收获，不止 SQL 优化

第 五 章

感受体系结构让 SQL 飞

E-Mail:yiren.job@gmail.com

目录

1.绑定变量的好处与坏处

◆ 详细问题：说说绑定变量的好处和坏处，请举场景说明。

1.1 好处和坏处原因概述

使用绑定变量的好处：在通常高并发的系统中，同一条 SQL 语句仅仅由于谓词部分变量的不同而在执行的时候需要重新解析，每一次解析都是一次硬解析，造成 SQL 执行计划不能共享，这极大地耗费了系统时间，假如银行的查询余额 SQL 语句，在成千上万次查询中可能只是账号名不同，其输出都一样，如果使用绑定变量，假设执行能够节省 0.001 秒，那么高并发下 10 万次、100 万次查询节省下来的时间将是很客观的，这就无形中提高了系统的响应时间。

使用绑定变量的坏处：使用绑定变量也不是万能的，不可一概而论，就目前随着业务系统复杂度越来越高，场景越来越多，在不同的场景下我们使用绑定变量有可能会取得适得其反的效果，因为使用绑定变量生成执行计划时数据库总是以第一个变量的值为准生成执行计划（变量窥探，这项技术在 11G 下已经有很多的改善），后者的变量将采用第一个值生成的执行计划直接使用，这可能会导致执行效果的偏差。

1.2 绑定变量好处的场景

```
SQL>
begin
    for i in 1 .. 100000
    loop
        execute immediate
        'insert into t values ( '||i||')';
    end loop;
    commit;
end;
  9  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:33.00
begin
    for i in 1 .. 100000
    loop
        execute immediate
        'insert into t values (:x)' using i;
    end loop;
        commit;
end;
  9  /

PL/SQL procedure successfully completed.

Elapsed: 00:00:04.77
```

上面没有使用绑定变量，下面使用了绑定变量，可以光从时间上就可以比较效率。

```
SQL> col sql_text for a30
select t.sql_text, t.sql_id, t.executions, t.parse_calls
  from v$sql t
  3   where sql_text like 'insert into t values%';

SQL_TEXT                       SQL_ID        EXECUTIONS PARSE_CALLS
------------------------------ ------------- ---------- -----------
insert into t values ( 99898)  8bfrzh2hms0av          1           1
insert into t values ( 99762)  0692hq6rxn0qa          1           1
insert into t values ( 99893)  bmpxvqbf2w0rw          1           1
insert into t values ( 99790)  9gt4jcv7a015g          1           1
insert into t values ( 99821)  cy31nkj0g01fc          1           1
insert into t values ( 99963)  39kus60wxs24s          1           1
insert into t values ( 99826)  7z4h4vd89w2tr          1           1
insert into t values ( 99776)  3uzm4zu1tn2xj          1           1
insert into t values ( 99892)  bjycy8pqm03f5          1           1
insert into t values ( 99822)  37dzbgqmr83j6          1           1
insert into t values ( 99801)  cj7mk23ja83vu          1           1

SQL_TEXT                       SQL_ID        EXECUTIONS PARSE_CALLS
------------------------------ ------------- ---------- -----------
insert into t values ( 99866)  bqh808z3t8458          1           1
insert into t values ( 99939)  gp0thbtpds4ra          1           1
insert into t values ( 99879)  d9s3sbx48n64f          1           1
insert into t values ( 99910)  22jktauhd465u          1           1
insert into t values ( 99905)  a3cht69fwc85b          1           1
insert into t values ( 99859)  b0s03fxj1s8by          1           1
insert into t values ( 99860)  aj92h6y10n8n4          1           1
```

这个相同语句只因谓词变量的不同，每个语句都需要重新解析执行

```
SQL>
select t.sql_text, t.sql_id, t.executions, t.parse_calls
  from v$sql t
  3   where sql_text like 'insert into t values (:x)%';

SQL_TEXT                       SQL_ID        EXECUTIONS PARSE_CALLS
------------------------------ ------------- ---------- -----------
insert into t values (:x)      bbz6pdq577unj     100000           1

Elapsed: 00:00:00.04
SQL>
```

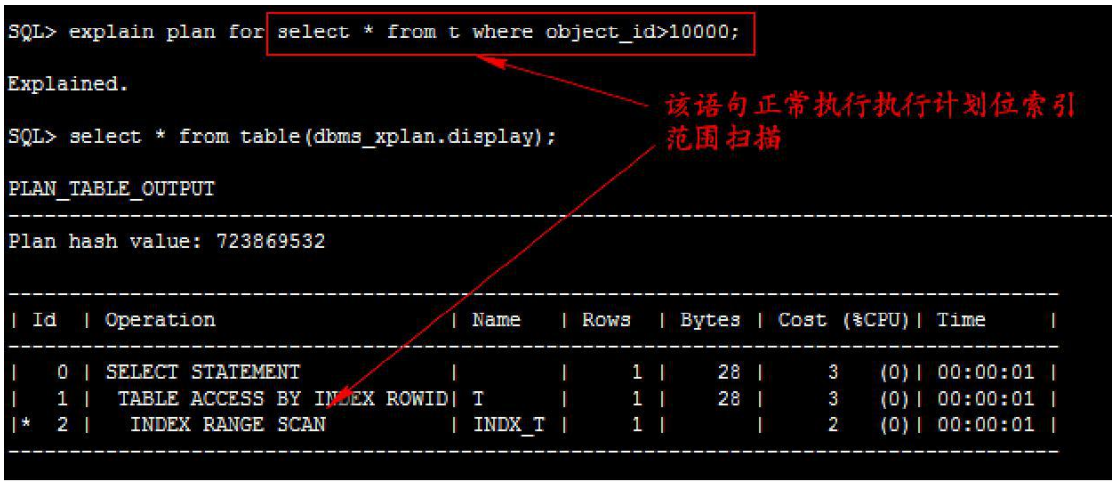
使用绑定变量后解析1次执行多次，省去解析时间

1.3 绑定变量坏处的场景

```
SQL> create table t as select rownum,object_name,object_type
  2  from dba_objects;
create table t as select rownum,object_name,object_type
                           *
ERROR at line 1:
ORA-00998: must name this expression with a column alias


SQL> create table t as select object_id,object_name,object_type
  2  from dba_objects;

Table created.

SQL> create index indx_t on t(object_id);

Index created.

SQL> exec dbms_stats.gather_table_stats(user,'T',cascade=>true);

PL/SQL procedure successfully completed.
```

```
SQL> set lines 120
SQL> explain plan for select * from t where object_id>10;

Explained.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 1601196873

------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |  9409 |   257K|    14   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |  9409 |   257K|    14   (0)| 00:00:01 |
------------------------------------------------------------------------------
```

该语句正常执行计划为全表扫描


SQL> explain plan for select * from t where object_id>10000;
Explained.
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
Plan hash value: 723869532
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	28	3 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T	1	28	3 (0)	00:00:01
* 2	INDEX RANGE SCAN	INDX_T	1		2 (0)	00:00:01
该语句正常执行执行计划位索引范围扫描


SQL> var v_cnt number;
SQL> exec :v_cnt:=10;
PL/SQL procedure successfully completed.
SQL> explain plan for select * from t where object_id>:v_cnt;
Explained.
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
Plan hash value: 723869532
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		471	13188	3 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T	471	13188	3 (0)	00:00:01
* 2	INDEX RANGE SCAN	INDX_T	85		2 (0)	00:00:01
使用绑定变量
该语句使用绑定变量时索引扫描与正常执行的执行计划明显是错误的。


SQL> exec :v_cnt:=10000;
PL/SQL procedure successfully completed.
SQL> explain plan for select * from t where object_id>:v_cnt;
Explained.
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
Plan hash value: 723869532
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		471	13188	3 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	T	471	13188	3 (0)	00:00:01
* 2	INDEX RANGE SCAN	INDX_T	85		2 (0)	00:00:01
其实现在不论给变量v_cnt赋予扫描值，该条SQL都是沿用上次生成的执行计划。

2. 说说直接路径访问方式为什么更快

2.1 直接路径访问概念

直接路径访问是绕过 SGA，直接把数据读入到 PGA 中，这个过程数据不经过 SGA 的缓冲，所以理论上应该是更快。在 PGA 中的数据只能有当前 PGA 进程能够访问，当其他会话需要访问这部分数据时需要从磁盘读取数据，发生磁盘读。

2.2 直接路径插入试验

```
SQL> create table t as select * from dba_objects;

Table created.

Elapsed: 00:00:00.25
SQL> insert into t select * from dba_objects;

9416 rows created.

Elapsed: 00:00:00.10
SQL> /                                    创建模拟测试环境

9416 rows created.

Elapsed: 00:00:00.09
SQL> /

9416 rows created.

Elapsed: 00:00:00.09
SQL> commit;

Commit complete.

Elapsed: 00:00:00.09
SQL> create table test1 as select * from dba_objects;

Table created.

Elapsed: 00:00:00.29
SQL> create table test2 as select * from dba_objects;

Table created.

Elapsed: 00:00:00.25
```

```
SQL> delete test1 where object_id between 300 and 1200;

866 rows deleted.

Elapsed: 00:00:00.03
SQL> delete test1 where object_id between 1500 and 3000;

1501 rows deleted.

Elapsed: 00:00:00.04
SQL> delete test1 where object_id between 4000 and 5000;

976 rows deleted.

Elapsed: 00:00:00.03
SQL> delete test2 where object_id between 300 and 1200;

866 rows deleted.

Elapsed: 00:00:00.04
SQL> delete test2 where object_id between 1500 and 3000;

1501 rows deleted.

Elapsed: 00:00:00.04
SQL>  delete test2 where object_id between 4000 and 5000;

976 rows deleted.

Elapsed: 00:00:00.03
SQL> commit;

Commit complete.
```

对TEST1和TEST2表执行相同操作

```
SQL> alter system flush buffer_cache;

System altered.

Elapsed: 00:00:00.40
SQL> insert /*+append*/into test1 select * from t;

37664 rows created.

Elapsed: 00:00:00.24
SQL> commit;

Commit complete.
SQL> alter system flush buffer_cache;

System altered.

Elapsed: 00:00:02.02
SQL> /

System altered.

Elapsed: 00:00:00.00
SQL> /

System altered.

Elapsed: 00:00:00.00
SQL> insert into test2 select * from t;

37664 rows created.

Elapsed: 00:00:00.31
```

从执行时间上，直接插入要比普通插入效率要高一些

3.缓存结果集、什么时候能用 keep 内存

◆ 详细问题：请用自己的话描述什么时候能用缓存结果集、什么时候能用 keep 内存。

3.1 什么时候能用缓存结果集

首先缓存结果集是 ORACLE 提供了一些设置接口使数据库管理员能够将经常访问的数据保存在内存中，从而为以后的相同结果集的查询减少或消除数据库的物理 I/O，提高系统的响应时间。

只有 result_cache_max_size 的参数值不为 0 的时候才可以使用缓存结果集。

```
SQL> Alter system set result_cache_max_size=10M;

System altered.

SQL> alter system set result_cache_max_result=20;

System altered.

SQL> alter system set result_cache_remote_expiration=3600;

System altered.

SQL> show parameter result_

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
client_result_cache_lag              big integer 3000
client_result_cache_size             big integer 0
result_cache_max_result              integer     20
result_cache_max_size                big integer 10M
result_cache_mode                    string      MANUAL
result_cache_remote_expiration       integer     3600
```

可以通过 dbms_result_cache.status() 查看是否开启了缓存结果集。缓存结果集是 ORACLE11G 的新特性。

3.2 什么时候能用 KEEP 内存

所谓 KEEP 内存就是把一些常用的数据对象常驻内存中（data buffer），不受 oracle LRU 算法的约束。在一些 OLTP 系统中一些关键的核心的小表可以 keep 到内存中，例如系统数据字典表，通过 keep 到内存中一些对象，可以提高 SQL 之间关联的效率。例如下面是演示 keep 一个对象：

```
SQL> show parameter db_keep_cache

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_keep_cache_size                   big integer 0
SQL> alter system set db_keep_cache_size=20m;

System altered.

SQL> create table test(id int ,city varchar(30));

Table created.

SQL> insert into test values(1,'beijing');

1 row created.

SQL> insert into test values(2,'shanghai');

1 row created.

SQL> commit;

Commit complete.

SQL> alter table test storage(buffer_pool keep);

Table altered.

SQL> create index idx_test on test(id);

Index created.

SQL> alter index idx_test storage(buffer_pool keep);

Index altered.
```

可以通过 alter xxx nocache;把取消 cache 一个对象。