

Oracle Database 10g: PL/SQL Fundamentals(한글판)

볼륨 I • 학생용

D17112KR30

Edition 3.0

2009년 11월

D63603

ORACLE

I

소개

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **과정 목표 설명**
- **과정 내용 설명**
- **본 과정에 사용되는 데이터베이스 테이블 식별**
- **완벽한 비즈니스 솔루션을 설계할 수 있는 Oracle 제품 식별**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 본 과정과 과정 흐름을 대략적으로 설명합니다. 본 과정에서 사용하는 데이터베이스 스키마와 테이블에 대해 알아봅니다. 또한 Oracle 10g 그리드 Infrastructure에 포함된 다양한 제품을 소개합니다.

과정 목표

이 과정을 마치면 다음을 수행할 수 있습니다.

- SQL의 프로그래밍 기능을 확장한 PL/SQL에 대한 설명
- 데이터베이스에 연결하는 PL/SQL 코드 작성
- 효과적으로 실행되는 PL/SQL 프로그램 단위 설계
- PL/SQL 프로그래밍 생성자 및 조건 제어문 사용
- 런타임 오류 처리
- 내장 프로시저 및 함수 설명

ORACLE

Copyright © 2009, Oracle. All rights reserved.

과정 목표

본 과정에서는 PL/SQL의 기본 사항을 설명합니다. PL/SQL 구문, 블록 및 프로그래밍 생성자에 대해 알아보고 이러한 생성자와 SQL을 통합할 때의 이점에 대해서도 알아봅니다. PL/SQL 프로그램 단위를 작성하고 효율적으로 실행하는 방법을 배웁니다. 또한 Oracle SQL Developer를 PL/SQL 개발 환경으로 사용하는 방법을 배우고, 프로시저 및 함수와 같이 재사용 가능한 프로그램 단위를 설계하는 방법도 배웁니다.

과정 내용

1일차 단원:

I. 소개

1. PL/SQL 소개
2. PL/SQL 변수 선언
3. 실행문 작성
4. Oracle 서버와 상호 작용
5. 제어 구조 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

과정 내용

2일차 단원:

6. 조합 데이터 유형 작업
7. 명시적 커서 사용
8. 예외 처리
9. 내장 프로시저 및 함수 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only



HR(Human Resources) 데이터 집합

슬라이드는 hr 스키마 테이블 및 해당 테이블의 관계를 보여줍니다.

departments는 사원의 소속 부서에 대한 세부 정보를 포함합니다. 각 부서에는 employees 테이블의 부서 관리자를 나타내는 관계가 있습니다.

HR(Human Resources) 데이터 집합(계속)

테이블 설명(계속)

`jobs`에는 각 사원이 보유할 수 있는 직무 유형이 포함되어 있습니다.

`job_history`에는 사원의 직무 기록이 포함되어 있습니다. 사원이 직무 내에서 부서를 변경하거나 부서 내에서 직무를 변경할 경우 새 행이 해당 사원의 이전 직무 정보와 함께 이 테이블에 삽입됩니다.

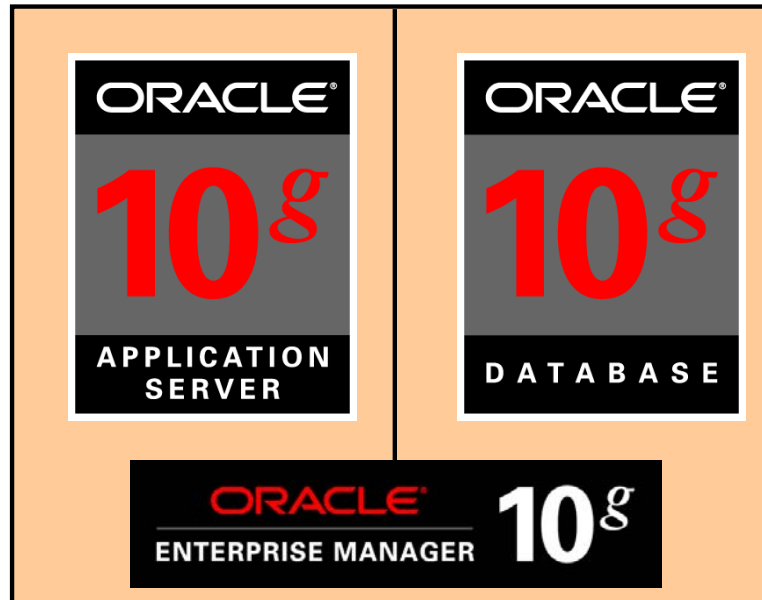
`locations`에는 국가에 있는 회사의 특정 지사, 도매점, 생산지 등의 특정 주소가 포함되어 있습니다.

`regions`에는 Americas, Asia 등과 같은 지역을 나타내는 행이 포함되어 있습니다.

`countries`에는 국가에 대한 행이 포함되어 있으며, 각 행은 `regions`와 연관되어 있습니다.

주: 이 단원에서는 hr 스키마의 다양한 테이블을 소개합니다. 각 테이블에 저장된 데이터를 보려면 부록 B ("테이블 설명 및 데이터")를 참조하십시오.

Oracle10g 그리드 Infrastructure



ORACLE

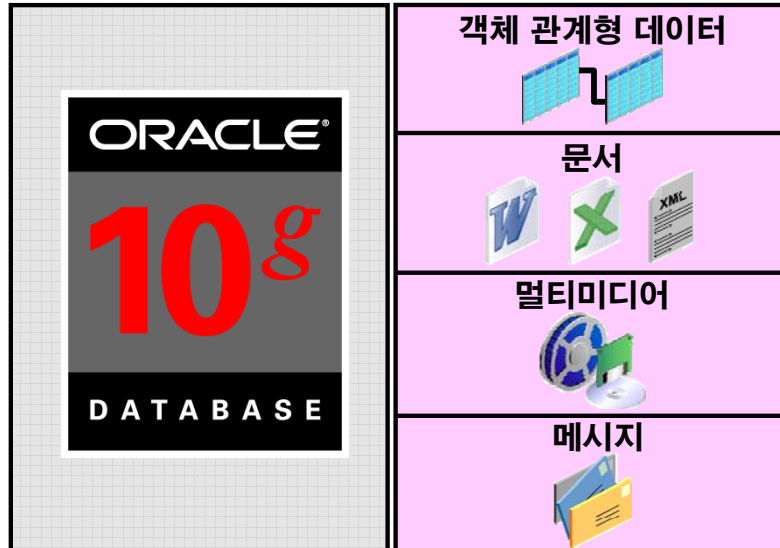
Copyright © 2009, Oracle. All rights reserved.

Oracle10g 그리드 Infrastructure

Oracle10g 릴리스의 세 가지 그리드 infrastructure 제품은 다음과 같습니다.

- Oracle Database 10g
- Oracle Application Server 10g
- Oracle Enterprise Manager 10g Grid Control

Oracle Database 10g



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 10g

Oracle Database 10g는 기업 정보를 저장하고 관리하도록 설계되었습니다. Oracle Database 10g를 사용하면 관리 비용을 줄이고 고품질 서비스를 보장할 수 있습니다. 줄어든 구성 및 관리 요구 사항과 자동 SQL 튜닝을 통해 환경 유지 관리 비용이 크게 절감됩니다.

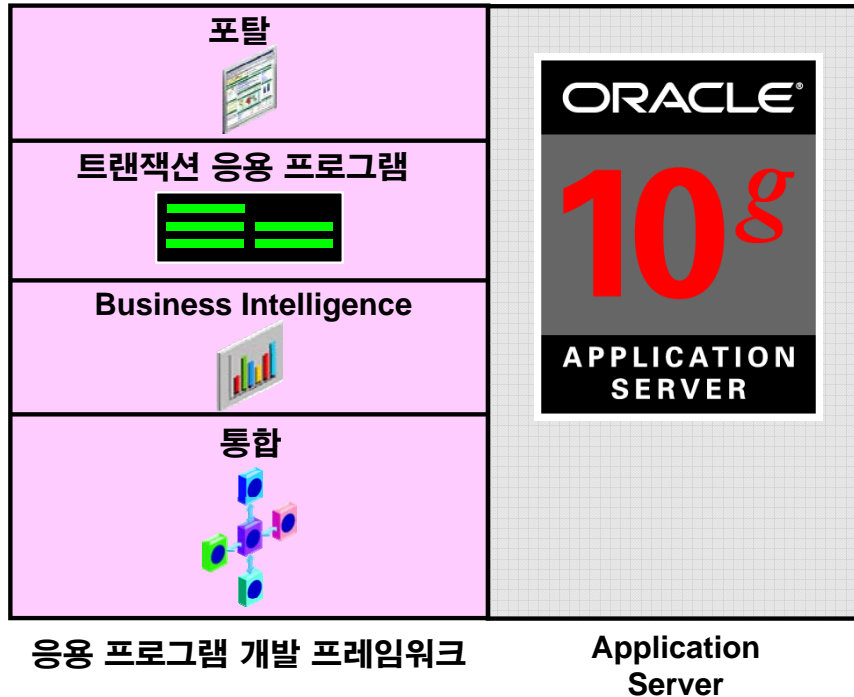
Oracle Database 10g는 Oracle 10g 릴리스의 그리드 infrastructure 제품 중 하나입니다. 그리드 컴퓨팅은 유틸리티로 활용할 수 있는 가장 완벽한 컴퓨팅입니다. 클라이언트는 데이터가 있는 위치나 데이터가 저장된 컴퓨터를 알 필요가 없습니다. 클라이언트가 정보를 요청하거나 데이터에 계산을 수행하면 클라이언트에 결과가 전달됩니다.

Oracle Database 10g는 여러분의 데이터를 모두 관리합니다. 기업 데이터베이스에서 관리할 데이터에 객체 관계형 데이터만 있는 것은 아닙니다. 다음과 같이 구조화되지 않은 데이터도 포함될 수 있습니다.

- 스프레드시트
- Word 문서
- PowerPoint 프리젠테이션
- XML
- MP3, 그래픽, 비디오 등의 멀티미디어 데이터 유형

데이터가 데이터베이스에 있어야 할 필요도 없습니다. Oracle Database 10g에는 파일 시스템에 저장된 정보에 대한 메타 데이터를 저장할 수 있는 서비스가 있습니다. 데이터베이스 서버를 사용함으로써 정보의 위치와 상관없이 정보를 관리 및 제공할 수 있습니다.

Oracle Application Server 10g



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Application Server 10g

Oracle Application Server 10g는 J2EE 및 웹 서비스 런타임 환경, 기업 포탈, 기업 통합 broker, Business Intelligence, 웹 캐싱 및 ID 관리 서비스를 포함한 다양한 기능이 통합된 기업 응용 프로그램 개발 및 배포를 위한 완벽한 infrastructure 플랫폼을 제공합니다. Oracle Application Server 10g는 수많은 고객이 실제 환경에서 기업 응용 프로그램으로 운용하고 있는 Oracle9i Application Server에 새 그리드 컴퓨팅 기능을 추가한 것입니다.

Oracle Application Server 10g는 실행하려는 모든 서로 다른 서버 응용 프로그램에 대한 서비스를 포함하는 유일한 응용 프로그램 서버입니다. 다음 항목을 실행할 수 있습니다.

- 포탈 및 웹 사이트
- Java 트랜잭션 응용 프로그램
- Business Intelligence 응용 프로그램

또한 이 제품은 조직 전반에서 유저, 응용 프로그램 및 데이터 사이의 통합 기능을 제공합니다.

Oracle Enterprise Manager 10g Grid Control

- 소프트웨어 프로비저닝
- 응용 프로그램 서비스 레벨 모니터



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Enterprise Manager 10g Grid Control

Oracle Enterprise Manager 10g Grid Control은 그리드 환경에서 여러 시스템 사이의 관리 작업을 자동화하는 완벽한 통합 중앙 관리 콘솔이며 기반 프레임워크입니다. Grid Control을 사용하면 여러 하드웨어 노드, 데이터베이스, Application Server 및 기타 대상을 하나의 논리적 엔티티로 그룹화할 수 있습니다. 또한 업무 실행, 표준 정책 적용, 성능 진단 및 모니터, 기타 작업 자동화 등을 여러 시스템에서 개별적으로 수행하는 대신 대상 그룹에서 수행함으로써 늘어나는 그리드에 따라 성능을 확장할 수 있습니다.

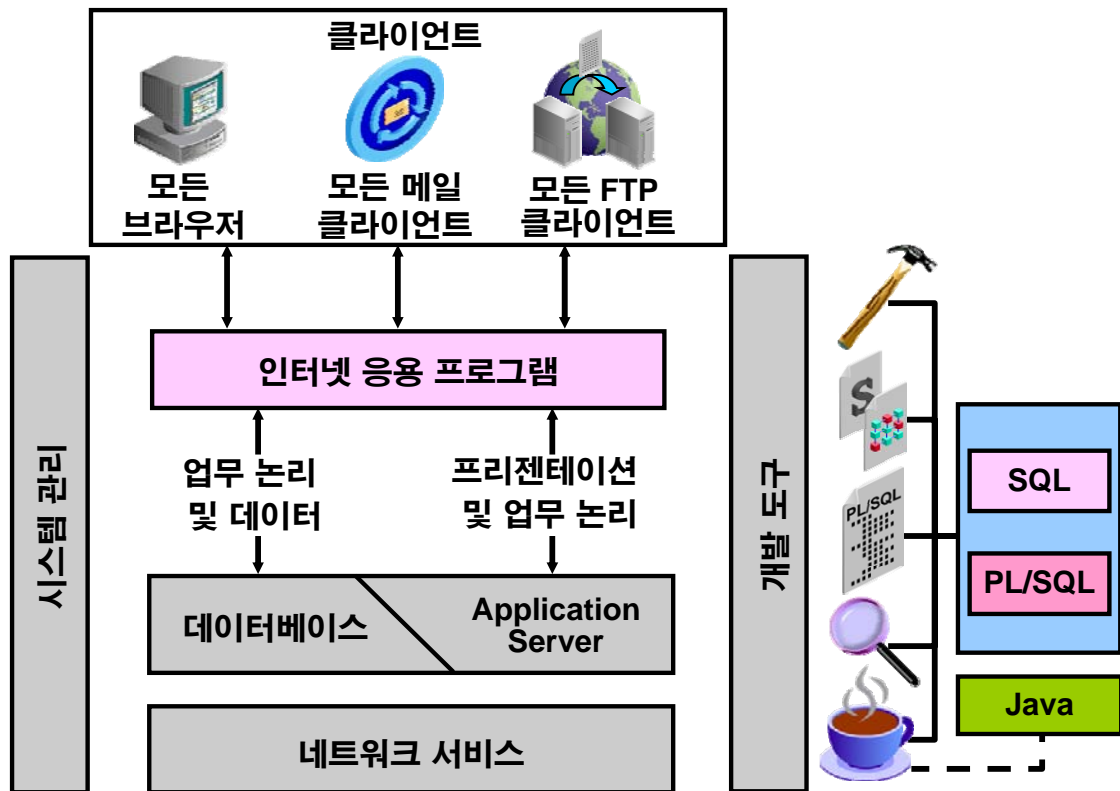
소프트웨어 프로비저닝

Oracle 10g 플랫폼은 Grid Control을 통해 여러 노드에서 Oracle Application Server 10g 및 Oracle Database 10g의 설치, 구성 및 복제를 자동화합니다. Oracle Enterprise Manager는 관리자가 필요에 따라 Application Server와 데이터베이스의 새 instance를 사용하여 새 서버를 생성, 구성, 배치 및 활용할 수 있도록 소프트웨어 프로비전 및 관리를 위한 공통 프레임워크를 제공합니다.

응용 프로그램 서비스 레벨 모니터

유저가 경험하는 것처럼 Grid Control은 그리드 infrastructure의 가용성과 성능을 저장 장치, 프로세싱 상자, 데이터베이스 및 Application Server처럼 분리된 대상이 아니라 하나로 통합된 대상으로 봅니다.

Oracle Internet Platform



Copyright © 2009, Oracle. All rights reserved.

Oracle Internet Platform

전자 상거래 응용 프로그램을 개발하려면 데이터를 저장 및 관리할 수 있고 업무 논리를 구현하는 응용 프로그램에 런타임 환경을 제공할 수 있고 응용 프로그램이 통합된 후에 모니터 및 진단할 수 있는 제품이 필요합니다. Oracle 10g 그리드 infrastructure 제품은 이전에 설명했듯이 기업 응용 프로그램 개발에 필요한 모든 구성 요소를 제공합니다. Oracle은 전자 상거래 및 데이터 웨어하우징을 위한 종합적인 고성능 인터넷 플랫폼을 제공합니다. 이러한 통합 플랫폼은 인터넷 응용 프로그램의 개발, 배치 및 관리에 필요한 모든 기능을 제공합니다.

Oracle Internet Platform은 다음의 세 가지 핵심 구성 요소 위에 구축되었습니다.

- 프리젠테이션을 처리하기 위한 브라우저 기반 클라이언트
- 업무 논리를 실행하고 브라우저 기반 클라이언트에 프리젠테이션 논리를 제공하기 위한 Application Server
- 데이터베이스 중심 업무 논리 및 서버 데이터를 실행하기 위한 데이터베이스

Oracle은 많은 영역의 업무 및 업종을 위한 대용량 소프트웨어 응용 프로그램 모음뿐 아니라 업무용 응용 프로그램을 구축하기 위한 광범위한 최고급 GUI(그래픽 사용자 인터페이스) 방식 개발 도구를 제공합니다. 내장 프로시저, 함수 및 패키지는 SQL, PL/SQL 또는 Java를 사용하여 작성할 수 있습니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 과정 목표 및 과정 내용 설명
- hr 스키마에서 테이블 및 테이블 사이의 관계 식별
- 완벽한 비즈니스 솔루션을 개발할 수 있는 Oracle 10g 그리드 infrastructure의 다양한 제품 식별

ORACLE

Copyright © 2009, Oracle. All rights reserved.

과정 연습

본 과정의 연습을 수행하면서 익명 블록을 사용하여 간단한 응용 프로그램을 개발하게 될 것입니다. 이 익명 블록에서는 다음 사항을 다룹니다.

- 선언 섹션 작성
- 스칼라 유형의 변수 선언
- %TYPE 속성을 사용하여 변수 선언
- 실행 섹션 작성
- 변수에 대한 유저 입력 받기
- 데이터베이스에서 값을 검색한 다음 INTO 절을 사용하여 변수에 값 저장
- 실행 섹션 내에서 중첩 블록 작성
- 실행 섹션에서 제어 구조를 사용하여 업무 논리 수행
- INDEX BY 테이블을 사용하여 값 저장 및 출력
- 예외 처리

연습 응용 프로그램의 기능

이 응용 프로그램은 간단한 HR 응용 프로그램이며 HR 부서에 근무하는 직원만 사용할 수 있습니다. employees 테이블에는 HR 부서에 한 명의 직원만 있습니다. 따라서 employee_id를 인증용으로 사용할 수 있습니다.

회사에서 이번 분기에 특정 부서의 직원 급여를 인상하기로 결정했습니다. 인상 비율은 직원의 현재 급여에 따라 결정됩니다.

다음 부서의 직원들은 이번 분기에 급여 인상 대상자입니다.

department_id	department_name
20	Marketing
60	IT
80	Sales
100	Finance
110	Accounting

급여 범위 및 결과 인상 비율은 다음과 같습니다.

salary	Raise percentage
< 6500	20
> 6500 < 9500	15
> 9500 < 12000	8
> 12000	3

1

PL/SQL 소개

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **PL/SQL의 필요성 설명**
- **PL/SQL의 이점 설명**
- **다양한 PL/SQL 블록 유형 식별**
- **Oracle SQL Developer를 PL/SQL의 개발 환경으로 사용**
- **PL/SQL에서 메시지 출력**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 PL/SQL 및 PL/SQL 프로그래밍 생성자를 소개하고 PL/SQL의 이점을 살펴봅니다. 또한 SQL Developer를 PL/SQL 개발 환경으로 사용하는 방법을 배웁니다.

PL/SQL이란?

PL/SQL:

- SQL을 확장한 절차적 언어 (Procedural Language)를 나타냅니다.
- 관계형 데이터베이스에서 사용되는 Oracle의 표준 데이터 액세스 언어입니다.
- 프로시저 생성자를 SQL과 완벽하게 통합합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL이란?

SQL(Structured Query Language)은 관계형 데이터베이스에서 데이터에 액세스하거나 데이터를 수정하는 데 사용되는 기본 언어입니다. 손쉽게 배우고 편리하게 사용할 수 있는 몇 가지 SQL 명령이 있습니다. 다음 예제를 살펴보십시오.

```
SELECT first_name, department_id, salary FROM employees;
```

위의 SQL 문은 간단하면서도 직관적입니다. 그러나 SQL에서 검색 데이터를 조건부로 변경하려면 제한이 따릅니다. 문제의 명령문을 약간 수정해 보겠습니다. 검색한 모든 사원에 대해 department_id와 salary를 확인합니다. 부서의 성과와 사원의 급여 수준에 따라 커미션을 차등 지급해야 하는 경우가 있습니다.

문제를 살펴보면 위의 SQL 문을 실행하고 데이터를 수집하며 데이터에 논리를 적용해야 한다는 사실을 알 수 있습니다. 하나의 해결 방법은 해당 부서의 사원에게 상여금을 주는 부서마다 SQL 문을 작성하는 것입니다. 상여금 액수를 결정하기 전에 급여 구성 요소를 확인해야 합니다.

이렇게 하려면 과정이 약간 복잡해집니다. 조건문을 사용하면 구문 작성이 훨씬 쉬워질 것입니다. PL/SQL은 이러한 요구 사항을 충족하도록 설계되었습니다. PL/SQL은 기존 SQL에 프로그래밍 확장 기능을 제공합니다.

PL/SQL 정보

PL/SQL:

- 코드 실행 단위에 블록 구조를 제공합니다. 잘 정의된 구조로 코드 유지 관리가 쉽습니다.
- 다음과 같은 프로시저 생성자를 제공합니다.
 - 변수, 상수 및 유형
 - 조건문 및 루프와 같은 제어 구조
 - 한 번 작성하면 여러 번 실행할 수 있는 재사용 가능한 프로그램 단위

ORACLE

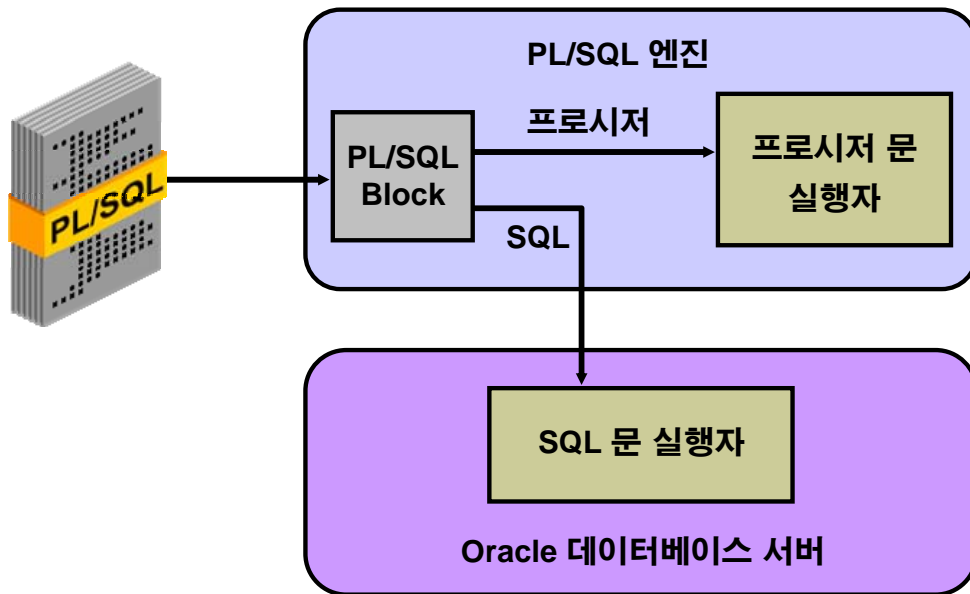
Copyright © 2009, Oracle. All rights reserved.

PL/SQL 정보

PL/SQL은 코드 작성을 위한 블록 구조를 정의합니다. 이러한 구조를 사용하면 코드 유지 관리 및 디버깅이 더욱 쉬워집니다. 프로그램 단위의 흐름과 실행을 쉽게 이해할 수 있습니다.

PL/SQL은 데이터 캡슐화, 예외 처리, 정보 숨기기, 객체 지향과 같은 최신 소프트웨어 엔지니어링 기능을 제공하며 Oracle 서버 및 툴세트에 최신 프로그래밍 기술을 접목시킵니다. PL/SQL은 3세대 언어(3GL)에서 사용할 수 있는 프로시저 생성자를 모두 제공합니다.

PL/SQL 환경



ORACLE

Copyright © 2009, Oracle. All rights reserved.

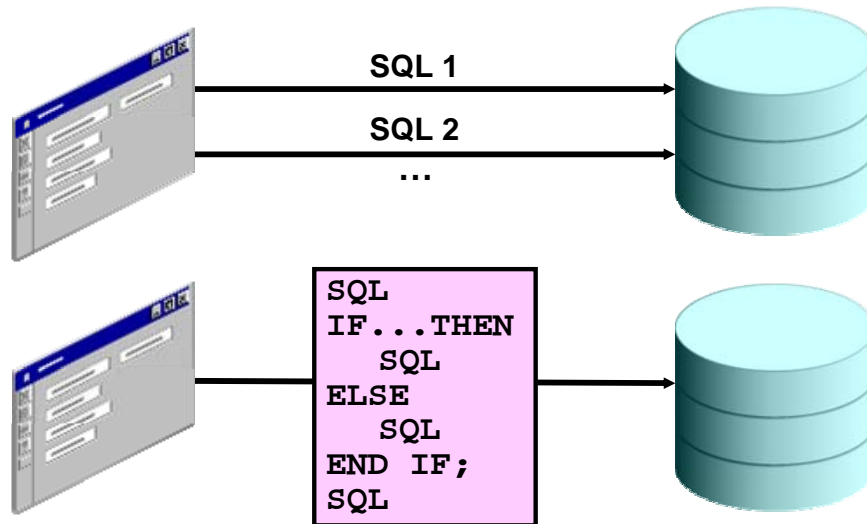
PL/SQL 환경

이 슬라이드는 Oracle 데이터베이스 서버에서의 PL/SQL 실행 환경을 보여줍니다. PL/SQL 블록에는 프로시저문과 SQL 문이 포함됩니다. PL/SQL 블록을 서버로 전송하면 먼저 PL/SQL 엔진이 해당 블록을 구문 분석합니다. PL/SQL 엔진은 프로시저문과 SQL 문을 식별합니다. 프로시저문은 프로시저문 실행자에, SQL 문은 SQL 문 실행자에 각각 전달합니다.

슬라이드의 도표는 데이터베이스 서버 내의 PL/SQL 엔진을 보여줍니다. Oracle 응용 프로그램 개발 툴은 PL/SQL 엔진도 포함할 수 있습니다. 이 툴은 블록을 해당하는 로컬 PL/SQL 엔진으로 전달합니다. 따라서 모든 프로시저문은 로컬로 실행되고 SQL 문만 데이터베이스에서 실행됩니다. 사용되는 엔진은 PL/SQL 블록이 호출되는 위치에 따라 다릅니다.

PL/SQL의 이점

- 프로시저 생성자와 SQL의 통합
- 성능 향상



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 이점

프로시저 생성자와 SQL의 통합: PL/SQL의 최대 이점은 프로시저 생성자와 SQL의 통합에 있습니다. SQL은 비절차적 언어입니다. SQL 명령을 실행하면 데이터베이스 서버가 수행할 작업을 알려줍니다. 그러나 수행하는 방법은 지정할 수 없습니다. PL/SQL은 제어문과 조건문을 SQL과 통합하여 SQL 문과 해당 명령문의 실행을 보다 잘 제어할 수 있도록 합니다. 이 단원 앞부분에서 그러한 통합의 필요성을 보여주는 예제를 살펴보았습니다.

성능 향상: PL/SQL을 사용하지 않을 경우 논리적으로 SQL 문을 한 단위로 결합할 수 없습니다. Form이 포함된 응용 프로그램을 설계하는 경우 각 Form에 필드가 있는 다양한 Form을 가질 수 있습니다. Form이 데이터를 전송하는 경우 많은 SQL 문을 실행해야 합니다. SQL 문은 한 번에 하나씩 데이터베이스로 보내집니다. 따라서 네트워크 이동이 자주 발생하고 SQL 문마다 데이터베이스를 한 번씩 호출하게 되어 (특히 클라이언트/서버 모델에서) 네트워크 트래픽은 증가하고 성능은 감소합니다.

PL/SQL을 사용하면 이러한 모든 SQL 문을 단일 프로그램 단위로 결합할 수 있습니다. 이 응용 프로그램은 SQL 문을 한 번에 하나씩 데이터베이스로 보내는 대신 전체 블록을 보낼 수 있습니다. 따라서 데이터베이스 호출 횟수가 상당히 줄어듭니다. 슬라이드에서 볼 수 있는 것처럼 SQL 중심 응용 프로그램인 경우 PL/SQL 블록을 사용하여 SQL 문을 Oracle 데이터베이스 서버로 보내 실행하기 전에 그룹화할 수 있습니다.

PL/SQL의 이점

- 모듈식 프로그램 개발
- Oracle 툴과의 통합
- 이식성
- 예외 처리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 이점(계속)

모듈식 프로그램 개발: 모든 PL/SQL 프로그램의 기본 단위는 블록입니다. 블록은 순차적으로 배치되거나 다른 블록에 중첩될 수 있습니다. 모듈식 프로그램 개발은 다음과 같은 장점이 있습니다.

- 블록 내의 관련 명령문을 논리적으로 그룹화할 수 있습니다.
- 블록을 더 큰 블록 내부에 중첩하여 강력한 프로그램을 작성할 수 있습니다.
- 응용 프로그램은 더 작은 모듈로 분리할 수 있습니다. 복잡한 응용 프로그램을 설계하는 경우 PL/SQL을 사용하면 해당 응용 프로그램을 더욱 작고 관리하기 쉬우며 논리적으로 관련된 모듈로 분리할 수 있습니다.
- 코드를 쉽게 유지 관리 및 디버깅할 수 있습니다.

툴과의 통합: PL/SQL 엔진은 Oracle Forms, Oracle Reports 등과 같은 Oracle 툴에 통합됩니다. 이러한 툴을 사용할 경우 논리적으로 사용할 수 있는 PL/SQL 엔진이 프로시저문을 처리하고 SQL 문만 데이터베이스에 전달됩니다.

PL/SQL의 이점(계속)

이식성: PL/SQL 프로그램은 운영 체제나 플랫폼에 상관없이 Oracle 서버가 실행되는 모든 환경에서 실행할 수 있습니다. 매번 새로운 환경에 맞게 변경할 필요가 없습니다. 이식 가능한 프로그램 패키지를 작성하고 서로 다른 환경에서 재사용 가능한 라이브러리를 만들 수 있습니다.

예외 처리: PL/SQL을 사용하면 예외를 효율적으로 처리할 수 있습니다. 예외 처리를 위한 별도의 블록을 정의할 수 있습니다. 이 과정 뒷부분에서 예외 처리에 대해 배우게 됩니다.

PL/SQL과 SQL은 데이터 유형 체계가 동일하고(일부 확장 기능 포함) 동일한 표현식 구문을 사용합니다.

PL/SQL 블록 구조

- **DECLARE (선택 사항)**
 - 변수, 커서, 유저 정의 예외
- **BEGIN (필수)**
 - SQL 문
 - PL/SQL 문
- **EXCEPTION (선택 사항)**
 - 오류 발생 시 수행할 작업
- **END; (필수)**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록 구조

이 슬라이드는 기본 PL/SQL 블록을 보여줍니다. PL/SQL 블록은 다음 세 개의 섹션으로 구성됩니다.

- **선언(선택 사항):** 선언 섹션은 DECLARE 키워드로 시작하며 실행 섹션 시작 부분에서 끝납니다.
- **실행(필수):** 실행 섹션은 BEGIN 키워드로 시작되며 END로 끝납니다. END는 세미콜론으로 종료됩니다. PL/SQL 블록의 실행 섹션은 PL/SQL 블록을 무제한 포함할 수 있습니다.
- **예외 처리(선택 사항):** 예외 섹션은 실행 섹션 내에 중첩됩니다. 이 섹션은 EXCEPTION 키워드로 시작됩니다.

PL/SQL 블록에서 DECLARE, BEGIN 및 EXCEPTION 키워드는 세미콜론으로 종료되지 않습니다. 그러나 END 키워드, 모든 SQL 문 및 PL/SQL 문은 세미콜론으로 종료해야 합니다.

섹션	설명	포함
선언(DECLARE)	실행 및 예외 섹션에서 참조하는 모든 변수, 상수, 커서 및 유저 정의 예외 선언을 포함합니다.	선택
실행 (BEGIN ... END)	데이터베이스에서 데이터를 검색하는 SQL 문과 블록에서 데이터를 조작하는 PL/SQL 문을 포함합니다.	필수
예외(EXCEPTION)	실행 섹션에서 오류 또는 비정상적인 상황이 발생할 경우에 수행할 조치를 지정합니다.	선택

블록 유형

익명	프로시저	함수
<pre>[DECLARE] BEGIN --statements [EXCEPTION] END;</pre>	<pre>PROCEDURE name IS BEGIN --statements [EXCEPTION] END;</pre>	<pre>FUNCTION name RETURN datatype IS BEGIN --statements RETURN value; [EXCEPTION] END;</pre>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

블록 유형

PL/SQL 프로그램은 하나 이상의 블록으로 구성됩니다. 이러한 블록은 완전히 별개이거나 다른 블록 내에 중첩될 수 있습니다. PL/SQL 프로그램을 구성하는 블록 유형 세 가지는 다음과 같습니다.

- 익명 블록
- 프로시저
- 함수

익명 블록: 익명 블록은 이름이 지정되지 않은 블록입니다. 응용 프로그램에서 해당 블록을 실행할 지점에 인라인으로 선언되고 응용 프로그램이 실행될 때마다 컴파일됩니다. 이러한 블록은 데이터베이스에 저장되지 않습니다. 런타임에 PL/SQL 엔진으로 전달되어 실행됩니다.

Oracle Developer 구성 요소의 트리거는 이러한 블록으로 구성됩니다. 이러한 익명 블록은 인라인 블록이기 때문에 런타임에 실행됩니다. 동일한 블록을 다시 실행하려면 해당 블록을 다시 작성해야 합니다. 이전에 작성한 블록은 익명이며 실행된 후에는 존재하지 않기 때문에 실행하거나 호출할 수 없습니다.

블록 유형(계속)

서브 프로그램: 서브 프로그램은 익명 블록을 보완합니다. 서브 프로그램은 데이터베이스에 저장되어 있는 명명된 PL/SQL 블록입니다. 이름이 지정되고 저장되기 때문에 응용 프로그램에 따라 필요한 경우 언제든지 호출할 수 있습니다. 서브 프로그램은 프로시저나 함수로 선언할 수 있습니다. 일반적으로 작업을 수행할 때는 프로시저를, 값을 계산하고 반환할 때는 함수를 사용합니다.

서브 프로그램은 서버나 응용 프로그램 레벨에서 저장할 수 있습니다. Oracle Developer 구성 요소(Forms, Reports)를 사용하면 프로시저와 함수를 응용 프로그램(Form 또는 보고서)의 일부로 선언하고 필요한 경우 언제든지 동일한 응용 프로그램 내의 다른 프로시저, 함수 및 트리거에서 호출할 수 있습니다.

주: 함수는 값을 반환해야 한다는 점을 제외하고 프로시저와 유사합니다.

프로그램 생성자



툴 생성자
익명 블록
응용 프로그램 프로시저 또는 함수
응용 프로그램 패키지
응용 프로그램 트리거
객체 유형

데이터베이스 서버 생성자
익명 블록
내장 프로시저 또는 함수
내장 패키지
데이터베이스 트리거
객체 유형

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램 생성자

다음 표에 기본 PL/SQL 블록을 사용하는 다양한 PL/SQL 프로그램 생성자가 요약되어 있습니다. 사용 가능한 프로그램 생성자는 실행 환경에 따라 다릅니다.

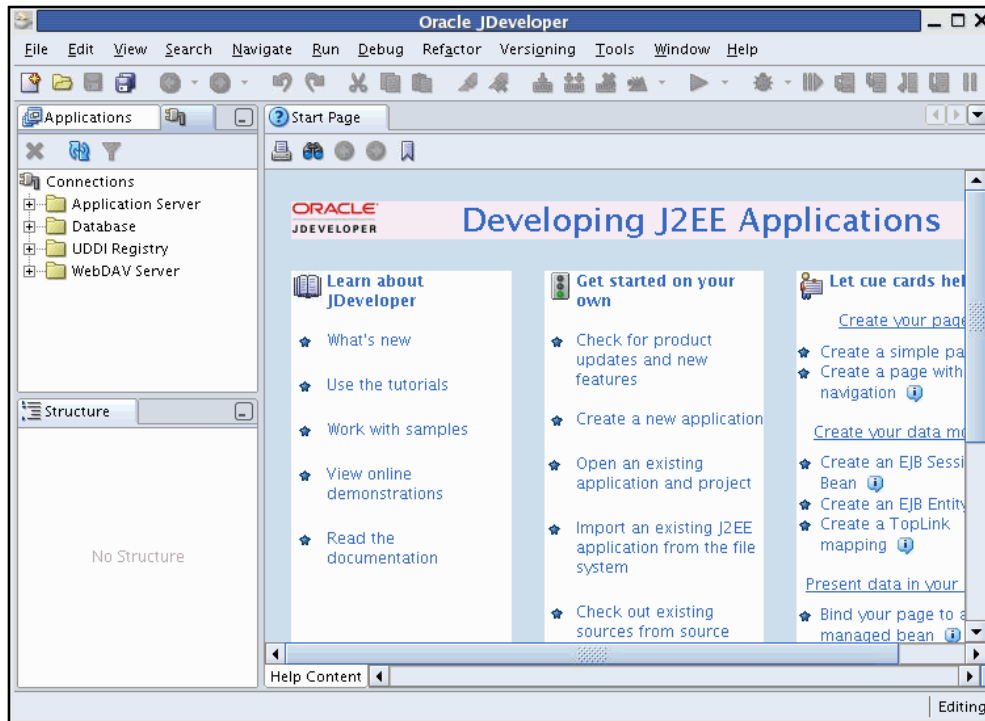
프로그램 생성자	설명	가용성
익명 블록	응용 프로그램에 포함되거나 대화식으로 실행되는 이름이 지정되지 않은 PL/SQL 블록	모든 PL/SQL 환경
응용 프로그램 프로시저 또는 함수	Oracle Forms Developer 응용 프로그램이나 공유 라이브러리에 저장된 명명된 PL/SQL 블록으로, 파라미터를 사용하고 이름으로 반복적으로 호출할 수 있습니다.	Oracle Developer 툴 구성 요소(예: Oracle Forms Developer, Oracle Reports)
내장 프로시저 또는 함수	Oracle 서버에 저장된 명명된 PL/SQL 블록이며 파라미터를 받고 이름으로 반복적으로 호출할 수 있습니다.	Oracle 서버 또는 Oracle Developer 툴
패키지 (응용 프로그램 또는 내장 패키지)	관련 프로시저, 함수 및 식별자를 그룹화한 명명된 PL/SQL 모듈	Oracle 서버 및 Oracle Developer 툴 구성 요소(예: Oracle Forms Developer)

프로그램 생성자(계속)

프로그램 생성자	설명	가용성
데이터베이스 트리거	데이터베이스 테이블과 연관되어 다양한 이벤트에 의해 트리거될 때 자동으로 실행되는 PL/SQL 블록	Oracle 서버 또는 DML을 실행하는 Oracle 툴
응용 프로그램 트리거	데이터베이스 테이블이나 시스템 이벤트와 연관된 PL/SQL 블록. 각각 DML이나 시스템 이벤트에 의해 트리거될 때 자동으로 실행됩니다.	Oracle Developer 툴 구성 요소 (예: Oracle Forms Developer)
객체 유형	데이터를 조작하는 데 필요한 함수 및 프로시저와 함께 데이터 구조를 캡슐화하는 유저 정의 조합 데이터 유형	Oracle 서버 및 Oracle Developer 툴

PL/SQL 프로그래밍 환경

Oracle JDeveloper



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 프로그래밍 환경

Oracle JDeveloper: J2EE 응용 프로그램, 웹 서비스 및 PL/SQL의 작성, 테스트 및 배치 과정을 모두 지원하는 통합 개발 환경입니다.

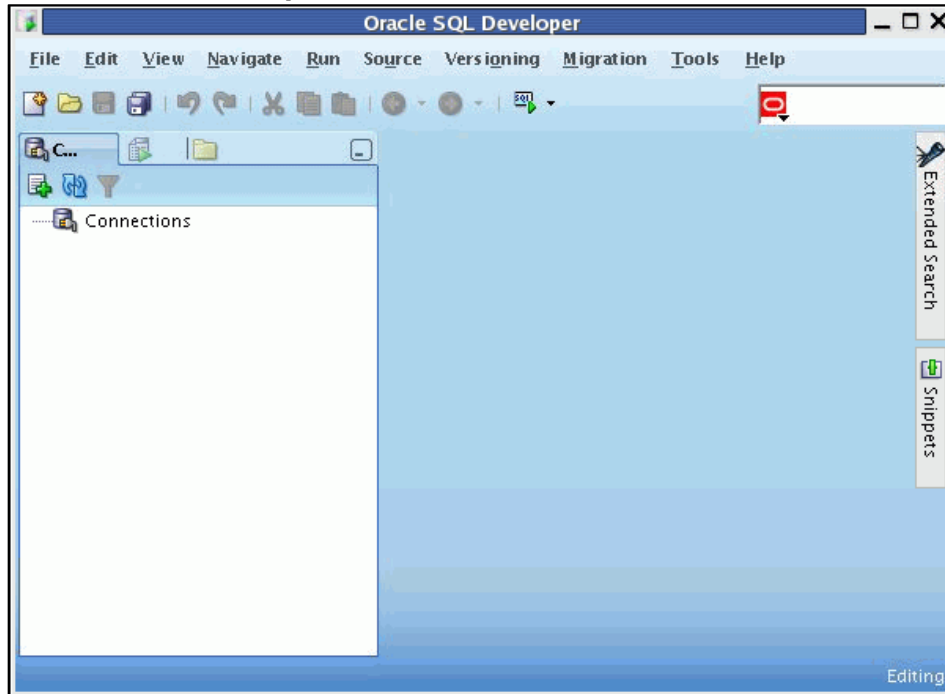
Oracle JDeveloper를 사용하여 다음을 수행할 수 있습니다.

- 유저에게 친숙한 마법사를 사용하여 데이터베이스 연결 설정
- 연결된 데이터베이스에서 객체 검색
- 데이터베이스 유저 및 객체 생성
- 프로시저, 함수 및 패키지과 같은 PL/SQL 프로그램 생성, 실행 및 디버그

주: Oracle JDeveloper 및 Oracle SQL Developer를 모두 프로그래밍 환경으로 사용할 수 있습니다. 그러나 이 과정에서는 모든 데모 및 연습에서 Oracle SQL Developer를 사용합니다.

PL/SQL 프로그래밍 환경

Oracle SQL Developer



ORACLE

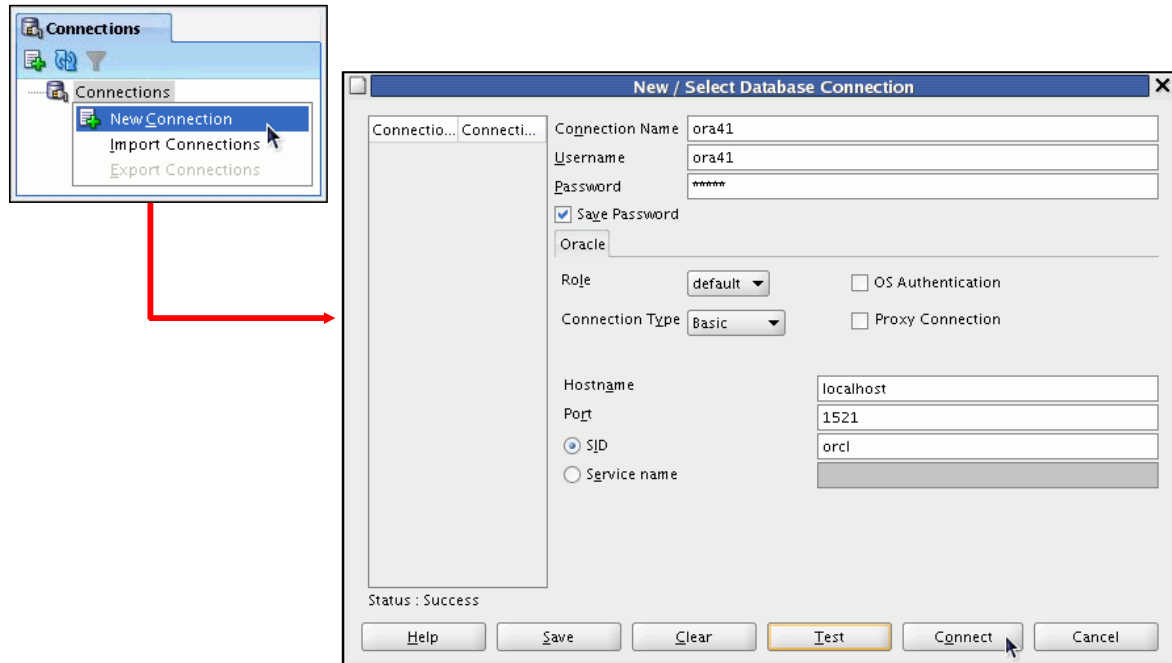
Copyright © 2009, Oracle. All rights reserved.

PL/SQL 프로그래밍 환경(계속)

Oracle SQL Developer: 이 툴은 Java로 개발되었으며 Windows, Linux 및 Mac OS(운영 체제) X 플랫폼에서 실행됩니다. 데이터베이스 서버에 SQL Developer를 설치하고 데스크톱에서 원격으로 연결할 수 있으므로 클라이언트/서버 네트워크 통신량이 발생하지 않습니다.

데이터베이스에 대한 기본 연결은 JDBC(Java Database Connectivity) Thin 드라이버를 통해 이루어지므로 Oracle 홈이 필요 없습니다. SQL Developer에는 Installer가 필요 없으며 다운로드한 파일의 압축을 풀기만 하면 됩니다.

데이터베이스 연결 생성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 연결 생성

SQL Developer를 사용하려면 하나 이상의 데이터베이스 연결이 있어야 합니다.

데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. `<your_path>\sqldeveloper\sqldeveloper.exe`를 두 번 누릅니다.
2. **Connections** 탭 페이지에서 **Connections**를 마우스 오른쪽 버튼으로 누르고 **New Connection**을 선택합니다.
3. 연결할 데이터베이스의 연결 이름, 유저 이름, 암호, 호스트 이름, 포트 번호 및 SID를 입력합니다.
4. **Test**를 눌러 연결이 올바르게 설정되었는지 확인합니다.
5. **Connect**를 누릅니다.

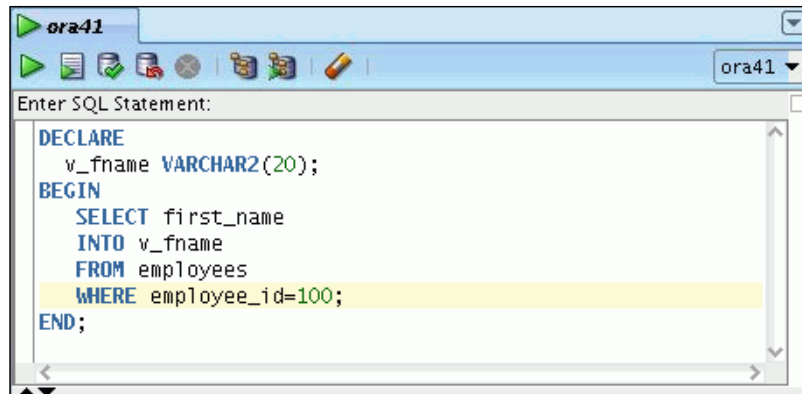
주: Save Password 체크 박스를 선택하면 암호가 XML 파일에 저장됩니다. 따라서 SQL Developer 연결을 닫았다가 다시 열었을 때 암호를 입력하라는 메시지가 표시되지 않습니다.

데이터베이스 연결을 생성한 후에 해당 연결에 대한 SQL Worksheet가 자동으로 열립니다.

Oracle SQL Developer에 대한 자세한 내용은 부록 E "SQL Developer 사용" 을 참조하십시오.

익명 블록 생성

SQL Developer 작업 영역에 익명 블록 입력:



ORACLE

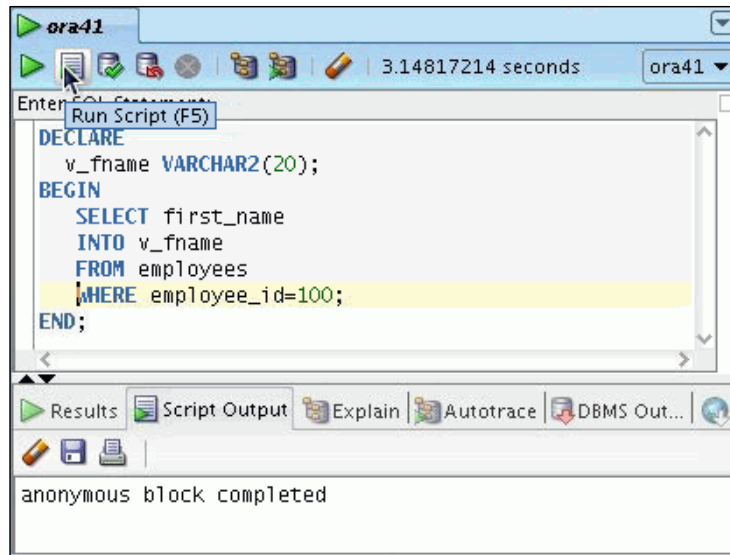
Copyright © 2009, Oracle. All rights reserved.

익명 블록 생성

SQL Developer를 사용하여 익명 블록을 생성하려면 슬라이드에 표시된 것처럼 SQL Worksheet에 블록을 입력하십시오. 블록은 선언 섹션과 실행 섹션을 가지고 있습니다. 블록에 있는 명령문의 구문은 신경 쓰지 않아도 됩니다. 구문은 본 과정 뒷부분에서 배우게 됩니다. 익명 블록은 employee_id가 100인 사원의 first_name을 가져와서 v_fname이라는 변수에 저장합니다.

익명 블록 실행

Run Script 버튼을 눌러 익명 블록 실행:



ORACLE

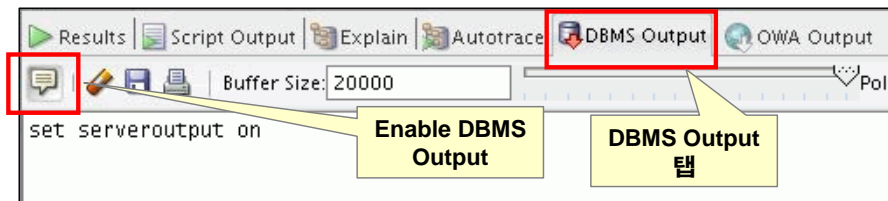
Copyright © 2009, Oracle. All rights reserved.

익명 블록 실행

Run Script 버튼을 눌러 작업 영역에 있는 익명 블록을 실행하십시오. 블록이 실행된 후 Script Output window에 "anonymous block completed"라는 메시지가 표시됩니다.

PL/SQL 블록 출력 테스트

- DBMS Output 탭의 Enable DBMS Output 버튼을 눌러 SQL Developer에서 출력 활성화:



- 미리 정의된 Oracle 패키지와 프로시저 사용:
 - DBMS_OUTPUT.PUT_LINE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

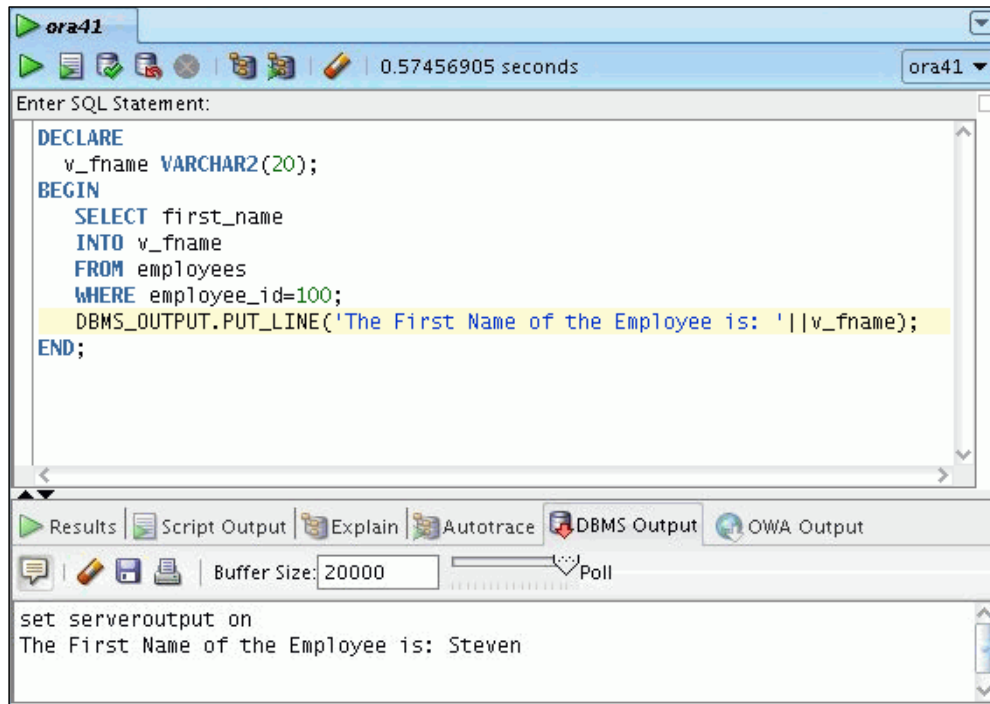
PL/SQL 블록 출력 테스트

이전 슬라이드의 예제에서 값이 `v_fname` 변수에 저장되었지만 출력되지는 않았습니다. 여기서는 값을 출력하는 방법을 설명합니다.

PL/SQL에는 입출력 기능이 내장되어 있지 않습니다. 따라서 입출력용으로 미리 정의된 Oracle 패키지를 사용해야 합니다. 출력을 생성하려면 다음과 같이 해야 합니다.

- DBMS Output 탭의 Enable Output 버튼을 눌러 SQL Developer에서 출력을 활성화합니다. 그러면 window에 표시되는 `SET SERVEROUTPUT ON` 명령이 실행됩니다. SQL*Plus에서 출력을 활성화하려면 `SET SERVEROUTPUT ON` 명령을 명시적으로 실행해야 합니다.
주: `SET SERVEROUTPUT ON` SQL*Plus 명령을 계속 사용하고 Script 탭에서 출력을 볼 수 있습니다. 이 과정의 코드 예제는 `SET SERVEROUTPUT ON` SQL*Plus 명령을 사용합니다.
- DBMS_OUTPUT 패키지의 `PUT_LINE` 프로시저를 사용하여 출력 결과를 표시합니다. 슬라이드에 표시된 것처럼 출력해야 할 값을 이 프로시저에 인수로 전달합니다. 그러면 이 프로시저가 인수를 출력합니다.

PL/SQL 블록 출력 테스트



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록 출력 테스트(계속)

슬라이드는 출력을 생성하는 코드를 포함시킨 후 PL/SQL 블록의 출력 결과를 보여줍니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- SQL 문과 PL/SQL 프로그램 생성자의 통합
- PL/SQL의 이점 파악
- 서로 다른 PL/SQL 블록 유형 구분
- Oracle SQL Developer를 PL/SQL의 프로그래밍 환경으로 사용
- PL/SQL에서 메시지 출력

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

PL/SQL은 SQL의 확장 역할을 하는 프로그래밍 기능을 갖춘 언어입니다. PL/SQL 프로그래밍 생성자는 비절차적 언어인 SQL을 절차적 언어로 만듭니다. PL/SQL 응용 프로그램은 Oracle 서버가 실행되는 모든 플랫폼이나 운영 체제에서 실행할 수 있습니다. 이 단원에서는 기본 PL/SQL 블록을 작성하는 방법을 배웠습니다.

연습 1: 개요

이 연습에서는 다음 내용을 다룹니다.

- 성공적으로 실행되는 PL/SQL 블록 식별
- 간단한 PL/SQL 블록 생성 및 실행

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 1: 개요

이 연습에서는 이 단원에서 다룬 PL/SQL의 기본 사항을 다룹니다.

- 1번 문제는 성공적으로 실행되는 PL/SQL 블록을 식별하기 위한 객관식 문제입니다.
- 2번 문제에는 간단한 PL/SQL 블록 생성 및 실행이 포함됩니다.

연습 1

labs 폴더가 작업 디렉토리입니다. labs 폴더에 스크립트를 저장할 수 있습니다. 강사에게 문의하여 본 과정에 사용할 labs 폴더를 찾으십시오. 모든 연습의 해답은 soln 폴더에 있습니다.

1. 다음 PL/SQL 블록 중 성공적으로 실행되는 블록은 무엇입니까?

- a.

```
BEGIN
END;
```
- b.

```
DECLARE
amount INTEGER(10);
END;
```
- c.

```
DECLARE
BEGIN
END;
```
- d.

```
DECLARE
amount INTEGER(10);
BEGIN
    DBMS_OUTPUT.PUT_LINE(amount);
END;
```

2. 다음 정보를 사용하여 SQL Developer에서 데이터베이스 연결을 생성합니다.

Connection Name: ora41
Username: ora41
Password: ora41
Hostname: localhost
Port: 1521
SID: orcl

3. "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다. 이 스크립트를 실행하여 lab_01_03_soln.sql로 저장합니다.

2

PL/SQL 변수 선언

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 적합한 식별자와 부적합한 식별자 구분
- 변수 사용 나열
- 변수 선언 및 초기화
- 다양한 데이터 유형 나열 및 설명
- %TYPE 속성 사용 시의 이점 파악
- 바인드 변수 선언, 사용 및 출력

ORACLE

Copyright © 2009, Oracle. All rights reserved.

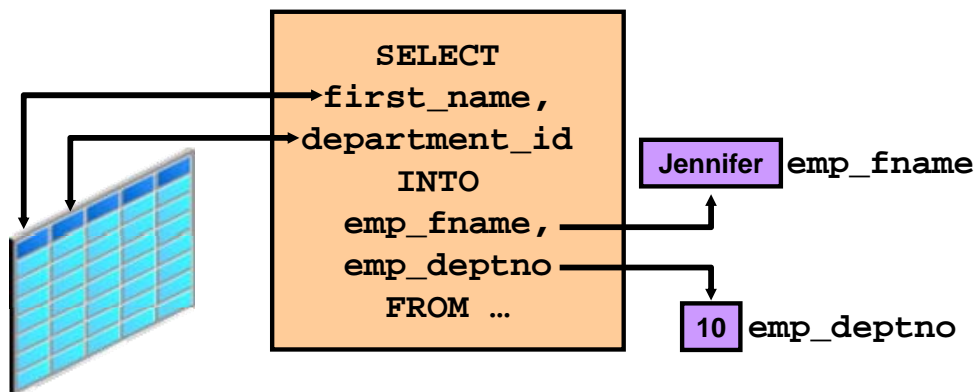
단원 목적

이미 기본 PL/SQL 블록 및 섹션에 대해 알아보았습니다. 이 단원에서는 유효한 식별자와 부적합한 식별자에 대해 배웁니다. PL/SQL 블록의 선언 섹션에서 변수를 선언하고 초기화하는 방법을 배웁니다. 이 단원에서는 다양한 데이터 유형을 설명합니다. %TYPE 속성 및 이점에 대해서도 알아봅니다.

변수 사용

변수는 다음 용도로 사용할 수 있습니다.

- 데이터의 임시 저장 영역
- 저장된 값 조작
- 재사용성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 사용

PL/SQL에서 변수를 선언하고 SQL 및 프로시저문에서 해당 변수를 사용할 수 있습니다.

변수는 주로 데이터 저장 및 저장된 값의 조작에 사용됩니다. 슬라이드의 SQL 문을 살펴보십시오. 이 명령문은 테이블에서 first_name과 department_id를 검색합니다. first_name이나 department_id를 조작해야 하는 경우 검색한 값을 저장해야 합니다. 변수는 값을 임시로 저장하는 데 사용됩니다. 이러한 변수에 저장된 값을 사용하여 데이터를 처리 및 조작할 수 있습니다. 변수는 모든 PL/SQL 객체(변수, 유형, 커서, 서브 프로그램 등)를 저장할 수 있습니다.

재사용성은 변수 선언 시의 또 다른 장점입니다. 변수를 선언한 후에 명령문에서 해당 변수를 참조하는 방식으로 응용 프로그램에서 반복적으로 사용할 수 있습니다.

식별자

식별자는 다음 용도로 사용됩니다.

- 변수 이름 지정
- 변수 이름 지정 규칙 제공
 - 문자로 시작해야 합니다.
 - 문자나 숫자를 포함할 수 있습니다.
 - 달러 기호, 밑줄, 파운드 기호 등과 같은 특수 문자를 포함할 수 있습니다.
 - 길이는 30자로 제한됩니다.
 - 예약어를 사용할 수 없습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

식별자

식별자는 주로 변수 이름 지정 규칙을 제공하는 데 사용됩니다. 슬라이드에 변수 이름 지정 규칙이 나열되어 있습니다.

변수와 식별자의 차이점

식별자는 변수의 이름입니다. 변수는 데이터 저장 위치입니다. 데이터는 메모리에 저장됩니다. 변수는 데이터를 읽어들이고 수정할 수 있는 메모리 위치를 가리킵니다. 식별자는 모든 PL/SQL 객체(변수, 유형, 커서, 서브 프로그램 등)의 이름을 지정하는 데 사용됩니다. 변수는 PL/SQL 객체를 저장하는 데 사용됩니다.

PL/SQL에서 변수 처리

변수는 다음과 같이 처리됩니다.

- 선언 섹션에서 선언 및 초기화됨
- 실행 섹션에서 사용되고 새 값이 할당됨
- PL/SQL 서브 프로그램에 파라미터로 전달됨
- PL/SQL 서브 프로그램의 출력 결과를 보유하는 데 사용됨

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에서 변수 처리

선언 섹션에서 선언 및 초기화됨

임의 PL/SQL 블록, 서브 프로그램 또는 패키지의 선언 부분에 변수를 선언할 수 있습니다. 선언은 값에 대한 저장 공간을 할당하고, 해당 데이터 유형을 지정하고, 참조할 수 있도록 저장 위치를 명명합니다. 선언은 또한 초기 값을 지정하고 변수에 NOT NULL 제약 조건을 지정할 수 있습니다. 전방 참조는 허용되지 않습니다. 다른 선언문을 포함하여 다른 명령문에서 참조하기 전에 변수를 선언해야 합니다.

실행 섹션에서 사용되고 새 값이 할당됨

실행 섹션에서 변수의 기존 값을 새 값으로 대체할 수 있습니다.

PL/SQL 서브 프로그램에 파라미터로 전달됨

서브 프로그램은 파라미터를 가질 수 있습니다. 서브 프로그램에 파라미터로 변수를 전달할 수 있습니다.

PL/SQL 서브 프로그램의 출력을 보유하는 데 사용됨

프로시저와 함수의 유일한 차이점으로 함수는 값을 반환해야 한다고 배웠습니다. 변수는 함수가 반환한 값을 보유하는 데 사용될 수 있습니다.

PL/SQL 변수 선언 및 초기화

구문

```
identifier [CONSTANT] datatype [NOT NULL]
[ := | DEFAULT expr ];
```

예제

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno      NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언 및 초기화

PL/SQL 블록에서 모든 PL/SQL 식별자를 참조하려면 먼저 선언 섹션에서 이들을 선언해야 합니다. 슬라이드에 표시된 것처럼 변수에 초기 값을 할당하는 옵션이 있습니다. 선언하려는 변수에는 값을 할당할 필요가 없습니다. 선언 섹션에서 다른 변수를 참조할 경우 이 변수가 이전 명령문에 별도로 선언되어 있어야 합니다.

이 구문에서 다음이 적용됩니다.

identifier 변수 이름입니다.

CONSTANT 변수 값을 변경할 수 없도록 변수에 제약 조건을 지정합니다. 상수는 초기화되어야 합니다.

data type 스칼라, 조합, 참조 또는 LOB 데이터 유형입니다. 본 과정에서는 스칼라, 조합 및 LOB 데이터 유형만 다룹니다.

NOT NULL 변수가 반드시 값을 포함하도록 변수에 제약 조건을 지정합니다. NOT NULL 변수는 초기화되어야 합니다.

expr 임의의 PL/SQL 식으로, 리터럴 표현식, 다른 변수, 연산자와 함수를 포함하는 표현식일 수 있습니다.

주: 선언 섹션에서 변수와 함께 커서와 예외를 선언할 수도 있습니다. 이 과정 뒷부분에서 커서와 예외를 선언하는 방법을 배우게 됩니다.

PL/SQL 변수 선언 및 초기화

1

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: ' || Myname);
  Myname := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || Myname);
END;
/
```

2

```
SET SERVEROUTPUT ON
DECLARE
  Myname VARCHAR2(20) := 'John';
BEGIN
  Myname := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || Myname);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언 및 초기화(계속)

슬라이드의 두 코드 블록을 살펴보세요.

1. 블록의 선언 섹션에서 Myname 변수가 선언됩니다. 이 변수는 동일한 블록의 실행 섹션에서 액세스할 수 있습니다. 실행 섹션에서 John 값이 변수에 할당됩니다. 문자열 리터럴은 작은 따옴표로 묶어야 합니다. "Today's Date"처럼 문자열이 큰 따옴표로 묶인 경우 "Today's Date"로 나타냅니다. ':= '는 할당 연산자입니다. Myname 변수를 전달하여 PUT_LINE 프로시저가 호출됩니다. 'My name is: ' 문자열에 변수의 값이 연결됩니다. 이 익명 블록의 출력 결과는 다음과 같습니다.

```
anonymous block completed
My name is:
My name is: John
```

2. 두번째 블록에서 변수 Myname이 선언 섹션에서 선언되고 초기화됩니다. Myname은 초기화 후에 값 John을 보유합니다. 이 값은 블록의 실행 섹션에서 조작됩니다. 이 익명 블록의 출력 결과는 다음과 같습니다.

```
anonymous block completed
My name is: Steven
```

문자열 리터럴의 구분자

```
SET SERVEROUTPUT ON
DECLARE
    event VARCHAR2(15);
BEGIN
    event := q'!Father's day!';
    DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is :
    ' || event);
    event := q'[Mother's day]';
    DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is :
    ' || event);
END;
/
```

```
anonymous block completed
3rd Sunday in June is : Father's day
2nd Sunday in May is : Mother's day
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

문자열 리터럴의 구분자

문자열이 아포스트로피(작은 따옴표와 같음)를 포함할 경우 다음 예제와 같이 두 개의 작은 따옴표를 사용해야 합니다.

```
event VARCHAR2(15):='Father' 's day' ;
```

첫번째 따옴표는 이스케이프 문자로 사용됩니다. 이와 같은 경우, 특히 문자열로 SQL 문을 가지고 있으면 문자열이 복잡해질 수 있습니다. 다음과 같이 작은 따옴표 대신 문자열에 나타나지 않는 모든 문자를 구분자로 지정할 수 있습니다. 슬라이드는 q' 표기를 사용하여 구분자를 지정하는 방법을 보여줍니다. 예제에서는 '!' 및 '['를 구분자로 사용합니다. 다음 예제를 살펴보십시오.

```
event := q'!Father's day!';
```

이 예제를 이 노트 페이지의 첫번째 예제와 비교할 수 있습니다. 구분자를 사용하려면 문자열을 q'로 시작합니다. 이 표기 다음의 문자는 사용되는 구분자입니다. 구분자를 지정한 후에 문자열을 입력하고 구분자를 닫고 작은 따옴표를 사용하여 표기를 닫으십시오. 다음 예제는 '['를 구분자로 사용하는 방법을 보여줍니다.

```
event := q'[Mother's day]';
```

변수 유형

- **PL/SQL 변수:**
 - 스칼라
 - 조합
 - 참조
 - LOB(대형 객체)
- **비PL/SQL 변수: 바인드 변수**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 유형

모든 PL/SQL 변수는 저장 형식, 제약 조건 및 유효한 값 범위를 지정하는 데이터 유형을 가지고 있습니다. PL/SQL은 변수, 상수 및 포인터를 선언하는 데 사용할 수 있는 다섯 개의 데이터 유형 범주인 스칼라, 조합, 참조, LOB(대형 객체) 및 객체를 지원합니다.

- **스칼라 데이터 유형:** 스칼라 데이터 유형은 단일 값을 보유합니다. 값은 변수의 데이터 유형에 따라 다릅니다. 예를 들어, 슬라이드 7의 예제에서 `Myname` 변수는 `VARCHAR2` 유형입니다. 따라서 `Myname`은 문자열 값을 보유할 수 있습니다. 또한 PL/SQL은 부울 변수를 지원합니다.
- **조합 데이터 유형:** 조합 데이터 유형은 스칼라 또는 조합을 내부 요소로 포함합니다. 레코드와 테이블은 조합 데이터 유형의 예입니다.
- **참조 데이터 유형:** 참조 데이터 유형은 저장 위치를 가리키는 *포인터*라고 하는 값을 보유합니다.
- **LOB 데이터 유형:** LOB 데이터 유형은 대형 객체의 위치를 지정하는 위치자라고 하는 값을 보유합니다. 대형 객체는 행 외에 저장되는 그래픽 이미지 등을 말합니다.

비PL/SQL 변수는 사전 컴파일러 프로그램에서 선언된 호스트 언어 변수, Forms 응용 프로그램의 화면 필드 및 호스트 변수를 포함합니다. 호스트 변수는 이 단원 뒷부분에서 배우게 됩니다.

LOB에 대한 자세한 내용은 *PL/SQL User's Guide and Reference*를 참조하십시오.

변수 유형

TRUE

25-JAN-01



The soul of the lazy man
desires, and he has nothing;
but the soul of the diligent
shall be made rich.

256120.08



Atlanta

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 유형(계속)

슬라이드는 다음 데이터 유형을 보여줍니다.

- TRUE는 부울 값을 나타냅니다.
- 25-JAN-01은 DATE를 나타냅니다.
- 이미지는 BLOB을 나타냅니다.
- 속담의 텍스트는 VARCHAR2 데이터 유형 또는 CLOB을 나타냅니다.
- 256120.08은 정밀도와 배율을 가진 NUMBER 데이터 유형을 나타냅니다.
- 필름 릴은 BFILE을 나타냅니다.
- 도시 이름 *Atlanta*는 VARCHAR2를 나타냅니다.

PL/SQL 변수 선언 및 초기화 지침

- 이름 지정 규칙을 따릅니다.
- 변수 이름으로 의미를 알 수 있는 이름을 사용합니다.
- NOT NULL 및 CONSTANT로 지정된 변수를 초기화합니다.
- 할당 연산자(:=) 또는 DEFAULT 키워드로 변수를 초기화합니다.

```
Myname VARCHAR2(20) := 'John';
```

```
Myname VARCHAR2(20) DEFAULT 'John';
```

- 가독성 및 코드 유지 관리 효율을 높이기 위해 각 행마다 하나씩 식별자를 선언합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언 및 초기화 지침

다음은 PL/SQL 변수 선언 시 따라야 하는 몇 가지 지침입니다.

- 이름 지정 규칙을 따릅니다. 예를 들어, name은 변수를 나타내고 c_name은 상수를 나타냅니다.
- 변수의 의미를 알 수 있는 알맞은 이름을 사용합니다. 예를 들어, salary1과 salary2 대신 salary와 sal_with_commission을 사용해 봅니다.
- NOT NULL 상수를 사용하지 않는 경우 변수 선언 시 값을 할당해야 합니다.
- 상수 선언 시 CONSTANT 키워드가 유형 지정자 앞에 나와야 합니다. 다음 선언은 NUMBER의 서브타입인 REAL의 상수 이름을 지정하고 이 상수에 50,000이라는 값을 할당합니다. 상수는 선언 시 초기화되어야 하며 그렇지 않으면 컴파일 오류가 발생합니다. 상수를 초기화한 후에 값을 변경할 수 없습니다.

```
sal CONSTANT REAL := 50000.00;
```

PL/SQL 변수 선언 및 초기화 지침

- 열 이름을 식별자로 사용하지 마십시오.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT      employee_id
  INTO        employee_id
  FROM        employees
  WHERE       last_name = 'Kochhar';
END;
/
```

- 변수가 값을 보유해야 하는 경우 NOT NULL 제약 조건을 사용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언 및 초기화 지침(계속)

- 할당 연산자 (:=) 또는 DEFAULT 예약어가 있는 표현식으로 변수를 초기화합니다. 초기 값을 할당하지 않으면 값을 할당할 때까지 새 변수에 기본적으로 NULL이 들어갑니다. 변수에 값을 할당하거나 재할당하려면 PL/SQL 할당문을 작성합니다. 모든 변수를 초기화하는 것은 좋은 프로그래밍 습관입니다.
- 두 개의 객체가 서로 다른 블록에서 정의된 경우에만 동일한 이름을 가질 수 있습니다. 두 객체가 병존하는 경우 레이블을 사용하여 한정하고 사용할 수 있습니다.
- 열 이름을 식별자로 사용하지 마십시오. PL/SQL 변수가 SQL 문에서 나타나고 이 변수 이름이 열 이름과 동일한 경우 Oracle 서버는 이 변수를 참조할 열로 간주합니다. 슬라이드의 예제 코드는 제대로 실행되지만 데이터베이스 테이블과 변수에 동일한 이름을 사용하여 작성된 코드는 관독이나 유지 관리가 쉽지 않습니다.
- 변수가 값을 포함해야 하는 경우 NOT NULL 제약 조건을 적용하십시오. NOT NULL로 정의된 변수에는 널을 할당할 수 없습니다. NOT NULL 제약 조건 다음에는 초기화 절이 나와야 합니다.

```
pincode NUMBER(15) NOT NULL := 'Oxford';
```

스칼라 데이터 유형

- 단일 값을 보유합니다.
- 내부 구성 요소가 없습니다.

TRUE

25-JAN-01

The soul of the lazy man
desires, and he has nothing;
but the soul of the diligent
shall be made rich.

256120.08

Atlanta

ORACLE

Copyright © 2009, Oracle. All rights reserved.

스칼라 데이터 유형

모든 상수, 변수 및 매개변수는 저장 형식, 제약 조건 및 유효한 값 범위를 지정하는 데이터 유형을 가집니다. PL/SQL은 다양한 사전 정의 데이터 유형을 제공합니다. 예를 들어 정수, 부동 소수점 수, 문자, 부울, 날짜, 컬렉션 및 LOB 유형 중에서 선택할 수 있습니다. 이 장에서는 PL/SQL 프로그램에서 자주 사용하는 기본 유형을 다룹니다.

스칼라 데이터 유형은 단일 값을 보유하며 내부 구성 요소가 없습니다. 스칼라 데이터 유형은 숫자, 문자, 날짜 및 부울의 네 가지 범주로 분류될 수 있습니다. 문자 및 숫자 데이터 유형은 기본 유형에 제약 조건을 적용한 서브타입을 가집니다. 예를 들어, INTEGER 및 POSITIVE는 NUMBER 기본 유형의 서브타입입니다.

스칼라 데이터 유형에 대한 자세한 내용과 전체 리스트는 *PL/SQL User's Guide and Reference*를 참조하십시오.

기본 스칼라 데이터 유형

- CHAR [(maximum_length)]
- VARCHAR2 (maximum_length)
- LONG
- LONG RAW
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 스칼라 데이터 유형

데이터 유형	설명
CHAR [(maximum_length)]	고정 길이 문자 데이터의 기본 유형이며 최대 길이는 32,767바이트입니다. 최대 길이를 지정하지 않으면 기본 길이가 1로 설정됩니다.
VARCHAR2 (maximum_length)	가변 길이 문자 데이터의 기본 유형이며 최대 길이는 32,767바이트입니다. VARCHAR2 변수와 상수는 기본 크기가 없습니다.
NUMBER [(precision, scale)]	정밀도 p 와 배율 s 를 가진 숫자입니다. 정밀도 p 의 범위는 1부터 38까지, 배율 s 의 범위는 -84부터 127까지입니다.
BINARY_INTEGER	-2,147,483,647과 2,147,483,647 사이의 정수에 대한 기본 유형입니다.

기본 스칼라 데이터 유형(계속)

데이터 유형	설명
PLS_INTEGER	-2,147,483,647과 2,147,483,647 사이의 부호 있는 정수에 대한 기본 유형입니다. PLS_INTEGER 값은 NUMBER 값보다 저장 공간이 적게 필요하고 연산 속도가 더 빠릅니다. Oracle Database 10g에서 PLS_INTEGER 및 BINARY_INTEGER 데이터 유형은 동일합니다. PLS_INTEGER 및 BINARY_INTEGER 값의 산술 연산은 NUMBER 값보다 빠릅니다.
BOOLEAN	논리적 계산에 사용 가능한 세 가지 값(TRUE, FALSE 및 NULL) 중 하나를 저장하는 기본 유형입니다.
BINARY_FLOAT	Oracle Database 10g에 도입된 새로운 데이터 유형입니다. IEEE 754 형식의 부동 소수점 수를 나타냅니다. 값을 저장하려면 5바이트가 필요합니다.
BINARY_DOUBLE	Oracle Database 10g에 도입된 새로운 데이터 유형입니다. IEEE 754 형식의 부동 소수점 수를 나타냅니다. 값을 저장하려면 9바이트가 필요합니다.

기본 스칼라 데이터 유형

- **DATE**
- **TIMESTAMP**
- **TIMESTAMP WITH TIME ZONE**
- **TIMESTAMP WITH LOCAL TIME ZONE**
- **INTERVAL YEAR TO MONTH**
- **INTERVAL DAY TO SECOND**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 스칼라 데이터 유형(계속)

데이터 유형	설명
DATE	날짜 및 시간에 대한 기본 유형입니다. DATE 값은 자정 이후 경과한 시간을 초 단위로 포함합니다. 날짜의 범위는 4712 B.C.와 9999 A.D 사이입니다.
TIMESTAMP	TIMESTAMP 데이터 유형은 DATE 데이터 유형을 확장하고 연도, 월, 일, 시, 분, 초 및 소수점 이하 자릿수를 저장합니다. 구문은 <code>TIMESTAMP[(precision)]</code> 입니다. 여기서 <code>precision</code> 은 선택적 파라미터로, 초 필드의 소수점 이하 자릿수를 지정합니다. 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으므로 0-9 범위의 정수 리터럴을 사용해야 합니다. 기본값은 6입니다.
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE 데이터 유형은 TIMESTAMP 데이터 유형을 확장하고 시간대 변위를 포함합니다. 시간대 변위는 로컬 시간과 UTC(Coordinated Universal Time 이전의 그리니치 표준시)의 차이(시간과 분)입니다. 구문은 <code>TIMESTAMP[(precision)] WITH TIME ZONE</code> 입니다. 여기서 <code>precision</code> 은 선택적 파라미터로, 초 필드의 소수점 이하 자릿수를 지정합니다. 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으므로 0-9 범위의 정수 리터럴을 사용해야 합니다. 기본값은 6입니다.

기본 스칼라 데이터 유형(계속)

데이터 유형	설명
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP WITH LOCAL TIME ZONE 데이터 유형은 TIMESTAMP 데이터 유형을 확장하고 시간대 변위를 포함합니다. 시간대 변위는 로컬 시간과 UTC(Coordinated Universal Time 이전의 그리니치 표준시)의 차이(시간과 분)입니다. 구문은 <code>TIMESTAMP[(precision)] WITH LOCAL TIME ZONE</code> 입니다. 여기서 <code>precision</code> 은 선택적 파라미터로, 초 필드의 소수점 이하 자릿수를 지정합니다. 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으므로 0-9 범위의 정수 리터럴을 사용해야 합니다. 기본값은 6입니다. 이 데이터 유형은 데이터베이스 열에 값을 삽입하면 해당 값이 데이터베이스 시간대로 정규화되고 시간대 변위가 열에 저장되지 않는다는 점에서 <code>TIMESTAMP WITH TIME ZONE</code> 과 다릅니다. 값을 검색할 때 Oracle 서버는 로컬 세션 시간대의 값을 반환합니다.
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH 데이터 유형은 연도와 월의 간격을 저장하거나 조작하는 데 사용됩니다. 구문은 <code>INTERVAL YEAR[(precision)] TO MONTH</code> 입니다. 여기서 <code>precision</code> 은 연도 필드의 자릿수를 지정합니다. <code>precision</code> 을 지정할 때 기호 상수 또는 변수는 사용할 수 없으므로 0-4 범위의 정수 리터럴을 사용해야 합니다. 기본값은 2입니다.
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND 데이터 유형은 일, 시, 분, 초의 간격을 저장하거나 조작하는 데 사용됩니다. 구문은 <code>INTERVAL DAY[(precision1)] TO SECOND[(precision2)]</code> 입니다. 여기서 <code>precision1</code> 과 <code>precision2</code> 는 각각 일 필드와 초 필드의 자릿수를 지정합니다. 두 경우 모두 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으므로 0-9 범위의 정수 리터럴을 사용해야 합니다. 기본값은 각각 2와 6입니다.

BINARY_FLOAT 및 BINARY_DOUBLE

- IEEE 754 형식의 부동 소수점 수를 나타냅니다.
- 더욱 뛰어난 상호 운용성 및 수행 속도를 제공합니다.
- NUMBER 데이터 유형에서 저장할 수 있는 값 범위를 벗어난 값을 저장합니다.
- 닫힌 산술 연산 및 투명한 반올림의 이점을 제공합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

BINARY_FLOAT 및 BINARY_DOUBLE

BINARY_FLOAT 및 BINARY_DOUBLE은 Oracle Database 10g에서 도입된 새로운 데이터 유형입니다.

- **IEEE 754 형식의 부동 소수점 수를 나타냄:** 이러한 데이터 유형은 과학 계산 및 IEEE(Institute of Electrical and Electronics Engineers) 형식을 따르는 프로그램 간의 데이터 교환에 사용할 수 있습니다.
- **이점:** 많은 컴퓨터 시스템이 고유 프로세서 명령어를 통해 IEEE 754 부동 소수점 연산을 지원합니다. 이러한 유형은 부동 소수점 데이터가 포함된 고성능 계산에서 효율적입니다. Oracle은 이러한 두 개의 데이터 유형에 동일한 형식을 적용하기 때문에 이러한 프로그램과의 상호 작용이 더욱 수월합니다.
- **더욱 뛰어난 상호 운용성 및 수행 속도:** 상호 운용성은 주로 이러한 두 데이터 유형의 형식에 기인합니다. 이러한 데이터 유형은 과학 데이터 처리와 같은 숫자 중심의 연산에서 뛰어난 성능을 발휘합니다.
- **Oracle NUMBER 범위를 벗어나는 값 저장:** 1-22바이트를 임의로 사용하는 Oracle NUMBER와 달리 BINARY_FLOAT는 5바이트가 필요하고 BINARY_DOUBLE은 9바이트가 필요합니다. 이러한 데이터 유형은 NUMBER 범위를 벗어나는 숫자 데이터를 저장할 수 있어야 한다는 숫자 데이터 유형의 요구 조건을 충족합니다.

BINARY_FLOAT 및 BINARY_DOUBLE(계속)

- **단한 산술 연산 및 투명한 반올림:** BINARY_FLOAT 및 BINARY_DOUBLE을 사용하는 모든 산술 연산은 단한 있습니다. 즉, 산술 연산 결과로 정상적인 값이나 특수 값이 산출됩니다. 명시적 변환에 대해 신경 쓸 필요가 없습니다. 예를 들어, BINARY_FLOAT 숫자를 다른 BINARY_FLOAT 숫자와 곱하면 결과는 BINARY_FLOAT 숫자가 됩니다. BINARY_FLOAT를 0으로 나누는 것은 정의되어 있지 않으며, 실제로 Inf(Infinite)라는 특수 값이 산출됩니다. 이러한 데이터 유형의 연산에는 반올림이 적용되며 반올림은 PL/SQL 유저에게 투명합니다. 기본 모드는 가장 가까운 이진 자리로 반올림하는 것입니다. 대부분의 회계 응용 프로그램은 소수점 반올림 기능이 필요하지만 순수 과학 응용 프로그램은 그렇지 않습니다.

예제

```
SET SERVEROUTPUT ON
DECLARE
    bf_var BINARY_FLOAT;
    bd_var BINARY_DOUBLE;
BEGIN
    bf_var := 270/35f;
    bd_var := 140d/0.35;
    DBMS_OUTPUT.PUT_LINE('bf: ' || bf_var);
    DBMS_OUTPUT.PUT_LINE('bd: ' || bd_var);
END;
/
```

```
anonymous block completed
bf: 7.71428585E+000
bd: 4.0E+002
```

스칼라 변수 선언

예제

```
DECLARE
  emp_job          VARCHAR2(9);
  count_loop       BINARY_INTEGER := 0;
  dept_total_sal   NUMBER(9,2) := 0;
  orderdate        DATE := SYSDATE + 7;
  c_tax_rate       CONSTANT NUMBER(3,2) := 8.25;
  valid            BOOLEAN NOT NULL := TRUE;
  ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

스칼라 변수 선언

슬라이드의 변수 선언 예제는 다음과 같이 정의됩니다.

- emp_job: 사원의 직위를 저장하는 변수입니다.
- count_loop: 루프 반복 횟수를 세는 변수이며 0으로 초기화되었습니다.
- dept_total_sal: 부서의 총 급여를 합산하는 변수이며 0으로 초기화되었습니다.
- orderdate: 주문의 출고 날짜를 저장하는 변수이며 오늘부터 일주일 후의 날짜로 초기화되었습니다.
- c_tax_rate: 세율에 대한 상수 변수이며 PL/SQL 블록 전체에서 변경되지 않고 8.25로 설정되었습니다.
- valid: 해당 데이터 부분이 유효한지 여부를 나타내는 플래그이며 TRUE로 초기화되었습니다.

%TYPE 속성

%TYPE 속성

- 다음에 따라 변수를 선언하는 데 사용됩니다.
 - 데이터베이스 열 정의
 - 다른 선언된 변수
- 다음 항목이 앞에 나옵니다.
 - 데이터베이스 테이블 및 열
 - 선언된 변수의 이름

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%TYPE 속성

PL/SQL 변수는 일반적으로 데이터베이스에 저장된 데이터를 보유하고 조작하기 위해 선언됩니다. 열 값을 보유할 PL/SQL 변수를 선언하는 경우 변수의 데이터 유형과 정밀도가 정확해야 합니다. 그렇지 않으면 실행하는 동안 PL/SQL 오류가 발생합니다. 큰 서브 프로그램을 설계해야 하는 경우 이 과정은 시간이 많이 걸리고 오류를 유발할 수 있습니다.

변수의 데이터 유형과 정밀도를 하드 코딩하는 대신 %TYPE 속성을 사용하여 이전에 선언한 다른 변수나 데이터베이스 열에 따라 변수를 선언할 수 있습니다. %TYPE 속성은 변수에 저장된 값이 데이터베이스의 테이블에서 파생될 경우에 가장 자주 사용됩니다. %TYPE 속성을 사용하여 변수를 선언할 경우 데이터베이스 테이블과 열 이름이 앞에 나와야 합니다. 이전에 선언한 변수를 참조할 경우 변수 이름이 속성보다 먼저 나와야 합니다.

%TYPE 속성의 이점

- 데이터 유형 불일치나 잘못된 정밀도로 인해 발생하는 오류를 방지할 수 있습니다.
- 변수의 데이터 유형을 하드 코딩하는 것을 피할 수 있습니다.
- 열 정의가 변경된 경우 변수 선언을 변경할 필요가 없습니다. %TYPE 속성을 사용하지 않고 이미 특정 테이블에 대해 일부 변수를 선언한 경우 변수가 선언된 열이 변경되면 PL/SQL 블록에서 오류가 발생할 수 있습니다. %TYPE 속성을 사용하는 경우 PL/SQL은 블록이 컴파일될 때 변수의 데이터 유형과 크기를 결정합니다. 따라서 이러한 변수가 항상 해당 변수를 채우는 데 사용되는 열과 호환되도록 보장합니다.

%TYPE 속성을 사용하여 변수 선언

구문

```
identifier      table.column_name%TYPE;
```

예제

```
...  
  emp_lname      employees.last_name%TYPE;  
  balance        NUMBER(7,2);  
  min_balance    balance%TYPE := 1000;  
...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%TYPE 속성을 사용하여 변수 선언

사원의 성을 저장할 변수를 선언합니다. emp_lname 변수는 employees 테이블의 last_name 열과 동일한 데이터 유형으로 정의됩니다. %TYPE 속성은 데이터베이스 열의 데이터 유형을 제공합니다.

최소 잔액을 1,000으로 하여 은행 계좌의 잔액을 저장할 변수를 선언합니다. min_balance 변수는 balance 변수와 동일한 데이터 유형으로 정의됩니다. %TYPE 속성은 변수의 데이터 유형을 제공합니다.

NOT NULL 데이터베이스 열 제약 조건은 %TYPE을 사용하여 선언된 변수에 적용되지 않습니다. 따라서 %TYPE 속성과 NOT NULL로 정의된 데이터베이스 열을 사용하여 변수를 선언할 경우 변수에 NULL 값을 할당할 수 있습니다.

부울 변수 선언

- TRUE, FALSE 및 NULL 값만 부울 변수에 할당할 수 있습니다.
- 조건식은 논리 연산자 AND 및 OR, 그리고 단항 연산자 NOT을 사용하여 변수 값을 확인합니다.
- 변수는 항상 TRUE, FALSE 또는 NULL을 반환합니다.
- 산술, 문자 및 날짜 표현식은 부울 값을 반환하는 데 사용될 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

부울 변수 선언

PL/SQL에서 SQL 문과 프로시저문의 변수를 비교할 수 있습니다. 이러한 비교를 부울 표현식이라고 하며 관계 연산자로 구분된 간단하거나 복잡한 표현식으로 구성됩니다. SQL 문에서는 부울 표현식을 사용하여 명령문에 적용할 테이블의 행을 지정할 수 있습니다. 프로시저문에서 부울 표현식은 조건부 제어의 기초가 됩니다. NULL은 누락된 값, 적용할 수 없는 값 또는 알 수 없는 값을 나타냅니다.

예제

```
emp_sal1 := 50000;  
emp_sal2 := 60000;
```

다음 표현식은 TRUE를 반환합니다.

```
emp_sal1 < emp_sal2
```

부울 변수를 선언하고 초기화합니다.

```
DECLARE  
    flag BOOLEAN := FALSE;  
BEGIN  
    flag := TRUE;  
END;  
/
```

바인드 변수

바인드 변수:

- 호스트 환경에서 생성됩니다.
- 호스트 변수라고도 합니다.
- VARIABLE 키워드를 사용하여 생성됩니다.
- SQL 문과 PL/SQL 블록에서 사용됩니다.
- PL/SQL 블록이 실행된 후에도 액세스할 수 있습니다.
- 앞에 콜론을 사용하여 참조합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

바인드 변수

바인드 변수는 호스트 환경에서 생성되는 변수입니다. 따라서 호스트 변수라고도 합니다.

바인드 변수의 사용

바인드 변수는 PL/SQL 블록의 선언 섹션이 아니라 호스트 환경에서 생성됩니다. PL/SQL 블록에서 선언된 변수는 해당 블록을 실행할 때만 사용할 수 있습니다. 블록이 실행된 후에는 변수에 할당된 메모리가 해제됩니다. 그러나 바인드 변수는 블록이 실행된 후에도 액세스할 수 있습니다. 따라서 바인드 변수를 생성하면 여러 서브 프로그램에서 사용하고 조작할 수 있습니다. 다른 변수와 마찬가지로 SQL 문과 PL/SQL 블록에서 사용할 수 있습니다. 이러한 변수는 PL/SQL 서브 프로그램 내부 또는 외부로 런타임 값으로 전달할 수 있습니다.

바인드 변수 생성

SQL Developer나 SQL*Plus에서 바인드 변수를 생성하려면 VARIABLE 명령을 사용합니다. 예를 들어, 다음과 같이 NUMBER 및 VARCHAR2 유형의 변수를 선언합니다.

```
VARIABLE return_code NUMBER
VARIABLE return_msg VARCHAR2(30)
```

SQL*Plus와 SQL Developer는 모두 바인드 변수를 참조할 수 있고 SQL Developer는 SQL*Plus PRINT 명령을 통해 바인드 변수의 값을 표시할 수 있습니다.

바인드 변수(계속)

예제

변수 앞에 콜론을 사용하여 PL/SQL 프로그램에서 바인드 변수를 참조할 수 있습니다.

```
VARIABLE result NUMBER
BEGIN
    SELECT (SALARY*12) + NVL(COMMISSION_PCT,0) INTO :result
    FROM employees WHERE employee_id = 144;
END;
/
PRINT result
```

```
anonymous block completed
result
-----
30000
```

주: NUMBER 유형의 바인드 변수를 생성하는 경우 정밀도와 배율을 지정할 수 없습니다. 그러나 문자열의 크기를 지정할 수 있습니다. Oracle NUMBER는 차원(Dimension)에 관계없이 동일한 방식으로 저장됩니다. 즉, Oracle 서버는 7, 70, .0734를 저장할 때 동일한 바이트 수를 사용합니다. 숫자 형식에서 Oracle 숫자 표현의 크기를 계산하는 것은 비효율적이므로 코드에서 항상 필요한 바이트를 할당하도록 합니다. 사용자가 입력하는 문자열의 크기에 따라 필요한 바이트 수를 할당할 수 있습니다.

바인드 변수 출력

예제

```
VARIABLE emp_salary NUMBER
BEGIN
  SELECT salary INTO :emp_salary
  FROM employees WHERE employee_id = 178;
END;
/
PRINT emp_salary
SELECT first_name, last_name FROM employees
WHERE salary=:emp_salary;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

바인드 변수 출력

SQL Developer에서 PRINT 명령을 사용하여 바인드 변수의 값을 표시할 수 있습니다. 슬라이드에 나오는 PL/SQL 블록을 실행하면 PRINT 명령을 실행할 때 다음과 같은 출력 결과가 나타납니다.

```
anonymous block completed
emp_salary
-----
7000
```

emp_salary는 바인드 변수입니다. 이제 SQL 문이나 PL/SQL 프로그램에서 이 변수를 사용할 수 있습니다. 바인드 변수를 사용하는 SQL 문을 잘 살펴보십시오. SQL 문의 출력 결과는 다음과 같습니다.

FIRST_NAME	LAST_NAME
Oliver	Tuvault
Sarath	Sewall
Kimberely	Grant
3 rows selected	

주: 모든 바인드 변수를 출력하려면 변수 없이 PRINT 명령을 사용하십시오.

바인드 변수 출력

예제

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
BEGIN
    SELECT salary INTO :emp_salary
    FROM employees WHERE employee_id = 178;
END;
/
```

```
anonymous block completed
emp_salary
-----
7000
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

바인드 변수 출력(계속)

성공적인 PL/SQL 블록에서 사용되는 바인드 변수를 자동으로 출력하려면 SET AUTOPRINT ON 명령을 사용합니다.

치환 변수

- 런타임에 유저 입력을 얻는 데 사용됩니다.
- 앞에 앰퍼샌드(&)를 붙여 PL/SQL 블록 내에서 참조됩니다.
- 런타임에 얻을 수 있는 값을 하드 코딩하는 것을 피할 수 있습니다.

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
  empno NUMBER(6) := &empno;
BEGIN
  SELECT salary INTO :emp_salary
  FROM employees WHERE employee_id = empno;
END;
/
```

ORACLE

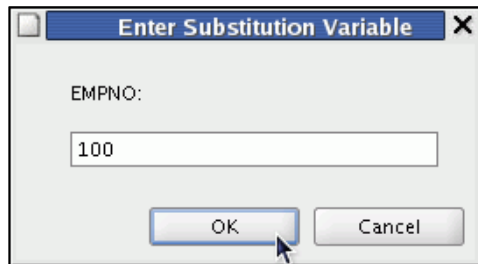
Copyright © 2009, Oracle. All rights reserved.

치환 변수

치환 변수는 PL/SQL 블록으로 런타임 값을 전달하는 데 사용될 수 있습니다. 앞에 앰퍼샌드(&)를 붙여 SQL 문과 PL/SQL 블록 내에서 치환 변수를 참조할 수 있습니다. 텍스트 값은 PL/SQL 블록이 실행되기 전에 PL/SQL 블록으로 치환됩니다. 따라서 루프를 사용하여 치환 변수를 다른 값으로 치환할 수 없습니다. 루프에 변수를 포함하는 경우에도 값을 입력하라는 메시지가 한 번만 나타납니다. 하나의 값만 치환 변수를 대신할 수 있습니다.

슬라이드의 블록을 실행하면 SQL Developer가 치환 변수인 empno의 값을 입력하라는 메시지를 표시합니다.

치환 변수



A screenshot of the 'Enter Substitution Variable' dialog box in SQL Developer. The title bar says 'Enter Substitution Variable' with a close button. Inside, there is a label 'EMPNO:' followed by a text input field containing the value '100'. At the bottom, there are two buttons: 'OK' and 'Cancel'. A mouse cursor is pointing at the 'OK' button.

1

```
VARIABLE emp_salary NUMBER
SET AUTOPRINT ON
DECLARE
empno NUMBER(6):=100;
BEGIN
SELECT salary INTO :emp_salary
FROM employees WHERE employee_id = empno;
END;
anonymous block completed
emp_salary
-----
24000
```

2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

치환 변수(계속)

1. 이전 슬라이드의 블록을 실행하면 SQL Developer가 치환 변수인 empno의 값을 입력하라는 메시지를 표시합니다. 슬라이드처럼 값을 입력하고 OK 버튼을 누릅니다.
2. 슬라이드와 같은 출력이 표시됩니다.

치환 변수를 확인하는 프롬프트 표시

```
SET VERIFY OFF
VARIABLE emp_salary NUMBER
ACCEPT empno PROMPT 'Please enter a valid employee
number: '
SET AUTOPRINT ON
DECLARE
    empno NUMBER(6) := &empno;
BEGIN
    SELECT salary INTO :emp_salary FROM employees
    WHERE employee_id = empno;
END;
/
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

치환 변수를 확인하는 프롬프트 표시

이전 슬라이드에서 기본 프롬프트 메시지는 "EMPNO:" 였습니다.

PROMPT 명령을 사용하여 슬라이드에 표시된 것처럼 메시지를 변경할 수 있습니다. 이 명령은 SQL*Plus 명령이므로 PL/SQL 블록에 포함될 수 없습니다.

유저 변수에 DEFINE 사용

예제

```
SET VERIFY OFF
DEFINE lname= Urman
DECLARE
    fname VARCHAR2(25);
BEGIN
    SELECT first_name INTO fname FROM employees
    WHERE last_name='&lname';
END;
/
```


ORACLE

Copyright © 2009, Oracle. All rights reserved.

유저 변수에 DEFINE 사용

DEFINE 명령은 유저 변수를 지정하고 CHAR 값을 할당합니다. CHAR 데이터 유형의 변수만 정의할 수 있습니다. 50000이라는 숫자를 입력해도 SQL Developer는 5,0,0,0,0 문자로 구성된 CHAR 값을 변수에 할당합니다. 슬라이드에 표시된 것처럼 앞에 앰퍼샌드(&)를 사용하여 이러한 변수를 참조할 수 있습니다.

조합 데이터 유형

TRUE	23-DEC-98	ATLANTA	
------	-----------	---------	---

PL/SQL 테이블 구조

1	SMITH
2	JONES
3	NANCY
4	TIM

PL/SQL 테이블 구조

1	5000
2	2345
3	12
4	3456

↑
↑
↑
↑
PLS_INTEGER
VARCHAR2

↑
↑
↑
↑
PLS_INTEGER
NUMBER

ORACLE

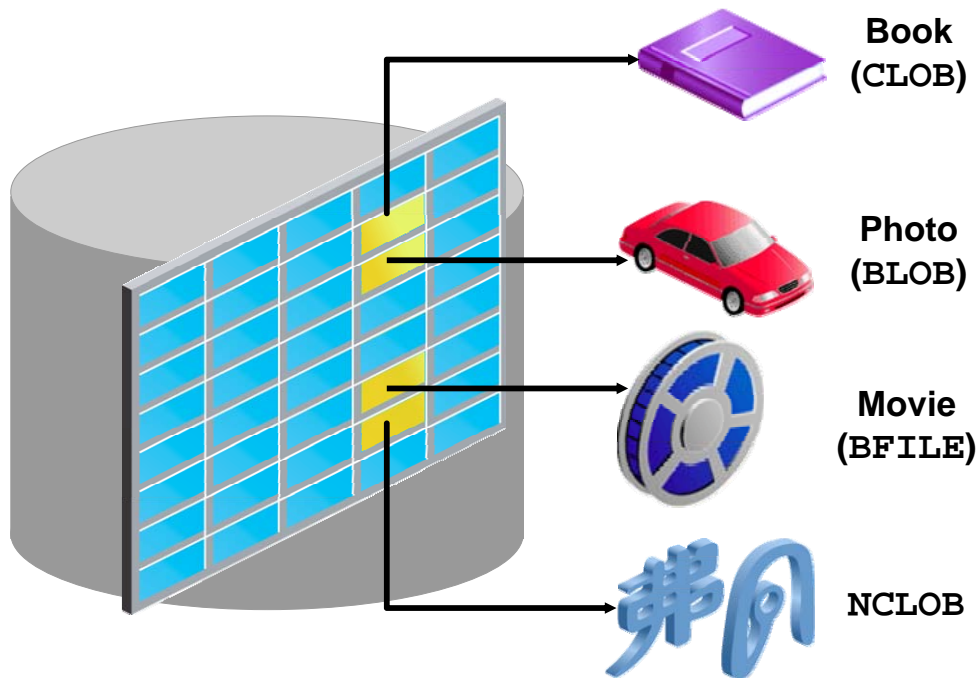
Copyright © 2009, Oracle. All rights reserved.

조합 데이터 유형

스칼라 유형은 내부 구성 요소가 없습니다. 조합 유형은 개별적으로 조작할 수 있는 내부 구성 요소를 가집니다. 조합 데이터 유형(컬렉션이라고도 함)은 TABLE, RECORD, NESTED TABLE 및 VARRAY 유형입니다.

TABLE 데이터 유형을 사용하여 데이터 컬렉션을 전체 객체로 참조하고 조작합니다. RECORD 데이터 유형을 사용하여 관련은 있지만 유사하지 않은 데이터를 논리적 단위로 처리합니다. NESTED TABLE 및 VARRAY 데이터 유형은 *Oracle Database 10g: PL/SQL 프로그램 단위 개발* 과정에서 다룹니다.

LOB 데이터 유형 변수



ORACLE

Copyright © 2009, Oracle. All rights reserved.

LOB 데이터 유형 변수

LOB(대형 객체)은 다량의 데이터를 저장하는 것을 의미합니다. 데이터베이스 열은 LOB 범주에 포함될 수 있습니다. LOB 범주의 데이터 유형(BLOB, CLOB 등)을 사용하여 텍스트, 그래픽 이미지, 비디오 클립 및 사운드 웨이브 형식과 같은 구조화되지 않은 데이터 블록을 최대 4GB까지 저장할 수 있습니다. LOB 데이터 유형은 데이터에 임의로 하나씩 효율적으로 액세스할 수 있으며 객체 유형의 속성이 될 수 있습니다.

- CLOB(Character Large Object) 데이터 유형은 대형 문자 데이터 블록을 데이터베이스에 저장하는 데 사용됩니다.
- BLOB(Binary Large Object) 데이터 유형은 구조화되지 않거나 구조화된 대형 바이너리 객체를 데이터베이스에 저장하는 데 사용됩니다. 이러한 데이터를 데이터베이스에 삽입하거나 데이터베이스에서 검색할 경우 데이터베이스는 해당 데이터를 해석하지 않습니다. 이러한 데이터는 해당 데이터를 사용하는 외부 응용 프로그램이 해석합니다.
- BFILE(Binary File) 데이터 유형은 대형 Binary File을 저장하는 데 사용됩니다. 다른 LOB과 달리 BFILE은 데이터베이스에 저장되지 않습니다. BFILE은 데이터베이스 외부에 저장됩니다. 운영 체제 파일이 여기에 속합니다. BFILE에 대한 포인터만 데이터베이스에 저장됩니다.
- NCLOB(National language Character Large Object) 데이터 유형은 대형 블록의 단일 바이트 또는 고정 너비 멀티바이트 NCHAR 유니코드 데이터를 데이터베이스에 저장하는 데 사용됩니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 유효한 식별자 및 부적합한 식별자 구분
- PL/SQL 블록의 선언 섹션에서 변수 선언
- 실행 섹션에서 변수 초기화 및 사용
- 스칼라 데이터 유형과 조합 데이터 유형 구별
- %TYPE 속성 사용
- 바인드 변수 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

익명 PL/SQL 블록은 이름이 지정되지 않은 PL/SQL 프로그램의 기본 단위입니다. SQL 또는 PL/SQL 문 집합으로 구성되어 논리적 기능을 수행합니다. 선언부는 PL/SQL 블록의 첫번째 부분이며 변수, 상수, 커서 및 예외라고 부르는 오류 상황 정의와 같은 객체를 선언하는 데 사용됩니다.

이 단원에서는 선언 섹션에서 변수를 선언하는 방법을 배웠습니다. 변수 선언 시 따라야 하는 몇 가지 지침을 살펴보았습니다. 변수를 선언할 때 초기화하는 방법을 배웠습니다.

PL/SQL 블록의 실행부는 필수 요소이며 데이터 query 및 조작을 위한 SQL 및 PL/SQL 문을 포함합니다. 이 단원에서는 실행 섹션에서 변수를 초기화하는 방법과 변수를 사용하고 변수 값을 조작하는 방법을 배웠습니다.

연습 2: 개요

이 연습에서는 다음 내용을 다룹니다.

- 유효한 식별자 판별
- 유효한 변수 선언 판별
- 익명 블록 내에서 변수 선언
- %TYPE 속성을 사용하여 변수 선언
- 바인드 변수 선언 및 출력
- PL/SQL 블록 실행

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 2: 개요

1, 2, 3번 문제는 객관식입니다.

연습 2

1. 적합한 식별자와 부적합한 식별자 이름을 구분합니다.
 - a. today
 - b. last_name
 - c. today's_date
 - d. Number_of_days_in_February_this_year
 - e. Isleap\$year
 - f. #number
 - g. NUMBER#
 - h. number1to7

2. 다음 변수 선언 및 초기화가 적합한지 식별합니다.
 - a. number_of_copies PLS_INTEGER;
 - b. printer_name constant VARCHAR2(10);
 - c. deliver_to VARCHAR2(10):=Johnson;
 - d. by_when DATE:= SYSDATE+1;

3. 다음 익명 블록을 검토하고 올바른 문장을 선택합니다.


```
SET SERVEROUTPUT ON
DECLARE
    fname VARCHAR2(20);
    lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
    DBMS_OUTPUT.PUT_LINE( FNAME || ' ' || lname);
END;
/
```

 - a. 블록이 성공적으로 실행되고 "fernandez"가 출력됩니다.
 - b. fname 변수가 초기화하지 않고 사용되었기 때문에 오류가 발생합니다.
 - c. 블록이 성공적으로 실행되고 "null fernandez"가 출력됩니다.
 - d. VARCHAR2 유형의 변수를 초기화하는 데 DEFAULT 키워드를 사용할 수 없기 때문에 오류가 발생합니다.
 - e. 블록에서 FNAME 변수가 선언되지 않았기 때문에 오류가 발생합니다.

4. 익명 블록을 생성합니다. SQL Developer에서 연습 1의 2번 문제에서 생성한 lab_01_02_soln.sql 스크립트를 엽니다.
 - a. 이 PL/SQL 블록에 선언 섹션을 추가합니다. 선언 섹션에서 다음 변수를 선언합니다.
 1. DATE 유형의 today 변수. today를 SYSDATE로 초기화합니다.
 2. today 유형의 tomorrow 변수. %TYPE 속성을 사용하여 이 변수를 선언합니다.
 - b. 실행 섹션에서 내일 날짜를 계산하는 표현식(today 값에 1 추가)을 사용하여 tomorrow 변수를 초기화합니다. "Hello World"를 출력한 후 today와 tomorrow의 값을 출력합니다.

연습 2(계속)

- c. 이 스크립트를 실행하고 lab_02_04_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello World
TODAY IS : 28-JAN-09
TOMORROW IS : 29-JAN-09
```

5. lab_02_04_soln.sql 스크립트를 편집합니다.
- 두 개의 바인드 변수를 생성하는 코드를 추가합니다. NUMBER 유형의 바인드 변수 basic_percent 및 pf_percent를 생성합니다.
 - PL/SQL 블록의 실행 섹션에서 basic_percent와 pf_percent에 각각 값 45와 12를 할당합니다.
 - "/"로 PL/SQL 블록을 종료하고 PRINT 명령을 사용하여 바인드 변수 값을 표시합니다.
 - 스크립트 파일을 실행하고 lab_02_05_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello World
TODAY IS : 28-JAN-09
TOMORROW IS : 29-JAN-09

basic_percent
--
45

pf_percent
--
12
```


3

실행문 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **PL/SQL 블록의 렉시칼 단위 파악**
- **PL/SQL의 내장 SQL 함수 사용**
- **암시적 변환이 발생하는 경우와 명시적 변환으로 처리해야 하는 경우 설명**
- **중첩 블록 작성 및 레이블로 변수 한정**
- **적절한 들여쓰기로 읽기 쉬운 코드 작성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

PL/SQL 블록에서 변수를 선언하고 실행문을 작성하는 방법을 배웠습니다. 이 단원에서는 렉시칼 단위가 PL/SQL 블록을 구성하는 방법을 설명합니다. 중첩 블록을 작성하는 방법을 배웁니다. 또한 중첩 블록에서 변수의 범위 및 가시성에 대해 배우고 레이블로 한정하는 방법을 공부합니다.

PL/SQL 블록의 렉시칼 단위

렉시칼 단위:

- 모든 PL/SQL 블록의 기본 구조입니다.
- 문자, 숫자, 탭, 공백, Return 및 기호를 포함한 문자 시퀀스입니다.
- 다음과 같이 분류할 수 있습니다.
 - 식별자
 - 구분자
 - 리터럴
 - 비교

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록의 렉시칼 단위

렉시칼 단위는 문자, 숫자, 특수 문자, 탭, 공백, Return 및 기호를 포함합니다.

- **식별자:** 식별자는 PL/SQL 객체에게 부여되는 이름입니다. 유효한 식별자와 잘못된 식별자를 구분하는 방법은 이미 배웠습니다. 키워드는 식별자로 사용할 수 없다는 점에 유의하십시오.

따옴표로 묶인 식별자:

- 식별자의 대소문자 구분
- 공백과 같은 문자 포함
- 예약어 사용

예제:

```
"begin date" DATE;  
"end date" DATE;  
"exception thrown" BOOLEAN DEFAULT TRUE;
```

이러한 변수를 연이어 사용할 때는 항상 큰 따옴표로 묶어야 합니다.

- **구분자:** 구분자는 특별한 의미를 지닌 기호입니다. 이미 앞에서 세미콜론(;)이 SQL 또는 PL/SQL 문을 종료하는 데 사용된다는 것을 배웠습니다. 따라서 ;은 구분자의 가장 좋은 예입니다. 자세한 내용은 *PL/SQL User's Guide and Reference*를 참조하십시오.

PL/SQL 블록의 렉시칼 단위(계속)

- 구분자 (계속) 구분자는 PL/SQL에서 특별한 의미를 가지는 간단한 또는 복잡한 기호입니다. 간단한 기호

기호	의미
+	더하기 연산자
-	빼기/부정 연산자
*	곱하기 연산자
/	나누기 연산자
=	등호 연산자
@	원격 액세스 표시자
;	명령문 종료자

복잡한 기호

기호	의미
<>	부등 연산자
!=	부등 연산자
	연결 연산자
--	단일 행 주석 표시자
/*	주석 시작 구분자
*/	주석 종료 구분자
:=	할당 연산자

주: 이 리스트에는 전체 구분자가 아니라 일부만 포함되어 있습니다.

- 리터럴:** 변수에 할당되는 모든 값은 리터럴입니다. 식별자가 아닌 모든 문자, 숫자, 부울 또는 날짜 값은 리터럴입니다. 리터럴은 다음과 같이 분류됩니다.
 - 문자 리터럴:** 모든 문자열 리터럴은 데이터 유형이 CHAR이므로 문자 리터럴이라고 합니다(예: John, 12C, 1234, 12-JAN-1923).
 - 숫자 리터럴:** 숫자 리터럴은 정수 또는 실수 값을 나타냅니다(예: 428 및 1.276).
 - 부울 리터럴:** 부울 변수에 할당된 값은 부울 리터럴입니다. TRUE, FALSE 및 NULL은 부울 리터럴이거나 키워드입니다.
- 주석:** 해당 코드 부분이 어떤 작업을 수행하는지 설명하는 것은 좋은 프로그래밍 습관입니다. PL/SQL 블록에 설명을 포함시켜도 컴파일러는 해당 내용을 해석할 수 없습니다. 해당 내용을 컴파일하지 않도록 알려줄 수 있는 방법이 있어야 합니다. 주석은 주로 이러한 목적에 사용됩니다. 주석으로 추가한 모든 내용은 컴파일러가 해석하지 않습니다.
 - 단일 행의 주석을 삽입할 경우 두 개의 하이픈(--)을 사용합니다.
 - 다중 행의 주석을 삽입할 경우 주석 시작 및 종료 구분자(/* 및 */)를 사용합니다.

PL/SQL 블록 구문 및 작성 지침

- **리터럴:**

- 문자 및 날짜 리터럴은 작은 따옴표로 묶어야 합니다.

```
name := 'Henderson';
```

- 숫자에는 간단한 값이나 과학적 표기법이 올 수 있습니다.

- 명령문은 여러 행으로 이어질 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록 구문 및 작성 지침

리터럴은 식별자로 나타내지 않는 명시적 숫자, 문자열, 날짜 또는 부울 값입니다.

- 문자 리터럴은 PL/SQL 문자 집합에서 출력 가능한 모든 문자(문자, 숫자, 공백 및 특수 기호)를 포함합니다.
- 숫자 리터럴은 간단한 값(예: -32.5)이나 과학적 표기법(예: 2E5는 $2 * 10^5 = 200,000$ 을 의미)으로 나타낼 수 있습니다.

코드 주석 처리

- 단일 행 주석은 행의 맨 앞에 두 개의 하이픈(--)을 사용합니다.
- 다중 행 주석은 /* 기호와 */ 기호 사이에 배치합니다.

예제

```
DECLARE
...
annual_sal NUMBER (9,2);
BEGIN      -- Begin the executable section

/* Compute the annual salary based on the
   monthly salary input from the user */
annual_sal := monthly_sal * 12;
END;       -- This is the end of the block
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

코드 주석 처리

각 단계를 문서화하고 디버깅을 지원하기 위해 코드를 주석 처리해야 합니다. 단일 행 주석의 경우에는 두 개의 하이픈(--)으로 PL/SQL 코드를 주석 처리하고 주석이 여러 행에 걸쳐 나오는 경우에는 주석을 /* 기호와 */ 기호 사이에 배치합니다.

주석은 정보 제공용으로만 엄격히 제한되며 논리나 데이터에 어떠한 조건이나 기능도 적용할 수 없습니다. 잘 정리된 주석은 코드 가독성 및 차후 코드 유지 관리를 위해 매우 중요합니다. 슬라이드의 예제에서 /* 기호와 */ 기호 사이에 배치된 행은 다음 코드를 설명하는 주석을 나타냅니다.

PL/SQL의 SQL 함수

- 프로시저문에서 사용할 수 있는 함수:
 - 단일 행 숫자
 - 단일 행 문자
 - 데이터 유형 변환
 - 날짜
 - 시간 기록
 - GREATEST 및 LEAST
 - 기타 함수
- 프로시저문에서 사용할 수 없는 함수:
 - DECODE
 - 그룹 함수

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SQL 함수

SQL은 SQL문에서 사용할 수 있는 수많은 미리 정의된 함수를 제공합니다. 이러한 함수들은 대부분 PL/SQL 식에서 유효합니다.

다음 함수는 프로시저문에서 사용할 수 없습니다.

- DECODE
- 그룹 함수: AVG, MIN, MAX, COUNT, SUM, STDDEV, VARIANCE
그룹 함수는 테이블의 행 그룹에 적용되므로 PL/SQL 블록의 SQL 문에서만 사용할 수 있습니다.

여기에서 언급한 함수는 전체 리스트 중 일부에 불과합니다.

PL/SQL의 SQL 함수: 예제

- 문자열의 길이를 구합니다.

```
desc_size INTEGER(5);  
prod_description VARCHAR2(70):='You can use this  
product with your radios for higher frequency';  
  
-- get the length of the string in prod_description  
desc_size:= LENGTH(prod_description);
```

- 사원 이름을 소문자로 변환합니다.

```
emp_name:= LOWER(emp_name);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SQL 함수: 예제

SQL 함수를 사용하여 데이터를 조작할 수 있습니다. 이러한 함수는 다음 범주로 그룹화됩니다.

- 숫자
- 문자
- 변환
- 날짜
- 기타

데이터 유형 변환

- 데이터를 유사한 데이터 유형으로 변환합니다.
- 두 가지 유형이 있습니다.
 - 암시적 변환
 - 명시적 변환
- 몇 개의 변환 함수:
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER
 - TO_TIMESTAMP

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 유형 변환

모든 프로그래밍 언어에서 데이터 유형 변환은 공통적인 요구 사항입니다. PL/SQL은 스칼라 데이터 유형을 사용하여 이러한 변환을 처리할 수 있습니다. 데이터 유형 변환에는 두 가지 유형이 있습니다.

암시적 변환: PL/SQL은 명령문에 데이터 유형이 혼용된 경우 동적으로 변환을 시도합니다. 다음 예제를 살펴보십시오.

```
DECLARE
    salary NUMBER(6):=6000;
    sal_hike VARCHAR2(5):='1000';
    total_salary salary%TYPE;
BEGIN
    total_salary:=salary+sal_hike;
END;/
```

위의 예제에서 sal_hike 변수는 VARCHAR2 유형입니다. 전체 급여를 계산할 때 PL/SQL은 먼저 sal_hike를 NUMBER로 변환한 다음 연산을 수행합니다. 결과는 NUMBER 유형입니다.

암시적 변환은 다음 데이터 유형 사이에 발생할 수 있습니다.

- 문자와 숫자
- 문자와 날짜

명시적 변환: 데이터 유형을 변환하기 위해 내장 함수를 사용합니다. 예를 들어, CHAR 값을 DATE 값으로 변환하려면 TO_DATE를 사용하고 NUMBER 값으로 변환하려면 TO_NUMBER를 사용합니다.

데이터 유형 변환

1

```
date_of_joining DATE:= '02-Feb-2000';
```

2

```
date_of_joining DATE:= 'February 02,2000';
```

3

```
date_of_joining DATE:= TO_DATE('February  
02,2000','Month DD, YYYY');
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 유형 변환(계속)

DATE 데이터 유형의 암시적 및 명시적 변환:

1. 이 암시적 변환 예제는 date_of_joining 날짜를 할당합니다.
2. 할당된 날짜가 기본 형식이 아니기 때문에 PL/SQL이 오류 메시지를 표시합니다.
3. TO_DATE 함수를 사용하여 지정된 날짜를 명시적으로 특정 형식으로 변환하고 DATE 데이터 유형의 date_of_joining 변수에 날짜를 할당합니다.

중첩 블록

PL/SQL 블록은 중첩될 수 있습니다.

- 실행 섹션(BEGIN ... END)은 중첩 블록을 포함할 수 있습니다.
- 예외 섹션은 중첩 블록을 포함할 수 있습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 블록

PL/SQL이 SQL에 비해 뛰어난 점 중 하나는 명령문을 중첩할 수 있다는 것입니다. 실행문에서 허용되는 범위까지 무제한 블록을 중첩할 수 있으므로 중첩 블록을 명령문으로 만들 수 있습니다. 여러 업무 요구 사항을 지원하기 위해 실행 섹션에 논리적으로 관련된 많은 기능들이 포함된 코드가 있는 경우 실행 섹션을 더 작은 블록으로 나눌 수 있습니다. 예외 섹션도 중첩 블록을 포함할 수 있습니다.

중첩 블록

예제:

```
DECLARE
  outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(inner_variable);
    DBMS_OUTPUT.PUT_LINE(outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(outer_variable);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 블록(계속)

슬라이드의 예제는 외부(상위) 블록과 중첩(하위) 블록을 보여줍니다. `outer_variable` 변수는 외부 블록에서 선언되고 `inner_variable` 변수는 내부 블록에서 선언되었습니다.

`outer_variable`은 외부 블록에 대해서는 로컬 변수지만 내부 블록에 대해서는 전역 변수입니다. 내부 블록에서 이 변수에 액세스하면 PL/SQL은 먼저 내부 블록에서 해당 이름을 가진 로컬 변수를 찾습니다. 내부 블록에 동일한 이름의 변수가 없으므로 PL/SQL은 외부 블록에서 해당 변수를 찾습니다. 따라서 `outer_variable`은 모든 내부 블록에 대해 전역 변수로 간주됩니다. 슬라이드에 나오는 것처럼 내부 블록에서 이 변수에 액세스할 수 있습니다. PL/SQL 블록에서 선언된 변수는 해당 블록에 대해 로컬 변수로 간주되고 모든 서브 블록에 대해 전역 변수로 간주됩니다.

`inner_variable` 변수는 중첩 블록이 없기 때문에 내부 블록에 대해 전역 변수가 아니라 로컬 변수입니다. 이 변수는 내부 블록 내에서만 액세스할 수 있습니다. PL/SQL은 로컬로 선언된 변수를 찾지 못하면 외부 블록의 선언 섹션에서 찾습니다. PL/SQL은 외부 블록에서 내부 블록을 찾지 않습니다.

변수 범위 및 가시성

```
DECLARE
  father_name VARCHAR2(20):='Patrick';
  date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    child_name VARCHAR2(20):='Mike';
    date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||date_of_birth);
END;
/
```

①

②

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 범위 및 가시성

이 슬라이드의 블록 출력 결과는 다음과 같습니다.

```
anonymous block completed
Father's Name: Patrick
Date of Birth: 12-DEC-02
Child's Name: Mike
Date of Birth: 20-APR-72
```

아버지와 자녀에 대해 출력되는 생일을 조사합니다.

변수의 범위는 변수를 선언하고 액세스할 수 있는 프로그램의 영역입니다.

변수의 가시성은 수식자를 사용하지 않고도 변수에 액세스할 수 있는 프로그램의 영역입니다.

범위

- father_name 및 date_of_birth 변수는 외부 블록에서 선언됩니다. 이러한 변수는 해당 변수를 선언하고 액세스할 수 있는 블록의 범위를 가집니다. 따라서 이러한 변수의 범위는 외부 블록으로 제한됩니다.

변수 범위 및 가시성(계속)

범위(계속)

- `child_name` 및 `date_of_birth` 변수는 내부 블록이나 중첩 블록에서 선언됩니다. 이러한 변수는 중첩 블록 내에서만 액세스할 수 있고 외부 블록에서는 액세스할 수 없습니다. 변수가 범위를 벗어나면 PL/SQL이 변수를 저장하는 데 사용된 메모리를 해제하므로 이러한 변수를 참조할 수 없습니다.

표시 여부

- 외부 블록에서 선언된 `date_of_birth` 변수는 내부 블록에서도 범위를 가집니다. 그러나 내부 블록에 동일한 이름의 로컬 변수가 있기 때문에 내부 블록에서 이 변수를 볼 수 없습니다.
 1. PL/SQL 블록의 실행 섹션에 있는 코드를 살펴보십시오. 아버지의 이름과 자녀의 이름 및 생일을 출력할 수 있습니다. 여기서는 아버지의 생일을 볼 수 없기 때문에 자녀의 생일만 출력할 수 있습니다.
 2. 여기서는 아버지의 생일을 볼 수 있기 때문에 출력할 수 있습니다.

한 블록에서 동일한 이름의 여러 변수를 가질 수 없습니다. 그러나 두 개의 블록이 서로 다른 경우(중첩 블록) 동일한 이름의 여러 변수를 선언할 수 있습니다. 식별자가 나타내는 두 개의 항목은 서로 구분되고 한 항목을 변경해도 다른 항목에 영향을 주지 않습니다.

식별자의 명확한 지정

```
BEGIN <<outer>>
DECLARE
  father_name VARCHAR2(20):='Patrick';
  date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    child_name VARCHAR2(20):='Mike';
    date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: ' || father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '
                          || outer.date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: ' || child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || date_of_birth);
END;
END outer;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

식별자의 명확한 지정

수식자는 블록에 지정되는 레이블입니다. 수식자를 사용하여 범위를 가지고 있지만 볼 수는 없는 변수에 액세스할 수 있습니다. 코드를 살펴보면 이제 내부 블록에서 아버지의 생일과 자녀의 생일을 출력할 수 있습니다. 외부 블록에는 outer라는 레이블이 지정됩니다. 이 레이블을 사용하여 외부 블록에서 선언된 date_of_birth 변수에 액세스할 수 있습니다.

레이블 지정이 외부 블록으로 제한되지 않기 때문에 모든 블록에 레이블을 지정할 수 있습니다. 슬라이드에 있는 코드의 출력 결과는 다음과 같습니다.

```
Father's Name: Patrick
Date of Birth: 20-APR-72
Child's Name: Mike
Date of Birth: 12-DEC-02
```

변수 범위 결정

```
BEGIN <<outer>>
DECLARE
    sal      NUMBER(7,2) := 60000;
    comm     NUMBER(7,2) := sal * 0.20;
    message  VARCHAR2(255) := ' eligible for commission';
BEGIN
    DECLARE
        sal      NUMBER(7,2) := 50000;
        comm     NUMBER(7,2) := 0;
        total_comp NUMBER(7,2) := sal + comm;
    BEGIN
        message := 'CLERK not' || message;
        outer.comm := sal * 0.30;
    END;
    message := 'SALESMAN' || message;
END;
END outer;
/
```

①

②

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 범위 결정

슬라이드의 PL/SQL 블록을 평가합니다. 범위 지정 규칙에 따라 다음 값을 결정합니다.

1. 위치 1에서의 MESSAGE 값
2. 위치 2에서의 TOTAL_COMP 값
3. 위치 1에서의 COMM 값
4. 위치 1에서의 outer.COMM 값
5. 위치 2에서의 COMM 값
6. 위치 2에서의 MESSAGE 값

PL/SQL의 연산자

- 논리적
 - 산술
 - 연결
 - 연산 순서를 제어하기 위한 괄호
 - 지수 연산자(**)
- } SQL과 동일

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 연산자

표현식 내의 연산은 우선 순위에 따라 특정 순서로 수행됩니다. 다음 표는 높은 우선 순위에서 낮은 우선 순위 순서로 연산의 기본 순서를 보여줍니다.

연산자	연산
**	제곱
+, -	일치, 부정
*, /	곱하기, 나누기
+, -,	더하기, 빼기, 연결
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	비교
NOT	논리 부정
AND	결합
OR	포함

PL/SQL의 연산자

예제:

- 루프의 카운터를 증가시킵니다.

```
loop_count := loop_count + 1;
```

- 부울 플래그의 값을 설정합니다.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- 사원 번호에 값이 포함되어 있는지 확인합니다.

```
valid := (empno IS NOT NULL);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 연산자(계속)

널 값과 관련하여 다음 규칙을 기억해 두면 일반적으로 발생할 수 있는 실수를 피할 수 있습니다.

- 널이 포함된 비교 결과는 항상 NULL입니다.
- 논리 연산자 NOT을 널에 적용하면 NULL이 발생합니다.
- 조건 제어문에서 조건이 NULL을 반환하면 연관된 명령문 시퀀스는 실행되지 않습니다.

프로그래밍 지침

코드 유지 관리를 더욱 쉽게 만드는 방법:

- 주석을 사용하여 코드에 대한 설명 추가
- 코드의 대소문자 규칙 개발
- 식별자 및 기타 객체의 이름 지정 규칙 개발
- 들여쓰기로 가독성 향상

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그래밍 지침

PL/SQL 블록 개발 시 슬라이드의 프로그래밍 지침을 따르면 명확한 코드를 작성할 수 있을 뿐 아니라 유지 관리 비용을 줄일 수 있습니다.

코드 규칙

다음 표의 지침에 따라 대문자나 소문자로 코드를 작성하면 키워드와 명명된 객체를 구분하는데 도움이 됩니다.

범주	대소문자 규칙	예제
SQL 문	대문자	SELECT, INSERT
PL/SQL 키워드	대문자	DECLARE, BEGIN, IF
데이터 유형	대문자	VARCHAR2, BOOLEAN
식별자 및 파라미터	소문자	v_sal, emp_cursor, g_sal, p_empno
데이터베이스 테이블 및 열	소문자	employees, employee_id, department_id

코드 들여쓰기

명확성을 위해 코드의 각 레벨을 들여 씁니다.

예제:

```
BEGIN
  IF x=0 THEN
    y:=1;
  END IF;
END;
/
```

```
DECLARE
  deptno      NUMBER(4);
  location_id NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    deptno,
          location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';

  ...
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

코드 들여쓰기

명확성과 가독성을 높이기 위해 코드의 각 레벨을 들여 씁니다. 구조를 표시하기 위해 캐리지 리턴을 사용하여 행을 구분하고 공백과 탭을 사용하여 행을 들여 쓸 수 있습니다. 다음 IF 문의 가독성을 비교하십시오.

```
IF x>y THEN max:=x;ELSE max:=y;END IF;
```

```
IF x > y THEN
  max := x;
ELSE
  max := y;
END IF;
```

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- PL/SQL의 내장 SQL 함수 사용
- 중첩 블록을 작성하여 논리적으로 관련된 기능 분리
- 명시적 변환 수행 시기 결정
- 중첩 블록에서 변수 한정

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

PL/SQL은 SQL의 확장이기 때문에 SQL에 적용되는 일반 구문 규칙이 PL/SQL에도 적용됩니다. 블록의 실행 부분 내에 중첩 블록을 무제한 정의할 수 있습니다. 블록 내에 정의된 블록을 서브 블록이라고 합니다. 블록의 실행 부분에서만 블록을 중첩할 수 있습니다. 예외 섹션도 실행 섹션에 있기 때문에 중첩 블록을 가질 수 있습니다. 중첩 블록이 있는 경우 변수의 범위와 가시성이 정확해야 합니다. 상위 블록과 하위 블록에서 동일한 식별자를 사용하지 마십시오.

SQL에서 사용할 수 있는 대부분의 함수는 PL/SQL 표현식에서도 유효합니다. 변환 함수는 값의 데이터 유형을 변환합니다. 비교 연산자는 한 표현식을 다른 표현식과 비교합니다. 결과는 항상 TRUE, FALSE 또는 NULL입니다. 일반적으로 조건 제어문과 SQL 데이터 조작용의 WHERE 절에서 비교 연산자를 사용합니다. 관계 연산자를 사용하면 복합 표현식을 임의로 비교할 수 있습니다.

연습 3: 개요

이 연습에서는 다음 내용을 다룹니다.

- 범위 지정 및 중첩 규칙 검토
- PL/SQL 블록 작성 및 테스트

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 3: 개요

1, 2번 문제는 객관식입니다.

연습 3

PL/SQL 블록

```
DECLARE
  weight    NUMBER(3) := 600;
  message   VARCHAR2(255) := 'Product 10012';
BEGIN
  DECLARE
    weight    NUMBER(3) := 1;
    message   VARCHAR2(255) := 'Product 11001';
    new_locn  VARCHAR2(50) := 'Europe';
  BEGIN
    weight := weight + 1;
    new_locn := 'Western ' || new_locn;
  ① →
  END;
  weight := weight + 1;
  message := message || ' is in stock';
  new_locn := 'Western ' || new_locn;
  ② →
END;
/
```

1. 위에 제시된 PL/SQL 블록을 검토하여 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형 및 값을 판별합니다.
 - a. 1 위치에서의 weight 값:
 - b. 1 위치에서의 new_locn 값:
 - c. 2 위치에서의 weight 값:
 - d. 2 위치에서의 message 값:
 - e. 2 위치에서의 new_locn 값:

연습 3(계속)

범위 예제

```
DECLARE
    customer          VARCHAR2(50) := 'Womansport';
    credit_rating      VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
        customer NUMBER(7) := 201;
        name      VARCHAR2(25) := 'Unisports';
    BEGIN
        credit_rating := 'GOOD';
        ...
    END;
    ...
END;
/
```

2. 위에 제시된 PL/SQL 블록에서 다음 각 경우에 해당하는 값 및 데이터 유형을 판별합니다.
- a. 중첩 블록의 `customer` 값:
 - b. 중첩된 블록의 `name` 값:
 - c. 중첩 블록의 `credit_rating` 값:
 - d. 주 블록의 `customer` 값:
 - e. 주 블록의 `name` 값:
 - f. 주 블록의 `credit_rating` 값:

연습 3(계속)

3. lab_02_05_soln.sql을 실행한 동일한 워크시트에서 편집합니다. 해당 워크시트를 닫은 경우 열어서 lab_02_05_soln.sql을 실행한 다음 편집합니다.
 - a. 단일 행 주석 구문을 사용하여 바인드 변수를 생성하는 행을 주석 처리합니다.
 - b. 실행 섹션에서 다중 행 주석을 사용하여 바인드 변수에 값을 할당하는 행을 주석 처리합니다.
 - c. 데이터 유형이 VARCHAR2이고 크기가 15인 fname 및 데이터 유형이 NUMBER이고 크기가 10인 emp_sal이라는 두 변수를 선언합니다.
 - d. 다음 SQL 문을 실행 섹션에 포함시킵니다.

```
SELECT first_name, salary
      INTO fname, emp_sal FROM employees
      WHERE employee_id=110;
```
 - e. 'Hello World'를 출력하는 행을 'Hello'와 이름을 출력하도록 변경합니다. 필요한 경우 날짜를 표시하고 바인드 변수를 출력하는 행을 주석 처리합니다.
 - f. 적립 기금(PF)에 대한 사원의 부담금을 계산합니다. PF는 기본 급여의 12%이며 기본 급여는 급여의 45%입니다. 계산할 때는 바인드 변수를 사용합니다. 표현식을 하나만 사용하여 PF를 계산합니다. 사원의 급여 및 PF 부담금을 출력합니다.
 - g. 스크립트를 실행하고 lab_03_03_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF: 442.8
```

4. lab_03_04.sql 스크립트를 실행합니다. 이 스크립트는 employee_details라는 테이블을 생성합니다.
 - a. employee 및 employee_details 테이블에는 동일한 데이터가 있습니다. employee_details 테이블의 데이터를 갱신합니다. employees 테이블의 데이터는 갱신하거나 변경하지 마십시오.
 - b. lab_03_04b.sql 스크립트를 열어 파일의 코드를 확인합니다. 이 코드는 사용자가 입력한 사원 번호 및 부서 번호를 받아들입니다. 이 스크립트는 "소개" 단원에서 설명한 응용 프로그램을 개발하는 데 기본 구조로 사용됩니다.

4

Oracle 서버와 상호 작용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- PL/SQL 실행 블록에 직접 포함할 수 있는 SQL 문 판별
- PL/SQL의 DML 문으로 데이터 조작
- PL/SQL에서 트랜잭션 제어문 사용
- INTO 절을 사용하여 SQL 문에서 반환한 값 보유
- 암시적 커서와 명시적 커서 구별
- SQL 커서 속성 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 표준 SQL SELECT, INSERT, UPDATE, DELETE, MERGE 문을 PL/SQL 블록에 포함시키는 방법에 대해 설명합니다. 또한 PL/SQL에 DDL(데이터 정의어) 및 트랜잭션 제어문을 포함시키는 방법에 대해 설명합니다. 커서의 필요성에 대해 알아보고 두 가지 유형의 커서를 구분합니다. 암시적 커서와 함께 사용할 수 있는 다양한 SQL 커서 속성에 대해서도 설명합니다.

PL/SQL의 SQL 문

- **SELECT 명령을 사용하여 데이터베이스에서 행을 검색합니다.**
- **DML 명령을 사용하여 데이터베이스에서 행을 변경합니다.**
- **COMMIT, ROLLBACK 또는 SAVEPOINT 명령을 사용하여 트랜잭션을 제어합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SQL 문

PL/SQL 블록에서 SQL 문을 사용하여 데이터베이스 테이블에서 데이터를 검색하고 수정합니다. PL/SQL은 DML(데이터 조작용어) 및 트랜잭션 제어 명령을 지원합니다. DML 명령을 사용하여 데이터베이스 테이블에서 데이터를 수정할 수 있습니다. 그러나 PL/SQL 블록에서 DML 문과 트랜잭션 제어 명령을 사용할 경우 다음 사항에 유의하십시오.

- **END** 키워드는 트랜잭션의 끝이 아니라 PL/SQL 블록의 끝을 나타냅니다. 블록이 다중 트랜잭션을 확장할 수 있는 것과 마찬가지로 트랜잭션도 다중 블록을 확장할 수 있습니다.
- PL/SQL은 **CREATE TABLE**, **ALTER TABLE** 또는 **DROP TABLE**과 같은 DDL(데이터 정의어) 문을 직접 지원하지 않습니다. PL/SQL은 초기 바인딩을 지원하지므로 컴파일 시간이 실행 시간보다 오래 걸립니다. 응용 프로그램이 값을 전달하여 런타임에 데이터베이스 객체를 생성해야 하는 경우에는 초기 바인딩이 발생하지 않습니다. DDL 문은 직접 실행될 수 없습니다. DDL 문은 동적 SQL 문입니다. 동적 SQL 문은 런타임에 문자열로 작성되며 파라미터의 위치 표시자를 포함할 수 있습니다. 따라서 동적 SQL을 사용하여 PL/SQL에서 DDL 문을 실행할 수 있습니다. SQL 문을 인수로 취하는 **EXECUTE IMMEDIATE** 문을 사용하여 DDL 문을 실행합니다. **EXECUTE IMMEDIATE** 문은 동적 SQL 문의 구문을 분석하고 실행합니다. 이 과정 뒷부분에서 동적 SQL 및 **EXECUTE IMMEDIATE** 문에 대해 자세히 배우게 됩니다.

PL/SQL의 SQL 문(계속)

- PL/SQL은 GRANT나 REVOKE와 같은 DCL(데이터 제어어) 문을 지원하지 않습니다. DCL 문을 실행하려면 EXECUTE IMMEDIATE 문을 사용합니다.
- 데이터베이스를 영구적으로 변경하거나 변경 사항을 되돌리려면 트랜잭션 제어문을 사용합니다. COMMIT, ROLLBACK 및 SAVEPOINT는 사용되는 세 가지 주요 트랜잭션 제어 명령문입니다. COMMIT은 데이터베이스를 영구적으로 변경하는 데 사용됩니다. ROLLBACK은 마지막 COMMIT 후 데이터베이스에 대한 변경 사항을 폐기하기 위한 것입니다. SAVEPOINT는 트랜잭션 처리 시 중간 지점을 표시하는 데 사용됩니다. 트랜잭션 제어 명령은 PL/SQL에서 유효하므로 PL/SQL 블록의 실행 섹션에서 직접 사용할 수 있습니다.

PL/SQL의 SELECT 문

SELECT 문을 사용하여 데이터베이스에서 데이터를 검색합니다.

구문:

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SELECT 문

SELECT 문을 사용하여 데이터베이스에서 데이터를 검색합니다.

<i>select_list</i>	하나 이상의 열 리스트이며 SQL 표현식, 행 함수 또는 그룹 함수를 포함할 수 있습니다.
<i>variable_name</i>	검색한 값을 보유하는 스칼라 변수입니다.
<i>record_name</i>	검색한 값을 보유하는 PL/SQL 레코드입니다.
<i>Table</i>	데이터베이스 테이블 이름을 지정합니다.
<i>Condition</i>	PL/SQL 변수와 상수를 포함한 열 이름, 표현식, 상수 및 비교 연산자로 구성됩니다.

PL/SQL에서의 데이터 검색을 위한 지침

- 세미콜론(;)으로 각 SQL 문을 종료합니다.
- INTO 절을 사용하여 검색한 모든 값을 변수에 저장해야 합니다.
- WHERE 절은 선택 사항이며 입력 변수, 상수, 리터럴 및 PL/SQL 표현식을 지정하는 데 사용할 수 있습니다. 그러나 INTO 절을 사용할 때는 하나의 행만 패치(fetch)해야 합니다. 이 경우 WHERE 절 사용은 필수입니다.
- INTO 절의 변수 개수를 SELECT 절의 데이터베이스 열 개수와 동일하게 지정합니다. 위치적으로 일치하고 데이터 유형이 호환되어야 합니다.
- 그룹 함수는 테이블의 행 그룹에 적용되기 때문에 SQL 문에서 SUM과 같은 그룹 함수를 사용합니다.

PL/SQL의 SELECT 문

- INTO 절은 필수입니다.
- query는 하나의 행만 반환해야 합니다.

예제

```
SET SERVEROUTPUT ON
DECLARE
  fname VARCHAR2(25);
BEGIN
  SELECT first_name INTO fname
  FROM employees WHERE employee_id=200;
  DBMS_OUTPUT.PUT_LINE(' First Name is : ' || fname);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SELECT 문(계속)

INTO 절

INTO 절은 필수이며 SELECT와 FROM 절 사이에 위치합니다. INTO 절은 SQL이 SELECT 절에서 반환하는 값을 보유할 변수의 이름을 지정하는 데 사용됩니다. 선택한 각 항목에 대해 하나의 변수를 지정해야 하며, 변수의 순서는 선택한 항목과 일치해야 합니다.

INTO 절을 사용하여 PL/SQL 변수나 호스트 변수를 채웁니다.

query는 하나의 행만 반환해야 합니다.

PL/SQL 블록 내의 SELECT 문은 내장 SQL의 ANSI 규정에 따라 query는 하나의 행만 반환해야 한다는 규칙이 적용됩니다. 반환되는 행이 없거나 여러 개일 경우 오류가 발생합니다.

PL/SQL은 표준 예외를 발생시켜 이러한 오류를 관리하는데, 이 오류는 NO_DATA_FOUND 및 TOO_MANY_ROWS 예외를 사용하여 블록의 예외 섹션에서 처리할 수 있습니다. SQL 문에 WHERE 조건을 포함시켜서 명령문이 단일 행을 반환하도록 합니다. 이 과정 뒷부분에서 예외 처리에 대해 배우게 됩니다.

테이블에서 다중 행을 검색하고 데이터를 조작하는 방법

INTO 절이 포함된 SELECT 문은 한 번에 하나의 행만 검색할 수 있습니다. 다중 행을 검색하고 데이터를 조작해야 할 경우 명시적 커서를 사용하면 됩니다. 이 단원 후반부에서 커서에 대해 배우게 됩니다.

PL/SQL에서 데이터 검색

지정된 사원의 hire_date 및 salary를 검색합니다.

예제:

```
DECLARE
  emp_hiredate    employees.hire_date%TYPE;
  emp_salary      employees.salary%TYPE;
BEGIN
  SELECT    hire_date, salary
  INTO      emp_hiredate, emp_salary
  FROM      employees
  WHERE     employee_id = 100;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에서 데이터 검색

슬라이드의 예제에서 emp_hiredate 및 emp_salary 변수는 PL/SQL 블록의 선언 섹션에서 선언됩니다. 실행 섹션에서 employees 테이블에서 employee_id가 100인 사원의 hire_date 및 salary 열 값을 검색하여 각각 emp_hiredate 및 emp_salary 변수에 저장합니다. SELECT 문과 함께 INTO 절이 PL/SQL 변수로 데이터베이스 열 값을 읽어 들이는 방식을 살펴 보십시오.

주: SELECT 문은 hire_date를 검색한 다음 salary를 검색합니다. 따라서 INTO 절에 있는 변수의 순서도 동일해야 합니다. 예를 들어, 슬라이드의 명령문에서 emp_hiredate와 emp_salary의 순서를 바꾸면 오류가 발생합니다.

PL/SQL에서 데이터 검색

지정된 부서에 속한 모든 사원의 급여 합계를 반환합니다.

예제:

```
SET SERVEROUTPUT ON
DECLARE
    sum_sal    NUMBER(10,2);
    deptno     NUMBER NOT NULL := 60;
BEGIN
    SELECT  SUM(salary)  -- group function
    INTO sum_sal FROM employees
    WHERE   department_id = deptno;
    DBMS_OUTPUT.PUT_LINE ('The sum of salary is '
        || sum_sal);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에서 데이터 검색(계속)

슬라이드의 예제에서 sum_sal 및 deptno 변수는 PL/SQL 블록의 선언 섹션에서 선언됩니다. 실행 섹션에서 department_id가 60인 부서에 근무하는 사원의 전체 급여는 SQL 집계 함수 SUM을 사용하여 계산됩니다. 계산된 전체 급여는 sum_sal 변수에 할당됩니다.

주: 그룹 함수는 PL/SQL 구문에서 사용할 수 없습니다. 예제에 나오는 것처럼 PL/SQL 블록 내의 SQL 문에서 사용됩니다. 다음과 같은 방식으로 그룹 함수를 사용할 수 없습니다.

```
sum_sal := SUM(employees.salary);
```

슬라이드에 있는 PL/SQL 블록의 출력 결과는 다음과 같습니다.

```
anonymous block completed
The sum of salary is 28800
```

이름 지정 규칙

```
DECLARE
  hire_date      employees.hire_date%TYPE;
  sysdate        hire_date%TYPE;
  employee_id     employees.employee_id%TYPE := 176;
BEGIN
  SELECT          hire_date, sysdate
  INTO            hire_date, sysdate
  FROM            employees
  WHERE           employee_id = employee_id;
END;
/
```

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause:      The number specified in exact fetch is less than the rows returned.
*Action:     Rewrite the query or change number of rows requested
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

이름 지정 규칙

모호할 가능성이 있는 SQL 문에서 데이터베이스 열의 이름은 로컬 변수의 이름보다 우선합니다. 슬라이드에 표시된 예제는 다음과 같이 정의됩니다. `employee_id 176`의 `employees` 테이블에서 채용 날짜와 오늘 날짜를 검색합니다. 이 예제는 `WHERE` 절에서 PL/SQL 변수 이름이 `employees` 테이블의 데이터베이스 열 이름과 같기 때문에 처리되지 않는 런타임 예외를 발생시킵니다.

Oracle 서버는 `WHERE` 절에 나오는 `last_name`이 모두 데이터베이스 열을 참조하는 것으로 간주하기 때문에 다음 `DELETE` 문은 `employees` 테이블에서 성이 "King"인 사원뿐 아니라 성이 null이 아닌 사원도 모두 제거합니다.

```
DECLARE
  last_name VARCHAR2(25) := 'King';
BEGIN
  DELETE FROM employees WHERE last_name = last_name;
  . . .
```


이름 지정 규칙

- **WHERE 절에서 모호성을 방지하기 위해 이름 지정 규칙을 사용합니다.**
- **데이터베이스 열 이름을 식별자로 사용하지 않습니다.**
- **PL/SQL은 먼저 데이터베이스를 검사하여 테이블의 열이 있는지 확인하기 때문에 구문 오류가 발생할 수 있습니다.**
- **로컬 변수와 형식 파라미터의 이름은 데이터베이스 테이블의 이름보다 우선합니다.**
- **데이터베이스 테이블 열의 이름은 로컬 변수의 이름보다 우선합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

이름 지정 규칙(계속)

데이터베이스 열 이름과 PL/SQL 변수 이름을 구분하는 이름 지정 규칙을 준수하여 WHERE 절에서 모호성을 방지하십시오.

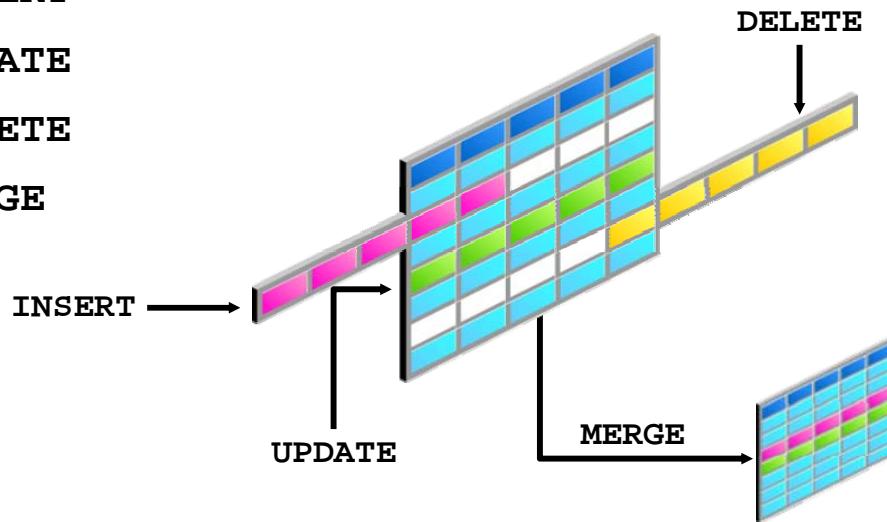
- 데이터베이스 열과 식별자의 이름은 명확해야 합니다.
- PL/SQL은 먼저 데이터베이스를 검사하여 테이블의 열이 있는지 확인하기 때문에 구문 오류가 발생할 수 있습니다.

주: SELECT 절의 모든 식별자는 데이터베이스 열 이름이기 때문에 SELECT 절에서 모호성이 발생할 가능성은 없습니다. INTO 절의 모든 식별자는 PL/SQL 변수이기 때문에 INTO 절에서 모호성이 발생할 가능성은 없습니다. 오직 WHERE 절에만 혼동이 발생할 가능성이 있습니다.

PL/SQL을 사용하여 데이터 조작

다음 DML 명령을 사용하여 데이터베이스 테이블을 변경합니다.

- INSERT
- UPDATE
- DELETE
- MERGE



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL을 사용하여 데이터 조작

DML 명령을 사용하여 데이터베이스의 데이터를 조작합니다. INSERT, UPDATE, DELETE, MERGE 등의 DML 명령은 PL/SQL에서 제약 없이 실행할 수 있습니다. 행 잠금 및 테이블 잠금은 PL/SQL 코드에 COMMIT 또는 ROLLBACK 문을 포함시켜서 해제합니다.

- INSERT 문은 테이블에 새 행을 추가합니다.
- UPDATE 문은 테이블에서 기존 행을 수정합니다.
- DELETE 문은 테이블에서 행을 제거합니다.
- MERGE 문은 한 테이블의 행을 선택하여 다른 테이블을 갱신하거나 다른 테이블에 삽입합니다. 대상 테이블을 갱신할지 대상 테이블에 삽입할지 여부는 ON 절의 조건에 따라 결정됩니다.

주: MERGE는 결정적 명령문(deterministic statement)입니다. 즉, 동일한 MERGE 문에서 대상 테이블의 동일한 행을 여러 번 갱신할 수 없습니다. 대상 테이블에서 객체에 대해 INSERT 및 UPDATE 권한이 있고 소스 테이블에서 SELECT 권한이 있어야 합니다.

데이터 삽입

EMPLOYEES 테이블에 새로운 직원 정보를 추가합니다.

예제:

```
BEGIN
  INSERT INTO employees
    (employee_id, first_name, last_name, email,
     hire_date, job_id, salary)
    VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores',
            'RCORES',sysdate, 'AD_ASST', 4000);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 삽입

슬라이드의 예제에서 INSERT 문은 PL/SQL 블록 내에서 employees 테이블에 레코드를 삽입하는 데 사용됩니다. PL/SQL 블록에서 INSERT 명령을 사용하여 다음 작업을 수행할 수 있습니다.

- USER 및 SYSDATE와 같은 SQL 함수를 사용합니다.
- 기존 데이터베이스 시퀀스를 사용하여 Primary Key 값 생성
- PL/SQL 블록에서 값 파생

주: employees 테이블의 데이터는 변경되지 않아야 합니다. 따라서 이 테이블에서 삽입, 갱신 및 삭제는 허용되지 않습니다.

데이터 갱신

자재 담당자 직책을 가진 모든 사원의 급여를 인상합니다.

예제:

```
DECLARE
    sal_increase    employees.salary%TYPE := 800;
BEGIN
    UPDATE          employees
    SET              salary = salary + sal_increase
    WHERE            job_id = 'ST_CLERK';
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 갱신

UPDATE 문의 SET 절에 모호한 점이 있습니다. 할당 연산자 왼쪽에 있는 식별자는 항상 데이터베이스 열이지만 오른쪽에 있는 식별자는 데이터베이스 열일 수도 있고 PL/SQL 변수일 수도 있기 때문입니다. WHERE 절에서 열 이름과 식별자 이름이 동일할 경우 Oracle 서버는 먼저 데이터베이스에서 이름을 검색합니다.

WHERE 절은 영향을 받는 행을 결정하는 데 사용됩니다. 수정된 행이 없을 경우 PL/SQL의 SELECT 문과 달리 오류가 발생하지 않습니다.

주: PL/SQL 변수 할당은 항상 :=를 사용하고, SQL 열 할당은 항상 =를 사용합니다.

데이터 삭제

employees 테이블에서 부서 10에 속하는 행을 삭제합니다.

예제:

```
DECLARE
  deptno    employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM employees
  WHERE department_id = deptno;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 삭제

DELETE 문은 테이블에서 불필요한 행을 제거합니다. 무결성 제약 조건이 없는 경우 WHERE 절을 사용하지 않으면 테이블의 모든 행이 제거될 수 있습니다.

행 병합

employees 테이블과 일치하도록 copy_emp 테이블에 행을 삽입하거나 행을 갱신합니다.

```
DECLARE
    empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
    USING employees e
    ON (e.employee_id = c.empno)
    WHEN MATCHED THEN
        UPDATE SET
            c.first_name      = e.first_name,
            c.last_name       = e.last_name,
            c.email           = e.email,
            . . .
    WHEN NOT MATCHED THEN
        INSERT VALUES(e.employee_id, e.first_name, e.last_name,
            . . . ,e.department_id);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

행 병합

MERGE 문은 한 테이블의 데이터를 사용하여 다른 테이블에 행을 삽입하거나 다른 테이블의 행을 갱신합니다. 각 행은 equijoin 조건에 따라 대상 테이블에 삽입되거나 대상 테이블에서 갱신됩니다.

슬라이드에 표시된 예제에서는 COPY_EMP 테이블의 employee_id가 employees 테이블의 employee_id와 일치하는지 조회합니다. 일치되는 행이 발견되면 해당 행이 employees 테이블의 행과 일치하도록 갱신됩니다. 행을 찾지 못하면 copy_emp 테이블에 행이 삽입됩니다. 다음 노트 페이지에 PL/SQL 블록에서 MERGE를 사용하는 자세한 예제가 나와 있습니다.

행 병합(계속)

```
DECLARE
    empno EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
    USING employees e
    ON (e.employee_id = c.empno)
WHEN MATCHED THEN
    UPDATE SET
        c.first_name = e.first_name,
        c.last_name = e.last_name,
        c.email = e.email,
        c.phone_number = e.phone_number,
        c.hire_date = e.hire_date,
        c.job_id = e.job_id,
        c.salary = e.salary,
        c.commission_pct = e.commission_pct,
        c.manager_id = e.manager_id,
        c.department_id = e.department_id
WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
        e.email, e.phone_number, e.hire_date, e.job_id,
        e.salary, e.commission_pct, e.manager_id,
        e.department_id);
END;
/
```

SQL 커서

- 커서는 Oracle 서버에서 할당한 전용(private) 메모리 영역에 대한 포인터입니다.
- 다음과 같은 두 가지 유형의 커서가 있습니다.
 - 암시적: Oracle 서버가 SQL 문을 처리하기 위해 내부적으로 생성하고 관리합니다.
 - 명시적: 프로그래머가 명시적으로 선언합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 커서

이미 앞에서 PL/SQL 블록에서 단일 행을 반환하는 SQL 문을 포함하는 방법에 대해 배웠습니다. SQL 문으로 검색된 데이터는 INTO 절을 사용하여 변수에 보유하고 있어야 합니다.

SQL 문이 처리되는 위치

Oracle 서버는 SQL 문을 처리하기 위해 컨텍스트 영역이라는 전용(private) 메모리 영역을 할당합니다. 이 영역에서 SQL 문의 구문을 분석하고 처리합니다. 처리에 필요한 정보와 처리 후 검색된 정보는 모두 이 영역에 저장됩니다. 이 영역은 Oracle 서버가 내부적으로 관리하기 때문에 사용자가 제어할 수 없습니다.

커서는 컨텍스트 영역에 대한 포인터입니다. 그러나 이 커서는 암시적 커서이며 Oracle 서버에 의해 자동으로 관리됩니다. 실행 블록에서 SQL 문을 실행하면 PL/SQL은 암시적 커서를 생성합니다.

SQL 커서(계속)

다음과 같은 두 가지 유형의 커서가 있습니다.

- **암시적:** 암시적 커서는 Oracle 서버가 생성하고 관리합니다. 여러분은 이 커서에 대한 액세스 권한이 없습니다. Oracle 서버는 SQL 문을 실행해야 하는 경우에 이러한 커서를 생성합니다.
- **명시적:** 프로그래머가 데이터베이스 테이블에서 다중 행을 검색하고 검색된 각 행에 포인터를 지정하며 한 번에 하나씩 행을 처리해야 하는 경우가 있습니다. 이러한 경우에 업무 요구 사항에 따라 명시적으로 커서를 선언할 수 있습니다. 프로그래머가 선언하는 커서를 명시적 커서라고 합니다. 이러한 커서는 PL/SQL 블록의 선언 섹션에서 선언합니다. 선언 섹션에서 변수와 예외도 선언할 수 있다는 사실을 기억하십시오.

암시적 커서에 대한 SQL 커서 속성

SQL 커서 속성을 사용하면 SQL 문의 결과를 테스트할 수 있습니다.

SQL%FOUND	가장 최근의 SQL 문의 한 행 이상을 반환한 경우 TRUE로 평가되는 부울 속성
SQL%NOTFOUND	가장 최근의 SQL 문의 한 행도 반환하지 않은 경우 TRUE로 평가되는 부울 속성
SQL%ROWCOUNT	가장 최근의 SQL 문의 의해 영향을 받은 행 수를 나타내는 정수 값

ORACLE

Copyright © 2009, Oracle. All rights reserved.

암시적 커서에 대한 SQL 커서 속성

SQL 커서 속성을 사용하면 암시적 커서가 마지막으로 사용되었을 때의 실행 결과를 평가할 수 있습니다. 이러한 속성은 SQL 문의 아닌 PL/SQL 문에서 사용합니다.

알맞은 DML 명령을 실행한 후 정보를 수집하기 위해 블록의 실행 섹션에서 SQL%ROWCOUNT, SQL%FOUND 및 SQL%NOTFOUND 속성을 테스트할 수 있습니다. DML 문의 기본 테이블의 행에 영향을 주지 않으면 PL/SQL은 오류를 반환하지 않습니다. 그러나 SELECT 문의 아무 행도 검색하지 않으면 PL/SQL은 예외를 반환합니다.

이 속성에는 SQL이라는 접두어가 붙습니다. 이러한 커서 속성은 암시적 커서와 함께 사용되며 암시적 커서는 PL/SQL에 의해 자동으로 생성되므로 이름을 알 수 없습니다. 따라서 커서 이름 대신 SQL을 사용합니다.

SQL%NOTFOUND 속성은 SQL%FOUND의 반대입니다. 이 속성은 루프에서 종료 조건으로 사용될 수 있습니다. 이 속성은 변경된 행이 없을 경우 예외가 반환되지 않기 때문에 UPDATE 및 DELETE 문의 유용합니다.

이 과정 뒷부분에서 명시적 커서 속성에 대해 배우게 됩니다.

암시적 커서에 대한 SQL 커서 속성

employees 테이블에서 지정된 사원 ID를 가진 행을 삭제합니다. 삭제된 행의 개수를 출력합니다.

예제

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  empno employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM employees
  WHERE employee_id = empno;
  :rows_deleted := (SQL%ROWCOUNT ||
                    ' row deleted. ');
END;
/
PRINT rows_deleted
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

암시적 커서에 대한 SQL 커서 속성(계속)

슬라이드의 예제는 employees 테이블에서 employee_id가 176인 행을 삭제합니다. SQL%ROWCOUNT 속성을 사용하여 삭제된 행 수를 출력할 수 있습니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- DML 문, 트랜잭션 제어문 및 DDL 문을 PL/SQL에 포함
- PL/SQL에서 모든 `SELECT` 문에 반드시 필요한 `INTO` 절 사용
- 암시적 커서와 명시적 커서 구별
- SQL 커서 속성을 사용하여 SQL 문의 결과 판별

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

DML 명령과 트랜잭션 제어문은 PL/SQL 프로그램에서 제약 없이 사용할 수 있습니다. 그러나 DDL 명령은 직접 사용할 수 없습니다.

PL/SQL 블록의 `SELECT` 문은 한 행만 반환할 수 있습니다. 반드시 `INTO` 절을 사용하여 `SELECT` 문으로 검색된 값을 보유해야 합니다.

커서는 메모리 영역에 대한 포인터입니다. 다음과 같은 두 가지 유형의 커서가 있습니다. 암시적 커서는 Oracle 서버가 SQL 문을 실행하기 위해 내부적으로 생성하고 관리합니다. 이러한 커서와 함께 SQL 커서 속성을 사용하여 SQL 문의 결과를 판별할 수 있습니다. 명시적 커서는 프로그래머가 선언합니다.

연습 4: 개요

이 연습에서는 다음 내용을 다룹니다.

- 테이블에서 데이터 선택
- 테이블에 데이터 삽입
- 테이블의 데이터 갱신
- 테이블에서 레코드 삭제

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 4

1. departments 테이블에서 최대 부서 ID를 선택하여 max_deptno 변수에 저장하는 PL/SQL 블록을 생성합니다. 최대 부서 ID를 표시합니다.
 - a. 선언 섹션에서 NUMBER 유형의 max_deptno 변수를 선언합니다.
 - b. BEGIN 키워드로 실행 섹션을 시작하고 departments 테이블에서 최대 department_id를 검색하는 SELECT 문을 포함시킵니다.
 - c. max_deptno를 표시하고 실행 블록을 종료합니다.
 - d. 스크립트를 실행하고 lab_04_01_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 270
```

2. 연습 1에서 생성한 PL/SQL 블록을 departments 테이블에 새 부서를 삽입하도록 수정합니다.
 - a. 스크립트 lab_04_01_soln.sql을 엽니다.
departments.department_name 유형의 dept_name 및 NUMBER 유형의 dept_id라는 두 개의 변수를 선언합니다. 선언 섹션에서 dept_name에 "Education"을 할당합니다.
 - b. 앞에서 이미 departments 테이블에서 현재 최대 부서 번호를 검색했습니다. 이 부서 번호에 10을 더하여 해당 결과를 dept_id에 할당합니다.
 - c. departments 테이블의 department_name, department_id 및 location_id 열에 데이터를 삽입하는 INSERT 문을 포함시킵니다.
department_name, department_id에는 dept_name, dept_id의 값을 사용하고 location_id에는 NULL을 사용합니다.
 - d. SQL 속성 SQL%ROWCOUNT를 사용하여 적용되는 행 수를 표시합니다.
 - e. select 문을 실행하여 새 부서가 삽입되었는지 확인합니다. "/"로 PL/SQL 블록을 종료하고 스크립트에 SELECT 문을 포함시킵니다.
 - f. 스크립트를 실행하고 lab_04_02_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 280
SQL%ROWCOUNT gives 1
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	280	Education	(null)	(null)

연습 4(계속)

3. 연습 문제 2에서 `location_id`를 널로 설정했습니다. 새 부서의 `location_id`를 3000으로 갱신하는 PL/SQL 블록을 생성합니다. 변수 `dept_id` 값을 사용하여 행을 갱신합니다.
 - a. `BEGIN` 키워드로 실행 블록을 시작합니다. 새 부서(`dept_id = 280`)의 `location_id`를 3000으로 설정하는 `UPDATE` 문을 포함시킵니다.
 - b. `END` 키워드로 실행 블록을 종료합니다. `"/` 로 PL/SQL 블록을 종료하고 갱신한 부서가 표시되도록 `SELECT` 문을 포함시킵니다.
 - c. 추가한 부서를 삭제하도록 `DELETE` 문을 포함시킵니다.
 - d. 스크립트를 실행하고 `lab_04_03_soln.sql`로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

anonymous block completed			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Education		3000
1 rows selected			
1 rows deleted			

4. `lab_03_05b.sql` 스크립트를 엽니다.
 - a. 코드에 중첩 블록이 있는지 확인합니다. 외부 블록의 선언 섹션을 볼 수 있습니다. `"INCLUDE EXECUTABLE SECTION OF OUTER BLOCK HERE"` 라는 주석을 찾아 실행 섹션을 시작합니다.
 - b. HR(Human Resources) 부서에서 근무하는 사원의 `employee_id`를 검색하는 단일 `SELECT` 문을 포함시킵니다. `INTO` 절을 사용하여 검색된 값을 `emp_authorization` 변수에 저장합니다.
 - c. 스크립트를 `lab_04_04_soln.sql`로 저장합니다.

5

제어 구조 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 제어 구조의 사용법 및 유형 식별
- IF 문 구성
- CASE 문 및 CASE 식 사용
- 다양한 LOOP 문 구성 및 식별
- 조건부 제어 구조 사용 지침 사용

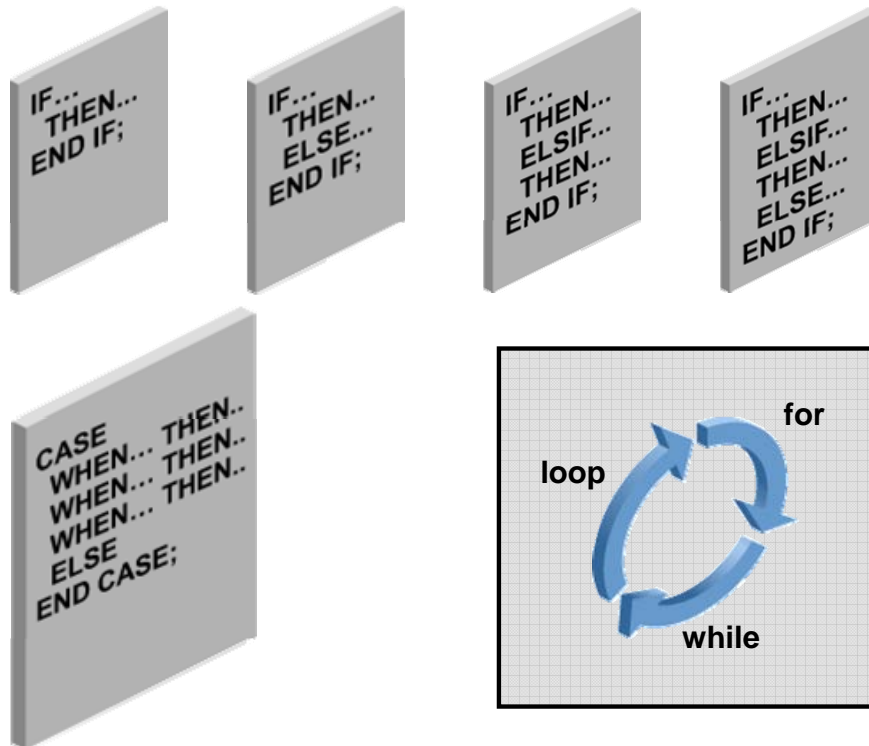
ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이미 앞에서 선언 섹션 및 실행 섹션을 포함하는 PL/SQL 블록을 작성하는 방법에 대해 배웠습니다. 또한 실행 블록에 표현식과 SQL 문을 포함시키는 방법에 대해서도 설명했습니다. 이 단원에서는 PL/SQL 블록에서 IF 문, CASE 식 및 LOOP 구조와 같은 제어 구조를 사용하는 방법에 대해 설명합니다.

실행 흐름 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

실행 흐름 제어

다양한 제어 구조를 사용하여 PL/SQL 블록 내에서 명령문의 논리적 흐름을 변경할 수 있습니다. 이 단원에서는 IF 문을 사용하는 조건 생성자, CASE 식 및 LOOP 제어 구조와 같이 세 가지 유형의 PL/SQL 제어 구조에 대해 설명합니다.

IF 문

구문:

```
IF condition THEN
    statements;
[ELSIF condition THEN
    statements;]
[ELSE
    statements;]
END IF;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IF 문

PL/SQL IF 문의 구조는 다른 절차적 언어로 된 IF 문의 구조와 비슷합니다. 이 구조에서는 PL/SQL에서 조건에 따라 선별적으로 작업을 수행할 수 있습니다.

이 구문에서 다음이 적용됩니다.

조건	TRUE, FALSE 또는 NULL을 반환하는 부울 변수 또는 표현식입니다.
THEN	부울 표현식을 뒤에 나오는 명령문 시퀀스와 연관시키는 절을 시작합니다.
명령문	하나 이상의 PL/SQL 또는 SQL 문이 올 수 있습니다. 여러 중첩 IF, ELSE 및 ELSIF 문을 포함하는 IF 문이 추가로 포함될 수 있습니다. THEN 절의 명령문은 연관된 IF 절의 조건이 TRUE로 평가되는 경우에만 실행됩니다.

IF 문(계속)

이 구문에서 다음이 적용됩니다.

ELSIF	부울 표현식을 시작하는 키워드입니다. 첫번째 조건의 결과가 FALSE 또는 NULL이면 ELSIF 키워드는 추가 조건을 시작합니다.
ELSE	IF 및 ELSIF로 시작되는 앞의 술어 중 어느 것도 TRUE가 아닌 경우에만 실행되는 기본 절을 시작합니다. True인 앞의 술어가 True일 수 있는
나중	술어를 선점할 수 있도록 차례로 테스트가 실행됩니다.
END IF	END IF는 IF 문의 끝을 표시합니다.

주: ELSIF 및 ELSE는 IF 문에서 선택적으로 사용됩니다. IF 문에서 ELSIF 키워드는 개수에 제한이 없지만 ELSE 키워드는 하나만 있어야 합니다. END IF는 IF 문의 끝을 표시하며 세미콜론으로 종료해야 합니다.

간단한 IF 문

```
DECLARE
  myage number:=31;
BEGIN
  IF myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  END IF;
END;
/
```

anonymous block completed

ORACLE

Copyright © 2009, Oracle. All rights reserved.

간단한 IF 문

이 슬라이드는 THEN 절을 사용하는 간단한 IF 문의 예제를 보여줍니다. myage 변수는 31로 초기화됩니다. Myage가 11 이상이기 때문에 IF 문의 조건이 FALSE를 반환합니다. 그러므로 이 제어는 THEN 절에 도달하지 않습니다. ELSE 및 ELSIF의 사용을 살펴보기 위해 이 예제에 코드를 추가하도록 합니다.

IF 문에는 AND, OR 및 NOT과 같은 논리 연산자와 관련된 조건식이 여러 개 있을 수 있습니다. 예를 들면 다음과 같습니다.

```
IF (myfirstname='Christopher' AND myage <11)
...
```

조건은 AND 연산자를 사용하므로 두 조건이 모두 TRUE로 평가되는 경우에만 TRUE로 평가됩니다. 조건식 수에는 제한이 없습니다. 그러나 이러한 명령문은 해당 논리 연산자와 관련되어 있어야 합니다.

IF THEN ELSE 문

```
SET SERVEROUTPUT ON
DECLARE
myage number:=31;
BEGIN
IF myage < 11
  THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IF THEN ELSE 문

이전 슬라이드의 코드에 ELSE 절이 추가되었습니다. 조건은 변경되지 않았으므로 여전히 FALSE로 평가됩니다. THEN 절의 명령문은 조건이 TRUE를 반환하는 경우에만 실행됩니다. 이 경우 조건이 FALSE를 반환하므로 제어가 ELSE 문으로 이동합니다. 블록 출력 결과가 슬라이드에 표시됩니다.

IF ELSIF ELSE 절

```
DECLARE
myage number:=31;
BEGIN
IF myage < 11
THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
ELSIF myage < 20
THEN
    DBMS_OUTPUT.PUT_LINE(' I am young ');
ELSIF myage < 30
THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
ELSIF myage < 40
THEN
    DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
ELSE
    DBMS_OUTPUT.PUT_LINE(' I am always young ');
END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IF ELSIF ELSE 절

이제 IF 절은 다중 ELSIF 절과 단일 ELSE 절을 포함합니다. ELSIF 절에는 ELSE 절과는 달리 조건이 있을 수 있습니다. ELSIF의 조건 뒤에는 ELSIF의 조건이 TRUE를 반환하는 경우에 실행되는 THEN 절이 와야 합니다.

여러 개의 ELSIF 절이 있을 때 첫번째 조건이 FALSE 또는 NULL이면 제어가 다음 ELSIF 절로 이동합니다. 조건은 맨 위부터 하나씩 차례로 평가됩니다. 모든 조건이 FALSE 또는 NULL이면 ELSE 절에 있는 명령문이 실행됩니다. 최종 ELSE 절은 선택 사항입니다.

IF 문의 NULL 값

```
DECLARE
myage number;
BEGIN
IF myage < 11
THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IF 문의 NULL 값

이 슬라이드의 예제에서는 `myage` 변수가 선언되었으나 초기화되지 않았습니다. IF 문에 있는 조건은 TRUE 또는 FALSE가 아닌 NULL을 반환합니다. 이 경우 제어는 ELSE 문으로 이동합니다.

지침:

- 충족되는 조건에 따라 선별적으로 작업을 수행할 수 있습니다.
- 코드 작성 시에는 키워드 단어를 기억해야 합니다.
 - ELSIF는 한 단어입니다.
 - END IF는 두 단어입니다.
- 제어 부울 조건이 TRUE인 경우 연관된 명령문 시퀀스가 실행되고 제어 부울 조건이 FALSE 또는 NULL인 경우 연관된 명령문 시퀀스가 무시됩니다. ELSIF 절의 수에는 제한이 없습니다.
- 명확한 구분을 위해 조건부로 실행되는 명령문은 들여 씁니다.

CASE 식

- CASE 식은 결과를 선택하여 반환합니다.
- 결과를 선택하기 위해 CASE 식은 표현식을 사용합니다.
이러한 표현식에서 반환되는 값은 여러 대안 중 하나를 선택하는 데 사용됩니다.

```
CASE selector
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CASE 식

CASE 식은 하나 이상의 대안에 기반한 결과를 반환합니다. 결과를 반환하기 위해 CASE 식은 선택자를 사용합니다. 이 선택자의 값을 사용하여 여러 대안 중 하나를 반환합니다. 선택자 다음에는 순차적으로 검사되는 하나 이상의 WHEN 절이 옵니다. 선택자의 값에 따라 반환 결과가 결정됩니다. 선택자의 값이 WHEN 절 표현식의 값과 같으면 WHEN 절이 실행되고 해당 결과가 반환됩니다.

또한 PL/SQL은 다음과 같은 형식의 검색된 CASE 식을 제공합니다.

```
CASE
  WHEN search_condition1 THEN result1
  WHEN search_condition2 THEN result2
  ...
  WHEN search_conditionN THEN resultN
  [ELSE resultN+1]
END;
```

검색된 CASE 식에는 선택자가 없습니다. 또한 WHEN 절은 모든 유형의 값을 생성할 수 있는 표현식이 아닌 부울 값을 생성하는 검색 조건을 포함합니다.

CASE 식: 예제

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DECLARE
    grade CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    appraisal :=
        CASE grade
            WHEN 'A' THEN 'Excellent'
            WHEN 'B' THEN 'Very Good'
            WHEN 'C' THEN 'Good'
            ELSE 'No such grade'
        END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || grade || '
                          Appraisal ' || appraisal);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CASE 식: 예제

이 슬라이드의 예제에서 CASE 식은 grade 변수의 값을 표현식으로 사용합니다. 이 값은 치환 변수를 사용하여 유저로부터 받아들입니다. 사용자가 입력한 값에 따라 CASE 식은 grade 값에 준하여 계산되는 appraisal 변수 값을 반환합니다. grade 값으로 a 또는 A를 입력할 때 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Grade: A Appraisal Excellent
```

검색된 CASE 식

```
DECLARE
  grade CHAR(1) := UPPER('&grade');
  appraisal VARCHAR2(20);
BEGIN
  appraisal :=
    CASE
      WHEN grade = 'A' THEN 'Excellent'
      WHEN grade IN ('B','C') THEN 'Good'
      ELSE 'No such grade'
    END;
  DBMS_OUTPUT.PUT_LINE ('Grade: ' || grade || '
                          Appraisal ' || appraisal);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

검색된 CASE 식

앞의 예제에서는 하나의 테스트 표현식(grade 변수)만 사용했습니다. WHEN 절은 이 테스트 표현식과 값을 비교했습니다.

검색된 CASE 문에는 테스트 표현식이 없습니다. 대신에 WHEN 절에는 부울 값을 생성하는 표현식이 포함되어 있습니다. 이 슬라이드에서는 동일한 예제를 재작성하여 검색된 CASE 문을 보여줍니다.

CASE 문

```
DECLARE
    deptid NUMBER;
    deptname VARCHAR2(20);
    emps NUMBER;
    mngid NUMBER:= 108;
BEGIN
    CASE mngid
        WHEN 108 THEN
            SELECT department_id, department_name
            INTO deptid, deptname FROM departments
            WHERE manager_id=108;
            SELECT count(*) INTO emps FROM employees
            WHERE department_id=deptid;
        WHEN 200 THEN
            ...
        END CASE;
    DBMS_OUTPUT.PUT_LINE ('You are working in the '|| deptname||
    ' department. There are '||emps ||' employees in this
    department');
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CASE 문

IF 문의 사용을 상기해 보십시오. n개의 PL/SQL 문을 THEN 절과 ELSE 절에 포함시킬 수 있습니다. 마찬가지로 CASE 문에도 명령문을 포함시킬 수 있습니다. CASE 문은 다중 IF 문과 ELSIF 문에 비해 훨씬 읽기 쉽습니다.

CASE 식과 CASE 문의 차이점

CASE 식은 조건을 평가하여 값을 반환하지만 CASE 문은 조건을 평가하여 작업을 수행합니다. CASE 문은 완전한 PL/SQL 블록이 될 수 있습니다. CASE 문은 END CASE;로 끝나지만 CASE 식은 END;로 끝납니다.

널 처리

널 값과 관련하여 다음 규칙을 기억해 두면 일반적으로 발생할 수 있는 실수를 피할 수 있습니다.

- 널을 사용하는 단순 비교는 항상 NULL을 반환합니다.
- 논리 연산자 NOT을 널에 적용하면 NULL이 발생합니다.
- 조건 제어문에서 조건이 NULL을 반환하면 연관된 명령문 시퀀스가 실행되지 않습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

널 처리

다음 예제를 살펴보십시오.

```
x := 5;
y := NULL;
...
IF x != y THEN -- yields NULL, not TRUE
    -- sequence_of_statements that are not executed
END IF;
```

x와 y가 같아 보이지 않으므로 명령문 시퀀스가 실행될 것으로 예상할 수 있습니다. 그러나 널은 불명확하므로 x와 y가 같은지 알 수 없습니다. 따라서 IF 조건이 NULL을 반환하면 명령문 시퀀스가 통과됩니다.

```
a := NULL;
b := NULL;
...
IF a = b THEN -- yields NULL, not TRUE
    -- sequence_of_statements that are not executed
END IF;
```

두번째 예제에서는 a와 b가 같아 보이므로 명령문 시퀀스가 실행될 것으로 예상할 수 있습니다. 그러나 여기서도 a와 b가 같은지 알 수 없으므로 IF 조건이 NULL을 반환하고 명령문 시퀀스가 통과됩니다.

논리 테이블

비교 연산자를 사용하여 단순 부울 조건을 작성합니다.

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

논리 테이블

비교 연산자로 숫자, 문자 또는 날짜 표현식을 조합하여 단순 부울 조건을 작성할 수 있습니다. AND, OR 및 NOT과 같은 논리 연산자로 단순 부울 조건을 조합하여 복합 부울 조건을 작성할 수 있습니다. 논리 연산자는 부울 변수 값을 검사하여 TRUE, FALSE 또는 NULL을 반환하는데 사용됩니다. 이 슬라이드의 논리 테이블의 설명은 다음과 같습니다.

- FALSE는 AND 조건에서 우선하고 TRUE는 OR 조건에서 우선합니다.
- AND는 피연산자가 둘 다 TRUE인 경우에만 TRUE를 반환합니다.
- OR은 피연산자가 둘 다 FALSE인 경우에만 FALSE를 반환합니다.
- NULL AND TRUE는 두번째 피연산자가 TRUE로 평가되는지 알 수 없으므로 항상 NULL로 평가됩니다.

주: 널 값은 불명확하기 때문에 NULL의 부정(NOT NULL)은 널 값을 반환합니다.

부울 조건

각 경우의 flag 값은?

```
flag := reorder_flag AND available_flag;
```

REORDER_FLAG	AVAILABLE_FLAG	FLAG
TRUE	TRUE	? (1)
TRUE	FALSE	? (2)
NULL	TRUE	? (3)
NULL	FALSE	? (4)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

부울 조건

AND 논리 테이블은 이 슬라이드의 부울 조건에 대한 확률을 평가하는 데 도움이 될 수 있습니다.

해답

1. TRUE
2. FALSE
3. NULL
4. FALSE

반복 제어: LOOP 문

- 루프는 명령문이나 명령문 시퀀스를 여러 번 반복합니다.
- 루프 유형은 다음 세 가지입니다.
 - 기본 루프
 - FOR 루프
 - WHILE 루프



ORACLE

Copyright © 2009, Oracle. All rights reserved.

반복 제어: LOOP 문

PL/SQL은 명령문이나 명령문 시퀀스를 여러 번 반복하는 루프를 구조화하기 위한 많은 기능을 제공합니다. 루프는 종료 조건에 도달할 때까지 명령문을 반복적으로 실행하는 데 주로 사용됩니다. 루프에는 종료 조건이 반드시 있어야 합니다. 그렇지 않으면 루프가 무한히 반복됩니다.

루프 생성자는 또 다른 제어 구조 유형입니다. PL/SQL에서 제공하는 루프 유형은 다음과 같습니다.

- 전반적인 조건 없이 반복적인 작업을 수행하는 기본 루프
- 개수를 기준으로 반복 작업을 수행하는 FOR 루프
- 조건을 기준으로 반복 작업을 수행하는 WHILE 루프

주: EXIT 문을 루프 종료에 사용할 수 있습니다. 기본 루프에는 EXIT가 있어야 합니다. 커서 FOR LOOP(FOR LOOP의 또 다른 유형)는 "명시적 커서 사용" 단원에서 설명합니다.

기본 루프

구문:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 루프

LOOP 문의 가장 간단한 형식은 기본(또는 무한) 루프이며, 키워드 LOOP와 END LOOP 사이에 명령문 시퀀스를 포함합니다. 실행 흐름이 END LOOP 문에 도달할 때마다 제어는 위쪽에 있는 해당 LOOP 문으로 돌아갑니다. 기본 루프를 사용하면 루프 시작과 함께 EXIT 조건이 충족되더라도 명령문을 적어도 한 번 실행할 수 있습니다. EXIT 문이 없으면 루프는 무한히 반복됩니다.

EXIT 문

EXIT 문을 사용하여 루프를 종료할 수 있습니다. 제어는 END LOOP 문 다음의 명령문으로 이동합니다. IF 문 내의 작업이나 루프 내의 독립형 명령문으로 EXIT를 실행할 수 있습니다. EXIT 문은 루프 안에 두어야 합니다. 독립형 명령문으로 EXIT를 실행하는 경우 조건부 루프 종료가 가능하도록 WHEN 절을 추가할 수 있습니다. EXIT 문에 도달하면 WHEN 절의 조건이 평가됩니다. 조건에서 TRUE를 반환하면 루프가 종료되고 제어가 루프 다음의 명령문으로 이동합니다. 기본 루프는 여러 개의 EXIT 문을 포함할 수 있지만 EXIT 포인트는 하나만 있는 것이 좋습니다.

기본 루프

예제

```
DECLARE
  countryid      locations.country_id%TYPE := 'CA';
  loc_id         locations.location_id%TYPE;
  counter        NUMBER(2) := 1;
  new_city       locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO loc_id FROM locations
  WHERE country_id = countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((loc_id + counter), new_city, countryid);
    counter := counter + 1;
    EXIT WHEN counter > 3;
  END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 루프(계속)

이 슬라이드에 나오는 기본 루프 예제는 국가 코드 CA 및 도시 Montreal에 대해 3개의 새 위치 ID를 삽입하는 것으로 정의되어 있습니다.

주: 기본 루프를 사용하면 루프 시작과 함께 조건이 충족되더라도 명령문을 한 번 이상 실행할 수 있습니다. 이러한 명령문이 실행될 때까지 조건이 검사되지 않도록 루프 안에 조건이 있는 경우에만 해당됩니다. 그러나 종료 조건이 다른 모든 실행문 앞, 즉 루프 위쪽에 있는 경우 해당 조건이 true이면 루프가 종료되고 명령문이 실행되지 않습니다.

WHILE 루프

구문:

```
WHILE condition LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

조건이 TRUE인 동안 명령문을 반복하려면 WHILE 루프를 사용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WHILE 루프

WHILE 루프를 사용하여 제어 조건이 더 이상 TRUE가 아닐 때까지 명령문 시퀀스를 반복할 수 있습니다. 이 조건은 반복이 시작될 때마다 평가됩니다. 조건이 FALSE 또는 NULL이면 루프가 종료됩니다. 루프 시작 시 조건이 FALSE 또는 NULL인 경우 더 이상 반복이 수행되지 않습니다.

이 구문에서 다음이 적용됩니다.

조건 부울 변수 또는 표현식(TRUE, FALSE 또는 NULL)입니다.

명령문 하나 이상의 PL/SQL 또는 SQL 문일 수 있습니다.

조건에 포함된 변수가 루프 본문에서 변경되지 않으면 조건은 TRUE인 상태로 유지되고 루프가 종료되지 않습니다.

주: 조건이 NULL을 반환하면 루프가 통과되고 제어는 다음 명령문으로 이동합니다.

WHILE 루프

예제

```
DECLARE
  countryid  locations.country_id%TYPE := 'CA';
  loc_id     locations.location_id%TYPE;
  new_city   locations.city%TYPE := 'Montreal';
  counter    NUMBER := 1;
BEGIN
  SELECT MAX(location_id) INTO loc_id FROM locations
  WHERE country_id = countryid;
  WHILE counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((loc_id + counter), new_city, countryid);
    counter := counter + 1;
  END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WHILE 루프(계속)

이 슬라이드의 예제에서는 CA 국가 코드 및 도시 Montreal에 대한 세 개의 새 위치 ID가 추가됩니다.

WHILE 루프를 통한 매회 반복 시 카운터(counter)가 증가합니다. 반복 횟수가 3회 이하이면 루프 안의 코드가 실행되고 행이 locations 테이블에 삽입됩니다. counter가 이 도시와 국가에 대한 새 위치 수를 초과하면 루프를 제어하는 조건이 FALSE로 평가되고 루프가 종료됩니다.

FOR 루프

- 반복 횟수에 대한 테스트를 단축하려면 FOR 루프를 사용합니다.
- 카운터는 암시적으로 선언되므로 선언하지 않아도 됩니다.
- 필요한 구문은 'lower_bound .. upper_bound' 입니다.

```
FOR counter IN [REVERSE]
  lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FOR 루프

FOR 루프는 기본 루프와 동일한 일반 구조를 가집니다. 또한 LOOP 키워드 앞에 PL/SQL에서 실행하는 반복 횟수를 설정하는 제어문이 있습니다. 이 구문에서 다음이 적용됩니다.

카운터 상한이나 하한에 도달할 때까지 루프 반복 시마다 값이 자동으로 1씩 증가하거나 감소하는(REVERSE 키워드를 사용하는 경우 감소) 암시적으로 선언되는 정수입니다.

REVERSE 매번 루프 반복 시마다 카운터를 상한에서 하한까지 감소시킵니다.

주: 여전히 하한이 먼저 참조됩니다.

lower_bound 카운터 값 범위에 대한 하한을 지정합니다.

upper_bound 카운터 값 범위에 대한 상한을 지정합니다.

카운터를 선언하지 마십시오. 암시적으로 정수로 선언됩니다.

FOR 루프(계속)

주: 상한과 하한 사이에서 카운터가 증가할 때마다 명령문 시퀀스가 실행됩니다. 루프 범위의 상한과 하한은 리터럴, 변수 또는 표현식이 될 수 있으나 정수로 평가되어야 합니다. 상한과 하한은 정수로 반올림됩니다. 즉, 11/3 및 8/5는 유효한 상한/하한입니다. 하한과 상한은 루프 범위에 포함됩니다. 루프 범위의 하한이 상한보다 큰 정수로 평가되면 명령문 시퀀스가 실행되지 않습니다. 예를 들어, 다음 명령문은 한번만 실행됩니다.

```
FOR i IN 3..3
LOOP
  statement1;
END LOOP;
```

FOR 루프

예제:

```
DECLARE
  countryid  locations.country_id%TYPE := 'CA';
  loc_id     locations.location_id%TYPE;
  new_city   locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO loc_id
    FROM locations
   WHERE country_id = countryid;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
      VALUES((loc_id + i), new_city, countryid );
  END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FOR 루프(계속)

이미 앞에서 기본 루프와 WHILE 루프를 사용하여 국가 코드 CA와 도시 Montreal에 대한 세 개의 새 위치를 삽입하는 방법에 대해 설명했습니다. 이 슬라이드에서는 FOR 루프를 사용하여 동일한 작업을 수행하는 방법을 보여줍니다.

FOR 루프

지침

- 루프 안에 있는 카운터만 참조하십시오. 루프 밖에서는 카운터가 정의되어 있지 않습니다.
- 카운터를 할당 대상으로 참조하지 마십시오.
- 루프 상한이나 하한은 NULL일 수 없습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FOR 루프(계속)

이 슬라이드는 FOR 루프 작성 시 따라야 하는 지침을 나열합니다.

주: LOOP 문의 하한과 상한은 숫자 리터럴이 아니어도 됩니다. 이러한 항목은 숫자 값으로 변환되는 표현식일 수 있습니다.

예제:

```
DECLARE
  lower  NUMBER := 1;
  upper  NUMBER := 100;
BEGIN
  FOR i IN lower..upper LOOP
    ...
  END LOOP;
END;
/
```


루프 지침

- 루프 안의 명령문이 적어도 한 번 실행되어야 하는 경우에는 기본 루프를 사용합니다.
- 매번 반복을 시작할 때마다 조건이 평가되어야 하는 경우에는 `WHILE` 루프를 사용합니다.
- 반복 횟수를 알 수 있는 경우에는 `FOR` 루프를 사용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

루프 지침

기본 루프를 사용하면 루프 시작과 함께 조건이 충족되더라도 명령문을 적어도 한 번 실행할 수 있습니다. `EXIT` 문이 없으면 루프는 무한히 반복됩니다.

`WHILE` 루프를 사용하여 제어 조건이 더 이상 `TRUE`가 아닐 때까지 명령문 시퀀스를 반복할 수 있습니다. 이 조건은 반복이 시작될 때마다 평가됩니다. 조건이 `FALSE`인 경우 루프가 종료됩니다. 루프 시작 시 조건이 `FALSE`인 경우 더 이상 반복이 수행되지 않습니다.

`FOR` 루프의 `LOOP` 키워드 앞에 `PL/SQL`에서 실행하는 반복 횟수를 설정하는 제어문이 있습니다. 반복 횟수가 미리 정의된 경우에는 `FOR` 루프를 사용합니다.

중첩 루프 및 레이블

- 여러 레벨로 루프를 중첩시킬 수 있습니다.
- 블록과 루프의 구분에 레이블을 사용합니다.
- 레이블을 참조하는 `EXIT` 문을 사용하여 외부 루프를 종료합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 루프 및 레이블

`FOR`, `WHILE` 및 기본 루프를 서로 중첩시킬 수 있습니다. 중첩 루프를 종료해도 예외 사항이 발생하지 않는 한 포함 루프는 종료되지 않습니다. 그러나 루프의 레이블을 지정하고 `EXIT` 문을 사용하여 외부 루프를 종료할 수 있습니다.

레이블 이름은 다른 식별자와 동일한 규칙을 따릅니다. 레이블은 명령문 앞에 같은 행이나 별도 행에 위치합니다. 내부 리터럴을 제외하고는 모든 `PL/SQL` 구문 분석 시 빈 칸은 중요하지 않습니다. `LOOP` 단어 앞에 레이블 구분자로 묶인(<<label>>) 레이블을 두어 기본 루프의 레이블을 지정합니다. `FOR` 및 `WHILE` 루프에서는 `FOR` 또는 `WHILE` 앞에 레이블을 배치합니다.

루프에 레이블이 지정되어 있으면 명확성을 위해 `END LOOP` 문 다음에 레이블 이름을 선택적으로 포함시킬 수 있습니다.

중첩 루프 및 레이블

```
...  
BEGIN  
  <<Outer_loop>>  
  LOOP  
    counter := counter+1;  
    EXIT WHEN counter>10;  
    <<Inner_loop>>  
    LOOP  
      ...  
      EXIT Outer_loop WHEN total_done = 'YES';  
      -- Leave both loops  
      EXIT WHEN inner_done = 'YES';  
      -- Leave inner loop only  
      ...  
    END LOOP Inner_loop;  
    ...  
  END LOOP Outer_loop;  
END;  
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 루프 및 레이블(계속)

이 슬라이드의 예제에는 두 개의 루프, 즉 <<Outer_Loop>> 레이블로 식별되는 외부 루프와 <<Inner_Loop>> 레이블로 식별되는 내부 루프가 있습니다. 식별자는 레이블 구분자(<<label>>)로 묶여 LOOP 단어 앞에 놓입니다. 내부 루프는 외부 루프 안에 중첩됩니다. 레이블 이름은 명확성을 위해 END LOOP 문 다음에 포함됩니다.

요약

이 단원에서는 다음 제어 구조를 사용하여 명령문의 논리적 흐름을 변경하는 방법에 대해 설명했습니다.

- 조건부 (IF 문)
- CASE 식 및 CASE 문
- 루프:
 - 기본 루프
 - FOR 루프
 - WHILE 루프
- EXIT 문

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

업무 논리 구현을 위한 제어 구조를 제공하는 경우에 한하여 언어를 프로그래밍 언어라 할 수 있습니다. 이러한 제어 구조는 프로그램의 흐름을 제어하는 데도 사용됩니다. PL/SQL은 프로그래밍 생성자와 SQL을 통합하는 프로그래밍 언어입니다.

조건부 제어 생성자는 조건의 유효성을 검사하고 해당 작업을 수행합니다. 명령문을 조건부로 실행하려면 IF 생성자를 사용합니다.

반복 제어 생성자는 지정된 조건이 TRUE인 경우 반복적으로 명령문 시퀀스를 실행합니다. 반복적인 작업을 수행하려면 다양한 루프 생성자를 사용합니다.

연습 5: 개요

이 연습에서는 다음 내용을 다룹니다.

- IF 문을 사용하여 조건부 작업 수행
- 루프 구조를 사용하여 반복적인 단계 수행

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 5: 개요

이 연습에서는 루프와 조건부 제어 구조를 포함하는 PL/SQL 블록을 생성합니다. 연습 문제에서는 다양한 IF 문과 LOOP 생성자 작성에 대한 이해도를 테스트합니다.

연습 5

1. lab_05_01.sql 파일에서 messages 테이블을 생성하는 명령을 실행합니다. messages 테이블에 숫자를 삽입하는 PL/SQL 블록을 작성합니다.
 - a. 1에서 10까지의 숫자를 삽입합니다(6, 8 제외).
 - b. 블록 종료 전에 커밋합니다.
 - c. SELECT 문을 실행하여 PL/SQL 블록이 실행되었는지 확인합니다. 출력은 다음과 같아야 합니다.

	RESULTS
1	1
2	2
3	3
4	4
5	5
6	7
7	9
8	10

2. lab_05_02.sql 스크립트를 실행합니다. 이 스크립트는 employees 테이블의 복제본인 emp 테이블을 생성합니다. 이 스크립트는 데이터 유형은 VARCHAR2이고 크기는 50인 새 열 stars를 추가하도록 emp 테이블을 변경합니다. 사원의 급여에 대해 \$1000 단위마다 stars 열에 별표를 삽입하는 PL/SQL 블록을 생성합니다. 스크립트를 lab_05_02_soln.sql로 저장합니다.
 - a. DEFINE 명령을 사용하여 empno라는 변수를 정의하고 176으로 초기화합니다.
 - b. 블록의 선언 섹션을 시작하고 치환 변수를 통해 empno 값을 PL/SQL 블록으로 전달합니다. emp.stars 유형의 asterisk 변수를 선언하고 NULL로 초기화합니다. emp.salary 유형의 sal 변수를 생성합니다.
 - c. 실행 섹션에서 급여 금액에 대해 \$1000 단위마다 별표(*)를 추가하는 논리를 작성합니다. 예를 들어, 사원의 급여가 \$8000이면 별표 문자열에는 8개의 별표가 있어야 합니다. 급여가 \$12500이면 별표 문자열에는 13개의 별표가 있어야 합니다.
 - d. 해당 사원의 stars 열을 별표 문자열로 갱신합니다. 블록 종료 전에 커밋합니다.

연습 5(계속)

- e. emp 테이블의 행을 출력하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다.
- f. 스크립트를 실행하고 lab_05_02_soln.sql로 저장합니다. 출력은 다음과 같습니다.

	EMPLOYEE_ID	SALARY	STARS
1	176	8600	*****

- 3. 연습 4의 질문 4에서 작성한 lab_04_04_soln.sql 스크립트를 엽니다.
 - a. "INCLUDE SIMPLE IF STATEMENT HERE" 라는 주석을 찾은 다음 간단한 IF 문을 포함시켜 emp_id와 emp_authorization의 값이 동일한지 확인합니다.
 - b. 스크립트를 lab_05_03_soln.sql로 저장합니다.

6

조합 데이터 유형 작업

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **유저 정의 PL/SQL 레코드 생성**
- **%ROWTYPE 속성을 사용하여 레코드 생성**
- **INDEX BY 테이블 생성**
- **INDEX BY 레코드 테이블 생성**
- **레코드, 테이블 및 레코드 테이블 간의 차이점 설명**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 목적

조합 데이터 유형에 대해서는 이미 설명했습니다. 이 단원에서는 조합 데이터 유형 및 사용법에 대해 자세히 설명합니다.

조합 데이터 유형

- 스칼라 유형과는 달리 다중 값을 보유할 수 있습니다.
- 두 가지 유형이 있습니다.
 - PL/SQL 레코드
 - PL/SQL 컬렉션
 - INDEX BY 테이블 또는 연관 배열
 - 중첩 테이블
 - VARRAY

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조합 데이터 유형

이미 앞에서 스칼라 데이터 유형의 변수는 단일 값만 보유할 수 있지만 조합 데이터 유형의 변수는 스칼라 데이터 유형이나 조합 데이터 유형의 다중 값을 보유할 수 있다는 것을 배웠습니다. 조합 데이터 유형은 다음 두 가지입니다.

- **PL/SQL 레코드:** 관련은 있지만 유사하지 않은 데이터를 논리적 단위로 처리하는 데 사용되는 레코드입니다. PL/SQL 레코드는 다양한 유형의 변수를 가질 수 있습니다. 예를 들어, 사원 세부 정보를 포함할 레코드를 정의할 수 있습니다. 여기에는 사원 번호를 NUMBER로 저장하고 이름과 성을 VARCHAR2로 저장하는 등의 작업이 포함됩니다. 사원 세부 정보를 저장할 레코드를 생성하면 집합적 논리 단위가 생성됩니다. 따라서 데이터 액세스와 조작이 쉬워집니다.
- **PL/SQL 컬렉션:** 컬렉션은 데이터를 단일 단위로 처리하는 데 사용됩니다. 컬렉션의 유형은 다음 세 가지입니다.
 - INDEX BY 테이블 또는 연관 배열
 - 중첩 테이블
 - VARRAY

조합 데이터 유형을 사용해야 하는 이유

모든 관련 데이터를 단일 단위로 보유하게 됩니다. 데이터를 쉽게 액세스 및 수정할 수 있습니다. 데이터가 조합 유형인 경우 데이터를 보다 쉽게 관리, 연관 및 전송할 수 있습니다. 쉽게 설명하자면 랩톱의 각 구성 요소를 별개의 가방에 넣는 대신 모든 구성 요소를 단일 가방에 넣는 것과 같습니다.

조합 데이터 유형

- 서로 다른 데이터 유형의 값을 저장하려는 경우 한 번에 하나씩만 저장하려면 PL/SQL 레코드를 사용합니다.
- 동일한 데이터 유형의 값을 저장하려는 경우 PL/SQL 컬렉션을 사용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조합 데이터 유형(계속)

PL/SQL 레코드와 PL/SQL 컬렉션 모두 조합 유형인데 둘 중 어느 것을 사용할지 어떻게 알 수 있을까요?

논리적으로 관련된 서로 다른 데이터 유형의 값을 저장하려는 경우 PL/SQL 레코드를 사용합니다. 사원 세부 정보를 보유할 레코드를 생성하는 경우 저장된 모든 값은 특정 사원에 대한 정보를 제공하므로 서로 관련되어 있습니다.

동일한 데이터 유형의 값을 저장하려는 경우 PL/SQL 컬렉션을 사용합니다. 이 데이터 유형은 레코드와 같은 조합 유형이 될 수도 있습니다. 모든 사원의 이름을 보유할 컬렉션을 정의할 수 있습니다. 컬렉션에 n 개의 이름을 저장했을 수 있는데, 여기에서 이름 1은 이름 2와 관련이 없습니다. 이들 이름 사이의 관계는 사원 이름이라는 것뿐입니다. 이러한 컬렉션은 C, C++ 및 Java와 같은 프로그래밍 언어의 배열과 유사합니다.

PL/SQL 레코드

- 스칼라, RECORD 또는 INDEX BY 테이블 데이터 유형의 구성 요소(필드라고 함)를 하나 이상 포함해야 합니다.
- C 및 C++을 포함하여 대부분 3GL 언어의 구조와 유사합니다.
- 유저 정의 데이터 유형이며 테이블 행의 부분 집합일 수 있습니다.
- 필드 모음을 논리적 단위로 처리합니다.
- 테이블에서 데이터 행을 패치(fetch)하여 처리하는데 편리합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드

레코드는 필드에 저장된 관련 데이터 항목의 그룹으로, 각 필드에는 고유한 이름과 데이터 유형이 있습니다.

- 정의된 각 레코드는 필요한 만큼 필드를 가질 수 있습니다.
- 레코드는 초기값을 할당 받을 수 있으며 NOT NULL로 정의될 수 있습니다.
- 초기값이 없는 필드는 NULL로 초기화됩니다.
- DEFAULT 키워드도 필드 정의에 사용할 수 있습니다.
- 블록, 서브 프로그램 또는 패키지의 선언 부분에서 RECORD 유형을 정의하고 유저 정의 레코드를 선언할 수 있습니다.
- 중첩 레코드를 선언 및 참조할 수 있습니다. 이 경우 한 레코드는 다른 레코드의 구성 요소가 됩니다.

PL/SQL 레코드 생성

구문:

1

```
TYPE type_name IS RECORD  
    (field_declaration [, field_declaration]...);
```

2

```
identifier    type_name;
```

field_declaration:

```
field_name {field_type | variable%TYPE  
            | table.column%TYPE | table%ROWTYPE}  
[[NOT NULL] {:= | DEFAULT} expr]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 생성

PL/SQL 레코드는 유저 정의 조합 유형입니다. PL/SQL 레코드를 사용하려면 다음 작업을 수행해야 합니다.

1. PL/SQL 블록의 선언 부분에서 레코드를 정의합니다. 레코드를 정의하는 구문은 슬라이드에 표시되어 있습니다.
2. 이 레코드 유형의 내부 구성 요소를 선언하고 선택적으로 초기화합니다.

이 구문에서 다음이 적용됩니다.

type_name RECORD 유형의 이름입니다. 이 식별자는 레코드 선언에 사용됩니다.

field_name 레코드 내의 필드 이름입니다.

field_type 필드의 데이터 유형입니다. REF CURSOR를 제외한 모든 PL/SQL 데이터 유형을 나타냅니다. %TYPE 및 %ROWTYPE 속성을 사용할 수

있습니다.

expr *field_type* 또는 초기값입니다.

NOT NULL 제약 조건은 이러한 필드에 널리 할당되지 않도록 합니다. NOT NULL 필드를 초기화해야 합니다.

REF CURSOR에 대해서는 부록 C "REF 커서"에서 다룹니다.

PL/SQL 레코드 생성

새로운 사원의 이름, 직무, 급여를 저장하도록 변수를 선언합니다.

예제:

```
...
TYPE emp_record_type IS RECORD
  (last_name  VARCHAR2(25),
   job_id     VARCHAR2(10),
   salary     NUMBER(8,2));
emp_record   emp_record_type;
...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 생성(계속)

레코드 정의에 사용되는 필드 선언은 변수 선언과 유사합니다. 각 필드에는 고유한 이름과 특정 데이터 유형이 있습니다. 스칼라 변수의 경우는 미리 정의된 데이터 유형이 있지만, PL/SQL 레코드의 경우는 미리 정의된 데이터 유형이 없습니다. 그러므로 먼저 레코드 유형을 생성한 다음 해당 유형을 사용하여 식별자를 선언해야 합니다.

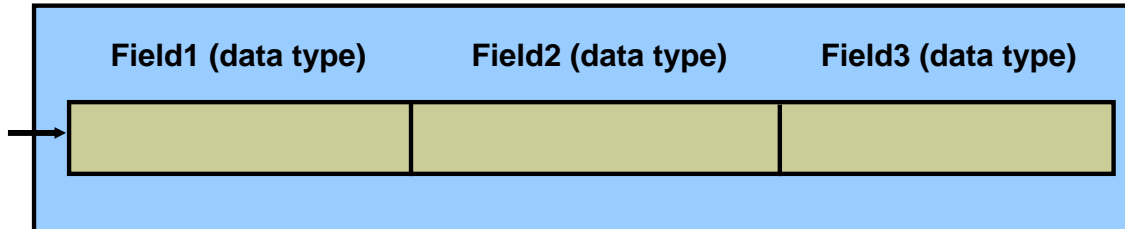
슬라이드의 예제에는 last_name, job_id 및 salary에 대한 값을 포함하도록 레코드 유형(emp_record_type)이 정의되어 있습니다. 다음 단계에서는 emp_record_type 유형의 레코드(emp_record)를 선언합니다.

다음 예제는 %TYPE 속성을 사용하여 필드 데이터 유형을 지정할 수 있음을 보여줍니다.

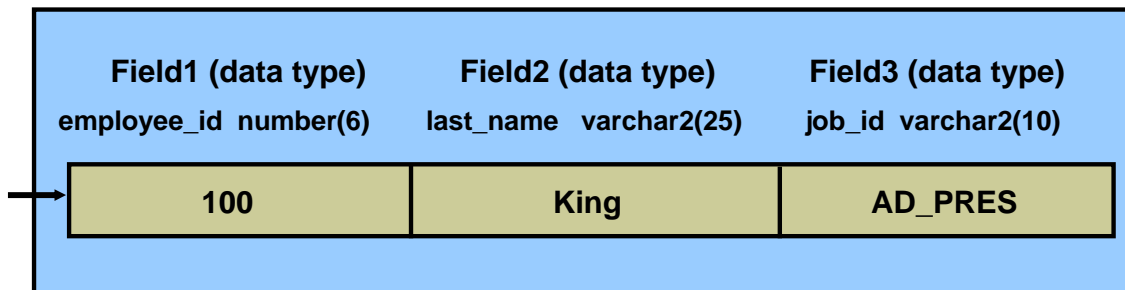
```
DECLARE
  TYPE emp_record_type IS RECORD
  (employee_id  NUMBER(6) NOT NULL := 100,
   last_name    employees.last_name%TYPE,
   job_id       employees.job_id%TYPE);
  emp_record    emp_record_type;
...
```

주: 필드 선언에 NOT NULL 제약 조건을 추가할 수 있으므로 해당 필드에 널이 할당되는 것을 방지할 수 있습니다. NOT NULL로 선언된 필드는 초기화되어야 한다는 점을 유념하십시오.

PL/SQL 레코드 구조



예제:



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 구조

레코드의 필드는 레코드 이름으로 액세스합니다. 개별 필드를 참조하거나 초기화하려면 다음과 같이 점 표기법을 사용합니다.

```
record_name.field_name
```

예를 들어, 다음과 같이 emp_record 레코드의 job_id 필드를 참조합니다.

```
emp_record.job_id
```

그런 다음 아래와 같이 레코드 필드에 값을 할당할 수 있습니다.

```
emp_record.job_id := 'ST_CLERK';
```

블록 또는 서브 프로그램에서 유저 정의 레코드는 블록 또는 서브 프로그램을 입력할 때 인스턴스화됩니다. 블록 또는 서브 프로그램을 종료하면 해당 레코드도 더 이상 존재하지 않게 됩니다.

%ROWTYPE 속성

- 데이터베이스 테이블 또는 뷰의 열 모음에 따라 변수를 선언합니다.
- %ROWTYPE 앞에는 데이터베이스 테이블 또는 뷰 이름이 접두어로 붙습니다.
- 레코드의 필드는 테이블 또는 뷰의 열에서 이름 및 데이터 유형을 가져옵니다.

구문:

```
DECLARE
    identifier reference%ROWTYPE;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%ROWTYPE 속성

앞에서 %TYPE이 열 유형의 변수를 선언하는 데 사용된다는 것을 배웠습니다. 변수는 테이블 열과 동일한 데이터 유형 및 크기를 가집니다. %TYPE의 이점은 열이 변경되는 경우에도 변수를 변경할 필요가 없다는 것입니다. 또한 변수가 계산에 사용될 경우 변수의 정밀도에 대해서 신경 쓸 필요가 없습니다.

%ROWTYPE 속성은 테이블 또는 뷰의 전체 행을 포함할 수 있는 레코드를 선언하는 데 사용됩니다. 레코드의 필드는 테이블 또는 뷰의 열에서 이름 및 데이터 유형을 가져옵니다. 레코드는 커서나 커서 변수에서 패치(fetch)한 전체 데이터 행을 저장할 수도 있습니다.

슬라이드는 레코드 선언을 위한 구문을 보여줍니다. 이 구문에서 다음이 적용됩니다.

<i>identifier</i>	레코드 전체에 대해 선택한 이름입니다.
<i>reference</i>	레코드의 기반이 될 테이블, 뷰, 커서 또는 커서 변수의 이름입니다. 이 참조가 유효하려면 테이블 또는 뷰가 존재해야 합니다.

다음 예제에서는 %ROWTYPE을 데이터 유형 지정자로 사용하여 레코드가 선언됩니다.

```
DECLARE
    emp_record employees%ROWTYPE;
    ...
```


%ROWTYPE 속성(계속)

emp_record 레코드는 각각 employees 테이블의 열을 나타내는 다음 필드로 이루어진 구조를 가집니다.

주: 이는 코드가 아니며 단순히 조합 변수의 구조를 나타냅니다.

```
(employee_id      NUMBER(6),
 first_name       VARCHAR2(20),
 last_name        VARCHAR2(20),
 email            VARCHAR2(20),
 phone_number     VARCHAR2(20),
 hire_date        DATE,
 salary           NUMBER(8,2),
 commission_pct   NUMBER(2,2),
 manager_id       NUMBER(6),
 department_id    NUMBER(4))
```

개별 필드를 참조하려면 다음과 같이 점 표기법을 사용합니다.

```
record_name.field_name
```

예를 들어, 다음과 같이 emp_record 레코드의 commission_pct 필드를 참조합니다.

```
emp_record.commission_pct
```

You can then assign a value to the record field:

```
emp_record.commission_pct := .35;
```

레코드에 값 할당

SELECT 또는 FETCH 문을 사용하여 레코드에 일반적인 값 리스트를 할당할 수 있습니다.

열 이름이 레코드의 필드와 동일한 순서로 나타나는지 확인합니다. 두 레코드의 해당 데이터 유형이 동일한 경우 한 레코드를 다른 레코드에 할당할 수도 있습니다. 유저 정의 레코드와 %ROWTYPE 레코드는 동일한 데이터 유형을 가질 수 없습니다.

%ROWTYPE 사용 시 이점

- 기본 데이터베이스 열의 개수와 데이터 유형을 알 필요가 없으며 실제로 런타임에 변경될 수 있습니다.
- %ROWTYPE 속성은 `SELECT *` 문을 사용하여 행을 검색할 때 유용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%ROWTYPE 사용 시 이점

%ROWTYPE 속성 사용 시 이점이 슬라이드에 나열되어 있습니다. 기본 데이터베이스 테이블의 구조에 대해 정확히 모르는 경우 %ROWTYPE 속성을 사용합니다.

%ROWTYPE 사용 시 주된 이점은 유지 관리가 단순화된다는 것입니다. %ROWTYPE을 사용하면 기본 테이블이 변경되는 경우 이 속성을 사용하여 선언된 변수의 데이터 유형이 동적으로 변경됩니다. DDL 문이 테이블에서 열을 변경할 경우 PL/SQL 프로그램 단위가 무효화됩니다. 프로그램이 재컴파일되면 자동으로 새로운 테이블 형식을 반영합니다.

%ROWTYPE 속성은 테이블에서 전체 행을 검색하려는 경우 특히 유용합니다. 이 속성이 없으면 select 문에 의해 검색된 각 열에 대해 일일이 변수를 선언해야 합니다.

%ROWTYPE 속성

```
...  
DEFINE employee_number = 124  
DECLARE  
    emp_rec    employees%ROWTYPE;  
BEGIN  
    SELECT * INTO emp_rec FROM employees  
    WHERE employee_id = &employee_number;  
    INSERT INTO retired_emps(empno, ename, job, mgr,  
        hiredate, leavedate, sal, comm, deptno)  
    VALUES (emp_rec.employee_id, emp_rec.last_name,  
        emp_rec.job_id, emp_rec.manager_id,  
        emp_rec.hire_date, SYSDATE, emp_rec.salary,  
        emp_rec.commission_pct, emp_rec.department_id);  
END;  
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%ROWTYPE 속성

%ROWTYPE 속성의 예제가 슬라이드에 표시되어 있습니다. 사원이 퇴직하는 경우 해당 사원에 대한 정보가 퇴직한 사원들에 대한 정보를 포함하는 테이블에 추가됩니다. 유저가 사원 번호를 입력한다고 합시다. 유저에 의해 지정된 사원의 레코드가 employees 테이블에서 검색되고 %ROWTYPE 속성을 사용하여 선언된 emp_rec 변수에 저장됩니다. retired_emps 테이블에 삽입되는 레코드는 다음과 같습니다.

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	02-FEB-09	5800	(null)	50

%ROWTYPE을 사용하여 레코드 삽입

```

...
DEFINE employee_number = 124
DECLARE
    emp_rec    retired_emps%ROWTYPE;
BEGIN
    SELECT employee_id, last_name, job_id, manager_id,
           hire_date, hire_date, salary, commission_pct,
           department_id INTO emp_rec FROM employees
    WHERE  employee_id = &employee_number;
    INSERT INTO retired_emps VALUES emp_rec;
END;
/
SELECT * FROM retired_emps;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%ROWTYPE을 사용하여 레코드 삽입

앞의 슬라이드에서 나왔던 insert 문과 이 슬라이드의 insert 문을 비교해 보십시오. emp_rec 레코드는 retired_emps 유형입니다. 레코드의 필드 수는 INTO 절의 필드 이름 수와 같아야 합니다. 테이블에 값을 삽입하는 데 이 레코드를 사용할 수 있습니다. 그러면 코드가 읽기 쉬워집니다.

다음은 retired_emps를 생성하는 create 문입니다.

```

CREATE TABLE retired_emps
(EMPNO      NUMBER(4), ENAME   VARCHAR2(10),
 JOB        VARCHAR2(9), MGR    NUMBER(4),
 HIREDATE   DATE, LEAVEDATE  DATE,
 SAL        NUMBER(7,2), COMM  NUMBER(7,2),
 DEPTNO     NUMBER(2))

```

슬라이드의 SELECT 문을 살펴보십시오. hire_date를 두 번 선택하고 hire_date 값을 retired_emps의 leavedate 필드에 삽입합니다. 채용 날짜에 퇴직하는 사원은 없습니다. 삽입되는 레코드는 다음과 같습니다.

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	16-NOV-99	5800	(null)	50

레코드를 사용하여 테이블의 행 갱신

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE employee_number = 124
DECLARE
    emp_rec retired_emps%ROWTYPE;
BEGIN
    SELECT * INTO emp_rec FROM retired_emps;
    emp_rec.leavedate:=SYSDATE;
    UPDATE retired_emps SET ROW = emp_rec WHERE
        empno=&employee_number;
END;
/
SELECT * FROM retired_emps;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

레코드를 사용하여 테이블의 행 갱신

앞에서 레코드를 사용하여 행을 삽입하는 방법에 대해 배웠습니다. 이 슬라이드는 레코드를 사용하여 행을 갱신하는 방법을 보여줍니다. ROW 키워드가 전체 행을 나타내는 데 사용됩니다. 슬라이드에 표시된 코드는 사원의 leavedate를 갱신합니다. 레코드가 갱신됩니다.

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	02-FEB-09	5800	(null)	50

INDEX BY 테이블 또는 연관 배열

- 다음과 같은 두 개의 열이 있는 PL/SQL 구조입니다.
 - 정수 또는 문자열 데이터 유형의 Primary Key
 - 스칼라 또는 레코드 데이터 유형의 열
- 크기의 제약이 없습니다. 그러나 크기는 키 데이터 유형이 포함할 수 있는 값에 따라 다릅니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 또는 연관 배열

INDEX BY 테이블은 조합 유형(컬렉션)이자 유저 정의 테이블입니다. INDEX BY 테이블은 Primary Key 값을 인덱스로 사용하여 데이터를 저장할 수 있습니다. 여기서 키 값은 순차적이지 않습니다. INDEX BY 테이블은 키-값 쌍 집합입니다. (키와 값 쌍이 정확하게 열에 저장되어 있지 않아도 두 열에 저장된 데이터를 예측할 수 있습니다.)

INDEX BY 테이블에는 다음 두 개의 열만 있습니다.

- Primary Key 역할을 하는 정수 또는 문자열 유형의 열. 키는 BINARY_INTEGER 또는 PLS_INTEGER 유형의 숫자일 수 있습니다. BINARY_INTEGER 및 PLS_INTEGER 키는 NUMBER보다 작은 저장 영역을 필요로 하며, 기계적 산술을 사용하여 수학 정수를 명료하게 나타내고 산술 연산을 구현하는 데 사용됩니다. 이러한 데이터 유형에 대한 산술 연산은 NUMBER 산술보다 빠릅니다. 키는 VARCHAR2 유형 또는 해당 서브타입 중 하나일 수도 있습니다. 이 과정의 예제에서는 키 열에 대한 데이터 유형으로 PLS_INTEGER를 사용합니다.
- 값을 포함할 스칼라 또는 레코드 데이터 유형의 열. 열이 스칼라 유형이면 단일 값만 포함할 수 있습니다. 열이 레코드 유형이면 다중 값을 포함할 수 있습니다.

INDEX BY 테이블은 크기의 제약이 없습니다. 그러나 PLS_INTEGER 열의 키는 PLS_INTEGER가 포함할 수 있는 최대값으로 제한됩니다. 키는 양수 및 음수 모두가 될 수 있습니다. INDEX BY 테이블의 키는 순차적이지 않습니다.

INDEX BY 테이블 생성

구문:

```
TYPE type_name IS TABLE OF
    {column_type | variable%TYPE
    | table.column%TYPE} [NOT NULL]
    | table%ROWTYPE
    [INDEX BY PLS_INTEGER | BINARY_INTEGER
    | VARCHAR2(<size>)];
identifier    type_name;
```

사원의 성을 저장할 INDEX BY 테이블을 선언합니다.

```
...
TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
...
ename_table ename_table_type;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 생성

INDEX BY 테이블을 생성하려면 다음 두 단계를 수행해야 합니다.

1. TABLE 데이터 유형을 선언합니다.
2. 해당 데이터 유형의 변수를 선언합니다.

이 구문에서 다음이 적용됩니다.

<i>type_name</i>	TABLE 유형의 이름입니다. PL/SQL 테이블 식별자의 이후 선언에 유형 지정자로 사용됩니다.
<i>column_type</i>	VARCHAR2, DATE, NUMBER 또는 %TYPE과 같은 스칼라 또는 조합 데이터 유형입니다. %TYPE 속성을 사용하여 열 데이터 유형을 제공할 수 있습니다.
<i>identifier</i>	전체 PL/SQL 테이블을 나타내는 식별자의 이름입니다.

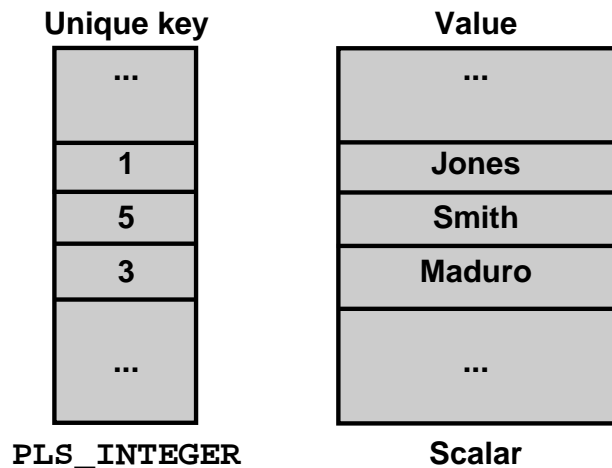
INDEX BY 테이블 생성(계속)

NOT NULL 제약 조건은 해당 유형의 PL/SQL 테이블에 널리 할당되지 않도록 합니다.
INDEX BY 테이블을 초기화하지 마십시오.

INDEX BY 테이블은 BINARY_INTEGER, BOOLEAN, LONG, LONG RAW, NATURAL, NATURALN, PLS_INTEGER, POSITIVE, POSITIVEN, SIGNTYPE 및 STRING 요소 유형을 가질 수 있습니다.

INDEX BY 테이블은 생성 시 자동으로 채워지지 않습니다. PL/SQL 프로그램에서 INDEX BY 테이블을 프로그래밍 방식으로 채운 다음 사용해야 합니다.

INDEX BY 테이블 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 구조

데이터베이스 테이블의 크기와 마찬가지로 INDEX BY 테이블의 크기에도 제약이 없습니다. 즉, INDEX BY 테이블의 행 수를 동적으로 늘릴 수 있으므로 새 행이 추가될 때마다 INDEX BY 테이블이 커집니다.

INDEX BY 테이블에는 하나의 열과 해당 열에 대한 고유 식별자가 있을 수 있으며 둘 다 이름이 지정되지 않을 수 있습니다. 테이블의 열은 스칼라 또는 레코드 데이터 유형에 속할 수 있으나 Primary Key는 반드시 PLS_INTEGER, BINARY_INTEGER 또는 VARCHAR2 유형에 속해야 합니다. INDEX BY 테이블은 자체 선언에서 초기화할 수 없습니다. INDEX BY 테이블은 선언 시 채워지지 않으며 키나 값을 포함하지 않습니다. INDEX BY 테이블을 채우려면 명시적 실행문이 필요합니다.

INDEX BY 테이블 생성

```
DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY PLS_INTEGER;
  ename_table          ename_table_type;
  hiredate_table        hiredate_table_type;
BEGIN
  ename_table(1)       := 'CAMERON';
  hiredate_table(8)    := SYSDATE + 7;
  IF ename_table.EXISTS(1) THEN
    INSERT INTO ...
    ...
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 생성

슬라이드의 예제에서는 두 개의 INDEX BY 테이블을 생성합니다.

INDEX BY 테이블의 키를 사용하여 테이블의 요소에 액세스합니다.

구문:

INDEX_BY_table_name(index)

여기서 인덱스는 PLS_INTEGER 유형에 속합니다.

다음 예제에서는 ename_table이라는 INDEX BY 테이블의 세번째 행을 참조하는 방법을 보여줍니다.

ename_table(3)

PLS_INTEGER의 크기 범위가 -2147483647 ~ 2147483647이므로 Primary Key 값은 음수가 될 수 있습니다. 인덱스는 1로 시작할 필요가 없습니다.

주: exists(i) 메소드는 i 인덱스를 가진 행이 반환되면 TRUE를 반환합니다. 존재하지 않는 테이블 요소 참조 시 발생하는 오류를 방지하려면 exists 메소드를 사용하십시오.

INDEX BY 테이블 메소드 사용

다음 메소드를 사용하면 INDEX BY 테이블 사용이 용이해집니다.

- EXISTS
- COUNT
- FIRST and LAST
- PRIOR
- NEXT
- DELETE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 메소드 사용

INDEX BY 테이블 메소드는 PL/SQL 테이블에서 작동하는 내장 프로시저 또는 함수이며 점 표기법을 통해 호출됩니다.

구문: `table_name.method_name[(parameters)]`

메소드	설명
EXISTS(<i>n</i>)	PL/SQL 테이블에 <i>n</i> 번째 요소가 존재하면 TRUE를 반환합니다.
COUNT	PL/SQL 테이블이 현재 포함하고 있는 요소의 수를 반환합니다.
FIRST LAST	<ul style="list-style-type: none">• PL/SQL 테이블에서 각각 첫번째와 마지막(최소 및 최대) 인덱스 번호를 반환합니다.• PL/SQL 테이블이 비어 있으면 NULL을 반환합니다.
PRIOR(<i>n</i>)	PL/SQL 테이블에서 인덱스 <i>n</i> 앞에 오는 인덱스 번호를 반환합니다.
NEXT(<i>n</i>)	PL/SQL 테이블에서 인덱스 <i>n</i> 뒤에 오는 인덱스 번호를 반환합니다.
DELETE	<ul style="list-style-type: none">• DELETE는 PL/SQL 테이블에서 모든 요소를 제거합니다.• DELETE(<i>n</i>)은 PL/SQL 테이블에서 <i>n</i>번째 요소를 제거합니다.• DELETE(<i>m</i>, <i>n</i>)은 PL/SQL 테이블에서 <i>m</i> ... <i>n</i> 범위의 모든 요소를 제거합니다.

INDEX BY 레코드 테이블

테이블의 전체 행을 포함할 INDEX BY 테이블 변수를 정의합니다.

예제:

```
DECLARE
  TYPE dept_table_type IS TABLE OF
    departments%ROWTYPE
    INDEX BY PLS_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 레코드 테이블

스칼라 데이터 유형의 테이블로 선언된 INDEX BY 테이블은 데이터베이스 테이블의 오직 한 열에 대한 세부 정보만 저장할 수 있습니다. query에 의해 검색되는 모든 열을 저장해야 하는 경우가 종종 있습니다. INDEX BY 레코드 테이블이 이 문제에 대한 해결책을 제공합니다. 하나의 테이블 정의만 있으면 데이터베이스 테이블의 모든 필드에 대한 정보를 저장할 수 있기 때문에 레코드 테이블은 INDEX BY 테이블의 기능을 크게 향상시킵니다.

레코드 테이블 참조

슬라이드의 예제에서 테이블의 각 요소는 레코드이므로 dept_table 레코드의 필드를 참조할 수 있습니다.

구문:

```
table(index).field
```

예제:

```
dept_table(15).location_id := 1700;
```

location_id represents a field in dept_table.

레코드 테이블 참조(계속)

%ROWTYPE 속성을 사용하여 데이터베이스 테이블의 행을 나타내는 레코드를 선언할 수 있습니다. %ROWTYPE 속성과 조합 데이터 유형의 PL/SQL 레코드의 차이점은 다음과 같습니다.

- PL/SQL 레코드 유형은 사용자가 정의할 수 있으나 %ROWTYPE은 레코드를 암시적으로 정의합니다.
- PL/SQL 레코드를 사용하면 선언하는 동안 필드와 데이터 유형을 지정할 수 있습니다. %ROWTYPE을 사용할 경우에는 필드를 지정할 수 없습니다. %ROWTYPE 속성은 해당 테이블의 정의에 기반한 모든 필드를 가진 테이블 행을 나타냅니다.
- 유저 정의 레코드는 정적입니다. %ROWTYPE 레코드는 테이블 구조가 데이터베이스에서 변경되기 때문에 동적입니다.

INDEX BY 레코드 테이블: 예제

```
SET SERVEROUTPUT ON
DECLARE
    TYPE emp_table_type IS TABLE OF
        employees%ROWTYPE INDEX BY PLS_INTEGER;
    my_emp_table    emp_table_type;
    max_count       NUMBER(3) := 104;
BEGIN
    FOR i IN 100..max_count
    LOOP
        SELECT * INTO my_emp_table(i) FROM employees
        WHERE employee_id = i;
    END LOOP;
    FOR i IN my_emp_table.FIRST..my_emp_table.LAST
    LOOP
        DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
    END LOOP;
END;
/
```

ORACLE

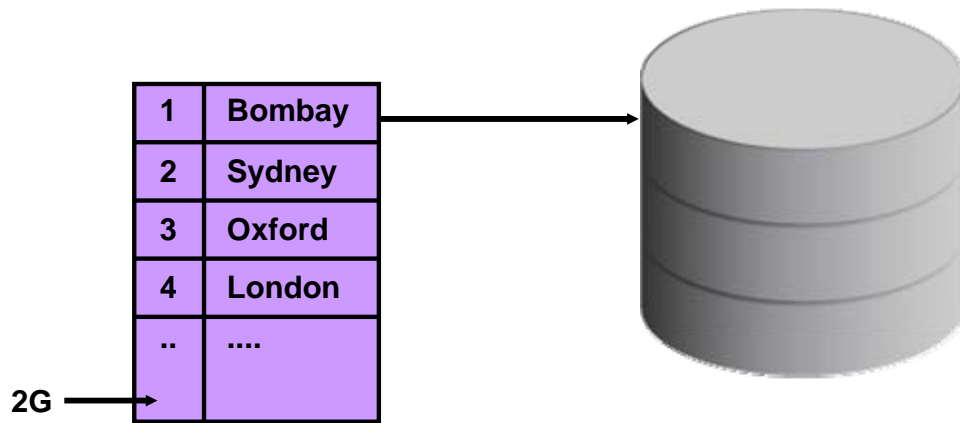
Copyright © 2009, Oracle. All rights reserved.

INDEX BY 레코드 테이블: 예제

슬라이드의 예제에서는 사원 ID가 100과 104 사이인 사원의 세부 정보를 임시 저장할 emp_table_type이라는 INDEX BY 레코드 테이블을 선언하며 루프를 통해 EMPLOYEES 테이블의 사원 정보를 검색하여 INDEX BY 테이블에 저장합니다. 또 다른 루프를 사용하여 INDEX BY 테이블에서 성을 출력합니다. 예제에서 첫번째 메소드와 마지막 메소드의 사용을 확인하십시오.

```
anonymous block completed
King
Kochhar
De Haan
Hunold
Ernst
```

중첩 테이블



ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 테이블

중첩 테이블의 기능은 INDEX BY 테이블과 비슷하지만 중첩 테이블 구현에 차이가 있습니다. 중첩 테이블은 스키마 레벨 테이블에 적합한 데이터 유형이지만 INDEX BY 테이블은 그렇지 않습니다. 중첩 테이블의 키 유형은 PLS_INTEGER가 아닙니다. INDEX BY 테이블과는 달리 키가 음수 값일 수 없습니다. 첫번째 열을 키라고 부르고 있지만 중첩 테이블에는 키가 없습니다. 대신에 키 열로 간주되는 순차적인 번호가 있는 열이 있습니다. 중첩 테이블의 어디서든 요소를 삭제할 수 있으며 이 경우 순차적이 아닌 키를 갖는 희소 테이블이 됩니다. 중첩 테이블의 행은 특정 순서로 되어 있지 않습니다. 중첩 테이블에서 값을 검색할 때 행에는 1부터 시작하는 연속 첨자가 주어집니다. 중첩 테이블은 INDEX BY 테이블과는 달리 데이터베이스에 저장할 수 있습니다.

구문:

```
TYPE type_name IS TABLE OF
    {column_type | variable%TYPE
    | table.column%TYPE} [NOT NULL]
    | table%ROWTYPE
```

Oracle Database 10g에서는 중첩 테이블이 동일한지 비교할 수 있습니다. 중첩 테이블에 요소가 존재하는지 확인할 수 있고 중첩 테이블이 다른 중첩 테이블의 부분 집합인지도 확인할 수 있습니다.

중첩 테이블(계속)

예제:

```
TYPE location_type IS TABLE OF locations.city%TYPE;  
offices location_type;
```

INDEX BY 테이블을 초기화하지 않은 경우 비어 있습니다. 중첩 테이블을 초기화하지 않은 경우 자동으로 NULL로 초기화됩니다. 생성자를 사용하여 offices 중첩 테이블을 초기화할 수 있습니다.

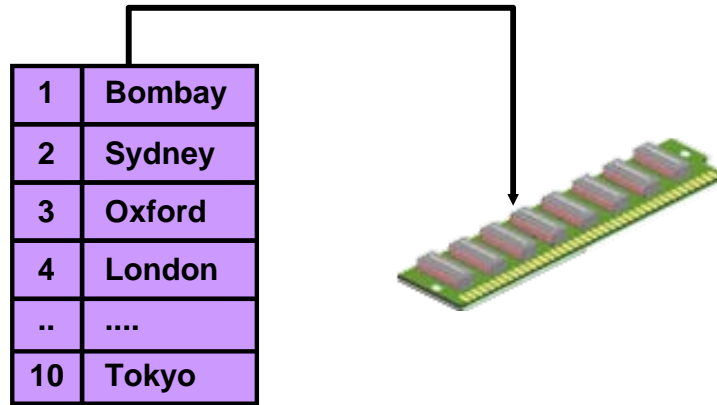
```
offices := location_type('Bombay', 'Tokyo','Singapore',  
    'Oxford');
```

완전한 예제:

```
SET SERVEROUTPUT ON  
DECLARE  
    TYPE location_type IS TABLE OF locations.city%TYPE;  
    offices location_type;  
    table_count NUMBER;  
BEGIN  
    offices := location_type('Bombay', 'Tokyo','Singapore',  
        'Oxford');  
    table_count := offices.count();  
    FOR i in 1..table_count LOOP  
        DBMS_OUTPUT.PUT_LINE(offices(i));  
    END LOOP;  
END;  
/
```

```
anonymous block completed  
Bombay  
Tokyo  
Singapore  
Oxford
```


VARRAY



ORACLE

Copyright © 2009, Oracle. All rights reserved.

VARRAY

가변 크기 배열 (VARRAY)은 VARRAY 크기에 제약이 있다는 점을 제외하고 PL/SQL 테이블과 비슷합니다. VARRAY는 스키마 레벨 테이블에 적합합니다. VARRAY 유형의 항목을 VARRAYs라고 합니다. VARRAYs에는 고정된 상한값이 있습니다. 선언 시 상한값을 지정해야 합니다. 이는 C 언어의 배열과 비슷합니다. VARRAY의 최대 크기는 중첩 테이블에서와 같이 2GB입니다. 중첩 테이블과 VARRAY가 뚜렷이 구별되는 점은 물리적 저장 모드입니다. VARRAY의 요소는 데이터베이스가 아닌 메모리에 인접하여 저장됩니다. SQL을 사용하여 데이터베이스에 VARRAY 유형을 생성할 수 있습니다.

예제:

```
TYPE location_type IS VARRAY(3) OF locations.city%TYPE;  
offices location_type;
```

VARRAY의 크기는 3으로 제한됩니다. 생성자를 사용하여 VARRAY를 초기화할 수 있습니다. 네 개 이상의 요소를 갖도록 VARRAY를 초기화하려고 시도하면 "Subscript outside of limit"이라는 오류 메시지가 표시됩니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 조합 데이터 유형의 PL/SQL 변수 정의 및 참조
 - PL/SQL 레코드
 - INDEX BY 테이블
 - INDEX BY 레코드 테이블
- %ROWTYPE 속성을 사용하여 PL/SQL 레코드 정의

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

PL/SQL 레코드는 테이블의 행을 나타내는 개별 필드의 모음입니다. 레코드를 사용하여 데이터를 단일 구조로 그룹화한 다음 해당 구조를 하나의 엔티티 또는 논리적 단위로 조작할 수 있습니다. 그러면 코딩 작업이 줄어들고 코드 유지 관리 및 이해가 쉬워집니다.

PL/SQL 레코드와 마찬가지로 테이블도 또 다른 조합 데이터 유형입니다. INDEX BY 테이블은 TABLE 유형의 객체이며 약간의 차이를 제외하고 데이터베이스 테이블과 비슷합니다. INDEX BY 테이블은 Primary Key를 사용하여 행에 대한 배열 방식의 액세스를 제공합니다. INDEX BY 테이블은 크기의 제약이 없습니다. INDEX BY 테이블에는 키-값 쌍이 저장됩니다. 키 열은 PLS_INTEGER, BINARY_INTEGER 또는 VARCHAR2 유형이어야 하고 값을 포함하는 열은 모든 데이터 유형이 될 수 있습니다.

중첩 테이블의 키 유형은 PLS_INTEGER가 아닙니다. 키는 INDEX BY 테이블의 경우와는 달리 음수 값을 가질 수 없습니다. 또한 키가 순차적이어야 합니다.

가변 크기 배열(VARRAYs)은 VARRAY 크기에 제약이 있다는 점을 제외하고 PL/SQL 테이블과 비슷합니다.

연습 6: 개요

이 연습에서는 다음 내용을 다룹니다.

- INDEX BY 테이블 선언
- INDEX BY 테이블을 사용하여 데이터 처리
- PL/SQL 레코드 선언
- PL/SQL 레코드를 사용하여 데이터 처리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 6: 개요

이 연습에서는 INDEX BY 테이블과 PL/SQL 레코드를 정의, 생성 및 사용합니다.

연습 6

1. 제공된 국가에 대한 정보를 출력하는 PL/SQL 블록을 작성합니다.
 - a. `countries` 테이블의 구조에 맞게 PL/SQL 레코드를 선언합니다.
 - b. `DEFINE` 명령을 사용하여 `countryid` 변수를 정의합니다. `countryid`에 `CA`를 할당합니다. 이 값을 치환 변수를 사용하여 PL/SQL 블록에 전달합니다.
 - c. 선언 섹션에서 `%ROWTYPE` 속성을 사용하여 `countries` 유형의 `country_record` 변수를 선언합니다.
 - d. 실행 섹션에서 `countryid`를 사용하여 `countries` 테이블의 모든 정보를 가져옵니다. 선택한 국가의 정보를 표시합니다. 다음은 출력 예입니다.
- ```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```
- e. ID가 DE, UK, US인 국가에 대해서도 PL/SQL 블록을 실행하여 테스트해 보십시오.
  2. `INDEX BY` 테이블과 통합하여 `departments` 테이블에서 일부 부서 이름을 검색하여 각 부서 이름을 화면에 출력하는 PL/SQL 블록을 생성합니다. 스크립트를 `lab_06_02_soln.sql`로 저장합니다.
    - a. `departments.department_name` 유형의 `INDEX BY` 테이블 `dept_table_type`을 선언합니다. `dept_table_type` 유형의 `my_dept_table` 변수를 선언하여 부서 이름을 임시로 저장합니다.
    - b. `NUMBER` 유형의 두 변수 `loop_count` 및 `deptno`를 선언합니다. `loop_count`에 10을 할당하고 `deptno`에 0을 할당합니다.
    - c. 루프를 사용하여 10개의 부서 이름을 검색하고 `INDEX BY` 테이블에 이름을 저장합니다. `department_id`를 10에서 시작합니다. 루프가 반복할 때마다 `deptno`를 10씩 증가시킵니다. 다음 테이블은 `department_name`을 검색하고 `INDEX BY` 테이블에 저장할 `department_id`를 보여줍니다.

| DEPARTMENT_ID | DEPARTMENT_NAME  |
|---------------|------------------|
| 10            | Administration   |
| 20            | Marketing        |
| 30            | Purchasing       |
| 40            | Human Resources  |
| 50            | Shipping         |
| 60            | IT               |
| 70            | Public Relations |
| 80            | Sales            |
| 90            | Executive        |
| 100           | Finance          |

## 연습 6(계속)

- d. 다른 루프를 사용하여 INDEX BY 테이블에서 부서 이름을 검색한 다음 정보를 출력합니다.
- e. 스크립트를 실행하고 lab\_06\_02\_soln.sql로 저장합니다. 출력 결과는 다음과 같습니다.

```
anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance
```

3. departments 테이블에서 각 부서 정보를 모두 검색하여 출력하도록 2번 문제에서 생성한 블록을 수정합니다. INDEX BY 레코드 테이블을 사용합니다.
  - a. lab\_06\_02\_soln.sql 스크립트를 엽니다.
  - b. INDEX BY 테이블은 departments.department\_name 유형으로 선언된 상태입니다. 부서의 번호, 이름, manager\_id 및 위치를 임시로 저장하도록 INDEX BY 테이블의 선언을 수정합니다. %ROWTYPE 속성을 사용합니다.
  - c. departments 테이블의 현재 모든 부서 정보를 검색하도록 select 문을 수정하고 INDEX BY 테이블에 저장합니다.
  - d. 다른 루프를 사용하여 INDEX BY 테이블에서 부서 정보를 검색한 다음 정보를 출력합니다. 다음은 출력 예입니다.

```
anonymous block completed
Department Number: 10 Department Name: Administration Manager Id: 200 Location Id: 1700
Department Number: 20 Department Name: Marketing Manager Id: 201 Location Id: 1800
Department Number: 30 Department Name: Purchasing Manager Id: 114 Location Id: 1700
Department Number: 40 Department Name: Human Resources Manager Id: 203 Location Id: 2400
Department Number: 50 Department Name: Shipping Manager Id: 121 Location Id: 1500
Department Number: 60 Department Name: IT Manager Id: 103 Location Id: 1400
Department Number: 70 Department Name: Public Relations Manager Id: 204 Location Id: 2700
Department Number: 80 Department Name: Sales Manager Id: 145 Location Id: 2500
Department Number: 90 Department Name: Executive Manager Id: 100 Location Id: 1700
Department Number: 100 Department Name: Finance Manager Id: 108 Location Id: 1700
```

4. lab\_05\_03\_soln.sql 스크립트를 엽니다.
  - a. "DECLARE AN INDEX BY TABLE OF TYPE VARCHAR2(50). CALL IT  
ename\_table\_type"이라는 주석을 찾아 선언을 포함시킵니다.
  - b. "DECLARE A VARIABLE ename\_table OF TYPE ename\_table\_type"이라는 주석을  
찾아 선언을 포함시킵니다.
  - c. 스크립트를 lab\_06\_04\_soln.sql로 저장합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 암시적 커서 및 명시적 커서 구분
- 명시적 커서를 사용하는 이유 설명
- 명시적 커서 선언 및 제어
- 간단한 루프 및 커서 FOR 루프를 사용하여 데이터 패치(fetch)
- 파라미터가 포함된 커서 선언 및 사용
- FOR UPDATE 절을 사용하여 행 잠금
- WHERE CURRENT OF 절을 사용하여 현재 행 참조

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 단원 목적

이미 앞에서 SQL SELECT 또는 DML 문을 실행할 때 PL/SQL에서 자동으로 생성하는 암시적 커서에 대해 배웠습니다. 이 단원에서는 명시적 커서에 대해 설명합니다. 암시적 커서와 명시적 커서를 구분하는 방법에 대해 설명하고 간단한 커서 및 파라미터가 포함된 커서를 선언하고 제어하는 방법에 대해서도 설명합니다.

# 커서

Oracle 서버에서 실행되는 모든 SQL 문에는 연관된 개별 커서가 있습니다.

- **암시적 커서:** 모든 DML 및 PL/SQL SELECT 문에 대해 PL/SQL이 선언하고 관리합니다.
- **명시적 커서:** 프로그래머가 선언하고 관리합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 커서

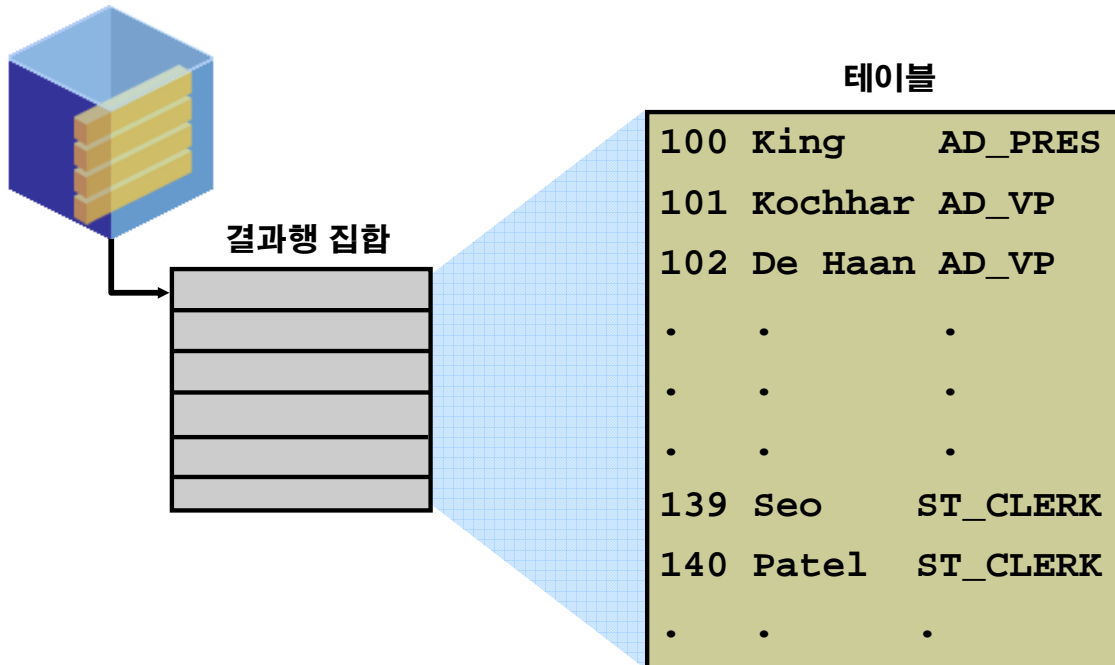
Oracle 서버는 전용 SQL 영역이라고 하는 작업 영역을 사용하여 SQL 문을 실행하고 처리 정보를 저장합니다. 명시적 커서를 사용하여 전용 SQL 영역을 명명하고 해당 영역에 저장된 정보에 액세스할 수 있습니다.

| 커서 유형 | 설명                                                                             |
|-------|--------------------------------------------------------------------------------|
| 암시적   | 암시적 커서는 모든 DML 및 PL/SQL SELECT 문에 대해 PL/SQL에서 암시적으로 선언합니다.                     |
| 명시적   | 명시적 커서는 둘 이상의 행을 반환하는 query에 대해 프로그래머가 선언하고 관리하며 블록의 실행 작업에서 특정 명령문을 통해 조작됩니다. |

Oracle 서버는 암시적으로 커서를 열어 명시적으로 선언된 커서와 연관되지 않은 각 SQL 문을 처리합니다. PL/SQL을 사용하여 가장 최근에 생성된 암시적 커서를 SQL 커서로 참조할 수 있습니다.



## 명시적 커서 작업



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 명시적 커서 작업

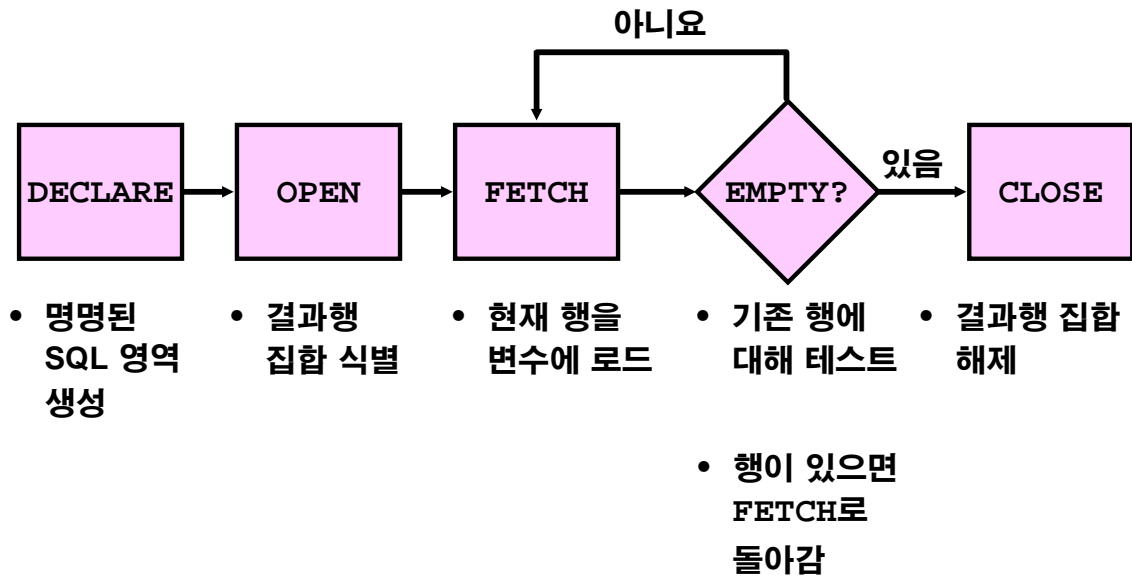
다중 행을 반환하는 SELECT 문이 있을 때 PL/SQL에서 명시적 커서를 선언합니다. SELECT 문에서 반환된 각 행을 처리할 수 있습니다.

다중 행 쿼리에서 반환된 행 집합을 결과행 집합이라고 합니다. 결과행 집합의 크기는 검색 조건을 만족하는 행의 수입니다. 위 슬라이드의 도표는 명시적 커서가 결과행 집합의 현재 행을 "가리키는" 방식을 보여줍니다. 이렇게 하면 프로그램에서 한 번에 하나씩 행을 처리할 수 있습니다.

명시적 커서의 기능은 다음과 같습니다.

- query에서 반환된 첫번째 행 이후에 행 단위 처리를 수행할 수 있습니다.
- 현재 처리 중인 행을 추적합니다.
- 프로그래머가 PL/SQL 블록에서 명시적 커서를 수동으로 제어할 수 있습니다.

## 명시적 커서 제어



ORACLE

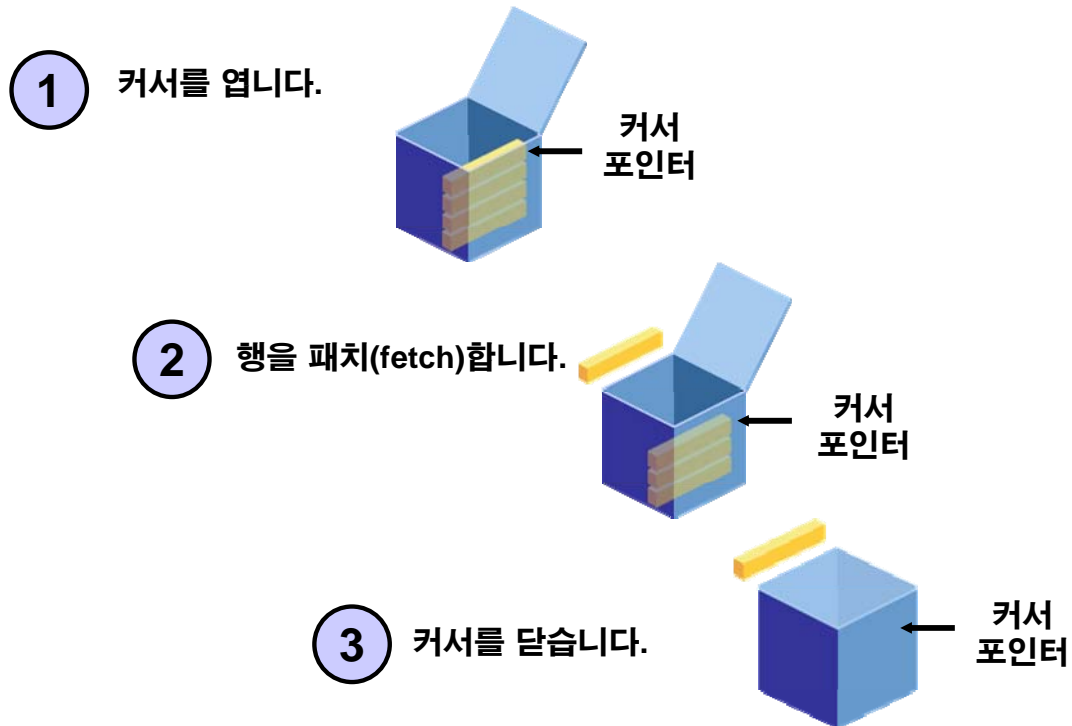
Copyright © 2009, Oracle. All rights reserved.

### 명시적 커서 제어

이제 커서에 대한 개념은 이해했으므로 커서의 사용 단계를 살펴봅니다.

1. 이름을 지정하고 연관될 query 구조를 정의하여 PL/SQL 블록의 선언 섹션에서 커서를 선언합니다.
2. 커서를 엽니다. OPEN 문은 query를 실행하고 참조된 모든 변수를 바인드합니다. query에 의해 식별된 행을 결과행 집합이라고 하며 현재 패치(fetch)할 수 있습니다.
3. 커서에서 데이터를 패치합니다. 위 슬라이드에 표시된 흐름도에서 각 패치 후에 기존 행이 있는지 커서를 테스트합니다. 더 이상 처리할 행이 없으면 커서를 닫아야 합니다.
4. 커서를 닫습니다. CLOSE 문은 결과행 집합을 해제합니다. 이제 커서를 다시 열어 새로운 활성 집합을 설정할 수 있습니다.

## 명시적 커서 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 명시적 커서 제어(계속)

PL/SQL 프로그램은 커서를 열고 query에 의해 반환된 행을 처리한 다음 커서를 닫습니다. 커서는 결과행 집합에서의 현재 위치를 표시합니다.

1. OPEN 문은 커서와 연관된 query를 실행하고 결과행 집합을 식별하며 커서를 첫번째 행에 배치합니다.
2. FETCH 문은 현재 행을 검색하여 행이 더 이상 없거나 지정된 조건에 부합할 때까지 커서를 다음 행으로 이동합니다.
3. CLOSE 문은 커서를 해제합니다.

# 커서 선언

## 구문:

```
CURSOR cursor_name IS
 select_statement;
```

## 예제:

```
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id =30;
```

```
DECLARE
 locid NUMBER:= 1700;
 CURSOR dept_cursor IS
 SELECT * FROM departments
 WHERE location_id = locid;
 ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 커서 선언

위 슬라이드는 커서를 선언하는 구문을 보여줍니다. 이 구문에서 다음이 적용됩니다.

|                         |                     |
|-------------------------|---------------------|
| <i>cursor_name</i>      | PL/SQL 식별자          |
| <i>select_statement</i> | INTO 절이 없는 SELECT 문 |

커서의 결과행 집합은 커서 선언의 SELECT 문에 의해 결정됩니다. 반드시 PL/SQL에 SELECT 문의 INTO 절이 있어야 합니다. 그러나 커서 선언의 SELECT 문은 INTO 절을 가질 수 없습니다. 이것은 커서가 선언 섹션에 한정되어 있고 커서로 행을 읽어 들이지 않기 때문입니다.

## 참고

- INTO 절은 나중에 FETCH 문에 나타나므로 커서 선언에 포함시키지 마십시오.
- 특정 시퀀스로 행을 처리해야 할 경우 query에 ORDER BY 절을 사용하십시오.
- 커서는 조인, subquery 등을 포함하는 유효한 ANSI SELECT 문일 수 있습니다.

## 커서 선언(계속)

`emp_cursor` 커서는 `department_id`가 30인 부서에서 근무하는 사원의 `employee_id` 및 `last_name` 열을 검색하도록 선언됩니다.

`dept_cursor` 커서는 `location_id`가 1700인 부서의 모든 세부 사항을 검색하도록 선언됩니다. 커서 선언 중에 변수가 사용됩니다. 이러한 변수는 커서를 선언 중일 때 표시되어야 하는 바인드 변수로 간주됩니다. 이 변수는 커서를 열 때 한 번만 검사됩니다. 이미 앞에서 명시적 커서가 **PL/SQL**에서 다중 행을 검색하고 처리할 때 사용된다는 것을 배웠습니다. 그러나 이 예제는 **SELECT** 문이 하나의 행만 반환하는 경우에도 명시적 커서를 사용할 수 있음을 보여줍니다.

## 커서 열기

```
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id =30;
 ...
BEGIN
 OPEN emp_cursor;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 열기

OPEN 문은 커서와 연관된 query를 실행하고 결과행 집합을 식별하며 커서 포인터를 첫번째 행에 배치합니다. OPEN 문은 PL/SQL 블록의 실행 섹션에 포함됩니다.

OPEN은 다음과 같은 작업을 수행하는 실행문입니다.

1. 컨텍스트 영역에 메모리를 동적으로 할당합니다.
2. SELECT 문의 구문을 분석합니다.
3. 입력 변수를 바인드합니다. 즉, 입력 변수의 메모리 주소를 확인하여 입력 변수 값을 설정합니다.
4. 결과행 집합(검색 조건을 만족하는 행 집합)을 식별합니다. 결과행 집합에 있는 행은 OPEN 문이 실행될 때 변수로 읽어 들여지지 않습니다. 대신 FETCH 문이 커서의 행을 변수로 읽어 들입니다.
5. 포인터를 결과행 집합의 첫번째 행에 배치합니다.

**주:** 커서가 열려 있을 때 query가 행을 반환하지 않으면 PL/SQL에서 예외가 발생하지 않습니다. 그러나 SQL%ROWCOUNT 커서 속성을 사용하여 패치(fetch) 이후의 암시적 커서 상태를 테스트할 수 있습니다. 명시적 커서에는 <cursor\_name>%ROWCOUNT를 사용하십시오.

## 커서에서 데이터 패치(fetch)

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id = 30;
 empno employees.employee_id%TYPE;
 lname employees.last_name%TYPE;
BEGIN
 OPEN emp_cursor;
 FETCH emp_cursor INTO empno, lname;
 DBMS_OUTPUT.PUT_LINE(empno || ' ' || lname);
 ...
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서에서 데이터 패치(fetch)

FETCH 문은 한 번에 하나씩 커서에서 행을 검색합니다. 각 패치 후에 커서가 결과행 집합의 다음 행으로 이동합니다. %NOTFOUND 속성을 사용하여 전체 결과행 집합을 검색했는지 확인할 수 있습니다.

위 슬라이드에 표시된 예제를 살펴보십시오. empno와 lname의 두 변수는 커서에서 패치된 값을 보유하도록 선언됩니다. FETCH 문을 검사하십시오.

PL/SQL 블록의 출력 결과는 다음과 같습니다.

```
anonymous block completed
114 Raphaely
```

커서의 값이 변수로 성공적으로 패치되었습니다. 그러나 부서 30에는 여섯 명의 사원이 있는데 한 행만 패치되었습니다. 모든 행을 패치하려면 루프를 사용해야 합니다. 다음 슬라이드에서 루프를 사용하여 모든 행을 패치하는 방법을 볼 수 있습니다.

FETCH 문은 다음과 같은 작업을 수행합니다.

1. 현재 행의 데이터를 출력 PL/SQL 변수로 읽어 들입니다.
2. 결과행 집합의 다음 행으로 포인터를 이동합니다.

### 커서에서 데이터 패치(fetch)(계속)

- `FETCH` 문의 `INTO` 절에 `SELECT` 문의 열과 동일한 수의 변수를 포함시키고 데이터 유형이 호환되는지 확인합니다.
- 위치에 따라 열에 각 변수를 대응시킵니다.
- 또는 커서에 대한 레코드를 정의하고 `FETCH INTO` 절에서 해당 레코드를 참조합니다.
- 커서에 행이 있는지 테스트합니다. 패치 후 획득한 값이 없으면 결과행 집합에 처리할 행이 남아 있지 않은 것이며 이것은 오류로 기록되지 않습니다.



## 커서에서 데이터 패치(fetch)

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id =30;
 empno employees.employee_id%TYPE;
 lname employees.last_name%TYPE;
BEGIN
 OPEN emp_cursor;
 LOOP
 FETCH emp_cursor INTO empno, lname;
 EXIT WHEN emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(empno || ' ' || lname);
 END LOOP;
 ...
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서에서 데이터 패치(fetch)(계속)

간단한 LOOP가 모든 행을 패치하는 데 사용됩니다. 또한 커서 속성 %NOTFOUND가 종료 조건을 테스트하는 데 사용됩니다. PL/SQL 블록의 출력 결과는 다음과 같습니다.

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

## 커서 닫기

```
...
 LOOP
 FETCH emp_cursor INTO empno, lname;
 EXIT WHEN emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(empno || ' ' || lname);
 END LOOP;
 CLOSE emp_cursor;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 닫기

CLOSE 문은 커서를 비활성화하고 컨텍스트 영역을 해제하고 결과행 집합의 정의를 해제합니다. FETCH 문 처리를 완료한 후 커서를 닫으십시오. 필요한 경우 커서를 다시 열 수 있습니다. 커서가 닫힌 경우에만 커서를 다시 열 수 있습니다. 커서가 닫힌 후 커서에서 데이터 패치(fetch)를 시도할 경우 INVALID\_CURSOR 예외가 발생합니다.

**주:** 커서를 닫지 않고 PL/SQL 블록을 종료할 수 있다 하더라도 자원을 해제하려면 명시적으로 선언한 커서를 항상 닫아야 합니다. 세션당 열 수 있는 최대 커서 수에는 데이터베이스 파라미터 파일의 OPEN\_CURSORS 파라미터에 의해 결정되는 제한이 있습니다.

OPEN\_CURSORS = 50(기본값).

# 커서 및 레코드

PL/SQL 레코드로 값을 패치하여 결과행 집합의 행을 처리합니다.

```
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id =30;
 emp_record emp_cursor%ROWTYPE;
BEGIN
 OPEN emp_cursor;
 LOOP
 FETCH emp_cursor INTO emp_record;
 ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 커서 및 레코드

이미 앞에서 테이블의 열 구조를 갖는 레코드를 정의할 수 있음을 확인했습니다. 또한 명시적 커서에서 선택한 열 리스트를 기반으로 레코드를 정의할 수 있습니다. 이 방법을 사용하면 간단하게 레코드로 패치(fetch)할 수 있으므로 결과행 집합의 행을 처리하는 경우 편리합니다. 따라서 행 값이 레코드의 해당 필드로 직접 로드됩니다.

## 커서 FOR 루프

### 구문:

```
FOR record_name IN cursor_name LOOP
 statement1;
 statement2;
 . . .
END LOOP;
```

- 커서 FOR 루프를 사용하면 명시적 커서를 간단히 처리할 수 있습니다.
- 열기, 패치(fetch), 종료 및 닫기 작업이 암시적으로 일어납니다.
- 레코드는 암시적으로 선언됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 FOR 루프

이미 앞에서 간단한 루프를 사용하여 커서에서 데이터를 패치(fetch)하는 방법을 배웠습니다. 이제 명시적 커서에서 행을 처리하는 커서 FOR 루프를 사용하는 방법을 배웁니다. 이것은 매우 간단한 방법으로, 커서가 열리고 루프가 반복될 때마다 한 번씩 행을 패치하고 마지막 행이 처리되면 루프가 종료되고 커서가 자동으로 닫힙니다. 루프 자체는 마지막 행이 패치되어 반복 작업이 끝날 때 자동으로 종료됩니다.

이 구문에서 다음이 적용됩니다.

*record\_name*    암시적으로 선언된 레코드 이름  
*cursor\_name*    이전에 선언된 커서에 대한 PL/SQL 식별자

### 지침

- 루프는 암시적으로 선언되므로 루프를 제어하는 레코드를 선언하지 마십시오.
- 필요한 경우 루프 중에 커서 속성을 테스트합니다.
- 필요한 경우 FOR 문에서 커서에 대한 파라미터를 괄호로 묶어 커서 이름 뒤에 붙입니다.

## 커서 FOR 루프

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR emp_cursor IS
 SELECT employee_id, last_name FROM employees
 WHERE department_id =30;
BEGIN
 FOR emp_record IN emp_cursor
 LOOP
 DBMS_OUTPUT.PUT_LINE(emp_record.employee_id
 || ' ' || emp_record.last_name);
 END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 FOR 루프(계속)

커서에서 데이터를 패치(fetch)하는 간단한 루프의 사용법을 보여줄 때 사용한 예제를 재작성하여 커서 FOR 루프를 사용하도록 했습니다.

emp\_record는 암시적으로 선언된 레코드입니다. 위 슬라이드에 표시된 것과 같이 이러한 암시적 레코드를 사용하여 패치 데이터에 액세스할 수 있습니다. INTO 절을 사용하여 패치된 데이터를 보유하도록 선언된 변수가 없습니다. 이 코드에는 커서를 열고 닫는 OPEN 및 CLOSE 문이 없습니다.

## 명시적 커서 속성

커서에 대한 상태 정보를 획득합니다.

| 속성        | 유형      | 설명                                                  |
|-----------|---------|-----------------------------------------------------|
| %ISOPEN   | Boolean | 커서가 열려 있으면 TRUE로 평가됩니다.                             |
| %NOTFOUND | Boolean | 가장 최근 패치(fetch)가 행을 반환하지 않으면 TRUE로 평가됩니다.           |
| %FOUND    | Boolean | 가장 최근 패치가 행을 반환하면 TRUE로 평가됩니다. %NOTFOUND 속성을 보완합니다. |
| %ROWCOUNT | Boolean | 지금까지 반환된 총 행 수로 평가됩니다.                              |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 명시적 커서 속성

암시적 커서와 마찬가지로, 커서에 대한 상태 정보를 획득하기 위한 네 개의 속성이 있습니다. 이러한 속성을 커서 변수 이름에 추가하면 커서 조작문 실행에 대한 유용한 정보가 반환됩니다.

주: SQL 문에서는 커서 속성을 직접 참조할 수 없습니다.

## %ISOPEN Attribute

- 커서가 열려 있을 때만 행을 패치합니다.
- 패치를 수행하기 전에 %ISOPEN 커서 속성을 사용하여 커서가 열려 있는지 테스트합니다.

### 예제:

```
IF NOT emp_cursor%ISOPEN THEN
 OPEN emp_cursor;
END IF;
LOOP
 FETCH emp_cursor...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### %ISOPEN 속성

- 커서가 열려 있을 때만 행을 패치(fetch)할 수 있습니다. %ISOPEN 커서 속성을 사용하여 커서가 열려 있는지 확인합니다.
- 루프에서 행을 패치합니다. 커서 속성을 사용하여 루프 종료 시점을 판별할 수 있습니다.
- %ROWCOUNT 커서 속성을 사용하여 다음을 수행할 수 있습니다.
  - 정확한 수의 행 처리
  - 루프에서 행을 패치하고 루프 종료 시점 확인

주: %ISOPEN은 커서가 열려 있으면 TRUE, 닫혀 있으면 FALSE로 커서 상태를 반환합니다.

## %ROWCOUNT 및 %NOTFOUND: 예제

```
SET SERVEROUTPUT ON
DECLARE
 empno employees.employee_id%TYPE;
 ename employees.last_name%TYPE;
 CURSOR emp_cursor IS SELECT employee_id,
 last_name FROM employees;
BEGIN
 OPEN emp_cursor;
 LOOP
 FETCH emp_cursor INTO empno, ename;
 EXIT WHEN emp_cursor%ROWCOUNT > 10 OR
 emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(TO_CHAR(empno)
 || ' ' || ename);
 END LOOP;
 CLOSE emp_cursor;
END ;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### %ROWCOUNT 및 %NOTFOUND: 예제

위 슬라이드의 예제는 처음 열 명의 사원을 한 명씩 읽어 들입니다. 이 예제는 루프의 종료 조건에 %ROWCOUNT 및 %NOTFOUND 속성을 사용하는 방법을 보여줍니다.



## subquery를 사용하는 커서 FOR 루프

커서를 선언할 필요가 없습니다.

예제:

```
SET SERVEROUTPUT ON
BEGIN
 FOR emp_record IN (SELECT employee_id, last_name
 FROM employees WHERE department_id =30)
 LOOP
 DBMS_OUTPUT.PUT_LINE(emp_record.employee_id || '
 ' || emp_record.last_name);
 END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### subquery를 사용하는 커서 FOR 루프

이 PL/SQL 블록에는 선언 섹션이 없습니다. subquery를 사용하는 커서 FOR 루프와 일반 커서 FOR 루프의 차이점은 커서 선언에 있습니다. subquery를 사용하여 커서 FOR 루프를 작성할 경우에는 선언 섹션에서 커서를 선언할 필요가 없습니다. 루프 자체에서 결과행 집합을 결정하는 SELECT 문을 제공해야 합니다.

커서 FOR 루프를 보여줄 때 사용한 예제를 재작성하여 subquery를 사용하는 커서 FOR 루프를 나타냈습니다.

**주:** 커서 FOR 루프에 subquery를 사용할 경우 커서에 명시적 이름을 부여할 수 없으므로 명시적 커서 속성을 참조할 수 없습니다.

## 파라미터가 포함된 커서

### 구문:

```
CURSOR cursor_name
 [(parameter_name datatype, ...)]
IS
 select_statement;
```

- 커서가 열리고 query가 실행되면 커서에 파라미터 값이 전달됩니다.
- 매번 다른 결과행 집합으로 여러 번 명시적 커서를 엽니다.

```
OPEN cursor_name(parameter_value,.....) ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 파라미터가 포함된 커서

커서 FOR 루프에서 커서에 파라미터를 전달할 수 있습니다. 이것은 블록에서 각 경우마다 다른 결과행 집합을 반환하는 명시적 커서를 여러 번 열고 닫을 수 있음을 의미합니다. 실행 시마다 이전 커서가 닫히고 새 파라미터 집합으로 다시 열립니다.

커서 선언의 각 형식 파라미터는 OPEN 문에 해당 실제 파라미터가 있어야 합니다. 파라미터 데이터 유형은 스칼라 변수의 데이터 유형과 동일하며 파라미터 크기는 지정하지 않습니다. 파라미터 이름은 커서의 표현식에서 참조용으로 사용됩니다.

이 구문에서 다음이 적용됩니다.

|                         |                       |
|-------------------------|-----------------------|
| <i>cursor_name</i>      | 선언된 커서에 대한 PL/SQL 식별자 |
| <i>parameter_name</i>   | 파라미터 이름               |
| <i>datatype</i>         | 파라미터의 스칼라 데이터 유형      |
| <i>select_statement</i> | INTO 절이 없는 SELECT 문   |

파라미터 표기법에서는 이 이상의 기능을 제공하지 않습니다. 이 표기법을 사용하면 쉽고 분명하게 입력 값을 지정할 수 있습니다. 이것은 특히 동일한 커서를 반복해서 참조하는 경우 유용합니다.

## 파라미터가 포함된 커서

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR emp_cursor (deptno NUMBER) IS
 SELECT employee_id, last_name
 FROM employees
 WHERE department_id = deptno;
 dept_id NUMBER;
 lname VARCHAR2(15);
BEGIN
 OPEN emp_cursor (10);
 ...
 CLOSE emp_cursor;
 OPEN emp_cursor (20);
 ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 파라미터가 포함된 커서(계속)

파라미터 데이터 유형은 스칼라 변수의 데이터 유형과 동일하며 파라미터 크기는 지정하지 않습니다. 파라미터 이름은 커서의 query에서 참조용으로 사용됩니다. 다음 예제에는 커서가 선언되고 하나의 파라미터로 정의되어 있습니다.

```
DECLARE
 CURSOR emp_cursor(deptno NUMBER) IS SELECT ...
```

다음 명령문은 커서를 열고 각기 다른 결과행 집합을 반환합니다.

```
OPEN emp_cursor(10);
OPEN emp_cursor(20);
```

커서 FOR 루프에 사용된 커서로 파라미터를 전달할 수 있습니다.

```
DECLARE
 CURSOR emp_cursor(p_deptno NUMBER, p_job VARCHAR2) IS
 SELECT ...
BEGIN
 FOR emp_record IN emp_cursor(10, 'Sales') LOOP ...
```

## FOR UPDATE 절

### 구문:

```
SELECT ...
FROM ...
FOR UPDATE [OF column_reference][NOWAIT | WAIT n];
```

- 명시적 잠금을 사용하여 트랜잭션 동안 다른 세션에 대한 액세스를 거부합니다.
- 갱신 또는 삭제 전에 행을 잠급니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FOR UPDATE 절

단일 데이터베이스에 대해 세션이 여럿인 경우 커서를 연 후 특정 테이블의 행이 갱신되었을 가능성이 있습니다. 커서를 다시 열어야만 갱신된 데이터를 볼 수 있습니다. 따라서 행을 갱신 또는 삭제하기 전에 행을 잠그는 것이 좋습니다. 커서 query에서 FOR UPDATE 절을 사용하여 행을 잠글 수 있습니다.

이 구문에서 다음이 적용됩니다.

|                         |                                        |
|-------------------------|----------------------------------------|
| <i>column_reference</i> | query가 수행된 테이블의 열(열 리스트가 사용될 수도 있습니다.) |
| NOWAIT                  | 다른 세션에서 행을 잠글 경우 Oracle 서버 오류를 반환합니다.  |

FOR UPDATE 절은 SELECT 문의 마지막 절로, ORDER BY가 있는 경우 ORDER BY 뒤에 옵니다. 다중 테이블을 query할 때 FOR UPDATE 절을 사용하여 행 잠금을 특정 테이블로 제한할 수 있습니다. FOR UPDATE OF col\_name (s)은 테이블에서 col\_name (s)을 포함하는 행만 잠급니다.

## FOR UPDATE 절(계속)

SELECT ... FOR UPDATE 문은 갱신 또는 삭제할 행을 식별한 다음 결과 집합에서 각 행을 잠급니다. 이것은 행의 기존 값을 기반으로 갱신을 수행하려는 경우 유용합니다. 이러한 경우 갱신 전에 다른 세션에서 행을 변경하지 않았는지 확인해야 합니다.

선택적 키워드인 NOWAIT를 사용하면 요청한 행을 다른 유저가 잠갔을 경우 Oracle 서버가 기다리지 않습니다. 잠금을 확보하려고 다시 시도하기 전에 다른 작업을 수행할 수 있도록 제어기가 즉시 프로그램으로 반환됩니다. NOWAIT 키워드를 생략하면 행을 사용할 수 있을 때까지 Oracle 서버가 기다립니다.

예제:

```
DECLARE
 emp_cursor IS
 SELECT employee_id, last_name, FROM employees
 WHERE department_id = 80 FOR UPDATE OF salary NOWAIT;
 ...
```

Oracle 서버는 SELECT FOR UPDATE에 필요한 행에 대한 잠금을 확보할 수 없을 경우 무기한으로 기다립니다. 이러한 상황을 해결하려면 NOWAIT를 사용합니다. NOWAIT를 지정했을 경우 다른 세션에서 행을 잠갔을 때 커서를 열면 오류가 발생합니다. 나중에 커서 열기를 시도할 수 있습니다. NOWAIT 대신 WAIT를 사용하고 대기 시간(초)을 지정하여 행 잠금이 해제되었는지 확인할 수 있습니다. *n*초 후에도 여전히 행이 잠겨 있으면 오류가 반환됩니다.

FOR UPDATE OF 절을 사용한 열 참조는 필수 사항은 아니지만 가독성 및 유지 관리가 향상되므로 사용할 것을 권장합니다.

## WHERE CURRENT OF 절

구문:

```
WHERE CURRENT OF cursor ;
```

- 커서를 사용하여 현재 행을 갱신 또는 삭제합니다.
- 먼저 행을 잠그도록 커서 query에 FOR UPDATE 절을 포함시킵니다.
- WHERE CURRENT OF 절을 사용하여 명시적 커서에서 현재 행을 참조합니다.

```
UPDATE employees
 SET salary = ...
 WHERE CURRENT OF emp_cursor;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### WHERE CURRENT OF 절

WHERE CURRENT OF 절은 FOR UPDATE 절과 함께 사용되어 명시적 커서의 현재 행을 참조합니다. WHERE CURRENT OF 절은 UPDATE 또는 DELETE 문에서 사용하는 반면 FOR UPDATE 절은 커서 선언에서 지정합니다. 두 절을 함께 사용하면 해당 데이터베이스 테이블에서 현재 행을 갱신 또는 삭제할 수 있습니다. 따라서 행 ID를 명시적으로 참조하지 않고도 현재 처리 중인 행에 갱신 및 삭제를 적용할 수 있습니다. FOR UPDATE 절을 커서 query에 포함시켜서 행이 OPEN에서 잠기도록 해야 합니다.

이 구문에서 다음이 적용됩니다.

*cursor*            선언된 커서의 이름(커서가 FOR UPDATE 절로 선언되었을 것입니다.)

## subquery가 포함된 커서

예제:

```
DECLARE
 CURSOR my_cursor IS
 SELECT t1.department_id, t1.department_name,
 t2.staff
 FROM departments t1, (SELECT department_id,
 COUNT(*) AS staff
 FROM employees
 GROUP BY department_id) t2
 WHERE t1.department_id = t2.department_id
 AND t2.staff >= 3;
 ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### subquery가 포함된 커서

subquery는 일반적으로 다른 SQL 문 안에 괄호로 묶여 있는 query입니다. subquery를 평가한 후 그 결과를 외부에 값 또는 값 집합으로 제공합니다. subquery는 때때로 select 문의 WHERE 절에 사용됩니다. 또한 FROM 절에 사용되어 해당 query에 대한 임시 데이터 소스를 생성합니다.

슬라이드의 예제에서 subquery는 부서 번호 및 각 부서의 사원 수(alias가 STAFF임)로 구성된 데이터 소스를 생성합니다. alias가 t2인 테이블이 FROM 절에서 이 임시 데이터 소스를 참조합니다. 이 커서가 열려 있으면 근무하는 사원이 세 명 이상인 부서의 부서 번호, 부서 이름 및 해당 부서에 근무하는 전체 사원 수가 결과행 집합에 포함됩니다.

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- **커서 유형 구분:**
  - 암시적 커서: 모든 DML 문 및 단일 행 query에 사용
  - 명시적 커서: 다중(0개 이상) 행 query에 사용
- **명시적 커서 생성 및 처리**
- **간단한 루프 및 커서 FOR 루프를 사용하여 커서의 다중 행 처리**
- **커서 속성을 사용하여 커서 상태 평가**
- **FOR UPDATE 및 WHERE CURRENT OF 절을 사용하여 현재 패치(fetch)된 행 갱신 또는 삭제**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

Oracle 서버는 작업 영역을 사용하여 SQL 문을 실행하고 처리 정보를 저장합니다. 커서라고 하는 PL/SQL 생성자를 사용하여 작업 영역 이름을 지정하고 해당 저장 정보에 액세스할 수 있습니다. 커서에는 암시적 커서와 명시적 커서 두 종류가 있습니다. PL/SQL은 단일 행만 반환하는 query를 포함한 모든 SQL 데이터 조작문에 대해 커서를 암시적으로 선언합니다. 둘 이상의 행을 반환하는 query의 경우 커서를 명시적으로 선언하여 개별적으로 행을 처리해야 합니다.

모든 명시적 커서 및 커서 변수에는 %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT의 네 개 속성이 있습니다. 이러한 속성을 커서 변수 이름에 추가하면 SQL 문 실행에 대한 유용한 정보가 반환됩니다. 커서 속성은 프로시저문에는 사용할 수 있지만 SQL 문에는 사용할 수 없습니다.

간단한 루프 또는 커서 FOR 루프를 사용하여 커서에서 패치(fetch)된 다중 행을 처리할 수 있습니다. 간단한 루프를 사용하는 경우 커서를 열고 패치하고 닫아야 하지만 커서 FOR 루프는 이러한 과정을 암시적으로 수행합니다. 행을 갱신 또는 삭제하는 경우 FOR UPDATE 절을 사용하여 행을 잠급니다. 이렇게 하면 커서를 연 후에 사용하고 있는 데이터가 다른 세션에 의해 갱신되지 않습니다. WHERE CURRENT OF 절과 FOR UPDATE 절을 함께 사용하면 커서에서 패치된 현재 행을 참조할 수 있습니다.



## 연습 7: 개요

이 연습에서는 다음 내용을 다룹니다.

- 명시적 커서를 선언 및 사용하여 테이블 행에 query 수행
- 커서 FOR 루프 사용
- 커서 속성을 적용하여 커서 상태 테스트
- 파라미터가 포함된 커서 선언 및 사용
- FOR UPDATE 및 WHERE CURRENT OF 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 7: 개요

이 연습에서는 그 동안 배운 커서 지식을 활용하여 테이블에서 여러 행을 처리하고 커서 FOR 루프를 사용하여 다른 테이블을 결과로 채워 봅니다. 또한 파라미터가 포함된 커서를 작성해 봅니다.

## 연습 7

1. 상위  $n$ 번째까지의 고액 급여를 판별하는 PL/SQL 블록을 생성합니다.
  - a. 사원의 급여를 저장하기 위해 lab\_07\_01.sql 스크립트를 실행하여 새 테이블 top\_salaries를 생성합니다.
  - b. 숫자  $n$ 은 유저가 입력하는 값입니다. 여기서  $n$ 은 employees 테이블의 급여 중 상위  $n$ 번째를 나타냅니다. 예를 들어, 상위 다섯 개의 급여를 보려면 5를 입력합니다.  
 주:  $n$ 에 대한 값을 제공하려면 DEFINE 명령을 사용하여 변수 p\_num을 정의합니다. 치환 변수를 통해 PL/SQL 블록으로 값이 전달됩니다.
  - c. 선언 섹션에서 치환 변수 p\_num을 수용할 NUMBER 유형의 num 변수와 employees.salary 유형의 sal 변수를 선언합니다. 사원의 급여를 내림차순으로 읽어 들이는 emp\_cursor 커서를 선언합니다. 급여는 중복될 수 없습니다.
  - d. 실행 섹션에서 루프를 열고 상위  $n$ 번째까지의 급여를 패치(fetch)한 다음 top\_salaries 테이블에 삽입합니다. 간단한 루프를 사용하여 데이터를 산출할 수 있습니다. 또한 종료 조건에 %ROWCOUNT 및 %FOUND 속성을 사용합니다.
  - e. top\_salaries 테이블에 삽입한 후 SELECT 문을 사용하여 행을 출력합니다. 결과로 employees 테이블의 상위 5개의 급여가 출력됩니다.

```

0 rows deleted
anonymous block completed
SALARY

24000
17000
14000
13500
13000

5 rows selected

```

- f.  $n$ 이 0이거나  $n$ 이 employees 테이블의 사원 수보다 더 큰 경우와 같이 다양한 경우를 테스트해 봅니다. 각 테스트 전후에는 top\_salaries 테이블을 비웁니다.
2. 다음을 수행하는 PL/SQL 블록을 생성합니다.
  - a. DEFINE 명령을 사용하여 부서 ID를 제공하는 p\_deptno 변수를 정의합니다.
  - b. 선언 섹션에서 NUMBER 유형의 deptno 변수를 선언하고 p\_deptno의 값을 할당합니다.
  - c. deptno에 지정된 부서에 근무하는 사원의 last\_name, salary 및 manager\_id를 읽어 들이는 emp\_cursor 커서를 선언합니다.

## 연습 7(계속)

- d. 실행 섹션에서 커서 FOR 루프를 사용하여 검색된 데이터에 대해 작업할 수 있습니다. 사원의 급여가 5000 미만이고 관리자 ID가 101 또는 124인 경우 `<<last_name>> Due for a raise` 메시지를 표시합니다. 그렇지 않은 경우 `<<last_name>> Not due for a raise` 메시지를 표시합니다.
- e. 다음 경우에 대해 PL/SQL 블록을 테스트합니다.

| Department ID | Message                                                                                                                                                                                        |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10            | Whalen Due for a raise                                                                                                                                                                         |
| 20            | Hartstein Not Due for a raise<br>Fay Not Due for a raise                                                                                                                                       |
| 50            | Weiss Not Due for a raise<br>Fripp Not Due for a raise<br>Kaufling Not Due for a raise<br>Vollman Not Due for a raise<br>Mourgas Not Due for a raise<br>. . .<br>. . .<br>Rajs Due for a raise |
| 80            | Russel Not Due for a raise<br>Partners Not Due for a raise<br>Errazuriz Not Due for a raise<br>Cambrault Not Due for a raise<br>. . .<br>. . .                                                 |

## 연습 7(계속)

3. 파라미터가 포함된 커서를 선언하고 사용하는 PL/SQL 블록을 작성합니다. 루프에서 커서를 사용하여 `departments` 테이블에서 `department_id`가 100보다 작은 부서의 부서 번호 및 부서 이름을 읽어 들입니다. 부서 번호를 다른 커서에 파라미터로 전달하여 `employees` 테이블에서 해당 부서에 근무하고 `employee_id`가 120보다 작은 사원의 성, 직위, 채용 날짜 및 급여 등의 상세 정보를 읽어 들입니다.
  - a. 선언 섹션에서 `dept_cursor` 커서를 선언하여 `department_id`가 100보다 작은 부서의 `department_id` 및 `department_name`을 검색합니다.  
`department_id`에 따라 정렬합니다.
  - b. 부서 번호를 파라미터로 사용하여 해당 부서에 근무하고 `employee_id`가 120보다 작은 사원의 `last_name`, `job_id`, `hire_date` 및 `salary`를 검색하는 다른 커서 `emp_cursor`를 선언합니다.
  - c. 각 커서에서 검색된 값을 보유하는 변수를 선언합니다. 변수 선언 시 `%TYPE` 속성을 사용합니다.
  - d. `dept_cursor`를 열고 간단한 루프를 사용하여 값을 선언된 변수에 패치(fetch)합니다. 부서 번호 및 부서 이름을 출력합니다.
  - e. 각 부서에 대해 현재 부서 번호를 파라미터로 전달하여 `emp_cursor`를 엽니다. 다른 루프를 시작하여 `emp_cursor`의 값을 변수에 패치하고 `employees` 테이블에서 검색된 모든 상세 정보를 출력합니다.  
**주:** 각 부서의 상세 정보를 표시한 다음 한 행을 출력할 수 있습니다. 종료 조건에 적합한 속성을 사용하십시오. 또한 커서를 열기 전에 이미 열려 있는지 확인하십시오.
  - f. 모든 루프 및 커서를 닫고 실행 섹션을 종료합니다. 스크립트를 실행합니다.

## 연습 7(계속)

다음은 출력 결과의 예입니다.

```
anonymous block completed
Department Number : 10 Department Name : Administration
```

```

Department Number : 20 Department Name : Marketing
```

```

Department Number : 30 Department Name : Purchasing
Raphaely PU_MAN 07-DEC-94 11000
Khoo PU_CLERK 18-MAY-95 3100
Baida PU_CLERK 24-DEC-97 2900
Tobias PU_CLERK 24-JUL-97 2800
Himuro PU_CLERK 15-NOV-98 2600
Colmenares PU_CLERK 10-AUG-99 2500
```

```

Department Number : 40 Department Name : Human Resources
```

```

Department Number : 50 Department Name : Shipping
```

```

Department Number : 60 Department Name : IT
Hunold IT_PROG 03-JAN-90 9000
Ernst IT_PROG 21-MAY-91 6000
Austin IT_PROG 25-JUN-97 4800
Pataballa IT_PROG 05-FEB-98 4800
Lorentz IT_PROG 07-FEB-99 4200
```

```

Department Number : 70 Department Name : Public Relations
```

```

Department Number : 80 Department Name : Sales
```

```

Department Number : 90 Department Name : Executive
King AD_PRES 17-JUN-87 24000
Kochhar AD_VP 21-SEP-89 17000
De Haan AD_VP 13-JAN-93 17000

```

## 연습 7(계속)

4. lab\_06\_04\_soln.sql 스크립트를 엽니다.

- "DECLARE A CURSOR CALLED emp\_records TO HOLD salary, first\_name, and last\_name of employees" 주석을 찾아 선언을 포함시킵니다. 사용자가 지정한 부서(치환 변수 emp\_deptid)에서 사원의 salary, first\_name 및 last\_name을 검색하는 커서를 생성합니다. FOR UPDATE 절을 사용합니다.
- "INCLUDE EXECUTABLE SECTION OF INNER BLOCK HERE" 주석을 찾아 실행 블록을 시작합니다.
- department\_id가 20, 60, 80, 100 및 110인 부서에 근무하는 직원만 이번 분기에 급여 인상 대상자입니다. 유저가 이 부서 ID 중 하나를 입력했는지 확인합니다. 값이 일치하지 않으면 "SORRY, NO SALARY REVISIONS FOR EMPLOYEES IN THIS DEPARTMENT" 메시지가 출력됩니다. 값이 일치하면 emp\_records 커서가 열립니다.
- 간단한 루프를 시작하여 값을 emp\_sal, emp\_fname 및 emp\_lname 에 패치(fetch)합니다. 종료 조건에 %NOTFOUND를 사용합니다.
- CASE 식을 포함시킵니다. 다음 테이블을 참조하여 CASE 식의 WHEN 절의 조건을 지정합니다.

주: CASE 식에 이미 선언된 c\_range1, c\_hike1 등과 같은 상수를 사용합니다.

| 급여             | 인상 비율 |
|----------------|-------|
| < 6500         | 20    |
| > 6500 < 9500  | 15    |
| > 9500 < 12000 | 8     |
| > 12000        | 3     |

예를 들어, 사원의 급여가 6500보다 적으면 급여를 20%까지 인상합니다. 모든 WHEN 절에서 사원의 first\_name과 last\_name을 연결하여 INDEX BY 테이블에 저장합니다. 바로 다음 위치에 문자열을 저장할 수 있도록 변수 i의 값을 증가시킵니다. WHERE CURRENT OF 절이 있는 UPDATE 문을 포함시킵니다.

- 루프를 닫습니다. %ROWCOUNT 속성을 사용하여 수정된 레코드 수를 출력합니다. 커서를 닫습니다.
- 급여가 조정된 모든 사원의 이름을 출력하는 간단한 루프를 포함시킵니다.

주: INDEX BY 테이블에 이러한 사원의 이름이 있습니다. "CLOSE THE INNER BLOCK"이라는 주석을 찾아 END IF 문과 END 문을 포함시킵니다.

- 스크립트를 lab\_07\_04\_soln.sql로 저장합니다.



# 8

## 예외 처리

ORACLE

Copyright © 2009, Oracle. All rights reserved.



## 목표

**이 단원을 마치면 다음을 수행할 수 있습니다.**

- PL/SQL 예외 정의
- 처리되지 않은 예외 인식
- 다양한 유형의 PL/SQL 예외 처리기 나열 및 사용
- 예상치 못한 오류 트랩
- 중첩 블록에서 예외 전달이 미치는 영향 설명
- PL/SQL 예외 메시지 커스터마이징

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 단원 목적

이미 앞에서 선언 섹션 및 실행 섹션에서 PL/SQL 블록을 작성하는 방법에 대해 배웠습니다. 실행해야 할 SQL 및 PL/SQL 코드는 모두 실행 블록에 작성됩니다.

지금까지 컴파일 시간 오류를 처리하면 코드가 제대로 작동한다고 가정했습니다. 그러나 런타임 시 코드에 예상치 못한 오류가 발생할 수 있습니다. 이 단원에서는 PL/SQL 블록에서 이러한 오류를 처리하는 방법에 대해 설명합니다.

## 예외의 예제

```
SET SERVEROUTPUT ON
DECLARE
 lname VARCHAR2(15);
BEGIN
 SELECT last_name INTO lname FROM employees WHERE
 first_name='John';
 DBMS_OUTPUT.PUT_LINE ('John's last name is : '
 ||lname);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외의 예제

위 슬라이드에 표시된 예제를 살펴보십시오. 코드에 구문 오류가 없으려면 익명 블록을 성공적으로 실행할 수 있어야 합니다. 블록의 select 문은 John의 last\_name을 읽어 들입니다. 코드를 실행하면 다음과 같은 출력 결과를 볼 수 있습니다.

```
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
```

코드가 예상대로 작동하지 않습니다. SELECT 문이 행을 하나만 읽어 들일 것으로 예상했지만 다중 행을 읽어 들이고 있습니다. 런타임 시 발생하는 이러한 오류를 예외라고 합니다. 예외가 발생하면 PL/SQL 블록이 종료됩니다. PL/SQL 블록에서 이러한 예외를 처리할 수 있습니다.

## 예외의 예제

```
SET SERVEROUTPUT ON
DECLARE
 lname VARCHAR2(15);
BEGIN
 SELECT last_name INTO lname FROM employees WHERE
 first_name='John';
 DBMS_OUTPUT.PUT_LINE ('John's last name is : '
 ||lname);
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE (' Your select statement
 retrieved multiple rows. Consider using a
 cursor. ');
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외의 예제(계속)

선언 섹션(DECLARE 키워드로 시작)과 실행 섹션(BEGIN 키워드로 시작하여 END 키워드로 종료)에서 PL/SQL 블록을 작성했습니다. 오류 처리를 위해 예외 섹션이라는 다른 선택적 섹션을 포함시킵니다. 이 섹션은 EXCEPTION 키워드로 시작됩니다. 예외 섹션이 존재하는 경우 이 섹션이 PL/SQL 블록의 마지막 섹션입니다. 슬라이드에 있는 코드의 EXCEPTION 섹션을 살펴보십시오. 구문과 명령문은 이 단원 뒷부분에서 배우게 되므로 신경 쓰지 않아도 됩니다.

이전 슬라이드의 코드를 재작성하여 발생한 예외를 처리하도록 했습니다. 코드의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Your select statement retrieved multiple rows. Consider using a cursor.
```

이전과 달리 PL/SQL 프로그램이 갑자기 종료되지 않습니다. 예외가 발생할 경우 제어가 예외 섹션으로 이동하여 예외 섹션의 모든 명령문이 실행됩니다. PL/SQL 블록이 성공적으로 완료되어 정상적으로 종료됩니다.

## PL/SQL로 예외 처리

- 예외는 프로그램 실행 중에 발생한 PL/SQL 오류입니다.
- 예외는 다음과 같이 발생할 수 있습니다.
  - Oracle 서버에 의해 암시적으로 발생
  - 프로그램에 의해 명시적으로 발생
- 예외를 다음과 같이 처리할 수 있습니다.
  - 처리기로 트랩
  - 호출 환경으로 전달

ORACLE

Copyright © 2009, Oracle. All rights reserved.

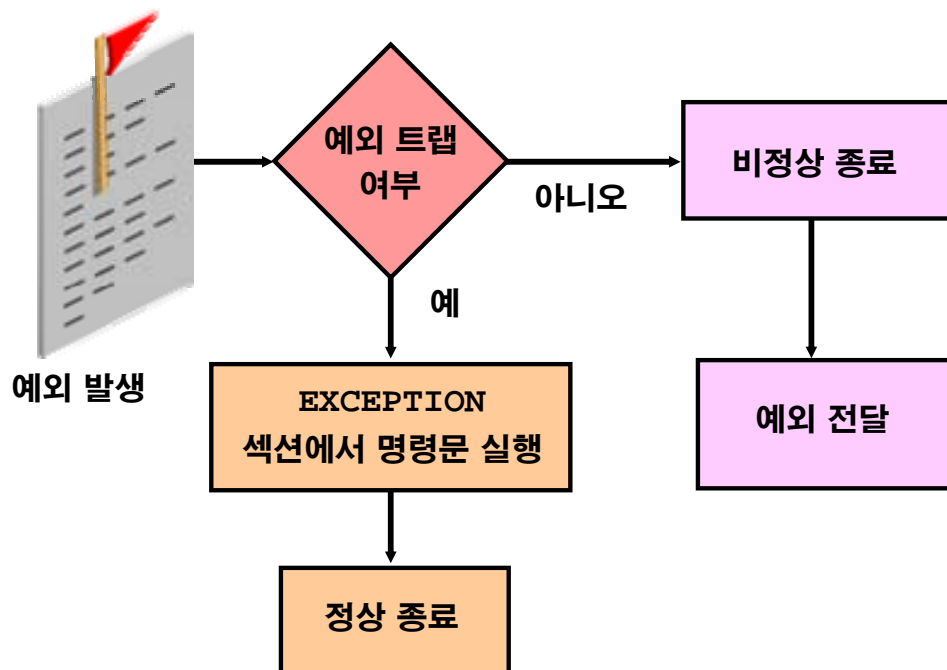
### PL/SQL로 예외 처리

예외는 블록 실행 중에 발생한 PL/SQL 오류입니다. 블록은 PL/SQL에서 예외가 발생하면 항상 종료되지만 사용자가 블록이 끝나기 전에 최종 작업을 수행하는 예외 처리기를 지정할 수 있습니다.

#### 예외를 발생시키기 위한 두 가지 방법

- Oracle 오류가 발생하면 연관된 예외가 자동으로 발생합니다. 예를 들어, `SELECT` 문 실행 시 데이터베이스에서 검색된 행이 없을 때 `ORA-01403` 오류가 발생하면 PL/SQL은 `NO_DATA_FOUND` 예외를 발생시킵니다. 이러한 오류는 미리 정의된 예외로 변환됩니다.
- 프로그램이 구현하는 업무 기능에 따라 예외를 명시적으로 발생시켜야 할 수 있습니다. 블록에서 `RAISE` 문을 실행하여 명시적으로 예외를 발생시킵니다. 발생하는 예외는 유저 정의 예외이거나 미리 정의된 예외일 수 있습니다. 미리 정의되지 않은 Oracle 오류도 있습니다. 이러한 오류는 미리 정의되지 않은 임의의 표준 Oracle 오류입니다. 명시적으로 예외를 선언하고 미리 정의되지 않은 Oracle 오류와 연관시킬 수 있습니다.

## 예외 처리



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 처리

#### 예외 트랩

예외를 트랩하려면 PL/SQL 프로그램에 EXCEPTION 섹션을 포함시키십시오. 블록의 실행 섹션에서 예외가 발생하면 처리가 블록의 예외 섹션에 있는 해당 예외 처리기로 분기됩니다. PL/SQL이 예외를 성공적으로 처리한 경우 예외는 포함하는 블록이나 호출 환경으로 전달되지 않습니다. 그러면 PL/SQL 블록이 성공적으로 종료됩니다.

#### 예외 전달

블록의 실행 섹션에서 예외가 발생하고 해당 예외 처리기가 없는 경우 PL/SQL 블록이 종료되고 오류가 발생하며, 예외가 포함하는 블록이나 호출 환경으로 전달됩니다. 호출 환경은 PL/SQL 프로그램을 실행하는 SQL\*Plus와 같은 응용 프로그램일 수 있습니다.

## 예외 유형

- 미리 정의된 Oracle 서버
  - 미리 정의되지 않은 Oracle 서버
- } 암시적으로 발생
- 
- 유저 정의
- 명시적으로 발생

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 유형

세 가지 유형의 예외가 있습니다.

| 예외                      | 설명                                       | 처리를 위한 지침                                                |
|-------------------------|------------------------------------------|----------------------------------------------------------|
| 미리 정의된 Oracle 서버 오류     | PL/SQL 코드에서 가장 자주 발생하는 약 20개 오류 중 하나입니다. | 이러한 예외는 선언할 필요가 없습니다. Oracle 서버에 의해 미리 정의되고 암시적으로 발생합니다. |
| 미리 정의되지 않은 Oracle 서버 오류 | 임의의 기타 표준 Oracle 서버 오류입니다.               | 선언 섹션 내에서 선언하여 Oracle 서버가 이 오류를 암시적으로 발생시키도록 하십시오.       |
| 유저 정의 오류                | 개발자가 비정상적인 것으로 결정한 조건입니다.                | 선언 섹션에서 선언하고 명시적으로 발생시키십시오.                              |

주: Oracle Developer Forms와 같이 클라이언트측 PL/SQL을 갖는 일부 응용 프로그램 틀은 자체적인 예외를 가집니다.

## 예외 트랩

### 구문:

```
EXCEPTION
 WHEN exception1 [OR exception2 . . .] THEN
 statement1;
 statement2;
 . . .
 [WHEN exception3 [OR exception4 . . .] THEN
 statement1;
 statement2;
 . . .]
 [WHEN OTHERS THEN
 statement1;
 statement2;
 . . .]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 트랩

PL/SQL 블록의 예외 처리 섹션 내에 해당 처리기를 포함시켜 오류를 트랩할 수 있습니다. 각 처리기에는 예외 이름을 지정하는 WHEN 절과 예외가 발생될 때 실행될 일련의 명령문이 차례로 나옵니다. 특정 예외를 처리하기 위해 EXCEPTION 섹션 내에 포함시킬 수 있는 처리기 수에는 제한이 없습니다. 그러나 단일 예외에 대해 다중 처리기를 가질 수 없습니다. 이 구문에서 다음이 적용됩니다.

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>exception</i> | 미리 정의된 예외의 표준 이름 또는 선언 섹션 내에 선언된 유저 정의 예외의 이름 |
| <i>statement</i> | 하나 이상의 PL/SQL 또는 SQL 문                        |
| OTHERS           | 명시적으로 처리되지 않은 예외를 트랩하는 선택적 예외 처리 절            |

## 예외 트랩(계속)

### WHEN OTHERS 예외 처리기

예외 처리 섹션은 지정된 예외만 트랩합니다. OTHERS 예외 처리기를 사용하지 않는 한 다른 예외는 트랩되지 않습니다. OTHERS 처리기는 아직 처리되지 않은 모든 예외를 트랩합니다. 따라서 OTHERS를 사용할 수 있는데, 이 경우 OTHERS는 마지막으로 정의된 예외 처리기여야 합니다.

```
WHEN NO_DATA_FOUND THEN
 statement1;
...
WHEN TOO_MANY_ROWS THEN
 statement1;
...
WHEN OTHERS THEN
 statement1;
```

앞의 예제를 살펴보십시오. 프로그램에서 NO\_DATA\_FOUND 예외가 발생하면 해당 처리기의 명령문이 실행됩니다. TOO\_MANY\_ROWS 예외가 발생하면 해당 처리기의 명령문이 실행됩니다. 그러나 기타 예외가 발생하면 OTHERS 예외 처리기의 명령문이 실행됩니다.

OTHERS 처리기는 아직 트랩되지 않은 모든 예외를 트랩합니다. 일부 Oracle 도구에는 응용 프로그램에서 이벤트를 유발하기 위해 발생시킬 수 있는 고유의 미리 정의된 예외가 있습니다. OTHERS 처리기는 이러한 예외도 트랩합니다.



## 예외 트랩에 대한 지침

- 예외 처리 섹션은 **EXCEPTION** 키워드로 시작합니다.
- 여러 예외 처리기를 사용할 수 있습니다.
- 블록을 종료하기 전 하나의 처리기만 실행됩니다.
- **WHEN OTHERS**는 마지막 절입니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 트랩에 대한 지침

- **EXCEPTION** 키워드로 블록의 예외 처리 섹션을 시작합니다.
- 블록에 대해 여러 예외 처리기를 정의합니다. 각각은 고유 동작 집합을 가집니다.
- 예외가 발생할 경우 **PL/SQL**은 블록 종료 전 하나의 처리기만 실행합니다.
- **OTHERS** 절은 다른 모든 예외 처리 절 뒤에 배치합니다.
- **OTHERS** 절은 하나만 지정할 수 있습니다.
- 할당문 또는 **SQL** 문에는 예외가 나타날 수 없습니다.

## 미리 정의된 Oracle 서버 오류 트랩

- 예외 처리 루틴에서 미리 정의된 이름을 참조합니다.
- 미리 정의된 예외 예제:
  - NO\_DATA\_FOUND
  - TOO\_MANY\_ROWS
  - INVALID\_CURSOR
  - ZERO\_DIVIDE
  - DUP\_VAL\_ON\_INDEX

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 미리 정의된 Oracle 서버 오류 트랩

해당하는 예외 처리 루틴 내에서 미리 정의된 이름을 참조하여 미리 정의된 Oracle 서버 오류를 트랩합니다.

미리 정의된 예외의 자세한 리스트는 *PL/SQL User's Guide and Reference*를 참조하십시오.

주: PL/SQL은 STANDARD 패키지에 미리 정의된 예외를 선언합니다.

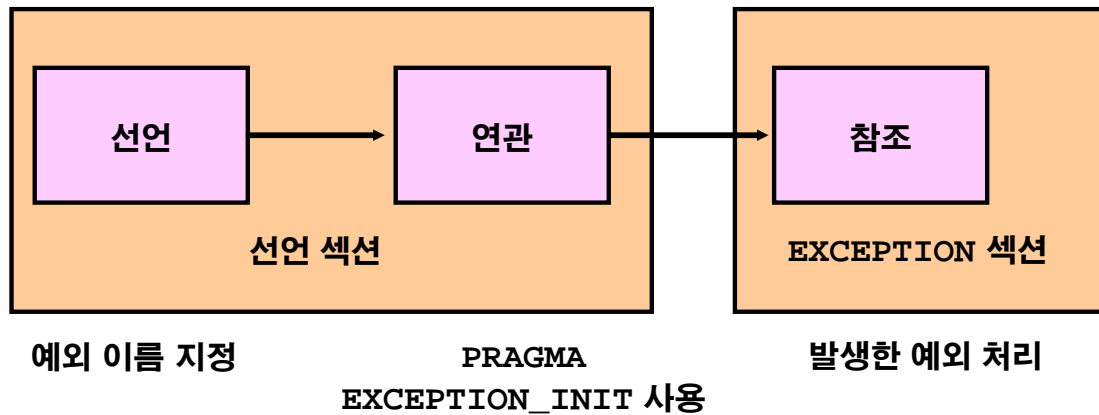
## 미리 정의된 예외

| 예외 이름               | Oracle서버<br>오류 번호 | 설명                                                         |
|---------------------|-------------------|------------------------------------------------------------|
| ACCESS_INTO_NULL    | ORA-06530         | 초기화되지 않은 객체의 속성에 값을 할당하려고 했습니다.                            |
| CASE_NOT_FOUND      | ORA-06592         | CASE 문의 WHEN 절에 있는 선택 사항 중에서 선택된 것이 없고 ELSE 절이 없습니다.       |
| COLLECTION_IS_NULL  | ORA-06531         | 초기화되지 않은 중첩 테이블 또는 VARRAY에 EXISTS가 아닌 컬렉션 메소드를 적용하려고 했습니다. |
| CURSOR_ALREADY_OPEN | ORA-06511         | 이미 열려 있는 커서를 열려고 했습니다.                                     |
| DUP_VAL_ON_INDEX    | ORA-00001         | 중복된 값을 삽입하려고 했습니다.                                         |
| INVALID_CURSOR      | ORA-01001         | 잘못된 커서 작업이 발생했습니다.                                         |
| INVALID_NUMBER      | ORA-01722         | 문자열을 숫자로 변환하는 데 실패했습니다.                                    |
| LOGIN_DENIED        | ORA-01017         | 잘못된 유저 이름 또는 암호로 Oracle 서버에 로그인하고 있습니다.                    |
| NO_DATA_FOUND       | ORA-01403         | 단일 행 SELECT가 데이터를 반환하지 않았습니다.                              |
| NOT_LOGGED_ON       | ORA-01012         | PL/SQL 프로그램이 Oracle 서버에 연결하지 않고 데이터베이스 호출을 실행합니다.          |
| PROGRAM_ERROR       | ORA-06501         | PL/SQL에 내부 문제가 있습니다.                                       |
| ROWTYPE_MISMATCH    | ORA-06504         | 할당에 관련된 호스트 커서 변수 및 PL/SQL 커서 변수가 호환되지 않는 반환 유형을 갖고 있습니다.  |

## 미리 정의된 예외(계속)

| 예외 이름                   | Oracle 서버<br>오류 번호 | 설명                                                               |
|-------------------------|--------------------|------------------------------------------------------------------|
| STORAGE_ERROR           | ORA-06500          | PL/SQL에 메모리가 부족하거나 메모리가 손상되었습니다.                                 |
| SUBSCRIPT_BEYOND_COUNT  | ORA-06533          | 컬렉션에 있는 요소 수보다 큰 인덱스 번호를 사용하여 중첩 테이블 또는 VARRAY 요소를 참조했습니다.       |
| SUBSCRIPT_OUTSIDE_LIMIT | ORA-06532          | 유효 범위를 벗어난 인덱스 번호 (예: -1)를 사용하여 중첩 테이블 또는 VARRAY 요소를 참조했습니다.     |
| SYS_INVALID_ROWID       | ORA-01410          | 문자열이 유효한 ROWID를 나타내지 않기 때문에 문자열을 universal ROWID로 변환하는 데 실패했습니다. |
| TIMEOUT_ON_RESOURCE     | ORA-00051          | Oracle 서버가 리소스를 대기하는 동안 시간 초과가 발생했습니다.                           |
| TOO_MANY_ROWS           | ORA-01422          | 단일 행 SELECT가 여러 행을 반환했습니다.                                       |
| VALUE_ERROR             | ORA-06502          | 산술, 변환, 잘림 또는 크기 제약 조건 오류가 발생했습니다.                               |
| ZERO_DIVIDE             | ORA-01476          | 0으로 나누려고 했습니다.                                                   |

## 미리 정의되지 않은 Oracle 서버 오류 트랩



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 미리 정의되지 않은 Oracle 서버 오류 트랩

미리 정의되지 않은 예외는 미리 정의된 예외와 유사하지만 Oracle 서버에 PL/SQL 예외로 정의되어 있지 않습니다. 미리 정의되지 않은 예외는 표준 Oracle 오류입니다. PRAGMA EXCEPTION\_INIT 함수를 사용하여 표준 Oracle 오류가 있는 예외를 생성합니다. 이러한 예외를 미리 정의되지 않은 예외라고 합니다.

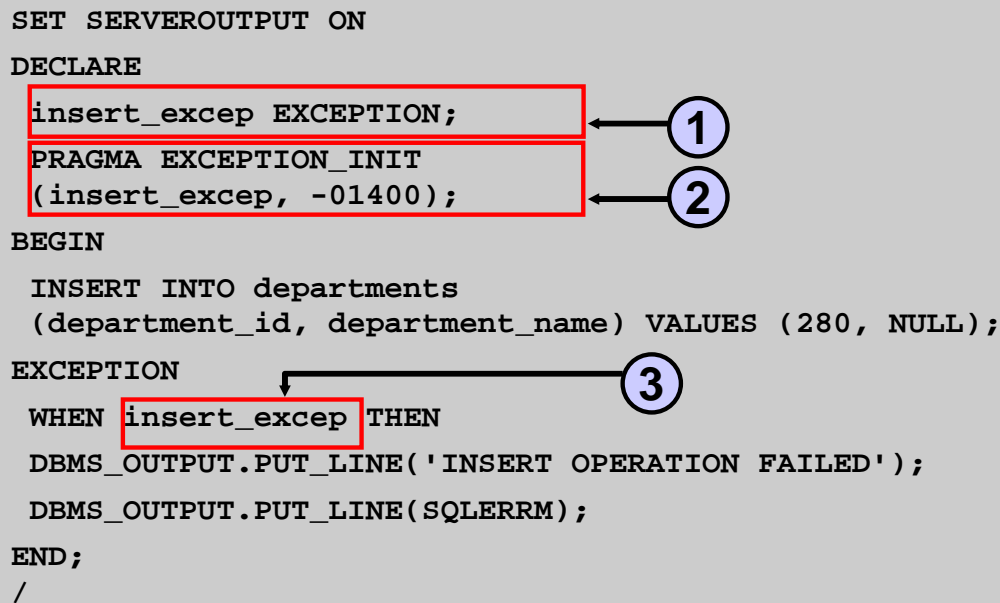
먼저 이 예외를 선언하면 미리 정의되지 않은 Oracle 서버 오류를 트랩할 수 있습니다. 선언된 예외는 암시적으로 발생합니다. PL/SQL에서 PRAGMA EXCEPTION\_INIT는 컴파일러에게 예외 이름을 Oracle 오류 번호와 연관시키도록 지시합니다. 이렇게 하면 모든 내부 예외를 이름으로 참조하고 이 예외에 대한 특정 처리기를 작성할 수 있습니다.

**주:** PRAGMA(의사 명령어라고도 함)는 명령문이 컴파일러 지시어임을 의미하는 키워드로서 PL/SQL 블록 실행 시 처리되지 않습니다. 이 키워드는 PL/SQL 컴파일러가 블록 내의 모든 예외 이름을 연관된 Oracle 서버 오류 번호로 해석하도록 지시합니다.

## 미리 정의되지 않은 오류

Oracle 서버 오류 번호 -01400("cannot insert NULL")을 트랩하려면 다음과 같이 하십시오.

```
SET SERVEROUTPUT ON
DECLARE
insert_excep EXCEPTION;
PRAGMA EXCEPTION_INIT
(insert_excep, -01400);
BEGIN
INSERT INTO departments
(department_id, department_name) VALUES (280, NULL);
EXCEPTION
WHEN insert_excep THEN
DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 미리 정의되지 않은 오류

- 선언 섹션에 예외 이름을 선언합니다.  
구문:  
`exception EXCEPTION;`  
이 구문에서 *exception*은 예외의 이름입니다.
- PRAGMA EXCEPTION\_INIT 함수를 사용하여 선언된 예외를 표준 Oracle 서버 오류 번호와 연관시킵니다.  
구문:  
`PRAGMA EXCEPTION_INIT(exception, error_number);`  
이 구문에서 *exception*은 앞서 선언한 예외이고 *error\_number*는 표준 Oracle 서버 오류 번호입니다.
- 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

#### 예제

슬라이드에 표시된 예제는 departments 테이블의 department\_name 열에 대해 NULL 값을 삽입하려고 합니다. 그러나 department\_name이 NOT NULL 열이기 때문에 작업이 성공적으로 수행되지 않습니다. 예제에서 다음 행을 살펴보십시오.

```
DBMS_OUTPUT.PUT_LINE(SQLERRM);
```

SQLERRM 함수는 오류 메시지를 읽어 들이는 데 사용됩니다. 다음 몇 개의 슬라이드에서 SQLERRM에 대해 자세히 설명합니다.

## 예외 트랩에 대한 함수

- **SQLCODE**: 오류 코드에 대한 숫자 값을 반환합니다.
- **SQLERRM**: 오류 번호와 연관된 메시지를 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 트랩에 대한 함수

예외가 발생할 때 아래의 두 함수를 사용하여 연관된 오류 코드 또는 오류 메시지를 식별할 수 있습니다. 코드 또는 메시지 값을 기반으로 수행할 후속 작업을 결정할 수 있습니다.

SQLCODE는 내부 예외에 대한 Oracle 오류 번호를 반환합니다. SQLERRM은 오류 번호와 연관된 메시지를 반환합니다.

| 함수      | 설명                                                 |
|---------|----------------------------------------------------|
| SQLCODE | 오류 코드에 대한 숫자 값을 반환합니다. 이 값을 NUMBER 변수에 할당할 수 있습니다. |
| SQLERRM | 오류 번호와 연관된 메시지를 포함하는 문자 데이터를 반환합니다.                |

### SQLCODE 값: 예제

| SQLCODE 값 | 설명                 |
|-----------|--------------------|
| 0         | 예외가 발생하지 않음        |
| 1         | 유저 정의 예외           |
| +100      | NO_DATA_FOUND 예외   |
| 음수        | 기타 Oracle 서버 오류 번호 |

## 예외 트랩에 대한 함수

예제:

```
DECLARE
 error_code NUMBER;
 error_message VARCHAR2(255);
BEGIN
 ...
EXCEPTION
 ...
 WHEN OTHERS THEN
 ROLLBACK;
 error_code := SQLCODE ;
 error_message := SQLERRM ;
 INSERT INTO errors (e_user, e_date, error_code,
 error_message) VALUES (USER, SYSDATE, error_code,
 error_message);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 예외 트랩에 대한 함수(계속)

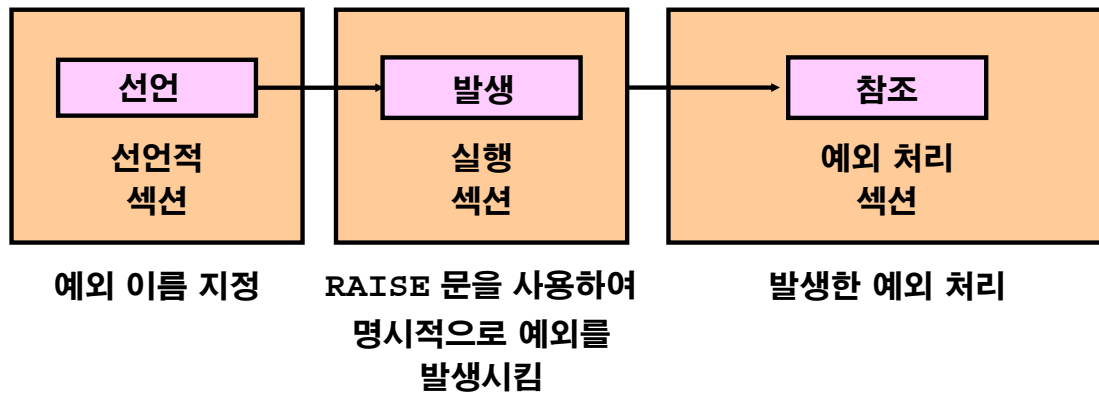
예외가 WHEN OTHERS 예외 처리기에서 트랩되면 일련의 일반 함수를 사용하여 오류를 식별할 수 있습니다. 슬라이드의 예제에서는 SQLCODE 및 SQLERRM 값을 변수에 할당한 다음 이 변수를 SQL 문에 사용하고 있습니다.

SQL 문에서 직접 SQLCODE 또는 SQLERRM을 사용할 수 없습니다. 대신 다음 예제와 같이 해당 값을 로컬 변수에 할당한 다음 SQL 문에서 이 변수를 사용해야 합니다.

```
DECLARE
 err_num NUMBER;
 err_msg VARCHAR2(100);
BEGIN
 ...
EXCEPTION
 ...
 WHEN OTHERS THEN
 err_num := SQLCODE;
 err_msg := SUBSTR(SQLERRM, 1, 100);
 INSERT INTO errors VALUES (err_num, err_msg);
END;
/
```



## 유저 정의 예외 트랩



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 유저 정의 예외 트랩

PL/SQL을 사용하면 응용 프로그램의 요구 사항에 따라 고유의 예외를 정의할 수 있습니다. 예를 들어, 유저에게 부서 번호를 입력하도록 요청할 수 있습니다. 입력 데이터의 오류 조건을 처리하는 예외를 정의합니다. 부서 번호가 존재하는지 여부를 확인합니다. 부서 번호가 존재하지 않을 경우 유저 정의 예외를 발생시켜야 합니다.

PL/SQL 예외 트랩 방법은 다음과 같습니다.

- PL/SQL 블록의 선언 섹션에서 선언해야 합니다.
- RAISE 문을 사용하여 명시적으로 발생시켜야 합니다.
- EXCEPTION 섹션에서 처리해야 합니다.

## 유저 정의 예외 트랩

```
ACCEPT deptno PROMPT 'Please enter the department number:'
ACCEPT name PROMPT 'Please enter the department name:'
DECLARE
 invalid_department EXCEPTION;
 name VARCHAR2(20) := '&name';
 deptno NUMBER := &deptno;
BEGIN
 UPDATE departments
 SET department_name = name
 WHERE department_id = deptno;
 IF SQL%NOTFOUND THEN
 RAISE invalid_department;
 END IF;
 COMMIT;
EXCEPTION
 WHEN invalid_department THEN
 DBMS_OUTPUT.PUT_LINE('No such department id.');
```

```
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 유저 정의 예외 트랩(계속)

유저 정의 예외를 선언한 다음 명시적으로 발생시켜 트랩합니다.

1. 선언 섹션 내에 유저 정의 예외에 대한 이름을 선언합니다.

구문:

```
exception EXCEPTION;
```

이 구문에서 *exception*은 예외의 이름입니다.

2. RAISE 문을 사용하여 실행 가능 섹션 내에서 해당 예외를 명시적으로 발생시킵니다.

구문:

```
RAISE exception;
```

이 구문에서 *exception*은 앞서 선언한 예외입니다.

3. 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

### 예제

이 블록은 부서의 department\_name을 갱신합니다. 유저가 부서 번호와 새 이름을 제공합니다. 유저가 존재하지 않는 부서 번호를 입력한 경우 departments 테이블에서 행이 갱신되지 않습니다. 예외를 발생시키고 유효하지 않은 부서 번호가 입력되었음을 알리는 메시지를 출력합니다.

주: 예외 처리기 내에서 RAISE 문만 사용하여 동일한 예외를 다시 발생시킨 다음 호출 환경으로 다시 전달하십시오.

## 호출 환경

|                        |                                                                       |
|------------------------|-----------------------------------------------------------------------|
| SQL*Plus               | 오류 번호 및 메시지를 화면에 표시합니다.                                               |
| SQL Developer          | 오류 번호 및 메시지를 화면에 표시합니다.                                               |
| Oracle Developer Forms | ERROR_CODE 및 ERROR_TEXT 패키지 함수를 통해 ON-ERROR 트리거의 오류 번호 및 메시지에 액세스합니다. |
| 선행 컴파일러 응용 프로그램        | SQLCA 데이터 구조를 통해 예외 번호에 액세스합니다.                                       |
| PL/SQL의 포함하는 블록        | 포함하는 블록의 예외 처리 루틴 내에서 예외를 트랩합니다.                                      |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 호출 환경

PL/SQL 블록 내의 예외를 트랩하는 대신 호출 환경이 처리할 수 있도록 예외를 전달합니다. 각 호출 환경에는 오류를 표시하고 액세스하는 고유한 방법이 있습니다.

## 서브 블록의 예외 전달

서브 블록은 예외를 처리하거나, 포함하는 블록으로 예외를 전달할 수 있습니다.

```
DECLARE
 . . .
 no_rows exception;
 integrity exception;
 PRAGMA EXCEPTION_INIT (integrity, -2292);
BEGIN
 FOR c_record IN emp_cursor LOOP
 BEGIN
 SELECT ...
 UPDATE ...
 IF SQL%NOTFOUND THEN
 RAISE no_rows;
 END IF;
 END;
 END LOOP;
EXCEPTION
 WHEN integrity THEN ...
 WHEN no_rows THEN ...
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 서브 블록의 예외 전달

서브 블록이 예외를 처리하면 서브 블록이 정상적으로 종료되고 서브 블록의 END 문 바로 뒤에 나오는 포함하는 블록에서 제어가 재개됩니다.

그러나 PL/SQL이 예외를 발생시켰을 때 현재 블록에 해당 예외에 대한 처리기가 없을 경우 처리기를 찾을 때까지 예외가 이어지는 포함 블록으로 전달됩니다. 이러한 블록 중 어떤 것도 예외를 처리하지 않는 경우 호스트 환경에 처리되지 않은 예외가 발생합니다.

예외가 포함하는 블록으로 전달될 때 그 블록의 나머지 실행 가능 작업은 생략됩니다.

이 동작의 한 가지 장점은 일반적인 예외 처리는 포함하는 블록에 남겨 두고, 고유의 배타적인 오류 처리가 필요한 명령문을 서브 블록에 묶을 수 있다는 것입니다.

예제를 살펴보면 예외(no\_rows 및 integrity)가 외부 블록에서 선언됩니다. 내부 블록에서 no\_rows 예외가 발생할 때 PL/SQL이 서브 블록에서 처리할 예외를 찾습니다.

서브 블록에서 예외가 처리되지 않으므로 PL/SQL이 처리기를 발견한 외부 블록으로 예외가 전달됩니다.

## RAISE\_APPLICATION\_ERROR 프로시저

### 구문:

```
raise_application_error (error_number,
 message[, {TRUE | FALSE}]);
```

- 이 프로시저를 사용하여 내장 서브 프로그램에서 유저 정의 오류 메시지를 실행할 수 있습니다.
- 응용 프로그램에 오류를 보고하고 처리되지 않은 예외가 반환되지 않도록 할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### RAISE\_APPLICATION\_ERROR 프로시저

RAISE\_APPLICATION\_ERROR 프로시저를 사용하면 비표준 오류 코드 및 오류 메시지를 반환하여 미리 정의된 예외를 대화식으로 전달할 수 있습니다. RAISE\_APPLICATION\_ERROR를 사용하여 응용 프로그램에 오류를 보고하고 처리되지 않은 예외가 반환되지 않도록 할 수 있습니다.

이 구문에서 다음이 적용됩니다.

|                     |                                                                                        |
|---------------------|----------------------------------------------------------------------------------------|
| <i>error_number</i> | 예외에 대한 유저 지정 번호로, -20000 ~ -20999 범위의 값입니다.                                            |
| <i>message</i>      | 예외에 대한 유저 지정 메시지로, 최대 길이가 2,048바이트인 문자열입니다.                                            |
| TRUE   FALSE        | 선택적 부울 파라미터입니다. TRUE인 경우 오류가 이전 오류의 스택에 위치하게 되고, FALSE(기본값)인 경우 이전 오류가 모두 해당 오류로 바뀝니다. |

## RAISE\_APPLICATION\_ERROR 프로시저

- 서로 다른 두 위치에 사용됩니다.
  - 실행 섹션
  - 예외 섹션
- 다른 Oracle 서버 오류와 일치하는 방식으로 유저에게 오류 조건을 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### RAISE\_APPLICATION\_ERROR 프로시저(계속)

RAISE\_APPLICATION\_ERROR 프로시저는 PL/SQL 프로그램의 실행 섹션이나 예외 섹션 또는 둘 다에서 사용할 수 있습니다. 반환된 오류는 Oracle 서버가 미리 정의된 오류, 미리 정의되지 않은 오류 또는 유저 정의 오류를 생성하는 방법과 일치합니다. 유저에게 오류 번호 및 메시지가 표시됩니다.

# RAISE\_APPLICATION\_ERROR 프로시저

## 실행 섹션:

```
BEGIN
...
DELETE FROM employees
WHERE manager_id = v_mgr;
IF SQL%NOTFOUND THEN
RAISE_APPLICATION_ERROR(-20202,
 'This is not a valid manager');
END IF;
...
```

## 예외 섹션:

```
...
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 RAISE_APPLICATION_ERROR (-20201,
 'Manager is not a valid employee.');
```

```
END; /
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## RAISE\_APPLICATION\_ERROR 프로시저(계속)

이 슬라이드는 PL/SQL 프로그램의 실행 섹션과 예외 섹션 모두에서 RAISE\_APPLICATION\_ERROR 프로시저를 사용할 수 있음을 보여줍니다.

다음은 RAISE\_APPLICATION\_ERROR 프로시저를 사용하는 또 다른 예제입니다.

```
DECLARE
 e_name EXCEPTION;
 PRAGMA EXCEPTION_INIT (e_name, -20999);
BEGIN
 ...
 DELETE FROM employees
 WHERE last_name = 'Higgins';
 IF SQL%NOTFOUND THEN
 RAISE_APPLICATION_ERROR(-20999, 'This is not a
 valid last name');
 END IF;
EXCEPTION
 WHEN e_name THEN
 -- handle the error
 ...
END;
/
```

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- PL/SQL 예외 정의
- PL/SQL 블록에 `EXCEPTION` 섹션을 추가하여 런타임 시 예외 처리
- 다양한 유형의 예외 처리:
  - 미리 정의된 예외
  - 미리 정의되지 않은 예외
  - 유저 정의 예외
- 중첩 블록 및 호출 응용 프로그램에서의 예외 전달

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

이 단원에서는 다양한 유형의 예외 처리 방법을 배웠습니다. PL/SQL에서 런타임 시 발생하는 경고 또는 오류 조건을 예외라고 합니다. 미리 정의된 예외는 Oracle 서버에 의해 정의된 오류 조건입니다. 미리 정의되지 않은 예외는 임의의 표준 Oracle 서버 오류일 수 있습니다. 유저 정의 예외는 유저의 응용 프로그램 고유의 예외입니다. 선언된 예외 이름과 Oracle 서버 오류를 연관시키기 위해 `PRAGMA EXCEPTION_INIT` 함수를 사용할 수 있습니다.

PL/SQL 블록의 선언 섹션에서 유저 고유의 예외를 정의할 수 있습니다. 예를 들어, 초과 인출된 은행 계좌를 플래그로 지정하는 `INSUFFICIENT_FUNDS`라는 예외를 정의할 수 있습니다.

오류가 발생하면 예외가 발생합니다. 정상적인 실행이 정지되고 PL/SQL 블록의 예외 처리 섹션으로 제어가 넘어갑니다. 내부 예외가 런타임 시스템에 의해 암시적으로 (자동으로) 발생하는 반면, 유저 정의 오류는 명시적으로 발생되어야 합니다. 발생한 예외를 처리하기 위해 예외 처리기라는 별도의 루틴을 작성합니다.



## 연습 8: 개요

이 연습에서는 다음 내용을 다룹니다.

- 명명된 예외 처리
- 유저 정의 예외 생성 및 호출

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 8: 개요

이 연습에서는 특정 상황의 예외 처리기를 생성하는 방법을 다룹니다.

## 연습 8

1. 이 연습의 목적은 미리 정의된 예외에 대한 사용법을 제공하는 것입니다. 제공된 급여 값을 가진 사원의 이름을 선택하는 PL/SQL 블록을 작성합니다.
  - a. `messages` 테이블의 모든 레코드를 삭제합니다. `DEFINE` 명령을 사용하여 `sal` 변수를 정의하고 6000으로 초기화합니다.
  - b. 선언 섹션에서 `employees.last_name` 유형의 `ename` 변수와 `employees.salary` 유형의 `emp_sal` 변수를 선언합니다. 대체 변수의 값을 `emp_sal`에 전달합니다.
  - c. 실행 섹션에서 급여가 `emp_sal`의 값과 동일한 사원의 이름을 검색합니다.  
**주:** 명시적 커서를 사용하지 마십시오. 입력된 급여가 행을 하나만 반환하는 경우 `messages` 테이블에 사원 이름 및 급여를 삽입하십시오.
  - d. 입력된 급여가 행을 반환하지 않으면 적합한 예외 처리기로 예외를 처리하고 `messages` 테이블에 "No employee with a salary of <salary>" 메시지를 삽입합니다.
  - e. 입력된 급여가 둘 이상의 행을 반환하면 적합한 예외 처리기로 예외를 처리하고 `messages` 테이블에 "More than one employee with a salary of <salary>" 메시지를 삽입합니다.
  - f. 기타 예외의 경우 적합한 예외 처리기로 처리하고 `messages` 테이블에 "Some other error occurred" 메시지를 삽입합니다.
  - g. `messages` 테이블의 행을 표시하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다. 다음은 출력 예입니다.

| RESULTS                                        |
|------------------------------------------------|
| 1 More than one employee with a salary of 6000 |

2. 이 연습의 목적은 표준 Oracle 서버 오류를 사용하여 예외를 선언하는 방법을 제공하는 것입니다. Oracle 서버 오류 ORA-02292(무결성 제약 조건 위반 - 하위 레코드 발견)를 사용합니다.
  - a. 선언 섹션에서 `childrecord_exists` 예외를 선언합니다. 선언된 예외를 표준 Oracle 서버 오류 -02292와 연결합니다.
  - b. 실행 섹션에서 "Deleting department 40...."을 출력합니다. `department_id`가 40인 부서를 삭제하는 `DELETE` 문을 포함시킵니다.
  - c. `childrecord_exists` 예외를 처리하고 적절한 메시지를 출력하는 예외 섹션을 포함시킵니다. 다음은 출력 예입니다.

|                                                                                              |
|----------------------------------------------------------------------------------------------|
| anonymous block completed                                                                    |
| Deleting department 40.....                                                                  |
| Cannot delete this department. There are employees in this department (child records exist.) |

## 연습 8(계속)

3. lab\_07\_04\_soln.sql 스크립트를 엽니다.

- a. 외부 블록의 선언 섹션을 살펴봅니다. no\_such\_employee 예외가 선언된 것을 확인할 수 있습니다.
- b. "RAISE EXCEPTION HERE"라는 주석을 찾습니다. emp\_id 값이 100과 206 사이에 없으면 no\_such\_employee 예외를 발생시킵니다.
- c. "INCLUDE EXCEPTION SECTION FOR OUTER BLOCK"이라는 주석을 찾아 no\_such\_employee 및 too\_many\_rows 예외를 처리합니다. 예외가 발생하면 해당 메시지를 표시합니다. employees 테이블에 HR 부서에서 근무하는 사원이 한 명 뿐이므로 그에 따라 코드가 작성됩니다. too\_many\_rows 예외가 처리되면 select 문을 실행했을 때 HR 부서에서 근무하는 사원이 두 명 이상 검색된 것입니다.
- d. 외부 블록을 닫습니다.
- e. 스크립트를 lab\_08\_03\_soln.sql로 저장합니다.
- f. 스크립트를 실행합니다. 사원 번호 및 부서 번호를 입력하고 출력을 확인합니다. 다른 값을 입력해 보고 다른 조건을 확인해 봅니다. 사원 ID가 203이고 부서 ID가 100인 예제 출력은 아래와 같습니다.

```
anonymous block completed
NUMBER OF RECORDS MODIFIED :6
The following employees' salaries are updated
Nancy Greenberg
Daniel Faviat
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
```

# 9

## 내장 프로시저 및 함수 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 익명 블록과 서브 프로그램 구분
- 간단한 프로시저 작성 및 익명 블록에서 프로시저 호출
- 간단한 함수 작성
- 파라미터를 받아들이는 간단한 함수 작성
- 프로시저와 함수 구분

ORACLE

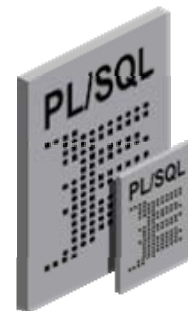
Copyright © 2009, Oracle. All rights reserved.

### 단원 목적

이미 앞에서 익명 블록에 대해 배웠습니다. 이 단원에서는 *서브 프로그램*이라고도 하는 명명된 블록을 소개합니다. 프로시저와 함수가 PL/SQL 서브 프로그램입니다. 이 단원에서는 익명 블록과 서브 프로그램을 구분하는 방법도 배웁니다.

## 프로시저 및 함수

- 명명된 PL/SQL 블록입니다.
- PL/SQL 서브 프로그램이라고 합니다.
- 익명 블록과 유사한 블록 구조를 가집니다.
  - 선택적 선언 섹션 (DECLARE 키워드를 사용하지 않음)
  - 필수 실행 섹션
  - 선택적 예외 처리 섹션



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로시저 및 함수

지금까지 이 과정에서 다룬 PL/SQL 코드의 예는 익명 블록뿐입니다. 이름에서 알 수 있듯이 **익명** 블록은 이름이 지정되지 않은 실행 가능한 블록입니다. 이름이 지정되지 않았기 때문에 재사용하거나 나중에 사용할 수 있도록 저장할 수 없습니다.

프로시저 및 함수는 명명된 PL/SQL 블록입니다. 이들을 서브 프로그램이라고도 합니다. 이러한 서브 프로그램은 컴파일한 다음 데이터베이스에 저장할 수 있습니다. 서브 프로그램의 블록 구조는 익명 블록의 구조와 유사합니다. 서브 프로그램은 스키마 레벨에서만 아니라 다른 PL/SQL 블록 내에도 선언할 수 있습니다. 서브 프로그램은 다음 섹션을 포함합니다.

- **선언 섹션:** 서브 프로그램은 선택적 선언 섹션을 가질 수 있습니다. 그러나 서브 프로그램의 선언 섹션은 익명 블록과 달리 DECLARE 키워드로 시작하지 않습니다. 서브 프로그램 선언에서 선택적 선언 섹션은 IS 또는 AS 키워드 다음에 나옵니다.
- **실행 섹션:** 서브 프로그램의 필수 섹션이며 업무 논리의 구현을 포함합니다. 이 섹션의 코드를 보면 서브 프로그램의 업무 기능을 쉽게 확인할 수 있습니다. 이 섹션은 BEGIN 키워드로 시작하여 END 키워드로 끝납니다.
- **예외 섹션:** 예외를 처리하기 위해 포함된 선택적 섹션입니다.

## 익명 블록과 서브 프로그램 간의 차이점

| 익명 블록                    | 서브 프로그램                          |
|--------------------------|----------------------------------|
| 이름이 지정되지 않은 PL/SQL 블록    | 명명된 PL/SQL 블록                    |
| 매번 컴파일됩니다.               | 한 번만 컴파일됩니다.                     |
| 데이터베이스에 저장되지 않습니다.       | 데이터베이스에 저장됩니다.                   |
| 다른 응용 프로그램에서 호출할 수 없습니다. | 명명되었으므로 다른 응용 프로그램에서 호출할 수 있습니다. |
| 값을 반환하지 않습니다.            | 함수라고 하는 서브 프로그램은 값을 반환해야 합니다.    |
| 파라미터를 사용할 수 없습니다.        | 파라미터를 사용할 수 있습니다.                |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 익명 블록과 서브 프로그램 간의 차이점

슬라이드의 테이블은 익명 블록과 서브 프로그램의 차이점을 보여주며 서브 프로그램의 일반적인 이점을 크게 강조하고 있습니다.

익명 블록은 영구적인 데이터베이스 객체가 아니며 오직 한 번 컴파일되고 실행됩니다. 또한 재사용할 수 있도록 데이터베이스에 저장되지 않습니다. 익명 블록을 재사용하려면 재컴파일과 실행이 수행되도록 익명 블록을 생성하는 스크립트를 다시 실행해야 합니다. 프로시저와 함수는 컴파일한 다음 컴파일된 형태로 데이터베이스에 저장할 수 있습니다. 프로시저와 함수는 변경된 경우에만 재컴파일됩니다. 데이터베이스에 저장되기 때문에 모든 응용 프로그램이 적절한 권한에 준하여 이러한 서브 프로그램을 사용할 수 있습니다. 프로시저가 파라미터를 받아들이도록 설계되었다면 호출하는 응용 프로그램이 파라미터를 프로시저로 전달할 수 있습니다. 마찬가지로 호출하는 응용 프로그램이 함수 또는 프로시저를 실행 중이라면 값을 검색할 수 있습니다.

## 프로시저: 구문

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(argument1 [mode1] datatype1,
 argument2 [mode2] datatype2,
 . . .)]
IS|AS
procedure_body;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로시저: 구문

이 슬라이드는 프로시저 작성을 위한 구문을 보여줍니다. 이 구문에서 다음이 적용됩니다.

|                       |                                                                                      |
|-----------------------|--------------------------------------------------------------------------------------|
| <i>procedure_name</i> | 생성할 프로시저 이름입니다.                                                                      |
| <i>argument</i>       | 프로시저 파라미터에 제공된 이름입니다. 모든 인수는 모드 및 데이터 유형과 연관되어 있습니다. 인수는 콤마로 구분하여 원하는 만큼 사용할 수 있습니다. |
| <i>mode</i>           | 인수 모드:<br>IN (기본값)<br>OUT<br>IN OUT                                                  |
| <i>datatype</i>       | 연관된 파라미터의 데이터 유형입니다. 파라미터의 데이터 유형은 명시적 크기를 갖지 않는 대신 %TYPE을 사용합니다.                    |
| <i>Procedure_body</i> | 코드를 구성하는 PL/SQL 블록입니다.                                                               |

프로시저 선언에서 인수 리스트는 선택적입니다. 프로시저에 대한 자세한 내용은 *Oracle Database 10g: Develop PL/SQL Program Units* 과정에서 다룹니다.



## 프로시저: 예제

```
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
 dept_id dept.department_id%TYPE;
 dept_name dept.department_name%TYPE;
BEGIN
 dept_id:=280;
 dept_name:='ST-Curriculum';
 INSERT INTO dept(department_id,department_name)
 VALUES(dept_id,dept_name);
 DBMS_OUTPUT.PUT_LINE(' Inserted ' ||
 SQL%ROWCOUNT || ' row ');
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로시저: 예제

슬라이드의 코드를 살펴보세요. add\_dept 프로시저는 부서 ID가 280이고 부서 이름이 ST-Curriculum인 새 부서를 삽입합니다. 이 프로시저는 선언 섹션에서 dept\_id와 dept\_name의 두 변수를 선언합니다. 프로시저의 선언 섹션은 프로시저 선언 바로 다음에 시작되고 DECLARE 키워드로 시작하지 않습니다. 프로시저는 암시적 커서 속성이나 SQL%ROWCOUNT SQL 속성을 사용하여 해당 행이 성공적으로 삽입되었는지 확인합니다. 이 경우 SQL%ROWCOUNT는 1을 반환합니다.

**주:** 테이블, 프로시저, 함수 등의 임의의 객체를 생성하면 해당 항목이 user\_objects 테이블에 작성됩니다. 슬라이드의 코드가 성공적으로 실행되면 다음 명령을 실행하여 user\_objects 테이블을 확인할 수 있습니다.

```
SELECT object_name,object_type FROM user_objects;
```

|   | OBJECT_NAME | OBJECT_TYPE |
|---|-------------|-------------|
| 1 | ADD_DEPT    | PROCEDURE   |
| 2 | DEPT        | TABLE       |

## 프로시저: 예제(계속)

프로시저의 소스는 user\_source 테이블에 저장됩니다. 다음 명령을 실행하여 프로시저의 소스를 확인할 수 있습니다.

```
SELECT * FROM user_source WHERE name='ADD_DEPT' ;
```

| LINE | NAME     | TYPE      | TEXT                                                                                                        |
|------|----------|-----------|-------------------------------------------------------------------------------------------------------------|
| 1    | ADD_DEPT | PROCEDURE | PROCEDURE add_dept IS                                                                                       |
| 2    | ADD_DEPT | PROCEDURE | dept_id dept.department_id%TYPE;                                                                            |
| 3    | ADD_DEPT | PROCEDURE | dept_name dept.department_name%TYPE;                                                                        |
| 4    | ADD_DEPT | PROCEDURE | BEGIN                                                                                                       |
| 5    | ADD_DEPT | PROCEDURE | dept_id:=280;                                                                                               |
| 6    | ADD_DEPT | PROCEDURE | dept_name:='ST-Curriculum';                                                                                 |
| 7    | ADD_DEPT | PROCEDURE | INSERT INTO dept(department_id,department_name)                                                             |
| 8    | ADD_DEPT | PROCEDURE | VALUES(dept_id,dept_name);DBMS_OUTPUT.PUT_LINE(' Inserted '  DBMS_OUTPUT.PUT_LINE(' SQL%ROWCOUNT   ' row'); |
| 9    | ADD_DEPT | PROCEDURE | END;                                                                                                        |

## 프로시저 호출

```
BEGIN
 add_dept;
END;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```

anonymous block completed  
Inserted 1 row

|   | DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---------------|-----------------|
| 1 | 280           | ST-Curriculum   |

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 프로시저 호출

이 슬라이드는 익명 블록에서 프로시저를 호출하는 방법을 보여줍니다. 익명 블록의 실행 섹션에 프로시저에 대한 호출을 포함시켜야 합니다. 마찬가지로 Forms 응용 프로그램, Java 응용 프로그램 등과 같은 응용 프로그램에서 프로시저를 호출할 수 있습니다. 코드의 select 문은 해당 행이 성공적으로 삽입되었는지 확인합니다.

CALL <procedure\_name> SQL 문을 사용하여 프로시저를 호출할 수도 있습니다.

## 함수: 구문

```
CREATE [OR REPLACE] FUNCTION function_name
 [(argument1 [mode1] datatype1,
 argument2 [mode2] datatype2,
 . . .)]
RETURN datatype
IS|AS
function_body;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 함수: 구문

이 슬라이드는 함수 작성을 위한 구문을 보여줍니다. 이 구문에서 다음이 적용됩니다.

|                        |                                                                                                           |
|------------------------|-----------------------------------------------------------------------------------------------------------|
| <i>function_name</i>   | 생성할 함수의 이름입니다.                                                                                            |
| <i>argument</i>        | 함수 파라미터에 제공된 이름입니다. 모든 인수는 모드 및 데이터 유형과 연관되어 있습니다. 인수는 콤마로 구분하여 원하는 만큼 사용할 수 있습니다. 함수를 호출하는 경우 인수를 전달합니다. |
| <i>mode</i>            | 파라미터 유형입니다. IN 파라미터만 선언되어야 합니다.                                                                           |
| <i>datatype</i>        | 연관된 파라미터의 데이터 유형입니다.                                                                                      |
| RETURN <i>datatype</i> | 함수에 의해 반환된 값의 데이터 유형입니다.                                                                                  |
| <i>function_body</i>   | 함수 코드를 구성하는 PL/SQL 블록입니다.                                                                                 |

함수 선언에서 인수 리스트는 선택적입니다. 프로시저와 함수의 차이점은 함수의 경우 호출 프로그램에 값을 반환해야 한다는 것입니다. 따라서 함수의 구문은 해당 함수가 반환하는 값의 데이터 유형을 지정하는 *return\_type*을 포함합니다. 프로시저는 OUT 또는 IN OUT 파라미터를 통해 값을 반환할 수 있습니다.

## 함수: 예제

```
CREATE FUNCTION check_sal RETURN Boolean IS
dept_id employees.department_id%TYPE;
empno employees.employee_id%TYPE;
sal employees.salary%TYPE;
avg_sal employees.salary%TYPE;
BEGIN
empno:=205;
SELECT salary,department_id INTO sal,dept_id
FROM employees WHERE employee_id= empno;
SELECT avg(salary) INTO avg_sal FROM employees
WHERE department_id=dept_id;
IF sal > avg_sal THEN
RETURN TRUE;
ELSE
RETURN FALSE;
END IF;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN NULL;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 함수: 예제

check\_sal 함수는 특정 사원의 급여가 해당 부서에 근무하는 모든 사원의 평균 급여보다 많거나 적은지를 확인하기 위해 작성되었습니다. 이 사원의 급여가 해당 부서에 근무하는 모든 사원의 평균 급여보다 많으면 TRUE가 반환되고 그렇지 않으면 FALSE가 반환됩니다. NO\_DATA\_FOUND 예외가 발생한 경우 함수는 NULL을 반환합니다.

함수는 사원 ID가 205인 사원을 검사합니다. 이 사원 ID만 검사하도록 함수가 하드 코딩되어 있습니다. 다른 사원도 검사하려면 함수 자체를 수정해야 합니다. 인수를 받아들이도록 함수를 선언하여 이 문제를 해결할 수 있습니다. 그런 다음 사원 ID를 파라미터로 전달할 수 있습니다.

## 함수 호출

```
SET SERVEROUTPUT ON
BEGIN
 IF (check_sal IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
 NULL due to exception');
 ELSIF (check_sal) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
END;
/
```

```
anonymous block completed
Salary > average
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 함수 호출

익명 블록의 실행 섹션에 함수에 대한 호출을 포함시킵니다. 함수는 명령문의 일부로 호출됩니다. check\_sal 함수는 부울 또는 NULL을 반환합니다. 따라서 함수 호출이 IF 블록의 조건식으로 포함됩니다.

주: 다음 예제와 같이 DESCRIBE 명령을 사용하여 함수의 인수 및 반환 유형을 확인할 수 있습니다.

```
DESCRIBE check_sal;
```

|                     |                |        |         |
|---------------------|----------------|--------|---------|
| DESCRIBE check_sal; |                |        |         |
| Argument Name       | Type           | In/Out | Default |
| <return value>      | PL/SQL BOOLEAN | OUT    | unknown |
| 1 rows selected     |                |        |         |

## 함수에 파라미터 전달

```
DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(empno employees.employee_id%TYPE)
RETURN Boolean IS
 dept_id employees.department_id%TYPE;
 sal employees.salary%TYPE;
 avg_sal employees.salary%TYPE;
BEGIN
 SELECT salary,department_id INTO sal,dept_id
 FROM employees WHERE employee_id=empno;
 SELECT avg(salary) INTO avg_sal FROM employees
 WHERE department_id=dept_id;
 IF sal > avg_sal THEN
 RETURN TRUE;
 ELSE
 RETURN FALSE;
 END IF;
EXCEPTION ...
...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 함수에 파라미터 전달

앞에서 사용한 함수는 사원 ID가 205인 사원의 급여를 검사하도록 하드 코딩되어 있었습니다. 위 슬라이드에 표시된 코드는 사원 번호를 파라미터로 사용하도록 재작성되었기 때문에 해당 제약 조건이 필요 없습니다. 이제 다른 사원 번호를 전달하여 해당 사원의 급여를 검사할 수 있습니다.

함수에 대한 자세한 내용은 *Oracle Database 10g: Develop PL/SQL Program Units* 과정에서 다룹니다.

## 파라미터가 포함된 함수 호출

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 205');
 IF (check_sal(205) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
 NULL due to exception');
 ELSIF (check_sal(205)) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 70');
 IF (check_sal(70) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
 NULL due to exception');
 ELSIF (check_sal(70)) THEN
 ...
 END IF;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 파라미터가 포함된 함수 호출

슬라이드의 코드는 파라미터를 전달하여 함수를 두 번 호출합니다. 코드의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception
```



## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 간단한 프로시저 작성
- 익명 블록에서 프로시저 호출
- 간단한 함수 작성
- 파라미터를 받아들이는 간단한 함수 작성
- 익명 블록에서 함수 호출

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

익명 블록을 사용하여 PL/SQL의 모든 기능을 설계할 수 있습니다. 그러나 익명 블록의 주된 제약 조건은 저장되지 않으므로 재사용할 수 없다는 것입니다.

익명 블록을 생성하는 대신 PL/SQL 서브 프로그램을 생성할 수 있습니다. 프로시저와 함수는 서브 프로그램이라고 하며 서브 프로그램은 명명된 PL/SQL 블록입니다. 서브 프로그램은 파라미터를 활용하여 재사용 가능한 논리를 나타냅니다. 프로시저나 함수의 구조는 익명 블록의 구조와 유사합니다. 이러한 서브 프로그램은 데이터베이스에 저장되므로 재사용할 수 있습니다.

## 연습 9: 개요

이 연습에서는 다음 내용을 다룹니다.

- 기존 익명 블록을 프로시저로 변환
- 파라미터를 받아들이도록 프로시저 수정
- 프로시저를 호출하는 익명 블록 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 연습 9

1. SQL Developer에서 연습 2의 질문 4에서 작성한 lab\_02\_04\_soln.sql 스크립트를 엽니다.
  - a. 익명 블록을 greet라는 프로시저로 변환하도록 스크립트를 수정합니다.
  - b. 스크립트를 실행하여 프로시저를 작성합니다.
  - c. 스크립트를 lab\_09\_01\_soln.sql로 저장합니다.
  - d. Clear 버튼을 눌러 작업 영역을 지웁니다.
  - e. greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 다음은 출력 예입니다.

```
anonymous block completed
Hello World
TODAY IS : 05-FEB-09
TOMORROW IS : 06-FEB-09
```

2. lab\_09\_01\_soln.sql 스크립트를 엽니다.
  - a. 다음 명령을 실행하여 greet 프로시저를 삭제합니다. DROP PROCEDURE greet
  - b. 이 프로시저를 VARCHAR2 유형의 인수를 받아들이도록 수정합니다. name 인수를 호출합니다.
  - c. Hello World 대신 Hello <name>을 출력합니다.
  - d. 스크립트를 lab\_09\_02\_soln.sql로 저장합니다.
  - e. 스크립트를 실행하여 프로시저를 작성합니다.
  - f. 파라미터가 포함된 greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 다음은 출력 예입니다.

```
anonymous block completed
Hello Neema
TODAY IS : 05-FEB-09
TOMORROW IS : 06-FEB-09
```

---

## A

### 연습 해답

---

## 연습 1

labs 폴더는 스크립트를 저장할 수 있는 작업 디렉토리입니다. 강사에게 문의하여 본 과정에 사용할 labs 폴더를 찾으십시오. 모든 연습의 해답은 soln 폴더에 있습니다.

1. 다음 PL/SQL 블록 중 성공적으로 실행되는 블록은 무엇입니까?

- a. BEGIN  
END;
- b. DECLARE  
amount INTEGER(10);  
END;
- c. DECLARE  
BEGIN  
END;
- d. DECLARE  
amount INTEGER(10);  
BEGIN  
DBMS\_OUTPUT.PUT\_LINE(amount);  
END;

*a*의 블록은 실행 섹션에 명령문이 없기 때문에 실행되지 않습니다.

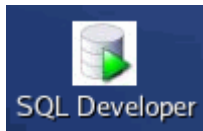
*b*의 블록에는 **BEGIN** 키워드로 시작하는 필수 실행 섹션이 없습니다.

*c*의 블록에는 필요한 항목이 모두 있지만 실행 섹션에 명령문이 없습니다.

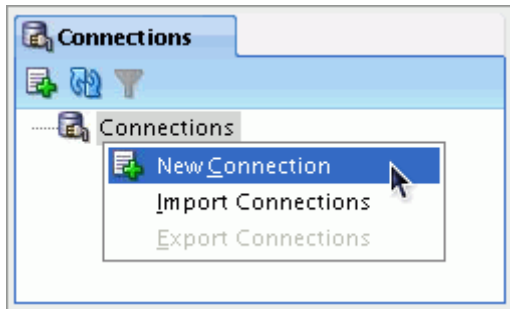
2. 다음 정보를 사용하여 SQL Developer 에서 데이터베이스 연결을 생성합니다.

Connection Name: ora41  
Username: ora41  
Password: ora41  
Hostname: localhost  
Port: 1521  
SID: orcl

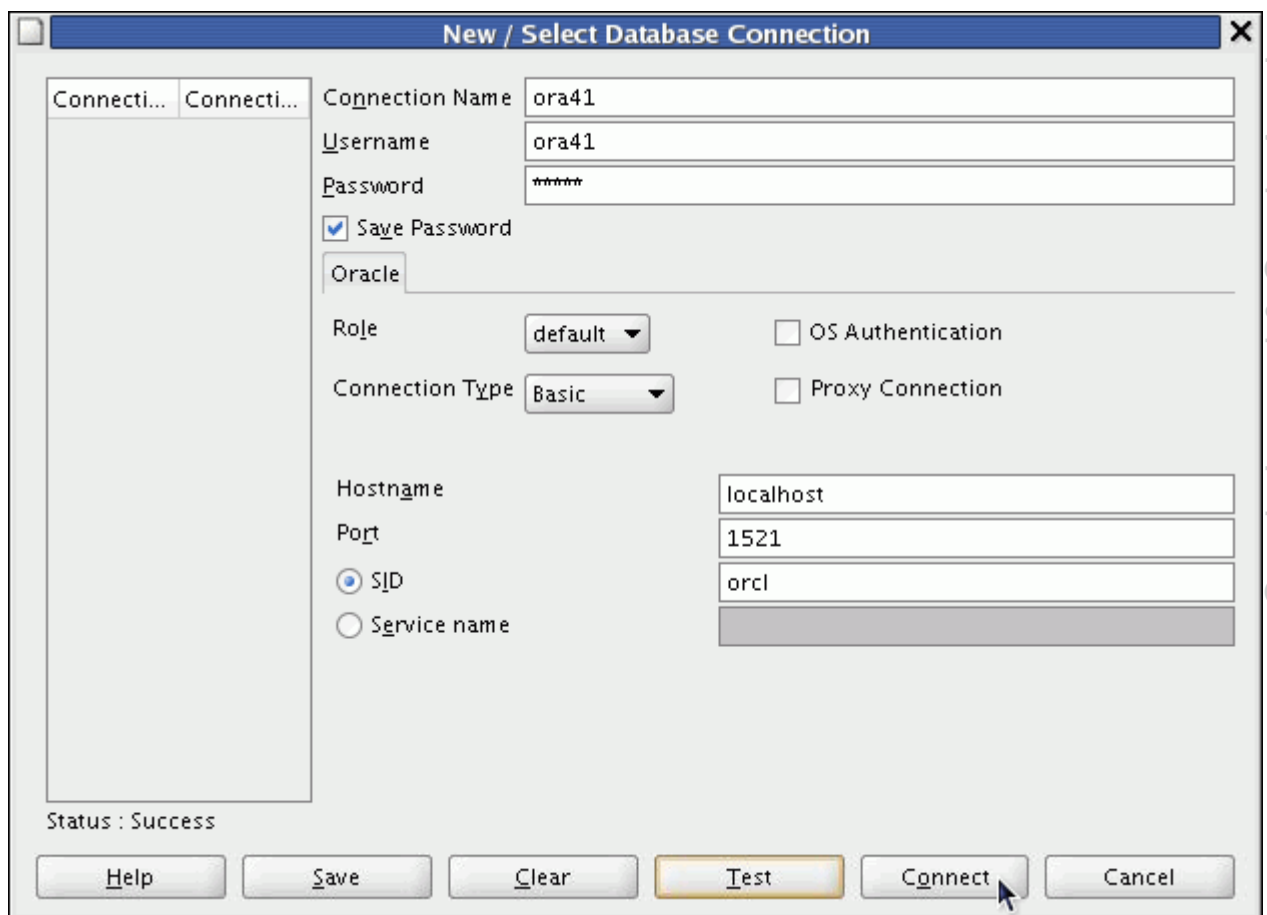
1. 바탕 화면의 아이콘에서 SQL Developer를 시작합니다.



2. Connections 탭에서 Connections를 마우스 오른쪽 버튼으로 누르고 New Connection을 선택합니다.



3. 세부 정보를 입력하고, 연결을 테스트하고 연결합니다.



3. "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다. 이 스크립트를 실행하여 lab\_01\_03\_soln.sql 로 저장합니다.

a. 작업 2 에서 생성한 ora41 연결을 엽니다. 해당 연결에 대한 SQL Worksheet 가 열립니다.

b. 작업 영역에 다음 코드를 입력합니다.

```
SET SERVEROUTPUT ON
BEGIN
DBMS_OUTPUT.PUT_LINE(' Hello World ');
END;
```

c. Run Script 아이콘을 누릅니다.

d. 출력 결과는 다음과 같아야 합니다.

```
anonymous block completed
Hello World
```

e. File>Save 를 선택하여 스크립트를 저장합니다. 파일을 저장할 폴더를 선택합니다. 파일 이름으로 lab\_01\_03\_soln.sql 을 입력하고 Save 버튼을 누릅니다.

## 연습 2

1. 적합한 식 별자와 부적합한 식별자를 구분합니다.

- |                                         |                          |
|-----------------------------------------|--------------------------|
| a. today                                | 유효                       |
| b. last_name                            | 유효                       |
| c. today's_date                         | 부적합 - ''' 문자는 허용되지 않습니다. |
| d. Number_of_days_in_February_this_year | 부적합 - 너무 길니다.            |
| e. Isleap\$year                         | 유효                       |
| f. #number                              | 부적합 - '#'으로 시작할 수 없습니다.  |
| g. NUMBER#                              | 유효                       |
| h. number1to7                           | 유효                       |

2. 다음 변수 선언 및 초기화가 적합한지 식별합니다.

- |                     |                        |     |
|---------------------|------------------------|-----|
| a. number_of_copies | PLS_INTEGER;           | 유효  |
| b. PRINTER_NAME     | constant VARCHAR2(10); | 부적합 |
| c. deliver_to       | VARCHAR2(10):=Johnson; | 부적합 |
| d. by_when          | DATE:= SYSDATE+1;      | 유효  |

상수 변수는 선언 시에 초기화되어야 하기 때문에 **b**의 선언은 유효하지 않습니다.

문자열 리터럴은 작은 따옴표로 묶어야 하기 때문에 **c**의 선언은 유효하지 않습니다.

3. 다음 익명 블록을 검토하고 올바른 문장을 선택합니다.

```
SET SERVEROUTPUT ON
DECLARE
fname VARCHAR2(20);
lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
DBMS_OUTPUT.PUT_LINE(FNAME || ' ' || lname);
END;
```

- 블록이 성공적으로 실행되고 "fernandez"가 출력됩니다.
- fname 변수가 초기화하지 않고 사용되었기 때문에 오류가 발생합니다.
- 블록이 성공적으로 실행되고 "null fernandez"가 출력됩니다.
- VARCHAR2 유형의 변수를 초기화하는 데 DEFAULT 키워드를 사용할 수 없기 때문에 오류가 발생합니다.
- fname 변수를 선언하지 않았기 때문에 오류가 발생합니다.
- 블록이 성공적으로 실행되고 "fernandez"가 출력됩니다.



4. 익명 블록을 생성합니다. SQL Developer 에서 연습 1 의 2 번 문제에서 생성한 lab\_01\_02\_soln.sql 스크립트를 다음 지침에 따라 로드합니다.  
File > Open 을 선택합니다.  
lab\_01\_02\_soln.sql 파일을 찾아 선택합니다. Open 버튼을 누릅니다. 이제 작업 영역에 .sql 파일의 코드가 나타납니다.

- a. 이 PL/SQL 블록에 선언 섹션을 추가합니다. 선언 섹션에서 다음 변수를 선언합니다.

1. DATE 유형의 변수 today.today를 SYSDATE 로 초기화합니다.

```
DECLARE
today DATE:=SYSDATE;
```

2. today 유형의 변수 tomorrow.%TYPE 속성을 사용하여 이 변수를 선언합니다.

```
tomorrow today%TYPE;
```

- b. 실행 섹션에서 내일 날짜를 계산하는 표현식(today 값에 1 추가)을 사용하여 tomorrow 변수를 초기화합니다. "Hello World"를 출력한 후 today 와 tomorrow 의 값을 출력합니다.

```
BEGIN
tomorrow:=today +1;
DBMS_OUTPUT.PUT_LINE(' Hello World ');
DBMS_OUTPUT.PUT_LINE('TODAY IS : ' || today);
DBMS_OUTPUT.PUT_LINE('TOMORROW IS : ' || tomorrow);
END;
```

- c. 스크립트를 실행하고 lab\_02\_04\_soln.sql 로 저장합니다.  
연습 1 의 2 단계 e 항목에 나오는 지침에 따라 파일을 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello World
TODAY IS : 28-JAN-09
TOMORROW IS : 29-JAN-09
```

5. lab\_02\_04\_soln.sql 스크립트를 편집합니다.

a. 두 개의 바인드 변수를 생성하는 코드를 추가합니다.

NUMBER 유형의 바인드 변수 basic\_percent 및 pf\_percent 를 생성합니다.

```
VARIABLE basic_percent NUMBER
VARIABLE pf_percent NUMBER
```

b. PL/SQL 블록의 실행 섹션에서 basic\_percent 와 pf\_percent 에 각각 값 45 와 12 를 할당합니다.

```
:basic_percent:=45;
:pf_percent:=12;
```

c. "/"로 PL/SQL 블록을 종료하고 PRINT 명령을 사용하여 바인드 변수 값을 표시합니다.

```
/
PRINT basic_percent
PRINT pf_percent
```

또는

```
PRINT
```

d. 스크립트를 실행하고 lab\_02\_05\_soln.sql 로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello World
TODAY IS : 28-JAN-09
TOMORROW IS : 29-JAN-09

basic_percent
--
45

pf_percent
--
12
```

### 연습 3

```
DECLARE
weight NUMBER(3) := 600;
message VARCHAR2(255) := 'Product 10012';
BEGIN
 DECLARE
 weight NUMBER(3) := 1;
 message VARCHAR2(255) := 'Product 11001';
 new_locn VARCHAR2(50) := 'Europe';
 BEGIN
 weight := weight + 1;
 new_locn := 'Western ' || new_locn;
1 →
 END;
weight := weight + 1;
message := message || ' is in stock';
new_locn := 'Western ' || new_locn;
2 →
END;
```

1. 앞의 PL/SQL 블록을 검토하여 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형 및 값을 판별합니다.

- a. 위치 1에서의 weight 값:  
**2**  
**데이터 유형은 NUMBER**
- b. 위치 1에서의 new\_locn 값:  
**Western Europe**  
**데이터 유형은 VARCHAR2**
- c. 위치 2에서의 weight 값:  
**601**  
**데이터 유형은 NUMBER**
- d. 위치 2에서의 message 값:  
**Product 10012 is in stock.**  
**데이터 유형은 VARCHAR2**

- e. 위치 2에서의 new\_locn 값:

**new\_locn** 은 하위 블록 외부에서 볼 수 없기 때문에 잘못된 구문입니다.

```
DECLARE
 customer VARCHAR2(50) := 'Womansport';
 credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
 DECLARE
 customer NUMBER(7) := 201;
 name VARCHAR2(25) := 'Unisports';
 BEGIN
 credit_rating := 'GOOD';
 ...
 END;
 ...
END;
```

2. 앞의 PL/SQL 블록에서 다음 각 경우에 해당하는 값 및 데이터 유형을 판별합니다.

- a. 중첩된 블록의 customer 값:

**201**

**데이터 유형은 NUMBER**

- b. 중첩된 블록의 name 값:

**Unisports**

**데이터 유형은 VARCHAR2**

- c. 중첩된 블록의 credit\_rating 값:

**GOOD**

**데이터 유형은 VARCHAR2**

- d. 기본 블록의 customer 값:

**Womansport**

**데이터 유형은 VARCHAR2**

- e. 기본 블록의 name 값:

**name** 은 기본 블록에서 볼 수 없으며 오류 메시지가 표시됩니다.

- f. 기본 블록의 credit\_rating 값:

**GOOD**

**데이터 유형은 VARCHAR2**

3. lab\_02\_05\_soln.sql 을 실행한 동일한 워크시트에서 편집합니다. 해당 워크시트를 닫은 경우 열고 lab\_02\_05\_soln.sql 을 실행한 다음 편집합니다.

- a. 단일 행 주석 구문을 사용하여 바인드 변수를 생성하는 행을 주석 처리합니다.

```
-- VARIABLE basic_percent NUMBER
-- VARIABLE pf_percent NUMBER
```

- b. 실행 섹션에서 다중 행 주석을 사용하여 바인드 변수에 값을 할당하는 행을 주석 처리합니다.

```
/* :basic_percent:=45;
:pf_percent:=12; */
```

- c. 데이터 유형이 VARCHAR2 이고 크기가 15 인 fname 및 데이터 유형이 NUMBER 이고 크기가 10 인 emp\_sal 이라는 두 변수를 선언합니다.

```
fname VARCHAR2(15);
emp_sal NUMBER(10);
```

- d. 다음 SQL 문을 실행 섹션에 포함시킵니다.

```
SELECT first_name, salary INTO fname, emp_sal
FROM employees WHERE employee_id=110;
```

- e. "Hello World"를 출력하는 행을 "Hello"와 이름을 출력하도록 변경합니다. 필요한 경우 날짜를 표시하고 바인드 변수를 출력하는 행을 주석 처리합니다.

```
DBMS_OUTPUT.PUT_LINE(' Hello ' || fname);
```

- f. 기업 연금(PF)에 대한 사원의 부담금을 계산합니다.  
PF는 기본 급여의 12%이며 기본 급여는 급여의 45%입니다. 계산할 때는 바인드 변수를 사용합니다. 표현식을 하나만 사용하여 PF 를 계산합니다. 사원의 급여 및 PF 부담금을 출력합니다.

```
DBMS_OUTPUT.PUT_LINE('YOUR SALARY IS : ' || emp_sal);
DBMS_OUTPUT.PUT_LINE('YOUR CONTRIBUTION TOWARDS PF:
' || emp_sal*basic_percent/100*pf_percent/100);
```

- g. 스크립트를 실행하고 lab\_03\_03\_soln.sql 로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
Hello John
YOUR SALARY IS : 8200
YOUR CONTRIBUTION TOWARDS PF: 442.8
```

4. lab\_03\_04.sql 스크립트를 실행합니다. 이 스크립트는 employee\_details 라는 테이블을 생성합니다.
- employee 및 employee\_details 테이블에는 동일한 데이터가 있습니다. employee\_details 테이블의 데이터를 갱신합니다. employees 테이블의 데이터는 갱신하거나 변경하지 마십시오.
  - lab\_03\_04b.sql 스크립트를 열어 파일의 코드를 확인합니다. 이 코드는 사용자가 입력한 사원 번호 및 부서 번호를 받아들입니다.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
ACCEPT emp_id PROMPT 'Please enter your employee number';
ACCEPT emp_deptid PROMPT 'Please enter the department number for which
salary revision is being done';

DECLARE
 emp_authorization NUMBER(5);
 emp_id NUMBER(5):=&emp_id;
 emp_deptid NUMBER(6):=&emp_deptid;
 no_such_employee EXCEPTION;
...
```

- 이 스크립트는 "소개" 단원에서 설명한 응용 프로그램을 개발하는 데 기본 구조로 사용됩니다.

## 연습 4

1. departments 테이블에서 최대 부서 ID를 선택하여 max\_deptno 변수에 저장하는 PL/SQL 블록을 생성합니다. 최대 부서 ID를 표시합니다.

a. 선언 섹션에서 NUMBER 유형의 max\_deptno 변수를 선언합니다.

```
SET SERVEROUTPUT ON
DECLARE
 max_deptno NUMBER;
```

b. BEGIN 키워드로 실행 섹션을 시작하고 departments 테이블에서 최대 department\_id를 검색하는 SELECT 문을 포함시킵니다.

```
BEGIN
 SELECT MAX(department_id) INTO max_deptno FROM departments;
```

c. max\_deptno를 표시하고 실행 블록을 종료합니다.

```
DBMS_OUTPUT.PUT_LINE('The maximum department_id is : ' || max_deptno);
END;
```

d. 스크립트를 실행하고 lab\_04\_01\_soln.sql로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 270
```

2. 연습 1에서 생성한 PL/SQL 블록을 departments 테이블에 새 부서를 삽입하도록 수정합니다.

a. lab\_04\_01\_soln.sql 스크립트를 엽니다.  
departments.department\_name 유형의 dept\_name 변수와 NUMBER 유형의 dept\_id 변수를 선언합니다.  
선언 섹션에서 dept\_name에 "Education"을 할당합니다.

```
VARIABLE dept_id NUMBER
...
dept_name departments.department_name%TYPE:= 'Education';
```

b. 앞에서 이미 departments 테이블에서 현재 최대 부서 번호를 검색했습니다.  
이 부서 번호에 10을 더하여 해당 결과를 dept\_id에 할당합니다.

```
dept_id := 10 + max_deptno;
...
```

- c. departments 테이블의 department\_name, department\_id 및 location\_id 열에 데이터를 삽입하기 위해 INSERT 문을 포함시킵니다. department\_name, department\_id 에 dept\_name, dept\_id 의 값을 사용하고 location\_id 에 NULL 을 사용합니다.

```
...
INSERT INTO departments (department_id, department_name, location_id)
VALUES (:dept_id, dept_name, NULL);
```

- d. SQL 속성 SQL%ROWCOUNT 를 사용하여 적용되는 행 수를 표시합니다.

```
DBMS_OUTPUT.PUT_LINE (' SQL%ROWCOUNT gives ' || SQL%ROWCOUNT);
...
```

- e. select 문을 실행하여 새 부서가 삽입되었는지 확인합니다. "/"로 PL/SQL 블록을 종료하고 스크립트에 SELECT 문을 포함시킵니다.

```
...
/
SELECT * FROM departments WHERE department_id=:dept_id;
```

- f. 스크립트를 실행하고 lab\_04\_02\_soln.sql 로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 280
SQL%ROWCOUNT gives 1
```

|   | DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|---------------|-----------------|------------|-------------|
| 1 | 280           | Education       | (null)     | (null)      |

3. 연습 2 에서 location\_id 를 NULL 로 설정했습니다. 새 부서의 location\_id 를 3000 으로 갱신하는 PL/SQL 블록을 생성합니다. 변수 dept\_id 값을 사용하여 행을 갱신합니다.

- a. BEGIN 키워드로 실행 블록을 시작합니다. 새 부서(dept id = 280)의 location\_id 를 3000 으로 설정하는 UPDATE 문을 포함시킵니다.

```
BEGIN
 UPDATE departments SET location_id=3000 WHERE
 department_id=280;
```

- b. END 키워드로 실행 블록을 종료합니다. "/"로 PL/SQL 블록을 종료하고 갱신한 부서가 표시되도록 SELECT 문을 포함시킵니다.

```
END;
/
SELECT * FROM departments WHERE department_id=:dept_id;
```



- c. 추가한 부서를 삭제하도록 DELETE 문을 포함시킵니다.

```
DELETE FROM departments WHERE department_id=280;
```

- d. 스크립트를 실행하고 lab\_04\_03\_soln.sql 로 저장합니다. 예제의 출력 결과는 다음과 같습니다.

| anonymous block completed |                 |            |             |
|---------------------------|-----------------|------------|-------------|
| DEPARTMENT_ID             | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
| -----                     |                 |            |             |
| 280                       | Education       |            | 3000        |
| 1 rows selected           |                 |            |             |
| 1 rows deleted            |                 |            |             |

4. lab\_03\_05b.sql 스크립트를 엽니다.

- a. 코드에 중첩 블록이 있는지 확인합니다. 외부 블록의 선언 섹션을 볼 수 있습니다. "INCLUDE EXECUTABLE SECTION OF OUTER BLOCK HERE"라는 주석을 찾아 실행 섹션을 시작합니다.

```
BEGIN
```

- b. HR(Human Resources) 부서에서 근무하는 사원의 employee\_id 를 검색하는 단일 SELECT 문을 포함시킵니다. INTO 절을 사용하여 검색된 값을 emp\_authorization 변수에 저장합니다.

```
SELECT employee_id into emp_authorization FROM
employee_details WHERE department_id=(SELECT department_id
FROM departments WHERE department_name='Human Resources');
```

- c. 스크립트를 lab\_04\_04\_soln.sql 로 저장합니다.

## 연습 5

1. lab\_05\_01.sql 파일에서 messages 테이블을 생성하는 명령을 실행합니다.  
messages 테이블에 숫자를 삽입하는 PL/SQL 블록을 작성합니다.

- a. 1에서 10까지의 숫자를 삽입합니다(6, 8 제외).
- b. 블록 종료 전에 커밋합니다.

```
BEGIN
FOR i in 1..10 LOOP
 IF i = 6 or i = 8 THEN
 null;
 ELSE
 INSERT INTO messages(results)
 VALUES (i);
 END IF;
END LOOP;
COMMIT;
END;
/
```

- c. SELECT 문을 실행하여 PL/SQL 블록이 작동하는지 확인합니다.

```
SELECT * FROM messages;
```

출력 결과는 다음과 같아야 합니다.

|   | RESULTS |
|---|---------|
| 1 | 1       |
| 2 | 2       |
| 3 | 3       |
| 4 | 4       |
| 5 | 5       |
| 6 | 7       |
| 7 | 9       |
| 8 | 10      |

2. lab\_05\_02.sql 스크립트를 실행합니다. 이 스크립트는 employees 테이블의 복제본인 emp 테이블을 생성합니다. 이 스크립트는 데이터 유형은 VARCHAR2 이고 크기는 50인 새 열 stars 를 추가하도록 emp 테이블을 변경합니다. 사원의 급여에 대해 \$1000 단위마다 stars 열에 별표를 삽입하는 PL/SQL 블록을 생성합니다. 스크립트를 lab\_05\_02\_soln.sql 로 저장합니다.

- a. DEFINE 명령을 사용하여 empno 라는 변수를 정의하고 176으로 초기화합니다.

```
SET VERIFY OFF
DEFINE empno = 176
```

- b. 블록의 선언 섹션을 시작하고 치환 변수를 통해 empno 값을 PL/SQL 블록으로 전달합니다. emp.stars 유형의 asterisk 변수를 선언하고 NULL 로 초기화합니다. emp.salary 유형의 sal 변수를 생성합니다.

```
DECLARE
 empno emp.employee_id%TYPE := TO_NUMBER(&empno);
 asterisk emp.stars%TYPE := NULL;
 sal emp.salary%TYPE;
```

- c. 실행 섹션에서 사원의 급여에 대해 \$1000 단위마다 문자열에 별표(\*)를 추가하는 논리를 작성합니다. 예를 들어, 사원의 급여가 \$8000 이면 별표 문자열에는 8 개의 별표가 있어야 합니다. 급여가 \$12500 이면 별표 문자열에는 13 개의 별표가 있어야 합니다.

```
BEGIN
 SELECT NVL(ROUND(salary/1000), 0) INTO sal
 FROM emp WHERE employee_id = empno;

 FOR i IN 1..sal
 LOOP
 asterisk := asterisk || '*';
 END LOOP;
```

- d. 해당 사원의 stars 열을 별표 문자열로 갱신합니다. 블록 종료 전에 커밋합니다.

```
UPDATE emp SET stars = asterisk
 WHERE employee_id = empno;
COMMIT;
END;
```

- e. emp 테이블의 행을 출력하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다.

```
SELECT employee_id,salary, stars
FROM emp WHERE employee_id=&empno;
```

- f. 스크립트를 실행하고 lab\_05\_02\_soln.sql 로 저장합니다. 출력 결과는 다음과 같습니다.

|   | EMPLOYEE_ID | SALARY | STARS |
|---|-------------|--------|-------|
| 1 | 176         | 8600   | ***** |

3. 연습 4 의 4 번 문제에서 작성한 lab\_04\_04\_soln.sql 스크립트를 엽니다.

- a. "INCLUDE SIMPLE IF STATEMENT HERE"라는 주석을 찾은 다음 간단한 IF 문을 포함시켜 emp\_id 와 emp\_authorization 의 값이 동일한지 확인합니다.

```
IF (emp_id=emp_authorization) THEN
```

- b. 스크립트를 lab\_05\_03\_soln.sql 로 저장합니다.

## 연습 6

1. 제공된 국가에 대한 정보를 출력하는 PL/SQL 블록을 작성합니다.
  - a. `countries` 테이블의 구조에 맞게 PL/SQL 레코드를 선언합니다.
  - b. `DEFINE` 명령을 사용하여 `countryid` 변수를 정의합니다.  
`countryid` 에 CA 를 할당합니다. 이 값을 치환 변수를 사용하여 PL/SQL 블록에 전달합니다.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE countryid = CA
```

- c. 선언 섹션에서 `%ROWTYPE` 속성을 사용하여 `countries` 유형의 `country_record` 변수를 선언합니다.

```
DECLARE
country_record countries%ROWTYPE;
```

- d. 실행 섹션에서 `countryid` 를 사용하여 `countries` 테이블의 모든 정보를 가져옵니다. 선택한 국가의 정보를 표시합니다. 예제의 출력 결과는 다음과 같습니다.

```
BEGIN
SELECT *
 INTO country_record
 FROM countries
 WHERE country_id = UPPER('&countryid');

DBMS_OUTPUT.PUT_LINE ('Country Id: ' || country_record.country_id ||
 ' Country Name: ' || country_record.country_name
 || ' Region: ' || country_record.region_id);

END;
```

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

- e. ID 가 DE, UK, US 인 국가에 대 PL/SQL 블록을 실행하여 테스트할 수도 있습니다.

2. INDEX BY 테이블과 통합하여 departments 테이블에서 일부 부서 이름을 검색하여 각 부서 이름을 화면에 출력하는 PL/SQL 블록을 생성합니다. 스크립트를 lab\_06\_02\_soln.sql 로 저장합니다.

- a. departments.department\_name 유형의 INDEX BY 테이블 dept\_table\_type 을 선언합니다. dept\_table\_type 유형의 my\_dept\_table 변수를 선언하여 부서 이름을 임시로 저장합니다.

```
SET SERVEROUTPUT ON
DECLARE
 TYPE dept_table_type is table of departments.department_name%TYPE
 INDEX BY PLS_INTEGER;
 my_dept_table dept_table_type;
```

- b. NUMBER 유형의 2 개의 변수 loop\_count 및 deptno 를 선언합니다. loop\_count 에 10 을 할당하고 deptno 에 0 을 할당합니다.

```
loop_count NUMBER (2) := 10;
deptno NUMBER (4) := 0;
```

- c. 루프를 사용하여 10 개의 부서 이름을 검색하고 INDEX BY 테이블에 이름을 저장합니다. department\_id 를 10 에서 시작합니다. 루프가 반복할 때마다 deptno 를 10 씩 증가시킵니다. 다음 테이블 department\_name 을 검색하고 INDEX BY 테이블에 저장할 department\_id 를 보여줍니다.

| DEPARTMENT_ID | DEPARTMENT_NAME  |
|---------------|------------------|
| 10            | Administration   |
| 20            | Marketing        |
| 30            | Purchasing       |
| 40            | Human Resources  |
| 50            | Shipping         |
| 60            | IT               |
| 70            | Public Relations |
| 80            | Sales            |
| 90            | Executive        |
| 100           | Finance          |

```

BEGIN

FOR i IN 1..loop_count
LOOP
 deptno:=deptno+10;
 SELECT department_name
 INTO my_dept_table(i)
 FROM departments
 WHERE department_id = deptno;
END LOOP;

```

- d. 다른 루프를 사용하여 INDEX BY 테이블에서 부서 이름을 검색한 다음 해당 정보를 출력합니다.

```

FOR i IN 1..loop_count
LOOP
 DBMS_OUTPUT.PUT_LINE (my_dept_table(i));
END LOOP;
END;

```

- e. 스크립트를 실행하고 lab\_06\_02\_soln.sql 로 저장합니다. 출력 결과는 다음과 같습니다.

```

anonymous block completed
Administration
Marketing
Purchasing
Human Resources
Shipping
IT
Public Relations
Sales
Executive
Finance

```

3. departments 테이블에서 각 부서 정보를 모두 검색하여 출력하도록 2 번 문제에서 생성한 블록을 수정합니다. INDEX BY 레코드 테이블을 사용합니다.
- lab\_06\_02\_soln.sql 스크립트를 엽니다.
  - INDEX BY 테이블을 departments.department\_name 유형으로 선언했습니다. 모든 부서의 번호, 이름, manager\_id 및 위치를 임시로 저장하도록 INDEX BY 테이블의 선언을 수정합니다. %ROWTYPE 속성을 사용합니다.

```

SET SERVEROUTPUT ON
DECLARE
 TYPE dept_table_type is table of departments%ROWTYPE
 INDEX BY PLS_INTEGER;
 my_dept_table dept_table_type;
 loop_count NUMBER (2):=10;
 deptno NUMBER (4):=0;

```

- c. departments 테이블의 현재 모든 부서 정보를 검색하도록 select 문을 수정하고 INDEX BY 테이블에 저장합니다.

```
BEGIN
 FOR i IN 1..loop_count
 LOOP
 deptno := deptno + 10;
 SELECT *
 INTO my_dept_table(i)
 FROM departments
 WHERE department_id = deptno;
 END LOOP;
```

- d. 다른 루프를 사용하여 INDEX BY 테이블에서 부서 정보를 검색한 다음 해당 정보를 출력합니다. 예제의 출력 결과는 다음과 같습니다.

```
FOR i IN 1..loop_count
LOOP
 DBMS_OUTPUT.PUT_LINE ('Department Number: ' ||
my_dept_table(i).department_id
 || ' Department Name: ' || my_dept_table(i).department_name
 || ' Manager Id: ' || my_dept_table(i).manager_id
 || ' Location Id: ' || my_dept_table(i).location_id);
END LOOP;
END;
```

```
anonymous block completed
Department Number: 10 Department Name: Administration Manager Id: 200 Location Id: 1700
Department Number: 20 Department Name: Marketing Manager Id: 201 Location Id: 1800
Department Number: 30 Department Name: Purchasing Manager Id: 114 Location Id: 1700
Department Number: 40 Department Name: Human Resources Manager Id: 203 Location Id: 2400
Department Number: 50 Department Name: Shipping Manager Id: 121 Location Id: 1500
Department Number: 60 Department Name: IT Manager Id: 103 Location Id: 1400
Department Number: 70 Department Name: Public Relations Manager Id: 204 Location Id: 2700
Department Number: 80 Department Name: Sales Manager Id: 145 Location Id: 2500
Department Number: 90 Department Name: Executive Manager Id: 100 Location Id: 1700
Department Number: 100 Department Name: Finance Manager Id: 108 Location Id: 1700
```

4. lab\_05\_03\_soln.sql 스크립트를 로드합니다.

- a. "DECLARE AN INDEX BY TABLE OF TYPE VARCHAR2(50). CALL IT ename\_table\_type"이라는 주석을 찾아 선언을 포함시킵니다.

```
TYPE ename_table_type IS TABLE OF
 VARCHAR2(50) INDEX BY PLS_INTEGER;
```

- b. "DECLARE A VARIABLE ename\_table OF TYPE ename\_table\_type"이라는 주석을 찾아 선언을 포함시킵니다.

```
ename_table ename_table_type;
```

- c. 스크립트를 lab\_06\_04\_soln.sql 로 저장합니다.



## 연습 7

1. 상위  $n$  번째까지의 고액 급여를 판별하는 PL/SQL 블록을 생성합니다.
  - a. 사원의 급여를 저장하기 위해 lab\_07\_01.sql 스크립트를 실행하여 새 테이블 top\_salaries 를 생성합니다.
  - b. 숫자  $n$  은 유저가 입력하는 값입니다. 여기서  $n$  은 employees 테이블의 급여 중 상위  $n$  번째를 나타냅니다. 예를 들어, 상위 다섯 개의 급여를 보려면 5 를 입력합니다.  
주:  $n$  에 대한 값을 제공하려면 DEFINE 명령을 사용하여 변수 p\_num 을 정의합니다. 치환 변수를 통해 PL/SQL 블록으로 값이 전달됩니다.

```
DELETE FROM top_salaries;
DEFINE p_num = 5
```

- c. 선언 섹션에서 치환 변수 p\_num 을 수용할 NUMBER 유형의 num 변수와 employees.salary 유형의 sal 변수를 선언합니다. 사원의 급여를 내림차순으로 읽어 들이는 emp\_cursor 커서를 선언합니다. 급여는 중복될 수 없습니다.

```
DECLARE
 num NUMBER(3) := &p_num;
 sal employees.salary%TYPE;
 CURSOR
 emp_cursor IS
 SELECT distinct salary
 FROM employees
 ORDER BY salary DESC;
```

- d. 실행 섹션에서 루프를 열고 상위  $n$  번째까지의 급여를 패치(fetch)한 다음 top\_salaries 테이블에 삽입합니다. 간단한 루프를 사용하여 데이터를 산출할 수 있습니다. 또한 종료 조건에 %ROWCOUNT 및 %FOUND 속성을 사용합니다.

```
BEGIN
 OPEN emp_cursor;
 FETCH emp_cursor INTO sal;
 WHILE emp_cursor%ROWCOUNT <= num AND emp_cursor%FOUND LOOP
 INSERT INTO top_salaries (salary)
 VALUES (sal);
 FETCH emp_cursor INTO sal;
 END LOOP;
 CLOSE emp_cursor;
END;
```

- e. `top_salaries` 테이블에 삽입한 후 `SELECT` 문을 사용하여 행을 출력합니다.  
결과로 `employees` 테이블의 상위 5 개의 급여가 출력됩니다.

```
/
SELECT * FROM top_salaries;
```

```
0 rows deleted
anonymous block completed
SALARY

24000
17000
14000
13500
13000

5 rows selected
```

- f.  $n$  이 0 이거나  $n$  이 `employees` 테이블의 사원 수보다 더 큰 경우와 같이 다양한 경우를 테스트해 봅니다. 각 테스트 전후에는 `top_salaries` 테이블을 비웁니다.

2. 다음을 수행하는 PL/SQL 블록을 생성합니다.

- a. `DEFINE` 명령을 사용하여 부서 ID 를 제공하는 `p_deptno` 변수를 정의합니다.

```
SET SERVEROUTPUT ON
SET VERIFY OFF
SET ECHO OFF
DEFINE p_deptno = 10
```

- b. 선언 섹션에서 `NUMBER` 유형의 `deptno` 변수를 선언하고 `p_deptno` 의 값을 할당합니다.

```
DECLARE
deptno NUMBER := &p_deptno;
```

- c. `deptno` 에 지정된 부서에 근무하는 사원의 `last_name`, `salary` 및 `manager_id` 를 검색하는 `emp_cursor` 커서를 선언합니다.

```
CURSOR emp_cursor IS
SELECT last_name, salary,manager_id
FROM employees
WHERE department_id = deptno;
```

- d. 실행 섹션에서 커서 FOR 루프를 사용하여 검색된 데이터에 대해 작업할 수 있습니다. 사원의 급여가 5000 미만이고 관리자 ID 가 101 또는 124 인 경우 <<last\_name>> Due for a raise 메시지를 표시합니다. 그렇지 않은 경우 <<last\_name>> Not due for a raise 메시지를 표시합니다.

```
BEGIN
 FOR emp_record IN emp_cursor
 LOOP
 IF emp_record.salary < 5000 AND (emp_record.manager_id=101 OR
emp_record.manager_id=124) THEN
 DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Due for a
raise');
 ELSE
 DBMS_OUTPUT.PUT_LINE (emp_record.last_name || ' Not Due for a
raise');
 END IF;
 END LOOP;
END;
```

- e. 다음 경우에 대해 PL/SQL 블록을 테스트합니다.

| Department ID | Message                                                                                                                                                                                        |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10            | Whalen Due for a raise                                                                                                                                                                         |
| 20            | Hartstein Not Due for a raise<br>Fay Not Due for a raise                                                                                                                                       |
| 50            | Weiss Not Due for a raise<br>Fripp Not Due for a raise<br>Kaufling Not Due for a raise<br>Vollman Not Due for a raise<br>Mourgas Not Due for a raise<br>. . .<br>. . .<br>Rajs Due for a raise |
| 80            | Russel Not Due for a raise<br>Partners Not Due for a raise<br>Errazuriz Not Due for a raise<br>Cambrault Not Due for a raise<br>. . .<br>. . .                                                 |

3. 파라미터가 포함된 커서를 선언하고 사용하는 PL/SQL 블록을 작성합니다.
- 루프에서 커서를 사용하여 `departments` 테이블에서 `department_id` 가 100 보다 작은 부서의 부서 번호 및 부서 이름을 읽어 들입니다. 부서 번호를 다른 커서에 파라미터로 전달하여 `employees` 테이블에서 해당 부서에 근무하고 `employee_id` 가 120 보다 작은 사원의 성, 직위, 채용 날짜 및 급여 등의 상세 정보를 읽어 들입니다.
- 선언 섹션에서 `dept_cursor` 커서를 선언하여 `department_id` 가 100 보다 작은 부서의 `department_id` `department_name` 을 검색합니다. `department_id` 에 따라 정렬합니다.

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR dept_cursor IS
 SELECT department_id, department_name
 FROM departments
 WHERE department_id < 100
 ORDER BY department_id;
```

- 부서 번호를 파라미터로 사용하여 해당 부서에 근무 `employee_id` 가 120 보다 작은 사원의 `last_name`, `job_id`, `hire_date` 및 `salary` 를 검색하는 다른 커서 `emp_cursor` 를 선언합니다.

```
CURSOR emp_cursor(v_deptno NUMBER) IS
 SELECT last_name, job_id, hire_date, salary
 FROM employees
 WHERE department_id = v_deptno
 AND employee_id < 120;
```

- 각 커서에서 검색된 값을 보유하는 변수를 선언합니다. 변수 선언 시 `%TYPE` 속성을 사용합니다.

```
current_deptno departments.department_id%TYPE;
current_dname departments.department_name%TYPE;
ename employees.last_name%TYPE;
job employees.job_id%TYPE;
hiredate employees.hire_date%TYPE;
sal employees.salary%TYPE;
```

- `dept_cursor` 를 열고 간단한 루프를 사용하여 값을 선언된 변수에 패치(fetch)합니다. 부서 번호 및 부서 이름을 출력합니다.

```
BEGIN
 OPEN dept_cursor;
 LOOP
 FETCH dept_cursor INTO current_deptno, current_dname;
 EXIT WHEN dept_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE ('Department Number : ' ||
current_deptno || ' Department Name : ' || current_dname);
```

- e. 각 부서에 대해 현재 부서 번호를 파라미터로 전달하여 emp\_cursor 를 엽니다. 다른 루프를 시작하여 emp\_cursor 의 값을 변수에 패치(fetch)하고 employees 테이블에서 검색된 모든 상세 정보를 출력합니다.

**주:** 각 부서의 상세 정보를 표시한 다음 한 행을 출력할 수 있습니다. 종료 조건에 적합한 속성을 사용하십시오. 또한 커서를 열기 전에 이미 열려 있는지 확인하십시오.

```
IF emp_cursor%ISOPEN THEN
 CLOSE emp_cursor;
END IF;
OPEN emp_cursor (current_deptno);
LOOP
 FETCH emp_cursor INTO ename,job,hiredate,sal;
 EXIT WHEN emp_cursor%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE (ename || ' ' || job || ' ' ||
hiredate || ' ' || sal);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE('-----');
 CLOSE emp_cursor;
```

- f. 모든 루프 및 커서를 닫고 실행 섹션을 종료합니다. 스크립트를 실행합니다.

```
END LOOP;
 CLOSE dept_cursor;
END;
```

```

anonymous block completed
Department Number : 10 Department Name : Administration

Department Number : 20 Department Name : Marketing

Department Number : 30 Department Name : Purchasing
Raphaely PU_MAN 07-DEC-94 11000
Khoo PU_CLERK 18-MAY-95 3100
Baida PU_CLERK 24-DEC-97 2900
Tobias PU_CLERK 24-JUL-97 2800
Himuro PU_CLERK 15-NOV-98 2600
Colmenares PU_CLERK 10-AUG-99 2500

Department Number : 40 Department Name : Human Resources

Department Number : 50 Department Name : Shipping

Department Number : 60 Department Name : IT
Hunold IT_PROG 03-JAN-90 9000
Ernst IT_PROG 21-MAY-91 6000
Austin IT_PROG 25-JUN-97 4800
Pataballa IT_PROG 05-FEB-98 4800
Lorentz IT_PROG 07-FEB-99 4200

Department Number : 70 Department Name : Public Relations

Department Number : 80 Department Name : Sales

Department Number : 90 Department Name : Executive
King AD_PRES 17-JUN-87 24000
Kochhar AD_VP 21-SEP-89 17000
De Haan AD_VP 13-JAN-93 17000

```

4. lab\_06\_04\_soln.sql 스크립트를 로드합니다.
- "DECLARE A CURSOR CALLED emp\_records TO HOLD salary, first\_name, and last\_name of employees" 주석을 찾아 선언을 포함시킵니다. 유저가 지정한 부서(치환 변수 emp\_deptid)에서 사원의 salary, first\_name 및 last\_name 을 검색하는 커서를 생성합니다. FOR UPDATE 절을 사용합니다.

```

CURSOR emp_records IS SELECT salary,first_name,last_name
FROM employee_details WHERE department_id=emp_deptid
FOR UPDATE;

```

- "INCLUDE EXECUTABLE SECTION OF INNER BLOCK HERE" 주석을 찾아 실행 블록을 시작합니다.

```

BEGIN

```

- c. department\_id 가 20, 60, 80, 100 및 110 인 부서에 근무하는 직원만 이번 분기에 급여 인상 대상자입니다. 사용자가 이 부서 ID 중 하나를 입력했는지 확인합니다. 값이 일치하지 않으면 "SORRY, NO SALARY REVISIONS FOR EMPLOYEES IN THIS DEPARTMENT" 메시지가 출력됩니다. 값이 일치하면 emp\_records 커서를 엽니다.

```
IF (emp_deptid NOT IN (20,60,80,100,110)) THEN
 DBMS_OUTPUT.PUT_LINE ('SORRY, NO SALARY REVISIONS FOR
 EMPLOYEES IN THIS DEPARTMENT');
ELSE
 OPEN emp_records;
```

- d. 간단한 루프를 시작하여 값을 emp\_sal, emp\_fname 및 emp\_lname 에 패치(fetch)합니다. 종료 조건에 %NOTFOUND 를 사용합니다.

```
LOOP
 FETCH emp_records INTO emp_sal,emp_fname,emp_lname;
 EXIT WHEN emp_records%NOTFOUND;
```

- e. CASE 식을 포함시킵니다. 다음 테이블을 참조하여 CASE 식의 WHEN 절의 조건을 지정합니다.

**주:** CASE 식에서 이미 선언한 c\_range1, c\_hike1 과 같은 상수를 사용하십시오.

| salary         | Hike percentage |
|----------------|-----------------|
| < 6500         | 20              |
| > 6500 < 9500  | 15              |
| > 9500 < 12000 | 8               |
| > 12000        | 3               |

예를 들어, 사원의 급여가 6500 보다 적으면 급여를 20%까지 인상합니다. 모든 WHEN 절에서 사원의 first\_name 과 last\_name 을 연결하여 INDEX BY 테이블에 저장합니다. 바로 다음 위치에 문자열을 저장할 수 있도록 변수 i 의 값을 증가시킵니다. WHERE CURRENT OF 절이 있는 UPDATE 문을 포함시킵니다.

```
CASE
 WHEN emp_sal < c_range1 THEN
 ename_table(i) := emp_fname || ' ' || emp_lname;
 i := i + 1;
 UPDATE employee_details SET salary = emp_sal + (emp_sal * c_hike1)
 WHERE CURRENT OF emp_records;
 WHEN emp_sal < c_range2 THEN
 ename_table(i) := emp_fname || ' ' || emp_lname;
 i := i + 1;
 UPDATE employee_details SET salary = emp_sal + (emp_sal * c_hike2)
 WHERE CURRENT OF emp_records;
 WHEN (emp_sal < c_range3) THEN
 ename_table(i) := emp_fname || ' ' || emp_lname;
```

```

 i:=i+1;
 UPDATE employee_details SET salary=emp_sal+(emp_sal*c_hike3)
 WHERE CURRENT OF emp_records;
 ELSE
 ename_table(i):=emp_fname||' '||emp_lname;
 i:=i+1;
 UPDATE employee_details SET salary=emp_sal+(emp_sal*c_hike4)
 WHERE CURRENT OF emp_records;
 END CASE;

```

- f. 루프를 닫습니다. %ROWCOUNT 속성을 사용하여 수정된 레코드 수를 출력합니다. 커서를 닫습니다.

```

END LOOP;

DBMS_OUTPUT.PUT_LINE ('NUMBER OF RECORDS MODIFIED :
'||emp_records%ROWCOUNT);
CLOSE emp_records;

```

- g. 급여가 조정된 모든 사원의 이름을 출력하는 간단한 루프를 포함시킵니다.  
 주: INDEX BY 테이블에 이러한 사원의 이름이 있습니다.  
 "CLOSE THE INNER BLOCK"이라는 주석을 찾아 END IF 문과 END 문을 포함시킵니다.

```

DBMS_OUTPUT.PUT_LINE ('The following employees' salaries are
updated');
FOR i IN ename_table.FIRST..ename_table.LAST
 LOOP
 DBMS_OUTPUT.PUT_LINE(ename_table(i));
 END LOOP;
END IF;
END;

```

- h. 스크립트를 lab\_07\_04\_soln.sql 로 저장합니다.



## 연습 8

1. 이 연습의 목적은 미리 정의된 예외에 대한 사용법을 제공하는 것입니다. 제공된 급여 값을 가진 사원의 이름을 선택하는 PL/SQL 블록을 작성합니다.

- a. `messages` 테이블의 모든 레코드를 삭제하십시오. `DEFINE` 명령을 사용하여 `sal` 변수를 정의하고 6000 으로 초기화합니다.

```
DELETE FROM MESSAGES;
SET VERIFY OFF
DEFINE sal = 6000
```

- b. 선언 섹션에서 `employees.last_name` 유형의 `ename` 변수와 `employees.salary` 유형의 `emp_sal` 변수를 선언합니다. 치환 변수의 값을 `emp_sal` 에 전달합니다.

```
DECLARE
 ename employees.last_name%TYPE;
 emp_sal employees.salary%TYPE := &sal;
```

- c. 실행 섹션에서 급여가 `emp_sal` 의 값과 동일한 사원의 성을 검색합니다.  
**주:** 명시적 커서를 사용하지 마십시오.  
입력된 급여가 행을 하나만 반환하는 경우 `messages` 테이블에 사원 이름 및 급여를 삽입합니다.

```
BEGIN
 SELECT last_name
 INTO ename
 FROM employees
 WHERE salary = emp_sal;
 INSERT INTO messages (results)
 VALUES (ename || ' - ' || emp_sal);
```

- d. 입력된 급여가 행을 반환하지 않으면 적합한 예외 처리기로 예외를 처리하고 `messages` 테이블에 "No employee with a salary of <salary>" 메시지를 삽입합니다.

```
EXCEPTION
 WHEN no_data_found THEN
 INSERT INTO messages (results)
 VALUES ('No employee with a salary of ' || TO_CHAR(emp_sal));
```

- e. 입력된 급여가 둘 이상의 행을 반환하면 적합한 예외 처리기로 예외를 처리하고 `messages` 테이블에 "More than one employee with a salary of <salary>" 메시지를 삽입합니다.

```
 WHEN too_many_rows THEN
 INSERT INTO messages (results)
 VALUES ('More than one employee with a salary of ' ||
 TO_CHAR(emp_sal));
```

- f. 기타 예외의 경우 적합한 예외 처리기로 처리하고 messages 테이블에 "Some other error occurred" 메시지를 삽입합니다.

```
WHEN others THEN
 INSERT INTO messages (results)
 VALUES ('Some other error occurred.');
```

END;

- g. messages 테이블의 행을 표시하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다. 예제의 출력 결과는 다음과 같습니다.

```
/
SELECT * FROM messages;
```

| RESULTS                                        |
|------------------------------------------------|
| 1 More than one employee with a salary of 6000 |

2. 이 예제의 목적은 표준 Oracle Server 오류를 사용하여 예외를 선언하는 방법을 보여주는 것입니다. Oracle 서버 오류 ORA-02292(무결성 제약 조건 위반 - 하위 레코드 발견)를 사용합니다.
- a. 선언 섹션에서 예외 childrecord\_exists 를 선언합니다. 선언된 예외를 표준 Oracle 서버 오류 -02292 와 연결합니다.

```
SET SERVEROUTPUT ON
DECLARE
 childrecord_exists EXCEPTION;
 PRAGMA EXCEPTION_INIT(childrecord_exists, -02292);
```

- b. 실행 섹션에서 "Deleting department 40...."을 출력합니다. department\_id 가 40 인 부서를 삭제하는 DELETE 문을 포함시킵니다.

```
BEGIN
 DBMS_OUTPUT.PUT_LINE(' Deleting department 40.....');
 delete from departments where department_id=40;
```

- c. childrecord\_exists 예외를 처리하고 적절한 메시지를 출력하는 예외 섹션을 포함시킵니다. 예제의 출력 결과는 다음과 같습니다.

```
EXCEPTION
 WHEN childrecord_exists THEN
 DBMS_OUTPUT.PUT_LINE(' Cannot delete this department. There are
employees in this department (child records exist.)');
END;
```

```
anonymous block completed
Deleting department 40.....
Cannot delete this department. There are employees in this department (child records exist.)
```

3. lab\_07\_04\_soln.sql 스크립트를 로드합니다.

- a. 외부 블록의 선언 섹션을 살펴봅니다. no\_such\_employee 예외가 선언된 것을 확인할 수 있습니다.
- b. "RAISE EXCEPTION HERE"라는 주석을 찾습니다. emp\_id 값이 100 과 206 사이에 없으면 no\_such\_employee 예외를 발생시킵니다.

```
IF (emp_id NOT BETWEEN 100 AND 206) THEN
 RAISE no_such_employee;
END IF;
```

- c. "INCLUDE EXCEPTION SECTION FOR OUTER BLOCK"이라는 주석을 찾아 no\_such\_employee 및 too\_many\_rows 예외를 처리합니다. 예외가 발생하면 해당 메시지를 표시합니다. employees 테이블에 HR 부서에서 근무하는 사원이 한 명 뿐이므로 그에 따라 코드가 작성됩니다. too\_many\_rows 예외가 처리되면 select 문을 실행했을 때 HR 부서에서 근무하는 사원이 두 명 이상 검색된 것입니다.

```
EXCEPTION
 WHEN no_such_employee THEN
 DBMS_OUTPUT.PUT_LINE ('NO EMPLOYEE EXISTS WITH THE
 GIVEN EMPLOYEE NUMBER: PLEASE CHECK');

 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.PUT_LINE (' THERE IS MORE THAN ONE
 EMPLOYEE IN THE HR DEPARTMENT. ');
```

- d. 외부 블록을 닫습니다.

```
END;
```

- e. 스크립트를 lab\_08\_03\_soln.sql 로 저장합니다.
- f. 스크립트를 실행합니다. 사원 번호 및 부서 번호를 입력하고 출력을 확인합니다. 다른 값을 입력해 보고 다른 조건을 확인해 봅니다. 사원 ID 가 203 이고 부서 ID 가 100 인 예제 출력은 다음과 같습니다.

```
anonymous block completed
NUMBER OF RECORDS MODIFIED :6
The following employees' salaries are updated
Nancy Greenberg
Daniel Faviet
John Chen
Ismael Sciarra
Jose Manuel Urman
Luis Popp
```

## 연습 9

1. SQL Developer 에서 연습 2 의 질문 4 에서 작성한 lab\_02\_04\_soln.sql 스크립트를 엽니다.

- a. 익명 블록을 greet 라는 프로시저로 변환하도록 스크립트를 수정합니다.

```
CREATE PROCEDURE greet IS
 today DATE:=SYSDATE;
 tomorrow today%TYPE;
 ...
```

- b. 스크립트를 실행하여 프로시저를 작성합니다.
- c. 이 스크립트를 lab\_09\_01\_soln.sql 로 저장합니다.
- d. Clear 버튼을 눌러 작업 영역을 지웁니다.
- e. greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 예제의 출력 결과는 다음과 같습니다.

```
BEGIN
 greet;
END;
```

```
anonymous block completed
Hello World
TODAY IS : 05-FEB-09
TOMORROW IS : 06-FEB-09
```

2. lab\_09\_01\_soln.sql 스크립트를 엽니다.

- a. 다음 명령을 실행하여 greet 프로시저를 삭제합니다.

```
DROP PROCEDURE greet
```

- b. 이 프로시저를 VARCHAR2 유형의 인수를 받아들이도록 수정합니다. name 인수를 호출합니다.

```
CREATE PROCEDURE greet(name VARCHAR2) IS
 today DATE:=SYSDATE;
 tomorrow today%TYPE;
```

- c. Hello World 대신 Hello <name>을 출력합니다.

```
BEGIN
 tomorrow:=today +1;
 DBMS_OUTPUT.PUT_LINE(' Hello ' || name);
```

- d. 스크립트를 lab\_09\_02\_soln.sql 로 저장합니다.
- e. 스크립트를 실행하여 프로시저를 작성합니다.
- f. 파라미터가 포함된 greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 예제의 출력 결과는 다음과 같습니다.

```
BEGIN
 greet('Neema');
END;
```

```
anonymous block completed
Hello Neema
TODAY IS : 05-FEB-09
TOMORROW IS : 06-FEB-09
```

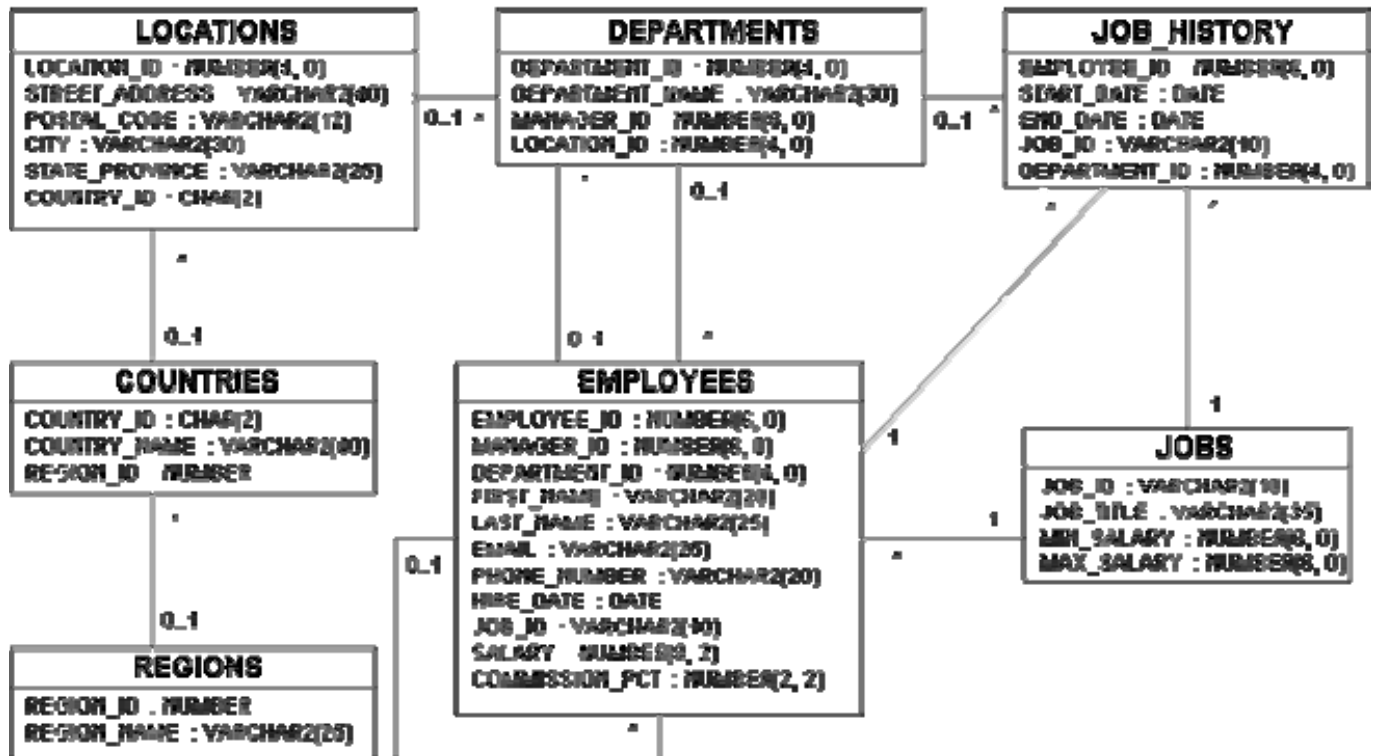
---

## **B**

### **테이블 설명 및 데이터**




---

## 엔티티 관계 도표



## 스키마의 테이블

```
SELECT * FROM tab;
```

|   |  TNAME |  TABTYPE |  CLUSTERID |
|---|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| 1 | REGIONS                                                                                 | TABLE                                                                                     | (null)                                                                                      |
| 2 | COUNTRIES                                                                               | TABLE                                                                                     | (null)                                                                                      |
| 3 | LOCATIONS                                                                               | TABLE                                                                                     | (null)                                                                                      |
| 4 | DEPARTMENTS                                                                             | TABLE                                                                                     | (null)                                                                                      |
| 5 | JOBS                                                                                    | TABLE                                                                                     | (null)                                                                                      |
| 6 | EMPLOYEES                                                                               | TABLE                                                                                     | (null)                                                                                      |
| 7 | JOB_HISTORY                                                                             | TABLE                                                                                     | (null)                                                                                      |
| 8 | EMP_DETAILS_VIEW                                                                        | VIEW                                                                                      | (null)                                                                                      |





## regions 테이블

```
DESCRIBE regions
```

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| -----           | -----    | -----        |
| REGION_ID       | NOT NULL | NUMBER       |
| REGION_NAME     |          | VARCHAR2(25) |
| 2 rows selected |          |              |

```
SELECT * FROM regions;
```


|   |  REGION_ID |  REGION_NAME |
|---|---------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| 1 | 1                                                                                           | Europe                                                                                        |
| 2 | 2                                                                                           | Americas                                                                                      |
| 3 | 3                                                                                           | Asia                                                                                          |
| 4 | 4                                                                                           | Middle East and Africa                                                                        |

## countries 테이블

DESCRIBE countries

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| COUNTRY_ID      | NOT NULL | CHAR(2)      |
| COUNTRY_NAME    |          | VARCHAR2(40) |
| REGION_ID       |          | NUMBER       |
| 3 rows selected |          |              |

SELECT \* FROM countries;

|    |  COUNTRY_ID |  COUNTRY_NAME |  REGION_ID |
|----|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| 1  | AR                                                                                           | Argentina                                                                                      | 2                                                                                           |
| 2  | AU                                                                                           | Australia                                                                                      | 3                                                                                           |
| 3  | BE                                                                                           | Belgium                                                                                        | 1                                                                                           |
| 4  | BR                                                                                           | Brazil                                                                                         | 2                                                                                           |
| 5  | CA                                                                                           | Canada                                                                                         | 2                                                                                           |
| 6  | CH                                                                                           | Switzerland                                                                                    | 1                                                                                           |
| 7  | CN                                                                                           | China                                                                                          | 3                                                                                           |
| 8  | DE                                                                                           | Germany                                                                                        | 1                                                                                           |
| 9  | DK                                                                                           | Denmark                                                                                        | 1                                                                                           |
| 10 | EG                                                                                           | Egypt                                                                                          | 4                                                                                           |
| 11 | FR                                                                                           | France                                                                                         | 1                                                                                           |
| 12 | HK                                                                                           | HongKong                                                                                       | 3                                                                                           |
| 13 | IL                                                                                           | Israel                                                                                         | 4                                                                                           |
| 14 | IN                                                                                           | India                                                                                          | 3                                                                                           |
| 15 | IT                                                                                           | Italy                                                                                          | 1                                                                                           |
| 16 | JP                                                                                           | Japan                                                                                          | 3                                                                                           |
| 17 | KW                                                                                           | Kuwait                                                                                         | 4                                                                                           |
| 18 | MX                                                                                           | Mexico                                                                                         | 2                                                                                           |
| 19 | NG                                                                                           | Nigeria                                                                                        | 4                                                                                           |
| 20 | NL                                                                                           | Netherlands                                                                                    | 1                                                                                           |
| 21 | SG                                                                                           | Singapore                                                                                      | 3                                                                                           |
| 22 | UK                                                                                           | United Kingdom                                                                                 | 1                                                                                           |
| 23 | US                                                                                           | United States of America                                                                       | 2                                                                                           |
| 24 | ZM                                                                                           | Zambia                                                                                         | 4                                                                                           |
| 25 | ZW                                                                                           | Zimbabwe                                                                                       | 4                                                                                           |

## locations 테이블

```
DESCRIBE locations;
```

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| LOCATION_ID     | NOT NULL | NUMBER(4)    |
| STREET_ADDRESS  |          | VARCHAR2(40) |
| POSTAL_CODE     |          | VARCHAR2(12) |
| CITY            | NOT NULL | VARCHAR2(30) |
| STATE_PROVINCE  |          | VARCHAR2(25) |
| COUNTRY_ID      |          | CHAR(2)      |
| 6 rows selected |          |              |

```
SELECT * FROM locations;
```

|    | LOC...       | STREET_ADDRESS            | POSTAL_CODE | CITY                | STATE_PROVINCE    | COUNTRY_ID |
|----|--------------|---------------------------|-------------|---------------------|-------------------|------------|
| 1  | 1000 1297    | Via Cola di Rie           | 00989       | Roma                | (null)            | IT         |
| 2  | 1100 93091   | Calle della Testa         | 10934       | Venice              | (null)            | IT         |
| 3  | 1200 2017    | Shinjuku-ku               | 1689        | Tokyo               | Tokyo Prefecture  | JP         |
| 4  | 1300 9450    | Kamiya-cho                | 6823        | Hiroshima           | (null)            | JP         |
| 5  | 1400 2014    | Jabberwocky Rd            | 26192       | Southlake           | Texas             | US         |
| 6  | 1500 2011    | Interiors Blvd            | 99236       | South San Francisco | California        | US         |
| 7  | 1600 2007    | Zagora St                 | 50090       | South Brunswick     | New Jersey        | US         |
| 8  | 1700 2004    | Charade Rd                | 98199       | Seattle             | Washington        | US         |
| 9  | 1800 147     | Spadina Ave               | M5V 2L7     | Toronto             | Ontario           | CA         |
| 10 | 1900 6092    | Boxwood St                | Y5W 9T2     | Whitehorse          | Yukon             | CA         |
| 11 | 2000 40-5-12 | Laogianggen               | 190518      | Beijing             | (null)            | CN         |
| 12 | 2100 1298    | Vileparle (E)             | 490231      | Bombay              | Maharashtra       | IN         |
| 13 | 2200 12-98   | Victoria Street           | 2901        | Sydney              | New South Wales   | AU         |
| 14 | 2300 198     | Clementi North            | 540198      | Singapore           | (null)            | SG         |
| 15 | 2400 8204    | Arthur St                 | (null)      | London              | (null)            | UK         |
| 16 | 2500         | Magdalen Centre, The ...  | OX9 9ZB     | Oxford              | Oxford            | UK         |
| 17 | 2600 9702    | Chester Road              | 09629850293 | Stretford           | Manchester        | UK         |
| 18 | 2700         | Schwanthalerstr. 7031     | 80925       | Munich              | Bavaria           | DE         |
| 19 | 2800         | Rua Frei Caneca 1360      | 01307-002   | Sao Paulo           | Sao Paulo         | BR         |
| 20 | 2900 20      | Rue des Corps-Saints      | 1730        | Geneva              | Geneve            | CH         |
| 21 | 3000         | Murtenstrasse 921         | 3095        | Bern                | BE                | CH         |
| 22 | 3100         | Pieter Breughelstraat 837 | 3029SK      | Utrecht             | Utrecht           | NL         |
| 23 | 3200         | Mariano Escobedo 9991     | 11932       | Mexico City         | Distrito Federal, | MX         |

## departments 테이블

DESCRIBE departments

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| DEPARTMENT_ID   | NOT NULL | NUMBER(4)    |
| DEPARTMENT_NAME | NOT NULL | VARCHAR2(30) |
| MANAGER_ID      |          | NUMBER(6)    |
| LOCATION_ID     |          | NUMBER(4)    |
| 4 rows selected |          |              |

SELECT \* FROM departments;





|    | DEPARTMENT_ID | DEPARTMENT_NAME      | MANAGER_ID | LOCATION_ID |
|----|---------------|----------------------|------------|-------------|
| 1  | 10            | Administration       | 200        | 1700        |
| 2  | 20            | Marketing            | 201        | 1800        |
| 3  | 30            | Purchasing           | 114        | 1700        |
| 4  | 40            | Human Resources      | 203        | 2400        |
| 5  | 50            | Shipping             | 121        | 1500        |
| 6  | 60            | IT                   | 103        | 1400        |
| 7  | 70            | Public Relations     | 204        | 2700        |
| 8  | 80            | Sales                | 145        | 2500        |
| 9  | 90            | Executive            | 100        | 1700        |
| 10 | 100           | Finance              | 108        | 1700        |
| 11 | 110           | Accounting           | 205        | 1700        |
| 12 | 120           | Treasury             | (null)     | 1700        |
| 13 | 130           | Corporate Tax        | (null)     | 1700        |
| 14 | 140           | Control And Credit   | (null)     | 1700        |
| 15 | 150           | Shareholder Services | (null)     | 1700        |
| 16 | 160           | Benefits             | (null)     | 1700        |
| 17 | 170           | Manufacturing        | (null)     | 1700        |
| 18 | 180           | Construction         | (null)     | 1700        |
| 19 | 190           | Contracting          | (null)     | 1700        |
| 20 | 200           | Operations           | (null)     | 1700        |
| 21 | 210           | IT Support           | (null)     | 1700        |
| 22 | 220           | NOC                  | (null)     | 1700        |
| 23 | 230           | IT Helpdesk          | (null)     | 1700        |
| 24 | 240           | Government Sales     | (null)     | 1700        |
| 25 | 250           | Retail Sales         | (null)     | 1700        |
| 26 | 260           | Recruiting           | (null)     | 1700        |
| 27 | 270           | Payroll              | (null)     | 1700        |

## jobs 테이블

```
DESCRIBE jobs
```

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| JOB_ID          | NOT NULL | VARCHAR2(10) |
| JOB_TITLE       | NOT NULL | VARCHAR2(35) |
| MIN_SALARY      |          | NUMBER(6)    |
| MAX_SALARY      |          | NUMBER(6)    |
| 4 rows selected |          |              |

```
SELECT * FROM jobs;
```

|    |  JOB_ID |  JOB_TITLE |  MIN_SALARY |  MAX_SALARY |
|----|------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| 1  | AD_PRES                                                                                  | President                                                                                   | 20000                                                                                        | 40000                                                                                          |
| 2  | AD_VP                                                                                    | Administration Vice President                                                               | 15000                                                                                        | 30000                                                                                          |
| 3  | AD_ASST                                                                                  | Administration Assistant                                                                    | 3000                                                                                         | 6000                                                                                           |
| 4  | FL_MGR                                                                                   | Finance Manager                                                                             | 8200                                                                                         | 16000                                                                                          |
| 5  | FL_ACCOUNT                                                                               | Accountant                                                                                  | 4200                                                                                         | 9000                                                                                           |
| 6  | AC_MGR                                                                                   | Accounting Manager                                                                          | 8200                                                                                         | 16000                                                                                          |
| 7  | AC_ACCOUNT                                                                               | Public Accountant                                                                           | 4200                                                                                         | 9000                                                                                           |
| 8  | SA_MAN                                                                                   | Sales Manager                                                                               | 10000                                                                                        | 20000                                                                                          |
| 9  | SA_REP                                                                                   | Sales Representative                                                                        | 6000                                                                                         | 12000                                                                                          |
| 10 | PU_MAN                                                                                   | Purchasing Manager                                                                          | 8000                                                                                         | 15000                                                                                          |
| 11 | PU_CLERK                                                                                 | Purchasing Clerk                                                                            | 2500                                                                                         | 5500                                                                                           |
| 12 | ST_MAN                                                                                   | Stock Manager                                                                               | 5500                                                                                         | 8500                                                                                           |
| 13 | ST_CLERK                                                                                 | Stock Clerk                                                                                 | 2000                                                                                         | 5000                                                                                           |
| 14 | SH_CLERK                                                                                 | Shipping Clerk                                                                              | 2500                                                                                         | 5500                                                                                           |
| 15 | IT_PROG                                                                                  | Programmer                                                                                  | 4000                                                                                         | 10000                                                                                          |
| 16 | MK_MAN                                                                                   | Marketing Manager                                                                           | 9000                                                                                         | 15000                                                                                          |
| 17 | MK_REP                                                                                   | Marketing Representative                                                                    | 4000                                                                                         | 9000                                                                                           |
| 18 | HR_REP                                                                                   | Human Resources Represen...                                                                 | 4000                                                                                         | 9000                                                                                           |
| 19 | PR_REP                                                                                   | Public Relations Representat...                                                             | 4500                                                                                         | 10500                                                                                          |

## employees 테이블

```
DESCRIBE employees
```

| Name             | Null     | Type         |
|------------------|----------|--------------|
| EMPLOYEE_ID      | NOT NULL | NUMBER(6)    |
| FIRST_NAME       |          | VARCHAR2(20) |
| LAST_NAME        | NOT NULL | VARCHAR2(25) |
| EMAIL            | NOT NULL | VARCHAR2(25) |
| PHONE_NUMBER     |          | VARCHAR2(20) |
| HIRE_DATE        | NOT NULL | DATE         |
| JOB_ID           | NOT NULL | VARCHAR2(10) |
| SALARY           |          | NUMBER(8,2)  |
| COMMISSION_PCT   |          | NUMBER(2,2)  |
| MANAGER_ID       |          | NUMBER(6)    |
| DEPARTMENT_ID    |          | NUMBER(4)    |
| 11 rows selected |          |              |

다음 스크린샷에서 EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY, COMMISSION\_PCT, MANAGER\_ID 및 DEPARTMENT\_ID 열의 머리글은 페이지에서 테이블 값을 맞추기 위해 각각 ID, FNAME, LNAME, SAL, COMM, MGRID 및 DEPTID로 설정되어 있습니다.

```
SELECT * FROM employees;
```

|    | ID  | FNAME     | LNAME     | EMAIL    | PHONE_NUMBER | HIRE_DATE | JOB_ID     | SAL   | COMM   | MGRID  | DEPTID |
|----|-----|-----------|-----------|----------|--------------|-----------|------------|-------|--------|--------|--------|
| 1  | 198 | Donald    | OConnell  | DOCONNEL | 650.507.9833 | 21-JUN-99 | SH_CLERK   | 2600  | (null) | 124    | 50     |
| 2  | 199 | Douglas   | Grant     | DGRANT   | 650.507.9844 | 13-JAN-00 | SH_CLERK   | 2600  | (null) | 124    | 50     |
| 3  | 200 | Jennifer  | Whalen    | JWHALEN  | 515.123.4444 | 17-SEP-87 | AD_ASST    | 4400  | (null) | 101    | 10     |
| 4  | 201 | Michael   | Hartstein | MHARTSTE | 515.123.5555 | 17-FEB-96 | MK_MAN     | 13000 | (null) | 100    | 20     |
| 5  | 202 | Pat       | Fay       | PFAY     | 603.123.6666 | 17-AUG-97 | MK_REP     | 6000  | (null) | 201    | 20     |
| 6  | 203 | Susan     | Mavris    | SMAVRIS  | 515.123.7777 | 07-JUN-94 | HR_REP     | 6500  | (null) | 101    | 40     |
| 7  | 204 | Hermann   | Baer      | HBAER    | 515.123.8888 | 07-JUN-94 | PR_REP     | 10000 | (null) | 101    | 70     |
| 8  | 205 | Shelley   | Higgins   | SHIGGINS | 515.123.8080 | 07-JUN-94 | AC_MGR     | 12000 | (null) | 101    | 110    |
| 9  | 206 | William   | Gietz     | WGIEZT   | 515.123.8181 | 07-JUN-94 | AC_ACCOUNT | 8300  | (null) | 205    | 110    |
| 10 | 100 | Steven    | King      | SKING    | 515.123.4567 | 17-JUN-87 | AD_PRES    | 24000 | (null) | (null) | 90     |
| 11 | 101 | Neena     | Kochhar   | NKOCHHAR | 515.123.4568 | 21-SEP-89 | AD_VP      | 17000 | (null) | 100    | 90     |
| 12 | 102 | Lex       | De Haan   | LDEHAAN  | 515.123.4569 | 13-JAN-93 | AD_VP      | 17000 | (null) | 100    | 90     |
| 13 | 103 | Alexander | Hunold    | AHUNOLD  | 590.423.4567 | 03-JAN-90 | IT_PROG    | 9000  | (null) | 102    | 60     |
| 14 | 104 | Bruce     | Ernst     | BERNST   | 590.423.4568 | 21-MAY-91 | IT_PROG    | 6000  | (null) | 103    | 60     |
| 15 | 105 | David     | Austin    | DAUSTIN  | 590.423.4569 | 25-JUN-97 | IT_PROG    | 4800  | (null) | 103    | 60     |
| 16 | 106 | Valli     | Pataballa | VPATABAL | 590.423.4560 | 05-FEB-98 | IT_PROG    | 4800  | (null) | 103    | 60     |
| 17 | 107 | Diana     | Lorentz   | DLORENTZ | 590.423.5567 | 07-FEB-99 | IT_PROG    | 4200  | (null) | 103    | 60     |
| 18 | 108 | Nancy     | Greenberg | NGREENBE | 515.124.4569 | 17-AUG-94 | FI_MGR     | 12000 | (null) | 101    | 100    |
| 19 | 109 | Daniel    | Faviet    | DFAVIET  | 515.124.4169 | 16-AUG-94 | FI_ACCOUNT | 9000  | (null) | 108    | 100    |
| 20 | 110 | John      | Chen      | JCHEN    | 515.124.4269 | 28-SEP-97 | FI_ACCOUNT | 8200  | (null) | 108    | 100    |

## employees 테이블(계속)

|    | ID  | FNAME       | LNAME       | EMAIL    | PHONE_NUMBER        | HIRE_DATE | JOB_ID     | SAL   | COMM   | MGRID | DEPTID |
|----|-----|-------------|-------------|----------|---------------------|-----------|------------|-------|--------|-------|--------|
| 21 | 111 | Ismael      | Sciarra     | ISCIARRA | 515.124.4369        | 30-SEP-97 | FI_ACCOUNT | 7700  | (null) | 108   | 100    |
| 22 | 112 | Jose Manuel | Urman       | JMURMAN  | 515.124.4469        | 07-MAR-98 | FI_ACCOUNT | 7800  | (null) | 108   | 100    |
| 23 | 113 | Luis        | Popp        | LPOPP    | 515.124.4567        | 07-DEC-99 | FI_ACCOUNT | 6900  | (null) | 108   | 100    |
| 24 | 114 | Den         | Raphaely    | DRAPHEAL | 515.127.4561        | 07-DEC-94 | PU_MAN     | 11000 | (null) | 100   | 30     |
| 25 | 115 | Alexander   | Khoo        | AKHOO    | 515.127.4562        | 18-MAY-95 | PU_CLERK   | 3100  | (null) | 114   | 30     |
| 26 | 116 | Shelli      | Baida       | SBAIDA   | 515.127.4563        | 24-DEC-97 | PU_CLERK   | 2900  | (null) | 114   | 30     |
| 27 | 117 | Sigal       | Tobias      | STOBIAS  | 515.127.4564        | 24-JUL-97 | PU_CLERK   | 2800  | (null) | 114   | 30     |
| 28 | 118 | Guy         | Himuro      | GHIMURO  | 515.127.4565        | 15-NOV-98 | PU_CLERK   | 2600  | (null) | 114   | 30     |
| 29 | 119 | Karen       | Colmenares  | KCOLMENA | 515.127.4566        | 10-AUG-99 | PU_CLERK   | 2500  | (null) | 114   | 30     |
| 30 | 120 | Matthew     | Weiss       | MWEISS   | 650.123.1234        | 18-JUL-96 | ST_MAN     | 8000  | (null) | 100   | 50     |
| 31 | 121 | Adam        | Fripp       | AFRIPP   | 650.123.2234        | 10-APR-97 | ST_MAN     | 8200  | (null) | 100   | 50     |
| 32 | 122 | Payam       | Kaufling    | PKAUFLIN | 650.123.3234        | 01-MAY-95 | ST_MAN     | 7900  | (null) | 100   | 50     |
| 33 | 123 | Shanta      | Vollman     | SVOLLMAN | 650.123.4234        | 10-OCT-97 | ST_MAN     | 6500  | (null) | 100   | 50     |
| 34 | 124 | Kevin       | Mourgos     | KMOURGOS | 650.123.5234        | 16-NOV-99 | ST_MAN     | 5800  | (null) | 100   | 50     |
| 35 | 125 | Julia       | Nayer       | JNAYER   | 650.124.1214        | 16-JUL-97 | ST_CLERK   | 3200  | (null) | 120   | 50     |
| 36 | 126 | Irene       | Mikkilineni | IMIKKILI | 650.124.1224        | 28-SEP-98 | ST_CLERK   | 2700  | (null) | 120   | 50     |
| 37 | 127 | James       | Landry      | JLANDRY  | 650.124.1334        | 14-JAN-99 | ST_CLERK   | 2400  | (null) | 120   | 50     |
| 38 | 128 | Steven      | Markle      | SMARKLE  | 650.124.1434        | 08-MAR-00 | ST_CLERK   | 2200  | (null) | 120   | 50     |
| 39 | 129 | Laura       | Bissot      | LBISSOT  | 650.124.5234        | 20-AUG-97 | ST_CLERK   | 3300  | (null) | 121   | 50     |
| 40 | 130 | Mozhe       | Atkinson    | MATKINSO | 650.124.6234        | 30-OCT-97 | ST_CLERK   | 2800  | (null) | 121   | 50     |
| 41 | 131 | James       | Marlow      | JAMRLOW  | 650.124.7234        | 16-FEB-97 | ST_CLERK   | 2500  | (null) | 121   | 50     |
| 42 | 132 | TJ          | Olson       | TJOLSON  | 650.124.8234        | 10-APR-99 | ST_CLERK   | 2100  | (null) | 121   | 50     |
| 43 | 133 | Jason       | Mallin      | JMALLIN  | 650.127.1934        | 14-JUN-96 | ST_CLERK   | 3300  | (null) | 122   | 50     |
| 44 | 134 | Michael     | Rogers      | MROGERS  | 650.127.1834        | 26-AUG-98 | ST_CLERK   | 2900  | (null) | 122   | 50     |
| 45 | 135 | Ki          | Gee         | KGEE     | 650.127.1734        | 12-DEC-99 | ST_CLERK   | 2400  | (null) | 122   | 50     |
| 46 | 136 | Hazel       | Philtanker  | HPHILTAN | 650.127.1634        | 06-FEB-00 | ST_CLERK   | 2200  | (null) | 122   | 50     |
| 47 | 137 | Renske      | Ladwig      | RLADWIG  | 650.121.1234        | 14-JUL-95 | ST_CLERK   | 3600  | (null) | 123   | 50     |
| 48 | 138 | Stephen     | Stiles      | SSTILES  | 650.121.2034        | 26-OCT-97 | ST_CLERK   | 3200  | (null) | 123   | 50     |
| 49 | 139 | John        | Seo         | JSEO     | 650.121.2019        | 12-FEB-98 | ST_CLERK   | 2700  | (null) | 123   | 50     |
| 50 | 140 | Joshua      | Patel       | JPATEL   | 650.121.1834        | 06-APR-98 | ST_CLERK   | 2500  | (null) | 123   | 50     |
| 51 | 141 | Trenna      | Rajs        | TRAJS    | 650.121.8009        | 17-OCT-95 | ST_CLERK   | 3500  | (null) | 124   | 50     |
| 52 | 142 | Curtis      | Davies      | CDAVIES  | 650.121.2994        | 29-JAN-97 | ST_CLERK   | 3100  | (null) | 124   | 50     |
| 53 | 143 | Randall     | Matos       | RMATOS   | 650.121.2874        | 15-MAR-98 | ST_CLERK   | 2600  | (null) | 124   | 50     |
| 54 | 144 | Peter       | Vargas      | PVARGAS  | 650.121.2004        | 09-JUL-98 | ST_CLERK   | 2500  | (null) | 124   | 50     |
| 55 | 145 | John        | Russell     | JRUSSEL  | 011.44.1344.4292... | 01-OCT-96 | SA_MAN     | 14000 | 0.4    | 100   | 80     |
| 56 | 146 | Karen       | Partners    | KPARTNER | 011.44.1344.4672... | 05-JAN-97 | SA_MAN     | 13500 | 0.3    | 100   | 80     |
| 57 | 147 | Alberto     | Errazuriz   | AERRAZUR | 011.44.1344.4292... | 10-MAR-97 | SA_MAN     | 12000 | 0.3    | 100   | 80     |
| 58 | 148 | Gerald      | Cambrault   | GCAMBRAU | 011.44.1344.6192... | 15-OCT-99 | SA_MAN     | 11000 | 0.3    | 100   | 80     |
| 59 | 149 | Eleni       | Zlotkey     | EZLOTKEY | 011.44.1344.4290... | 29-JAN-00 | SA_MAN     | 10500 | 0.2    | 100   | 80     |
| 60 | 150 | Peter       | Tucker      | PTUCKER  | 011.44.1344.1292... | 30-JAN-97 | SA_REP     | 10000 | 0.3    | 145   | 80     |
| 61 | 151 | David       | Bernstein   | DBERNSTE | 011.44.1344.3452... | 24-MAR-97 | SA_REP     | 9500  | 0.25   | 145   | 80     |
| 62 | 152 | Peter       | Hall        | PHALL    | 011.44.1344.4789... | 20-AUG-97 | SA_REP     | 9000  | 0.25   | 145   | 80     |
| 63 | 153 | Christopher | Olsen       | COLSEN   | 011.44.1344.4987... | 30-MAR-98 | SA_REP     | 8000  | 0.2    | 145   | 80     |
| 64 | 154 | Nanette     | Cambrault   | NCAMBRAU | 011.44.1344.9876... | 09-DEC-98 | SA_REP     | 7500  | 0.2    | 145   | 80     |
| 65 | 155 | Oliver      | Tuvault     | OTUVAULT | 011.44.1344.4865... | 23-NOV-99 | SA_REP     | 7000  | 0.15   | 145   | 80     |

## employees 테이블(계속)

| ID  | FNAME         | LNAME      | EMAIL    | PHONE_NUMBER        | HIRE_DATE | JOB_ID   | SAL   | COMM   | MGRID | DEPTID |
|-----|---------------|------------|----------|---------------------|-----------|----------|-------|--------|-------|--------|
| 66  | 156 Janette   | King       | JKING    | 011.44.1345.4292... | 30-JAN-96 | SA_REP   | 10000 | 0.35   | 146   | 80     |
| 67  | 157 Patrick   | Sully      | PSULLY   | 011.44.1345.9292... | 04-MAR-96 | SA_REP   | 9500  | 0.35   | 146   | 80     |
| 68  | 158 Allan     | McEwen     | AMCEWEN  | 011.44.1345.8292... | 01-AUG-96 | SA_REP   | 9000  | 0.35   | 146   | 80     |
| 69  | 159 Lindsey   | Smith      | LSMITH   | 011.44.1345.7292... | 10-MAR-97 | SA_REP   | 8000  | 0.3    | 146   | 80     |
| 70  | 160 Louise    | Doran      | LDORAN   | 011.44.1345.6292... | 15-DEC-97 | SA_REP   | 7500  | 0.3    | 146   | 80     |
| 71  | 161 Sarath    | Sewall     | SSEWALL  | 011.44.1345.5292... | 03-NOV-98 | SA_REP   | 7000  | 0.25   | 146   | 80     |
| 72  | 162 Clara     | Vishney    | CVISHNEY | 011.44.1346.1292... | 11-NOV-97 | SA_REP   | 10500 | 0.25   | 147   | 80     |
| 73  | 163 Danielle  | Greene     | DGREENE  | 011.44.1346.2292... | 19-MAR-99 | SA_REP   | 9500  | 0.15   | 147   | 80     |
| 74  | 164 Mattea    | Marvins    | MMARVINS | 011.44.1346.3292... | 24-JAN-00 | SA_REP   | 7200  | 0.1    | 147   | 80     |
| 75  | 165 David     | Lee        | DLEE     | 011.44.1346.5292... | 23-FEB-00 | SA_REP   | 6800  | 0.1    | 147   | 80     |
| 76  | 166 Sundar    | Ande       | SANDE    | 011.44.1346.6292... | 24-MAR-00 | SA_REP   | 6400  | 0.1    | 147   | 80     |
| 77  | 167 Amit      | Banda      | ABANDA   | 011.44.1346.7292... | 21-APR-00 | SA_REP   | 6200  | 0.1    | 147   | 80     |
| 78  | 168 Lisa      | Ozer       | LOZER    | 011.44.1343.9292... | 11-MAR-97 | SA_REP   | 11500 | 0.25   | 148   | 80     |
| 79  | 169 Harrison  | Bloom      | HBLOOM   | 011.44.1343.8292... | 23-MAR-98 | SA_REP   | 10000 | 0.2    | 148   | 80     |
| 80  | 170 Tayler    | Fox        | TFOX     | 011.44.1343.7292... | 24-JAN-98 | SA_REP   | 9600  | 0.2    | 148   | 80     |
| 81  | 171 William   | Smith      | WSMITH   | 011.44.1343.6292... | 23-FEB-99 | SA_REP   | 7400  | 0.15   | 148   | 80     |
| 82  | 172 Elizabeth | Bates      | EBATES   | 011.44.1343.5292... | 24-MAR-99 | SA_REP   | 7300  | 0.15   | 148   | 80     |
| 83  | 173 Sundita   | Kumar      | SKUMAR   | 011.44.1343.3292... | 21-APR-00 | SA_REP   | 6100  | 0.1    | 148   | 80     |
| 84  | 174 Ellen     | Abel       | EABEL    | 011.44.1644.4292... | 11-MAY-96 | SA_REP   | 11000 | 0.3    | 149   | 80     |
| 85  | 175 Alyssa    | Hutton     | AHUTTON  | 011.44.1644.4292... | 19-MAR-97 | SA_REP   | 8800  | 0.25   | 149   | 80     |
| 86  | 176 Jonathon  | Taylor     | JTAYLOR  | 011.44.1644.4292... | 24-MAR-98 | SA_REP   | 8600  | 0.2    | 149   | 80     |
| 87  | 177 Jack      | Livingston | JLIVINGS | 011.44.1644.4292... | 23-APR-98 | SA_REP   | 8400  | 0.2    | 149   | 80     |
| 88  | 178 Kimberely | Grant      | KGRANT   | 011.44.1644.4292... | 24-MAY-99 | SA_REP   | 7000  | 0.15   | 149   | (null) |
| 89  | 179 Charles   | Johnson    | CJOHNSON | 011.44.1644.4292... | 04-JAN-00 | SA_REP   | 6200  | 0.1    | 149   | 80     |
| 90  | 180 Winston   | Taylor     | WTAYLOR  | 650.507.9876        | 24-JAN-98 | SH_CLERK | 3200  | (null) | 120   | 50     |
| 91  | 181 Jean      | Fleaur     | JFLEAUR  | 650.507.9877        | 23-FEB-98 | SH_CLERK | 3100  | (null) | 120   | 50     |
| 92  | 182 Martha    | Sullivan   | MSULLIVA | 650.507.9878        | 21-JUN-99 | SH_CLERK | 2500  | (null) | 120   | 50     |
| 93  | 183 Girard    | Geoni      | GGEONI   | 650.507.9879        | 03-FEB-00 | SH_CLERK | 2800  | (null) | 120   | 50     |
| 94  | 184 Nandita   | Sarchand   | NSARCHAN | 650.509.1876        | 27-JAN-96 | SH_CLERK | 4200  | (null) | 121   | 50     |
| 95  | 185 Alexis    | Bull       | ABULL    | 650.509.2876        | 20-FEB-97 | SH_CLERK | 4100  | (null) | 121   | 50     |
| 96  | 186 Julia     | Dellinger  | JDELLING | 650.509.3876        | 24-JUN-98 | SH_CLERK | 3400  | (null) | 121   | 50     |
| 97  | 187 Anthony   | Cabrio     | ACABRIO  | 650.509.4876        | 07-FEB-99 | SH_CLERK | 3000  | (null) | 121   | 50     |
| 98  | 188 Kelly     | Chung      | KCHUNG   | 650.505.1876        | 14-JUN-97 | SH_CLERK | 3800  | (null) | 122   | 50     |
| 99  | 189 Jennifer  | Dilly      | JDILLY   | 650.505.2876        | 13-AUG-97 | SH_CLERK | 3600  | (null) | 122   | 50     |
| 100 | 190 Timothy   | Gates      | TGATES   | 650.505.3876        | 11-JUL-98 | SH_CLERK | 2900  | (null) | 122   | 50     |
| 101 | 191 Randall   | Perkins    | RPERKINS | 650.505.4876        | 19-DEC-99 | SH_CLERK | 2500  | (null) | 122   | 50     |
| 102 | 192 Sarah     | Bell       | SBELL    | 650.501.1876        | 04-FEB-96 | SH_CLERK | 4000  | (null) | 123   | 50     |
| 103 | 193 Britney   | Everett    | BEVERETT | 650.501.2876        | 03-MAR-97 | SH_CLERK | 3900  | (null) | 123   | 50     |
| 104 | 194 Samuel    | McCain     | SMCCAIN  | 650.501.3876        | 01-JUL-98 | SH_CLERK | 3200  | (null) | 123   | 50     |
| 105 | 195 Vance     | Jones      | VJONES   | 650.501.4876        | 17-MAR-99 | SH_CLERK | 2800  | (null) | 123   | 50     |
| 106 | 196 Alana     | Walsh      | AWALSH   | 650.507.9811        | 24-APR-98 | SH_CLERK | 3100  | (null) | 124   | 50     |
| 107 | 197 Kevin     | Feeney     | KFEENEY  | 650.507.9822        | 23-MAY-98 | SH_CLERK | 3000  | (null) | 124   | 50     |



## job\_history 테이블

```
DESCRIBE job_history
```

| Name            | Null     | Type         |
|-----------------|----------|--------------|
| EMPLOYEE_ID     | NOT NULL | NUMBER(6)    |
| START_DATE      | NOT NULL | DATE         |
| END_DATE        | NOT NULL | DATE         |
| JOB_ID          | NOT NULL | VARCHAR2(10) |
| DEPARTMENT_ID   |          | NUMBER(4)    |
| 5 rows selected |          |              |

```
SELECT * FROM job_history;
```

|    | EMPLOYEE_ID | START_DATE | END_DATE  | JOB_ID     | DEPARTMENT_ID |
|----|-------------|------------|-----------|------------|---------------|
| 1  | 102         | 13-JAN-93  | 24-JUL-98 | IT_PROG    | 60            |
| 2  | 101         | 21-SEP-89  | 27-OCT-93 | AC_ACCOUNT | 110           |
| 3  | 101         | 28-OCT-93  | 15-MAR-97 | AC_MGR     | 110           |
| 4  | 201         | 17-FEB-96  | 19-DEC-99 | MK_REP     | 20            |
| 5  | 114         | 24-MAR-98  | 31-DEC-99 | ST_CLERK   | 50            |
| 6  | 122         | 01-JAN-99  | 31-DEC-99 | ST_CLERK   | 50            |
| 7  | 200         | 17-SEP-87  | 17-JUN-93 | AD_ASST    | 90            |
| 8  | 176         | 24-MAR-98  | 31-DEC-98 | SA_REP     | 80            |
| 9  | 176         | 01-JAN-99  | 31-DEC-99 | SA_MAN     | 80            |
| 10 | 200         | 01-JUL-94  | 31-DEC-98 | AC_ACCOUNT | 90            |



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 커서 변수

- 커서 변수는 C 또는 Pascal의 포인터와 같습니다. 즉, 항목 자체 대신 항목의 메모리 위치(주소)를 보유합니다.
- PL/SQL에서 포인터는 REF x로 선언됩니다.여기서 REF는 REFERENCE의 줄임말이며 x는 객체 클래스를 나타냅니다.
- 커서 변수는 REF CURSOR 데이터 유형을 가집니다.
- 커서는 정적이며 커서 변수는 동적입니다.
- 커서 변수는 융통성을 향상시킵니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 변수

커서 변수는 C 또는 Pascal의 포인터와 같습니다. 즉, 항목 자체 대신 항목의 메모리 위치(주소)를 보유합니다. 따라서 커서 변수를 선언하면 항목이 아닌 포인터가 생성됩니다. PL/SQL에서 포인터는 REF x 데이터 유형을 가집니다. 여기서 REF는 REFERENCE의 줄임말이며 x는 객체 클래스를 나타냅니다. 커서 변수는 REF CURSOR 데이터 유형을 가집니다.

커서와 마찬가지로 커서 변수는 다중 행 쿼리의 결과 집합에서 현재 행을 가리킵니다. 그러나 상수와 변수가 다른 것처럼 커서는 커서 변수와 다릅니다. 커서는 정적이지만 커서 변수는 특정 query에 종속되지 않으므로 동적입니다. 유형 호환이 가능한 query에 대해 커서 변수를 열 수 있습니다. 커서 변수는 융통성을 향상시킵니다.

커서 변수는 모든 PL/SQL 클라이언트에서 사용 가능합니다. 예를 들어, OCI 또는 Pro\*C 프로그램과 같은 PL/SQL 호스트 환경에서 커서 변수를 선언한 다음 PL/SQL에 입력 호스트 변수 바인드 변수로 전달할 수 있습니다. 또한 PL/SQL 엔진이 있는 Oracle Forms 및 Oracle Reports와 같은 응용 프로그램 개발 툴에서도 클라이언트측에서만 커서 변수를 사용할 수 있습니다. Oracle 서버에도 PL/SQL 엔진이 있습니다. 따라서 RPC(원격 프로시저 호출)를 통해 응용 프로그램과 서버 간에 커서 변수를 전달할 수 있습니다.

## 커서 변수를 사용하는 이유

- PL/SQL 내장 서브 프로그램과 다양한 클라이언트 간에 query 결과 집합을 전달하는 데 커서 변수를 사용할 수 있습니다.
- PL/SQL은 결과 집합이 저장되는 query 작업 영역에 대한 포인터를 공유할 수 있습니다.
- 커서 변수 값을 한 범위에서 다른 범위로 자유롭게 전달할 수 있습니다.
- PL/SQL 블록은 한 번의 왕복으로 여러 개의 호스트 커서 변수를 열거나 닫을 수 있으므로 네트워크 통신량을 줄일 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 커서 변수를 사용하는 이유

PL/SQL 내장 서브 프로그램과 다양한 클라이언트 간에 query 결과 집합을 전달하는 데 커서 변수를 사용합니다. PL/SQL도 해당 클라이언트도 결과 집합을 소유하지 않으며 결과 집합이 저장되는 query 작업 영역에 대한 포인터를 공유할 뿐입니다. 예를 들어, OCI 클라이언트, Oracle Forms 응용 프로그램 및 Oracle 서버는 모두 동일한 작업 영역을 참조할 수 있습니다.

커서 변수가 query 작업 영역을 가리키는 한 이 영역은 액세스 가능한 상태를 유지합니다. 그러므로 커서 변수 값을 한 범위에서 다른 범위로 자유롭게 전달할 수 있습니다. 예를 들어, Pro\*C 프로그램에 포함된 PL/SQL 블록에 호스트 커서 변수를 전달하는 경우 커서 변수가 가리키는 작업 영역은 블록 종료 후에도 액세스 가능한 상태로 존재합니다.

클라이언트측에 PL/SQL 엔진이 있으면 클라이언트에서 서버로 호출하는 데 아무 제한이 없습니다. 예를 들어, 클라이언트측에서 커서 변수를 선언하고, 서버측에서 커서 변수를 열어 패치(fetch)한 다음, 계속해서 클라이언트측에서 다시 커서 변수에서 패치할 수 있습니다. 또한 PL/SQL 블록은 한 번의 왕복으로 여러 개의 호스트 커서 변수를 열거나 닫을 수 있으므로 네트워크 통신량을 줄일 수 있습니다.

커서 변수는 정적 이름으로 커서 작업 영역을 지정하는 대신 PGA에 커서 작업 영역에 대한 참조를 보유합니다. 이 영역을 참조에 의해 지정하기 때문에 변수를 융통성 있게 사용할 수 있습니다.

## REF CURSOR 유형 정의

- REF CURSOR 유형 정의:

```
Define a REF CURSOR type
TYPE ref_type_name IS REF CURSOR [RETURN
return_type];
```

- 해당 유형의 커서 변수 선언:

```
ref_cv ref_type_name;
```

- 예제:

```
DECLARE
TYPE DeptCurTyp IS REF CURSOR RETURN
departments%ROWTYPE;
dept_cv DeptCurTyp;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REF CURSOR 유형 정의

REF CURSOR를 정의하려면 다음 두 단계를 수행합니다. 먼저 REF CURSOR 유형을 정의한 다음 해당 유형의 커서 변수를 선언합니다. 다음 구문을 사용하여 PL/SQL 블록, 서브 프로그램 또는 패키지에서 REF CURSOR 유형을 정의할 수 있습니다.

```
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

구문 설명:

|               |                              |
|---------------|------------------------------|
| ref_type_name | 커서 변수의 이후 선언에 사용되는 유형 지정자입니다 |
| return_type   | 데이터베이스 테이블의 레코드 또는 행을 나타냅니다. |

위의 예제에서 DEPARTMENT 데이터베이스 테이블의 행을 나타내는 반환 유형을 지정합니다.

REF CURSOR 유형은 strong(제한) 또는 weak(비제한)일 수 있습니다. 다음 예제에서 알 수 있듯이 strong REF CURSOR 유형 정의는 반환 유형을 지정하지만 weak 정의는 지정하지 않습니다.

```
DECLARE
```

```
TYPE EmpCurTyp IS REF CURSOR RETURN employees%ROWTYPE; -- strong
TYPE GenericCurTyp IS REF CURSOR; -- weak
```

## REF CURSOR 유형 정의(계속)

PL/SQL 컴파일러가 strong 유형의 커서 변수를 유형 호환이 가능한 query와만 연관시키므로 strong REF CURSOR 유형은 오류 발생 가능성이 낮습니다. 그러나 컴파일러가 weak 유형의 커서 변수를 사용하여 어떠한 query와도 연관시킬 수 있으므로 weak REF CURSOR 유형은 보다 큰 융통성을 지닙니다.

## 커서 변수 선언

REF CURSOR 유형을 정의한 후 PL/SQL 블록이나 서브 프로그램에서 해당 유형의 커서 변수를 선언할 수 있습니다. 다음 예제에서는 DEPT\_CV 커서 변수를 선언합니다.

```
DECLARE

 TYPE DeptCurTyp IS REF CURSOR RETURN departments%ROWTYPE;
 dept_cv DeptCurTyp; -- declare cursor variable
```

**주:** 패키지에는 커서 변수를 선언할 수 없습니다. 패키지 변수와 달리 커서 변수는 지속 상태를 갖지 않습니다. 커서 변수를 선언하면 항목이 아닌 포인터가 생성됨을 기억하십시오. 커서 변수는 데이터베이스에 저장될 수 없으며 일반적인 범위 지정 및 instantiation 규칙을 따릅니다.

REF CURSOR 유형 정의의 RETURN 절에서 다음과 같이 %ROWTYPE을 사용하여 strong(weak 가 아닌) 유형의 커서 변수에 의해 반환된 행을 나타내는 레코드 유형을 지정할 수 있습니다.

```
DECLARE

 TYPE TmpCurTyp IS REF CURSOR RETURN employees%ROWTYPE;
 tmp_cv TmpCurTyp; -- declare cursor variable
 TYPE EmpCurTyp IS REF CURSOR RETURN tmp_cv%ROWTYPE;
 emp_cv EmpCurTyp; -- declare cursor variable
```

마찬가지로 다음 예제에서 알 수 있듯이 %TYPE을 사용하여 레코드 변수의 데이터 유형을 제공할 수 있습니다.

```
DECLARE

 dept_rec departments%ROWTYPE; -- declare record variable
 TYPE DeptCurTyp IS REF CURSOR RETURN dept_rec%TYPE;
 dept_cv DeptCurTyp; -- declare cursor variable
```

마지막 예제에서 RETURN 절에 유저 정의 RECORD 유형을 지정합니다.

```
DECLARE

 TYPE EmpRecTyp IS RECORD (
 empno NUMBER(4),
 ename VARCHAR2(10),
 sal NUMBER(7,2));
 TYPE EmpCurTyp IS REF CURSOR RETURN EmpRecTyp;
 emp_cv EmpCurTyp; -- declare cursor variable
```

## 파라미터로서의 커서 변수

커서 변수를 함수와 프로시저의 형식 파라미터로 선언할 수 있습니다. 다음 예제에서 REF CURSOR 유형의 EmpCurTyp을 정의한 다음 해당 유형의 커서 변수를 프로시저의 형식 파라미터로 선언합니다.

```
DECLARE
```

```
 TYPE EmpCurTyp IS REF CURSOR RETURN emp%ROWTYPE;
```

```
 PROCEDURE open_emp_cv (emp_cv IN OUT EmpCurTyp) IS ...
```

## OPEN-FOR, FETCH 및 CLOSE 문 사용

- OPEN-FOR 문은 커서 변수를 다중 행 쿼리와 연관시키고 query를 실행하며 결과 집합을 식별하고 결과 집합의 첫번째 행을 가리키도록 커서의 위치를 지정합니다.
- FETCH 문은 다중 행 쿼리의 결과 집합에서 행을 반환하고 select-list 항목의 값을 INTO 절의 해당 변수 또는 필드에 할당하며 %ROWCOUNT를 사용하여 유지되는 수를 증가시키고 커서를 다음 행으로 이동합니다.
- CLOSE 문은 커서 변수를 비활성화합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### OPEN-FOR, FETCH 및 CLOSE 문 사용

동적 다중 행 쿼리를 처리하기 위해 OPEN-FOR, FETCH 및 CLOSE 문을 사용합니다. 먼저 다중 행 쿼리에 "대한" 커서 변수를 "열니다". 그런 다음 한 번에 하나씩 결과 집합에서 행을 "페치(fetch)"합니다. 모든 행이 처리되면 커서 변수를 "닫습니다".

#### 커서 변수 열기

OPEN-FOR 문은 커서 변수를 다중 행 쿼리와 연관시키고 query를 실행하며 결과 집합을 식별하고 결과 집합의 첫번째 행을 가리키도록 커서의 위치를 지정한 다음 %ROWCOUNT를 사용하여 유지되는 처리된 행 수를 0으로 설정합니다. 정적 형식인 OPEN-FOR와는 달리 동적 형식은 선택적인 USING 절을 가집니다. 런타임에 USING 절의 바인드 인수는 동적 SELECT 문의 해당 자리 표시자를 대신합니다. 구문은 다음과 같습니다.

```
OPEN {cursor_variable | :host_cursor_variable} FOR
dynamic_string

[USING bind_argument[, bind_argument]...];
```

여기서 CURSOR\_VARIABLE은 weak 유형의 커서 변수이고(반환 유형 없음)

HOST\_CURSOR\_VARIABLE은 OCI 프로그램과 같은 PL/SQL 호스트 환경에서 선언된 커서 변수이며 dynamic\_string은 다중 행 쿼리를 나타내는 문자열 식입니다.



## OPEN-FOR, FETCH 및 CLOSE 문 사용(계속)

다음 예제의 구문에서는 커서 변수를 선언한 다음 employees 테이블에서 행을 반환하는 동적 SELECT 문과 연관시킵니다.

```
DECLARE
 TYPE EmpCurTyp IS REF CURSOR; -- define weak REF CURSOR type
 emp_cv EmpCurTyp; -- declare cursor variable
 my_ename VARCHAR2(15);
 my_sal NUMBER := 1000;
BEGIN
 OPEN emp_cv FOR -- open cursor variable
 'SELECT last_name, salary FROM employees WHERE salary >
 :s'
 USING my_sal;
 ...
END;
```

query의 바인드 인수는 커서 변수가 열려 있는 경우에만 평가됩니다. 그러므로 다른 바인드 값을 사용하여 커서에서 행을 패치(fetch)하려면 새 값으로 설정된 바인드 인수로 커서 변수를 다시 열어야 합니다.

### 커서 변수에서 패치(fetch)

FETCH 문은 다중 행 쿼리의 결과 집합에서 행을 반환하고 select-list 항목의 값을 INTO 절의 해당 변수 또는 필드에 할당하며 %ROWCOUNT를 사용하여 유지되는 수를 증가시키고 커서를 다음 행으로 이동합니다. 다음 구문을 사용하십시오.

```
FETCH {cursor_variable | :host_cursor_variable}
 INTO {define_variable[, define_variable]... | record};
```

예제를 계속 수행하면 EMP\_CV 커서 변수에서 MY\_ENAME 및 MY\_SAL 정의 변수로 행을 패치합니다.

```
LOOP
 FETCH emp_cv INTO my_ename, my_sal; -- fetch next row
 EXIT WHEN emp_cv%NOTFOUND; -- exit loop when last row is
 fetched
 -- process row
END LOOP;
```

커서 변수와 연관된 query에 의해 반환된 각 열 값의 경우 INTO 절에 유형 호환이 가능한 해당 변수 또는 필드가 있어야 합니다. 동일한 커서 변수를 사용하여 개별 패치(fetch)마다 다른 INTO 절을 사용할 수 있습니다. 각 패치는 동일한 결과 집합에서 다른 행을 읽어 들입니다. 단히거나 전혀 열린 경우가 없는 커서 변수에서 패치하려고 하면 PL/SQL은 미리 정의된 예외 INVALID\_CURSOR를 발생시킵니다.

## OPEN-FOR, FETCH 및 CLOSE 문 사용(계속)

### 커서 변수 닫기

CLOSE 문은 커서 변수를 비활성화합니다. 그 이후에 연관된 결과 집합은 미정의 상태가 됩니다. 다음 구문을 사용하십시오.

```
CLOSE {cursor_variable | :host_cursor_variable};
```

In this example, when the last row is processed, close the EMP\_CV cursor variable:

```
LOOP
 FETCH emp_cv INTO my_ename, my_sal;
 EXIT WHEN emp_cv%NOTFOUND;
 -- process row
END LOOP;
CLOSE emp_cv; -- close cursor variable
```

If you try to close an already-closed or never-opened cursor variable, PL/SQL raises INVALID\_CURSOR.

## 패치(Fetch) 예제

```
DECLARE
 TYPE EmpCurTyp IS REF CURSOR;
 emp_cv EmpCurTyp;
 emp_rec employees%ROWTYPE;
 sql_stmt VARCHAR2(200);
 my_job VARCHAR2(10) := 'ST_CLERK';
BEGIN
 sql_stmt := 'SELECT * FROM employees
 WHERE job_id = :j';
 OPEN emp_cv FOR sql_stmt USING my_job;
 LOOP
 FETCH emp_cv INTO emp_rec;
 EXIT WHEN emp_cv%NOTFOUND;
 -- process record
 END LOOP;
 CLOSE emp_cv;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 패치(Fetch) 예제

슬라이드의 예제는 동적 다중 행 쿼리의 결과 집합에서 레코드로 행을 패치하는 방법을 보여줍니다. 먼저 REF CURSOR 유형의 EmpCurTyp을 정의해야 합니다. 그런 다음 EmpCurTyp 유형의 emp\_cv 커서 변수를 정의합니다. PL/SQL 블록의 실행 섹션에서 OPEN-FOR 문은 EMP\_CV 커서 변수를 sql\_stmt 다중 행 쿼리와 연관시킵니다. FETCH 문은 다중 행 쿼리의 결과 집합에서 행을 반환하고 select-list 항목의 값을 INTO 절의 EMP\_REC에 할당합니다. 마지막 행이 처리되면 EMP\_CV 커서 변수를 닫습니다.

---

## 추가 연습

---

## 추가 연습 개요

이 추가 연습은 *Oracle Database 10g: PL/SQL Fundamentals* 과정의 보충 자료로 제공됩니다.

이 연습에서는 *Oracle Database 10g: PL/SQL Fundamentals*에서 배운 개념을 적용합니다.

이 추가 연습에서는 변수 선언, 실행문 작성, Oracle 서버와의 상호 작용, 제어 구조 작성, 복합 데이터 유형, 커서 사용 및 예외 처리 등을 보완할 수 있는 연습을 제공합니다. 이 추가 연습 부분에 사용된 테이블에는 `employees`, `jobs`, `job_history`, `departments` 등이 있습니다.

## 추가 연습 1 및 2

주: 이 연습은 변수를 선언하고 실행문을 작성하는 방법을 설명할 때 추가 연습으로 사용될 수 있습니다.

1. 다음에 나열된 선언을 검토하여 그 중 잘못된 선언을 판별하고 그 이유를 설명합니다.

a. DECLARE

```
name,dept VARCHAR2(14);
```

b. DECLARE

```
test NUMBER(5);
```

c. DECLARE

```
MAXSALARY NUMBER(7,2) = 5000;
```

d. DECLARE

```
JOINDATE BOOLEAN := SYSDATE;
```

2. 다음 각 할당에서 결과 표현식의 데이터 유형을 파악합니다.

a. email := firstname || to\_char(empno);

b. confirm := to\_date('20-JAN-1999', 'DD-MON-YYYY');

c. sal := (1000\*12) + 500

d. test := FALSE;

e. temp := temp1 < (temp2/ 3);

f. var := sysdate;

### 추가 연습 3

```
DECLARE
 custid NUMBER(4) := 1600;
 custname VARCHAR2(300) := 'Women Sports Club';
 new_custid NUMBER(3) := 500;
BEGIN
DECLARE
 custid NUMBER(4) := 0;
 custname VARCHAR2(300) := 'Shape up Sports Club';
 new_custid NUMBER(3) := 300;
 new_custname VARCHAR2(300) := 'Jansports Club';
BEGIN
 custid := new_custid;
 custname := custname || ' ' || new_custname;
END;

 custid := (custid *12) / 10;
END;

/
```

1

END;

2

END;

3. 위에 제시된 PL/SQL 블록을 검토하여 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형 및 값을 판별합니다.
  - a. 1 위치에서의 CUSTID 값:
  - b. 1 위치에서의 CUSTNAME 값:
  - c. 2 위치에서의 NEW\_CUSTID 값:
  - d. 1 위치에서의 NEW\_CUSTNAME 값:
  - e. 2 위치에서의 CUSTID 값:
  - f. 2 위치에서의 CUSTNAME 값:

**주:** 이 연습은 Oracle 서버와 상호 작용하고 제어 구조를 작성하는 방법을 설명할 때 추가 연습으로 사용될 수 있습니다.

4. 연도를 받아들이는 PL/SQL 블록을 작성하고 연도가 윤년인지 여부를 확인합니다. 예를 들어, 입력한 연도가 1990이면 "1990 is not a leap year"가 출력되어야 합니다.  
**힌트:** 윤년은 4로 정확히 나누어 떨어지며 100의 배수는 아닙니다. 그렇지만 400의 배수는 윤년입니다.

## 추가 연습 4 및 5

다음 연도로 솔루션을 테스트합니다.

|      |                 |
|------|-----------------|
| 1990 | Not a leap year |
| 2000 | Leap year       |
| 1996 | Leap year       |
| 1886 | Not a leap year |
| 1992 | Leap year       |
| 1824 | Leap year       |

```
anonymous block completed
1990 is not a leap year
```

5. a. 아래 연습의 경우 결과를 저장할 임시 테이블이 필요합니다. 테이블을 직접 생성하거나 테이블을 자동으로 생성할 `lab_ap_05.sql` 스크립트를 실행할 수 있습니다. 다음 세 개의 열을 가진 `TEMP`라는 테이블을 생성합니다.

| 열 이름         | NUM_STORE | CHAR_STORE | DATE_STORE |
|--------------|-----------|------------|------------|
| 키 유형         |           |            |            |
| Nulls/Unique |           |            |            |
| FK 테이블       |           |            |            |
| FK 열         |           |            |            |
| 데이터 유형       | Number    | VARCHAR2   | Date       |
| 길이           | 7,2       | 35         |            |

- b. 두 개의 변수 `MESSAGE` 및 `DATE_WRITTEN`을 포함하는 `PL/SQL` 블록을 작성합니다. `MESSAGE`를 길이가 35인 `VARCHAR2` 데이터 유형으로 선언하고 `DATE_WRITTEN`을 `DATE` 데이터 유형으로 선언합니다. 다음 값을 해당 변수에 할당합니다.

| 변수                        | 내용                              |
|---------------------------|---------------------------------|
| <code>MESSAGE</code>      | This is my first PL/SQL program |
| <code>DATE_WRITTEN</code> | Current date                    |

`TEMP` 테이블의 해당 열에 값을 저장합니다. `TEMP` 테이블을 query하여 결과를 확인합니다.

|   | NUM_STORE | CHAR_STORE                     | DATE_STORE |
|---|-----------|--------------------------------|------------|
| 1 | (null)    | This is my first PLSQL Program | 26-FEB-09  |



## 추가 연습 6 및 7

6. a. 부서 번호를 대체 변수에 저장합니다.  
b. 해당 부서에서 근무하는 사원 수를 출력하는 PL/SQL 블록을 작성합니다.  
**힌트:** SET SERVEROUTPUT ON을 사용하여 DBMS\_OUTPUT을 활성화합니다.

```
anonymous block completed
6 employee(s) work for department number 30
```

7. 사원의 급여를 저장하는 sal이라는 변수를 선언하는 PL/SQL 블록을 작성합니다.  
프로그램의 실행 부분에서 다음을 수행합니다.
  - a. 사원 이름을 대체 변수에 저장합니다.
  - b. sal 변수에 사원의 급여를 저장합니다.
  - c. 급여가 3,000 미만인 경우 급여를 500 인상하고 window에 "<Employee Name>'s salary updated" 메시지를 표시합니다.
  - d. 급여가 3,000 이상인 경우 "<Employee Name> earns ....." 형식으로 사원의 급여를 출력합니다.
  - e. 다음 성에 대해 PL/SQL 블록을 테스트합니다.

| LAST_NAME | SALARY |
|-----------|--------|
| Pataballa | 4800   |
| Greenberg | 12000  |
| Ernst     | 6000   |

**주:** 스크립트 끝에서 사원 이름을 저장하는 변수의 정의를 해제하십시오.

## 추가 연습 8 및 9

8. 사원의 급여를 대체 변수에 저장하는 PL/SQL 블록을 작성합니다. 프로그램의 실행 부분에서 다음을 수행합니다.
- 급여에 12를 곱하여 연봉을 계산합니다.
  - 아래와 같이 상여금을 계산합니다.

| 연봉              | 상여금   |
|-----------------|-------|
| >= 20,000       | 2,000 |
| 19,999 - 10,000 | 1,000 |
| <= 9,999        | 500   |

- 다음과 같은 형식으로 상여금 액수를 window에 표시합니다.  
"The bonus is \$......"
- 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

| SALARY | BONUS |
|--------|-------|
| 5000   | 2000  |
| 1000   | 1000  |
| 15000  | 2000  |

**주:** 이 연습은 복합 데이터 유형, 커서 및 예외 처리를 다루는 방법을 설명할 때 추가 연습으로 사용될 수 있습니다.

9. a. lab\_ap\_09\_a.sql 스크립트를 실행하여 emp라는 임시 테이블을 생성합니다. 사원 번호, 새 부서 번호 및 급여 인상률을 대체 변수에 저장하는 PL/SQL 블록을 작성합니다.
- b. 사원의 부서 ID를 새 부서 번호로 갱신하고 급여를 새 급여로 갱신합니다. 갱신에 emp 테이블을 사용합니다. 갱신이 완료된 후에는 "Update complete" 메시지가 window에 표시됩니다. 일치하는 레코드를 찾지 못하면 "No Data Found"를 표시합니다. 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

| EMPLOYEE_ID | NEW_DEPARTMENT_ID | % INCREASE | MESSAGE         |
|-------------|-------------------|------------|-----------------|
| 100         | 20                | 2          | Update Complete |
| 10          | 30                | 5          | No Data found   |
| 126         | 40                | 3          | Update Complete |

## 추가 연습 10 및 11

10. employees 테이블에서 사원 이름, 급여 및 채용 날짜를 선택하도록 EMP\_CUR 커서를 선언하는 PL/SQL 블록을 생성합니다. 커서에서 각 행을 처리하고 급여가 15,000 이상이고 채용 날짜가 01-FEB-1988 이전인 경우 아래의 샘플 출력과 같은 형식으로 window에 사원 이름, 급여 및 채용 날짜를 표시합니다.

```
anonymous block completed
Kochhar earns 17000 and joined the organization on 21-SEP-89
De Haan earns 17000 and joined the organization on 13-JAN-93
```

11. EMPLOYEE\_ID가 114 미만인 사원에 대해 employees 테이블에서 각 사원의 성 및 부서 ID를 검색하는 PL/SQL 블록을 생성합니다. employees 테이블에서 검색된 값으로 사원의 성에 대한 레코드를 저장하는 PL/SQL 테이블과 사원의 부서 ID에 대한 레코드를 저장하는 PL/SQL 테이블을 채웁니다. 루프를 사용하여 PL/SQL 테이블에서 사원 이름 정보와 급여 정보를 검색하고 DBMS\_OUTPUT.PUT\_LINE을 사용하여 window에 표시합니다. PL/SQL 테이블에 있는 처음 15명의 사원에 대한 세부 정보를 표시합니다.

```
anonymous block completed
Employee Name: King Department_id: 90
Employee Name: Kochhar Department_id: 90
Employee Name: De Haan Department_id: 90
Employee Name: Hunold Department_id: 60
Employee Name: Ernst Department_id: 60
Employee Name: Austin Department_id: 60
Employee Name: Pataballa Department_id: 60
Employee Name: Lorentz Department_id: 60
Employee Name: Greenberg Department_id: 100
Employee Name: Faviet Department_id: 100
Employee Name: Chen Department_id: 100
Employee Name: Sciarra Department_id: 100
Employee Name: Urman Department_id: 100
Employee Name: Popp Department_id: 100
Employee Name: Raphaely Department_id: 30
```

## 추가 연습 12, 13 및 14

12. a. DATE\_CUR라는 커서를 선언하는 PL/SQL 블록을 생성합니다. DATE 데이터 유형의 파라미터를 이 커서에 전달하고 해당 날짜 이후에 입사한 모든 사원에 대한 상세 정보를 출력합니다.

```
DEFINE P_HIREDATE = 08-MAR-00
```

- b. 채용 날짜 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99에 대해 PL/SQL 블록을 테스트합니다.

```
anonymous block completed
166 Aude 24-MAR-00
167 Banda 21-APR-00
173 Kumar 21-APR-00
```

13. lab\_ap\_09\_a.sql 스크립트를 실행하여 emp 테이블을 재생성합니다. 급여가 3,000을 초과하는 사원을 SR CLERK로 승진시키고 급여를 10% 인상하는 PL/SQL 블록을 생성합니다. 이 연습에는 EMP 테이블을 사용합니다. emp 테이블을 query하여 결과를 확인합니다.

**힌트:** 커서를 FOR UPDATE 및 CURRENT OF 구문과 함께 사용합니다.

14. a. 아래 연습의 경우 결과를 저장하는 테이블이 필요합니다. analysis 테이블을 직접 생성하거나 해당 테이블을 자동으로 생성할 lab\_ap\_14\_a.sql 스크립트를 실행할 수 있습니다. 다음 세 개의 열을 가진 analysis라는 테이블을 생성합니다.

| 열 이름         | ENAME    | YEARS  | SAL    |
|--------------|----------|--------|--------|
| 키 유형         |          |        |        |
| Nulls/Unique |          |        |        |
| FK 테이블       |          |        |        |
| FK 열         |          |        |        |
| 데이터 유형       | VARCHAR2 | Number | Number |
| 길이           | 20       | 2      | 8,2    |

- b. analysis 테이블을 employees 테이블의 정보로 채우는 PL/SQL 블록을 생성합니다. 대체 변수를 사용하여 사원의 성을 저장합니다.

### 추가 연습 12, 13 및 14(계속)

- c. employees 테이블을 query하여 사원의 근무 연수가 5년 이상이고 급여가 3,500 이하일 경우 예외가 발생하는지 확인합니다. 사원의 성, 근무 연수 및 현재 급여를 analysis 테이블에 삽입하는 적합한 예외 처리기로 예외를 처리합니다. 그렇지 않으면 Not due for a raise를 window에 표시합니다. analysis 테이블을 query하여 결과를 확인합니다. 다음 테스트를 사용하여 PL/SQL 블록을 테스트합니다.

| LAST_NAME | MESSAGE             |
|-----------|---------------------|
| Austin    | Not due for a raise |
| Nayer     | Not due for a raise |
| Fripp     | Not due for a raise |
| Khoo      | Due for a raise     |

---

## 추가 연습: 해답

---

## 추가 연습 1: 해답

1. 다음에 나열된 선언을 검토하여 그 중 잘못된 선언을 판별하고 그 이유를 설명합니다.

a.

```
DECLARE
 name,dept VARCHAR2(14);
```

각 선언마다 하나의 식별자만 허용되기 때문에 잘못된 선언입니다.

b.

```
DECLARE
 test NUMBER(5);
```

올바른 선언입니다.

c.

```
DECLARE
 MAXSALARY NUMBER(7,2) = 5000;
```

할당 연산자가 틀리기 때문에 잘못된 선언입니다. :=를 사용해야 합니다.

d.

```
DECLARE
 JOINDATE BOOLEAN := SYSDATE;
```

데이터 유형이 일치하지 않기 때문에 잘못된 선언입니다. Boolean 데이터 유형에는 날짜 값을 할당할 수 없습니다. Date 데이터 유형을 사용해야 합니다.

## 추가 연습 2: 해답

2. 다음 각 할당에서 결과 표현식의 데이터 유형을 파악합니다.

a. `email := firstname || to_char(empno);`

**Character string**

b. `confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');`

**Date**

c. `sal := (1000*12) + 500`

**Number**

d. `test := FALSE;`

**Boolean**

e. `temp := temp1 < (temp2/ 3);`

**Boolean**

f. `var := sysdate;`

**Date**



### 추가 연습 3: 해답

```
DECLARE
 custid NUMBER(4) := 1600;
 custname VARCHAR2(300) := 'Women Sports Club';
 new_custid NUMBER(3) := 500;
BEGIN
 DECLARE
 custid NUMBER(4) := 0;
 custname VARCHAR2(300) := 'Shape up Sports Club';
 new_custid NUMBER(3) := 300;
 new_custname VARCHAR2(300) := 'Jansports Club';
 BEGIN
 custid := new_custid;
 custname := custname || ' ' || new_custname;
 END;
 custid := (custid *12) / 10;
END;
/
```

3. 위에 제시된 PL/SQL 블록을 검토하여 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형 및 값을 판별합니다.

a. 위치 1에서의 CUSTID 값:

**300. 데이터 유형은 NUMBER**

b. 위치 1에서의 CUSTNAME 값:

**Shape up Sports Club Jansports Club. 데이터 유형은 VARCHAR2**

c. 위치 1에서의 NEW\_CUSTID 값:

**500. 데이터 유형은 NUMBER(또는 INTEGER)**

d. 위치 1에서의 NEW\_CUSTNAME 값:

**Jansports Club. 데이터 유형은 VARCHAR2**

e. 위치 2에서의 CUSTID 값:

**1920. 데이터 유형은 NUMBER**

f. 위치 2에서의 CUSTNAME 값:

**Women Sports Club. 데이터 유형은 VARCHAR2**

## 추가 연습 4: 해답

4. 연도를 받아들이는 PL/SQL 블록을 작성하고 연도가 윤년인지 여부를 확인합니다. 예를 들어, 입력한 연도가 1990 이면 "1990 is not a leap year"가 출력되어야 합니다.

**힌트:** 윤년은 4로 정확히 나누어 떨어지며 100의 배수는 아닙니다. 그렇지만 400의 배수는 윤년입니다.

다음 연도로 솔루션을 테스트합니다.

|      |                 |
|------|-----------------|
| 1990 | Not a leap year |
| 2000 | Leap year       |
| 1996 | Leap year       |
| 1886 | Not a leap year |
| 1992 | Leap year       |
| 1824 | Leap year       |

```
SET SERVEROUTPUT ON
DECLARE
 YEAR NUMBER(4) := &P_YEAR;
 REMAINDER1 NUMBER(5,2);
 REMAINDER2 NUMBER(5,2);
 REMAINDER3 NUMBER(5,2);
BEGIN
 REMAINDER1 := MOD(YEAR,4);
 REMAINDER2 := MOD(YEAR,100);
 REMAINDER3 := MOD(YEAR,400);
 IF ((REMAINDER1 = 0 AND REMAINDER2 <> 0)
 OR REMAINDER3 = 0) THEN
 DBMS_OUTPUT.PUT_LINE(YEAR || ' is a leap year');
 ELSE
 DBMS_OUTPUT.PUT_LINE (YEAR || ' is not a leap year');
 END IF;
END;
/
```

## 추가 연습 5: 해답

5. a. 아래 연습의 경우 결과를 저장할 임시 테이블이 필요합니다.

테이블을 직접 생성하거나 테이블을 자동으로 생성할 lab\_ap\_05.sql 스크립트를 실행할 수 있습니다. 다음 세 개의 열을 가진 TEMP라는 테이블을 생성합니다.

| 열 이름         | NUM_STORE | CHAR_STORE | DATE_STORE |
|--------------|-----------|------------|------------|
| 키 유형         |           |            |            |
| Nulls/Unique |           |            |            |
| FK 테이블       |           |            |            |
| FK 열         |           |            |            |
| 데이터 유형       | Number    | VARCHAR2   | Date       |
| 길이           | 7,2       | 35         |            |

```
CREATE TABLE temp
(num_store NUMBER(7,2),
char_store VARCHAR2(35),
date_store DATE);
```

- b. MESSAGE 및 DATE\_WRITTEN이라는 2개의 변수를 포함한 PL/SQL 블록을 작성합니다. 길이가 35인 VARCHAR2 데이터 유형으로 MESSAGE를 선언하고 DATE 데이터 유형으로 DATE\_WRITTEN을 선언합니다. 다음 값을 해당 변수에 할당합니다.

| 변수           | 내용                              |
|--------------|---------------------------------|
| MESSAGE      | This is my first PL/SQL program |
| DATE_WRITTEN | Current date                    |

TEMP 테이블의 해당 열에 값을 저장합니다. TEMP 테이블을 query 하여 결과를 확인합니다.

```
SET SERVEROUTPUT ON
DECLARE
MESSAGE VARCHAR2(35);
DATE_WRITTEN DATE;
BEGIN
MESSAGE := 'This is my first PLSQL Program';
DATE_WRITTEN := SYSDATE;
INSERT INTO temp(char_store,date_store)
VALUES (MESSAGE,DATE_WRITTEN);
END;
/
SELECT * FROM TEMP;
```

## 추가 연습 6: 해답

6. a. 부서 번호를 대체 변수에 저장합니다.

```
DEFINE P_DEPTNO = 30
```

- b. 해당 부서에서 근무하는 직원 수를 출력하는 PL/SQL 블록을 작성합니다.

**힌트:** SET SERVEROUTPUT ON을 사용하여 DBMS\_OUTPUT을 활성화합니다.

```
SET SERVEROUTPUT ON
DECLARE
 HOWMANY NUMBER(3);
 DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
BEGIN
 SELECT COUNT(*) INTO HOWMANY FROM employees
 WHERE department_id = DEPTNO;
 DBMS_OUTPUT.PUT_LINE (HOWMANY || ' employee(s) work for
department number ' || DEPTNO);
END;
/
SET SERVEROUTPUT OFF
```

## 추가 연습 7: 해답

7. 사원의 급여를 저장하는 sal 이라는 변수를 선언하는 PL/SQL 블록을 작성합니다. 프로그램의 실행 부분에서 다음을 수행합니다.
- 사원 이름을 대체 변수에 저장합니다.  
SET SERVEROUTPUT ON  
DEFINE P\_LASTNAME = Pataballa
  - sal 변수에 해당 사원의 급여를 저장합니다.
  - 급여가 3,000 미만인 경우 급여를 500 인상하고 window에 "<Employee Name>'s salary updated" 메시지를 표시합니다.
  - 급여가 3,000을 초과할 경우 "<Employee Name> earns ..... " 형식으로 사원의 급여를 출력합니다.
  - 성에 대해 PL/SQL 블록을 테스트합니다.

| LAST_NAME | SALARY |
|-----------|--------|
| Pataballa | 4800   |
| Greenberg | 12000  |
| Ernst     | 6000   |

**주:** 스크립트 끝에서 사원 이름을 저장하는 변수의 정의를 해제하십시오.

```

DECLARE
 SAL NUMBER(7,2);
 LASTNAME EMPLOYEES.LAST_NAME%TYPE;
BEGIN
 SELECT salary INTO SAL
 FROM employees
 WHERE last_name = INITCAP('&P_LASTNAME') FOR UPDATE of
 salary;

 LASTNAME := INITCAP('&P_LASTNAME');
 IF SAL < 3000 THEN
 UPDATE employees SET salary = salary + 500
 WHERE last_name = INITCAP('&P_LASTNAME') ;
 DBMS_OUTPUT.PUT_LINE (LASTNAME || ''s salary
updated');
 ELSE
 DBMS_OUTPUT.PUT_LINE (LASTNAME || ' earns ' ||
TO_CHAR(SAL));
 END IF;
END;
/
SET SERVEROUTPUT OFF
UNDEFINE P_LASTNAME

```

## 추가 연습 8: 해답

8. 사원의 급여를 대체 변수에 저장하는 PL/SQL 블록을 작성합니다. 프로그램의 실행 부분에서 다음을 수행합니다.

- 급여에 12를 곱하여 연봉을 계산합니다.
- 아래와 같이 상여금을 계산합니다.

| 연봉              | 상여금   |
|-----------------|-------|
| >= 20,000       | 2,000 |
| 19,999 - 10,000 | 1,000 |
| <= 9,999        | 500   |

- 다음과 같은 형식으로 상여금 액수를 window에 표시합니다.  
"The bonus is \$....."
- 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

| SALARY | BONUS |
|--------|-------|
| 5000   | 2000  |
| 1000   | 1000  |
| 15000  | 2000  |

```

SET SERVEROUTPUT ON
DEFINE P_SALARY = 5000
DECLARE
 SAL NUMBER(7,2) := &P_SALARY;
 BONUS NUMBER(7,2);
 ANN_SALARY NUMBER(15,2);
BEGIN
 ANN_SALARY := SAL * 12;
 IF ANN_SALARY >= 20000 THEN
 BONUS := 2000;
 ELSIF ANN_SALARY <= 19999 AND ANN_SALARY >= 10000 THEN
 BONUS := 1000;
 ELSE
 BONUS := 500;
 END IF;
 DBMS_OUTPUT.PUT_LINE ('The Bonus is $ ' ||
 TO_CHAR(BONUS));
END;
/
SET SERVEROUTPUT OFF

```

## 추가 연습 9: 해답

9. a. lab\_ap\_09\_a.sql 스크립트를 실행하여 emp 라는 임시 테이블을 생성합니다. 사원 번호, 새 부서 번호 및 급여 인상률을 대체 변수에 저장하는 PL/SQL 블록을 작성합니다.

```
SET SERVEROUTPUT ON
DEFINE P_EMPNO = 100
DEFINE P_NEW_DEPTNO = 10
DEFINE P_PER_INCREASE = 2
```

- b. 사원의 부서 ID를 새 부서 번호로 갱신하고 급여를 새 급여로 갱신합니다. 갱신에 emp 테이블을 사용합니다. 갱신이 완료된 후에는 "Update complete" 메시지가 window에 표시됩니다. 일치하는 레코드를 찾을 수 없으면 "No Data Found" 메시지를 표시합니다. 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

| EMPLOYEE_ID | NEW_DEPARTMENT_ID | % INCREASE | MESSAGE         |
|-------------|-------------------|------------|-----------------|
| 100         | 20                | 2          | Update Complete |
| 10          | 30                | 5          | No Data found   |
| 126         | 40                | 3          | Update Complete |

```
DECLARE
 EMPNO emp.EMPLOYEE_ID%TYPE := &P_EMPNO;
 NEW_DEPTNO emp.DEPARTMENT_ID%TYPE := &P_NEW_DEPTNO;
 PER_INCREASE NUMBER(7,2) := &P_PER_INCREASE;
BEGIN
 UPDATE emp
 SET department_id = NEW_DEPTNO,
 salary = salary + (salary * PER_INCREASE/100)
 WHERE employee_id = EMPNO;
 IF SQL%ROWCOUNT = 0 THEN
 DBMS_OUTPUT.PUT_LINE ('No Data Found');
 ELSE
 DBMS_OUTPUT.PUT_LINE ('Update Complete');
 END IF;
END;
/
SET SERVEROUTPUT OFF
```

## 추가 연습 10: 해답

10. employees 테이블에서 사원 이름, 급여 및 채용 날짜를 선택하도록 EMP\_CUR 커서를 선언하는 PL/SQL 블록을 생성합니다. 커서에서 각 행을 처리하고 급여가 15,000 이상이고 채용 날짜가 01-FEB-1988 이전인 경우 window 에 사원 이름, 급여 및 채용 날짜를 표시합니다.

```
SET SERVEROUTPUT ON
DECLARE
 CURSOR EMP_CUR IS
 SELECT last_name,salary,hire_date FROM EMPLOYEES;
 ENAME VARCHAR2(25);
 SAL NUMBER(7,2);
 HIREDATE DATE;
BEGIN
 OPEN EMP_CUR;
 FETCH EMP_CUR INTO ENAME,SAL,HIREDATE;
 WHILE EMP_CUR%FOUND
 LOOP
 IF SAL > 15000 AND HIREDATE >= TO_DATE('01-FEB-1988','DD-MON-
 YYYY') THEN
 DBMS_OUTPUT.PUT_LINE (ENAME || ' earns ' || TO_CHAR(SAL) || '
 and joined the organization on ' || TO_DATE(HIREDATE,'DD-
 Mon-YYYY'));
 END IF;
 FETCH EMP_CUR INTO ENAME,SAL,HIREDATE;
 END LOOP;
 CLOSE EMP_CUR;
END;
/
SET SERVEROUTPUT OFF
```



## 추가 연습 11: 해답

11. EMPLOYEE\_ID 가 114 미만인 사원에 대해 employees 테이블에서 각 사원의 성 및 부서 ID를 검색하는 PL/SQL 블록을 생성합니다. employees 테이블에서 검색된 값으로 사원의 성에 대한 레코드를 저장하는 PL/SQL 테이블과 사원의 부서 ID에 대한 레코드를 저장하는 PL/SQL 테이블을 채웁니다. 루프를 사용하여 PL/SQL 테이블에서 사원 이름 정보와 급여 정보를 검색하고 DBMS\_OUTPUT.PUT\_LINE 을 사용하여 window 에 표시합니다. PL/SQL 테이블에 있는 처음 15 명의 사원에 대한 세부 정보를 표시합니다.

```
SET SERVEROUTPUT ON
DECLARE
 TYPE Table_Ename is table of employees.last_name%TYPE
 INDEX BY BINARY_INTEGER;
 TYPE Table_dept is table of employees.department_id%TYPE
 INDEX BY BINARY_INTEGER;
 Tename Table_Ename;
 Tdept Table_dept;
 i BINARY_INTEGER :=0;
 CURSOR Namedept IS SELECT last_name,department_id from employees
WHERE employee_id < 115;
 TRACK NUMBER := 15;
BEGIN
 FOR emprec in Namedept
 LOOP
 i := i +1;
 Tename(i) := emprec.last_name;
 Tdept(i) := emprec.department_id;
 END LOOP;
 FOR i IN 1..TRACK
 LOOP
 DBMS_OUTPUT.PUT_LINE ('Employee Name: ' ||
Tename(i) || ' Department_id: ' || Tdept(i));
 END LOOP;
END;
/
SET SERVEROUTPUT OFF
```

## 추가 연습 12: 해답

12. a DATE\_CUR 라는 커서를 선언하는 PL/SQL 블록을 생성합니다. DATE 데이터 유형의 파라미터를 이 커서에 전달하고 해당 날짜 이후에 입사한 모든 사원에 대한 상세 정보를 출력합니다.

```
SET SERVEROUTPUT ON
DEFINE P_HIREDATE = 08-MAR-00
```

- b. 채용 날짜 08-MAR-00, 25-JUN-97, 28-SEP-98, 07-FEB-99에 대해 PL/SQL 블록을 테스트합니다.

```
DECLARE
 CURSOR DATE_CURSOR(JOIN_DATE DATE) IS
 SELECT employee_id,last_name,hire_date FROM employees
 WHERE HIRE_DATE >JOIN_DATE ;
 EMPNO employees.employee_id%TYPE;
 ENAME employees.last_name%TYPE;
 HIREDATE employees.hire_date%TYPE;
 HDATE employees.hire_date%TYPE := '&P_HIREDATE';
BEGIN
 OPEN DATE_CURSOR(HDATE);
 LOOP
 FETCH DATE_CURSOR INTO EMPNO,ENAME,HIREDATE;
 EXIT WHEN DATE_CURSOR%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE (EMPNO || ' ' || ENAME || ' ' ||
 HIREDATE);
 END LOOP;
END;
/
SET SERVEROUTPUT OFF;
```

### 추가 연습 13: 해답

13. lab\_ap\_09\_a.sql 스크립트를 실행하여 emp 테이블을 다시 생성합니다. 급여가 3,000 이상인 사원을 SR CLERK 로 승진시키고 급여를 10% 인상하는 PL/SQL 블록을 생성합니다. 이 연습에는 emp 테이블을 사용합니다. emp 테이블을 query 하여 결과를 확인합니다.

**힌트:** 커서를 FOR UPDATE 및 CURRENT OF 구문과 함께 사용합니다.

```
DECLARE
 CURSOR Senior_Clerk IS
 SELECT employee_id, job_id FROM emp
 WHERE job_id = 'ST_CLERK' AND salary > 3000
 FOR UPDATE OF job_id;
BEGIN
 FOR Emrec IN Senior_Clerk
 LOOP
 UPDATE emp
 SET job_id = 'SR_CLERK',
 salary = 1.1 * salary
 WHERE CURRENT OF Senior_Clerk;
 END LOOP;
 COMMIT;
END;
/
SELECT * FROM emp;
```

## 추가 연습 14: 해답

14. a. 아래 연습의 경우 결과를 저장할 테이블이 필요합니다. `analysis` 테이블을 직접 생성하거나 테이블을 자동으로 생성하는 `lab_ap_14_a.sql` 스크립트를 실행할 수 있습니다. 다음 세 개의 열을 가진 `analysis` 라는 테이블을 생성합니다.

| 열 이름         | ENAME    | YEARS  | SAL    |
|--------------|----------|--------|--------|
| 키 유형         |          |        |        |
| Nulls/Unique |          |        |        |
| FK 테이블       |          |        |        |
| FK 열         |          |        |        |
| 데이터 유형       | VARCHAR2 | Number | Number |
| 길이           | 20       | 2      | 8,2    |

```
CREATE TABLE analysis
 (ename Varchar2(20),
 years Number(2),
 sal Number(8,2));
```

- b. `employees` 테이블에 있는 정보로 `analysis` 테이블을 채우는 PL/SQL 블록을 생성합니다. 대체 변수를 사용하여 사원의 성을 저장합니다.

```
SET SERVEROUTPUT ON
DEFINE P_ENAME = Austin
```

## 추가 연습 14: 해답(계속)

- c. employees 테이블을 query하여 사원의 근무 연수가 5년 이상이고 급여가 3,500 이하일 경우 예외가 발생하는지 확인합니다. 사원의 성, 근무 연수 및 현재 급여를 analysis 테이블에 삽입하는 적합한 예외 처리기로 예외를 처리합니다. 그렇지 않으면 window에 Not due for a raise를 표시합니다. analysis 테이블을 query하여 결과를 확인합니다. 다음 테스트를 사용하여 PL/SQL 블록을 테스트합니다.

| LAST_NAME | MESSAGE             |
|-----------|---------------------|
| Austin    | Not due for a raise |
| Nayer     | Not due for a raise |
| Fripp     | Not due for a raise |
| Khoo      | Due for a raise     |

```
DECLARE
 DUE_FOR_RAISE EXCEPTION;
 HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
 ENAME EMPLOYEES.LAST_NAME%TYPE := INITCAP('& P_ENAME');
 SAL EMPLOYEES.SALARY%TYPE;
 YEARS NUMBER(2);
BEGIN
 SELECT LAST_NAME, SALARY, HIRE_DATE
 INTO ENAME, SAL, HIREDATE
 FROM employees WHERE last_name = ENAME;
 YEARS := MONTHS_BETWEEN(SYSDATE, HIREDATE) / 12;
 IF SAL < 3500 AND YEARS > 5 THEN
 RAISE DUE_FOR_RAISE;
 ELSE
 DBMS_OUTPUT.PUT_LINE ('Not due for a raise');
 END IF;
EXCEPTION
 WHEN DUE_FOR_RAISE THEN
 INSERT INTO ANALYSIS(ENAME, YEARS, SAL)
 VALUES (ENAME, YEARS, SAL);
END;
/
```