ORACLE PL/SQL PROGRAMMING

개발자를 위한 PL/SQL 프로그래밍

개발자를 위한 PL/SQL 프로그래밍

초판 1쇄 발행 2017년 6월 30일

지은이 나장근
펴낸이 장성두
펴낸곳 제이펍

출판신고 2009년 11월 10일 제406-2009-000087호
주소 경기도 파주시 회동길 159 3층 3-B호
전화 070-8201-9010 / **팩스** 02-6280-0405
홈페이지 www.jpub.kr / **원고투고** jeipub@gmail.com
독자문의 readers.jpub@gmail.com / **교재문의** jeipubmarketer@gmail.com

편집부 이민숙, 황혜나, 이 슬, 이주원 / **소통·기획팀** 민지환, 현지환
표지디자인 미디어픽스
용지 에스에이치페이퍼 / **인쇄** 한승인쇄사 / **제본** 광우제책사

ISBN 979-11-85890-91-3 (93000)
값 28,000원

ORACLE PL/SQL PROGRAMMING

개발자를 위한 PL/SQL 프로그래밍

나장근 지음

Jpub 제이펍

※ 드리는 말씀

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 등장하는 각 회사명, 제품명은 일반적으로 각 회사의 등록 상표 또는 상표입니다. 본문 중에는 ™, ©, ® 마크 등이 표시되어 있지 않습니다.
- 이 책에서 사용하고 있는 제품 버전은 독자의 학습 시점이나 환경에 따라 책의 내용과 다를 수 있습니다.
- 책 내용과 관련된 문의사항은 지은이나 출판사로 연락해 주시기 바랍니다.
  - 지은이: jangkeunna.books@gmail.com
  - 출판사: readers.jpub@gmail.com

차례

머리말

'책을 한 권 써 볼까?'라는 생각에 목차를 잡기 시작해서 첫 원고가 나올 때까지 거의 3년이 걸렸다. 딱히 언제까지 초고를 완성하겠다는 목표를 잡고 시작한 게 아니었다는 것도 원인이겠지만, 기간이 길어진 가장 큰 이유는 아마도 게으름이 아닌가 싶다. 처음에 목차를 잡고 나서 본문을 쓰기 시작할 때까지 1년 걸렸고, 다시 시작했지만 역시 게으름으로 인해 중단하고 한참을 버려 두었다가 또 다시 시작하기까지 또 1년이 걸렸다. 사실은 전에도 몇 번 책을 쓰려고 시도했지만 귀차니즘 때문에 몇 쪽 끄적거리다가 만 전과가 이미 있다. 이번에는 끝을 본 게 그나마 다행이 아닌가 하는 생각도 한다.

오라클 PL/SQL 프로그래밍을 주제로 책을 쓰게 된 이유는 간단하다. 책을 쓰기로 마음먹은 3년 전에는 번역서와 국내서 포함해 한국어 서적이 아직 한 권도 출간되지 않은 분야가 바로 오라클 PL/SQL 분야였다. 'SQL과 PL/SQL'이라는 제목으로 나온 책이 몇 권 있었지만, 주 내용은 모두 SQL이었고 PL/SQL은 양념 수준이었다. 이미 몇 사람이 지나간 영역이라면 굳이 내가 거기에 책을 한 권 더 보태어 서가를 무겁게 만들 필요는 없다는 생각이 들어서 자연적으로 아직 지나간 사람이 없는 이 주제로 책을 쓰기 시작했는데, 한 2년 미적거리는 동안에 번역서 한 권이 나와버렸다. 역시 게을러서는 될 일이 없다. 계속할까 말까 고민하다가 '개발자를 위한 PL/SQL 프로그래밍'이라는 방향으로, 이미 출판된 책과는 다른 색깔의 책을 써 보기로 했다. 결과적으로 이 책은 PL/SQL만을 사용하여 프로그래밍하는 경우에 필요한 지식은 물론이고, 자바나 .NET 같은 외부 프로그래밍 언어 개발자가 PL/SQL을 어떻게 사용할 수 있는지에 대한 주제까지 포함하게 되었다. 나 스스로가 오랫동안 개발자로서 일을 해왔으므로 개발자의 관점에서 저술하기 위해 많이 노력했다.

이 책은 입문자부터 중급자까지의 오라클 PL/SQL 개발자를 위해 쓰였다. PL/SQL을 막 시작해서부터 어느 정도 수준에 이를 때까지 여러분들이 필요로 할 PL/SQL의 다양한 기술들을

습득하기에 이 한 권으로 충분할 것이다. 상급자를 위한 책이 아니라고 해서 이 책의 수준이 낮은 것은 결코 아니다. 이 책에서 다루지 않는 상급 수준의 주제들은 대부분이 일반적인 상황에서 잘 사용되지 않는 것들이거나, 설명의 복잡함에 비해 얻을 수 있는 지식의 활용도가 높지 않아서 불필요하게 책의 무게를 늘리게 되는 것들이다. 사용할 일이 별로 없는 주제라면 굳이 이 책에 포함시킬 이유가 없다. 나는 얇은 책을 선호한다.

그건 그렇고, 여러분이 이 책을 사야 하는 이유는 무엇인가?

첫째로, 이 책은 오라클 PL/SQL 프로그래밍 영역에서 입문자부터 중급자까지의 개발자가 필요로 할 주제들을 거의 빠짐없이 담고 있기 때문이다. 처음부터 끝까지 정독하는 경우에도 유용할 것이고, 특정 주제만을 따로 찾아보기에도 유용할 것이다. 둘째로, 이 책은 PL/SQL 자체를 배우고 활용하는 데 필요한 지식과 더불어 자바나 .NET과 같은 언어에서 PL/SQL을 응용하는 프로그램을 작성하거나 자바 언어로 저장 프로시저를 작성하는 데 필요한 주제까지 포함하고 있어서 프로그래머가 PL/SQL 프로그램을 개발하고 이를 응용 프로그램에서 활용하는 과정 전체를 익히는 데 적합하기 때문이다. 한마디로 이 책은 개발자 관점에서 DB 응용프로그램을 개발하는 데 필요한 주제를 폭넓게 다루고 있다. 셋째로, SQL만 잘 아는 프로그래머보다는 PL/SQL까지도 잘 아는 프로그래머가 훨씬 더 높은 생산성을 낼 수 있기 때문이다. SQL을 모르고는 비즈니스 애플리케이션 구축이 불가능한 것은 엄연한 사실이다. 하지만 실제에서 SQL만으로 애플리케이션을 구축하기에는 제약 사항이 상당히 많다. PL/SQL은 SQL의 약점을 보충해 줄 수 있는 매우 유용한 도구다. SQL로 구현하기 어렵거나 불가능한 것들을 PL/SQL로 능숙하게 구현할 수 있게 된다면, 당신은 최고의 생산성을 내는 유능한 프로그래머가 될 수 있다. PL/SQL이 SQL의 가치를 높여 주는 도구이듯이, 이 책은 당신의 가치를 높여 주는 도구가 될 것이다.

이 책과 같이 봐야 하는 필수적인 문서가 하나 있다. 바로 오라클 매뉴얼 'PL/SQL Packages and Types Reference'다. 이 매뉴얼에는 오라클 PL/SQL에 막강한 힘을 더해 주는 수많은 라이브러리에 대한 설명이 담겨 있다. 이 책에서도 많이 사용할 패키지인 DBMS_OUTPUT을 포함해서 암호화 관련 패키지인 DBMS_CRYPTO, 작업 스케줄 작성을 위한 DBMS_SCHEDULER와 DBMS_JOB, DB Object의 DDL 스크립트를 생성하는 DBMS_METADATA, 동적 SQL 수행을 위한 DBMS_SQL, 통계 정보 생성을 위한 DBMS_STATS 등 수많은 유틸리티들에 대한 설명이 이 매뉴얼에 들어 있다. 문서는 오라클 웹 사이트인 docs.oracle.com에서 온라인으로 볼 수도 있고, PDF나 ePub, Mobi 형태로 다운로드받을 수도 있다. 어느 정도 PL/

SQL의 사용에 익숙해지고 나면 이 매뉴얼을 찾아볼 일이 점점 더 많아질 것이다. 전문가가 되려면 항상 매뉴얼을 가까이 해야 한다.

이 책을 출판하는 데까지 도움을 주신 분들에게 감사드린다. 깐깐한 내 성격 때문에 힘깨나 들었을 거라 짐작되지만, 끝까지 교정과 편집에 힘써 주신 제이펍 출판사의 모든 분들에게 고마움을 전한다. 아울러 만나보지는 못했지만, 베타리딩을 통해 책의 품질을 한 단계 올려준 여러 분들에게도 지면을 빌려 감사 인사를 드린다. 그리고 책 쓴답시고 방에 내내 틀어박혀서 모니터만 쳐다보던 남편을 말없이 지지해 주고 IT 문외한이면서도 때로는 지루한 교정 작업마저 함께 해준, 사랑하는 아내에게 가장 많은 고마움을 전한다.

아무쪼록 이 책이 여러분의 가치를 높이는 데 많은 도움이 되기를 바란다.

2017년 여름에
지은이 나장근

이 책에 사용된 표기법

영어 소문자	PL/SQL 코드에 나타나는 영어 소문자는 사용자가 생성하거나 작성한 테이블, 칼럼, 저장 함수 등을 의미한다. 단, 예제로 사용하는 scott 계정의 테이블은 모두 소문자로 표기한다.
영어 대문자	PL/SQL 코드에 나타나는 영어 대문자는 사용자가 생성한 것이 아니라 시스템에서 제공되는 것을 사용했음을 의미한다. 예를 들어 시스템 테이블, 딕셔너리 뷰, 데이터 타입, 내장 함수, 오라클 DBMS 설치 시에 기본으로 제공되는 패키지 등은 모두 영어 대문자로 표기한다.
강조	굵은 글씨(고딕체)는 강조를 나타낸다(의미 강조 또는 키워드).
사용자대체	이탤릭체 부분은 프로그래머가 적절한 값으로 바꿔 넣어야 하는 부분이다. 사용자 대체 부분은 일부분을 제외하고는 일부러 띄어쓰기를 무시하고 하나의 의미 단위로 붙여쓰기했다. 예를 들어 '*데이터 타입*'이라고 사용하면 하나의 의미 단위인 '데이터 타입'인지 '데이터'와 '타입' 두 개를 말하는 지 혼란이 올 수 있기 때문에 띄어쓰기를 하지 않고 '*데이터타입*'으로 표기했다.
`Coding 인용`	고정 피치 폰트를 사용한 단어는 코딩에 나타난 단어를 인용한 부분이다.
`코딩 본문`	회색 음영이 들어간 코드 박스는 코딩 본문을 의미한다.
...	세 개의 점으로 표시한 부분은 생략을 의미한다. 이 부분에는 한 줄 이상의 문장(Statement), 주석, PL/SQL 블록 등이 올 수 있다. 예제 프로그램 중에 생략 기호가 포함되어 있는 것은 실행 가능한 완전한 프로그램이 아니다.

[옵션] 문법(Syntax) 설명에서 '['와 ']'로 둘러싼 항목은 옵션이다(필요하면 사용하고, 필요가 없다면 생략하자).

선택-1 | 선택-2 | … 여러 항목 중 하나를 선택하는 경우에 사용된다.

줄 번호 예제 소스 코드 중 일부는 설명을 위해서 다음과 같이 소스 코드 박스 바깥에 줄 번호를 달았다. 아래와 같이 회색 박스 바깥에 줄 번호가 달린 경우, 줄 번호는 소스 코드가 아니며, 박스 안쪽만 소스 코드다.

```
01  BEGIN
02    DBMS_OUTPUT.PUT_LINE('Hello, World') ;
03  END ;
```

SQL*Plus 출력 소스 코드 중 일부는 다음의 예와 같이 SQL*Plus의 출력을 그대로 복사하여 붙여넣기 했다. 이 경우 첫 줄에는 'SCOTT> '와 같이 SQL*Plus 프롬프트가 나오고, 두 번째 줄부터는 줄 번호가 나오는데, SQL*Plus 프롬프트 'SCOTT> '와 줄 번호가 차지하는 첫 다섯 개 문자(아래 예제에서는 '__2__')는 소스 코드가 아니다.

```
SCOTT> BEGIN
  2    DBMS_OUTPUT.PUT_LINE('Hello, World') ;
  3  END ;
  4  /
Hello, World
```

베타리더 후기

이상현(SI 개발자)

읽기 쉽게 구성됐고, 친절하며, 세세한 부분까지 잘 다루어 따라 하기 쉬워서 좋았습니다. 간단히 해볼 수 있는 예제 위주라 읽는 동안 내내 즐거웠네요. 누군가 PL/SQL 책을 추천해달라고 한다면 망설임 없이 이 책을 추천하고 싶습니다! 잘 만들어지고 잘 다듬어진, 훌륭한 책입니다! 이런 좋은 책이 많은 분께 선택받기를 진심으로 바랍니다.

이아름

오라클을 잘 모르시는 분이면 한 번 쓱 읽고 필요한 것만 취하며 지나칠 수 있는 책은 아닌 것 같습니다. 여러 번 읽으며 필요한 추가 지식을 직접 찾아보게 만드는, 정말 '공부'하며 읽어야 할 책이지만 오라클을 좀 더 고급스럽게 쓰고 싶다면 읽어 두어야 할 책이라고 생각합니다. 개발할 때 써먹을 만한 양질의 PL/SQL 도서입니다.

이요셉(솔루티스)

최고, 최고, 최고! 진정으로 PL/SQL로 배우는 프로그래밍이라는 이름을 붙일 만한 도서입니다! 오라클 PL/SQL이 이렇게 잘 짜인 프로그래밍 언어인지 이 책을 읽기 전엔 미처 알지 못했네요. 쉽고, 재미있고 체계적입니다!

차준성

책의 첫 장은 간단한 PL/SQL문을 직접 작성해 보는 것으로 시작합니다. 간단한 예제를 먼저 작성해 보는 덕에 2장부터 시작되는 PL/SQL의 개념과 문법 설명이 더 효과적으로 전달되는 것 같습니다. 이후로도 차근차근 읽어가다 보면 잘 짜인 커리큘럼을 내세운 수업을 받는 느낌이 들 만큼 쉽고 친절한 설명으로 가득합니다. 개중에서도 간단하고 쉬운 예제와 친절한 주석

이 책을 이해하는 데 도움이 많이 되었습니다. 예제를 하나씩 직접 실행해 보고, 실제 업무에서는 어떻게 활용할 수 있을지 생각하면서 책을 읽는다면 더욱 효과적일 것 같습니다. 무엇보다 전반적으로 친절한 설명 때문에 PL/SQL 입문자용으로도 좋은 책인 것 같습니다.

최승호(Naver)

만약 PL/SQL을 사용해야 할 상황이 닥친다면 이 책으로 입문할 것을 강력히 추천합니다! 그만큼 쉽게 쓰인 책이자, PL/SQL의 내용 전반을 살펴보는 데 적합한 책입니다. 책 내용상의 예제 코드도 주어져서 더욱더 수월하게 살펴볼 수 있었고, 학습에도 정말 많은 도움이 되었습니다. PL/SQL에 대해 상당히 자세하게 쓰인 좋은 책입니다.

한홍근(서울옥션블루)

SQL 입문자 또는 오라클의 DB 시스템을 처음 다루는 사용자에게 추천합니다. 파트 I에서 책의 학습 방향과 설치 환경을 설명한 뒤, 상세한 설명과 많은 예시로 이론, 실습을 진행하는 매우 체계적인 구성을 보여 줍니다. 게다가 각 명령에 대한 설명과 예외 상황까지 짚어주어서 예상치 못한 문제가 닥쳐도 두려움 없이 극복하며 빠르게 PL/SQL을 익힐 수 있을 것으로 생각합니다. 개인적으로는 경쟁 관계에 있는(?) 기술을 비교하면서 공부, 베타리딩을 하니 장단점이 더욱 보여서 도움이 되었습니다. 책 내용 중에도 다른 SQL과 비교하는 점이 나와서 머릿속으로 정리하기가 참 좋았네요.

제이펍은 책에 대한 애정과 기술에 대한 열정이 뜨거운 베타리더들로 하여금
출간되는 모든 서적에 사전 검증을 시행하고 있습니다.

PART

I

PL/SQL 시작하기

관계형 데이터베이스가 사용되기 시작한 초기부터 사용자가 데이터베이스에 접근할 수 있도록 하는 공통적인 도구는 SQL(Structured Query Language)이었다. SQL을 사용하여 처리할 데이터 소스와 처리 결과만을 명시하면 나머지는 DBMS가 알아서 처리 절차와 알고리즘을 결정하고 최적화한 후 원하는 결과를 찾아 준다. 데이터베이스 사용자에게 SQL은 매우 편리한 언어다.

그렇다고 해서 SQL이 만능인 것은 아니다. SQL만으로 업무 로직을 처리하기에는 너무 복잡하거나 심지어는 불가능한 경우도 있다. 실제 업무 처리 로직에는 절차적인 요소가 매우 많다. PL/SQL이 없던 시절에는 이런 절차적인 요소를 코볼(COBOL)과 같은 언어가 담당했다. 메인프레임에서 실행되는 코볼 프로그램에 데이터베이스와의 인터페이스를 위한 임베디드 SQL(Embedded SQL) 처리 기능을 추가하여 절차적인 업무들을 처리했다. 과거에는 대규모 비즈니스 업무 처리에 사용된 언어가 거의 코볼 일색이었기 때문에 어쩌면 이것이 자연스러웠을 것이다.

이제는 시대가 변해서 다양한 프로그래밍 언어가 코볼의 자리를 대신하고 있다. 데이터베이스를 처리할 수 있는 프로그래밍 언어의 다양화에는 장점도 있지만, 구현 언어의 파편화로 인한 개발과 유지보수의 어려움이나 성능의 저하와 같은 문제점도 같이 가지고 있다. 이로 인해 데이터베이스에서도 절차적인 비즈니스 로직을 구현할 수 있는 단일 언어가 필요해졌다. 오라클 데이터베이스에서 이 역할을 위해 태어난 언어가 바로 PL/SQL이다.

PL/SQL 언어는 오라클 버전 6에서 처음 도입됐다. 이때의 PL/SQL은 프로그램을 데이터베이스에 저장할 수 있는 형태가 아니었고, 오라클사의 폼즈Forms라는 제품에 사용되는 언어였다. 다시 말해, 이때는 오라클 데이터베이스가 PL/SQL이라는 언어를 이해하지 못하던 시기였다. PL/SQL을 데이터베이스 내장 언어로 사용할 수 있게 된 것은 오라클 버전 7 이후부터다. 이때부터는 오라클 데이터베이스가 데이터뿐만 아니라 비즈니스 로직까지도 저장할 수 있게 되어 데이터베이스 관리 시스템DBMS으로서의 기능이 보다 막강해지고, 보다 프로그래밍 언어 중립적이 되었다고 할 수 있다.

지금은 데이터 처리를 위한 많은 기능이 PL/SQL로 작성되고 있다. PL/SQL 프로그램은 개발의 용이성과 표준화, 재사용성, 효율성, 유지보수성이 높고, 한 번 작성된 프로그램은 오라클이 설치된 운영체제의 종류와 무관하게 배포 가능한 장점이 있다. 게다가 오라클 버전 8*i*부터는 데이터베이스 내에 자바Java 컴파일러와 JVM(Java Virtual Machine)을 내장해서 자바로 저장 프로시저를 작성할 수도 있게 되어 개발자와 한층 더 가까워졌다.

물론, 모든 언어가 그렇듯이 PL/SQL에 장점만 있는 것은 아니다. 늘 그렇듯, 주의 깊은 검토를 통한 취사선택이 필요하다. 장점과 단점, PL/SQL로 할 수 있는 것과 없는 것들을 충분히 이해한 후에 사용한다면 PL/SQL은 여러분들에게 더 없이 훌륭한 도구가 되어 줄 것이다.

예제로 시작하기

새로운 프로그래밍 언어를 배우는 가장 빠른 길은 그 언어로 프로그램을 작성해 보는 것이다. 프로그램을 직접 작성하고 버그를 수정하여 실행해 보는 동안 머릿속에 프로그래밍 언어에 대한 구체적인 이미지가 자리 잡게 된다. 그러므로 본격적으로 PL/SQL에 대해 알아보기 전에 먼저 간단한 예제를 몇 개 살펴보도록 하자. PL/SQL에 대한 대략적인 감을 잡을 수 있을 것이다. 1장에서는 오라클과 SQL*Plus에 익숙하지 않은 독자분의 이해를 돕기 위해 예제 소스 코드와 실행 과정과 실행 결과를 상세하게 보여 줄 것이다. 2장부터는 실행 과정이나 실행 결과를 보여주지 않아도 이해하는 데 문제가 없거나 결과가 너무 뻔한 경우는 이를 생략하였다.

1.1 오라클 scott 예제 스키마

오라클을 설치하면 scott이라는 예제 스키마가 설치되어 있고, 이 계정에는 emp, dept, bonus, salgrade라는 네 개의 테이블이 들어 있다. 앞으로 이 책에 나오는 대부분의 예제는 이 네 개의 테이블을 사용할 것이다. scott 계정이 설치되지 않았다면 먼저 예제 1-1을 실행하여 계정 scott을 생성하라. 스크립트는 사용자 생성 권한이 있는 DBA 계정으로 접속해서 실행해야 한다. 오라클 버전 12c를 사용하는 경우에는 주의가 필요하다. 오라클 버전 12c에서 데이터베이스를 멀티테넌트 유형으로 생성한 경우에는 컨테이너 DB(CDB)와 플러거블

DB(PDB)라는 구분이 있는데, CDB에 접속하여 예제를 실행하면 오류 'ORA-65096: 공통 사용자 또는 롤 이름이 부적합합니다.'가 발생한다. 이를 피하려면 CDB가 아니라 PDB에 접속해야 한다. 12c 멀티테넌트 데이터베이스를 사용하는 독자는 먼저 '3.2.3 오라클 버전 12c 사용 시의 주의사항'을 읽어보기 바란다.

예제 1-1 scott 계정 생성

```
% sqlplus / as sysdba
REM scott 계정 생성
CREATE USER scott IDENTIFIED BY tiger ;
ALTER USER scott DEFAULT TABLESPACE USERS ;
ALTER USER scottt QUOTA UNLIMITED ON USERS ;
GRANT RESOURCE,CONNECT TO scott ;
```

다음으로 예제 1-2를 실행하여 예제 테이블과 예제 데이터를 생성한다.

예제 1-2 예제 데이터 생성

```
REM scott 계정으로 접속
CONN scott/tiger

REM 테이블을 생성하고 데이터를 삽입

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, DATE'1980-12-17',  800, NULL, 20);
INSERT INTO emp VALUES (7499, 'ALLEN',  'SALESMAN',  7698, DATE'1981-02-20', 1600,  300, 30);
INSERT INTO emp VALUES (7521, 'WARD',   'SALESMAN',  7698, DATE'1981-02-22', 1250,  500, 30);
INSERT INTO emp VALUES (7566, 'JONES',  'MANAGER',   7839, DATE'1981-04-02', 2975, NULL, 20);
INSERT INTO emp VALUES (7654, 'MARTIN', 'SALESMAN',  7698, DATE'1981-09-28', 1250, 1400, 30);
INSERT INTO emp VALUES (7698, 'BLAKE',  'MANAGER',   7839, DATE'1981-05-01', 2850, NULL, 30);
INSERT INTO emp VALUES (7782, 'CLARK',  'MANAGER',   7839, DATE'1981-06-09', 2450, NULL, 10);
INSERT INTO emp VALUES (7788, 'SCOTT',  'ANALYST',   7566, DATE'1987-04-19', 3000, NULL, 20);
INSERT INTO emp VALUES (7839, 'KING',   'PRESIDENT', NULL, DATE'1981-11-17', 5000, NULL, 10);
INSERT INTO emp VALUES (7844, 'TURNER', 'SALESMAN',  7698, DATE'1981-09-08', 1500,    0, 30);
INSERT INTO emp VALUES (7876, 'ADAMS',  'CLERK',     7788, DATE'1987-05-23', 1100, NULL, 20);
INSERT INTO emp VALUES (7900, 'JAMES',  'CLERK',     7698, DATE'1981-12-03',  950, NULL, 30);
```

```
INSERT INTO emp VALUES (7902, 'FORD',   'ANALYST',   7566, DATE'1981-12-03', 3000, NULL, 20);
INSERT INTO emp VALUES (7934, 'MILLER', 'CLERK',     7782, DATE'1982-01-23', 1300, NULL, 10);

CREATE TABLE dept
 (deptno NUMBER(2),
  dname  VARCHAR2(14),
  loc    VARCHAR2(13) );

INSERT INTO dept VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept VALUES (20, 'RESEARCH',   'DALLAS');
INSERT INTO dept VALUES (30, 'SALES',      'CHICAGO');
INSERT INTO dept VALUES (40, 'OPERATIONS', 'BOSTON');

CREATE TABLE bonus
  (ename VARCHAR2(10),
   job   VARCHAR2(9),
   sal   NUMBER,
   comm  NUMBER);

CREATE TABLE salgrade
(
  grade   NUMBER,
  losal   NUMBER,
  hisal   NUMBER
) ;

INSERT INTO salgrade VALUES (1,  700, 1200);
INSERT INTO salgrade VALUES (2, 1201, 1400);
INSERT INTO salgrade VALUES (3, 1401, 2000);
INSERT INTO salgrade VALUES (4, 2001, 3000);
INSERT INTO salgrade VALUES (5, 3001, 9999);

COMMIT;
```

오라클 예제 스키마 scott의 테이블 네 개에는 당연히 있어야 할 인덱스와 제약 조건이 존재하지 않는다. 추가로 예제 1-3을 실행하여 인덱스와 PRIMARY KEY 제약 조건을 생성하자.

예제 1-3 예제 스키마의 제약 조건 생성

```
REM dept 테이블 인덱스와 제약 조건
CREATE UNIQUE INDEX dept_pk ON dept(deptno) ;
ALTER TABLE dept ADD CONSTRAINT dept_pk PRIMARY KEY(deptno) ;

REM emp 테이블 인덱스와 제약 조건
CREATE UNIQUE INDEX emp_pk ON emp(empno) ;
ALTER TABLE emp ADD CONSTRAINT emp_pk PRIMARY KEY(empno) ;
CREATE UNIQUE INDEX emp_ename_uk ON emp(ename) ;
ALTER TABLE emp ADD CONSTRAINT emp_ename_uk UNIQUE(ename) ;
```

```
REM bonus 테이블 인덱스와 제약 조건
CREATE UNIQUE INDEX bonus_pk ON bonus(ename) ;
ALTER TABLE bonus ADD CONSTRAINT bonus_pk PRIMARY KEY(ename) ;

REM salgrade 테이블 인덱스와 제약 조건
CREATE UNIQUE INDEX salgrade_pk ON salgrade(grade) ;
ALTER TABLE salgrade ADD CONSTRAINT salgrade_pk PRIMARY KEY(grade) ;
```

1.2 예제 프로그램

이제 간단한 PL/SQL 프로그램을 작성해 보자. 예제는 서버에 저장되지 않는 유형인 익명(Anonymous) PL/SQL 한 개와 서버에 저장되는 유형인 함수(Function) 한 개, 프로시저(Procedure) 한 개로, 총 세 개를 작성할 것이다(각 유형들에 대한 자세한 설명은 뒤에서 다시 할 예정이다). 예제들은 PL/SQL 프로그램이 어떤 것인지를 보여 주기 위한 것이므로 가능하면 간단하게 작성할 것이다.

1.2.1 익명 PL/SQL 예제

첫 번째 예제는 익명 PL/SQL이다. 익명 PL/SQL은 이름을 가지지 않는(Anonymous) PL/SQL 프로그램으로, 서버에 저장되지 않고 일회성으로 실행되는 유형의 PL/SQL 프로그램을 말한다.

scott 계정의 테이블 emp를 사용하여 다음과 같은 요구 사항을 처리하는 PL/SQL 프로그램을 생각해 보자.

프로그램 요구 사항

(사번, 사원명, 업무)가 값 (v_empno, v_ename, v_job)로 주어졌을 때,

1. 사번 v_empno가 테이블 emp에 존재하면,
   - 테이블의 해당 로우의 (사원명, 업무)를 (v_ename, v_job)으로 변경
2. 존재하지 않으면,
   - 해당 정보를 테이블 emp에 등록
   - 등록 시 부서 코드 값은 20을 기본값으로 사용

위의 요구 사항을 처리하는 프로그램은 예제 1-4와 같다. 왕 초보자의 경우라면 예제가 쉽게 이해되지 않을 수도 있지만, 실망하지 말기 바란다. PL/SQL을 이미 접해 보았거나 다른 프로그래밍 언어를 사용해 본 분들은 어렵지 않게 이해할 것이고, 그렇지 않은 분들은 한 번에 이

해하기에는 낯설 수도 있다. 하지만 모든 일이 그렇듯, 몇 번 보다 보면 금세 익숙해진다.

예제 1-4 익명 PL/SQL

```
DECLARE
  -- 상수
  c_default_deptno CONSTANT NUMBER := 20 ; -- 기본 부서 코드

  -- 처리 대상 사원 정보를 값으로 가지는 변수 정의
  v_empno     NUMBER(4)     := 7788    ;     -- 처리 대상 사번
  v_ename     VARCHAR2(10) := 'SCOTT' ;     -- 처리 대상 사원명
  v_job       VARCHAR2(9)   := 'ANALYST' ;  -- 처리 대상 사원의 업무

  -- 추가 변수
  v_cnt       NUMBER ;        -- 건수
BEGIN
  -- 주어진 사번의 존재 여부 확인
  --   v_cnt > 0 : 존재
  --         = 0 : 없음
  SELECT COUNT(*)
    INTO v_cnt
    FROM emp
   WHERE empno = v_empno ;

  -- 1. 해당 사번이 emp 테이블에 존재하면
  IF v_cnt > 0 THEN
    --  1.1 (사원명, 업무)를(v_ename, v_job)으로 변경
    UPDATE emp
       SET ename = v_ename,
           job   = v_job
     WHERE empno = v_empno ;

    DBMS_OUTPUT.PUT_LINE('사원 "' || v_ename || '"의 정보가 변경되었습니다.') ;
  ELSE  -- 2. 해당 사번이 emp 테이블에 존재하지 않으면
    -- 새로운 사원 정보를 테이블에 등록
    INSERT INTO emp(empno, ename, job, deptno)
    VALUES (v_empno, v_ename, v_job, c_default_deptno) ;

    DBMS_OUTPUT.PUT_LINE('신입사원 "' || v_ename || '"이(가) 등록되었습니다.') ;
  END IF ;
  COMMIT ;
EXCEPTION WHEN OTHERS THEN
  ROLLBACK ; -- 모든 변경 취소
  DBMS_OUTPUT.PUT_LINE('응용 프로그램 오류 발생' || CHR(10) || SQLERRM) ;
END ;
/
```

SQL에 능숙한 분들은 위의 요구 조건을 처리하는 이 PL/SQL 프로그램 대신에 동일한 기능을

하는 단 한 문장의 SQL 쿼리를 작성할 수도 있을 것이다(DBMS_OUTPUT을 사용하는 메시지 출력은 제외하고). 예제는 순전히 PL/SQL 프로그램이 어떤 형식으로 작성되고 어떤 기능들을 수행할 수 있는지를 보여 주기 위한 목적으로 작성되었으므로 최적화된 프로그램이 아니라는 것을 이해하기 바란다.

프로그램을 각 줄별로 설명하면 다음과 같다.

01: 키워드 DECLARE를 사용하여 PL/SQL 상수/변수/서브프로그램 등을 선언하고 정의하는 구역을 시작한다. 대부분의 익명 PL/SQL 프로그램은 DECLARE절을 가지고 있다. DECLARE절에 선언할 것이 없다면 DECLARE는 생략 가능하다. 그렇다고 하더라도 묵시적으로 DECLARE절은 존재한다.

02: 두 개의 연속된 대시는 줄 단위 주석을 의미한다. 줄 주석 구분자 -- 뒤의 모든 내용은 무시된다. 여러 줄에 걸친 주석은 /* ... */를 사용한다.

03: 이름이 c_default_deptno인 NUMBER형 상수(CONSTANT)를 선언하고 초깃값을 20으로 정의한다.

변수 또는 상수에 값을 할당할 때는 :=를 사용한다. 혼동하기 쉬운데, =는 논리 비교고 :=는 할당 연산자다. =와 :=는 완전히 다르다.

06 ~ 08: 처리 대상 사원의 정보 세 가지(사번, 사원명, 업무)를 값으로 가지는 변수 v_empno, v_ename, v_job을 선언하면서 초깃값을 정의한다. 실제에서라면 초깃값(7788, 'SCOTT', 'ANALYST')이 오는 자리에 바인드(Bind) 변수를 사용하겠지만, 이 예제에서는 이해를 쉽게 하기 위해 리터럴 값으로 처리했다.

11: 쿼리의 COUNT(*) 결괏값을 저장할 변수 v_cnt를 선언한다.

12: 키워드 BEGIN을 사용하여 DECLARE 구역을 종료하고 PL/SQL 실행문을 작성하는 구역을 시작한다. 모든 PL/SQL 프로그램은 BEGIN으로 시작하는 실행 구역을 가져야 한다. BEGIN 키워드는 생략할 수 없다.

16 ~ 19: 테이블 emp에서 사번이 v_empno인 로우의 수를 계산하는 SELECT 쿼리를 수행하여 결과를 변수 v_cnt에 저장한다.

22: 키워드 IF를 사용하여 조건 처리를 시작한다.

24 ~ 27: UPDATE SQL문을 사용하여 사번이 v_empno인 로우의 칼럼(ename, job) 값을 (v_ename, v_job)으로 변경한다.

29: PL/SQL 프로그램에서 DB 서버로 문자열을 출력하는 오라클 내장 패키지 함수인 DBMS_OUTPUT.PUT_LINE을 사용하여 DB 서버로 메시지를 출력한다. SQL*Plus에

서 SET SERVEROUTPUT ON 명령을 실행하면, 이후부터는 실행되는 프로그램에서 DBMS_OUTPUT.PUT_LINE으로 출력한 메시지를 화면에 보여 준다.

30: 22번 줄의 조건 처리 IF문의 조건에 해당하지 않는 경우에 대한 처리를 시작(ELSE)한다.

32 ~ 33: INSERT SQL문을 사용하여 새로운 사원 정보를 테이블 emp에 삽입한다.

35: 서버 메시지를 출력한다.

36: END IF 키워드를 사용하여 22번 줄에서 시작한 IF절을 종료한다.

37: 트랜잭션을 COMMIT하여 변경을 확정하고, 다른 사용자가 변경된 값을 볼 수 있도록 한다.

38: 키워드 EXCEPTION를 사용하여 13 ~ 37번 줄 사이의 문장 수행 중에 발생한 예외 사항을 처리하는 예외 처리 구역을 시작한다.

39: 예외가 발생했을 시에는 트랜잭션을 취소한다.

41: END 키워드를 사용하여 PL/SQL 블록을 종료한다. 모든 PL/SQL 프로그램은 최소한 하나의 END 키워드를 가져야 한다.

42: 입력된 PL/SQL문을 실행한다.

SQL*Plus에서 입력된 SQL이나 PL/SQL을 실행하라고 지시하는 명령은 /다. 예제에서 42번 줄의 /는 41번 줄까지 작성된 프로그램을 실행하라는 의미다. /는 SQL*Plus 명령이며, PL/SQL이나 SQL의 일부가 아니다.

앞서 나온 예제는 여러분이 PL/SQL 언어를 사용하여 작성하게 될 프로그램의 전형적인 템플릿을 보여 준다. 예제에는 다음과 같은 주제들이 포함되어 있다.

1. PL/SQL 블록(DECLARE-BEGIN-END)의 사용
2. 상수와 변수의 선언 및 초깃값의 지정
3. 기본적인 네 가지 쿼리(SELECT, INSERT, UPDATE, DELETE) 중 DELETE문을 제외한 나머지 세 가지 쿼리의 사용 방법
4. IF-THEN-ELSE-END IF절을 사용하여 조건에 따른 처리 수행
5. DBMS_OUTPUT 내장 패키지를 사용하여 메시지를 서버로 출력
6. COMMIT/ROLLBACK을 사용한 트랜잭션 처리
7. EXCEPTION 블록을 사용한 예외 처리

각 주제들은 뒤따르는 장에서 하나씩 자세히 설명할 것이다.

SQL*Plus를 실행하여 scott 계정으로 접속한 후, 위의 예제 1-4를 실행하면 다음과 같은 메시지가 출력된다. SQL*Plus에 접속하여 실행한 첫 명령(세 번째 줄) SET SERVEROUTPUT ON은 프로그램에서 사용된 오라클 내장 패키지의 서브프로시저인 DBMS_OUTPUT.PUT_LINE에 의한 출력 메시지를 SQL*Plus 화면에 출력하라는 명령이다. 이 명령이 누락되면 출력 메시지가 SQL*Plus 화면에 출력되지 않는다.

```
% sqlplus scott/tiger

SCOTT> SET SERVEROUTPUT ON
SCOTT>
SCOTT> DECLARE
  2    -- 상수
  3    c_default_deptno CONSTANT NUMBER := 20 ; -- 기본 부서 코드
  4
  5    -- 처리 대상 사원 정보를 값으로 가지는 변수 정의
  6    v_empno     NUMBER(4)    := 7788  ;       -- 처리 대상 사번
  7    v_ename     VARCHAR2(10) := 'SCOTT' ;     -- 처리 대상 사원명
  8    v_job       VARCHAR2(9)  := 'ANALYST' ;   -- 처리 대상 사원의 업무
  9
 10    -- 추가 변수
 11    v_cnt       NUMBER ;        -- 건수
 12  BEGIN
 13    -- 주어진 사번의 존재 여부 확인
 14    --   v_cnt > 0 : 존재
 15    --         = 0 : 없음
 16    SELECT COUNT(*)
 17      INTO v_cnt
 18      FROM emp
 19     WHERE empno = v_empno ;
 20
 21    -- 1. 해당 사번이 emp 테이블에 존재하면
 22    IF v_cnt > 0 THEN
 23      --  1.1 (사원명, 업무)를 (v_ename, v_job)으로 변경
 24      UPDATE emp
 25         SET ename = v_ename,
 26             job   = v_job
 27       WHERE empno = v_empno ;
 28
 29      DBMS_OUTPUT.PUT_LINE('사원 "' || v_ename || '"의 정보가 변경되었습니다.') ;
 30    ELSE  -- 2. 해당 사번이 emp 테이블에 존재하지 않으면
 31      -- 새로운 사원 정보를 테이블에 등록
 32      INSERT INTO emp(empno, ename, job, deptno)
 33      VALUES (v_empno, v_ename, v_job, c_default_deptno) ;
 34
```

```
 35      DBMS_OUTPUT.PUT_LINE('신입사원 "' || v_ename || '"이(가) 등록되었습니다.') ;
 36    END IF ;
 37    COMMIT ;
 38  EXCEPTION WHEN OTHERS THEN
 39    ROLLBACK ; -- 모든 변경 취소
 40    DBMS_OUTPUT.PUT_LINE('응용 프로그램 오류 발생' || CHR(10) || SQLERRM) ;
 41  END ;
 42  /
사원 "SCOTT"의 정보가 변경되었습니다.

PL/SQL 처리가 정상적으로 완료되었습니다.
```

예제의 6번 줄에서 변수 v_empno를 사용하여 정의한 사번 7788이 테이블 emp에 이미 등록되어 있기 때문에 IF-THEN-ELSE-END IF 조건 처리에서 THEN 부분이 처리되어 24번 줄의 UPDATE 쿼리가 수행되었고, 29번 줄의 출력문이 수행되어 메시지 사원 "SCOTT"의 정보가 변경되었습니다가 출력되었다(사실은 ename과 job의 원래 값과 동일한 값으로 UPDATE했기 때문에 값 변경이 없다).

6번 줄의 변수 v_empno의 값을 테이블 emp에 존재하지 않는 사번인 9000으로 변경하고, 7번 줄의 사원명을 정의하는 변수 v_ename의 값을 '홍길동'으로 변경하여 다시 수행하자. 이렇게 하면 THEN 부분이 아닌 ELSE 부분이 수행되는데, 32번 줄의 INSERT문이 실행되고 35번 줄의 메시지 신입사원 ... 등록되었습니다.가 출력된다.

예제 1-5 존재하지 않는 사원으로 변경하여 실행

```
SCOTT> DECLARE
  2    -- 상수
  3    c_default_deptno CONSTANT NUMBER := 20 ; -- 기본 부서 코드
  4
  5    -- 처리 대상 사원 정보를 값으로 가지는 변수 정의
  6    v_empno    NUMBER(4)    := 9000    ;  -- 처리 대상 사번
  7    v_ename    VARCHAR2(10) := '홍길동' ;  -- 처리 대상 사원명
  8    v_job      VARCHAR2(9)  := 'CLERK' ;  -- 처리 대상 사원의 업무
  9
 10    -- 추가 변수
 11    v_cnt      NUMBER ;       -- 건수
 12  BEGIN
 13    -- 주어진 사번의 존재 여부 확인
 14    --   v_cnt > 0 : 존재
 15    --         = 0 : 없음
 16    SELECT COUNT(*)
 17      INTO v_cnt
 18      FROM emp
 19     WHERE empno = v_empno ;
```

```
20
21    -- 1. 해당 사번이 emp 테이블에 존재하면
22    IF v_cnt > 0 THEN
23      --  1.1 (사원명, 업무)를 (v_ename, v_job)으로 변경
24      UPDATE emp
25         SET ename = v_ename,
26             job   = v_job
27       WHERE empno = v_empno ;
28
29      DBMS_OUTPUT.PUT_LINE('사원 "' || v_ename || '"의 정보가 변경되었습니다.') ;
30    ELSE  -- 2. 해당 사번이 emp 테이블에 존재하지 않으면
31      -- 새로운 사원 정보를 테이블에 등록
32      INSERT INTO emp(empno, ename, job, deptno)
33      VALUES (v_empno, v_ename, v_job, c_default_deptno) ;
34
35      DBMS_OUTPUT.PUT_LINE('신입사원 "' || v_ename || '"이(가) 등록되었습니다.') ;
36    END IF ;
37    COMMIT ;
38  EXCEPTION WHEN OTHERS THEN
39    ROLLBACK ; -- 모든 변경 취소
40    DBMS_OUTPUT.PUT_LINE('응용 프로그램 오류 발생' || CHR(10) || SQLERRM) ;
41  END ;
42  /
신입사원 "홍길동"이(가) 등록되었습니다.

PL/SQL 처리가 정상적으로 완료되었습니다.
```

스크립트의 수정 없이 한 번 더 실행하면, 사번 7370이 위에서 이미 등록되었으므로 IF 조건 처리에서 THEN 부분의 UPDATE문과 29번 줄의 출력이 실행되어 사원 "홍길동"의 정보가 변경되었습니다.가 출력된다. 다음 스크립트의 첫 번째 줄과 같이 SQL*Plus의 프롬프트에서 첫 번째 문자로 /만을 입력한 후 엔터 키를 누르면 마지막으로 실행한 스크립트(예제 1-5의 PL/SQL문)를 다시 실행한다.

```
SCOTT> /
사원 "홍길동"의 정보가 변경되었습니다.

PL/SQL 처리가 정상적으로 완료되었습니다.
```

1.2.2 저장 함수 예제

저장 함수(Stored Function)는 서버에 저장되는 PL/SQL 서브프로그램이다. 서버에 저장되는 서브프로그램은 컴파일되어 서버에 저장된 후, 그 이름과 매개변수를 주어 실행할 수 있는 프로

그램을 말한다. 함수는 지정된 로직을 실행한 후 하나의 결과를 반환한다.

함수를 실행하기 위해서는 두 단계를 거쳐야 한다. 첫 번째는 저장 함수를 생성하는 단계다. 여러분은 함수를 정의하는 PL/SQL 프로그램을 작성하여 이를 실행해야 한다. 이 첫 번째 실행은 작성된 함수를 실행하는 것이 아니라 함수를 컴파일하여 서버에 저장하기만 한다. 두 번째 단계는 첫 번째 단계에서 작성하여 서버에 저장한 함수를 실제로 실행하는 단계다. 서버에 저장된 함수를 실행하는 방법은 다음과 같이 세 가지가 있다.

1. 익명 PL/SQL 프로그램에서 실행
2. 다른 저장 PL/SQL 프로그램에서 실행
3. SQL문에 포함시켜 실행

다음 예제 1-6은 첫 번째 단계로, 함수를 컴파일하여 서버에 저장하도록 만드는 프로그램이다. 함수의 이름은 get_dept_employee_count다(1번 줄에 이름이 명시되었다). 함수는 한 개의 매개변수를 사용하도록 정의되었는데, 부서 번호를 입력으로 받는다(2번 줄의 a_deptno). 또한, 함수는 NUMBER 타입의 결과를 반환하도록 정의되었는데(3번 줄의 RETURN NUMBER), 매개변수로 받은 부서 번호의 소속 사원 수를 반환한다.

예제 1-6 저장 함수

```
01  CREATE OR REPLACE FUNCTION get_dept_employee_count(
02            a_deptno NUMBER -- 사원 수를 계산할 부서 번호
03          ) RETURN NUMBER    -- 부서의 사원 수를 반환
04  IS
05    -- 변수
06    v_cnt NUMBER ;  -- 건수
07  BEGIN
08    -- 테이블 emp에 들어 있는, 부서 번호 a_deptno를 가진 사원의 수를 계산
09    SELECT COUNT(*)
10      INTO v_cnt
11      FROM emp
12     WHERE deptno = a_deptno ;
13
14    RETURN v_cnt ; -- 건수를 반환
15
16  EXCEPTION WHEN OTHERS THEN
17    DBMS_OUTPUT.PUT_LINE('응용 프로그램 오류 발생' || CHR(10) || SQLERRM) ;
18    RETURN -1 ;
19  END ;
```

이 프로그램을 SQL*Plus에서 다음과 같이 실행한다.

```
% sqlplus scott/tiger

SCOTT> CREATE OR REPLACE FUNCTION get_dept_employee_count(
  2            a_deptno NUMBER -- 사원 수를 계산할 부서 번호
  3          ) RETURN NUMBER   -- 부서의 사원 수를 반환
  4  IS
  5    -- 변수
  6    v_cnt NUMBER ;  -- 건수
  7  BEGIN
  8    -- 테이블 emp에서 부서 번호 a_deptno를 가진 사원의 수를 계산
  9    SELECT COUNT(*)
 10      INTO v_cnt
 11      FROM emp
 12     WHERE deptno = a_deptno ;
 13
 14    RETURN v_cnt ; -- 건수를 반환
 15
 16  EXCEPTION WHEN OTHERS THEN
 17    DBMS_OUTPUT.PUT_LINE('응용 프로그램 오류 발생' || CHR(10) || SQLERRM) ;
 18    RETURN -1 ;
 19  END ;
 20  /

함수가 생성되었습니다.
```

오류 없이 컴파일되었으므로 함수는 서버에 저장되었다. 이제부터 함수 get_dept_employee_count는 실행 가능하다. 다음 예제는 서버에 저장된 함수 get_dept_employee_count를 실행하는 SQL 쿼리다.

예제 1-7 SELECT문을 사용한 저장 함수의 실행

```
01   SELECT deptno  부서번호
02        , dname   부서명
03        , loc     위치
04        , get_dept_employee_count(deptno) 사원수
05     FROM dept ;
```

보다시피 쿼리는 매우 간단하다. 테이블 dept에서 부서번호, 부서명, 위치와 함께 SELECT문의 4번 줄에서 부서번호를 매개변수로 하여 함수 get_dept_employee_count를 호출하여 부서별 사원 수를 같이 출력한다. 이제 이 쿼리를 SQL*Plus에서 실행해 보자.

```
SCOTT> SET SERVEROUTPUT ON
SCOTT>
SCOTT> SELECT deptno  부서번호
```

```
  2       , dname   부서명
  3       , loc     위치
  4       , get_dept_employee_count(deptno) 사원수
  5    FROM dept ;

    부서번호  부서명          위치                사원수
---------- -------------- ------------- ----------
        10 ACCOUNTING     NEW YORK               3
        20 RESEARCH       DALLAS                 6
        30 SALES          CHICAGO                6
        40 OPERATIONS     BOSTON                 0
```

익명 PL/SQL에 포함하여 실행하는 예제는 다음과 같다.

예제 1-8 익명 PL/SQL문을 사용한 저장 함수의 실행

```
SCOTT> DECLARE
  2    v_cnt PLS_INTEGER ;
  3  BEGIN
  4    v_cnt := get_dept_employee_count(10) ;
  5    DBMS_OUTPUT.PUT_LINE('사원 수: '|| v_cnt) ;
  6  END ;
  7  /
사원 수: 3

PL/SQL 처리가 정상적으로 완료되었습니다.
```

1.2.3 저장 프로시저 예제

저장 프로시저(Stored Procedure)는 저장 함수(Stored Function)와 마찬가지로 서버에 저장되는 서브프로그램이다. 프로시저가 함수와 다른 점 중 하나는 결과를 반환하지 않는다는 것이다. 사실, 첫 번째 예제와 같은 익명 PL/SQL 프로그램처럼 DML을 수행하는 프로그램은 함수보다는 프로시저로 작성하는 것이 더 적합하다.

예제로 위의 익명 PL/SQL 프로그램을 프로시저로 바꿔 보도록 하겠다. 프로시저를 실행하기 위해서는 함수의 경우와 마찬가지로 두 단계를 거쳐야 한다. 첫 번째는 저장 프로시저를 생성하는 단계다. 여러분은 프로시저를 정의하는 PL/SQL 프로그램을 작성하여 이를 실행해야 한다. 저장 함수에서와 마찬가지로, 첫 번째 실행은 프로시저를 실행하는 것이 아니라 프로시저를 컴파일하여 서버에 저장하는 실행이다. 프로시저는 컴파일 시 서버에 저장되기만 할 뿐, 실행되지는 않는다. 두 번째 단계는 첫 번째 단계에서 작성하여 서버에 저장된 프로시저를 실제로 실행하는 단계다. 서버에 저장된 프로시저를 실행하는 방법은 다음과 같이 두 가지가 있다.

1. 익명 PL/SQL 프로그램에서 실행하는 방법
2. 다른 저장 PL/SQL 프로그램에서 실행하는 방법

프로시저는 함수와는 달리 SQL문에 포함하여 실행할 수 없다.

다음 예제 1-9는 첫 번째 단계로 프로시저를 컴파일하여 서버에 저장하기 위한 프로그램으로, 프로시저의 이름은 register_employee다(1번 줄에 이름이 명시되었다). 프로시저는 다섯 개의 매개변수를 사용하도록 정의되었다. 처음 세 개의 매개변수는 차례로 사번, 사원명, 업무다(2~4번 줄). 네 번째와 다섯 번째 매개변수에는 매개변수명과 데이터 타입 사이에 OUT이라는 키워드가 들어 있다. OUT은 이 변수가 프로시저에서 변경되어 결과가 호출자에게로 반환된다는 의미다. 네 번째 매개변수는 실행 결과가 성공인지 여부를 나타내기 위해서 BOOLEAN형으로 정의되었으며(5번 줄), 다섯 번째 매개변수는 처리 결과 메시지를 반환하기 위해서 VARCHAR2형으로 정의되었다(6번 줄). 31번 줄과 37번 줄, 43번 줄에서는 처리 결과를 직접 출력하는 대신에 출력 매개변수 a_msg_out에 저장한다. 39번 줄과 44번 줄에서는 처리 성공 여부를 출력 매개변수 a_rslt_out에 저장한다. 나머지 부분은 익명 PL/SQL 버전과 동일하다.

예제 1-9 저장 프로시저

```
01  CREATE OR REPLACE PROCEDURE register_employee(
02            a_empno        NUMBER,   -- 입력 변수: 등록할 사번 매개변수
03            a_ename        VARCHAR2, -- 입력 변수: 등록할 이름 매개변수
04            a_job          VARCHAR2, -- 입력 변수: 등록할 업무 매개변수
05            a_rslt_out OUT BOOLEAN,  -- 출력 변수: 처리 성공 여부
06            a_msg_out  OUT VARCHAR2  -- 출력 변수: 처리 결과를 반환하는 변수
07          )
08  IS
09    -- 상수
10    c_default_deptno CONSTANT NUMBER := 20 ; -- DEFAULT 부서 코드
11
12    -- 변수
13    v_cnt      NUMBER ;       -- 건수
14  BEGIN
15    -- 주어진 사번의 존재 여부 확인
16    --   v_cnt > 0 : 존재
17    --         = 0 : 없음
18    SELECT COUNT(*)
19      INTO v_cnt
20      FROM emp
21     WHERE empno = a_empno ;
22
23    -- 1. 해당 사번이 emp 테이블에 존재하면
```

```
24      IF v_cnt > 0 THEN
25        --  1.1 (사원명, 업무)를 (v_ename, v_job)으로 변경
26        UPDATE emp
27           SET ename = a_ename,
28               job   = a_job
29         WHERE ename = a_ename ;
30
31        a_msg_out := '사원 "' || a_ename || '"의 정보가 변경되었습니다.' ;
32      ELSE
33        -- 새로운 사원 정보를 테이블에 등록
34        INSERT INTO emp(empno, ename, job, deptno)
35        VALUES (a_empno, a_ename, a_job, c_default_deptno) ;
36
37        a_msg_out := '신입사원 "' || a_ename || '"이(가) 등록되었습니다.' ;
38      END IF ;
39      a_rslt_out := TRUE ;
40
41    EXCEPTION WHEN OTHERS THEN
42      ROLLBACK ; -- 모든 변경 취소
43      a_msg_out  := '응용 프로그램 오류 발생' || CHR(10) || SQLERRM ;
44      a_rslt_out := FALSE ;
45    END ;
```

저장 프로시저 예제 1-9를 SQL*Plus에서 다음과 같이 실행한다.

```
SCOTT> CREATE OR REPLACE PROCEDURE register_employee(
  2             a_empno        NUMBER,   -- 입력 변수: 등록할 사번 매개변수
  3             a_ename        VARCHAR2, -- 입력 변수: 등록할 이름 매개변수
  4             a_job          VARCHAR2, -- 입력 변수: 등록할 업무 매개변수
  5             a_rslt_out OUT BOOLEAN,  -- 출력 변수: 처리 성공 여부
  6             a_msg_out  OUT VARCHAR2  -- 출력 변수: 처리 결과를 반환하는 변수
  7           )
  8  IS
  9    -- 상수
 10    c_default_deptno CONSTANT NUMBER := 20 ; -- DEFAULT 부서 코드
 11
 12    -- 변수
 13    v_cnt      NUMBER ;       -- 건수
 14  BEGIN
 15    -- 주어진 사번의 존재 여부 확인
 16    --   v_cnt > 0 : 존재
 17    --         = 0 : 없음
 18    SELECT COUNT(*)
 19      INTO v_cnt
 20      FROM emp
 21     WHERE empno = a_empno ;
 22
 23    -- 1. 해당 사번이 emp 테이블에 존재하면
```

```
  IF v_cnt > 0 THEN
    --  1.1 (사원명, 업무)를 (v_ename, v_job)으로 변경
    UPDATE emp
       SET ename = a_ename,
           job   = a_job
     WHERE ename = a_ename ;

    a_msg_out := '사원 "' || a_ename || '"의 정보가 변경되었습니다.' ;
  ELSE
    -- 새로운 사원 정보를 테이블에 등록
    INSERT INTO emp(empno, ename, job, deptno)
    VALUES (a_empno, a_ename, a_job, c_default_deptno) ;

    a_msg_out := '신입사원 "' || a_ename || '"이(가) 등록되었습니다.' ;
  END IF ;
  a_rslt_out := TRUE ;

EXCEPTION WHEN OTHERS THEN
  ROLLBACK ; -- 모든 변경 취소
  a_msg_out  := '응용 프로그램 오류 발생' || CHR(10) || SQLERRM ;
  a_rslt_out := FALSE ;
END ;
/

프로시저가 생성되었습니다.
```

오류 없이 컴파일되었으므로 프로시저는 서버에 저장되었고, 이제부터 프로시저 register_employee는 실행 가능하다. 다음 예제 1-10은 서버에 저장된 프로시저 register_employee를 실행하는 프로그램이다. 여기서는 익명 PL/SQL 프로그램에서 실행하는 방법을 사용한다.

예제 1-10 저장 프로시저의 실행

```
DECLARE
  a_rslt_out BOOLEAN ;         -- 처리 성공 여부
  a_msg_out  VARCHAR2(1000) ; -- 처리 결과를 반환하는 변수
BEGIN
  register_employee(7788, 'SCOTT', 'ANALYST', a_rslt_out, a_msg_out) ;
  DBMS_OUTPUT.PUT_LINE(a_msg_out) ;
  IF a_rslt_out = TRUE THEN
    DBMS_OUTPUT.PUT_LINE('등록 성공!') ;
    COMMIT ;
  ELSE
    DBMS_OUTPUT.PUT_LINE('등록 실패!') ;
    ROLLBACK ;
  END IF ;
END ;
```

프로그램은 DECLARE절에서 프로시저의 출력 매개변수를 돌려받아 저장하기 위한 두 개의 변수를 선언하였다(2 ~ 3번 줄). 5번 줄은 프로시저를 실행하는 문장이고, 6 ~ 13번 줄은 반환받은 처리 결과를 출력하고 결과에 따라서 COMMIT 또는 ROLLBACK 처리를 하는 문장이다. 이제 예제 1-10을 SQL*Plus에서 실행하자.

```
SCOTT> SET SERVEROUTPUT ON
SCOTT>
SCOTT> DECLARE
  2    a_rslt_out BOOLEAN ;        -- 처리 성공 여부
  3    a_msg_out  VARCHAR2(1000) ; -- 처리 결과를 반환하는 변수
  4  BEGIN
  5    register_employee(7788, 'SCOTT', 'ANALYST', a_rslt_out, a_msg_out) ;
  6    DBMS_OUTPUT.PUT_LINE(a_msg_out) ;
  7    IF a_rslt_out = TRUE THEN
  8      DBMS_OUTPUT.PUT_LINE('등록 성공!') ;
  9      COMMIT ;
 10    ELSE
 11      DBMS_OUTPUT.PUT_LINE('등록 실패!') ;
 12      ROLLBACK ;
 13    END IF ;
 14  END ;
 15  /
사원 "SCOTT"의 정보가 변경되었습니다.
등록 성공!

PL/SQL 처리가 정상적으로 완료되었습니다.
```

지금까지 소개하는 의미로 간단한 PL/SQL 프로그램들을 살펴보았다. PL/SQL 프로그램이 가지는 개략적인 구조와 기능을 구현했으며, 간단한 PL/SQL 프로그램이라면 위의 예제를 변형해서 충분히 작성할 수 있을 것이다.

테스트용으로 생성한 데이터 한 건은 예제 1-11의 SQL을 실행하여 삭제한다.

예제 1-11 테스트 데이터 삭제

```
SCOTT> DELETE FROM emp WHERE empno = 9000;

1 행이 삭제되었습니다.

SCOTT> COMMIT;

커밋이 완료되었습니다.
```