第 1 章　Eygle 的 DBA 工作手记

写下这一章节的目的源于很多朋友的建议，他们建议我描述一下 DBA 的日常工作，一方面可以将真实的 DBA 生活展现给那些将要入行的朋友看，另一方面又可以将工作中遇到的问题真实地描写出来。前者对初学者有益，后者对大家有参考价值。于是有了这一章 DBA 手记，内容，也许有些断断续续、只言片语，但是这就是我们在一直面对的工作与持续的思考。

1.1　DBA 2.0 的时代

在 2008~2009 年，随着 Oracle Database 10g 的成熟与广泛应用，以及 Oracle Database 11g 的改进与推行，Oracle 公司开始对 DBA（即 Database Administrator，数据库管理员）这个词进行了重新界定，进一步推出了 DBA 2.0 的概念。

当然 DBA 2.0 不仅仅是一个概念，更是对我们一直以来进行的长期思考的一个阶段性总结和升华。那么什么是 DBA 2.0 呢？

回忆起来，DBA 这个职业从诞生、发展到成熟，其实时间是非常短的，记得 2000 年左右，DBA 的从业人群还非常小，而到了 2008、2009 年，DBA 的圈子已经越来越大，甚至传统意义上的 DBA 已经成熟得需要革新。这个行业的发展和变化如此之快，我们甚至举办过一个系列的高校巡回演讲活动，主题是如何成为一个 Oracle DBA，类似的很多活动已经进一步将 DBA 这个词引入校园（Oracle 公司已经在面向高校推进 OCP 认证），现在的学生能够接触到 DBA 这个概念的时间早得超乎我们当年的想象（很多人一毕业就可以加入到 DBA 行列）。

很多业界朋友都问过自己这样的问题，在数据库软件的自动化程度越来越高，应用越来越普及和简单之后，DBA 当何以为生？实际上这也正是 DBA 2.0 时代我们要面对的问题。

说起来，DBA 2.0 时代，直接同 Oracle Database 10g 引入的一个新产品表象相关，这个产品就是 Grid / Database Control，这个工具将原来基于客户端的 OEM 通过 Web 形式来展现，并且基于后台众多新特性的支持，提供了强大的功能。

通过这个工具，以前要用 SQL 工具来追踪的 SQL 问题、性能问题等，现在使用新版的 Database Control 就可以通过 Web 页面清晰快速地展现和定位。

如图 1-1 的 Oracle Database Control 主页面清晰地展示了系统资源的使用情况及诊断概要信息等。

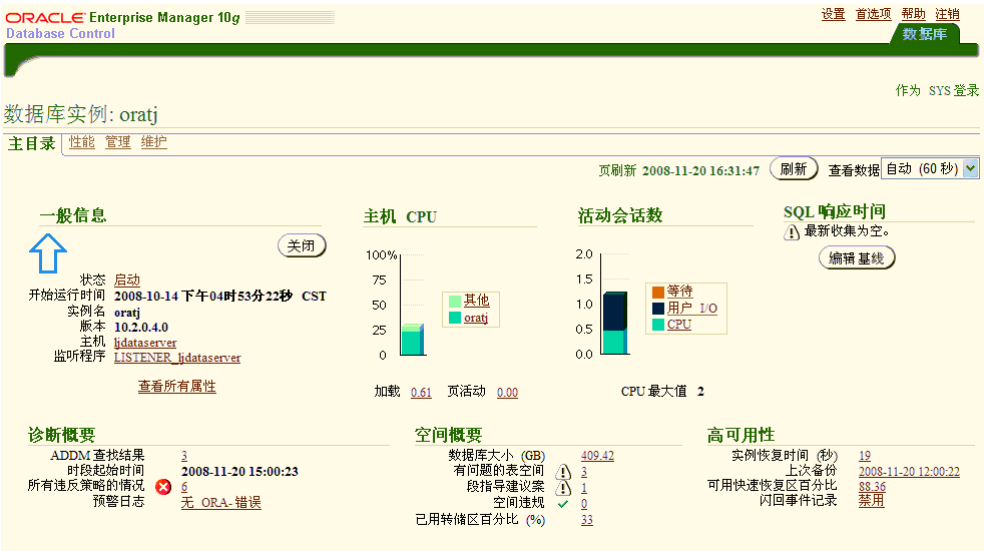


图 1-1 Oracle Database Control 主页面

而在 SQL 诊断部分*自动数据库诊断监视器*（Automatic Database Diagnostic Monitor，ADDM）更能够自动进行数据库问题的诊断并且给出调整和优化的建议，图 1-2 来自一个真实的客户系统诊断，ADDM 给出了存在显著性能问题 SQL 的调整建议：

图 1-2 Oracle ADDM 给出的 SQL 调整建议

很多客户对于 Database Control 的感觉就是，这个工具真实地简化了用户对数据库的管理和监控工作，提高了用户的工作效率，改变就是如此简单。

而在传统的数据库层面，数据库的自动管理与自我维护性在不断提高，Database Control 可以帮助我们更好地监控和管理数据库，AWR（自动工作负载信息库）使得信息的收集实现自动化，ADDM（自动数据库诊断监控程序）使得数据库可以根据 AWR 等信息自动地进行性

能分析和诊断，SQL Advisor、SPM（SQL Plan Management）可以帮助我们进行 SQL 的调整和提供建议…..

总体说来，**Oracle 更倾向于将新的数据库特性描述成一个具有主动性（Proactive）的产品，能够自主地、主动地发现数据库的问题，并提出优化和解决方案**，这些功能在 Oracle Database 11g 中被进一步深化。实际上，**Proactive 这个词也正是 2.0 时代的 DBA 应该具备的素质**。一个优秀的 DBA，在数据库越来越完善的时代，应该拥有更多的主动性、预见性，应该能够对系统作出良好的规划和预期，将错误或故障消灭在萌芽阶段，从而使数据库环境拥有更佳的稳定性；更进一步，一个 2.0 时代的 DBA 应该能够从企业的发展及大局出发，为企业规划更合理的数据管理方式、更有效的数据使用方式，从而不仅为企业节省投资，而且能够为企业创造更多的价值，DBA 的发挥空间还远远不止于此！

然而一切并不如此简单，**Oracle 一方面在提高数据库软件的自动化水平，另一方面却将更多的知识、技术囊括入数据库的领域，在这个层面，实际上使得 DBA 需要了解的内容愈加广阔**。比如，自动存储管理（Automated Storage Management，ASM）技术的引入使得 DBA 不得不更加深入地介入存储的管理和维护；集群软件（Clusterware）的引入，使得 DBA 不得不深入了解和维护 Cluster 软件；如果加上 Oracle 的 OEL（Oracle Enterprise Linux）和 2008 年推出的 Exadata 以及 HP Oracle Database Machine，那么现在主机、操作系统、OS 都要求一个 Oracle DBA 能深层次地介入和理解；2009 年 Oracle 公司收购了 SUN 公司并随之推出了 Exadata V2 和 OLTP Database Machine，现在 Oracle 能够提供基于主机、存储甚至外加 MySQL 的整体解决方案。这一切都使得 DBA 们不断面临新的挑战。

总结一下那就是，**在传统的数据库层面，Oracle 不断在强化自动化管理，提高数据库的自我管理性，减少用户的干预和工作量；而在数据库之外，更后端，DBA 需要不断向系统、存储甚至网络领域延伸，在前端，DBA 则需要不断向应用层面进行扩展**。而根据经验，不断向应用和业务方向延伸，是 DBA 职业发展的一个重要趋势。

DBA 2.0 的使命大致可以概括为：**在不断完善的数据库管理工作之外，向更广阔体系层面延伸，包括应用（Application）、系统（System）、存储（Storage）、网络（Network）、架构（Architecture）等方面，为企业提供更深远的架构与决策支持，促进企业向更全面合理的可持续性 IT 架构方向发展**。以前，很多 DBA 是由开发转型过来的，现在，**DBA 向开发回溯必将为企业和个人带来更高的价值实现**。

既然 DBA 2.0 意味着更广泛的层面介入、更深入的知识内涵，那么 Oracle 也不断在加强自己的产品，来降低 DBA 工作的复杂度，加强数据库的稳定性与可靠性等。实际上通过不断的收购以及长期的布局，Oracle 已经打造了以数据库为核心的全系列 DBA 辅助工具与产品。

比如，Oracle 最近推行的 **Real User Experience Insight** 产品一些情况，这是一款用于全方位监视实际系统运行性能的软件，用以确保基于 Web 的应用程序性能能够达到期望水平；在达不到期望水平时进行分析和通知，并提供用户应采取相关措施的软件。图 1-3 是其原理图，从 Page request 到请求返回，所有流程的响应都将被记录用于预警和评估：

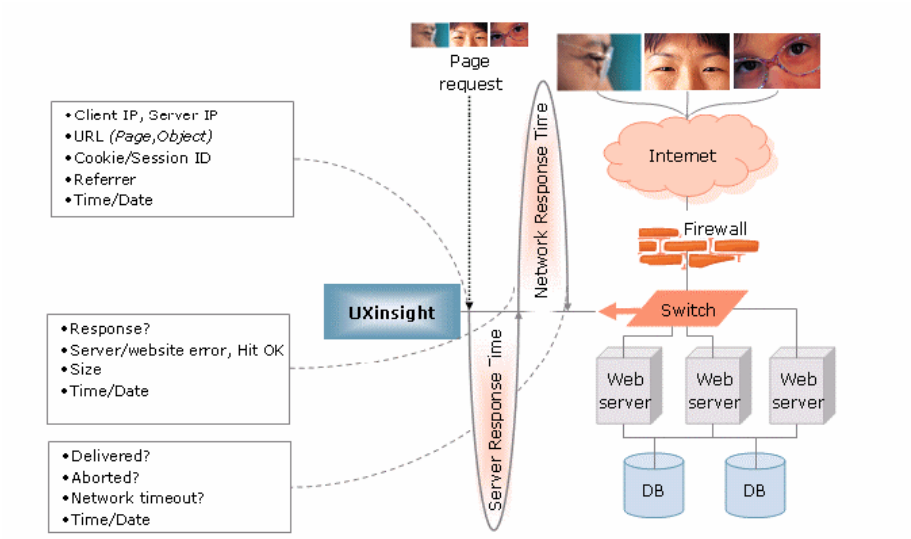


图 1-3 Oracle Uxinsight 产品的逻辑图

这是 Oracle 在应用监控方面的软件，实际上这是在向前端、应用层、网络层迈进，这是 Oracle 的扩展策略之一。

而 **Oracle Enterprise Manager 产品则是在数据库层面的巨大增强，Oracle 一如既往不断在数据库层面加强其产品能力的体现**。图 1-4 是 Oracle 在从开发到产品所有阶段自顶向下的应用管理示意图（摘自 Oracle 官方演示文档），其中数据库层面的 Diagnostic and Tuning、Provisioning、Configuration Management 等产品都是 OEM 产品的增强。

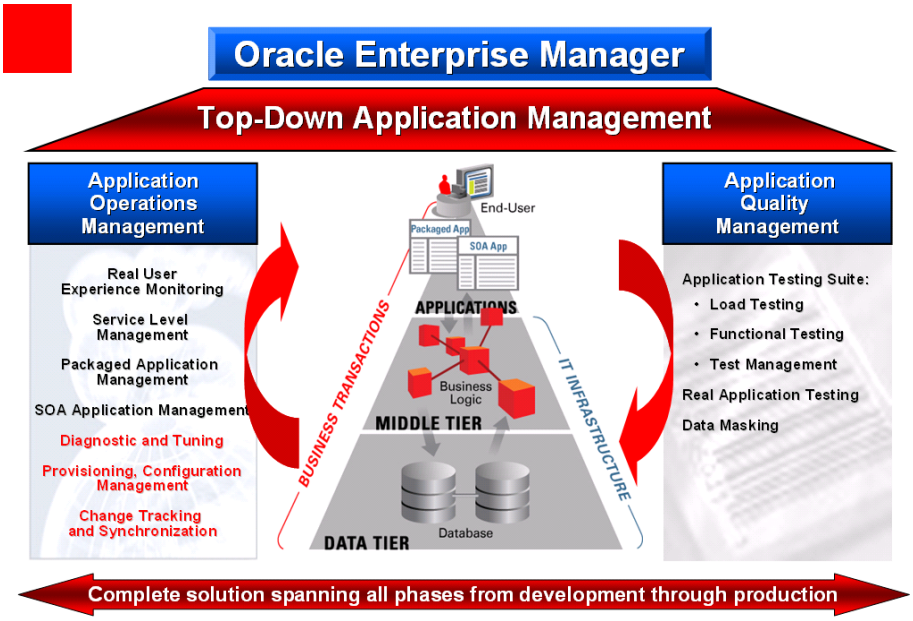


图 1-4 Oracle 自顶向下的管理产品结构

图 1-4 右侧的应用测试阶段，Oracle 也有了很多新的产品，包括 Real Application Testing（这是 Oracle Database 11g 中的功能，也整合了在 Oracle 10.2.0.4 及之后的版本中），而 Data Masking（Data Masking 组件在进行数据克隆时，可以对生产数据进行转换来保护原来的敏感

信息，但是仍然能够保持数据的完整关系用于真实的应用测试）也是最新包含进 OEM 中的组件；另外 Timesten 内存数据库现在也打包进了 Oracle Database 11g 数据库产品中，Oracle In-Memory Database Cache 现在可以作为数据库的一个组件销售。Oracle 确实已经能够提供从前端到中端再到后端的全方位的数据库产品。而在 2009 年 7 月，Oracle 又收购了数据同步复制领域的重要厂商 GoldenGate，意图强化异构数据库之间的数据交换及迁移。Oracle 以数据库为核心的产品战略布局已经发散到各个角落。

如果深入体验一下新的 OEM 产品，可以发现其涵盖的范围越来越广，**Oracle 的意图是将 OEM 逐渐演进成为一个全面的系统管理、维护与监控工具，不仅涵盖数据库，还要涉及系统、网络、存储等方方面面**。

现在我们能够透过 OEM 看到主机负载（如图 1-5）、IO 以及 ASM 等全面的性能及负荷信息（如图 1-6）：

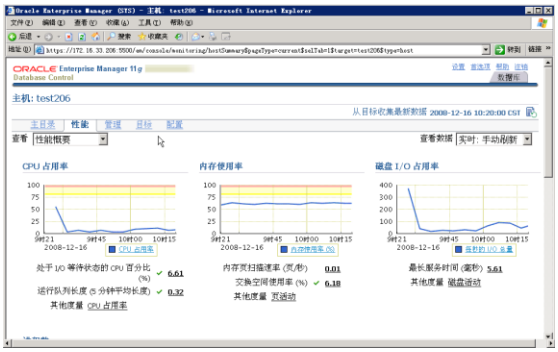


图 1-5　主机负载曲线

图 1-6 ASM 存储负载曲线

而在 Grid Control 中，更广泛的信息会被监控和收集，比如应用的响应与处理时间等（如图 1-7），通过进一步的研究与分析，可以在不同层面对性能问题进行掌控和解读：

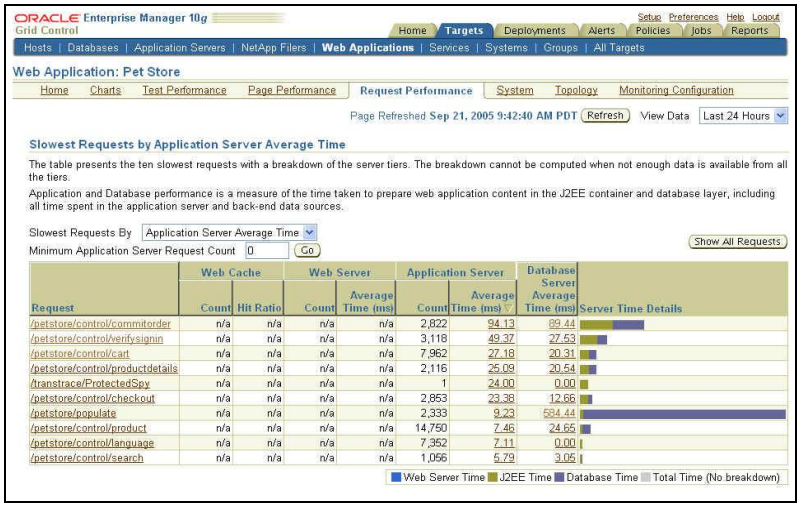


图 1-7 Web Application 的性能跟踪

虽然 OEM 还不能解决我们所有的问题，但是在日常工作方面，已经可以成为我们很好的助手。

OEM 通过全面的监控部署，使原本须要手工处理的我们进行大量工作可自动进行，以前

要脚本编写处理的工作，现在 OEM 可以内置地自动完成，这部分增强对于 DBA 具有普遍的价值。

比如在监控方面，我们可以定义各种各样的度量条件（如图 1-8），达到当特定阈值时，就触发报警：

图 1-8 OEM 的管理度量

如果配置了 SMTP 及邮件设置，数据库出现相关告警就可以自动发出告警邮件。**通用功能的增强，是 OEM 中非常重要的部分，因为这部分增强可以将 DBA 普遍需要进行的工作简化，实现用户效率的提升**。

OEM 中的 Diagnostic Pack 和 Tuning Package 对于 DBA 具有更高的诊断与优化价值，通过这两个工具包，DBA 可以很容易捕获并分析性能问题，并且可以参考数据库的建议进行改进。使用 OEM 进行问题 SQL 的捕获（注意，**不要等待问题出现时才去关注，日常对于数据库的关注与监控尤为重要，也只有对数据库进行更全面的跟踪才能使 DBA 更具有预见性**）与诊断变得非常快速与简便，这些变化甚至可以让对数据库仅有初步认识的人在解决数据库问题时也变得十分有效（如图 1-9）：

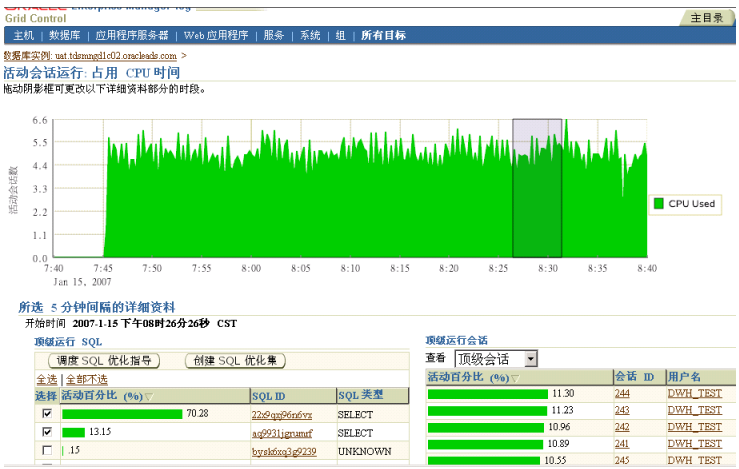


图 1-9 活动会话 SQL 诊断

而这些在 OEM 中可以快速完成的工作，在以前由于其复杂度可能会让很多对数据库了解不多的技术人员望而却步！Oracle 在不断变革，DBA 的世界也在不断改变，我们需要做的是

跟上这些变化，继续出发。

1.2　DBA 日常工作职责——我对 DBA 的 7 点建议

DBA 的工作职责是什么？每天 DBA 应该做哪些工作？稳定环境中的 DBA 该如何成长与优化？这是很多人都曾经提出过的问题，下面是我的观点和建议，供参考：

1. 实时监控数据库告警日志

作为一个 DBA，或者哪怕仅仅是和 Oracle 数据库打交道的技术人员，你都必须知道告警日志是什么，在何处。

而对于 DBA 来说，实时的监控数据库的告警日志是必须进行的工作，监控并且应该根据不同的严重级别，发送不同级别的告警信息（通过邮件、短信），这可以帮助我们及时了解数据库的变化与异常，及时响应并介入处理。

2. 实时监控数据库的重要统计信息

实施监控对于数据库运行至关重要、要高度关注那些能够表征数据库重要变化的统计信息，并且据此发送报警信息。那么应当监控哪些统计信息呢？大家应当区别条件深入思考，对于单机、RAC 环境等各不相同。

3. 部署自动的 Statspack/AWR 报告生成机制

每天检查前日的 AWR 报告，熟悉数据库的运行状况，做到对于数据库了如指掌

4. 每天至少优化和熟悉一个 Top SQL

根据 AWR 或 Statspack 报告,每天至少了解或熟悉一个 Top SQL,能优化的要提出优化和调整建议。一个 DBA 应当对稳定系统中的 SQL 非常熟悉和了解，这样才可能在系统出现性能问题时见微知著，快速地作出判断和响应。

5. 部署完善的监控和数据采样系统

DBA 应该对数据库部署完善的监控系统，并对重要信息进行采样，能够实时或定期生成数据库重要指标的曲线图，展现数据库的运行趋势。

6. 全面深入地了解应用架构

不了解应用的 DBA 是没有前途的 DBA，对应用了解不深入的 DBA 算不上 Expert，所以一定要深入了解应用。

在数据库本身变得更加自动化和简化之后，未来的 DBA 应该不断走向前端，加深对于应用的了解，从应用角度对数据库及全局进行把握和优化。

7. 撰写系统架构、现状、调整备忘录

根据对数据库的研究和了解，不断记录数据库的状况，撰写数据库架构、现状及调整备忘录，不放过任何可能的优化与改进的机会。

1.3　DBA 最重要的素质有哪些

在招聘 DBA 的时候，很多朋友经常问起我对于 DBA 的要求。其实在打算培养一个

DBA 时，我们的要求并不复杂，甚至很多要求跟技术无关。

对于一个准备进入 DBA 领域的人，我希望他**勤奋、严谨、具有钻研精神及独立思考能力**。如果不是要求特别高的职位，其实一般技术往往并不是我们最关心的内容，因为具备了前面的素质之后，经过 1~2 年的锻炼，一个人绝对不会知道的太少，而我们通过简单的提问就很容易知道一个人是否曾经深入的思考或者研究某些技术问题，在简历上的任何伪装都是不明智的，太多精通的字眼已经使这两个字的可信度大大降低，所以**诚实这种品质也非常重要**。

也许勤奋,严谨,具有钻研精神及独立思考能力已经被每个人写在了自己的简历之中,可是真正做到的却并不多。我在论坛上亲眼看到很多人从入门到迅速成长为技术专家，而另外很多人却是数年如一日,没有太大的进步,说白了就是缺乏我前面提到的四点,或者没有很好的理解这些简单的道理。

如果一个人真正对某件事情很投入且执著,那么他的进步一定会稳健而迅速，这样的人,在具备了一定的基础之后,通常所有的公司都会给他时间学习成长。技术可以逐步学习,态度和风格却很难改变。

我见到过一些大学生，在大学中已经对 Oracle 进行了基于兴趣和爱好的长期学习和钻研，这使得他们更容易获得工作机会；而另外一些声称喜欢或具备兴趣的人，往往在大学四年中甚至没有 Google 一下去更加深入地了解一下 Oracle；这其中的差异是显而易见的。

所以我曾经在《Oracle 数据库性能优化》一书的序言中写到:

兴趣 + 勤奋 + 坚持 + 方法 ≈ 成功

很遗憾我不能给以上公式画上“=”，但只要具备了以上条件，我想每个人都离成功不远了。

明白了这些道理之后，在工作中应当切忌浮躁，我们每个人从毕业到工作、再到找到适合自己的位置，这通常都要经过一个较长的时期，学习、思考、进步、再次出发，所以要知道有时候等待是过程的必须。一个人从毕业到成长为具有稳定职业发展观，总需要一个过程，在这个过程中，要保持冷静，拒绝浮躁。

如果你手上已经有了一份工作，那你需要做的是，做好它，哪怕那不是你喜欢的！

你必须证明给别人看你有做好一件事情的能力，别人才会给你下一个机会。

总结一下，我想说的是，十年磨一剑，有时候等待是必须的，正视这个历程，珍重这个历程才是正确的态度。

我上面说，十年磨一剑，在这个历程中，你应当一直在琢磨，使自己成长，具备进一步跳跃的素质，如果你喜欢 Oracle，那你应当在这个阶段完成积累过程。很多人经常是偶然有一天头脑发热，说我要去搞 Oracle，这未尝不可，不过没有积累，你的起步要困难得多。

在具备了一定的积累之后，你需要的其实是机会，机会在哪里？

机会喜欢光顾有准备的人，你做好准备之后，静静的去寻找和等待机会，这是你应该做的。很少有机会会从天而降，降临到毫无准备的人头上，如果你希望获得机会，那

机会会看你准备得如何。

学习是没有止境的，在工作中认真处理遇到的问题，从每一个可能遇到的问题入手，深入再深入，这是对现实的把握。把握好现实才能把握未来。

去除浮躁，认真学习，不断积累，寻找机遇，这是我给很多准备成为 DBA 的朋友的建议。

1.4　DBA 职业生涯之误删除篇

前面我提到严谨这一素质对于 DBA 的重要性，在 ITPUB 论坛上曾经有过一个热烈的主题：“请列出你在从事 DBA 生涯中,最难以忘怀的一次误操作”

这一主题引起了大家普遍的兴趣，很多有趣的案例呈现出来，我摘录了一些和误删除有关的案例，看看都有哪些一时马虎、不够严谨会犯下的过失，大家应引以为戒，共同警醒：

1. 在 linux 平台上，一次不小心操作，把 oradata 下所有的东西全删除了
   这样的误操作，教训是极其惨痛的，所以备份对于 DBA 来说仍然是最重要的。
2. 一次误删了个表，最后恢复了，丢了一天的数据。加了一晚上班，至今记得。
   人越累的时候就越容易犯错误，我就是在最后快下班的几分钟犯的错误。一定要记住墨菲定律，越着急就越容易犯错误，DBA 千万不要赶着去做一件事。
3. 在一次测试过程中，把一个在本机执行的删除所有非系统用户的脚本，错误地粘到一个开发数据库的 sqlplus 窗口中了。如果你不注意剪贴板，它就会害你！
4. 有一次一不小心把一个表给 truncate 了，上千万条记录一眨眼就没了。
   DDL 操作一定要谨慎啊！
5. rm -rf /opt/ora92/* 在测试库中本来想删除数据库，结果错误地把 Oracle 软件删除了.
   rm -rf 是相当恐怖的，每个 DBA 都应该学会不要直接使用这个命令。
6. 不小心用 rm -rf /home 目录下的所有文件,/home 目录下放的有账务系统的 app。
   一看到删除的路径不对时，挽救已经来不及了。
   又是一个 rm -rf 的狠操作。
7. 删除一些 trace 文件，然后就直接删除 rm orcl*，结果通过 vpn 到生产的网络太慢，命令刚慢慢地显示出来，看都没看直接按回车，结果执行的命令却是 rm orcl *，因为 orcl 和星号中间有个空格，所以把这个目录下面所有的内容全部删除了。
   网络慢的确害惨过很多人，所以网络可能存在问题时，任何操作都要慎重，不行可以使用脚本在后台跑。
8. 有很多在 OEM / Toad 中误击键盘删除用户或表空间的情况。

实际上，我们看到，很多误操作都和技术能力基本无关，只是要细致、严谨，再认真一点。DBA 有些素质和习惯是必须养成的。

很多时候，只要多思考一些，就可以将工作做得更加完美，就可以严谨的避免很多

问题，举一个 DBA 可能经常面对的工作任务作为例子：

如果在数据库中执行 DDL 操作，比如：

CREATE INDEX / REBUILD INDEX … ONLINE COMPUTE STATISTICS / GRANT / REVOKE / COMPILE / ANALYZE / DBMS_STATS….

进行严密严谨的思考，这些操作可能带来什么后果?

通常我们都知道，DDL 会使解析过的 SQL 失效、会使存储过程等对象失效发生重编译、也有可能引起 SQL 执行计划的改变……

这些可能在一个繁忙的业务数据库中带来灾难，如果大量的 SQL 同时失效，同时重新解析，就可能带来大量的 Shared Pool 与 Library Cache 的竞争，SQL 解析与竞争又会导致 CPU 的高消耗，这些在繁忙系统中可能会导致负荷突然升高，严重的甚至会导致系统阶段性地挂起。

已经有很多 DBA 在这些问题上付出了惨痛代价，而我只提醒大家，做事时要能够明确工作的性质、分析潜在的风险、回避可能引发的问题。

1.5 DBA 警世录——有些习惯 DBA 须要养成

既然 DBA 这个职业如此危险，那么哪些习惯是 DBA 必须养成的呢？

我总结过几条简单的习惯命令，通过这些习惯性命令，可以减少我们出错的可能。写下这段内容时，刚刚完成一个客户数据库的恢复，客户造成故障的原因很简单，因为维护升级时错误地连接到生产主机，结果导致生产库故障，数据文件被删除并部分覆盖。

1. 经常使用 hostname 命令

在 Linux/Unix 上，我们使用 ssh 或 telnet 等通过多次跳转，很容易变更了连接主机，如果不经过确认就可能在不正确的主机上执行了错误的操作。

通过 hostname 命令可以确认我们连接到的主机，避免发生不应该的误操作。在执行操作之前一定要通过hostname命令确认连接主机,这是DBA或者系统管理员应该养成的习惯:

```
[oracle@jumper oracle]$ hostname
jumper.hurray.com.cn
```

2. 使用 pwd 确认路径

经常有朋友在错误地路径下错误的执行了"rm -rf *"等命令，这类错误的发生率居然也是很高的。

所以作为一个 DBA，应经常性地执行如下的 pwd 命令来确认自己的工作路径:

```
[oracle@jumper oracle]$ pwd
/opt/oracle
```

3. 确认 instance_name 等数据库重要信息

在执行 truncate/drop 等操作之前，应该确认连接到了哪个数据库，从 v$database 或 v$instance（代码如下）等视图中可以获得这些信息(可能需要授权)：

```
SQL> select instance_name,host_name from v$instance;
```

```
INSTANCE_NAME HOST_NAME
---------------- ----------------------------------------------------------------
eygle  jumper.eygle.com
```

4. 通过 id 命令确认用户信息

要经常通过如下的 id 命令确认用户信息，以免切换用户而导致不自觉的异常操作：

```
[gqgai@jumper gqgai]$ id
uid=2003(gqgai) gid=101(dba) groups=101(dba)
```

我见到过有的案例，用户切换为 root，误操作删除过大量系统文件，导致了严重的故障。

5. 对 DDL 语句心存敬畏

DBA 应该知道 TRUNCATE / DROP 等 DDL 操作可能带来的影响，所以应该对这些 DDL 操作心存敬畏，甚至应该避免执行或避免草率执行这样的操作，最好养成在 DDL 清除数据之前备份的习惯。

通过一些良好习惯的养成，可以使得我们少犯错误。
学会总结，学会从别人的教训中积累经验，这对 DBA 来说必不可少！

1.6　RAC 环境下故障处理一则

这是客户的一个 Oracle9iR2 RAC 数据库环境，数据库版本为 Oracle 9.2.0.8，主机为 IBM P55a 小型机。最初客户集群环境运行稳定，后来一台主机出现硬件故障退出集群，问题出现在主机维修完毕之后，当故障主机重新加入现有环境运行，客户发现应用性能出现衰减，前端收费系统响应缓慢，反而不如一个节点工作时的性能。

首先可以看一下这个系统的逻辑读变化曲线，图 1-10 清晰地显示在 17 日左右系统变成单节点运行后，逻辑读有了一个双倍的攀升变化，非常线性和清晰：

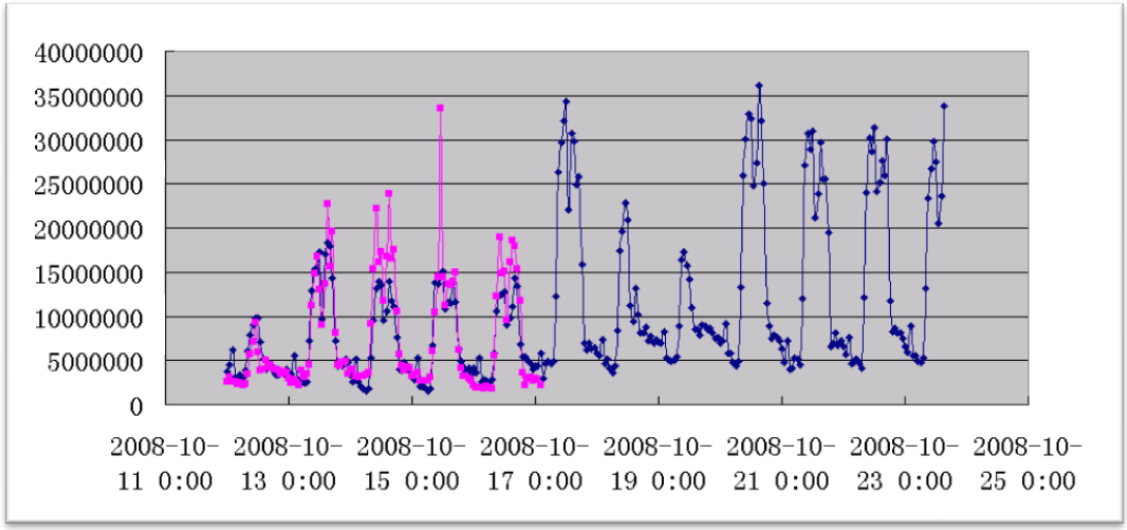


图 1-10 系统的逻辑读变化曲线

现在的问题是，两个节点运行反而不如当初的单节点运行，那么很有可能是新恢复运行的节点存在问题。登录该主机首先使用 vmstat 来收集系统信息,从以下摘要数据可以看到，系统的内存消耗殆尽，Page In/Out 较高，同时 Wait 较高：

```
p55a1#vmstat 2

System configuration: lcpu=4 mem=1840MB

kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 0  0 802203  2591   0  22  51  64  247   0 600 3182 1434  3  1 91  5
 0  3 802477  2776   0  38 151 345 1606   0 629 2509 1270  1  1 87 10
 0  1 801897  3098   0 207  76   0    0   0 910 3306 1762  3  2 74 22
 0  0 802229  2549   0 131  92 193  692   0 2138 12989 6000 10  6 69 15
 3  0 801876  2812   0  71  26   0    0   0 953 5782 2145 10  2 79  8
 1  1 801877  2814   0 139  95 141  503   0 1095 6617 2810 12  3 70 14
 0  2 801887  2546   0 128   0   0    0   0 780 3566 1909  5  2 79 14
 0  2 801883  3042   0 150 284 401 4301   0 1079 3067 1820  3  2 56 39
 0  1 802061  2532   0 162   0   0    0   0 1191 5444 2630  2  2 81 14
 1  1 802060  3193   0  51 270 381 2387   0 704 2468 977  3  1 77 18
 1  3 802535  2827   0 256 201 338 2011   0 975 2808 1624 11  2 51 36
 0  4 802852  2606   0 263 221 287 1535   0 1181 3217 2044  5  2 51 42
 1  2 802455  2838   0 339 166 258 1154   0 1181 4040 2181 13  2 47 37
```

通过 topas 可以做进一步的诊断，从图 1-11 的输出中同样可以看到类似数据，系统的 I/O Wait 较高，Paging 频繁，而磁盘 hdisk5 和 hdisk6 的 Busy 程度达到了近 100%：

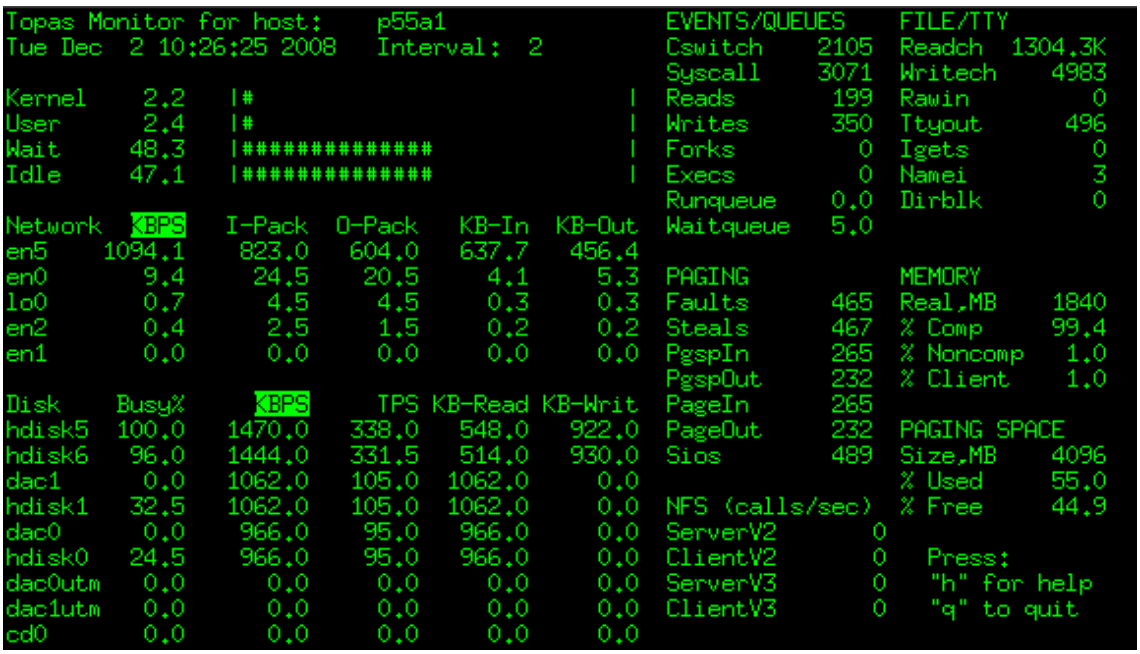


图 1-11　执行 topas 命令的输出

这说明由于主机的交换频繁导致了 I/O 负荷升高，从而导致了数据库响应缓慢。那么为什么会出现这种问题呢？内存的紧张和交换频繁直接和数据库的内存使用有关。

登录数据库，查看一下 SGA 设置就能够明白问题的原因所在（见如下代码）：

```
p55a1$sqlplus "/ as sysdba"

SQL*Plus: Release 9.2.0.8.0 - Production on Tue Dec 2 10:34:20 2008
Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.
SQL> show sga

Total System Global Area 2031584960 bytes
Fixed Size                  743104 bytes
Variable Size             956301312 bytes
Database Buffers         1073741824 bytes
Redo Buffers                798720 bytes
```

我们注意到数据库的 SGA 设置，分配了大约 2GB 的内存，而根据如下系统显示，主机配置的内存也仅为 2GB:

```
System configuration: lcpu=4 mem=1840MB
```

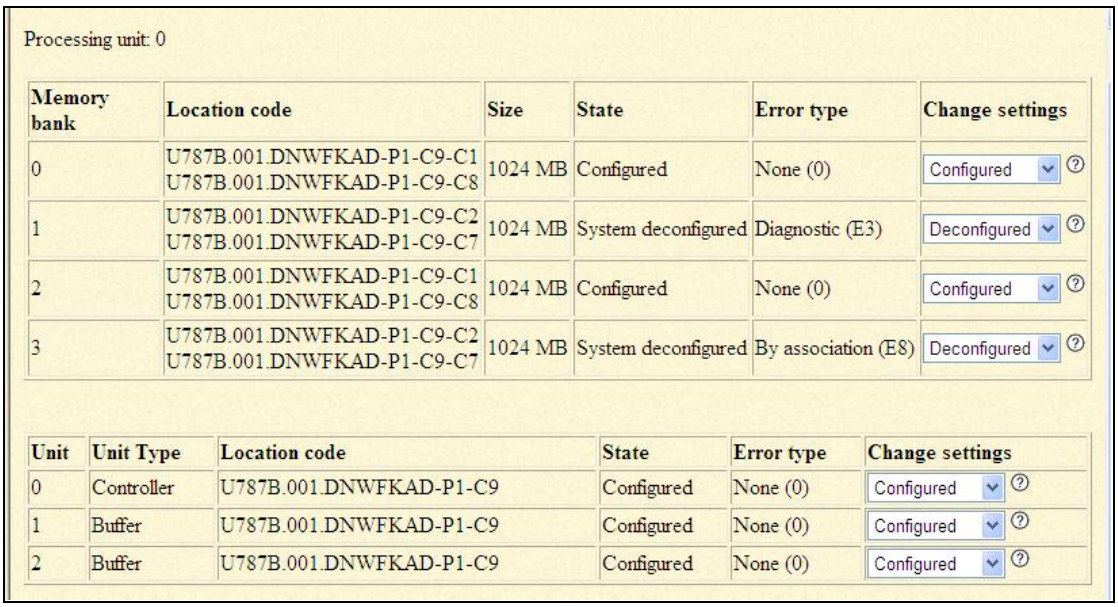
看来问题的原因已经找到，由于数据库的 SGA 设置过高，导致了过高的内存消耗，使得系统在业务繁忙时段产生了大量的交换，影响了系统性能。那么解决这个问题的方式是要么降低 SGA，要么扩展系统内存。

但是还有一个问题，为什么以前一切是正常的，而机器修复后反而出现了问题呢？其实

系统除了维修并未做其他数据库参数方面的变更。

适当猜测加上询问客户和查看记录就发现了进一步的原因，那就是在维修之前该主机共有 4GB 内存，维修之后只剩下了 2GB，而客户并未主动变更过，那么显然还有故障存在在主机上（虽然 errpt 并未显示任何警告）。

在机房通过笔记本连接 P55A 小型机后端的 HMC1 口，通过浏览器连接主机，发现根本原因所在，原来是有两个内存被 Deconfigured 了（见图 1-12）：



Processing unit: 0

Memory bank	Location code	Size	State	Error type	Change settings
0	U787B.001.DNWFKAD-P1-C9-C1 U787B.001.DNWFKAD-P1-C9-C8	1024 MB	Configured	None (0)	Configured
1	U787B.001.DNWFKAD-P1-C9-C2 U787B.001.DNWFKAD-P1-C9-C7	1024 MB	System deconfigured	Diagnostic (E3)	Deconfigured
2	U787B.001.DNWFKAD-P1-C9-C1 U787B.001.DNWFKAD-P1-C9-C8	1024 MB	Configured	None (0)	Configured
3	U787B.001.DNWFKAD-P1-C9-C2 U787B.001.DNWFKAD-P1-C9-C7	1024 MB	System deconfigured	By association (E8)	Deconfigured

Unit	Unit Type	Location code	State	Error type	Change settings
0	Controller	U787B.001.DNWFKAD-P1-C9	Configured	None (0)	Configured
1	Buffer	U787B.001.DNWFKAD-P1-C9	Configured	None (0)	Configured
2	Buffer	U787B.001.DNWFKAD-P1-C9	Configured	None (0)	Configured

图 1-12 Memory Deconfigration

也就是说可能这两条内存出现了故障，或者可能是内存插槽有问题，导致 2GB 的内存未被加载，这才是这次故障的根本原因。通过联系硬件厂商，解决了内存故障后，数据库系统恢复正常。

1.7 SQL_TRACE 跟踪与诊断

2004 年，笔者曾经帮客户处理过一则案例，其中涉及 SQL_TRACE 的使用，我们首先来回顾一下这个案例。

客户的应用是一个后台新闻发布系统，主要性能问题是通过连接访问新闻页极其缓慢，通常需要十数秒才能返回，这是用户不能忍受的。客户操作系统是 SunOS 5.8，数据库版本为 8.1.7。

面对这个问题，首先想到的是 SQL 问题，但如何定位具体的问题 SQL 成为我们考虑的主要目标，通过 Statspack 采样是一个全局手段，而通过 sql_trace 则可以实时对会话进行跟踪。诊断时是晚上，在无集中用户访问情况下，让用户在前台进行相关页面的访问，同时进行进程跟踪。

查询 v$session 视图，获取进程信息：

```
SQL> select sid,serial#,username from v$session where username is not null;
       SID    SERIAL# USERNAME
---------- ---------- ------------------------------
         7        284 IFLOW
        11        214 IFLOW
        12        164 SYS
        16       1042 IFLOW
```

然后对相应的应用会话启用 sql_trace 跟踪如下：

```
SQL> exec dbms_system.set_sql_trace_in_session(7,284,true)
SQL> exec dbms_system.set_sql_trace_in_session(11,214,true)
SQL> exec dbms_system.set_sql_trace_in_session(16,1042,true)
```

应用执行一段时间后，关闭 sql_trace：

```
SQL> exec dbms_system.set_sql_trace_in_session(7,284,false)
SQL> exec dbms_system.set_sql_trace_in_session(11,214,false)
SQL> exec dbms_system.set_sql_trace_in_session(16,1042,false)
```

检查 trace 文件，可以找到跟踪过程中前台执行的 SQL 调用，检查发现以下语句是可疑的性能瓶颈点：

```
********************************************************************************
select auditstatus,categoryid,auditlevel from
 categoryarticleassign a,category b where b.id=a.categoryid and articleId=
  20030700400141 and auditstatus>0

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.81       0.81          0       3892          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.81       0.81          0       3892          0           1
********************************************************************************
```

这里的查询显然是根据 articleId 进行新闻读取的，但是注意到逻辑读有 3892，这是较高的一个数字，这个内容引起了我的注意。

接下来的类似查询跟踪得到的执行计划显示，全表访问被执行：

```
select auditstatus,categoryid from
 categoryarticleassign where articleId=20030700400138 and categoryId in ('63',
  '138','139','140','141','142','143','144','168','213','292','341''-1')
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
```

```
Execute      1      0.00       0.00          0          0          0          0
Fetch        1      4.91       4.91          0       2835          7          1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      4.91       4.91          0       2835          7          1

Rows     Row Source Operation
-------  ---------------------------------------------------
      1 TABLE ACCESS FULL CATEGORYARTICLEASSIGN
```

登录数据库，检查相应表结构，看是否存在有效的索引，以下输出中的 IDX_ARTICLEID 基于 ARTICLEID 创建，但是在以上查询中都没有被用到：

```
SQL> select index_name,table_name,column_name from user_ind_columns
  2  where table_name=upper('categoryarticleassign');
INDEX_NAME                     TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------ --------------------
IDX_ARTICLEID                  CATEGORYARTICLEASSIGN          ARTICLEID
IND_ARTICLEID_CATEG            CATEGORYARTICLEASSIGN          ARTICLEID
IND_ARTICLEID_CATEG            CATEGORYARTICLEASSIGN          CATEGORYID
IDX_SORTID                     CATEGORYARTICLEASSIGN          SORTID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          ARTICLEID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          CATEGORYID
PK_CATEGORYARTICLEASSIGN       CATEGORYARTICLEASSIGN          ASSIGNTYPE
```

检查如下表结构:

```
SQL> desc categoryarticleassign
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CATEGORYID                                NOT NULL NUMBER
 ARTICLEID                                 NOT NULL VARCHAR2(14)
 ASSIGNTYPE                                NOT NULL VARCHAR2(1)
 AUDITSTATUS                               NOT NULL NUMBER
 SORTID                                    NOT NULL NUMBER
 UNPASS                                             VARCHAR2(255)
```

我们发现了问题所在，因为 ARTICLEID 是个字符型数据,查询中给入的 articleId=20030700400141 是一个数字值，Oracle 发生潜在的数据类型转换，从而导致了索引失效：

```
SQL>  select auditstatus,categoryid
  2   from
  3    categoryarticleassign where articleId=20030700400132;
AUDITSTATUS CATEGORYID
----------- ----------
          9         94
```

```
          0        383
          0        695
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=110 Card=2 Bytes=38)
   1    0   TABLE ACCESS (FULL) OF 'CATEGORYARTICLEASSIGN' (Cost=110 Card=2 Bytes=38)
```

解决方法很简单，只须在参数两侧各增加一个单引号，即可解决这个问题，对于类似的查询，我们发现 Query 模式读取降低为 2，占用 CPU 时间也大大减少：

```
********************************************************************************
select unpass from
 categoryarticleassign where articleid='20030320000682' and categoryid='113'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.00       0.00          0          2          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3      0.00       0.00          0          2          0           0

Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 20

Rows     Row Source Operation
-------  ---------------------------------------------------
      0  TABLE ACCESS BY INDEX ROWID CATEGORYARTICLEASSIGN
      1   INDEX RANGE SCAN (object id 3080)
********************************************************************************
```

至此，这个问题得到了完满的解决。

1.8　临时表空间组导致递归 SQL 高度解析案例

在 2009 年的一次客户现场服务中，很惊奇地再次看到了熟悉的表名和用户，这套系统（如图 1-13）就是 1.7 我写到过的那个发布系统，作为一个 DBA，多年以后看到似曾相识的数据库，是多么令人感慨。

select * from (select rownum r, t.* from (select ID, TITLE, IMAGE, RELEASETIME from iflow.article a, iflow.categoryarticleassign c where a.id = c.articleid and c.auditstatus =9 and c.categoryid =:1 order by (a.releasetime + c.puttoptime) DESC) t where rownum <=:2) where r>:3
select * from (select rownum r, t.* from (select ID, TITLE, IMAGE, RELEASETIME from iflow.article a, iflow.categoryarticleassign c where a.id = c.articleid and c.auditstatus =9 and c.categoryid =:1 order by (a.releasetime + c.puttoptime) DESC) t where rownum <=:2) where r>:3
select * from (SELECT c.articleid FROM categoryarticleassign c, article a, category b WHERE a.deleted=0 and c.categoryid =87113 AND c.articleid = a.id and b.id=c.categoryid and c.auditstatus=9 ORDER BY a.releasetime DESC) where rownum < 16
select * from (SELECT c.articleid FROM categoryarticleassign c, article a, category b WHERE a.deleted=0 and c.categoryid =113434 AND c.articleid = a.id and b.id=c.categoryid and c.auditstatus=9 ORDER BY a.releasetime DESC) where rownum < 16

图 1-13 AWR 中采样的部分 SQL

从图 1-13 中摘录的是 AWR 中采样的一些 SQL，这些 SQL 没有绑定变量，是性能影响的一个因素。

客户反映这个系统的主要问题是 CPU 消耗很高，经常处于 90%以上运行。我们对数据库 AWR 采样生成了一个 10 小时的采样报告（如图 1-14 所示），这是一个运行于 IBM P595 之上的 Oracle 10.2.0.2 版本的 RAC 集群数据库：

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
P5DB	554354975	p5dbc	1	10.2.0.2.0	YES	p595c

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	20436	08-Jun-09 09:00:32	645	13.0
End Snap:	20446	08-Jun-09 19:00:13	638	26.2
Elapsed:		599.68 (mins)		
DB Time:		5,182.06 (mins)		

图 1-14 AWR 生成的 10 小时采样报告

从图 1-14 报告的 DB Time/Elapsed = 8.6，可以获得的整体印象是，数据库处于相当繁忙的运行状态。

如图 1-15 的负载概要信息(Load Profile)进一步显示，数据库每秒的逻辑读高达 857 104.76 次，SQL Parses 每秒为 750.16 次，频繁的 SQL 解析和 User Calls 是 CPU 消耗的另外一方面的体现：

Load Profile

	Per Second	Per Transaction
Redo size:	47,546.76	1,380.30
Logical reads:	857,104.76	24,882.14
Block changes:	310.25	9.01
Physical reads:	241.41	7.01
Physical writes:	18.09	0.53
User calls:	1,686.25	48.95
Parses:	750.16	21.78
Hard parses:	6.19	0.18
Sorts:	4,716.05	136.91
Logons:	2.08	0.06
Executes:	1,034.00	30.02
Transactions:	34.45	

图 1-15　负载概要信息

为了了解 SQL 分析调用的信息，我们可以进一步来查看 SQL Statistics 部分的 Parse Calls 模块内容（如图 1-16 所示），在这部分信息中，发现了一条可疑的高解析执行的 SQL：

SQL ordered by Parse Calls

- Total Parse Calls: 26,991,433
- Captured SQL account for 90.4% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
21,220,546	21,298,871	78.62	8sph6b5p41afr		select min(bitmapped) from ts$...
531,686	531,682	1.97	8q7uku4ra1w29	JDBC Thin Client	SELECT * FROM site WHERE sitei...
266,963	266,940	0.99	2mfrm3zvjhs66		select * from CONTENT where NO...
250,411	250,442	0.93	7h35uxf5uhmm1	Admin Connection	select sysdate from dual

图 1-16 Parse Calls 的输出

排在第一位的这条 SQL 占据了 78.62%的解析比重，在 10 小时的 AWR 报告采样中，共执行了两千多万次，平均每秒解析执行约 600 次，这个高解析执行的 SQL 以超乎寻常的频率解析执行引起了我们的注意，其 SQL 的完整文本为：

```
select min(bitmapped) from ts$ where dflmaxext =:1 and bitand(flags, 1024) =1024
```

直观判断这条 SQL 是和系统递归调用相关的，查询了底层的 ts$视图，其调用如此频繁必然和大多数查询有关，尝试跟踪一下普通查询，我们发现这个 SQL 有很高的解析度。比如跟踪如下的 SQL：

```
SQL> alter session set events '10046 trace name context forever, level 12';
Session altered.

SQL> select count(*) from dba_indexes;
```

```
  COUNT(*)
----------
      5890
```

tkprof 格式化后台跟踪文件可以发现，在这个查询中，后台 ts$递归查询高达 3305 次，并且逻辑读很高：

```
select min(bitmapped) from
 ts$ where dflmaxext =:1 and bitand(flags,1024) =1024

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse     3305      0.02       0.04          0          0          0           0
Execute   3305      0.24       0.30          0          0          0           0
Fetch     6610      0.90       1.05          0     317280          0        3305
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total    13220      1.16       1.39          0     317280          0        3305
```

这使得我怀疑可能是某个 Bug 在作祟，检索 Metalink，马上发现了相关 Bug，Bug 号为：5455880。该 Bug 的影响版本如图 1-17 所示：

Product (*Component*)	Oracle Server (Rdbms)
Range of versions *believed* to be affected	Versions < 11
Versions *confirmed* as being affected	• 10.1.0.5 • 10.2.0.3
Platforms affected	Generic (all / most platforms affected)

图 1-17 Bug 号为 5455880 的影响版本

客户的数据库版本为 10.2.0.2，正好在受影响之列，这个 Bug 是说，当使用了 Oracle 10g 的临时表空间组特性时，后台的递归 SQL 可能会发生高昂的解析及执行：

```
When using a tablespace group as the temporary tablespace
excessive recursive queried against TS$ can impact performance.
The offending SQL is of the form:
  "select min(bitmapped) from ts$  where dflmaxext =:1 and bitand(flags,1024)=1024"
```

这个 Bug 在 10.2.0.4 之后修正，暂时的解决方案是停用临时表空间组。用户调整了临时表空间组的使用之后，这个 SQL 立即消失了，系统的解析等负载概要信息也发生了较大变化（使用$ORACLE_HOME/rdbms/admin/awrddrpt.sql 可以生成两个时段的 AWR 比较报告），如图 1-18 所示：

	1st Per Sec	2nd Per Sec	%Diff	1st Per Txn	2nd Per Txn	%Diff
Parses:	743.69	145.64	**-80.42**	22.40	5.78	-74.20
Sorts:	4,534.49	2,670.65	**-41.10**	136.60	106.02	-22.39
Executes:	1,032.02	425.26	**-58.79**	31.09	16.88	-45.71

图 1-18 两个时段的 AWR 比较报告

在使用 Oracle 的一些新特性时，一定要注意观察，看是否会引发一些新的问题，而**DBA 应该对系统中的一些异常 SQL 具有一定的敏锐性，要认真细致及时审查确认，才能保障数据库的持续稳定运行**。

1.9　使用闪回查询恢复误删除的数据

某日，一个朋友的数据库数据被误操作删除（DELETE）掉了，并且已经提交请求请我帮忙进行恢复。数据库版本是 Oracle10g Release 2 的，我首先想到的是使用 Flashback Query 进行闪回恢复，不幸的是出现了 ORA-01555，数据已经不能被闪回了。

Oracle 从 9i 开始推出的闪回查询特性被一直不断增强着，闪回查询通过对回滚段中存储的前镜像数据进行追溯，可以获得变更之前的数据，从而在前镜像被覆盖之前，提供了一种快捷的恢复和回退方式，在 Oracle Database 11g 中，Oracle 更加引入了闪回数据归档的新特性，允许将前镜像数据通过独立的表空间进行归档，从而提供长达数年的历史追溯。

闪回特性受到初始化参数 undo_retention 的影响，这个参数被在 10g 中默认地设置为 900 秒，这个时间长度一般是不够的，我们可以根据数据库的具体情况，酌情调高这个参数，将前镜像的期望保留时间延长，一般习惯将这个参数修改为 10800 秒，即 3 个小时（代码如下）：

```
ALTER SYSTEM SET undo_retention=10800 SCOPE=BOTH;
```

但请注意，增大这个参数可能会导致更多的 UNDO 表空间使用，以前在 Oracle 9iR2 中，这个参数的默认值一度被设为 10800 秒，可是随之而来的问题可能是 UNDO 表空间的过分扩展，不易回收，所以 Oracle 在不同版本中，也在进行不停地折中，不断尝试不同的初始化参数设置。Oracle 也许会这样想:如果很少有人使用 Flashback Query,而过大的 undo_retention 又会带来麻烦，那么干脆，设小点。

而根据我们的经验，在初始化创建数据库之后，undo_retention 的设置应该被修改，这一经验性调整应该被更新入安装手册，在数据库创建之后即刻进行。

然而在生产环境中修改这样一个参数需要谨慎，事实上，我们提醒 DBA 们要养成一些良好的习惯，如：

1. 尽量在测试后才在生产环境中执行某些维护操作。
2. 在生产环境空闲时才执行某些特定的维护操作。

在 RAC 环境中，修改某些数据库参数更须谨慎，有客户遇到了这样的问题，在 Oracle 10gR1 RAC 环境下修改 undo_retention 参数，使用如下命令：

```
alter system set undo_retention=18000 sid='*';
```

这条命令直接导致了 RAC 的其他节点挂起，Oracle 记录了一个相关 Bug，Bug 号为：4220405 ，Oracle 提示不指定具体 SID 的 UNDO_RETENTION 修改在该版本中不被支持，其 Workaround 就是，分别修改不同实例：

```
alter system set undo_retention=18000 sid='RAC1';
alter system set undo_retention=18000 sid='RAC2';
```

```
alter system set undo_retention=18000 sid='RAC3';
......
```

Oracle 声明在 Oracle Database 11g 中改正了以上这一问题，笔者曾经在 10.2.0.4 版本中测试过，证实已经不存在这个问题了。

这样一个简单的案例告诉我们，数据库的 Bug 可能无处不在，生产数据库调整应当极其谨慎，对于 DBA 来说，养成一些良好的工作习惯对于减少故障非常重要。

以下是一个简单的通过闪回查询恢复误删除数据的案例，这一特性应当为 DBA 所熟知。

某日下午接到研发工程师的电话，说误删除了部分重要数据，并且已经提交，要求恢复。登录到数据库上查看，由于是 Oracle9iR2，首先尝试使用 flashback query 闪回数据。

数据库运行在归档模式，可以首先通过 V$ARCHIVED_LOG 视图来确认数据库的 SCN 变化如下所示。

```
SQL> col fscn for 9999999999999999999
SQL> col nscn for 9999999999999999999
SQL> select name,FIRST_CHANGE# fscn,NEXT_CHANGE# nscn,FIRST_TIME from v$archived_log;
NAME                                 FSCN                 NSCN FIRST_TIME
------------------------------ ------------ -------------------- -------------------
/mwarch/oracle/1_52413.dbf      12929941968          12929942881 2005-06-22 14:38:28
/mwarch/oracle/1_52414.dbf      12929942881          12929943706 2005-06-22 14:38:32
/mwarch/oracle/1_52415.dbf      12929943706          12929944623 2005-06-22 14:38:35
/mwarch/oracle/1_52416.dbf      12929944623          12929945392 2005-06-22 14:38:38
/mwarch/oracle/1_52417.dbf      12929945392          12929945888 2005-06-22 14:38:41
/mwarch/oracle/1_52418.dbf      12929945888          12929945965 2005-06-22 14:38:44
/mwarch/oracle/1_52419.dbf      12929945965          12929948945 2005-06-22 14:38:45
/mwarch/oracle/1_52420.dbf      12929948945          12929949904 2005-06-22 14:46:05
/mwarch/oracle/1_52421.dbf      12929949904          12929950854 2005-06-22 14:46:08
/mwarch/oracle/1_52422.dbf      12929950854          12929951751 2005-06-22 14:46:11
/mwarch/oracle/1_52423.dbf      12929951751          12929952587 2005-06-22 14:46:14
..................
/mwarch/oracle/1_52498.dbf      12930138975          12930139212 2005-06-22 15:55:57
/mwarch/oracle/1_52499.dbf      12930139212          12930139446 2005-06-22 15:55:59
/mwarch/oracle/1_52500.dbf      12930139446          12930139682 2005-06-22 15:56:00
/mwarch/oracle/1_52501.dbf      12930139682          12930139915 2005-06-22 15:56:02
/mwarch/oracle/1_52502.dbf      12930139915          12930140149 2005-06-22 15:56:03
/mwarch/oracle/1_52503.dbf      12930140149          12930140379 2005-06-22 15:56:05
/mwarch/oracle/1_52504.dbf      12930140379          12930140610 2005-06-22 15:56:05
/mwarch/oracle/1_52505.dbf      12930140610          12930140845 2005-06-22 15:56:07
```

获得当前的 SCN 如下：

```
SQL> select dbms_flashback.get_system_change_number fscn from dual;
                FSCN
--------------------
         12930142214
```

使用应用用户尝试闪回

```
SQL> connect username/password
Connected.
```

现有数据为：

```
SQL> select count(*) from hs_passport;
  COUNT(*)
----------
    851998
```

创建恢复表如下：

```
SQL> create table hs_passport_recov as select * from hs_passport where 1=0;


Table created.
```

根据开发人员提供的大致误操作时间，结合 v$archived_log 视图，选择适当 SCN 向前执行闪回查询，代码如下：

```
SQL> select count(*) from hs_passport as of scn 12929970422;
  COUNT(*)
----------
    861686
```

尝试多个 SCN，获取最佳值（如果能得知具体时间，那么可以获得准确的数据闪回）：

```
SQL> select count(*) from hs_passport as of scn &scn;
Enter value for scn: 12929941968
  COUNT(*)
----------
    861684
SQL> /
Enter value for scn: 12927633776
select count(*) from hs_passport as of scn 12927633776
                     *
ERROR at line 1:
ORA-01466: unable to read data - table definition has changed


SQL> /
Enter value for scn: 12929928784
  COUNT(*)
----------
```

```
    825110

SQL> /
Enter value for scn: 12928000000
select count(*) from hs_passport as of scn 12928000000
                     *
ERROR at line 1:
ORA-01466: unable to read data - table definition has changed
```

最后选择恢复到 SCN 为 12929941968 的时间点，代码如下：

```
SQL> insert into hs_passport_recov select * from hs_passport as of scn 12929941968;
861684 rows created.
SQL> commit;
Commit complete.
```

然后由研发人员通过 hs_passport_recov 表确认，向当前表中补回误删除的数据，至此闪回恢复成功。如果没有闪回特性，这样的恢复就或者须要通过物理备份进行不完全恢复，或者找出足够及时的逻辑备份来进行恢复，其过程都可能是极其复杂的。

1.10 使用 ERRORSTACK 进行错误跟踪及诊断

在使用 Oracle 数据库的过程中，可能会遇到各种各样的错误或异常，很多异常的提示并不具体，我们有必要了解一下 Oracle 的 ErrorStack 跟踪方式。

ErrorStack 是 Oracle 提供的一种对于错误堆栈进行跟踪的方法，通过设置跟踪可以将一些错误的后台信息详尽地转储出来，写入跟踪文件，对于错误的研究与诊断非常有效。

设置 errorstack 主要有有 4 个级别：

0 仅转储错误堆栈（0 级已经被逐渐废弃）

1 转储错误堆栈和函数调用堆栈

2 Level 1 + ProcessState

3 Level 2 + Context area (显示所有 cursors，着重显示当前 cursor)

Errorstack 可以在实例级或会话级别设置，也可以在参数文件中设置，这个设置仅当某个特定的错误出现时才被触发，如设置 ORA-00942 事件的跟踪：

alter session set events '942 trace name errorstack level 1';

一个客户曾经出现如下 ORA-01438 错误，提示数据的精度超过允许值，是后台 Job 调度的任务：

```
Mon Jul 13 10:27:31 2009
Errors in file /admin/erpdb/bdump/erpdb1_j000_447020.trc:
ORA-12012: error on auto execute of job 22
ORA-01438: value larger than specified precision allowed for this column
```

```
ORA-06512: at "ERP.TIMRDU", line 13
ORA-06512: at line 1
```

跟踪文件中默认的不会记录具体的 SQL、绑定变量等信息，我们可以通过 ErrorStack 进行后台跟踪，获得更详细的信息，执行如下代码中 SQL：

```
alter system set events='1438 trace name errorstack forever,level 3';
```

然后可以手工执行出错的存储过程(代码如下)，获得跟踪文件，再关闭跟踪：

```
alter system set events='1438 trace name errorstack off';
```

在 Oracle 10g 中，这样的操作会被记录到日志文件中：

```
Mon Jul 13 10:48:39 2009
OS Pid: 541528 executed alter system set events '1438 trace name Errorstack forever,level 3'
Mon Jul 13 10:56:06 2009
Errors in file /admin/erpdb/udump/erpdb1_ora_267056.trc:
ORA-01438: value larger than specified precision allowed for this column
Mon Jul 13 10:56:08 2009
Trace dumping is performing id=[cdmp_20090713105608]
Mon Jul 13 10:57:15 2009
OS Pid: 541528 executed alter system set events '1438 trace name Errorstack off'
```

接下来分析获得的跟踪文件，就可以获得 SQL 文本线索，找到根本问题。在这个案例中，我们得到的跟踪文件，其关键 SQL 内容如下，通过这个 SQL 对照数据表很快就找到了精度超过的 Number 型字段：

```
*** SESSION ID:(857.16304) 2009-07-13 10:56:06.429
*** 2009-07-13 10:56:06.429
ksedmp: internal or fatal error
ORA-01438: value larger than specified precision allowed for this column
Current SQL statement for this session:
INSERT INTO CONTRAPAYM (IHCODE,GTICODE,IDX,HCODE,PORDATE,FCODE,
FCY,CCODE,ECODE,FLAG,MATCHFLAG,BCODE,MONTHZL,STATUS,AUTOFLAG ,REMARK) SELECT DISTINCT
IHCODE,'0000000000000000' GTICODE,(IDX+100) IDX,HCODE,:B3 PORDATE,FCODE,:B2 FCY,CCODE,ECODE,
FLAG,0 MATCHFLAG,BCODE,MONTHZL,STATUS,4 AUTOFLAG ,' Proc_AdjustContractpayment' REMARK FROM
CONTRAPAYM WHERE IHCODE=:B1 AND IDX=(SELECT MAX(IDX) FROM CONTRACTPAYMENT WHERE IHCODE=:B1 )
AND ROWNUM=1
----- PL/SQL Call Stack -----
  object      line  object
  handle    number  name
7000002ca366e80       100  procedure ERP.PROC_ADJCONTRAPAYM
700000336a1a070       236  procedure ERP.PROC_AUTOBATPROC
7000002ca367df0         5  procedure ERP.TIMRDU
700000342eb7c20         1  anonymous block
```

在跟踪文件中，还有大量的堆栈信息，对于复杂的问题，还可以通过进一步细致的堆栈

分析进行深入追踪。

可以很容易地测试这一功能的使用，比如使用如下代码段中的测试过程：

```
SQL>  alter system set events '984 trace name errorstack off';
SQL> connect eygle/eygle
SQL> create table t (name varchar2(10),id number);
Table created.
SQL> insert into t values(a,1);
insert into t values(a,1)
                     *
ERROR at line 1:
ORA-00984: column not allowed here
SQL> alter system set events '984 trace name errorstack off';
```

在报警日志文件中就可以获得如下信息：

```
Mon Jul 13 22:55:59 2009
OS Pid: 2431 executed alter system set events '984 trace name ERRORSTACK level 3'
Mon Jul 13 22:59:12 2009
Errors in file /opt/oracle/admin/mmstest/udump/mmstest_ora_2520.trc:
ORA-00984: column not allowed here
Mon Jul 13 23:01:01 2009
OS Pid: 2431 executed alter system set events '984 trace name errorstack off'
```

获得的跟踪文件里记录了 insert 的相关信息：

```
*** 2009-07-13 22:59:12.928
ksedmp: internal or fatal error
ORA-00984: column not allowed here
Current SQL statement for this session:
insert into t values(a,1)
----- Call Stack Trace -----
calling              call     entry               argument values in hex
location             type     point               (? means dubious value)
-------------------- -------- ------------------- ----------------------------
ksedst()+27          call     ksedst1()           0 ? 1 ?
```

1.1 断电故障导致 ASM DiskGroup 故障及恢复案例

ASM 在 RAC 环境中的使用已经极其广泛，但是往往由于对 ASM 的认识不够，很多时候在处理故障时会陷入谜团，前一段在客户环境中就遇到了一个 ASM 的棘手问题。

客户由于断电导致了存储故障，进而使得部分磁盘对主机不可见，某个磁盘组无法加载，

此时客户尝试过重启数据库，结果遇到了如下错误：

```
Thu Jun 25 05:00:11 2009
Errors in file /u01/app/oracle/admin/billing/udump/billing1_ora_8184.trc:
ORA-15062: ASM disk is globally closed
ORA-15025: could not open disk '/dev/rdsk/c12t0d2'
ORA-27041: unable to open file
HPUX-ia64 Error: 6: No such device or address
Additional information: 3
Thu Jun 25 05:00:29 2009
Errors in file /u01/app/oracle/admin/billing/udump/billing1_ora_8761.trc:
ORA-15062: ASM 磁盘已全局关闭
ORA-15025: 无法打开磁盘 '/dev/rdsk/c12t0d2'
ORA-27041: 无法打开文件
HPUX-ia64 Error: 6: No such device or address
Additional information: 3
Thu Jun 25 05:00:29 2009
Errors in file /u01/app/oracle/admin/billing/udump/billing1_ora_8759.trc:
ORA-15062: ASM disk is globally closed
ORA-15025: could not open disk '/dev/rdsk/c12t0d2'
ORA-27041: unable to open file
HPUX-ia64 Error: 6: No such device or address
Additional information: 3
```

注意在这个提示中，有一个重要提示：ORA-15062: ASM 磁盘已全局关闭。也就是说，由于磁盘无法访问，ASM 将磁盘在全局关闭，ASM 磁盘组也不可用。

如果强制打开数据库，Oracle 会将该磁盘组的所有文件 Offline 离线处理，然后 Open 数据库：

```
Thu Jun 25 06:10:41 2009
KCF: write/open error block=0xce0db online=1
     file=148 +DG_DATA_03/billing/datafile/tbs_table_20.256.654268217
     error=15081 txt: ''
Automatic datafile offline due to write error on
file 148: +DG_DATA_03/billing/datafile/tbs_table_20.256.654268217
KCF: write/open error block=0x72b online=1
     file=21 +DG_DATA_03/billing/datafile/tbs_idx_20.265.654273237
     error=15078 txt: ''
Automatic datafile offline due to write error on
file 21: +DG_DATA_03/billing/datafile/tbs_idx_20.265.654273237
KCF: write/open error block=0x4c6e9 online=1
     file=50 +DG_DATA_03/billing/datafile/tbs_dailytable_20.270.656595577
```

```
    error=15078 txt: ''
```

这就造成了进一步的一个现象，在数据库看来，始终有一个磁盘组处于 Mounted 的状态，数据库无法连接，也就无法访问其中的数据：

```
SQL> select name,state from v$asm_diskgroup_stat;
NAME                            STATE
------------------------------ -----------
DG_DATA_01                      CONNECTED
DG_DATA_02                      CONNECTED
DG_DATA_03                      MOUNTED
```

这其实很正常，Oracle 不访问该 DG 中的磁盘，该磁盘就保持了 MOUNTED 状态，我们只要尝试访问该磁盘中的文件，该磁盘组就会显示为数据库连接的 CONNECTED 状态。简单地通过如下 RMAN 的 COPY 命令就可激活该磁盘组的磁盘访问：

```
RMAN> copy datafile '+DG_DATA_03/billing/datafile/tbs_20.264.654269073' to '/backup/a.dbf';
```

接下来通过 Recover 那些被 Offline 的文件，再执行 Online 操作，就将数据库恢复到了正常状态。

1.2 共享池的改进与 ORA-04031 的变化

我们知道，从 Oracle 9i 开始，Shared Pool 可以被分割为多个子缓冲池（SubPool）进行管理，以提高并发性，减少竞争。

Shared Pool 的每个 SubPool 可以被看作是一个 Mini Shared Pool，拥有自己独立的 Free List、内存结构以及 LRU List。同时 Oracle 提供多个 Latch 对各个子缓冲池进行管理，从而避免单个 Latch 的竞争（Shared Pool Reserved Area 同样进行分割管理）。SubPool 最多可以有 7 个，Shared Pool Latch 也从原来的一个增加到现在的 7 个。如果系统有 4 个或 4 个以上的 CPU，并且 SHARED_POOL_SIZE 大于 250MB，Oracle 可以把 Shared Pool 分割为多个子缓冲池（SubPool）进行管理，在 Oracle 9i 中，每个 SubPool 至少为 128MB。

如果你看到过类似如下信息，那就意味着你可能遇到了 SubPool 的问题，如下所示：

```
Tue Dec 11 17:14:49 2007
Errors in file /oracle/app/admin/ctais2/udump/ctais2_ora_778732.trc:
ORA-04031: unable to allocate 4216 bytes of shared memory
("shared pool","IDX_DJ_NSRXX_P_NSRMCCTAIS2","sga heap(2,0)","library cache")
ORA-04031: unable to allocate 4216 bytes of shared memory
("shared pool","IDX_DJ_NSRXX_P_NSRMCCTAIS2","sga heap(2,0)","library cache")
Tue Dec 11 17:14:51 2007
Errors in file /oracle/app/admin/ctais2/bdump/ctais2_pmon_393248.trc:
ORA-04031: unable to allocate 4216 bytes of shared memory
("shared pool","unknown object","sga heap(2,0)","library cache")
```

Oracle 9i 中多个子缓冲池的结构如图 1-19 所示。

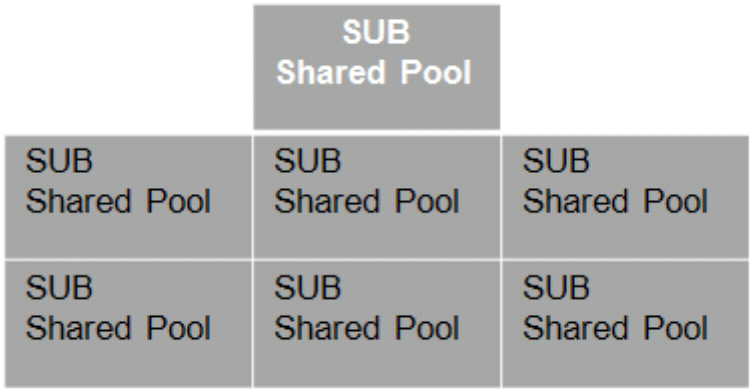


图 1-19 Oracle 9i 中多个子缓冲池的结构示意图

子缓冲池的数量受一个新引入的隐含参数**_KGHDSIDX_COUNT** 影响。可以手工调整该参数（仅限于试验环境研究用），观察共享池管理的变化，可以通过如下步骤转储默认情况以及修改后的 Shared Pool，再进行观察：

```
alter session set events 'immediate trace name heapdump level 2';
alter system set "_kghdsidx_count"=2 scope=spfile;
startup force;
alter session set events 'immediate trace name heapdump level 2';
```

以下是概要输出，注意在前者的跟踪文件中，sga heap(1,0)指共享池只存在一个子缓冲，后者则存在 sga heap(1,0)以及 sga heap(2,0)两个子缓冲池：

```
 [oracle@jumper udump]$ grep "sga heap" eygle_ora_25766.trc
HEAP DUMP heap name="sga heap"  desc=0x5000002c
HEAP DUMP heap name="sga heap(1,0)"  desc=0x5001ef0c
[oracle@jumper udump]$ grep "sga heap" eygle_ora_25786.trc
HEAP DUMP heap name="sga heap"  desc=0x5000002c
HEAP DUMP heap name="sga heap(1,0)"  desc=0x5001ef0c
HEAP DUMP heap name="sga heap(2,0)"  desc=0x50023c04
```

子缓冲池的分配的算法很简单：

- 每个子缓冲池必须满足一定的内存约束条件；
- 每 4 颗 CPU 可以分配一个子缓冲池，子缓冲池的数量最多 7 个。

在 Oracle 9i 中，每个 SubPool 容量至少 128MB，而在 Oracle 10g 中，每个子缓冲池至少为 256MB。如前所述，SubPool 的数量可以通过_kghdsidx_count 参数来控制，但是没有参数可以显式地控制 SubPool 的大小。

根据以上规则，在一个 12 颗 CPU 的系统中，如果分配容量为 300MB 的 Shared Pool，Oracle 9i 将创建两个 SubPool，每个容量大约 150MB，如果共享池容量增加到 500MB，Oracle 将创建 3 个 SubPool，每个大约 166MB。

不管 Oracle 9i 中的 128MB 以及 Oracle10g 中的 256MB，这样的 SubPool 在许多复杂的系统中，都可能是过小的，在这些情况下，可能要增大 SubPool。可以通过控制 Shared Pool 大

小以及 SubPool 的数量来改变 SubPool 的大小。一些 Bug 以及内部测试表明 500MB 的 SubPool 可能会带来更好的性能，所以从 Oracle 11g 开始，每个 SubPool 至少为 512MB。

除大小控制之外，在 Oracle 10g 中，Oracle 仍然对共享池的管理做出了进一步改进，那就是对单个子缓冲池进行进一步的细分。现在默认，Oracle 10g 会将单个缓冲池分割为 4 个子分区进行管理(这可能是因为通常 4 颗 CPU 才分配一个 SubPool)，使用类似如上的方法在 Oracle 10gR2 中进行测试：

```
alter session set events 'immediate trace name heapdump level 2';
alter system set "_kghdsidx_count"=2 scope=spfile;
startup force;
alter session set events 'immediate trace name heapdump level 2';
```

分析得到的日志，当仅有一个子缓冲池时，Shared Pool 被划分为 sga heap(1,0)～sga heap(1,3)共 4 个子分区：

```
 [oracle@eygle udump]$ grep "sga heap" eygle_ora_13577.trc
HEAP DUMP heap name="sga heap"  desc=0x2000002c
HEAP DUMP heap name="sga heap(1,0)"  desc=0x2001b550
HEAP DUMP heap name="sga heap(1,1)"  desc=0x2001c188
HEAP DUMP heap name="sga heap(1,2)"  desc=0x2001cdc0
HEAP DUMP heap name="sga heap(1,3)"  desc=0x2001d9f8
```

当使用两个子缓冲池时，Shared Pool 则被划分为 8 个子分区进行管理如下：

```
 [oracle@eygle udump]$ grep "sga heap" eygle_ora_13618.trc
HEAP DUMP heap name="sga heap"  desc=0x2000002c
HEAP DUMP heap name="sga heap(1,0)"  desc=0x2001b550
HEAP DUMP heap name="sga heap(1,1)"  desc=0x2001c188
HEAP DUMP heap name="sga heap(1,2)"  desc=0x2001cdc0
HEAP DUMP heap name="sga heap(1,3)"  desc=0x2001d9f8
HEAP DUMP heap name="sga heap(2,0)"  desc=0x20020640
HEAP DUMP heap name="sga heap(2,1)"  desc=0x20021278
HEAP DUMP heap name="sga heap(2,2)"  desc=0x20021eb0
HEAP DUMP heap name="sga heap(2,3)"  desc=0x20022ae8
```

Oracle 10g 中多缓冲池结构如图 1-20 所示。

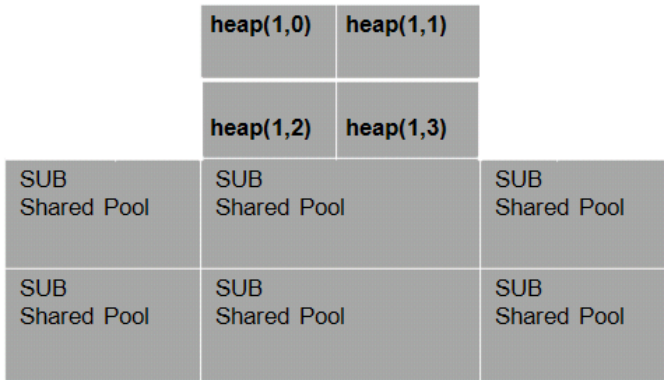


图 1-20 Oracle 10g 中多缓冲池结构示意图

通过一个内部表 X$KGHLU（[K]ernel [G]eneric memory [H]eap manager State of [L]R[U] Of Unpinned Recreatable chunks）可以查询这些子缓冲池的分配：

```
SQL> select addr,indx,kghluidx,kghludur,kghluops,kghlurcr from x$kghlu;
ADDR      INDX   KGHLUIDX   KGHLUDUR   KGHLUOPS   KGHLURCR
--------  ----  ----------  ---------  ---------  ---------
B5F4C5B4     0           2          3      12773        257
B5F4C1AC     1           2          2      43675       1042
B5F4D9C8     2           2          1      18831       1518
B5F4D5C0     3           2          0          0          0
B5F4D1B8     4           1          3     144697        327
B5F4E9E4     5           1          2     483428       1462
B5F4E5DC     6           1          1       6558        982
B5F4E1D4     7           1          0          0          0
8 rows selected.
```

通过这一系列的算法改进，Oracle 中 Shared Pool 管理得以不断增强，较好地解决了大 Shared Pool 的性能问题；Oracle 8i 中，过大 Shared Pool 设置可能带来的栓锁争用等性能问题在某种程度上得以解决。从 Oracle 10g 开始，Oracle 开始提供自动共享内存管理，使用该特性，用户可以不必显示设置共享内存参数，Oracle 会自动进行分配和调整，虽然 Oracle 为用户提供了极大的便利，但是了解自动化后面的原理对于理解 Oracle 的运行机制仍然是十分重要的。

虽然多缓冲池技术使 Oracle 可以管理更大的共享池，但是 SubPool 的划分可能也会导致各分区之间的协调问题，甚至可能因为内存分散而出现 ORA-04031 错误。最常见的问题是某个子缓冲池（SubPool）可能出现过度使用，当新的进程仍然被分配到这个 SubPool 时，可能会导致内存请求失败（而此时其他 SubPool 可能还有很多内存空间）。

因为子缓冲池存在的种种问题，从 Oracle 10g 开始，允许内存请求在不同 SubPool 之间进行切换（Switch），从而提高了请求成功的可能（但是显然切换不可能是无限制的，所以问题仍然可能存在）。

以下是来自客户系统的一个实际案例，在一个 Oracle9i 的系统中，经常出现 ORA-04031

的错误，客户系统的主要配置如下：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle9i Enterprise Edition Release 9.2.0.6.0 - 64bit Production
SQL> show parameter cpu_count
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
cpu_count                            integer     48
SQL> select * from v$sga;
NAME                                     VALUE
-------------------------------- ----------------
Fixed Size                             762240
Variable Size                         2600468480
Database Buffers                     18975031296
Redo Buffers                           6578176
```

我们检查其参数设置，默认的子池设置是 7 个，代码如下：

```
SQL> select a.ksppinm, b.ksppstvl from   x$ksppi a, x$ksppsv b
  2  where  a.indx = b.indx and a.ksppinm = '_kghdsidx_count';
KSPPINM                                                          KSPPSTVL
---------------------------------------------------------------- --------------------
_kghdsidx_count                                                  7
```

7 个子池都被使用，其 Latch 使用情况如下：

```
SQL> select child#, gets from v$latch_children
  2  where name = 'shared pool' order by child#;
    CHILD#       GETS
---------- ----------
         1  333403016
         2  355720323
         3  273944301
         4  197980497
         5  282347697
         6  354398593
         7  468809111
```

看一下具体的子池使用及内存情况，注意到各个 Shared Pool 子池平均分配了 320MB 内存左右，共享池合计约 2256MB：

```
SELECT        'shared pool (' || NVL (DECODE (TO_CHAR (ksmdsidx), '0', '0 - Unused',
ksmdsidx),'Total') || '):' subpool,
        SUM (ksmsslen) BYTES, ROUND (SUM (ksmsslen) / 1048576, 2) mb
```

```
    FROM x$ksmss WHERE ksmsslen > 0
GROUP BY ROLLUP (ksmdsidx) ORDER BY subpool ASC
/
SUBPOOL                              BYTES         MB
------------------------------ ---------- ----------
shared pool (1):                 352321536        336
shared pool (2):                 335544320        320
shared pool (3):                 335544320        320
shared pool (4):                 335544320        320
shared pool (5):                 335544320        320
shared pool (6):                 335544320        320
shared pool (7):                 335544320        320
shared pool (Total):            2365587456       2256

8 rows selected.
```

进一步可以查询一下各个子池的剩余内存，注意到各个子池剩余内存约在 7MB~15MB 之间，而这些剩余内存又可能是零散的碎片：

```
SELECT   subpool, NAME, SUM (BYTES), ROUND (SUM (BYTES) / 1048576, 2) mb
    FROM (SELECT    'shared pool (' ||DECODE (TO_CHAR (ksmdsidx), '0', '0 - Unused', ksmdsidx)
                || '):' subpool, ksmssnam NAME, ksmsslen BYTES
            FROM x$ksmss WHERE ksmsslen > 0 AND LOWER (ksmssnam) LIKE LOWER ('%free memory%'))
GROUP BY subpool, NAME ORDER BY subpool ASC, SUM (BYTES) DESC
/
SUBPOOL                        NAME                            SUM(BYTES)         MB
------------------------------ ------------------------------ ---------- ----------
shared pool (1):               free memory                        8158640       7.78
shared pool (2):               free memory                        7414472       7.07
shared pool (3):               free memory                        7831608       7.47
shared pool (4):               free memory                       10690992       10.2
shared pool (5):               free memory                       17201856       16.4
shared pool (6):               free memory                        8239920       7.86
shared pool (7):               free memory                       13925416      13.28
```

通过以下查询可以详细列举不同子池的 Free 内存块情况，从输出可以观察到，每个子池大于 10KB 的内存块都很少，这也就意味着，当有大块的共享内存请求时就可能出现 ORA-04031 错误（注意：R-free 指保留池的剩余空间）：

```
SQL> SELECT   ksmchidx "SubPool", 'sga heap(' || ksmchidx || ',0)' sga_heap,
  2           ksmchcom chunkcomment,
  3           DECODE (ROUND (ksmchsiz / 1000),
  4                   0, '0-1K', 1, '1-2K',2, '2-3K',3, '3-4K',4, '4-5K', 5, '5-6k',
```

```
  5                   6, '6-7k', 7, '7-8k',8, '8-9k',9, '9-10k', '> 10K'
  6                  ) "size",
  7           COUNT (*), ksmchcls status, SUM (ksmchsiz) BYTES
  8      FROM x$ksmsp WHERE ksmchcom = 'free memory'
  9  GROUP BY ksmchidx, ksmchcls, 'sga heap(' || ksmchidx || ',0)',ksmchcom, ksmchcls,
 10           DECODE (ROUND (ksmchsiz / 1000),
 11                   0, '0-1K', 1, '1-2K',2, '2-3K',3, '3-4K',4, '4-5K', 5, '5-6k',
 12                   6, '6-7k', 7, '7-8k',8, '8-9k',9, '9-10k', '> 10K' );
SUBPOOL SGA_HEAP            CHUNKCOMMENT     size    COUNT(*) STATUS        BYTES
------- ------------------- ---------------- ----- ---------- -------- ----------
      1 sga heap(1,0)       free memory      0-1K        5173 free         922568
      1 sga heap(1,0)       free memory      1-2K        5422 free        5274920
.........
      1 sga heap(1,0)       free memory      6-7k           2 R-free        11968
      1 sga heap(1,0)       free memory      7-8k           9 R-free        62096
      1 sga heap(1,0)       free memory      8-9k          12 R-free        95480
      1 sga heap(1,0)       free memory      9-10k         11 R-free        99192
      1 sga heap(1,0)       free memory      > 10K         25 R-free       434272
      2 sga heap(2,0)       free memory      0-1K        4919 free         848864
.......
      2 sga heap(2,0)       free memory      9-10k          5 R-free        46056
      2 sga heap(2,0)       free memory      > 10K         43 R-free       769144
      3 sga heap(3,0)       free memory      0-1K        6921 free        1058264
。。。。。。
      3 sga heap(3,0)       free memory      9-10k          9 R-free        81344
      3 sga heap(3,0)       free memory      > 10K         64 R-free      1212424
      4 sga heap(4,0)       free memory      0-1K        6430 free         928688
.......
      4 sga heap(4,0)       free memory      9-10k          9 R-free        80464
      4 sga heap(4,0)       free memory      > 10K         34 R-free       689640
      5 sga heap(5,0)       free memory      0-1K        4416 free         779096
......
      5 sga heap(5,0)       free memory      9-10k          4 R-free        36344
      5 sga heap(5,0)       free memory      > 10K         40 R-free      1669384
      6 sga heap(6,0)       free memory      0-1K        6203 free         863104
。。。。。。
      6 sga heap(6,0)       free memory      9-10k         11 R-free        99464
      6 sga heap(6,0)       free memory      > 10K         56 R-free      1758912
      7 sga heap(7,0)       free memory      0-1K        3814 free         607616
```

```
……
      7 sga heap(7,0)       free memory      9-10k          6 R-free        54432
      7 sga heap(7,0)       free memory      > 10K         52 R-free      2816480

120 rows selected.
```

针对这种情况，我们可以相应减少 Shared Pool 子池的数量，以使得每个子池可以有足够的空闲内存可用。在这个客户环境中，首先将_kghdsidx_count 调整为 3，ORA-04031 错误即没有再次出现，调整之后，每个子池的内存扩大到 750MB 左右：

```
SUBPOOL                              BYTES         MB
------------------------------ ---------- ----------
shared pool (1):                 788529152        752
shared pool (2):                 788529192        752
shared pool (3):                 771751936        736
shared pool (Total):            2348810280       2240
```

现在每个子池的空闲内存达到了 20MB~60MB 左右：

```
SUBPOOL                        NAME                      SUM(BYTES)         MB
------------------------------ ------------------------- ---------- ----------
shared pool (1):               free memory                 56014080      53.42
shared pool (2):               free memory                 20292704      19.35
shared pool (3):               free memory                 67884912      64.74
```

调整后具体的内存使用情况如下，我们注意到，保留池的大块的空闲内存（R-free）数量大大增加，这样在要请求大块内存时，就更容易获得共享内存资源：

```
SUBPOOL SGA_HEAP         CHUNKCOMMENT     size    COUNT(*) STATUS         BYTES
------- ---------------- ---------------- ------ ---------- -------- ----------

     。。。。。。
      1 sga heap(1,0)     free memory      8-9k           6 free          48016
      1 sga heap(1,0)     free memory      > 10K          4 free          45448
  。。。。。。
      1 sga heap(1,0)     free memory      9-10k         22 R-free       197536
      1 sga heap(1,0)     free memory      > 10K        144 R-free      2606992
 。。。。。。
      2 sga heap(2,0)     free memory      9-10k          8 free          72784
      2 sga heap(2,0)     free memory      > 10K         15 free         172616
……
      2 sga heap(2,0)     free memory      9-10k         22 R-free       195280
      2 sga heap(2,0)     free memory      > 10K        155 R-free      2839248
。。。。。。
      3 sga heap(3,0)     free memory      8-9k          14 free         111736
```

```
      3 sga heap(3,0)     free memory       9-10k          1 free           8808
。。。。。。
      3 sga heap(3,0)     free memory       9-10k         29 R-free       261272
      3 sga heap(3,0)     free memory       > 10K        186 R-free      3434512
```

客户的系统是一个双节点 RAC 环境，在运行中，应用设置为只连接其中的一个节点，另外一个空闲节点的 Shared Pool 使用情况如下，列举供参考：

```
SUBPOOL SGA_HEAP         CHUNKCOMMENT      size    COUNT(*) STATUS        BYTES
------- ---------------- ----------------- ------ ---------- -------- ----------
      1 sga heap(1,0)    free memory       0-1K          373 free          41144
      1 sga heap(1,0)    free memory       1-2K            1 free           1488
      1 sga heap(1,0)    free memory       2-3K            1 free           1936
      1 sga heap(1,0)    free memory       3-4K            1 free           2704
      1 sga heap(1,0)    free memory       4-5K            1 free           3776
      1 sga heap(1,0)    free memory       9-10k           4 free          34864
      1 sga heap(1,0)    free memory       > 10K         157 free      460271664
      1 sga heap(1,0)    free memory       > 10K          38 R-free     25520800
      2 sga heap(2,0)    free memory       0-1K          357 free          37376
      2 sga heap(2,0)    free memory       3-4K            2 free           6152
      2 sga heap(2,0)    free memory       4-5K            1 free           3776
      2 sga heap(2,0)    free memory       > 10K         130 free      454592888
      2 sga heap(2,0)    free memory       > 10K          38 R-free     25520800
      3 sga heap(3,0)    free memory       0-1K          425 free          51280
      3 sga heap(3,0)    free memory       3-4K            1 free           2704
      3 sga heap(3,0)    free memory       7-8k            1 free           6664
      3 sga heap(3,0)    free memory       > 10K          44 free      467930312
      3 sga heap(3,0)    free memory       > 10K          38 R-free     25520800
```

ORA-04031 出现时，可能共享池没有足够空闲内存，但是 Shared Pool 保留池（shared_pool_reserved_size）还有一定的内存空闲，所以我们可以释放降低使用保留池的内存大小，在这个案例中，降低_shared_pool_reserved_min_alloc 参数设置，也帮助数据库更好地利用了保留内存。

1.3 共享内存无法正常释放的处理

在数据库启动之后，需要从操作系统上分配共享内存和信号量（Semaphore）资源，而在某些情况下，数据库异常关闭后，这些资源有可能无法正常释放，则在下次启动时，数据库可能遭遇错误，无法正常启动。

在一个客户环境（操作系统为 SUN Solaris 平台）中出现了 ORA-04031 错误之后，使用了 shutdown abort 选项关闭了数据库：

```
Sun Jul 19 19:20:23 2009
Errors in file /ora9i/oracle/admin/ora9i/bdump/ora9i1_j000_14337.trc:
ORA-12012: error on auto execute of job 721
ORA-04031: unable to allocate 4088 bytes of shared memory ("shared pool","STANDARDSYS","PL/SQL
MPCODE","BAMIMA: Bam Buffer")
Sun Jul 19 19:24:24 2009
Errors in file /ora9i/oracle/admin/ora9i/bdump/ora9i1_j000_17497.trc:
ORA-12012: error on auto execute of job 721
ORA-04031: unable to allocate 4088 bytes of shared memory ("shared pool","STANDARDSYS","PL/SQL
MPCODE","BAMIMA: Bam Buffer")
Sun Jul 19 19:26:56 2009
Shutting down instance (abort)
License high water mark = 1012
Instance terminated by USER, pid = 19157
```

接下来的尝试启动失败，告警日志记录了如下所示的错误信息：

```
Sun Jul 19 19:35:27 2009
Errors in file /ora9i/oracle/admin/ora9i/udump/ora9i1_ora_25055.trc:
ORA-27154: post/wait create failed
ORA-27300: OS system dependent operation:semget failed with status: 28
ORA-27301: OS failure message: No space left on device
ORA-27302: failure occurred at: sskgpsemsper
```

注意以上信息中，提示失败的操作是 semget，其实如果仔细阅读这个提示，就可以获知错误的真正原因，在操作系统可以通过 man semget 查看手册，了解出错的真实原因。以下输出告诉我们，semget 的任务是获得信号量集（get set of semaphores），所以也就可以知道之前的提示 No space left on device 并不是指存储空间，而是信号量资源：

```
oracle@db-server # man semget
NAME
     semget - get set of semaphores
SYNOPSIS
     #include <sys/types.h>
     #include <sys/ipc.h>
     #include <sys/sem.h>
     int semget(key_t key, int nsems, int semflg);
DESCRIPTION
     The semget() function returns the semaphore identifier associated with key.
```

知道了这个原因以后，就可以通过 ipcs 命令来找到数据库的信号量资源占用，该主机上有三个数据库，Oracle 用户的信号量集就是没有正常释放的那个：

```
oracle@db-server # ipcs -sa
IPC status from <running system> as of Sun Jul 19 22:01:09 CST 2009
```

```
T      ID      KEY         MODE         OWNER GROUP  CREATOR   CGROUP NSEMS   OTIME    CTIME
Semaphores:
s  851977   0x462fcb40 --ra-r----- orabil9 dbabil9  orabil9  dbabil9   604 22:07:58 12:54:47
s 1769482   0x6b997db0 --ra-r----- oracle9    dba9  oracle9     dba9  2004 22:07:15 17:03:39
s 1245195   0x8ea0d4d4 --ra-ra----  oracle   dba   oracle      dba  2004 no-entry 19:27:06
```

接下来就可以通过操作系统的 ipcrm 命令来移除信号量集：

```
oracle@db-server # man ipcrm
NAME
       ipcrm - remove a message queue, semaphore set or shared memory id
```

执行过程如下所示：

```
oracle@db-server # ipcrm -s 1245195
oracle@db-server # ipcs -sa
IPC status from <running system> as of Sun Jul 19 22:09:01 CST 2009
T       ID       KEY          MODE          OWNER     GROUP   CREATOR    CGROUP NSEMS    OTIME
CTIME
Semaphores:
s  851977   0x462fcb40 --ra-r----- orabil9 dbabil9  orabil9  dbabil9   604 22:07:58 12:54:47
s 1769482   0x6b997db0 --ra-r----- oracle9    dba9  oracle9     dba9  2004 22:07:15 17:03:39
```

完成这个操作之后，数据库就可以正常启动了。

这个案例告诉我们的是，如果采用特殊手段来进行维护和管理，那么必须清楚这些操作可能带来的影响和后果。

1.4 Log_buffer 设置与变迁

在 Oracle 10g 引入了自动的 SGA 调整之后，Oracle 的内存管理变得简单了，但是，很多 Buffer 的技术实现、管理等都发生了重大变化，这也包括 Log_buffer。

在 Oracle 10g 之前，该参数的默认设置为 Max（512 KB ，128 KB * CPU_COUNT），按照默认设置，log_buffer 消耗的内存都不会太高，而由于 LGWR 对 Log Buffer 内容的写出非常频繁，所以很小的 Log Buffer 也可以工作得很好，根据经验，有很多对 Log Buffer 的指导性设置，比如经常提到的 3MB 大小，但是在 Oracle10g 中，Redo Log Buffer 默认值已经大大超过了原来的想象。

这和 Oracle 9i 引入了 Granule 的概念有关，在动态 SGA 管理中，Granule 是最小的内存分配单元，其大小与 SGA 及操作系统平台有关。在 Oracle10g 中，Oracle 的内存分配会为'Fixed SGA Size'和'Redo Buffers'共享整数倍个 Granule。看看不同 SGA 设置下的内存分配情况：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
```

```
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - Prod
SQL> select * from v$sgainfo where name in ('Fixed SGA Size','Redo Buffers','Granule Size');
NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                       1267212 No
Redo Buffers                        15507456 No
Granule Size                        16777216 No
SQL> select sum(bytes)/1024/1024 from v$sgainfo where name in ('Fixed SGA Size','Redo
Buffers');
SUM(BYTES)/1024/1024
--------------------
            15.99757
```

而在另外一个生产系统中，共分配了 2 个 Granule，每个 4MB。

```
 SQL> select * from v$sgainfo where name in ('Fixed SGA Size','Redo Buffers','Granule Size');
NAME                                  BYTES RES
-------------------------------- ---------- ---
Fixed SGA Size                       1263320 No
Redo Buffers                         7122944 No
Granule Size                         4194304 No
SQL> select sum(bytes)/1024/1024 from v$sgainfo where name in ('Fixed SGA Size','Redo
Buffers');
SUM(BYTES)/1024/1024
--------------------
          7.99776459
```

1.5　Logmnr 简单而强大的工具

在 Oracle 数据库中，LGWR 进程将数据库中进行的 DML 等操作信息记录在日志文件中，在归档模式下，日志文件还会写出到归档日志文件中。在数据库发生故障崩溃后，恢复时数据库可以根据日志信息来重演事务，完成恢复，从而保证成功提交的事务不丢失。

很多第三方工具以及 Oracle 的一些工作，都可以通过解析日志文件来实现数据复制和同步。Oracle 随数据库软件提供了一个 Logmnr 工具，可以很容易实现对于日志的解析，熟悉 Logmnr 的使用在很多时候可以帮助我们分析数据库问题，找出根本原因。

从 Oracle9i 开始，LOGMNR 的使用大大简化，可以使用 LOGMNR 在线分析和挖掘日志，使用当前在线的数据字典，非常方便，以下是一个简要的测试和说明。

首先执行一个 DDL（或 DML）操作，以记录重做信息：

```
SQL> connect eygle/eygle
Connected.
```

```
SQL> alter system switch logfile;
System altered.
SQL> create table eygle as select * from dba_users;
Table created.
SQL> select count(*) from eygle;
  COUNT(*)
----------
        19
```

然后可以执行 LOGMNR 解析工作：

```
SQL> connect / as sysdba
Connected.
SQL> select * from v$log where status='CURRENT';
GROUP#    THREAD#  SEQUENCE#      BYTES    MEMBERS ARC STATUS   FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- -------------
     2          1         100   52428800          1 NO  CURRENT     12729697 01-JUL-09
SQL> SELECT MEMBER from v$logfile where group#=2;
MEMBER
--------------------------------------------------------------------------------
/opt/oracle/oradata/mmstest/redo02.log
SQL>exec
dbms_logmnr.add_logfile('/opt/oracle/oradata/mmstest/redo02.log',dbms_logmnr.new);
PL/SQL procedure successfully completed.
SQL> exec dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);
PL/SQL procedure successfully completed.
SQL> select count(*) from v$logmnr_contents;
  COUNT(*)
----------
       136
```

解析之后，就可以通过 v$logmnr_contents 视图来查出数据库执行所有操作，以下查询出来 SQL_REDO，通过这些 SQL 就可以重演 CREATE TABLE 的 DDL 操作，通过以下的重做信息也可以看到，DDL 的后台操作实际上是转换为对字典表的一系列 DML 操作。

```
SQL> select sql_redo from v$logmnr_contents;
SQL_REDO
--------------------------------------------------------------------------------
set transaction read write;
insert      into      "SYS"."OBJ$"("OBJ#","DATAOBJ#","OWNER#","NAME","NAMESPACE","SUBNAME",
"TYPE#","CTIME","MTIME","STIME","STATUS","REMOTEOWNER","LINKNAME","FLAGS","OID$","SPARE1"
,"SPARE2","SPARE3","SPARE4","SPARE5","SPARE6")     values     ('25847','25847','31','EYGLE',
'1',NULL,'2',TO_DATE('01-JUL-09','DD-MON-RR'),TO_DATE('01-JUL-09','DD-MON-RR'),TO_DATE('0
```

```
1-JUL-09', 'DD-MON-RR'),'1',NULL,NULL,'0',NULL,'6','1',NULL,NULL,NULL,NULL);

set transaction read write;
update "SYS"."CON$" set "CON#" = '10823' where "CON#" = '10822' and ROWID =
'AAAAAcAABAAAACqAAM';
commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10824' where "CON#" = '10823' and ROWID =
'AAAAAcAABAAAACqAAM';
commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10825' where "CON#" = '10824' and ROWID =
'AAAAAcAABAAAACqAAM';
commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10826' where "CON#" = '10825' and ROWID =
'AAAAAcAABAAAACqAAM';
commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10827' where "CON#" = '10826' and ROWID =
'AAAAAcAABAAAACqAAM';

commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10828' where "CON#" = '10827' and ROWID =
'AAAAAcAABAAAACqAAM';

commit;
set transaction read write;
update "SYS"."CON$" set "CON#" = '10829' where "CON#" = '10828' and ROWID =
'AAAAAcAABAAAACqAAM';

commit;
create table eygle as select * from dba_users;
set transaction read write;
```

查询完成之后，可以通过如下命令结束日志解析过程：

```
SQL> exec dbms_logmnr.end_logmnr
PL/SQL procedure successfully completed.
```

熟悉和熟练使用 LOGMNR 可以帮助我们解决很多棘手的问题，并加深对于数据库的理

解。

1.6 从数据字典中获得更多知识

Oracle 的数据字典记录了数据库管理的元数据，对于数据库来说是生死攸关的“核心档案”，认真理解数据字典的内容不仅有助于加深对于数据库管理的认识，也有助于理解很多数据库运行的机制，接下来我们通过几则数据库管理与学习中和数据字典的交互，向大家介绍如何通过数据字典获得更多的数据库知识。

1. sql.bsq 文件数据字典

要了解数据字典需要关注的一个重要文件是$ORACLE_HOME/rdbms/admin/sql.bsq，这个文件在数据库创建时（CREATE DATABASE）被调用，用于创建一系列的内部数据字典，该文件的调用由一个隐含参数设定：

```
SQL> SELECT x.ksppinm NAME, y.ksppstvl VALUE, x.ksppdesc describ
  2    FROM SYS.x$ksppi x, SYS.x$ksppcv y
  3   WHERE x.indx = y.indx AND x.ksppinm ='_init_sql_file';
NAME            VALUE                    DESCRIB
--------------- ------------------------ ------------------------------------------------
_init_sql_file  ?/rdbms/admin/sql.bsq  File containing SQL statements to execute
                                       upon database creation
```

在这个文件中，我们可以找到很多关于数据库重要特性的实现方法，比如和闪回特性相关的重要数据表 SMON_SCN_TIME，这里就有详细的说明与注释。如图 1-21 中注释提到：SMON 维护这个表的信息，用于创建基于 SCN 到时间的映射关系。该表可以存储 144000 条记录，保留 5 天，最快可以每 3 秒记录一次信息：

```
rem create the scn<->time tracking table that smon will maintain
rem as a circular queue - notice that we populate the entire
rem table with at least 144000 entries (enough for 5 days).
rem
rem -"thread" is for backward compatibility and is always 0
rem -"orig_thread" is for upgrade/downgrade
rem - scn_wrp, scn_bas, and time_dp are for backward compatibility
rem   and not queried by the ktf layer.
rem

create cluster smon_scn_to_time (
  thread number                          /* thread, compatibility */
)
/
create index smon_scn_to_time_idx on cluster smon_scn_to_time
/
create table smon_scn_time (
  thread number,                          /* thread, compatibility */
  time_mp number,                         /* time this recent scn represents */
  time_dp date,                           /* time as date, compatibility */
  scn_wrp number,                         /* scn.wrp, compatibility */
  scn_bas number,                         /* scn.bas, compatibility */
  num_mappings number,
  tim_scn_map raw(1200),
  scn number default 0,                   /* scn */
  orig_thread number default 0            /* for downgrade */
) cluster smon_scn_to_time (thread)
/

create unique index smon_scn_time_tim_idx on smon_scn_time(time_mp)
/

create unique index smon_scn_time_scn_idx on smon_scn_time(scn)
/
```

图 1-21 sql.bsq 文件数据库字典

这里的说明及创建过程是非常详尽的，如果对比 Oracle 9i 的相关部分，我们会注意到这个表的内容被极大地扩充，9i 只能维护最快每 5 分钟一次记录，这也决定了闪回的最小时间间隔。

进一步，这个表如果只存储 5 天的数据，Oracle 是如何来选择覆盖和放弃过期数据的呢？在 Oracle Database 10g 的一个 Bug 异常中，我们可以观察到这个内部的操作。

在我们的一个客户系统（Oracle 10g 10.2.0.3 版本）中曾经遇到过这样一个错误：

ORA-1461 ENCOUNTERED WHEN GENERATING SERVER ALERT SMG-3500

这个错误是一个 Bug 导致的，根据 Metalink Note 461911.1 记载，这个 Bug 仅发生在 Oracle Database 10.2.0.3 版本中。具体的错误是：

ORA-01461: can bind a LONG value only for insert into a LONG column

在后台的跟踪文件里，可以找到错误出现的详细 SQL，这个 SQL 是对 smon_scn_time 表执行 update 操作：

```
ORA-01461: can bind a LONG value only for insert into a LONG column
```

```
Current SQL statement for this session:
update smon_scn_time set orig_thread=0, time_mp=:1, time_dp=:2, scn=:3,
scn_wrp=:4, scn_bas=:5, num_mappings=:6, tim_scn_map=:7 where thread=0 and
scn = (select min(scn) from smon_scn_time where thread=0)
```

我们注意到，这个 SQL 正是通过寻找最小的 SCN，获得历史最久的记录，对其进行更新以完成覆盖和数据舍弃的，在 Oracle 学习过程中，经常会有这样非常有趣的过程，曲曲折折的我们可以发现数据库的很多内部工作机制。

对于这个 Bug，Patch 6602742 修正了它。这个 Bug 并不会经常出现，对数据库没有太大影响，如果有充足停机时间，可以考虑应用这个小 Patch，否则也可以予以忽略。

2. OBJECT_ID、DATA_OBJECT_ID 与 Truncate 的本质

在熟悉 DBA_OBJECTS 字典视图时，很多朋友曾经提出过这样一个问题，那就是 OBJECT_ID 和 DBA_OBJECT_ID 有何区别：

```
SQL> desc dba_objects
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 OWNER                                              VARCHAR2(30)
 OBJECT_NAME                                        VARCHAR2(128)
 SUBOBJECT_NAME                                     VARCHAR2(30)
 OBJECT_ID                                          NUMBER
 DATA_OBJECT_ID                                     NUMBER
.....
```

两者可以这样区别：OBJECT_ID 可以看做是对象的一个逻辑 ID，在对象创建时分配，一经分配即不再改变；而 DATA_OBJECT_ID 则是一个物理 ID，在对象物理存储发生变化时可能会发生改变。在 sql.bsq 文件中，我们可以看到如图 1-22 中一段注释说明（OBJ$是 DBA_OBJECTS 的底层数据表），OBJ$表的 DATAOBJ#为 Data Layer 的对象号，Oracle 提示不要在该字段上创建索引，因为该标识会在 TRUNCATE 时发生改变：

```
rem NOTE
rem Logminer/Streams uses contents of this table.
rem Please do not reuse any flags without verifying the impact of your
rem changes on inter-op.
create table obj$                                                /* object table */
( obj#          number not null,                                /* object number */
  /* DO NOT CREATE INDEX ON DATAOBJ#  AS IT WILL BE UPDATED IN A SPACE
   * TRANSACTION DURING TRUNCATE */
  dataobj#      number,                             /* data layer object number */
  owner#        number not null,                          /* owner user number */
  name          varchar2("M_IDEN") not null,                /* object name */
  namespace     number not null,          /* namespace of object (see KQD.H): */
 /* 1 = TABLE/PROCEDURE/TYPE, 2 = BODY, 3 = TRIGGER, 4 = INDEX, 5 = CLUSTER, */
                                                  /* 8 = LOB, 9 = DIRECTORY, */
  /* 10 = QUEUE, 11 = REPLICATION OBJECT GROUP, 12 = REPLICATION PROPAGATOR, */
                                       /* 13 = JAVA SOURCE, 14 = JAVA RESOURCE */
                                                  /* 58 = (Data Mining) MODEL */
```

图 1-22 sql.bsq 文件数据库字典中 OBJ$的注释

这个注释道出了 TRUNCATE 操作的本质，我们知道 TRUNCATE 操作可以很快速地清除表数据，也可以降低 HWM 并释放存储空间，同时也知道这是一个极其危险的操作，如果发生误操作一般要通过备份来恢复，如果没有备份，恢复将变得相当困难。

Truncate 的本质实质上就是通过为对象重新映射一段存储空间，修改对象的 DATAOB 指向，从而完成了快速的数据清除，而在根本上，数据库仅仅是将数据文件上的空间标记为可用，并未真正清除数据，不过如果未来释放的空间被其他对象使用，则数据将被真正覆盖。所以如果真的发生了误操作 Truncate 了数据表，并且没有备份，那么最好的办法就是尽快将相关文件备份出来，然后通过各种方法和工具，将文件上的数据抽取出来，完成数据恢复。

以下我们可以通过简单的测试来验证一下 Truncate 操作的实现，首先创建一个测试表，可以看到初始的 OBJECT_ID 和 DATA_OBJECT_ID 是相同的：

```
SQL> create table eygle as select * from dba_users;
Table created.
SQL> select object_name,object_id,data_object_id from dba_objects where object_name='EYGLE';
OBJECT_NAME            OBJECT_ID DATA_OBJECT_ID
-------------------- ---------- --------------
EYGLE                       34697          34697
```

接下来将这个对象的前两个数据块转储到跟踪文件中：

```
SQL>     select     segment_name,file_id,block_id,blocks     from     dba_extents     where
segment_name='EYGLE';
SEGMENT_NAME            FILE_ID   BLOCK_ID     BLOCKS
-------------------- ---------- ---------- ----------
EYGLE                         5     150265          8
SQL> alter system dump datafile 5 block min 150265 block max 150266;

System altered.
```

再通过一个新的会话连接入数据块，Truncate 之后再次转储数据块：

```
SQL> truncate table eygle;
Table truncated.

SQL> alter system dump datafile 5 block min 150265 block max 150266;
System altered.
```

对比一下两个跟踪文件如图 1-23 所示（在 Linux 下 Oracle 9.2.0.4 得出测试结果）的显著不同，我们发现除了 SCN、Highwater Mark 发生了变化之外，数据段的对象号也发生了变化：

```
[oracle@jumper udump]$ ls -sort|tail -2
   8 -rw-r-----    1 oracle       7511 Jul 19 15:53 eygle_ora_9284.trc
   8 -rw-r-----    1 oracle       7510 Jul 19 15:54 eygle_ora_9317.trc
[oracle@jumper udump]$ diff eygle_ora_9284.trc eygle_ora_9317.trc
20c20
< scn: 0x081b.dd02797f seq: 0x01 flg: 0x00 tail: 0x797f1001
---
> scn: 0x081b.dd0279d3 seq: 0x01 flg: 0x00 tail: 0x79d31001
26c26
<       Highwater::  0x01424afb ext#: 0      blk#: 1      ext size: 7
---
>       Highwater::  0x01424afa ext#: 0      blk#: 0      ext size: 7
28c28
<   #blocks below: 1
---
>   #blocks below: 0
31c31
<       Map Header:: next  0x00000000  #extents: 1    obj#: 34697  flag: 0x40000000
---
>       Map Header:: next  0x00000000  #extents: 1    obj#: 34698  flag: 0x40000000
```

图 1-23 两个跟踪文件的比较结果

查询数据库可以获得同样的内容如下（注意，转储数据段头的 OBJ#即指 DATAOBJ#）：

```
SQL> select object_name,object_id,data_object_id from dba_objects where object_name='EYGLE';
OBJECT_NAME                    OBJECT_ID DATA_OBJECT_ID
------------------------------ ---------- --------------
EYGLE                               34697          34698
```

注意，在比较文件中，除了这些不同外，其他数据内容完全相同，这也就意味着，Truncate 掉的数据并不会立即消失。

3. user$与 profile 的约束

在 Oracle Database 10g 中，默认的用户管理上有个小的改进，就是对默认的失败登录次数的限制，用户的 PROFILE 中，FAILED_LOGIN_ATTEMPTS 设置口令失败尝试次数为 10，如果连续进行了 10 次口令失败的登录尝试，用户账号将被锁定。

```
SQL> select * from dba_profiles where resource_name=' FAILED_LOGIN_ATTEMPTS';
PROFILE          RESOURCE_NAME                    RESOURCE LIMIT
---------------- -------------------------------- -------- ----------------------------------
DEFAULT          FAILED_LOGIN_ATTEMPTS            PASSWORD 10
```

在一个客户环境中，曾经遇到的一个问题就是，无法定位的某个客户端，会连续通过错误的口令进行尝试，时常导致用户账号锁定。

10g 的这个默认改进在某些环境下的确可能存在问题，当然我们可以通过如下命令恢复之前的无限制：

```
alter profile default limit FAILED_LOGIN_ATTEMPTS unlimited;
```

可是有朋友提出一个问题，当前失败了多少次从何处查询，如何跟踪？

这个问题开始我并不知道答案，但是我的习惯是看一下字典的底层表，用户状态等信息是通过 DBA_USERS 来记录展现的：

代码 1-81

```
SQL> desc dba_users
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 USERNAME                                  NOT NULL VARCHAR2(30)
 USER_ID                                   NOT NULL NUMBER
 PASSWORD                                           VARCHAR2(30)
 ACCOUNT_STATUS                            NOT NULL VARCHAR2(32)
 LOCK_DATE                                          DATE
 EXPIRY_DATE                                        DATE
 DEFAULT_TABLESPACE                        NOT NULL VARCHAR2(30)
 TEMPORARY_TABLESPACE                      NOT NULL VARCHAR2(30)
 CREATED                                   NOT NULL DATE
 PROFILE                                   NOT NULL VARCHAR2(30)
 INITIAL_RSRC_CONSUMER_GROUP                        VARCHAR2(30)
 EXTERNAL_NAME                                      VARCHAR2(4000)
```

使用 autotrace 看一下查询 DBA_USERS 展现的执行计划，其底层表包含 PROFILE$、USER$等：

```
 SQL> set autotrace trace explain
SQL> select count(*) from dba_users;
-------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                         |     1 |    78 |
|   1 |  SORT AGGREGATE              |                         |     1 |    78 |
|*  2 |   HASH JOIN                  |                         |    13 |  1014 |
|*  3 |    HASH JOIN                 |                         |    13 |   975 |
|*  4 |     HASH JOIN                |                         |    13 |   936 |
|*  5 |      HASH JOIN OUTER         |                         |    13 |   897 |
|*  6 |       HASH JOIN              |                         |    13 |   572 |
|*  7 |        HASH JOIN             |                         |    13 |   546 |
|   8 |         MERGE JOIN CARTESIAN |                         |     1 |    16 |
|*  9 |          TABLE ACCESS FULL   | PROFILE$                |     1 |     8 |
|  10 |          BUFFER SORT         |                         |     1 |     8 |
|* 11 |           TABLE ACCESS FULL  | PROFILE$                |     1 |     8 |
|* 12 |         TABLE ACCESS FULL    | USER$                   |    18 |   468 |
|  13 |        TABLE ACCESS FULL     | PROFNAME$               |     1 |     2 |
|* 14 |       TABLE ACCESS FULL      | RESOURCE_GROUP_MAPPING$ |     2 |    50 |
|  15 |      TABLE ACCESS FULL       | USER_ASTATUS_MAP        |     9 |    27 |
|  16 |     TABLE ACCESS FULL        | TS$                     |    23 |    69 |
```

```
|  17 |    TABLE ACCESS FULL         | TS$                          |    23 |    69 |
-------------------------------------------------------------------------------------
```

查看底层表 USER$的字段，其中 LCOUNT 字段立刻引起了我们的注意：

```
SQL> desc user$
 Name                           Null?    Type
 ------------------------------ -------- --------------------
 USER#                          NOT NULL NUMBER
 NAME                           NOT NULL VARCHAR2(30)
 TYPE#                          NOT NULL NUMBER
 PASSWORD                                VARCHAR2(30)
 DATATS#                        NOT NULL NUMBER
 TEMPTS#                        NOT NULL NUMBER
 DEFROLE                        NOT NULL NUMBER
 DEFGRP#                                 NUMBER
 DEFGRP_SEQ#                             NUMBER
 ASTATUS                        NOT NULL NUMBER
 LCOUNT                         NOT NULL NUMBER
。。。。。。。。
```

找个用户测试一下，可以发现 Oracle 数据库正是通过这个字段来记录登录失败的次数的（一旦成功登录之后，该计数会被清零）：

```
SQL> select name,lcount from user$ where name='EYGLEE';
NAME                             LCOUNT
------------------------------ ----------
EYGLEE                                0
SQL> connect eyglee/ee
ERROR:
ORA-01017: invalid username/password; logon denied
Warning: You are no longer connected to ORACLE.
SQL> connect / as sysdba
Connected.
SQL> select name,lcount from user$ where name='EYGLEE';
NAME                             LCOUNT
------------------------------ ----------
EYGLEE                                1
```

进一步，我们可以通过 sql.bsq 文件来确认一下，这个文件提示 lcount 正是失败的登录尝试计数（count of failed login attempts）：

```
create table user$                                                    /* user table */
( user#         number not null,                    /* user identifier number */
  name          varchar2("M_IDEN") not null,                 /* name of user */
  type#         number not null,                          /* 0 = role, 1 = user */
  password      varchar2("M_IDEN"),                       /* encrypted password */
  datats#       number not null, /* default tablespace for permanent objects */
  tempts#       number not null,  /* default tablespace for temporary tables */
  ctime         date not null,                 /* user account creation time */
  ptime         date,                            /* password change time */
  exptime       date,                      /* actual password expiration time */
  ltime         date,                         /* time when account is locked */
  resource$     number not null,                          /* resource profile# */
  audit$        varchar2("S_OPFL"),                       /* user audit options */
  defrole       number not null,                  /* default role indicator: */
               /* 0 = no roles, 1 = all roles granted, 2 = roles in defrole$ */
  defgrp#       number,                                   /* default undo group */
  defgrp_seq#   number,                /* global sequence number for  the grp *
  spare         varchar2("M_IDEN"),                     /* reserved for future */
  astatus       number default 0 not null,          /* status of the account */
                /* 1 = Locked, 2 = Expired, 3 = Locked and Expired, 0 - open */
  lcount        number default 0 not null, /* count of failed login attempts */
  defschclass   varchar2("M_IDEN"),                  /* initial consumer group */
  ext_username  varchar2("M_VCSZ"),                     /* external username */
  spare1        number, /* used for schema level supp. logging: see ktscts.h */
```

图 1-24 sql.bsq 文件中的 lcount

虽然我们可以从数据字典中获得很多知识，但是切记不要轻易修改数据字典的内容，数据字典是数据库的核心数据存储地，修改其内容很容易导致数据库崩溃和无法启动。

曾经一度最常见的案例是修改 props$以变更数据库字符集（这种方法是不可行的），如果修改错误，则数据库将无法启动；也曾经有人修改 props$中的 global_name 将其值置空，这也会导致数据库无法启动：

```
update global_name set global_name=″
```

然后启动数据库时出现如下错误：

```
Fri Nov 21 08:58:29 2008
Errors in file d:\oracle\product\10.2.0\admin\govt\udump\govt_ora_4972.trc:
ORA-00600: 内部错误代码, 参数: [18062], [], [], [], [], [], [], []
ORA-1092 signalled during: alter database open...
```

在 Metalink 上最近发布的一个 BUG（Bug 7176385）与此有关，因为数据库在启动中须校验 GLOBAL_NAME，空值会导致无法启动。

还遇到过一次故障是，某公司技术人员将数据库中的几个数据字典表 Truncate 掉，原因是他们发现这几张字典表占用空间很大，这直接导致了数据库不可用。数据库环境为 Oracle 9.2.0.7 RAC 环境。现场检查确认，主要被截断的表有如下一些：

```
SQL> select object_name,object_type from dba_objects where object_name like 'IDL%';
OBJECT_NAME                    OBJECT_TYPE
------------------------------ -------------------
IDL_CHAR$                       TABLE
IDL_SB4$                       TABLE
```

```
IDL_UB1$                        TABLE
IDL_UB2$                        TABLE
```

IDL_UB1$等字典是记录数据库对象编译信息的，丢失了其中的数据，所有过程、Package 等都将无法执行。在这个客户案例中，数据库会出现大量的 ORA-00600 错误，所有事务不能进行：

```
ORA-00600: internal error code, arguments: [17069], [0xC0000000DDDFA690], [], [], [], [], [],
[]
```

参考文献：

Thomas Kyte Effective Oracle By Design