ORACLE 数据库 SQL 优化原则①

郭　珉 (山西财经大学 信息管理学院 山西 太原 030006)

摘　要： Oracle 数据库是当前应用最广泛的大型数据库之一，其系统结构复杂，性能受多方面因素影响，其中 SQL 语句的执行效率是影响其性能的关键因素。以一个省级通信运营商的 ORACLE ERP 系统为例，从 ORACLE 数据库的 SQL 共享原理和 SQL 执行过程入手，指出合理配置数据库参数，提高 SQL 语句共享、提高数据缓存命中率是 SQL 语句性能提高的前提；并在此基础提出了 SQL 语句优化的四个原则。

关键词： Oracle 数据库；SQL；性能优化；优化器；共享池

Optimization Principles of SQL in ORACLE Database

GUO Min

(School of Information Management, Shanxi University of Finance and Economics, Taiyuan 030006, China)

Abstract: Oracle is one of the most widely-used large-scale databases at present. For its complex architecture, Oracle's performance is subjected to various factors, and query operation is one of the key factors. This paper analyzes the sharing principle and execution process of SQL statement in Oracle database. It finds out that improving the sharing of SQL statements and the data cache hit rate of the SQL statement is the premise of performance tuning. Lastly, it presents four principles for SQL performance tuning.

Keywords: oracle database; SQL; performance tuning; optimizer; shared buffer pool

从很多系统的应用来看，SQL 语句的书写技巧和优劣往往是系统在使用一段时间后性能是否下降的关键因素。书写 SQL 语句的开发人员往往只注重 SQL 语句的执行结果，忽略语句的执行效率。开发人员对 ORCLE 优化器原理[1,2]和 SQL 语句执行原理[1]不关心、不了解，往往使简单的 SQL 语句不能共享，导致 SQL 语句频繁解析；不能正确使用索引[3]，导致数据库缓存命中率降低，加重系统 I/O 负载。

本文以一个省级通信运营商的 ORACLE ERP 系统为例，从 ORACLE 选择数据库优化器、提高 SQL 共享和 SQL 语句执行过程入手，强调合理配置数据库参数，提高数据缓存和库缓存命中率是数据库高效执行 SQL 语句的前提；并在此基础上提出了优化 SQL 语句的四个基本原则。

1　优化器选择和提高SQL语句共享

1.1 选用适合的 Oracle 优化器

Oracle 的优化器共有 3 种：RULE (基于规则)、COST (基于成本)、CHOOSE(选择性)。通过对 init.ora 文件中的 OPTIMIZER_MODE 参数的各种声明(RULE，COST，CHOOSE，ALL_ROWS，FIRST_ROWS)，设置缺省的优化器。或者在 SQL 句级或是会话(session)级对其进行覆盖。为了使用基于成本的优化器(CBO, Cost-Based Optimizer)，必须经常运行 analyze 命令来增加数据库中的对象统计信息(object statistics)的准确性。如果数据库的优化器模式设置为选择性(CHOOSE)，那么实际的优化器模式将和是否运行过 analyze 命令有关。如果 table 已经被 analyze 过，优化器模式将自动成为 CBO，反之，数据库将采用 RULE 形式的优化器。

在缺省情况下，Oracle 采用 CHOOSE 优化器，为了避免那些不必要的全表扫描(full table scan)，须尽量避免使用 CHOOSE 优化器，而直接采用基于规则或者基于成本的优化器。

1.2 共享 SQL 语句

为了不重复解析相同的 SQL 语句，在第一次解析

① 收稿时间:2009-07-19;收到修改稿时间:2009-08-21

之后， Oracle 将 SQL 语句存放在内存中。这块位于系统全局区域 SGA (system global area)的共享池[4,5](shared buffer pool)中的内存可以被所有的数据库用户共享，节省内存的使用。

当向 Oracle 提交一个 SQL 语句，Oracle 首先在这块内存中查找相同的语句。这里需要注明的是，Oracle 对两者采取的是一种严格匹配。因此，当执行一个 SQL 语句时，如果它和之前执行过的语句完全相同，Oracle 就能很快获得已经被解析的语句以及最好的执行路径。Oracle 的这个功能大大地提高了 SQL 的执行性能并共享，SQL 语句必须完全相同(包括空格，换行等)。共享的语句必须满足三个条件：

(1) 字符级的比较。当前被执行的语句和共享池中的语句必须完全相同。例如：SELECT * FROM FND_USER；

和下列每一个都不同

SELECT * from FND_USER；

SELECT * FROM fnd_user；

(2) 两个语句所指的对象必须完全相同。例如：

表 1 员工基本信息

用户	对象	访问方法
APPS	FND_USER	PUBLIC SYNONYM
APPS	CUST_COMP_MOD	TABLE OWNER
APPSREAD	FND_USER	PUBLIC SYNONYM
APPSREAD	CUST_COMP_MOD	PUBLIC SYNONYM

当 APPS 和 APPSREAD 执行 SELECT * FROM FND_USER 时，都通过 PUBLIC SYNONYM 访问对象，SQL 语句能够共享。

而当 APPS 和 APPSREAD 执行 SELECT * FROM CUST_COMP_MOD 时，由于 APPSREAD 通过 PUBLIC SYNONYM 访问对象，APPS 是对象的所有者，虽然 SQL 语句完全相同，但是不能实现语句共享。

(3) 两个 SQL 语句必须使用相同名字的绑定变量(bind variables)。例如：第一组的两个 SQL 语句是相同的(可以共享)，而第二组中的两个语句是不同的(即使在运行时,赋予不同绑定变量相同的值) 。

第一组：

```
SELECT USER_NAME, EMAIL_ADDRESS
FROM   FND_USER WHERE USER_ID=:USER_ ID;
SELECT USER_NAME, EMAIL_ADDRESS
FROM   FND_USER WHERE USER_ID=:USER_ID;
```

第二组：

```
SELECT USER_NAME, EMAIL_ADDRESS
FROM   FND_USER WHERE USER_ID=:USER_ID;
SELECT USER_NAME, EMAIL_ADDRESS
FROM   FND_USER WHERE USER_ID=:FUSER_ID;
```

2 Oracle SQL语句的执行过程

当用户提交一条 SQL 语句时，Oracle 会做如下操作：

第一步：客户端把语句发给服务器端执行。

第二步：语句解析。当客户端把 SQL 语句传送到服务器后，服务器进程会对该语句进行解析。服务器端具体做以下工作：(1)查询高速缓存。如果在高速缓存中刚好有其用户使用过这个查询语句，则服务器进程就会直接执行这个 SQL 语句，省去后续的工作。(2)语句合法性检查。 (3)语言含义检查。 (4)获得对象解析锁。语法、语义都正确后，系统就会对需要查询的对象加锁。 (5)数据访问权限的核对。(6)确定最佳执行计划。服务器进程会根据一定的规则，对这条语句进行优化。当服务器进程的优化器确定这条查询语句的最佳执行计划后，就会将这条 SQL 语句与执行计划保存到数据高速缓存。等以后还有这个查询时，就会省略以上的语法、语义与权限检查的步骤，而直接执行 SQL 语句，提高 SQL 语句处理效率。

第三步：语句执行。分两种情况：(1)被选择的数据块已经被读取到数据缓冲区，服务器进程会直接把这个数据传递给客户端。(2)数据不在缓冲区中，服务器进程将从数据库文件中查询相关数据，并把这些数据放入到数据缓冲区中。

第四步：提取数据。返回语句执行结果。

从 SQL 语句执行过程来看，要想提高 SQL 语句执行效率，除了必备的 SQL 语句书写规范和技巧外，数据库的 DB_CACHE_SIZE 和 SHARE_POOL_SIZE[5,6]两个参数起着至关重要的作用。

本文将 ERP 系统的 DB_CACHE_SIZE 由 4G 该成 8G，数据库的数据缓存命中率相应提高了 3 个百分点，达到 98%，下面是监控数据库数据缓存命中率的语句：

```
Select round((1-(sum(decode(name, 'phy- sical reads', value, 0))/(sum(decode(name, 'db block gets', value, 0))+sum(decode(name, 'consistent gets',
```

value, 0))))), 2) "data buffer hit ratio " from v$sysstat;

将 SHARE_POOL_SIZE 由 2000M 扩大到 2500M，库缓存命中率由 95%提高到 99%，下面是监控数据库库缓存命中率的语句：

Select sum(pins)"total pins", sum(reloads) "total reloads", round ((1-(sum (reloads) / sum (pins))), 4) "library cache hit ratio" from v$libr- arycache;

3 SQL查询性能优化原则

3.1 优化排序操作

3.1.1 对经常进行排序和连接操作的字段建立索引

建立索引后，当服务器向这些字段发出排序请求时，将直接引用索引而不进行排序操作；为了使用基于成本的优化器，必须经常运行 analyze 命令，以增加数据库中的对象统计信息(object statistics)的准确性。

随着基表数据的不断增加，索引块会不断分裂以保持二叉树的平衡，树的层次也不断增加，层次大于 4 的索引宜重建;基表数据删除时，索引项也随之删除，但叶块上的空间并不能被重用，除非该叶块上所有的索引项都被删除，当删除项占所有项超过 20%时，这个索引也需要重建。

有三种重建索引的方法：

(1) 先删除再重建。这种方法耗费最多的资源，是早期版本的唯一方法；

(2) ALTER INDEX … REBUILD。这种方法高效快速，但需要额外的磁盘空间。用这种方法可以指定许多选项如 ONLINE(在线重建可减少锁争用)，TABLESPACE(移动段到其它表空间)，COMPUTE STATISTICS(统计)，PARALLEL(并行)， NOLOG- GING(尽可能少地产生日志)；

(3) ALTER INDEX … COALESCE。这种方法快速，无需额外空间，锁争用也少，缺点是选项少。

3.1.2 书写 SQL 语句要尽可能使用索引

创建索引后，如果 SQL 语句书写不当，优化器生成的执行计划也不会包含索引，应该避免下列情况：

(1) 对 WHERE 条件后的列使用函数。

```
SELECT ITEM_ID,ITEM_DESC FROM CHECK_ ITEM
WHERE UPPER(ITEM_ID)= ‘ A_INC ’
```

即使对该表的 ITEM_ID 字段创建了索引，该执行计划也不会使用到索引，建议使用下面语句建立函数索引。

```
CREATE INDEX UPPER_ITEM_ID ON CHECK_ ITEM(UPER(ITEM_ID));
```

同时可以将执行语句简化为下面书写方式，就可以正常使用索引。

```
SELECT ITEM_ID,ITEM_DESC FROM CHECK_ ITEM
WHERE ITEM_ID= ’ A_INC ’
```

(2) 避免 ‘ < > ’ 和 ‘ != ’ 操作符使用。

```
SELECT ITEM_ID,ITEM_DESC FROM CHECK_ ITEM
WHERE ITEM_ID< > ‘ A_INC ’
```

可以改写为：

```
SELECT ITEM_ID,ITEM_DESC FROM CHECK_ ITEM
WHERE ITEM_ID> ‘ A_INC ’ OR ITEM_ID< ‘ A_INC ’
```

(3) 比较不匹配的数据类型，ITEM_ID 字段是 VARCHAR2 类型,如下查询：

```
SELECT ITEM_ID,ITEM_DESC FROM CHECK_ ITEM
WHERE ITEM_ID=100
```

Oracle 会自动将语句变为 WHERE TO_NUM- BER(ITEM_ID)=100，导致索引无法使用，应将语句写为：SELECT ITEM_ID,ITEM_DESC

```
FROM CHECK_ITEM  WHERE ITEM_ID= ‘ 100 ’
```

3.2 减少排序

3.2.1 用 UNION ALL 代替 UNION

UNION 在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表 UNION，如：

```
SELCT * FROM CW_INCOME
UNION
SELECT * FROM CW_REALINCOM
```

这个 SQL 在运行时先取出两个表的结果，再用排序空间进行排序删除重复的记录，最后返回结果集，如果表数据量大的话可能会导致用磁盘进行排序。

推荐方案 :采用 UNION ALL 操作符替代 UNION，因为 UNION ALL 操作只是简单的将两个结果合并后就返回。

3.2.2 使用 WHERE 代替 HAVING

避免使用 HAVING 子句，HAVING 只会在检索出所有记录之后才对结果集进行过滤。这个处理需要排序、总计等操作。 如果能通过 WHERE 子句限制记录的数目，那就能减少这方面的开销。例如：

低效：

```
SELECT  SET_OF_BOOKS_ID_PERIOD_NAME,
SUM(QUARTER_TO_DATE_DR)
FROM GL_BALANCES
GROUP BY PERIOD_NAME
HAVING PERIOD_NAME<>‘JAN-06’
```

高效：

```
SELECT  SET_OF_BOOKS_ID_PERIOD_NAME,
SUM(QUARTER_TO_DATE_DR)
FROM GL_BALANCES
WHERE PERIOD_NAME<>‘JAN-06’
GROUP BY PERIOD_NAME
```

3.3 减少 I/O

过多的I/O操作会占用CPU时间、消耗大量内存，因此有必要对 SQL 的 I/O 进行优化。

(1) 尽量使用索引,避免全表扫描

尤其是在频繁使用的、数据量较大的表上建立合理索引，如函数索引、复合索引。

(2) WHERE 子句中的连接顺序

Oracle 采用自下而上 WHERE 子句，根据这个原理，表之间的连接必须写在其他 WHERE 条件之前，那些可以过滤掉最大数量记录的条件必须写在 WHERE 子句的末尾。如：

```
SELECT PERIOD_NAME,SUM(QUARTER_TO_ DATE_
DR)
FROM GL_BALANCES
WHER EPERIOD_NAME=‘JAN-06’
AND ACTUAL_FLAG=‘A’
GROUP BY PERIOD_NAME
```

(低效的，执行时间为 15.2 秒)

```
SELECT  PERIOD_NAME,SUM(QUARTER_TO_DATE_
DR)
FROM GL_BALANCES
WHERE ACTUAL_FLAG = ‘A’
AND PERIOD_NAME = ‘JAN-06’
GROUP BY PERIOD_NAME
```

(高效的，执行时间为 3.6 秒)

(3) 选择最有效率的表名顺序

Oracle的解析器按照从右到左的顺序处理FROM子句中的表名。在 FROM 子句中包含多个表的情况下，必须选择记录条数最少的表作为基础表。当 Oracle 处理多个表时，会运用排序及合并的方式连接它们。首先，扫描第一个表(FROM 子句中最后的那个表)并对记录进行排序，然后扫描第二个表(FROM 子句中最后第二个表)，最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。例如：表 TAB1 有 10000 条记录,表 TAB2 中有 200 条记录。

选择 TABLE2 作为基础表，效果比较好。

```
SELECT CONL1,CONL2 FROM TAB1,TAB2
WHERE …
```

选择 TABLE1 作为基础表，执行效率不理想

```
SELECT CONL1,CONL2 FROM TAB1,TAB2
WHERE …
```

3.4 减少表的访问次数

执行 SQL 语句时，数据库内部要执行许多工作，对系统资源消耗严重。如果能将相关的 SQL 语句进行有效整合，可以大大降低数据库的访问次数，从而提高系统性能。例如将下面两条语句整合为一条：

```
SELECT  SET_OF_BOOKS_ID, PERIOD_NAME,
    SUM(QUUARTER_TO_DATE_DR)
FROM    GL_BALANCES
GROUP BY PERIOD_NAME
HAVING PERIOD_NAME=‘JAN-06’
SELECT  SET_OF_BOOKS_ID, PERIOD_NAME,
    SUM(QUUARTER_TO_DATE_DR)
FROM    GL_BALANCES
GROUP BY PERIOD_NAME
HAVING PERIOD_NAME=‘JAN-08’
```

整合后：

```
SELECT  SET_OF_BOOKS_ID, PERIOD_NAME,
    SUM(QUUARTER_TO_DATE_DR)
FROM    GL_BALANCES
WHERE   PERIOD_NAME IN (‘JAN-06’,‘JAN-08’)
GROUP BY PERIOD_NAME
```

4 结论

通过使用本文提出的SQL优化原则对某省级通信运营商的 Oracle ERP 系统进行优化，取得了不错的效果。用户终端响应时间明显减少，系统的数据缓存命中率和库缓存命中率有了相应提高。ORACLE 数据库内部结构复杂，影响系统性能因素较多，但本文认为，

(下转第 165 页)

(上接第173页)

在系统硬件性能一定的情况下，影响系统性能的原因万变不离其宗，SQL 语句的优化是性能得以提高的根本，通过优化和整合低效的 SQL 语句，往往会带来意想不到的性能提高。SQL 优化方法众多，遵循本文提出的优化原则，数据库 SQL 优化从四个角度着手，可以使优化效果指标化，数据库性能提高显著。

参考文献

1 杨小艳,尹明,戴学丰.Oracle 数据库查询优化方法研究.计算机与现代化, 2008(4):4－7.

2 Belknap P, Dageville B, Dias K, et al. Selt-Tuning for SQL Performance in Oracle Database 11g.IEEE International Conference on Data Engineerint, 2009. 1694－1700.

3 谷小秋,李得昌.索引调整优化 Oracle9i 工作性能的研究.计算机工程与应用, 2005,(26):174－176.

4 杨厚云,龚汉明,武装.Oracle 数据库性能优化方案.北京机械工业学院学报, 2006,(4):55－59.

5 戴小平.Oracle 9i 数据库性能调整与优化.安徽工业大学学报, 2006,(3):315－319.

6 杜庆峰,张卫山.Oracle 的中大型应用系统性能优化分析.计算机工程, 2005,(14):91－93.