取的最小单位为数据库块(由多个连续的操作系统块组成)
- 一次读取的最大值由操作系统一次I/O的最大值与multiblock参数共同决定，所以即使只需要一行数据，也是将该行所在的数据库块读入内存。

逻辑读的方法

（1) 全表扫描(Full Table Scans, FTS)

- 为实现全表扫描，Oracle读取表中所有的行，并检查每一行是否满足语句的WHERE限制条件。
- Oracle顺序地读取分配给表的每个数据块，直到读到表的最高水线处(high water mark, HWM，标识表的最后一个数据块)。
- 一个多块读操作可以使一次I/O能读取多块数据块(db_block_multiblock_read_count参数设定)，而不是只读取一个数据块，这极大的减少了I/O总次数，提高了系统的吞吐量，所以利用多块读的方法可以十分高效地实现全表扫描，而且只有在全表扫描的情况下才能使用多块读操作。

- 在这种访问模式下，每个数据块只被读一次。由于HWM标识最后一块被读入的数据，而delete操作不影响HWM值，所以一个表的所有数据被delete后，其全表扫描的时间不会有改善，一般我们需要使用truncate命令来使HWM值归为0。
- 幸运的是oracle 10G后，可以人工收缩HWM的值。

- 使用FTS的前提条件：在较大的表上不建议使用全表扫描，除非取出数据的比较多，超过总量的5% -- 10%，或你想使用并行查询功能时。
- 例如：

```
Set autotrace on
Set autotrace traceonly
explain plan for select * from dual;
```

(2) 通过ROWID的表存取(Table Access by ROWID或rowid lookup)

- 行的ROWID指出该行所在的数据文件、数据块以及行在该块中的位置，所以通过ROWID来存取数据可以快速定位到目标数据上，是Oracle存取单行数据的最快方法。

- 为了通过ROWID存取表，Oracle 首先要获取被选择行的ROWID，或者从语句的WHERE子句中得到，或者通过表的一个或多个索引的索引扫描得到。Oracle然后以得到的ROWID为依据定位每个被选择的行。

- explain plan for select * from dept where rowi d = 'AAAAyGAADAAAAAATAAF‘

- select * from table(DBMS_XPLAN.DISPLAY)

Oracle ROWID格式及rdba

- Oracle 8i以上smallfile表空间的ROWID格式是：OOOOOO.FFF.BBBBBB.RRR，其中:
- O－－对象号
- F－－文件号
- B－－块号
- R－－行号
- ROWID是一个64位数，共18位。

64位编码表如下：

64位码	对应十进制数值
A～Z	0～25
a～z	26～51
0～9	52～61
+	62
/	63

- ROWID存储为10个字节，共80位，组成形式：32bit obj# +10bit rfile#+22bit block# +16bit row#
- 可以使用dbms_rowid包对ROWID进行解析：

```
select dbms_rowid.rowid_object('&&rowid') obj_id#,
dbms_rowid.rowid_relative_fno('&&rowid') rfile#,
dbms_rowid.rowid_block_number('&&rowid') block#,
dbms_rowid.rowid_row_number('&&rowid') row#
from dual;
```

- 从Oracle10g开始，Oracle引入了bigfile表空间，ROWID格式随之变为：OOOOOO.LLLLLLLLL.RRR，其中：
- O－－对象号
- L－－块号
- R－－行号
- 存储格式也调整为：32bit obj# + 32bit block# +16bit row#

- RDBA是relative data block address, 即相对数据块地址。在DUMP数据库块时会经常看到RDBA，RDBA是一个16进制数，组成形式：10bit rfile#+22bit block#。

- 以下两种方法可以用来转换RDBA：

➢ 1. 使用dbms_utility包

```
select
dbms_utility.data_block_address_file(to_number('&&rdba','
xxxxxxxxxxxxx')) file_id,
dbms_utility.data_block_address_block(to_number('&&rdb
a','xxxxxxxxxxxxx')) block_id from dual;
```

➢ 2. 直接解析

```
select
trunc(to_number('&&rdba','xxxxxxxxxxxxx')/power(2,22))
file_id,
mod(to_number('&&rdba','xxxxxxxxxxxxx'),power(2,22))
block_id from dual;
```

➢(3)索引扫描(Index Scan或index lookup)

- 通过index查找到数据对应的rowid值(对于非唯一索引可能返回多个rowid值)，然后根据rowid直接从表中得到具体的数据，这种查找方式称为索引扫描或索引查找(index lookup)。
- 一个rowid唯一的表示一行数据，该行对应的数据块是通过一次i/o得到的，在此情况下该次i/o只会读取一个数据库块。

```
select empno, ename from emp where empno=10
select empno, ename from emp
where empno > 7876 order by empno;
```

4种类型的索引扫描

- 索引唯一扫描(index unique scan)
- 索引范围扫描(index range scan)
- 索引全扫描(index full scan)
- 索引快速扫描(index fast full scan)

(1) 索引唯一扫描(index unique scan)

- 通过唯一索引查找一个数值经常返回单个ROWID。如果该唯一索引有多个列组成(即组合索引)，则至少要有组合索引的引导列参与到该查询中
  - 如创建一个索引：create index idx_test on emp(ename, deptno, loc)。则select ename from emp where ename = ‘JACK’ and deptno = ‘DEV’ 语句可以使用该索引。如果该语句只返回一行，则存取方法称为索引唯一扫描。
  - 而select ename from emp where deptno = ‘DEV’语句则不会使用该索引，因为where子句种没有引导列。如果存在UNIQUE 或 PRIMARY KEY 约束(它保证了语句只存取单行)的话，Oracle经常实现唯一性扫描。
  - select empno,ename from emp where empno=10;

(2) 索引范围扫描(index range scan)

- 使用一个索引存取多行数据

  如果索引是组合索引，而且select ename from emp where ename = ‘JACK’ and deptno = ‘DEV’语句返回多行数据，虽然该语句还是使用该组合索引进行查询，可此时的存取方法称为索引范围扫描。在唯一索引上使用索引范围扫描的典型情况下是在谓词(where限制条件)中使用了范围操作符(如>、<、<>、>=、<=、between)

使用index rang scan的3种情况

- (a) 在唯一索引列上使用了range操作符(> < <> >= <= between)。
- (b) 在组合索引上，只使用部分列进行查询，导致查询出多行。
- SNO,CNO,
- (c) 对非唯一索引列上进行的任何查询。

(3) 索引全扫描(index full scan)

- 与全表扫描对应，也有相应的全索引扫描。在某些情况下，可能进行全索引扫描而不是范围扫描，需要注意的是全索引扫描只在CBO模式下才有效。CBO根据统计数值得知进行全索引扫描比进行全表扫描更有效时，才进行全索引扫描，而且此时查询出的数据都必须从索引中可以直接得到。

(4) 索引快速扫描(index fast full scan)

- 扫描索引中的所有的数据块，与 index full scan很类似，但是一个显著的区别就是它不对查询出的数据进行排序，即数据不是以排序顺序被返回。在这种存取方法中，可以使用多块读功能，也可以使用并行读入，以便获得最大吞吐量与缩短执行时间。

表连接的方法

- 连接的分类：
  - 等值连接(如WHERE A.COL3 = B.COL4)
  - 非等值连接(WHERE A.COL3 > B.COL4)
  - 外连接(WHERE A.COL3 = B.COL4(+))。
- 以下将以这个例子：

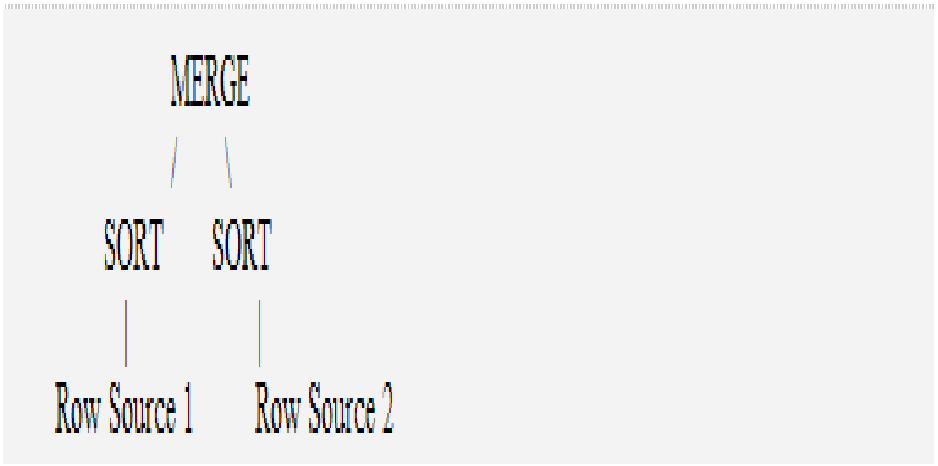
```
SELECT A.COL1, B.COL2 FROM A, B
WHERE A.COL3 = B.COL4;
```

连接类型

- 排序 - - 合并连接(Sort Merge Join (SMJ))
- 嵌套循环(Nested Loops (NL))
-
- 哈希连接(Hash Join)

排序 - - 合并连接(Sort Merge Join, SMJ)

- 内部连接过程：
  - 1) 首先生成row source1需要的数据，然后对这些数据按照连接操作关联列(如A.col3)进行排序。
  - 2) 随后生成row source2需要的数据，然后对这些数据按照与sort source1对应的连接操作关联列(如B.col4)进行排序。
  - 3) 最后两边已排序的行被放在一起执行合并操作，即将2个row source按照连接条件连接起来



- 合并两个row source的过程是串行的，但是可以并行访问这两个row source(如并行读入数据，并行排序).

SMJ连接的例子

```
select /*+ ordered */ e.deptno, d.deptno
 from emp e, dept d
 where e.deptno = d.deptno
 order by e.deptno, d.deptno;
```

```
Query Plan
----------------------------------------
SELECT STATEMENT [CHOOSE] Cost=17
  MERGE JOIN
    SORT JOIN
      TABLE ACCESS FULL EMP [ANALYZED]
    SORT JOIN
      TABLE ACCESS FULL DEPT [ANALYZED]
```

SMJ的缺点

- 排序是一个费时、费资源的操作，特别对于大表。基于这个原因，SMJ经常不是一个特别有效的连接方法。
- 如果2个row source都已经预先排序，则这种连接方法的效率也是比较高的。

嵌套循环(Nested Loops, NL)

内部连接过程：

```
Row source1的Row 1 ------------      -- Probe ->      Row source 2
Row source1的Row 2 ------------      -- Probe ->      Row source 2
Row source1的Row 3 ------------      -- Probe ->      Row source 2
.....
Row source1的Row n ------------      -- Probe ->      Row source 2
```

- 从内部连接过程来看，需要用row source1中的每一行，去匹配row source2中的所有行，所以此时保持row source1尽可能的小与高效的访问row source2(一般通过索引实现)是影响这个连接效率的关键问题。这只是理论指导原则，目的是使整个连接操作产生最少的物理I/O次数，而且如果遵守这个原则，一般也会使总的物理I/O数最少。但是如果不遵从这个指导原则，反而能用更少的物理I/O实现连接操作。

- NESTED LOOPS有其它连接方法没有的的一个优点是：可以先返回已经连接的行，而不必等待所有的连接操作处理完才返回数据，这可以实现快速的响应时间。
- 如：

```
select a.dname,b.sql
from dept a,emp b
where a.deptno = b.deptn
```

```
Query Plan
----------------------
SELECT STATEMENT [CHOOSE] Cost=5
  NESTED LOOPS
    TABLE ACCESS FULL DEPT [ANALYZED]
    TABLE ACCESS FULL EMP [ANALYZED]
```

哈希连接(Hash Join, HJ)

Hash join的工作方式是将一个表（通常是小一点的那个表）做hash运算，将列数据存储到hash列表中，从另一个表中抽取记录，做hash运算，到hash 列表中找到相应的值，做匹配。

笛卡儿乘积(Cartesian Product)

当两个row source做连接，但是它们之间没有关联条件时，就会在两个row source中做笛卡儿乘积，这通常由编写代码疏漏造成(即程序员忘了写关联条件)。

如何产生执行计划？

- Set autotrace on
- Set autotrace traceonly
- 调用**dbms_system存储过程生成执行计划**

干预执行计划

- 在**ORACLE**中，是通过为语句添加**hints**(提示)来实现**干预优化器优化**的目的。

- hints是oracle提供的一种机制，用来告诉优化器按照我们的告诉它的方式生成执行计划。我们可以用hints来实现：
  - 1) 使用的优化器的类型
  - 2) 基于代价的优化器的优化目标，是all_rows还是first_rows。
  - 3) 表的访问路径，是全表扫描，还是索引扫描，还是直接利用rowid。
-

- Hints只应用在它们所在sql语句块(statement block，由select、update、delete关键字标识)上，对其它SQL语句或语句的其它部分没有影响

Hints的语法

使用hints的语法：

{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */

or

{DELETE|INSERT|SELECT|UPDATE} --+ hint [text] [hint[text]]...

注解：

1) DELETE、INSERT、SELECT和UPDATE是标识一个语句块开始的关键字，包含提示的注释只能出现在这些关键字的后面，否则提示无效。

2) “+”号表示该注释是一个hints，该加号必须立即跟在”/*”的后面，中间不能有空格。

3) hint是下面介绍的具体提示之一，如果包含多个提示，则每个提示之间需要用一个或多个空格隔开。

4) text 是其它说明hint的注释性文本

- 4) 表之间的连接类型
- 5) 表之间的连接顺序
- 6) 语句的并行程度
- 除了”RULE”提示外，一旦使用的别的提示，语句就会自动的改为使用CBO优化器，此时如果你的数据字典中没有统计数据，就会使用缺省的统计数据。所以建议大家如果使用CBO或HINTS提示，则最好对表和索引进行定期的分析。

使用全套的hints

下面是一个使用全套hints的例子，ORDERED提示指出了连接的顺序，而且为不同的表指定了连接方法：

```
SELECT /*+ ORDERED INDEX (b, jl_br_balances_n1) USE_NL (j b)
USE_NL (glcc glf) USE_MERGE (gp gsb) */
b.application_id, b.set_of_books_id ,
b.personnel_id, p.vendor_id Personnel,
p.segment1 PersonnelNumber, p.vendor_name Name
FROM jl_br_journals j, jl_br_balances b,
gl_code_combinations glcc, fnd_flex_values_vl glf,
gl_periods gp, gl_sets_of_books gsb, po_vendors p
WHERE ...
```

指示优化器的方法与目标的hints

ALL_ROWS -- 基于代价的优化器，以吞吐量为目标

FIRST_ROWS(n) -- 基于代价的优化器，以响应时间为目标

CHOOSE -- 根据是否有统计信息，选择不同的优化器

RULE -- 使用基于规则的优化器

例子：

```
SELECT /*+ FIRST_ROWS(10) */ employee_id, last_name, salary, job_id
FROM employees
WHERE department_id = 20;
SELECT /*+ CHOOSE */ employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;
SELECT /*+ RULE */ employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;
```

- **指示存储路径的hints：**

  FULL /*+ FULL (table) */

  指定该表使用全表扫描

  ROWID /*+ ROWID (table) */

  指定对该表使用rowid存取方法，该提示用的较少

- INDEX /*+ INDEX (table [index]) */
- 使用该表上指定的索引对表进行索引扫描

  INDEX_FFS /*+ INDEX_FFS (table [index]) */
- 使用快速全表扫描

  NO_INDEX /*+ NO_INDEX (table [index]) */

  不使用该表上指定的索引进行存取，仍然可以使用其它的索引进行索引扫描

```
SELECT /*+ FULL(e) */ employee_id, last_name
FROM employees e
WHERE last_name LIKE :b1;
SELECT /*+ROWID(employees)*/ *
FROM employees
WHERE rowid > 'AAAAtkAABAAAFNTAAA' AND employee_id = 155;
SELECT /*+ INDEX(A sex_index) use sex_index because there are few
male patients */ A.name, A.height, A.weight
FROM patients A
WHERE A.sex = 'm';
SELECT /*+NO_INDEX(employees emp_empid)*/ employee_id
FROM employees
WHERE employee_id > 200;
```

指示连接顺序的hints

- ORDERED /*+ ORDERED */

  按from 字句中表的顺序从左到右的连接
- STAR /*+ STAR */

  指示优化器使用星型查询

```
SELECT /*+ORDERED */ o.order_id, c.customer_id, l.unit_price * l.quantity
FROM customers c, order_items l, orders o
WHERE c.cust_last_name = :b1
AND o.customer_id = c.customer_id
AND o.order_id = l.order_id;
/*+ ORDERED USE_NL(FACTS) INDEX(facts fact_concat) */
```

指示连接类型的hints：

USE_NL /*+ USE_NL (table [,table, ...]) */

使用嵌套连接

USE_MERGE /*+ USE_MERGE (table [,table, ...]) */

使用排序- -合并连接

USE_HASH /*+ USE_HASH (table [,table, ...]) */

使用HASH连接

注意：如果表有alias(别名)，则上面的table指的是表的别名，而不是真实的表名。

具体实例

- 见晚上实验

跟踪一个客户程序SQL

1) 识别要跟踪的客户端程序到数据库的连接(后面都用session代替)，主要找出能唯一识别一个session的sid与serial#.
2) 设定相应的参数，如打开时间开关(可以知道一个sql执行了多长时间)，存放跟踪数据的文件的位置、最大值。
3) 启动跟踪功能
4) 让系统运行一段时间，以便可以收集到跟踪数据
5) 关闭跟踪功能
6) 格式化跟踪数据，得到我们易于理解的跟踪结果。

1) 识别要跟踪的客户端程序到数据库的数据库连接q2

- set linesize 190
  col machine format a30 wrap
  col program for a40
  col username format a15 wrap
  set pagesize 500
  select s.sid sid, s.SERIAL# "serial#", s.username,
  s.machine, s.program,
  p.spid ServPID, s.server
  from v$session s, v$process p
  where p.addr = s.paddr ;

- username : 程序连接数据库的用户名
- machine : 连接数据库的程序所在的机器的机器名，可以hostname得到
- program : 连接数据库的程序名，所有用java jdbc thin的程序的名字都一样，
- servpid : 与程序对应的服务器端的服务器进程的进程号，在unix下比较有用
- server : 程序连接数据库的模式：专用模式(dedicaed)、共享模式(shared)。
- 只有在专用模式下的数据库连接，对其进程跟踪才有效
- logon_time : 程序连接数据库的登陆时间
- 根据machine, logon_time 可以方便的识别出一个数据库连接对应的session，从而得到该sesion的唯一标识sid, serial#, 为对该session进行跟踪做好准备

2) 设定相应的参数

- 参数说明：

  timed_statistics : 收集跟踪信息时，是否将收集时间信息，如果收集，则可以知道一个sql的各个执行阶段耗费的时间情况

  user_dump_dest : 存放跟踪数据的文件的位置

  max_dump_file_size : 放跟踪数据的文件的最大值，防止由于无意的疏忽，使跟踪数据的文件占用整个硬盘，影响系统的正常运行

  show parameter user_dump_dest

设置跟踪参数

```
exec
   sys.dbms_system.set_int_param_in_session( -
       sid => 161, -
       serial# => 10, -
       parnam => 'max_dump_file_size', -
       intval => 2147483647)
```

3) 启动跟踪功能

- exec sys.dbms_system.set_sql_trace_in_session(161, 10, true);

4) 让系统运行一段时间，以便可以收集到跟踪数据

- **格式化跟踪数据，得到我们易于理解的跟踪结果**

对产生的trace文件进行格式化：

在命令提示符下，运行下面的命令

tkprof orcl_ora_3444.trc dsdb2_trace.out SYS=NO EXPLAIN=SCOTT/oracle

启用Autotrace

- 通过以下方法可以把Autotrace的权限授予Everyone，如果你需要限制Autotrace权限，可以把对public的授权改为对user的授权。

创建PLAN_TABLE

```
D:\oracle\ora92>sqlplus /nolog

SQL*Plus: Release 9.2.0.1.0 - Production on 星期二 6月 3 15:16:03 2003

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

SQL> connect sys as sysdba
请输入口令:
已连接。
SQL> @?\rdbms\admin\utlxplan

表已创建。

SQL> create public synonym plan_table for plan_table;

同义词已创建。

SQL> grant all on plan_table to public ;

授权成功。
```

创建PLUSTRACE角色

```
SQL> @?\sqlplus\admin\plustrce
SQL>
SQL> drop role plustrace;
drop role plustrace
         *
ERROR 位于第 1 行:
ORA-01919: 角色'PLUSTRACE'不存在


SQL> create role plustrace;

角色已创建
```

```
SQL>
SQL> grant select on v_$sesstat to plustrace;

授权成功。

SQL> grant select on v_$statname to plustrace;

授权成功。

SQL> grant select on v_$session to plustrace;

授权成功。

SQL> grant plustrace to dba with admin option;

授权成功。

SQL>
SQL> set echo off
```

```
SQL> grant plustrace to public ;

授权成功。

SQL> connect eqsp/eqsp
已连接。
SQL> set autotrace on
SQL> set timing on

SQL>
```

然后执行相应的**SQL**语句就可以看到执行计划了。

Autotrace选项

SET AUTOTRACE OFF - 不生成AUTOTRACE 报告，这是缺省模式。
SET AUTOTRACE ON EXPLAIN - AUTOTRACE只显示优化器执行路径报告
SET AUTOTRACE ON STATISTICS - 只显示执行统计信息
SET AUTOTRACE ON - 包含执行计划和统计信息
SET AUTOTRACE TRACEONLY - 同set autotrace on，但是不显示查询输出

Example

```
SQL> set autotrace traceonly
SQL> select table_name from user_tables;

已选择98行。

已用时间:  00: 00: 00.04

Execution Plan
----------------------------------------------------------
   0     SELECT STATEMENT Optimizer=CHOOSE
   1    0  NESTED LOOPS
   2    1    NESTED LOOPS (OUTER)
   3    2      NESTED LOOPS (OUTER)
   4    3        NESTED LOOPS (OUTER)
   5    4          NESTED LOOPS (OUTER)
   6    5            NESTED LOOPS
   7    6              TABLE ACCESS (BY INDEX ROWID) OF 'OBJ$'
   8    7                INDEX (RANGE SCAN) OF 'I_OBJ2' (UNIQUE)
   9    6              TABLE ACCESS (CLUSTER) OF 'TAB$'
  10    9                INDEX (UNIQUE SCAN) OF 'I_OBJ#' (NON-UNIQUE)
  11    5            TABLE ACCESS (BY INDEX ROWID) OF 'OBJ$'
  12   11              INDEX (UNIQUE SCAN) OF 'I_OBJ1' (UNIQUE)
  13    4          INDEX (UNIQUE SCAN) OF 'I_OBJ1' (UNIQUE)
  14    3        TABLE ACCESS (CLUSTER) OF 'USER$'
  15   14          INDEX (UNIQUE SCAN) OF 'I_USER#' (NON-UNIQUE)
  16    2      TABLE ACCESS (CLUSTER) OF 'SEG$'
  17   16        INDEX (UNIQUE SCAN) OF 'I_FILE#_BLOCK#' (NON-UNIQUE)
  18    1    TABLE ACCESS (CLUSTER) OF 'TS$'
  19   18      INDEX (UNIQUE SCAN) OF 'I_TS#' (NON-UNIQUE)
```

```
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       1389  consistent gets
          0  physical reads
          0  redo size
       2528  bytes sent via SQL*Net to client
        569  bytes received via SQL*Net from client
          8  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
         98  rows processed

SQL>
```