
Expert
oracle
Database

Expert Oracle Database Architecture:

Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions, Second Edition

by Thomas Kyte

전문가를 위한 오라클 데이터베이스 아키텍처 제2판

오라클 데이터베이스 9i, 10g, 11g 프로그래밍 기법과 해법

2판 1쇄 발행 2011년 6월 28일

지은이 토마스 카이트
옮긴이 오선경, 한준희, 유동오, 임영섭, 김창배 | **펴낸이** 장성두 | **책임편집** 안주연

본문디자인 임성민 | **표지디자인** 미디어픽스

주소 경기도 파주시 교하읍 파주신도시 에이15-1블록 한빛마을 휴먼빌 201-502
전화 070-8201-9010 | **팩스** 02-6280-0405
홈페이지 www.jpub.kr | **펴낸곳** 제이펍

출판신고 2009년 11월 10일 제406-2009-000087호

용지 신승지류유통 | **인쇄** 해외정판사 | **제본** 춘산제본

ISBN 978-89-94506-19-7 (93560)

값 45,000원

전문가를 위한 오라클 데이터베이스 아키텍처

제2판

오라클 데이터베이스 9i, 10g, 11g 프로그래밍 기법과 해법

토마스 카이트 지음 | 오선경, 한준희, 유동오, 임영섭, 김창배 옮김

Jpub 제이펍

차 례

추천사 I

우리가 살고 있는 스마트한 시대에서 데이터에 관한 관심과 중요성은 날로 증가하고 있으며, 데이터(Data) → 정보(Information) → 지식(Knowledge) → 지혜/예지(Wisdom)를 원하는 욕구는 갈수록 폭발적일 수밖에 없다. 시대의 변화와 시장의 요구에 능동적으로, 빠르게, 그리고 적시에 대응하기 위한 의사결정, 즉 고품질의 가치 있는 정보와 지식 인프라를 갖추는 것은 비즈니스 부서의 핵심 역량이라 해도 과언이 아닐 것이다. 고부가가치 자산인 데이터를 저장하고 활용을 가능하게 하는 데이터베이스 기술은 매우 빠르게 발전하고 다양한 기능을 제공하고 있다.

어느 분야건 개념과 작동 원리를 이해하고 사용해야 최대의 효율을 낼 수 있듯이, 데이터베이스 역시 마찬가지다. 데이터베이스를 사용하는 시스템 환경이 온라인 처리계 환경인지, 정보 활용 분석계 환경인지에 따라 최적화 방법이 다르며, 비즈니스의 성격에 따라 활용방안도 다를 수밖에 없다. 데이터베이스의 개념 및 작동 원리를 이해하지 못한다면, 데이터베이스를 효과적으로 사용하지 못할 뿐 아니라 데이터베이스가 기본적으로 쉽게 제공하는 기능을 애플리케이션으로 개발하기 위해 방대한 비용을 지불하며 어렵게 구현하게 된다. 실제로 이런 어처구니없는 과오를 범하는 경우를 주변에서 어렵지 않게 찾아볼 수 있다.

데이터베이스의 물리설계에서부터 개발, 성능최적화(튜닝), 성능고도화에 이르는 과정에는 적용 데이터베이스의 원리를 이해한 전문가가 절실히 필요하며, 이런 전문가들이 함께 했을 경우에만 저비용으로 최고의 효율을 이끌어 낼 수 있을 것이다. 데이터베이스의 개념과 작동 원리를 이해해야 하는 대상자는 개발자, 설계자, 시스템 관리자, 정책결정자에게까지 매우 다양하다.

오라클 데이터베이스 전문가인 토마스 카이트의 이 책에는 다른 어느 책에서도 쉽게 설명하지 못하는 중요한 개념을 상세하게 소개하고 있으며, 실제 사례를 통해 오라클 기능의 장단점을 안내하고 있다. 오라클 매뉴얼은 사용방법에 관해서는 소개하고 있지만, '어떤 때 무엇을 사용하는 것이 효과적이며,

어떠한 기술은 적합하지 않는다' 라는 식으로 설명하지 않기 때문에 토마스 카이트의 이 책은 오라클 사용자들에게 실질적인 도움을 주는 베스트셀러라 할 수 있다. 특히 구현 시스템에 정말로 필요한 기능이 무엇이고, 그 기능의 장단점에 대해 잘 소개하고 있다.

2003년에 초판이 국내에 소개된 후 8년 만에 국내 최고의 데이터베이스 컨설팅 전문가 그룹인 ㈜비투엔컨설팅의 우수한 컨설턴트들이 실무의 경험을 기반으로 개정판을 공동 번역하여 출간하였다. 이 책을 접하는 독자 모두가 오라클 데이터베이스 전문가로 활약할 수 있기를 기대한다.

조 광 원
한국DB산업협의회 컨설팅분과 위원장 / ㈜비투엔컨설팅 대표이사

추천사 Ⅱ

내가 Oracle RDBMS를 처음 만난 건 1987년이나 1988년쯤 됐을 때다. 어느 날 상사가 내 책상에 작은 상자를 던지며 "Oracle이라고 하는 새로운 제품이 새롭게 출시된 것 같으니 이거 가지고 몇 주 놀아보고 우리한테 도움이 될 만한 게 있는지 좀 말해주게나"라고 말했던 것 같다.

그때 버전은 5.0.22였던 것 같고, 며칠 후에 오라클을 시작했는데 매우 쉽게 시작할 수 있었다. Forms 2.0과 SQL*Report 및 다른 모든 것을 포함하여 출력한 매뉴얼 세트가 아주 작은 서류가방에 들어갈 수 있을 정도로 적은 양이었고, create table 구문에 대한 문서도 기껏해야 세 페이지 정도밖엔 안 됐다.

만약 여러분이 11.2 SQL Reference 매뉴얼 PDF 파일을 조사해본다면, 아마 create table 문은 16-6페이지에서 시작해 16-79페이지까지 총 74페이지에 걸쳐 발견할 수 있을 것이다. 내가 가장 최근 조사한 9i에서 총 페이지 수는 무려 20,000페이지가 넘었고, 10*g*와 11*g*에서 그 수가 줄어들지는 않았으리라 생각한다.

아주 얇은 5.0.22 버전의 세 권의 매뉴얼로 오라클이 우리가 작업하는 데 어떤 도움을 주며, 어떻게 효과적으로 일하는지 모든 것을 아는 데는 그리 많은 시간이 걸리지 않았던 것 같다. 옵션도 많이 없어서 잘못을 저지를 위험조차 제공하지 않았다. 하지만 요즘은 어떤가? 오라클의 핵심은 거대한 옵션과 기능 속에 파묻혀 보이지도 않은 상태에서 오라클을 시작해야 하지 않은가? 더욱 안 좋은 점은 여러분이 진정 이해하고자 하는 세부 기능들은 쉽게 얻을 수는 있지만, 그걸 가지고 시작하는 것이 너무 어려운 정보의 산에 묻혀 있다는 점이다.

대답은 아주 간단하다.

1단계: 오라클 콘셉트 매뉴얼(concept manual)을 읽어보라. 오라클이 무엇인지 모든 것을 얻게 될 것이다.

2단계: 토마스 카이트의 책을 읽어보라. 여러분이 제일 처음 "select 'hello world' from dual"부터 시작해서 아주 자신 있게 "이 테이블에 대해 이 컬럼들을 오브플로우 영역으로 저장하는 range partition IOT를 사용할 것인데, 그 이유는~"이라고 말할 수 있도록 이성적인 절차를 통해 학습하고 경험할 수 있도록 인도할 것이다.

톰(토마스)은 이 책에서 세 가지를 조합하였다. 첫 번째로, 세부 기술과 '어떻게'에 숨어 있는 '왜'를 손쉽게 이해할 수 있도록 대화체를 사용하였다. 두 번째로, 서로 연결되지 못하는 임의의 팁들을 그저 수집하는 것보다 목표를 향하여 지식을 축적할 수 있도록 '줄거리(storyline)'를 구축하였다. 세 번째로, 기능이 어떻게 작동하는지, 여러분이 어떻게 작업하고 생각할 수 있는지 알려주기 위해 조심스럽게 시연시스템을 통해 증명하였다.

인덱스를 한 예로 들어 보자. 다양한 유형의 인덱스가 존재하며, 각각의 차이를 분리하여 간단한 소개가 필요하다. B-tree 인덱스의 약점과 강점을 이해함으로써 실제로 어떻게 작동하는지 잘 이해할 수 있다. B-tree 인덱스에 대해 잘 알게 되었다면 '실제로는 존재하지 않는 데이터'에 대해 색인을 생성하는 function-based Index의 개념으로 이동할 수 있게 된다. 이런 관점은 오라클이 무엇을 하는지 이해하는 견해를 얻을 수 있을 뿐 아니라, 더 나아가 오라클을 통해 우리가 할 수 있는 일이 무엇인지를 알 수 있는 기회를 얻게 된다.

원래 이 모든 사실은 매뉴얼에 담겨 있다. 하지만 가용한 명령어의 원리를 가지고 통찰력을 얻었다면 여러분은 실제 문제에 대한 해결책을 적절하게 구축할 수 있을 것이다. 톰은 통찰력을 제공하며, 여러분 자신의 통찰력을 개발할 수 있도록 열심히 응원할 것이다.

솔직히 세상에 있는 모든 DBA와 개발자들이 이 책을 읽고 조심스럽게 작업하게 된다면, 나는 아마 오라클 컨설팅이 필요한 많은 수의 고객들이 떨어져 나가고 향후에는 SQL Server 사용자들에게 컨설팅 서비스를 제공할지도 모르겠다.

조나단 루이스(Jonathan Lewis)

추천사 Ⅲ

※ 이 추천사는 1판에 실렸던 추천사다(편집자주).

1914년, 토마스 J. 왓슨(Thomas J. Watson, Sr.)은 훗날 IBM이 된 회사에 합류하고, 'THINK' 라는 한 단어의 아주 간단한 모토를 주창하였다. 그는 모든 IBM 사원들에게 역할이 무엇이든 의사결정 과정에 참여하고 지능적으로 그들의 일을 처리할 수 있도록 이 모토를 간곡하게 권고하였다. 'THINK' 모토는 곧 그들의 아이콘이 되었고, 출판물, 달력과 IT와 비즈니스 관리자의 사무실에 걸려 있는 액자에도 나타나기 시작했다. 심지어 『*The New Yorker*』 잡지의 만화에도 등장할 정도였다. 'THINK' 는 1914년뿐 아니라 현재까지도 본받을 만한 아이디어라 생각한다.

최근 '다르게 생각하기(Think Different)' 는 애플 컴퓨터가 채택한 회사의 브랜드 이미지를 향상시키고 오랫동안 광고 캠페인에서 사용하는 사랑받는 슬로건으로, 많은 사람들이 일생 생활에서 아주 가깝게 활용되는 기술의 혁명을 가져오게 한 구심점이 되었다. 영문법에서 일상적으로 생각하는 법을 제안하는 'think differently' 라는 단어와는 달리, 애플의 슬로건에서 사용한 'different' 는 'think' 라는 단어가 뜻하는 것과 같이(think big과 유사) 무엇을 생각해야 하는지를 제안하고 있다. 이 광고 캠페인은 애플 컴퓨터만이 혁명적인 솔루션을 제공하며 환상적인 업무성과를 제공한다는 의미를 함축하여, 애플 컴퓨터와 함께 하면 보다 창의적인 사람이 될 수 있다고 강조하고 있다.

필자가 1981년 오라클에 합류하였을 때(그땐 Oracle이 아니고 Relational Software Incorporated이었음), 관계형 모델을 통합한 데이터베이스 시스템은 새롭고 놀랄 만한 기술이었다. 개발자, 프로그래머와 떠오르는 차세대 주자인 데이터베이스 관리자는 정규화 방법론을 이용한 데이터베이스 설계 원칙을 배웠다. 그 당시 잘 알려지지 않았진 비(非)절차형 SQL 언어는 과거의 절차형 프로그래밍 기법을 기반으로 데이터를 가공하는 능력을 가지고 있던 사람들에게도 매우 깊은 감동을 주었다. 다시 현재로 돌아와서도 생각해볼 만한 것들이 아주 많다. 새로운 기술들은 새로운 아이디어와 접근 방식뿐만 아니라 새로운 방식으로 생각하는 법까지도, 이 새로운 기술을 배우는 사람들을 변하게 만들고 있다. 과거든 현재든 이런 새로운 시도를 직접 실행하는 사람들은 데이터베이스 기술이 갖는 최고의 이점을 끌어올려 성공적으로 업무혁명을 가져 올 수 있으며, 비즈니스 문제를 해결할 효과적인 해결책을 제시하는 사람들이다.

오라클이 상업적으로 가장 먼저 소개한 SQL이라는 데이터베이스 언어에 대해 생각해보자. SQL은 애플리케이션 설계자들이 한 번에 하나의 레코드를 처리하는 반복적인 루프를 활용하는 전통적인 언어를 사용하는 대신, 비절차형(선언적) 언어를 사용하여 많은 양의 로우를 다룰 수 있도록 허용한다. 필자가 처음 SQL을 접했을 때 조인이나 서브쿼리처럼 집합을 다루는 연산으로 원하는 결과를 얻기 위해서는 절차적 사고방식과는 다른 각도로 문제를 바라봐야 한다는 것을 깨달았다. 이 새로운 기술이 진정으로 필자에게 요구했던 '이 새로운 기술'로 인해 필자는 진정으로 다르게 생각하게 되었고, 'think different'가 무엇인지 고민하는 기회가 되었다.

세트 프로세싱은 한 번에 하나씩 처리하는 방식보다 훨씬 더 효과적이므로, SQL을 최대로 활용한 애플리케이션은 그렇지 않은 애플리케이션보다 훨씬 더 잘 수행된다. 하지만 아직까지도 애플리케이션에 다른 차선책을 적용하는 경우를 놀라울 정도로 많이 본다. 사실, 대부분의 경우 오라클 파라미터 설정이나 다른 환경설정 대신 애플리케이션 설계가 전체 성능을 결정하는 주요한 요인이 된다. 그러므로 애플리케이션 개발자는 반드시 데이터베이스 기능과 프로그래밍 인터페이스뿐만 아니라 문제에 대한 접근 방식과 애플리케이션에 SQL과 같은 기능을 적용하는 방식을 배워야 할 것이다.

오라클 커뮤니티에는 보다 나은 성능을 얻거나 다양한 오라클 기능을 사용하기 위해 가장 효과적일 거라는 전통적인 지식들이 넘쳐나고 있다. 이런 지식은 가끔씩 아주 구닥다리 지식이 되거나 심지어 근거 없는 신화가 되기도 하는데, 개발자와 데이터베이스 관리자는 여전히 무비판적으로 이런 지식들을 채택하거나 합리적인 추리나 증명도 없이 확대 재생산하기도 한다.

그 사례로, "한 개의 결과가 좋으면 다량의 결과도 좋다"라는 지식이다. 이 지식은 매우 일반적이지만, 안타깝게도 옳은 경우는 매우 드물다. 한 예로 단일 시스템 콜에서 다중 로우를 입력하거나 추출할 수 있는 기능인 오라클의 Array 인터페이스를 선택했다고 가정해보자. 아주 명확하게, 애플리케이션과 데이터베이스 사이의 네트워크 메시지의 수를 줄이는 것은 매우 훌륭한 효과를 볼 수 있다. 하지만 … 100개의 로우를 한번에 fetch하는 것은 한 번에 하나의 로우를 fetch하는 것보다 훨씬 더 성능상 효과가 있다. 그런데 1,000개의 로우를 한번에 fetch하는 것이 효율 면에서 일반적으로 더 낫다고 할 수는 없으며, 특히 여러분이 메모리에 대한 요구사항을 고려한다면 더욱 더 그렇다.

무비판적인 생각의 또 다른 사례로는 더 나은 성능향상이나 신뢰성, 가용성이나 보안에 관심을 두기보다 시스템 설계나 환경에 대한 잘못된 견해에만 몰입하여 분석하고 집중하는 것이다. buffer hit ratio를 최대로 끌어올리는 시스템 튜닝 방법에 대한 일반적인 지식에 대해 잠깐 생각해보기로 하자. 몇몇 애플리케이션에서 요청한 데이터를 모두 메모리에서 처리하는 것이 성능을 최대로 높이는 것은 가능하다. 하지만 대부분의 애플리케이션 튜닝에서는 병목현상(대기 상태라고 부르기도 함)이 성능에 미치는 영향에 주목하는 것이 오히려 시스템 수준의 특정 메트릭에 집중하는 것보다 훨씬 더 나은 결과를 얻을 수 있다. 애플리케이션의 지연을 초래하는 설계 측면의 문제점을 제거하는 것이 여러분에게 훨씬 더 좋

은 성능을 보장할 것이다.

애플리케이션을 설계할 때는 문제를 작은 부분으로 분할하여 각 부분을 하나씩 해결하는 것이 가장 좋은 방법이라는 것을 알았다. 이런 접근 방식에서 종종 애플리케이션의 요구사항을 해결하기 위해 격조 높고 창의적으로 SQL을 사용할 수 있다. 종종 처음 보면 아주 복잡한 절차형 프로그램으로 해결해야 할 것 같은 문제를 하나의 SQL 구문으로도 훌륭히 처리하는 게 가능하다. 한 번에 많은 수의 로우 집합을 처리하기 위해 parallel과 같은 기법을 활용한 강력한 힘을 가진 SQL을 활용할 수 있다면, 애플리케이션 개발자들의 개발생산성을 높일 수 있을 뿐만 아니라 애플리케이션 또한 좀 더 빨리 수행할 수 있다.

때때로 어느 정도 진실에 바탕을 둔 모범 사례도(물론 100% 따르지 않더라도) 사실이 변하면 더 이상 적용할 수 없는 법이다. 과거 "최상의 성능을 얻기 위해 인덱스와 데이터는 각각 독립된 테이블스페이스에 저장하라"라는 이론이 있었다. 종종 데이터베이스 관리자가 디스크 속도나 용량, 주어진 작업환경을 고려하여 계정 변경은 전혀 고려하지 않은 채 이 개념이 가진 가치에만 매료되어 매우 강한 어조로 억지를 부리는 것을 자주 목격했다. 이 특별한 '법칙'을 평가함에 있어 여러분은 오라클 데이터베이스는 빈번하게, 그리고 최근에 사용한 데이터베이스 블록(인덱스와 관계된 블록들일 수도 있음)을 메모리에 캐시한다는 사실과, 사용자의 요청에 따라 인덱스와 데이터 블록은 동시가 아닌 순차적으로 사용한다는 것을 명심해야 한다.

컴퓨터가 얼마나 빠르든지, 얼마나 정교한 데이터베이스가 등장하든지, 프로그래밍 도구의 능력이 얼마나 강력한지에 상관없이 '사고원리(thinking discipline)'를 갖는 인간의 지능을 대신할 것은 없다. 그러므로 우리의 애플리케이션을 구성하고 있는 복잡한 기술을 배우는 것이 중요한 만큼 그 기술을 적절하게 사용하는 방법에 대해 배우는 것도 매우 중요하다.

토마스 카이트는 내가 알고 있는 아주 총명한 사람으로 오라클 데이터베이스, SQL, 성능 튜닝 및 애플리케이션 설계에 대해서도 엄청난 지식을 보유하고 있는 사람이다. 나는 톰이 'THINK'와 'Think different' 슬러건의 맹신자라 확신한다. 톰은 "물고기를 주면 하루를 살 수 있지만, 물고기 잡는 방법을 알려주면 평생을 먹고 살 수 있을 것이다"라는 속담을 맹신하는 사람이다. 톰은 오라클에 대한 지식 나눔에 열정을 다하는 사람으로서, 우리 커뮤니티에 지대한 영향과 이익을 주었다. 그러나 별것 아닌 간단한 질문에 대한 답변으로 많은 시간을 허비하기도 하였다. 아무튼 그의 도움으로 많은 사람들이 문제에 대한 접근 방식의 변화를 얻을 수 있었다.

톰은 그의 웹 사이트인 http://asktom.oracle.com에서 그의 개인적인 발언을 하기도 하고, 오라클과 그의 책으로 데이터베이스 애플리케이션을 설계하는 사람들로 하여금 '다르게 생각하는' 습관이 들게끔 노력하고 있다. 톰은 관습적인 지식과 추측을 거부하고, 대신 충분한 사례를 통해 증명한 사실을 근거로 하여 지식을 검증해야 한다고 강조하고 있다. 문제에 대한 매우 실용적이고 간단한 접근 방식을 그를 아는 주변사람들에게 전파하고 있으며, 그의 조언과 방법론을 따르면 좀 더 생산적이고 효율적인

애플리케이션을 개발할 수 있게 된다.

톰의 책은 오라클의 기능에 대한 설명과 사용 방법뿐만 아니라 기능에 대해 매우 쉽고 다양하게 접근할 수 있는 방식을 조명하고 있다.

- 미신을 믿지 마라. 모두 여러분에게 달렸다.
- 관습에 얽매인 지식을 따르지 마라. 모든 사람들이 알고 있는 지식이 틀리는 경우가 많다.
- 루머나 주관적인 의견을 맹신하지 마라. 스스로 테스트를 게을리 하지 말고, 예를 증명함으로 결정하라.
- 문제를 좀 더 단순한 질문으로 분할하고, 각 질문에 대해 단계별로 얻은 대답을 정교하게, 그리고 효율적으로 하나의 솔루션으로 취합하라.
- 여러분의 프로그램이 데이터베이스에서 좀 더 잘, 그리고 빨리 구동될 때는 더 이상 프로그램에 손을 대지 마라.
- 이상과 실생활과의 차이점을 이해하라. 즉, 현실은 이상과는 다르다.
- 기술 표준에 대해 옳지 않은 회사의 정책에 대해 끊임없이 질문하고 의심하라.
- 가까이에서 요구사항에 대한 최적의 결론이 무엇인지 전체적인 관점에서 고려하라.
- 'THINK' 에 시간을 투자하라.

톰은 오라클을 블랙박스보다 더 소중히 다루라고 권하고 있다. 또한 오라클에 데이터를 넣고 추출하는 데 중점을 두는 대신, 여러분이 오라클이 어떻게 일하고, 힘을 발휘하는지 이해하도록 도울 것이다. 오라클 기술을 창의적으로 적용하는 방법을 배움으로써 대부분의 애플리케이션 설계로 인한 문제를 빠르고 효과적으로 해결할 수 있을 것이다.

여러분이 이 책을 즐겁게 읽어나간다면 오라클 데이터베이스 기술에 대한 다양한 새로운 사실과 애플리케이션 설계에 대한 중요한 개념을 배울 것이다. 여러분이 직면한 도전과제에 '다르게 생각하는' 계기가 될 것이라고 확신한다.

IBM의 왓슨(Watson)은 "시간이 생긴 이후의 사고(thought)는 모든 진보의 시조였다. '생각해 본 적이 없다(I didn' t think)' 는 말은 엄청난 비용을 지불하는 단어다"라고 말했다. 이 책에서 배울 지식과 기술로 자신을 무장하고 생각하는 것을 즐긴다면 엄청난 금액의 비용을 절감할 수 있을 것이며, 여러분은 그 작업을 잘 수행하여 얻은 만족감을 느긋하게 즐길 수 있을 것이라 확신한다.

켄 제이콥스(Ken Jacobs)
오라클 기업 서버 테크놀로지 제품 전략 부사장

옮긴이 머리말

작년 여름이 막 시작될 무렵 회사로부터 메일 한 통을 받았다. 회사에 번역 제의가 들어왔는데 직원들이 참여할 의향이 있는지 묻는 것이었다. 메일을 받고 며칠 동안 많은 고민을 했다. 몇 년 전에 『비용기반 오라클』을 번역하면서 힘들어 하던 동료들의 모습이 떠오르면서 만만치 않은 작업에 대한 두려움과 비록 직접 쓴 책은 아니지만 좋은 책을 번역할 수 있는 기회를 놓칠 수 없다는 생각 때문이었다. 마무리 시점이라 그럴까? 지금 생각해보니 참여하길 잘한 것 같다. 나름대로 오라클에 대해 많은 것을 알고 있다고 생각했는데 이 책을 번역하면서 새롭게 알게 된 것도 많고, 기본에 충실하다는 것이 얼마나 중요한지도 다시 한 번 깨닫는 계기가 되었다.

이 책은 오라클 데이터베이스의 구조, 파일, 메모리, 락킹, 테이블, 인덱스, 병렬 처리 등 데이터베이스 구조와 오브젝트에 대한 기본 및 활용에 대한 내용을 다루고 있다. 새로운 개념이라기보다는 오라클 매뉴얼에도 나와 있는 기본적인 내용에 충실하다고 볼 수도 있다. 그리고 여기에 더해 언제 어떻게 사용해야 좋은지에 대해 설명하고 있다. 개발자가 보기에 어떤 부분은 SQL 작성과 직접적인 관련이 없다고 느낄 수 있고, DBA는 다 아는 내용이라고 생각할 수도 있다. 우리나라는 응용과학 수준은 높은 데 기초과학은 아직 낮다는 말을 많이 들었을 것이다. 언뜻 보기에 별로 상관없는 듯하지만, 기초가 약하면 복잡해지고 세월이 흐를수록 한계를 절실히 느끼기 마련이다. 기본적인 개념과 활용 방향에 대해 이해하고 나면 다른 부분도 이해하기 쉽고 어려운 문제도 해결할 수 있는 능력이 생길 것이다.

여러 프로젝트를 하면서 느끼는 점은 예전에 비해 훨씬 많은 개발자들이 오라클에 대해 비교적 잘 알고 있으며 새로운 기능을 활용한다는 것이다. 그러나 아직도 매뉴얼이나 전문 서적을 통한 기술 습득이 아닌 선배나 누군가로부터 얻은 지식을 활용하는 경우를 많이 볼 수 있다. 개발자가 테이블이나 인덱스에 대해 충분히 알고 있다는 얘기를 듣고 나름대로 SQL을 잘 작성할 것이라고 생각했다. 그러나 튜닝을 하다 보면 "이렇게 하면 인덱스 타는 거 아닌가요?"라는 질문을 받는 경우가 종종 있다. 인덱스를 사용

하는 문제가 아니라 어떻게 사용하느냐에 대한 고민은 없었던 듯하다. 기본을 알고 접근한다는 것이 얼마나 중요한지를 다시 한 번 느끼게 된다.

원서를 옮기면서 번역이라는 특성과 지식 전달 사이에서 용어에 대한 고민을 많이 했다. 가급적 한글로 번역하되 의미 전달이 어려운 부분은 많이 사용하는 용어를 선택했으며, 내용이 어려운 부분은 풀어 쓰거나 부연 설명을 추가했다.

그동안 번역작업을 하면서 많은 지원을 아끼지 않으신 (주)비투엔컨설팅 조광원 사장님과 정광용 팀장님, 조시형 수석님, 임성민 과장님께 감사드리고, 꼼꼼한 검토와 출판에 정성을 기울여 주신 제이펍 출판사 장성두 실장님께 깊은 감사의 마음을 전한다.

끝으로 옮긴이와 출판사가 여러 번 검토하긴 했으나 저자의 의도와 다르거나 문맥이 어색한 부분이 있으리라 생각된다. 앞으로 계속 수정해 나갈 것이며, 틀린 부분이나 바로 잡아야 할 내용이 있으면 바로 연락 주기를 바란다. 아무쪼록 이 책을 읽는 여러분께 많은 도움이 되길 바란다.

2011년 6월
옮긴이 일동

옮긴이 소개

■ 오선경

세종대학교 컴퓨터공학과를 졸업하였고, (주)MPC, 코리안클릭을 거쳐 현재 (주)비투엔컨설팅에서 수석 컨설턴트로 재직 중이다. 교보생명, 대교 직판시스템, 인터넷 사용자 분석시스템, 인터파크, CJ오쇼핑, 삼성화재 정보계 프로젝트를 진행하였고, 데이터베이스 성능개선, 데이터 이행, DW구축, 데이터 모델링 및 데이터 아키텍처 구축컨설팅 업무를 진행하고 있다.

■ 한준희

한국교원대학교 수학교육과를 졸업하였고, KCC정보통신, (주)엔코아와 보건복지부 보건의료정보사업추진단 정보지식화 팀장을 거쳐 현재 (주)비투엔컨설팅에서 수석 컨설턴트로 재직 중이다. 2006년부터 (주)엔코아에서 대용량 데이터베이스 솔루션, 튜닝 워크샵과 한국 데이터베이스 진흥원에서 데이터 아키텍처 및 데이터베이스 튜닝 전문가 과정을 담당하였다. 고속철도 통합정보시스템, 한국교육학술정보원(KERIS), 금융감독원, 국방 표준 데이터관리체계 구축 등 다수의 사업에서 데이터 아키텍처 구축컨설팅과 데이터베이스 성능개선 사업 및 데이터 표준화 컨설팅 사업을 수행하였다.

■ 유동오

한양대학교 글로벌MBA를 졸업하였고, (주)엔코아컨설팅을 거쳐 현재 (주)비투엔컨설팅에서 수석 컨설턴트로 재직 중이다. 한국산업기술평가원, 우리은행, 한국고용정보원 등 여러 고객사에서 데이터모델링, 데이터베이스 성능개선 등 데이터 컨설팅을 수행했으며, 현재는 삼성화재 퇴직연금시스템 데이터 모델링을 진행 중이다.

■ 임영섭

서강대학교 컴퓨터공학과 대학원을 졸업하였고, 삼성전자를 거쳐 현재 (주)비투엔컨설팅에서 책임 컨설턴트로 재직 중이다. 한국과학기술정보연구원, 한국수출보험공사, 삼성화재, 보건복지부, 한국국방연구원, 한국고용정보원 등 여러 고객사에서 ISP, 차세대 시스템 구축, 데이터베이스 성능개선 및 데이터 표준화 컨설팅 업무를 수행하였다.

■ 김창배

서강대 정보통신대학원을 졸업하였고, (주)비투엔컨설팅에서 선임 컨설턴트로 재직 중이다. KTF, 우리은행, SK 건설, BC 카드, 삼성화재 등 여러 고객사에서 데이터베이스 성능개선, DW구축, DB설계, 데이터 표준화 등과 같은 시스템 개발 및 컨설팅 업무를 수행하였다.

저자 소개

■ **토마스 카이트**(Thomas Kyte)

필자는 오라클 7.0.9 버전(오라클 데이터베이스 6 버전 이후로 새로운 버전이 출시된 1993년)부터 오라클에서 일해 왔지만, 오라클 5.1.5c 버전 때부터 오라클과 함께 작업을 해왔다(360KB 플로피 디스크의 단일 사용자 DOS 버전이 99$였다). 오라클 근무 전에는 6년 동안 주로 군대와 정부 고객을 대상으로 이기종 데이터베이스와 애플리케이션을 구축하는 시스템 통합 업무를 수행하였다. 요즘은 오라클 데이터베이스를 사용하는 사람들을 지원하면서 데이터베이스를 다루는 작업에 대부분의 시간을 보내고 있다.

고객들의 시스템을 설계하고 시스템을 구축하기도 하지만, 그보다는 더 자주 고객이 시스템을 재구축하거나 튜닝하는 일을 도와준다('튜닝' 은 데이터베이스를 재구축하는 의미로 자주 사용된다). 또한 『오라클 매거진』에서 'Ask Tom' 코너를 맡고 있으며, http://asktom.oracle.com 사이트에서는 오라클 데이터베이스와 툴에 관해 매일 수십여 통의 질문에 답해주고 있다. 두 달마다 한 번씩 매거진에 가장 좋은 질문을 싣고 있으며(물론 웹에서 언제든지 볼 수 있으며, 내용은 오라클 데이터베이스에 저장된다), 또한 이 책에서 볼 수 있는 대부분의 소재를 다루는 기술 세미나를 하고 있다. 필자는 다른 사람들이 오라클 데이터베이스를 성공적으로 사용하는 데 도움이 되도록 많은 시간을 보내고 있으며, 오라클 기업 내에서도 애플리케이션을 구축하고 소프트웨어를 개발하고 있다.

이 책은 필자가 매일 하고 있는 일을 담은 책이다. 다시 말해서 사람들이 매일 겪고 있는 문제와 자주 물어보는 질문 등을 모두 다루고 있다. "필자가 오라클 데이터베이스를 사용한다면 이런 방식으로 한다"는 마음가짐으로 이런 이슈를 다루었다. 이 이슈들은 갖가지 상황에서 데이터베이스를 사용하면서 필자가 겪었던 다양한 경험에서 우러나온 것들이다.

기술 검토자 소개

■ **크리스토퍼 벡**(Christopher Beck)은 러트거스 뉴저지주립대학교에서 컴퓨터 공학 학위를 받았으며, 20년 가까이 다수의 DBMS를 다루어 왔다. 현재 핵심 데이터베이스 기술을 담당하는 최고 책임자로서 오라클에 15년 이상 몸담아 왔다. 또한 그는 Oracle Application Express 제품을 개발하는 데 사용된 소프트웨어 개발방법론에 관한 두 가지 US 특허의 공동 발명자기도 하다. 크리스는 톰의 첫 번째 책 『*Expert One-On-One*』을 포함한 다른 오라클 책을 리뷰했으며, 『*Beginning Oracle Programming*』과 『*Mastering Oracle PL/SQL*』의 공동 저자기도 하다. 그는 아내 마르타와 네 자녀와 함께 버지니아 북부에 거주하고 있으며, 가족과 함께 있지 않을 때는 보통 비디오 게임을 즐기거나 미식축구 경기를 시청한다.

■ **멜라니 캐프리**(Melanie Caffrey)는 다양한 고객의 비즈니스 요구사항을 해결하는 프런트엔드와 백엔드 오라클 솔루션을 제공하는 오라클 기업의 선임 개발 관리자다. 그녀는 『*Expert Oracle Practices: Oracle Database Administration from the Oak Table*』뿐만 아니라 『*Oracle Web Application Programming for PL/SQL Developers*』, 『*The Oracle DBA Interactive Workbook*』, 그리고 『*Oracle Database Administration: The Complete Video Course*』 등 여러 기술 서적의 공동 저자다. 그녀는 뉴욕시에 소재한 콜롬비아 대학에서 운영하는 컴퓨터 기술과 애플리케이션 프로그램을 통해 오라클 데이터베이스 관리와 PL/SQL 개발을 가르쳤다. 오라클 컨퍼런스 강연자로서도 자주 활동한다.

■ **제이슨 스트라우브**(Jason Straub)는 수학 학위를 갖고 있으며, 수학을 컴퓨터 공학 분야에 적용해 데이터베이스를 사용하는 웹 애플리케이션을 15년 동안 개발해왔다. 오라클에서 근무하고 있으며, 그 전에는 마이크로소프트에서 일했다. 그는 오라클 버전 8.0.6부터 오라클 데이터베이스 애플리케이션을 개발해왔고, 마이크로소프트에서는 SQL 서버 2000부터 다루었다. 현재 나날이 인기를 더해가는 웹 개발 툴인 Oracle Application Express 제품에서 웹 서비스 기능을 구축하는 일을 맡고 있는 주요 개발자다. 또한 오라클 기술 네트워크에서 언제든지 구할 수 있는 몇 가지 백서의 저자이기도 하며, ODTUG Kaleidoscope, RMOUG와 오라클 Open World 같은 오라클 기술 컨퍼런스에서 자주 등장하는 발표자기도 하다.

감사의 글

이 책을 완성하기까지 도와준 많은 사람들에게 감사의 인사를 전하고 싶다.

먼저 독자 여러분께 감사드린다. 이 책을 읽고 있는 독자라면 어떤 방식으로든 방문해 한두 가지 질문을 남겼으리라 짐작된다. 질문을 받고 답변하는 과정이 이 책을 집필하게 해준 소재도 되고 소재의 배경지식이 되었다. 그런 질문이 없었다면 오라클 데이터베이스에 관해 필자는 현재와 같은 식견을 갖지 못했을 것이다. 따라서 이 책을 있게 해준 독자 여러분께 감사의 말을 전한다.

필자의 글을 잘 읽을 수 있도록 글을 다듬어 준 토니 데이비스(Tony Davis)에게 감사한다. 절(sections) 간의 부드러운 연결과 단락 개수, 명료성은 일정 부분 그의 노고가 있었기 때문에 가능한 일이었다. 필자는 2000년부터 토니와 함께 기술 서적을 집필해 왔고, 그 동안 오라클에 관한 토니의 지식도 함께 성장해 온 것을 지켜봐 왔다. 이제 그는 내용 편집을 넘어 기술 편집도 가능한 수준이다. 이 책에서 든 풍부한 예제는 평범한 독자의 눈높이로 지적해준 토니가 있었기 때문에 가능하였다. 토니는 이 책의 개정판에는 참여하지 않았어도 내용은 대개 초판에 의거하기 때문에 그가 없었다면 이 책은 나올 수 없었을 것이다.

이 책과 이전 판에 대한 기술 리뷰를 담당해 준 유능한 팀에게도 감사의 인사를 전한다. 초판에서 기술 리뷰는 조나단 루이스(Jonathan Lewis), 로더릭 마날락(Roderick Manalac), 마이클 뫼러(Michael Möller), 게이브 로마네스크(Gabe Romanescu)가 담당하였고, 이들은 자료를 열심히 연구하고, 기술적으로 정확하며, 실제로도 유용한지를 검증하는 데 많은 시간을 할애했다. 이번 두 번째 개정판에도 역량 있는 팀원들의 리뷰가 있었는데, 멜라니 카프리(Melanie Caffrey), 크리스토퍼 벡(Christopher Beck), 제이슨 스트라우브(Jason Straub)가 그들이다. 기술서적은 저자뿐만 아니라 기술 검토자에 의해서도 판가름 난다고 믿어 의심치 않는다. 이들 일곱 명의 검토자들이 있었기 때문에 내용에 확신을 가지게 되었다.

오라클에서 필자는 가장 명석한 동료들과 함께 일하고 그들 모두 나름대로 필자에게 도움을 주었지만, 특별히 켄 제이콥스(Ken Jacobs)의 수년에 걸친 지원과 열의에 감사를 전한다. 아쉽게도 켄은 더 이상 오라클에서 일하고 있지 않지만 그의 영향은 오래도록 지속될 것이다.

마지막으로 끊임없이 도와 준 가족에게 많은 감사를 전한다. 지속적으로 지원을 아끼지 않으며 기술 검토를 도맡아준 나의 아내 멜라니(Melanie), 그리고 아들 알란(Alan)과 딸 메건(Megan)이 있었기에 이 책을 마칠 수 있었음을 전하고 싶다.

이 책에 대하여

이 책의 주제에 관한 영감은 오라클 소프트웨어 개발 경험과 오라클 데이터베이스 기반에서 신뢰할 만하고 안정적인 애플리케이션을 구축하려는 오라클 동료 개발자들을 돕는 작업으로부터 시작되었다. 이 책은 기본적으로 필자가 매일 수행하는 작업의 내용이자 당면하는 실질적인 이슈들을 반영한 것이다.

이 책에서는 필자가 가장 중요하다고 생각하는 오라클 데이터베이스와 아키텍처에 관하여 다루었다. 필자는 EJB(Enterprise JavaBeans)를 호출하고 오라클과 통신하기 위해 JDBC를 사용하는 JSP(JavaServer Pages)와 같은 특정 프로그래밍 언어와 구조 기반의 애플리케이션 개발 기법에 관한 책을 쓸 수 있지만, 궁극적으로 애플리케이션을 성공적으로 개발하기 위해 정말로 필요한 것은 이 책에서 다루고 있는 주제라고 생각한다. 이 책은 독자가 ODBC를 이용하는 Visual Basic 프로그래머이거나, EJB와 JDBC를 이용하는 Java 프로그래머이거나, DBI Perl을 이용하는 Perl 개발자이건 간에 오라클 환경에서 성공적인 애플리케이션을 개발하기 위해서 전반적으로 알아야 할 내용이라고 생각되는 주제를 다루고 있다. 이 책은 특정 애플리케이션이나 3-티어(three tier) 또는 클라이언트/서버 환경 등과 같은 특정한 구조의 장단점을 비교하지는 않는다. 단지 데이터베이스가 할 수 있는 것과 데이터베이스의 작동방법에 대해 반드시 이해해야 하는 것을 다루고 있다. 데이터베이스는 애플리케이션 구조의 중심이기 때문에 다양한 독자에게 도움을 줄 수 있을 것이다.

이 책의 제목과 같이 데이터베이스 구조와 데이터베이스가 어떻게 일하는지를 살펴보고, 오라클 데이터베이스와 인스턴스를 구성하는 파일, 메모리 구조, 프로세스에 관해 자세히 다루었다. 그리고 중요한 주제인 락킹, 동시성 제어, 트랜잭션의 이해, 리두와 언두에 관하여 살펴보고, 왜 이것들이 중요한가에 대해 언급하였다. 마지막으로, 물리적 구조 이해를 위해 테이블, 인덱스, 데이터타입에 관하여 살펴보았고, 이것들을 가장 적절하게 사용하는 기술을 알아보았다.

이 책의 주제는 무엇인가?

다양한 개발 방법이 존재할 때의 문제는 당면한 문제를 해결하기 위한 최선의 선택이 무엇인지를 결정하기 어렵다는 것이다. 사용자는 다양하고 많은 선택이 가능한 유연성을 원하지만, 또한 쉽게 선택할 수 있기를 원한다. 오라클은 개발자에게 매우 다양한 방법들을 제공한다. 즉, 누구도 "오라클에서 할 수 없습니다"라고 말하지 않는다. 단지 "오라클로 구현할 수 있는 방법이 얼마나 다양합니까?"라고 묻는다. 이 책이 올바른 개발 방법을 선택하는 데 도움이 되기를 바란다.

이 책의 목적은 최선의 선택을 해야 할 뿐 아니라 어떤 지침이 필요하고 실제로 구현 가능한 오라클 요소와 기능에 대해 세부적으로 알아야 하는 사용자를 위한 책이다. 예를 들어, 오라클은 실제로 병렬 처

리(parallel execution)라 불리는 적합한 기능이 있는데, 오라클 도큐먼트는 이 요소를 어떻게 사용하는지, 그리고 무엇을 할 수 있는지에 대해 설명하고 있다. 그러나 더 중요한 것을 말하지는 않는다. 즉, 언제 사용해야 하고, 언제 사용하지 않아야 하는지에 대해서는 설명하지 않고 있다. 그리고 구체적인 구현 방법도 설명하지 않으며, 주의하지 않으면 어떤 결과가 도출되는지도 설명하지 않고 있다(필자는 버그를 언급하지 않을 것이다. 그러나 기능들이 수행되는 방법과 무엇을 위해 설계되었는지는 설명할 것이다).

이 책에서는 어떻게 수행하는지를 설명할 뿐 아니라 특정한 기능을 언제, 왜 사용해야 하는지를 설명하고 있다. '어떻게' 수행하는지를 이해하는 것뿐 아니라 '언제' 그리고 '왜' 사용해야 하는지, 그리고 언제 사용하지 않아야 하고 왜 사용하지 않아야 하는지를 이해하는 것은 매우 중요한 문제다.

이 책은 누가 읽어야 할까?

이 책의 주요 독자는 오라클 기반에서 애플리케이션을 개발하려는 모든 사용자를 대상으로 할 뿐 아니라 오라클 세부 지원 기능에 대한 이해가 필요한 전문 오라클 개발자들을 대상으로 한다. 또한 이 책의 실제적인 원리는 DBA에게도 주요 관심사가 될 것이다. 책에서 설명하고 있는 대부분의 예제는 핵심사항을 설명하기 위해 SQL*Plus를 사용하고 있으며, GUI 툴에서는 어떻게 개발하는지 설명하고 있지 않지만, 오라클 데이터베이스의 동작 방법과 주요 기능을 알게 되고, 언제 사용해야 하는지 알게 될 것이다.

이 책은 오라클 데이터베이스를 효과적으로 사용하기를 원하는 모든 독자를 위한 책이다. 존재하던 기능을 새로운 방법으로 구현하기를 원하거나, 오라클의 다양한 기능에 대한 실제 적용방법을 습득하려는 독자를 위한 책이다(단지 사용법만을 설명하지 않고 왜 이 기능이 유용한가를 제일 먼저 설명하고 있다). 이 책을 읽어야 하는 또 다른 독자층은 오라클 기반 개발자들의 기술적 관리자다. 어떤 면에서 그들은 프로젝트를 성공적으로 수행하기 위해 데이터베이스의 어떤 기능이 필요한가를 아는 것이 더 중요하기 때문이다. 팀원들의 올바른 기술 습득을 독려하거나 알아야 하는 기술을 이미 알고 있는 팀원을 격려해야 하는 관리자에게 이 책은 유용할 것이다.

이 책의 내용을 더 많이 이해하기 위해 독자는 다음과 같은 사항을 알고 있어야 한다.

- SQL에 대한 이해 : SQL 고급 개발자일 필요는 없지만, SQL에 대한 기본지식이 있으면 이 책을 더욱 쉽게 이해할 것이다.
- PL/SQL에 대한 이해 : 반드시 필요한 요구사항은 아니지만, 이 책의 예제를 완전하게 이해하는 데 도움이 될 것이다. 예를 들어, 이 책은 FOR 루프를 작성하는 방법이나 record type을 정의하는 방법을 설명하지 않는다. 이러한 내용은 오라클 도큐먼트 문서와 수많은 책에서 소개하고 있다. 그렇지만 이 책을 통해서도 PL/SQL에 관한 많은 부분을 공부할 수 있을 것이다. PL/SQL의 수많은

특징을 알게 될 것이고, 새롭게 접근하는 방법을 알게 될 것이며, 전에 알지 못하던 패키지/특징 등에 관하여 많이 알게 될 것이다.

- C 또는 Java와 같은 3세대 프로그램 언어에 대한 이해 : 3세대 언어로 프로그래밍을 해본 독자라면 이 책의 예제를 읽고 이해하는 데 전혀 문제가 없을 것이다.
- 오라클 개념 매뉴얼에 대한 숙지

끝으로 언급할 것은, 방대한 오라클 도큐먼트 문서에 부담을 느끼는 독자는 가장 먼저는『*Oracle Concepts*』매뉴얼을 보기 바란다. 이 문서는 데이터베이스를 시작하는 독자에게 가장 적합한 문서며, 약 400페이지 분량이다(필자는 그 중에 일정 분량을 직접 작성했고 대부분을 검토했기 때문에 알고 있다). 독자가 알아야 할 오라클의 주요한 개념에 대해 많은 부분을 설명하고 있다. 각각에 대한 기술적인 세부사항은 설명하지 않고 있지만(이는 10,000에서 20,000페이지에 달하는 도큐먼트 문서에 기술되어 있다), 주요한 개념에 대해서는 충분히 이해할 수 있을 것이다. 이 매뉴얼은 다음과 같은 주제를 언급하고 있다.

- 데이터베이스 구조와 데이터를 구조화하고 저장하는 방법
- 분산 처리
- 오라클 메모리 구조
- 오라클 프로세스 구조
- 스키마 객체(테이블, 인덱스, 클러스터 등)
- 내장형(built-in) 데이터타입과 사용자 정의 데이터타입
- SQL 저장 프로시저
- 트랜잭션의 처리방법
- 옵티마이저
- 데이터 무결성
- 동시성 제어

위의 내용은 오라클의 기본 개념이기 때문에 이 책에서도 반복적으로 계속해서 언급할 것이다. 위의 개념에 대한 이해가 없다면, 잘못된 오라클 애플리케이션을 만들 수밖에 없을 것이다. 그러므로 오라클 매뉴얼 읽기를 권하며, 이러한 주제의 일부라도 이해하기를 바란다.

이 책은 어떻게 구성되어 있는가?

다음의 목록을 통해 설명하겠지만, 이 책은 일반적으로 4개 절로 나누어 구성되어 있다. 4개 절을 정확

하게 나누지는 않지만, 각 절은 이 책의 정보를 더 빠르게 찾을 수 있도록 도와줄 것이다. 이 책은 16개 장으로 구성되어 있고, 각각은 하나의 독립적인 작은 책과 같이 구성되어 있다. 간혹 다른 장의 예제나 특징을 언급하기도 하지만, 한 장 내에서 대부분을 설명하고 있기 때문에 다른 장을 참조할 필요는 없다. 예를 들어, 14장 병렬 처리를 이해하기 위해 10장 데이터베이스 테이블을 읽을 필요는 없다는 것이다.

각각의 구성과 스타일은 다음과 같이 정리되어 있다.

- 주요 기능과 역할 소개
- 기능과 역할의 필요성 및 언제 이 기능을 사용해야 하고 언제 사용하지 않아야 하는지에 대한 소개
- 기능을 사용하는 방법. 단지 주제와 관련된 SQL을 참조하는 방식이 아니라 어떤 것이 필요하고 무엇을 해야 하는지, 그리고 진행과 중지를 위한 방법에 대해 단계적으로 설명하고 있다. 주제를 설명하기 위해 다음과 같은 내용을 포함한다.
  - 기능이 어떻게 구현되는지에 관한 설명
  - 다양한 예제
  - 기능의 실행을 추적하는 방법
  - 기능을 사용했을 때의 주의사항
  - 예방차원에서의 오류를 처리하는 방법
- 전체 정리

이 책에서 다루는 다양한 예제와 코드는 Apress 출판사(www.apress.com)나 제이펍 출판사(www.jpub.kr)에서 다운로드가 가능하다. 다음 절에서 각 장의 내용을 조금 더 자세히 설명하고 있다.

1장: 성공적인 오라클 애플리케이션 개발

이 장에서는 데이터베이스 프로그래밍에서 필요한 주요 접근 방법을 설명하고 있다. 모든 데이터베이스가 똑같지 않기 때문에, 데이터베이스를 이용하는 애플리케이션을 주어진 기간 안에 성공적으로 개발하기 위해서는 사용하는 데이터베이스가 어떤 기능을 가졌는지 정확하게 이해해야 하며, 또한 어떻게 수행하는지 알고 있어야 한다. 데이터베이스가 무엇을 할 수 있는지 알지 못한다면, 데이터베이스가 이미 제공하는 기능을 애플리케이션으로 개발하기 위해 쓸데없이 시간을 낭비할 수 있다. 또한 데이터베이스가 어떻게 수행하는지 알지 못한다면, 오류를 발생시키거나 제대로 동작하지 않는 애플리케이션을 개발할 수 있다.

이 장에서는 데이터베이스에 대한 이해부족으로 말미암은 프로젝트 실패 사례에서 볼 수 있는 몇몇 애

플리케이션을 살펴볼 것이다. 예제를 살펴보면서 개발자로서 이해할 필요가 있는 기본적인 특징과 기능에 대해 알아볼 것이다. 결론적으로 말하면, 데이터베이스를 단지 결과를 뱉어내는 블랙박스처럼 다루지 말아야 하며, 확장성과 성능에 관하여 관심을 기울여야 한다.

2장 : 아키텍처 개요

이 장에서는 오라클 아키텍처의 기본 개념을 알아본다. 먼저 오라클 데이터베이스에서 자주 잘못 이해되고 있는 '인스턴스'와 '데이터베이스' 두 가지 용어에 대해 정확하게 정의한다. 또한 오라클 인스턴스를 구성하는 SGA(System Global Area)와 프로세스에 대해 빠르게 살펴보고 있으며, 간단한 오라클의 접속 방식에 대해서도 살펴본다.

3장 : 파일

이 장에서는 오라클 데이터베이스와 인스턴스를 구성하는 8가지 파일 유형을 자세히 살펴본다. 파라미터 파일부터 데이터 파일, 리두 로그 파일까지 각 파일이 어떤 파일인지 살펴보고, 존재하는 이유와 어떻게 다루어야 하는지에 대해 살펴본다.

4장 : 메모리 구조

이 장에서는 오라클이 메모리를 어떻게 사용하는지를 살펴본다. 사용자 레벨에서 사용하는 PGA(Process Global Area) 메모리와 데이터베이스 전체적으로 사용하는 SGA(shared memory)에 대해 알아본다. 오라클 10*g*에서 PGA 및 SGA를 수동으로 관리하는 방법과 자동으로 관리하는 방식 간의 차이점을 살펴보고, 각각이 어느 경우에 유리한지 살펴본다. 이 장을 읽고 나면 오라클이 메모리를 어떻게 사용하고 관리하는지 정확하게 이해할 수 있을 것이다.

5장 : 오라클 프로세스

이 장은 오라클의 서버 프로세스와 백그라운드 프로세스의 유형에 대해 전반적으로 살펴본다. 또한 shared server 프로세스를 통한 접속 방식과 dedicated server 프로세스를 통한 접속 방식 간의 차이점을 자세히 설명하고 있다. 또한 오라클 인스턴스가 구동할 때 생성되는 백그라운드 프로세스(LGWR, DBWR, PMON, SMON 등)의 대부분을 살펴보고 있으며, 각각의 역할에 대해 언급하고 있다.

6장 : 락킹과 래칭

데이터베이스는 데이터베이스만의 고유한 방식을 가지고 있기 때문에(SQL Server는 오라클과 같은 방식으로 수행하지 않는다), 오라클이 락킹과 동시성 제어를 어떻게 구현하는지 이해하는 것은 성공적인 애플리케이션 구현을 위해서 반드시 필요한 주제다. 이 장에서는 이러한 주제에 접근하는 기본적인 접근 방법을 설명하고 있고, DML, DDL, 래치에 따라 적용될 수 있는 락의 유형에 대해 알아보고, 락킹을 주

의 깊게 구현하지 않았을 때에 발생할 수 있는 데드락킹(deadlocking), 블로킹(blocking), 락 상승(escalation) 등과 같은 문제점에 대해 살펴본다.

7장: 동시성과 멀티버저닝

이 장에서는 필자가 선호하는 오라클의 특징인 멀티버저닝과 동시성 제어를 살펴보고, 애플리케이션의 올바른 설계에 어떠한 영향을 미치는지 알아볼 것이다. 모든 데이터베이스가 동일하게 생성되지 않았기 때문에 데이터베이스마다의 작동 방법은 애플리케이션 설계에 영향을 미친다. 먼저 ANSI SQL에 기본적으로 정의되는 다양한 트랜잭션 고립화 수준에 대해 살펴보고, 오라클 구현 방식과 어떻게 연결되는지 알아본다. 또한 다른 데이터베이스에서도 이 표준과 어떻게 대응되는지 살펴본다. 그리고 오라클이 제공하는 멀티버저닝이 무엇인지 알아보고, 이 기능으로 인해 데이터베이스에서 non-blockng 읽기가 어떻게 가능한지 알아볼 것이다.

8장: 트랜잭션

트랜잭션은 모든 데이터베이스의 기본 개념으로 파일 시스템과 구별되는 데이터베이스의 특징 중 하나지만, 아직도 트랜잭션을 잘못 이해하고 있는 것을 종종 볼 수 있으며, 많은 개발자가 실수로 트랜잭션을 이용하지 않고 있다는 것조차 모르는 경우도 있다. 이 장에서는 오라클에서 트랜잭션을 어떻게 사용해야 하는지를 살펴보고, 다른 데이터베이스에서 사용하는 방식으로 잘못 사용하는 경우에 대해서도 알아볼 것이다. 특히 트랜잭션의 특징 중 하나인 원자성(atomicity)에 대해 알아보고, 오라클 명령문에 어떻게 영향을 미치는지도 살펴볼 것이다. 그리고 트랜잭션 제어 명령문인 COMMIT, SAVEPOINT, ROLLBACK에 대해서 알아보고, 무결성 제약조건(integrity constraints), 분산 트랜잭션(two-phase commit, 2PC)을 알아보고, 마지막으로 자율 트랜잭션에 대해 살펴볼 것이다.

9장: 리두와 언두

개발자들은 DBA만큼 리두와 언두에 대해 자세히 알 필요는 없지만, 개발자들도 데이터베이스에서 리두와 언두의 역할에 대해서는 알아야 한다. 제일 먼저 리두를 정의하고, COMMIT 명령문이 정확하게 무엇을 하는지 조사한다. 그리고 얼마만큼의 리두가 생성되는지를 확인하는 방법을 알아보고, 어떤 오퍼레이션에 의해 생성된 대량의 리두를 줄이기 위해 NOLOGGING 절을 사용하는 방법에 대해 살펴본다. 또한 블록 클린아웃(cleancout)과 로그 경합과 같은 이슈와 관련된 리두 생성에 관하여 알아본다.

언두 절에서는 언두 데이터의 필요성을 알아보고, 언두 데이터를 최대로 발생시키는 오퍼레이션과 최소로 발생시키는 오퍼레이션을 알아본다. 마지막으로, ORA-01555: snapshot too old error가 어떤 경우에 발생하는지 알아보고 피할 수 있는 방법을 살펴본다.

10장: 데이터베이스 테이블

오라클은 다양한 테이블 유형을 지원하고 있다. 이 장에서는 힙 구조 테이블(기본으로 생성되는 일반적인 테이블), 인덱스 구조 테이블, 인덱스 클러스터 테이블, 해시 클러스터 테이블, 중첩 테이블, 템포러리 테이블, 객체 테이블의 다양한 테이블 유형을 알아본다, 그리고 이 테이블이 언제 사용하는 것이 유리한가, 어떻게 사용해야 하는가, 왜 사용하는가를 알아볼 것이다. 대부분은 힙 구조 테이블로 구현해도 문제가 없지만, 이 장을 통해 다른 테이블 유형이 어떤 환경에서는 더 유리한 구조라는 것을 알게 될 것이다.

11장 : 인덱스

인덱스는 애플리케이션 설계에 있어서 중요한 역할을 한다. 올바른 구현을 위해서는 데이터를 깊이 이해할 필요가 있고, 데이터를 어떻게 분산하는지, 어떻게 사용할 것인지에 대한 이해가 필요하다. 인덱스는 애플리케이션의 최종 마무리 단계에서 다뤄지는 경우가 대부분이어서 성능에 문제를 겪게 된다.

이 장에서는 B*Tree, 비트맵, 함수 기반과 애플리케이션 인덱스에 관한 다양한 인덱스 유형을 자세히 살펴보고, 인덱스 유형별로 어떤 경우에 사용해야 하고 사용하지 않아야 하는지 살펴본다. '뷰에서도 인덱스가 사용되는가?', '해당 인덱스를 왜 사용하지 않을까?' 등과 같은 인덱스에 관한 잦은 질문에 대해 살펴보고, 몇 가지 오해에 대해서도 알아본다.

12장 : 데이터타입

오라클에는 다양한 데이터타입이 존재한다. 이 장에서는 22개 내장형 데이터타입에 대해 살펴보고, 데이터타입별로 어떤 특징이 있는지 알아보고, 언제 어떻게 어떤 데이터타입을 선택해야 하는지 살펴본다. 먼저 NLS(National Language Support)에 대해 간단하게 살펴보고, 오라클에서 문자 타입을 충분히 이해하기 위한 기본 지식을 자세히 살펴본다. 그리고 익숙한 NUMBER 타입에 대해 살펴보고, 오라클 10*g* 버전에서 제공하는 숫자의 저장방식에 대해서도 살펴본다. 다음으로, 이제는 잘 사용하지 않는 LONG과 LONG RAW 데이터타입에 대해 알아본다. 애플리케이션에서 과거의 LONG 컬럼을 어떻게 다뤄야 할지 보여주고, LOB 타입으로 이전하는 방법에 대해 설명한다. 다음으로, 일자와 시간을 다루는 다양한 데이터타입을 살펴보고, 원하는 결과를 얻기 위해 다양한 데이터타입을 다루는 방법에 대해서도 살펴보고 있으며, 타임 존(time zone)을 지원하는 방식에 대해서도 다루고 있다.

또한 LOB 데이터타입을 살펴보고 있으며, 어떻게 저장하는지 알아보고 IN ROW, CHUNK, RETENTION, CACHE와 같은 다양한 설정의 의미에 대해 알아본다. LOB 데이터타입을 다루기 위해 기본적으로 어떻게 구현되고 어떻게 저장되는지를 이해하는 것은 중요하며, 특히 추출결과와 저장공간의 활용방식을 이해하는 것이 중요하다. 마지막으로 ROWID와 UROWID 타입을 살펴보며, 이는 오라클에 국한된 특별한 데이터타입으로서 로우의 주소를 의미한다. 테이블에서 데이터타입으로 언제 사용해야 하는지 알아보고, 결과적으로 사용하지 않아야 한다는 결론을 얻게 된다.

13장 : 파티셔닝

파티셔닝은 대용량 테이블과 인덱스를 분할정복(divide-and-conquer) 방식으로 관리하는 도구로서 고안되었고, 논리적인 단위로 테이블을 쪼개거나 관리할 수 있는 더 작은 조각으로 인덱스를 나누어 관리할 수 있게 한다. DBA와 개발자는 애플리케이션의 유용성과 성능 향상을 최대화하기 위해 협력해야 한다. 이 장에서는 테이블과 인덱스 파티셔닝에 대해 다루고 있으며, 데이터 웨어하우스에서 주로 사용하는 로컬 인덱스 파티셔닝과 주로 OLTP 환경에서 사용하는 글로벌 인덱스 파티셔닝에 대해 살펴본다.

14장 : 병렬 처리

이 장에서는 오라클에서 병렬 처리의 개념과 처리방식에 대해 살펴본다. 먼저 언제 병렬 처리방식이 유용한가에 대해 살펴보고, 언제 사용해야 하는지, 또한 언제 사용하지 않아야 하는지를 살펴본다. 그리고 병렬 처리 메커니즘을 살펴보고, 병렬 처리와 관련된 기능을 알아본다. 다음으로, 병렬 처리 방식으로 수정을 처리하는 병렬 DML(PDML)에 대해 살펴보고, PDML이 물리적으로 어떻게 구현되는가와 PDML에 관련된 일련의 제한이 존재하는 이유에 대해 살펴본다.

또한 병렬 DDL을 살펴보는데, 필자가 생각하기에 병렬 DDL이야말로 병렬 처리의 백미라고 생각한다. 일반적으로 DBA는 거대한 작업을 관리하는 작은 단위 작업을 가지고 있게 마련인데, 병렬 DDL은 DBA가 사용 가능한 최대한의 시스템 자원을 이용하여 거대하고 복잡한 작업을 단일 프로세스로 처리했을 때보다 극적으로 짧은 시간에 마무리할 수 있게 한다.

마지막으로 애플리케이션을 병렬로 실행하는 방식으로 이해되는 프로시저 병렬 처리 방식에 대해 설명하며, 이를 구현하기 위한 두 가지 방식을 다룬다. 하나는 파이프라인 함수 또는 다이내믹 병렬을 구현하는 스토어드 함수로 구현하는 방식이고, 또 하나는 애플리케이션을 동시에 수행하도록 하는 'do it yourself(DIY)' 병렬 처리 방식이다.

15장 : 데이터 로딩과 언로딩

전반부는 SQL*Loader(SQLLDR)를 설명하고, 이 툴을 이용하여 데이터를 로드하고 수정하는 다양한 방식에 대해 알아본다. 구분자를 가진 데이터를 로딩하고, 기존에 존재하는 데이터를 수정하고, 새로운 로우를 삽입하고, 데이터를 언로딩하고, 저장 프로시저에서 SQLLDR을 호출하는 방식에 대해 살펴본다. SQLLDR은 매우 훌륭하고 유용한 툴이지만, 실제 사용해보면 많은 질문이 생기는 툴이기도 하다. 후반부는 External 테이블을 집중적으로 살펴보며, 벌크 데이터를 로드하고 언로드하는 데는 의심의 여지가 없는 가장 강력한 수단이라는 것을 알 수 있을 것이다.

16장 : 데이터 암호화

이 장은 오라클 데이터베이스에서 데이터 암호화를 위한 방법을 살펴본다. DBMS_CRYPTO 패키지를

이용하여 수동으로 설정하는 'do it yourself' 암호화 방식으로 중요하게 다루지는 않는다. 오히려 패키지를 이용하는 암호화 방식을 왜 사용하지 않아야 하는지를 설명하고 있다. 이 장의 주요 관심사는 오라클 데이터베이스에서 TDE(Transparent Data Encryption) 방식에 대해 자세히 살펴보고 있다. 컬럼 수준과 테이블스페이스 수준에서 어떻게 암호화를 구현하는지 알아보고, 개발자 또는 DBA에게 어떤 의미가 있는지를 설명한다. 가능한 모든 설정 방법을 설명하지는 않고(이것은 오라클 도큐먼트를 참조하면 된다), 실제적인 구현을 위한 세부사항을 살펴보고 어떻게 사용자에게 영향을 주는지를 알아본다.

소스 코드와 업데이트

이 책에 있는 예제를 테스트할 때는 독자가 모든 코드를 직접 타이핑하여 작성해보기를 추천한다. 상당수의 독자들은 이렇게 하는 것이 관련된 기술을 사용하고 친숙하게 되는 좋은 방법이라고 생각하기 때문이다.

코드를 타이핑하거나 그렇지 않든 간에, 이 책의 모든 소스 코드는 Apress 출판사(www.apress.com)의 원서 소개페이지에서 다운로드할 수 있다. 제이펍 출판사(www.jpub.kr)나 (주)비투엔컨설팅(www.b2en.com)의 이 책의 소개페이지에서도 다운로드가 가능하다. 타이핑하려는 독자라면 다운로드받은 소스 코드를 실행하여 결과가 같은지 확인하는 데 사용할 수 있으며, 타이핑하지 않으려는 독자라면 반드시 다운로드해야만 한다. 어쨌든 소스 코드 파일은 수정과 디버깅을 도와줄 것이다.

정오표 및 문의사항

Apress 출판사는 본문과 소스 코드에 어떠한 오류도 발생하지 않도록 노력했지만, 오류는 존재할 수 있다. 어떠한 실수를 발견하게 되거나 정정해야 할 것을 발견하게 되면 Apress 출판사(www.apress.com)의 원서 소개페이지에 존재하는 이 책의 정오표(Errata)를 살펴보고, 이미 알려진 오류가 아니라면 필자에게 알려주길 바란다. 제이펍 출판사(www.jpub.kr)나 (주)비투엔컨설팅(www.b2en.com)의 이 책의 소개페이지에서는 번역서에 대한 정오표 확인이 가능하다.

환경 설정

이 절에서는 이 책의 예제를 실행하는 데 필요한 환경을 설정하는 법을 다루고자 한다.

- 예제 스키마 SCOTT/TIGER 올바르게 설정하는 법
- 실행하고 호출하는 데 필요한 환경
- AUTOTRACE, SQL*Plus 기능 설정
- Statspack 설치
- Runstats와 이 책에서 사용된 다른 사용자 정의 유틸리티 설치 및 실행
- 이 책에서 사용한 코딩 규칙

SCOTT/TIGER 스키마 설정

SCOTT/TIGER 스키마는 데이터베이스에 이미 있고, 일반적으로 설치 시에 포함된다. 그러나 데이터베이스의 필수 구성요소는 아니다. SCOTT 계정 사용에 관한 제한은 없으므로 여러분도 데이터베이스 계정에 예제 스키마인 SCOTT를 설치할 수 있다. 또한 여러분의 데이터베이스 계정에도 직접 EMP/DEPT 테이블을 설치할 수 있다.

이 책의 많은 예제는 SCOTT 스키마의 테이블을 사용했다. 여러분이 필자처럼 수행하길 원한다면, 여러분은 이 테이블이 필요할 것이다. 공유되는 데이터베이스에서 수행한다면, 다른 사용자가 같은 데이터를 적재하는 문제를 피하기 위해 SCOTT 계정보다는 특정 계정에 여러분만의 테이블 복사본을 설치하는 것이 바람직하다.

스크립트 실행

SCOTT 예제 테이블을 생성하려면 다음과 같이 한다.

- cd [ORACLE_HOME]/sqlplus/demo

• 사용할 유저(어떠한 유저든 가능)로 접속하여 demobld.sql 실행

| Note | 오라클 10*g* 이상에서 컴패니언(Companion) CD로 예제 서브티렉토리를 설치해야만 한다. 필자 역시 아래의 demobld.sql의 필수 구성요소를 재설치했다.

---

demobld.sql은 5개의 테이블을 생성한다. 생성이 완료되었을 때 자동으로 SQL*Plus에서 빠져나가므로 스크립트 수행 후에 SQL*Plus가 사라졌더라도 놀라지 마라.

기본 데모 테이블은 참조 무결성이 정의되어 있지 않다. 그러나 필자의 특정 예제 테이블은 참조 무결성이 정의되어 있으므로 demobld.sql 수행 후에 아래 내용도 실행할 것을 추천한다.

```
alter table emp add constraint emp_pk primary key(empno);
alter table dept add constraint dept_pk primary key(deptno);
alter table emp add constraint emp_fk_dept
                                foreign key(deptno) references dept;
alter table emp add constraint emp_fk_emp foreign key(mgr) references emp;
```

이로써 SCOTT 스키마의 설치가 끝났다. 언제라도 이 스키마를 제거하길 원하면, 간단히 [ORACLE_HOME]/sqlplus/demo/demodrop.sql을 실행시키면 된다. 이 스크립트는 5개의 테이블을 삭제하고 SQL*Plus에서 빠져나간다.

스크립트 없이 스키마 생성하기

demobld.sql을 사용할 수 없다면, 이 책의 예제를 수행하기 위해 아래 스크립트를 수행해도 된다.

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
 ENAME VARCHAR2(10),
 JOB VARCHAR2(9),
 MGR NUMBER(4),
 HIREDATE DATE,
 SAL NUMBER(7, 2),
 COMM NUMBER(7, 2),
 DEPTNO NUMBER(2)
);

INSERT INTO EMP VALUES (7369, 'SMITH',  'CLERK',     7902,
TO_DATE('17-DEC-1980', 'DD-MON-YYYY'),  800, NULL, 20);
INSERT INTO EMP VALUES (7499, 'ALLEN',  'SALESMAN',  7698,
TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600,  300, 30);
```

```
INSERT INTO EMP VALUES (7521, 'WARD',   'SALESMAN',  7698,
TO_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250,  500, 30);
INSERT INTO EMP VALUES (7566, 'JONES',  'MANAGER',   7839,
TO_DATE('2-APR-1981', 'DD-MON-YYYY'),  2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN',  7698,
TO_DATE('28-SEP-1981', 'DD-MON-YYYY'), 1250, 1400, 30);
INSERT INTO EMP VALUES (7698, 'BLAKE',  'MANAGER',   7839,
TO_DATE('1-MAY-1981', 'DD-MON-YYYY'),  2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK',  'MANAGER',   7839,
TO_DATE('9-JUN-1981', 'DD-MON-YYYY'),  2450, NULL, 10);
INSERT INTO EMP VALUES (7788, 'SCOTT',  'ANALYST',   7566,
TO_DATE('09-DEC-1982', 'DD-MON-YYYY'), 3000, NULL, 20);
INSERT INTO EMP VALUES (7839, 'KING',   'PRESIDENT', NULL,
TO_DATE('17-NOV-1981', 'DD-MON-YYYY'), 5000, NULL, 10);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN',  7698,
TO_DATE('8-SEP-1981', 'DD-MON-YYYY'),  1500,    0, 30);
INSERT INTO EMP VALUES (7876, 'ADAMS',  'CLERK',     7788,
TO_DATE('12-JAN-1983', 'DD-MON-YYYY'), 1100, NULL, 20);
INSERT INTO EMP VALUES (7900, 'JAMES',  'CLERK',     7698,
TO_DATE('3-DEC-1981', 'DD-MON-YYYY'),   950, NULL, 30);
INSERT INTO EMP VALUES (7902, 'FORD',   'ANALYST',   7566,
TO_DATE('3-DEC-1981', 'DD-MON-YYYY'),  3000, NULL, 20);
INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK',     7782,
TO_DATE('23-JAN-1982', 'DD-MON-YYYY'), 1300, NULL, 10);

CREATE TABLE DEPT
(DEPTNO NUMBER(2),
 DNAME VARCHAR2(14),
 LOC VARCHAR2(13)
);

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEWYORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH',   'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES',      'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

위의 명령어를 실행하여 스키마를 생성했다면, 이전 항목으로 돌아가서 제약조건을 생성하는 명령어를 실행해야 한다는 것을 기억하길 바란다.

환경 설정

이 책의 대부분의 예제는 SQL*Plus 환경에서 100% 수행되도록 설계되었다. 필자는 SQL*Plus 사용을 제안하며, 이 책의 거의 모든 예제는 DBMS_OUTPUT을 사용했다. DBMS_OUTPUT을 사용하기 위

해서는 아래와 같이 SQL*Plus 명령어를 수행해야만 한다.

```
SQL> set serveroutput on
```

필자처럼 했다면, 매번 성가시게 타이핑을 해야만 할 것이다. 다행히도 login.sql 파일에 설정하면 SQL*Plus를 실행할 때마다 매번 스크립트(login.sql)가 수행된다. 게다가 login.sql 스크립트가 어느 디렉터리에 있든지 찾을 수 있도록 SQLPATH 환경 변수도 설정할 수 있다.

이 login.sql 스크립트는 이 책의 모든 예제에 사용된다.

```
define _editor=vi
set serveroutput on size 1000000
set trimspool on
set long 5000
set linesize 100
set pagesize 9999
column plan_plus_exp format a80
column global_name new_value gname
set termout off
define gname=idle
column global_name new_value gname
select lower(user) ¦¦ '@' ¦¦ substr( global_name, 1, decode( dot, 0, length(global_name), dot-1) )
global_name
  from (select global_name, instr(global_name,'.') dot from global_name );
set sqlprompt '&gname> '
set termout on
```

이 파일에 대한 설명은 아래와 같다.

- define _editor=vi – SQL*Plus에서 사용하는 기본 편집기를 설정한다. 여러분이 애용하는 Notepad나 emacs 같은 텍스트 편집기를 설정할 수도 있다.
- set serveroutput on size unlimited – 기본값으로 DBMS_OUTPUT이 설정되므로(매번 set serveroutput on을 설정할 필요는 없다) 가능한 큰 크기로 기본 버퍼 크기를 설정한 것이다.
- set trimspool on – 텍스트로 스풀(spool)할 때 라인 공백을 제거하여 고정 길이로 출력되지 않는다. 기본 설정은 off이고, linesize를 설정한 만큼 스풀된 라인이 넓어진다.
- set long 5000 – LONG과 CLOB 컬럼을 조회할 때 보이는 바이트의 기본값을 설정한다.
- set linesize 100 – SQL*Plus에서 보이는 라인의 넓이를 100자로 설정한다.
- set pagesize 9999 – 자주 SQL*Plus에서 헤딩(heading) 출력을 제어하기 위해 pagesize 설정

을 한다. 페이지당 한 개의 헤딩 집합을 출력하기 위해 큰 숫자를 사용한다.

- column plan_plus_exp format a80 – 이것은 AUTOTRACE를 사용하여 출력되는 실행계획의 기본 넓이 값을 설정하는 것이다. a80은 전체 실행계획을 나타내는 데 일반적으로 충분한 넓이다.

다음은 필자의 필요에 의해 SQL*Plus 프롬프트를 login.sql에 설정한 부분이다.

```
define gname=idle
column global_name new_value gname
select lower(user) || '@' || substr( global_name, 1, decode( dot, 0, length(global_name), dot-1) )
global_name
  from (select global_name, instr(global_name,'.') dot from global_name );
set sqlprompt '&gname> '
```

column global_name new_value gname 지시어는 global_name으로 명명된 컬럼을 조회하여 최신 값을 가져오고, 대체 변수 gname에 최신 값을 넣으라고 SQL*Plus에게 말하는 것이다. 데이터베이스의 global_name을 조회하고 난 후, 접속한 사용자명과 global_name을 연결하여 필자의 prompt를 아래에 보이는 것처럼 만드는 것이다.

```
ops$tkyte@ora11gr2>
```

이로 인해 어떤 사용자명으로 접속했고 어느 디렉터리에 있는지도 알 수 있다.

SQL*Plus에서 Autotrace 설정

AUTOTRACE는 SQL*Plus의 기능으로, 사용자가 사용한 자원과 실행한 쿼리의 실행계획을 보여준다. 이 책에서는 이 기능을 광범위하게 사용한다. AUTOTRACE 설정은 여러 가지 방법이 있다.

초기 설정

AUTOTRACE를 수행하려면 아래와 같이 해야 한다.

- cd [ORACLE_HOME]/rdbms/admin
- SYSTEM으로 SQL*Plus에 접속
- @utlxplan 실행
- CREATE PUBLIC SYNONYM PLAN_TABLE FOR PLAN_TABLE; 실행
- GRANT ALL ON PLAN_TABLE TO PUBLIC; 실행

GRANT TO PUBLIC을 사용자가 원하는 특정 유저로 바꿀 수 있다. public을 사용하면 SQL*Plus를 사용하는 모든 유저가 트레이스를 사용할 수 있다(필자 관점에서는 나쁜 것은 아니다). 플랜 테이블을 모든 사용자가 각각 설치하는 것을 막아준다. 모든 스키마가 AUTOTRACE를 사용하려면, @utlxplan을 실행시켜도 된다.

다음 단계는 PLUSTRACE 롤(role)을 부여하고 생성하는 것이다.

- cd [ORACLE_HOME]/sqlplus/admin
- SYS 또는 AS SYSDBA로 SQL*Plus에 접속
- @plustrce 실행
- GRANT PLUSTRACE TO PUBLIC;실행

다시 말하지만, GRANT 명령어에 PUBLIC은 사용자가 원하는 특정 유저로 변경할 수 있다.

리포트 제어

SQL 옵티마이저가 사용한 실행 경로와 문장 실행 통계정보를 자동으로 출력할 수 있다. 이 리포트는 SQL DML(SELECT, DELETE, UPDATE, MERGE와 INSERT) 문을 성공적으로 수행한 후에 생성된다. 이는 문장의 성능 튜닝과 모니터링에 유용하다.

AUTOTRACE 시스템 변수 설정으로 리포트를 조정할 수 있다.

- SET AUTOTRACE OFF – AUTOTRACE 리포트가 생성되지 않으며, 이것이 기본값이다.
- SET AUTOTRACE ON EXPLAIN – AUTOTRACE 리포트는 옵티마이저 실행 경로만을 보여준다.
- SET AUTOTRACE ON STATISTICS – AUTOTRACE 리포트는 SQL 문 실행 통계정보만을 보여준다.
- SET AUTOTRACE ON – AUTOTRACE 리포트는 옵티마이저 실행 경로와 SQL 문 실행 통계정보를 모두 보여준다.
- SET AUTOTRACE TRACEONLY – SET AUTOTRACE ON과 같으나 사용자 쿼리 결과값을 보여주지 않는다.

Statspack 설정

StatsPack은 SYSDBA(CONNECT/AS SYSDBA)로 접속하여 설치되도록 설계되었다. StatsPack을 설

치하는 것은 DBA나 관리자에게 요청해 수행해야만 하는 작업이다.

접속하고 StatsPack을 설치하기는 매우 쉽다. 단순히 @spcreate.sql을 실행하면 된다. [ORACLE_HOME]\rdbms\admin에서 이 스크립트를 찾을 수 있고, SQL*Plus에 SYSDBA로 접속해서 실행하면 된다.

spcreate.sql 스크립트를 실행하기 전에 세 가지 정보를 알 필요가 있다.

- PERFSTAT 스키마를 생성할 때 사용한 비밀번호를 사용한다.
- PERFSTAT의 기본 테이블 스페이스를 사용한다.
- PERFSTAT의 템포러리 테이블스페이스를 사용한다.

스크립트를 수행하면 아래와 같이 보일 것이다.

```
$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Fri May 28 10:52:522010
Copyright (c) 1982, 2009, Oracle.  All rights reserved.
Connected to:
Oracle Database 11g EnterpriseEditionRelease11.2.0.1.0-Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

sys%ORA11GR2> @spcreate

Choose the PERFSTAT user's password
----------------------------------
Not specifying a password will result in the installation FAILING
Enter value for perfstat_password:
… <output omitted for brevity> …
```

이 스크립트를 실행하는 데 필요한 정보를 입력해야 한다. 오자나 부주의로 인해 설치가 취소된 것이다. 다른 사람에 의해 이미 설치된 statspack 뷰와 유저가 있어서 이를 제거하려면 $ORACLE_HOME/rdbms/admin에 있는 spdrop.sql을 사용하면 된다. StatsPack 설치 시 spcpkg.lis라 불리는 파일이 생성된다. 오류가 발생하였는지 확인하려면 이 파일을 검토해야 할 것이다. 유효한 테이블스페이스 이름을 사용하는 한 유저, 뷰, PL/SQL 코드는 완벽하게 설치될 것이다.

사용자 정의 스크립트

이 절에서는 이 책에 사용된 다양한 스크립트 내의 코드를 설명하고자 한다.

Runstats

Runstats는 같은 작업을 하는 두 개의 다른 방법을 비교하고 우위에 있는 하나를 보여주기 위해 개발한 툴이다. 두 가지 다른 방법과 Runstats를 제공한다. Runstats는 세 가지 핵심사항을 측정한다.

- Wall clock or elapsed time – 이것은 알면 유용하나, 가장 중요한 정보는 아니다.
- System statistics – 이것은 단계적으로 둘 간의 차이점과 각각 무언가(예를 들어, parse call과 같은)를 액세스한 횟수를 보여준다.
- Latching – 이것이 리포트의 핵심 출력물이다.

이 책을 보면 알겠지만, 래치는 가벼운 락의 일종이다. 락은 직렬화 장치다. 직렬화 장치는 동시성을 금지한다. 동시성을 금지한 애플리케이션은 확장이 떨어지고, 적은 사용자만을 지원할 수 있으며, 적은 자원을 필요로 한다. 우리의 목적은 항상 잠재적인 확장성을 가진 한 유저뿐만 아니라 1,000이나 10,000 유저까지 서비스할 수 있는 애플리케이션을 만드는 것이다. 애플리케이션에서 래치가 적게 발생할수록 더 좋을 것이다. 필자는 수행하는 데 시간이 더 걸리는 방법을 선택할 것이나 래치의 10%로만 사용한다. 필자는 많은 래치를 사용하는 방법보다 래치를 줄여 더 적은 래치를 사용하는 방법을 알고 있다.

Runstats는 독립된 환경에 적합하다. 즉, 한 유저가 사용하는 데이터베이스에서 유용하다. 실행 중인 통계정보와 래치(락)의 결과를 측정할 것인데, 수행 동안에 다른 세션에 의해 발생한 시스템 로드 및 래치를 원하지 않는다. 이런 테스트는 소규모 테스트 데이터베이스 환경이 적당하다. 필자는 종종 PC나 랩탑에서 Runstats를 사용한다.

| Note | 필자는 모든 개발자가 모든 것을 DBA에게 요청하지 않고, 자신의 생각을 시험하기 위한 테스트 데이터베이스를 가지고 있다고 생각한다. 개발자들은 자신의 데스크톱에 배포용이 아닌 개발하고 테스트하는 데 사용할 personal developer 버전 라이선스가 있는 데이터베이스가 반드시 있어야 한다. 이로 인해 손해 볼 것은 없다. 필자는 컨퍼런스나 세미나에서 비공식적으로 조사한 자료가 있는데, 그곳의 거의 모든 DBA는 개발자부터 시작했다. 그 개발자들은 실제로 어떻게 작동하고 결과가 발생하는지 볼 수 있는 자신만의 데이터베이스를 가짐으로써 경험과 훈련을 할 수 있었다.

Runstats를 사용하려면, 통계정보를 보관하기 위한 테이블 생성 및 여러 V$ 뷰에 액세스해서 설정해야 하고, Runstats 패키지를 생성해야 한다. 4개의 V$ 테이블(V$STATNAME, V$MYSTAT, V$TIMER와 V$LATCH)을 액세스해야 한다. 아래는 필자가 사용한 뷰다.

```
create or replace view stats
as select 'STAT...' ¦¦ a.name name, b.value
      from v$statname a, v$mystat b
```

```
 where a.statistic# = b.statistic#
union all
select 'LATCH.' ¦¦ name,  gets
  from v$latch
union all
select 'STAT...Elapsed Time', hsecs from v$timer;
```

| Note | 액세스 권한 부여가 필요한 실제 객체 이름은 V_$STATNAME, V_$MYSTAT 등이다. 즉, 객체 이름은 V$가 아닌 V_$를 액세스할 수 있는 권한을 사용해야 한다. V$는 V_$로 시작하는 이름을 가진 원래 뷰를 가리키는 시노님(synonym)이다. 그래서 V$STATNAME은 V_$STATNAME 뷰를 가리키는 시노님이다. 여러분은 뷰에 접근할 권한이 있어야만 한다.

V$STATNAME, V$MYSTAT, V$TIMER와 V$LATCH를 SELECT할 수 있는 권한을 직접 부여받거나, 이 객체들을 SELECT할 수 있는 뷰를 생성하고 뷰에 대한 SELECT 권한을 부여한다.

일단 설정을 하면 통계정보를 수집하기 위한 작은 테이블이 필요하다.

```
create global temporary table run_stats
( runid varchar2(15),
  name varchar2(80),
  value int )
on commit preserve rows;
```

마지막으로, 간단한 API 콜이 포함된 Runstats 패키지를 생성하는 것이 필요하다.

- RS_START(Runstats 시작)는 Runstats 테스트의 시작 부분을 말한다.
- RS_MIDDLE은 여러분이 생각한 것처럼 중간 부분을 말한다.
- RS_STOP은 리포트를 출력하고 종료하는 것이다.

상세 설명은 아래에 있다.

```
ops$tkyte%ORA11GR2> create or replace package runstats_pkg
  2  as
  3      procedure rs_start;
  4      procedure rs_middle;
  5      procedure rs_stop( p_difference_threshold in number default 0 );
  6  end;
  7  /
Package created.
```

파라미터 p_difference_threshold는 최종 데이터 출력량을 조정하는 데 사용한다. Runstats는 각 수행별로 통계정보와 래치 정보를 수집하고 각 테스트에서 사용된 자원의 양과 그것들의 차이를 리포트로 출력한다. 입력 파라미터를 사용하여 이 숫자보다 큰 차이가 발생한 통계정보와 래치만을 볼 수 있다. 기본값이 0이므로 모두 출력해볼 수 있다.

다음으로, 단계적으로 패키지 보디(package body) 프로시저를 볼 것이다. 패키지는 글로벌 변수로 시작하고, 실행에 대한 elapsed times를 기록하는 데 사용할 것이다.

```
ops$tkyte%ORA11GR2> create or replace package body runstats_pkg
  2  as
  3
  4  g_start number;
  5  g_run1  number;
  6  g_run2  number;
  7
```

다음은 RS_START며, 테이블에 저장된 통계정보를 지우고, 'before', 통계정보, 래치를 입력할 것이다. 0.01초까지 계산된 elapsed times를 사용하여 현재 타이머 값을 캡처할 것이다.

```
  8  procedure rs_start
  9  is
 10  begin
 11      delete from run_stats;
 12
 13      insert into run_stats
 14      select 'before', stats.* from stats;
 15
 16      g_start := dbms_utility.get_cpu_time;
 17  end;
 18
```

다음은 RS_MIDDLE 루틴이며, 이 프로시저는 G_RUN1에서 테스트의 첫 번째 수행 elapsed time을 기록하고 난 후 현재의 통계정보와 래치 정보를 입력한다. RS_START에서 이미 저장한 값에서 이 값을 빼면, 첫 번째 방법에서 얼마나 많은 래치와 커서(통계정보)를 사용했는지 등을 알 수 있다.

마지막으로, 다음 수행을 위해 시작 시간을 저장한다.

```
 19  procedure rs_middle
 20  is
 21  begin
```

```
22      g_run1 := (dbms_utility.get_cpu_time-g_start);
23
24      insert into run_stats
25      select 'after 1', stats.* from stats;
26      g_start := dbms_utility.get_cpu_time;
27
28  end;
29
```

이 패키지의 마지막 루틴은 RS_STOP 루틴이다. 각 수행에 대한 총 CPU 시간을 출력하고, 두 수행 간에 통계정보/래치 값을 비교하여 차이를 출력한다(임계치를 초과하는 것들만 출력한다).

```
30  procedure rs_stop(p_difference_threshold in number default 0)
31  is
32  begin
33      g_run2 := (dbms_utility.get_cpu_time-g_start);
34
35      dbms_output.put_line
36      ( 'Run1 ran in ' ¦¦ g_run1 ¦¦ ' cpu hsecs' );
37      dbms_output.put_line
38      ( 'Run2 ran in ' ¦¦ g_run2 ¦¦ ' cpu hsecs' );
39          if ( g_run2 <> 0 )
40          then
41      dbms_output.put_line
42      ( 'run 1 ran in ' ¦¦ round(g_run1/g_run2*100,2) ¦¦
43        '% of the time' );
44          end if;
45      dbms_output.put_line( chr(9) );
46
47      insert into run_stats
48      select 'after 2', stats.* from stats;
49
50      dbms_output.put_line
51      ( rpad( 'Name', 30 ) ¦¦ lpad( 'Run1', 12 ) ¦¦
52        lpad( 'Run2', 12 ) ¦¦ lpad( 'Diff', 12 ) );
53
54      for x in
55      ( select rpad( a.name, 30 ) ¦¦
56               to_char( b.value-a.value, '999,999,999' ) ¦¦
57               to_char( c.value-b.value, '999,999,999' ) ¦¦
58               to_char( ( (c.value-b.value)-(b.value-a.value)),
                                  '999,999,999' ) data
59          from run_stats a, run_stats b, run_stats c
60         where a.name = b.name
61           and b.name = c.name
```

```
 62           and a.runid = 'before'
 63           and b.runid = 'after 1'
 64           and c.runid = 'after 2'
 65
 66           and abs( (c.value-b.value) - (b.value-a.value) )
 67                 > p_difference_threshold
 68         order by abs( (c.value-b.value)-(b.value-a.value))
 69      ) loop
 70          dbms_output.put_line( x.data );
 71      end loop;
 72
 73      dbms_output.put_line( chr(9) );
 74      dbms_output.put_line
 75      ( 'Run1 latches total versus runs -- difference and pct' );
 76      dbms_output.put_line
 77      ( lpad( 'Run1', 12 ) ¦¦ lpad( 'Run2', 12 ) ¦¦
 78        lpad( 'Diff', 12 ) ¦¦ lpad( 'Pct', 10 ) );
 79
 80      for x in
 81      ( select to_char( run1, '999,999,999' ) ¦¦
 82               to_char( run2, '999,999,999' ) ¦¦
 83               to_char( diff, '999,999,999' ) ¦¦
 84               to_char( round( run1/decode( run2, 0,
                            to_number(0), run2) *100,2 ), '99,999.99' ) ¦¦ '%' data
 85          from ( select sum(b.value-a.value) run1, sum(c.value-b.value) run2,
 86                        sum( (c.value-b.value)-(b.value-a.value)) diff
 87                   from run_stats a, run_stats b, run_stats c
 88                  where a.name = b.name
 89                    and b.name = c.name
 90                    and a.runid = 'before'
 91                    and b.runid = 'after 1'
 92                    and c.runid = 'after 2'
 93                    and a.name like 'LATCH%'
 94                  )
 95      ) loop
 96          dbms_output.put_line( x.data );
 97      end loop;
 98  end;
 99
100  end;
101  /
Package body created.
```

Runstats를 사용할 준비가 되었다. 한 예로, 한 개의 벌크 INSERT와 로우별 처리의 영향도를 보기 위해 Runstats를 사용하는 법을 소개할 것이다. 1,000,000로우를 INSERT할 두 개의 테이블을 설정하는 것부터 시작할 것이다(BIG_TABLE 생성 스크립트는 이 절 후반부에 있다).

```
ops$tkyte%ORA11GR2> create table t1
  2  as
  3  select * from big_table.big_table
  4  where 1=0;
Table created.

ops$tkyte%ORA11GR2> create table t2
  2  as
  3  select * from big_table.big_table
  4  where 1=0;
Table created.
```

첫 번째 방법으로, 한 개의 SQL 문을 사용하여 레코드를 입력한다. RUNSTATS_PKG.RS_START를 호출하여 시작한다.

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_start;
PL/SQL procedure successfully completed.

ops$tkyte%ORA11GR2> insert into t1
  2  select *
  3    from big_table.big_table
  4   where rownum <= 1000000;
1000000 rows created.

ops$tkyte%ORA11GR2> commit;
Commit complete.
```

두 번째 방법으로, 데이터를 로우별로 입력한다.

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_middle;
PL/SQL procedure successfully completed.

ops$tkyte%ORA11GR2> begin
  2          for x in ( select *
  3                       from big_table.big_table
  4                      where rownum <= 1000000 )
  5          loop
  6                  insert into t2 values X;
  7          end loop;
  8          commit;
  9  end;
 10  /
PL/SQL procedure successfully completed.
```

마지막으로, 리포트를 생성한다.

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_stop(1000000)
Run1 ran in 411 cpu hsecs
Run2 ran in 6192 cpu hsecs
run 1 ran in 6.64% of the time

Name                                  Run1        Run2        Diff
STAT...opened cursors cumulati         213   1,000,365   1,000,152
STAT...execute count                   213   1,000,372   1,000,159
LATCH.shared pool                    2,820   1,006,421   1,003,601
STAT...recursive calls               3,256   1,014,103   1,010,847
STAT...physical read total byt 122,503,168 124,395,520   1,892,352
STAT...cell physical IO interc 122,503,168 124,395,520   1,892,352
STAT...physical read bytes     122,503,168 124,395,520   1,892,352
STAT...db block changes            110,810   2,087,125   1,976,315
STAT...file io wait time         5,094,828     438,102  -4,656,726
LATCH.cache buffers chains         571,469   5,510,416   4,938,947
STAT...undo change vector size   3,885,808  67,958,316  64,072,508
STAT...redo size               120,944,520 379,497,588 258,553,068

Run1 latches total versus runs -- difference and pct
Run1        Run2        Diff       Pct
804,522   6,840,599   6,036,077     11.76%

PL/SQL procedure successfully completed.
```

RUNSTATS_PKG 패키지를 설치하고 가능하면 애플리케이션을 개발할 때 절차적으로 프로그램을 작성하는 것보다 한 개의 SQL 문을 왜 사용해야 하는지를 보여준다.

Mystat

Mystat.sql과 mystat2.sql은 연산 전후의 오라클 '통계정보' 증가세를 보여주는 데 사용된다. Mystat.sql은 단순히 특정 통계정보의 시작 값을 캡처한 것이다.

```
set echo off
set verify off
column value new_val V
define S="&1"

set autotrace off
select a.name, b.value
from v$statname a, v$mystat b
where a.statistic# = b.statistic#
```

```
and lower(a.name) like '%' ¦¦ lower('&S')¦¦'%'
/
set echo on
```

mystat2.sql은 차이를 보여준다(&V는 첫 번째 스크립트 mystat.sql 수행 시 SQL*Plus NEW_VAL 기능을 사용하여 입력된 것으로 위 쿼리에서 조회된 마지막 VALUE다).

```
set echo off
set verify off
select a.name, b.value V, to_char(b.value-&V,'999,999,999,999') diff
from v$statname a, v$mystat b
where a.statistic# = b.statistic#
and lower(a.name) like '%' ¦¦ lower('&S')¦¦'%'
/
set echo on
```

예를 들어, UPDATE 문에 의해 생성되는 리두 크기를 볼 수 있다.

```
big_table@ORA11GR2> @mystat "redo size"
big_table@ORA11GR2> set echo off

NAME                                 VALUE
---------------------------- ----------
redo size                              496

big_table@ORA11GR2> update big_table set owner = lower(owner)
  2  where rownum <= 1000;

1000 rows updated.

big_table@ORA11GR2> @mystat2
big_table@ORA11GR2> set echo off

NAME                                     V DIFF
---------------------------- ---------- ---------------
redo size                             89592          89,096
```

1,000 로우의 UPDATE가 89,096바이트의 리두를 생성했음을 보여준다.

Show_Space

SHOW_SPACE 루틴은 데이터베이스 세그먼트의 공간 활용 정보를 자세히 출력해준다. SHOW_

SPACE 인터페이스는 다음과 같다.

```
ops$tkyte@ORA11GR2> desc show_space
PROCEDURE show_space
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 P_SEGNAME                      VARCHAR2                IN
 P_OWNER                        VARCHAR2                IN     DEFAULT
 P_TYPE                         VARCHAR2                IN     DEFAULT
 P_PARTITION                    VARCHAR2                IN     DEFAULT
```

인수는 아래와 같다.

- P_SEGNAME – 테이블 또는 인덱스 세그먼트의 이름이다.
- P_OWNER – 현재 유저가 기본값이지만, 다른 스키마로 이 루틴을 사용할 수도 있다.
- P_TYPE – TABLE이 기본값이고 객체의 유형을 나타낸다. 예를 들어, select distinct segment_type from dba_segments 리스트의 valid 세그먼트 유형이다.
- P_PARTITION – 파티션 객체에 관한 공간을 볼 때 파티션의 이름이다. SHOW_SPACE는 한 번에 한 파티션 공간만을 보여준다.

ASSM(Automatic Segment Space Management) 테이블스페이스 내의 세그먼트일 때 이 루틴의 출력은 아래와 같다.

```
big_table@ORA11GR2> exec show_space('BIG_TABLE');
Unformatted Blocks .....                    0
FS1 Blocks (0-25)  .....                    0
FS2 Blocks (25-50) .....                    0
FS3 Blocks (50-75) .....                    0
FS4 Blocks (75-100).....                    0
Full Blocks        .....                14,469
Total Blocks............................ 15,360
Total Bytes............................ 125,829,120
Total MBytes............................    120
Unused Blocks...........................    728
Unused Bytes............................  5,963,776
Last Used Ext FileId....................      4
Last Used Ext BlockId...................  43,145
Last Used Block.........................    296

PL/SQL procedure successfully completed.
```

리포트된 아이템은 다음과 같다.

- Unformatted Blocks – 테이블의 하이 워터 마크 아래에 할당된 블록 수로, 사용된 적이 없는 블록이다. unformatted와 unused 블록을 더하면 ASSM 객체 내에서 데이터를 저장하는 데 사용된 적이 없는 테이블에 할당된 블록 수의 합이다.
- FS1 Blocks–FS4 Blocks – 데이터를 가진 포맷된 블록으로, 각 블록의 '빈 공간'의 범위를 나타내는 것이다. 예를 들어, (0-25)는 0에서 25%가 비어 있는 블록 수다.
- Full Blocks – 더 이상 insert할 수 없는 가득 찬 블록 수다.
- Total Blocks, Total Bytes, Total Mbytes – 데이터베이스 블록, 바이트, 메가바이트로 측정한 할당된 세그먼트의 총 공간이다.
- Unused Blocks, Unused Bytes – 사용된 적이 없는 공간을 나타낸다. 이 블록은 세그먼트에 할당은 되었으나 세그먼트의 하이(high) 워터 마크 위에 있는 공간이다.
- Last Used Ext FileId – 데이터가 있는 마지막 익스텐트 파일의 파일 ID다.
- Last Used Ext BlockId – 마지막 인스텐트의 시작 블록 ID다. 마지막으로 사용된 파일 내의 블록 ID다.
- Last Used Block – 마지막 익스텐트에 사용된 마지막 블록의 블록 ID 오프셋이다.

SHOW_SPACE를 사용하여 테이블스페이스에 관리되는 사용자 공간에 객체를 볼 때 출력물은 아래와 유사하다.

```
big_table@ORA11GR2> exec show_space( 'BIG_TABLE' )
Free Blocks.............................              1
Total Blocks............................        147,456
Total Bytes.............................  1,207,959,552
Total MBytes............................          1,152
Unused Blocks...........................          1,616
Unused Bytes............................     13,238,272
Last Used Ext FileId....................              7
Last Used Ext BlockId...................        139,273
Last Used Block.........................          6,576

PL/SQL procedure successfully completed.
```

단지 다른 것은 리포트 시작에 Free Blocks뿐이다. 이것은 세그먼트의 첫 번째 프리리스트 그룹의 개수다. 필자의 스크립트 리포트는 프리리스트 그룹이 더 있다. 다중 프리리스트 그룹을 반영하려면 스크립트 수정이 필요하다.

코드 설명은 아래와 같다. 이 유틸리티는 데이터베이스에 DBMS_SPACE API의 상위 레이어다.

```
create or replace procedure show_space
( p_segname in varchar2,
  p_owner   in varchar2 default user,
  p_type    in varchar2 default 'TABLE',
  p_partition in varchar2 default NULL )
-- this procedure uses authid current user so it can query DBA_*
-- views using privileges from a ROLE and so it can be installed
-- once per database, instead of once per user that wants to use it
authid current_user
as
    l_free_blks                 number;
    l_total_blocks              number;
    l_total_bytes               number;
    l_unused_blocks             number;
    l_unused_bytes              number;
    l_LastUsedExtFileId         number;
    l_LastUsedExtBlockId        number;
    l_LAST_USED_BLOCK           number;
    l_segment_space_mgmt        varchar2(255);
    l_unformatted_blocks number;
    l_unformatted_bytes number;
    l_fs1_blocks number; l_fs1_bytes number;
    l_fs2_blocks number; l_fs2_bytes number;
    l_fs3_blocks number; l_fs3_bytes number;
    l_fs4_blocks number; l_fs4_bytes number;
    l_full_blocks number; l_full_bytes number;

    -- inline procedure to print out numbers nicely formatted
    -- with a simple label
    procedure p( p_label in varchar2, p_num in number )
    is
    begin
        dbms_output.put_line( rpad(p_label,40,'.') ||
                              to_char(p_num,'999,999,999,999') );
    end;
begin
   -- this query is executed dynamically in order to allow this procedure
   -- to be created by a user who has access to DBA_SEGMENTS/TABLESPACES
   -- via a role as is customary.
   -- NOTE: at runtime, the invoker MUST have access to these two
   -- views!
   -- this query determines if the object is an ASSM object or not
begin
      execute immediate
          'select ts.segment_space_management
```

```
          from dba_segments seg, dba_tablespaces ts
         where seg.segment_name      = :p_segname
           and (:p_partition is null or
               seg.partition_name = :p_partition)
           and seg.owner = :p_owner
           and seg.tablespace_name = ts.tablespace_name'
         into l_segment_space_mgmt
        using p_segname, p_partition, p_partition, p_owner;
exception
    when too_many_rows then
       dbms_output.put_line
       ( 'This must be a partitioned table, use p_partition => ');
       return;
end;


-- if the object is in an ASSM tablespace, we must use this API
-- call to get space information, else we use the FREE_BLOCKS
-- API for the user managed segments
if l_segment_space_mgmt = 'AUTO'
then
  dbms_space.space_usage
  ( p_owner, p_segname, p_type, l_unformatted_blocks,
    l_unformatted_bytes, l_fs1_blocks, l_fs1_bytes,
    l_fs2_blocks, l_fs2_bytes, l_fs3_blocks, l_fs3_bytes,
    l_fs4_blocks, l_fs4_bytes, l_full_blocks, l_full_bytes, p_partition);

  p( 'Unformatted Blocks ', l_unformatted_blocks );
  p( 'FS1 Blocks (0-25)  ', l_fs1_blocks );
  p( 'FS2 Blocks (25-50) ', l_fs2_blocks );
  p( 'FS3 Blocks (50-75) ', l_fs3_blocks );
  p( 'FS4 Blocks (75-100)', l_fs4_blocks );
  p( 'Full Blocks        ', l_full_blocks );
else
  dbms_space.free_blocks(
    segment_owner     => p_owner,
    segment_name      => p_segname,
    segment_type      => p_type,
    freelist_group_id => 0,
    free_blks         => l_free_blks);

  p( 'Free Blocks', l_free_blks );
end if;

-- and then the unused space API call to get the rest of the
-- information
dbms_space.unused_space
```

```
  ( segment_owner     => p_owner,
    segment_name      => p_segname,
    segment_type      => p_type,
    partition_name    => p_partition,
    total_blocks      => l_total_blocks,
    total_bytes       => l_total_bytes,
    unused_blocks     => l_unused_blocks,
    unused_bytes      => l_unused_bytes,
    LAST_USED_EXTENT_FILE_ID => l_LastUsedExtFileId,
    LAST_USED_EXTENT_BLOCK_ID => l_LastUsedExtBlockId,
    LAST_USED_BLOCK => l_LAST_USED_BLOCK );

    p( 'Total Blocks', l_total_blocks );
    p( 'Total Bytes', l_total_bytes );
    p( 'Total MBytes', trunc(l_total_bytes/1024/1024) );
    p( 'Unused Blocks', l_unused_blocks );
    p( 'Unused Bytes', l_unused_bytes );
    p( 'Last Used Ext FileId', l_LastUsedExtFileId );
    p( 'Last Used Ext BlockId', l_LastUsedExtBlockId );
    p( 'Last Used Block', l_LAST_USED_BLOCK );
end;
/
```

Big_Table

이 책의 예제에서 필자는 BIG_TABLE이라 불리는 테이블을 사용한다. 시스템에 따라 이 테이블은 한 레코드에서 4백만 레코드 사이에 있고, 200MB에서 800MB까지 다양한 크기로 사용된다. 모든 경우에서 테이블의 구조는 같다.

BIG_TABLE을 생성하려고 아래와 같은 스크립트를 작성했다.

- ALL_OBJECTS를 기초로 빈 테이블을 생성한다. 이 딕셔너리 뷰는 BIG_TABLE에 데이터를 입력하는 데 사용된다.
- 테이블을 NOLOGGING으로 만든다. 이는 선택사항이며, 필자는 성능을 위해 NOLOGGING을 사용했다. 테스트 테이블을 NOLOGGING으로 만드는 것은 안전하다. 오라클 Data Guard 같은 기능이 설정되지 않기 때문에 상용 시스템에서는 NOLOGGING을 사용하지 말아야 한다.
- ALL_OBJECTS의 내용으로 테이블에 데이터 생성 후에 테이블 자체로 반복적으로 입력할 때마다 크기가 정확히 두 배가 된다.
- 테이블에 기본 키 제약조건 생성
- 통계정보 수집

BIG_TABLE 테이블을 만들려면, SQL*Plus에서 아래 스크립트를 실행시키면 테이블에 원하는 데이터 수를 프롬프트에 입력할 수 있다.

```
create table big_table
as
select rownum id, a.*
  from all_objects a
 where 1=0
/
alter table big_table nologging;

declare
    l_cnt number;
    l_rows number := &1;
begin
    insert /*+ append */
    into big_table
    select rownum, a.*
      from all_objects a
     where rownum <= &1;

    l_cnt := sql%rowcount;

    commit;

    while (l_cnt < l_rows)
    loop
        insert /*+ APPEND */ into big_table
        select rownum+l_cnt,
               OWNER, OBJECT_NAME, SUBOBJECT_NAME, OBJECT_ID, DATA_OBJECT_ID,
               OBJECT_TYPE, CREATED, LAST_DDL_TIME, TIMESTAMP, STATUS,
               TEMPORARY, GENERATED, SECONDARY, NAMESPACE, EDITION_NAME
          from big_table
         where rownum <= l_rows-l_cnt;
        l_cnt := l_cnt + sql%rowcount;
        commit;
    end loop;
end;
/
alter table big_table add constraint
big_table_pk primary key(id);

exec dbms_stats.gather_table_stats( user, 'BIG_TABLE', estimate_percent=> 1);
```

테이블에 estimate로 baseline 통계정보를 생성했다. 기본 키와 연관된 인덱스는 생성될 때 자동으로

compute로 통계정보가 생성된다.

코딩 규칙

이 책에서 필자가 사용한 한 가지 코딩 규칙인 PL/SQL 코드에서 변수 명 설정 방법을 주목하길 바란다. 예를 들어, 아래와 같은 패키지 보디를 고려해보자.

```
create or replace package body my_pkg
as
   g_variable varchar2(25);

   procedure p( p_variable in varchar2 )
   is
      l_variable varchar2(25);
   begin
      null;
   end;
end;
/
```

여기에는 세 가지 변수인 글로벌 패키지 변수 G_VARIABLE, 프로시저 형식 매개 변수 P_VARIABLE과 로컬 변수 L_VARIABLE이 있다. 필자는 세 개(G_VARIABLE, P_VARIABLE, L_VARIABLE)의 뒷부분을 변수라 부른다. 글로벌은 G_, 매개 변수는 P_ 그리고 로컬 변수는 L_로 모두 시작한다. 데이터베이스 테이블의 컬럼과 PL/SQL 변수를 구분하는 것이 가장 큰 이유다. 예를 들어, 아래와 같은 프로시저가 있다.

```
create procedure p( ENAME in varchar2 )
as
begin
   for x in ( select * from emp where ename = ENAME ) loop
      Dbms_output.put_line( x.empno );
   end loop;
end;
```

위와 같은 식으로 프로시저를 만들면 EMP 테이블에서 ENAME이 NULL이 아닌 로우를 모두 출력한다. ENAME 컬럼을 자신의 값으로 비교(ename = ENAME)하도록 SQL 문을 작성했으니 당연한 결과다. 이를 피하려면 조건절을 ename = P.ENAME으로 살짝 바꿔주기만 하면 된다. 즉, PL/SQL 변수를 참조하려는 것임을 명시하기 위해 프로시저 이름을 붙이는 방식이다. 그렇더라도 이런 식으로 코딩하는 것은 바람직하지 않다. 실수하기 쉽고 오류를 유발하는 원인이 되기 때문이다.

필자는 항상 뒤쪽에 변수를 명명한다. 이 방법은 로컬 변수와 글로벌 변수 파라미터를 쉽게 구분할 수 있도록 해준다. 게다가 컬럼 이름과 변수 이름에 관한 혼동을 없앨 수 있다.