ACOUG
All China Oracle User Group
中国 Oracle 用户组

ITPUB

NO.2 2010/10

Oracle DBA手记2

数据库诊断案例与内部恢复实践

电信运营商DBA工作手记

DBA小故事之SQL诊断

Ora-600错误深入探究案例两则

深入解析DEPENDENCY$对象的恢复

主 编
盖国强 崔华
编 著
郭岳 晶晶小妹 等

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

主 编
盖国强 崔华
编 著
郭岳 晶晶小妹 等

Oracle DBA手记2

数据库诊断案例与内部恢复实践

NO.2 2010/10

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书由多位数据库技术专家和爱好者合著而成，集合了各行业 DBA 的工作经验与思考，包含了精心挑选的数据库诊断案例与数据库恢复实践。内容涉及 Oracle 典型错误的分析和诊断，Oracle 600 内部错误的处理和解决，优化器与 SQL 的行为与分析，以及很多内部深入技术的实践。

本书的主要内容以原理分析、内部实践、故障解决为依据，将 Oracle 数据库的深层技术层剖缕析抽丝剥茧地展示给读者。希望能够帮助读者加深对于 Oracle 技术的认知和理解，并将这些技术应用到实践中去。

图书在版编目（CIP）数据

Oracle DBA 手记 2：数据库诊断案例与内部恢复实践 / 盖国强，崔华主编.
—北京：电子工业出版社，2010.11
ISBN 978-7-121-11946-0
Ⅰ. ①O… Ⅱ. ①盖… ②崔… Ⅲ. ①关系数据库—数据库管理系统，Oracle Ⅳ. ①TP311.138

中国版本图书馆 CIP 数据核字（2010）第 194482 号

策划编辑：周筠
责任编辑：杨绣国
印　　刷：北京天宇星印刷厂
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　**邮编：**100036
开　　本：787×1092　1/16　**印张：**18.25　**字数：**426 千字
印　　次：2010 年 11 月第 1 次印刷
印　　数：4 000 册　　**定价：**45.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 *zlts@phei.com.cn*，盗版侵权举报请发邮件至 *dbqq@phei.com.cn*。服务热线：（010）88258888。

本书编委会成员介绍

主编：盖国强、崔华

编著：张晶晶、郭岳、颜志芳、梁敬彬、怀晓明

盖国强
网名Eygle

Oracle ACE总监，恩墨科技创始人，ITPUB论坛超级版主，远程DBA服务的倡导者和实践者，致力于以技术服务客户。著有《深入解析Oracle》、《循序渐进Oracle》、《深入浅出Oracle》等书；从2010年开始，致力于《Oracle DBA手记》的撰写与编辑工作，并与张乐奕共同创立了ACOUG用户组，在国内推进公益自由的Oracle技术交流活动。

个人网站：http://www.eygle.com

崔华
网名dbsnake

2004年开始从事DBA工作，在Oracle的安装、升级、开发、性能调整、故障处理方面有丰富的经验，对Oracle的体系结构具有深入了解；深入理解Oracle的内存结构、物理存储（各种块格式）、锁机制、优化机制等；深入了解Oracle的备份恢复机制，熟悉Oracle的各种备份方法，能够处理各种情况下的复杂数据恢复情况。
崔华也是热心的技术分享者，多次在ACOUG的活动上与技术爱好者分享技术心得。

个人网站:http://www.dbsnake.com

张晶晶
网名晶晶小妹

晶晶小妹是ITPUB上的明星，以女儿之身于2008年前后20岁许迅速崛起，其光亮闪耀一时。晶晶在ITPUB上发表的一系列技术研究和个人感悟文章，以自我独特的学习研究视角，将技术内容细致入微的展现出来，形成了自己独特的技术风格。
晶晶做过程序员、DBA、讲师，目前在北京从事OCP认证培训等相关工作。

郭岳
网名三少
David.Guo

资深Oracle DBA，Oracle 10G OCM 。目前就职于杭州某手机软件平台型企业，任DB技术经理；曾服务于某Oracle服务提供商，负责维护亚洲最大，全球第二的电信行业的Oracle数据库，包括性能，监控，问题处理，方案制定及项目协调。超过6年的专职Oracle维护经验，ACOUG成员。曾经的ITPUB以及CNOUG版主。目前致力于Oracle高可用和性能优化。专注于互联网企业数据库架构。

颜志芳
网名yanzfcu

现任职于上航某民航业IT公司系统架构师兼开发DBA，主要从事数据库及BI应用的架构设计与开发工作。2004年在工作上开始接触Oracle，后来逐渐被其之博大精深所吸引，故而也产生了浓厚的学习兴趣，2008年获得9i OCP证书，拜读了Tom Kyte、Cary Millsap、Jonathan Lewis、Christian Antognini等大师的许多书籍后，才发现自己的Oracle之旅才刚刚开始，相信旅途中一定充满着挑战和快乐。

梁敬彬
网名wabjtam123

ITPUB数据库新技术板块版主，福建富士通公司在聘数据库专家，多年从事电信相关行业工作，负责系统架构设计、优化等工作，有着丰富的数据库管理、设计、开发、培训经验和电信行业经验。

怀晓明
网名lastwinner

ITPUB Oracle开发版等版块版主，兴趣广泛，视野广阔，曾获得第一届ITPUB最佳建议奖。在多个大型IT企业多年的工作历练中，积累了大量的系统架构设计经验，擅长数据库和Web的设计、开发，精于故障诊断与处理，具有丰富的省部级电子政务行业工作经验及项目管理经验。

会泽百家 始成江河

——《Oracle DBA 手记 2》编辑始末

《Oracle DBA 手记——数据库诊断案例与性能优化实践》一书已经出版近一年，现在我完成了《Oracle DBA 手记 2 —— 数据库诊断案例与内部恢复实践》的组稿工作，可以将我的计划与梦想进一步呈现给读者们。

做"DBA 手记"这个系列，就一直希望能够在我力所能及的时间维度内，不断推动更多的技术人开始分享自己的经验心得，在这个分享的途中，也可以使得我能够不断地从各位作者那里偷师学艺，不断进步。

我得承认，实践这个梦想有不小的难度，我个人的时间、能力都构成了制约，然而最初的想法一直鼓舞着我，我继续着自己的编辑尝试。我是一个有了想法就无法释怀的人，我想我还是要坚持下去。

于是，我开始继续寻找好的作者，好的文章，去汇集散落在互联网海洋里的粒粒珍珠，这个过程有点辛苦，但是完全值得。一篇一篇的精彩文章，我深深了解其中蕴含的真知灼见、缜密思考、无私分享。文章读得越多，我越能坚定自己的信念：能够将这些知识汇集起来分享给更多的读者，这本身就是一件功德。我坚信这些文章里的经验积累、思路方法、分析过程能够帮助更多的读者去更好地面对工作，解决疑难。

我很珍视自己努力去完成一件工作的过程，而且在这个过程中，我又可以结识真挚的朋友，这是让人十分欣喜的事情。

崔华是我们的新朋友，但也迅速变成了老朋友。他在数据恢复领域的造诣与认知深刻，能够圆转如意地运用种种内部工具进行数据恢复，基于对 Oracle 的深入理解，他能够从本质上对很多 Oracle 问题进行解构和揭秘，这种能力在我熟知的领域无人能及。在我们刚刚完成《海量数据库解决方案 I》一书的校译时，他就开始为《DBA 手记 2》撰稿，他担心自己传达的内容太过深入而不受欢迎，我说，只要是真正有价值的东西，我们就欢迎和鼓励，也愿意出版，最终读者们会发现书中的知识无价。

晶晶小妹是 ITPUB 上的明星，以 20 岁出头的女儿身于 2008 年前后迅速崛起，其光亮闪耀一时。晶晶发表的一系列文章从自己的角度出发，深入探究 Oracle 的种种知识难点，将其浅显明了地展现出来，这些无私的奉献为晶晶赢得了大批追随者。我请晶晶将整理好的文字拿出来，与更多的读者分享。这个愿望最终达成，按照全书的内容需要，我会逐渐将晶晶的学习历程展示出来。任何一个初学者看一看 20 岁的晶晶曾经走过的学习历程，我想都会有所收获。在学习 Oracle 技术的过程中，一定要具备从一个点深入下去的能力，并且能够通过实践来验证书本知识和自己的想法，晶晶这条路就是如此走来。

怀晓明（野花）是开发经验丰富的编程高手，我觉得书中应该也包含与开发相关的内容，这样就想到了在 ACOUG 活动上野花分享的 Connect by 的相关主题。他欣然接受了我的请求，将文档

整理成流畅的文章，投稿给《DBA 手记》，这是一个可贵的开始，以后《DBA 手记》的内容应当不断扩展到更广泛的领域，我们期待有更多有意思、独特的主题融入到书中来。

郭岳是我多年老友，辗转于各移动运营商旗下工作多年，常见他在 Blog 上发表技术感悟，遂邀其著文分享。老郭思揣多日，慨然允诺，于是有了他的“电信运营商 DBA 工作手记”。这些内容有的来自故障导致的惨痛教训，有的来自经验的真知总结，有的寥寥数笔即可传神，有的条条框框又可以作为警示为我们常记不忘，这样的行业经验分享于我们尤其宝贵。

颜志芳是我在上海结识的朋友，他在航空领域工作十多年，对各类航空数据库应用经验丰富，航空业的安全稳定与可用性要求极高，老颜记录了他在 DBA 工作中遇到的一些常规问题和经验总结，让我们有机会一窥又一个重要行业的 DBA 生活。

梁敬彬是一个热情上进，活力无限的小伙子，在上海的一次技术交流中，我们互相结识。在 ITPUB 上他分享了自己总结的大量文档和案例，在与大家的不断探讨中迅速成长起来，他把总结出来有代表性的文章发送给我，作为和读者的一次分享，我自己就是这样一路走来，现在又可以欣喜地见到年轻一代继续成长。

在本书的第一部分，我收录了年初应博文视点周筠老师之邀，发表在《程序员》杂志的一篇个人总结文章，在这篇文章中，我回忆了自己走过的一段段历程，由年少至而立，从读书到就业，乃至创业。

根据本书的内容，我们将全书分为四篇：

第一篇　DBA 之路

这一篇包括一章内容，计划以后每辑《DBA 手记》均邀请一位资深 DBA 或相关从业人士来分享其从业经历和成长历程，希望他们的这些经验和心路历程，能够为 DBA 朋友们提供参考借鉴。本篇内容由盖国强撰写。

第二篇　DBA 手记

这一篇包括四章内容，以手记形式记录了 DBA 们的工作点滴，经验分享，本篇的四章内容分别由盖国强、崔华、郭岳、颜志芳撰写。

第三篇　开发基础

这一篇包括两章内容，将 DBA 的一些相关基础知识探索、讲解以及开发相关的知识内容纳入这一篇章。本篇的两章分别由晶晶小妹和怀晓明撰写。

第四篇　诊断案例

这一篇包括五章内容，涵盖了 DBA 在工作中遇到的案例诊断与解决方案，本章内容由梁敬彬、盖国强、崔华撰写。

感谢为本书供稿的朋友们，感谢好友杨廷琨帮助我审阅了本书的部分章节，感谢崔华、张乐奕在本书成书过程中给予我的大力支持，感谢恩墨科技的罗晓程，他帮助我做了大量文字校对、书稿整理的工作，正是因为有了朋友们的无私分享和大力支持，本书才得以呈现在读者面前！

本书的很多作者都来自于 ITPUB 论坛（http://www.itpub.net），在我的学习成长过程中，ITPUB 论坛给予了我们巨大的帮助，那里永远是我们网络上的精神家园。

基于技术分享的目的，我和张乐奕（Kamus）在 2010 年创建了 Oracle 用户组（ACOUG – http://www.acoug.org），并且每个月在北京开展一次面对面的技术交流活动，我们希冀通过线下的交流活动，将 Oracle 的技术分享不断推动下去。

网络以及通过网络来到现实的朋友，永远是我无比珍视的巨大财富。

关于本书，除了选入的这些文章之外，我还收到了许多其他朋友的慷慨赐稿和精彩分享，这些稿件将会收入《DBA 手记 3》中，编辑工作远未完成，我的梦想要继续！

盖国强（Eygle）

2010-09-03 于北京

Oracle DBA手记

——数据库诊断案例与性能优化实践

NO.1 2009/12

本书由多位工作在数据库维护一线的工程师合著而成，包含了精心挑选的数据库诊断案例与性能优化实践经验，内容涉及Oracle典型错误的分析和诊断，各种SQL优化方法（如调整索引、处理表碎片、优化分页查询、改善执行计划等），以及优化系统性能的经验。

作者不仅强调案例的实用性和可操作性，更着重再现解决问题的过程和思路，并总结经验教训，希望将多年积累的工作方法，以及对DBA的职业发展感悟展现出来，供广大Oracle DBA借鉴参考。

作者们的话：

在网络上，我经常能够看到大量精彩的诊断案例与故障处理过程，又常常遗憾这些文字被互联网的海量信息所淹没，于是我萌生了对这些文字进行“编辑”的想法，想通过自己的阅读、学习和选择，让更多的作者能将自己的经验分享出来。想想一本全部来自第一线DBA的经验集合，将会是多么宝贵的财富啊！——Eygle

目录

Part 1 DBA之路

0

天道酬勤

——从头细数来时路

转眼就过了三十岁，想想小时候整天梦想着快点长大，现在看着儿子在屋子里跑来跑去，一切宛如梦幻。年纪渐长，随着那些在不同年龄段结下的挚友散落天涯，也就越来越少去表露自己，所有年少时的梦想，成长时的奋斗，挫折时之困惑，都渐渐学会了独自去承担，有时候觉得，男人的宽厚与沉稳，必须经过岁月与风霜，舍此别无他途。

想起毛姆在《刀锋》一书的扉页上引用《迦托-奥义书》的话：

一把刀的锋刃很不容易越过，因此智者说得救之道是困难的。

每个人都会遇到自己的锋刃，每个人也都在不停地寻找自己的解救之道，然而人生的意义就在于此。在于途中的欢喜忧愁、起伏跌宕，**艰难困苦，则玉汝于成**。

而如果静下心来，回首想想来时之路，其实一切皆有因由。

阅读童年

关于童年的记忆，只有一些惊鸿片影还留在脑海里。

2 岁以前的记忆早已经消失不见，那时候妈妈说还住在城市里，因为爸爸在粮库里面当临时工，所以那时候最熟悉的一个词是“扛麻袋”，在很多年里，那是我脑海里非常重要的一个词汇。当然那是一个相当辛苦的工作，而且是临时的，后来爸爸带着我们举家迁回农村，我的记忆从这时候开始。

5 岁左右时，开始上生产队的幼儿园，那时候叫“育红班”，不收钱，妈妈说目的是腾出劳动力来去建设社会主义。印象最深刻的是，有一次幼儿园发了一块绘有牛郎织女的手帕，挚爱不已，不记得在什么时候失去了。爸爸结婚前曾经参过军，在丹东服役，环境极其艰苦，也许正是艰苦的环境养成了父亲正直坚毅的性格。小时候家里有一个上锁的抽屉，在我的记忆里至为神秘，那里面有一本毛泽东选集、几枚子弹壳和毛主席像章，这些东西在那个年代再普通不过。父亲有一张在天安门广场的照片，是文革时串联到北京，据说在天安门广场的人山人海中望见毛主席挥手。那是我小时候知道的关于那个年代微弱的信息之一。

当初回到村里时，据说村长排斥我们，指定了一块洼地给我们盖房子，父母都是倔强的人，有强烈的自尊，他们在那块土地上铺垫了很多泥土，盖起了 2 间土坯房，所有花费是 200 元，在那里我一直住到 12 岁。我仍然记得童年里，每次大雨，我都要和爸爸去外面淘水，以免水流到屋子里，然而于我来说，那时候获得更多的是快乐。房子很矮，所以记得小时候经常会爬上屋顶跳下来，可以安然无恙，除了被父母责备。

在那间屋子里，我开始学习用田字格的纸张写字，爸爸说我写的字很秀气，从来不会超出格子，我想这正和我细致细腻的性格相映，而姐姐写的字就极大气，不是格子所能够束缚的。

从识字开始就爱上了读书，疯狂的阅读能够找到的一切带字的东西，我清晰地记得有一次病的厉害，爸爸带我到市里看病，出来以后问我要什么，我说要买书。最后买了一套 2 本 5 元钱的《薛仁贵征东》，那套书是我童年里最珍贵的礼物，至今仍存。展示一下这本书的封面，不知道是不是有读者会和我拥有类似的记忆。

对于读书的爱好不知因何而起，也许是因为求知欲的自然而然的推动，这种爱好让我总是自发的、不需要推动的去读书和学习，父母从来没有因为读书而责罚我。

再后来在舅舅家发现了他订阅的杂志《新村》，就一本本借回来读，里面很多的楹联、典故、诗词至今还能够流利的背出。这些书几乎成为童年里最重要的知识读本。那时候常以昆明大观楼的“古今第一长联”作为朗读范本，有一次，一个亲戚偶然的教我朗诵岳飞的满江红，让我学会了什么叫抑扬顿挫，转承起合，在大学里我还以诗朗诵拿过奖状。

当然后来也疯狂的迷恋上武侠小说，在小学三年级左右，金庸、古龙、梁羽生盛行，当时班上有个女生的哥哥买了很多书，我们就整天怂恿她偷来给我们看，每有新书到手，就是废寝忘食，毕读方休，那时候印象深刻的就是读古龙的《多情剑客无情剑》，虽然确定性不能理解其中的复杂与人性，可是书中的爱恨情仇不可避免的让我们怦然心动。都说武侠小说是成年人的童话，可是真实的是，那些书成为了我们的启蒙读物。

在需要阅读时有书读是一种幸运，然而在需要阅读时无从得合适之书，实在是一种遗憾。

东北农村孩子关于学校的记忆一定和其他地方不同，那时候最痛苦的是干农活，家里的不算，记得小学拥有几十公顷的田地，而学生就是免费的劳动力，从除草到施肥、收割、扒苞米、装车运输都是我们的必修课，一般从春天开始就是无尽的劳动，除了学校还要帮助老师家干活，手上的老茧、血泡是常事，到冬天还有无尽的扫雪、砍柴、烧炉子等工作等着我们，这种情况一直持续到高中。**对繁重农活的恐惧进一步促使我去读书，试图通过这唯一的途径离开农村。**

中学之路

按照农村的习惯，小学毕业很多孩子就停止了学业，然后回家种田、放牛、养猪或者外出务工来赚钱，记得小学有个女同学在 5 年级时就辍学回家嫁了人，没两年就生了宝宝，那就是那个时代的农村。可是父母支持我们读书，小学毕业后继续到镇上的初中学习，中学离家里有 12 里山路，走路要一个小时，爸爸小时候就在这条路上来回奔走，日未出出行，夕阳下方归，到我的时候要好很多，爸爸四处借来一些零件，为我组装了一辆最简捷的自行车，难忘的在那条路上跋涉的岁月。一旦有大雨下时，路上就泥泞不堪，自行车也没办法行走，那时候就数着田垄将自行车藏到田里，放学再来找，记忆里非常多的瓢泼大雨，经常的浑身湿透。

至于学习，由于语文偏好，一直都受到语文老师的偏爱，作文经常被作为范文在学校朗读，其他学科的学习就要相对差一些，虽然成绩也还是不错，通常在班里排在前几名，但是据说和县市里面的重点中学要相差很多。

在中学，学校里有一个小小的图书馆，那里

的藏书成了我们整天觊觎的对象，可是当时的图书馆显然是一个摆设，不对学生开放，我只好辗转托老师去帮忙借书，唯一借到的一本是普希金的传记，然后开始了如饥似渴的阅读和大量的笔记摘录，虽然现在并不能记得太多，再次了解十二月党人也已经是大学时事了，但是普希金自由浪漫的风格深深的植入了我幼小的思绪。

在中学，也开始学绘画，一个深受爱戴的老师每天中午和假期义务的教我们大约十几个孩子，**他第一次让我感受到了因材施教的意义**。那时候，他根据每个人的性格、爱好和特长来教授不同的画风，比如工笔，我就无法一根根去拉仕女的秀发。后来我学版画、篆刻和素描，作品还意外的在市里的展览上获奖。

一位好的教师会为我们的一生开启最宝贵的宝藏之门，回想起来，我总是心存感激。所以即便现在，在我自己很忙的情况下，我也总是会定时去回复邮箱里很多陌生朋友的来信，有时候写很多的话，总是希望，即便能有一句话对一个人产生良好的影响，那就值得我们的付出。

那时候，还开始写作的尝试，四处去投稿，期望自己的文字变成铅字的印刷品，也得了不少关于写作的奖状。有一次在北京读书的表哥回来，说在什么地方看到我写的东西获奖，对我进行了一番夸奖，当时小小的虚荣心得到了极大的满足。

然而兴趣广泛、涉猎过多最终让我尝到了苦头，最惨痛的打击随中考而来。那时候一个老师说，你学的不错，估计复读一下考重点高中没问题。我当时不以为然，谁知中考前，得了严重的水痘，在家里闭门闷了很长时间才康复，父母说这影响了我的考试，第一年差了几分没有考取。那时候流行复读再考，浑浑噩噩中又浪费了一年时光，仍然没有考取重点高中。**中考的失利是我那个幼小的年代经历的最大挫折，也让我认清了自己的能力，时时提醒自己为了成功要付出更多的努力。**

于是又到了一次抉择的路口，当时的情况是，如果考取重点高中，那么你就相当于跨进了大学的门口，而如果落榜，最好回家种田，这一道门槛，至少有 80%的人选择了放弃。而我几乎是没经过任何思考，就决定找一所普通高中，继续读书是父母和我的共同愿望。于是在离家近 50 里左右的另外一个镇上，找到了一所高中，开始住校学习。在这所学校里，我开始刻苦学习，并且第一次基本摆脱了学校里的农业劳动（这一点当时让我兴奋不已啊！）。

在学习上，我喜欢学文，但是升学率让我选择了理科，高中里，我仍然以语文出众，我曾经的梦想是成为一个作家，能够出版印着自己名字的图书。但是那时候仍然改不了兴趣广泛的毛病，又爱上了集邮、集火花、烟盒、糖纸，收集一切有点意思又不需要花费的东西；还爱上剪报，分门别类收集一切能够得到的有意思的报纸和文章。

高中毕业时，**班主任也是最喜欢我的语文老师给我的评语是：博采众长，也应该学有所专！**她跟我解释说：说明白点，将来你千万不要搞得样样精通，样样稀松。

这样的话同样对我影响至深，现在在工作上，技术方面，我一直极力在数据库技术方面做的专业，安身立命之后方能旁顾其他。

好的老师真的就是一所学校，感谢命运的眷顾，在人生每个重要的阶段，都有一些老师和话语能够指引我。

在学习中，父母从不干涉我的学习情况，他们一直以来从不曾更改的就是对我的无条件的支持。高中的成绩在班上一直在前三名左右，并不觉得太困难，只是在那个环境中不知道如何做的更好。一直没有停歇的是阅读，仍然不停的读一切能够找到的各种各样的图书。高中是动荡的三年，高一很快过去，高二开始文理分班，我被调整到另外一个班级，高三又更换了几乎所有的老师，直到最后那一年才知道什么是高考，几乎所有的面向高考的学习都是从那一年开始的，我所就读的普通高中，所有好的老师都部署在高三，所以直到那一年我才知道，前两年的时间几乎都

是白白浪费掉了。

高三遇到的物理老师对我产生了极大的影响。他当时的形象总是杂乱的头发，深陷的眼眶，乱糟糟的胡须，一本从不翻开的教案，一只粉笔。上课时可以随意的从任何一个章节开始讲起，精彩绝伦。在做高考模拟题讲解时，他总是能够端详一下就告诉我们答案，清晰的记得一次学生根据答案置疑他的推演，他就说：那我们再来回顾一下，这里的公式没有错，这里的计算没问题，那么就是答案出错了。他的自信与严谨一下子就交给了我，从那时候**我明白，只要有严谨的思考，缜密的推演，就可以毫不置疑的相信自己，这一理念从那时候起就固化进我的脑海。**

高考填报志愿时，没有任何参考，父母对于高考的认识要比我们还少，在学校里胡乱做出了这一生最为重要的决定。第一志愿选择了北京的一所学校，没有考取；第二志愿是同桌选择了一个名字听起来不错的“昆明理工大学”，好几个朋友都填了这个学校（当时我甚至不知道昆明之于东北是什么概念），后来就只有我一个人去了万里之外的这个第二故乡。

成长之途

大学使我第一次踏上万里之途，在那之前，自己没有离开家超过 100 里的地方，这次远足使我出山海关、经北京，南下两湖，继而登上了云贵高原，古人说**读万卷书不如行万里路**，诚然如此！这路途的艰辛自不足提，那时候从家里到学校要坐 63 个小时的火车，然而我从旅途中真正理解了山川之广大，人土风情之不同。记得当时火车经过溆浦站，第一次找到了屈原诗中的“入溆浦余儃佪兮，迷不知吾所如”的意境。屈原流放时，曾经在溆浦做过一段停留，而我当时的感觉时，我要走的路是比屈原还要远啊，旅途之中满耳听不懂的方言俚语，这一切真真正正让我有了去国怀乡的感伤和悲从中来。

虽然当时曾经犹豫和怀疑，但是后来我深深的爱上了那片土地，我的第二故乡，并且从未对自己偶然的选择后悔，一切的偶然之中，必有必然之果。

在大学里，我读的是自动化专业，在高中时我自己并不能够知道这一专业意味着什么，等明白时一切为时已晚。可是这一切也不再重要，我认为**大学的四年时光最终要的是予我们时间学习、思考、成长，专业的学习反而成为次要**。

专业课上整天研究的是电机、电路、控制这些知识，复杂的计算与公式让我头晕，象千千万万以为自己选错了专业的同学一样，我在大二开始尝试第一次转变，在学校里辅修了企业管理的课程，之所以选择这个课程，一方面是因为自己在高中就偏爱文科，另一方面是因为自己早就有一个管理的梦想。这一尝试最终在大四考研时折戟沉沙。

大学里的最大收获是有了一个真正开放的图书馆，除了阅读之外，学会了真正的思考，开始阅读哲学、心理学和历史的书籍，**知识可以通过学习得来，而学问则要通过思考获得**。在大学里通过家教和打工来维持生计，同时参加了很多校内外的活动，结识了几个一生中至为重要的知己好友。这些是我在那四年中最大的收获！

我的第一份工作

大学毕业，我不得不在 2000 年面对第一次择业的考验。

基于当时热火朝天、风起云涌的 IT 浪潮，再加上对自己的专业知识没有信心，所以毕业时我考虑转行去做 IT（那时候 IT 这两个字母对我来说同样非常抽象和遥远）。虽然 2000 年的时候就业压力还没有如今这么大，但是我仍然清晰的记得那时候招聘会上人山人海、无法抬脚的情形。那一天我投出了 3 份简历，其中的一个公司给了我面谈的机会，在第二天这个公司已经签下了我同班的一位同学，所以他们决定不再招我这个同班的，我大急之下拿着自己所有的证书、奖状等材料找到公司的总经理，向其陈情，我永远记得那一天的情形，当他听了我的情况后说：**行，没**

问题，我相信学自动化的搞什么都没有问题（后来得知他本人就是自动化系出身）!

就这样我颇为惊险的一头扎进了 IT 圈，成为了一个程序员，从对程序、代码、数据库、操作系统等几乎一无所知开始，一干就是 3 年。

我从最基本的编程开始，学习 VB、PowerBuilder、Delphi 等技术，投入到一个开发全套企业级 ERP 软件的团队中，最初的学习是艰苦的，但也是快乐的。在大学时几乎没怎么接触电脑，公司里却可以有属于自己的笔记本，一切都是新奇的，我就在这样的兴奋与新奇中迅速熟悉了软件与计算机技术。当时公司选定的主要开发工具是 PowerBuilder，PowerBuilder 是当时最为流行的开发工具之一，而今天却已经销声匿迹，IT 领域的变化就是如此迅捷。

在开发软件的过程中，不可避免的接触到了后台数据库，了解了庞大而复杂的 Oracle 数据库产品，并对其产生了浓厚的兴趣。在做好开发工作的前提下，公司开始给我机会让我接触、学习和协助管理 Oracle 数据库及 HP-Unix 主机。

就这样我接触到了一个项目中核心的几部分内容，并且在这三年中，从开发 ERP 软件到管理 Unix 系统、维护 Oracle 数据库，逐渐积累了知识成长起来。

在这个阶段，我可以分享的两个字是：勤奋！我通过刻苦的努力，不懈的坚持，最终获得了公司的认可，并且给予了我更多的机会。

在后来总结这段工作时，我给一些朋友的建议是：**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！**

而且，这个积累的过程需要时间，只有**去除浮躁，认真学习，不断积累，寻找机遇**，才能够更好的把握自己的职业生涯。

我的网络生涯

在工作之余（或者说工作之间），我的网络生涯也随之展开。

在 2000 年左右，我注册了网名 Eygle，开始混迹于各种网络社区，那时候坚守的两个社区是：**榕树下、ITPUB**。潜在榕树下是因为从小就怀有的作家梦，而 ITPUB 论坛则是探讨和学习技术知识的大本营。

榕树下后来失去了，而我一直在 ITPUB.Net 技术论坛坚持到今天，并且最终成为了这个论坛的核心成员之一。

在注册 ITPUB 论坛时，我对 Oracle 数据库技术（目前我主要从事的工作内容）还知之甚少，在论坛中，靠着对技术的热情与执着，不停的学习、思考、阅读，在帮助别人的过程中也不断提升了自己。记得那时候，几乎所有和 Oracle 技术相关的帖子都会通读、关注，并且力所能及的帮助解答、探讨，甚至搭建测试环境帮助网友们寻求解决方案，那一时期论坛里极其活跃的讨论、学习氛围最终帮助一批数据库技术爱好者们成长了起来，现在这些人分布在全国各地，已经成为各个企业的骨干力量。

靠着求知欲、热情与互助，我在这个论坛里先后担任了微软技术版版主、Oracle 数据库管理版版主、超级版主，进而成为了核心团队的一员。最终，ITPUB 在 2006 年被 IT168 网站所收购，交易金额大约为 100 万美元，这是一个终结，也是一个新的开始，在互联网的收购中，我们真正熟悉了资本的力量、网络的价值。

在这些年的网络生涯中，我收获了知识，也结识了很多志同道合的朋友，从网络到现实，这生活已经成为我生命中的一个重要部分。

在这一阶段我可以和大家分享的话是：**不论做什么事，有付出就一定有回报！**在开始做一件事情的时候，不要计较太多，也不要顾虑太多。

在 2004 年 4 月 13 日，也就是我到北京后 1 年多之后，我在网络上开启了自己个人的博客站点，注册的域名就是 Eygle.com，在随后的日子里，我基本坚持每天在网站上发表一篇或技术、或生活的个人文章。

几年坚持下来，我的网站上已经累积了上千篇技术和生活感悟文章，这些内容对我来说是无比宝贵的财富。通过网站，我还结识了很多的朋友，最高兴的是，很多文章能够帮助别人，朋友们经常发邮件来和我探讨技术内容或者对某个有帮助的技术文章表示感谢，甚至有很多朋友来自台湾、香港和国外的很多地方，这些都成为鼓励我坚持下去的动力！

现在经常有网友问我，这么多内容是如何积累起来的，如何构建一个个人站点？其实排除技术之外，只有两个字：**坚持！**如果你能够坚持数年如一日的做一件事，那么最后的成绩一定会让你自己也吃惊的。我们每个人在学习和成长的过程中，都做过无数的思考和学习，而很多时候，我们都只是将这些经验和过程记录在自己的头脑中，时过境迁就可能模糊、遗忘，而如果记录总结出来，不仅可以帮助其他人，还可以对自己做个记录，当然这要有所付出，可是如同我前面提到的，**有付出就一定会有收获。**

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“=”，但是无关紧要，只要具备了以上因素，我想我们每个人都会离成功不远了。

在从事一件工作或事业时，能够坚持不懈是多么重要啊！

我的第二份工作

2003 年 4 月 1 日，我离开原来学习生活了 7 年的城市来到北京，开始寻找新的机会、新的起点。

结束一份熟悉多年的工作，离开一个生活多年的城市，走向一个陌生的城市陌生的街道，这并不是一件容易的事。很多朋友问起我当时的抉择，回想起来感觉重要的有两点，一是感觉遇到了瓶颈；二是做好了知识的积累。然后再有一些契机和触发条件，就很容易做出选择了，当然最重要的，那时候我还算年轻。

在北京寻找工作的过程中，我已经有了两个可以选择的方向，一是作为一个程序员继续做 ERP 软件的开发工作；二是作为数据库管理员（DBA - Database Administrator），寻找一个数据库管理的工程师职位。

也许是命中注定，也许只是机缘巧合，ITPUB

上的一位朋友为我介绍了一份 DBA 的工作（虽然那只是我众多面试的其中之一），在非典前夕的兵荒马乱之中，这家公司的快速响应让我在 4 月 17 日正式上岗，接受了这份工作，而在非典之后，还有一些公司陆续通知我去复试或者考虑入职。

在这之后，ITPUB 上几个素未谋面的好友陆续来到北京，叶梁（ITPUB 上的 Coolyl）从广州回到北京，冯春培（ITPUB 上的 biti_rainy）从珠海回到北京，他们和我住到了一起，那所租来的房子成为了几年间我们在北京的重要据点，我们家的技术实力空前壮大，我用四个字来形容那个阶段：黄金时代。那个时侯，很多时间在讨论与学习中度过，我们经常彼此提出值得研究的内容探讨研究，这一段时间，进步飞快。我们家还有很多非常住固定成员，ORA-600、Kamus、橘子、Seay 都是家里的常客，大家经常在一起打牌，讨论，甚至注册马甲上网吵架，那一个阶段结下的朋友和友谊将永不磨灭。

在北京的第一家公司是一个快速成长的企业，我和北京共历了非典，也和这个公司共历了快速成长到衰退的整个过程。这个公司给了我宽松的工作环境、良好的同事氛围、足够的成长空间，使得我能够全心的投入到工作和学习之中去。很多同事十分优秀，在他们中间，我感受到了工作的乐趣与成长的快乐，从他们身上学到了很多优秀的品质。工作的锻炼、同事的影响、自我的学习，让我快速的成长起来，并且最终在职业上成熟起来。

在职业上成熟、获得充分的自信、能够清晰、冷静、严谨的思考，对于一个技术人员来说尤其重要，我庆幸的是，有机会在这家公司获得了这些。

随着技术上的进步和发展，我的职位开始有所变化，在公司工作的 5 年中，我的职位从工程师到部门经理，再然后是部门总监。职位上的变化让我开始接触新的内容，那就是如何领导和带领更多的同事为了一个共同的目标而努力。作为一个领头人，**你还要学习如何为他人着想，从他人的角度看问题，如何带动大家共同进步与发展。**在这个职位上的思考与经历同样让我受益匪浅。

我并不能够清晰地回忆起，何时建立起从他人的角度看问题的思想，但是这一习惯对我至为重要。我一直以来的习惯是，从不轻易对一件事情下判断，哪怕于别人来说可能是一件理所当然的事情。有时候某个人所作的看起来似乎不可理喻的事情，了解起来总有其可原的情理。**理解、宽容、不要以恶意去揣度别人，是我一直遵循的人生法则。**

我在北京的第一个公司工作了 5 年，然后我离开了这家公司，开始单飞。

我的学习与积累

在北京工作这几年，除了做好自己的本职工作之外，我还不断学习，根据自己的实践与积累，写作出版了 6 本 Oracle 数据库方面的技术书籍（其中三本为与其他作者联合撰写）：

写作的最初想法很简单，那就是把自己积累的知识与经验分享出来，并且可以和朋友们一起为社区与网络生涯留下一点记忆。就这样一路走下来到了今天，自己也从坚持之中受益匪浅。

在北京最初的两年中，我还利用自己的业余时间，为 ITPUB 的培训课程担任讲师，主讲过几十次各类数据库的技术课程，积累了大量的培训经验，这些经验使得即使在今天，我仍然很了解学员们的需求以及培训这个市场。

2006 年 8 月，我和很多朋友一起参加了“中国首届杰出数据库工程师评选”活动，并且获评为“十大杰出数据库工程师”之一，这是外界对我做出的一个非常积极的肯定（右图是北京大学教授-唐世渭老师为我颁奖的照片）。

这个阶段我可以作出的总结是：**积累知识，分享经验，收获快乐！**写作的过程是艰辛艰苦的，然而分享的收获会超出你的想象，能够帮助别人，分享有价值的经验实在是一件快乐的事情。我计划将这个工作一直坚持下去。

由于个人对于技术的执着和热爱，这么多年来，不管在怎样的工作岗位上，我从来没有停止过对于技术的研究与探索。刚开始在北京做 DBA 的工作时，经常为一个个技术问题废寝忘食，记得有一次在公司思考一个问题未果，吃饭时一直思索，思路顿开时，立即丢下饭不吃跑回去做实验来推理验证；有时候会持续很多年关注和跟踪某个技术问题，直到某一天豁然开朗，融会贯通。我相信在学习的过程中，**每个人都会在不同的阶段遇到自己的瓶颈，然而必须在山重水复之后才能有技进乎道的感觉，我相信所有的技艺在最后的层面上都会如此，而只有具备毅力与坚持者方能抵达。**

有一年我去兰州大学做技术交流，兰大的一位李老师对我说，最近看我网站上提到的学习方法等内容，感觉到一个字：虚！我当时跟他开玩笑说，我还有更虚无的 8 个字可以送给你，那就是：运用之妙，存乎一心。

这是玩笑，也不是玩笑，有时候对 Oracle 进行了深入的研究与探索之后，剩下的如何运用这些知识去解决问题，实际上是非常灵活的，很多时候简单的常规方法经过巧妙运用之后就可以化为神奇，发挥出你意想不到的作用。所以根本的，我们应该花力气去做的仍然是积累、深入、思考，然后才能在遇到问题时举重若轻、运用自如。

这些年在技术方面不断的努力带来的一个额外收获就是 Oracle 公司官方的认同，在 2007 年 3 月，我被 Oracle 公司授予 Oracle ACE 称号，进一步的在 2008 年 2 月，被 Oracle 公司授予 Oracle ACE Director（ACE 总监）称号，这是 Oracle 公司对 Oracle 公司之外的人所能授予的最高荣誉称号，在数据库领域，国内目前仅有 4 人获得该称号。

所有的这些积累，都是今天我尝试创业必不可少的重要条件！

我的太太

在一生中，很多重要的支持必不可少，我很幸运有我的太太坚定、理智的支持我所做的一切。她能够宽容我在网络上浪费的时光，她支持我坚持不懈的去写作技术书籍，她愿意陪我四处乱跑去出席各类的技术会议，这一切都使得我们的生活因为工作而丰富，从未因此而影响到家庭的和睦。

有一次去出席《程序员》杂志社举办的年终答谢晚宴，我带着我的太太 Julia 一起，几乎所有的技术人都是独自前来的，只有 Julia 自我介绍时说自己并不懂 IT 技术，是陪我来结识各路英雄的。我发言的时候说：**记得《东邪西毒》里面洪七说过的一句话“谁说不能带着老婆闯江湖？”，我无论走到哪里，都带着老婆一起闯荡，我的生活也就是她的生活。**现场的朋友们为这句话给了我很多的掌声。

在 2007 年 5 月，儿子出生前几个月，太太还陪着我去杭州参加阿里巴巴举办的侠客行大会，在几乎所有我的演讲现场，太太都是前排我最忠

实的听众和支持者，看到她我就会踏实而有信心。

说起来，我和太太也是通过网络上认识的，很偶然的通过孟静的博客链接到另外一个地方，在某篇文章下的留言里冲突起来，就此结识。后来在请周筠老师、孟静一起吃饭时，还专门感谢过她成就了我们的姻缘。

人生中充满了偶然，在偶然之中也存在必然。只要我们保有一颗好奇、敏感且善良的心灵，就始终能够体验到生命于我们的眷顾。

太太的善良、孝顺与宽容，让我在家庭上几乎没有遇到任何麻烦，可以集中精力去经营自己的事业，而在事业上，她又始终是我最坚定的支持者与拥护者，提醒我在顺境中不骄傲，鼓励我在逆境中不灰心，就这样一路走下来。

当然在家庭生活中，一些矛盾不可避免，可是我们总是会想起在一起的甜蜜时光，想起彼此的支持与爱护，这样就没有什么是不可以化解的，这样就不再忍心给对方伤害。太太定的规则是，一切的矛盾不能带过午夜，我是这一规则的坚定执行者。我们始终要明白的一点是，家庭的矛盾不能靠逻辑去推理，要靠爱去化解。

2007 年，我们的儿子出生，一个孩子会彻底改变你对于世界的观点，一个孩子会让你走路、工作想到他都可以笑出声来，我的妻子和儿子，是这么多年来我最为宝贵的收获。

单飞与创业

2008 年 2 月 20 日，我选择离开了服务多年的公司，出来开始自己的事业。

按照自我的评估，我本人并不是一个充满冒险精神的创业者，我其实更愿意做一个较为安稳的技术人员，能够认认真真做好自己的工作，在自己的价值不被低估的前提下，快乐的工作。这样的期望在职业发展中，很快遇到了瓶颈，我从一个工程师被逐渐推上了管理者的岗位，而慢慢的我认识到，这并不是我喜欢的位置，有越来越多的会议和跨部门的矛盾，有越来越多的管理和沟通工作而不是条理分明的技术，然而从技术走向管理在中国又是一个两难的选择，很多企业给技术人员提供的空间并不多，所以造成的局面往往是：如果不在岗位上提升，一个人的价值仿佛就变得有限。

如同第一次辞职一样，选择总是痛苦的，当我们面对熟悉的环境、稳定的工作时，做出变化时就可能会患得患失，这里唯一支持我的还是，多年不懈的学习与积累！而且，我们终究要明白：有付出才有收获，能舍弃才能获得。

在面对这次离开时，实际上我可以有两个选择：一是选择一家更好的公司，和许多志同道合的朋友一起，继续自己的职业生涯，而且已经有朋友多次真诚的向我发出了邀约；一是开始自己的事业，选择充满风险的单飞与创业生涯！

在这两个选择中，选择前者远比选择后者容易得多，可是我花了很长时间思考，自己更想要的是怎样一种生活？一个良好的企业平台会给个人提供更广阔的发展空间，而且除了工作之外，没有太多压力，会有满意的收入与可预期的未来；如果自己发展，一切就都变得不确定，唯一确定的是，可以获得充分的自由与自主，而这是我更加期待的。

走一条不可预期的路，对自己充满挑战，也充满乐趣，这体验不管成功还是失败，都将是全新的！就这样我最终还是选择了挑战自我。

有时候常常想，如果充分发挥自己的能力，自己到底能创造多少价值？现在我可以全力去寻找这个答案。

在单飞的第一年中，靠着朋友的推荐与自己多年的客户积累，顺利的展开了自己的事业里程。我最初从事的工作很简单，就是**基于 Oracle 数据库的顾问咨询、技术服务与培训**。这一工作已经有很多公司在做，而且面对的是惨烈的竞争，许多朋友说这条路不见得能够走得通，不过我固执的要试一试。

在开始之前，我对这个市场做过很多细致的调研，我发现很多经营多年的同类公司，已经慢慢丧失了活力以及技术力，靠技术服务的公司，

失去了技术跟进与理解，也就失去了价值。而我由于个人对于技术的爱好、个人的性格，我想我可以一直领导一个团队，认真、细致的坚持下去。

这就是我的切入点：**以更认真、更细致、更贴近革新的技术服务客户，真正以客户为中心，用心经营。**

这些话说起来并不难，但是落到实处却绝不容易，我经常和技术人员说的一句话就是：**要比客户更了解数据库的运行，给予的要永远比用户期待的还多！**

如果能做到这些，那么得到和留住用户就不是一件困难的事。

所以在未来公司发展的过程中，我最重要的工作就是坚持不懈的把这些想法落到实处，而且，从网络成长起来的我，**更希望这个全新的公司能够带有一些网络的、Web 2.0 的气息**，能够跟进时代与潮流，更好的为用户提供服务。

技术的出路

时至今天，IT 这个行业仍然是最为吸引毕业生的一个重要行业。记得多年前榕树下的一位朋友“落花如雨”说过一句话：喜欢这个行业，因为这个行业里汇聚了这个时代最聪明的人才与最快速增长的财富。

就因为这两点，众多的年轻人前仆后继的开始涌入这个圈子。那么然后，出路又在何方呢？一直以来大家都认为，程序员或者 IT 领域是年轻人的天下，因为这里有变幻迅速的技术和产品，而机遇和压力一直是呈正比增加的。

然而在中国，如果单靠工作来衡量，技术并不足以支撑每个人获得可观的财富，所以在谋求出路上，所有人都在尝试不同的道路。在常规的企业里，**技术人员要么成为管理者，通过职位的晋升来获得提高；要么成为企业技术的核心人员，靠技术赢得一切；要么走出去，靠一技之长来尝试开创事业。**当然我们也看到中国有很多优秀的企业也已经做的越来越好，为技术人员提供更好的待遇与更好的成长空间。

按照目前的市场行情（2010 年，我所了解的北京、上海、杭州等大城市的大致数据），就我所知的 Oracle 数据库从业领域，初级者的薪水可能在 3000 元/月左右，中级的 DBA 薪水可能在 5000 元/月~7000 元/月，而高级 DBA 的薪水范围可能在 8000 元/月~20000 元/月左右，至为出色的或者占有重要位置的薪水可能在 30000 元/月~50000 元/月左右（当然很多优秀的公司还会有可观的期权收益和其他奖励）。

那么一个入门者从初级到高级需要多长时间呢？我觉得可能需要大致 2~3 年的时间，可是即便在技术上做的较为出色，在当前社会不断攀升的房价、高昂的生活成本压力下，这样的收入水平离经济自由也还有相当的距离。

所以不管走哪一条路，这里面都有一句潜台词就是：你必须面临严峻的竞争，取得快速的成长！**张爱玲说过，成名要趁早。做技术的也是如此，成长越早越好，越快越好。**在经历了足够的积累和成长之后，在尽快到达天花板并且超越之后，你会发现前方供你选择的道路会更多、更宽广。

这个快速变化的时代给我的压迫感随时都在，时间与时机总是稍纵即逝，所以进入 IT 这个领域，注定我们要不断跋涉，不得停息。

创业的准备

不管做怎样一件事，我们都不能盲目，当你要开始创业时，要考虑的事情就更多，对我来说，**最重要的事情有两件：一是市场，二是人才。**

你所要从事的事业，市场在哪里，客户在哪里，如何让客户找到你，你如何找到客户，你的客户群能否支撑你的增长？如何汇聚更多的人才为客户服务？这些都是在创业前必须回答的问题。

作为数据服务领域的参与者，我在创业之前已经对这些问题有了基本的解答：

1. Oracle 数据库在全球的市场份额已经占到了 48.6%（根据 Gartner 2007 年市场数据），遥遥领先的居于第一位（占据第二位的 IBM db2 市场

份额为 22%左右）；根据经验及部分参考数据，Oracle 数据库在国内的市场占有率还要更高。

2. Oracle 的高份额市场占有率决定了服务市场的广泛存在。

3. 用户对于服务的认同近几年已经有了很大的提升，很多国际性大公司不断在强调以 Service 为中心的战略理念，而且服务在众多实践中的确能够为客户创造价值。

4. 数据服务与顾问公司在国外已经发展了多年，是成熟的商业模式；在国内也有很多公司在该领域深植多年，市场普遍存在。

市场存在了，那么你的市场在哪里呢？你靠什么吸引客户？这些问题对一个公司来说是更直接面对的问题：

1. 多年的积累使得我了解这个市场并且有机会接触客户，能够接触到客户，让客户了解你，是非常重要的企业公关。我的 Eygle.com 也成为了非常有效的信息发布平台。

2. 通过培训、服务积累的口碑，**良好的服务与口碑是企业生存的关键。**

3. 不经意的回头，虽然没有刻意的准备，但是多年的职业生涯、网络生涯以及著作培训经历都为我积累了相当的客户资源。

4. 我相信自己团队的实力、坚持与专注。**唯有专注、专业，才能为客户提供保障、创造价值，**而这正是我们的核心竞争力。

当你明白了自己拥有什么之后，就会充满信心，无所畏惧，只要我们能够提供好的产品、好的服务，接下来就可以满怀信心的迎接用户的挑战！

说了那么多，其实有一个核心的思想就是：**积累非常重要！**不管在哪一个行业，做什么工作，如果你能够利用自己以前的学习、工作经验，发挥自己积累的技能，那么做事情就会事半功倍；而如果你试图进入一个全新的领域，那么一定要做好充足的功课才行。

曾经有一个 IT 圈的朋友在听过某个销售的成功宣讲之后，立即热血沸腾，向我表示他找到了人生的新的方向，决定立即转行。我当时对他说，如果你觉得这是你的兴趣与志向所在，那么在过去的 20 多年间，你为此做过哪些准备？你学习过什么有关销售的课程？市场营销学之父是何人？我们承认有很多人可以无师自通、自学成才，可是那也必然是经过无数困惑与思考而得来的成绩，没有无缘无故的成功，一时的热情必须要靠理智来支撑。可惜的是，我最终没能说服那位朋友。

总结一下这些年走过的路，零零碎碎有一些话可以和大家分享：

1. **勤奋、坚持，**这两点非常重要，当然如果能够找到自己的兴趣，作为职业，用正确的方法，走正确的路，那么取得成绩是早晚的事情，我经常写给读者的座右铭就是：天道酬勤！

2. **在看不清方向的时候，低下头来把手中的工作做好**

3. 向他人学习，向聪明人学习，借鉴成功者、同行者的经验非常重要。

4. 敞开心胸，平淡看得失。

5. 在正确的时间做正确的事，比如结婚、生子。

6. 行动有时候比思想更重要。

在本文的最后，我还想说几句的是，**除了工作之外，不要忘记了生活，没有什么比生活更重要的，家是世界上最重要的地方。**

想一想你匆忙的脚步是否已经很久没有为一览风景而停留？想一想你是否已经很久没有陪家人与朋友出游谈天？要记住我们是为了生活而工作，而不是为了工作而生活。在 IT 圈子的朋友们尤其如此，高强度的工作，大量的加班，黑白颠倒，这一切绝不是生活的目标。

在我的一本书的结尾，我写过如下一段话，与大家分享：

2008 年的 9 月 21 日～9 月 25 日，应 Oracle 公司的邀请，我到旧金山参加了 Oracle 2008 Open World 大会。你能猜到大会上最打动我的一句话是什么？

不是 Oracle 发布的 Exadata Programmable Storage Server 也不是 HP Oracle Database Machine（Oracle 软件公司划时代发布的两款硬

件产品），而是 Larry 在 Keynote 上发表的演讲时讲到的一段话，他说：在过去七八年间，我的主要工作是去赢得美洲杯（American's Cup），然后才是在 Oracle 的工作。

不管 Larry 想传达的意思是什么，我的理解是，能够快乐的做自己喜爱的事情，才是人生最值得追求的，而工作不过是生活的另一面。

工作是永无穷尽的，而生活则是有限的，快乐的生活比什么都要重要。

作为芸芸众生中的普通一员，在为理想与未来奋斗之余，让我们用更多一点的时间去经历更加快乐的生活吧！

最后祝愿我的恩墨科技能够越走越好，也祝愿所有的朋友们都能够走向自己开阔的前方！路在脚下，让我们一起起步前行吧！

盖国强（Eygle）

2009-5-13 初稿

2010-2-28 改毕

Part 2 DBA手记

1

Eygle（盖国强）

Eygle 的 DBA 工作手记

我一直习惯记录分享，也一直珍视来自实践的经验总结，这一章中记录了最近一年以来的工作点滴、领悟，与大家分享。

承前启后——Failed Login Count 带来的性能问题

在《Oracle DBA 手记》一书的最后一则案例中，曾经提到关于 Failed Login Count 的问题，简要的引用一下用来承前启后。

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有了限制。在用户的 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
```

那么这里的 10 次登陆失败计数是如何完成的呢？查看底层表 USER$的字段，其中 LCOUNT 字段就是用来完成这个功能的：

```
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
......
```

可以通过 sql.bsq 文件来进一步确认，这个文件提示 lcount 正是失败的登录尝试计数(count of failed login attempts)，如图 1-1 所示：

```
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_VC$2"),                        /* external username */
  spare1        number, /* used for schema level supp. logging: see ktscts.h */
```

图 1-1　lcount 为失败的登录尝试计数

在最近的一次客户数据库性能优化中，再次遇到了类似的一个案例。这是一个 Oracle Database 11g 11.1.0.6 的数据库环境，如图 1-2 所示：

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
GVDB	2402750861	gvdb1	1	12-7月 -10 22:07	11.1.0.6.0	YES

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
CARDCLUSTER1	Microsoft Windows 64-bit for AMD	16	4		8.00

图 1-2　Oracle Database 11g 11.1.0.6 的数据库环境

在这个数据库的 SQL ordered by Gets 诊断中，发现了一条可疑的 SQL，如图 1-3 所示，这个 SQL 的逻辑读排在第三位，占整体数据库逻辑读的 14.23%，其 SQL Module 是：Oracle Enterprise Manager.Metric Engine。

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1,832,581	40	45,814.53	19.59	166.77	361.69	dsp6uxu05kcca6		DECLARE job BINARY_INTEGER := ...
1,466,068	32	45,814.63	15.68	133.81	295.07	baq98krgfyf2		UPDATE CARD_MEMC SET CARD_STAT...
1,353,730	4	338,432.50	14.47	14.23	61.44	d15cdr0at3vtp	Oracle Enterprise Manager.Metric Engine	SELECT TO_CHAR(current_timesta...
344,776	1	344,776.00	3.69	1.27	4.51	akuxnr9ws30x2	ORACLE.EXE	SELECT "A1"."CARD_FACADE_CD", ...
299,070	2	149,535.00	3.20	2.25	10.40	[illegible]	ORACLE.EXE	SELECT "A1"."CARD_FACADE_CD", ...
274,014	1	274,014.00	2.93	19.58	21.26	f3qt2kgjrgbg1	exp.exe	SELECT /*+NESTED_TABLE_GET_REF...
254,373	1	254,373.00	2.72	237.52	243.60	0m2dta30frbuh		DECLARE job BINARY_INTEGER := ...
219,956	40	5,498.90	2.35	2.22	3.85	hjjc9c4gg99jj	OEM.CacheModelWaitPool	BEGIN EMDW_LOG.set_context(MGMI...
184,579	984	187.58	1.97	1.94	2.14	74t8gppdnm1cw	ORACLE.EXE	SELECT SUM("A1"."CBY_TOTAL_DIS...
131,274	121	1,084.91	1.40	8.16	13.78	8qvch1xu9ca3g		DECLARE job BINARY_INTEGER := ...

图 1-3　一条可疑的 SQL

在这里我想强调一点的是，很多时候 DBA 在遇到数据库系统自身调用的内部 SQL 时，常常下意识的选择回避，认为数据库的自身功能不会存在太大的问题，而事实往往相反。我的一个座右铭是，决不放过任何一句可疑的 SQL 代码。这里的 Module 显示，该 SQL 是 OEM 的 Metric 引擎发起的，一个

数据库的内部功能在任何时候都不应该消耗大量的系统资源。

格式化一下该 SQL 代码得到如下完整输出：

```
SELECT TO_CHAR(current_timestamp AT TIME ZONE 'GMT',
               'YYYY-MM-DD HH24:MI:SS TZD') AS curr_timestamp,
       COUNT(username) AS failed_count
  FROM sys.dba_audit_session
 WHERE returncode != 0
   AND TO_CHAR(timestamp, 'YYYY-MM-DD HH24:MI:SS') >=
       TO_CHAR(current_timestamp - TO_DSINTERVAL('0 0:30:00'), 'YYYY-MM-DD HH24:MI:SS')
```

从这段代码可以看到，该 SQL 是用于监控和计算失败登陆次数（failed_count）的，这一监控结果可以在某用户发生失败登陆尝试时给出告警。这里的 DBA_AUDIT_SESSION 用于记录审计对于数据库所有的 CONNECT 和 DISCONNECT 操作，底层表为 AUD$。在 Database / Grid Control 中如果启用了 Failed Login Count Metric 监控，就可能遇到这个问题，一个建议的解决方案就是停用这个监控。

但是为什么会出现这样的问题呢？检查这个 SQL 的执行计划，可以发现一些端倪，如图 1-4 所示，对 AUD$表的访问出现了一个全表扫描，然后进行 NESTED LOOPS OUTER 连接。

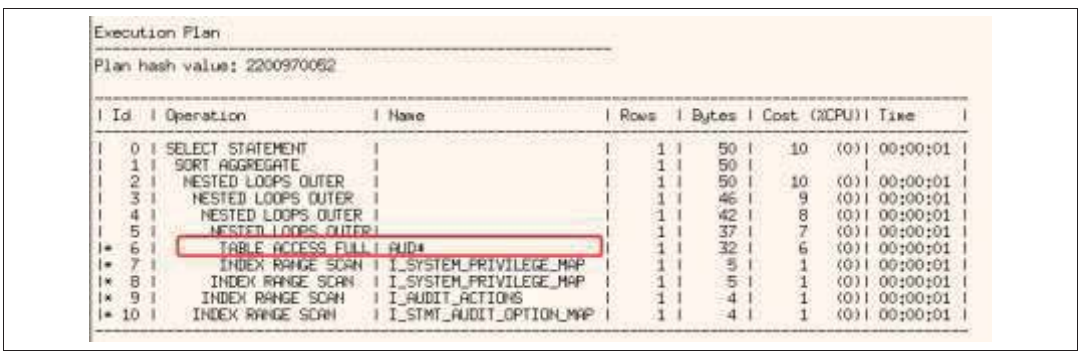


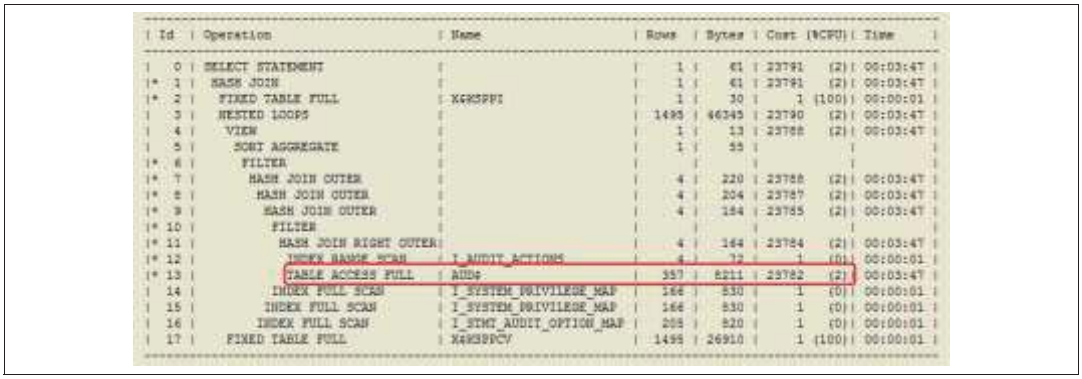
图 1-4　对 AUD$表的全表扫描

如果此处 AUD$表的数据量较大，就可能产生非常大量的逻辑读，影响性能，恰恰 AUD$表经常会存在大量的数据，这就是原因所在。在新版本中，Oracle 正在尝试通过对该表进行分区，提升数据清理效率，并通过适当的索引提升访问性能。

对于 DBA_AUDIT_SESSION 的各种访问都可能遇到类似的问题，另外一则报告的问题 SQL 如下：

```
select a.CURRENT_AUDIT_SETTING, b.TOTAL_SUCC_LOGINS
 from (select value as CURRENT_AUDIT_SETTING
         from v$parameter
        where name = 'audit_trail') a,
     (select count(*) as TOTAL_SUCC_LOGINS
        from dba_audit_session
       where (action_name = 'LOGON' and returncode = 0 or
              action_name like 'LOGOFF%')
         and timestamp > EMIP_BIND_START_DATE
         and timestamp < EMIP_BIND_END_DATE) b
```

这段 SQL 在客户环境中的执行计划如图 1-5 所示，类似的执行计划和全表访问，导致了 SQL 执行成本的上升，极大的影响了性能：



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	61	23791 (2)	00:03:47
* 1	HASH JOIN		1	61	23791 (2)	00:03:47
* 2	FIXED TABLE FULL	X$KSPPI	1	30	1 (100)	00:00:01
3	NESTED LOOPS		1495	46345	23790 (2)	00:03:47
4	VIEW		1	13	23788 (2)	00:03:47
5	SORT AGGREGATE		1	55		
* 6	FILTER					
* 7	HASH JOIN OUTER		4	220	23788 (2)	00:03:47
* 8	HASH JOIN OUTER		4	204	23787 (2)	00:03:47
* 9	HASH JOIN OUTER		4	184	23785 (2)	00:03:47
* 10	FILTER					
* 11	HASH JOIN RIGHT OUTER		4	164	23784 (2)	00:03:47
* 12	INDEX RANGE SCAN	I_AUDIT_ACTIONS	4	72	1 (0)	00:00:01
* 13	TABLE ACCESS FULL	AUD$	357	8211	23782 (2)	00:03:47
14	INDEX FULL SCAN	I_SYSTEM_PRIVILEGE_MAP	166	830	1 (0)	00:00:01
15	INDEX FULL SCAN	I_SYSTEM_PRIVILEGE_MAP	166	830	1 (0)	00:00:01
16	INDEX FULL SCAN	I_STMT_AUDIT_OPTION_MAP	205	820	1 (0)	00:00:01
17	FIXED TABLE FULL	X$KSPPCV	1495	26910	1 (100)	00:00:01

图 1-5　全表扫描 AUD$的执行计划

任何时候，我们都应当对系统的功能与 SQL 心存警惕，不能掉以轻心。

OEM 罪几何？——空间监控的性能问题

在某金融行业用户的 ERP 数据库中，一个小时的采样报告，位于 Elapsed Time 消耗排行第二位的 SQL 消耗了 19.41%的 DB Time（如图 1-6 所示），该 SQL 同样是 OEM 发出来的，其 SQL Module 是 Oracle Enterprise Manager.Metric Engine，这个 SQL 每次执行需要 245.77 秒的时间，是极其缓慢的，数据库环境是 Oracle Database 10g 10.2.0.4 版本。

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
538	513	324,406	0.00	21.25	[redacted]	GLUEENT	SELECT 'Y' FROM DUAL WHERE EXI...
492	39	2	245.77	19.41	4d6m2q3ngjcv9	Oracle Enterprise Manager.Metric Engine	insert into mgmt_db_size_gtt ...
362	348	1	362.15	14.30	[redacted]	GLUEENT	SELECT PERIOD_NAME, BATCH_NAME...
258	26	1,169	0.22	10.19	[redacted]	fnd.framework.navigate.server.OANavigatePortletA	BEGIN :1 := FND_AOLJ_UTIL.is_V...
237	12	6	39.55	9.37	[redacted]	fnd.wf.worklist.server.WorklistAM.R	SELECT TRANSACTION_ID FROM ICX...
174	11	1	173.86	6.87	[redacted]	FAS400	SELECT cc SEGMENT1 COMP_CODE ...
167	94	1	166.62	6.58	[redacted]	Oracleware4	SELECT (101360 as E101360, (1...

图 1-6　Elapsed Time SQL 列表

该 SQL 的文本内容是：

```
insert into mgmt_db_size_gtt
 select tablespace_name, NVL(sum(bytes) / 1048576, 0) sz
   from sys.dba free space
  group by tablespace_name
```

这显然是通过 dba_free_space 来计算各表空间的 Free 空间的，这个 SQL 同样是 OEM 发出的，其执行计划可以通过 AWR 获得：

```
SQL> select * from table(dbms_xplan.display_awr('4d6m2q3ngjcv9'));
insert into mgmt_db_size_gtt  select  tablespace_name,NVL(sum(bytes)/1048576, 0) sz
from sys.dba_free_space  group by tablespace_name

Plan hash value: 2413628916
```

```
-------------------------------------------------------------------------------------
| Id | Operation                      | Name             | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------------------
|  0 | INSERT STATEMENT               |                  |       |       |   82  |
|  1 | SORT GROUP BY                  |                  |  189  |  5670 |   82  |
|  2 | VIEW                           | DBA_FREE_SPACE   |  189  |  5670 |   35  |
|  3 | UNION-ALL                      |                  |       |       |       |
|  4 | NESTED LOOPS                   |                  |   68  |  2584 |    6  |
|  5 | NESTED LOOPS                   |                  |   68  |  2176 |    6  |
|  6 | TABLE ACCESS FULL              | TS$              |    1  |    23 |    5  |
|  7 | TABLE ACCESS CLUSTER           | FET$             |  136  |  1224 |    1  |
|  8 | INDEX UNIQUE SCAN              | I_FILE2          |    1  |     6 |       |
|  9 | NESTED LOOPS                   |                  |  119  |  5355 |    6  |
| 10 | NESTED LOOPS                   |                  |  119  |  4641 |    6  |
| 11 | TABLE ACCESS FULL              | TS$              |   19  |   551 |    5  |
| 12 | FIXED TABLE FIXED INDEX        | X$KTFBFE (ind:1) |    6  |    60 |    1  |
| 13 | INDEX UNIQUE SCAN              | I_FILE2          |    1  |     6 |       |
| 14 | NESTED LOOPS                   |                  |    1  |   126 |   20  |
| 15 | NESTED LOOPS                   |                  |    1  |   120 |   20  |
| 16 | NESTED LOOPS                   |                  |    1  |    68 |    3  |
| 17 | TABLE ACCESS FULL              | RECYCLEBIN$      |    1  |    39 |    2  |
| 18 | TABLE ACCESS CLUSTER           | TS$              |    1  |    29 |    1  |
| 19 | INDEX UNIQUE SCAN              | I_TS#            |    1  |       |       |
| 20 | FIXED TABLE FIXED INDEX        | X$KTFBUE (ind:1) |  100  |  5200 |   17  |
| 21 | INDEX UNIQUE SCAN              | I_FILE2          |    1  |     6 |       |
| 22 | NESTED LOOPS                   |                  |    1  |    81 |    3  |
| 23 | NESTED LOOPS                   |                  |    1  |    58 |    2  |
| 24 | NESTED LOOPS                   |                  |    1  |    52 |    2  |
| 25 | TABLE ACCESS FULL              | RECYCLEBIN$      |    1  |    39 |    2  |
| 26 | TABLE ACCESS CLUSTER           | UET$             |    1  |    13 |       |
| 27 | INDEX UNIQUE SCAN              | I_FILE#_BLOCK#   |    1  |       |       |
| 28 | INDEX UNIQUE SCAN              | I_FILE2          |    1  |     6 |       |
| 29 | TABLE ACCESS CLUSTER           | TS$              |    1  |    23 |    1  |
| 30 | INDEX UNIQUE SCAN              | I_TS#            |    1  |       |       |
-------------------------------------------------------------------------------------
```

通过执行计划可以看到，在 Oracle Database 10g 引入了回收站功能后，会将回收站（RECYCLEBIN$）中的空间计算为自由空间，加入到 dba_free_space 字典中。

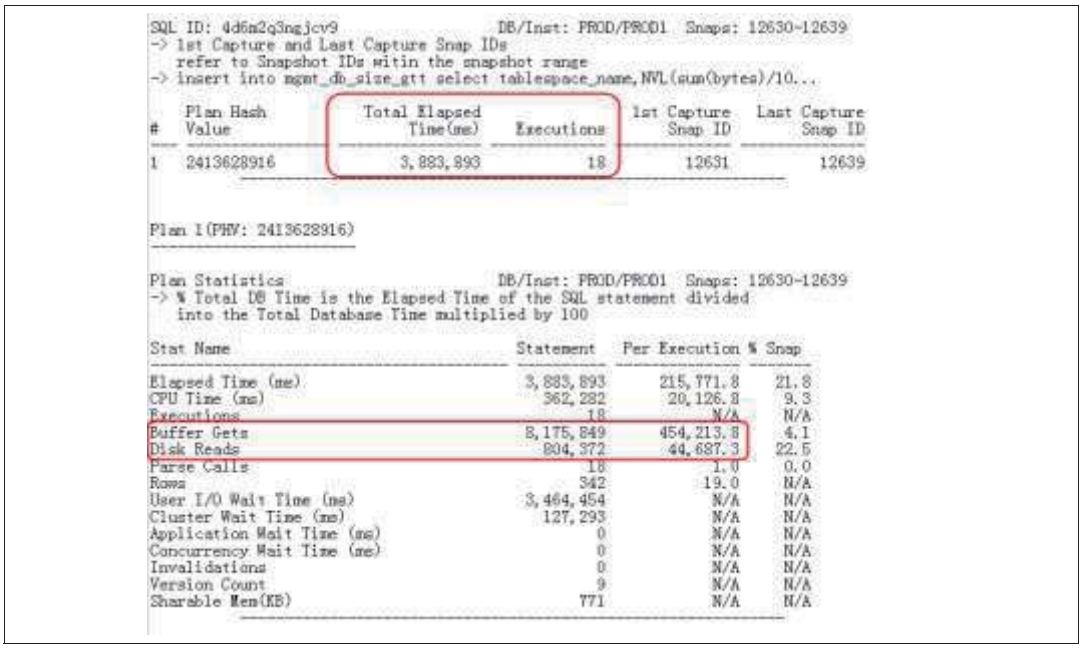
如果数据库中存在大量的回收站对象，则这部分回收站空间的计算将会极为耗时，在这个数据库环境中，共有 5 万多个回收站对象：

```
SQL> select count(*) from RECYCLEBIN$;

  COUNT(*)
----------
     51986
```

清理这些回收站对象可以大幅提升这个 SQL 查询的性能，在 OEM 中禁用这个 Metric 监控则可以彻底去除这个 SQL 访问。

在 SQL 报告中，显示了该 SQL 的详细信息（如图 1-7 所示）：



```
SQL ID: 4d6m2q3ngjcv9                    DB/Inst: PROD/PROD1  Snaps: 12630-12639
-> 1st Capture and Last Capture Snap IDs
   refer to Snapshot IDs within the snapshot range
-> insert into mgmt_db_size_gtt select tablespace_name,NVL(sum(bytes)/10...

                                Total Elapsed                    1st Capture   Last Capture
#   Plan Hash Value               Time(ms)      Executions         Snap ID       Snap ID
---  ---------------  ---------------  ---------  ------------  ------------
1   2413628916             3,883,893          18         12631         12639

Plan 1(PHV: 2413628916)
-----------------

Plan Statistics                          DB/Inst: PROD/PROD1  Snaps: 12630-12639
-> % Total DB Time is the Elapsed Time of the SQL statement divided
   into the Total Database Time multiplied by 100

Stat Name                                  Statement   Per Execution % Snap
----------------------------------------  ----------  --------------  ------
Elapsed Time (ms)                          3,883,893      215,771.8    21.8
CPU Time (ms)                                362,282       20,126.8     9.3
Executions                                        18            N/A     N/A
Buffer Gets                                8,175,849      454,213.8     4.1
Disk Reads                                   804,372       44,687.3    22.5
Parse Calls                                       18            1.0     0.0
Rows                                             342           19.0     N/A
User I/O Wait Time (ms)                    3,464,454            N/A     N/A
Cluster Wait Time (ms)                       127,293            N/A     N/A
Application Wait Time (ms)                         0            N/A     N/A
Concurrency Wait Time (ms)                         0            N/A     N/A
Invalidations                                      0            N/A     N/A
Version Count                                      9            N/A     N/A
Sharable Mem(KB)                                 771            N/A     N/A
```

图 1-7　SQL 报告的详细信息

在$ORACLE_HOME/rdbms/admin/catspace.sql 脚本中，可以找到创建 DBA_FREE_SPACE 视图的脚本：

```
create or replace view DBA_FREE_SPACE
   (TABLESPACE_NAME, FILE_ID, BLOCK_ID,BYTES, BLOCKS, RELATIVE_FNO)
as
select ts.name, fi.file#, f.block#, f.length * ts.blocksize, f.length, f.file#
from sys.ts$ ts, sys.fet$ f, sys.file$ fi
where ts.ts# = f.ts# and f.ts# = fi.ts# and f.file# = fi.relfile# and ts.bitmapped = 0
union all
select /*+ ordered use_nl(f) use_nl(fi) */ ts.name, fi.file#, f.ktfbfebno,
     f.ktfbfeblks * ts.blocksize, f.ktfbfeblks, f.ktfbfefno
from sys.ts$ ts, sys.x$ktfbfe f, sys.file$ fi
where ts.ts# = f.ktfbfetsn and f.ktfbfetsn = fi.ts# and f.ktfbfefno = fi.relfile#
  and ts.bitmapped <> 0 and ts.online$ in (1,4) and ts.contents$ = 0
union all
select /*+ ordered use_nl(u) use_nl(fi) */ ts.name, fi.file#, u.ktfbuebno,
     u.ktfbueblks * ts.blocksize, u.ktfbueblks, u.ktfbuefno
from sys.recyclebin$ rb, sys.ts$ ts, sys.x$ktfbue u, sys.file$ fi
where ts.ts# = rb.ts# and rb.ts# = fi.ts# and u.ktfbuefno = fi.relfile#
  and u.ktfbuesegtsn = rb.ts# and u.ktfbuesegfno = rb.file# and u.ktfbuesegbno = rb.block#
  and ts.bitmapped <> 0 and ts.online$ in (1,4) and ts.contents$ = 0
union all
select ts.name, fi.file#, u.block#,u.length * ts.blocksize, u.length, u.file#
from sys.ts$ ts, sys.uet$ u, sys.file$ fi, sys.recyclebin$ rb
where ts.ts# = u.ts# and u.ts# = fi.ts# and u.segfile# = fi.relfile#
  and u.ts# = rb.ts# and u.segfile# = rb.file#
  and u.segblock# = rb.block# and ts.bitmapped = 0
/
```

以上脚本中，后面两个 UNION ALL 查询块是 Oracle 10g 引入的，并且为了修正这个视图带来的 Bug，Oracle 一直不停的在改进视图语句。注意视图中 Hints 的制定对于执行计划的强制影响。

我们要时刻牢记的是：**当 Oracle 引入了某个新功能时，同时也会引入很多问题，所以在使用新功能、新特性时要加强监控，及时发现和解决可能出现的问题。**

Grid Control 的必要监控——进程累积导致的宕机

某用户 Oracle Database 10g 10.2.0.4 数据库，运行在 HP 平台上，数据库出现大量系统累积进程，最后导致数据库挂起，影响了业务使用，造成了严重故障。

在数据库的进程记录信息中，我们发现大量的 crs_stat.bin –t 进程，这些进程部分是 Grid Control 调度的监控，另外一部分来自用户自定制的监控脚本，这些脚本中最早未能完成的脚本时间为 03:32:48。也就是说，从这一时间起，数据库出现异常导致大量 crs_stat 进程累积，最后耗尽资源，系统挂起。

```
oracle  7085  7084  0 03:52:49 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  9016  9015  0 03:57:48 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  1194  1193  0 03:37:49 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  7237  7231  0 16:31:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  5119  5118  0 03:47:48 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  6325  6319  0 03:51:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle 25629 25619  0 04:41:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle 28872 28871  0 03:32:48 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle  2328  2322  0 03:41:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle 11040 11039  0 04:02:49 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle 10294 10288  0 04:01:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
oracle 29596 29590  0 04:51:00 ? 0:00 /u01/app/oracle/product/10.2.0/crs/bin/crs_stat.bin -t
```

那么是否这些进程就是罪魁祸首呢?

除了这些进程，系统中还累积了大量的 racgmain check 进程，这些进程自 Mar 14 日已经开始累积，最终数量达到了 400 个左右：

```
oracle 21315  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 16268  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 27453  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 12214  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  6824  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  8249  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  1884  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 12224  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  8232  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 11227  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 26275  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle 16272  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  6829  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
oracle  1889  1  0  Mar 14  ?   0:00 /u01/app/oracle/product/10.2.0/db_1/bin/racgmain check
```

正常情况下，在每次检测之后 racgmain 进程应当退出，但是在故障出现时，这些进程没有退出，

而且这种情况持续了多天。这说明 racgmain 的大量异常累积及之后的 crs_stat 检测异常是导致最终故障的来龙去脉，而 racgmain 的异常是主要起因。Oracle 的 Bug 6196746 说明了这个问题。可以通过如下步骤修复这个问题：

```
1. Make a copy of racgwrap located under $ORACLE_HOME/bin and $CRS_HOME/bin on ALL Nodes
2. Edit the file racgwrap and modify the last 3 lines from:
~~~
$ORACLE_HOME/bin/racgmain "$@"
status=$?
exit $status

to:

# Line added to fix for Bug 6196746
exec $ORACLE_HOME/bin/racgmain "$@"
~~~
3. Kill all the orphan racgmain processes running.
$ ps -ef|grep "racgmain check"
oracle 18701 1 0 Aug 1 ? 0:00 /oracle/product/10.2.0/database/bin/racgmain check
oracle 14653 1 0 Aug 1 ? 0:00 /oracle/product/10.2.0/database/bin/racgmain check
oracle 24517 1 0 Aug 1 ? 0:00 /oracle/product/10.2.0/database/bin/racgmain check

$ kill -9 <PID of racgmain>
```

注意，数据库在运行过程中，会调用该系统$ORACLE_HOME/bin 和$CRS_HOME/bin 下的 racgwrap 脚本，进而执行 racgmain 检测，这两个脚本应该同时修正，而在客户系统中我们发现这两个目录下的脚本并不一致，这可能是之前客户修正时疏忽遗漏所致：

```
db1::/u01/app/oracle/product/10.2.0/crs/bin>diff racgwrap $ORACLE_HOME/bin/racgwrap
3c3
< # Copyright (c) 2001, 2008, Oracle and/or its affiliates. All rights reserved.
---
> # Copyright (c) 2001, 2007, Oracle. All rights reserved.
9c9
< ORACLE_HOME=/u01/app/oracle/product/10.2.0/crs
---
> ORACLE_HOME=/u01/app/oracle/product/10.2.0/db_1
12c12
< ORACLE_BASE=/u01/app/oracle/product/10.2.0/crs
---
> ORACLE_BASE=/u01/app/oracle/product/10.2.0/db_1
62c62,64
< exec $ORACLE_HOME/bin/racgmain "$@"
---
> $ORACLE_HOME/bin/racgmain "$@"
> status=$?
> exit $status
```

在检查 crsd.log 日志文件中，我们可以发现大量如下错误提示：

```
2010-03-14 17:31:21.682: [ CRSEVT][67763] CAAMonitorHandler :: 0:Action Script
/u01/app/oracle/product/10.2.0/crs/bin/racgwrap(check) timed out for ora.xgp4.db!
```

```
(timeout=600)
2010-03-14 17:31:21.683: [  CRSAPP][67763] CheckResource error for ora.xgp4.db error code = -2
2010-03-14 17:37:32.382: [  CRSEVT][67784] CAAMonitorHandler :: 0:Could not join
/u01/app/oracle/product/10.2.0/db_1/bin/racgwrap(check)
category: 1234, operation: scls_process_join, loc: childcrash, OS error: 0, other: Abnormal
termination of the child

2010-03-14 17:37:32.382: [  CRSEVT][67784] CAAMonitorHandler :: 0:Action Script
/u01/app/oracle/product/10.2.0/db_1/bin/racgwrap(check) timed out for ora.xgp4.cmp.cs!
(timeout=600)
2010-03-14 17:37:32.382: [  CRSAPP][67784] CheckResource error for ora.xgp4.cmp.cs error code
= -2
2010-03-14 17:47:32.093: [  CRSEVT][67819] CAAMonitorHandler :: 0:Could not join
/u01/app/oracle/product/10.2.0/db_1/bin/racgwrap(check)
category: 1234, operation: scls_process_join, loc: childcrash, OS error: 0, other: Abnormal
termination of the child
```

值得注意的是，在以上提示中，Oracle 数据库会调用$ORACLE_HOME/bin/racgwrap 和$CRS_HOME/bin/racgwrap 两个脚本来执行检查，来自$ORACLE_HOME/bin/racgwrap 的脚本未修正，会导致进程的异常挂起。

这个 Bug 在 CRS 的 PSU 中被修正，根据环境检查，在 Patch 应用中可能出现问题，导致脚本错误，当数据库调用$ORACLE_HOME/bin/racgwrap 进程时，就可能出现问题。

图 1-8 是 CRS 10.2.0.4 中相关的 Bug 修正列表。

图 1-8　CRS 10.2.0.4 Bundle Patch 中相关的 Bug 修正列表

而为何累计了大量 racgmain 进程，crs_stat 操作无法返回结果，这和 HP-UX Itanium 平台的 Bug 有关，根据 Metalink 的 883801.1 文档记录应当应用操作系统 PHKL_40208 补丁集。**并且应当检查以下补丁或替代补丁已经被应用：**PHKL_38715、PHCO_38837、PHKL_38623、PHKL_38733, PHKL_38715, PHKL_38762。

这个案例带给我们的经验是：

1．在安装 CRS 时，应当应用最新的 Patch Bundle。

2．应当监控 CRS 日志，以及时发现其中的错误或告警信息。

3．应当部署操作系统的进程监控，以便帮助我们及早发现类似进程累积的问题。

4．Grid Control 或 Database Control 存在较多 Bug，部署后应该密切进行监控。

强化的监控和管理是及时发现，快速解决问题的根本，在数据库管理中必须加强。

DBA 诊断利器——Event 10046 和 10053

一次某优化工具厂商的朋友，发来一个案例请求协助诊断，朋友的优化工具在客户的环境中执行某个 SQL 查询时，需要 10 分钟时间才能出结果，这是无法接受的，而同样的查询在其他环境上都可以快速的获得输出结果，数据库环境是 9.2.0.8。

首先我获得了一个 10046 跟踪文件，通过 tkprof 格式化之后，这个 SQL 的输出结果展现出来。

首先该 SQL 代码如图 1-9 所示：

```
SELECT B.OWNER, B.CONSTRAINT_NAME, B.CONSTRAINT_TYPE, B.OWNER, B.TABLE_NAME,
  A.OWNER, A.CONSTRAINT_NAME, A.OWNER, A.TABLE_NAME
FROM
 SYS.DBA_CONSTRAINTS A, SYS.DBA_CONSTRAINTS B, SYS.DBA_OBJECTS O WHERE
  A.CONSTRAINT_TYPE='R' AND A.R_CONSTRAINT_NAME=B.CONSTRAINT_NAME AND
  A.R_OWNER=B.OWNER  AND O.OWNER=A.OWNER AND O.OBJECT_NAME=A.TABLE_NAME AND
  O.OBJECT_TYPE='TABLE'  AND B.OWNER='BOM' AND B.TABLE_NAME=
  'BOM_DEPARTMENT_CLASSES' ORDER BY B.OWNER, B.TABLE_NAME

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.12       0.16          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1    403.63     607.11       9585   15568040          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        3    403.75     607.27       9585   15568040          0           0
```

图 1-9　SQL 代码及执行统计信息

该段 SQL 的 Elapsed 时间超过了 600 秒，Query 模式逻辑读也非常高，对于一个优化工具来说显然是不可接受的。

接下来的跟踪文件中显示了 SQL 的执行计划：

```
   Rows  Row Source Operation
------- ---------------------------------------------------
      0  NESTED LOOPS OUTER
      0   NESTED LOOPS OUTER
      0    NESTED LOOPS OUTER
      0     NESTED LOOPS OUTER
      0      NESTED LOOPS OUTER
      0       NESTED LOOPS
      0        NESTED LOOPS OUTER
      0         NESTED LOOPS
      0          NESTED LOOPS OUTER
      0           NESTED LOOPS OUTER
      0            NESTED LOOPS
      0             NESTED LOOPS
      0              NESTED LOOPS
1935557               NESTED LOOPS
   2863                NESTED LOOPS
   2863                 NESTED LOOPS
   2863                  NESTED LOOPS
```

```
      1              NESTED LOOPS
      1               TABLE ACCESS BY INDEX ROWID USER$
      1                INDEX UNIQUE SCAN I_USER1 (object id 44)
      1               TABLE ACCESS BY INDEX ROWID USER$
      1                INDEX UNIQUE SCAN I_USER1 (object id 44)
   2863              TABLE ACCESS FULL CDEF$
   2863             TABLE ACCESS BY INDEX ROWID CON$
   2863              INDEX UNIQUE SCAN I_CON2 (object id 49)
   2863            TABLE ACCESS CLUSTER USER$
   2863             INDEX UNIQUE SCAN I_USER# (object id 11)
1935557           VIEW
1935557            UNION-ALL PARTITION
1935557             FILTER
1935557              NESTED LOOPS
   2863               TABLE ACCESS BY INDEX ROWID USER$
   2863                INDEX UNIQUE SCAN I_USER1 (object id 44)
1935557               TABLE ACCESS FULL OBJ$
      0              TABLE ACCESS BY INDEX ROWID IND$
      0               INDEX UNIQUE SCAN I_IND1 (object id 39)
      0              FILTER
      0               NESTED LOOPS
      0                TABLE ACCESS BY INDEX ROWID USER$
      0                 INDEX UNIQUE SCAN I_USER1 (object id 44)
      0                INDEX RANGE SCAN I_LINK1 (object id 113)
      0           TABLE ACCESS BY INDEX ROWID CON$
1935557            INDEX UNIQUE SCAN I_CON2 (object id 49)
      0          TABLE ACCESS BY INDEX ROWID CON$
      0           INDEX UNIQUE SCAN I_CON1 (object id 48)
      0         TABLE ACCESS BY INDEX ROWID CDEF$
      0          INDEX UNIQUE SCAN I_CDEF1 (object id 50)
      0         TABLE ACCESS BY INDEX ROWID CON$
      0          INDEX UNIQUE SCAN I_CON2 (object id 49)
      0        TABLE ACCESS CLUSTER USER$
      0         INDEX UNIQUE SCAN I_USER# (object id 11)
      0       TABLE ACCESS BY INDEX ROWID OBJ$
      0        INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0       INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0      TABLE ACCESS BY INDEX ROWID OBJ$
      0       INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0     TABLE ACCESS BY INDEX ROWID OBJ$
      0      INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0    TABLE ACCESS CLUSTER USER$
      0     INDEX UNIQUE SCAN I_USER# (object id 11)
      0    INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0   TABLE ACCESS BY INDEX ROWID OBJ$
      0    INDEX UNIQUE SCAN I_OBJ1 (object id 36)
      0  TABLE ACCESS CLUSTER USER$
      0   INDEX UNIQUE SCAN I_USER# (object id 11)
```

以上执行计划中，最可疑的部分是对于 OBJ$的全表扫描，这个环节的行数返回有 1 935 557 行，这个量级一直向上传递，所以我们首先怀疑这里的执行计划选择错误，如果选择索引，执行性能肯定会有极大的不同。

可是 10046 的跟踪事件显示的信息比较有限，不能够准确定位错误的原因，我请朋友通过 10053 事件来生成一个执行计划的跟踪，使用 10053 极为简便，通过如下方式就可以捕获 SQL 的解析过程：

```
alter session set events '10053 trace name context forever,level 1';
explain plan for you_select_query;
```

例如：

```
SQL> alter session set events '10053 trace name context forever,level 1';
Session altered.

SQL> explain plan for select count(*) from obj$;
Explained.
```

然后在 udump 目录下就可以找到 10053 生成的跟踪文件，在该文件中，找到了查询相关表的统计信息，其中 OBJ$的信息如下所示，其中 CDN（CarDiNality）指表中包含的记录数量，此处显示 OBJ$表中有 24 万条左右的记录，使用了 2941 个数据块：

```
***********************
Table stats   Table: OBJ$  Alias: SYS_ALIAS_1
  TOTAL :: CDN: 245313 NBLKS: 2941 AVG_ROW_LEN: 79
Column:    OWNER# Col#: 3     Table: OBJ$  Alias: SYS_ALIAS_1
   NDV: 221     NULLS: 0       DENS: 4.5249e-03 LO: 0 HI: 259
   NO HISTOGRAM: #BKT: 1 #VAL: 2
-- Index stats
  INDEX NAME: I_OBJ1 COL#: 1
    TOTAL :: LVLS: 1  #LB: 632 #DK: 245313 LB/K: 1 DB/K: 1 CLUF: 4184
  INDEX NAME: I_OBJ2 COL#: 3 4 5 12 13 6
    TOTAL :: LVLS: 2  #LB: 1904 #DK: 245313 LB/K: 1 DB/K: 1 CLUF: 180286
  INDEX NAME: I_OBJ3 COL#: 15
    TOTAL :: LVLS: 1  #LB: 19 #DK: 2007 LB/K: 1 DB/K: 1 CLUF: 340
_OPTIMIZER_PERCENT_PARALLEL = 0
*************************************
```

而在前面的 10046 跟踪信息中，显示 OBJ$包含大约 200 万条记录，这是非常巨大的不同，对于 USER$表，显示具有 2863 条记录，而统计信息中显示仅有 253 条记录：

```
***********************
Table stats   Table: USER$  Alias: U
  TOTAL :: CDN: 253 NBLKS: 16 AVG_ROW_LEN: 82
Column:    USER# Col#: 1     Table: USER$  Alias: U
   NDV: 253     NULLS: 0       DENS: 3.9526e-03 LO: 0 HI: 261
   NO HISTOGRAM: #BKT: 1 #VAL: 2
-- Index stats
  INDEX NAME: I_USER# COL#: 1
    TOTAL :: LVLS: 0  #LB: 1 #DK: 258 LB/K: 1 DB/K: 1 CLUF: 13
  INDEX NAME: I_USER1 COL#: 2
    TOTAL :: LVLS: 0  #LB: 1 #DK: 253 LB/K: 1 DB/K: 1 CLUF: 87
***********************
```

这说明数据字典中记录的统计信息与真实情况不符合，导致了 SQL 选择了错误的执行计划，在使用 CBO 的 Oracle9i 数据库中，这种情况极为普遍，通过删除表的统计信息，或者重新收集正确的

统计信息，可以使 SQL 执行恢复到正常合理的范畴内。Oracle10g 开始时自动统计信息收集，就是为了防止出现统计信息陈旧的现象。

以下是通过 dbms_stats 包清除和重新收集表的统计信息的简单参考：

```
SQL> exec dbms_stats.delete_table_stats(user,'OBJ$');

PL/SQL procedure successfully completed.

SQL> exec dbms stats.gather table stats(user,'OBJ$');

PL/SQL procedure successfully completed.
```

基于这样的判断我们建议客户做出修正，最后的客户反馈结果是：按照你的建议，在更新 OBJ$的统计信息后，该语句的执行时间由 10 分钟减少到了 2 分钟。接下来，按照类似的思路，又更新了 USER$、CON$、CDEF$表的统计信息，这时，语句的执行时间减少到了零点几秒。至此，问题解决。

ORA-00600 kcratr1_lostwrt 之解决与原理分析

客户的一个数据库因为断电遇到了 ORA-600 kcratr1_lostwrt 错误，数据库无法启动。

错误信息类似：

```
ksedmp: internal or fatal error
ORA-00600: internal error code, arguments: [kcratr1_lostwrt], [], [], [], [], [], [], []
Current SQL statement for this session:
alter database open
```

这个错误不难解决，但是其具体成因有点意思。Metalink 对这个错误的解释只有一句关键：

```
When an instance is restarted following an instance crash, Oracle carries
out some checks against the last block that was written to disk prior to
the instance crash.If Oracle finds an old block, then this suggests a lost
write and the error is raised.
```

这句话是说，当实例崩溃之后启动，Oracle 会去检查崩溃前最后一个写出的数据块，通过控制文件校验其是否一致，如果这个块是旧的，则说明最后的写操作丢失了。

这是一个非常快捷巧妙地判断，如果有写丢失，自然必须引入恢复。

在跟踪文件中记录了详细的信息：

```
Last BWR afn: 6 rdba: 0x18f9590(blk 1021328) ver: 0x0001.5c21fd6e.01 flg: 0x04
Disk version: 0x0001.5c1ec4f0.04 flag: 0x04
```

提示说，控制文件记录的最后一次写的数据块是 file6 block 1021328，SCN 版本为：0x0001.5c21fd6e，而数据文件上记录的 SCN 则是 0x0001.5c1ec4f0，后者陈旧，小于前者，这说明丢失了写操作。

那能否恢复呢？跟踪文件里还会记录这样的信息：

```
Thread checkpoint rba:0x00dfeb.00000002.0010 scn:0x0001.5c1ee5b7
On-disk rba:0x00dfeb.0001161f.0000 scn:0x0001.5c2266d6
```

线程检查点的 SCN 为 0x0001.5c1ee5b7，而 On-Disk Rba 的 SCN 为 0x0001.5c2266d6，完全可以推演超过 5c21fd6e，可以恢复。

所以这样的问题：

```
SQL>startup mount;
SQL>recover database;
SQL>alter database open;
```

一般就可以完成恢复了，如果不幸的，你的 On-Disk Rba 不足以恢复丢失的写操作，则问题就会稍微复杂一些。

在这个案例中我们学会的是：**认真阅读跟踪文件的信息，学会理解体会其深层含义，对于加深对于数据库的了解非常重要。**

ORA-00600 kcratr_nab_less_than_odr 案例一则

有朋友遇到了 ORA-00600: 内部错误代码，参数: [kcratr_nab_less_than_odr]错误，具体环境为 Solaris 10(SPARC) Oracle Database 11.2.0.1，错误信息类似如下：

```
SQL> alter database open ;
alter database open
*
第 1 行出现错误:
ORA-00600: 内部错误代码, 参数: [kcratr_nab_less_than_odr], [1], [3313],[2328320], [2334233], [],
[], [], [], [], [], []
```

后台记录了如下日志：

```
Incident 63078 created, dump file:
   /u01/diag/rdbms/orcl/orcl/incident/incdir_63078/orcl_ora_1916_i63078.trc
   ORA-00600: 内部错误代码, 参数: [kcratr_nab_less_than_odr], [1], [3313], [2328320], [2334233],
[], [], [], [], [], [], []
   ORA-00600: 内部错误代码, 参数: [kcratr_nab_less_than_odr], [1], [3313], [2328320], [2334233],
[], [], [], [], [], [], []
   ORA-00600: 内部错误代码, 参数: [kcratr_nab_less_than_odr], [1], [3313], [2328320], [2334233],
[], [], [], [], [], [], []
```

这个错误当时在 metalink 上找不到任何有用信息。其实，很多时候在处理问题时，我们可能都无法找到针对性的参考，这时候，猜测就很重要了。当学习到一定阶段，**猜测将成为一种重要的学习能力**。

猜测错误号为[kcratr_nab_less_than_odr]，根据 less than 字样，可以判断是在进行某个比较时，出现问题，剩下的 kcratr、nab、odr 都可以进行猜测，比如 odr 可能是 Oracle Data Redo/Oracle Data Recovery 等等的缩写。

再通过进一步的日志信息可以看到：

```
Thread 1 checkpoint: logseq 3313, block 2, scn 5965899084787
 cache-low rba: logseq 3313, block 1484161
  on-disk rba: logseq 3313, block 2334233, scn 5965899135009
 start recovery at logseq 3313, block 1484161, scn 0
```

这里就知道了 600 错误中的[1]、[3313]的含义，是 Thread 1 logseq 3313，那么剩下的就应该是恢复时的 rba 地址，错误提示中的 2334233 是 On-Disk Rba，是恢复应该到达的终点，而 2328320 小于 On-Disk Rba，就应该是 Less Than 里面提到的恢复的终点因为没有到达 Redo 的最后位置，被认为是非法的，可能丢失数据。

再向下检查错误跟踪文件，可以验证我们的推测：

```
WARNING! Crash recovery of thread 1 seq 3313 is
ending at redo block 2328320 but should not have ended before
redo block 2334233
```

现在 600 错误中的另外重要信息出现了，2328320 是恢复的中止位置，小于了 on-disk rba。

那么现在不需要 Metalink，我们就可以判断和解决这个 600 问题了，数据库无法启动的原因就是恢复的进度不够，进度不够可能是因为断电导致某些写操作丢失，进而控制文件和数据文件不一致，无法完成正常恢复。

在最终恢复时，通过重建控制文件，应用 ONLINE 日志进行恢复，在日志无损失的情况下，恢复得以顺利完成：

```
指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/redolog/redo01.log
已应用的日志。
完成介质恢复。
```

然后通过 Resetlogs 打开数据库：

```
SQL> alter database open RESETLOGS;
数据库已更改。
```

在这个案例中，我们学到的经验时：**猜测是一种很重要的能力。**

Cache-Low RBA 与 On-Disk RBA 的恢复证明

在最近（2010 年 9 月 6 日）的一次培训中，有位朋友问起上节案例，该如何证明和验证 Oracle 介于 Cache-Low RBA 和 On-Disk RBA 之间的恢复过程。我们可以通过如下的过程来做一些观察和证明。

首先执行一个建表的 CTAS 操作，这个操作是为了多生成一些脏块（Dirty Buffer），然后紧接着执行两次控制文件转储，两次转储是为了确认对比一下控制文件的检查点没有变化，然后紧接着执行强制关闭数据库（Abort 方式），再启动数据库（如图 1-10 所示）。

现在来分析一下跟踪文件，看看其中的相关信息，选取第二次转储的控制文件信息，在数据库 Entry 部分，可以找到检查点记录：

```
**********************************************************************
DATABASE ENTRY
**********************************************************************
 (size = 316, compat size = 316, section max = 1, section in-use = 1,
  last-recid= 0, old-recno = 0, last-recno = 0)
```

```
(extent = 1, blkno = 1, numrecs = 1)
07/31/2010 16:35:48
DB Name "ENMO"
Database flags = 0x00404000 0x00001000
Controlfile Creation Timestamp  07/31/2010 16:35:49
Incmplt recovery scn: 0x0000.00000000
Resetlogs scn: 0x0000.00089c75 Resetlogs Timestamp  07/31/2010 16:35:52
Prior resetlogs scn: 0x0000.00000001 Prior resetlogs Timestamp  03/14/2008 18:46:22
Redo Version: compatible=0xa200300
#Data files = 4, #Online files = 4
Database checkpoint: Thread=1 scn: 0x0000.00119459
Threads: #Enabled=1, #Open=1, Head=1, Tail=1
```

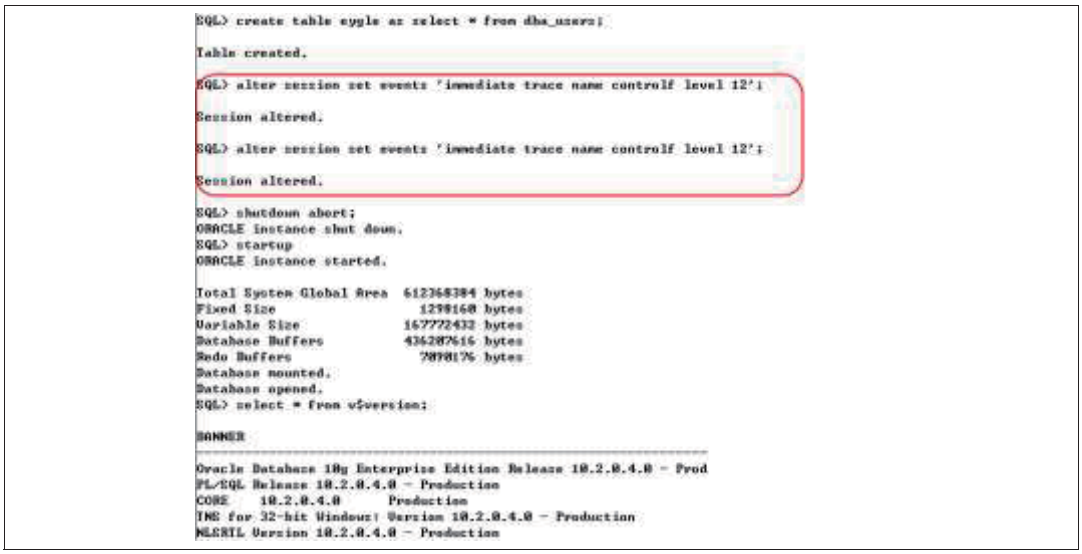


图 1-10　转储控制文件

此时记录数据库的检查点 SCN 是 119459，这是 16 进制，10 进制是 1152089。

继续检查，在检查点进程记录部分，获得如下信息，这里就包含了 Low Cache RBA 和 On Disk RBA 的信息，也记录了 Dirty Buffer 的数量是 48：

```
*******************************************************************************
CHECKPOINT PROGRESS RECORDS
*******************************************************************************
 (size = 8180, compat size = 8180, section max = 11, section in-use = 0,
  last-recid= 0, old-recno = 0, last-recno = 0)
 (extent = 1, blkno = 2, numrecs = 11)
THREAD #1 - status:0x2 flags:0x0 dirty:48
low cache rba:(0x27.6c.0) on disk rba:(0x27.f9.0)
on disk scn: 0x0000.001195a5 09/10/2010 14:55:25
resetlogs scn: 0x0000.00089c75 07/31/2010 16:35:52
heartbeat: 729376761 mount id: 570757625
```

把这里的 RBA 信息简单分析一下（见表 1-1）：

表 1-1　简单分析 RBA 信息

	RBA 信息	Log Sequence	Blcok Number
Low Cache RBA	0x27.6c.0	0x27 = 39	6c = 108
On Disk RBA	0x27.f9.0	0x27 = 39	F9 = 249

在启动数据库时，进行恢复产生了一个跟踪文件，记录了恢复的过程，恢复从 39 号日志文件的第 108 块恢复至 249 块，正是以上数据库关闭之前的 RBA 地址范围：

```
*** SESSION ID:(158.4) 2010-09-10 14:56:11.738
Successfully allocated 2 recovery slaves
Using 545 overflow buffers per recovery slave
Thread 1 checkpoint: logseq 39, block 2, scn 1152089
 cache-low rba: logseq 39, block 108
   on-disk rba: logseq 39, block 249, scn 1152421
 start recovery at logseq 39, block 108, scn 0
----- Redo read statistics for thread 1 -----
Read rate (ASYNC): 70Kb in 0.20s => 0.34 Mb/sec
Total physical reads: 4096Kb
Longest record: 8Kb, moves: 0/243 (0%)
Change moves: 2/29 (6%), moved: 0Mb
Longest LWN: 53Kb, moves: 0/6 (0%), moved: 0Mb
Last redo scn: 0x0000.001195a4 (1152420)
---------------------------------------------
```

数据库恢复的检查点起点是 SCN 1152089，也就是控制文件中记录的数据库最后完成的检查点，On-Disk RBA 的 SCN 是 1152421，转换为 16 进制也就是 1195A5，也和控制文件中记录的 On Disk SCN 完全相符。

数据库的恢复 SCN 范围也就由此确定，即 SCN 范围：1152089~1152241。

启动数据库之后，查询一下日志信息，可以看到 39 号日志文件正是执行恢复的日志文件，其 SCN 范围处于 1152088 和 1172422 之间，一个日志就满足了之前恢复的 SCN 范围，恢复完成之后日志切换，当前使用了 40 号日志：

```
SQL> select * from v$log;

GROUP#  THREAD#  SEQUENCE# BYTES    MEMBERS  ARC  STATUS   FIRST_CHANGE# FIRST_TIME
------ ------- ---------- -------- ------- ---- -------- ------------- ------------
     1       1         40 52428800       1 NO   CURRENT       1172422 10-SEP-10
     2       1         38 52428800       1 NO   INACTIVE      1131823 10-SEP-10
     3       1         39 52428800       1 NO   INACTIVE      1152088 10-SEP-10
```

至此我们清晰地看到了数据库恢复从 Low Cache RBA 至 On Disk RBA 的过程。

定时任务带来的问题——auto_space_advisor_job_proc

在另外一个客户的系统中，再次遇到了类似于上一则案例提到的情况。在客户的 SAP 系统中，某个高负载的时段，数据库遇到了 DBMS_SCHEDULER 任务的一个 Bug，其数据库版本为 10.2.0.2。

在 SQL Ordered By Elapsed Time 的采样中，Top 6 都是 DBMS_SCHEDULER 调度的任务，而且耗时显著（如图 1-11 所示）。

图 1-11　SQL Ordered By Elapsed Time 列表

处在第一位的，是和上一则案例相同的 auto_space_advisor_job_proc，CPU Time 消耗高达 4226 秒：

```
call dbms_space.auto_space_advisor_job_proc ( )
```

执行花费了大量的时间，3000 多秒，进而执行的 SQL：

```
insert into wri$_adv_objspace_trend_data select timepoint, space_usage, space_alloc, quality
fromtable(dbms_space.object_growth_trend(:1,:2,:3,:4,NULL,NULL,NULL,'FALSE',:5,'FALSE'))
```

也花费了 2514 秒的时间，这显然是不正常的。

在正常情况下，单独跟踪一下 SQL*Plus 手工执行，可以获得这个 SQL 的执行统计信息（如图 1-12 所示）：

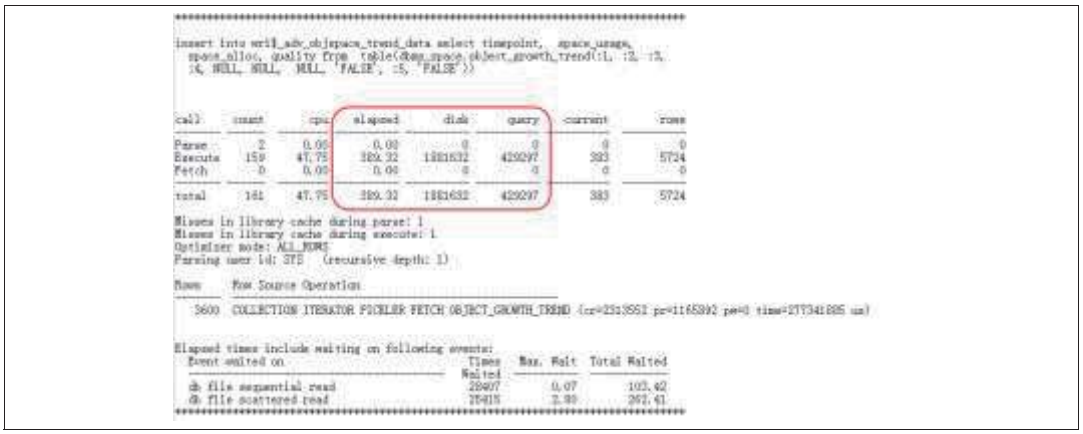


图 1-12　跟踪 SQL*Plus 手工执行获得的统计信息

注意到，这个 Insert 仍然消耗了 389 秒的时间，逻辑读 429 297，性能是存在问题的。在 Metalink 上存在如下一个 Bug。

```
Bug 5376783: DBMS_SPACE.OBJECT_GROWTH_TREND CALL TAKES A LOT OF DISK READS
```

这个 Bug 在 DBMS_SPACE.OBJECT_GROWTH_TREND 进行空间分析时被触发，根本原因在于内部算法在执行空间检查时，耗费了大量的评估 IO 成本，导致了大量的 IO 资源使用。

临时的处理办法是，暂时关闭这个自动任务：

```
execute dbms_scheduler.disable('AUTO_SPACE_ADVISOR_JOB');
```

这个 Bug 在 10.2.0.2 之后的版本中被修正。

既然 Oracle 的缺省定时任务可能会带来如此多的问题，我们就很有必要去关注一下系统有哪些缺省的任务，执行情况如何。以下是一个 10.2.0.5 版本的数据库中一些自动任务的调度设置情况：

```
SQL> select job_name,state,enabled,last_start_date from dba_scheduler_jobs;

JOB NAME                  STATE         ENABL  LAST START DATE
------------------------- ------------- ------ -----------------------------------
AUTO_SPACE_ADVISOR_JOB    SCHEDULED     TRUE   07-AUG-10 06.00.03.792886 AM +08:00
GATHER_STATS_JOB          SCHEDULED     TRUE   07-AUG-10 06.00.03.783957 AM +08:00
FGR$AUTOPURGE_JOB         DISABLED      FALSE
PURGE_LOG                 SCHEDULED     TRUE   07-AUG-10 03.00.00.353023 AM PRC
MGMT_STATS_CONFIG_JOB     SCHEDULED     TRUE   01-AUG-10 01.01.01.822354 AM +08:00
MGMT_CONFIG_JOB           SCHEDULED     TRUE   07-AUG-10 06.00.03.767320 AM +08:00
```

在以上的调度任务中，GATHER_STATS_JOB 是 Oracle Database 10g 开始引入的自动统计信息收集的任务，该任务缺省的调度是，工作日每晚 22：00 至凌晨 6：00 进行分析，周末全天进行分析。在以下输出中，我们可以看到任务无法完成，STOP 的情况：

```
SQL> SELECT log_id, job_name, status,
  2  TO_CHAR(ACTUAL_START_DATE,'DD-MON-YYYY HH24:MI') start_date,TO_CHAR (log_date,
'DD-MON-YYYY HH24:MI') log_date
  3  FROM dba_scheduler_job_run_details
  4  WHERE job_name = 'GATHER_STATS_JOB' order by 4;

    LOG_ID  JOB_NAME          STATUS     START_DATE          LOG_DATE
---------- ----------------- ---------- ------------------- ------------------
      1480  GATHER_STATS_JOB  SUCCEEDED  02-AUG-2010 22:00   03-AUG-2010 00:58
      1561  GATHER_STATS_JOB  STOPPED    03-AUG-2010 22:00   04-AUG-2010 06:00
      1640  GATHER_STATS_JOB  SUCCEEDED  04-AUG-2010 22:00   05-AUG-2010 05:36
      1680  GATHER_STATS_JOB  SUCCEEDED  05-AUG-2010 22:00   05-AUG-2010 22:25
      1741  GATHER_STATS_JOB  SUCCEEDED  06-AUG-2010 22:00   06-AUG-2010 22:27
      1800  GATHER_STATS_JOB  SUCCEEDED  07-AUG-2010 06:00   07-AUG-2010 06:02
       384  GATHER_STATS_JOB  STOPPED    07-JUL-2010 22:00   08-JUL-2010 06:00
       463  GATHER_STATS_JOB  SUCCEEDED  08-JUL-2010 22:00   09-JUL-2010 05:06
       503  GATHER_STATS_JOB  SUCCEEDED  09-JUL-2010 22:00   09-JUL-2010 22:05
       544  GATHER_STATS_JOB  SUCCEEDED  10-JUL-2010 06:00   10-JUL-2010 06:02
       589  GATHER_STATS_JOB  SUCCEEDED  12-JUL-2010 22:00   12-JUL-2010 22:04
       597  GATHER_STATS_JOB  SUCCEEDED  13-JUL-2010 22:00   13-JUL-2010 22:03
```

在一些大型数据库中，这个任务不一定能够有效执行，以下是某用户的数据库环境，输出显示，多日数据库都因为 ORA-04031 错误未能完成统计信息收集采样：

```
SQL> SELECT LOG_DATE,RUN_DURATION,JOB_NAME,STATUS,ERROR#
  2  FROM DBA_SCHEDULER_JOB_RUN_DETAILS
  3  WHERE JOB_NAME='GATHER_STATS_JOB'
  4  order by 1 desc;
LOG_DATE                    RUN_DURATION             JOB_NAME          STATUS        ERROR#
--------------------------- ------------------------ ----------------- ----------- ----------
26-MAY-10 10.00.09.290291 PM +08:00 +000 00:00:05 GATHER_STATS_JOB   FAILED         22303
25-MAY-10 10.00.08.973684 PM +08:00 +000 00:00:06 GATHER_STATS_JOB   FAILED          4031
24-MAY-10 10.00.22.977244 PM +08:00 +000 00:00:18 GATHER_STATS_JOB   FAILED          4031
22-MAY-10 06.00.16.950362 AM +08:00 +000 00:00:13 GATHER_STATS_JOB   FAILED          4031
21-MAY-10 10.00.49.653788 PM +08:00 +000 00:00:47 GATHER_STATS_JOB   FAILED          4031
```

```
20-MAY-10 10.00.14.028432 PM +08:00 +000 00:00:11  GATHER_STATS_JOB  FAILED          4031
19-MAY-10 10.00.20.828607 PM +08:00 +000 00:00:18  GATHER_STATS_JOB  FAILED          4031
19-MAY-10 05.54.27.871444 AM +08:00 +000 07:54:25  GATHER_STATS_JOB  SUCCEEDED          0
18-MAY-10 05.36.01.494920 AM +08:00 +000 07:35:59  GATHER_STATS_JOB  SUCCEEDED          0
15-MAY-10 07.06.05.793257 AM +08:00 +000 01:06:01  GATHER_STATS_JOB  SUCCEEDED          0
15-MAY-10 03.56.50.898303 AM +08:00 +000 05:56:48  GATHER_STATS_JOB  SUCCEEDED          0
```

在 GATHER_STATS_JOB 任务不能够有效的执行时，我们必须及时的介入去手工处理，不及时的统计信息可能使数据库产生错误的执行计划。

正常的 AUTO_SPACE_ADVISOR_JOB 调度可能应该有着类似以下输出的执行结果。

```
SQL> SELECT log_id, job_name, status,TO_CHAR(ACTUAL_START_DATE,'DD-MON-YYYY HH24:MI')
start_date,
  2     TO_CHAR (log_date, 'DD-MON-YYYY HH24:MI') log_date
  3   FROM dba_scheduler_job_run_details
  4  WHERE job_name = 'AUTO_SPACE_ADVISOR_JOB' order by 4;

   LOG_ID JOB_NAME                  STATUS        START_DATE           LOG_DATE
--------- ------------------------- ------------ -------------------- ----------------
     1460 AUTO_SPACE_ADVISOR_JOB    SUCCEEDED     02-AUG-2010 22:00   02-AUG-2010 22:16
     1520 AUTO_SPACE_ADVISOR_JOB    SUCCEEDED     03-AUG-2010 22:00   03-AUG-2010 23:18
     1600 AUTO_SPACE_ADVISOR_JOB    SUCCEEDED     04-AUG-2010 22:00   04-AUG-2010 22:19
     1681 AUTO_SPACE_ADVISOR_JOB    SUCCEEDED     05-AUG-2010 22:00   05-AUG-2010 22:28
     1740 AUTO_SPACE_ADVISOR_JOB    SUCCEEDED     06-AUG-2010 22:00   06-AUG-2010 22:17
```

定时任务 GATHER_STATS_JOB 与 SQL 执行

上一节中，我们曾经说过，GATHER_STATS_JOB 任务有时候会影响数据库的使用，特别是对那些 24 小时 × 7 天都是繁忙时段的数据库，还有那些部署在中国，为国外业务提供服务的数据库。你可以想想，当你的业务正常热火朝天的进行时，忽然 GATHER_STATS_JOB 调度 DBMS_STATS 开始执行，系统的性能可能会骤然衰减，甚至影响业务的可用性。

在 Oracle Database 10g 刚刚开始普及时，我就曾接到过朋友的电话，他说观察到数据库的一个奇怪现象，就是每天夜里 10 点开始，整个应用开始拥堵，业务进行缓慢，1 个小时左右之后恢复正常。我说，这肯定是 GATHER_STATS_JOB 惹的祸。事实正是如此。

于是，很多客户和软件厂商选择了自定制的统计信息收集，比如选择每个周日做一次全库收集等。然而这样也不一定保险，在一个客户环境里，我们就遇到了以下这样一个案例。

首先是收到客户报告，系统的 CPU 使用率频繁冲高，基本是 100%的使用消耗，有部分进程消耗大量的 CPU，图 1-13 是 IBM 小型机下 Topas 的输出截图。

我们可以生成一个 AWR 报告，观察一下数据库的性能瓶颈所在。

首先记录一下数据库的基本信息，通过 DB Time 和 Elapsed Time 的比较，可以知道数据库确实处于一个比较繁忙的状态（如图 1-14 所示）。

在 Top SQL 中我们发现，很多 SQL 执行时间较长，也正是这些 SQL 消耗了大量的 CPU 资源，报告基本信息如图 1-15 所示。

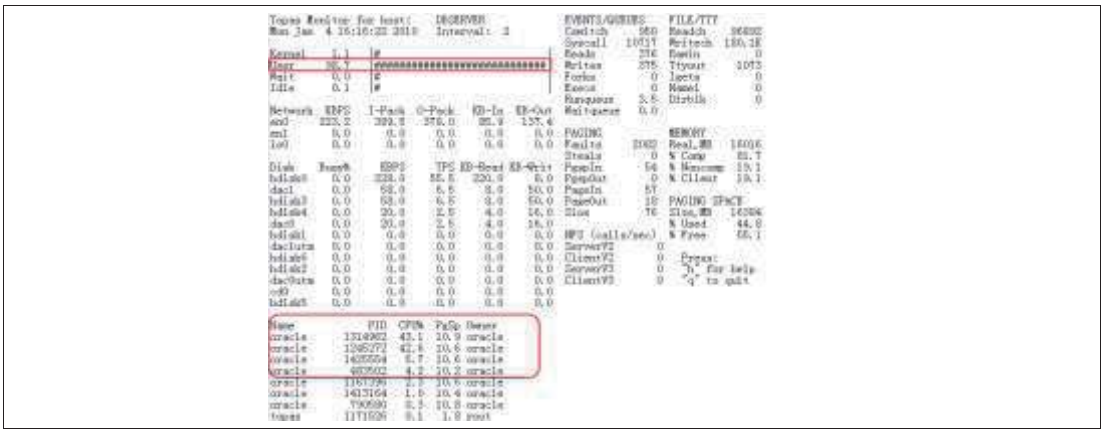


图 1-13　IBM 小型机下 Topas 的输出截图

图 1-14　比较 DB Time 和 Elapsed Time

图 1-15　SQL ordered by Elapsed Time 列表

其中几个主要 SQL 执行时间近 5 分钟，达到了无法接受的地步。通过 AWR SQL Report 我们可以获得相关 SQL 的执行计划等详细信息，进一步分析出现异常的原因。

图 1-16 捕获的是 SQL ID 为 fqd75cr78fr6x 的 SQL 执行计划。

该 SQL 每次执行的 Elapsed Time 是 270 秒，单次执行逻辑读也高达 57，085，374（如图 1-17 所示）：

该 SQL 的执行计划如图 1-18 所示，我们注意到执行计划中有一个 MERGE JOIN CARTESIAN 的操作，一般来说 MERGE JOIN 已经值得怀疑，而 CARTESIAN 对于稍大的数据量更会雪上加霜。所以我们马上怀疑这个执行计划出了问题，之所以出现这个执行计划，很可能是统计信息出了问题，使得每个步骤评估出来的行数（Rows）是 1，而这个 1 又极有可能是因为评估无满足条件的记录，计算取整得出来的：

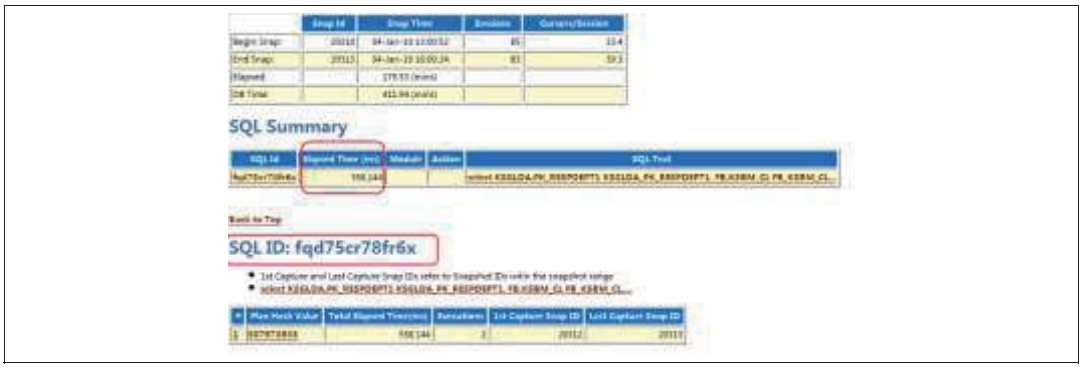


图 1-16　SQL ID 为 fqd75cr78fr6x 的 SQL 的执行计划

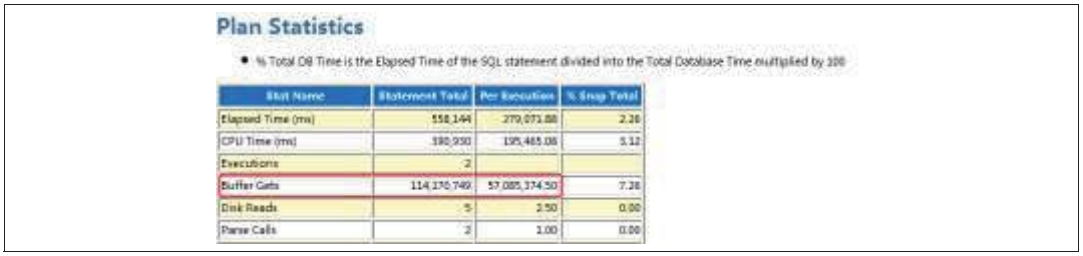


图 1-17　单次执行逻辑读高达 57, 085, 374

图 1-18　SQL 的执行计划中有一个 MERGE JOIN CARTESIAN 操作

AWR SQL Report 最后部分是 SQL 代码（如图 1-19 所示）：

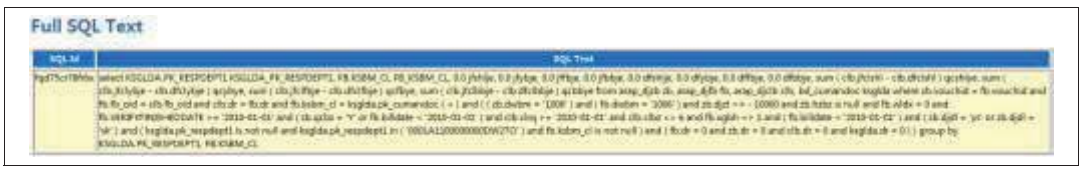


图 1-19　AWR SQL Report 最后部分的 SQL 代码

格式化一下这段代码，获得 SQL 的直观印象：

```
select KSGLDA.PK_RESPDEPT1 KSGLDA_PK_RESPDEPT1,
     FB.KSBM_CL FB_KSBM_CL,0.0 jfshlje,0.0 jfybje,0.0 jffbje,
     0.0 jfbbje,0.0 dfshlje,0.0 dfybje,0.0 dffbje,0.0 dfbbje,
     sum(clb.jfclshl - clb.dfclshl) qcshlye,sum(clb.jfclybje - clb.dfclybje) qcybye,
     sum(clb.jfclfbje - clb.dfclfbje) qcfbye,sum(clb.jfclbbje - clb.dfclbbje) qcbbye
  from arap_djzb zb, arap_djfb fb, arap_djclb clb, bd_cumandoc ksglda
```

```
 where zb.vouchid = fb.vouchid
   and fb.fb_oid = clb.fb_oid
   and clb.dr = fb.dr
   and fb.ksbm_cl = ksglda.pk_cumandoc(+)
   and ((zb.dwbm = '1006') and (fb.dwbm = '1006') and zb.djzt >= -10000 and
      zb.hzbz is null and fb.wldx = 0 and
      fb.VERIFYFINISHEDDATE >= '2010-01-01' and
      (zb.qcbz = 'Y' or fb.billdate < '2010-01-01') and
      clb.clrq >= '2010-01-01' and clb.clbz <> 6 and fb.xgbh <> 1 and
      (fb.billdate < '2010-01-01') and (zb.djdl = 'ys' or zb.djdl = 'sk') and
      (ksglda.pk_respdept1 is not null and
      ksglda.pk_respdept1 in ('0001A1100000000DW2TO') and
      fb.ksbm_cl is not null) and
      (fb.dr = 0 and zb.dr = 0 and clb.dr = 0 and ksglda.dr = 0))
 group by KSGLDA.PK_RESPDEPT1, FB.KSBM_CL
```

检查数据库统计信息收集的任务，发现未有统计信息执行的调度任务：

```
SYS@ora10g>col job_name for a20
SYS@ora10g>SELECT log_id, job_name, status,
     2  TO_CHAR (log_date, 'DD-MON-YYYY HH24:MI') log_date
     3  FROM dba_scheduler_job_run_details
     4  WHERE job_name = 'GATHER_STATS_JOB';

no rows selected
```

再进一步检查数据库中主要业务表的最后一次统计信息收集时间，发现最后一次是在 2009 年 12 月 30 日进行的统计数据收集：

```
SYS@ora10g>select table_name,LAST_ANALYZED from dba_tables where table_name='ARAP_DJCLB';

TABLE_NAME                     LAST_ANALYZED
------------------------------ -------------------
ARAP_DJCLB                     2009-12-30 02:10:05

SYS@ora10g>select table_name,LAST_ANALYZED from dba_tables where table_name like 'ARAP%';

TABLE_NAME                     LAST_ANALYZED
------------------------------ -------------------
ARAP_XZFK                      2009-12-30 02:52:17
ARAP_TB_ITEM_B                 2009-12-30 02:52:17
ARAP_TB_ITEM                   2009-12-30 02:52:17
ARAP_SYSCODE                   2009-12-30 02:52:16
ARAP_RQQJB                     2009-12-30 02:52:16
ARAP_QRYOBJ                    2009-12-30 02:52:16
ARAP_POWER_DJLX                2009-12-30 02:52:16
ARAP_PARAM                     2009-12-30 02:52:16
ARAP_NOTICE_B                  2009-12-30 02:52:16
ARAP_NOTICE                    2009-12-30 02:52:16
……
```

统计信息的缺失可能会导致了 SQL 执行计划的异常。考察前面的 SQL 查询谓词，可以看到有相关的时间限定，限定查询 2010 年的数据，而根据统计信息则不包含 2010 年的数据分布，通过指定谓词密

度计算，就得出来多个步骤的评估 Rows 为 1，根据这样的判断，进而选择了 MERGE JOIN CARTESIAN 的连接方式，影响了性能：

```
fb.VERIFYFINISHEDDATE >= '2010-01-01' and
     (zb.qcbz = 'Y' or fb.billdate < '2010-01-01') and
     clb.clrq >= '2010-01-01' and clb.clbz <> 6 and fb.xgbh <> 1 and
     (fb.billdate < '2010-01-01')
```

为了证实我们的猜测，可以通过 10053 事件对 SQL 的执行计划解析进行跟踪，以下信息来自 10053 事件的跟踪输出。

在单表访问路径中，处理表（CLB）的 CLRQ 字段（CHARACTER 类型）就显示谓词的选择性超出了表中数值范围（Out-Of-Range）。

```
***************************************
SINGLE TABLE ACCESS PATH
 Column (#9): CLRQ(CHARACTER)
  AvgLen: 11.00 NDV: 798 Nulls: 0 Density: 0.0015234
  Histogram: HtBal  #Bkts: 75  UncompBkts: 75  EndPtVals: 76
 Using prorated density: 7.8950e-08 of col #9 as selectivity of out-of-range value pred
 Column (#4): CLBZ(NUMBER)
  AvgLen: 2.00 NDV: 4 Nulls: 0 Density: 0.25 Min: 0 Max: 6
 Table: ARAP_DJCLB  Alias: CLB
  Card: Original: 6333094  Rounded: 1  Computed: 0.37  Non Adjusted: 0.37
 Access Path: TableScan
  Cost:  94331.80  Resp: 94331.80  Degree: 0
   Cost_io: 92005.00  Cost_cpu: 7982504250
   Resp_io: 92005.00  Resp_cpu: 7982504250
kkofmx: index filter:"CLB"."DR"=0 AND "KSGLDA"."DR"=0
 Using prorated density: 7.8950e-08 of col #9 as selectivity of out-of-range value pred
 Access Path: index (RangeScan)
  Index: I_ARAP_DJCLB005
  resc_io: 4.00  resc_cpu: 29780
  ix_sel: 7.8950e-08  ix_sel_with_filters: 7.8950e-08
  Cost: 1.60  Resp: 1.60  Degree: 1
 Access Path: index (skip-scan)
  SS sel: 1  ANDV (#skips): 1117461
  SS io: 1117460.91 vs. table scan io: 92005.00
  Skip Scan rejected
 Access Path: index (FullScan)
  Index: I_ARAP_DJCLB_JSZC_02
  resc_io: 2576496.00  resc_cpu: 27768838880
  ix_sel: 1  ix_sel_with_filters: 1
  Cost: 1033873.03  Resp: 1033873.03  Degree: 1
 Best:: AccessPath: IndexRange  Index: I_ARAP_DJCLB005
      Cost: 1.60  Degree: 1  Resp: 1.60  Card: 0.37  Bytes: 0
***************************************
```

那么为了解决这个问题，可以将相关数据表进行统计信息收集：

```
NC5X@ora10g>exec dbms_stats.gather_table_stats(user, 'BD_CUMANDOC', cascade=>true);

PL/SQL procedure successfully completed.
```

```
NC5X@ora10g>select count(*) from ARAP_DJFB;

  COUNT(*)
----------
   4035495

Elapsed: 00:00:08.69
NC5X@ora10g>exec dbms_stats.gather_table_stats(user, 'ARAP_DJFB', cascade=>true);

PL/SQL procedure successfully completed.

Elapsed: 00:20:02.30
NC5X@ora10g>select count(*) from ARAP_DJCLB;

  COUNT(*)
----------
   6360072

NC5X@ora10g>exec dbms_stats.gather_table_stats(user,'ARAP_DJCLB',cascade=>true);

PL/SQL procedure successfully completed.

Elapsed: 00:16:16.54
```

此后 SQL 的执行计划恢复了正常，SQL 的逻辑读降低到单次 43,513：

```
<……略去 SQL 部分……>
9 rows selected.


Execution Plan
----------------------------------------------------------
Plan hash value: 3055746956

-------------------------------------------------------------------------------------------------
| Id |Operation                    | Name            |Rows | Bytes|Cost(%CPU)| Time      |
-------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT            |                 |   1 |  222 |    6 (17)| 00:00:01  |
|  1 | HASH GROUP BY               |                 |   1 |  222 |    6 (17)| 00:00:01  |
|  2 | NESTED LOOPS                |                 |   1 |  222 |    5  (0)| 00:00:01  |
|  3 | NESTED LOOPS                |                 |   1 |  188 |    4  (0)| 00:00:01  |
|  4 | NESTED LOOPS                |                 |   1 |  148 |    3  (0)| 00:00:01  |
|* 5 | TABLE ACCESS BY INDEX ROWID | ARAP_DJCLB      |   1 |   51 |    2  (0)| 00:00:01  |
|* 6 | INDEX RANGE SCAN            | I_ARAP_DJCLB005 |   1 |      |    1  (0)| 00:00:01  |
|* 7 | TABLE ACCESS BY INDEX ROWID | ARAP_DJFB       |   1 |   97 |    1  (0)| 00:00:01  |
|* 8 | INDEX UNIQUE SCAN           | PK_ARAP_DJFB    |   1 |      |    1  (0)| 00:00:01  |
|* 9 | TABLE ACCESS BY INDEX ROWID | ARAP_DJZB       |   1 |   40 |    1  (0)| 00:00:01  |
|*10 | INDEX UNIQUE SCAN           | PK_ARAP_DJZB    |   1 |      |    1  (0)| 00:00:01  |
|*11 | TABLE ACCESS BY INDEX ROWID | BD_CUMANDOC     |   1 |   34 |    1  (0)| 00:00:01  |
|*12 | INDEX UNIQUE SCAN           | PK_BD_CUMANDOC  |   1 |      |    1  (0)| 00:00:01  |
-------------------------------------------------------------------------------------------------
Statistics
----------------------------------------------------------
```

```
      0  recursive calls
      0  db block gets
  43513  consistent gets
      0  physical reads
    172  redo size
   1765  bytes sent via SQL*Net to client
    469  bytes received via SQL*Net from client
      2  SQL*Net roundtrips to/from client
      0  sorts (memory)
      0  sorts (disk)
      9  rows processed
```

最后用了两个半小时将用户的统计数据重新收集，整体数据库恢复了正常：

```
NC5X@ora10g>exec dbms_stats.gather_schema_stats('NC5X');

PL/SQL procedure successfully completed.

Elapsed: 02:23:41.87
```

另外一个类似的 SQL 执行计划发生了同样的改变，收集统计信息之前的执行计划（如图 1-20 所示）：

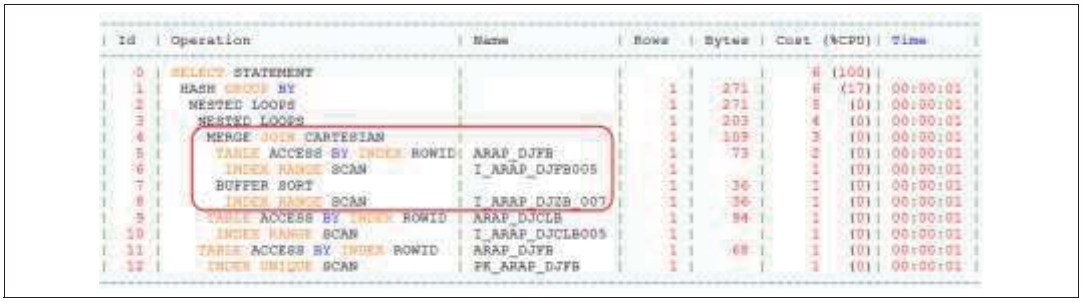


图 1-20　收集统计信息之前的执行计划

收集了统计信息之后的执行计划（如图 1-21 所示）：

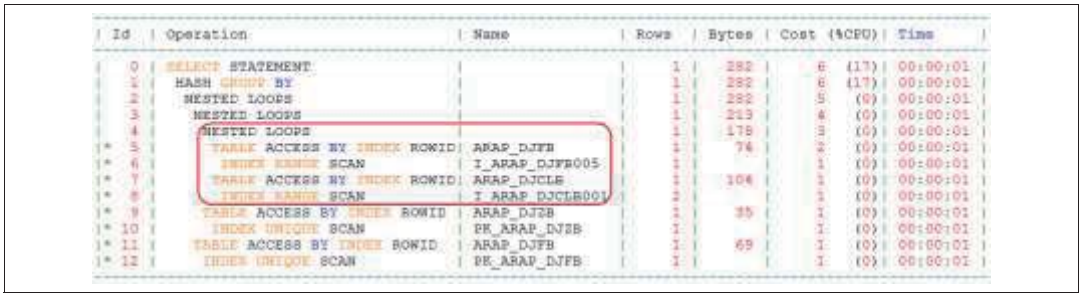


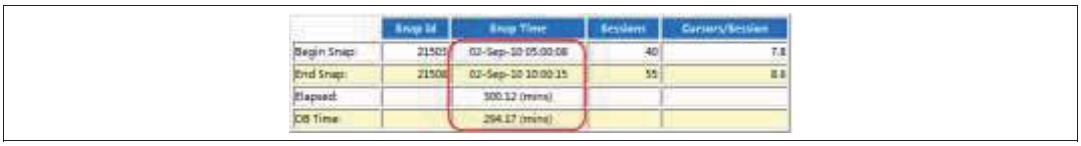
图 1-21　收集统计信息之后的执行计划

GATHER_STATS_JOB 跨月的“BUG”

有客户在 2010 年 9 月 2 日报告，说数据库中的某个 JOB 任务执行了近 4 个小时（从早 6 点至上午

10 点）未能成功完成，最后只能被迫杀掉中止了任务。

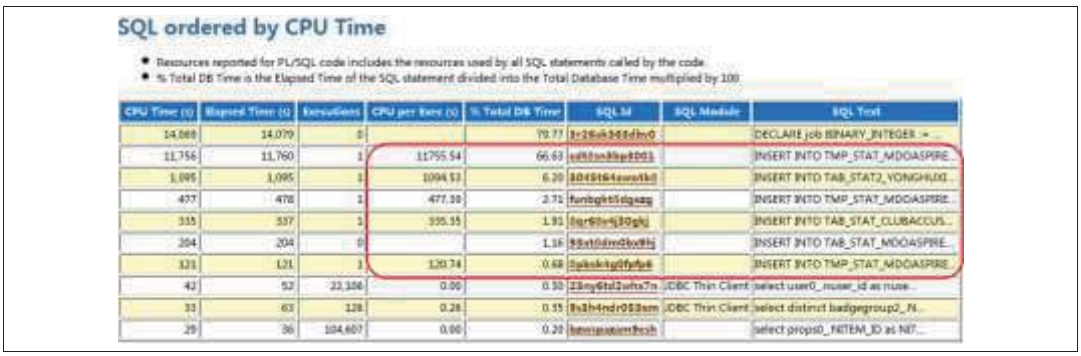
马上介入检查这个问题，首先生成一个该时段的 AWR 报告，看当时的运行情况，报告时间从 5:00~10:00（如图 1-22 所示）：



	Snap Id	Snap Time	Sessions	Curs/Session
Begin Snap:	21503	02-Sep-10 05:00:08	40	7.8
End Snap:	21508	02-Sep-10 10:00:15	55	8.8
Elapsed:		300.12 (mins)		
DB Time:		294.17 (mins)		

图 1-22　AWR 报告采样时段

在 Top SQL 列表，具有如下 SQL 消耗了大量的执行时间（如图 1-23 所示），注意排在第一位的是 JOB 任务，后面的一系列 SQL 语句为 JOB 任务中执行的 DML 操作，JOB 为执行完，所以其执行次数为 0，单次执行 CPU 时间也未知。



SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
14,080	14,079	0		79.77	3r26uh34t4hn0		DECLARE job BINARY_INTEGER := ...
11,756	11,760	1	11755.54	66.63	[redacted]		INSERT INTO TMP_STAT_MDOASPIRE...
1,095	1,095	1	1094.53	6.20	[redacted]		INSERT INTO TAB_STAT2_YONGHUXI...
477	478	1	477.38	2.71	[redacted]		INSERT INTO TMP_STAT_MDOASPIRE...
335	337	1	335.35	1.91	[redacted]		INSERT INTO TAB_STAT_CLUBACCUS...
204	204	0		1.16	[redacted]		INSERT INTO TAB_STAT_MDOASPIRE...
121	121	1	120.74	0.68	[redacted]		INSERT INTO TMP_STAT_MDOASPIRE...
42	52	22,106	0.00	0.30	23ny6td2wtx7n	JDBC Thin Client	select user0_.user_id as use...
33	63	128	0.26	0.35	8v3h4ndr053am	JDBC Thin Client	select distinct badgegroup2_.N...
29	36	104,607	0.00	0.20	[redacted]		select propso_.NITEM_ID as NIT...

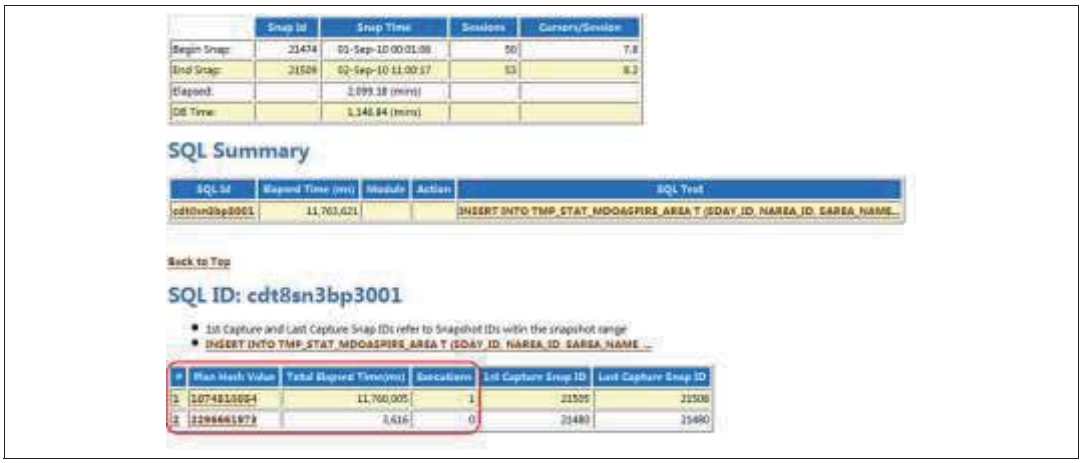
图 1-23　消耗了大量执行时间的 SQL

在以上 SQL 中，我们注意到其中一条消耗了 11755.54 秒的 CPU 时间，显然这条 SQL 隐含着核心的 CPU 消耗，正常情况下这个 SQL 应当执行很快，否则用户早就会报告异常了。

该 SQL 的主要逻辑如下：

```
INSERT INTO TMP_STAT_MDOASPIRE_AREA T
 SELECT TO_CHAR(:B1, 'yyyyMMdd'),
      AREA.OID,       AREA.PROVINCE,       COUNT(1),       0,
      COUNT(DISTINCT ACC.SMOBILE),
      0,       0,       0,       0,       0,       0
  FROM TAB_STAT_CLUBACCUSER_MDO MDO,
      TAB_WAPLOG_CLUB2       ACC,
      TAB_PROVINCE          AREA
 WHERE MDO.SDAY_ID = TO_CHAR(:B1, 'yyyyMMdd')
   AND MDO.SMOBILE = ACC.SMOBILE
   AND ACC.DACCESS >= TRUNC(:B1)
   AND ACC.DACCESS < TRUNC(:B1) + 1
   AND ACC.NTYPE = 0
   AND MDO.NOID = AREA.OID
 GROUP BY AREA.OID, AREA.PROVINCE
```

针对这个 SQLID，生成一个 AWR SQL 报告，看看 SQL 的执行计划情况，通常都是 SQL 执行计划的突然改变等异常导致的性能严重衰减。在生成的 SQL 报告中，果然发现该 SQL 存在两个执行计划版本，其中一条正是执行极为缓慢的带有 SELECT 查询的 INSERT 语句（如图 1-24 所示）。



	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	21434	01-Sep-10 00:01:08	50	7.8
End Snap:	21509	02-Sep-10 11:00:17	53	8.2
Elapsed:		2,099.18 (mins)		
DB Time:		1,148.84 (mins)		

SQL Summary

SQL Id	Elapsed Time (ms)	Module	Action	SQL Text
cdt8sn3bp3001	11,761,621			INSERT INTO TMP_STAT_MDOASPIRE_AREA T (SDAY_ID, NAREA_ID, SAREA_NAME...

Back to Top

SQL ID: cdt8sn3bp3001

- 1st Capture and Last Capture Snap IDs refer to Snapshot IDs within the snapshot range
- INSERT INTO TMP_STAT_MDOASPIRE_AREA T (SDAY_ID, NAREA_ID, SAREA_NAME ...

#	Plan Hash Value	Total Elapsed Time(ms)	Executions	1st Capture Snap ID	Last Capture Snap ID
1	1074810694	11,760,005	1	21505	21508
2	2296661973	1,616	0	21480	21480

图 1-24　SQL 的两个执行计划版本

我们分别来查看一下两个执行计划，分析一下性能衰减的原因，第一个执行计划，其中包含了一个显著的 MERGE JOIN CARTESIA 操作（如图 1-25 所示）：

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	INSERT STATEMENT				8 (100)			
1	SORT GROUP BY		1	66	8 (13)	00:00:01		
2	FILTER							
3	TABLE ACCESS BY INDEX ROWID	TAB_STAT_CLUBACCUSER_MDO	1	24	2 (0)	00:00:01		
4	NESTED LOOPS		1	66	7 (0)	00:00:01		
5	MERGE JOIN CARTESIAN		1	42	5 (0)	00:00:01		
6	PARTITION RANGE ITERATOR		1	34	0 (0)		KEY	KEY
7	TABLE ACCESS BY LOCAL INDEX ROWID	TAB_WAPLOG_CLUB2	1	34	0 (0)		KEY	KEY
8	INDEX RANGE SCAN	IDX_CLUB2_DACCESS	1		0 (0)		KEY	KEY
9	BUFFER SORT		52	250	5 (0)	00:00:01		
10	TABLE ACCESS FULL	TAB_PROVINCE	52	250	5 (0)	00:00:01		
11	INDEX RANGE SCAN	IDX_TAT_CLUBACCUSER_MDO	1		1 (0)	00:00:01		

图 1-25　包含 MERGE JOIN CARTESIA 操作的执行计划

笛卡儿积通常是我们不希望看到的，这里 JOIN 的两个对象还包含一个分区表的分区，这显然可能存在问题。

再观察另外一个执行计划，这里的 HASH JOIN 看起来是更加合理的选择（如图 1-26 所示）：

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	INSERT STATEMENT				12 (100)			
1	SORT GROUP BY		1	55	12 (17)	00:00:01		
2	FILTER							
3	HASH JOIN		1	55	11 (10)	00:00:01		
4	TABLE ACCESS BY LOCAL INDEX ROWID	TAB_WAPLOG_CLUB2	1	25	3 (0)	00:00:01		
5	NESTED LOOPS		1	47	5 (0)	00:00:01		
6	TABLE ACCESS BY INDEX ROWID	TAB_STAT_CLUBACCUSER_MDO	1	24	2 (0)	00:00:01		
7	INDEX RANGE SCAN	IDX_TAT_CLUBACCUSER_MDO	1		1 (0)	00:00:01		
8	PARTITION RANGE ITERATOR		8		2 (0)	00:00:01	KEY	KEY
9	INDEX RANGE SCAN	IDX_CLUB2_DACCESS	8		2 (0)	00:00:01	KEY	KEY
10	TABLE ACCESS FULL	TAB_PROVINCE	52	250	5 (0)	00:00:01		

图 1-26　包含 HASH JOIN 操作的执行计划

那么可能是什么地方出现问题呢？最值得怀疑的地方当然就是统计信息，我们查询一下分区的统计信息，发现果然 9 月份的分区统计信息未被收集。

```
SQL> select table_name,partition_name,partition_position,num_rows,blocks,last_analyzed
  2  from dba_tab_partitions a
  3  where a.table_name='TAB_WAPLOG_CLUB2' and a.partition_name like '%2010%'
  4  /

TABLE_NAME         PARTITION_NAME      ITION   NUM_ROWS  BLOCKS  LAST_ANALYZED
------------------ ------------------- ----- ---------- -------  -------------------
TAB_WAPLOG_CLUB2   PART_CLUB2_201001      15   31882877  923374  2010/2/1  22:33:00
TAB_WAPLOG_CLUB2   PART_CLUB2_201002      16   21827605  635691  2010/2/27  6:08:30
TAB_WAPLOG_CLUB2   PART_CLUB2_201003      17   26856055  784543  2010/4/1   22:22:30
TAB_WAPLOG_CLUB2   PART_CLUB2_201004      18   19129756  557686  2010/4/27 22:26:55
TAB_WAPLOG_CLUB2   PART_CLUB2_201005      19    9209552  245925  2010/5/31 22:10:07
TAB_WAPLOG_CLUB2   PART_CLUB2_201006      20    8774146  234721  2010/7/1   22:07:59
TAB_WAPLOG_CLUB2   PART_CLUB2_201007      21    8756020  236267  2010/8/12 22:09:17
TAB_WAPLOG_CLUB2   PART_CLUB2_201008      22   10072596  293444  2010/8/30 22:27:10
TAB_WAPLOG_CLUB2   PART_CLUB2_201009      23          0       0  2010/8/12 22:10:25
TAB_WAPLOG_CLUB2   PART_CLUB2_201010      24          0       0  2010/8/12 22:10:25
TAB_WAPLOG_CLUB2   PART_CLUB2_201011      25          0       0  2010/8/12 22:10:25
TAB_WAPLOG_CLUB2   PART_CLUB2_201012      26          0       0  2010/8/12 22:10:25

12 rows selected
```

然后客户又问，为什么会出现这样的问题，统计信息不是自动收集的么？

的确在这个客户环境中，统计信息是自动收集的，并且在 9 月 1 日晚 22：00 成功完成了：

```
SQL> SELECT log_id, job_name, status,
  2   TO_CHAR (log_date, 'DD-MON-YYYY HH24:MI') log_date
  3  FROM dba_scheduler_job_run_details
  4  WHERE job_name = 'GATHER_STATS_JOB' order by 1 desc;

   LOG_ID JOB_NAME                       STATUS                         LOG_DATE
--------- ---------------------------- ------------------------------ ----------------
   328617 GATHER_STATS_JOB             SUCCEEDED                      01-SEP-2010 22:16
   327857 GATHER_STATS_JOB             SUCCEEDED                      31-AUG-2010 22:02
   327217 GATHER_STATS_JOB             SUCCEEDED                      30-AUG-2010 22:28
   325418 GATHER_STATS_JOB             SUCCEEDED                      28-AUG-2010 06:02
   325237 GATHER_STATS_JOB             SUCCEEDED                      27-AUG-2010 22:10
   324517 GATHER_STATS_JOB             SUCCEEDED                      26-AUG-2010 22:02
   323838 GATHER_STATS_JOB             SUCCEEDED                      25-AUG-2010 22:08
   323157 GATHER_STATS_JOB             SUCCEEDED                      24-AUG-2010 22:01
   322458 GATHER_STATS_JOB             SUCCEEDED                      23-AUG-2010 22:09
   320638 GATHER_STATS_JOB             SUCCEEDED                      21-AUG-2010 06:03
   320397 GATHER_STATS_JOB             SUCCEEDED                      20-AUG-2010 22:08
```

那么剩下的唯一可能就是，数据是在统计信息收集之后发生的变化，这样统计信息虽然收集成功了，但是却无法预知 22：16 分之后的数据变化，而一旦数据入库滞后，则就可能出现“BUG”。查询确认一下数据，果然发现当日（9 月 2 日）的数据并未入库，数据库中只存在 9 月 1 日的数据，由此判断日志的入库是滞后的：

```
SQL> select trunc(sysdate) from dual;

TRUNC(SYSDATE)
-------------------
```

```
2010-09-02 00:00:00
SQL> select count(*) from statistic.tab_waplog_club2 where DACCESS>trunc(sysdate);

  COUNT(*)
---------
         0

SQL> select count(*) from statistic.tab_waplog_club2 where DACCESS>trunc(sysdate-1);

  COUNT(*)
---------
    482113
```

后经用户证实，数据是在每日凌晨两点入库，这样就导致了这一问题的出现。

那么这一问题在什么时候出现呢？仅仅会在每月 1 日。

当一个新分区在每月 1 日启用，数据在凌晨两点入库，也就是 9 月 2 日凌晨两点入库 9 月 1 日的数据；而统计信息是在每日晚 22：00 收集，这就导致了 9 月 1 日晚 22：00 收集的统计信息认为表中不存在 9 月份的数据，在凌晨 6：00 开始的 JOB 任务，在制定执行计划时就发生了错误的判断和选择。而如果再过一天，9 月的数据被收集，笛卡儿积的执行计划是不会被选择出来的。

所以这个“Bug”仅会在每月 1 日发生。我们可以选择在加载数据之后进行统计信息收集就可以避免这个问题。在用户环境中，通过对改变进行统计信息收集之后，性能马上恢复了正常：

```
SQL> analyze table TAB_WAPLOG_CLUB2 partition (PART_CLUB2_201009) compute statistics;

Table analyzed.

SQL> select table_name,partition_name,partition_position,num_rows,blocks,last_analyzed
  2  from dba_tab_partitions a
  3  where a.table_name='TAB_WAPLOG_CLUB2' and a.partition_name like '%2010%'
  4  /

TABLE_NAME         PARTITION_NAME     ITION   NUM_ROWS BLOCKS LAST_ANALYZED
------------------ ------------------ ----- ---------- ------ -------------------
TAB_WAPLOG_CLUB2   PART_CLUB2_201001     15   31882877 923374 2010/2/1 22:33:00
TAB_WAPLOG_CLUB2   PART_CLUB2_201002     16   21827605 635691 2010/2/27 6:08:30
TAB_WAPLOG_CLUB2   PART_CLUB2_201003     17   26856055 784543 2010/4/1 22:22:30
TAB_WAPLOG_CLUB2   PART_CLUB2_201004     18   19129756 557686 2010/4/27 22:26:55
TAB_WAPLOG_CLUB2   PART_CLUB2_201005     19    9209552 245925 2010/5/31 22:10:07
TAB_WAPLOG_CLUB2   PART_CLUB2_201006     20    8774146 234721 2010/7/1 22:07:59
TAB_WAPLOG_CLUB2   PART_CLUB2_201007     21    8756020 236267 2010/8/12 22:09:17
TAB_WAPLOG_CLUB2   PART_CLUB2_201008     22   10072596 293444 2010/8/30 22:27:10
TAB_WAPLOG_CLUB2   PART_CLUB2_201009     23     482113  14684 2010/9/2 16:26:04
TAB_WAPLOG_CLUB2   PART_CLUB2_201010     24          0      0 2010/8/12 22:10:25
TAB_WAPLOG_CLUB2   PART_CLUB2_201011     25          0      0 2010/8/12 22:10:25
TAB_WAPLOG_CLUB2   PART_CLUB2_201012     26          0      0 2010/8/12 22:10:25

12 rows selected
```

通过 AWR 的统计信息，可以对比一下两次执行的资源消耗（如图 1-27 所示），我们看到调整后 SQL 的执行时间大约是 17 秒，逻辑读也大大降低。

Plan Statistics

- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	11,760,005	11,760,005.04	12.58
CPU Time (ms)	11,755,542	11,755,541.57	33.94
Executions	1		
Buffer Gets	450,362,425	450,362,425.00	34.83
Disk Reads	9,237	9,237.00	0.03
Parse Calls	1	1.00	0.00
Rows	32	32.00	
User I/O Wait Time (ms)	14,112		
Cluster Wait Time (ms)	0		
Application Wait Time (ms)	0		
Concurrency Wait Time (ms)	6		
Invalidations	0		
Version Count	4		
Sharable Mem(KB)	95		

Plan Statistics

- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total

Stat Name	Statement Total	Per Execution	% Snap Total
Elapsed Time (ms)	37,510	37,510.00	0.02
CPU Time (ms)	1,026	1,026.29	0.00
Executions	1		
Buffer Gets	99,242	99,242.00	0.01
Disk Reads	1,740	1,740.00	0.01
Parse Calls	1	1.00	0.00
Rows	32	32.00	
User I/O Wait Time (ms)	16,545		
Cluster Wait Time (ms)	0		
Application Wait Time (ms)	0		
Concurrency Wait Time (ms)	0		
Invalidations	0		
Version Count	1		
Sharable Mem(KB)	6		

图 1-27　对比两次执行的资源消耗

执行计划的 cardinality (rows)评估

在前面我们曾经提到，当查询 SQL 访问的谓词超出了数据库中统计信息记录的值限，比如某个数值的范围是 1~100，当访问 200 的取值，超出了这个范围，由于统计信息收集通常是通过采样、定时来实现的，所以真实的数据中可能仍然存在满足条件的数据，这是数据库会估算指定一个 Density 用于 cardinality 的计算。

首先创建一个简单的测试表用于说明这个问题：

```
SQL> create table eygle as select object_id,owner,object_name
  2 from dba_objects where rownum <20001;
Table created.

SQL> insert into eygle select * from eygle;
20000 rows created.
SQL> insert into eygle select * from eygle;
40000 rows created.
SQL> insert into eygle select * from eygle;
80000 rows created.
SQL> commit;
Commit complete.
```

收集该表的统计信息：

```
SQL> exec dbms_stats.gather_table_stats(user,'EYGLE');

PL/SQL procedure successfully completed.
```

查询记录一下相关的统计信息：

```
SQL> select table_name,num_rows,blocks,avg_row_len,sample_size
  2  from user_tables where table_name='EYGLE';

TABLE_NAME                     NUM_ROWS     BLOCKS  AVG_ROW_LEN  SAMPLE_SIZE
------------------------------ ---------- ---------- ------------ -----------
EYGLE                            159340        884           32       39835
```

```
SQL> select column_name,data_type,num_distinct,density,1/num_distinct
comp_den ,last_analyzed,
     sample_size,histogram
  2  from user_tab_cols where table_name='EYGLE'
  3  /

COLUMN_NAME  DATA_TYPE NUM_DISTINCT    DENSITY    COMP_DEN LAST_ANALYZED SAMPLE_SIZE HISTOGRAM
------------ --------- ------------ ---------- ---------- ------------- ----------- ---------
OBJECT_ID    NUMBER           20054  4.98653635 4.98653635 2010/8/8 9:48       39835 NONE
OWNER        VARCHAR2             7  0.14285714 0.14285714 2010/8/8 9:48        6845 NONE
OBJECT_NAME  VARCHAR2         12424  8.04893754 8.04893754 2010/8/8 9:48       39835 NONE
```

注意以上的 DENSITY 计算，在不存在柱状图时，Density = 1/ NUM_DISTINCT，此时计算 cardinality 时，会选择使用公式：

```
cardinality = NUM_ROWS / NUM_DISTINCT = NUM_ROWS*(1 / NUM_DISTINCT)
```

数据库中的基本信息如下：

```
SQL> select max(object_id) from eygle;

MAX(OBJECT_ID)
--------------
         20535
SQL> select count(*) from eygle;

  COUNT(*)
----------
    160000
SQL> select count(*) from eygle where object_id=1000;

  COUNT(*)
----------
         8
```

通过 10053 事件跟踪以下查询：

```
SQL> alter session set events '10053 trace name context forever,level 1';

Session altered.

SQL> explain plan for select count(*) from eygle where object_id=1000;

Explained.

SQL> explain plan for select count(*) from eygle where object_id=10000;

Explained.

SQL> explain plan for select count(*) from eygle where object_id=25000;

Explained.

SQL> explain plan for select count(*) from eygle where object_id=30000;
```

```
Explained.

SQL> explain plan for select count(*) from eygle where object_id=40000;

Explained.

SQL> explain plan for select count(*) from eygle where object_id=50000;

Explained.
```

此时来分析一下跟踪文件，将非常有助于我们理解 Oracle 是如何完成计算。对于以下查询块：

```
****************
QUERY BLOCK TEXT
****************
select count(*) from eygle where object_id=1000
*********************
```

首先基本统计信息部分我们可以看到关于表中行数、数据块数、平均行长等信息。注意这里的 Rows 为 159340 是采样估算值，实际表中的记录数为 160000 条：

```
BASE STATISTICAL INFORMATION
***********************
Table Stats::
 Table: EYGLE  Alias: EYGLE
   #Rows: 159340  #Blks:  884  AvgRowLen:  32.00
```

在单表访问路径部分，紧接着的信息是 Cardinality 的评估，评估的结果由：

```
(Card: Original: 159340 )* (Density: 4.9865e-005)= (Computed: 7.95 Non Adjusted: 7.95)
                                                 = Rounded: 8
**************************************
SINGLE TABLE ACCESS PATH
  ----------------------------------------
  BEGIN Single Table Cardinality Estimation
  ----------------------------------------
  Column (#1): OBJECT_ID(NUMBER)
    AvgLen: 5.00 NDV: 20054 Nulls: 0 Density: 4.9865e-005 Min: 4 Max: 20530
  Table: EYGLE  Alias: EYGLE
    Card: Original: 159340  Rounded: 8  Computed: 7.95  Non Adjusted: 7.95
  ----------------------------------------
  END  Single Table Cardinality Estimation
```

全表访问路径的成本是：197.71。

```
  ----------------------------------------
  Access Path: TableScan
    Cost:  197.71  Resp: 197.71  Degree: 0
      Cost_io: 195.00  Cost_cpu: 38163353
      Resp_io: 195.00  Resp_cpu: 38163353
  Best:: AccessPath: TableScan
        Cost: 197.71  Degree: 1  Resp: 197.71  Card: 7.95  Bytes: 0
**************************************
```

这个查询最后的执行计划，其 Rows 就是评估的 8，完全符合真实的数据情况：

```
sql_id=3sukwzrqc7dy1.
Current SQL statement for this session:
explain plan for select count(*) from eygle where object_id=1000

============
Plan Table
============
-------------------------------------+-------------------------------------+
| Id  | Operation           | Name   |  Rows  | Bytes |  Cost   | Time       |
-------------------------------------+-------------------------------------+
| 0   | SELECT STATEMENT    |        |        |       |  198    |            |
| 1   | SORT AGGREGATE      |        |    1   |    5  |         |            |
| 2   | TABLE ACCESS FULL   | EYGLE  |    8   |   40  |  198    | 00:00:03   |
-------------------------------------+-------------------------------------+
Predicate Information:
----------------------
2 - filter("OBJECT_ID"=1000)

Content of other_xml column
===========================
 db_version    : 10.2.0.4
 parse_schema  : SCOTT
 plan_hash     : 3602634261
```

当查询块访问 object_id=25000 时，就超出了最大数据范围：

```
****************
QUERY BLOCK TEXT
****************
select count(*) from eygle where object_id=25000
*********************
```

在统计信息中显示，OBJECT_ID 的 Min 为 4，Max 为 20530：

```
***************************************
SINGLE TABLE ACCESS PATH
 ----------------------------------------
 BEGIN Single Table Cardinality Estimation
 ----------------------------------------
 Column (#1): OBJECT_ID(NUMBER)
  AvgLen: 5.00 NDV: 20054 Nulls: 0 Density: 4.9865e-005 Min: 4 Max: 20530
 Using prorated density: 3.9006e-005 of col #1 as selectivity of out-of-range value pred
 Table: EYGLE  Alias: EYGLE
  Card: Original: 159340  Rounded: 6  Computed: 6.22  Non Adjusted: 6.22
 ----------------------------------------
 END  Single Table Cardinality Estimation
 ----------------------------------------
```

注意，此时 Oracle 分配了一个估算密度，**3.9006e-005** 最为 OUT-OF-RANGE 的计算，对于此次查询，估算的 Cardinality 是 6，其执行计划是：

```
-------------------------------------+-------------------------------------+
| Id  | Operation           | Name   | Rows   | Bytes  | Cost   | Time     |
-------------------------------------+-------------------------------------+
```

```
| 0   | SELECT STATEMENT    |         |        |        |  198   |          |
| 1   |  SORT AGGREGATE     |         |    1   |    5   |        |          |
| 2   |   TABLE ACCESS FULL | EYGLE   |    6   |   30   |  198   | 00:00:03 |
-------------------------------------+-----------------------------------+
```

进一步的，当 OBJECT_ID 更加远离实际值时，则 prorated density 趋向于一个确定值：

```
 Using prorated density: 3.1379e-006 of col #1 as selectivity of out-of-range value pred
```

此时的 Cardinality 估算为 1 ：

```
 ----------------------------------------
 Column (#1): OBJECT_ID(NUMBER)
  AvgLen: 5.00 NDV: 20054 Nulls: 0 Density: 4.9865e-005 Min: 4 Max: 20530
 Using prorated density: 3.1379e-006 of col #1 as selectivity of out-of-range value pred
 Table: EYGLE  Alias: EYGLE
  Card: Original: 159340  Rounded: 1  Computed: 0.50  Non Adjusted: 0.50
 ----------------------------------------
```

执行计划显示的就是：

```
-------------------------------------+-----------------------------------+
| Id  | Operation           | Name   | Rows   | Bytes  | Cost   | Time       |
-------------------------------------+-----------------------------------+
| 0   | SELECT STATEMENT    |        |        |        |  198   |            |
| 1   |  SORT AGGREGATE     |        |    1   |    5   |        |            |
| 2   |   TABLE ACCESS FULL | EYGLE  |    1   |    5   |  198   |  00:00:03  |
-------------------------------------+-----------------------------------+
```

X$KTUXE 与 Oracle 的死事务恢复

X$KTUXE 是数据库中非常神秘的一个对象表，当然其本质上是 C 定义的一个结构体，在数据库中可以看到其结构：

```
SQL> desc x$ktuxe
 Name                            Null?    Type
 ------------------------------- -------- -------- -----------------
 ADDR                                     RAW(4)
 INDX                                     NUMBER
 INST_ID                                  NUMBER
 KTUXEUSN                                 NUMBER
 KTUXESLT                                 NUMBER
 KTUXESQN                                 NUMBER
 KTUXERDBF                                NUMBER
 KTUXERDBB                                NUMBER
 KTUXESCNB                                NUMBER
 KTUXESCNW                                NUMBER
 KTUXESTA                                 VARCHAR2(16)
 KTUXECFL                                 VARCHAR2(24)
 KTUXEUEL                                 NUMBER
 KTUXEDDBF                                NUMBER
```

```
KTUXEDDBB                                       NUMBER
KTUXEPUSN                                       NUMBER
KTUXEPSLT                                       NUMBER
KTUXEPSQN                                       NUMBER
KTUXESIZ                                        NUMBER
```

那么这个表的 KTUXE 代表的含义是什么呢?

通过数据库的 V$TYPE_SIZE 视图，我们可以找到 Oracle 的自解释信息：

```
SQL> select * from v$type size where component='KTU';

COMPONEN TYPE    DESCRIPTION                      TYPE_SIZE
-------- ------- -------------------------------- ----------
KTU      KTUBH   UNDO HEADER                              16
KTU      KTUXE   UNDO TRANSACTION ENTRY                   40
KTU      KTUXC   UNDO TRANSACTION CONTROL                104
```

所以 KTUXE 代表的就是：[K]ernel [T]ransaction [U]ndo Transa[x]tion [E]ntry (table)。在这个表中，数据库展示了回滚段头的事务表信息。通过这个结构体我们可以获得数据库事务相关的重要信息。

最早接触到这个表是在 Oracle 8i 年代，当我们试图获得最接近当前 SCN 的数值时，可以使用如下 SQL 来查询：

```
SQL> select max(ktuxescnw*power(2,32)+ktuxescnb) SCN from x$ktuxe;

      SCN
---------
   900538
```

这个查询，是获取回滚段事务表中最大的 SCN Base 和 SCN Wrap，通过两者计算出最大的 SCN。由于 SCN 和事务紧密相关，通常这个 SCN 就很接近当前系统最大的 SCN 值。

当然，在后期的版本中，我们通过 dbms_flashback.get_system_change_number 可以很容易的获得当前的 SCN 值：

```
SQL> select max(ktuxescnw*power(2,32)+ktuxescnb) SCN,
  2  dbms_flashback.get_system_change_number from x$ktuxe;

       SCN  GET_SYSTEM_CHANGE_NUMBER
---------- ------------------------
    900538                   900599
```

现在看起来这两者之间是具有一定的差距的。不过 SCN 本质上来自于内存地址，通过内存变量的转储，则可以获得 SCN 的最本质来源：

```
SQL> oradebug setmypid
Statement processed.
SQL> oradebug dumpvar sga kcsgscn_
kcslf kcsgscn_ [3E494C0, 3E494E0) = 0000 0000 000DBE17 000004B3 00000000 00000000 00000000 00000000
03E492B8
SQL> select to_number('DBE17','xxxxx') from dual;
```

```
TO_NUMBER('DBE17','XXXXX')
--------------------------
                    900631
```

X$KTUXE 表的另外一个重要功能是，可以获得无法通过 v$transaction 来观察的死事务信息，当一个数据库发生异常中断，或者进行延迟事务恢复时，数据库启动后，无法通过 V$TRANSACTION 来观察事务信息，但是 X$KTUXE 可以帮助我们获得这些信息。

该表中的 KTUXECFL 代表了事务的 Flag 标记，通过这个标记可以找到那些 Dead 事务：

```
17:30:37 SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;
KTUXECFL               COUNT(*)
-------------------- ----------
DEAD                          1
NONE                       2393
SCO|COL                       8
```

KTUXESIZ 用来记录事务使用的回滚段块数，可以通过观察这个字段来评估恢复进度：

```
16:59:47 SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ
         2 from x$ktuxe where  KTUXECFL ='DEAD';

ADDR               KTUXEUSN   KTUXESLT   KTUXESQN   KTUXESIZ
---------------- ---------- ---------- ---------- ----------
FFFFFFFF7D07B91C         10         39    2567412    1086075
17:02:12 SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ
         2 from x$ktuxe where  KTUXECFL ='DEAD';

ADDR               KTUXEUSN   KTUXESLT   KTUXESQN   KTUXESIZ
---------------- ---------- ---------- ---------- ----------
FFFFFFFF7D07B91C         10         39    2567412    1086067
```

曾经遇到的一个案例，某个事务回滚经过测算需要大约 2.55 天：

```
SQL> declare
 2  l_start number;
 3  l_end   number;
 4  begin
 5    select ktuxesiz into l_start from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
 6    dbms_lock.sleep(60);
 7    select ktuxesiz into l_end from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
 8    dbms_output.put_line('time est Day:'|| round(l_end/(l_start -l_end)/60/24,2));
 9  end;
10  /

time est Day:2.55
```

可以进一步通过我们熟悉的手段将 KTUXE 中的 Entry 信息转储出来，辅助查看。这一次让我们选择 SYSTEM 回滚段：

```
SQL> select * from v$rollname where usn=0;

       USN NAME
---------- ------------------------------
         0 SYSTEM
```

```
SQL> select file_id,block_id,blocks from dba_extents where segment_name='SYSTEM';
  FILE_ID  BLOCK_ID    BLOCKS
--------- --------- ---------
        1         9         8
        1        17         8
        1       385         8
        1       393         8
        1       401         8
        1       409         8
```

我们将 BLOCK 9 转储出来：

```
SQL> alter system dump datafile 1 block 9;

System altered.
```

以下是跟踪文件的摘录：

```
Start dump data blocks tsn: 0 file#: 1 minblk 9 maxblk 9
buffer tsn: 0 rdba: 0x00400009 (1/9)
scn: 0x0000.000db42c seq: 0x01 flg: 0x04 tail: 0xb42c0e01
frmt: 0x02 chkval: 0xf71c type: 0x0e=KTU UNDO HEADER W/UNLIMITED EXTENTS
Hex dump of block: st=0, typ_found=1
 Extent Control Header
 -------------------------------------------------------------
 Extent Header:: spare1: 0     spare2: 0     #extents: 6     #blocks: 47
                 last map  0x00000000  #maps: 0     offset: 4128
    Highwater::  0x004001a0  ext#: 5     blk#: 7     ext size: 8
 #blocks in seg. hdr's freelists: 0
 #blocks below:  0
 mapblk  0x00000000  offset: 5
              Unlocked
    Map Header:: next  0x00000000  #extents: 6   obj#: 0     flag: 0x40000000
 Extent Map
 -------------------------------------------------------------
  0x0040000a  length: 7
  0x00400011  length: 8
  0x00400181  length: 8
  0x00400189  length: 8
  0x00400191  length: 8
  0x00400199  length: 8

 TRN CTL:: seq: 0x0035 chd: 0x0010 ctl: 0x001a inc: 0x00000000 nfb: 0x0001
           mgc: 0x8002 xts: 0x0068 flg: 0x0001 opt: 2147483646 (0x7ffffffe)
           uba: 0x004001a0.0035.20 scn: 0x0000.000bcb32
Version: 0x01
 FREE BLOCK POOL::
   uba: 0x004001a0.0035.20 ext: 0x5  spc: 0x2d6
   uba: 0x00000000.0033.21 ext: 0x3  spc: 0x11e4
   uba: 0x00000000.0032.37 ext: 0x2  spc: 0x62c
   uba: 0x00000000.002b.0c ext: 0x1  spc: 0x19de
   uba: 0x00000000.0000.00 ext: 0x0  spc: 0x0
 TRN TBL::
```

```
 index  state cflags  wrap#   uel     scn             dba         parent-xid
 ------------------------------------------------------------------------------
  0x00   9    0x00    0x002c  0x002f  0x0000.000db422  0x004001a0  0x0000.000.00000000
  0x01   9    0x00    0x002c  0x0013  0x0000.000d9c2a  0x0040019f  0x0000.000.00000000
  0x02   9    0x00    0x002c  0x004e  0x0000.000d9763  0x0040019f  0x0000.000.00000000
  0x03   9    0x00    0x002c  0x0047  0x0000.000d9c0e  0x0040019f  0x0000.000.00000000
  0x04   9    0x00    0x002c  0x005c  0x0000.000d975b  0x0040019f  0x0000.000.00000000
  0x05   9    0x00    0x002c  0x0024  0x0000.000d9c24  0x0040019f  0x0000.000.00000000
......
  0x60   9    0x00    0x002b  0x003e  0x0000.000d974e  0x0040019e  0x0000.000.00000000
  0x61   9    0x00    0x002b  0x000f  0x0000.000db40e  0x004001a0  0x0000.000.00000000
End dump data blocks tsn: 0 file#: 1 minblk 9 maxblk 9
```

如果用 BBED 再来查看以下段头的数据结构，一切就会变得清晰明了：

```
D:\oracle\9.2.0\bin>bbed parfile=parfile.txt
口令:

BBED: Release 2.0.0.0.0 - Limited Production on 星期四 8 月 12 15:50:08 2010

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

************* !!! For Oracle Internal Use only !!! ***************

BBED> set file 1 block 10
      FILE#             1
      BLOCK#           10

BBED> map /v
 File: D:\ORACLE\ORADATA\ENMO\SYSTEM01.DBF (1)
 Block: 10                            Dba:0x0040000a
------------------------------------------------------------
 Unlimited Undo Segment Header

 struct kcbh, 20 bytes                @0
   ub1 type_kcbh                      @0
   ub1 frmt_kcbh                      @1
   ub1 spare1_kcbh                    @2
   ub1 spare2_kcbh                    @3
   ub4 rdba_kcbh                      @4
   ub4 bas_kcbh                       @8
   ub2 wrp_kcbh                       @12
   ub1 seq_kcbh                       @14
   ub1 flg_kcbh                       @15
   ub2 chkval_kcbh                    @16
   ub2 spare3_kcbh                    @18

 struct ktech, 72 bytes               @20
   ub4 spare1_ktech                   @20
   word tsn_ktech                     @24
   ub4 lastmap_ktech                  @28
   ub4 mapcount_ktech                 @32
   ub4 extents_ktech                  @36
   ub4 blocks_ktech                   @40
```

```
  ub2 mapend_ktech                  @44
  struct hwmark_ktech, 32 bytes     @48
  struct locker_ktech, 8 bytes      @80
  ub4 flag_ktech                    @88

struct ktemh, 16 bytes              @92
  ub4 count_ktemh                   @92
  ub4 next_ktemh                    @96
  ub4 obj_ktemh                     @100
  ub4 flag_ktemh                    @104

struct ktetb[6], 48 bytes           @108
  ub4 ktetbdba                      @108
  ub4 ktetbnbk                      @112

struct ktuxc, 104 bytes             @4148
  struct ktuxcscn, 8 bytes          @4148
  struct ktuxcuba, 8 bytes          @4156
  sb2 ktuxcflg                      @4164
  ub2 ktuxcseq                      @4166
  sb2 ktuxcnfb                      @4168
  ub4 ktuxcinc                      @4172
  sb2 ktuxcchd                      @4176
  sb2 ktuxcctl                      @4178
  ub2 ktuxcmgc                      @4180
  ub4 ktuxcopt                      @4188
  struct ktuxcfbp[5], 60 bytes      @4192

struct ktuxe[204], 8160 bytes       @4252
  ub4 ktuxexid                      @4252
  ub4 ktuxebrb                      @4256
  struct ktuxescn, 8 bytes          @4260
  sb4 ktuxesta                      @4268
  ub1 ktuxecfl                      @4269
  sb2 ktuxeuel                      @4270

ub4 tailchk                         @8188
```

结合前面的 V$TYPE_SIZE 视图，参考上面的 BBED 输出，可以看到 KTU 等事务信息，以及 KTE 的区间使用信息等：

```
SQL> select * from v$type size where component in ('KTE','KTU');

COMPONEN TYPE     DESCRIPTION                    TYPE_SIZE
-------- -------- ------------------------------ ----------
KTE      KTECT    EXTENT CONTROL                         44
KTE      KTECH    EXTENT CONTROL                         72
KTE      KTETB    EXTENT TABLE                            8
KTU      KTUBH    UNDO HEADER                            16
KTU      KTUXE    UNDO TRANSACTION ENTRY                 40
KTU      KTUXC    UNDO TRANSACTION CONTROL              104
```

参考文献

[1] 《Oracle DBA 手记》编委会. Oracle DBA 手记——数据库诊断案例与性能优化实践. 北京：电子工业出版社, 2010 年 1 月

[2] 盖国强. 循序渐进 Oracle——数据库管理、优化与备份恢复. 北京：人民邮电出版社, 2007 年 9 月

[3] **Metalink:Slow Performance Of DBA_AUDIT_SESSION Query From EM [ID 829103.1]**

[4] **Metalink**:Node Crash Due To Large Amount Of Racgimon Threads, OS bug (QX:QXCR1000940361) [ID 883801.1]

[5] **Metalink**:BUG 3591101 - AWR / DBSMP USER ISSUES QUERIES FROM SYS.DBA_FREE_SPACE THAT TAKES TOO LONG

[6] **Metalink:BUG 5376783: DBMS_SPACE.OBJECT_GROWTH_TREND CALL TAKES A LOT OF DISK READS**

[7] Metalink:Note.61552.1 DIAGNOSING DATABASE HANGING ISSUES

[8] **Metalink**:Note:39282.1 - ORA-600 [4193] "seq# mismatch while adding undo record"

[9] **Metalink**:ORA-600 [kcratr1_lostwrt] Doc ID: Note:248718.1

[10] 案例:Move 系统表 DEPENDENCY$导致索引失效的数据库故障.
http://www.eygle.com/archives/2005/11/move_dependency_index_unusable.html

[11] Oracle® Database SQL Reference.
http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/queries003.htm

[12] Oracle Concepts.
http://download.oracle.com/docs/cd/B19306_01/server.102/b14220/toc.htm

索引

北京博文视点（www.broadview.com.cn）资讯有限公司成立于2003年，是工业和信息化部直属的中央一级科技与教育出版社——电子工业出版社（PHEI）下属旗舰级子公司，在六年的开拓、探索和成长中，已成为中国颇具影响力的专业IT图书策划和服务提供商。

六年来，博文视点以开发IT类图书选题为主业，励精图治、兢兢业业，打造了一支团结一心的专业队伍，并形成了自身独特的竞争优势。一直以来，博文视点始终以传播完美知识为己任，用诚挚之心奉献精品佳作，年组织策划图书达300个品种，同时开展相关信息和知识增值服务，赢得了众多才华横溢的作者朋友和肝胆相照的合伙伙伴，已经成为IT图书领域的高端品牌。

我们的理念：创新专业图书服务体制；培养职业策划图书服务队伍；打造精品图书品牌；完善全面出版服务平台。

我们的目标：面向IT专业人员的出版物提供相关服务。

我们的团队：一个整合了专业技术人员和专业服务人员的团队；一个充满创新意识和创作激情的团队；一个不断进取、追求卓越的团队。

我们的服务：善待作者　尊重作者　提升作者

我们的实力：优秀的专业编辑队伍
全方位立体化的强大的市场推广平台
实力雄厚的电子工业出版社的渠道平台

“走出软件作坊独辟蹊经　人道编程之美，
追踪加密解密庖丁解牛　精雕夜读天书。”

路漫漫其修远，博文视点愿与所有曾经帮助、关心过我们的朋友、作者、合作伙伴携手奋斗。未来之路，不可限量！

地址：北京市万寿路173信箱电子工业出版社博文视点资讯有限公司
邮编：100036　**总机：**010-88254356　**传真：**010-88254356-802

武汉分部地址：武汉市洪山区吴家湾湖北信息产业科技大厦1402室
邮编：430074　**总机：**027-87690813　**传真：**027-87690013

欢迎投稿：bvtougao@gmail.com
读者邮箱：reader@broadview.com.cn

博文视点官方博客：http://blog.csdn.net/bvbook
博文视点官方网站：http://www.broadiew.com.cn

反侵权盗版声明

定价：45.00元

ISBN 978-7-121-11946-0

策划编辑：周　筠　　责任编辑：杨绣国　　封面设计：杨小勤

上架建议　数据库