

8

第 8 章

数据库服务性能问题诊断分析

随着 IT 技术的发展，各行业业务发展需要，软件系统也随之发展得越来越复杂，数据量也越来越大，数据库作为应用系统的基础组成部分，其重要性不言而喻。

对某些企业来说，数据就是金钱，例如金融行业，一旦数据库服务器崩溃了或者数据库的性能降低了，这样直接导致系统无法正常使用，甚至导致客户的流失和利润的下降，更严重的是可能会引起不良的社会影响。像证券交易系统，一旦性能降低或者服务器崩溃，那后果是极其严重的，估计不是简单的会引起短时间股民的情绪不满那么简单。

而对于普通企业来说，如果日常的运营完全依赖于该业务系统，那么一旦该系统的数据库崩溃或性能问题严重，那么对企业造成的影响也是可想而知的。

8.1 Oracle 性能监控方法

Oracle 是目前世界上大型应用系统广泛使用的数据库，Oracle 数据库产品为财富排行榜上的前 1000 家公司所采用，许多大型网站也选用了 Oracle 系统。Oracle 内部结构比较复杂，如图 8.1 所示，出现性能问题的可能性是比较大的。因此在性能测试过程中，应该密切关注 Oracle 的性能表现，收集和监控 Oracle 性能数据。

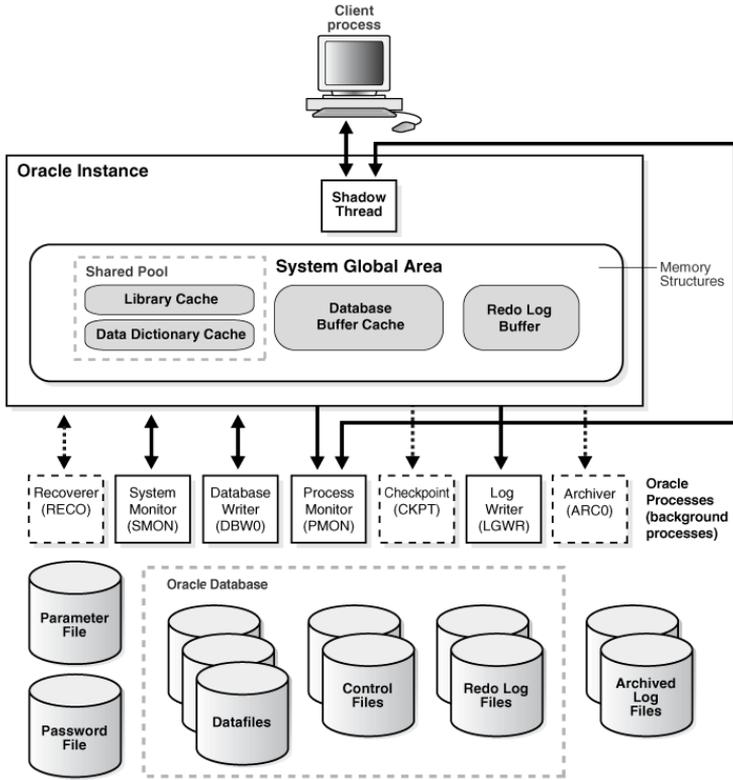


图 8.1 Oracle 内部结构工作示意图

8.1.1 在 LoadRunner 中配置监控 Oracle

要通过 LoadRunner 监控 Oracle，首先必须在 Controller 所在的机器上安装 Oracle 客户端，然后配置好服务名，用 sqlplus 确认可以连接 Oracle，之后就可以在 Controller 中配置 Oracle 连接，配置的方法如下。

- (1) 打开 Controller，选择监控图中的“Database Server Resource Graphs - Oracle”，如图 8.2 所示。
- (2) 添加监控，监控引擎选择“LoadRunner native monitors”单选按钮，如图 8.3 所示。
- (3) 然后添加需要监控的 Oracle 所在的服务器 IP 地址，如图 8.4 所示。
- (4) 输入 Oracle 服务器的登录账号，如图 8.5 所示。

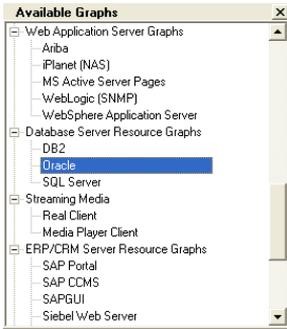


图 8.2 在 Controller 中添加 Oracle 监控

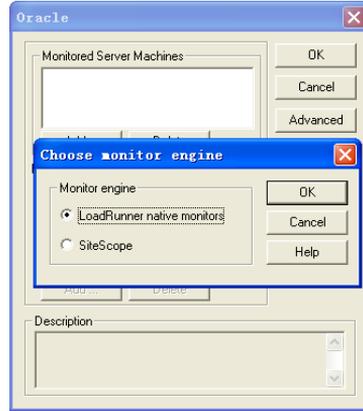


图 8.3 监控引擎选择“LoadRunner native monitors”单选按钮

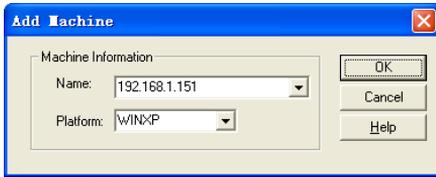


图 8.4 添加 Oracle 所在的服务器 IP 地址

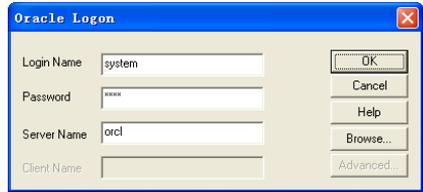


图 8.5 输入 Oracle 账号

(5) 选择需要监控的 Oracle 计数器，如图 8.6 所示。

添加完监控 IP 后如图 8.7 所示。

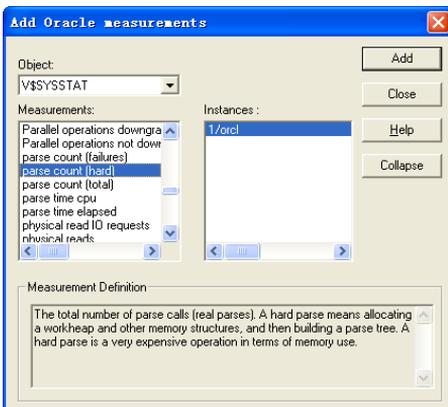


图 8.6 选择需要监控的 Oracle 计数器

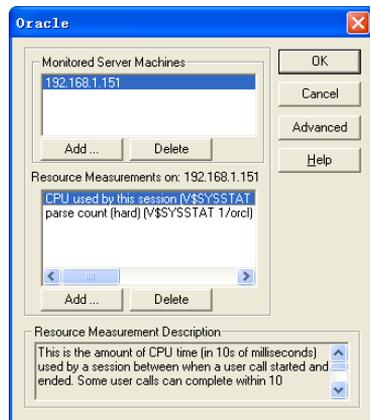


图 8.7 确认添加 Oracle 计数器

确定之后即可在 Controller 的监控图表中看到收集的 Oracle 性能数据，如图 8.8 所示。

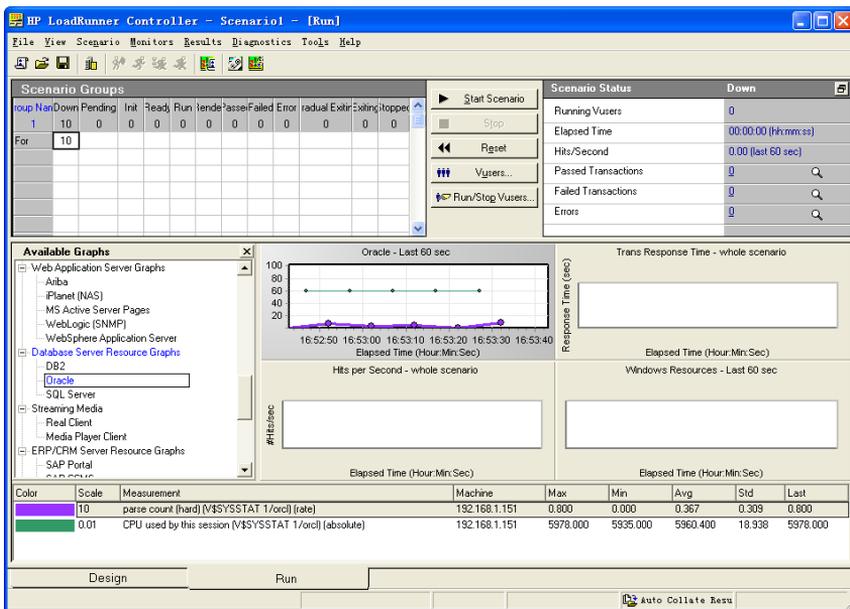


图 8.8 在 Controller 监控图中查看 Oracle 性能数据

(6) 修改 Oracle 监控数据的收集频率。

LoadRunner 默认间隔 10 秒收集一个 Oracle 性能数据，数据取样的间隔太短会对 Oracle 数据库性能造成一定的影响，如果想调整收集频率，可打开 LoadRunner 安装目录中的 dat\monitors\vmom.cfg 文件，修改其中的 sample rate。

需要注意的是 LoadRunner 限制监控 Oracle 收集性能数据的最小间隔是 10 秒，如果设置低于这个值，会按 10 秒间隔来收集。

(7) 添加自定义计数器。

LoadRunner 在 vmom.cfg 文件中设置了可收集的 Oracle 性能计数器，如果希望自己定义收集的性能数据，可修改此文件，添加相应的 SQL 语句来定义计数器。

例如，可在 LoadRunner 安装路径的\dat\monitors 找到 vmom.cfg 文件并修改如下：

```
[V$ Monitor]
Counters=150
CustomCounters=9
;How many seconds for each data sample?
SamplingRate=10
```

```
[Custom0]
;Name must be unique
Name=库快命中率
Description=该计数器返回当前库快命中率
Query=SELECT 100*((sum(pins-reloads))/sum(pins)) from v$librarycache
IsRate=0

[Custom1]
;Name must be unique
Name=高速缓存区命中率
Description=oracle database shoot straight
Query=SELECT          round(1-SUM(PHYSICAL_READS)/(SUM(DB_BLOCK_GETS) +
SUM(CONSISTENT_GETS)), 4) * 100 FROM (SELECT CASE WHEN NAME='physical reads' THEN
VALUE END PHYSICAL_READS,CASE WHEN NAME = 'db block gets' THEN VALUE END
DB_BLOCK_GETS,CASE WHEN NAME = 'consistent gets' THEN VALUE END CONSISTENT_GETS
FROM V$SYSSTAT WHERE Name IN ('physical reads','db block gets','consistent
gets'))
IsRate=0

[Custom2]
;Name must be unique
Name=共享区库缓存区命中率
Description=命中率应大于 0.99
Query=Select round(sum(pins-reloads)/sum(pins) * 100, 2) from v$librarycache
IsRate=0

[Custom3]
;Name must be unique
Name=共享区字典缓存区命中率
Description=命中率应大于 0.85
Query=Select round(sum(gets-getmisses-usage-fixed)/sum(gets) * 100, 2) from
v$rowcache
IsRate=0

[Custom4]
;Name must be unique
Name=检测回滚段的争用
Description=应该小于 1%
Query=select round(sum(waits)/sum(gets) * 100, 2) from v$rollstat
```

```
IsRate=0

[Custom5]
;Name must be unique
Name=检测回滚段收缩次数
Description=应该小于 1%
Query=select sum(shrinks) from v$rollstat, v$rollname where v$rollstat.usn
= v$rollname.usn
IsRate=0

[Custom6]
;Name must be unique
Name=监控表空间的 I/O 读总数
Description=监控表空间的 I/O
Query=select sum(f.phyrds) pyr from v$filestat f, dba_data_files df where
f.file# = df.file_id
IsRate=0

[Custom7]
;Name must be unique
Name=监控表空间的 I/O 块读总数
Description=监控表空间的 I/O
Query=select sum(f.phyblkrd) pbr from v$filestat f, dba_data_files df where
f.file# = df.file_id
IsRate=0

[Custom8]
;Name must be unique
Name=监控表空间的 I/O 写总数
Description=监控表空间的 I/O
Query=select sum(f.phywrt) pyw from v$filestat f, dba_data_files df where
f.file# = df.file_id
IsRate=0
```

8.1.2 使用 SiteScope 监控 Oracle

LoadRunner 除了可以用本地监控器来收集 Oracle 的性能数据外，还可以整合 SiteScope 对 Oracle 进行监控。

根据 SiteScope 帮助文档，需要把 oracle 的 jdbc 驱动 classes12.zip 放到 <SiteScope root directory>\Web-INF\lib 和 <SiteScope root directory>\java\lib\ext，之后重启 SiteScope 才能对 Oracle 进行监控。添加监控器时，选择“数据库计数器”选项，如图 8.9 所示，然后配置 Oracle 的 JDBC 连接即可。



图 8.9 在 SiteScope 中添加 Oracle 数据库监控

另外需注意的是：连接 Oracle 的数据库用户必须是 DBA 权限的用户。

8.1.3 使用 Oracle 企业管理器查看数据库性能

Oracle 自带的企业管理器中包含了数据库性能监控的功能，安装好 Oracle 后，可打开 IE 浏览器，输入“http://<IP 地址>:5500/em”访问企业管理器，用 SYS 用户登录后，打开“性能”进行监控，如图 8.10 所示。



注意

如果启动不了企业管理器，有可能是企业管理器的服务未启动，可先启动：

```
emctl start dbconsole
```



图 8.10 Oracle 企业管理器性能监控页面

另外，由于 Oracle 的 V\$视图存储了 Oracle 数据库的核心信息，包括 Oracle 数据库的性能数据，因此可以通过 sqlplus 等工具连接数据库查询这些性能数据来进行 Oracle 的性能分析。例如：查询 v\$sysstat 视图（如下程序清单所示）可以查看从内存中读取数据的频率。它提供了数据库中设置的数据块缓存区的命中率。这个信息可以帮助我们判断系统是否需要更多的数据缓存（DB_CACHE_SIZE），或者系统的状态是否调整得不佳（二者均将导致较低的命中率）。通常情况下，应当确保读数据的命中率保持在 95%以上。

```
select 1-(sum(decode(name, 'physical reads', value,0))/
(sum(decode(name, 'db block gets', value,0) +
(sum(decode(name, 'consistent gets', value,0))))
"Read Hit Ratio"
from v$sysstat;
```

8.1.4 使用 SpotLight 监控数据库性能

SpotLight On Oracle 是由 Quest 公司出品的一款针对 Oracle 进行监控的软件。SpotLight 监控 Oracle 的基本原理与 LoadRunner 监控类似，通过获取 Oracle 的数据字典和动态性能视图，然后把性能数据按直观的方式展现出来，如图 8.11 所示。



图 8.11 Spotlight On Oracle 监控数据库

下面简要介绍使用 Spotlight 对 Oracle 进行监控的过程。

1. 建立 Oracle 连接

第一步要建立 Connection，如图 8.12 所示，这样才能够使用 Spotlight 连接到要监测的数据库。

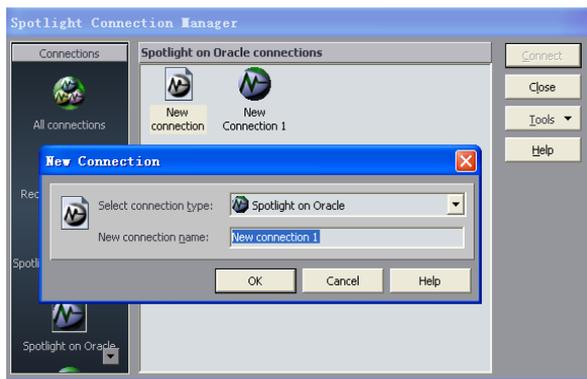


图 8.12 建立 Oracle 连接

新建连接，然后输入 Oracle 连接用户账号，确定之后即可进入监控主页面。

2. 查看系统主界面进行 Oracle 监控

系统主界面反映了系统的整体运行情况，如果系统哪方面出现问题，会报相应的警告，最严重为红色警告。然后根据警告可转到相应的子窗口，查看相应的情况。下面介绍各子窗口。

1) Sessions 面板

- **Response:** 系统的响应时间。
- **Total Users:** 总的用户 Session 数量。
- **Active Users:** 当前正在执行的用户 Session 数量。

2) Host 面板

Host 面板主要显示 CPU 利用率和内存使用情况。

3) Server Processes 面板

Server Processes 面板主要显示服务器进程的信息。主要关注以下几点。

- **PGA Target/Used:** PGA 目标总数及当前使用数。
- **Dedicated:** 专用服务器进程的个数。
- **Shared:** 共享服务器进程的个数。
- **Job Queue:** 作业进程的个数。

4) SGA 面板

SGA 面板主要显示 SGA 中各组件的内存使用情况，主要关注以下几点。

- **CurrentSize:** 当前 SGA 的使用量。
- **BufferCache:** 数据缓冲区的内存情况。
- **SharedPool:** 共享池的使用情况。

5) 后台进程面板

后台进程面板主要显示与磁盘 I/O 相关的后台进程，包括 DBWR、LGWR、ARCH。

6) 磁盘存储面板

磁盘存储面板主要显示数据库文件的情况（控制文件除外），包括以下几点。

- **DatabaseFiles:** 数据文件使用情况。
- **Redo Logs:** 联机日志文件情况，包括组数及大小。

- Archive Log: 归档日志情况。

3. Top Sessions 的监控

主面板用一个宏观的视图展现了 Oracle 的全貌，另外，还可以打开其他视图查看 Oracle 的性能表现，例如 Top Sessions 面板。

通过 Top Sessions 面板可以查看当前哪个 Session 占用了大量的资源，以此定位数据库问题。这是主从式的面板，单击上部列表，会在 Session Information 中显示该会话的所有信息，如图 8.13 所示。

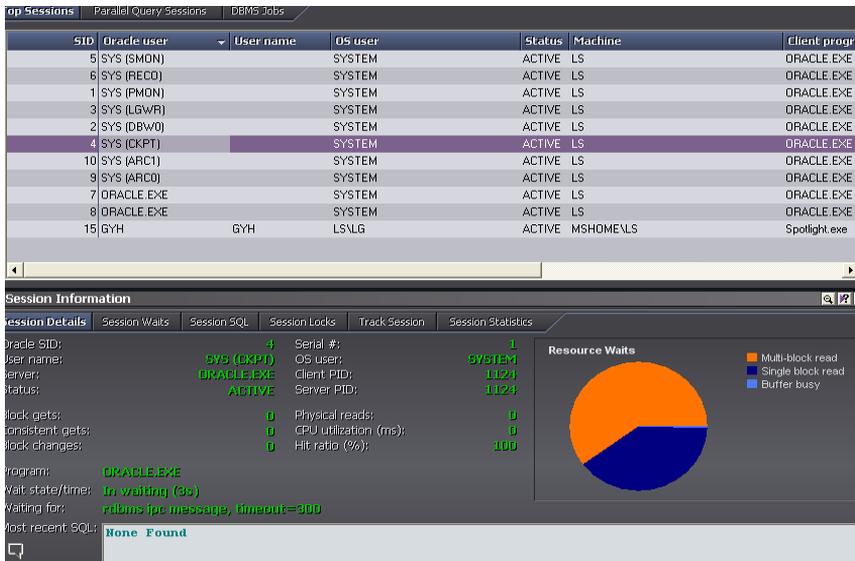


图 8.13 在 Session Information 中显示会话信息

其中，Most recent SQL 可以用来确定当前占用资源最多的 SQL 语句。切换到 Session waits 选项卡可以找出与该 Session 相关的等待事件是什么；切换到 Session locks 页可以看到 Session 相关的锁信息。

4. Top SQL 视图

在 Top SQL 视图中，可根据条件来筛选查看 LibraryCache 中相应的 SQL 语句，如图 8.14 所示，可以协助我们找出对性能影响比较大的 SQL 语句。

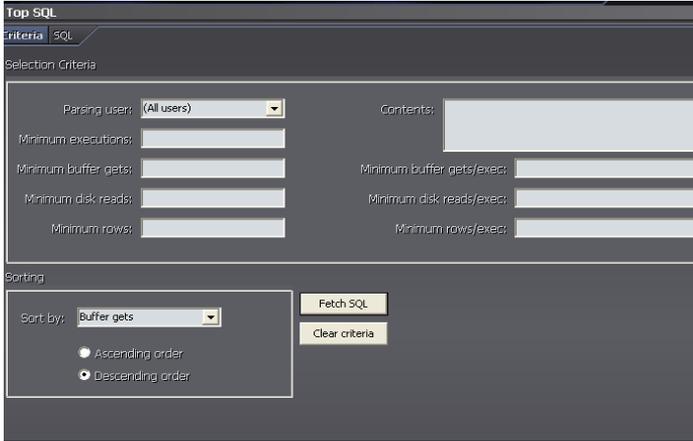


图 8.14 Top SQL 视图

5. Activity 视图

Activity 视图主要提供了等待事件，锁等待，闫锁等待，当前事务等，如图 8.15 所示。

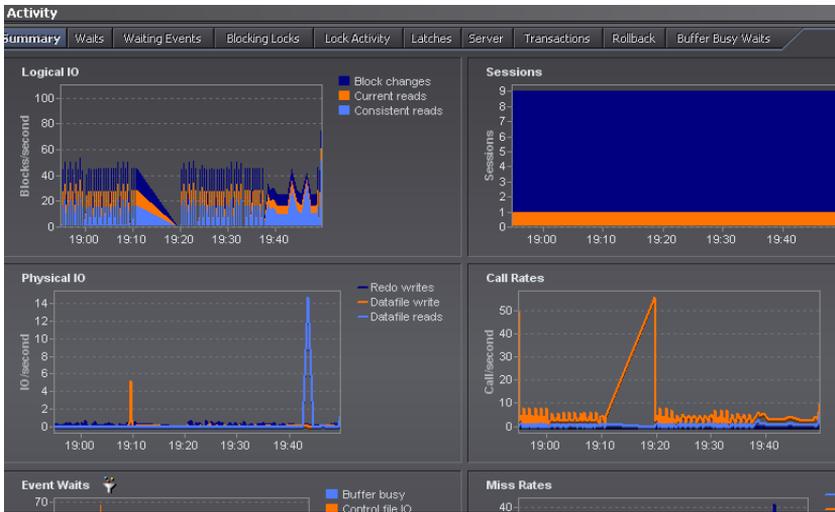


图 8.15 Activity 视图

6. Configuration & Memory 视图

Configuration & Memory 视图主要显示 SGA 的使用情况及参数配置情况，如图 8.16 所示，可修改调整相应的参数。

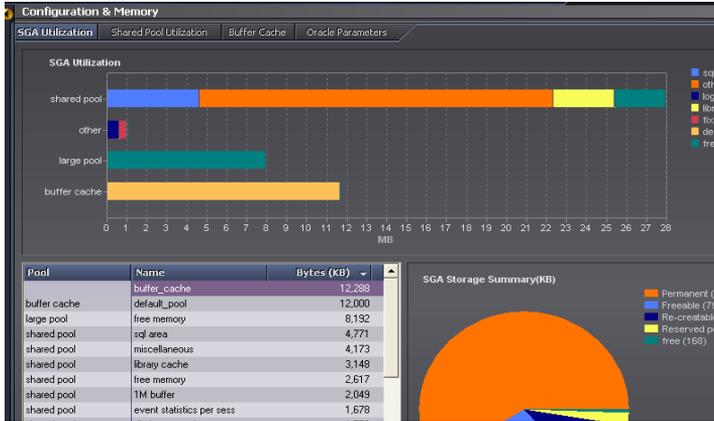


图 8.16 Configuration & Memory 视图

7. OS 视图

OS 视图主要展现的是 Oracle 服务器的操作系统性能情况，包括磁盘利用率和进程情况等，如图 8.17 所示。

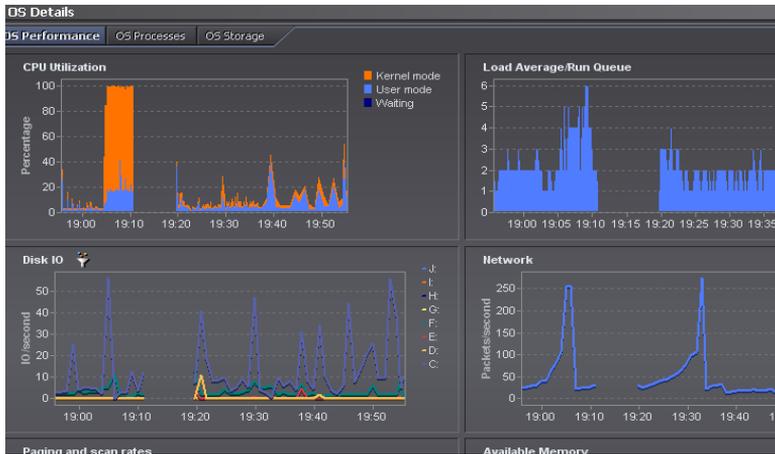


图 8.17 OS 视图

8. Disk Storage

Disk Storage 视图主要显示磁盘存储情况，包括 Oracle 表空间利用率和在线日志的存储空间使用情况，如图 8.18 所示。

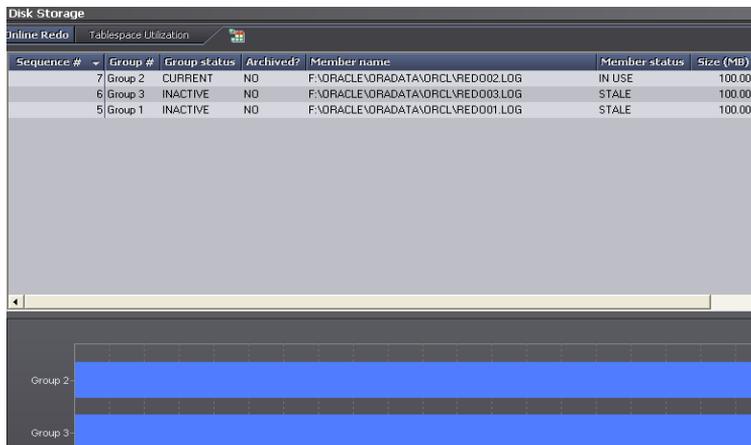


图 8.18 Disk Storage 视图

8.2 Oracle 性能分析与诊断

8.2.1 排序对 Oracle 性能的影响

Oracle 的 v\$sysstat 视图存储了排序的信息，包括磁盘排序和内存排序：

(1) sorts(memory) 是在 SORT_AREA_SIZE 中的排序操作的数量（由于是在 SORT_AREA_SIZE 中，因此不需要在磁盘进行排序）。

(2) sorts(disk) 则是由于排序所需空间太大，SORT_AREA_SIZE 不能满足而不得不在磁盘进行排序操作的数量。

这两项统计通常用于计算 In-memory sort ratio。In-memory sort ratio 表示内存中完成的排序所占的比例。理想状态下，在 OLTP 系统中，大部分排序不仅小并且能够完全在内存里完成排序。In-memory sort ratio 的计算公式如下：

$$\text{sorts (memory)} / (\text{sorts (memory)} + \text{sorts (disk)})$$

可通过执行以下 SQL 语句计算得出 In-memory sort ratio：

```
select a.value/(b.value+c.value)
from v$sysstat a,v$sysstat b,v$sysstat c
where a.name='sorts (memory)' and
      b.name='sorts (memory)' and c.name='sorts (disk)';
```

性能好的话，应该是大部分排序在内存中进行。对于要做大量排序操作的 SQL 语句的执行（如 `select * from tt order by 1,2,3,4;`），可监控到 `sort(disk)` 和 `sort(memory)` 都会有所上升。读者可在 Oracle 中做如下实验来进行验证。

(1) 创建表：

```
create table tt as select * from all_objects;
commit;
```

(2) 打开时间统计开关：

```
Set timing on;
```

(3) 执行 SQL 语句，监控 In-memory sort ratio：

```
select * from tt order by 1,2,3,4;
```

由于该 SQL 语句将执行大量的排序操作，因此可看到 `sort (disk)` 和 `sort (memory)` 的上升，如果 Oracle 的 `SORT_AREA_SIZE` 参数设置比较小，则大量排序被放到磁盘中进行，导致执行效率低下。

`Sort (disk) / Sort (memory)` 应该保持小于 5% 的状态，如果超过 5%，就应该考虑增加 `sort_area_size` 的值（通过调整 PGA 来完成）。下面是用于诊断的 SQL 语句：

```
SELECT disk.Value disk,mem.Value mem,(disk.Value/mem.Value)*100 ratio
      FROM v$sysstat disk,v$sysstat mem
      WHERE mem.NAME='sorts (memory)' AND disk.NAME='sorts (disk)';
```

8.2.2 Buffer Cache Hit Ratio 诊断

Oracle 查询数据时可分 3 种情况读取数据：`db block gets`、`consistent gets`、`physical reads`，它们三者之间的关系大致可概括为：逻辑读（`logical reads`）指的是 Oracle 从内存读到的数据块数量。一般来说是 '`consistent gets`' + '`db block gets`'。当在内存中找不到所需的数据块的话就需要从磁盘中获取，于是就产生了 '`physical reads`'。

产生 `Physical reads`（物理读）的主要原因如下。

- 在数据库高速缓存中不存在这些块。
- 全表扫描。
- 磁盘排序。

要想查询速度快，应该尽量避免 `Physical reads`，尽量从缓存读取数据。可通过执行以下 SQL 语句来查看 Oracle 缓存的命中率：

```

select 1 - ((physical.value - direct.value - lobs.value) / logical.value)
"Buffer Cache Hit Ratio"
from v$sysstat physical,v$sysstat direct,v$sysstat lobs,v$sysstat logical
where physical.name = 'physical reads'
and direct.name='physical reads direct'
and lobs.name='physical reads direct (lob)'
and logical.name='session logical reads';
    
```

该项显示 buffer cache 大小是否合适，通常在 OLTP 系统中，这个值应该大于 90%，否则需要考虑增加 buffer cache 的大小。

在考虑调整 buffer cache hit ratio 时，需要注意：如果上次增加 buffer cache 的大小以后，没有对提高 hit ratio 产生很大效果的话，不要盲目增加 buffer cache 的大小以提高性能。因为对于排序操作或并行读，Oracle 是绕过 buffer cache 进行的。在调整 buffer cache 时，尽量避免增加很多的内存而只是提高少量 hit ratio 的情况出现。

可通过执行以下语句查看 Oracle 的 buffer cache size:

```
show parameter _size
```

8.2.3 优化 Oracle 软解析率

通常，Oracle 对 SQL 语句的处理过程是这样的：当发送一条 SQL 语句到 Oracle 时，在执行和获取数据之前，Oracle 会对此 SQL 语句进行几个步骤的处理。

(1) 语法检查 (syntax check): 检查此 SQL 的拼写是否语法。

(2) 语义检查 (semantic check): 如检查 SQL 语句中的访问对象是否存在，以及该用户是否具备相应的权限。

(3) 对 SQL 语句进行解析 (parse): 利用内部算法对 SQL 语句进行解析，生成解析树 (parse tree) 及执行计划 (execution plan)。

(4) 执行 SQL，返回结果 (execute and return)。

其中，软、硬解析就发生在第三步中。Oracle 利用内部的 HASH 算法来取得该 SQL 语句的 HASH 值，然后在 Library Cache 里查找是否存在该 HASH 值，如果存在，则将此 SQL 语句与 cache 中的 SQL 语句进行比较；如果两个 SQL 语句“相同”，就会利用已有的解析树与执行计划，而省略了优化器的相关工作。这就是软解析的过程。

如果上面的 2 个条件中任有一个不成立，那么优化器都将进行创建解析树、生成执行计划的动作。这个过程就叫硬解析。

创建解析树、生成执行计划对于 SQL 的执行来说是开销比较大的工作，因为它需要 Oracle 在 Shared Pool 中重新分配内存，然后再确定执行计划，最终 SQL 语句才会被执行；所以，应当尽量避免硬解析，尽量使用软解析。在项目开发中，倡导开发人员对功能相同的 SQL 代码要努力保持代码的一致性，在程序中多使用绑定变量，以避免硬解析的发生。

可通过以下 SQL 语句查询 Oracle 的软解析率，软解析率 = $1 - (\text{parse count (hard)} / \text{parse count (total)})$):

```
select 1 - (a.value/b.value)
  from v$sysstat a,v$sysstat b
  where a.name='parse count (hard)'
        and b.name='parse count (total)';
```

如果软解析率小于 0.2，则表示硬解析率太高，不过，如果总解析量 (parse count total) 偏低，这项值可以被忽略。

软解析率偏低通常是由于存在较多不能被共享利用 (重用) 的 SQL 语句导致的，因此，可以通过查询 v\$sqlarea 视图来找出 Library Cache 中执行次数偏低的 SQL 语句，从而分析这些 SQL 语句为什么不能被重用，查询 v\$sqlarea 视图的 SQL 语句如下所示：

```
SELECT sql_text
  FROM v$sqlarea
  WHERE executions < 5
  ORDER BY UPPER(sql_text);
```

另外，也可以查找 SQL 执行次数和 SQL 解析次数 (hard parse)，然后对比两个值的差，查询 v\$sqlarea 的语句可以这样写：

```
SELECT sql_text , parse_calls , executions
  FROM v$sqlarea
  ORDER BY parse_calls;
```

还可以通过查询 v\$librarycache 视图的 Reloads 值 (reparsing 的值) 来进行诊断，该值应该接近 0，invalidations 的值也应该接近 0，否则应该考虑调整 shared pool size，通过调整 Shared Pool 来调整 Library Cache。查询 v\$librarycache 视图的 SQL 语句如下所示：

```
select namespace, gethitratio, pinhitratio, reloads, invalidations
  from v$librarycache;
```

检查 v\$librarycache 中 sql area 的 gethitratio 是否超过 90%，如果未超过 90%，则应该检查应用代码：

```
Select gethitratio
  from v$librarycache
  where namespace='SQL AREA';
```

v\$sqllibrarycache 中 reloads/pins 的比率应该小于 1%，如果大于 1%，应该考虑增加参数 shared_pool_size 的值：

```
select sum(pins) "executions", sum(reloads) "cache
misses",sum(reloads)/sum(pins)
from v$sqllibrarycache;
```

reloads/pins>1%有两种可能，一种是 library cache 空间不足，另一种是 SQL 语句中引用的对象不合法。如果知道具体某个连接的 Session，则可以直接查看某个 Session 的 hard parse 个数：

```
select a.sid,a.value
      from v$sesstat a,v$session b ,v$statname c
      where a.sid=b.sid
            and a.statistic#=c.statistic#
            and a.sid = 137
            and c.name='parse count (hard)';
```

如果确认是由于 Library Cache 空间不足导致的软解析率低，则可以通过调整 Library Cache 来进行优化，需要通过调整 Shared Pool 来调整 Library Cache：

```
SELECT shared_pool_size_for_estimate AS pool_size,estd_lc_size,estd_lc_time_
saved
FROM v$shared_pool_advice;
```

根据 shared_pool_advice 的提示来进行调整，其中 ESTD_LC_SIZE 是估计 Library Cache 的大小值，ESTD_LC_TIME_SAVED 是在当前指定共享池的大小中找到库缓存对象所节省的时间（秒）。

如果不是 Library Cache 空间的问题，则需要考虑对 SQL 程序进行调优，下面列举两个调优的例子。

1. 书写程序时尽量使用变量，不要过多的使用常量

如果编写 SQL 代码时没有使用绑定变量，则可能造成硬解析率偏高，从而影响 SQL 语句的执行效率，读者可做以下实验来进行验证。

(1) 创建表格：

```
SQL>CREATE TABLE m(x int);
```

(2) 创建存储过程 proc1，使用绑定变量：

```
SQL>CREATE OR REPLACE PROCEDURE proc1
AS
BEGIN
  FOR i IN 1..10000
  LOOP
```

```
Execute immediate
  'INSERT INTO m VALUES (:x)' USING i;
END LOOP;
END;
/
```

(3) 创建存储过程 **proc2**，不使用绑定变量：

```
SQL>CREATE OR REPLACE PROCEDURE proc2
AS
BEGIN
  FOR i IN 1..10000
  LOOP
Execute immediate
  'INSERT INTO m VALUES ('||i||')' ;
END LOOP;
END;
/
```

(4) 执行 **proc2** 和 **proc1**，对比执行效率。

打开计时开关：

```
SQL>SET TIMING ON
执行 proc2:
SQL> exec proc2;
PL/SQL procedure successfully completed.
Elapsed: 00:00:08.93
```

清空数据，再执行 **proc1**：

```
SQL> TRUNCATE TABLE m;
Table truncated.
Elapsed: 00:00:01.76
```

```
SQL> exec proc1;
PL/SQL procedure successfully completed.
Elapsed: 00:00:01.85
```

可看到使用绑定变量的 **proc1** 执行效率要比不使用变量的 **proc2** 效率高。

2. 修改 `cursor_sharing` 参数为 `similar`，让类似的 SQL 语句不做 `hard parse`

有时候我们的应用程序没有使用绑定变量，而修改程序可能有点困难，我们可能需要设置 `cursor_sharing=similar` 来强制 ORACLE 使用绑定变量。可用以下语句查看当前 Oracle 采用什么

样的方式处理共享游标:

```
Show parameter cursor
```

设置 `cursor_sharing=similar` 来强制 ORACLE 使用绑定变量:

```
Alter system set cursor_sharing=similar
```

8.2.4 诊断 Oracle 的 CPU 使用率

从 `v$sysstat` 视图中可以找出 Session 使用 CPU 的情况, 例如, 可以统计 `Parse CPU to total CPU ratio`: 该项显示总的 CPU 花费在执行及解析上的比率。如果这项比率较低, 说明系统执行了太多的解析。

总的 CPU 花费在执行及解析上的比率 = $1 - (\text{parse time cpu} / \text{CPU used by this session})$, 可通过以下 SQL 语句统计出该项数据。

```
select 1-(a.value/b.value)
from v$sysstat a,v$sysstat b
where a.name='parse time cpu' and
      b.name='CPU used by this session';
```

如果发现 CPU 使用比较多, 可用以下语句进一步找出使用 CPU 多的用户 Session:

```
select          a.sid,spid,status,substr(a.program,1,40)
prog,a.terminal,osuser,value/60/100 value
from v$session a,v$process b,v$sesstat c
where c.statistic#=12
      and c.sid=a.sid
      and a.paddr=b.addr
      order by value desc;
```



注意

SQL 语句中的 12 是指 `cpu used by this session`。

再进一步找出使用 CPU 多的 SQL 语句, 可查找指定 SPID 正在执行的 SQL 语句:

```
SELECT P.pid pid,S.sid sid,P.spid spid,S.username username,
S.osuser osname,P.serial# S_#,P.terminal,P.program
program,P.background,S.status,RTRIM(SUBSTR(a.sql_text, 1, 80)) SQL
FROM v$process P, v$session S,v$sqlarea A
WHERE P.addr = s.paddr
      AND S.sql_address = a.address (+)
      AND P.spid LIKE '%&1%';
```

注意 在 Linux 环境中可以通过 ps 查看进程信息，包括 pid；而 Windows 中任务管理器的 PID 与 Oracle 的 v\$process 中的 pid 不能一一对应。Windows 是多线程服务器，每个进程包含一系列线程，这点与 UNIX 等不同，UNIX 每个 Oracle 进程独立存在，在 NT 上所有线程由 Oracle 进程衍生。

也可以指定 SID 查看正在执行的 SQL 语句：

```
SELECT P.pid pid,S.sid sid,P.spid spid,S.username username,
S.osuser osname,P.serial# S_#,P.terminal,P.program
program,P.background,S.status,RTRIM(SUBSTR(a.sql_text, 1, 80)) SQL
FROM v$process P, v$session S,v$sqlarea A
WHERE P.addr = s.paddr
AND S.sql_address = a.address (+)
AND s.sid = '136';
```

8.2.5 跟踪诊断和优化 SQL 语句

SQL 语句的执行效率对于数据库性能的影响非常重大，在性能测试过程中，需要查找和定位可能存在性能瓶颈的 SQL 语句。

PL/SQL Developer 是一个 Oracle 的 PL/SQL 开发工具，同时具备 Oracle 的一些管理功能，可利用它来跟踪客户端向 Oracle 服务器提交的所有 SQL 语句，方法是选择 Session→SQL Trace 命令，如图 8.19 所示。

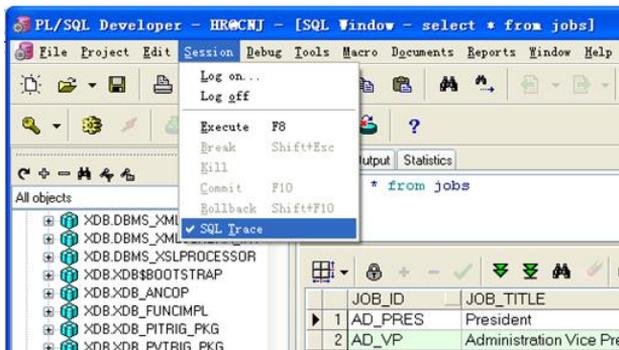


图 8.19 使用 PL/SQL Developer 跟踪 SQL 语句

也可以指定跟踪某个 Session 的 SQL 语句，如图 8.20 所示。

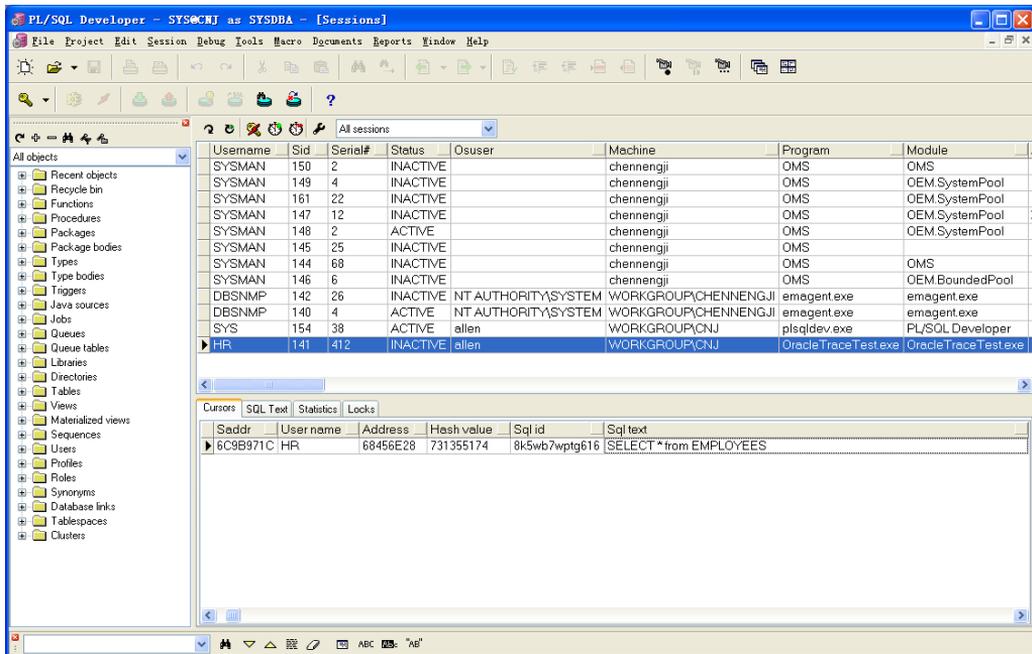


图 8.20 指定跟踪某个 Session 的 SQL 语句

如果不用 PL/SQL Developer 这类工具，也可以通过修改 Oracle 全局参数 `sql_trace`，来产生 Trace 文件，之后再分析 Trace 文件来分析提交到 Oracle 的 SQL 语句。如果想针对某个 Session 进行 SQL 语句的跟踪，则可以采用以下办法（以下操作需要在 SYS 用户下进行）。

(1) 首先使用以下语句从 Oracle 的 `v$session` 表查出所有跟 Oracle 连接的客户端程序的进程：

```
select SID, SERIAL#, USERNAME, OSUSER, MACHINE, TERMINAL, PROGRAM from v$session
```

(2) 然后利用 `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` 包来跟踪特定的进程向 Oracle 提交的 SQL 语句。

例如，下面语句为 SID 为 13，SERIAL# 为 96 的进程执行 SQL 语句的跟踪功能：

```
execute dbms_system.set_sql_trace_in_session(13,96,TRUE);
```

(3) 如果要停止跟踪，则提交以下语句：

```
execute dbms_system.set_sql_trace_in_session(13,96,FALSE);
```

然后在利用 Oracle 的 `tkprof` 工具对记录下来的 SQL 跟踪文件进行分析，如图 8.21 所示。

```

Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\allen>tkprof
Usage: tkprof tracefile outputfile [explain= ] [table= ]
      [print= ] [insert= ] [sys= ] [sort= ]
table=schema.tablename Use 'schema.tablename' with 'explain=' option.
explain=user/password Connect to ORACLE and issue EXPLAIN PLAN.
print=integer List only the first 'integer' SQL statements.
aggregate=yes|no
insert=filename List SQL statements and data inside INSERT statements.
sys=no TKPROF does not list SQL statements run as user SYS.
record=filename Record non-recursive statements found in the trace file.
waits=yes|no Record summary for any wait events found in the trace file.
sort=option Set of zero or more of the following sort options:
  pscnt number of times parse was called
  pscpu cpu time parsing
  psela elapsed time parsing
  prdsk number of disk reads during parse
  prqry number of buffers for consistent read during parse
  prscu number of buffers for current read during parse
  prsmis number of misses in library cache during parse
  exectn number of execute was called
  execpu cpu time spent executing
  exeela elapsed time executing
  exedsk number of disk reads during execute
  exeqry number of buffers for consistent read during execute
  execu number of buffers for current read during execute
  exerow number of rows processed during execute
  exemis number of library cache misses during execute
  fchcnt number of times fetch was called
  fchcpu cpu time spent fetching
  fchela elapsed time fetching
  fchdsk number of disk reads during fetch
  fchqry number of buffers for consistent read during fetch
  fchcu number of buffers for current read during fetch
  fchrow number of rows fetched
  userid userid of user that parsed the cursor
    
```

图 8.21 利用 tkprof 对 SQL 跟踪文件进行分析

另外，也可以利用前面讲过的 Spotlight 这个工具进行 SQL 跟踪，在 Spotlight 中跟踪指定 Session 的 SQL 语句，并查看执行计划，如图 8.22 所示。

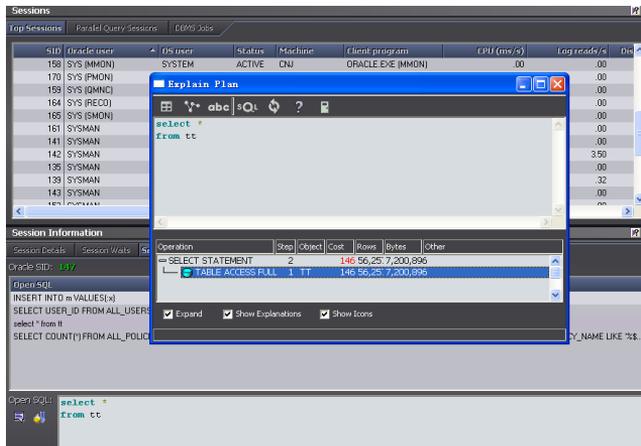


图 8.22 在 Spotlight 中跟踪指定 Session 的 SQL 语句

还可以利用 SQL Tuning 的调优建议功能对 SQL 语句进行分析，SQL Tuning 将自动查找更优的可替代 SQL 语句，如图 8.23 所示。

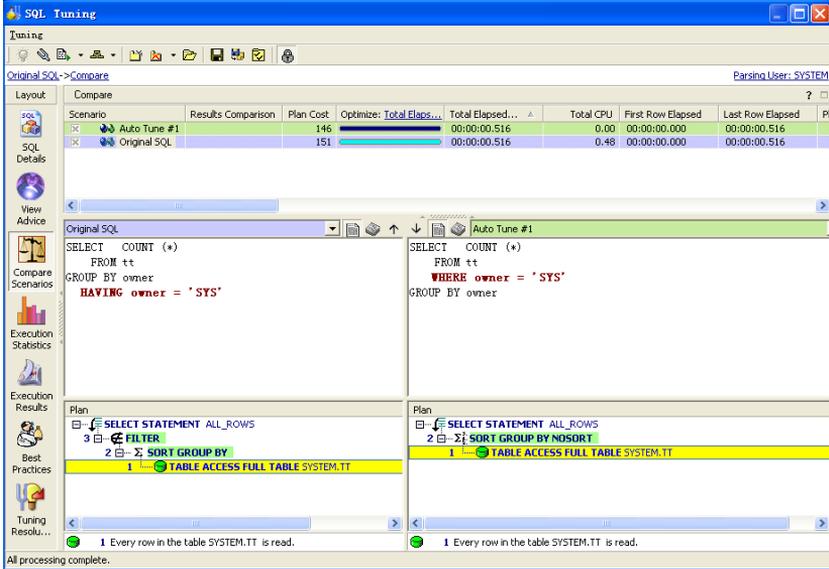


图 8.23 使用 SQL Tuning 优化 SQL 语句

图 8.23 中，SQL Tuning 把原来使用 Having 进行条件过滤的 SQL 语句改写为使用 Where 进行条件过滤，这样的 SQL 语句执行效率更高。

8.2.6 Oracle 索引问题诊断与优化

数据库表设计不恰当往往是数据库性能表现低下的主要原因，其中索引设计不大是常见的问题。在进行数据库性能问题诊断时，应该重点关注索引的设计，以及 SQL 语句的写法对索引的利用是否恰当。

在适当的表字段建立索引，能有效加快查询速度，例如创建两个表：

```
create table s1 as select * from SH.SALES;
create table s2 as select * from SH.SALES;
```

其中，s1 表没有建立索引，s2 表有建立索引。分别测试两个表的查询速度：

```
set timing on;
select * from s1 where prod_id=1;
2.45s
select * from s2 where prod_id=1;
0.59s
```

有建立索引的表比未建立索引的表查询速度要快很多，可见索引对于表查询速度的重要性。

另外，即使创建了索引，如果 SQL 查询语句写法有问题，也会造成索引不能被利用的情况，例如，在 Where 关键字后应该尽量避免使用函数，否则将抑制索引的使用，例如下面 SQL 语句：

```
SELECT * FROM dwtable2 WHERE to_number(empno)=783;
Elapsed: 00:00:00.15
```

这个 SQL 语句在 Where 关键字后使用了不必要的 to_number 函数，导致 empno 这个字段上的索引不能被查询利用，这个 SQL 语句可以改写成更效率的写法：

```
SELECT * FROM dwtable2 WHERE empno=783;
Elapsed: 00:00:00.01
```

为了诊断数据库的索引设计，可利用下面的 SQL 语句来查看 Oracle 数据库 index 信息：

```
SELECT  A.OWNER, A.TABLE_OWNER, A.TABLE_NAME, A.INDEX_NAME, A.INDEX_TYPE,
        B.COLUMN_POSITION, B.COLUMN_NAME, C.TABLESPACE_NAME,
        A.TABLESPACE_NAME, A.UNIQUENESS
FROM DBA_INDEXES A, DBA_IND_COLUMNS B, DBA_TABLES C
WHERE A.OWNER = UPPER ('hr')
AND A.OWNER = B.INDEX_OWNER
      AND A.OWNER = C.OWNER
      AND A.TABLE_NAME LIKE UPPER ('DEPARTMENTS')
      AND A.TABLE_NAME = B.TABLE_NAME
      AND A.TABLE_NAME = C.TABLE_NAME
      AND A.INDEX_NAME = B.INDEX_NAME
ORDER BY  A.OWNER,      A.TABLE_OWNER,      A.TABLE_NAME,      A.INDEX_NAME,
        B.COLUMN_POSITION
```

还可以用下面的 SQL 语句直接查出某个库中没有建立 index 的表，分析是否有必要补充建立索引：

```
SELECT OWNER, TABLE_NAME
      FROM ALL_TABLES
      WHERE OWNER NOT IN ('SYS','SYSTEM','OUTLN','DBSNMP')
      AND OWNER = UPPER ('scott')
MINUS
SELECT OWNER, TABLE_NAME
      FROM ALL_INDEXES
      WHERE OWNER NOT IN ('SYS','SYSTEM','OUTLN','DBSNMP')
```

一个表可以有几百个索引，但是对于频繁插入和更新表，索引越多系统 CPU，I/O 负担就越重；建议每张表不超过 5 个索引。可用以下 SQL 语句查出建立了过量 index 的表：

```
SELECT  OWNER, TABLE_NAME, COUNT (*) "count"
      FROM ALL_INDEXES
```

```

WHERE OWNER NOT IN ('SYS','SYSTEM','OUTLN','DBSNMP')
      AND OWNER = UPPER ('hr')
      GROUP BY OWNER, TABLE_NAME
      HAVING COUNT (*) > ('4')
    
```

为了验证过量索引对性能的影响，读者可进行如下实验。

(1) 创建两个表：

```

create table table1 as select * from SH.SALES;
create table table2 as select * from SH.SALES;
    
```

其中，table1 只在 prod_id 列建索引，table2 在所有列建索引。

(2) 分别更新两个表的相同 prod_id 的数据，共 29282 条数据：

```

SELECT count(*) FROM table1 where prod_id=30;
29282
    
```

(3) 对比更新的速度：

```

set timing on;
update table1 set cust_id=1 where prod_id=30;
10.56s
update table2 set cust_id=1 where prod_id=30;
11.35s
    
```

可见过量的索引对于做更新操作的 SQL 语句而言，会造成执行效率下降的情况，如果要更新的数据量比较大的话，效率的影响会更加明显。

对于一个 SQL 语句的执行，如果没有可利用的索引，Oracle 将进行全表扫描，这将对性能造成比较大的影响，尤其是大数据量的全表扫描，效率是非常低下的，因此在性能测试和诊断分析过程中，寻找发生了全表扫描的 Sid 和 SQL 就尤为关键，下面的 SQL 语句可以从 v\$sysstat 视图中找出有多少全表扫描在进行：

```

SELECT name, value
      FROM v$sysstat
      WHERE name LIKE '%table %'
      ORDER BY name;
    
```

下面的 SQL 语句将找出正在做全表扫描的 Session：

```

SELECT      ss.username
           || '('
           || se.sid
           || ') ' "User Process",
SUM (DECODE (NAME, 'table scans (short tables)', VALUE)) "Short Scans",
    
```

```
SUM (DECODE (NAME, 'table scans (long tables)', VALUE)) "Long Scans",
SUM (DECODE (NAME, 'table scan rows gotten', VALUE)) "Rows Retrieved"
  FROM v$session ss, v$sesstat se, v$statname sn
  WHERE se.statistic# = sn.statistic#
        AND ( NAME LIKE '%table scans (short tables)%'
              OR NAME LIKE '%table scans (long tables)%'
              OR NAME LIKE '%table scan rows gotten%'
        )
  AND se.sid = ss.sid
  AND ss.username IS NOT NULL
GROUP BY ss.username
|| '('
|| se.sid
|| ') ';
```