

Oracle Database 11g: PL/SQL Fundamentals(한글판)

학생용

D49990KR20

Edition 2.0

2009년 12월

D64314

ORACLE®

저자

Brian Pottle

기술 제공자 및 검토자

Tom Best

Christoph Burandt

Yanti Chang

Laszlo Czinkoczki

Ashita Dhir

Peter Driver

Gerlinde Frenzen

Nancy Greenberg

Chaitanya Kortamaddi

Tim Leblanc

Bryan Roberts

Abhishek X Singh

Puja Singh

Lex Van Der Werff

Copyright © 2009, Oracle. All rights reserved.

Disclaimer

본 문서는 독점적 정보를 포함하고 있으며 저작권법 및 기타 지적 재산법에 의해 보호됩니다. 본 문서는 오라클 교육 과정에서 자신이 사용할 목적으로만 복사하고 인쇄할 수 있습니다. 어떤 방법으로도 본 문서를 수정하거나 변경할 수 없습니다. 저작권법에 따라 "공정"하게 사용하는 경우를 제외하고, 오라클의 명시적 허가 없이 본 문서의 전체 또는 일부를 사용, 공유, 다운로드, 업로드, 복사, 인쇄, 표시, 실행, 재생산, 게시, 라이센스, 우편 발송, 전송 또는 배포할 수 없습니다.

본 문서의 내용은 사전 공지 없이 변경될 수 있습니다. 만일 본 문서의 내용상 문제점을 발견하면 서면으로 통지해 주기 바랍니다. Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. 오라클은 본 문서에 오류가 존재하지 않음을 보증하지 않습니다.

Restricted Rights Notice

만일 본 문서를 미국 정부나 또는 미국 정부를 대신하여 문서를 사용하는 개인이나 법인에게 배송하는 경우, 다음 공지 사항이 적용됩니다.

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle은 Oracle Corporation 또는 그 자회사의 등록 상표입니다. 기타의 명칭들은 각 해당 명칭을 소유한 회사의 상표일 수 있습니다.

그래픽 디자이너

Satish Bettegowda

편집자

Vijayalakshmi Narasimhan

Daniel Milne

발행인

Jobi Varghese

목차

I 소개

- 단원 목표 I-2
- 과정 목표 I-3
- 본 과정에서 사용하는 HR(Human Resources) 스키마 I-4
- 과정 일정 I-5
- 클래스 계정 정보 I-6
- 본 과정에 사용되는 부록 I-7
- PL/SQL 개발 환경 I-8
- Oracle SQL Developer 란? I-9
- SQL*Plus에서 PL/SQL 코딩 I-10
- Oracle JDeveloper에서 PL/SQL 코딩 I-11
- Oracle 11g SQL 및 PL/SQL 설명서 I-12
- 요약 I-13
- 연습 1: 개요: 시작하기 I-14

1 PL/SQL 소개

- 목표 1-2
- 다루는 내용 1-3
- PL/SQL 정보 1-4
- PL/SQL 런타임 구조 1-6
- PL/SQL의 이점 1-7
- PL/SQL 블록 구조 1-10
- 다루는 내용 1-12
- 블록 유형 1-13
- 프로그램 생성자 1-15
- 익명 블록 검사 1-17
- 익명 블록 실행 1-18
- 다루는 내용 1-19
- PL/SQL 블록의 출력 활성화 1-20
- PL/SQL 블록의 출력 보기 1-21
- 퀴즈 1-22
- 요약 1-23
- 연습 1: 개요 1-24

2 PL/SQL 변수 선언

목표	2-2
다루는 내용	2-3
변수 사용	2-4
변수 이름에 대한 요구 사항	2-5
PL/SQL에서 변수 처리	2-6
PL/SQL 변수 선언 및 초기화	2-7
문자열 리터럴의 구분자	2-9
다루는 내용	2-10
변수 유형	2-11
PL/SQL 변수 선언 및 초기화 지침	2-13
PL/SQL 변수 선언 지침	2-14
이 과정에서 사용하는 PL/SQL 구조의 이름 지정 규칙	2-15
스칼라 데이터 유형	2-16
기본 스칼라 데이터 유형	2-17
스칼라 변수 선언	2-21
%TYPE 속성	2-22
%TYPE 속성을 사용하여 변수 선언	2-24
부울 변수 선언	2-25
LOB 데이터 유형 변수	2-26
조합 데이터 유형: 레코드 및 컬렉션	2-27
다루는 내용	2-28
바인드 변수	2-29
바인드 변수 참조	2-31
바인드 변수와 함께 AUTOPRINT 사용	2-32
퀴즈	2-33
요약	2-34
연습 2: 개요	2-35

3 실행문 작성

목표	3-2
다루는 내용	3-3
PL/SQL 블록의 렉시칼 단위	3-4
PL/SQL 블록 구문 및 작성 지침	3-6
코드 주석 처리	3-7
PL/SQL의 SQL 함수	3-8
PL/SQL의 SQL 함수: 예제	3-9
PL/SQL 표현식에서 시퀀스 사용	3-10
데이터 유형 변환	3-11

다루는 내용 3-13
중첩 블록 3-14
중첩 블록 예제 3-15
변수 범위 및 가시성 3-16
중첩 블록에 수식자 사용 3-18
문제: 변수 범위 결정 3-19
다루는 내용 3-21
PL/SQL의 연산자 3-22
PL/SQL의 연산자: 예제 3-23
프로그래밍 지침 3-24
코드 들여쓰기 3-25
퀴즈 3-26
요약 3-27
연습 3: 개요 3-28

4 오라클 데이터베이스 서버와 상호 작용: PL/SQL 프로그램의 SQL 문

목표 4-2
다루는 내용 4-3
PL/SQL의 SQL 문 4-4
PL/SQL의 SELECT 문 4-5
PL/SQL을 사용하여 데이터 검색: 예제 4-9
PL/SQL을 사용하여 데이터 검색 4-10
이름 지정 모호성 4-11
이름 지정 규칙 4-12
다루는 내용 4-13
PL/SQL을 사용하여 데이터 조작 4-14
데이터 삽입: 예제 4-15
데이터 간신: 예제 4-16
데이터 삭제: 예제 4-17
행 병합 4-18
다루는 내용 4-20
SQL 커서 4-21
암시적 커서에 대한 SQL 커서 속성 4-22
퀴즈 4-24
요약 4-25
연습 4: 개요 4-26

5 제어 구조 작성

목표 5-2

실행 흐름 제어 5-3

다루는 내용 5-4

IF 문 5-5

간단한 IF 문 5-7

IF THEN ELSE 문 5-8

IF ELSIF ELSE 절 5-9

IF 문의 NULL 값 5-10

다루는 내용 5-11

CASE 식 5-12

CASE 식: 예제 5-13

검색된 CASE 식 5-14

CASE 문 5-15

널 처리 5-16

논리 테이블 5-17

부울 식 또는 논리식? 5-18

다루는 내용 5-19

반복 제어: LOOP 문 5-20

기본 루프 5-21

기본 루프: 예제 5-22

WHILE 루프 5-23

WHILE 루프: 예제 5-24

FOR 루프 5-25

FOR 루프: 예제 5-27

FOR 루프 규칙 5-28

루프의 권장 사용법 5-29

중첩 루프 및 레이블 5-30

중첩 루프 및 레이블: 예제 5-31

PL/SQL CONTINUE 문 5-32

PL/SQL CONTINUE 문: 예제 1 5-33

PL/SQL CONTINUE 문: 예제 2 5-34

퀴즈 5-35

요약 5-36

연습 5: 개요 5-37

6 조합 데이터 유형 작업

목표 6-2

다루는 내용 6-3

조합 데이터 유형 6-4

PL/SQL 레코드 또는 컬렉션? 6-5

다루는 내용 6-6

PL/SQL 레코드 6-7

PL/SQL 레코드 생성 6-8

PL/SQL 레코드 구조 6-9

%ROWTYPE 속성 6-10

PL/SQL 레코드 생성: 예제 6-12

%ROWTYPE 속성 사용 시 이점 6-13

또 다른 %ROWTYPE 속성 예제 6-14

%ROWTYPE 을 사용하여 레코드 삽입 6-15

레코드를 사용하여 테이블의 행 간신 6-16

다루는 내용 6-17

연관 배열(INDEX BY 테이블) 6-18

연관 배열 구조 6-19

연관 배열 생성 단계 6-20

연관 배열 생성 및 액세스 6-21

INDEX BY 테이블 메소드 사용 6-22

INDEX BY 레코드 테이블 옵션 6-23

INDEX BY 레코드 테이블 옵션: 예제 2 6-24

중첩 테이블 6-25

VARRAY 6-27

컬렉션 유형 요약 6-28

퀴즈 6-29

요약 6-30

연습 6: 개요 6-31

7 명시적 커서 사용

목표 7-2

다루는 내용 7-3

커서 7-4

명시적 커서 작업 7-5

명시적 커서 제어 7-6

다루는 내용 7-8

커서 선언 7-9

커서 열기 7-11

커서에서 데이터 패치(fetch)	7-12
커서 닫기	7-15
커서 및 레코드	7-15
커서 FOR 루프	7-16
명시적 커서 속성	7-18
%ISOPEN 속성	7-19
%ROWCOUNT 및 %NOTFOUND: 예제	7-20
subquery 를 사용하는 커서 FOR 루프	7-21
다루는 내용	7-22
파라미터가 포함된 커서	7-23
다루는 내용	7-25
FOR UPDATE 절	7-26
WHERE CURRENT OF 절	7-28
퀴즈	7-29
요약	7-30
연습 7: 개요	7-31

8 예외 처리

목표	8-2
다루는 내용	8-3
예외란?	8-4
예외 처리: 예제	8-5
PL/SQL 예외 이해	8-6
예외 처리	8-7
예외 유형	8-8
다루는 내용	8-9
예외 트랩 구문	8-10
예외 트랩에 대한 지침	8-12
미리 정의된 Oracle 서버 오류 트랩	8-13
미리 정의되지 않은 Oracle 서버 오류 트랩	8-16
미리 정의되지 않은 오류 트랩: 예제	8-17
예외 트랩에 대한 함수	8-18
유저 정의 예외 트랩	8-20
서브 블록의 예외 전달	8-22
RAISE_APPLICATION_ERROR 프로시저	8-23
퀴즈	8-26
요약	8-27
연습 8: 개요	8-28

9 내장 프로시저 및 함수 소개

목표 9-2

다루는 내용 9-3

프로시저 및 함수 9-4

익명 블록과 서브 프로그램의 차이 9-5

다루는 내용 9-6

프로시저: 구문 9-7

프로시저 작성 9-8

프로시저 호출 9-10

다루는 내용 9-11

함수: 구문 9-12

함수 작성 9-13

함수 호출 9-14

함수에 파라미터 전달 9-15

파라미터가 포함된 함수 호출 9-16

퀴즈 9-17

요약 9-18

연습 9: 개요 9-19

A 연습 및 해답

B 테이블 설명 및 데이터

C SQL Developer 사용

목표 C-2

Oracle SQL Developer 란? C-3

SQL Developer 사양 C-4

SQL Developer 1.5 인터페이스 C-5

데이터베이스 연결 생성 C-7

데이터베이스 객체 탐색 C-10

테이블 구조 표시 C-11

파일 탐색 C-12

스키마 객체 생성 C-13

새 테이블 생성: 예제 C-14

SQL Worksheet 사용 C-15

SQL 문 실행 C-18

SQL 스크립트 저장 C-19

저장된 SQL 스크립트 실행: 방법 1 C-20

저장된 SQL 스크립트 실행: 방법 2 C-21

SQL 코드 형식 지정	C-22
Snippet 사용	C-23
Snippet 사용: 예제	C-24
프로시저 및 함수 디버깅	C-25
데이터베이스 보고	C-26
유저 정의 보고서 작성	C-27
검색 엔진 및 External 도구	C-28
환경 설정	C-29
SQL Developer 레이아웃 재설정	C-30
요약	C-31

D SQL*Plus 사용

목표	D-2
SQL 과 SQL*Plus 의 상호 작용	D-3
SQL 문과 SQL*Plus 명령 비교	D-4
SQL*Plus: 개요	D-5
SQL*Plus 에 로그인	D-6
테이블 구조 표시	D-7
SQL*Plus 편집 명령	D-9
LIST, n 및 APPEND 사용	D-11
CHANGE 명령 사용	D-12
SQL*Plus 파일 명령	D-13
SAVE 및 START 명령 사용	D-14
SERVERROUTPUT 명령	D-15
SQL*Plus SPOOL 명령 사용	D-16
AUTOTRACE 명령 사용	D-17
요약	D-18

E JDeveloper 사용

- Oracle JDeveloper E-2
- Database Navigator E-3
- 연결 생성 E-4
- 데이터베이스 객체 탐색 E-5
- SQL 문 실행 E-6
- 프로그램 단위 생성 E-7
- 컴파일 E-8
- 프로그램 단위 실행 E-9
- 프로그램 단위 삭제 E-10
- Structure window E-11
- Editor window E-12
- Application Navigator E-13
- Java 내장 프로시저 배치 E-14
- PL/SQL에 Java 게시(publishing) E-15
- JDeveloper 11g에 대해 자세히 배울 수 있는 방법 E-16

F REF 커서

- 커서 변수 F-2
- 커서 변수 사용 F-3
- REF CURSOR 유형 정의 F-4
- OPEN-FOR, FETCH 및 CLOSE 문 사용 F-7
- 패치(fetch) 예제 F-10

추가 연습 및 해답

I

소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 과정 목표 설명
- 과정에서 사용하는 HR 데이터베이스 스키마 설명
- 본 과정에서 사용할 수 있는 유저 인터페이스 환경 식별
- 사용 가능한 부록, 설명서 및 기타 리소스 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목표

이 단원에서는 본 과정과 과정 흐름을 대략적으로 설명합니다. 본 과정에서 사용하는 데이터베이스 스키마와 테이블에 대해 알아봅니다. 또한 Oracle 11g 그리드 Infrastructure에 포함된 다양한 제품을 소개합니다.

과정 목표

이 과정을 마치면 다음을 수행할 수 있습니다.

- PL/SQL이 SQL에 제공하는 프로그래밍 확장 기능 파악
- 데이터베이스에 연결하는 PL/SQL 코드 작성
- 효율적으로 실행되는 PL/SQL 익명 블록 설계
- PL/SQL 프로그래밍 생성자 및 조건 제어문 사용
- 런타임 오류 처리
- 내장 프로시저 및 함수 설명

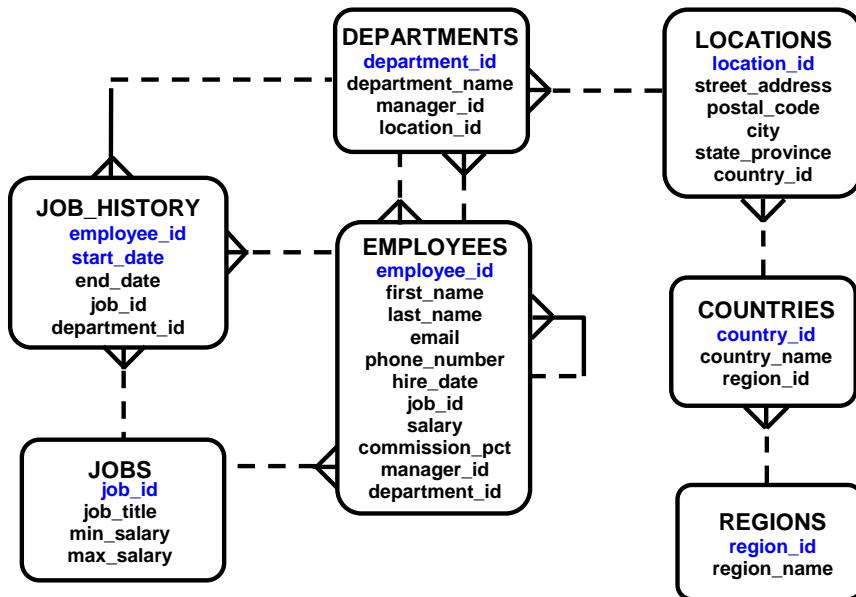
ORACLE®

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 SQL Developer를 PL/SQL 개발 환경으로 사용하는 방법을 배우고, 또한 프로시저 및 함수처럼 재사용 가능한 프로그램 단위를 설계하는 방법을 배웁니다.

본 과정에서 사용하는 HR(Human Resources) 스키마



ORACLE

Copyright © 2009, Oracle. All rights reserved.

본 과정에서 사용하는 HR(Human Resources) 스키마

Human Resources(HR) 스키마는 오라클 데이터베이스에 설치할 수 있는 Oracle 예제 스키마의 일부입니다. 본 과정의 연습 세션에서는 HR 스키마의 데이터를 사용합니다.

테이블 설명

- REGIONS에는 Americas 또는 Asia와 같은 지역을 나타내는 행이 포함되어 있습니다.
- COUNTRIES에는 국가에 대한 행이 포함되어 있으며, 각 행은 REGION과 연관되어 있습니다.
- LOCATIONS에는 특정 국가에 있는 회사의 특정 지사, 도매점, 생산지 등의 주소가 포함되어 있습니다.
- DEPARTMENTS는 사원의 소속 부서에 대한 세부 정보를 표시합니다. 각 부서에는 EMPLOYEES 테이블의 부서 관리자를 나타내는 관계가 있습니다.
- EMPLOYEES에는 특정 부서에서 근무하는 각 사원에 대한 세부 정보가 포함되어 있습니다. 부서가 할당되지 않은 사원이 있을 수도 있습니다.
- JOBS에는 각 사원이 보유할 수 있는 직무 유형이 포함되어 있습니다.
- JOB_HISTORY에는 사원의 작업 기록이 포함되어 있습니다. 사원이 직무 내에서 부서를 변경하거나 부서 내에서 직무를 변경할 경우 새 행이 해당 사원의 이전 직무 정보와 함께 이 테이블에 삽입됩니다.

과정 일정

첫째 날:

- I. 소개
1. PL/SQL 소개
2. PL/SQL 변수 선언
3. 실행문 작성
4. 오라클 데이터베이스 서버와 상호 작용: PL/SQL 프로그램의 SQL 문
5. 제어 구조 작성

둘째 날:

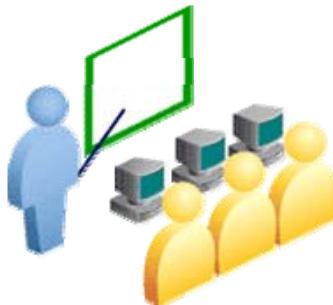
6. 조합 데이터 유형 작업
7. 명시적 커서 사용
8. 예외 처리
9. 내장 프로시저 및 함수 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

클래스 계정 정보

- 복제된 HR 계정 ID가 설정됩니다.
- 계정 ID는 ora41입니다.
- 암호는 계정 ID와 일치합니다.
- 각 시스템에는 고유의 완전한 환경이 있으며 동일한 계정이 할당됩니다.
- 강사는 별도의 ID를 가지고 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

본 과정에 사용되는 부록

- **부록 A: 연습 및 해답**
- **부록 B: 테이블 설명 및 데이터**
- **부록 C: SQL Developer 사용**
- **부록 D: SQL*Plus 사용**
- **부록 E: JDeveloper 사용**
- **부록 F: REF 커서**
- **부록 AP: 추가 연습 및 해답**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 개발 환경

이 과정 설정에서는 PL/SQL 코드를 개발하기 위한 다음 도구를 제공합니다.

- Oracle SQL Developer(본 과정에서 사용)
- Oracle SQL*Plus
- Oracle JDeveloper IDE

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 개발 환경

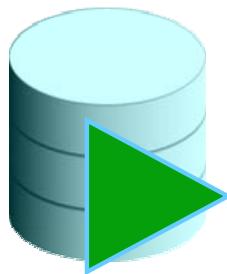
오라클은 PL/SQL 코드를 작성하는 데 사용할 수 있는 여러 도구를 제공합니다. 본 과정에서 사용할 수 있는 몇 가지 개발 툴은 다음과 같습니다.

- Oracle SQL Developer: 그래픽 도구
- Oracle SQL*Plus: window 또는 명령행 응용 프로그램
- Oracle JDeveloper: window 기반의 IDE(통합 개발 환경)

참고: 과정 참고 사항에 제시되는 코드와 화면 예제는 SQL Developer 환경에서 출력된 결과입니다.

Oracle SQL Developer란?

- Oracle SQL Developer는 생산성을 높이고 데이터베이스 개발 작업을 간소화하기 위해 무료로 제공되는 그래픽 도구입니다.
- 표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다.
- 본 과정에서는 SQL Developer를 사용합니다.
- 부록 C에는 SQL Developer 사용에 대한 세부 정보가 포함되어 있습니다.



SQL Developer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle SQL Developer란?

Oracle SQL Developer는 생산성을 높이고 일상적인 데이터베이스 작업의 개발을 간소화하기 위해 무료로 제공되며, 간단히 버튼을 몇 번 누르기만 하면 손쉽게 내장 프로시저를 생성 및 유지 관리하고, SQL 문을 테스트하고, 옵티마이저 계획을 볼 수 있습니다.

데이터베이스 개발을 위한 시각적 도구인 SQL Developer는 다음 작업을 단순화합니다.

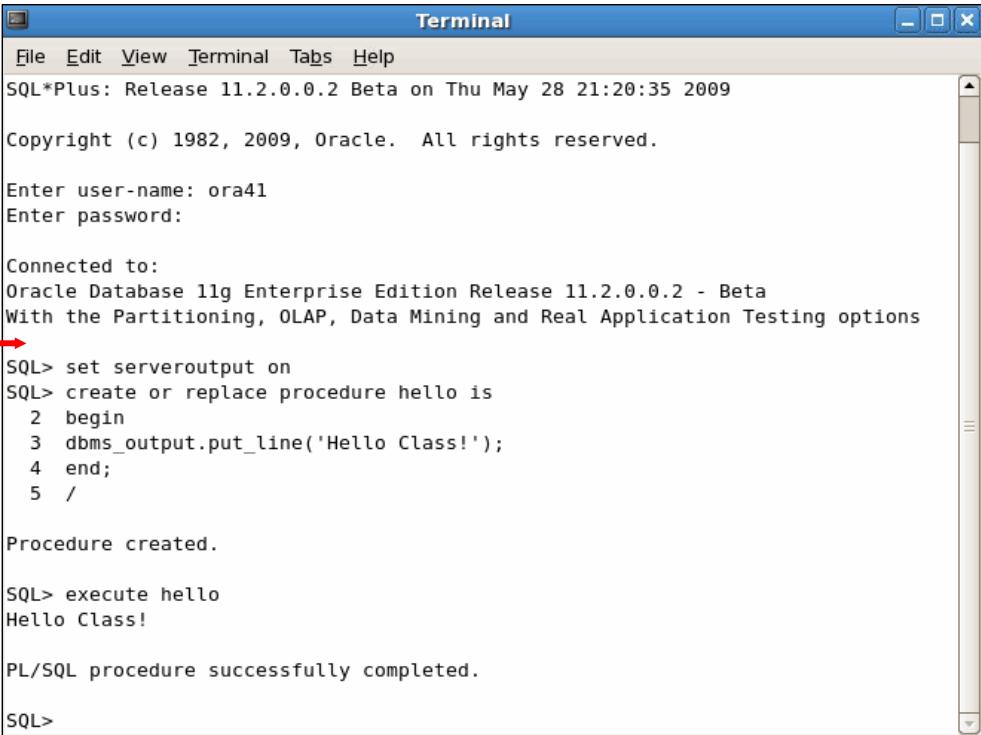
- 데이터베이스 객체 탐색 및 관리
- SQL 문 및 스크립트 실행
- PL/SQL 문 편집 및 디버깅
- 보고서 작성

표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다. 연결되면 데이터베이스의 객체에 대해 작업을 수행할 수 있습니다.

부록 C

이 과정의 부록 C에서는 SQL Developer 인터페이스 사용에 대해 소개합니다. 데이터베이스 연결 생성 및 SQL과 PL/SQL을 통한 데이터와의 상호 작용에 대한 자세한 내용은 이 부록을 참조하십시오.

SQL*Plus에서 PL/SQL 코딩



The screenshot shows a Windows-style terminal window titled "Terminal". A red arrow points from a blue "Terminal" icon on the left towards the window. Inside the window, the SQL*Plus environment is displayed:

```

Terminal
File Edit View Terminal Tabs Help
SQL*Plus: Release 11.2.0.0.2 Beta on Thu May 28 21:20:35 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Enter user-name: ora41
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> set serveroutput on
SQL> create or replace procedure hello is
  2 begin
  3 dbms_output.put_line('Hello Class!');
  4 end;
  5 /

Procedure created.

SQL> execute hello
Hello Class!

PL/SQL procedure successfully completed.

SQL>

```

At the bottom of the window, there is a red bar with the "ORACLE" logo and the text "Copyright © 2009, Oracle. All rights reserved."

SQL*Plus에서 PL/SQL 코딩

Oracle SQL*Plus는 실행할 SQL 문과 PL/SQL 블록을 실행한 다음 그 결과를 응용 프로그램이나 명령 window에서 수신할 수 있는 명령행 인터페이스입니다.

SQL*Plus는 다음 특징을 갖습니다.

- 데이터베이스와 함께 제공됩니다.
- 클라이언트 및 데이터베이스 서버 시스템에 설치됩니다.
- 아이콘이나 명령행을 사용하여 액세스됩니다.

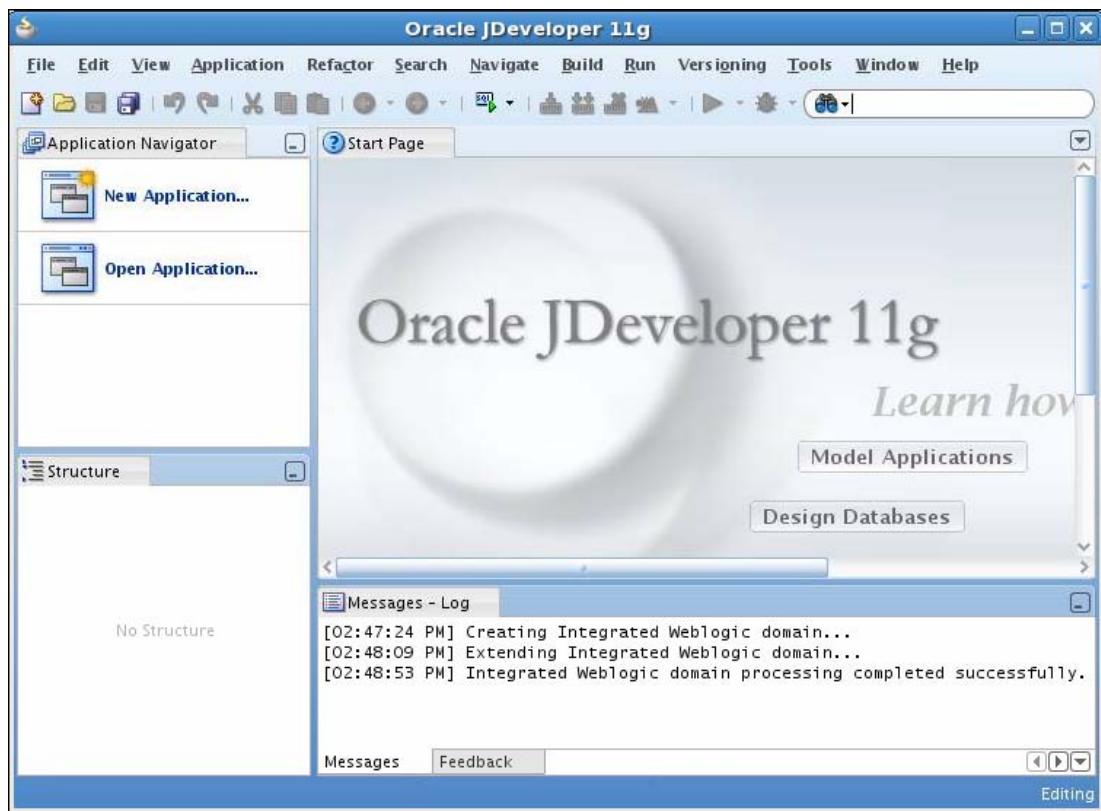
SQL*Plus를 사용하여 PL/SQL 서브 프로그램을 코딩할 때는 다음 사항을 고려하십시오.

- CREATE SQL 문을 사용하여 서브 프로그램을 생성합니다.
- 익명 PL/SQL 블록이나 EXECUTE 명령을 사용하여 서브 프로그램을 실행합니다.
- DBMS_OUTPUT 패키지 프로시저를 사용하여 화면에 텍스트를 출력할 경우 먼저 세션에서 SET SERVEROUTPUT ON 명령을 실행해야 합니다.

참고

- Linux 환경에서 SQL*Plus를 시작하려면 Terminal window를 열고 sqlplus 명령을 입력합니다.
- SQL*Plus 사용에 대한 자세한 내용은 부록 D를 참조하십시오.

Oracle JDeveloper에서 PL/SQL 코딩



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle JDeveloper에서 PL/SQL 코딩

Oracle JDeveloper를 사용하면 개발자는 정교한 GUI를 사용하여 PL/SQL 코드를 생성, 편집, 테스트 및 디버그할 수 있습니다. Oracle JDeveloper는 Oracle Developer Suite에 속하지만 독립된 제품으로도 사용할 수 있습니다.

JDeveloper에서 PL/SQL을 코딩할 때 다음 사항을 고려하십시오.

- 먼저 데이터베이스 연결을 생성하여 JDeveloper에서 서브 프로그램의 데이터베이스 스키마 소유자에 액세스할 수 있도록 합니다.
- 그런 다음 데이터베이스 연결의 JDeveloper 컨텍스트 메뉴를 사용하여 내장된 JDeveloper 코드 편집기에서 새 서브 프로그램 생성자를 생성합니다.
- 명명된 서브 프로그램의 컨텍스트 메뉴에서 Run 명령을 사용하여 서브 프로그램을 호출합니다. 결과는 위 스크린샷의 하단에 표시된 것처럼 JDeveloper Log Message window에 나타납니다.

참고

- JDeveloper는 JDeveloper 코드 편집기에서 색상 코딩 구문을 제공하며 PL/SQL 언어 생성자와 명령문을 쉽게 구분할 수 있게 해 줍니다.
- JDeveloper 사용에 대한 자세한 내용은 부록 E를 참조하십시오.

Oracle 11g SQL 및 PL/SQL 설명서

- ***Oracle Database New Features Guide 11g Release 2 (11.2)***
- ***Oracle Database Advanced Application Developer's Guide 11g Release 2 (11.2)***
- ***Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)***
- ***Oracle Database Reference 11g Release 2 (11.2)***
- ***Oracle Database SQL Language Reference 11g Release 2 (11.2)***
- ***Oracle Database Concepts 11g Release 2 (11.2)***
- ***Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)***
- ***Oracle Database SQL Developer User's Guide Release 1.5***



Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 과정 목표 설명
- 과정에서 사용하는 HR 데이터베이스 스키마 설명
- 본 과정에서 사용할 수 있는 유저 인터페이스 환경 식별
- 사용 가능한 부록, 설명서 및 기타 리소스 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 I: 개요: 시작하기

이 연습에서는 다음 내용을 다룹니다.

- SQL Developer 시작
- 새 데이터베이스 연결 생성
- HR 스키마 테이블 탐색
- SQL Developer 환경 설정 구성



Copyright © 2009, Oracle. All rights reserved.

연습 I: 개요

이 연습에서는 SQL Developer에서 SQL 문을 실행하여 HR 스키마의 데이터를 검사합니다. 또한 간단한 익명 블록을 생성합니다.

참고: 모든 연습 문제는 SQL Developer를 개발 환경으로 사용합니다. SQL Developer를 사용하는 것이 권장되지만 이 과정에서 사용 가능한 SQL*Plus 또는 JDeveloper 환경을 사용할 수도 있습니다.

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
- PL/SQL에서 메시지 출력

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 PL/SQL 및 PL/SQL 프로그래밍 생성자를 소개하고 PL/SQL의 이점에 대해서도 알아봅니다.

다루는 내용

- **PL/SQL의 이점과 구조 이해**
- **PL/SQL 블록 검사**
- **PL/SQL에서 출력 메시지 생성**

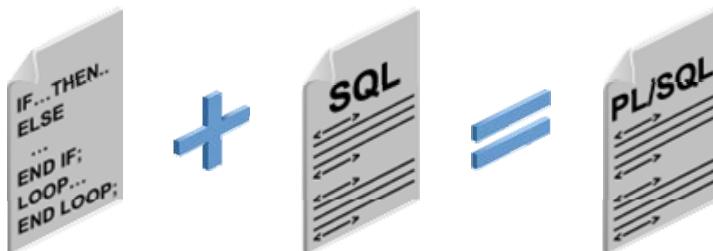
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 정보

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/SQL 정보

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 부서 ID와 급여를 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 구조 요소를 확인해야 합니다. 이렇게 하려면 과정이 약간 복잡해집니다.
- 더욱 효과적인 해결 방법에는 조건문이 포함될 수 있습니다. 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 데이터 유형
 - 조건문 및 루프와 같은 제어 구조
 - 한 번 작성하면 여러 번 실행할 수 있는 재사용 가능한 프로그램 단위

ORACLE®

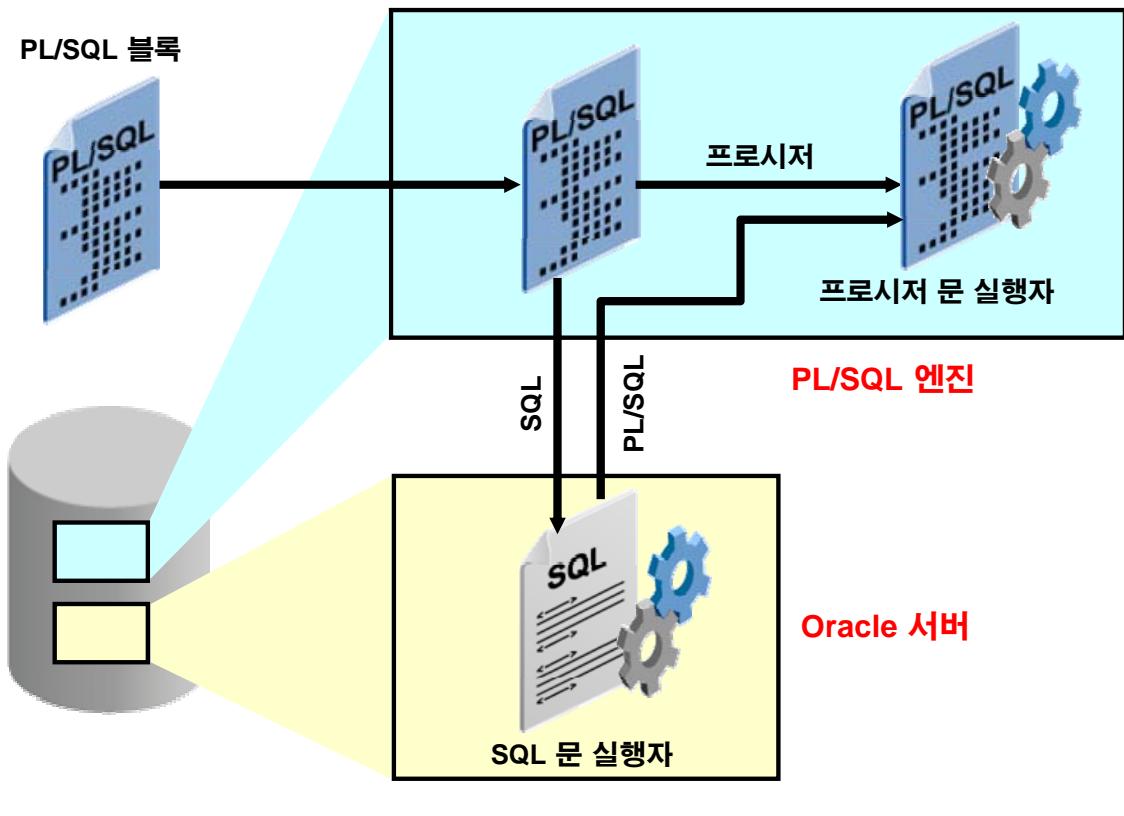
Copyright © 2009, Oracle. All rights reserved.

PL/SQL 정보 (계속)

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

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 런타임 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 런타임 구조

슬라이드의 도표는 PL/SQL 엔진에서 실행되는 PL/SQL 블록을 보여줍니다. PL/SQL 엔진이 상주하는 곳은 다음과 같습니다.

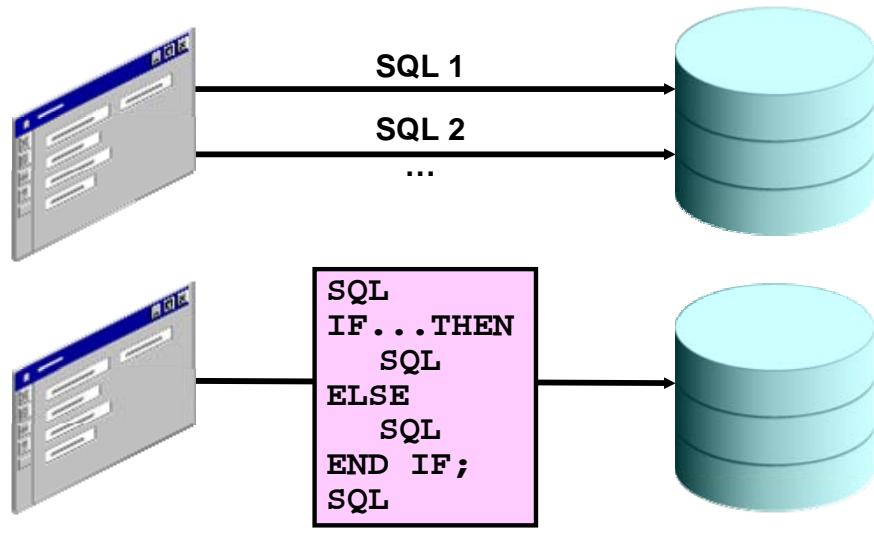
- 내장 서브 프로그램을 실행하는 오라클 데이터베이스
- 클라이언트/서버 응용 프로그램을 실행할 경우에는 Oracle Forms 클라이언트. Oracle Forms Services를 사용하여 웹에서 Forms를 실행할 경우에는 Oracle Application Server

PL/SQL 런타임 환경과 관계없이 기본 구조는 동일하게 유지됩니다. 그러므로 모든 PL/SQL 문은 프로시저 문 실행자에서 처리되며, 모든 SQL 문은 Oracle 서버 프로세스에서 처리할 수 있도록 SQL 문 실행자로 보내야 합니다. SQL 환경에서 PL/SQL 환경을 호출할 수도 있습니다. 예를 들어, SELECT 문에서 PL/SQL 함수를 사용하면 PL/SQL 환경이 호출됩니다.

PL/SQL 엔진은 메모리에 상주하며 PL/SQL m-code 명령을 처리하는 가상 시스템입니다. PL/SQL 엔진에서 SQL 문을 발견할 경우 SQL 문을 Oracle 서버 프로세스로 전달하도록 컨텍스트 전환이 이루어집니다. PL/SQL 엔진은 SQL 문이 완료되고 그 결과가 반환될 때까지 기다린 다음 PL/SQL 블록에서 이후 명령문을 계속 처리합니다. Oracle Forms PL/SQL 엔진은 클라이언트/서버를 구현하는 경우에는 클라이언트에서 실행되고, Forms Services를 구현하는 경우에는 Application Server에서 실행됩니다. 어느 경우든지 SQL 문은 대개 처리를 위해 네트워크를 통해 Oracle 서버로 전송됩니다.

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 설계한 경우 각 Form에 필드가 있는 다양한 Form을 사용할 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 떨어집니다.

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 관리하고 디버깅할 수 있습니다.

PL/SQL에서 모듈화는 "내장 프로시저 및 함수 소개" 단원에서 설명하는 프로시저, 함수 및 패키지를 사용하여 구현됩니다.

툴과의 통합: PL/SQL 엔진은 Oracle Forms 및 Oracle Reports와 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 커스터마이즈하지 않아도 됩니다. 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. 예외 처리는 "예외 처리" 단원에서 자세히 배웁니다.

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)**
 - 예외 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 네 개의 섹션으로 구성됩니다.

- **선언(선택 사항):** 선언 섹션은 DECLARE 키워드로 시작하며 실행 섹션 시작 부분에서 끝납니다.
- **시작(필수):** 실행 섹션은 BEGIN 키워드로 시작합니다. 이 섹션에는 명령문이 하나 이상 있어야 합니다. 그러나 PL/SQL 블록의 실행 섹션에는 PL/SQL 블록이 무제한 포함될 수 있습니다.
- **예외 처리(선택 사항):** 예외 섹션은 실행 섹션 내에 중첩됩니다. 이 섹션은 EXCEPTION 키워드로 시작됩니다.
- **종료(필수):** 모든 PL/SQL 블록은 END 문으로 끝나야 합니다. END는 세미콜론으로 종료됩니다.

PL/SQL 블록 구조 (계속)

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 사항
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 사항

다루는 내용

- PL/SQL의 이점과 구조 이해
- PL/SQL 블록 검사
- PL/SQL에서 출력 메시지 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

블록 유형

프로시저

```
PROCEDURE name
IS
BEGIN
    --statements
[ EXCEPTION ]
END;
```

함수

```
FUNCTION name
RETURN datatype
IS
BEGIN
    --statements
    RETURN value;
[ EXCEPTION ]
END;
```

익명 블록

```
[DECLARE]
BEGIN
    --statements
[ EXCEPTION ]
END;
```

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 다음 세 가지 유형의 블록이 있습니다.

- 프로시저
- 함수
- 익명 블록

프로시저: 프로시저는 SQL 및/또는 PL/SQL 문이 포함되어 있는 명명된 객체입니다.

함수: 함수는 SQL 및/또는 PL/SQL 문이 포함되어 있는 명명된 객체입니다. 프로시저와 달리 함수는 지정된 데이터 유형의 값을 반환합니다.

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.

동일한 블록을 다시 실행하려면 해당 블록을 다시 작성해야 합니다. 이 블록은 익명이며 실행된 후에는 존재하지 않기 때문에 이전에 작성한 블록을 실행하거나 호출할 수 없습니다.

블록 유형 (계속)

서브 프로그램

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

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

프로그램 생성자



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)	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

익명 블록 검사

SQL Developer 작업 영역의 익명 블록:



```
SQL Worksheet History
DECLARE
    v_fname VARCHAR2(20);
BEGIN
    SELECT first_name INTO v_fname FROM employees
    WHERE employee_id=100;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

익명 블록 검사

SQL Developer를 사용하여 익명 블록을 생성하려면 슬라이드에 표시된 것처럼 작업 영역에 블록을 입력하십시오.

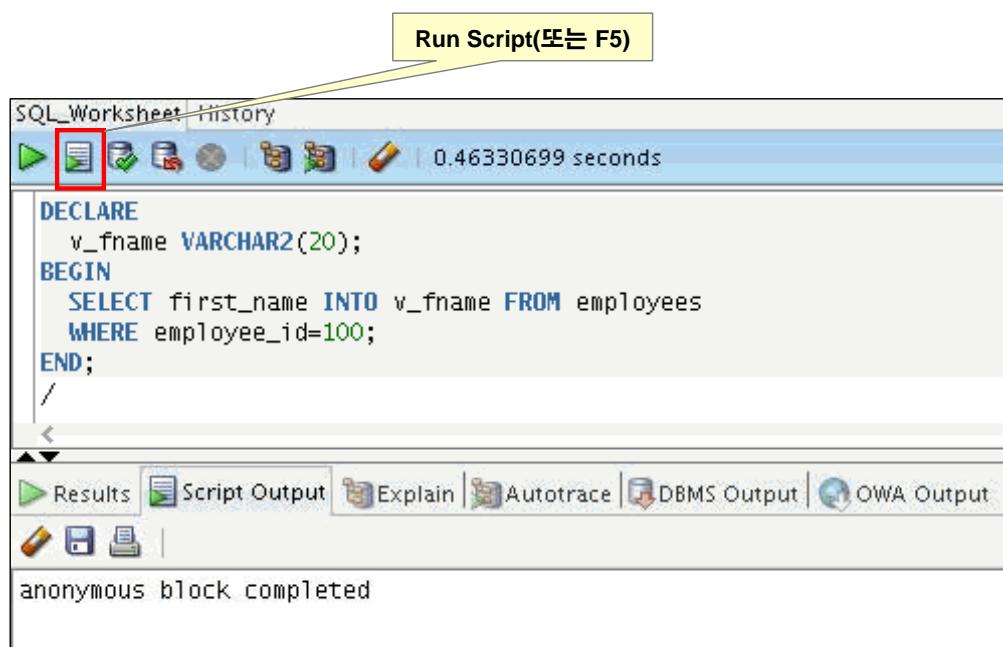
예제

블록에는 선언 섹션과 실행 섹션이 있습니다. 블록에 있는 명령문의 구문은 신경 쓰지 않아도 됩니다. 구문은 본 과정 뒷부분에서 배우게 됩니다.

익명 블록은 `employee_id`가 100인 사원의 `first_name`을 가져와서 `v_fname`이라는 변수에 저장합니다.

익명 블록 실행

Run Script 버튼을 눌러 익명 블록 실행:



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

익명 블록 실행

익명 블록을 실행하려면 Run Script 버튼이나 F5 키를 누릅니다.

참고: 블록이 실행된 후 Script Output window에 "anonymous block completed"라는 메시지가 표시됩니다.

다루는 내용

- PL/SQL의 이점과 구조 이해
- PL/SQL 블록 검사
- PL/SQL에서 출력 메시지 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록의 출력 활성화

- SQL Developer에서 출력을 활성화하려면 PL/SQL 블록을 실행하기 전에 다음 명령을 실행합니다.

```
SET SERVEROUTPUT ON
```

- 미리 정의된 Oracle 패키지와 해당 프로시저를 익명 블록에 사용합니다.

- DBMS_OUTPUT.PUT_LINE

```
DBMS_OUTPUT.PUT_LINE('The First Name of the
Employee is ' || v_fname);
```

Copyright © 2009, Oracle. All rights reserved.



PL/SQL 블록의 출력 활성화

이전 슬라이드의 예제에서 값이 v_fname 변수에 저장되었지만 출력되지는 않았습니다. PL/SQL에는 입출력 기능이 내장되어 있지 않습니다. 따라서 입출력용으로 미리 정의된 Oracle 패키지를 사용해야 합니다. 출력을 생성하려면 다음을 수행해야 합니다.

- 다음 명령을 실행합니다.

```
SET SERVEROUTPUT ON
```

참고: SQL*Plus에서 출력을 활성화하려면 SET SERVEROUTPUT ON 명령을 명시적으로 실행해야 합니다.

- PL/SQL 블록에서 DBMS_OUTPUT 패키지의 PUT_LINE 프로시저를 사용하여 출력을 표시합니다. 슬라이드에 표시된 것처럼 출력해야 할 값을 이 프로시저에 인수로 전달합니다. 그러면 이 프로시저가 인수를 출력합니다.

PL/SQL 블록의 출력 보기

The screenshot shows the Oracle SQL Worksheet interface. At the top, there's a toolbar with various icons and a status bar indicating "0.14278294 seconds". Below the toolbar is a code editor window containing the following PL/SQL code:

```

SET SERVEROUTPUT ON

DECLARE
    v_fname VARCHAR(20);
BEGIN
    SELECT first_name
    INTO v_fname
    FROM employees
    WHERE employee_id = 100;
    DBMS_OUTPUT.PUT_LINE('The First Name of the Employee is ' || v_fname);
END;
/

```

A callout bubble points to the "F5 키를 눌러 명령과 PL/SQL 블록을 실행합니다." (Press F5 to run the command and PL/SQL block) text. In the bottom panel, there are several tabs: "Results" (selected), "Script Output" (highlighted with a red box), "Explain", "Autotrace", "DBMS Output", and "OWA Output". The "Script Output" tab displays the output of the anonymous block:

```

anonymous block completed
The First Name of the Employee is Steven

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록의 출력 보기

PL/SQL 블록의 출력을 보려면 F5 키나 Run Script 아이콘을 누릅니다. 이 작업에 따라 다음이 수행됩니다.

1. SET SERVEROUTPUT ON 명령을 실행합니다.
2. 익명 PL/SQL 블록을 실행합니다.

Script Output 탭에 출력이 나타납니다.

퀴즈

PL/SQL 블록은 다음 세 개의 섹션으로 구성되어야 합니다.

- **DECLARE 키워드로 시작하며 실행 섹션 시작 부분에서 끝나는 선언 섹션**
- **BEGIN 키워드로 시작하여 END 키워드로 끝나는 실행 섹션**
- **EXCEPTION 키워드로 시작하여 실행 섹션 내에 중첩되는 예외 처리 섹션**

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 2

PL/SQL 블록은 다음 세 개의 섹션으로 구성됩니다.

- **선언(선택 사항):** 선택적 선언 섹션은 DECLARE 키워드로 시작하여 실행 섹션 시작 부분에서 끝납니다.
- **실행(필수):** 필수 실행 섹션은 BEGIN 키워드로 시작하여 END 키워드로 끝납니다. 이 섹션에는 반드시 적어도 하나의 명령문이 있어야 합니다. END는 세미콜론으로 종료됩니다. PL/SQL 블록의 실행 섹션은 PL/SQL 블록을 무제한 포함할 수 있습니다.
- **예외 처리(선택 사항):** 선택적인 예외 섹션은 실행 섹션 내에 중첩됩니다. 이 섹션은 EXCEPTION 키워드로 시작됩니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- SQL 문과 PL/SQL 프로그램 생성자의 통합
- PL/SQL의 이점 설명
- PL/SQL 블록 유형 간의 차이 설명
- PL/SQL에서 메시지 출력

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

PL/SQL은 SQL의 확장으로 사용되는 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 다집니다.

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

PL/SQL 변수 선언

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

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

다루는 내용

- **변수 소개**
- 변수 데이터 유형 및 %TYPE 속성 검사
- 바인드 변수 검사

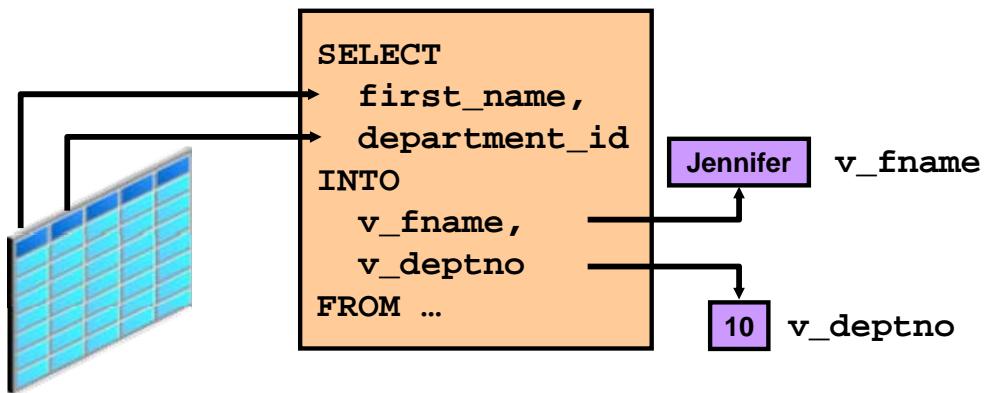
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 문과 프로시저문에서 해당 변수를 사용할 수 있습니다.

변수는 주로 데이터 저장 및 저장된 값의 조작에 사용됩니다. 슬라이드의 PL/SQL 문을 살펴보십시오. 이 명령문은 테이블에서 `first_name`과 `department_id`를 검색합니다. `first_name`이나 `department_id`를 조작해야 하는 경우 검색한 값을 저장해야 합니다. 변수는 값을 임시로 저장하는 데 사용됩니다. 이러한 변수에 저장된 값을 사용하여 데이터를 처리하고 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. 선언된 변수는 다양한 명령문에서 여러 번 참조하여 응용 프로그램에서 반복적으로 사용할 수 있습니다.

변수 이름에 대한 요구 사항

변수 이름:

- 문자로 시작해야 합니다.
- 문자나 숫자를 포함할 수 있습니다.
- 특수 문자(\$, _, # 등)를 포함할 수 있습니다.
- 30자 이하의 문자만 포함해야 합니다.
- 예약어를 포함하면 안됩니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

변수 이름에 대한 요구 사항

슬라이드에 변수 이름 지정 규칙이 나열되어 있습니다.

PL/SQL에서 변수 처리

변수는 다음과 같이 처리됩니다.

- 선언 섹션에서 선언 및 선택적으로 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

다음과 같은 방법으로 변수를 사용할 수 있습니다.

- **선언 섹션에서 선언 및 초기화:** PL/SQL 블록, 서브 프로그램 또는 패키지의 선언 부분에서 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 사용 및 새 값 할당:** 실행 섹션에서 변수의 기존 값을 새 값으로 바꿀 수 있습니다.
- **PL/SQL 서브 프로그램에 파라미터로 전달:** 서브 프로그램은 파라미터를 사용할 수 있습니다. 서브 프로그램에 파라미터로 변수를 전달할 수 있습니다.
- **PL/SQL 서브 프로그램의 출력 결과를 보유하는 데 사용:** 변수를 사용하여 함수에서 반환된 값을 보유할 수 있습니다.

PL/SQL 변수 선언 및 초기화

구문:

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

예제:

```
DECLARE
    v_hiredate      DATE;
    v_deptno        NUMBER(2) NOT NULL := 10;
    v_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. 선언에서 다른 변수를 참조할 경우 해당 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 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.

참고: 선언 섹션에서 변수와 함께 커서와 예외를 선언할 수도 있습니다. 커서 선언은 "명시적 커서 사용" 단원에서 배우고, 예외는 "예외 처리" 단원에서 배웁니다.

PL/SQL 변수 선언 및 초기화

```

1
DECLARE
  v_myName VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
  v_myName := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
END;
/

```

```

2
DECLARE
  v_myName VARCHAR2(20) := 'John';
BEGIN
  v_myName := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언 및 초기화 (계속)

슬라이드의 두 코드 블록을 살펴보십시오.

- 첫 블록에서는 `v_myName` 변수가 선언되지만 초기화되지 않습니다. 실행 �セ션에서 `John` 값이 변수에 할당됩니다.
 - 문자열 리터럴은 작은 따옴표로 묶어야 합니다. "Today's Date"처럼 문자열이 큰 따옴표로 묶인 경우 '`Today'`'s Date'로 나타납니다.
 - 할당 연산자는 `:=`입니다.
 - `v_myName` 변수를 전달하여 `PUT_LINE` 프로시저가 호출됩니다. 'My name is:' 문자열에 변수의 값이 연결됩니다.
 - 이 익명 블록의 출력은 다음과 같습니다.

```

Results Script Output Explain
anonymous block completed
My name is:
My name is: John

```

- 두번째 블록에서 `v_myName` 변수는 선언 셜션에서 선언되고 초기화됩니다. `v_myName`은 초기화 후에 `John` 값을 보유합니다. 이 값은 블록의 실행 셜션에서 조작됩니다. 이 익명 블록의 출력 결과는 다음과 같습니다.

```

anonymous block completed
My name is: Steven

```

문자열 리터럴의 구분자

```

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

```

결과 출력

```

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.

```
v_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. 예제는 !와 [를 구분자로 사용합니다. 다음 예제를 살펴보십시오.

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

이 예제를 이 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.

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

다루는 내용

- 변수 소개
- 변수 데이터 유형 및 %TYPE 속성 검사
- 바인드 변수 검사

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

변수 유형

- **PL/SQL 변수:**
 - 스칼라
 - 참조
 - LOB(대형 객체)
 - 조합
- **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 스칼라, 참조, LOB(대형 객체), 조합 등의 여러 데이터 유형 범주를 지원합니다.

- **스칼라 데이터 유형:** 스칼라 데이터 유형은 단일 값을 보유합니다. 값은 변수의 데이터 유형에 따라 다릅니다. 예를 들어, 이 단원의 "PL/SQL 변수 선언 및 초기화 지침" 섹션에 있는 예제의 `v_myName` 변수는 `VARCHAR2` 유형입니다. 따라서 `v_myName`은 문자열 값을 보유할 수 있습니다. 또한 PL/SQL은 부울 변수를 지원합니다.
- **참조 데이터 유형:** 참조 데이터 유형은 저장 위치를 가리키는 포인터라는 값을 보유합니다.
- **LOB 데이터 유형:** LOB 데이터 유형은 대형 객체의 위치를 지정하는 위치자라는 값을 보유합니다. 테이블 외부에 저장되는 그래픽 이미지 등이 대형 객체에 해당합니다.
- **조합 데이터 유형:** 조합 데이터 유형은 PL/SQL 컬렉션과 레코드 변수를 통해 사용할 수 있습니다. PL/SQL 컬렉션과 레코드에는 개별 변수로 처리할 수 있는 내부 요소가 포함됩니다.

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

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

변수 유형

TRUE



15-JAN-09

Snow White

Long, long ago,
in a land far, far away,
there lived a princess called
Snow White...

Atlanta

256120.08

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변수 유형 (계속)

슬라이드는 다음 데이터 유형을 보여줍니다.

- TRUE는 부울 값을 나타냅니다.
- 15-JAN-09는 DATE를 나타냅니다.
- 이미지는 BLOB를 나타냅니다.
- 콜아웃의 텍스트는 VARCHAR2 데이터 유형이나 CLOB를 나타낼 수 있습니다.
- 256120.08은 정밀도와 배율을 가진 NUMBER 데이터 유형을 나타냅니다.
- 필름 릴은 BFILE을 나타냅니다.
- 도시 이름 Atlanta는 VARCHAR2 데이터 유형을 나타냅니다.

PL/SQL 변수 선언 및 초기화 지침

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

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

```
v_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.

- 일관성 있는 이름 지정 규칙을 따릅니다. 예를 들어, name을 사용하여 변수를 나타내고 c_name을 사용하여 상수를 나타낼 수 있습니다. 마찬가지로 변수 이름을 지정하기 위해 v_fname을 사용할 수 있습니다. 중요한 점은 쉽게 식별할 수 있도록 일관성 있게 이름 지정 규칙을 적용하는 것입니다.
- 변수의 의미를 알 수 있는 알맞은 식별자를 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 선언에서 CONSTANT 키워드는 유형 지정자 앞에 와야 합니다. 다음 선언은 NUMBER 유형의 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 NUMBER := 50000.00;
```

PL/SQL 변수 선언 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 지침

- 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. 그러나 모든 변수를 초기화하는 것은 좋은 프로그래밍 습관입니다.
- 두 개의 객체가 서로 다른 블록에서 정의된 경우에만 동일한 이름을 가질 수 있습니다. 두 객체가 명존하는 경우 레이블을 사용하여 한정하고 사용할 수 있습니다.
- 열 이름을 식별자로 사용하지 마십시오. PL/SQL 변수가 SQL 문에 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 판독이나 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 와야 합니다.

```
pincode VARCHAR2(15) NOT NULL := 'Oxford';
```

이 과정에서 사용하는 PL/SQL 구조의 이름 지정 규칙

PL/SQL 구조	표기법	예제
변수	v_variable_name	v_rate
상수	c_constant_name	c_rate
서브 프로그램 파라미터	p_parameter_name	p_id
바인드(호스트) 변수	b_bind_name	b_salary
커서	cur_cursor_name	cur_emp
레코드	rec_record_name	rec_emp
유형	type_name_type	ename_table_type
예외	e_exception_name	e_products_invalid
파일 처리	f_file_handle_name	f_file

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

이 과정에서 사용하는 PL/SQL 구조의 이름 지정 규칙

슬라이드의 표에는 이 과정에서 사용되는 PL/SQL 구조의 몇 가지 이름 지정 규칙 예제가 나와 있습니다.

스칼라 데이터 유형

- 단일 값을 보유합니다.
- 내부 구성 요소가 없습니다.

TRUE

15-JAN-09

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

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 단원에서는 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)**
- **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. <i>maximum_length</i> 를 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 <i>p</i> 와 배율 <i>s</i> 를 가진 숫자입니다. 정밀도 <i>p</i> 의 범위는 1부터 38까지, 배율 <i>s</i> 의 범위는 -84부터 127까지입니다.
BINARY_INTEGER	-2,147,483,647 - 2,147,483,647 사이의 정수에 대한 기본 유형입니다.

기본 스칼라 데이터 유형 (계속)

데이터 유형	설명
PLS_INTEGER	-2,147,483,647 - 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 11g에서 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, NULL) 중 하나를 저장하는 기본 유형입니다.
BINARY_FLOAT	IEEE 754 형식의 부동 소수점 수를 나타냅니다. 값을 저장하기 위해 5바이트가 필요합니다.
BINARY_DOUBLE	IEEE 754 형식의 부동 소수점 수를 나타냅니다. 값을 저장하기 위해 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. - 9999 A.D 사이입니다.
TIMESTAMP	TIMESTAMP 데이터 유형은 DATE 데이터 유형을 확장하고 연도, 월, 일, 시, 분, 초 및 소수로 표시되는 초 단위를 저장합니다. 구문은 <code>TIMESTAMP[(precision)]</code> 이며 여기서 선택적 파라미터 <code>precision</code> 은 초 필드의 소수 부분 자릿수를 지정합니다. 자릿수를 지정하려면 0~9 범위의 정수를 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> 이며 여기서 선택적 파라미터 <code>precision</code> 은 초 필드의 소수 부분 자릿수를 지정합니다. 자릿수를 지정하려면 0~9 범위의 정수를 사용해야 합니다. 기본값은 6입니다.

기본 스칼라 데이터 유형 (계속)

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

스칼라 변수 선언

예제:

```
DECLARE
    v_emp_job          VARCHAR2(9);
    v_count_loop       BINARY_INTEGER := 0;
    v_dept_total_sal  NUMBER(9,2) := 0;
    v_orderdate        DATE := SYSDATE + 7;
    c_tax_rate         CONSTANT NUMBER(3,2) := 8.25;
    v_valid            BOOLEAN NOT NULL := TRUE;
    ...

```

Copyright © 2009, Oracle. All rights reserved.

스칼라 변수 선언

슬라이드의 변수 선언 예제는 다음과 같이 정의됩니다.

- **v_emp_job:** 사원의 직위를 저장하는 변수입니다.
- **v_count_loop:** 루프 반복 횟수를 세는 변수이며 0으로 초기화되었습니다.
- **v_dept_total_sal:** 부서의 총 급여를 합산하는 변수이며 0으로 초기화되었습니다.
- **v_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.
- **v_valid:** 해당 데이터 부분이 유효한지 여부를 나타내는 플래그이며 TRUE로 초기화되었습니다.

%TYPE 속성

- 다음에 따라 변수를 선언하는 데 사용됩니다.
 - 데이터베이스 열 정의
 - 다른 선언된 변수
- 다음 항목이 앞에 나옵니다.
 - 데이터베이스 테이블과 열 이름
 - 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 선언할 변수 앞에 이전에 선언한 변수의 변수 이름을 놓으십시오.

%TYPE 속성 (계속)

%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

```
...
v_emp_lname      employees.last_name%TYPE;
...
```

```
...
v_balance        NUMBER(7,2);
v_min_balance    v_balance%TYPE := 1000;
...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

%TYPE 속성을 사용하여 변수 선언

사원의 성을 저장할 변수를 선언합니다. `v_emp_lname` 변수는 사원 테이블의 `v_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. `v_min_balance` 변수는 `v_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;
```

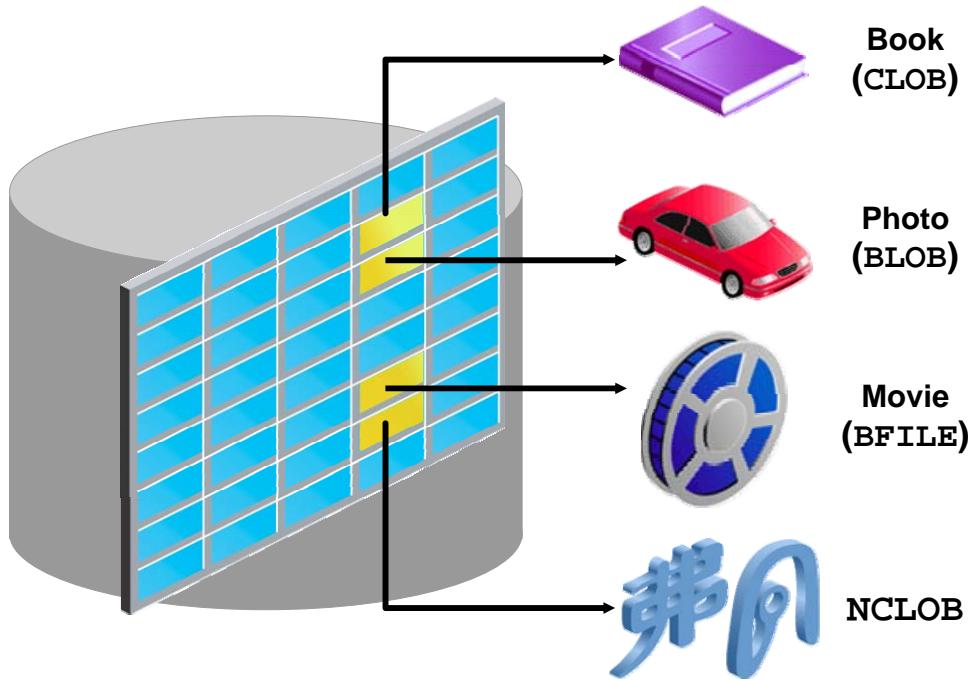
다음 표현식은 TRUE를 반환합니다.

```
emp_sal1 < emp_sal2
```

부울 변수를 선언하고 초기화합니다.

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

LOB 데이터 유형 변수



ORACLE

Copyright © 2009, Oracle. All rights reserved.

LOB 데이터 유형 변수

LOB(대형 객체)는 다량의 데이터를 저장하는 것을 의미합니다. 데이터베이스 열은 LOB 범주에 포함될 수 있습니다. LOB 범주의 데이터 유형(BLOB, CLOB 등)을 사용하여 데이터베이스 블록 크기에 따라 최대 128TB까지 텍스트, 그래픽 이미지, 비디오 클립, 사운드 웨이브 형식 등의 구조화되지 않은 데이터 블록을 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 external 응용 프로그램이 해석합니다.
- BFILE(Binary File) 데이터 유형은 대형 Binary File을 저장하는 데 사용됩니다. 다른 LOB 와 달리 BFILES는 데이터베이스 외부에 저장되며 데이터베이스에 저장되지 않습니다. 운영 체제 파일이 여기에 속합니다. BFILE에 대한 포인터만 데이터베이스에 저장됩니다.
- NCLLOB(National language Character Large Object) 데이터 유형은 대형 블록의 단일 바이트 또는 고정 너비 멀티바이트 NCHAR 유니코드 데이터를 데이터베이스에 저장하는 데 사용됩니다.

조합 데이터 유형: 레코드 및 컬렉션

PL/SQL 레코드:

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

PL/SQL 컬렉션:

1	SMITH	1	5000
2	JONES	2	2345
3	NANCY	3	12
4	TIM	4	3456

↓ ↓ ↓ ↓
 PLS_INTEGER VARCHAR2 PLS_INTEGER NUMBER
 ← ← ← ←

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조합 데이터 유형: 레코드 및 컬렉션

앞에서 언급한 바와 같이, 스칼라 데이터 유형은 단일 값을 보유하며 내부 구성 요소가 없습니다. PL/SQL 레코드 및 PL/SQL 컬렉션이라는 조합 데이터 유형에는 개별 변수로 처리할 수 있는 내부 구성 요소가 있습니다.

- PL/SQL 레코드에서 내부 구성 요소는 다양한 데이터 유형일 수 있으며 필드라고 합니다. `record_name.field_name` 구문을 사용하여 각 필드에 액세스합니다. 레코드 변수는 테이블 행이나 테이블 행의 일부 열을 보유할 수 있습니다. 각 레코드 필드는 테이블 열에 해당합니다.
- PL/SQL 컬렉션에서 내부 구성 요소는 항상 동일한 데이터 유형이며 요소라고 합니다. 고유의 하위 스크립트를 통해 각 요소에 액세스합니다. 컬렉션의 대표적인 예는 리스트와 배열입니다. 연관 배열, 중첩 테이블 및 VARRAY 유형이라는 세 가지 유형의 PL/SQL 컬렉션이 있습니다.

참고

- PL/SQL 레코드와 연관 배열은 "조합 데이터 유형 작업" 단원에서 다룹니다.
- NESTED TABLE 및 VARRAY 데이터 유형은 *Oracle Database 11g: Advanced PL/SQL* 과정에서 다룹니다.

다루는 내용

- 변수 소개
- 변수 데이터 유형 및 %TYPE 속성 검사
- 바인드 변수 검사

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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.

PRINT 명령을 사용하여 값을 출력할 수 있습니다.

* SQL*Plus 및 SQL Developer를 사용할 경우 필수입니다.

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. 그러므로 바인드 변수는 블록이 실행된 후에도 액세스할 수 있습니다. 생성된 바인드 변수는 다중 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 외부에 런타임 값으로 전달할 수 있습니다.

참고: 바인드 변수는 환경 변수지만 전역(global) 변수는 아닙니다.

바인드 변수 생성

SQL Developer에서 바인드 변수를 생성하려면 VARIABLE 명령을 사용하십시오. 예를 들어, 다음과 같이 NUMBER 및 VARCHAR2 유형의 변수를 선언합니다.

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

바인드 변수의 값 보기

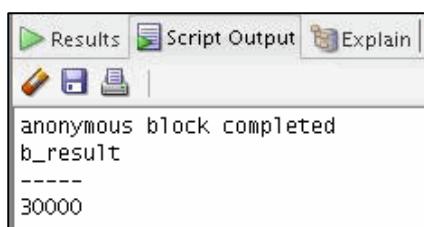
SQL Developer를 사용하여 바인드 변수를 참조할 수 있으며 PRINT 명령을 사용하여 바인드 변수의 값을 볼 수 있습니다.

바인드 변수 (계속)

예제

변수 앞에 콜론을 사용하여 PL/SQL 프로그램에서 바인드 변수를 참조할 수 있습니다. 예를 들어, 다음 PL/SQL 블록은 b_result 바인드 변수를 생성하고 사용합니다. PRINT 의 출력 결과는 코드 아래에 표시됩니다.

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



참고: 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. 유저는 필요한 바이트 수를 할당할 수 있도록 문자열의 크기를 지정해야 합니다.

바인드 변수 참조

예제:

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

anonymous block completed
b_emp_salary

7000

FIRST_NAME	LAST_NAME
Oliver	Tuvault
Sarath	Sewall
Kimberely	Grant

3 rows selected

ORACLE

Copyright © 2009, Oracle. All rights reserved.

바인드 변수 참조

앞에서 설명한 바와 같이, 바인드 변수를 생성한 후에는 다른 모든 SQL 문 또는 PL/SQL 프로그램에서 이 변수를 참조할 수 있습니다.

예제에서는 b_emp_salary가 PL/SQL 블록에 바인드 변수로 생성되고 그 다음에 오는 SELECT 문에 사용됩니다.

슬라이드와 같이 PL/SQL 블록을 실행하면 다음 출력이 나타납니다.

- PRINT 명령이 실행됩니다.

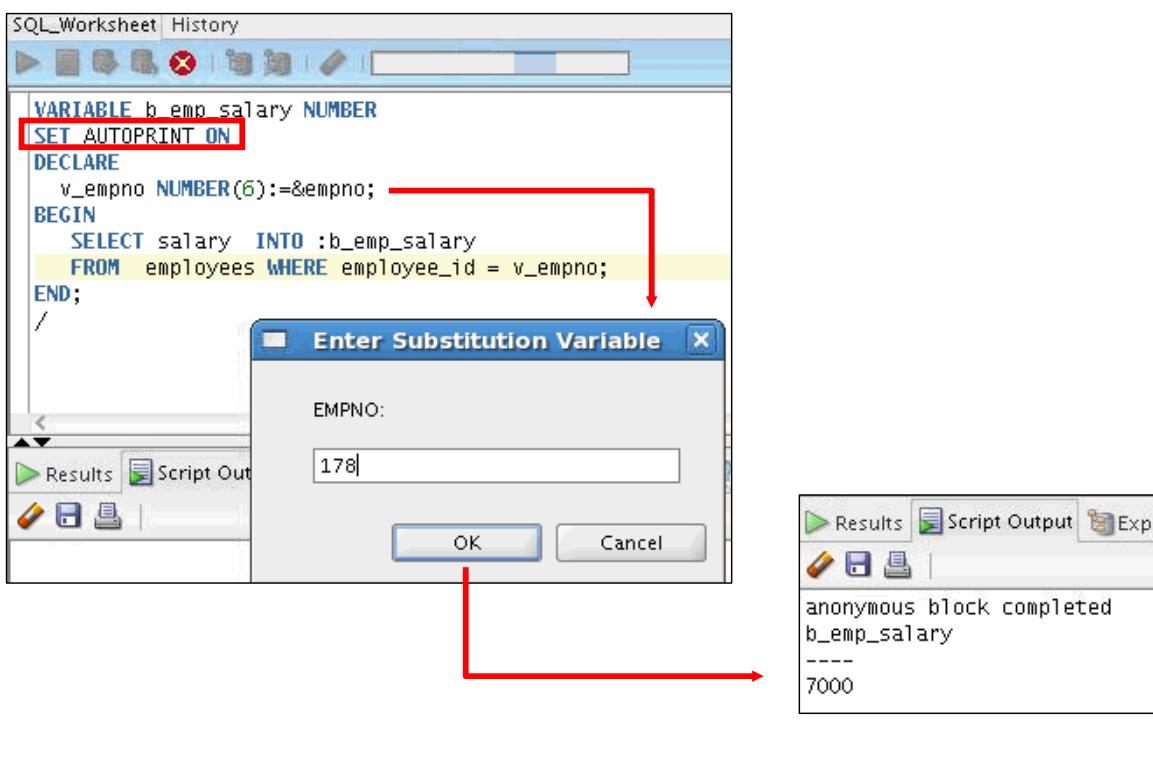
```
b_emp_salary
-----
7000
```

- 그런 다음 SQL 문이 출력됩니다.

```
FIRST_NAME      LAST_NAME
-----          -----
Oliver          Tuvault
Sarah           Sewall
Kimberely       Grant
```

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

바인드 변수와 함께 AUTOPRINT 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

바인드 변수와 함께 AUTOPRINT 사용

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

예제

코드 예제 설명:

- `b_emp_salary`라는 바인드 변수가 생성되고 AUTOPRINT가 켜집니다.
- `v_empno`라는 변수가 선언되고, 유저 입력을 받기 위해 치환 변수가 사용됩니다.
- 마지막으로 PL/SQL 블록의 실행 섹션에 바인드 변수와 임시 변수가 사용됩니다.

유효한 사원 번호(이 경우 178)를 입력하면 바인드 변수의 출력이 자동으로 인쇄됩니다.
바인드 변수에는 유저가 제공하는 사원 번호의 급여가 포함됩니다.

퀴즈

%TYPE 속성:

- 1. 데이터베이스 열 정의에 따라 변수를 선언하는 데 사용됩니다.**
- 2. 데이터베이스 테이블 또는 뷰의 열 모음에 따라 변수를 선언하는 데 사용됩니다.**
- 3. 선언된 또 다른 변수의 정의에 따라 변수를 선언하는 데 사용됩니다.**
- 4. 데이터베이스 테이블 및 열 이름이나 선언된 변수의 이름이 접두어로 불습니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3, 4

%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 선언할 변수 앞에 이전에 선언한 변수의 변수 이름을 놓으십시오. %TYPE의 이점은 열이 변경되는 경우에도 변수를 변경할 필요가 없다는 것입니다. 또한 변수가 계산에 사용될 경우 변수의 정밀도에 대해서 신경 쓸 필요가 없습니다.

%ROWTYPE 속성

%ROWTYPE 속성은 테이블 또는 뷰의 전체 행을 포함할 수 있는 레코드를 선언하는 데 사용됩니다. 이 속성은 "조합 데이터 유형 작업" 단원에서 배웁니다.

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 선언하는데 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.

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
- PL/SQL 표현식에서 시퀀스 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

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 블록에 있는 변수의 범위 및 가시성에 대해 배우고 레이블로 변수를 한정하는 방법을 배웁니다.

다루는 내용

- PL/SQL 블록에서 실행문 작성
- 중첩 블록 작성
- 연산자 사용 및 읽기 쉬운 코드 개발

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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: `v_fname, c_percent`
 - 구분자: `, +, -`
 - 리터럴: `John, 428, True`
 - 주석: `--, /* */`

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. 그러나 따옴표로 묶인 식별자를 사용하지 않는 것이 좋습니다.

- **구분자:** 구분자는 특별한 의미를 지닌 기호입니다. 이미 앞에서 세미콜론(;)이 SQL 또는 PL/SQL 문을 종료하는 데 사용된다는 것을 배웠습니다. 따라서 ;은 구분자의 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
<>	부등호 연산자
!=	부등호 연산자
	연결 연산자
--	단일 행 주석 표시자
/*	주석 시작 구분자
*/	주석 종료 구분자
:=	할당 연산자

참고: 이 리스트에는 전체 구분자가 아니라 일부만 포함되어 있습니다.

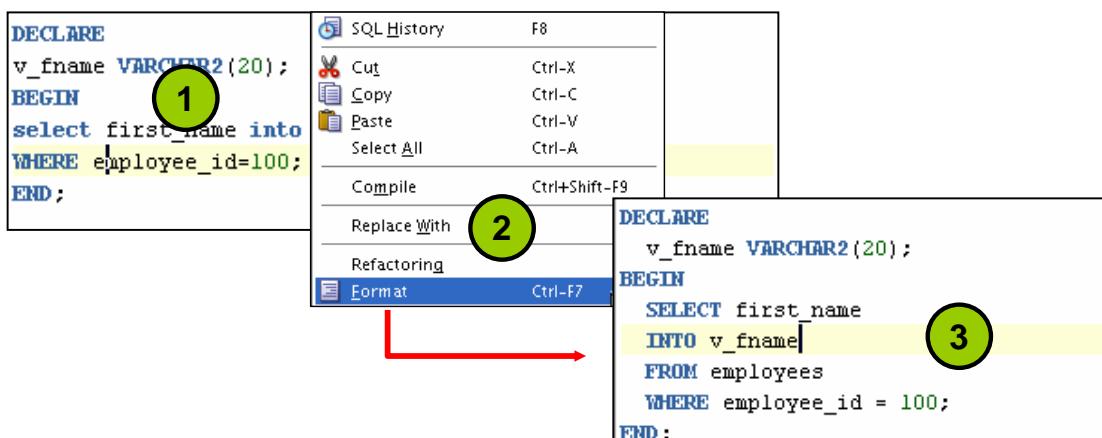
- **리터럴:** 변수에 할당되는 모든 값은 리터럴입니다. 식별자가 아닌 모든 문자, 숫자, 부울 또는 날짜 값은 리터럴입니다. 리터럴은 다음과 같이 분류됩니다.
 - **문자 리터럴:** 모든 문자열 리터럴은 데이터 유형이 CHAR 또는 VARCHAR2이므로 문자 리터럴이라고 합니다(예: John 및 12C).
 - **숫자 리터럴:** 숫자 리터럴은 정수 또는 실수 값을 나타냅니다(예: 428 및 1.276).
 - **부울 리터럴:** 부울 변수에 할당된 값은 부울 리터럴입니다. TRUE, FALSE 및 NULL은 부울 리터럴이거나 키워드입니다.
- **주석:** 해당 코드가 어떤 작업을 수행하는지 설명하는 것은 좋은 프로그래밍 습관입니다. 그러나 PL/SQL 블록에 설명을 포함해도 컴파일러는 해당 내용을 해석할 수 없습니다. 따라서 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 사용**
 - 문자 및 날짜 리터럴은 작은 따옴표로 묶어야 합니다.
 - 숫자에는 간단한 값이나 과학적 표기법이 올 수 있습니다.

```
v_name := 'Henderson';
```

- **코드 형식 정의: 명령문은 여러 행에 걸쳐 배열될 수 있습니다.**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록 구문 및 작성 지침

리터럴 사용

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

- 문자 리터럴은 PL/SQL character set에서 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 형식 정의

PL/SQL 블록에서 SQL 문은 여러 행에 걸쳐 배열될 수 있습니다(슬라이드의 예제 3 참조).

SQL Worksheet 단축 메뉴를 사용하여 형식이 정의되지 않은 SQL 문(슬라이드의 예제 1 참조)의 형식을 정의할 수 있습니다. 활성 SQL 워크시트를 마우스 오른쪽 버튼으로 누른 다음 표시되는 단축 메뉴에서 Format 옵션을 선택합니다(예제 2 참조).

참고: Ctrl+F7의 단축 키 조합을 사용하여 코드의 형식을 정의할 수도 있습니다.

코드 주석 처리

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

예제:

```
DECLARE
  ...
v_annual_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

코드 주석 처리

각 단계를 문서화하고 디버깅을 지원하기 위해 코드를 주석 처리해야 합니다. PL/SQL 코드에서는 다음과 같습니다.

- 단일 행 주석에는 일반적으로 하이픈(--) 두 개가 접두어로 붙습니다.
- /* 기호와 */ 기호 사이에 주석을 넣을 수도 있습니다.

참고: 다중 행 주석의 경우 각 주석 행 앞에 하이픈 두 개를 넣거나 블록 주석 형식을 사용할 수 있습니다.

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.

PL/SQL의 SQL 함수

- **프로시저문에서 사용할 수 있는 함수:**
 - 단일 행 함수
- **프로시저문에서 사용할 수 없는 함수:**
 - DECODE
 - 그룹 함수

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SQL 함수

SQL은 SQL 문에서 사용할 수 있는 많은 미리 정의된 함수를 제공합니다. 이러한 함수 중 대부분(단일 행 숫자 및 문자 함수, 데이터 유형 변환 함수, 날짜 및 시간 기록 함수 등)은 PL/SQL 표현식에서 유효합니다.

다음 함수는 프로시저문에서 사용할 수 없습니다.

- DECODE
- 그룹 함수: AVG, MIN, MAX, COUNT, SUM, STDDEV 및 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.

```
v_desc_size INTEGER(5);
v_prod_description VARCHAR2(70):='You can use this
product with your radios for higher frequency';

-- get the length of the string in prod_description
v_desc_size:= LENGTH(v_prod_description);
```

- 사원이 근무한 개월 수를 구합니다.

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 SQL 함수: 예제

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

- 번호
- 문자
- 변환
- 날짜
- 기타

PL/SQL 표현식에서 시퀀스 사용

11g 이상:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    v_new_id := my_seq.NEXTVAL;
END;
/
```

11g 이전:

```
DECLARE
    v_new_id NUMBER;
BEGIN
    SELECT my_seq.NEXTVAL INTO v_new_id FROM Dual;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

시퀀스 값 액세스

Oracle Database 11g에서는 NUMBER 데이터 유형의 표현식이 올바르게 나타날 수 있는 PL/SQL 쿼리문에서 NEXTVAL 및 CURRVAL의 Pseudo Column을 사용할 수 있습니다. SELECT 문을 사용하여 시퀀스를 query하는 이전 스타일도 여전히 유효하지만 사용하지 않는 것이 좋습니다.

Oracle Database 11g 이전에는 PL/SQL 서브 루틴에서 시퀀스 객체 값을 사용하기 위해 SQL 문을 작성해야 했습니다. NEXTVAL 및 CURRVAL의 Pseudo Column을 참조하는 SELECT 문을 작성하여 시퀀스 번호를 획득하는 것이 일반적이었으나, 이 방법을 사용하면 사용성 문제가 발생합니다.

Oracle Database 11g에서는 시퀀스 값을 검색하는 SQL 문을 작성해야 하는 부담이 사라졌습니다. 시퀀스 향상 기능으로 얻을 수 있는 이점은 다음과 같습니다.

- 시퀀스 사용성이 개선되었습니다.
- 개발자가 입력해야 하는 내용이 줄었습니다.
- 결과 코드가 보다 분명해졌습니다.

데이터 유형 변환

- 데이터를 유사한 데이터 유형으로 변환합니다.
- 두 가지 유형이 있습니다.
 - 암시적 변환
 - 명시적 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
    v_salary NUMBER(6) := 6000;
    v_sal_hike VARCHAR2(5) := '1000';
    v_total_salary v_salary%TYPE;
BEGIN
    v_total_salary := v_salary + v_sal_hike;
END;
/
```

이 예제에서 `sal_hike` 변수는 `VARCHAR2` 유형입니다. 전체 급여를 계산할 때 PL/SQL은 먼저 `sal_hike`를 `NUMBER`로 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_DATE`를 사용하고 `NUMBER` 값으로 변환하려면 `TO_NUMBER`를 사용합니다.

데이터 유형 변환

1

-- 암시적 데이터 유형 변환

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

2

-- 데이터 유형 변환 오류

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

3

-- 명시적 데이터 유형 변환

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 유형 변환 (계속)

슬라이드에서 DATE 데이터 유형의 암시적 및 명시적 변환을 보여주는 세 가지 예제를 살펴보십시오.

1. date_of_joining에 할당되는 문자열 리터럴이 기본 형식이기 때문에 이 예제는 암시적 변환을 수행하고 지정된 날짜를 date_of_joining에 할당합니다.
2. 할당되는 날짜가 기본 형식이 아니기 때문에 PL/SQL이 오류를 반환합니다.
3. TO_DATE 함수를 사용하여 제공된 날짜를 명시적으로 특정 형식으로 변환하여 DATE 데이터 유형 변수 date_of_joining에 할당합니다.

다루는 내용

- PL/SQL 블록에서 실행문 작성
- 중첩 블록 작성
- 연산자 사용 및 읽기 쉬운 코드 개발

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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

프로시저가 될 경우 PL/SQL은 명령문을 중첩할 수 있게 됩니다. 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
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;

```

anonymous block completed
 LOCAL VARIABLE
 GLOBAL VARIABLE
 GLOBAL VARIABLE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 블록 (계속)

슬라이드의 예제는 외부(상위) 블록과 중첩(하위) 블록을 보여줍니다. `v_outer_variable` 변수는 외부 블록에서 선언되고 `v_inner_variable` 변수는 내부 블록에서 선언됩니다.

`v_outer_variable`은 외부 블록에 대해서는 로컬 변수지만 내부 블록에 대해서는 전역(global) 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 `v_outer_variable`은 모든 포함하는 블록에 대해 전역(global) 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(global) 변수로 간주됩니다.

내부 블록에 중첩 블록이 없기 때문에 `v_inner_variable`은 내부 블록에 대해 전역(global) 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
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth); ←
    DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
  END;
  → DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/

```

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 범위와 가시성이 올바르게 적용되지 않기 때문에 출력에서 올바른 정보가 제공되지 않습니다.

- 변수의 범위는 변수를 선언하고 액세스할 수 있는 프로그램의 영역입니다.
- 변수의 가시성은 수식자를 사용하지 않고도 변수에 액세스할 수 있는 프로그램의 영역입니다.

범위

- v_father_name 변수와 v_date_of_birth 변수의 첫번째 발생은 외부 블록에 선언됩니다. 이 변수에는 변수가 선언되는 블록의 범위가 있습니다. 따라서 이러한 변수의 범위는 외부 블록으로 제한됩니다.

변수 범위 및 가시성 (계속)

범위 (계속)

- `v_child_name` 및 `v_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 `v_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. 외부 블록에서 아버지의 생일을 볼 수 있으므로 출력할 수 있습니다.

참고: 한 블록에 동일한 이름의 변수가 여러 개 있을 수 없습니다. 그러나 예제와 같이 두 개의 다른 블록(중첩 블록)에서 동일한 이름의 변수를 여러 개 선언할 수 있습니다. 식별자가 나타내는 두 개의 항목은 서로 구분되며, 한 항목을 변경해도 다른 항목에 영향을 주지 않습니다.

중첩 블록에 수식자 사용

```

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

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중첩 블록에 수식자 사용

수식자는 블록에 지정되는 레이블입니다. 수식자를 사용하여 범위를 가지고 있지만 볼 수는 없는 변수에 액세스할 수 있습니다.

예제

코드 예제 설명:

- 외부 블록에는 outer라는 레이블이 지정됩니다.
- 내부 블록 내에서는 outer 수식자를 사용하여 외부 블록에서 선언되는 v_date_of_birth 변수에 액세스합니다. 따라서 내부 블록 내에서 아버지의 생일과 자녀의 생일을 모두 출력할 수 있습니다.
- 슬라이드의 코드 출력에는 올바른 정보가 표시됩니다.

```

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

```

참고: 레이블 지정은 외부 블록에 제한되지 않습니다. 모든 블록에 레이블을 지정할 수 있습니다.

문제: 변수 범위 결정

```
BEGIN <>outer>>
DECLARE
    v_sal      NUMBER(7,2) := 60000;
    v_comm     NUMBER(7,2) := v_sal * 0.20;
    v_message  VARCHAR2(255) := ' eligible for commission';
BEGIN
    DECLARE
        v_sal      NUMBER(7,2) := 50000;
        v_comm     NUMBER(7,2) := 0;
        v_total_comp NUMBER(7,2) := v_sal + v_comm;
    BEGIN
        1-> v_message := 'CLERK not'||v_message;
        outer.v_comm := v_sal * 0.30;

        END;
        2-> v_message := 'SALESMAN'||v_message;
    END;
END outer;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

문제: 변수 범위 결정

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

1. 위치 1에서 v_message의 값
2. 위치 2에서 v_total_comp의 값
3. 위치 1에서 v_comm의 값
4. 위치 1에서 outer.v_comm의 값
5. 위치 2에서 v_comm의 값
6. 위치 2에서 v_message의 값

정답: 변수 범위 결정

범위 문제에 대한 해답은 다음과 같습니다.

1. 위치 1에서 v_message의 값: 커미션 대상자가 아닌 CLERK
2. 위치 2에서 v_total_comp의 값: Error. v_total_comp는 내부 블록 안에서 정의되므로 여기서 볼 수 없습니다.
3. 위치 1에서 v_comm의 값: 0
4. 위치 1에서 outer.v_comm의 값: 12000
5. 위치 2에서 v_comm의 값: 15000
6. 위치 2에서 v_message의 값: 커미션 대상자가 아닌 SALESMANCLERK

다루는 내용

- PL/SQL 블록에서 실행문 작성
- 중첩 블록 작성
- 연산자 사용 및 읽기 쉬운 코드 개발

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL의 연산자

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

SQL과 동일

Copyright © 2009, Oracle. All rights reserved.

ORACLE

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	소문자, 복수	employees, departments
데이터베이스 열	소문자, 단수	employee_id, department_id

코드 들여쓰기

명확성을 위해 코드의 각 레벨을 들여 씁니다.

```
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;
```

퀴즈

PL/SQL 표현식에서 숫자, 문자, 변환, 날짜 단일 행 함수와 같은 대부분의 SQL 단일 행 함수를 사용할 수 있습니다.

1. 맞습니다.
2. 틀립니다.



Copyright © 2009, Oracle. All rights reserved.

정답: 1

PL/SQL의 SQL 함수

SQL은 SQL 문에서 사용할 수 있는 많은 미리 정의된 함수를 제공합니다. 이러한 함수 중 대부분(단일 행 숫자 및 문자 함수, 데이터 유형 변환 함수, 날짜 및 시간 기록 함수 등)은 PL/SQL 표현식에서 유효합니다.

다음 함수는 프로시저문에서 사용할 수 없습니다.

- DECODE
- 그룹 함수: AVG, MIN, MAX, COUNT, SUM, STDDEV 및 VARIANCE 그룹 함수는 테이블의 행 그룹에 적용되므로 PL/SQL 블록의 SQL 문에서만 사용할 수 있습니다. 여기에서 언급한 함수는 전체 리스트 중 일부에 불과합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- PL/SQL 블록의 렉시칼 단위 파악
- PL/SQL의 내장 SQL 함수 사용
- 중첩 블록을 작성하여 논리적으로 관련된 기능 분리
- 명시적 변환 수행 시기 결정
- 중첩 블록에서 변수 한정
- PL/SQL 표현식에서 시퀀스 사용

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 �кции은 실행 셙션에도 속하기 때문에 중첩 블록도 포함할 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 표현식을 비교합니다. 결과는 항상 TRUE, FALSE 또는 NULL입니다. 일반적으로 조건 제어문과 SQL 데이터 조작문의 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.

4 오라클 데이터베이스 서버와 상호 작용: PL/SQL 프로그램의 SQL 문

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- PL/SQL 실행 블록에 직접 포함될 수 있는 SQL 문 판별
- PL/SQL에서 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.

목표

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

다루는 내용

- **PL/SQL을 사용하여 데이터 검색**
- **PL/SQL을 사용하여 데이터 조작**
- **SQL 커서 소개**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 조작어) 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 TABLE과 같은 DDL(데이터 정의어) 문을 직접 지원하지 않습니다. PL/SQL은 응용 프로그램에서 값을 전달하여 런타임에 데이터베이스 객체를 생성해야 할 경우에 발생할 수 없는 초기 바인딩을 지원합니다. 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. 동적 SQL 작업에 대한 자세한 내용은 *Oracle Database 11g: Develop PL/SQL Program Units* 과정에서 다룹니다.
- PL/SQL은 GRANT 또는 REVOKE와 같은 DCL(데이터 제어어) 문을 직접 지원하지 않습니다. 동적 SQL을 사용하여 DCL 문을 실행할 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/SQL에서 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 한 행만 패치(fetch)해야 합니다. 이 경우 WHERE 절 사용은 필수입니다.

PL/SQL의 SELECT 문 (계속)

- 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 한 행만 반환해야 합니다.

```

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

```

anonymous block completed
First Name is : Jennifer

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 한 행만 반환해야 합니다.

PL/SQL 블록 내의 SELECT 문에는 내장 SQL의 ANSI 규정에 따라 query는 한 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. 예외 처리는 "예외 처리" 단원에서 배웁니다.

참고: 코드 예제에서 DBMS_OUTPUT.PUT_LINE이 사용되는 모든 경우 SET SERVEROUTPUT ON 문이 블록 앞에 놓입니다.

PL/SQL의 SELECT 문 (계속)

테이블에서 여러 행을 검색하고 데이터를 조작하는 방법

INTO 절이 포함된 SELECT 문은 한 번에 한 행만 검색할 수 있습니다. 여러 행을 검색하고 데이터를 조작해야 할 경우 명시적 커서를 사용하면 됩니다. 이 단원 뒷부분에서 커서를 소개하고 "명시적 커서 사용" 단원에서 명시적 커서에 대해 배웁니다.

PL/SQL을 사용하여 데이터 검색: 예제

지정된 사원의 hire_date 및 salary를 검색합니다.

```

DECLARE
    v_emp_hiredate    employees.hire_date%TYPE;
    v_emp_salary       employees.salary%TYPE;
BEGIN
    SELECT    hire_date, salary
    INTO      v_emp_hiredate, v_emp_salary
    FROM      employees
    WHERE     employee_id = 100;
    DBMS_OUTPUT.PUT_LINE ('Hire date is :'|| v_emp_hiredate);
    DBMS_OUTPUT.PUT_LINE ('Salary is :'|| v_emp_salary);
END;
/

```

```

anonymous block completed
Hire date is : 17-JUN-87
Salary  is : 24000

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL을 사용하여 데이터 검색

슬라이드의 예제에서 v_emp_hiredate 및 v_emp_salary 변수는 PL/SQL 블록의 선언 섹션에 선언됩니다. 실행 섹션에서 employee_id가 100인 사원에 대한 hire_date 열의 값과 salary 열의 값이 employees 테이블에서 검색된 다음 emp_hiredate 변수와 emp_salary 변수에 각각 저장됩니다. INTO 절을 SELECT 문과 함께 사용하여 데이터베이스 열 값을 검색하고 값을 PL/SQL 변수에 저장하는 방법을 살펴보십시오.

참고: SELECT 문은 hire_date를 검색한 다음 salary를 검색합니다. 따라서 INTO 절에 있는 변수의 순서도 동일해야 합니다. 예를 들어, 슬라이드의 명령문에서 v_emp_hiredate와 v_emp_salary를 교환하면 명령문에 오류가 발생합니다.

PL/SQL을 사용하여 데이터 검색

지정된 부서에 속한 모든 사원의 급여 합계를 반환합니다.

예제:

```

DECLARE
    v_sum_sal    NUMBER(10,2);
    v_deptno     NUMBER NOT NULL := 60;
BEGIN
    SELECT SUM(salary) -- group function
    INTO v_sum_sal FROM employees
    WHERE department_id = v_deptno;
    DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;

```

anonymous block completed
The sum of salary is 28800

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL을 사용하여 데이터 검색 (계속)

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

참고: 그룹 함수는 PL/SQL 구문에서 사용할 수 없습니다. 슬라이드의 예제와 같이 이 함수는 PL/SQL 블록 내의 SQL 문에 사용됩니다.

예를 들어, 다음 구문을 사용하여 그룹 함수를 사용할 수 없습니다.

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

이름 지정 모호성

```

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 모호성

모호할 가능성이 있는 SQL 문에서 데이터베이스 열의 이름은 로컬 변수의 이름보다 우선합니다. 슬라이드에 표시된 예제는 다음과 같이 정의됩니다. `employee_id` 176의 `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 널이 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

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.

참고: SELECT 절의 모든 식별자는 데이터베이스 열 이름이기 때문에 SELECT 절에 모호성이 발생할 가능성은 없습니다. INTO 절의 식별자는 PL/SQL 변수이기 때문에 INTO 절에 모호성이 발생할 가능성은 없습니다. 혼동이 발생할 가능성은 WHERE 절에만 있습니다.

다루는 내용

- PL/SQL을 사용하여 데이터 검색
- PL/SQL을 사용하여 데이터 조작
- SQL 커서 소개

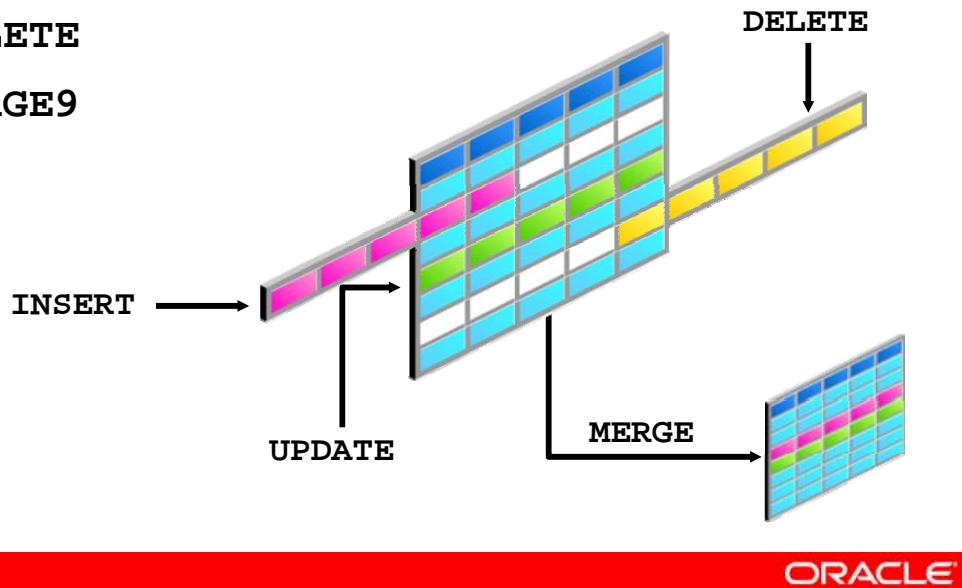
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL을 사용하여 데이터 조작

다음 DML 명령을 사용하여 데이터베이스 테이블을 변경합니다.

- **INSERT**
- **UPDATE**
- **DELETE**
- **MERGE**



Copyright © 2009, Oracle. All rights reserved.

ORACLE

PL/SQL을 사용하여 데이터 조작

DML 명령을 사용하여 데이터베이스의 데이터를 조작합니다. PL/SQL에서 INSERT, UPDATE, DELETE, MERGE 등의 DML 명령을 제약 없이 실행할 수 있습니다. 행 잠금 및 테이블 잠금은 PL/SQL 코드에 COMMIT 또는 ROLLBACK 문을 포함하여 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 아니면 대상 테이블에 삽입할지는 ON 절의 조건에 따라 결정됩니다.

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

데이터 삽입: 예제

EMPLOYEES 테이블에 새로운 사원 정보를 추가합니다.

```
BEGIN
  INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
  VALUES(employees_seq.NEXTVAL, 'Ruth', 'Cores',
    'RCORES', CURRENT_DATE, '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 CURRENT_DATE와 같은 SQL 함수 사용
- 기존 데이터베이스 시퀀스를 사용하여 Primary Key 값 생성
- PL/SQL 블록에서 값 파생

참고: employees 테이블의 데이터는 변경되지 않아야 합니다. employees 테이블이 읽기 전용이 아니더라도 code_04_15_s.sql 코드 예제와 같이 출력의 일관성을 보장하기 위해 이 테이블에서 삽입, 갱신 및 삭제가 허용되지 않습니다.

데이터 갱신: 예제

자재 담당자 직책을 가진 모든 사원의 급여를 인상합니다.

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

anonymous block completed	
FIRST_NAME	SALARY
Julia	4000
Irene	3500
James	3200
Steven	3000
...	

Curtis	3900
Randall	3400
Peter	3300

20 rows selected

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.

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

데이터 삭제: 예제

employees 테이블에서 부서 10에 속하는 행을 삭제합니다.

```
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.

```
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.

표시된 예제에서는 `copy_emp` 테이블의 `empno` 열을 `employees` 테이블의 `employee_id` 열과 일치시킵니다. 일치되는 행이 발견되면 해당 행이 `employees` 테이블의 행과 일치하도록 갱신됩니다. 행을 찾지 못하면 `copy_emp` 테이블에 행이 삽입됩니다.

다음 페이지에 PL/SQL 블록에서 MERGE를 사용하는 자세한 예제가 나와 있습니다.

행 병합 (계속)

```
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;  
/
```

다루는 내용

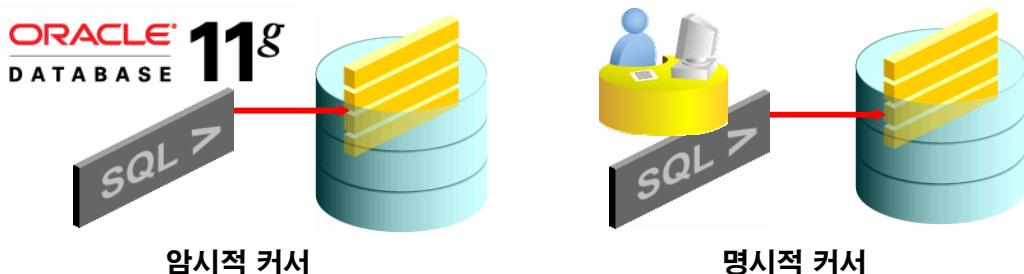
- PL/SQL을 사용하여 데이터 검색
- PL/SQL을 사용하여 데이터 조작
- SQL 커서 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 커서

- 커서는 Oracle 서버에서 할당한 전용(private) 메모리 영역에 대한 포인터입니다. 커서는 SELECT 문의 결과 집합을 처리하는데 사용됩니다.
- 암시적 커서와 명시적 커서라는 두 가지 유형의 커서가 있습니다.
 - 암시적: Oracle 서버에서 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.

Oracle 서버가 SQL 문을 처리하는 장소

Oracle 서버는 SQL 문을 처리하기 위해 컨텍스트 영역이라는 전용 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 유저가 제어할 수 없습니다.

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

커서 유형

다음과 같은 두 가지 유형의 커서가 있습니다.

- 암시적:** 암시적 커서는 Oracle 서버에서 생성하고 관리합니다. 프로그래머에게는 이 커서에 대한 액세스 권한이 없습니다. Oracle 서버는 SQL 문을 실행해야 하는 경우에 이러한 커서를 생성합니다.
- 명시적:** 프로그래머가 데이터베이스 테이블에서 여러 행을 검색하고 검색된 각 행에 포인터를 지정하며 한 번에 한 행씩 처리해야 하는 경우가 있습니다. 이러한 경우에 업무 요구 사항에 따라 명시적으로 커서를 선언할 수 있습니다. 프로그래머가 선언한 커서를 명시적 커서라고 합니다. 이러한 커서는 PL/SQL 블록의 선언 섹션에서 선언합니다.

암시적 커서에 대한 SQL 커서 속성

SQL 커서 속성을 사용하면 SQL 문의 결과를 테스트할 수 있습니다.

SQL%FOUND	가장 최근의 SQL 문이 한 행 이상에 영향을 미친 경우 TRUE로 평가되는 부울 속성
SQL%NOTFOUND	가장 최근의 SQL 문이 한 행에도 영향을 미치지 않은 경우 TRUE로 평가되는 부울 속성
SQL%ROWCOUNT	가장 최근의 SQL 문에 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 문이 아닌 PL/SQL 문에서 사용합니다.

해당 DML 명령이 실행된 후 블록의 실행 섹션에 있는 SQL%ROWCOUNT, SQL%FOUND 및 SQL%NOTFOUND 속성을 테스트하여 정보를 수집할 수 있습니다. DML 문이 기본 테이블의 행에 영향을 주지 않으면 PL/SQL은 오류를 반환하지 않습니다. 그러나 SELECT 문이 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 문에서 유용합니다.

명시적 커서 속성은 "명시적 커서 사용" 단원에서 배웁니다.

암시적 커서에 대한 SQL 커서 속성

`employees` 테이블에서 지정된 사원 ID를 가진 행을 삭제합니다. 삭제된 행 수를 출력합니다.

예제:

```
DECLARE
    v_rows_deleted VARCHAR2(30)
    v_empno employees.employee_id%TYPE := 176;
BEGIN
    DELETE FROM employees
    WHERE employee_id = v_empno;
    v_rows_deleted := (SQL%ROWCOUNT ||
        ' row deleted.');
    DBMS_OUTPUT.PUT_LINE (v_rows_deleted);
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

암시적 커서에 대한 SQL 커서 속성 (계속)

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

퀴즈

PL/SQL에서 SELECT 문을 사용할 경우 INTO 절이 필요하며 query에서 한 행 이상 반환할 수 있습니다.

- 1. 맞습니다.**
- 2. 틀립니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2

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 한 행만 반환해야 합니다.

PL/SQL 블록 내의 SELECT 문에는 내장 SQL의 ANSI 규정에 따라 query는 한 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 포함시켜 명령문이 단일 행을 반환하도록 합니다. 이 과정 뒷부분에서 예외 처리에 대해 배웁니다.

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 서버에서 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.

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
- LOOP 문 작성 및 식별
- 조건부 제어 구조 사용 지침 사용

ORACLE®

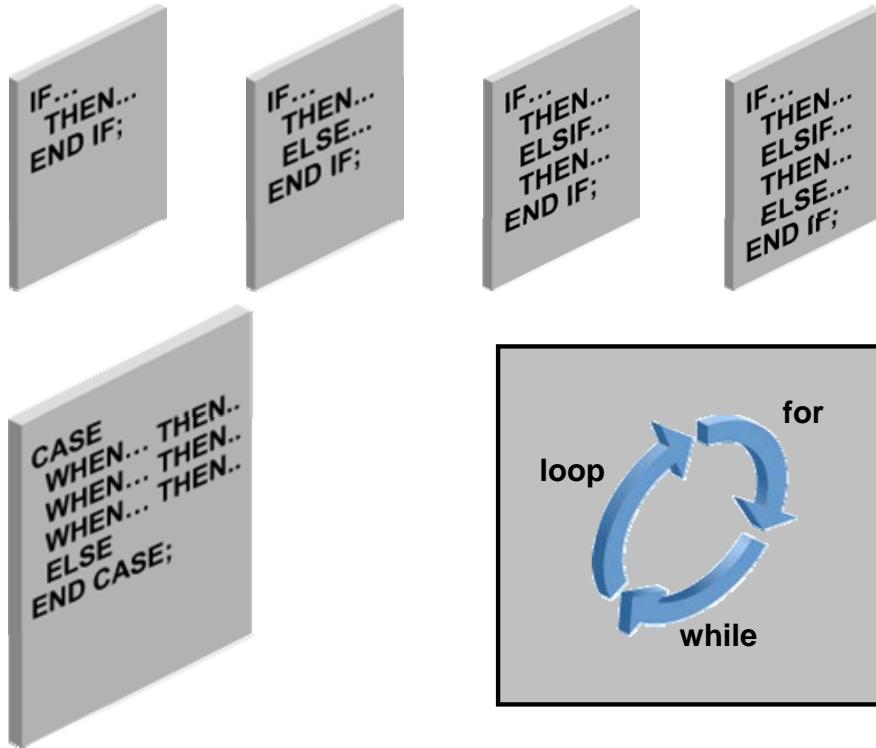
Copyright © 2009, Oracle. All rights reserved.

목표

이미 앞에서 선언 섹션 및 실행 섹션을 포함하는 PL/SQL 블록을 작성하는 방법에 대해 배웠습니다. 또한 실행 블록에 표현식과 SQL 문을 포함하는 방법에 대해서도 설명했습니다.

이 단원에서는 PL/SQL 블록에서 IF 문, CASE 식, LOOP 구조 등의 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 문이 있는 조건부 생성자, CASE 식이 있는 조건부 생성자, LOOP 제어 구조가 있는 조건부 생성자, CONTINUE 문이 있는 조건부 생성자 등의 네 가지 PL/SQL 제어 구조를 다룹니다.

다루는 내용

- **IF 문 사용**
- **CASE 문 및 CASE 식 사용**
- **LOOP 문 작성 및 식별**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

IF 문

Syntax:

```
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.

condition TRUE, FALSE 또는 NULL을 반환하는 부울 변수 또는 표현식입니다.

THEN 부울 표현식을 뒤에 나오는 명령문 시퀀스와 연관시키는 절을 시작합니다.

statements 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 선점하도록 차례로 테스트가 실행됩니다.

END IF IF 문의 끝을 표시합니다.

참고: 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
    v_myage number:=31;
BEGIN
    IF v_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

간단한 IF 예제

이 슬라이드는 THEN 절을 사용하는 간단한 IF 문의 예제를 보여줍니다.

- v_myage 변수는 31로 초기화됩니다.
- v_myage가 11보다 작지 않기 때문에 IF 문의 조건은 FALSE를 반환합니다.
- 따라서 이 제어는 THEN 절에 도달하지 않습니다.

조건식 추가

IF 문에는 AND, OR 및 NOT과 같은 논리 연산자와 관련된 다중 조건식이 있을 수 있습니다.

예를 들면 다음과 같습니다.

```

IF (myfirstname='Christopher' AND v_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

```
DECLARE
  v_myage number:=31;
BEGIN
  IF v_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.

블록의 출력은 코드 아래에 표시됩니다.

IF ELSIF ELSE 절

```

DECLARE
    v_myage number:=31;
BEGIN
    IF v_myage < 11 THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSIF v_myage < 20 THEN
        DBMS_OUTPUT.PUT_LINE(' I am young ');
    ELSIF v_myage < 30 THEN
        DBMS_OUTPUT.PUT_LINE(' I am in my twenties ');
    ELSIF v_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

IF 절은 다중 ELSIF 절과 단일 ELSE 절을 포함할 수 있습니다. 예제에서는 이 절의 다음 특징을 설명합니다.

- ELSE 절과 달리 ELSIF 절에는 조건이 있을 수 있습니다.
- ELSIF의 조건 뒤에는 ELSIF의 조건이 TRUE를 반환하는 경우에 실행되는 THEN 절이 와야 합니다.
- 다중 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.

예제에서 블록의 출력은 코드 아래에 표시됩니다.

IF 문의 NULL 값

```
DECLARE
    v_myage number;
BEGIN
    IF v_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 `v_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.
- 코드를 작성할 때는 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 수는 제한이 없습니다.
- 명확한 구분을 위해 조건부로 실행되는 명령문은 들여 씁니다.

다루는 내용

- **IF 문 사용**
- **CASE 문 및 CASE 식 사용**
- **LOOP 문 작성 및 식별**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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;
```



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 CASE 식의 WHEN 절은 모든 유형의 값을 생성할 수 있는 표현식이 아닌 부울 값을 생성하는 검색 조건을 포함합니다.

CASE 식: 예제

```

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

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

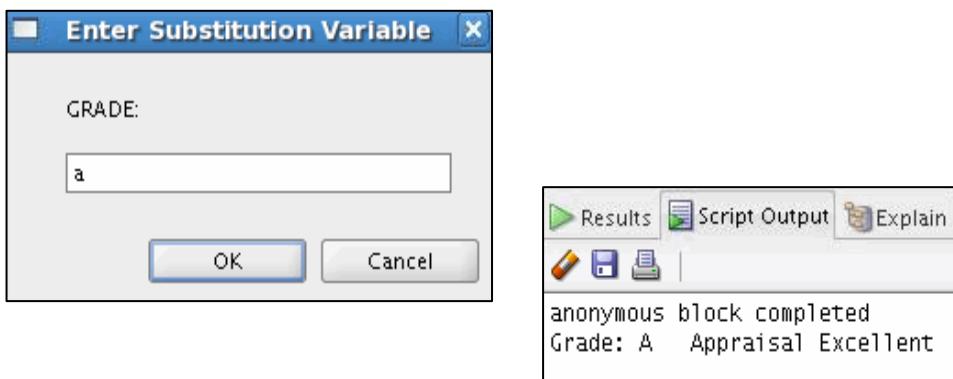
CASE 식: 예제

이 슬라이드의 예제에서 CASE 식은 v_grade 변수의 값을 표현식으로 사용합니다.

이 같은 치환 변수를 사용하여 유저로부터 받아들입니다. 유저가 입력한 값에 따라 CASE 식은 v_grade 값에 기반한 v_appraisal 변수 값을 반환합니다.

결과

Substitution Variable window에 표시된 것처럼 v_grade에 대해 a 또는 A를 입력할 경우 예제의 출력은 다음과 같습니다.



검색된 CASE 식

```

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

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

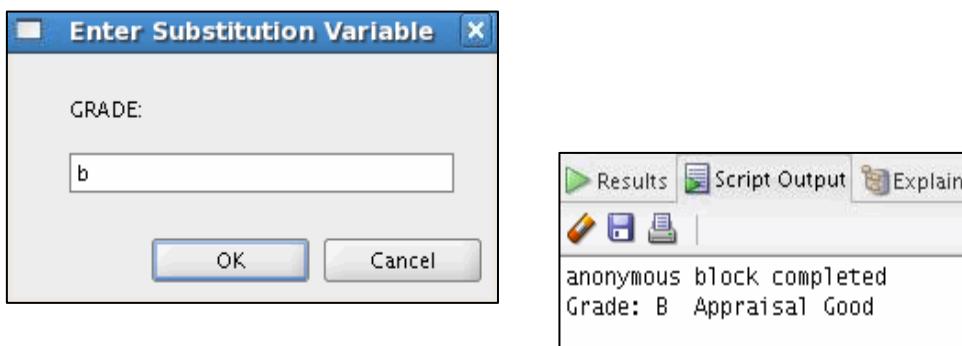
검색된 CASE 식

앞의 예제에서는 하나의 테스트 표현식(v_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.

결과

v_grade 값으로 b 또는 B를 입력할 경우 예제의 출력은 다음과 같습니다.



CASE 문

```
DECLARE
    v_deptid NUMBER;
    v_deptname VARCHAR2(20);
    v_emps NUMBER;
    v_mngid NUMBER:= 108;
BEGIN
    CASE v_mngid
    WHEN 108 THEN
        SELECT department_id, department_name
        INTO v_deptid, v_deptname FROM departments
        WHERE manager_id=108;
        SELECT count(*) INTO v_emps FROM employees
        WHERE department_id=v_deptid;
    WHEN 200 THEN
        ...
    END CASE;
    DBMS_OUTPUT.PUT_LINE ('You are working in the '|| v_deptname ||
    ' department. There are '||v_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 포함할 수 있습니다. 마찬가지로 CASE 문에 명령문을 포함하면 다중 IF 및 ELSIF 문보다 더 읽기 쉽습니다.

CASE 식과 CASE 문의 차이점

CASE 식은 조건을 평가하여 값을 반환하는 반면에 CASE 문은 조건을 평가하여 작업을 수행합니다. CASE 문은 완전한 PL/SQL 블록이 될 수 있습니다.

- CASE 문은 END CASE;로 끝납니다.
- CASE 식은 END;로 끝납니다.

슬라이드 코드 예제의 출력은 다음과 같습니다.

The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the following text:

```
anonymous block completed
You are working in the Finance department. There are 6 employees in this department
```

참고: IF 문은 아무 작업도 수행할 수 없지만 (조건은 모두 False일 수 있으며 ELSE 절은 필수가 아님) CASE 문은 특정 PL/SQL 문을 실행해야 합니다.

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 반환하는 데 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.

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

부울 식 또는 논리식?

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 식 또는 논리식?

AND 논리 테이블은 슬라이드에서 부울 조건의 확률을 평가하는 데 도움이 됩니다.

해답

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

다루는 내용

- **IF 문 사용**
- **CASE 문 및 CASE 식 사용**
- **LOOP 문 작성 및 식별**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 유형의 제어 구조입니다. 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

참고: 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

가장 간단한 형식의 LOOP 문은 LOOP 키워드와 END LOOP 키워드로 명령문 시퀀스를 묶는 기본 루프입니다. 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 종료를 활성화하도록 WHEN 절을 추가할 수 있습니다. EXIT 문에 도달하면 WHEN 절의 조건이 평가됩니다. 조건에서 TRUE를 반환하면 루프가 종료되고 제어가 루프 다음의 명령문으로 이동합니다. 기본 루프는 다중 EXIT 문을 포함할 수 있지만 EXIT 포인트는 하나만 있는 것이 좋습니다.

기본 루프: 예제

```

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

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 루프: 예제

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

참고

- 기본 루프를 사용하면 EXIT WHEN 조건이 충족될 때까지 명령문을 실행할 수 있습니다.
- 조건이 LOOP 문 다음에 검사되도록 루프 안에 배치되면 루프가 적어도 한 번 실행됩니다.
- 그러나 종료 조건이 다른 모든 실행문 앞, 즉 루프 위쪽에 있는 경우 해당 조건이 true이면 루프가 종료되고 명령문이 실행되지 않습니다.

결과

출력을 보려면 code_05_22_s.sql 코드 예제를 실행합니다.

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. 따라서 루프 내의 어떠한 명령문도 실행되지 않을 수 있습니다.

이 구문에서 다음이 적용됩니다

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

statement 하나 이상의 PL/SOL 또는 SOL 문일 수 있습니다.

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

참고: 조건이 NULL을 반환하면 루프가 통과되고 제어가 다음 명령문으로 이동합니다.

WHILE 루프: 예제

```

DECLARE
    v_countryid    locations.country_id%TYPE := 'CA';
    v_loc_id        locations.location_id%TYPE;
    v_new_city      locations.city%TYPE := 'Montreal';
    v_counter       NUMBER := 1;
BEGIN
    SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
    WHILE v_counter <= 3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
        v_counter := v_counter + 1;
    END LOOP;
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WHILE 루프: 예제

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

- WHILE 루프를 통한 매회 반복 시 카운터 (v_counter) 가 증가합니다.
- 반복 횟수가 3회보다 작거나 같으면 루프 안의 코드가 실행되고 행이 locations 테이블에 삽입됩니다.
- v_counter가 이 도시와 국가에 대한 새 위치 수를 초과하면 루프를 제어하는 조건이 FALSE로 평가되고 루프가 종료됩니다.

결과

출력을 보려면 code_05_24_s.sql 코드 예제를 실행합니다.

FOR 루프

- 반복 횟수에 대한 테스트를 단축하려면 FOR 루프를 사용합니다.
- 카운터는 암시적으로 선언되므로 선언하지 않아도 됩니다.

```
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 앞에는 PL/SQL에서 실행하는 반복 횟수를 설정하는 제어문이 있습니다.

이 구문에서 다음이 적용됩니다.

<i>counter</i>	상한이나 하한에 도달할 때까지 루프 반복 시마다 값이 자동으로 1씩 증가 또는 감소(REVERSE 키워드를 사용하는 경우 감소)하는 암시적으로 선언되는 정수입니다.
REVERSE	반복할 때마다 카운터를 상한에서 하한으로 감소시킵니다. 참고: 여전히 하한이 먼저 참조됩니다.
<i>lower_bound</i>	카운터 값 범위에 대한 하한을 지정합니다.
<i>upper_bound</i>	카운터 값 범위에 대한 상한을 지정합니다.

카운터를 선언하지 마십시오. 암시적으로 정수로 선언됩니다.

FOR 루프 (계속)

참고: 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 $8/5$ 는 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
    v_countryid    locations.country_id%TYPE := 'CA';
    v_loc_id        locations.location_id%TYPE;
    v_new_city      locations.city%TYPE := 'Montreal';
BEGIN
    SELECT MAX(location_id) INTO v_loc_id
    FROM locations
    WHERE country_id = v_countryid;
    FOR i IN 1..3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + i), v_new_city, v_countryid );
    END LOOP;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FOR 루프: 예제

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

결과

출력을 보려면 code_05_27_s.sql 코드 예제를 실행합니다.

FOR 루프 규칙

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

FOR 루프 규칙

이 슬라이드는 FOR 루프 작성 시 따라야 하는 지침을 나열합니다.

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

예제:

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

루프의 권장 사용법

- **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.

루프의 권장 사용법

기본 루프를 사용하면 루프를 시작할 때 조건이 충족되더라도 명령문을 적어도 한 번 실행할 수 있습니다. EXIT 문이 없으면 루프는 무한히 반복됩니다.

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

FOR 루프에는 LOOP 키워드 앞에 PL/SQL에서 실행할 반복 횟수를 설정하는 제어문이 있습니다. 반복 횟수가 미리 정의된 경우에는 FOR 루프를 사용합니다.

중첩 루프 및 레이블

- **다중 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 예외가 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. 루프의 레이블이 지정되어 있으면 명확성을 위해 END LOOP 문 다음에 레이블 이름을 포함시킬 수 있습니다(선택 사항).

중첩 루프 및 레이블: 예제

```
...
BEGIN
    <<Outer_loop>>
    LOOP
        v_counter := v_counter+1;
        EXIT WHEN v_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: 예제

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.

PL/SQL CONTINUE 문

- 정의
 - 다음 루프 반복을 시작하는 기능을 추가합니다.
 - 프로그래머에게 제어를 다음 루프 반복으로 전이하는 기능을 제공합니다.
 - EXIT 문에 대한 의미와 병렬 구조를 사용합니다.
- 이점
 - 프로그래밍 프로세스를 용이하게 합니다.
 - CONTINUE 문을 시뮬레이트하는 이전 프로그래밍 해결책보다 다소 성능이 향상될 수 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL CONTINUE 문

CONTINUE 문을 사용하면 루프 내의 제어를 새 반복으로 다시 전이하거나 루프를 벗어날 수 있습니다. 다른 많은 프로그래밍 언어에 이 기능이 있습니다. Oracle Database 11g 버전을 사용할 경우 PL/SQL에서도 이 기능을 제공합니다. Oracle Database 11g 버전 이전에는 부울 변수와 조건문을 통해 CONTINUE 프로그램 기능을 시뮬레이트하여 해결책을 코딩할 수 있었습니다. 어떤 경우에는 이러한 해결책이 덜 효율적입니다.

CONTINUE 문을 사용하면 단순화된 방법으로 루프 반복을 제어할 수 있으며 이전 코딩 해결책보다 더 효율적일 수 있습니다.

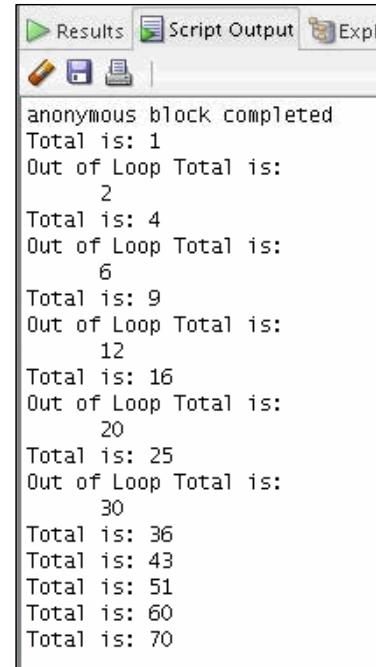
CONTINUE 문은 일반적으로 주 처리가 시작되기 전에 루프 본문 내의 데이터를 필터링하는데 사용됩니다.

PL/SQL CONTINUE 문: 예제 1

```

DECLARE
    v_total SIMPLE_INTEGER := 0;
BEGIN
    FOR i IN 1..10 LOOP
        1 v_total := v_total + i;
        dbms_output.put_line
            ('Total is: '|| v_total);
        CONTINUE WHEN i > 5;
        2 v_total := v_total + i;
        dbms_output.put_line
            ('Out of Loop Total is:
             '|| v_total);
    END LOOP;
END;
/

```



```

Results Script Output Expl
anonymous block completed
Total is: 1
Out of Loop Total is:
2
Total is: 4
Out of Loop Total is:
6
Total is: 9
Out of Loop Total is:
12
Total is: 16
Out of Loop Total is:
20
Total is: 25
Out of Loop Total is:
30
Total is: 36
Total is: 43
Total is: 51
Total is: 60
Total is: 70

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL CONTINUE 문: 예제 1

예제에는 `v_total` 변수를 사용하는 할당이 두 개 있습니다.

1. 첫번째 할당은 루프의 10회 반복 각각에 대해 실행됩니다.
2. 두번째 할당은 루프의 처음 5회 반복에 대해 실행됩니다. `CONTINUE` 문이 루프 내의 제어를 새 반복으로 다시 전이하므로 루프의 마지막 5회 반복에 대해서는 두번째 TOTAL 할당이 실행되지 않습니다.

TOTAL 변수의 최종 결과는 70입니다.

PL/SQL CONTINUE 문: 예제 2

```
DECLARE
  v_total NUMBER := 0;
BEGIN
  <<BeforeTopLoop>>
  FOR i IN 1..10 LOOP
    v_total := v_total + 1;
    dbms_output.put_line
      ('Total is: ' || v_total);
    FOR j IN 1..10 LOOP
      CONTINUE BeforeTopLoop WHEN i + j > 5;
      v_total := v_total + 1;
    END LOOP;
  END LOOP;
END two_loop;
```



Results	Script Output	Exp
anonymous block completed		
Total is: 1		
Total is: 6		
Total is: 10		
Total is: 13		
Total is: 15		
Total is: 16		
Total is: 17		
Total is: 18		
Total is: 19		
Total is: 20		

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL CONTINUE 문: 예제 2

CONTINUE 문을 사용하여 외부 루프의 다음 반복으로 건너 뛸 수 있습니다. 이렇게 하려면 외부 루프에 CONTINUE 문이 이동해야 할 위치를 식별하는 레이블을 지정하십시오.

맨 안쪽 루프에 있는 CONTINUE 문은 WHEN 조건이 true일 때마다 해당 루프를 종료합니다 (EXIT 키워드와 마찬가지임). 이 예제에서는 맨 안쪽 루프가 CONTINUE 문에 의해 종료된 후 BeforeTopLoop라는 레이블이 지정된 맨 바깥쪽 루프의 다음 반복으로 제어가 전이됩니다. 이 루프 쌍(pair)이 완료될 때 TOTAL 변수의 값은 20입니다.

코드의 내부 블록 안에서 CONTINUE 문을 사용할 수도 있습니다. 그러면 블록이 해당 외부 루프 안에 중첩되는 경우 루프가 포함되지 않습니다.

제한 사항

- CONTINUE 문은 루프 밖에 나타날 수 없습니다. 그러면 컴파일러 오류가 발생합니다.
- 프로시저, 함수 또는 메소드 경계를 통과하기 위해 CONTINUE 문을 사용할 수 없습니다. 그러면 컴파일러 오류가 발생합니다.

퀴즈

기본, FOR 및 WHILE이라는 세 가지 유형의 루프가 있습니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1

루프 유형

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

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
- CONTINUE 문

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

업무 논리 구현을 위한 제어 구조를 제공하는 경우에만 언어를 프로그래밍 언어라고 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
- LOOP 구조를 사용하여 반복 단계 수행

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 5: 개요

이 연습에서는 루프와 조건부 제어 구조를 통합하는 PL/SQL 블록을 생성합니다.

연습 문제에서는 다양한 IF 문과 LOOP 생성자 작성에 대한 이해도를 테스트합니다.

조합 데이터 유형 작업

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- PL/SQL 컬렉션 및 레코드 설명
- 유저 정의 PL/SQL 레코드 생성
- %ROWTYPE 속성을 사용하여 PL/SQL 레코드 생성
- 연관 배열 생성
 - INDEX BY 테이블
 - INDEX BY 레코드 테이블

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

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

다루는 내용

- 조합 데이터 유형 소개
- PL/SQL 레코드 사용
 - PL/SQL 레코드를 사용하여 데이터 조작
 - %ROWTYPE 속성의 이점
- PL/SQL 컬렉션 사용
 - 연관 배열 검사
 - 중첩 테이블 소개
 - VARRAY 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

조합 데이터 유형

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

조합 데이터 유형을 사용해야 하는 이유

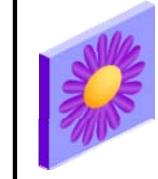
모든 관련 데이터를 단일 단위로 보유하게 됩니다. 데이터를 쉽게 액세스하고 수정할 수 있습니다. 데이터가 조합 유형인 경우 데이터를 보다 쉽게 관리, 연관 및 전송할 수 있습니다. 쉽게 설명하자면 랙톱의 각 구성 요소를 별개의 가방에 넣는 대신 모든 구성 요소를 하나의 가방에 넣는 것과 같습니다.

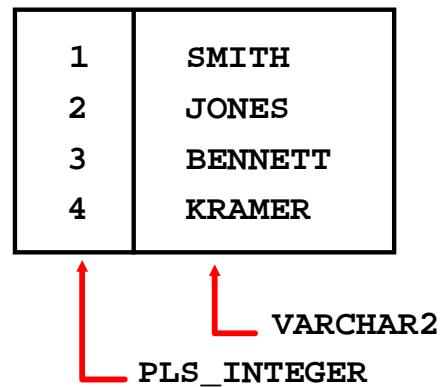
PL/SQL 레코드 또는 컬렉션?

- 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.

PL/SQL 컬렉션:

PL/SQL 레코드:

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 또는 컬렉션?

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. 예를 들어, 사원 세부 정보를 보유할 PL/SQL 레코드를 생성하고, 저장된 모든 값은 특정 사원에 대한 정보를 제공하므로 서로 관련되어 있음을 표시할 수 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.

다루는 내용

- 조합 데이터 유형 검사
- PL/SQL 레코드 사용
 - PL/SQL 레코드를 사용하여 데이터 조작
 - %ROWTYPE 속성의 이점
- PL/SQL 컬렉션 사용
 - 연관 배열 검사
 - 중첩 테이블 소개
 - VARRAY 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드

- **스칼라, RECORD 또는 INDEX BY 테이블 데이터 유형의 구성 요소(필드라고 함)를 하나 이상 포함해야 합니다.**
- **C와 C++를 포함한 대부분의 3세대 언어 구조와 유사합니다.**
- **사용자 정의 데이터 유형이며 테이블 행의 부분 집합일 수 있습니다.**
- **필드 컬렉션을 논리적 단위로 처리합니다.**
- **테이블에서 데이터 행을 패치(fetch)하여 처리하는 데 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.
- 필드를 초기화할 때 DEFAULT 키워드와 :=를 사용할 수 있습니다.
- 블록, 서브 프로그램 또는 패키지의 선언 부분에서 RECORD 유형을 정의하고 유저 정의 레코드를 선언할 수 있습니다.
- 중첩 레코드를 선언하고 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. 이 레코드를 사용하려면 다음 단계를 수행하십시오.

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 조건을 사용하면 지정된 필드에 널이 할당되지 않습니다. NOT NULL 필드를 초기화해야 합니다.

PL/SQL 레코드 구조

필드 선언



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 컬렉션에 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;
```

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 숫자이고 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.

identifier 레코드 전체에 대해 선택한 이름입니다.

reference 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.

참고: 이 레코드는 코드가 아니며 단순히 조합 변수의 구조입니다.

```
(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
```

그런 다음 아래와 같이 레코드 필드에 값을 할당할 수 있습니다.

```
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. `employees%ROWTYPE` 유형의 레코드와 `employees` 테이블의 유사 필드를 가지는 유저 정의 레코드는 동일한 데이터 유형을 가집니다. 따라서 유저 정의 레코드에 `%ROWTYPE` 레코드의 필드와 유사한 필드가 포함되어 있으면 해당 유저 정의 레코드를 `%ROWTYPE` 레코드에 할당할 수 있습니다.

PL/SQL 레코드 생성: 예제

```

DECLARE
  TYPE t_rec IS RECORD
    (v_sal number(8),
     v_minsal number(8) default 1000,
     v_hire_date employees.hire_date%type,
     v_rec1 employees%rowtype);
  v_myrec t_rec;
BEGIN
  v_myrec.v_sal := v_myrec.v_minsal + 500;
  v_myrec.v_hire_date := sysdate;
  SELECT * INTO v_myrec.v_rec1
    FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name ||' ' ||
    to_char(v_myrec.v_hire_date) ||' '|| to_char(v_myrec.v_sal));
END;

```

anonymous block completed
 King 16-FEB-09 1500

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 생성: 예제

레코드 정의에 사용되는 필드 선언은 변수 선언과 유사합니다. 필드마다 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 예제에서는 필수적인 2단계 프로세스를 사용하여 PL/SQL 레코드가 생성됩니다.

1. 레코드 유형(t_rec)이 정의됩니다.
2. t_rec 유형의 레코드(v_myrec)가 선언됩니다.

참고

- 레코드에는 v_sal, v_minsal, v_hire_date, v_rec1 등의 네 가지 필드가 포함됩니다.
- v_rec1은 %TYPE 속성과 비슷한 %ROWTYPE 속성을 사용하여 정의됩니다. %TYPE을 사용하면 필드는 지정된 열의 데이터 유형을 상속합니다. %ROWTYPE을 사용하면 필드는 참조된 테이블에 있는 모든 열의 열 이름과 데이터 유형을 상속합니다.
- PL/SQL 레코드 필드는 %ROWTYPE 속성으로 정의되는 <record>.<field> 표기법이나 <record>.<field>.<column> 표기법을 사용하여 참조됩니다.
- 필드 선언에 NOT NULL 제약 조건을 추가하여 해당 필드에 널이 할당되지 않도록 할 수 있습니다. NOT NULL로 선언되는 필드는 반드시 초기화해야 합니다.

%ROWTYPE 속성 사용 시 이점

- 기본 데이터베이스 열의 개수와 데이터 유형을 알 필요가 없으며 실제로 런타임에 변경될 수 있습니다.
- %ROWTYPE 속성은 다음을 사용하여 행을 검색하려는 경우에 유용합니다.
 - SELECT * 문
 - 행 레벨 INSERT 및 UPDATE 문

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 검색되는 각 열에 대해 일일이 변수를 선언해야 합니다.

또 다른 %ROWTYPE 속성 예제

```

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

```

The screenshot shows the Oracle SQL Developer interface. In the top-left pane, there is a code editor containing the provided PL/SQL code. In the bottom-right pane, there is a results grid showing the output of the query. The results grid has columns labeled: EMPNO, ENAME, JOB, MGR, HIREDATE, LEAVEDATE, SAL, COMM, DEPTNO. There is one row of data: 1, 124, Mourgos, ST_MAN, 100, 16-NOV-99, 16-JUN-09, 5800, (null), 50.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

또 다른 %ROWTYPE 속성 예제

%ROWTYPE 속성의 또 다른 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.

다음은 retired_emps 테이블을 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 ))

```

참고

- retired_emps 테이블에 삽입되는 레코드가 슬라이드에 표시됩니다.
- 슬라이드에 표시된 출력을 보려면 SQL Developer에서 코드 예제 하단의 SELECT 문에 커서를 놓고 F9 키를 누르십시오.
- 전체 코드 예제는 code_6_14_n-s.sql에 있습니다.

%ROWTYPE을 사용하여 레코드 삽입

```

...
DECLARE
    v_employee_number number:= 124;
    v_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 v_emp_rec FROM employees
    WHERE employee_id = v_employee_number;
    INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
SELECT * FROM retired_emps;

```

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

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 코드를 쉽게 읽을 수 있습니다.

슬라이드의 `SELECT` 문을 살펴보십시오. `hire_date`를 두 번 선택하고 `hire_date` 값을 `retired_emps`의 `leavedate` 필드에 삽입합니다. 채용 날짜에 퇴직하는 사원은 없습니다. 삽입된 레코드가 슬라이드에 표시됩니다. 다음 슬라이드에서는 이러한 레코드를 생성하는 방법에 대해 설명합니다.

참고: 슬라이드에 표시된 출력을 보려면 SQL Developer에서 코드 예제 하단의 `SELECT` 문에 커서를 놓고 F9 키를 누르십시오.

레코드를 사용하여 테이블의 행 갱신

```

SET VERIFY OFF
DECLARE
    v_employee_number number:= 124;
    v_emp_rec retired_emps%ROWTYPE;
BEGIN
    SELECT * INTO v_emp_rec FROM retired_emps;
    v_emp_rec.leavedate:=CURRENT_DATE;
    UPDATE retired_emps SET ROW = v_emp_rec WHERE
        empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;

```

Results										
Script Output Explain Autotrace DBMS Output OWA Output										
Results:										
	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO	
1	124	Mourgos	ST_MAN	100	16-NOV-99	16-NOV-99	5800	(null)	50	

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.
- 슬라이드에 표시된 것과 같이 레코드가 갱신됩니다.

참고: 슬라이드에 표시된 출력을 보려면 SQL Developer에서 코드 예제 하단의 SELECT 문에 커서를 놓고 F9 키를 누르십시오.

다루는 내용

- 조합 데이터 유형 검사
- PL/SQL 레코드 사용
 - PL/SQL 레코드를 사용하여 데이터 조작
 - %ROWTYPE 속성의 이점
- PL/SQL 컬렉션 사용
 - 연관 배열 검사
 - 중첩 테이블 소개
 - VARRAY 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다루는 내용

앞에서 설명한 바와 같이, 동일한 데이터 유형의 값을 저장하려는 경우 PL/SQL 컬렉션을 사용합니다. 이 데이터 유형은 레코드와 같은 조합 유형일 수도 있습니다.

따라서 컬렉션은 데이터를 단일 단위로 처리하는 데 사용됩니다. 컬렉션의 유형은 다음 세 가지입니다.

- 연관 배열
- 중첩 테이블
- VARRAY

참고: 이 단원에서는 이 세 가지 컬렉션 중 연관 배열을 중점적으로 다룹니다. 중첩 테이블과 VARRAY는 비교 목적으로만 소개됩니다. 이 두 컬렉션은 *Oracle Database 11g: Advanced PL/SQL* 과정에서 자세히 다룹니다.

연관 배열(INDEX BY 테이블)

연관 배열은 다음 두 열이 있는 PL/SQL 컬렉션입니다.

- 정수 또는 문자열 데이터 유형의 Primary Key
- 스칼라 또는 레코드 데이터 유형의 열

키	값
1	JONES
2	HARDEY
3	MADURO
4	KRAMER

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

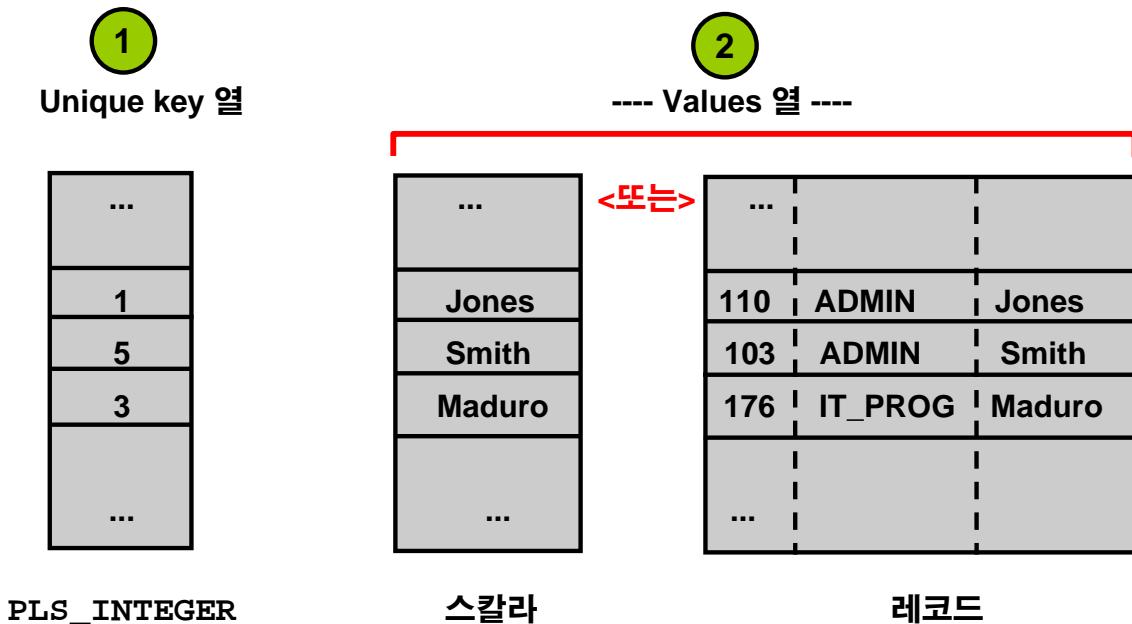
연관 배열(INDEX BY 테이블)

연관 배열은 PL/SQL 컬렉션의 한 유형으로 사용자 정의 조합 데이터 유형입니다. 연관 배열은 키-값 쌍의 집합입니다. 연관 배열은 Primary Key값을 인덱스로 사용하여 데이터를 저장할 수 있습니다. 여기서 키 값은 순차적이지 않을 수도 있습니다. 연관 배열은 *INDEX BY* 테이블이라고도 합니다.

연관 배열에는 두 열만 있습니다. 두 열 모두 이름을 지정할 수 없습니다.

- 정수 또는 문자열 유형의 첫번째 열은 Primary Key역할을 합니다.
- 스칼라 또는 레코드 데이터 유형의 두번째 열은 값을 보유합니다.

연관 배열 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

연관 배열 구조

앞에서 설명한 바와 같이, 연관 배열에는 두 열이 있습니다. 두번째 열은 행당 하나의 값을 보유하거나 여러 값을 보유합니다.

Unique Key 열: 키 열의 데이터 유형은 다음일 수 있습니다.

- BINARY_INTEGER 또는 PLS_INTEGER인 숫자. 이 두 가지 숫자 데이터 유형은 NUMBER보다 적은 저장 영역이 필요하며 해당 데이터 유형에 대한 산술 연산은 NUMBER 산술보다 빠릅니다.
- VARCHAR2 또는 하위 유형 중 하나

"Value" 열: Value열은 스칼라 데이터 유형 또는 레코드 데이터 유형일 수 있습니다. 스칼라 데이터 유형의 열은 행당 하나의 값만 보유할 수 있지만, 레코드 데이터 유형의 열은 행당 여러 값을 보유할 수 있습니다.

기타 특성

- 연관 배열은 선언 시 채워지지 않으며 키나 값을 포함하지 않으므로 선언에서 연관 배열을 초기화할 수 없습니다.
- 연관 배열을 채우려면 명시적 실행문이 필요합니다.
- 데이터베이스 테이블의 크기와 마찬가지로 연관 배열의 크기에도 제약이 없습니다. 따라서 새 행이 추가됨에 따라 연관 배열이 증가하도록 행 수가 동적으로 늘어날 수 있습니다. 키는 순차적이 아닐 수 있으며 양수 및 음수일 수 있습니다.

연관 배열 생성 단계

구문:

```

1   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>);

2   identifier type_name;

```

예제:

```

...
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.

연관 배열 생성 단계

연관 배열을 생성하려면 다음 2단계를 수행해야 합니다.

1. INDEX BY 옵션을 사용하여 TABLE 데이터 유형을 선언합니다.
2. 해당 데이터 유형의 변수를 선언합니다.

구문

type_name Is the name of the TABLE type (This name is used in the subsequent declaration of the array identifier.)

column_type Is any scalar or composite data type such as VARCHAR2, DATE, NUMBER, or %TYPE (You can use the %TYPE attribute to provide the column data type.)

identifier Is the name of the identifier that represents an entire associative array

참고: NOT NULL 제약 조건을 사용하면 연관 배열에 널이 할당되지 않습니다.

예제

예제에서는 사원의 성을 저장하기 위해 변수 이름 `ename_table` 포함된 연관 배열이 선언됩니다.

연관 배열 생성 및 액세스

```

...
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;
/
...

```

anonymous block completed	
ENAME	HIREDT
CAMERON	23-JUN-09
1 rows selected	

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연관 배열 생성 및 액세스

슬라이드의 예제에서는 ename_table 및 hiredate_table 식별자를 사용하여 두 개의 연관 배열을 생성합니다.

각 연관 배열의 키는 다음 구문을 사용하여 배열의 요소에 액세스하는 데 사용됩니다.

식별자(index)

두 배열 모두에서 인덱스 값은 PLS_INTEGER 유형에 속합니다.

- ename_table 연관 배열의 첫번째 행을 참조하려면 ename_table(1)을 지정합니다.
- hiredate_table 연관 배열의 여덟번째 행을 참조하려면 hiredate_table(8)을 지정합니다.

참고

- PLS_INTEGER의 크기 범위가 $-2,147,483,647 \sim 2,147,483,647$ 으로 Primary Key 값은 음수가 될 수 있습니다. 인덱스는 1로 시작할 필요가 없습니다.
- i 인덱스가 있는 행이 반환되면 exists(i) 메소드는 TRUE를 반환합니다. 존재하지 않는 데이터 요소 참조 시 발생하는 오류를 방지하려면 exists 메소드를 사용하십시오.
- 전체 코드 예제는 code_6_21_s.sql에 있습니다.

INDEX BY 테이블 메소드 사용

다음 메소드를 사용하면 연관 배열을 더 쉽게 사용할 수 있습니다.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 테이블 메소드 사용

INDEX BY 테이블 메소드는 연관 배열에서 작동하는 내장 프로시저 또는 함수이며 점 표기법을 사용하여 호출됩니다.

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

메소드	설명
EXISTS(<i>n</i>)	연관 배열에 <i>n</i> 번째 요소가 존재하면 TRUE를 반환합니다.
COUNT	현재 연관 배열에 포함된 요소 수를 반환합니다.
FIRST	<ul style="list-style-type: none"> • 연관 배열에서 첫번째(가장 작은) 인덱스 번호를 반환합니다. • 연관 배열이 비어 있으면 NULL을 반환합니다.
LAST	<ul style="list-style-type: none"> • 연관 배열에서 마지막(가장 큰) 인덱스 번호를 반환합니다. • 연관 배열이 비어 있으면 NULL을 반환합니다.
PRIOR(<i>n</i>)	연관 배열에서 인덱스 <i>n</i> 앞에 나오는 인덱스 번호를 반환합니다.
NEXT(<i>n</i>)	연관 배열에서 인덱스 <i>n</i> 뒤에 나오는 인덱스 번호를 반환합니다.
DELETE	<ul style="list-style-type: none"> • DELETE 연관 배열에서 모든 요소를 제거합니다. • DELETE(<i>n</i>) 연관 배열에서 <i>n</i>번째 요소를 제거합니다. • DELETE(<i>m, n</i>) 연관 배열에서 <i>m ... n</i> 범위의 모든 요소를 제거합니다.

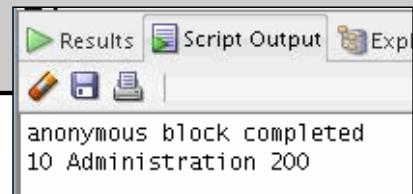
INDEX BY 레코드 테이블 옵션

테이블의 전체 행을 보유하도록 연관 배열을 정의합니다.

```

DECLARE
    TYPE dept_table_type IS TABLE OF
        departments%ROWTYPE INDEX PLS_INTEGER;
    dept_table dept_table_type;
    -- Each element of dept_table is a record
Begin
    SELECT * INTO dept_table(1) FROM departments
    WHERE department_id = 10;
    DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id || ||
                         dept_table(1).department_name || ||
                         dept_table(1).manager_id);
END;
/

```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

INDEX BY 레코드 테이블 옵션

앞에서 설명한 바와 같이, 스칼라 데이터 유형의 테이블로 선언된 연관 배열은 데이터베이스 테이블의 한 열에 대한 세부 정보만 저장할 수 있습니다. 그러나 query에서 검색된 모든 열을 저장해야 하는 경우가 종종 있습니다. INDEX BY 레코드 테이블 옵션을 사용하면 데이터베이스 테이블의 모든 필드에 대한 정보를 하나의 배열 정의에 저장할 수 있습니다.

레코드 테이블 생성 및 참조

슬라이드의 연관 배열 예제와 같이 다음을 수행할 수 있습니다.

- %ROWTYPE 속성을 사용하여 데이터베이스 테이블의 한 행을 나타내는 레코드를 선언합니다.
- 배열의 각 요소는 레코드이므로 dept_table 배열 내의 필드를 참조합니다.

%ROWTYPE 속성과 조합 데이터 유형 PL/SQL 레코드 간의 차이점은 다음과 같습니다.

- PL/SQL 레코드 유형은 유저가 정의할 수 있지만, %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.
- 유저가 정의한 레코드는 정적이지만, %ROWTYPE 레코드는 동적이며 테이블 구조를 기반으로 합니다. 테이블 구조가 변경되면 레코드 구조도 해당 변경 사항을 적용합니다.

INDEX BY 레코드 테이블 옵션: 예제 2

```

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 2

슬라이드의 예제는 사원 ID가 100 ~ 104 범위에 있는 사원의 세부 정보를 임시로 저장하도록 INDEX BY 레코드 테이블 옵션을 사용하여 연관 배열을 선언합니다. 배열의 변수 이름은 emp_table_type입니다.

루프를 사용하여 EMPLOYEES 테이블에서 사원 정보가 검색되고 배열에 저장됩니다. 또 다른 루프는 배열에서 성을 출력하는 데 사용됩니다. 예제에서 첫번째 메소드와 마지막 메소드의 사용을 확인하십시오.

참고: 슬라이드에서는 INDEX BY 레코드 테이블 메소드를 사용하는 연관 배열 작업 방법 중 하나를 보여 줍니다. 그러나 커서를 사용하여 동일한 작업을 더 효율적으로 수행할 수 있습니다. 커서는 "명시적 커서 사용" 단원에서 설명합니다.

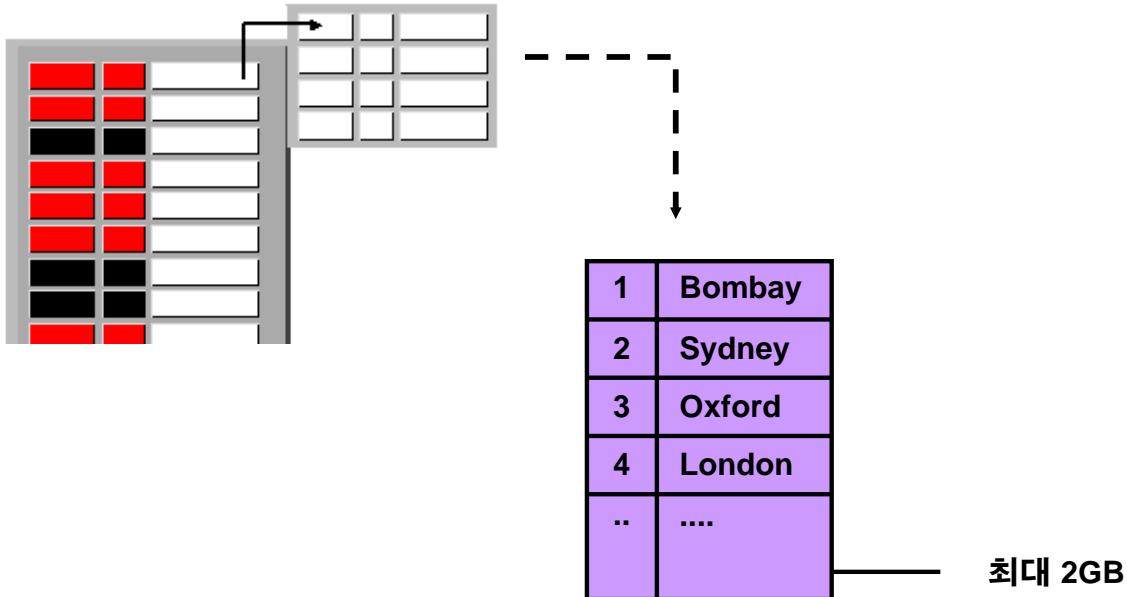
코드 예제의 결과는 다음과 같습니다.

```

Results Script Output Explain
anonymous block completed
King
Kochhar
De Haan
Hunold
Ernst

```

중첩 테이블



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

중첩 테이블

중첩 테이블의 기능은 연관 배열과 비슷하지만 중첩 테이블 구현에 차이가 있습니다.

- 중첩 테이블은 스키마 레벨 테이블에 적합한 데이터 유형이지만 연관 배열은 그렇지 않습니다. 따라서 연관 배열과 달리 중첩 테이블은 데이터베이스에 저장할 수 있습니다.
- 중첩 테이블의 크기는 동적으로 증가할 수 있지만 최대 크기는 2GB입니다.
- 연관 배열과 달리 "키"는 음수 값일 수 없습니다. 첫번째 열을 키로 참조하지만 중첩 테이블에는 키가 없으며 숫자 열이 있습니다.
- 중첩 테이블의 어디서든 요소를 삭제하고 순차적이 아닌 "키"가 있는 희소 테이블을 남겨둘 수 있습니다. 중첩 테이블의 행은 특정 순서로 되어 있지 않습니다.
- 중첩 테이블에서 값을 검색하면 1부터 시작하는 연속적인 하위 스크립트가 행에 지정됩니다.

구문

```

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

중첩 테이블 (계속)

예제:

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

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

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

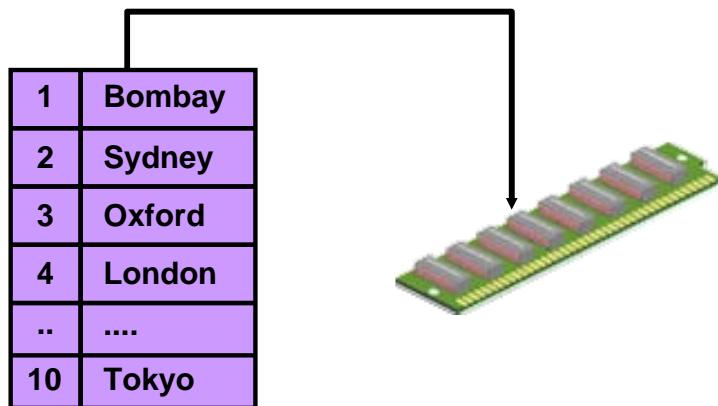
전체 코드 예제와 출력은 다음과 같습니다.

```
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',
        'London');
    FOR i in 1.. offices.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 연관 배열과 비슷합니다.

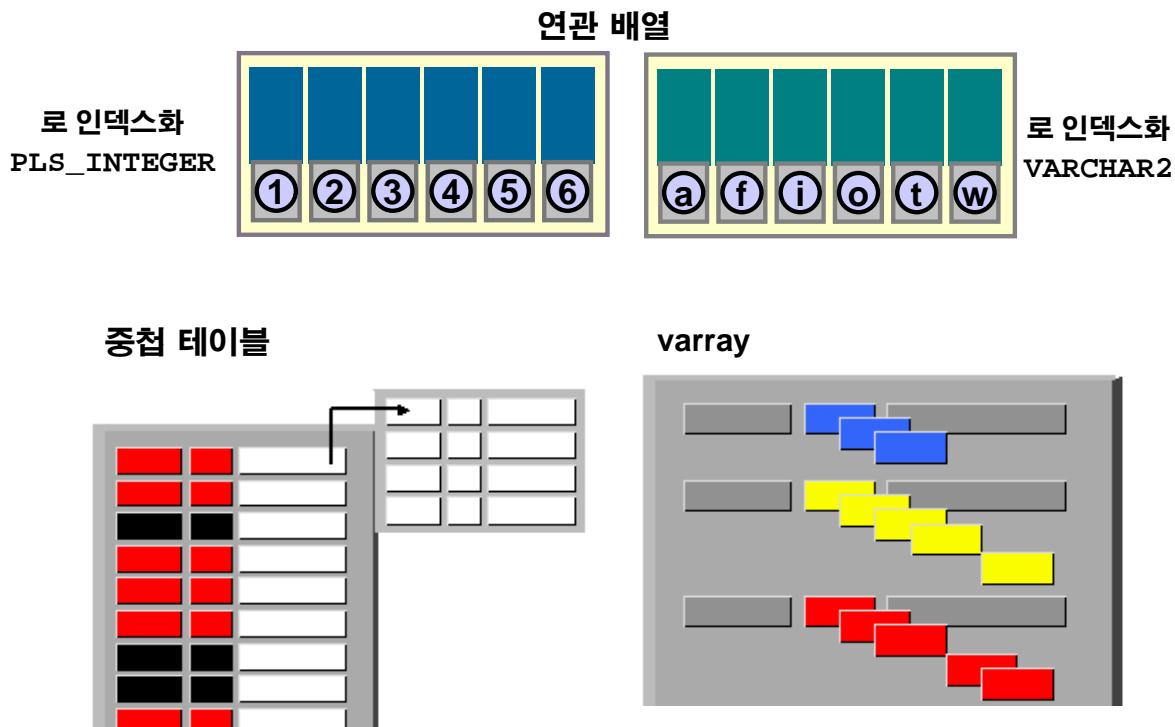
- VARRAY는 스키마 레벨 테이블에 적합합니다.
- VARRAY 유형의 항목을 VARRAYs라고 합니다.
- VARRAYs에는 고정된 상한값이 있습니다. 선언 시 상한값을 지정해야 합니다.
이 점은 C 언어의 배열과 비슷합니다. VARRAY의 최대 크기는 중첩 테이블에서와 같이 2GB입니다.
- 중첩 테이블과 VARRAY가 뚜렷이 구별되는 점은 물리적 저장 모드입니다. VARRAY의 크기가 4KB 보다 크지 않으면 VARRAY의 요소는 테이블 데이터와 함께 인라인으로 저장됩니다. 이 점은 항상 순서 없이 저장되는 중첩 테이블과 대조됩니다.
- 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 하면 "Subscript outside of limit"라는 오류 메시지가 표시됩니다.

컬렉션 유형 요약



Copyright © 2009, Oracle. All rights reserved.

ORACLE

컬렉션 유형 요약

연관 배열

연관 배열은 키-값 쌍 집합입니다. 이 집합에서 각 키는 고유하며 배열에서 해당하는 값을 찾는데 사용됩니다. 키는 정수 또는 문자 기반일 수 있습니다. 배열 값은 스칼라 데이터 유형(단일 값) 또는 레코드 데이터 유형(다중 값)일 수 있습니다.

연관 배열은 임시 데이터를 저장하기 위한 것이므로 `INSERT` 및 `SELECT INTO`와 같은 SQL 문에서는 사용할 수 없습니다.

중첩 테이블

중첩 테이블은 값 집합을 보유합니다. 달리 말하면 테이블 내에 테이블이 있는 것입니다. 중첩 테이블은 범위가 제한되지 않아 테이블 크기가 동적으로 증가할 수 있습니다. 중첩 테이블은 PL/SQL과 데이터베이스 모두에서 사용할 수 있습니다. PL/SQL에서 중첩 테이블은 크기가 동적으로 증가할 수 있는 1차원 배열과 같습니다.

varray

가변 크기의 배열, 즉 `varray`도 런타임에 요소 수를 변경할 수 있지만 고정 개수의 요소를 보유하는 동종 요소로 된 컬렉션입니다. `varray`는 일련 번호를 하위 스크립트로 사용합니다. 상용하는 SQL 유형을 정의하면 `varray`를 데이터베이스 테이블에 저장할 수 있습니다.

퀴즈

다음 중 %ROWTYPE 속성을 사용할 수 있는 경우를 고르십시오.

1. 기본 데이터베이스 테이블의 구조에 대해 확실히 모르는 경우
2. 테이블에서 전체 행을 검색하려는 경우
3. 이전에 선언한 변수 또는 데이터베이스 열에 따라 변수를 선언하려는 경우

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2

%ROWTYPE 속성 사용 시 이점

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 검색되는 각 열에 대해 일일이 변수를 선언해야 합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 조합 데이터 유형의 PL/SQL 변수 정의 및 참조
 - PL/SQL 레코드
 - 연관 배열
 - INDEX BY 테이블
 - INDEX BY 레코드 테이블
- %ROWTYPE 속성을 사용하여 PL/SQL 레코드 정의
- 세 가지 PL/SQL 컬렉션 유형 비교 및 대조:
 - 연관 배열
 - 중첩 테이블
 - VARRAY

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

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

PL/SQL 레코드와 마찬가지로 PL/SQL 컬렉션도 또 다른 조합 데이터 유형입니다. PL/SQL 컬렉션에는 다음이 포함됩니다.

- 연관 배열(INDEX BY 테이블이라고도 함): 연관 배열은 TABLE 유형의 객체이며 약간의 차이를 제외하고 데이터베이스 테이블과 비슷합니다. INDEX BY 테이블이라고도 하는 연관 배열은 Primary Key를 사용하여 배열 방식으로 행에 액세스할 수 있습니다. 연관 배열은 크기 제약이 없습니다.
- 중첩 테이블: INDEX BY 테이블과 달리 중첩 테이블은 음수 값을 포함할 수 없습니다. 또한 키가 순차적이어야 합니다.
- 가변 크기 배열(VARRAY): VARRAY는 크기 제약이 있다는 점을 제외하고 연관 배열과 비슷합니다.

연습 6: 개요

이 연습에서는 다음 내용을 다룹니다.

- 연관 배열 선언
- 연관 배열을 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 연관 배열과 PL/SQL 레코드를 정의, 생성 및 사용합니다.



명시적 커서 사용

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.

목표

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 방법과 간단한 커서 및 파라미터가 포함된 커서를 선언하고 제어하는 방법도 배웁니다.

다루는 내용

- 명시적 커서란?
- 명시적 커서 사용
- 파라미터가 포함된 커서 사용
- 행 잠금 및 현재 행 참조

ORACLE®

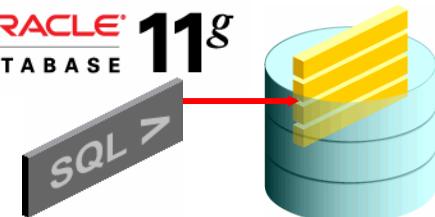
Copyright © 2009, Oracle. All rights reserved.

커서

Oracle 서버에서 실행되는 모든 SQL 문에는 연관된 개별 커서가 있습니다.

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

ORACLE®
DATABASE
11g



암시적 커서



명시적 커서

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

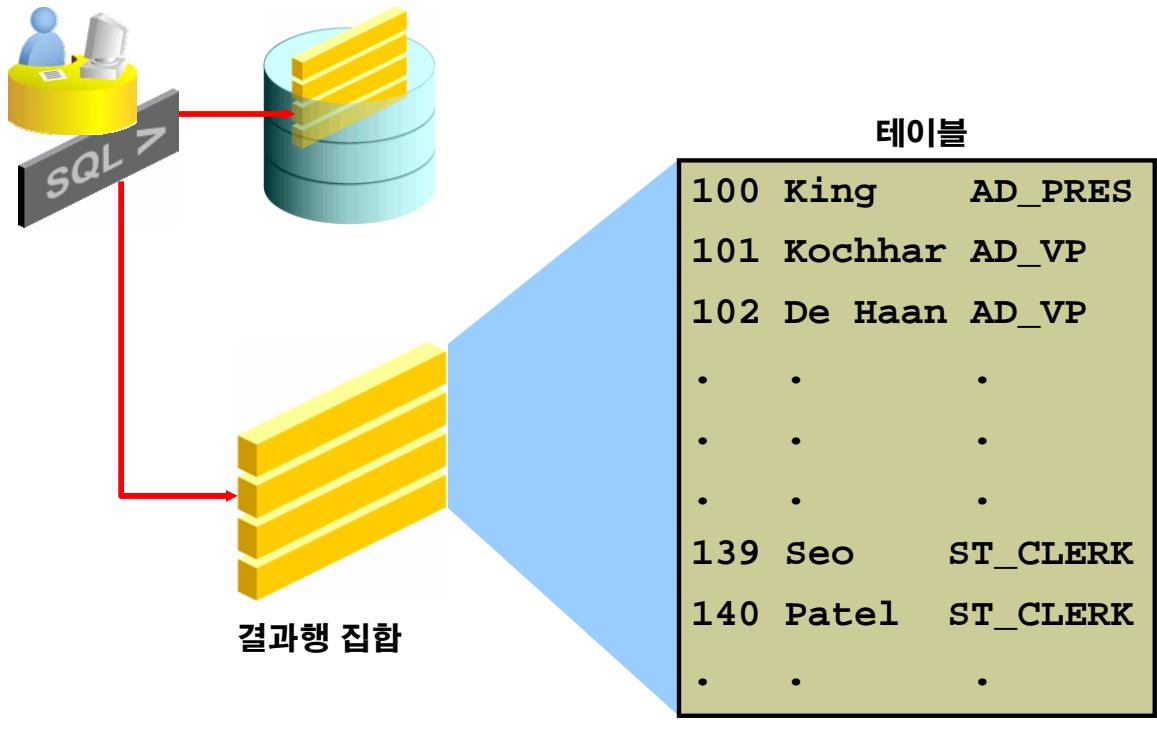
커서

Oracle 서버는 전용(private) SQL 영역이라는 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 다중 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



Copyright © 2009, Oracle. All rights reserved.

명시적 커서 작업

여러 행을 반환하는 SELECT 문이 있을 때 PL/SQL에서 명시적 커서를 선언합니다.

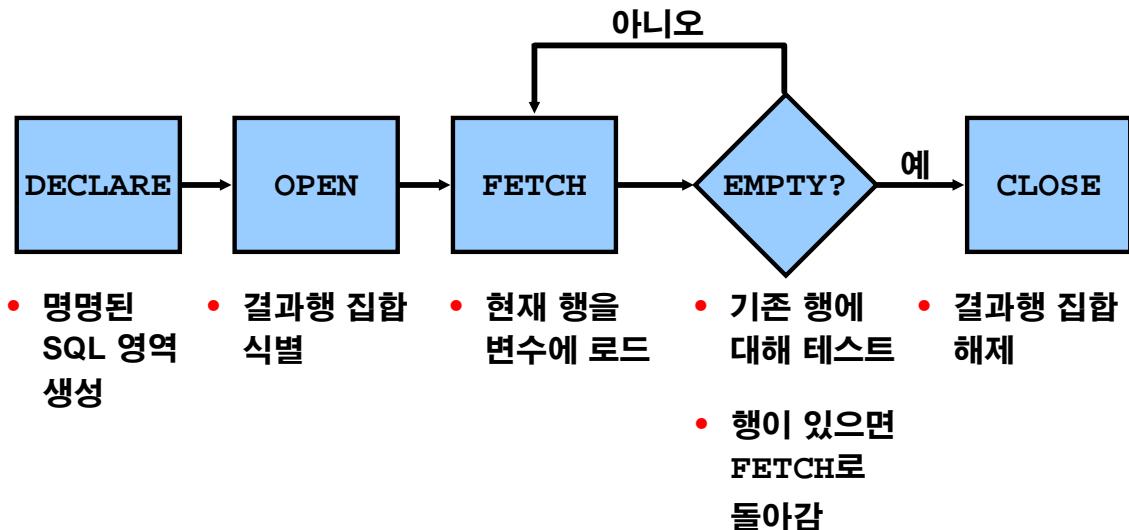
SELECT 문에서 반환된 각 행을 처리할 수 있습니다.

여러 행 query에서 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 한 행씩 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.

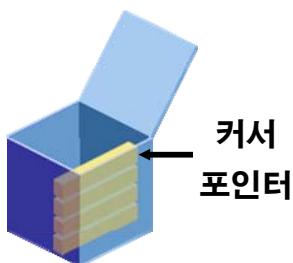
명시적 커서 제어

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 query 를 실행하고 참조된 모든 변수를 바인드합니다.
Query에 의해 식별된 행을 결과행 집합이라고 하며 현재 패치(fetch)할 수 있습니다.
3. 커서에서 데이터를 패치(fetch)합니다. 위 슬라이드에 표시된 흐름도에서 각 패치(fetch) 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 결과행 집합을 설정할 수 있습니다.

명시적 커서 제어

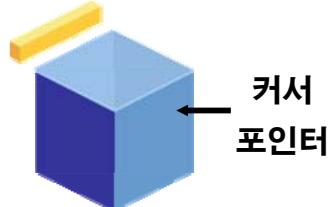
1 커서를 엽니다.



2 행을 패치 (fetch)합니다.



3 커서를 닫습니다.



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 검색하고 행이 더 이상 없거나 지정된 조건에 부합할 때까지 커서를 다음 행으로 이동합니다.
3. CLOSE 문은 커서를 해제합니다.

다루는 내용

- 명시적 커서란?
- **명시적 커서 사용**
- 파라미터가 포함된 커서 사용
- 행 잠금 및 현재 행 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

커서 선언

구문:

```
CURSOR cursor_name IS  
    select_statement;
```

예제:

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서 선언

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

cursor_name PL/SQL 식별자
select_statement 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 처리하려는 경우 query 에 ORDER BY 절을 사용합니다.
- 커서는 조인, subquery 등을 포함하는 유효한 SELECT 문일 수 있습니다.

커서 선언 (계속)

c_emp_cursor 커서는 department_id가 30인 부서에서 근무하는 사원의 employee_id 및 last_name 열을 검색하도록 선언됩니다.

c_dept_cursor 커서는 location_id가 1700 인 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/SQL에서 여러 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 c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
    ...
BEGIN
    OPEN c_emp_cursor;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서 열기

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

OPEN은 다음과 같은 작업을 수행하는 실행문입니다.

1. 쿼리 텍스트 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.

참고: 커서가 열려 있을 때 query 가 행을 반환하지 않으면 PL/SQL에서 예외가 발생하지 않습니다. <cursor_name>%ROWCOUNT 속성을 사용하여 명시적 커서와 함께 반환된 행 수를 확인할 수 있습니다.

커서에서 데이터 패치(fetch)

```

DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
        v_empno employees.employee_id%TYPE;
        v_lname employees.last_name%TYPE;
BEGIN
    OPEN c_emp_cursor;
    FETCH c_emp_cursor INTO v_empno, v_lname;
    DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
END;
/

```

```

anonymous block completed
114 Raphaely

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서에서 데이터 패치(fetch)

FETCH 문은 한 번에 하나씩 커서에서 행을 검색합니다. 각 패치(fetch) 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 패치(fetch)된 값을 보유하도록 선언됩니다. FETCH 문을 검사하십시오.

커서의 값이 변수로 성공적으로 패치(fetch)되었습니다. 그러나 부서 30에는 사원이 여섯 명 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.

SELECT 문에 있는 열과 동일한 수의 변수를 FETCH 문의 INTO 절에 포함시킬 수 있습니다. 데이터 유형이 호환되는지 확인해야 합니다. 위치에 따라 열에 각 변수를 대응시킵니다. 또는 커서에 대한 레코드를 정의하고 FETCH INTO 절에서 해당 레코드를 참조할 수도 있습니다. 마지막으로 커서에 행이 있는지 테스트합니다. 패치(fetch) 후 획득한 값이 없으면 결과행 집합에 처리할 행이 남아 있지 않은 것이며 이것은 오류로 기록되지 않습니다.

커서에서 데이터 패치(fetch)

```

DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id =30;
    v_empno employees.employee_id%TYPE;
    v_lname employees.last_name%TYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_empno, v_lname;
        EXIT WHEN c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
    END LOOP;
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서에서 데이터 패치(fetch) (계속)

간단한 LOOP가 모든 행을 패치(fetch)하는 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 c_emp_cursor INTO empno, lname;
  EXIT WHEN c_emp_cursor%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE( v_empno || ' ' || v_lname);
END LOOP;
CLOSE c_emp_cursor;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서 닫기

CLOSE 문은 커서를 비활성화하고 컨텍스트 영역을 해제하며 결과행 집합의 "정의를 해제" 합니다. 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 시도하면 INVALID_CURSOR 예외가 발생합니다.

참고: 커서를 닫지 않고 PL/SQL 블록을 종료할 수 있지만, 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 패치(fetch)하여 결과행 집합의 행을 처리합니다.

```

DECLARE
    CURSOR c_emp_cursor IS
        SELECT employee_id, last_name FROM employees
        WHERE department_id = 30;
        v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_emp_record;
        EXIT WHEN c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                               || ' ' || v_emp_record.last_name);
    END LOOP;
    CLOSE c_emp_cursor;
END;

```

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.

```

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

```

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.

<i>record_name</i>	암시적으로 선언된 레코드 이름
<i>cursor_name</i>	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

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

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

커서 FOR 루프 (계속)

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

`emp_record`은 암시적으로 선언된 레코드입니다. 위 슬라이드에 표시된 것과 같이 이러한 암시적 레코드를 사용하여 꽤치(fetch) 데이터에 액세스할 수 있습니다. `INTO` 절을 사용하여 꽤치된 데이터를 보유하도록 선언된 변수가 없습니다. 이 코드에는 커서를 열고 닫는 `OPEN` 및 `CLOSE` 문이 없습니다.

명시적 커서 속성

명시적 커서 속성을 사용하여 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 패치(fetch)가 행을 반환하면 TRUE로 평가됩니다. %NOTFOUND 속성을 보완합니다.
%ROWCOUNT	Number	지금까지 반환된 총 행 수로 평가됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

명시적 커서 속성

암시적 커서와 마찬가지로, 커서에 대한 상태 정보를 얻기 위한 네 개의 속성이 있습니다.
이러한 속성을 커서 변수 이름에 추가하면 커서 조작문 실행에 대한 유용한 정보가 반환됩니다.
참고: SQL 문에서는 커서 속성을 직접 참조할 수 없습니다.

%ISOPEN 속성

- 커서가 열려 있을 때만 행을 패치(fetch)할 수 있습니다.
- 패치(fetch)를 수행하기 전에 %ISOPEN 커서 속성을 사용하여 커서가 열려 있는지 테스트합니다.

예제:

```
IF NOT c_emp_cursor%ISOPEN THEN
    OPEN c_emp_cursor;
END IF;
LOOP
    FETCH c_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 패치(fetch)합니다. 커서 속성을 사용하여 루프 종료 시점을 판별할 수 있습니다.
- %ROWCOUNT 커서 속성을 사용하여 다음을 수행할 수 있습니다.
 - 정확한 수의 행 처리
 - 루프에서 행을 패치(fetch)하고 루프 종료 시점 확인

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

%ROWCOUNT 및 %NOTFOUND: 예제

```
DECLARE
  CURSOR c_emp_cursor IS SELECT employee_id,
    last_name FROM employees;
  v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
      c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
      || ' ' || v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END ; /
```

anonymous block completed
174 Abel
166 Ande
130 Atkinson
105 Austin
204 Baer
116 Baida
167 Banda
172 Bates
192 Bell
151 Bernstein

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.

```
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;
/
```

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

subquery를 사용하는 커서 FOR 루프

이 PL/SQL 블록에는 선언 섹션이 없습니다. subquery 를 사용하는 커서 FOR 루프와 일반 커서 FOR 루프의 차이점은 커서 선언에 있습니다. subquery 를 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 subquery 를 사용하는 커서 FOR 루프를 나타냈습니다.

참고: 커서 FOR 루프에 subquery 를 사용할 경우 커서에 명시적 이름을 부여할 수 없으므로 명시적 커서 속성을 참조할 수 없습니다.

다루는 내용

- 명시적 커서란?
- 명시적 커서 사용
- **파라미터가 포함된 커서 사용**
- 행 잠금 및 현재 행 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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

파라미터를 커서로 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 query 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

```

DECLARE
  CURSOR c_emp_cursor (deptno NUMBER) IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id = deptno;
    ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...

```

```

anonymous block completed
200 Whalen
201 Hartstein
202 Fay

```

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 c_emp_cursor(deptno NUMBER) IS SELECT ...

```

다음 명령문은 커서를 열고 각기 다른 결과행 집합을 반환합니다.

```

OPEN c_emp_cursor(10);
OPEN c_emp_cursor(20);

```

커서 FOR 루프에 사용된 커서로 파라미터를 전달할 수 있습니다.

```

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

```

다루는 내용

- 명시적 커서란?
- 명시적 커서 사용
- 파라미터가 포함된 커서 사용
- 행 잡금 및 현재 행 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 잠금니다.

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.

column_reference query 가 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 query 하려는 경우 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 잠금니다.

FOR UPDATE 절 (계속)

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

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

예제:

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

```

Oracle 서버는 `SELECT FOR UPDATE` 작업에 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 포함합니다.
- WHERE CURRENT OF 절을 사용하여 명시적 커서에서 현재 행을 참조합니다.

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

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 row 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.)

퀴즈

암시적 커서는 PL/SQL에서 모든 DML 및 PL/SQL SELECT 문에 대해 암시적으로 선언됩니다. Oracle 서버는 암시적으로 커서를 열어 명시적으로 선언된 커서와 연관되지 않은 각 SQL 문을 처리합니다.

- 1. 맞습니다.**
- 2. 틀립니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 여러 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 query 를 포함한 모든 SQL 데이터 조작문에 대해 커서를 암시적으로 선언합니다. 여러 행을 반환하는 query 의 경우 행을 개별적으로 처리하도록 커서를 명시적으로 선언해야 합니다.

모든 명시적 커서 및 커서 변수에는 %FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT의 네 가지 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 여러 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 잠금니다. 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 선언하고 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.

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 커스터마이즈

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

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 런타임에 예상치 못한 코드 오류가 발생할 수 있습니다. 이 단원에서는 PL/SQL 블록에서 이러한 오류를 처리하는 방법에 대해 설명합니다.

다루는 내용

- PL/SQL 예외 이해
- 예외 트랩

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

예외란?

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

```
Error starting at line 3 in command:
DECLARE
    v_lname VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_lname FROM employees WHERE
        first_name='John';
    DBMS_OUTPUT.PUT_LINE ('John''s last name is : ' ||v_lname);
END;
Error report:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 4
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.

예외란?

위 슬라이드에 표시된 예제를 살펴보십시오. 코드에 구문 오류가 없으므로 익명 블록을 성공적으로 실행할 수 있어야 합니다. 블록의 SELECT 문은 John 의 성을 읽어 들입니다.

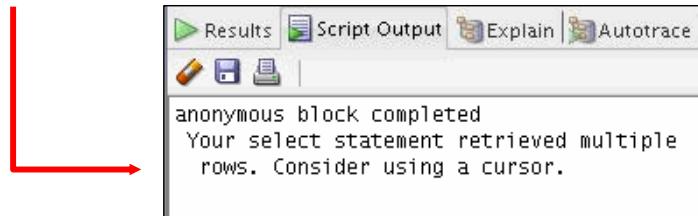
그러나 코드를 실행하면 다음과 같은 오류 보고서를 볼 수 있습니다.

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

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

예외 처리: 예제

```
DECLARE
    v_lname VARCHAR2(15);
BEGIN
    SELECT last_name INTO v_lname
    FROM employees
    WHERE first_name='John';
    DBMS_OUTPUT.PUT_LINE ('John''s last name is : ' || v_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.

예외 처리: 예제

앞에서 이미 선언 섹션 (DECLARE 키워드로 시작) 과 실행 섹션 (BEGIN 키워드로 시작하여 END 키워드로 종료) 으로 PL/SQL 블록을 작성하는 방법을 배웠습니다.

오류 처리를 위해 예외 섹션이라는 다른 선택적 섹션을 포함시킵니다.

- 이 섹션은 EXCEPTION 키워드로 시작합니다.
- 예외 섹션이 존재하는 경우 이 섹션이 PL/SQL 블록의 마지막 섹션입니다.

예제

슬라이드의 예제에서는 발생한 예외를 처리하도록 이전 슬라이드의 코드가 다시 작성됩니다. 코드의 출력도 슬라이드에 표시됩니다.

코드의 EXCEPTION 섹션을 추가하면 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/SQL 예외 이해

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

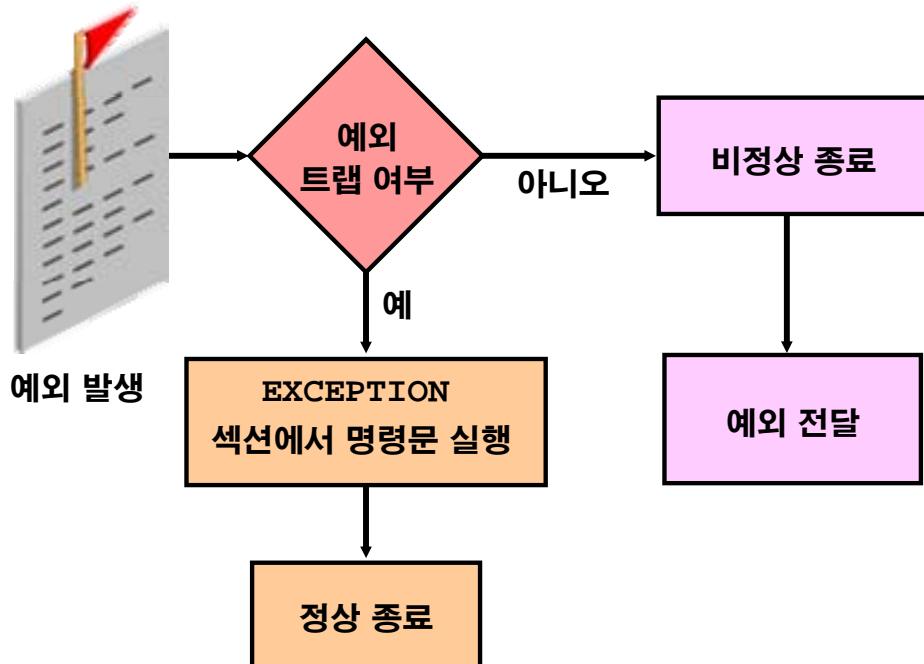
PL/SQL 예외 이해

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

예외를 발생시키기 위한 두 가지 방법

- Oracle 오류가 발생하면 연관된 예외가 자동으로 발생됩니다. 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. 발생되는 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 포함하십시오. 블록의 실행 섹션에서 예외가 발생하면 처리 과정이 블록 예외 섹션의 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
-
- 사용자 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.	이러한 예외는 선언 셋션 내에서 선언해야 합니다. Oracle 서버는 암시적으로 오류를 발생시키며 예외 처리기에서 오류를 처리할 수 있습니다.
유저 정의 오류	개발자가 비정상적인 것으로 결정한 조건입니다.	선언 셋션에서 선언하고 명시적으로 발생시켜야 합니다.

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

다루는 내용

- PL/SQL 예외 이해
- 예외 트랩

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 구문

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 포함할 수 있는 처리기 수에는 제한이 없습니다. 그러나 단일 예외에 대해 다중 처리기를 가질 수 없습니다.

예외 트랩 구문에는 다음 요소가 포함됩니다.

<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 구문 (계속)

WHEN OTHERS 예외 처리기

앞에서 설명한 대로 예외 처리 섹션은 지정된 예외만 트랩합니다.

지정되지 않은 예외를 트랩하려면 OTHERS 예외 처리기를 사용합니다. 이 옵션은 아직 처리되지 않은 모든 예외를 트랩합니다. 따라서 OTHERS 처리기를 사용할 때는 마지막으로 정의해야 합니다.

예를 들면 다음과 같습니다.

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

예제

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 OTHERS는 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/SQL은 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



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.

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

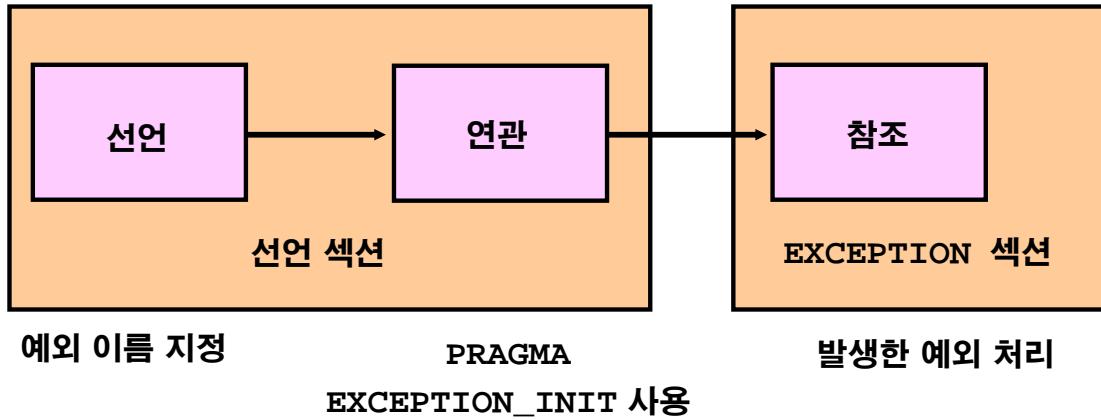
미리 정의된 예외

예외 이름	Oracle 서버 오류 번호	설명
ACCESS_INTO_NULL	ORA-06530	초기화되지 않은 객체의 속성에 값을 할당하려고 했습니다.
CASE_NOT_FOUND	ORA-06592	CASE 문의 WHEN 절에서 어떤 것도 선택할 수 없으며 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 로그온하고 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 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 적합한 ROWID를 나타내지 않기 때문에 문자열을 Universal ROWID로 변환하는 데 실패했습니다.
TIMEOUT_ON_RESOURCE	ORA-00051	Oracle 서버가 자원을 대기하는 동안 시간 초과가 발생했습니다.
TOO_MANY_ROWS	ORA-01422	단일 행 SELECT가 다중 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 서버에서 PL/SQL 예외로 정의되지 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 선언하여 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.

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

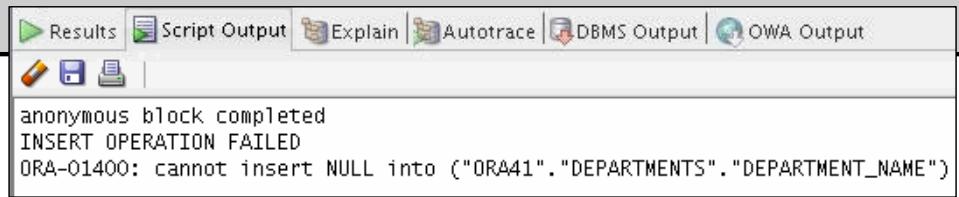
미리 정의되지 않은 오류 트랩: 예제

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

```

DECLARE
    e_insert_excep EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);
BEGIN
    INSERT INTO departments
    (department_id, department_name) VALUES (280, NULL);
EXCEPTION
    WHEN e_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 트랩: 예제

이 예제는 미리 정의되지 않은 오류 트랩과 관련된 세 단계를 설명합니다.

1. 다음 구문을 사용하여 선언 섹션에 예외 이름을 선언합니다.

```
exception EXCEPTION;
```

이 구문에서 `exception`은 예외의 이름입니다.

2. PRAGMA EXCEPTION_INIT 함수를 사용하여 선언된 예외를 표준 Oracle 서버 오류 번호와 연관시킵니다. 다음 구문을 사용하십시오.

```
PRAGMA EXCEPTION_INIT(exception, error_number);
```

이 구문에서 `exception`은 앞서 선언한 예외이고 `error_number`는 표준 Oracle 서버 오류 번호입니다.

3. 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

예제

슬라이드에 표시된 예제는 `departments` 테이블의 `department_name` 열에 NULL 값을 삽입하려고 합니다. 그러나 `department_name`이 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 `SQLERRM`에 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

```

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.

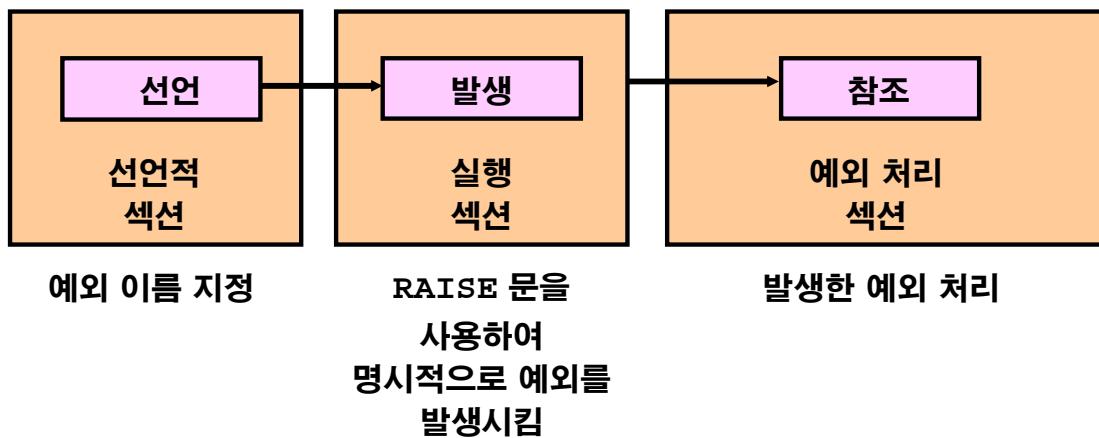
SQL 문에서 직접 SQLCODE 또는 SQLERRM을 사용할 수 없습니다. 대신 다음 예제와 같이 해당 값을 로컬 변수에 할당한 다음 SQL 문에서 이 변수를 사용해야 합니다.

```

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.

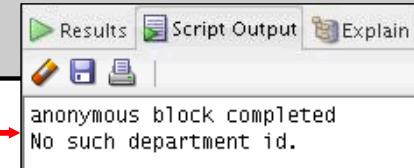
- PL/SQL 블록의 선언 섹션에서 선언해야 합니다.
- RAISE 문을 사용하여 명시적으로 발생시켜야 합니다.
- EXCEPTION 섹션에서 처리해야 합니다.

유저 정의 예외 트랩

```

DECLARE
    v_deptno NUMBER := 500;
    v_name VARCHAR2(20) := 'Testing';
    e_invalid_department EXCEPTION;
BEGIN
    UPDATE departments
    SET department_name = v_name
    WHERE department_id = v_deptno;
    IF SQL%NOTFOUND THEN
        RAISE e_invalid_department;
    END IF;
    COMMIT;
EXCEPTION
    WHEN e_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.

- 선언 섹션 내에 유저 정의 예외에 대한 이름을 선언합니다.

구문:

```
exception EXCEPTION;
```

이 구문에서 *exception*은 예외의 이름입니다.

- RAISE 문을 사용하여 실행 섹션 내에서 해당 예외를 명시적으로 발생시킵니다.

구문:

```
RAISE exception;
```

이 구문에서 *exception*은 앞서 선언한 예외입니다.

- 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

예제

슬라이드에 표시된 블록은 부서의 *department_name*을 갱신합니다. 유저가 부서 번호와 새 이름을 제공합니다. 제공된 부서 번호가 존재하지 않으면 *departments* 테이블에서 행이 갱신되지 않습니다. 예외가 발생하고 유효하지 않은 부서 번호가 입력되었음을 알리는 메시지가 유저에게 출력됩니다.

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

서브 블록의 예외 전달

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

```

DECLARE
    ...
    e_no_rows exception;
    e_integrity exception;
    PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
    FOR c_record IN emp_cursor LOOP
        BEGIN
            SELECT ... ;
            UPDATE ... ;
            IF SQL%NOTFOUND THEN
                RAISE e_no_rows;
            END IF;
        END;
    END LOOP;
EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_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 환경에서 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.

error_number 예외에 대한 유저 지정 번호로, -20,000 ~ -20,999 범위의 값입니다.

message 유저가 지정한 예외 메시지로, 최대 길이 2,048바이트의 문자열입니다.

TRUE | FALSE 선택적 부울 파라미터입니다.(TRUE인 경우 오류가 이전 오류의 스택에 추가되고, FALSE인 경우 이전 오류가 모두 해당 오류로 바뀝니다.)

RAISE_APPLICATION_ERROR 프로시저

- 다음과 같은 서로 다른 두 위치에 사용됩니다.
 - 실행 섹션
 - 예외 섹션
- 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 셠택션:

```
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 셋션:

```
...
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 셋션과 예외 셋션 모두에서 RAISE_APPLICATION_ERROR 프로시저를 사용할 수 있음을 보여줍니다.

다음은 RAISE_APPLICATION_ERROR 프로시저를 사용하는 또 다른 예제입니다.

```
DECLARE
    e_name EXCEPTION;
BEGIN
    ...
    DELETE FROM employees
        WHERE last_name = 'Higgins';
    IF SQL%NOTFOUND THEN RAISE e_name;
    END IF;
EXCEPTION
    WHEN e_name THEN
        RAISE_APPLICATION_ERROR (-20999, 'This is not a valid
last name'); ...
END;
/
```

퀴즈

PL/SQL 블록의 예외 처리 섹션에 해당 처리기를 포함하여 모든 오류를 트랩할 수 있습니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1

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 포함할 수 있는 처리기 수에는 제한이 없습니다. 그러나 단일 예외에 대해 다중 처리기를 가질 수 없습니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- PL/SQL 예외 정의
- PL/SQL 블록에 EXCEPTION 섹션을 추가하여 런타임에 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 런타임에 발생하는 경고 또는 오류 조건을 예외라고 합니다. 미리 정의된 예외는 Oracle 서버에서 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_FUNDS라는 예외를 정의할 수 있습니다.

오류가 발생하면 예외가 발생됩니다. 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.

- 유저 정의 예외 생성 및 호출
- 명명된 Oracle 서버 예외 처리



Copyright © 2009, Oracle. All rights reserved.

연습 8: 개요

이 연습에서는 미리 정의된 예외와 표준 Oracle 서버 예외에 대한 예외 처리기를 생성합니다.

내장 프로시저 및 함수 소개

9

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.

목표

이미 앞에서 익명 블록에 대해 배웠습니다. 이 단원에서는 서브 프로그램이라고도 하는 명명된 블록을 소개합니다. 프로시저와 함수가 PL/SQL 서브 프로그램입니다. 이 단원에서는 익명 블록과 서브 프로그램을 구분하는 방법도 배웁니다.

다루는 내용

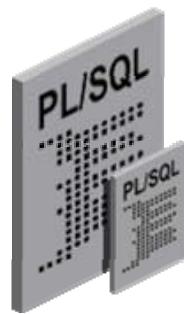
- **프로시저 및 함수 소개**
- **프로시저 미리 보기**
- **함수 미리 보기**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 다른 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.

프로시저와 함수는 서브 프로그램이라고도 하는 명명된 PL/SQL 블록입니다. 이러한 서브 프로그램은 컴파일한 다음 데이터베이스에 저장할 수 있습니다. 서브 프로그램의 블록 구조는 익명 블록의 구조와 유사합니다. 서브 프로그램은 스키마 레벨에서뿐만 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 프로그램의 차이

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.	함수일 경우 값을 반환해야 합니다.
파라미터를 사용할 수 없습니다.	파라미터를 사용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

익명 블록과 서브 프로그램의 차이

슬라이드의 테이블은 익명 블록과 서브 프로그램의 차이를 보여주며 서브 프로그램의 일반적인 이점을 크게 강조하고 있습니다.

익명 블록은 영구적인 데이터베이스 객체가 아니며 실행할 때마다 컴파일됩니다. 또한 재사용할 수 있도록 데이터베이스에 저장되지 않습니다. 재사용하려면 재컴파일과 실행이 일어나도록 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.

다루는 내용

- **프로시저 및 함수 소개**
- **프로시저 미리 보기**
- **함수 미리 보기**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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>	생성 할 프로시저 이름입니다.
<i>argument</i>	프로시저 파라미터에 제공된 이름입니다. 모든 인수는 모드 및 데이터 유형과 연관됩니다. 쉼표로 구분된 인수를 원하는 만큼 가질 수 있습니다.
<i>mode</i>	인수 모드 IN(기본값) OUT IN OUT
<i>datatype</i>	연관된 파라미터의 데이터 유형입니다. 파라미터의 데이터 유형은 명시적인 크기를 가질 수 없고 대신 %TYPE을 사용합니다.
<i>Procedure_body</i>	코드를 구성하는 PL/SQL 블록입니다.

프로시저 선언에서 인수 리스트는 선택적입니다. 프로시저에 대한 자세한 내용은 *Oracle Database 11g: Develop PL/SQL Program Units* 과정에서 다룹니다.

프로시저 작성

```

...
CREATE TABLE dept AS SELECT * FROM departments;
CREATE PROCEDURE add_dept IS
  v_dept_id dept.department_id%TYPE;
  v_dept_name dept.department_name%TYPE;
BEGIN
  v_dept_id:=280;
  v_dept_name:='ST-Curriculum';
  INSERT INTO dept(department_id,department_name)
  VALUES(v_dept_id,v_dept_name);
  DBMS_OUTPUT.PUT_LINE(' Inserted '|| SQL%ROWCOUNT
  ||' row ');
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 작성

코드 예제에서 add_dept 프로시저는 부서 ID가 280이고 부서 이름이 ST-Curriculum 인 새 부서를 삽입합니다.

또한 예제에는 다음도 표시됩니다.

- 프로시저의 선언 섹션은 프로시저 선언 바로 다음에 시작되며 DECLARE 키워드로 시작하지 않습니다.
- 프로시저는 dept_id와 dept_name이라는 두 변수를 선언합니다.
- 프로시저는 암시적 커서 속성이나 SQL%ROWCOUNT SQL 속성을 사용하여 해당 행이 성공적으로 삽입되었는지 확인합니다. 이 경우 값 1이 반환되어야 합니다.

참고: 예제에 대한 추가 참고 사항은 다음 페이지를 참조하십시오.

프로시저: 예제

참고

- 객체를 작성하면 user_objects 테이블에 항목이 만들어집니다. 슬라이드의 코드가 성공적으로 실행되면 다음 명령을 실행하여 user_objects 테이블에서 새 객체를 확인할 수 있습니다.

```
SELECT object_name,object_type FROM user_objects;
```

OBJECT_NAME	OBJECT_TYPE
41 COPY_EMP	TABLE
42 DEPT	TABLE
43 GREET	PROCEDURE
44 ADD_DEPT	PROCEDURE
45 MY_SEQ	SEQUENCE

- 프로시저의 소스는 user_source 테이블에 저장됩니다. 다음 명령을 실행하여 프로시저의 소스를 확인할 수 있습니다.

```
SELECT * FROM user_source WHERE name='ADD_DEPT' ;
```

NAME	TYPE	LINE	TEXT
1 ADD_DEPT	PROCEDURE	1	PROCEDURE add_dept IS
2 ADD_DEPT	PROCEDURE	2	v_dept_id dept.department_id%TYPE;
3 ADD_DEPT	PROCEDURE	3	v_dept_name dept.department_name%TYPE;
4 ADD_DEPT	PROCEDURE	4	BEGIN
5 ADD_DEPT	PROCEDURE	5	v_dept_id:=280;
6 ADD_DEPT	PROCEDURE	6	v_dept_name:='ST-Curriculum';
7 ADD_DEPT	PROCEDURE	7	INSERT INTO dept(department_id,department_name)
8 ADD_DEPT	PROCEDURE	8	VALUES(v_dept_id,v_dept_name);
9 ADD_DEPT	PROCEDURE	9	DBMS_OUTPUT.PUT_LINE(' Inserted' SQL%ROWCOUNT ' row');
10 ADD_DEPT	PROCEDURE	10	END;

프로시저 호출

```
...
BEGIN
  add_dept;
END;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```

DEPARTMENT_ID	DEPARTMENT_NAME
280	ST-Curriculum
1 rows selected	

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 호출

이 슬라이드는 익명 블록에서 프로시저를 호출하는 방법을 보여줍니다. 익명 블록의 실행 섹션에 프로시저에 대한 호출을 포함해야 합니다. 마찬가지로 Forms 응용 프로그램이나 Java 응용 프로그램 등의 응용 프로그램에서 프로시저를 호출할 수 있습니다. 코드의 SELECT 문은 해당 행이 성공적으로 삽입되었는지 확인합니다.

CALL <procedure_name> SQL 문을 사용하여 프로시저를 호출할 수도 있습니다.

다루는 내용

- **프로시저 및 함수 소개**
- **프로시저 미리 보기**
- **함수 미리 보기**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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 함수 이름입니다.
<i>argument</i>	함수 파라미터에 부여된 이름입니다.(모든 인수는 모드 및 데이터 유형과 연관됩니다. 쉼표로 구분된 인수를 원하는 만큼 가질 수 있습니다. 함수를 호출하는 경우 인수를 전달합니다.)
<i>mode</i>	파라미터의 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.

인수 리스트는 함수 선언에서 선택 사항입니다. 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 작성

```
CREATE FUNCTION check_sal RETURN Boolean IS
v_dept_id employees.department_id%TYPE;
v_empno   employees.employee_id%TYPE;
v_sal      employees.salary%TYPE;
v_avg_sal  employees.salary%TYPE;
BEGIN
  v_empno:=205;
  SELECT salary,department_id INTO v_sal,v_dept_id FROM
employees
  WHERE employee_id= v_empno;
  SELECT avg(salary) INTO v_avg_sal FROM employees WHERE
department_id=v_dept_id;
  IF v_sal > v_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.

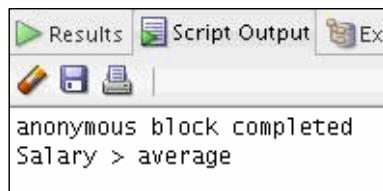
함수: 예제

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 205 인 사원을 검사합니다. 함수는 이 사원 ID만 검사하도록 하드 코딩되어 있습니다. 다른 사원을 검사하려면 함수 자체를 수정해야 합니다. 인수를 받아들이도록 함수를 선언하여 이 문제를 해결할 수 있습니다. 그런 다음 사원 ID를 파라미터로 전달할 수 있습니다.

함수 호출

```
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;
/
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 호출

익명 블록의 실행 세션에 함수에 대한 호출을 포함시킵니다. 함수는 명령문의 일부로 호출됩니다. check_sal 함수는 Boolean 또는 NULL을 반환합니다. 따라서 함수 호출이 IF 블록의 조건식으로 포함됩니다.

참고: 다음 예제와 같이 DESCRIBE 명령을 사용하여 함수의 인수 및 반환 유형을 확인할 수 있습니다.

```
DESCRIBE check_sal;
```

함수에 파라미터 전달

```

DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(p_empno employees.employee_id%TYPE)
RETURN Boolean IS
    v_dept_id employees.department_id%TYPE;
    v_sal      employees.salary%TYPE;
    v_avg_sal  employees.salary%TYPE;
BEGIN
    SELECT salary,department_id INTO v_sal,v_dept_id FROM employees
        WHERE employee_id=p_empno;
    SELECT avg(salary) INTO v_avg_sal FROM employees
        WHERE department_id=v_dept_id;
    IF v_sal > v_avg_sal THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
EXCEPTION
    ...

```

ORACLE

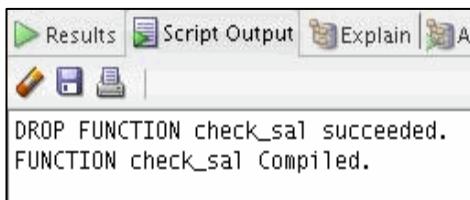
Copyright © 2009, Oracle. All rights reserved.

함수에 파라미터 전달

앞에서 사용한 함수는 사원 ID가 205 인 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 11g: Develop PL/SQL Program Units* 과정에서 다룹니다.

이 슬라이드의 코드 예제 출력은 다음과 같습니다.



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. It displays the following text:

```

Results | Script Output | Explain | A
| P | P |
DROP FUNCTION check_sal succeeded.
FUNCTION check_sal Compiled.

```

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;
/

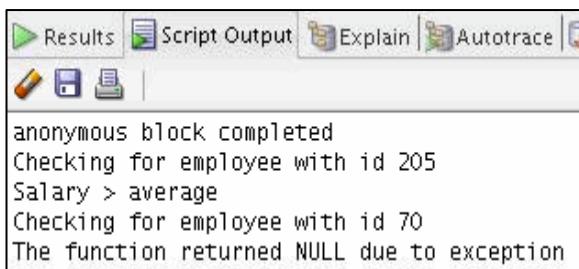
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

파라미터가 포함된 함수 호출

슬라이드의 코드는 파라미터를 전달하여 함수를 두 번 호출합니다. 코드의 출력 결과는 다음과 같습니다.



The screenshot shows the Oracle SQL developer interface with the 'Results' tab selected. The output window displays the following text:

```

anonymous block completed
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception

```

퀴즈

서브 프로그램:

1. 명명된 PL/SQL 블록이며 다른 응용 프로그램에서 호출할 수 있습니다.
2. 한 번만 컴파일됩니다.
3. 데이터베이스에 저장됩니다.
4. 함수인 경우 값을 반환할 필요가 없습니다.
5. 파라미터를 사용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 5

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



Copyright © 2009, Oracle. All rights reserved.

부록 A

연습 및 해답

목차

단원 I 의 연습 및 해답	3
연습 I-1: SQL Developer 리소스에 액세스	3
연습 I-2: 시작하기	4
해답 I-1: SQL Developer 리소스에 액세스	6
해답 I-2: 시작하기	7
단원 1 의 연습 및 해답	15
연습 1: PL/SQL 소개	15
해답 1: PL/SQL 소개	16
단원 2 의 연습 및 해답	17
연습 2: PL/SQL 변수 선언	17
해답 2: PL/SQL 변수 선언	19
단원 3 의 연습 및 해답	22
연습 3: 실행문 작성	22
해답 3: 실행문 작성	24
단원 4 의 연습 및 해답	28
연습 4: Oracle 서버와 상호 작용	28
해답 4: Oracle 서버와 상호 작용	30
단원 5 의 연습 및 해답	33
연습 5: 제어 구조 작성	33
해답 5: 제어 구조 작성	35
단원 6 의 연습 및 해답	38
연습 6: 조합 데이터 유형 작업	38
해답 6: 조합 데이터 유형 작업	41
단원 7 의 연습 및 해답	45
연습 7-1: 명시적 커서 사용	45
연습 7-2: 명시적 커서 사용 – 선택 사항	48
해답 7-1: 명시적 커서 사용	49
해답 7-2: 명시적 커서 사용 – 선택 사항	54
단원 8 의 연습 및 해답	56
연습 8-1: 미리 정의된 예외 처리	56
연습 8-2: 표준 Oracle 서버 예외 처리	58
해답 8-1: 미리 정의된 예외 처리	59
해답 8-2: 표준 Oracle 서버 예외 처리	61
단원 9 의 연습 및 해답	62
연습 9: 내장 프로시저 생성 및 사용	62
해답 9: 내장 프로시저 생성 및 사용	64

단원 I의 연습 및 해답

이 연습에서는 SQL Developer에 대한 정보 리소스를 식별하고, SQL Developer를 사용하여 SQL 문을 실행하고, 클래스 스키마에서 데이터를 검사합니다. 특히 다음을 수행합니다.

- SQL Developer 시작
- 새 데이터베이스 연결 생성
- 스키마 테이블 탐색
- SQL Developer 환경 설정 구성

참고: 모든 연습 문제는 SQL Developer를 개발 환경으로 사용합니다.

SQL Developer를 사용하는 것이 권장되지만 이 과정에서 사용 가능한 SQL*Plus 또는 JDeveloper 환경을 사용할 수도 있습니다.

연습 I-1: SQL Developer 리소스에 액세스

이 연습에서는 SQL Developer 홈 페이지로 이동하여 도구에 대한 유용한 정보를 탐색합니다.

SQL Developer 홈 페이지에 액세스합니다.

- a) 다음 위치의 온라인 SQL Developer 홈 페이지에 액세스합니다.

http://www.oracle.com/technology/products/database/sql_developer/index.html

- b) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.

- 1) <http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>의 온라인 SQL Developer 자습서에 액세스합니다. 다음 섹션 및 관련 데모를 검토합니다.
 - a) What to Do First
 - b) Working with Database Objects
 - c) Accessing Data

연습 I-2: 시작하기

- 1) SQL Developer를 시작합니다.
- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다(힌트: Save Password 체크 박스 선택).
 - a) Connection Name: MyConnection
 - b) Username: ora41
 - c) Password: ora41
 - d) Hostname: localhost
 - e) Port: 1521
 - f) SID: orcl
- 3) 새 연결을 테스트합니다. 상태가 Success이면 새로운 이 연결을 사용하여 데이터베이스에 연결합니다.
 - a) Database Connection window에서 Test 버튼을 누릅니다.
참고: window의 왼쪽 하단 모서리에 연결 상태가 나타납니다.
 - b) 상태가 Success이면 Connect 버튼을 누릅니다.
- 4) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.
 - a) 옆에 있는 더하기 기호를 눌러 MyConnection 연결을 확장합니다.
 - b) Tables 아이콘 옆에 있는 더하기 기호를 눌러 해당 아이콘을 확장합니다.
 - c) EMPLOYEES 테이블의 구조를 표시합니다.
- 5) EMPLOYEES 탭을 사용하여 EMPLOYEES 테이블의 데이터를 봅니다.
- 6) SQL Worksheet를 사용하여 연봉이 \$10,000보다 많은 모든 사원의 성과 급여를 선택합니다. Execute Statement (F9) 및 Run Script (F5) 아이콘을 모두 사용하여 SELECT 문을 실행합니다. 해당 탭에서 SELECT 문을 실행하여 두 방법의 결과를 검토합니다.
참고: 몇 분 동안 데이터를 살펴보거나, 이 과정에서 사용할 HR 스키마의 모든 테이블에 대한 설명과 데이터를 제공하는 부록 B를 참조하십시오.
- 7) SQL Developer 메뉴에서 Tools > Preferences를 선택합니다. Preferences window가 나타납니다.
- 8) Database > Worksheet Parameters를 선택합니다. "Select default path to look for scripts" 텍스트 상자에서 Browse 버튼을 사용하여 /home/oracle/labs/plsf 폴더를 선택합니다. 이 과정에서 사용하는 코드 예제 스크립트, 연습 스크립트 및 연습 해답 스크립트가 이 폴더에 있습니다. 그런 다음 Preferences window에서 OK를 눌러 Worksheet Parameter 설정을 저장합니다.

연습 I-2: 시작하기 (계속)

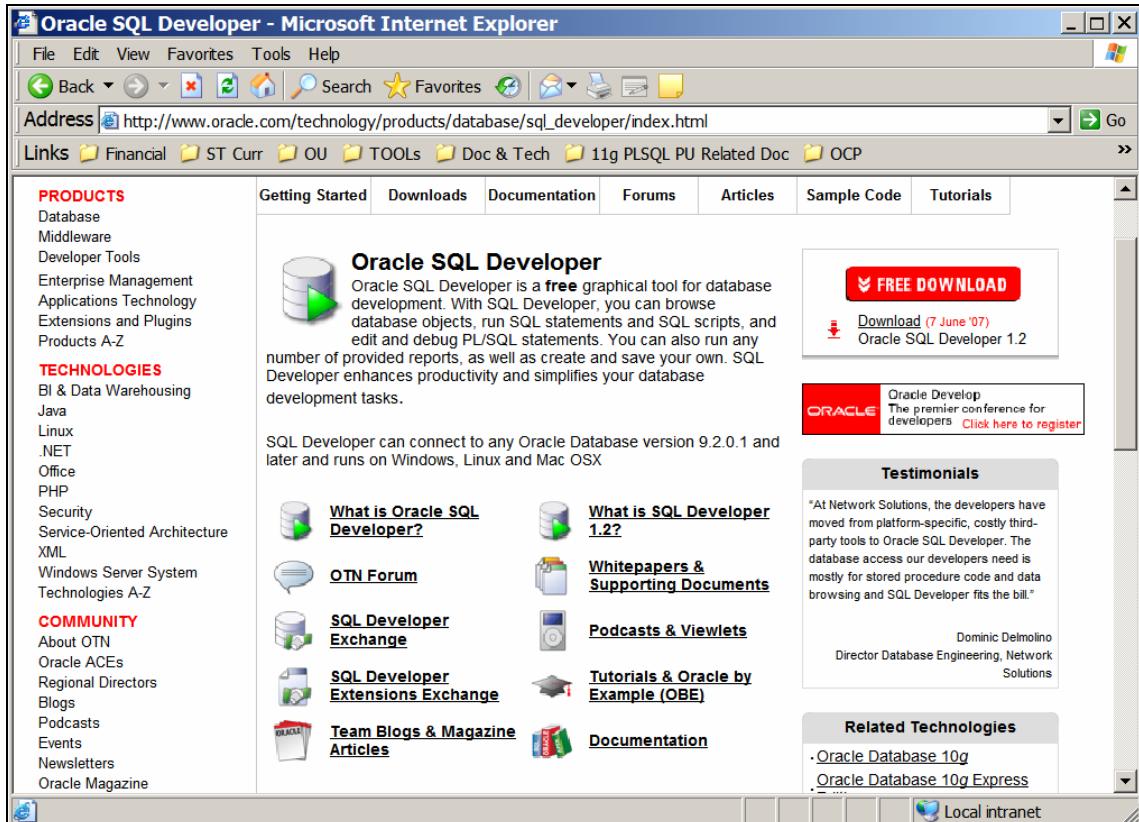
- 9) /home/oracle/labs/plsf 폴더의 구조를 익힙니다.
- a) File > Open을 선택합니다. Open window에서 .../plsf 폴더가 시작 위치로 자동 선택됩니다. 이 폴더에는 다음과 같은 세 개의 하위 폴더가 있습니다.
- /code_ex 폴더에는 과정 교재에 나오는 코드 예제가 있습니다. 각 .sql 스크립트는 단원의 특정 페이지와 연관됩니다.
 - /labs 폴더에는 특정 단원 연습에서 사용되는 코드가 있습니다. 해당 연습에서 필요한 스크립트를 실행합니다.
 - /soln 폴더에는 각 연습에 대한 해답이 있습니다. 각 .sql 스크립트는 연관된 practice_exercise 참조로 번호가 매겨집니다.
- b) Files 탭을 사용하여 폴더를 탐색하고 스크립트 파일을 열 수도 있습니다.
- c) Open window 및 Files 탭을 사용하여 폴더를 탐색하고 코드 실행 없이 스크립트 파일을 엽니다.
- d) SQL Worksheet를 닫습니다.

해답 I-1: SQL Developer 리소스에 액세스

1) SQL Developer 홈 페이지에 액세스합니다.

- a) http://www.oracle.com/technology/products/database/sql_developer/index.html의
온라인 SQL Developer 홈 페이지에 액세스합니다.

다음과 같이 SQL Developer 홈 페이지가 표시됩니다.



- b) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.
- 2) <http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>의 온라인 SQL Developer 자습서에 액세스합니다. 다음 섹션 및 관련 데모를 검토합니다.
- What to Do First
 - Working with Database Objects
 - Accessing Data

해답 I-2: 시작하기

- 1) SQL Developer를 시작합니다.

바탕 화면에서 SQL Developer 아이콘을 누릅니다.



- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다(힌트: Save Password 체크 박스 선택).

- a) Connection Name: MyConnection
- b) Username: ora41
- c) Password: ora41
- d) Hostname: localhost
- e) Port: 1521
- f) SID: orcl

Connections 탭 페이지의 Connections 노드를 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 New Database Connection을 선택합니다. 결과: New>Select Database Connection window가 나타납니다.

위에 나와 있는 정보를 사용하여 새 데이터베이스 연결을 생성합니다.
Save Password 체크 박스도 선택합니다. 다음 그림을 참조하십시오.

 A screenshot of the 'New Database Connection' dialog box. The 'Connection Name' field contains 'MyConnection'. The 'Username' field contains 'ora41'. The 'Password' field contains '*****'. The 'Save Password' checkbox is checked. Under the 'Access' tab, the 'Role' dropdown is set to 'default' and the 'Connection Type' dropdown is set to 'Basic'. Under the 'Basic' tab, the 'Hostname' is 'localhost', 'Port' is '1521', and 'SID' is selected with the value 'orcl'. Other options like 'OS Authentication', 'Kerberos Authentication', and 'Proxy Connection' are available but not selected.

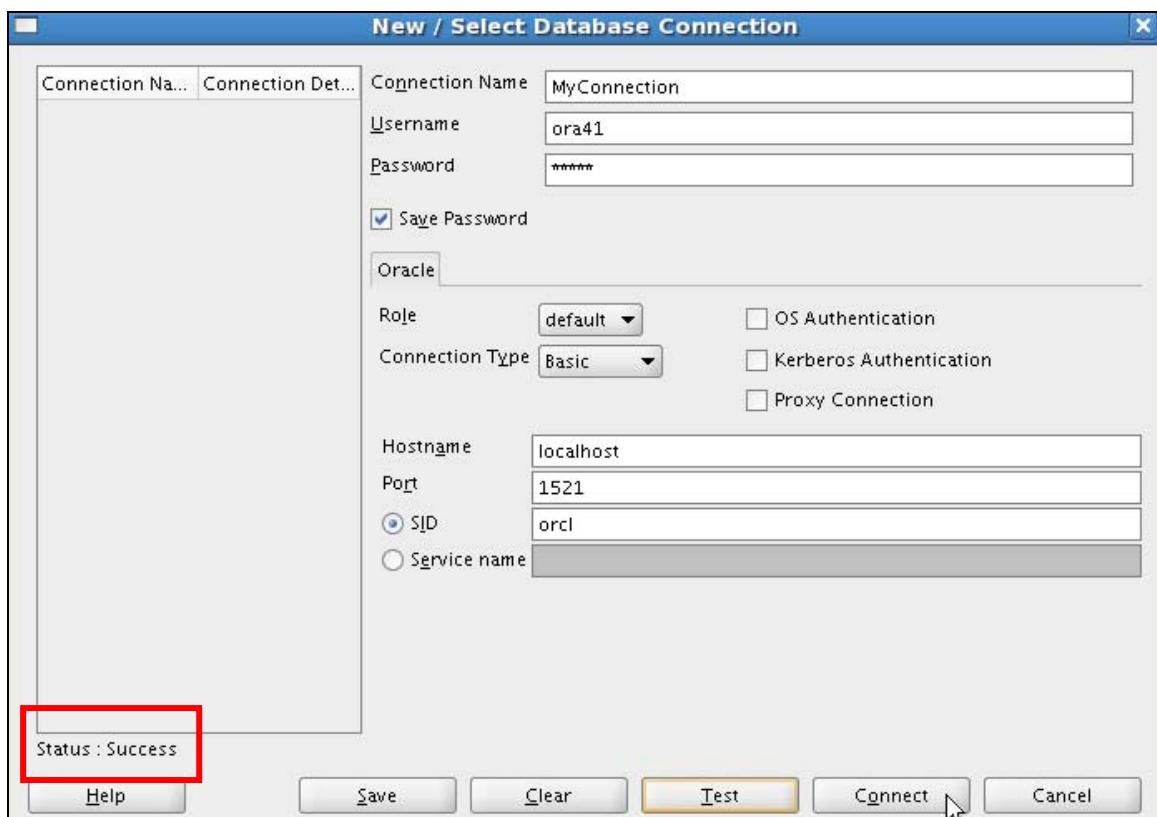
해답 I-2: 시작하기 (계속)

3) 새 연결을 테스트합니다. 상태가 Success이면 새로운 이 연결을 사용하여 데이터베이스에 연결합니다.

a) Database Connection window에서 Test 버튼을 누릅니다.

참고: window의 왼쪽 하단 모서리에 연결 상태가 나타납니다.

b) 상태가 Success이면 Connect 버튼을 누릅니다.



참고: 기존 연결의 속성을 표시하려면 Connections 탭에서 연결 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Properties를 선택합니다.

해답 I-2: 시작하기 (계속)

- 4) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.
- 옆에 있는 더하기 기호를 눌러 MyConnection 연결을 확장합니다.
 - 옆에 있는 더하기 기호를 눌러 Tables를 확장합니다.
 - EMPLOYEES 테이블의 구조를 표시합니다.

옆에 있는 더하기 기호를 눌러 EMPLOYEES 테이블로 드릴 다운합니다.

EMPLOYEES 테이블을 누릅니다.

결과: Columns 탭에 EMPLOYEES 테이블의 열이 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Developer interface. On the left, the Connections pane displays a connection named 'MyConnection' which is expanded to show 'Tables'. Under 'Tables', the 'EMPLOYEES' table is selected and expanded, showing its columns: 'EMPLOYEE_ID', 'FIRST_NAME', 'LAST_NAME', 'EMAIL', 'PHONE_NUMBER', 'HIRE_DATE', 'JOB_ID', 'SALARY', 'COMMISSION_PCT', 'MANAGER_ID', and 'DEPARTMENT_ID'. On the right, the main workspace shows the 'EMPLOYEES' table structure in the 'Columns' tab. The table has 11 columns, each with a column name, data type, nullability, default value, column ID, and primary key status. The primary key is 'EMPLOYEE_ID'.

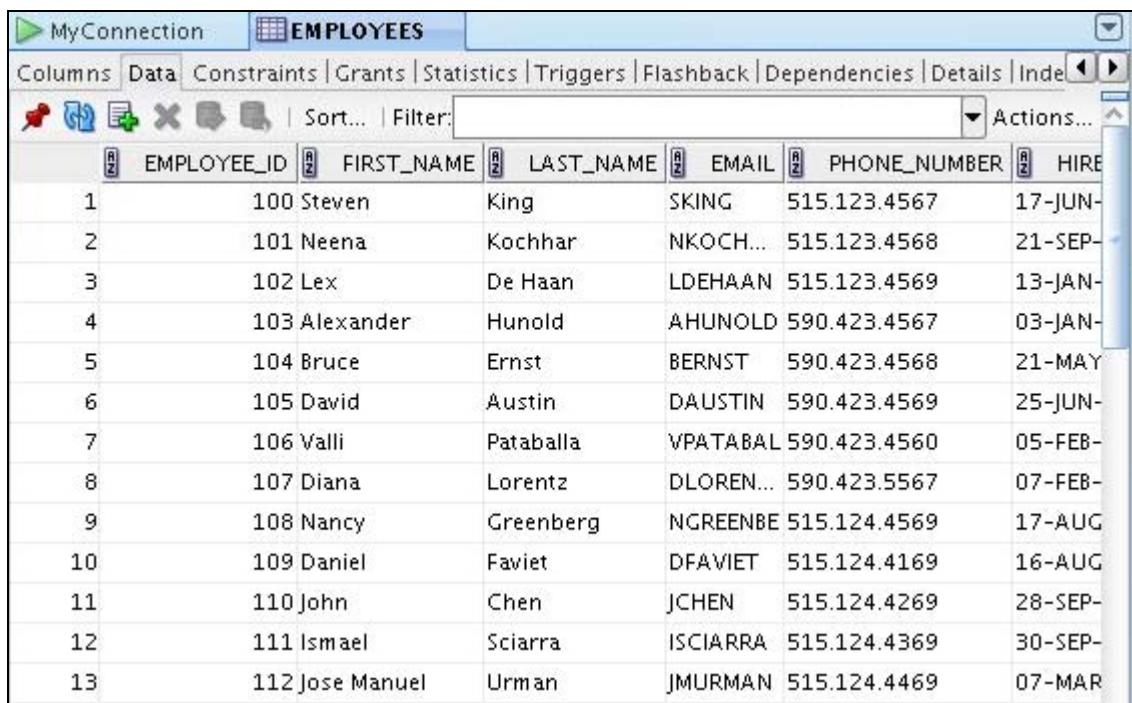
Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1 F
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null) F
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	(null) L
EMAIL	VARCHAR2(25 BYTE)	No	(null)	4	(null) E
PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5	(null) F
HIRE_DATE	DATE	No	(null)	6	(null) C
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	(null) C
SALARY	NUMBER(8,2)	Yes	(null)	8	(null) N
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	(null) C
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	(null) N
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	(null) C

해답 I-2: 시작하기 (계속)

- 5) EMPLOYEES 탭을 사용하여 EMPLOYEES 테이블의 데이터를 봅니다.

사원의 데이터를 표시 하려면 Data 탭을 누릅니다.

결과: EMPLOYEES 테이블 데이터가 다음과 같이 표시됩니다.



The screenshot shows the Oracle Database SQL Workshop interface with the 'EMPLOYEES' table selected. The table contains 13 rows of employee data with columns: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, and HIRE_DATE. The data is as follows:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
1	100	Steven	King	SKING	515.123.4567	17-JUN-03
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-03
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-03
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-03
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-03
6	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-03
7	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-03
8	107	Diana	Lorentz	DLORENZ	590.423.5567	07-FEB-03
9	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-03
10	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-03
11	110	John	Chen	JCHEN	515.124.4269	28-SEP-03
12	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-03
13	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-04

- 6) SQL Worksheet를 사용하여 연봉이 \$10,000보다 많은 모든 사원의 성과 급여를 선택합니다. Execute Statement (F9) 및 Run Script (F5) 아이콘을 모두 사용하여 SELECT 문을 실행합니다. 해당 탭에서 SELECT 문을 실행하여 두 방법의 결과를 검토합니다.

참고: 몇 분 동안 데이터를 살펴보거나, 이 과정에서 사용할 HR 스키마의 모든 테이블에 대한 설명과 데이터를 제공하는 부록 B를 참조하십시오.

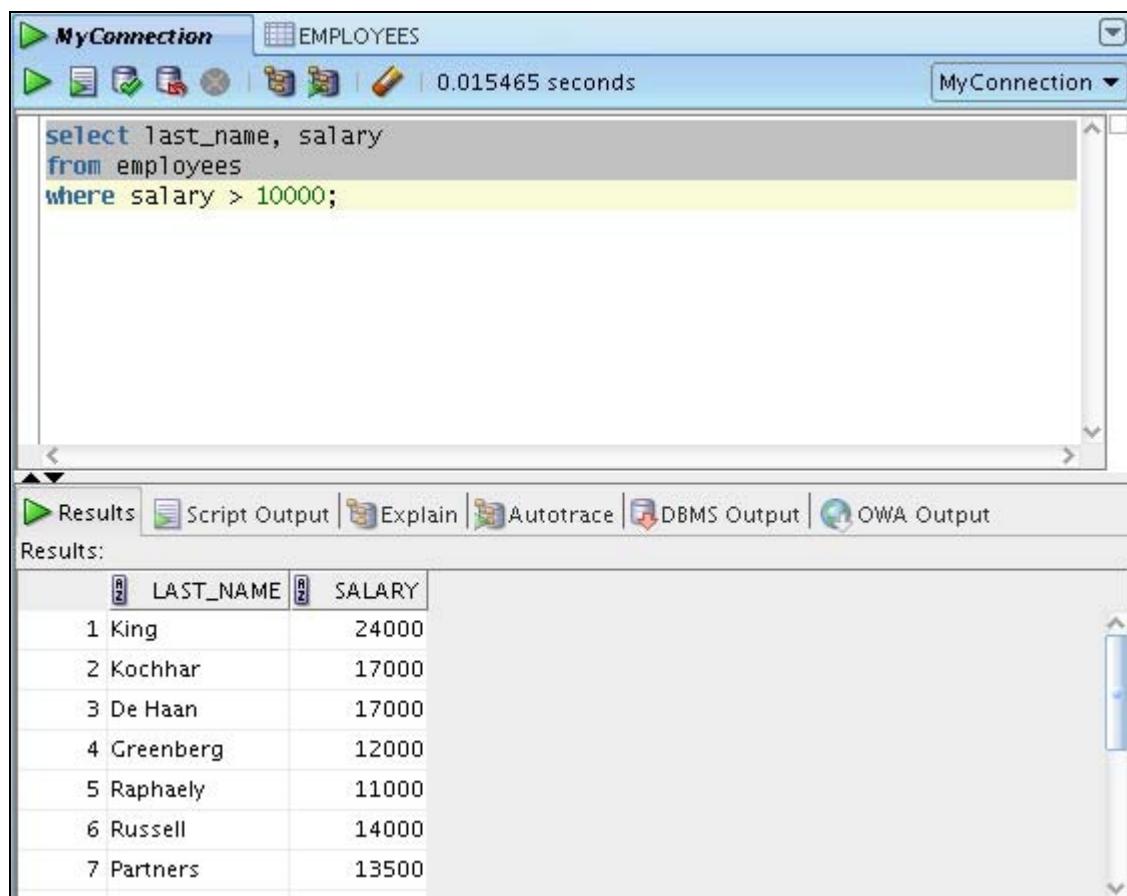
SQL Worksheet를 표시 하려면 MyConnection 탭을 누릅니다.

참고: 이전에 데이터베이스 연결로 드릴 다운 할 때 이 탭이 열렸습니다.

해답 I-2: 시작하기 (계속)

적절한 SELECT 문을 입력합니다. Query를 실행하려면 F9 키를 누르고, Run Script 메소드를 사용하여 query를 실행하려면 F5 키를 누릅니다.

예를 들어, F9 키를 누르면 다음과 비슷한 결과가 나타납니다.



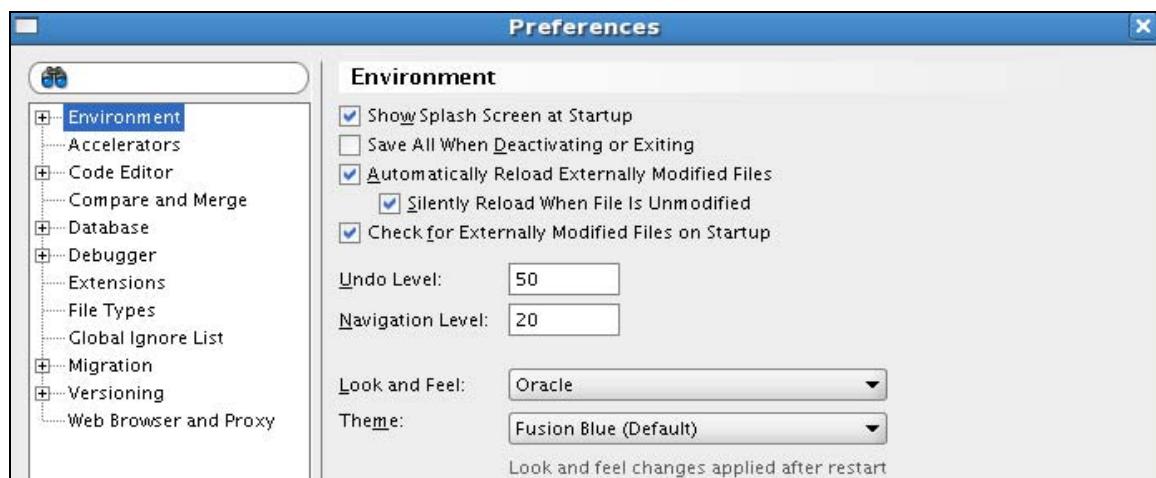
The screenshot shows the SQL Developer interface. At the top, there's a toolbar with various icons and a status bar indicating "MyConnection" and "EMPLOYEES". Below the toolbar is a query editor window containing the following SQL code:

```
select last_name, salary
from employees
where salary > 10000;
```

Below the query editor is a results grid titled "Results". The grid displays the following data:

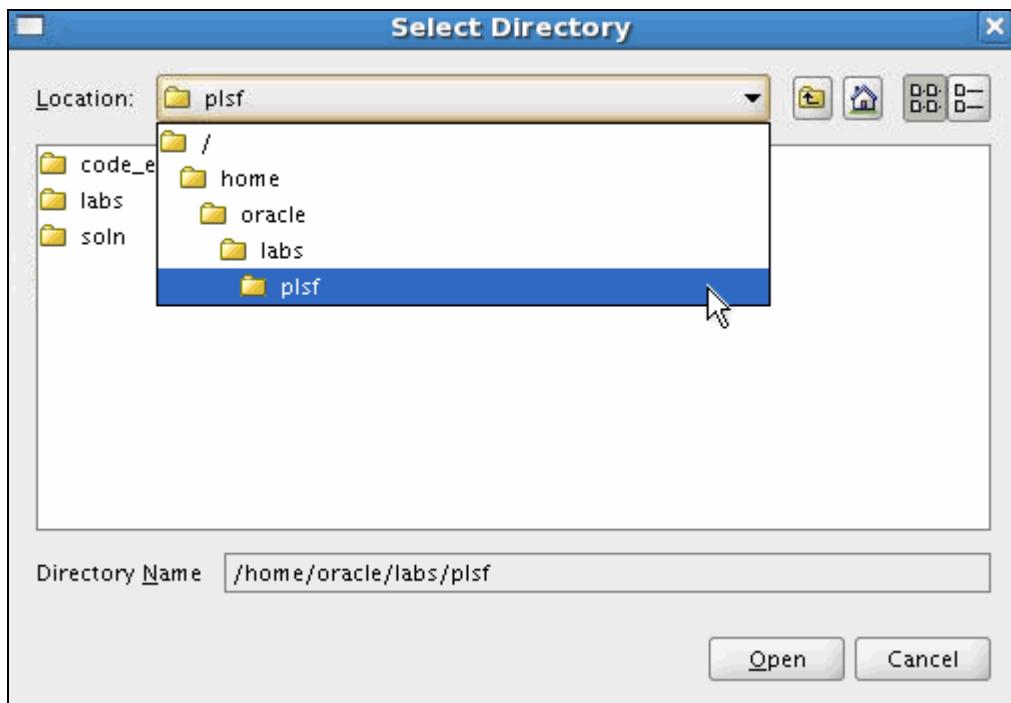
	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Greenberg	12000
5	Raphaely	11000
6	Russell	14000
7	Partners	13500

- 7) SQL Developer 메뉴에서 Tools > Preferences를 선택합니다. Preferences window가 나타납니다.



해답 I-2: 시작하기 (계속)

- 8) Database > Worksheet Parameters를 선택합니다. "Select default path to look for scripts" 텍스트 상자에서 Browse 버튼을 사용하여 /home/oracle/labs/plsf 폴더를 선택합니다.

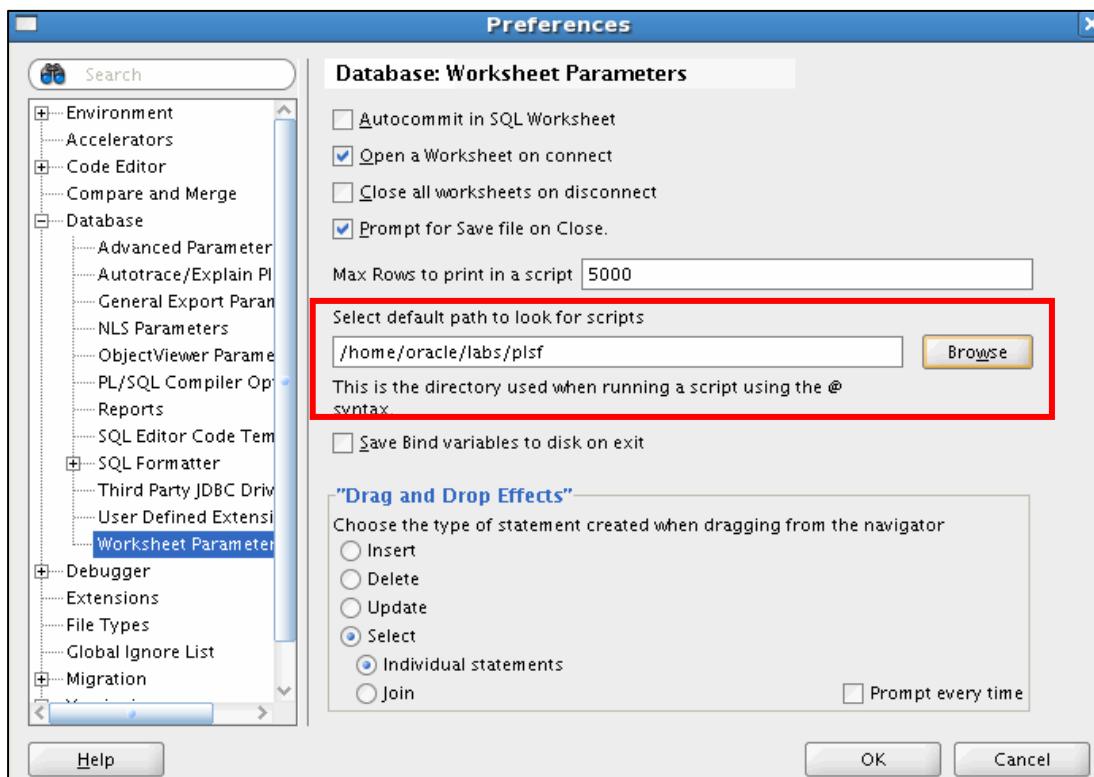


이 과정에서 사용하는 코드 예제 스크립트, 연습 스크립트 및 연습 해답 스크립트가 이 폴더에 있습니다.

Open을 눌러 폴더를 선택합니다.

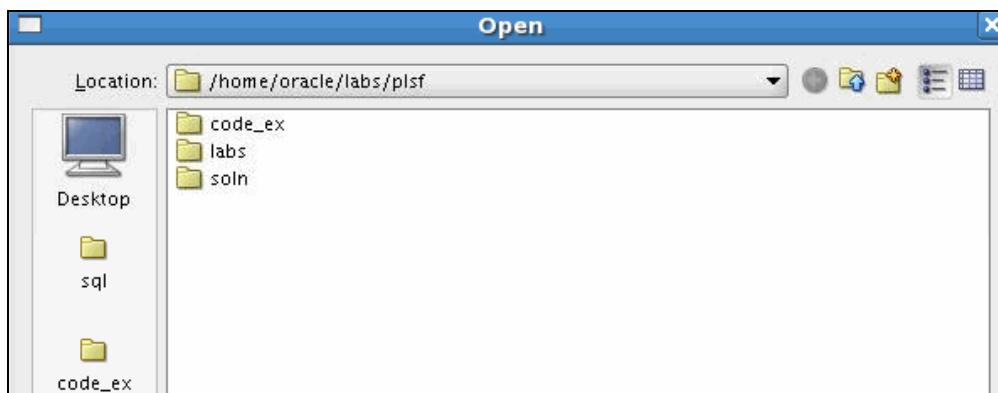
해답 I-2: 시작하기 (계속)

그런 다음 Preferences window에서 OK를 눌러 Worksheet Parameter 설정을 저장합니다.



9) /home/oracle/labs/plsf 폴더의 구조를 익힙니다.

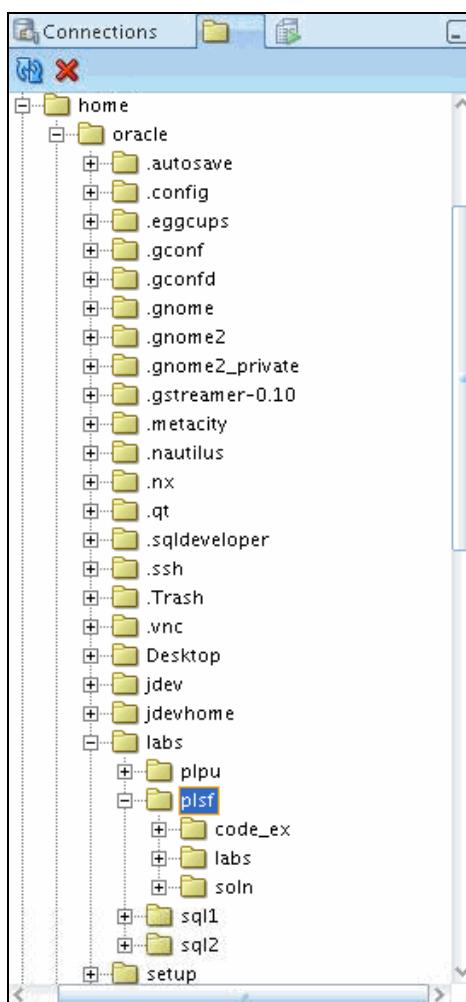
- a) File > Open을 선택합니다. Open window에서 .../plsf 폴더가 시작 위치로 자동 선택됩니다. 이 폴더에는 다음과 같은 세 개의 하위 폴더가 있습니다.



- /code_ex 폴더에는 과정 교재에 나오는 코드 예제가 있습니다. 각 .sql 스크립트는 단원의 특정 페이지와 연관됩니다.
- /labs 폴더에는 특정 단원 연습에서 사용되는 코드가 있습니다. 해당 연습에서 필요한 스크립트를 실행합니다.
- /soln 폴더에는 각 연습에 대한 해답이 있습니다. 각 .sql 스크립트는 연관된 practice_exercise 참조로 번호가 매겨집니다.

해답 I-2: 시작하기 (계속)

- b) Files 탭을 사용하여 폴더를 탐색하고 스크립트 파일을 열 수도 있습니다.



- c) Open window 및 Files 탭을 사용하여 폴더를 탐색하고 코드 실행 없이 스크립트 파일을 엽니다.
- d) SQL Worksheet를 닫습니다.

SQL Worksheet 탭을 닫으려면 다음과 같이 탭에서 X를 누릅니다.



단원 1의 연습 및 해답

/home/oracle/labs 폴더는 생성한 스크립트를 저장하는 작업 디렉토리입니다.
모든 연습의 해답은 /home/oracle/labs/plsf/soln 폴더에 있습니다.

연습 1: PL/SQL 소개

- 1) 다음 PL/SQL 블록 중 성공적으로 실행되는 블록은 무엇입니까?
 - a) BEGIN
END ;
 - b) DECLARE
v_amount INTEGER(10);
END ;
 - c) DECLARE
BEGIN
END ;
 - d) DECLARE
v_amount INTEGER(10);
BEGIN
DBMS_OUTPUT.PUT_LINE(amount);
END ;
- 2) "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다.
이 스크립트를 실행하여 lab_01_02_soln.sql로 저장합니다.

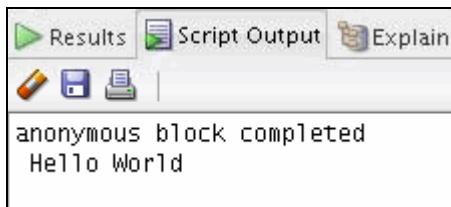
해답 1: PL/SQL 소개

- 1) 다음 PL/SQL 블록 중 성공적으로 실행되는 블록은 무엇입니까?
- BEGIN
END;
 - DECLARE
v_amount INTEGER(10);
END;
 - DECLARE
BEGIN
END;
 - DECLARE
v_amount INTEGER(10);
BEGIN
DBMS_OUTPUT.PUT_LINE(amount);
END;
- a의 블록은 실행되지 않습니다. 이 블록에는 실행문이 없습니다.
b의 블록에는 BEGIN 키워드로 시작하는 필수 실행 섹션이 없습니다.
c의 블록에는 필요한 모든 부분이 있지만 실행문이 없습니다.
d의 블록은 성공적으로 실행됩니다.

- 2) "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다.
 이 스크립트를 실행하여 lab_01_02_soln.sql로 저장합니다.
- 작업 영역에 다음 코드를 입력한 다음 F5 키를 누릅니다.

```
SET SERVEROUTPUT ON
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
```

Script Output 탭에 다음 출력 결과가 표시됩니다.



Save 버튼을 누릅니다. 파일을 저장할 폴더를 선택합니다.
lab_01_02_soln.sql을 파일 이름으로 입력하고 Save를 누릅니다.

단원 2의 연습 및 해답

연습 2: PL/SQL 변수 선언

이 연습에서는 PL/SQL 변수를 선언합니다.

1) 적합한 식별자와 부적합한 식별자를 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:=CURRENT_DATE+1;

3) 다음 익명 블록을 검사하고 적합한 명령문을 선택합니다.

```
DECLARE
    v_fname VARCHAR2(20);
    v_lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
    DBMS_OUTPUT.PUT_LINE(v_fname || ' ' || v_lname);
END;
```

- a) 블록이 성공적으로 실행되고 "fernandez"가 출력됩니다.
- b) fname 변수를 초기화하지 않고 사용했기 때문에 블록에서 오류가 발생합니다.
- c) 블록이 성공적으로 실행되고 "null fernandez"가 출력됩니다.
- d) DEFAULT 키워드를 사용하여 VARCHAR2 유형의 변수를 초기화할 수 없기 때문에 블록에서 오류가 발생합니다.
- e) v_fname 변수를 선언하지 않았기 때문에 블록에서 오류가 발생합니다.

연습 2: PL/SQL 변수 선언 (계속)

- 4) 기존 익명 블록을 수정하고 새 스크립트로 저장합니다.
- 연습 1에서 생성한 `lab_01_02_soln.sql` 스크립트를 업니다.
 - o) PL/SQL 블록에서 다음 변수를 선언합니다.
 - DATE 유형의 `v_today`. SYSDATE를 사용하여 `today`를 초기화합니다.
 - `today` 유형의 `v_tomorrow`. %TYPE 속성을 사용하여 이 변수를 선언합니다.
 - 실행 셋션에서 다음을 수행합니다.
 - 내일 날짜를 계산(`today` 값 + 1)하는 표현식을 사용하여 `v_tomorrow` 변수를 초기화합니다.
 - "Hello World"를 출력한 후 `v_today`와 `v_tomorrow`의 값을 출력합니다.
 - 스크립트를 `lab_02_04_soln.sql`로 저장한 다음 실행합니다.

예제 출력은 다음과 같습니다. 여기서 `v_today`와 `v_tomorrow`의 값은

오늘 날짜와 내일 날짜를 반영하여 달라집니다.

```
anonymous block completed
Hello World
TODAY IS : 05-JUN-09
TOMORROW IS : 06-JUN-09
```

- 5) `lab_02_04_soln.sql` 스크립트를 편집합니다.
- `b_basic_percent`과 `b_pf_percent`라는 두 개의 바인드 변수를 생성하는 코드를 추가합니다. 두 개의 바인드 변수는 모두 NUMBER 유형입니다.
 - PL/SQL 블록의 실행 셋션에서 `b_basic_percent`과 `b_pf_percent`에 각각 값 45와 12를 할당합니다.
 - "/"로 PL/SQL 블록을 종료하고 PRINT 명령을 사용하여 바인드 변수의 값을 표시합니다.
 - 스크립트를 실행하고 `lab_02_05_soln.sql`로 저장합니다. 예제 출력 결과는 다음과 같습니다.

```
anonymous block completed
b_basic_percent
--
45

b_pf_percent
--
12
```

해답 2: PL/SQL 변수 선언

1) 적합한 식별자와 부적합한 식별자를 구분합니다.

- | | |
|---|--------------------------|
| a) today | 적합 |
| b) last_name | 적합 |
| c) today's_date | 부적합 - 문자 "'"는 허용되지 않습니다. |
| d) Number_of_days_in_February_this_year | 부적합 - 너무 길니다. |
| e) Isleap\$year | 적합 |
| f) #number | 부적합 - "#"으로 시작할 수 없습니다. |
| g) NUMBER# | 적합 |
| h) number1to7 | 적합 |

2) 다음 변수 선언 및 초기화가 적합한지 식별합니다.

- | | |
|--|--|
| a) number_of_copies PLS_INTEGER; | 적합 |
| b) PRINTER_NAME constant VARCHAR2(10); | 부적합 |
| c) deliver_to VARCHAR2(10):=Johnson; | 부적합 |
| d) by_when DATE:= CURRENT_DATE+1; | 적합
상수 변수는 선언 시 초기화되어야 하기 때문에 b 의 선언은 적합하지 않습니다.
문자열 리터럴은 작은 따옴표로 묶어야 하기 때문에 c 의 선언은 적합하지 않습니다. |

3) 다음 익명 블록을 검사하고 적합한 명령문을 선택합니다.

```

DECLARE
    v_fname VARCHAR2(20);
    v_lname VARCHAR2(15) DEFAULT 'fernandez';
BEGIN
    DBMS_OUTPUT.PUT_LINE(v_fname || ' ' || v_lname);
END;
  
```

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

해답 2: PL/SQL 변수 선언 (계속)

4) 기존 익명 블록을 수정하고 새 스크립트로 저장합니다.

a) 연습 1에서 생성한 lab_01_02_soln.sql 스크립트를 업니다.

b) PL/SQL 블록에서 다음 변수를 선언합니다.

1. DATE 유형의 변수 v_today. SYSDATE를 사용하여 today를 초기화합니다.

```
DECLARE
    v_today DATE := SYSDATE;
```

2. today 유형의 변수 v_tomorrow. %TYPE 속성을 사용하여 이 변수를 선언합니다.

```
v_tomorrow v_today%TYPE;
```

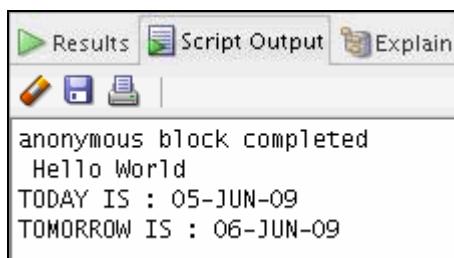
c) 실행 셋션에서 다음을 수행합니다.

1. 내일 날짜를 계산(today 값 + 1)하는 표현식을 사용하여 v_tomorrow 변수를 초기화합니다.
2. "Hello World"를 출력한 후 v_today와 v_tomorrow의 값을 출력합니다.

```
BEGIN
    v_tomorrow := v_today + 1;
    DBMS_OUTPUT.PUT_LINE('Hello World');
    DBMS_OUTPUT.PUT_LINE('TODAY IS : ' || v_today);
    DBMS_OUTPUT.PUT_LINE('TOMORROW IS : ' || v_tomorrow);
END;
```

d) 스크립트를 lab_02_04_soln.sql로 저장한 다음 실행합니다.

예제 출력은 다음과 같습니다. 여기서 v_today와 v_tomorrow의 값은 오늘 날짜와 내일 날짜를 반영하여 달라집니다.



해답 2: PL/SQL 변수 선언 (계속)

5) lab_02_04_soln.sql 스크립트를 편집합니다.

- a) b_basic_percent와 b_pf_percent라는 두 개의 바인드 변수를 생성하는 코드를 추가합니다. 두 개의 바인드 변수는 모두 NUMBER 유형입니다.

```
VARIABLE b_basic_percent NUMBER
VARIABLE b_pf_percent NUMBER
```

- b) PL/SQL 블록의 실행 섹션에서 b_basic_percent와 b_pf_percent에 각각 값 45와 12를 할당합니다.

```
:b_basic_percent:=45;
:b_pf_percent:=12;
```

- c) "/"로 PL/SQL 블록을 종료하고 PRINT 명령을 사용하여 바인드 변수의 값을 표시합니다.

```
/
PRINT b_basic_percent
PRINT b_pf_percent
```

OR

```
PRINT
```

- d) 스크립트를 실행하고 lab_02_05_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.

```
anonymous block completed
b_basic_percent
--
45

b_pf_percent
--
12
```

단원 3의 연습 및 해답

연습 3: 실행문 작성

이 연습에서는 실행문을 검사하고 작성합니다.

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

- 1) 위의 PL/SQL 블록을 평가하고 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형과 값을 결정합니다.
 - a) 위치 1의 v_weight 값:
 - b) 위치 1의 v_new_locn 값:
 - c) 위치 2의 v_weight 값:
 - d) 위치 2의 v_message 값:
 - e) 위치 2의 v_new_locn 값:

```
DECLARE
    v_customer      VARCHAR2(50) := 'Womansport';
    v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
        v_customer  NUMBER(7) := 201;
        v_name      VARCHAR2(25) := 'Unisports';
    BEGIN
        v_credit_rating := 'GOOD';
        ...
    END;
    ...
END;
```

연습 3: 실행문 작성 (계속)

- 2) 위의 PL/SQL 블록에서 다음 각 경우에 대한 값과 데이터 유형을 결정합니다.
- 중첩 블록의 v_customer 값:
 - 중첩 블록의 v_name 값:
 - 중첩 블록의 v_credit_rating 값:
 - 주 블록의 v_customer 값:
 - 주 블록의 v_name 값:
 - 주 블록의 v_credit_rating 값:
- 3) "PL/SQL 변수 선언" 단원의 연습을 실행하는 데 사용한 것과 동일한 세션을 사용합니다. 새 세션을 열었으면 lab_02_05_soln.sql을 실행합니다. 그런 다음 lab_02_05_soln.sql을 다음과 같이 편집합니다.
- 단일 행 주석 구문을 사용하여 바인드 변수를 생성하는 행을 주석 처리하고 SERVEROUTPUT을 캡니다.
 - 실행 셋션에서 다중 행 주석을 사용하여 바인드 변수에 값을 할당하는 행을 주석 처리합니다.
 - 선언 셋션에서 다음을 수행합니다.
 - 주석 처리된 바인드 변수를 대체할 두 개의 임시 변수를 선언하고 초기화합니다.
 - 유형이 VARCHAR2이고 크기가 15인 v_fname과 유형이 NUMBER이고 크기가 10인 v_emp_sal이라는 두 개의 변수를 추가로 선언합니다.
 - 다음 SQL 문을 실행 셋션에 포함합니다.

```
SELECT first_name, salary INTO v_fname, v_emp_sal
FROM employees WHERE employee_id=110;
```

- "Hello World"를 출력하는 행을 "Hello"와 이름을 출력하도록 변경합니다. 그런 다음 날짜를 표시하는 행과 바인드 변수를 출력하는 행을 주석 처리합니다.
- 기업 연금(PF)에 대한 사원의 부담금을 계산합니다. PF는 기본 급여의 12%이며 기본 급여는 급여의 45%입니다. 로컬 변수를 계산에 사용합니다. 표현식을 하나만 사용하여 PF를 계산합니다. 사원의 급여 및 PF 부담금을 출력합니다.
- 스크립트를 실행하고 lab_03_03_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.

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

해답 3: 실행문 작성

이 연습에서는 실행문을 검사하고 작성합니다.

```

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

```

- 1) 위의 PL/SQL 블록을 평가하고 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형과 값을 결정합니다.
 - a) 위치 1의 v_weight 값:
2
데이터 유형은 **NUMBER**
 - b) 위치 1의 new_locn 값:
Western Europe
데이터 유형은 **VARCHAR2**
 - c) 위치 2의 v_weight 값:
601
데이터 유형은 **NUMBER**
 - d) 위치 2의 v_message 값:
제품 10012는 재고 있음
데이터 유형은 **VARCHAR2**

해답 3: 실행문 작성 (계속)

e) 위치 2의 v_new_locn 값:

v_new_locn은 하위 블록 외부에서 볼 수 없기 때문에 잘못된 구문임

```

DECLARE
    v_customer      VARCHAR2(50) := 'Womansport';
    v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
    DECLARE
        v_customer  NUMBER(7) := 201;
        v_name      VARCHAR2(25) := 'Unisports';
    BEGIN
        v_credit_rating := 'GOOD';
        ...
    END;
    ...
END;

```

2) 위의 PL/SQL 블록에서 다음 각 경우에 대한 값과 데이터 유형을 결정합니다.

a) 중첩된 블록의 v_customer 값:

201

데이터 유형은 NUMBER

b) 중첩된 블록의 v_name 값:

Unisports

데이터 유형은 VARCHAR2

c) 중첩된 블록의 v_credit_rating 값:

GOOD

데이터 유형은 VARCHAR2

d) 주 블록의 v_customer 값:

Womansport 데이터 유형은 VARCHAR2

e) 주 블록의 v_name 값:

Null. name은 주 블록에서 볼 수 없으며 오류가 표시됨

f) 주 블록의 v_credit_rating 값:

EXCELLENT

데이터 유형은 VARCHAR2

3) "PL/SQL 변수 선언" 단원의 연습을 실행하는 데 사용한 것과 동일한 세션을 사용합니다. 새 세션을 열었으면 lab_02_05_soln.sql을 실행합니다. 그런 다음 lab_02_05_soln.sql을 다음과 같이 편집합니다.

a) 단일 행 주석 구문을 사용하여 바인드 변수를 생성하는 행을 주석 처리하고 SERVEROUTPUT을 캡니다.

```

-- VARIABLE b_basic_percent NUMBER
-- VARIABLE b_pf_percent NUMBER
SET SERVEROUTPUT ON

```

해답 3: 실행문 작성 (계속)

- b) 실행 셕션에서 다중 행 주석을 사용하여 바인드 변수에 값을 할당하는 행을 주석 처리합니다.

```
/* :b_basic_percent:=45;
:b_pf_percent:=12; */
```

- c) 선언 셕션에서 다음을 수행합니다.

1. 주석 처리된 바인드 변수를 대체할 두 개의 임시 변수를 선언하고 초기화합니다.
2. 유형이 VARCHAR2이고 크기가 15인 v_fname과 유형이 NUMBER이고 크기가 10인 v_emp_sal이라는 두 개의 변수를 추가로 선언합니다.

```
DECLARE
    v_basic_percent NUMBER:=45;
    v_pf_percent NUMBER:=12;
    v_fname VARCHAR2(15);
    v_emp_sal NUMBER(10);
```

- d) 다음 SQL 문을 실행 셕션에 포함합니다.

```
SELECT first_name, salary INTO v_fname, v_emp_sal
FROM employees WHERE employee_id=110;
```

- e) "Hello World"를 출력하는 행을 "Hello"와 이름을 출력하도록 변경합니다. 그런 다음 날짜를 표시하는 행과 바인드 변수를 출력하는 행을 주석 처리합니다.

```
DBMS_OUTPUT.PUT_LINE('Hello '|| v_fname);
/* DBMS_OUTPUT.PUT_LINE('TODAY IS : '|| v_today);
DBMS_OUTPUT.PUT_LINE('TOMORROW IS : ' || v_tomorrow); */
...
...
/
--PRINT b_basic_percent
--PRINT b_pf_percent
```

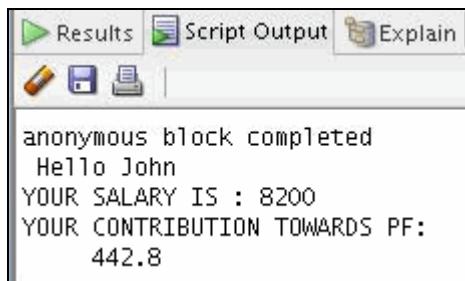
- f) 기업 연금(PF)에 대한 사원의 부담금을 계산합니다.

PF는 기본 급여의 12%이며 기본 급여는 급여의 45%입니다. 로컬 변수를 계산에 사용합니다. 표현식을 하나만 사용하여 PF를 계산합니다. 사원의 급여 및 PF 부담금을 출력합니다.

```
DBMS_OUTPUT.PUT_LINE('YOUR SALARY IS : '||v_emp_sal);
DBMS_OUTPUT.PUT_LINE('YOUR CONTRIBUTION TOWARDS PF:
    '||v_emp_sal*v_basic_percent/100*v_pf_percent/100);
END;
```

해답 3: 실행문 작성 (계속)

- g) 스크립트를 실행하고 lab_03_03_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.



The screenshot shows the 'Script Output' tab of the Oracle SQL Developer interface. The output window displays the results of an anonymous block execution. The text in the window is:

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

연습 4: Oracle 서버와 상호 작용

이 연습에서는 PL/SQL 코드를 사용하여 Oracle 서버와 상호 작용합니다.

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

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

- 2) 단계 1에서 생성한 PL/SQL 블록을 수정하여 departments 테이블에 새 부서를 삽입합니다.
 - a) lab_04_01_soln.sql 스크립트를 로드합니다. 다음 두 개의 변수를 선언합니다.
departments.department_name 유형의 v_dept_name 및
NUMBER 유형의 v_dept_id
선언 섹션에서 v_dept_name에 'Education'을 할당합니다.
 - b) 앞에서 이미 departments 테이블에서 현재 최대 부서 번호를 검색했습니다. 이 부서 번호에 10을 더하여 해당 결과를 v_dept_id에 할당합니다.
 - c) departments 테이블의 department_name, department_id 및 location_id 열에 데이터를 삽입하기 위해 INSERT 문을 포함합니다. department_name과 department_id에 각각 dept_name과 dept_id의 값을 사용하고 location_id에 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 포함합니다.

연습 4: Oracle 서버와 상호 작용 (계속)

- f) 스크립트를 실행하고 lab_04_02_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 270
SQL%ROWCOUNT gives 1

DEPARTMENT_ID      DEPARTMENT_NAME      MANAGER_ID      LOCATION_ID
-----              -----              -----              -----
280                Education           null             null

1 rows selected
```

- 3) 단계 2에서 location_id를 NULL로 설정했습니다. 새 부서의 location_id를 3000으로 갱신하는 PL/SQL 블록을 생성합니다.
참고: 단계 2를 성공적으로 완료했으면 단계 3a로 계속합니다. 그렇지 않으면 먼저 해답 스크립트 /soln/sol_04_02.sql을 실행합니다.
- BEGIN 키워드로 실행 블록을 시작합니다. 새 부서(dep_id = 280)의 location_id를 3000으로 설정하는 UPDATE 문을 포함합니다.
 - END 키워드로 실행 블록을 종료합니다. "/"로 PL/SQL 블록을 종료하고 갱신한 부서가 표시되도록 SELECT 문을 포함합니다.
 - 추가한 부서를 삭제하도록 DELETE 문을 포함합니다.
 - 스크립트를 실행하고 lab_04_03_soln.sql로 저장합니다. 예제 출력은 다음과 같습니다.

```
anonymous block completed
DEPARTMENT_ID      DEPARTMENT_NAME      MANAGER_ID      LOCATION_ID
-----              -----              -----              -----
280                Education           null             3000

1 rows selected
1 rows deleted
```

해답 4: Oracle 서버와 상호 작용

이 연습에서는 PL/SQL 코드를 사용하여 Oracle 서버와 상호 작용합니다.

- 1) departments 테이블에서 최대 부서 ID를 선택하여 v_max_deptno 변수에 저장하는 PL/SQL 블록을 생성합니다. 최대 부서 ID를 표시합니다.

 - a) 선언 섹션에서 NUMBER 유형의 v_max_deptno 변수를 선언합니다.

```
DECLARE
    v_max_deptno  NUMBER;
```

- b) BEGIN 키워드로 실행 섹션을 시작하고 departments 테이블에서 최대 department_id를 검색하는 SELECT 문을 포함합니다.

```
BEGIN
    SELECT MAX(department_id)  INTO v_max_deptno  FROM
        departments;
```

- c) v_max_deptno를 표시하고 실행 블록을 종료합니다.

```
DBMS_OUTPUT.PUT_LINE('The maximum department_id is : ' ||
v_max_deptno);
END;
```

- d) 스크립트를 실행하고 lab_04_01_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.

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

- 2) 단계 1에서 생성한 PL/SQL 블록을 수정하여 departments 테이블에 새 부서를 삽입합니다.

- a) lab_04_01_soln.sql 스크립트를 로드합니다. 다음 두 개의 변수를 선언합니다.

departments.department_name 유형의 v_dept_name 및
NUMBER 유형의 v_dept_id
선언 섹션에서 v_dept_name에 'Education'을 할당합니다.

```
v_dept_name departments.department_name%TYPE:= 'Education';
v_dept_id NUMBER;
```

- b) 앞에서 이미 departments 테이블에서 현재 최대 부서 번호를 검색했습니다. 이 부서 번호에 10을 더하여 해당 결과를 v_dept_id에 할당합니다.

```
v_dept_id := 10 + v_max_deptno;
```

해답 4: Oracle 서버와 상호 작용 (계속)

- c) departments 테이블의 department_name, department_id 및 location_id 열에 데이터를 삽입하기 위해 INSERT 문을 포함합니다. department_name과 department_id에 각각 dept_name 과 dept_id의 값을 사용하고 location_id에 NULL을 사용합니다.

```
...
INSERT INTO departments (department_id, department_name,
location_id)
VALUES (v_dept_id, v_dept_name, NULL);
```

- d) SQL 속성 SQL%ROWCOUNT를 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 포함합니다.

```
...
/
SELECT * FROM departments WHERE department_id= 280;
```

- f) 스크립트를 실행하고 lab_04_02_soln.sql로 저장합니다. 예제 출력 결과는 다음과 같습니다.

```
anonymous block completed
The maximum department_id is : 270
SQL%ROWCOUNT gives 1

DEPARTMENT_ID      DEPARTMENT_NAME      MANAGER_ID      LOCATION_ID
-----            -----
280                Education           null            null
1 rows selected
```

- 3) 단계 2에서 location_id를 NULL로 설정했습니다. 새 부서의 location_id를 3000으로 갱신하는 PL/SQL 블록을 생성합니다.
참고: 단계 2를 성공적으로 완료했으면 단계 3a로 계속합니다. 그렇지 않으면 먼저 해답 스크립트 /soln/sol_04_02.sql을 실행합니다.

- a) BEGIN 키워드로 실행 블록을 시작합니다. 새 부서(dept_id = 280)의 location_id를 3000으로 설정하는 UPDATE 문을 포함합니다.

```
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 포함합니다.

```
END;
/
SELECT * FROM departments WHERE department_id=280;
```

해답 4: Oracle 서버와 상호 작용 (계속)

- c) 추가한 부서를 삭제하도록 DELETE 문을 포함합니다.

```
DELETE FROM departments WHERE department_id=280;
```

- d) 스크립트를 실행하고 lab_04_03_soln.sql로 저장합니다. 예제 출력은 다음과 같습니다.

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

연습 5: 제어 구조 작성

이 연습에서는 루프와 조건부 제어 구조를 포함하는 PL/SQL 블록을 생성합니다.

이 연습에서는 다양한 IF 문과 LOOP 생성자에 대한 이해를 점검합니다.

- 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 작동하는지 확인합니다.

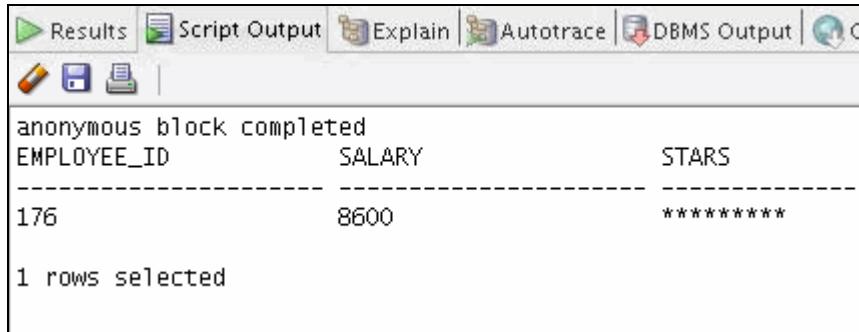
결과: 출력 결과는 다음과 같아야 합니다.

```
anonymous block completed
RESULTS
-----
1
2
3
4
5
7
9
10
8 rows selected
```

- 2) lab_05_02.sql 스크립트를 실행합니다. 이 스크립트는 employees 테이블의 복제본인 emp 테이블을 생성합니다. 이 스크립트는 데이터 유형이 VARCHAR2이고 크기가 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) 블록의 선언 섹션에서 emp.employee_id 유형의 v_empno 변수를 선언하고 176으로 초기화합니다. emp.stars 유형의 v_asterisk 변수를 선언하고 NULL로 초기화합니다. emp.salary 유형의 v_sal 변수를 생성합니다.
 - b) 실행 섹션에서 사원의 급여에 대해 \$1,000 단위마다 문자열에 별표(*)를 추가하는 논리를 작성합니다. 예를 들어, 사원의 급여가 \$8,000이면 별표 문자열에는 8개의 별표가 있어야 합니다. 급여가 \$12,500이면 별표 문자열에는 13개의 별표가 있어야 합니다.

연습 5: 제어 구조 작성 (계속)

- c) 해당 사원의 stars 열을 별표 문자열로 갱신합니다. 블록 종료 전에 커밋합니다.
- d) emp 테이블의 행을 출력하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다.
- e) 스크립트를 실행하고 lab_05_02_soln.sql로 저장합니다. 출력 결과는 다음과 같습니다.



```
anonymous block completed
EMPLOYEE_ID      SALARY      STARS
-----
176              8600      *****
1 rows selected
```

해답 5: 제어 구조 작성

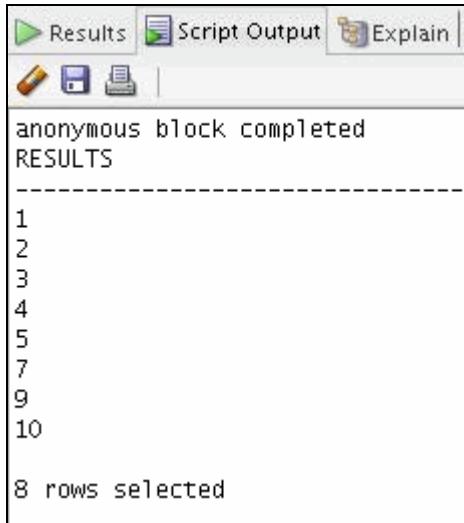
- 1) lab_05_01.sql 파일에서 messages 테이블을 생성하는 명령을 실행합니다.
 messages 테이블에 숫자를 삽입하는 PL/SQL 블록을 작성합니다.
- 1에서 10까지의 숫자를 삽입합니다(6, 8 제외).
 - 블록 종료 전에 커밋합니다.

```
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;
```

결과: 출력 결과는 다음과 같아야 합니다.



The screenshot shows the Oracle SQL developer interface with the 'Results' tab selected. The output window displays the results of an anonymous block execution:

```
anonymous block completed
RESULTS
-----
1
2
3
4
5
7
9
10
8 rows selected
```

해답 5: 제어 구조 작성 (계속)

2) lab_05_02.sql 스크립트를 실행합니다. 이 스크립트는 employees 테이블의 복제본인 emp 테이블을 생성합니다. 이 스크립트는 데이터 유형이 VARCHAR2이고 크기가 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) 블록의 선언 섹션에서 emp.employee_id 유형의 v_empno 변수를 선언하고 176으로 초기화합니다. emp.stars 유형의 v_asterisk 변수를 선언하고 NULL로 초기화합니다. emp.salary 유형의 v_sal 변수를 생성합니다.

```
DECLARE
    v_empno      emp.employee_id%TYPE := 176;
    v_asterisk   emp.stars%TYPE := NULL;
    v_sal        emp.salary%TYPE;
```

- b) 실행 섹션에서 사원의 급여에 대해 \$1,000 단위마다 문자열에 별표(*)를 추가하는 논리를 작성합니다. 예를 들어, 사원의 급여가 \$8,000이면 별표 문자열에는 8개의 별표가 있어야 합니다. 급여가 \$12,500이면 별표 문자열에는 13개의 별표가 있어야 합니다.

```
BEGIN
    SELECT NVL(ROUND(salary/1000), 0) INTO v_sal
    FROM emp WHERE employee_id = v_empno;

    FOR i IN 1..v_sal
    LOOP
        v_asterisk := v_asterisk || '*';
    END LOOP;
```

- c) 해당 사원의 stars 열을 별표 문자열로 갱신합니다. 블록 종료 전에 커밋합니다.

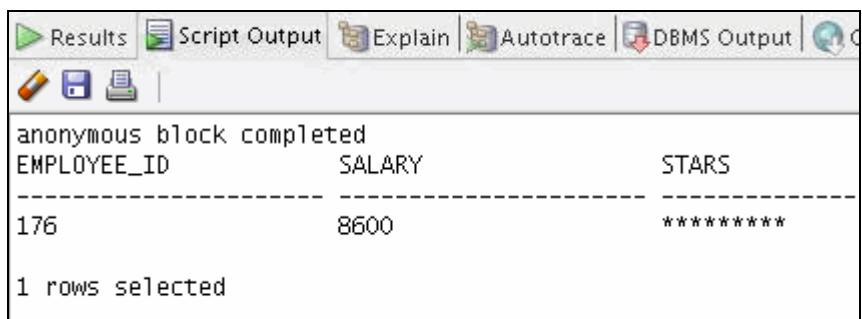
```
UPDATE emp SET stars = v_asterisk
WHERE employee_id = v_empno;
COMMIT;
END;
/
```

- d) emp 테이블의 행을 출력하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다.

```
SELECT employee_id, salary, stars
FROM emp WHERE employee_id = 176;
```

해답 5: 제어 구조 작성 (계속)

- e) 스크립트를 실행하고 lab_05_02_soln.sql로 저장합니다. 출력 결과는 다음과 같습니다.



The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output window displays the results of an anonymous block. The output is as follows:

```
anonymous block completed
EMPLOYEE_ID      SALARY      STARS
-----
176              8600        *****
1 rows selected
```

단원 6의 연습 및 해답

연습 6: 조합 데이터 유형 작업

- 1) 제공된 국가에 대한 정보를 출력하는 PL/SQL 블록을 작성합니다.
 - a) countries 테이블의 구조에 맞게 PL/SQL 레코드를 선언합니다.
 - b) v_countryid 변수를 선언합니다. CA를 v_countryid에 할당합니다.
 - c) 선언 섹션에서 %ROWTYPE 속성을 사용하여 countries 유형의 v_country_record 변수를 선언합니다.
 - d) 실행 섹션에서 v_countryid를 사용하여 countries 테이블의 모든 정보를 가져옵니다. 선택한 국가의 정보를 표시합니다. 예제 출력은 다음과 같습니다.

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

- e) ID가 DE, UK 및 US인 국가에 대해 PL/SQL 블록을 실행하여 테스트할 수도 있습니다.
- 2) 연관 배열과 통합하여 departments 테이블에서 일부 부서의 이름을 검색하고 각 부서 이름을 화면에 출력하는 PL/SQL 블록을 생성합니다. 스크립트를 lab_06_02_soln.sql로 저장합니다.
 - a) departments.department_name 유형의 INDEX BY 테이블 dept_table_type을 선언합니다. dept_table_type 유형의 my_dept_table 변수를 선언하여 부서 이름을 임시로 저장합니다.
 - b) NUMBER 유형의 두 가지 변수 f_loop_count 및 v_deptno를 선언합니다. f_loop_count에 10을 할당하고 v_deptno에 0을 할당합니다.

연습 6: 조합 데이터 유형 작업 (계속)

- c) 루프를 사용하여 10개 부서의 이름을 검색하고 연관 배열에 이름을 저장합니다. department_id 10으로 시작합니다. 루프가 반복할 때마다 v_deptno를 10씩 증가시킵니다. 다음 표는 department_name을 검색할 department_id를 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

- d) 다른 루프를 사용하여 연관 배열에서 부서 이름을 검색하고 표시합니다.
- 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 부서에 대한 모든 정보를 검색하고 표시하도록 연습 2에서 생성한 블록을 수정합니다. INDEX BY 레코드 테이블 메소드와 함께 연관 배열을 사용합니다.
- lab_06_02_soln.sql 스크립트를 로드합니다.
 - 연관 배열을 departments.department_name 유형으로 선언했습니다. 모든 부서의 번호, 이름 및 위치를 임시로 저장하도록 연관 배열의 선언을 수정합니다. %ROWTYPE 속성을 사용합니다.

연습 6: 조합 데이터 유형 작업 (계속)

- c) 현재 departments 테이블에 있는 모든 부서 정보를 검색하도록 SELECT 문을 수정하고 연관 배열에 저장합니다.
- d) 다른 루프를 사용하여 연관 배열에서 부서 정보를 검색하고 표시합니다.

예제 출력 결과는 다음과 같습니다.

```
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
```

해답 6: 조합 데이터 유형 작업

- 1) 제공된 국가에 대한 정보를 출력하는 PL/SQL 블록을 작성합니다.
 - a) countries 테이블의 구조에 맞게 PL/SQL 레코드를 선언합니다.
 - b) v_countryid 변수를 선언합니다. CA를 v_countryid에 할당합니다.

```
SET SERVEROUTPUT ON

SET VERIFY OFF
DECLARE
  v_countryid varchar2(20):= 'CA';
```

- c) 선언 섹션에서 %ROWTYPE 속성을 사용하여 countries 유형의 v_country_record 변수를 선언합니다.

```
v_country_record countries%ROWTYPE;
```

- d) 실행 섹션에서 v_countryid를 사용하여 countries 테이블의 모든 정보를 가져옵니다. 선택한 국가의 정보를 표시합니다. 예제 출력은 다음과 같습니다.

```
BEGIN
  SELECT *
  INTO   v_country_record
  FROM   countries
  WHERE  country_id = UPPER(v_countryid);

  DBMS_OUTPUT.PUT_LINE ('Country Id: ' ||
    v_country_record.country_id ||
    ' Country Name: ' || v_country_record.country_name
    || ' Region: ' || v_country_record.region_id);

END;
```

```
anonymous block completed
Country Id: CA Country Name: Canada Region: 2
```

- e) ID가 DE, UK 및 US인 국가에 대해 PL/SQL 블록을 실행하여 테스트할 수도 있습니다.

해답 6: 조합 데이터 유형 작업 (계속)

2) 연관 배열과 통합하여 departments 테이블에서 일부 부서의 이름을 검색하고 각 부서 이름을 화면에 출력하는 PL/SQL 블록을 생성합니다. 스크립트를 lab_06_02_soln.sql로 저장합니다.

- a) departments.department_name 유형의 INDEX BY 테이블 dept_table_type을 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 두 가지 변수 f_loop_count 및 v_deptno를 선언합니다. f_loop_count에 10을 할당하고 v_deptno에 0을 할당합니다.

```
loop_count NUMBER (2):=10;
deptno          NUMBER (4):=0;
```

- c) 루프를 사용하여 10개 부서의 이름을 검색하고 연관 배열에 이름을 저장합니다. department_id를 10에서 시작합니다. 루프가 반복할 때마다 v_deptno를 10씩 증가시킵니다. 다음 표는 department_name을 검색하고 연관 배열에 저장할 department_id를 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..f_loop_count
    LOOP
        v_deptno:=v_deptno+10;
        SELECT department_name
        INTO my_dept_table(i)
        FROM departments
        WHERE department_id = v_deptno;
    END LOOP;
```

해답 6: 조합 데이터 유형 작업 (계속)

- d) 다른 루프를 사용하여 연관 배열에서 부서 이름을 검색하고 표시합니다.

```
FOR i IN 1..f_loop_count
LOOP
    DBMS_OUTPUT.PUT_LINE (my_dept_table(i));
END LOOP;
END;
```

- 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 부서에 대한 모든 정보를 검색하고 표시하도록 연습 2에서 생성한 블록을 수정합니다. INDEX BY 레코드 테이블 메소드와 함께 연관 배열을 사용합니다.
- lab_06_02_soln.sql 스크립트를 로드합니다.
 - 연관 배열을 departments.department_name 유형으로 선언했습니다. 모든 부서의 번호, 이름 및 위치를 임시로 저장하도록 연관 배열의 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;
    f_loop_count      NUMBER (2):=10;
    v_deptno         NUMBER (4):=0;
```

해답 6: 조합 데이터 유형 작업 (계속)

- c) 현재 departments 테이블에 있는 모든 부서 정보를 검색하도록 SELECT 문을 수정하고 연관 배열에 저장합니다.

```
BEGIN
    FOR i IN 1..f_loop_count
    LOOP
        v_deptno := v_deptno + 10;
        SELECT *
        INTO my_dept_table(i)
        FROM departments
        WHERE department_id = v_deptno;
    END LOOP;
```

- d) 다른 루프를 사용하여 연관 배열에서 부서 정보를 검색하고 표시합니다.

```
FOR i IN 1..f_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;
```

예제 출력 결과는 다음과 같습니다.

```
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
```

연습 7-1: 명시적 커서 사용

이 연습에서는 다음 두 가지 연습을 수행합니다.

- 먼저 명시적 커서를 사용하여 테이블의 여러 행을 처리하고, 커서 FOR 루프를 사용하여 다른 테이블을 결과로 채웁니다.
- 그런 다음 파라미터를 사용하는 커서를 포함한 두 개의 커서로 정보를 처리하는 PL/SQL 블록을 작성합니다.

1) 다음을 수행하는 PL/SQL 블록을 생성합니다.

- 선언 섹션에서 NUMBER 유형의 v_deptno 변수를 선언하고 초기화합니다. 적합한 부서 ID 값을 할당합니다(값은 단계 d 참조).
- v_deptno에서 지정된 부서에 근무하는 사원의 last_name, salary 및 manager_id를 검색하는 c_emp_cursor 커서를 선언합니다.
- 실행 섹션에서 커서 FOR 루프를 사용하여 검색된 데이터에 대해 작업할 수 있습니다. 사원의 급여가 5,000 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.
- 다음 경우에 대해 PL/SQL 블록을 테스트합니다.

부서 ID	메시지
10	Whalen Due for a raise
20	Hartstein Not Due for a raise Fay Not Due for a raise
50	Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise Vollman Not Due for a raise. OConnell Due for a raise Grant Due for a raise
80	Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise . . . Livingston Not Due for a raise Johnson Not Due for a raise

연습 7-1: 명시적 커서 사용 (계속)

- 2) 그런 다음 두 개의 커서(하나는 파라미터가 없고 다른 하나는 파라미터가 있음)를 선언하고 사용하는 PL/SQL 블록을 작성합니다. 첫번째 커서는 departments 테이블에서 ID 번호가 100보다 작은 모든 부서의 부서 번호와 부서 이름을 검색합니다. 두번째 커서는 부서 번호를 파라미터로 수신한 다음, 해당 부서에서 근무하고 employee_id가 120보다 작은 사원의 세부 정보를 검색합니다.
- department_id가 100보다 작은 부서의 department_id와 department_name을 검색하는 c_dept_cursor 커서를 선언합니다. department_id를 기준으로 정렬합니다.
 - 부서 번호를 파라미터로 사용하여 employees 테이블에서 해당 부서에 근무하고 employee_id가 120보다 작은 사원의 last_name, job_id, hire_date, salary 등의 데이터를 검색하는 다른 c_emp_cursor 커서를 선언합니다.
 - 각 커서에서 검색된 값을 보유하는 변수를 선언합니다. 변수 선언 시 %TYPE 속성을 사용합니다.
 - c_dept_cursor를 열고 간단한 루프를 사용하여 값을 선언된 변수에 패치(fetch)합니다. 부서 번호 및 부서 이름을 출력합니다. 해당 커서 속성을 사용하여 루프를 종료합니다.
 - 현재 부서 번호를 파라미터로 전달하여 c_emp_cursor를 엽니다. 다른 루프를 시작하여 emp_cursor의 값을 변수에 패치(fetch)하고 employees 테이블에서 검색된 모든 상세 정보를 출력합니다.

참고

- 커서를 열기 전에 c_emp_cursor가 이미 열려 있는지 확인하십시오.
- 종료 조건에 적합한 커서 속성을 사용하십시오.
- 루프가 완료되면 각 부서의 세부 정보를 표시한 후 행을 출력하고 c_emp_cursor를 닫으십시오.

연습 7-1: 명시적 커서 사용 (계속)

- f) 첫번째 루프를 종료하고 c_dept_cursor를 닫습니다. 그런 다음 실행 셕션을 종료합니다.
- g) 스크립트를 실행합니다. 예제 출력 결과는 다음과 같습니다.

```

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 7-2: 명시적 커서 사용 - 선택 사항

시간 여유가 있을 경우 선택 사항인 다음 연습을 완료하십시오. 여기서는 명시적 커서를 사용하여 사원의 상위 n 개 급여를 결정하는 PL/SQL 블록을 생성합니다.

- 1) 사원의 급여를 저장하기 위해 lab_07-2.sql 스크립트를 실행하여 top_salaries 테이블을 생성합니다.
- 2) 선언 섹션에서 employees 테이블의 상위 n 개 급여를 받는 사원 수를 나타내는 숫자 n 을 보유하는 NUMBER 유형의 v_num 변수를 선언합니다. 예를 들어, 상위 다섯 개의 급여를 보려면 5를 입력합니다. employees.salary 유형의 또 다른 변수 sal 을 선언합니다. 사원의 급여를 내림차순으로 검색하는 c_emp_cursor 커서를 선언합니다. 급여는 중복될 수 없습니다.
- 3) 실행 섹션에서 루프를 열고 상위 n 개의 급여를 패치(fetch)한 다음 top_salaries 테이블에 삽입합니다. 간단한 루프를 사용하여 데이터를 산출할 수 있습니다. 또한 종료 조건에 %ROWCOUNT 및 %FOUND 속성을 사용합니다.
참고: 무한 루프가 발생하지 않도록 종료 조건을 추가해야 합니다.
- 4) top_salaries 테이블에 데이터를 삽입한 후 SELECT 문을 사용하여 행을 표시합니다. 결과로 employees 테이블의 상위 5개 급여가 출력됩니다.

SALARY
24000
17000
17000
14000
13500

- 5) $v_num =$ 또는 $v_num \leq$ employees 테이블의 사원 수보다 더 큰 경우와 같이 다양한 특수 경우를 테스트해 봅니다. 각 테스트 후에는 top_salaries 테이블을 비웁니다.

해답 7-1: 명시적 커서 사용

이 연습에서는 다음 두 가지 연습을 수행합니다.

- 먼저 명시적 커서를 사용하여 테이블의 여러 행을 처리하고, 커서 FOR 루프를 사용하여 다른 테이블을 결과로 채웁니다.
- 그런 다음 파라미터를 사용하는 커서를 포함한 두 개의 커서로 정보를 처리하는 PL/SQL 블록을 작성합니다.

1) 다음을 수행하는 PL/SQL 블록을 생성합니다.

- a) 선언 섹션에서 NUMBER 유형의 v_deptno 변수를 선언하고 초기화합니다.
적합한 부서 ID 값을 할당합니다(값은 단계 d 참조).

```
DECLARE
  v_deptno NUMBER := 10;
```

- b) v_deptno에서 지정된 부서에 근무하는 사원의 last_name, salary 및 manager_id를 검색하는 c_emp_cursor 커서를 선언합니다.

```
CURSOR c_emp_cursor IS
  SELECT      last_name, salary, manager_id
  FROM        employees
  WHERE       department_id = v_deptno;
```

- c) 실행 섹션에서 커서 FOR 루프를 사용하여 검색된 데이터에 대해 작업할 수 있습니다. 사원의 급여가 5,000 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.

```
BEGIN
  FOR emp_record IN c_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;
```

해답 7-1: 명시적 커서 사용 (계속)

- d) 다음 경우에 대해 PL/SQL 블록을 테스트합니다.

부서 ID	메시지
10	Whalen Due for a raise
20	Hartstein Not Due for a raise Fay Not Due for a raise
50	Weiss Not Due for a raise Fripp Not Due for a raise Kaufling Not Due for a raise Vollman Not Due for a raise. OConnell Due for a raise Grant Due for a raise
80	Russell Not Due for a raise Partners Not Due for a raise Errazuriz Not Due for a raise Cambrault Not Due for a raise . . . Livingston Not Due for a raise Johnson Not Due for a raise

- 2) 그런 다음 두 개의 커서(하나는 파라미터가 없고 다른 하나는 파라미터가 있음)를 선언하고 사용하는 PL/SQL 블록을 작성합니다. 첫번째 커서는 departments 테이블에서 ID 번호가 100보다 작은 모든 부서의 부서 번호와 부서 이름을 검색합니다. 두번째 커서는 부서 번호를 파라미터로 수신한 다음, 해당 부서에서 근무하고 employee_id가 120보다 작은 사원의 세부 정보를 검색합니다.
- a) department_id가 100보다 작은 부서의 department_id와 department_name을 검색하는 c_dept_cursor 커서를 선언합니다. department_id를 기준으로 정렬합니다.

```
DECLARE
  CURSOR c_dept_cursor IS
    SELECT department_id,department_name
    FROM departments
    WHERE department_id < 100
    ORDER BY      department_id;
```

해답 7-1: 명시적 커서 사용 (계속)

- b) 부서 번호를 파라미터로 사용하여 employees 테이블에서 해당 부서에 근무하고 employee_id가 120보다 작은 사원의 last_name, job_id, hire_date, salary 등의 데이터를 검색하는 다른 c_emp_cursor 커서를 선언합니다.

```
CURSOR c_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;
```

- c) 각 커서에서 검색된 값을 보유하는 변수를 선언합니다. 변수 선언 시 %TYPE 속성을 사용합니다.

```
v_current_deptno departments.department_id%TYPE;
v_current_dname departments.department_name%TYPE;
v_ename employees.last_name%TYPE;
v_job employees.job_id%TYPE;
v_hiredate employees.hire_date%TYPE;
v_sal employees.salary%TYPE;
```

- d) c_dept_cursor를 열고 간단한 루프를 사용하여 값을 선언된 변수에 패치(fetch)합니다. 부서 번호 및 부서 이름을 출력합니다. 해당 커서 속성을 사용하여 루프를 종료합니다.

```
BEGIN
    OPEN c_dept_cursor;
    LOOP
        FETCH c_dept_cursor INTO v_current_deptno,
                           v_current_dname;
        EXIT WHEN c_dept_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE ('Department Number : ' ||
                               v_current_deptno || ' Department Name : ' ||
                               v_current_dname);
    END LOOP;
END;
```

해답 7-1: 명시적 커서 사용 (계속)

- e) 현재 부서 번호를 파라미터로 전달하여 c_emp_cursor를 엽니다. 다른 루프를 시작하여 emp_cursor의 값을 변수에 패치(fetch)하고 employees 테이블에서 검색된 모든 상세 정보를 출력합니다.

참고

- 커서를 열기 전에 c_emp_cursor가 이미 열려 있는지 확인하십시오.
- 종료 조건에 적합한 커서 속성을 사용하십시오.
- 루프가 완료되면 각 부서의 세부 정보를 표시한 후 행을 출력하고 c_emp_cursor를 닫으십시오.

```
IF c_emp_cursor%ISOPEN THEN
    CLOSE c_emp_cursor;
END IF;
OPEN c_emp_cursor (v_current_deptno);
LOOP
    FETCH c_emp_cursor INTO v_ename,v_job,v_hiredate,v_sal;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE (v_ename || ' ' || v_job
                          || ' ' || v_hiredate || ' ' ||
                          v_sal);
END LOOP;
DBMS_OUTPUT.PUT_LINE ('-----');
CLOSE c_emp_cursor;
```

- f) 첫번째 루프를 종료하고 c_dept_cursor를 닫습니다. 그런 다음 실행 섹션을 종료합니다.

```
END LOOP;
CLOSE c_dept_cursor;
END;
```

해답 7-1: 명시적 커서 사용 (계속)

g) 스크립트를 실행합니다. 예제 출력 결과는 다음과 같습니다.

```
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
```

해답 7-2: 명시적 커서 사용 - 선택 사항

시간 여유가 있을 경우 선택 사항인 다음 연습을 완료하십시오. 여기서는 명시적 커서를 사용하여 사원의 상위 n개 급여를 결정하는 PL/SQL 블록을 생성합니다.

- 1) 사원의 급여를 저장하기 위해 lab_07-02.sql 스크립트를 실행하여 새 테이블 top_salaries를 생성합니다.
- 2) 선언 섹션에서 employees 테이블의 상위 n개 급여를 받는 사원 수를 나타내는 숫자 n을 보유하는 NUMBER 유형의 v_num 변수를 선언합니다. 예를 들어, 상위 다섯 개의 급여를 보려면 5를 입력합니다. employees.salary 유형의 또 다른 변수 sal을 선언합니다. 사원의 급여를 내림차순으로 검색하는 c_emp_cursor 커서를 선언합니다. 급여는 중복될 수 없습니다.

```
DECLARE
    v_num          NUMBER(3) := 5;
    v_sal          employees.salary%TYPE;
    CURSOR         c_emp_cursor IS
        SELECT      salary
        FROM        employees
        ORDER BY    salary DESC;
```

- 3) 실행 섹션에서 루프를 열고 상위 n개의 급여를 패치(fetch)한 다음 top_salaries 테이블에 삽입합니다. 간단한 루프를 사용하여 데이터를 산출할 수 있습니다. 또한 종료 조건에 %ROWCOUNT 및 %FOUND 속성을 사용합니다.

참고: 무한 루프가 발생하지 않도록 종료 조건을 추가해야 합니다.

```
BEGIN
    OPEN c_emp_cursor;
    FETCH c_emp_cursor INTO v_sal;
    WHILE c_emp_cursor%ROWCOUNT <= v_num AND
c_emp_cursor%FOUND LOOP
        INSERT INTO top_salaries (salary)
        VALUES (v_sal);
        FETCH c_emp_cursor INTO v_sal;
    END LOOP;
    CLOSE c_emp_cursor;
END;
```

해답 7-2: 명시적 커서 사용 – 선택 사항 (계속)

- 4) top_salaries 테이블에 데이터를 삽입한 후 SELECT 문을 사용하여 행을 표시합니다. 결과로 employees 테이블의 상위 5개 급여가 출력됩니다.

```
/  
SELECT * FROM top_salaries;
```

예제 출력 결과는 다음과 같습니다.

SALARY

24000
17000
17000
14000
13500

- 5) v_num = 0 또는 v_num> employees 테이블의 사원 수보다 더 큰 경우와 같이 다양한 특수 경우를 테스트해 봅니다. 각 테스트 후에는 top_salaries 테이블을 비웁니다.

연습 8-1: 미리 정의된 예외 처리

이 연습에서는 한 번에 한 레코드만 처리하기 위해 미리 정의된 예외를 적용하는 PL/SQL 블록을 작성합니다. 이 PL/SQL 블록은 제공된 급여 값을 가진 사원의 이름을 선택합니다.

- 1) lab_05_01.sql 파일에서 messages 테이블을 다시 생성하는 명령을 실행합니다.
- 2) 선언 섹션에서 employees.last_name 유형의 v_ename 변수와 employees.salary 유형의 v_emp_sal 변수를 선언합니다. v_emp_sal 변수를 6000으로 초기화합니다.
- 3) 실행 섹션에서 급여가 v_emp_sal의 값과 동일한 사원의 성을 검색합니다. 입력된 급여가 한 행만 반환하는 경우 messages 테이블에 사원 이름 및 급여를 삽입합니다.
참고: 명시적 커서를 사용하지 마십시오.
- 4) 입력된 급여가 행을 반환하지 않으면 적합한 예외 처리기로 예외를 처리하고 messages 테이블에 "No employee with a salary of <salary>" 메시지를 삽입합니다.
- 5) 입력된 급여가 여러 행을 반환하면 적합한 예외 처리기로 예외를 처리하고 messages 테이블에 "More than one employee with a salary of <salary>" 메시지를 삽입합니다.
- 6) 기타 예외의 경우 적합한 예외 처리기로 처리하고 messages 테이블에 "Some other error occurred" 메시지를 삽입합니다.

연습 8-1: 미리 정의된 예외 처리 (계속)

- 7) messages 테이블의 행을 표시하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다. 출력 결과는 다음과 같습니다.

```
RESULTS
-----
More than one employee with a salary of 6000
1 rows selected
```

- 8) v_emp_sal의 초기화된 값을 2000으로 변경하고 다시 실행합니다. 출력 결과는 다음과 같습니다.

```
RESULTS
-----
More than one employee with a salary of 6000
No employee with a salary of 2000
2 rows selected
```

연습 8-2: 표준 Oracle 서버 예외 처리

이 연습에서는 Oracle 서버 오류 ORA-02292(integrity constraint violated - child record found)에 대한 예외를 선언하는 PL/SQL 블록을 작성합니다. 이 블록은 예외를 테스트하고 오류 메시지를 출력합니다.

- 1) 선언 섹션에서 예외 e_childrecord_exists를 선언합니다. 선언된 예외를 표준 Oracle 서버 오류 -02292와 연결합니다.
- 2) 실행 섹션에서 "Deleting department 40...."을 표시합니다. department_id가 40인 부서를 삭제하는 DELETE 문을 포함합니다.
- 3) e_childrecord_exists 예외를 처리하고 적절한 메시지를 출력하는 예외 섹션을 포함합니다.

예제 출력 결과는 다음과 같습니다.

```
anonymous block completed
Deleting department 40.....
Cannot delete this department. There are employees in this department (child records exist.)
```

해답 8-1: 미리 정의된 예외 처리

이 연습에서는 한 번에 한 레코드만 처리하기 위해 미리 정의된 예외를 적용하는 PL/SQL 블록을 작성합니다. 이 PL/SQL 블록은 제공된 급여 값을 가진 사원의 이름을 선택합니다.

- 1) lab_05_01.sql 파일에서 messages 테이블을 다시 생성하는 명령을 실행합니다.
- 2) 선언 섹션에서 employees.last_name 유형의 v_ename 변수와 employees.salary 유형의 v_emp_sal 변수를 선언합니다. v_emp_sal 변수를 6000으로 초기화합니다.

```
DECLARE
    v_ename      employees.last_name%TYPE;
    v_emp_sal   employees.salary%TYPE := 6000;
```

- 3) 실행 섹션에서 급여가 v_emp_sal의 값과 동일한 사원의 성을 검색합니다. 입력된 급여가 한 행만 반환하는 경우 messages 테이블에 사원 이름 및 급여를 삽입합니다.

참고: 명시적 커서를 사용하지 마십시오.

```
BEGIN
    SELECT last_name
    INTO      v_ename
    FROM     employees
    WHERE    salary = v_emp_sal;
    INSERT INTO messages (results)
    VALUES (v_ename || ' - ' || v_emp_sal);
```

- 4) 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(v_emp_sal));
```

- 5) 입력된 급여가 여러 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(v_emp_sal));
```

해답 8-1: 미리 정의된 예외 처리 (계속)

- 6) 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;
```

- 7) messages 테이블의 행을 표시하여 PL/SQL 블록이 성공적으로 실행되었는지 확인합니다.

```
/  
SELECT * FROM messages;
```

출력 결과는 다음과 같습니다.

RESULTS

More than one employee with a salary of 6000
1 rows selected

- 8) v_emp_sal의 초기화된 값을 2000으로 변경하고 다시 실행합니다. 출력 결과는 다음과 같습니다.

RESULTS

More than one employee with a salary of 6000
No employee with a salary of 2000
2 rows selected

해답 8-2: 표준 Oracle 서버 예외 처리

이 연습에서는 Oracle 서버 오류 ORA-02292(integrity constraint violated - child record found)에 대한 예외를 선언하는 PL/SQL 블록을 작성합니다. 이 블록은 예외를 테스트하고 오류 메시지를 출력합니다.

- 선언 섹션에서 예외 e_childrecord_exists를 선언합니다. 선언된 예외를 표준 Oracle 서버 오류 -02292와 연결합니다.

```
SET SERVEROUTPUT ON
DECLARE
    e_childrecord_exists EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_childrecord_exists, -02292);
```

- 실행 섹션에서 "Deleting department 40...."을 표시합니다. department_id가 40인 부서를 삭제하는 DELETE 문을 포함합니다.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Deleting department 40.....');
    delete from departments where department_id=40;
```

- e_childrecord_exists 예외를 처리하고 적절한 메시지를 출력하는 예외 섹션을 포함합니다.

```
EXCEPTION
    WHEN e_childrecord_exists THEN
        DBMS_OUTPUT.PUT_LINE(' Cannot delete this department.
There are employees in this department (child records
exist.) ');
END;
```

예제 출력 결과는 다음과 같습니다.

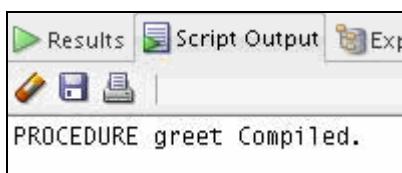
```
anonymous block completed
Deleting department 40.....
Cannot delete this department. There are employees in this department (child records exist.)
```

단원 9의 연습 및 해답

연습 9: 내장 프로시저 생성 및 사용

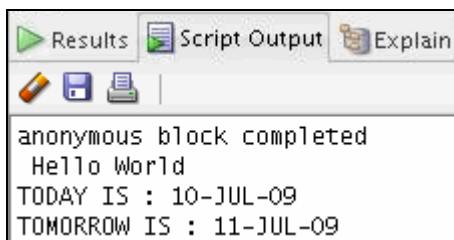
이 연습에서는 내장 프로시저를 생성하고 사용하도록 기존 스크립트를 수정합니다.

- 1) /home/oracle/plsf/soln/ 폴더에서 sol_02_04.sql 스크립트를 로드합니다.
 - a) 익명 블록을 greet라는 프로시저로 변환하도록 스크립트를 수정합니다.
(힌트: SET SERVEROUTPUT ON 명령도 제거합니다.)
 - b) 스크립트를 실행하여 프로시저를 작성합니다. 출력 결과는 다음과 같아야 합니다.



- c) 이 스크립트를 lab_09_01_soln.sql로 저장합니다.
 - d) Clear 버튼을 눌러 작업 영역을 지웁니다.
 - e) greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다.
(힌트: 블록 시작 부분에서 SERVEROUTPUT을 활성화해야 합니다.)

출력 결과는 다음과 유사해야 합니다.

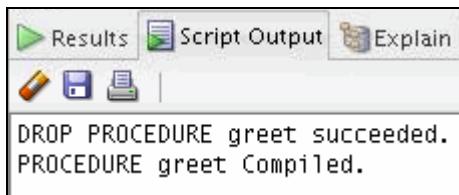


- 2) lab_09_01_soln.sql 스크립트를 다음과 같이 수정합니다.
 - a) 다음 명령을 실행하여 greet 프로시저를 삭제합니다.

```
DROP PROCEDURE greet;
```
 - b) VARCHAR2 유형의 인수를 받아들이도록 프로시저를 수정합니다. p_name 인수를 호출합니다.
 - c) Hello World를 출력하는 대신 Hello <name>(인수 내용)을 출력합니다.
 - d) 스크립트를 lab_09_02_soln.sql로 저장합니다.

연습 9: 내장 프로시저 생성 및 사용 (계속)

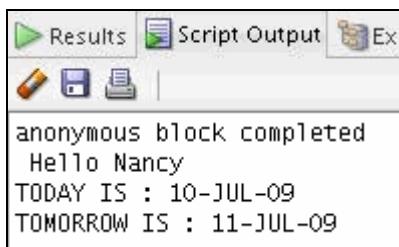
- e) 스크립트를 실행하여 프로시저를 작성합니다. 출력 결과는 다음과 같아야 합니다.



```
DROP PROCEDURE greet succeeded.  
PROCEDURE greet Compiled.
```

- f) 파라미터 값을 사용하여 greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 블록에서 출력도 생성해야 합니다.

예제 출력 결과는 다음과 유사해야 합니다.



```
anonymous block completed  
Hello Nancy  
TODAY IS : 10-JUL-09  
TOMORROW IS : 11-JUL-09
```

해답 9: 내장 프로시저 생성 및 사용

이 연습에서는 내장 프로시저를 생성하고 사용하도록 기존 스크립트를 수정합니다.

- 1) /home/oracle/plsf/soln/ 폴더에서 sol_02_04.sql 스크립트를 로드합니다.

- a) 익명 블록을 greet라는 프로시저로 변환하도록 스크립트를 수정합니다.
(힌트: SET SERVEROUTPUT ON 명령도 제거합니다.)

```
CREATE PROCEDURE greet IS
    V_today DATE:=SYSDATE;
    V_tomorrow today%TYPE;
...

```

- b) 스크립트를 실행하여 프로시저를 작성합니다. 출력 결과는 다음과 같아야 합니다.

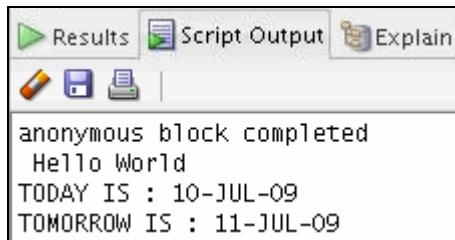


- c) 이 스크립트를 lab_09_01_soln.sql로 저장합니다.
- d) Clear 버튼을 눌러 작업 영역을 지웁니다.
- e) greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다.
(힌트: 블록 시작 부분에서 SERVEROUTPUT을 활성화해야 합니다.)

```
SET SERVEROUTPUT ON

BEGIN
    greet;
END;
```

출력 결과는 다음과 유사해야 합니다.



해답 9: 내장 프로시저 생성 및 사용 (계속)

2) lab_09_01_soln.sql 스크립트를 다음과 같이 수정합니다.

- a) 다음 명령을 실행하여 greet 프로시저를 삭제합니다.

```
DROP PROCEDURE greet;
```

- b) VARCHAR2 유형의 인수를 받아들이도록 프로시저를 수정합니다. p_name 인수를 호출합니다.

```
CREATE PROCEDURE greet(p_name VARCHAR2) IS
    V_today DATE:=SYSDATE;
    V_tomorrow today%TYPE;
```

- c) Hello World 대신 Hello <name>을 출력합니다.

```
BEGIN
    V_tomorrow:=v_today +1;
    DBMS_OUTPUT.PUT_LINE('Hello '|| p_name);
    ...

```

- d) 스크립트를 lab_09_02_soln.sql로 저장합니다.

- e) 스크립트를 실행하여 프로시저를 작성합니다. 출력 결과는 다음과 같아야 합니다.

```
DROP PROCEDURE greet succeeded.
PROCEDURE greet Compiled.
```

- f) 파라미터 값을 사용하여 greet 프로시저를 호출하는 익명 블록을 생성하고 실행합니다. 블록에서 출력도 생성해야 합니다.

```
SET SERVEROUTPUT ON;
BEGIN
    greet('Nancy');
END;
```

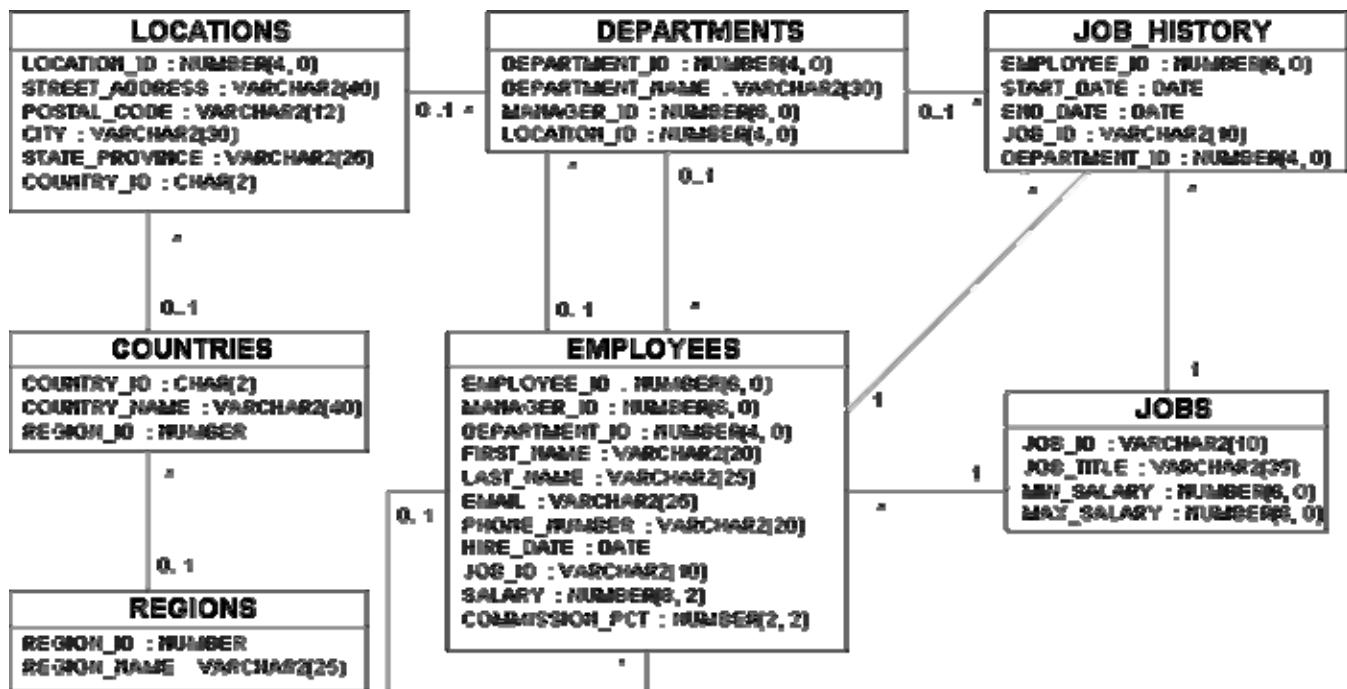
예제 출력 결과는 다음과 유사해야 합니다.

```
anonymous block completed
Hello Nancy
TODAY IS : 10-JUL-09
TOMORROW IS : 11-JUL-09
```

B

테이블 설명 및 데이터

엔티티 관계 도표



스키마의 테이블

```
SELECT * FROM tab;
```

TNAME	TABTYPE	CLUSTERID
COUNTRIES	TABLE	
DEPARTMENTS	TABLE	
EMPLOYEES	TABLE	
EMP_DETAILS_VIEW	VIEW	
JOB_HISTORY	TABLE	
JOBS	TABLE	
LOCATIONS	TABLE	
REGIONS	TABLE	

8 rows selected.

regions 테이블

```
DESCRIBE regions
```

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions;
```

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
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

SELECT * FROM countries;

CO	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4
FR	France	1
HK	HongKong	3
IL	Israel	4
IN	India	3
CO	COUNTRY_NAME	REGION_ID
IT	Italy	1
JP	Japan	3
KW	Kuwait	4
MX	Mexico	2
NG	Nigeria	4
NL	Netherlands	1
SG	Singapore	3
UK	United Kingdom	1
US	United States of America	2
ZM	Zambia	4
ZW	Zimbabwe	4

25 rows selected.

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)

SELECT * FROM locations;

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
1000	1297 Via Cola di Rie	00989	Roma		IT
1100	93091 Calle della Testa	10934	Venice		IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	6823	Hiroshima		JP
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	Y5W 9T2	Whitehorse	Yukon	CA
2000	40-5-12 Laogianggen	190518	Beijing		CN
2100	1298 Vileparle (E)	490231	Bombay	Maharashtra	IN
LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	CO
2400	8204 Arthur St		London		UK
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK
2600	9702 Chester Road	09629850293	Stretford	Manchester	UK
2700	Schwanthalerstr. 7031	80925	Munich	Bavaria	DE
2800	Rua Frei Caneca 1360	01307-002	Sao Paulo	Sao Paulo	BR
2900	20 Rue des Corps-Saints	1730	Geneva	Geneve	CH
3000	Murtenstrasse 921	3095	Bern	BE	CH
3100	Pieter Breughelstraat 837	3029SK	Utrecht	Utrecht	NL
3200	Mariano Escobedo 9991	11932	Mexico City	Distrito Federal,	MX

23 rows selected.

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)

SELECT * FROM departments;

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

27 rows selected.

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)

SELECT * FROM jobs;

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
FI_MGR	Finance Manager	8200	16000
FI_ACCOUNT	Accountant	4200	9000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
PU_MAN	Purchasing Manager	8000	15000
PU_CLERK	Purchasing Clerk	2500	5500
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
SH_CLERK	Shipping Clerk	2500	5500
JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000
HR_REP	Human Resources Representative	4000	9000
PR_REP	Public Relations Representative	4500	10500

19 rows selected.

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)

employees 테이블 (계속)

다음 스크린샷에서 commission_pct, manager_id 및 department_id 열의 머리글은 페이지에 테이블 값을 맞추기 위해 comm, mgrid 및 deptid로 설정되어 있습니다.

```
SELECT * FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-97	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-98	IT_PROG	4800		103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200		103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-94	FI_MGR	12000		101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-94	FI_ACCOUNT	9000		108	100
110	John	Chen	JCHEN	515.124.4269	28-SEP-97	FI_ACCOUNT	8200		108	100
111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-97	FI_ACCOUNT	7700		108	100
112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-98	FI_ACCOUNT	7800		108	100
113	Luis	Popp	LPOPP	515.124.4567	07-DEC-99	FI_ACCOUNT	6900		108	100
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-94	PU_MAN	11000		100	30
115	Alexander	Khoo	AKHOO	515.127.4562	18-MAY-95	PU_CLERK	3100		114	30
116	Shelli	Baida	SBAIDA	515.127.4563	24-DEC-97	PU_CLERK	2900		114	30
117	Sigal	Tobias	STOBIAS	515.127.4564	24-JUL-97	PU_CLERK	2800		114	30
118	Guy	Himuro	GHIMURO	515.127.4565	15-NOV-98	PU_CLERK	2600		114	30
119	Karen	Colmenares	KCOLMENA	515.127.4566	10-AUG-99	PU_CLERK	2500		114	30
120	Matthew	Weiss	MWEISS	650.123.1234	18-JUL-96	ST_MAN	8000		100	50
121	Adam	Fripp	AFRIPP	650.123.2234	10-APR-97	ST_MAN	8200		100	50
122	Payam	Kaufling	PKAUFLIN	650.123.3234	01-MAY-95	ST_MAN	7900		100	50
123	Shanta	Vollman	SVOLLMAN	650.123.4234	10-OCT-97	ST_MAN	6500		100	50
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800		100	50
125	Julia	Nayer	JNAYER	650.124.1214	16-JUL-97	ST_CLERK	3200		120	50
126	Irene	Mikkilineni	IMIKKILILI	650.124.1224	28-SEP-98	ST_CLERK	2700		120	50
127	James	Landry	JLANDRY	650.124.1334	14-JAN-99	ST_CLERK	2400		120	50

employees 테이블 (계속)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
128	Steven	Markle	SMARKLE	650.124.1434	08-MAR-00	ST_CLERK	2200		120	50
129	Laura	Bissot	LBISSOT	650.124.5234	20-AUG-97	ST_CLERK	3300		121	50
130	Mozhe	Atkinson	MATKINSO	650.124.8234	30-OCT-97	ST_CLERK	2800		121	50
131	James	Marlow	JAMRLOW	650.124.7234	16-FEB-97	ST_CLERK	2500		121	50
132	T.J.	Olson	TJOLSON	650.124.8234	10-APR-99	ST_CLERK	2100		121	50
133	Jason	Mallin	JMALLIN	650.127.1934	14-JUN-96	ST_CLERK	3300		122	50
134	Michael	Rogers	MROGERS	650.127.1834	28-AUG-98	ST_CLERK	2900		122	50
135	Ki	Gee	KGEE	650.127.1734	12-DEC-99	ST_CLERK	2400		122	50
136	Hazel	Philtanker	PHILHTAN	650.127.1634	06-FEB-00	ST_CLERK	2200		122	50
137	Renske	Ladwig	RLADWIG	650.121.1234	14-JUL-95	ST_CLERK	3600		123	50
138	Stephen	Stiles	SSTILES	650.121.2034	28-OCT-97	ST_CLERK	3200		123	50
139	John	Seo	JSEO	650.121.2019	12-FEB-98	ST_CLERK	2700		123	50
140	Joshua	Patel	JPATEL	650.121.1834	06-APR-98	ST_CLERK	2500		123	50
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3600		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
142	Curtis	Davies	CDAMES	650.121.2994	29-JAN-97	ST_CLERK	3100		124	50
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600		124	50
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500		124	50
145	John	Russell	JRUSSEL	011.44.1344.429268	01-OCT-96	SA_MAN	14000	.4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05-JAN-97	SA_MAN	13500	.3	100	80
147	Alberto	Etrazuriz	AERRAZUR	011.44.1344.429278	10-MAR-97	SA_MAN	12000	.3	100	80
148	Gerald	Cambrault	GCAMBRAU	011.44.1344.619268	15-OCT-99	SA_MAN	11000	.3	100	80
149	Beni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	.2	100	80
150	Peter	Tucker	PTUCKER	011.44.1344.129268	30-JAN-97	SA REP	10000	.3	145	80
151	David	Bernstein	DBERNSTE	011.44.1344.345268	24-MAR-97	SA REP	9500	.25	145	80
152	Peter	Hall	PHALL	011.44.1344.478968	20-AUG-97	SA REP	9000	.25	145	80
153	Christopher	Olsen	COLSEN	011.44.1344.498718	30-MAR-98	SA REP	8000	.2	145	80
154	Nanette	Cambrault	NCAMBRAU	011.44.1344.987668	09-DEC-98	SA REP	7500	.2	145	80
155	Oliver	Tuvault	OTUVVAULT	011.44.1344.486508	23-NOV-99	SA REP	7000	.15	145	80
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
156	Janette	King	JKING	011.44.1345.429268	30-JAN-96	SA REP	10000	.35	146	80
157	Patrick	Sully	PSULLY	011.44.1345.929268	04-MAR-96	SA REP	9500	.35	146	80
158	Allan	McEwen	AMCEWEN	011.44.1345.829268	01-AUG-96	SA REP	9000	.35	146	80
159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-97	SA REP	8000	.3	146	80
160	Louise	Doran	LDORAN	011.44.1345.629268	15-DEC-97	SA REP	7500	.3	146	80
161	Sarath	Sewall	SSEWALL	011.44.1345.529268	03-NOV-98	SA REP	7000	.25	146	80
162	Clara	Wshney	CWISHNEY	011.44.1346.129268	11-NOV-97	SA REP	10500	.25	147	80
163	Danielle	Greene	DGREENE	011.44.1346.229268	19-MAR-99	SA REP	9500	.15	147	80
164	Mattea	Marvins	MMARVINS	011.44.1346.329268	24-JAN-00	SA REP	7200	.1	147	80
165	David	Lee	DLEE	011.44.1346.529268	23-FEB-00	SA REP	6800	.1	147	80
166	Sundar	Ande	SANDE	011.44.1346.629268	24-MAR-00	SA REP	6400	.1	147	80
167	Amit	Banda	ABANDA	011.44.1346.729268	21-APR-00	SA REP	6200	.1	147	80
168	Lisa	Ozer	LOZER	011.44.1343.929268	11-MAR-97	SA REP	11500	.25	148	80
169	Hamison	Bloom	HBLOOM	011.44.1343.829268	23-MAR-98	SA REP	10000	.2	148	80

employees 테이블 (계속)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
170	Taylor	Fox	TFOX	011.44.1343.729268	24-JAN-98	SA_REP	9600	.2	148	80
171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-99	SA_REP	7400	.15	148	80
172	Elizabeth	Bates	EBATES	011.44.1343.529268	24-MAR-99	SA_REP	7300	.15	148	80
173	Sundita	Kumar	SKUMAR	011.44.1343.329268	21-APR-00	SA_REP	6100	.1	148	80
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	.3	149	80
175	Alyssa	Hutton	AHUTTON	011.44.1644.429266	19-MAR-97	SA_REP	8800	.25	149	80
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	.2	149	80
177	Jack	Livingston	JLIVINGSTON	011.44.1644.429264	23-APR-98	SA_REP	8400	.2	149	80
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	.15	149	
179	Charles	Johnson	CJOHNSON	011.44.1644.429262	04-JAN-00	SA_REP	6200	.1	149	80
180	Winston	Taylor	WTAYLOR	650.507.9876	24-JAN-98	SH_CLERK	3200		120	50
181	Jean	Fleaur	JFLEAUR	650.507.9877	23-FEB-98	SH_CLERK	3100		120	50
182	Martha	Sullivan	MSULLIVA	650.507.9878	21-JUN-99	SH_CLERK	2500		120	50
183	Girard	Geoni	GGEOANI	650.507.9879	03-FEB-00	SH_CLERK	2800		120	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
184	Nandita	Sarchand	NSARCHAN	650.509.1876	27-JAN-96	SH_CLERK	4200		121	50
185	Alexis	Bull	ABULL	650.509.2876	20-FEB-97	SH_CLERK	4100		121	50
186	Julia	Dellinger	JDELLING	650.509.3876	24-JUN-98	SH_CLERK	3400		121	50
187	Anthony	Cabrio	ACABRIO	650.509.4876	07-FEB-99	SH_CLERK	3000		121	50
188	Kelly	Chung	KCHUNG	650.505.1876	14-JUN-97	SH_CLERK	3800		122	50
189	Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-97	SH_CLERK	3600		122	50
190	Timothy	Gates	TGATES	650.505.3876	11-JUL-98	SH_CLERK	2900		122	50
191	Randall	Perkins	RPERKINS	650.505.4876	19-DEC-99	SH_CLERK	2500		122	50
192	Sarah	Bell	SBELL	650.501.1876	04-FEB-96	SH_CLERK	4000		123	50
193	Britney	Everett	BEVERETT	650.501.2876	03-MAR-97	SH_CLERK	3900		123	50
194	Samuel	McCain	SMCCAIN	650.501.3876	01-JUL-98	SH_CLERK	3200		123	50
195	Vance	Jones	VJONES	650.501.4876	17-MAR-99	SH_CLERK	2800		123	50
196	Alana	Walsh	AWALSH	650.507.9811	24-APR-98	SH_CLERK	3100		124	50
197	Kevin	Feeney	KFEENEY	650.507.9822	23-MAY-98	SH_CLERK	3000		124	50
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	comm	mgrid	deptid
198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-99	SH_CLERK	2800		124	50
199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-00	SH_CLERK	2800		124	50
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400		101	10
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000		100	20
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK REP	6000		201	20
203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-94	HR REP	6500		101	40
204	Hermann	Baer	HBAER	515.123.8888	07-JUN-94	PR REP	10000		101	70
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000		101	110
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300		205	110

107 rows selected.

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)

```
SELECT * FROM job_history;
```

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	deptid
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.



SQL Developer 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- Oracle SQL Developer의 주요 기능 숙지
- Oracle SQL Developer의 메뉴 항목 식별
- 데이터베이스 연결 생성
- 데이터베이스 객체 관리
- SQL Worksheet 사용
- SQL 스크립트 저장 및 실행
- 보고서 생성 및 저장

ORACLE®

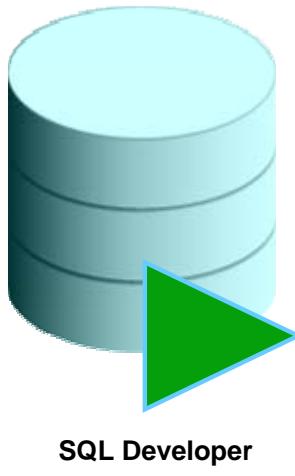
Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 SQL Developer라는 그래픽 도구에 대해 알아봅니다. 먼저, 데이터베이스 개발 작업에 SQL Developer를 사용하는 방법에 대해 알아봅니다. SQL Worksheet를 사용하여 SQL 문과 SQL 스크립트를 실행하는 방법에 대해 설명합니다.

Oracle SQL Developer란?

- Oracle SQL Developer는 생산성을 향상하고 데이터베이스 개발 작업을 단순화하는 그래픽 도구입니다.
- 표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다.



SQL Developer

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle SQL Developer란?

Oracle SQL Developer는 생산성을 향상하고 일상적인 데이터베이스 작업의 개발을 단순화하기 위해 설계된 무료 그래픽 도구입니다. 마우스를 몇 번 누르는 것만으로 간단하게 내장 프로시저를 생성 및 디버그하고, SQL 문을 테스트하고, 옵티マイ저 계획을 볼 수 있습니다.

데이터베이스 개발을 위한 시각적 도구인 SQL Developer는 다음 작업을 단순화합니다.

- 데이터베이스 객체 탐색 및 관리
- SQL 문 및 스크립트 실행
- PL/SQL 문 편집 및 디버깅
- 보고서 작성

표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다. 연결되면 데이터베이스의 객체에 대해 작업을 수행할 수 있습니다.

SQL Developer 1.2 버전은 유저가 단일 위치에서 third-party 데이터베이스의 데이터 및 데이터베이스 객체를 탐색하고 오라클에서 이러한 데이터베이스를 이전할 수 있도록 하는 *Developer Migration Workbench*와 긴밀하게 통합됩니다. 또한 MySQL, Microsoft SQL Server 및 Microsoft Access를 비롯한 일부 third-party 데이터베이스의 스키마에 연결할 수 있으며 이러한 데이터베이스의 메타 데이터와 데이터를 볼 수도 있습니다.

뿐만 아니라 SQL Developer에서는 Oracle Application Express 3.0.1(Oracle APEX)도 지원합니다.

SQL Developer 사양

- Oracle Database 11g Release 2와 함께 제공
- Java로 개발
- Windows, Linux 및 Mac OS X 플랫폼 지원
- JDBC Thin 드라이버를 사용하여 기본 연결 제공
- Oracle Database 9.2.0.1 이상 버전에 연결
- 다음 링크에서 무료로 다운로드할 수 있습니다.
 - http://www.oracle.com/technology/products/database/sql_developer/index.html

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 사양

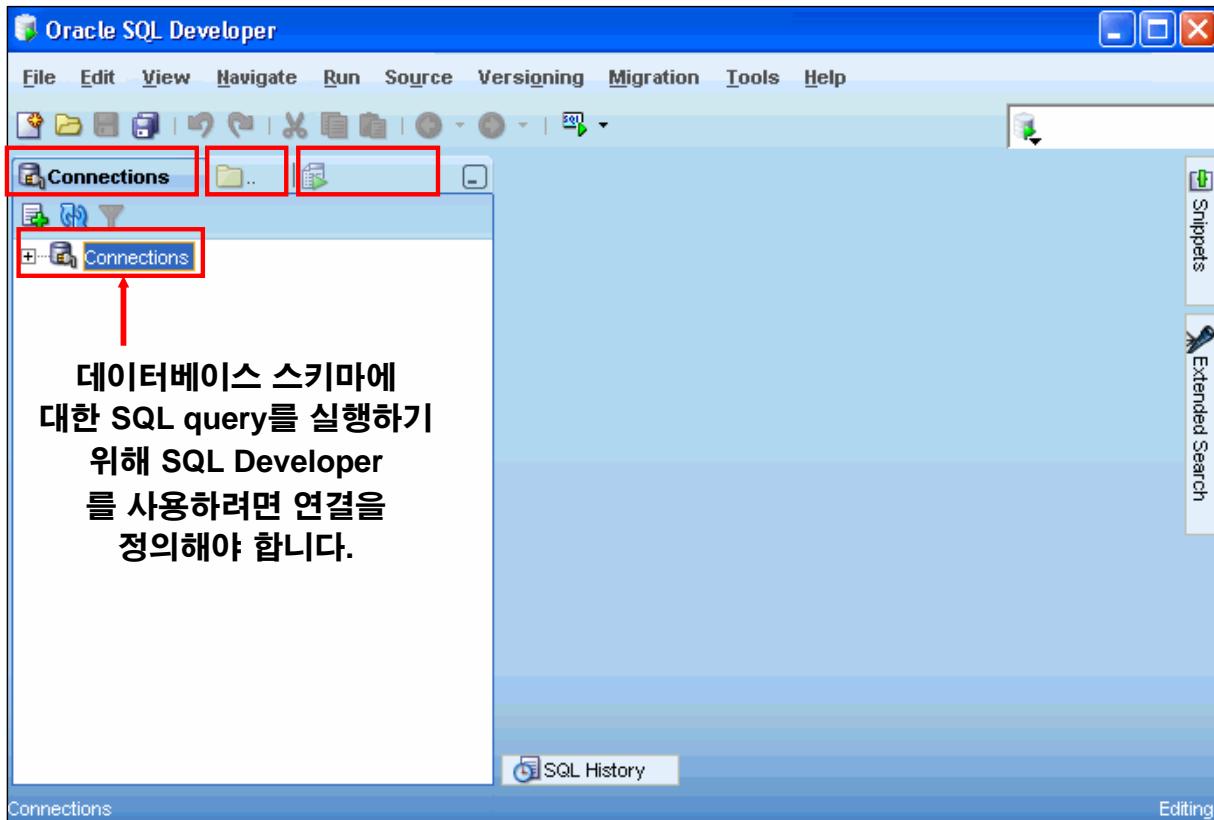
Oracle SQL Developer 1.5는 Oracle Database 11g Release 2와 함께 제공됩니다. SQL Developer는 Java로 개발되며 Oracle JDeveloper 통합 개발 환경(IDE)을 활용합니다. 교차 플랫폼 도구로, Windows, Linux 및 Mac OS(운영 체제) X 플랫폼에서 실행됩니다.

또한 JDBC(Java Database Connectivity) Thin 드라이버를 통해 데이터베이스에 대한 기본 연결이 제공되므로 Oracle Home이 필요하지 않습니다. SQL Developer는 설치 프로그램이 필요하지 않습니다. 다운로드한 파일의 압축을 풀기만 하면 됩니다. SQL Developer 유저는 Oracle Databases 9.2.0.1 이상 버전 및 Express Edition을 포함한 모든 Oracle Database Edition에 연결할 수 있습니다.

참고

- Oracle Database 11g Release 2 이전의 Oracle Database 버전을 사용하는 경우 SQL Developer를 다운로드하여 설치해야 합니다. SQL Developer 1.5는 다음 링크에서 무료로 다운로드할 수 있습니다.
 - http://www.oracle.com/technology/products/database/sql_developer/index.html
- SQL Developer를 설치하는 방법을 보려면 다음 링크를 방문하십시오.
 - http://download.oracle.com/docs/cd/E12151_01/index.htm

SQL Developer 1.5 인터페이스



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 1.5 인터페이스

SQL Developer 1.5에는 다음과 같은 세 개의 기본 탐색 템(왼쪽부터)이 있습니다.

- **Connections 템:** 이 템을 사용하면 액세스할 수 있는 데이터베이스 객체 및 유저를 찾아볼 수 있습니다.
- **Files 템:** Files 폴더 아이콘으로 식별됩니다. 이 템을 사용하면 File > Open 메뉴를 사용하지 않고도 로컬 시스템에서 파일에 액세스할 수 있습니다.
- **Reports 템:** Reports 아이콘으로 식별됩니다. 이 템을 사용하면 미리 정의된 보고서를 실행하거나 사용자 보고서를 작성하고 추가할 수 있습니다.

일반 탐색 및 사용

SQL Developer는 왼쪽 탐색 영역에서 객체를 찾아 선택하면 오른쪽 탐색 영역에 선택한 객체에 대한 정보가 표시됩니다. 환경 설정을 통해 SQL Developer의 다양한 모양과 동작을 커스터마이즈할 수 있습니다.

참고: 데이터베이스 스키마에 연결하여 SQL Query를 실행하거나 프로시저/함수를 실행하려면 연결을 하나 이상 정의해야 합니다.

SQL Developer 1.5 인터페이스 (계속)

메뉴

다음 메뉴에는 표준 항목과 SQL Developer 특정 기능에 대한 항목이 있습니다.

- **View:** SQL Developer 인터페이스에 표시되는 사항에 영향을 주는 옵션이 있습니다.
- **Navigate:** 창 이동 및 서브 프로그램 실행을 위한 옵션이 있습니다.
- **Run:** 함수나 프로시저가 선택될 때 연관되는 Run File 및 Execution Profile 옵션과 디버깅 옵션이 있습니다.
- **Source:** 함수와 프로시저를 편집할 때 사용하는 옵션이 있습니다.
- **Versioning:** CVS(Concurrent Versions System) 및 Subversion과 같은 버전 관리 및 소스 제어 시스템을 통합 지원합니다.
- **Migration:** Third-party 데이터베이스를 오라클로 이전할 때 관련되는 옵션이 있습니다.
- **Tools:** SQL*Plus, Preferences 및 SQL Worksheet 등의 SQL Developer 도구를 호출합니다.

참고: Run 메뉴에는 디버깅용으로 함수나 프로시저가 선택될 때 연관되는 옵션이 있습니다.

이러한 옵션은 1.2 버전의 Debug 메뉴에 있는 옵션과 동일합니다.

데이터베이스 연결 생성

- SQL Developer를 사용하려면 하나 이상의 데이터베이스 연결이 있어야 합니다.
- 다음 대상에 대해 연결을 생성하고 테스트할 수 있습니다.
 - 하나 이상의 데이터베이스
 - 하나 이상의 스키마
- SQL Developer는 시스템의 `tnsnames.ora` 파일에 정의된 모든 연결을 자동으로 임포트합니다.
- XML(Extensible Markup Language) 파일로 연결을 эксп포트할 수 있습니다.
- 추가로 생성된 각각의 데이터베이스 연결은 Connections Navigator 계층에 나열됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 연결 생성

연결은 특정 데이터베이스에 해당 데이터베이스의 특정 유저로 연결하는 데 필요한 정보를 지정하는 SQL Developer 객체입니다. SQL Developer를 사용하려면 하나 이상의 데이터베이스 연결이 있어야 합니다. 기존 연결을 사용하거나 연결을 새로 생성 또는 임포트할 수도 있습니다. 여러 데이터베이스 및 여러 스키마에 대해 연결을 생성하고 테스트할 수 있습니다.

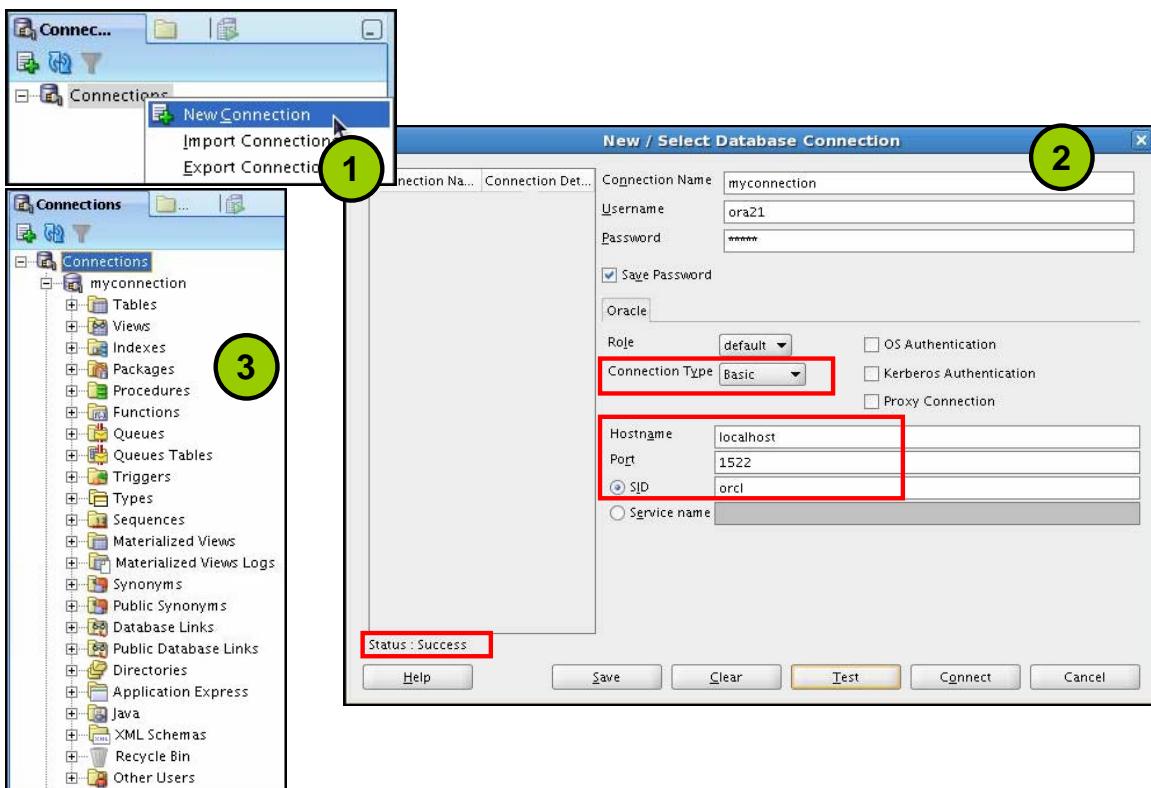
기본적으로 `tnsnames.ora` 파일은 `$ORACLE_HOME/network/admin` 디렉토리에 있지만, `TNS_ADMIN` 환경 변수 또는 레지스트리 값에 지정된 디렉토리에 있을 수도 있습니다. SQL Developer를 시작하고 Database Connections 대화상자를 표시하면 SQL Developer는 시스템의 `tnsnames.ora` 파일에 정의된 모든 연결을 자동으로 임포트합니다.

참고: Windows에서 `tnsnames.ora` 파일이 존재하지만 SQL Developer가 해당 연결을 사용하고 있지 않으면 `TNS_ADMIN`을 시스템 환경 변수로 정의하십시오.

연결을 XML 파일로 эксп포트하여 나중에 재사용할 수 있습니다.

동일한 데이터베이스에 다른 유저로 연결하거나 다른 데이터베이스에 연결하도록 추가 연결을 생성할 수 있습니다.

데이터베이스 연결 생성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 연결 생성 (계속)

데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. Connections 탭 페이지에서 **Connections**를 마우스 오른쪽 버튼으로 누르고 **New Connection**을 선택합니다.
2. New>Select Database Connection window에 연결 이름을 입력합니다. 연결하려는 스키마의 username 및 암호를 입력합니다.
 - a) Role drop-down box에서 *default* 또는 SYSDBA를 선택할 수 있습니다. sys 유저 또는 데이터베이스 관리자 권한이 있는 모든 유저에 대해 SYSDBA를 선택하면 됩니다.
 - b) 연결 유형은 다음 중에서 선택하면 됩니다.
 - **Basic:** 이 유형의 경우 연결하려는 데이터베이스의 호스트 이름 및 SID를 입력합니다. 포트는 이미 1521로 설정되어 있습니다. 원격 데이터베이스 연결을 사용하는 경우에는 서비스 이름을 직접 입력할 수도 있습니다.
 - **TNS:** tnsnames.ora 파일에서 임포트한 데이터베이스 alias 중 하나를 선택할 수 있습니다.
 - **LDAP:** Oracle Identity Management의 구성 요소인 Oracle Internet Directory에서 데이터베이스 서비스를 조회할 수 있습니다.
 - **Advanced:** 데이터베이스에 연결할 커스텀 JDBC(Java Database Connectivity) URL을 정의할 수 있습니다.

데이터베이스 연결 생성 (계속)

- c) **Test**를 눌러 연결이 올바르게 설정되었는지 확인합니다.
- d) **Connect**를 누릅니다.

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

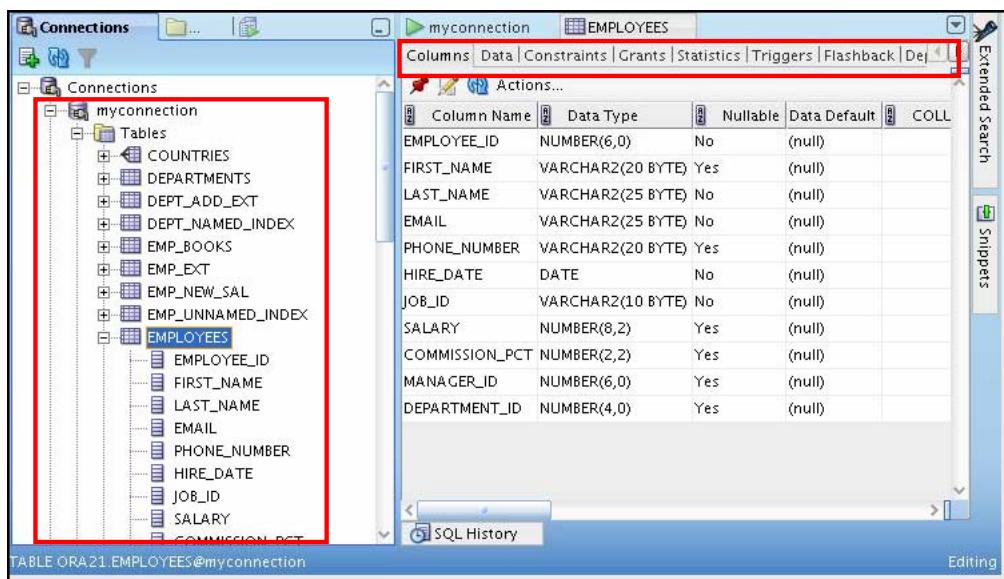
3. Connections Navigator에 연결이 추가됩니다. 연결을 확장하여 데이터베이스 객체를 볼 수 있으며 종속성, 상세 내역, 통계 등의 객체 정의를 볼 수 있습니다.

참고: 같은 New>Select Database Connection window에서 Access, MySQL 및 SQL Server 탭을 사용하여 third-party 데이터 소스에 대한 연결을 정의할 수 있습니다. 그러나 이러한 연결은 읽기 전용 연결이며 해당 데이터 소스에서 객체 및 데이터를 찾아볼 수만 있습니다.

데이터베이스 객체 탐색

Connections Navigator를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스 스키마에서 여러 객체 탐색
- 여러 객체의 정의를 한 번에 검토



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 객체 탐색

데이터베이스 연결을 생성한 후 Connections Navigator를 사용하여 데이터베이스 스키마에서 테이블, 뷰, 인덱스, 패키지, 프로시저, 트리거, 유형 등의 여러 객체를 탐색할 수 있습니다.

SQL Developer는 왼쪽 탐색 영역에서 객체를 찾아 선택하면 오른쪽 탐색 영역에 선택한 객체에 대한 정보가 표시됩니다. 환경 설정을 통해 SQL Developer의 다양한 모양을 커스터마이즈할 수 있습니다.

데이터 딕셔너리에서 추출된 객체 정의가 여러 정보 템에 분할되어 있는 것을 볼 수 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 읽기 쉬운 템 페이지에 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 세부 정보가 표시됩니다.

슬라이드에서처럼 EMPLOYEES 테이블의 정의를 보려면 다음 단계를 수행하십시오.

1. Connections Navigator에서 Connections 노드를 확장합니다.
2. Tables를 확장합니다.
3. EMPLOYEES를 누릅니다. 기본적으로 Columns 탭이 선택되며, 이 탭에는 해당 테이블의 열 설명이 표시됩니다. Data 탭을 통해 테이블 데이터를 볼 수 있으며 새 행을 입력하고, 데이터를 갱신하고, 해당 변경 내용을 데이터베이스로 커밋할 수 있습니다.

테이블 구조 표시

DESCRIBE 명령을 사용하여 테이블의 구조를 표시합니다.

The screenshot shows the SQL Developer interface with the query 'DESC EMPLOYEES' run against a connection named 'myconnection'. The results tab displays the structure of the 'EMPLOYEES' table:

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

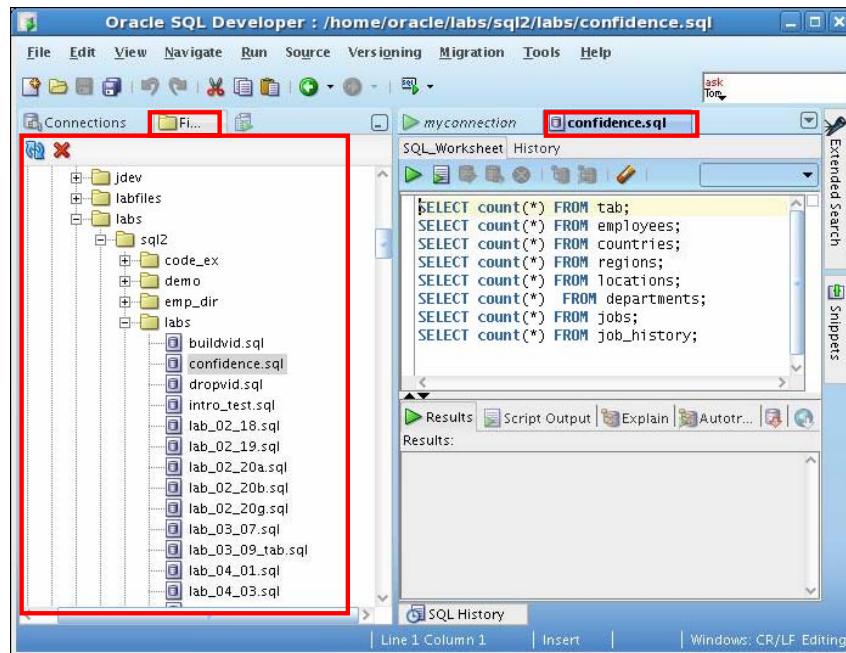
테이블 구조 표시

SQL Developer에서 DESCRIBE 명령을 사용하여 테이블 구조를 표시할 수도 있습니다.

이 명령의 결과로 열 이름 및 데이터 유형이 표시되고 열에 반드시 데이터가 포함되어야 하는지 여부가 나타납니다.

파일 탐색

Files Navigator를 사용하여 파일 시스템을 탐색하고 시스템 파일을 열 수 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

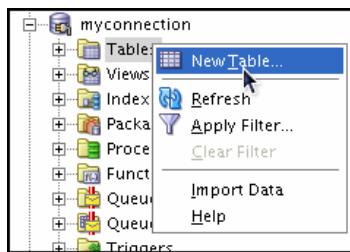
데이터베이스 객체 탐색

Files Navigator를 사용하여 시스템 파일을 찾아서 열 수 있습니다.

- Files Navigator를 보려면 Files 탭을 누르거나 View > Files를 선택합니다.
- 파일 내용을 보려면 파일 이름을 두 번 눌러 SQL Worksheet 영역에 파일 내용을 표시합니다.

스키마 객체 생성

- SQL Developer는 다음 방법을 통한 스키마 객체 생성을 지원합니다.
 - SQL Worksheet에서 SQL 문 실행
 - 컨텍스트 메뉴 사용
- 편집 대화상자나 문맥에 따른 여러 가지 메뉴 중 하나를 사용하여 객체를 편집합니다.
- 새 객체 생성 또는 기존 스키마 객체 편집과 같은 조정을 위해 DDL(데이터 정의어)을 봅니다.



ORACLE

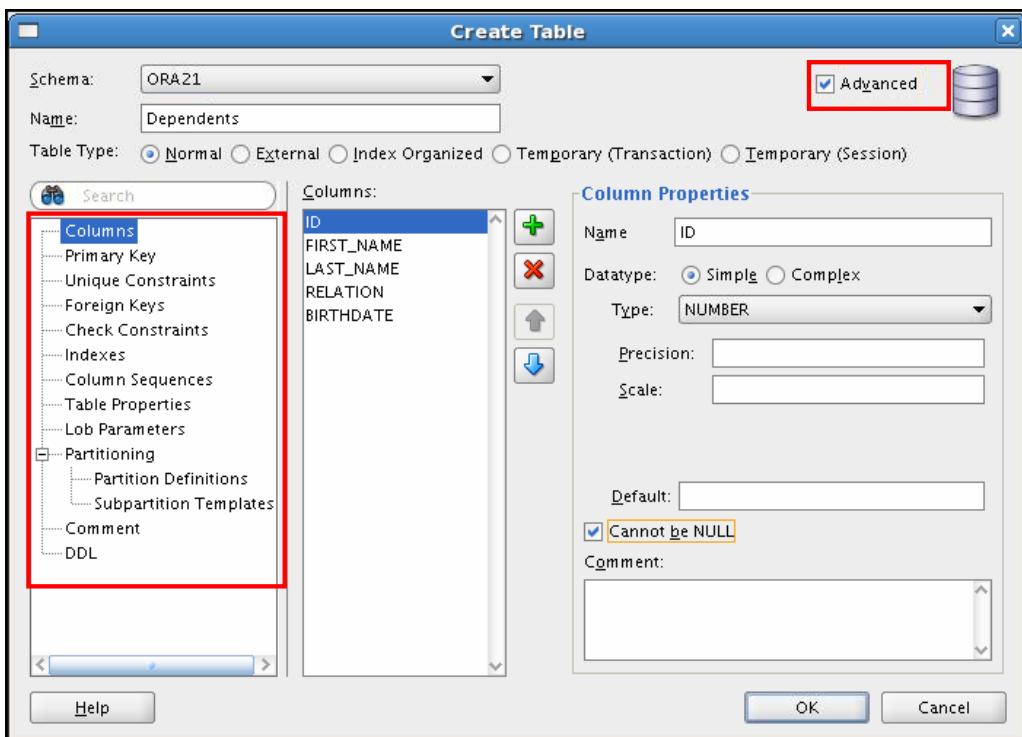
Copyright © 2009, Oracle. All rights reserved.

스키마 객체 생성

SQL Developer는 SQL Worksheet에서 SQL 문을 실행하여 스키마 객체 생성을 지원합니다. 또는 컨텍스트 메뉴를 사용하여 객체를 생성할 수도 있습니다. 객체를 생성한 후 편집 대화상자나 문맥에 따른 여러 메뉴 중 하나를 사용하여 객체를 편집할 수 있습니다. 새 객체를 생성하거나 기존 객체를 편집할 때 이러한 조정 작업을 위한 DDL을 확인할 수 있습니다. 스키마의 여러 객체에 대한 전체 DDL을 생성하려는 경우 Export DDL 옵션을 사용할 수 있습니다.

슬라이드에서는 컨텍스트 메뉴를 사용하여 테이블을 생성하는 방법을 보여줍니다. 새 테이블 생성 대화상자를 열려면 Tables를 마우스 오른쪽 버튼으로 누르고 New Table을 선택합니다. 데이터베이스 객체를 생성하고 편집하는 대화상자에는 여러 탭이 있으며 각 탭은 해당 객체 유형의 논리적 속성 그룹화를 반영합니다.

새 테이블 생성: 예제



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

새 테이블 생성: 예제

Create Table 대화상자에서 Advanced 체크 박스를 선택하지 않는 경우 열과 자주 사용하는 몇 가지 기능을 지정하여 테이블을 신속하게 생성할 수 있습니다.

Advanced 체크 박스를 선택한 경우에는 Create Table 대화상자에 여러 옵션이 표시되며, 여기에서 테이블을 생성하는 동안 확장된 기능을 지정할 수 있습니다.

슬라이드의 예제는 Advanced 체크 박스를 선택하여 DEPENDENTS 테이블을 생성하는 방법을 보여줍니다.

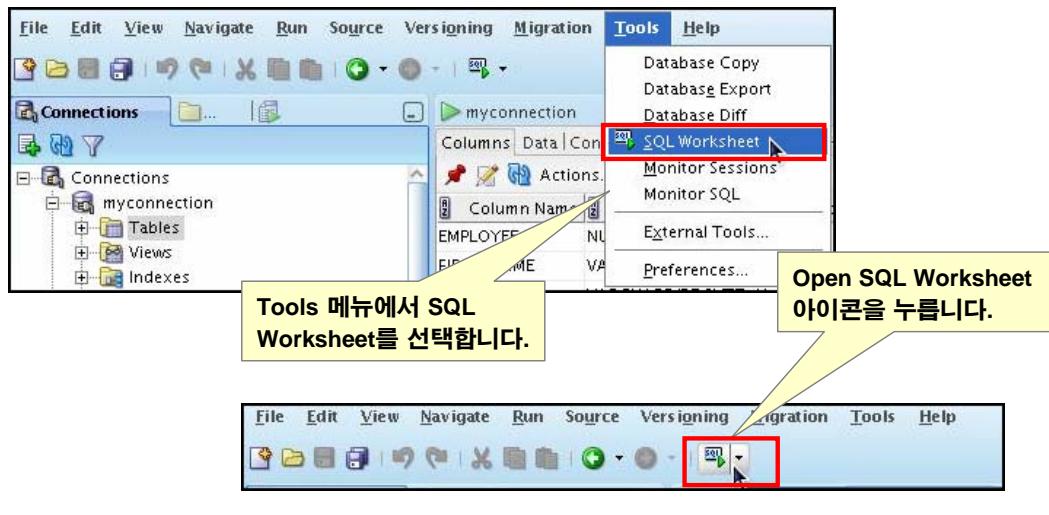
새 테이블을 생성하려면 다음 단계를 수행하십시오.

1. Connections Navigator에서 Tables를 마우스 오른쪽 버튼으로 누릅니다.
2. New Table을 선택합니다.
3. Create Table 대화상자에서 Advanced를 선택합니다.
4. 열 정보를 지정합니다.
5. OK를 누릅니다.

또한 필수 사항은 아니지만 대화상자의 Primary Key 탭을 사용하여 Primary Key를 지정하는 것이 좋습니다. 생성한 테이블을 편집해야 하는 경우도 있습니다. 테이블을 편집하려면 Connections Navigator에서 테이블을 마우스 오른쪽 버튼으로 누르고 Edit를 선택합니다.

SQL Worksheet 사용

- SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행합니다.
- 워크시트와 연관된 데이터베이스 연결로 처리할 수 있는 모든 작업을 지정합니다.



SQL Worksheet 사용

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet을 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. SQL Worksheet은 특정 범위까지 SQL*Plus 문을 지원합니다. 그러나 SQL Worksheet에서 지원되지 않는 SQL*Plus 문은 무시되고 데이터베이스에 전달되지 않습니다.

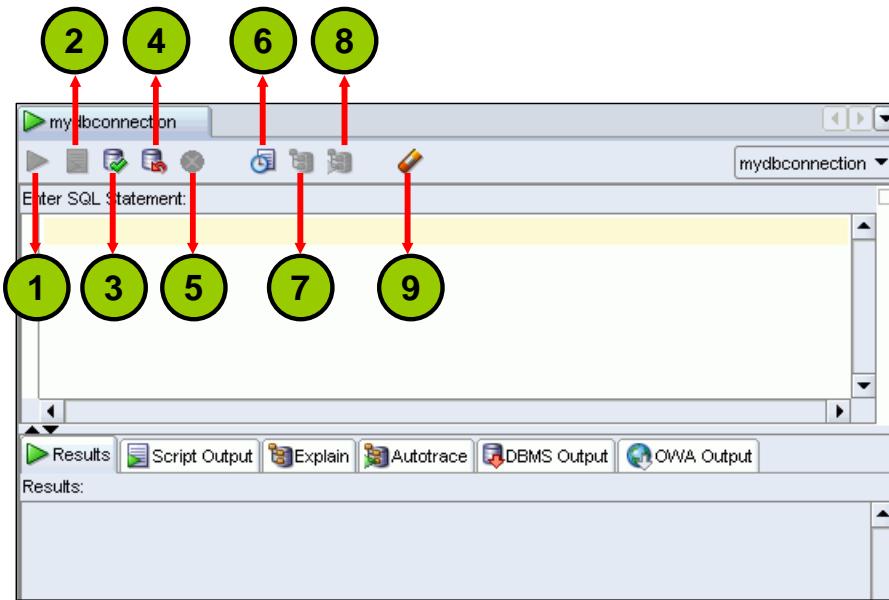
워크시트와 연관된 데이터베이스 연결에 의해 처리될 수 있는 다음과 같은 작업을 지정할 수 있습니다.

- 테이블 생성
- 데이터 삽입
- 트리거 생성 및 편집
- 테이블에서 데이터 선택
- 파일에 선택한 데이터 저장

다음 중 하나를 사용하여 SQL Worksheet을 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- Open SQL Worksheet 아이콘을 누릅니다.

SQL Worksheet 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

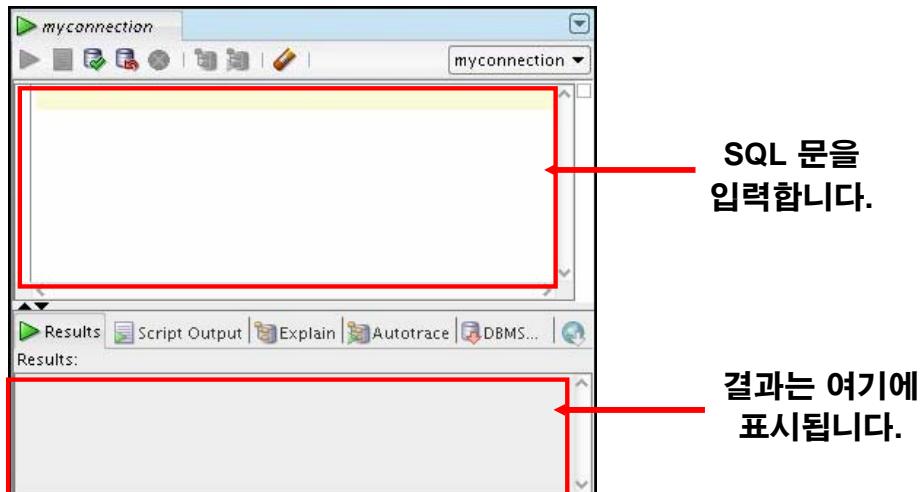
SQL Worksheet 사용 (계속)

단축키 또는 아이콘을 사용하여 SQL 문 실행, 스크립트 실행, 실행한 SQL 문 기록 보기 등의 특정 작업을 수행하는 경우가 있습니다. 여러 아이콘이 있는 SQL Worksheet 도구 모음을 사용하여 다음 작업을 수행할 수 있습니다.

1. **Execute Statement:** Enter SQL Statement window에서 커서가 위치한 명령문을 실행합니다. SQL 문에 바인드 변수를 사용할 수 있지만 치환 변수는 사용할 수 없습니다.
2. **Run Script:** Script Runner를 사용하여 Enter SQL Statement window에 있는 모든 명령문을 실행합니다. SQL 문에 치환 변수를 사용할 수 있지만 바인드 변수는 사용할 수 없습니다.
3. **Commit:** 데이터베이스에 대한 모든 변경 사항을 기록하고 트랜잭션을 종료합니다.
4. **Rollback:** 데이터베이스에 대한 모든 변경 사항을 데이터베이스에 기록하지 않은 채 폐기하고 트랜잭션을 종료합니다.
5. **Cancel:** 현재 실행 중인 모든 명령문의 실행을 정지합니다.
6. **SQL History:** 실행한 SQL 문에 대한 정보가 있는 대화상자를 표시합니다.
7. **Execute Explain Plan:** Explain 탭을 눌러 볼 수 있는 실행 계획을 생성합니다.
8. **Autotrace:** 명령문에 대한 추적 정보를 생성합니다.
9. **Clear:** Enter SQL Statement window에서 명령문을 지웁니다.

SQL Worksheet 사용

- SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행합니다.
- 워크시트와 연관된 데이터베이스 연결로 처리할 수 있는 모든 작업을 지정합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Worksheet 사용 (계속)

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. 모든 SQL 및 PL/SQL 명령이 지원되며 이러한 명령은 SQL Worksheet에서 오라클 데이터베이스로 직접 전달됩니다. 그러나 SQL Developer에서 사용되는 SQL*Plus 명령은 데이터베이스로 전달하기 전에 SQL Worksheet에서 해석되어야 합니다.

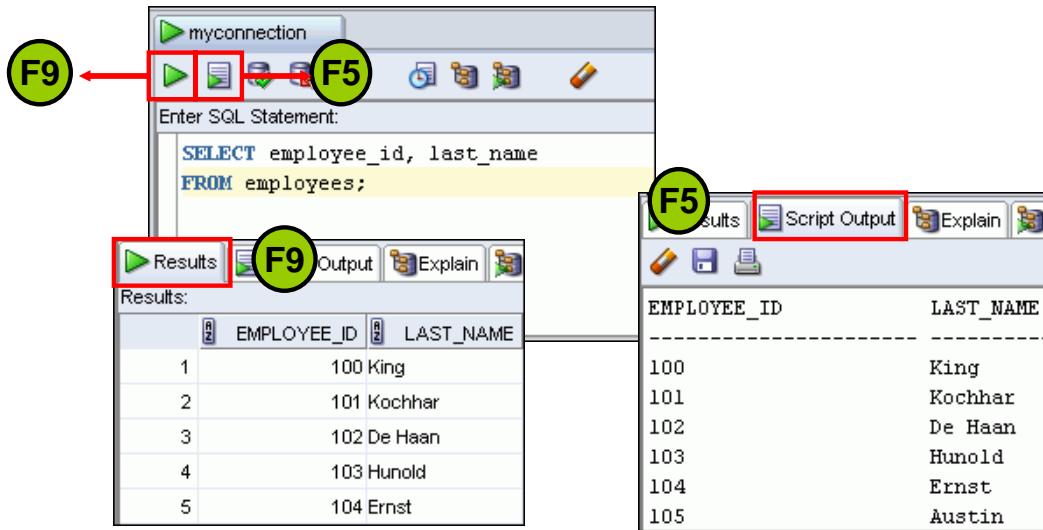
현재 SQL Worksheet에서는 다수의 SQL*Plus 명령을 지원합니다. 그러나 SQL Worksheet에서 지원하지 않는 명령은 무시되며 오라클 데이터베이스로 전달되지 않습니다. SQL Worksheet를 통해 SQL 문을 실행하거나 SQL*Plus 명령 중 일부를 실행할 수 있습니다.

다음 두 가지 옵션 중 하나를 사용하여 SQL Worksheet을 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- Open SQL Worksheet 아이콘을 누릅니다.

SQL 문 실행

Enter SQL Statement window를 사용하여 SQL 문을 하나 또는 여러 개 입력합니다.



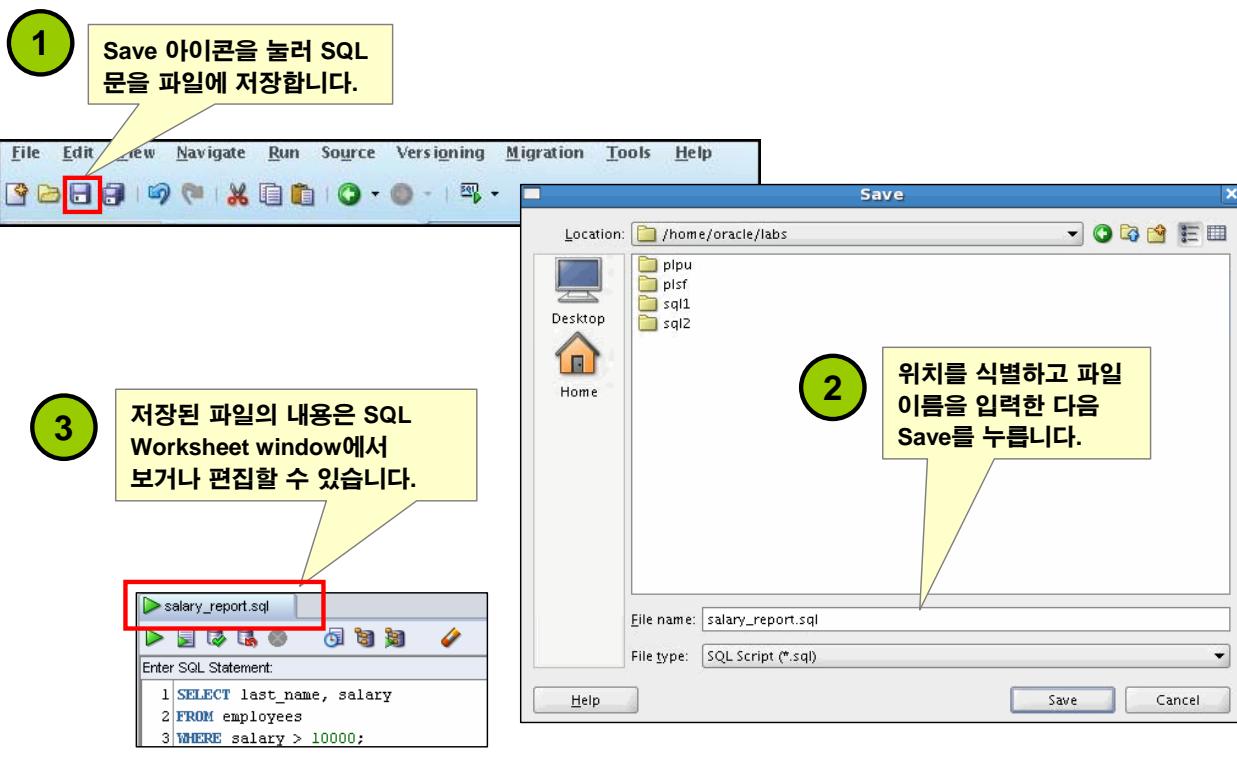
ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문 실행

슬라이드의 예제에서는 동일한 query에 대해 F9 키 또는 Execute Statement 를 사용한 출력과 F5 키 또는 Run Script를 사용한 출력의 차이를 보여줍니다.

SQL 스크립트 저장



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 스크립트 저장

SQL 문을 SQL Worksheet에서 텍스트 파일로 저장할 수 있습니다. Enter SQL Statement window의 내용을 저장하려면 다음 단계를 따르십시오.

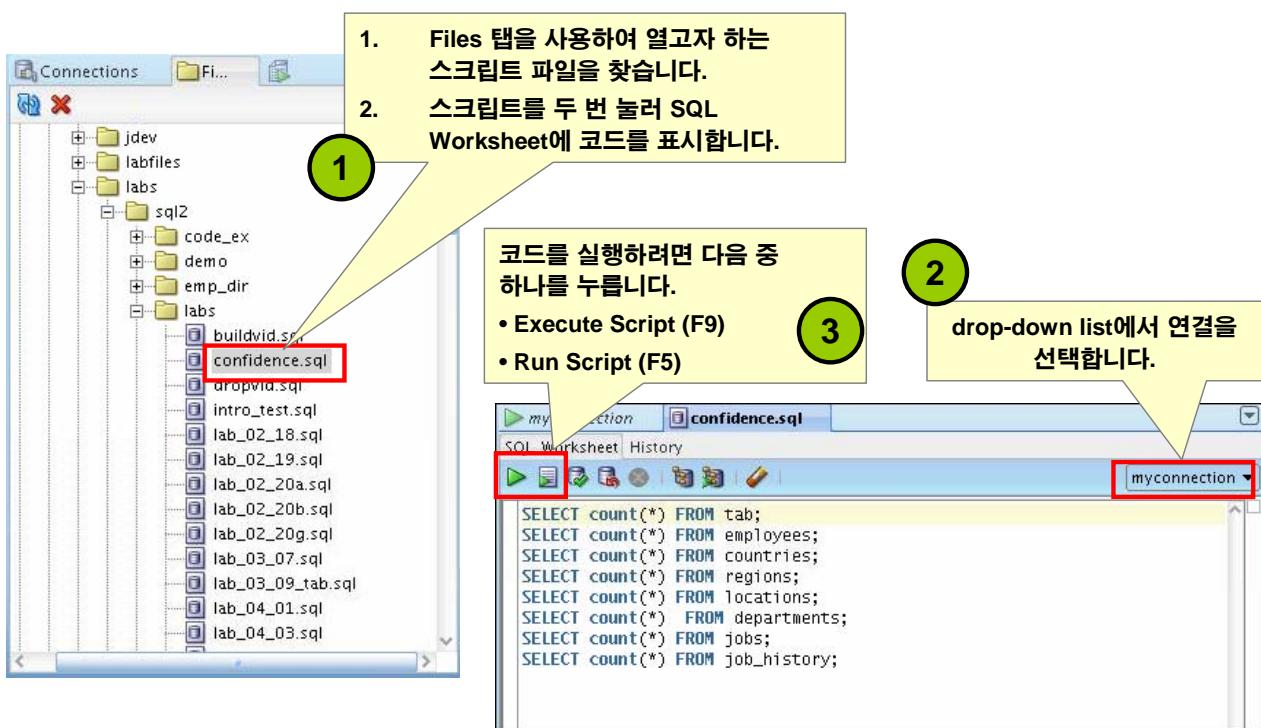
1. Save 아이콘을 누르거나 File > Save 메뉴 옵션을 사용합니다.
2. Windows Save 대화상자에서 파일 이름과 파일을 저장할 위치를 입력합니다.
3. Save를 누릅니다.

내용을 파일에 저장하면 Enter SQL Statement window가 파일 내용의 탭 페이지를 표시합니다. 여러 개의 파일을 동시에 열 수 있으며 각 파일은 탭 페이지로 표시됩니다.

스크립트 경로

스크립트를 찾고 저장할 기본 경로를 선택할 수 있습니다. Tools > Preferences > Database > Worksheet Parameters 아래에서 "Select default path to look for scripts" 필드에 값을 입력하십시오.

저장된 SQL 스크립트 실행: 방법 1



ORACLE

Copyright © 2009, Oracle. All rights reserved.

저장된 SQL 스크립트 실행: 방법 1

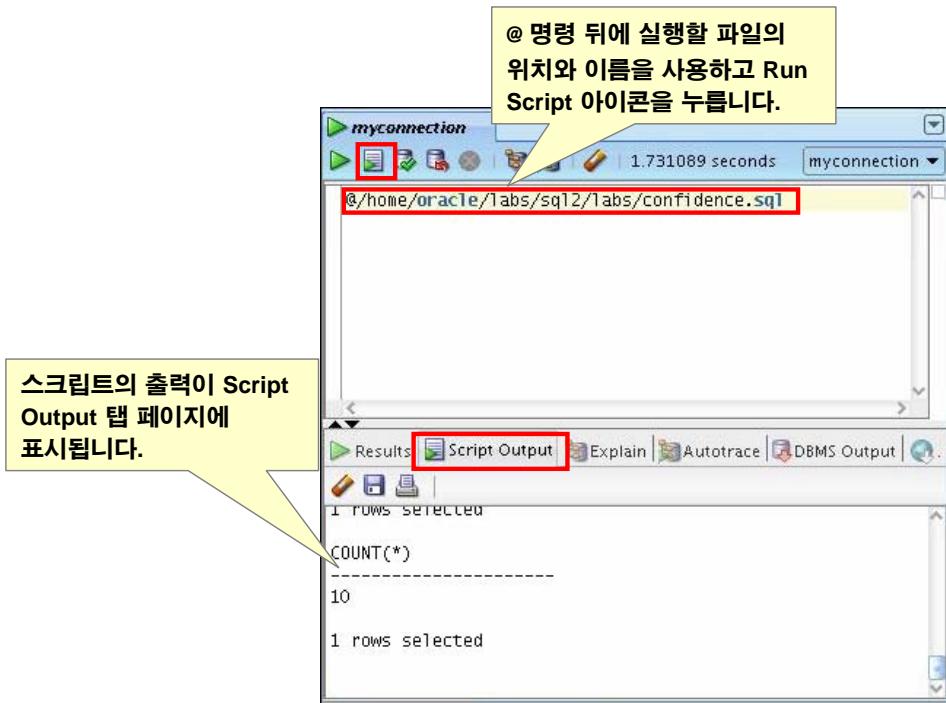
스크립트 파일을 열고 SQL Worksheet 영역에 코드를 표시하려면 다음을 수행합니다.

1. Files Navigator에서 열고자 하는 스크립트 파일을 선택합니다. 또는 해당 파일이 있는 위치로 이동합니다.
2. 두 번 눌러 엽니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
3. connection drop-down list에서 연결을 선택합니다.
4. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다. connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다. 스크립트 실행에 사용할 연결을 선택합니다.

또는 다음을 수행할 수 있습니다.

1. File > Open을 선택합니다. Open 대화상자가 나타납니다.
2. Open 대화상자에서 열고자 하는 스크립트 파일을 선택합니다. (또는 해당 파일이 있는 위치로 이동합니다.)
3. Open을 누릅니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
4. connection drop-down list에서 연결을 선택합니다.
5. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다. connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다. 스크립트 실행에 사용할 연결을 선택합니다.

저장된 SQL 스크립트 실행: 방법 2



Copyright © 2009, Oracle. All rights reserved.

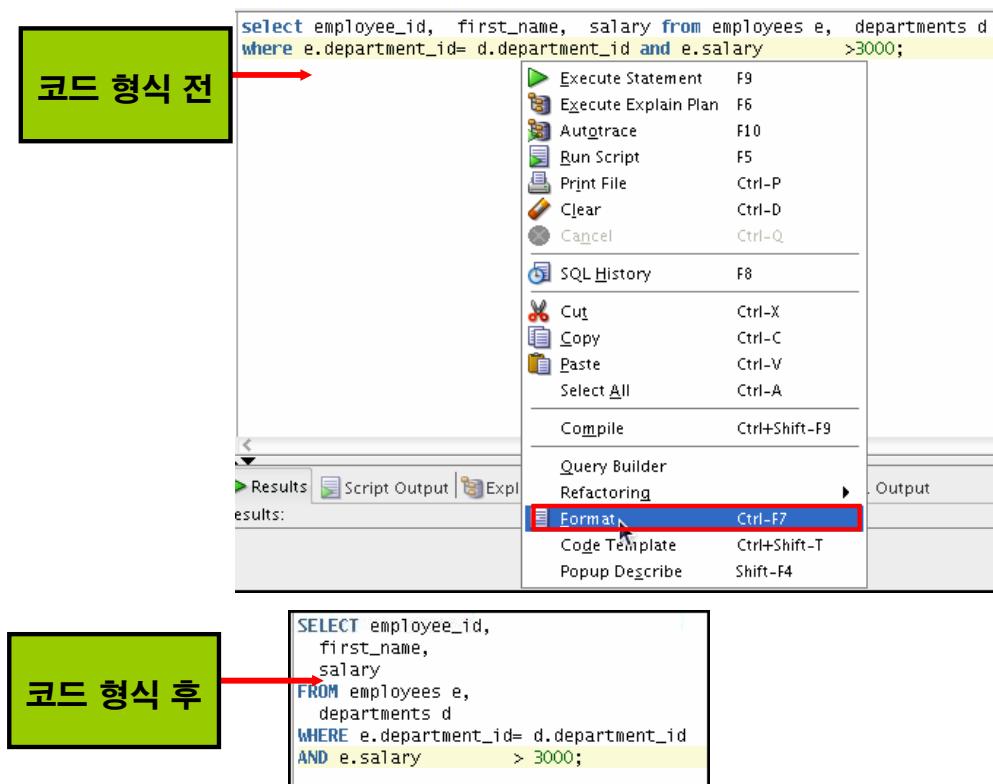
저장된 SQL 스크립트 실행: 방법 2

저장된 SQL 스크립트를 실행하려면 다음을 수행합니다.

1. Enter SQL Statement window에서 @ 명령을 사용하고 그 뒤에 실행할 파일의 위치와 이름을 입력합니다.
2. Run Script 아이콘을 누릅니다.

파일의 실행 결과가 Script Output 탭 페이지에 표시됩니다. 또한 Script Output 탭 페이지에서 Save 아이콘을 눌러 스크립트 출력을 저장할 수도 있습니다. Windows File Save 대화상자가 나타나고 파일의 이름 및 위치를 선택할 수 있습니다.

SQL 코드 형식 지정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 코드 형식 지정

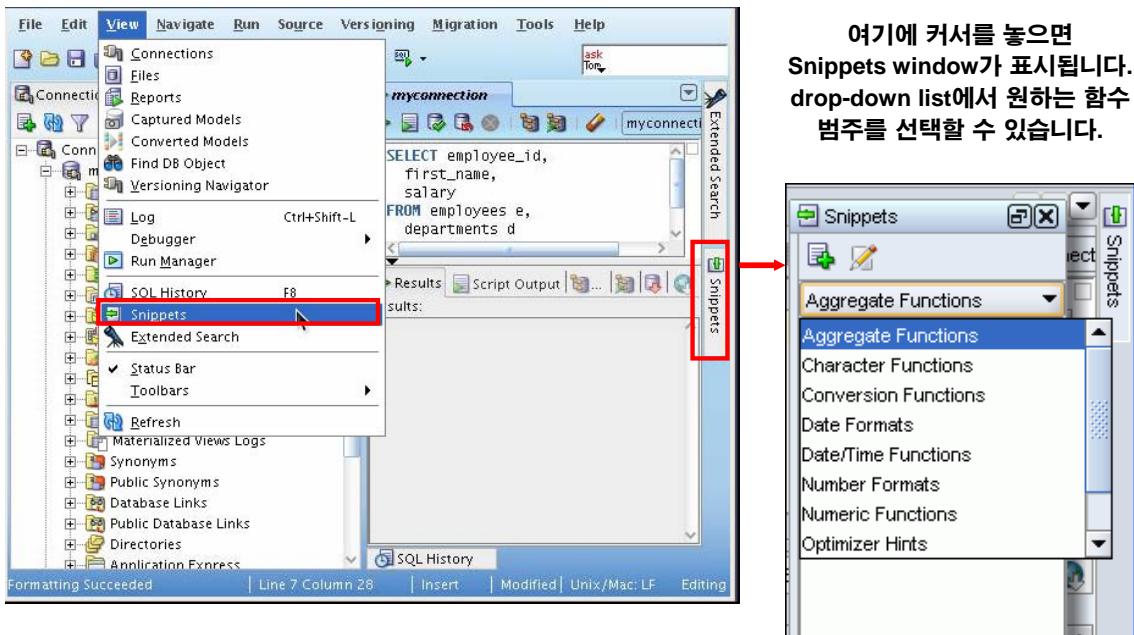
SQL 코드의 들여쓰기, 간격, 대소문자 및 줄 구분을 보기 좋게 지정해야 할 경우가 있습니다.
SQL Developer에는 SQL 코드의 형식을 지정하는 기능이 있습니다.

SQL 코드에 형식을 지정하려면 명령문 영역을 마우스 오른쪽 버튼으로 누른 다음 Format SQL을 선택합니다.

슬라이드의 예제에서 형식이 지정되기 전의 SQL 코드에는 키워드가 대문자로 표시되지 않고 명령문의 들여쓰기가 제대로 되어 있지 않습니다. 형식 지정 이후의 SQL 코드에서는 키워드가 대문자로 표시되고 명령문이 적절히 들여쓰기되어 보기 좋게 다듬어졌습니다.

Snippet 사용

Snippet은 구문이나 예제 등의 코드 부분입니다.



여기에서 커서를 놓으면
Snippets window가 표시됩니다.
drop-down list에서 원하는 함수
범주를 선택할 수 있습니다.

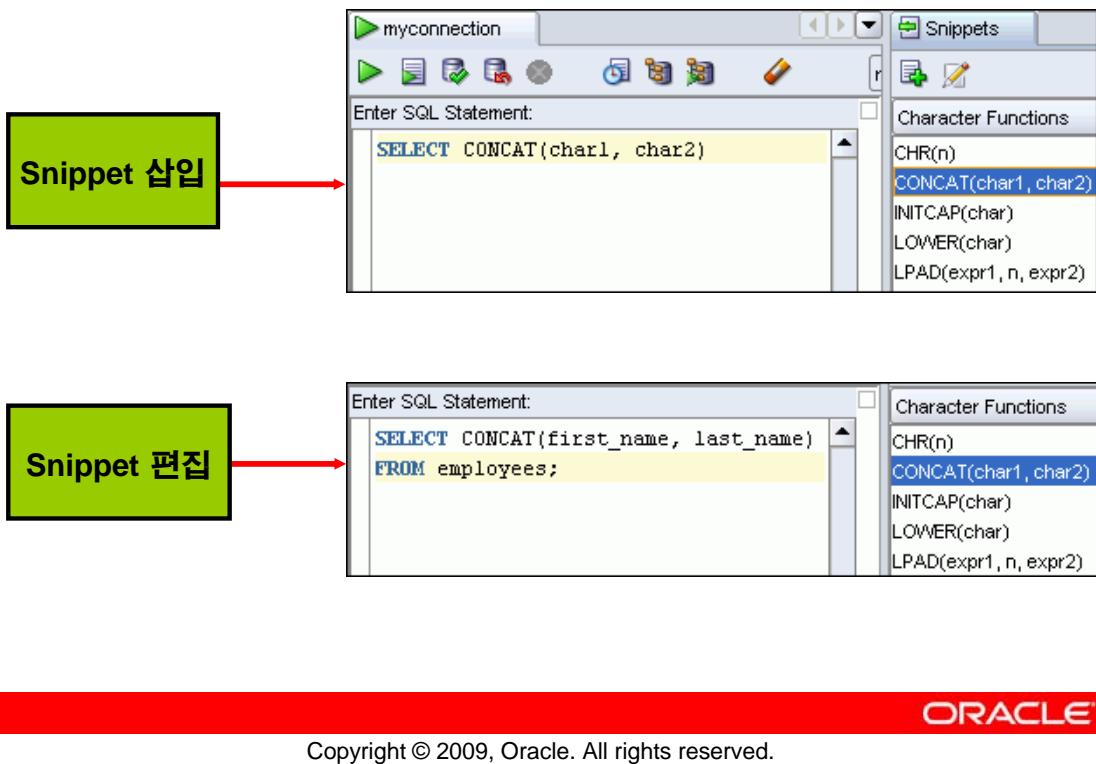
Snippet 사용

SQL Worksheet을 사용하거나 PL/SQL 함수 또는 프로시저를 생성하거나 편집할 때 특정 코드 부분을 사용해야 하는 경우가 있습니다. SQL Developer에는 SQL 함수, 옵티마이저 힌트 및 기타 PL/SQL 프로그래밍 기법과 같은 코드 부분인 Snippet이라는 기능이 있습니다. Snippet 을 Editor window로 끌어올 수 있습니다.

Snippet을 표시하려면 View > Snippets를 선택합니다.

그리면 Snippets window가 오른쪽에 표시됩니다. drop-down list를 사용하여 그룹을 선택할 수 있습니다. Snippets window가 숨겨진 경우 이를 표시할 수 있도록 오른쪽 window 여백에 Snippets 버튼이 표시됩니다.

Snippet 사용: 예제



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Snippet 사용: 예제

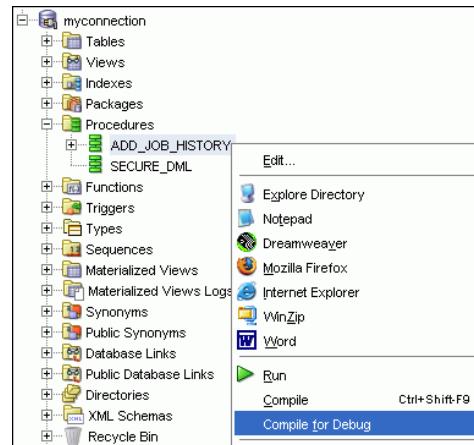
SQL Worksheet 또는 PL/SQL 함수나 프로시저에서 코드에 Snippet을 삽입하려면 Snippet을 Snippets window에서 코드의 원하는 위치로 끌어옵니다. 그런 다음 SQL 함수가 현재 컨텍스트에서 유효하도록 구문을 편집할 수 있습니다. 도구 설명에서 SQL 함수에 대한 간단한 설명을 보려면 커서를 함수 이름 위에 둡니다.

슬라이드의 예제는 Snippets window의 Character Functions 그룹에서 CONCAT(char1, char2)를 끌어오는 과정을 보여줍니다. 이어서 다음과 같이 CONCAT 함수 구문을 편집하고 나머지 명령문을 추가합니다.

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

프로시저 및 함수디 버깅

- **SQL Developer를 사용하여 PL/SQL 함수 및 프로시저를 디버그합니다.**
- **프로시저를 디버그할 수 있도록 PL/SQL 컴파일을 수행하려면 Compile for Debug 옵션을 사용합니다.**
- **중단점을 설정하고 Step Into 및 Step Over 작업을 수행하려면 Debug 메뉴 옵션을 사용합니다.**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 및 함수 디버깅

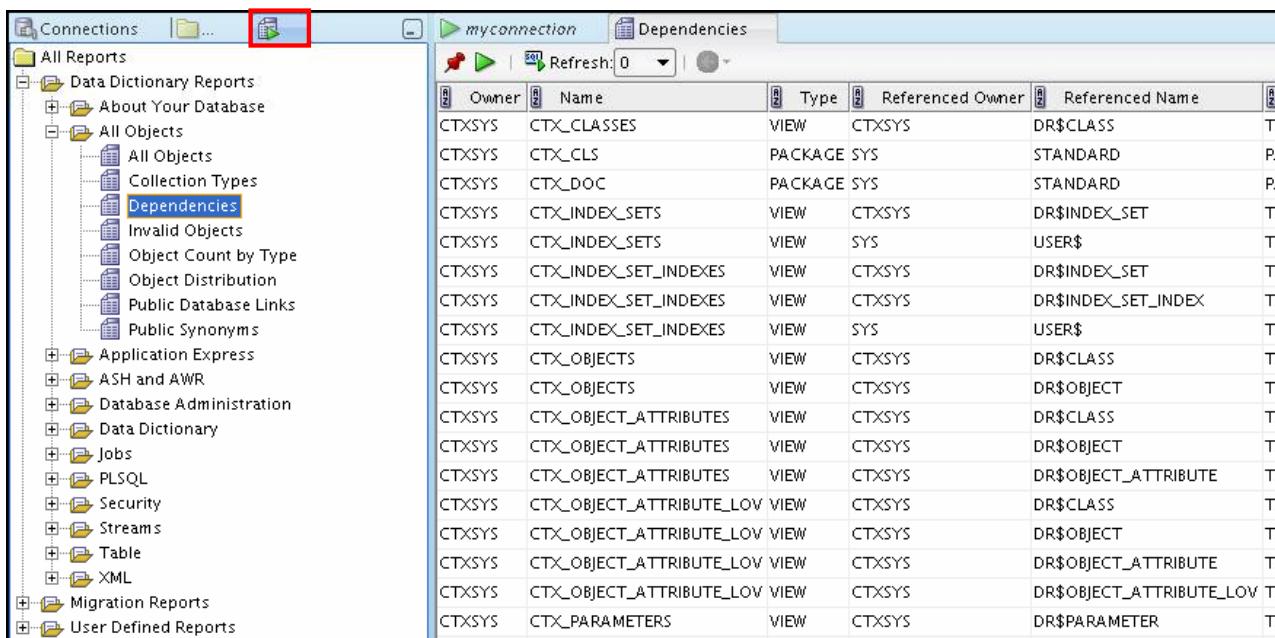
SQL Developer에서는 PL/SQL 프로시저 및 함수를 디버깅 할 수 있습니다. Debug 메뉴 옵션을 사용하여 다음 디버깅 작업을 수행할 수 있습니다.

- **Find Execution Point**를 사용하면 다음 실행 지점으로 이동합니다.
- **Resume**을 사용하면 실행이 계속됩니다.
- **Step Over**를 사용하면 다음 메소드를 건너뛰고 해당 메소드 뒤의 다음 명령문으로 이동합니다.
- **Step Into**를 사용하면 다음 메소드의 첫번째 명령문으로 이동합니다.
- **Step Out**을 사용하면 현재 메소드에서 나와 다음 명령문으로 이동합니다.
- **Step to End of Method**를 사용하면 현재 메소드의 마지막 명령문으로 이동합니다.
- **Pause**를 사용하면 실행이 중지되지만 종료되지는 않으므로 나중에 실행을 재개할 수 있습니다.
- **Terminate**를 사용하면 실행이 중지 및 종료됩니다. 이 지점에서는 실행을 재개할 수 없습니다. 대신 함수나 프로시저 시작 부분부터 실행이나 디버깅을 시작하려면 Source 탭 도구 모음의 Run 또는 Debug 아이콘을 누르십시오.
- **Garbage Collection**을 사용하면 캐시에서 무효한 객체가 제거되고 자주 액세스하는 유효한 객체가 사용됩니다.

이러한 옵션은 디버깅 도구 모음에서 아이콘으로도 사용할 수 있습니다.

데이터베이스 보고

SQL Developer에서는 데이터베이스 및 해당 객체에 대한 여러 가지 미리 정의된 보고서가 제공됩니다.



The screenshot shows the SQL Developer interface with the 'Reports' tab selected in the top navigation bar. On the left, there is a tree view of report categories. Under 'All Objects', the 'Dependencies' report is highlighted with a red box. On the right, a table titled 'Dependencies' displays data for various objects, with the first few rows shown below:

Owner	Name	Type	Referenced Owner	Referenced Name
CTXSYS	CTX_CLASSES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_CLS	PACKAGE	SYS	STANDARD
CTXSYS	CTX_DOC	PACKAGE	SYS	STANDARD
CTXSYS	CTX_INDEX_SETS	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SETS	VIEW	SYS	USER\$
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET_INDEX
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	SYS	USER\$
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE_LOV
CTXSYS	CTX_PARAMETERS	VIEW	CTXSYS	DR\$PARAMETER

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 보고

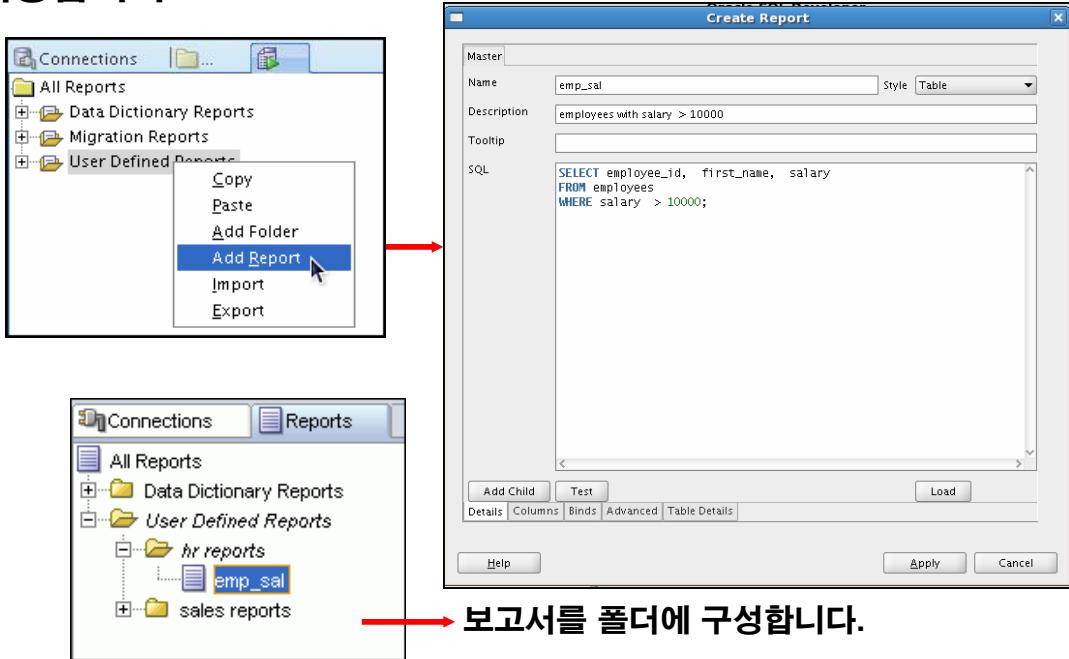
SQL Developer는 데이터베이스 및 객체에 대한 다양한 보고서를 제공합니다. 이러한 보고서는 다음 범주로 그룹화할 수 있습니다.

- About Your Database 보고서
- Database Administration 보고서
- Table 보고서
- PL/SQL 보고서
- Security 보고서
- XML 보고서
- Jobs 보고서
- Streams 보고서
- All Objects 보고서
- Data Dictionary 보고서
- 사용자 정의 보고서

보고서를 표시하려면 window 왼쪽의 Reports 탭을 누르십시오. Window 오른쪽의 탭 창에 개별 보고서가 표시됩니다. 각 보고서에 대해 drop-down list를 사용하여 보고서를 표시할 데이터베이스 연결을 선택할 수 있습니다. 객체에 대한 보고서의 경우 선택한 데이터베이스 연결과 연관된 데이터베이스 유저가 볼 수 있는 객체만 표시되고 행은 일반적으로 Owner별로 정렬됩니다. 고유한 유저 정의 보고서를 생성할 수도 있습니다.

유저 정의 보고서 작성

반복적으로 사용할 수 있도록 유저 정의 보고서를 생성하고 저장합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

유저 정의 보고서 작성

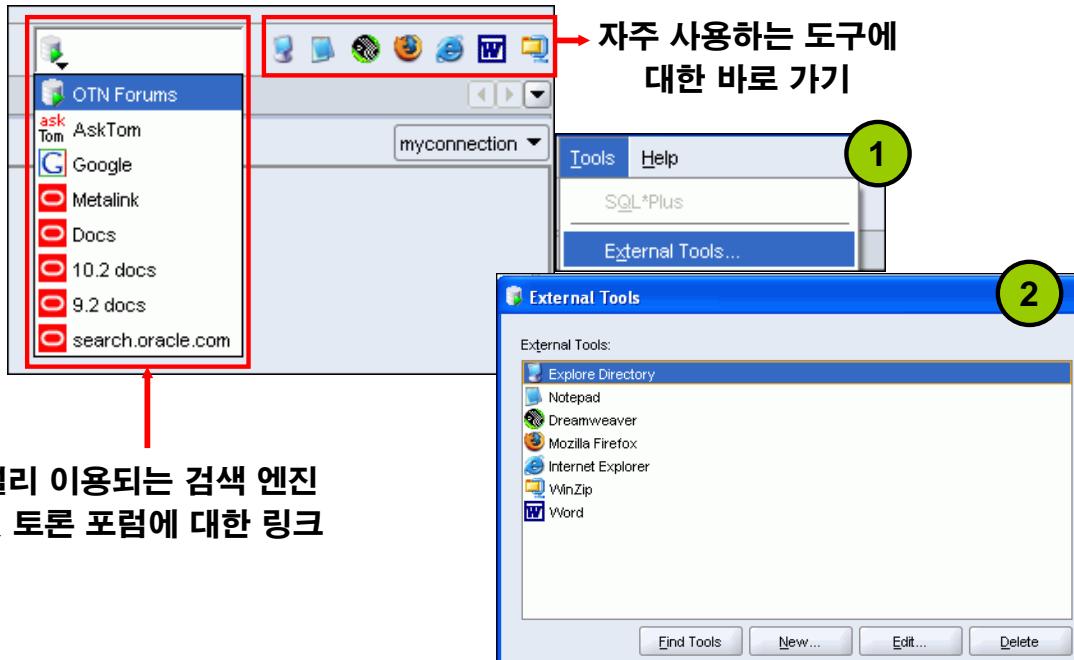
유저 정의 보고서는 SQL Developer 유저가 생성하는 보고서입니다. 유저 정의 보고서를 생성하려면 다음 단계를 수행하십시오.

1. Reports 아래에서 User Defined Reports 노드를 마우스 오른쪽 버튼으로 누르고 Add Report를 선택합니다.
2. Create Report 대화상자에서 보고서 이름을 지정하고 보고서 정보를 검색할 SQL query를 지정합니다. 그런 다음 Apply를 누릅니다.

슬라이드의 예제에서 보고서 이름은 emp_sal로 지정됩니다. 보고서에 급여 ≥ 10000 인 사원에 대한 세부 정보가 포함되어 있음을 나타내는 선택적 설명이 제공됩니다. 유저 정의 보고서에 표시할 정보를 검색하기 위한 전체 SQL 문이 SQL 상자에서 지정됩니다. Reports Navigator 화면에 있는 보고서 이름 위에 커서를 놓으면 표시될 선택적 도구 설명도 포함할 수 있습니다.

유저 정의 보고서를 폴더에 구성하고 폴더와 하위 폴더의 계층을 생성할 수 있습니다. 유저 정의 보고서에 대한 폴더를 생성하려면 User Defined Reports 노드나 해당 노드 아래에 있는 임의의 폴더 이름을 마우스 오른쪽 버튼으로 누르고 Add Folder를 선택합니다. 이러한 보고서에 대한 폴더를 포함한 유저 정의 보고서에 대한 정보는 유저 특정 정보를 위한 디렉토리 아래 UserReports.xml이라는 파일에 저장됩니다.

검색 엔진 및 External 도구



널리 이용되는 검색 엔진
및 토론 포럼에 대한 링크

자주 사용하는 도구에
대한 바로 가기

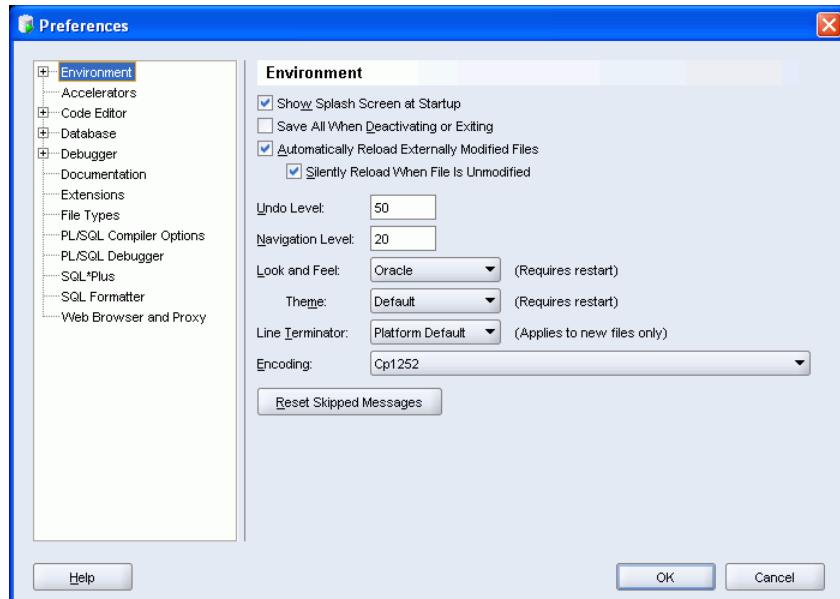
검색 엔진 및 External 도구

SQL 개발자의 생산성을 높이기 위해 SQL Developer에는 AskTom, Google 등의 유명 검색 엔진과 토의 포럼에 대한 빠른 링크가 포함되었습니다. 또한 메모장, Microsoft Word, Dreamweaver 등의 자주 사용하는 일부 도구에 대한 단축키 아이콘도 사용할 수 있습니다. 기존 리스트에 external 도구를 추가하거나 자주 사용하지 않는 도구에 대한 단축키를 삭제할 수도 있습니다. 이를 위해 다음을 수행하십시오.

1. Tools 메뉴에서 External Tools를 선택합니다.
2. 새 도구를 추가하려면 External Tools 대화상자에서 New를 선택합니다. 리스트에서 도구를 제거하려면 Delete를 선택합니다.

환경 설정

- SQL Developer 인터페이스와 환경을 커스터마이즈합니다.
- Tools 메뉴에서 Preferences를 선택합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

환경 설정

필요에 따라 SQL Developer 환경 설정을 수정하여 SQL Developer 인터페이스 및 환경의 여러 측면을 커스터마이즈할 수 있습니다. SQL Developer 환경 설정을 수정하려면 Tools, Preferences 를 차례로 선택합니다.

환경 설정은 다음 범주로 그룹화됩니다.

- Environment
- Accelerators (Keyboard shortcuts)
- Code Editors
- Database
- Debugger
- Documentation
- Extensions
- File Types
- Migration
- PL/SQL Compilers
- PL/SQL Debugger 등

SQL Developer 레이아웃 재설정

```
[oracle@EDRSR5P1 ~]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.40/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
[oracle@EDRSR5P1 ~]$ cd /home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ ls
Debugging.layout  Editing.layout  projects      windowinglayout.xml
dtcache.xml       preferences.xml   settings.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ rm windowinglayout.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 레이아웃 재설정

SQL Developer로 작업하는 동안 Connections Navigator가 사라지거나 Log window를 원래의 위치에 도킹할 수 없는 경우 다음 단계를 수행하여 문제를 해결합니다.

1. SQL Developer를 종료합니다.
2. 터미널 window를 열고 locate 명령을 사용하여 windowinglayout.xml의 위치를 찾습니다.
3. windowinglayout.xml이 있는 디렉토리로 이동하여 해당 파일을 삭제합니다.
4. SQL Developer를 재시작합니다.

요약

이 부록에서는 SQL Developer를 사용하여 다음 작업을 수행하는 방법을 배웠습니다.

- 데이터베이스 객체 탐색, 생성 및 편집
- SQL Worksheet에서 SQL 문 및 스크립트 실행
- 커스텀 보고서 생성 및 저장



Copyright © 2009, Oracle. All rights reserved.

요약

SQL Developer는 데이터베이스 개발 작업을 간편하게 수행할 수 있는 무료 그래픽 도구입니다. SQL Developer를 사용하여 데이터베이스 객체를 탐색, 생성 및 편집할 수 있습니다. 또한 SQL Worksheet를 사용하여 SQL 문 및 스크립트를 실행할 수 있습니다. SQL Developer를 통해 고유의 특수 보고서 집합을 생성하여 저장해 두었다가 반복해서 사용할 수 있습니다.

D

SQL*Plus 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- SQL*Plus에 로그인
- SQL 명령 편집
- SQL*Plus 명령을 사용하여 출력 형식 지정
- 스크립트 파일과 상호 작용

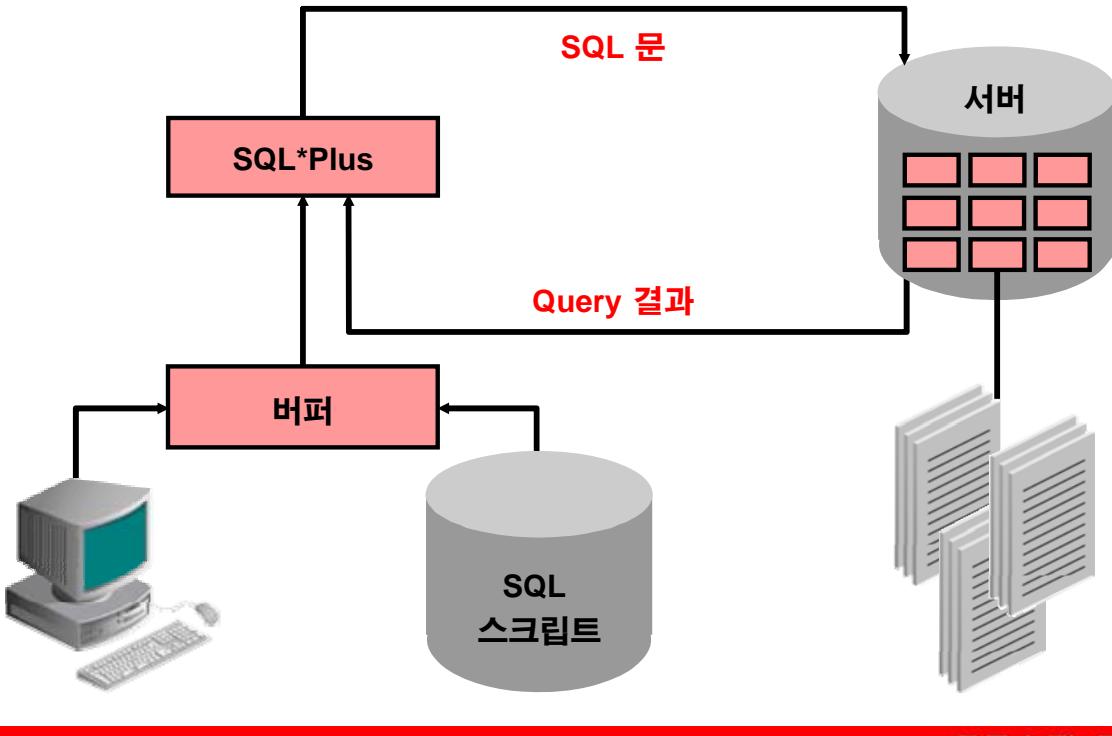
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

몇 번이고 사용할 수 있는 SELECT 문을 생성할 수 있습니다. 이 부록에서는 SQL*Plus 명령을 사용하여 SQL 문을 실행하는 과정도 다릅니다. SQL*Plus 명령을 사용하여 출력 형식을 지정하고, SQL 명령을 편집하고, SQL*Plus로 스크립트를 저장하는 방법을 배웁니다.

SQL과 SQL*Plus의 상호 작용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus

SQL은 임의의 도구 또는 응용 프로그램에서 Oracle 서버와 통신하는 데 사용하는 명령 언어입니다. Oracle SQL은 많은 확장을 포함합니다. SQL 문을 입력하면 SQL 버퍼(SQL Buffer)라고 하는 메모리 부분에 저장되어 새 SQL 문을 입력할 때까지 그대로 남아 있습니다. SQL*Plus는 SQL 문을 인식하여 실행을 위해 Oracle9i Server로 제출하는 오라클 도구입니다. SQL*Plus는 자체 명령어를 포함합니다.

SQL의 특성

- 프로그래밍 경험이 별로 없거나 전혀 없는 유저를 포함하여 다양한 유저가 사용할 수 있습니다.
- 비절차적 언어입니다.
- 시스템 생성 및 유지 관리에 필요한 시간을 단축합니다.
- 영어와 유사한 언어입니다.

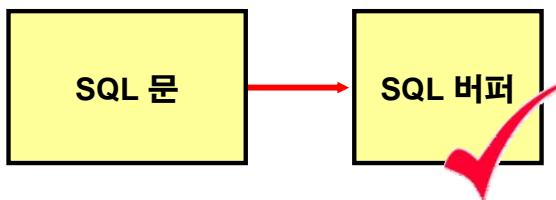
SQL*Plus의 특성

- 명령문의 임시 항목을 받아들입니다.
- 파일을 사용하여 SQL을 입력할 수 있습니다.
- SQL 문 수정을 위한 행 편집기를 제공합니다.
- 환경 설정을 제어합니다.
- Query 결과를 기본 보고서 형식으로 만듭니다.
- 로컬 및 원격 데이터베이스에 액세스합니다.

SQL 문과 SQL*Plus 명령 비교

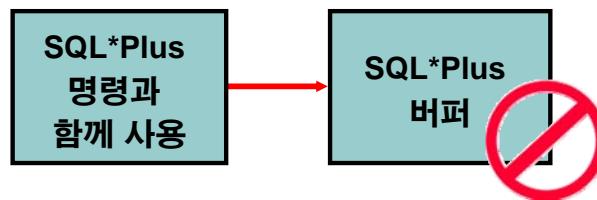
SQL

- 언어
- ANSI 표준
- 키워드를 약어로 표기할 수 없음
- 명령문으로 데이터베이스의 데이터 및 테이블 정의를 조작함



SQL*Plus

- 환경
- Oracle 고유
- 키워드는 약어로 표시할 수 있음
- 명령으로 데이터베이스의 값을 조작할 수 없음



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus (계속)

다음 표에서는 SQL과 SQL*Plus를 비교합니다.

SQL	SQL*Plus
Oracle 서버와 통신하여 데이터에 액세스하기 위한 언어	SQL 문을 인식하고 서버로 보냄
ANSI(American National Standards Institute) 표준 SQL을 기반으로 함	SQL 문을 실행하기 위한 오라클 고유의 인터페이스
데이터베이스의 데이터 및 테이블을 조작함	데이터베이스의 값을 조작할 수 없음
SQL 버퍼에 하나 이상의 행이 입력됨	한 번에 한 행씩 입력되고 SQL 버퍼에 저장되지 않음
연속 문자가 없음	명령이 한 줄보다 긴 경우 연속 문자로 대시(-)를 사용
약어를 사용할 수 없음	약어를 사용할 수 있음
종료 문자를 사용하여 명령을 즉시 실행	종료 문자를 사용할 필요 없이 명령을 즉시 실행
함수를 사용하여 일부 형식 지정 수행	명령을 사용하여 데이터의 형식 지정

SQL*Plus: 개요

- SQL*Plus에 로그인합니다.
- 테이블 구조를 설명합니다.
- SQL 문을 편집합니다.
- SQL*Plus에서 SQL을 실행합니다.
- SQL 문을 파일에 저장하고 첨부합니다.
- 저장된 파일을 실행합니다.
- 파일에서 버퍼로 명령을 로드하여 편집합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus

SQL*Plus는 다음 작업을 수행할 수 있는 환경입니다.

- 데이터베이스에서 데이터를 검색, 수정, 추가 및 제거하는 SQL 문을 실행합니다.
- query 결과의 형식을 지정하고 계산을 수행하고, 저장하고, 보고서 형식으로 인쇄합니다.
- 앞으로 반복 사용할 수 있도록 SQL 문을 저장하는 스크립트 파일을 생성합니다.

SQL*Plus 명령은 다음의 주요 범주로 나눌 수 있습니다.

범주	목적
환경	세션에 대한 SQL 문의 일반 동작에 영향을 줍니다.
형식	Query 결과의 형식을 지정합니다.
파일 조작	스크립트 파일을 저장, 로드, 실행합니다.
실행	SQL 문을 SQL 버퍼에서 Oracle 서버로 보냅니다.
편집	버퍼의 SQL 문을 수정합니다.
상호 작용	변수를 생성하여 SQL 문에 전달하고, 변수 값을 인쇄하고, 화면에 메시지를 출력합니다.
기타	데이터베이스에 연결하고, SQL*Plus 환경을 조작하며, 열 정의를 표시합니다.

SQL*Plus에 로그인

```
[oracle@EDRSR5P1 ~]$sqlplus
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:59:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Enter user-name: ora21@orcl
Enter password:
```

sqlplus [username[/password[@database]]]

```
[oracle@EDRSR5P1 ~]$sqlplus ora21/ora21@orcl
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:58:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.0.2 - Beta
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL>
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus에 로그인

SQL*Plus를 호출하는 방법은 오라클 데이터베이스를 실행하는 운영 체제의 종류에 따라 다릅니다.

Linux 환경에서 로그인 하려면 다음 단계를 수행하십시오.

1. Oracle Database 11g: PL/SQL Fundamentals D - 6
2. SQL*Plus에 로그인
3. SQL*Plus를 호출하는 방법은 오라클 데이터베이스를 실행하는 운영 체제의 종류에 따라 다릅니다.

이 구문에서 다음이 적용됩니다.

`username` 은 데이터베이스 `username`입니다.

`password` 는 데이터베이스 암호입니다(여기에 암호를 입력하면 암호가 보입니다.).

`@database` 는 데이터베이스 연결 문자열입니다.

참고: 암호의 무결성을 보장하려면 운영 체제 프롬프트에서 암호를 입력하지 마십시오. 대신 `username`만 입력하십시오. 암호는 암호 프롬프트에서 입력하십시오.

테이블 구조 표시

SQL*Plus DESCRIBE 명령을 사용하여 테이블의 구조를 표시합니다.

```
DESC[RIBE] tablename
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시

SQL*Plus에서 DESCRIBE 명령을 사용하여 테이블의 구조를 표시할 수 있습니다. 이 명령의 결과로 열 이름 및 데이터 유형이 표시되고 열에 반드시 데이터가 포함되어야 하는지 여부가 나타납니다.

이 구문에서 다음이 적용됩니다.

tablename 은 유저가 액세스할 수 있는 기존의 테이블, 뷰 또는 동의어의 이름입니다.
DEPARTMENTS 테이블을 기술하려면 다음 명령을 사용합니다.

```
SQL> 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)
```

테이블 구조 표시

```
DESCRIBE departments
```

Name	Null?	Type
<hr/>		
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시 (계속)

슬라이드의 예제는 DEPARTMENTS 테이블의 구조에 대한 정보를 표시합니다. 결과에서 다음이 적용됩니다.

Null?: 열에 데이터가 포함되어야 하는지 여부를 지정합니다. NOT NULL은 열에 데이터가 포함되어야 함을 나타냅니다.

Type: 열의 데이터 유형을 표시합니다.

SQL*Plus 편집 명령

- **A[PPEND] *text***
- **C[HANGE] / *old* / *new***
- **C[HANGE] / *text* /**
- **CL[EAR] BUFF[ER]**
- **DEL**
- **DEL *n***
- **DEL *m* *n***

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 편집 명령

SQL*Plus 명령은 한 번에 한 행씩 입력되고 SQL 버퍼에 저장되지 않습니다.

명령	설명
A[PPEND] <i>text</i>	현재 행의 끝에 텍스트를 추가합니다.
C[HANGE] / <i>old</i> / <i>new</i>	현재 행에서 <i>old</i> 텍스트를 <i>new</i> 로 변경합니다.
C[HANGE] / <i>text</i> /	현재 행에서 <i>text</i> 를 삭제합니다.
CL[EAR] BUFF[ER]	SQL 버퍼에서 모든 행을 삭제합니다.
DEL	현재 행을 삭제합니다.
DEL <i>n</i>	<i>n</i> 행을 삭제합니다.
DEL <i>m</i> <i>n</i>	<i>m</i> 행부터 <i>n</i> 행까지 삭제합니다.

자침

- 명령을 완료하기 전에 Enter 키를 누르면 SQL*Plus에서 행 번호를 표시합니다.
- 종료 문자(세미콜론이나 슬래시) 중 하나를 입력하거나 Enter 키를 두 번 눌러 SQL 버퍼를 종료합니다. 그러면 SQL 프롬프트가 나타납니다.

SQL*Plus 편집 명령

- **I[NPUT]**
- **I[NPUT] *text***
- **L[IST]**
- **L[IST] *n***
- **L[IST] *m n***
- **R[UN]**
- ***n***
- ***n text***
- **0 *text***

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 편집 명령 (계속)

명령	설명
I[NPUT]	임의 개수의 행을 삽입합니다.
I[NPUT] <i>text</i>	<i>text</i> 로 구성된 행을 삽입합니다.
L[IST]	SQL 버퍼에 있는 모든 행을 나열합니다.
L[IST] <i>n</i>	한 행(<i>n</i> 으로 지정된 행)을 나열합니다.
L[IST] <i>m n</i>	일정 범위(<i>m-n</i>)의 행을 나열합니다.
R[UN]	버퍼에 있는 현재 SQL 문을 표시하고 실행합니다.
<i>n</i>	<i>n</i> 행을 현재 행으로 지정합니다.
<i>n text</i>	<i>n</i> 행을 <i>text</i> 로 대체합니다.
0 <i>text</i>	1행 앞에 행을 삽입합니다.

참고: 각 SQL 프롬프트에서 하나의 SQL*Plus 명령만 입력할 수 있습니다. SQL*Plus 명령은 버퍼에 저장되지 않습니다. SQL*Plus 명령이 다음 행으로 이어지는 경우 첫 행의 끝에 하이픈 (-)을 추가합니다.

LIST, n 및 APPEND 사용

LIST

```
1  SELECT last_name
2* FROM   employees
```

1

```
1* SELECT last_name
```

A , job_id

```
1* SELECT last_name, job_id
```

LIST

```
1  SELECT last_name, job_id
2* FROM   employees
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LIST, n 및 APPEND 사용

- L[IST] 명령을 사용하여 SQL 버퍼의 내용을 표시합니다. 버퍼에서 2행 옆의 별표(*)는 해당 행이 현재 행임을 나타냅니다. 편집한 내용은 현재 행에 적용됩니다.
- 편집하려는 행 번호(n)를 입력하여 현재 행의 번호를 바꿉니다. 새로운 현재 행이 표시됩니다.
- A[PPEND] 명령을 사용하여 현재 행에 텍스트를 추가합니다. 새로 편집한 행이 표시됩니다. LIST 명령을 사용하여 버퍼의 새 내용을 확인합니다.

참고: LIST 및 APPEND를 비롯한 많은 SQL*Plus 명령은 첫번째 문자만 사용하여 약어로 표기할 수 있습니다. LIST는 L, APPEND는 A라는 약어로 표기할 수 있습니다.

CHANGE 명령 사용

LIST

```
1* SELECT * from employees
```

c/employees/departments

```
1* SELECT * from departments
```

LIST

```
1* SELECT * from departments
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

CHANGE 명령 사용

- L[IST]를 사용하여 버퍼의 내용을 표시합니다.
- C[HANGE] 명령을 사용하여 SQL 버퍼에서 현재 행의 내용을 변경합니다. 이 경우 employees 테이블을 departments 테이블로 대체합니다. 새로운 현재 행이 표시됩니다.
- L[IST] 명령을 사용하여 버퍼의 새 내용을 확인합니다.

SQL*Plus 파일 명령

- **SAVE filename**
- **GET filename**
- **START filename**
- **@ filename**
- **EDIT filename**
- **SPOOL filename**
- **EXIT**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 파일 명령

SQL 문은 Oracle 서버와 통신합니다. SQL*Plus 명령은 환경을 제어하고 query 결과의 서식을 지정하고 파일을 관리합니다. 다음 표에 설명된 명령을 사용할 수 있습니다.

명령	설명
SAV[E] <i>filename</i> [.ext] [REP[LACE]APP[END]]	SQL 버퍼의 현재 내용을 파일에 저장합니다. 기존 파일에 추가하려면 APPEND를 사용하고 기존 파일을 덮어 쓰려면 REPLACE를 사용합니다. 기본 확장자는 .sql입니다.
GET <i>filename</i> [.ext]	이전에 저장한 파일의 내용을 SQL 버퍼에 씁니다. 파일 이름의 기본 확장자는 .sql입니다.
STA[RT] <i>filename</i> [.ext]	이전에 저장한 명령 파일을 실행합니다.
@ <i>filename</i>	이전에 저장한 명령 파일을 실행합니다(START와 동일).
ED[IT]	편집기를 호출하여 버퍼 내용을 afiedt.buf라는 파일에 저장합니다.
ED[IT] [<i>filename</i> [.ext]]	편집기를 호출하여 저장된 파일의 내용을 편집합니다.
SPO[OL] [<i>filename</i> [.ext]] OFF OUT	Query 결과를 파일에 저장합니다. OFF는 스팔 파일을 닫습니다. OUT은 스팔 파일을 닫고 파일 결과를 프린터로 보냅니다.
EXIT	SQL*Plus를 종료합니다.

SAVE 및 START 명령 사용

LIST

```
1  SELECT last_name, manager_id, department_id
2* FROM employees
```

SAVE my_query

Created file my_query

START my_query

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
107 rows selected.		

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SAVE 및 START 명령 사용

SAVE

SAVE 명령을 사용하여 버퍼의 현재 내용을 파일에 저장합니다. 이와 같은 방법으로 자주 사용되는 스크립트를 나중에 사용할 수 있도록 저장할 수 있습니다.

START

START 명령을 사용하여 SQL*Plus에서 스크립트를 실행합니다. 또는 기호 @을 사용하여 스크립트를 실행할 수도 있습니다.

`@my_query`

SERVEROUTPUT 명령

- SET SERVEROUT[PUT] 명령을 사용하여 내장 프로시저 또는 PL/SQL 블록의 출력을 SQL*Plus에 표시할지 여부를 제어할 수 있습니다.
- DBMS_OUTPUT 행 길이 제한이 255바이트에서 32,767 바이트로 늘어났습니다.
- 이제 기본 크기에 제한이 없습니다.
- SERVEROUTPUT을 설정하는 경우 리소스가 미리 할당되지 않습니다.
- Physical memory를 절약하려는 경우가 아니라면 UNLIMITED를 사용하십시오. 그래도 성능 저하가 발생하지 않습니다.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNLIMTED}]  
[FOR[MAT] {WRA[PPED] | WOR[D_WWRAPPED] | TRU[NCATED]}]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SERVEROUTPUT 명령

대부분의 PL/SQL 프로그램에서는 SQL 문을 통해 입력 및 출력을 수행하여 데이터를 데이터베이스 테이블에 저장하거나 해당 테이블을 query합니다. 다른 모든 PL/SQL 입/출력은 다른 프로그램과 상호 작용하는 API를 통해 수행됩니다. 예를 들어, DBMS_OUTPUT 패키지에는 PUT_LINE과 같은 프로시저가 포함되어 있습니다. PL/SQL 외부에서 결과를 보려면 DBMS_OUTPUT에 전달된 데이터를 읽고 표시할 수 있는 SQL*Plus와 같은 다른 프로그램이 필요합니다.

SQL*Plus에서 DBMS_OUTPUT 데이터를 표시하려면 먼저 다음과 같이 SQL*Plus 명령 SET SERVEROUTPUT ON을 실행해야 합니다.

```
SET SERVEROUTPUT ON
```

참고

- SIZE는 오라클 데이터베이스 서버 내에서 버퍼될 수 있는 출력 크기를 바이트 수로 나타낸 값입니다. 기본값은 UNLIMITED입니다. N은 2,000보다 작거나 1,000,000보다 클 수 없습니다.
- SERVEROUTPUT에 대한 자세한 내용은 *Oracle Database PL/SQL User's Guide and Reference 11g*를 참조하십시오.

SQL*Plus SPOOL 명령 사용

```
SPO[OL] [file_name[.ext]] [CRE[ATE] | REP[LACE] |
APP[END]] | OFF | OUT]
```

옵션	설명
file_name[.ext]	출력을 지정된 파일 이름으로 스플합니다.
CRE[ATE]	지정한 이름으로 새 파일을 생성합니다.
REP[LACE]	기존 파일의 내용을 바꿉니다. 파일이 존재하지 않을 경우 REPLACE를 사용하면 파일이 생성됩니다.
APP[END]	지정한 파일의 끝에 버퍼의 내용을 추가합니다.
OFF	스풀을 정지합니다.
OUT	스풀 작업을 정지하고 파일을 컴퓨터의 표준(기본) 프린터로 보냅니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus SPOOL 명령 사용

SPOOL 명령은 query 결과를 파일에 저장하거나 선택적으로 파일을 프린터로 보냅니다. SPOOL 명령이 향상되었습니다. 이전에는 SPOOL 명령을 사용하여 파일을 생성하거나 바꿀 수만 있었지만 이제는 기존 파일을 바꾸는 것뿐 아니라 기존 파일에 내용을 추가할 수도 있습니다. 기본값은 REPLACE입니다.

화면에 출력을 표시하지 않고 스크립트의 명령으로 생성된 출력을 스플하려면 SET TERMOUT OFF를 사용하십시오. SET TERMOUT OFF는 대화식으로 실행하는 명령의 출력에는 영향을 주지 않습니다.

공백이 들어 있는 파일 이름은 따옴표로 묶어야 합니다. SPOOL APPEND 명령을 사용하여 유효한 HTML 파일을 생성하려면 PROMPT나 유사한 명령을 사용하여 HTML 페이지 헤더 및 바닥글을 생성해야 합니다. SPOOL APPEND 명령은 HTML 태그 구문을 분석하지 않습니다. CREATE, APPEND 및 SAVE 파라미터를 비활성화하려면 SET SQLPLUSCOMPAT[IBILITY]를 9.2 이하로 설정합니다.

AUTOTRACE 명령 사용

- **SELECT, INSERT, UPDATE, DELETE 같은 SQL DML 문을 성공적으로 실행한 후에 보고서를 표시합니다.**
- **이제 보고서에 실행 통계 및 query 실행 경로가 포함될 수 있습니다.**

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]  
[STATISTICS]
```

```
SET AUTOTRACE ON  
-- The AUTOTRACE report includes both the optimizer  
-- execution path and the SQL statement execution  
-- statistics
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

AUTOTRACE 명령 사용

EXPLAIN은 EXPLAIN PLAN을 수행하여 query 실행 경로를 보여줍니다. STATISTICS는 SQL 문 통계를 표시합니다. AUTOTRACE 보고서의 형식은 연결되어 있는 서버의 버전과 서버의 구성에 따라 달라질 수 있습니다. DBMS_XPLAN 패키지를 사용하면 EXPLAIN PLAN 명령의 출력을 미리 정의된 여러 형식으로 간단히 표시할 수 있습니다.

참고

- 패키지 및 서브 프로그램에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g* 설명서를 참조하십시오.
- EXPLAIN PLAN에 대한 자세한 내용은 *Oracle Database SQL Reference 11g*를 참조하십시오.
- 실행 계획 및 통계에 대한 자세한 내용은 *Oracle Database Performance Tuning Guide 11g*를 참조하십시오.

요약

이 부록에서는 다음 작업을 수행하기 위한 환경으로 SQL*Plus를 사용하는 방법을 배웠습니다.

- SQL 문 실행
- SQL 문 편집
- 출력 형식 지정
- 스크립트 파일과 상호 작용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

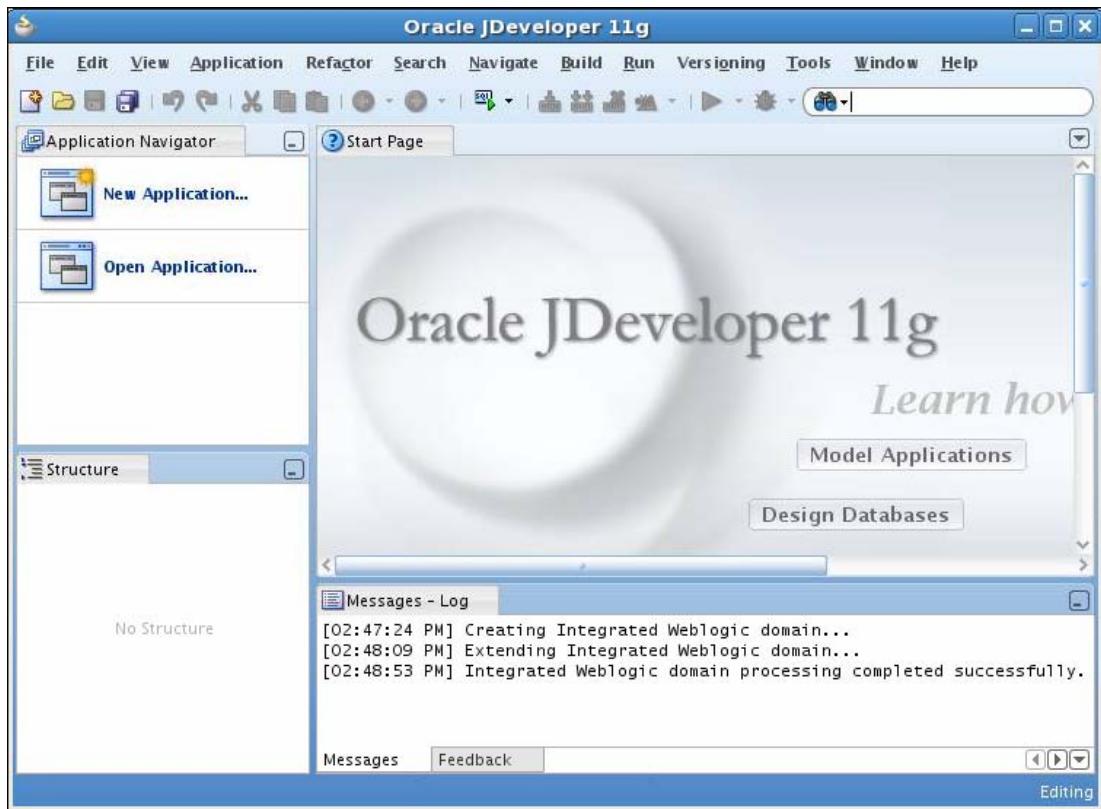
SQL*Plus는 SQL 명령을 데이터베이스 서버로 보내고 SQL 명령을 편집 및 저장하는 데 사용할 수 있는 실행 환경입니다. SQL 프롬프트 또는 스크립트 파일에서 명령을 실행할 수 있습니다.

JDeveloper 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Oracle JDeveloper



ORACLE®

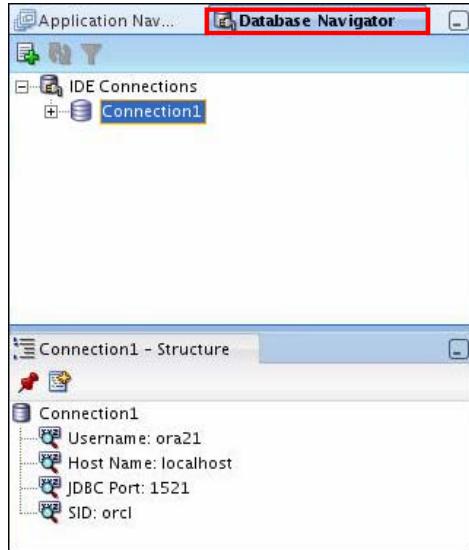
Copyright © 2009, Oracle. All rights reserved.

Oracle JDeveloper

Oracle JDeveloper는 Java 응용 프로그램과 웹 서비스를 개발하여 배치 할 수 있는 IDE(통합 개발 환경)입니다. 모델링에서 배치까지 전 단계의 SDLC(소프트웨어 개발 주기)를 지원합니다. 이 도구에는 응용 프로그램 개발 시 Java, XML 및 SQL에 대한 최신 산업 표준을 사용하는 기능이 있습니다.

Oracle JDeveloper 11g는 시각적/선언적 개발을 지원하는 기능으로 J2EE 개발에 대한 새로운 접근법을 시도합니다. 이 혁신적 접근은 J2EE 개발을 간단하고 효율적으로 만듭니다.

Database Navigator



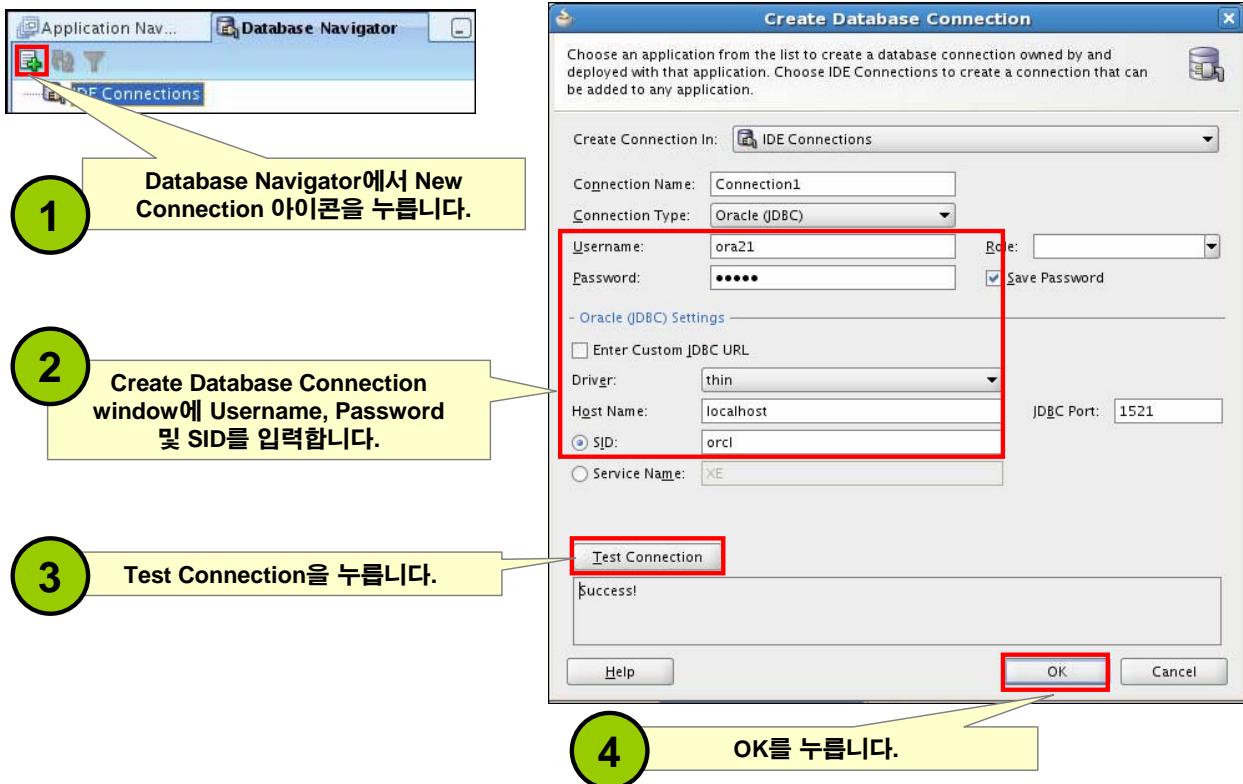
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Database Navigator

Oracle JDeveloper를 사용하면 데이터베이스에 연결하는 데 필요한 정보를 "Connection"이라는 객체에 저장할 수 있습니다. 연결은 IDE 설정의 일부로 저장되며 유저 그룹 간에 공유하기 쉽도록 엑스포트하고 임포트할 수 있습니다. 또한 데이터베이스를 찾고 응용 프로그램을 구축하는 것에서부터 배치에 이르기까지 다양한 용도로 사용됩니다.

연결 생성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

연결 생성

연결은 특정 데이터베이스에 해당 데이터베이스의 특정 유저로 연결하는 데 필요한 정보를 지정하는 객체입니다. 여러 데이터베이스 및 여러 스키마에 대해 연결을 생성하고 테스트할 수 있습니다.

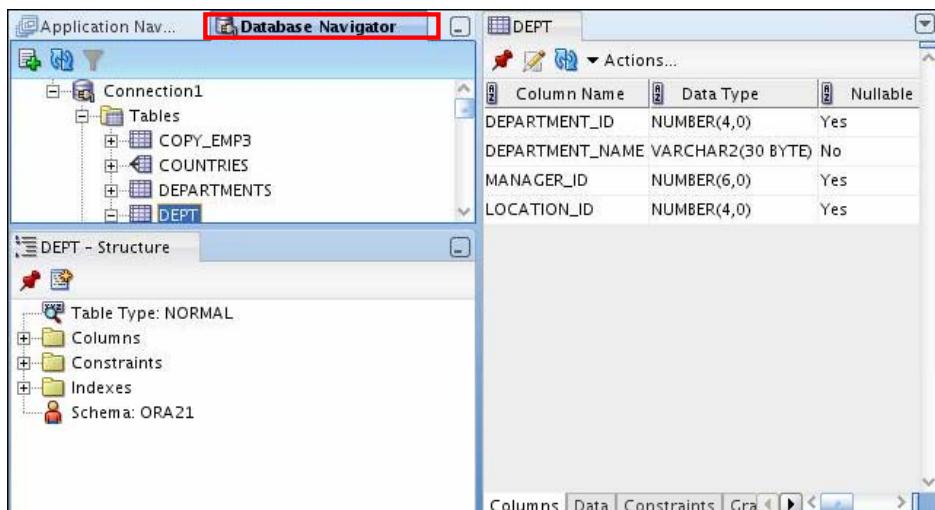
데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. Database Navigator에서 New Connection 아이콘을 누릅니다.
2. Create Database Connection window에 연결 이름을 입력합니다. 연결하려는 스키마의 username 및 암호를 입력합니다. 연결하려는 데이터베이스의 SID를 입력합니다.
3. Test를 눌러 연결이 올바르게 설정되었는지 확인합니다.
4. OK를 누릅니다.

데이터베이스 객체 탐색

Database Navigator를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스 스키마에서 여러 객체 탐색
- 여러 객체의 정의를 한 번에 검토

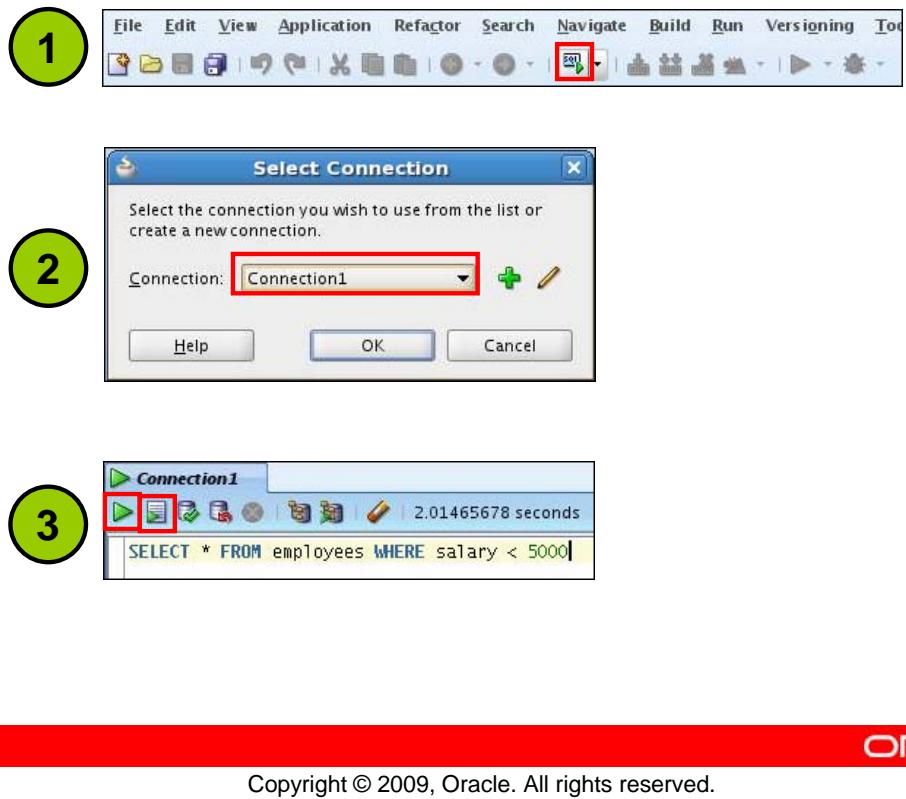


데이터베이스 객체 탐색

데이터베이스 연결을 생성한 후 Database Navigator를 사용하여 데이터베이스 스키마에서 테이블, 뷰, 인덱스, 패키지, 프로시저, 트리거, 유형 등의 여러 객체를 탐색할 수 있습니다.

데이터 딕셔너리에서 추출된 객체 정의가 여러 정보 탭에 분할되어 있는 것을 볼 수 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 읽기 쉬운 탭 페이지에 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 세부 정보가 표시됩니다.

SQL 문 실행



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문 실행

SQL 문을 실행하려면 다음 단계를 수행하십시오.

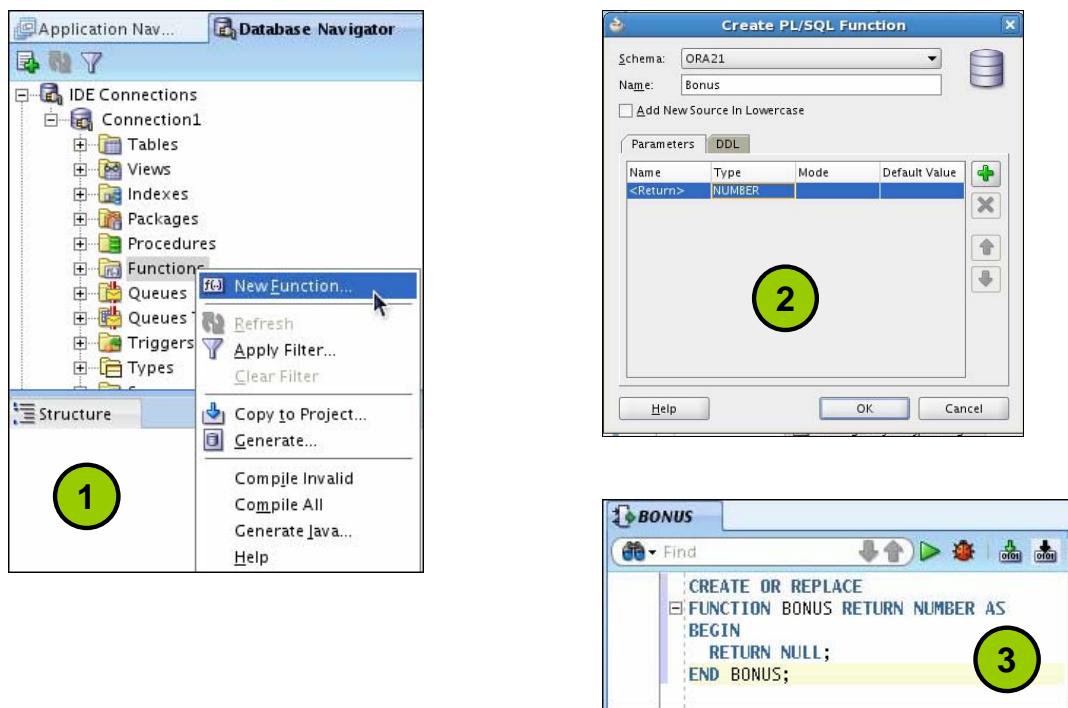
1. Open SQL Worksheet 아이콘을 누릅니다.
2. 연결을 선택합니다.
3. 다음 중 하나를 눌러 SQL 명령을 실행합니다.
 - **Execute statement** 버튼 또는 F9 키를 누릅니다. 출력 결과는 다음과 같습니다.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	100	Steven	King
2	101	Neena	Kochhar

- **Run Script** 버튼 또는 F5 키를 누릅니다. 출력 결과는 다음과 같습니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King

프로그램 단위 생성



함수의 기본 구조

ORACLE

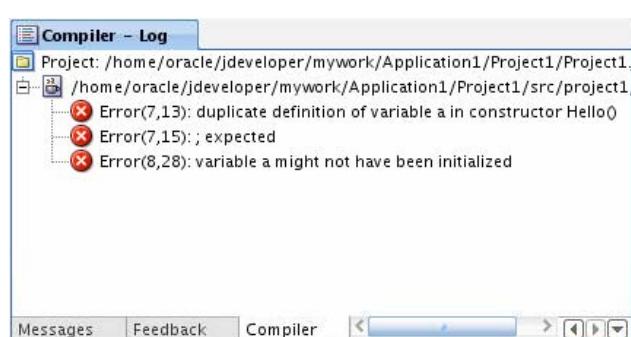
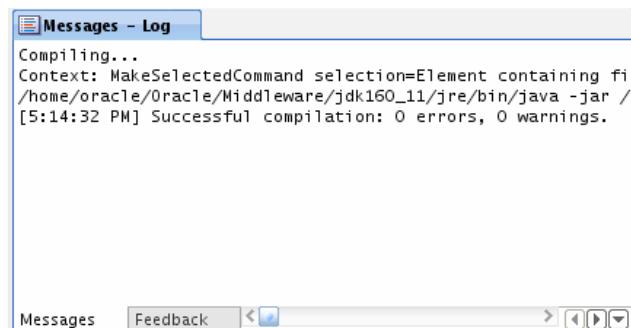
Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 생성

PL/SQL 프로그램 단위를 생성하려면 다음 단계를 수행하십시오.

1. View > Database Navigator를 선택합니다. 데이터베이스 연결을 선택하고 확장합니다. 객체 유형에 해당하는 폴더(Procedures, Packages, Functions)를 마우스 오른쪽 버튼으로 누릅니다. "New [Procedures | Packages | Functions]"를 선택합니다.
2. 함수, 패키지 또는 프로시저에 대한 유효한 이름을 입력하고 OK를 누릅니다.
3. 기본 구조 정의가 생성되어 Code Editor에서 열립니다. 그런 다음 요구에 맞게 서브 프로그램을 편집할 수 있습니다.

컴파일



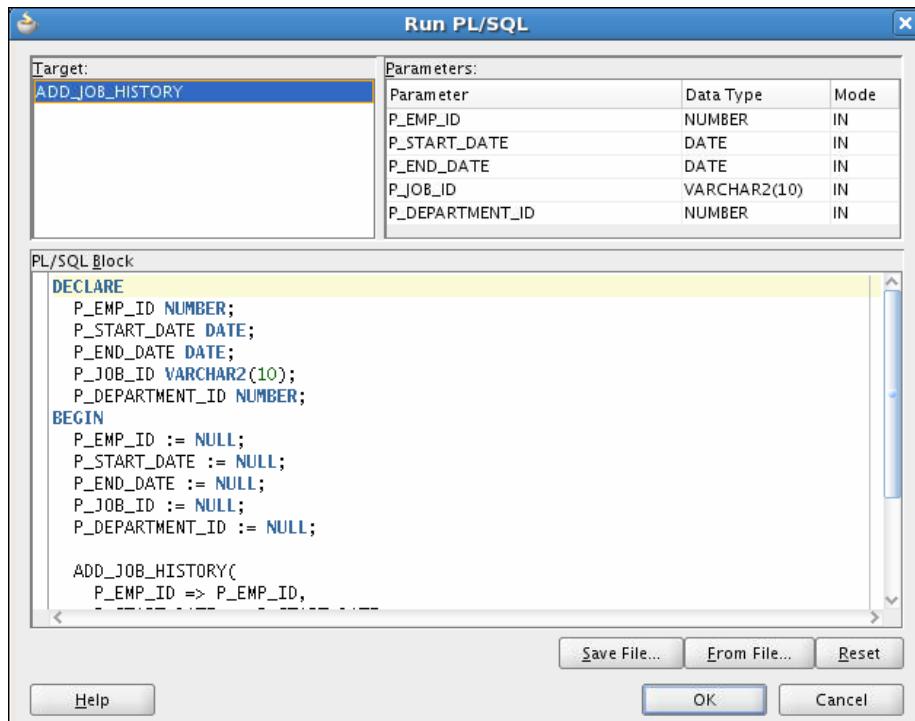
ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일

기본 구조 정의를 편집한 후 프로그램 단위를 컴파일해야 합니다. Connection Navigator에서 컴파일해야 하는 PL/SQL 객체를 마우스 오른쪽 버튼으로 누르고 Compile을 선택합니다. 또는 Ctrl+Shift+F9를 눌러 컴파일해도 됩니다.

프로그램 단위 실행



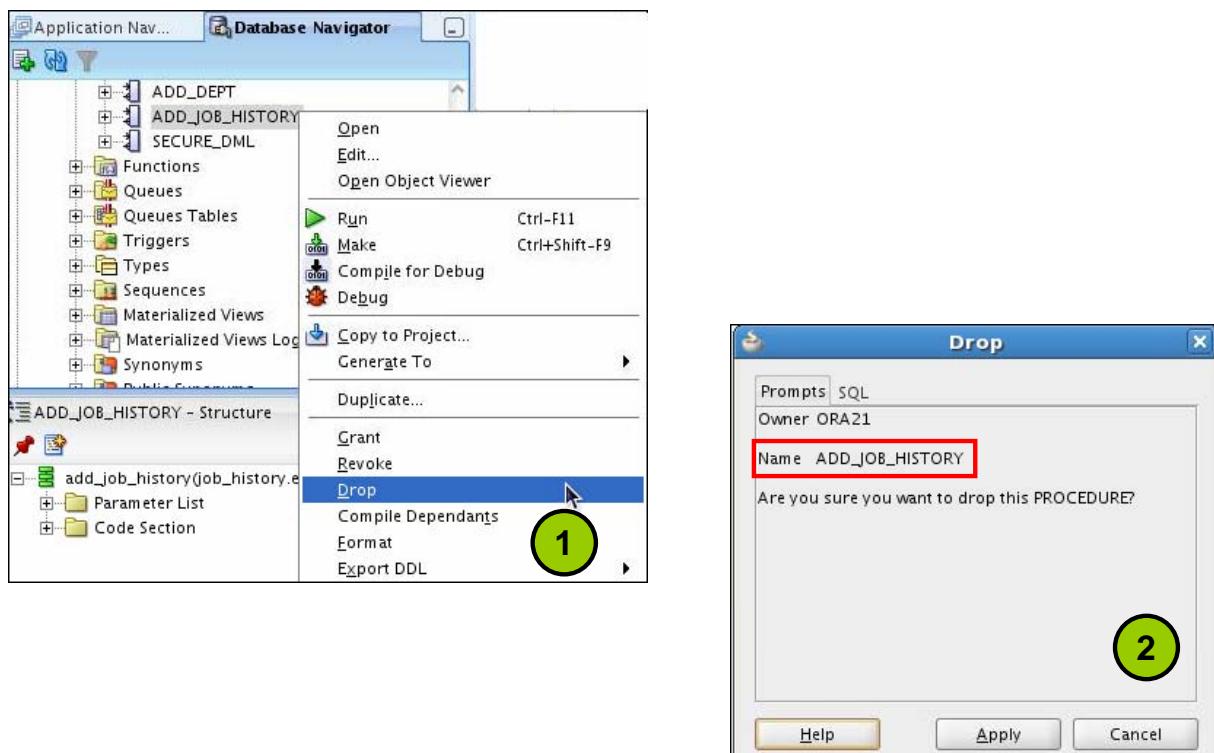
ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 실행

프로그램 단위를 실행하려면 객체를 마우스 오른쪽 버튼으로 누르고 Run을 선택합니다. Run PL/SQL 대화상자가 나타납니다. NULL 값을 프로그램 단위로 전달하기에 적절한 값으로 변경해야 할 수도 있습니다. 해당 값을 변경한 후 OK를 누릅니다. Message-Log window에 출력 결과가 표시됩니다.

프로그램 단위 삭제



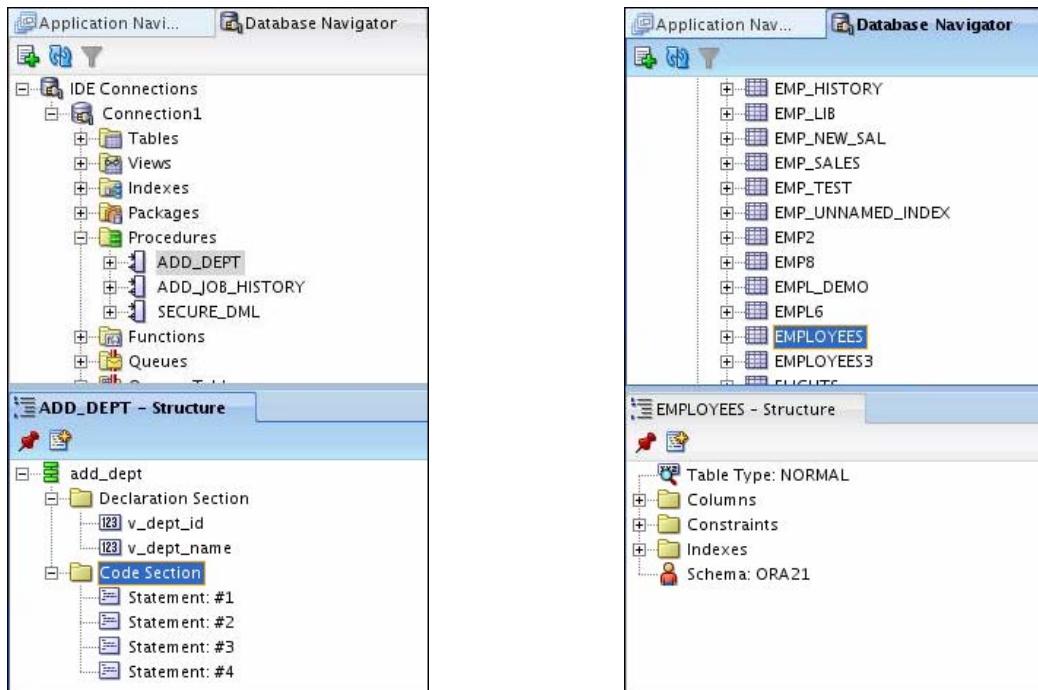
ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 삭제

프로그램 단위를 삭제하려면 객체를 마우스 오른쪽 버튼으로 누르고 Drop을 선택합니다. Drop Confirmation 대화상자가 나타납니다. Apply를 누릅니다. 객체가 데이터베이스에서 삭제됩니다.

Structure window



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Structure window

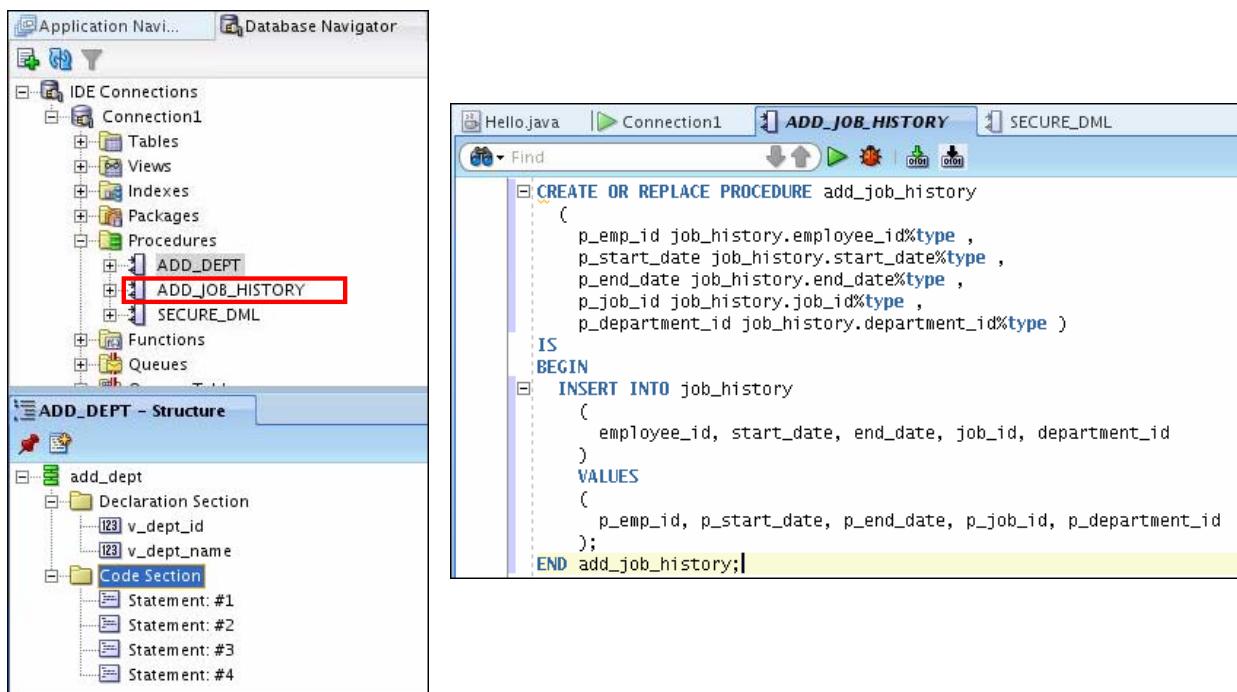
Structure window는 구조를 제공하는 데 관여하는 Navigator, Editor, Viewer, Property Inspector와 같은 window 중 활성 window에 현재 선택된 문서 데이터의 구조적 뷰를 제공합니다.

View > Structure window를 선택하여 Structure window를 봅니다.

Structure window에서 다양한 방식으로 문서 데이터를 볼 수 있습니다. 표시할 수 있는 구조는 문서 유형에 따라 결정됩니다. Java 파일의 경우 코드 구조, UI 구조 또는 UI 모델 데이터를 볼 수 있습니다. XML 파일의 경우 XML 구조, 디자인 구조 또는 UI 모델 데이터를 볼 수 있습니다.

Structure window는 현재의 활성 Editor와 연관되어 특정 뷰에 window 내용을 고정하지 않는 한 항상 동적으로 활성 window의 현재 선택 사항을 추적합니다. 현재 선택 사항이 Navigator의 노드일 때를 기본 편집기로 간주합니다. 현재 선택 사항의 구조에 대한 뷰를 변경하려면 다른 구조 탭을 누릅니다.

Editor window



ORACLE

Copyright © 2009, Oracle. All rights reserved.

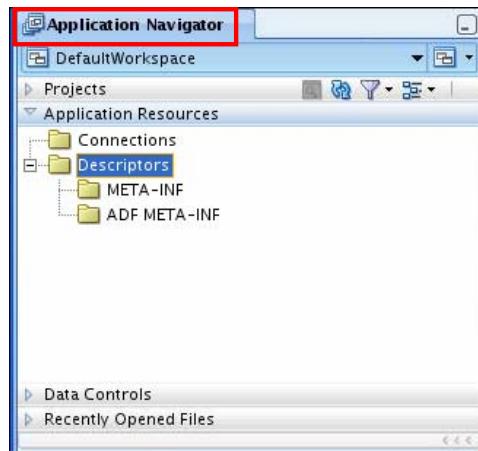
Editor window

프로그램 단위의 이름을 두 번 누르면 Editor window에 해당 프로그램 단위가 열립니다. 단일 편집기 window에서 프로젝트 파일을 모두 보거나, 한 파일을 여러 뷰로 열거나, 다양한 파일을 여러 뷰로 열 수 있습니다.

편집기 window의 상단에 있는 탭은 문서 탭입니다. 문서 탭을 누르면 해당 파일을 현재 편집기의 window 맨 앞으로 가져와서 해당 파일에 집중시킵니다.

주어진 파일에 대해 편집기 window의 하단에 있는 탭은 편집기 탭입니다. 편집기 탭을 선택하면 해당 편집기에 파일이 열립니다.

Application Navigator



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

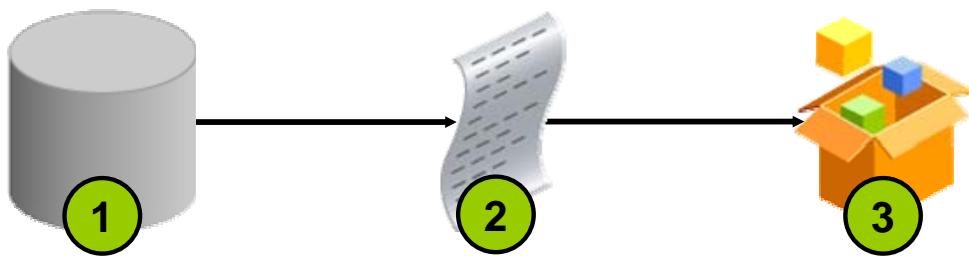
Application Navigator

Application Navigator는 응용 프로그램 및 이 응용 프로그램이 포함하는 데이터의 논리적 뷰를 제공합니다. Application Navigator는 다양한 확장을 플러그인하여 일관된 요약 방식으로 해당 데이터 및 메뉴를 구성하는 데 사용할 수 있는 Infrastructure를 제공합니다. Application Navigator는 Java 소스 파일과 같은 개별 파일을 포함할 수 있으며 이때 복잡한 데이터를 통합합니다. 엔티티 객체, UML(Unified Modeling Language) 다이어그램, EJB(Enterprise JavaBeans) 또는 웹 서비스와 같은 복잡한 데이터 유형이 Navigator에 단일 노드로 표시됩니다. 이러한 요약 노드를 구성하는 Raw File은 Structure window에 나타납니다.

Java 내장 프로시저 배치

Java 내장 프로시저를 배치하기 전에 다음 단계를 수행하십시오.

1. 데이터베이스 접속을 생성합니다.
2. 배치 프로파일을 생성합니다.
3. 객체를 배치합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Java 내장 프로시저 배치

Java 내장 프로시저에 대한 배치 프로파일을 생성한 다음 프로파일의 설정을 사용하여 JDeveloper에서 클래스를 배치하고 선택적으로 공용(public) 정적 메소드를 배치합니다.

데이터베이스가 배치되면 Deployment Profile Wizard 및 두 가지 오라클 데이터베이스 유ти리티에서 제공하는 정보가 사용됩니다.

- `loadjava`는 내장 프로시저를 포함하는 Java 클래스를 오라클 데이터베이스에 로드합니다.
- `publish`는 로드된 공용(public) 정적 메소드에 대해 PL/SQL 호출 특정 래퍼를 생성합니다. 게시(Publishing) 후에는 Java 메소드를 PL/SQL 함수 또는 프로시저로 호출할 수 있습니다.

PL/SQL에 Java 게시(publishing)

```

public class TrimLob
{
    public static void main (String args []) throws SQLException {
        Connection conn=null;
        if (System.getProperty("oracle.jserver.version") != null)
        {
            conn = DriverManager.getConnection("jdbc:default:connection:");
        }
        else
        {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:thin:scott/tiger");
        }
    }
}

```

```
[oracle@edrsr5p1-orcl ~]$ cd wkdir
[oracle@edrsr5p1-orcl wkdir]$ loadjava -u hr/hr TrimLob.java
[oracle@edrsr5p1-orcl wkdir]$
```

```

CREATE OR REPLACE PROCEDURE TRIMLOBPROC
as Language java
    name 'TrimLob.main(java.lang.String[])';
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에 Java 게시(publishing)

위 슬라이드는 Java 코드 및 이 Java 코드를 PL/SQL 프로시저에 게시(publish)하는 방법을 보여줍니다.

1. Java 루틴을 생성하고 컴파일합니다.
2. 슬라이드에 표시된 명령을 실행하여 TrimLob Java 루틴을 데이터베이스에 로드합니다.
3. Java 클래스 메소드를 참조하는 TrimLobProc PL/SQL 호출 지정을 생성하여 Java 클래스 메소드를 게시(publish)합니다. TrimLobProc PL/SQL 사양 내에서 Java 클래스 메소드의 이름과 파라미터를 식별합니다. 이 작업을 Java 클래스 메소드 "게시(publishing)"라고 합니다.

데이터베이스에서 Java 루틴을 실행하는 방법에 대한 자세한 내용은 다음 OBE:

http://www.oracle.com/technology/obe/hol08/11gR1_JDBC_Java/java_otn.htm을 참조하십시오.

JDeveloper 11g에 대해 자세히 배울 수 있는 방법

항목	웹 사이트
Oracle JDeveloper 제품 페이지	http://www.oracle.com/technology/products/jdev/index.html
Oracle JDeveloper 11g 학습서	http://www.oracle.com/technology/obe/obe11jdev/11/index.html
Oracle JDeveloper 11g 제품 설명서	http://www.oracle.com/technology/documentation/jdev.html
Oracle JDeveloper 11g 토의 포럼	http://forums.oracle.com/forums/forum.jspa?forumID=83



Copyright © 2009, Oracle. All rights reserved.

R

REF 커서

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 여러 행 query의 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 변수 사용

- 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 변수 사용

PL/SQL 내장 서브 프로그램과 다양한 클라이언트 간에 query 결과 집합을 전달하는 데 커서 변수를 사용합니다. PL/SQL도 해당 클라이언트도 결과 집합을 소유하지 않으면 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 블록이 있으면 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 PGA(프로그램 글로벌 영역)의 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];
```

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
```

참고: 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(weak가 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 예제와 같이 %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
```

파라미터로서 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 여러 행 query와 연관시키고 query를 실행하며 결과 집합을 식별하고 결과 집합의 첫번째 행을 가리키도록 커서의 위치를 지정합니다.
- FETCH 문은 여러 행 query의 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 여러 행 query를 처리하기 위해 OPEN-FOR, FETCH 및 CLOSE 문을 사용합니다. 먼저 여러 행 query에 "대한" 커서 변수를 "엽니다". 그런 다음 한 번에 하나씩 결과 집합에서 행을 "迨치(fetch)"합니다. 모든 행이 처리되면 커서 변수를 "닫습니다".

커서 변수 열기

OPEN-FOR 문은 커서 변수를 여러 행 query와 연관시키고 query를 실행하고 결과 집합을 식별하고 그 결과 집합의 첫번째 행을 가리키도록 커서의 위치를 지정한 다음 %ROWCOUNT 를 사용하여 처리된 행 수를 0으로 유지합니다. 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 여러 행 query를 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 매번 다시 열어야 합니다.

커서 변수에서 패치(fetch)

FETCH 문은 여러 행 query의 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 수행하여 emp_cv 커서 변수에서 MY_ENAME 및 MY_SAL 정의 변수로 행을 패치(fetch)합니다.

```
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 패치(fetch)는 동일한 결과 집합에서 다른 행을 읽어 들입니다. 닫히거나 전혀 열린 경우가 없는 커서 변수에서 패치(fetch)하려고 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};
```

이 예제에서 마지막 행이 처리되면 emp_cv 커서 변수를 닫습니다.

```
LOOP
```

```
    FETCH emp_cv INTO my_ename, my_sal;
```

```
    EXIT WHEN emp_cv%NOTFOUND;
```

```
    -- process row
```

```
END LOOP;
```

```
CLOSE emp_cv; -- close cursor variable
```

이미 닫혔거나 전혀 열린 경우가 없는 커서 변수를 닫으려고 하면 PL/SQL은 INVALID_CURSOR를 발생시킵니다.

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 여러 행 query의 결과 집합에서 레코드로 행을 패치(fetch)하는 방법을 보여줍니다. 먼저 REF CURSOR 유형 EmpCurTyp를 정의해야 합니다. 그런 다음 EmpcurTyp 유형의 커서 변수 emp_cv를 정의합니다. PL/SQL 블록의 실행 섹션에서 OPEN-FOR 문은 emp_cv 커서 변수를 sql_stmt 여러 행 query와 연관시킵니다. FETCH 문은 여러 행 query의 결과 집합에서 행을 반환하고 select-list 항목의 값을 INTO 절의 EMP_REC에 할당합니다. 마지막 행이 처리되면 emp_cv 커서 변수를 닫습니다.

부록 AP

추가 연습 및 해답

목차

개요	3
추가 연습 및 해답: 단원 1 및 2	4
연습 1: 선언 평가	4
연습 2: 표현식 평가	4
해답 1: 선언 평가	5
해답 2: 표현식 평가	5
추가 연습 및 해답: 단원 3	6
연습 3: 실행문 평가	6
해답 3: 실행문 평가	7
단원 4 의 추가 연습 및 해답	8
연습 4-1: Oracle 서버와 상호 작용	8
연습 4-2: Oracle 서버와 상호 작용	9
해답 4-1: Oracle 서버와 상호 작용	9
해답 4-2: Oracle 서버와 상호 작용	11
단원 5 의 추가 연습 및 해답	12
연습 5-1: 제어 구조 작성	12
연습 5-2: 제어 구조 작성	12
해답 5-1: 제어 구조 작성	13
해답 5-2: 제어 구조 작성	14
단원 6 및 7 의 추가 연습 및 해답	15
연습 6/7-1: 명시적 커서를 사용하여 데이터 패치	15
연습 6/7-2: 연관 배열 및 명시적 커서 사용	16
해답 6/7-1: 명시적 커서를 사용하여 데이터 패치	17
해답 6/7-2: 연관 배열 및 명시적 커서 사용	18
단원 8 의 추가 연습 및 해답	20
연습 8-1: 예외 처리	20
해답 8-1: 예외 처리	21

개요

이 추가 연습은 *Oracle Database 11g: PL/SQL Fundamentals* 과정의 보충 자료로 제공됩니다. 이 연습에서는 본 과정에서 설명한 개념을 활용합니다.

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. 추가 연습의 이 부분에 사용되는 테이블은 employees, jobs, job_history 및 departments 입니다.

추가 연습 및 해답: 단원 1 및 2

이 필기형 연습은 변수 선언 및 실행문 작성에 대한 추가 연습에 사용됩니다.

연습 1: 선언 평가

다음에 나열된 선언을 검토하여 그 중 잘못된 선언을 판별하고 그 이유를 설명합니다.

- 1)

```
DECLARE  
    name,dept      VARCHAR2(14);
```
- 2)

```
DECLARE  
    test          NUMBER( 5 );
```
- 3)

```
DECLARE  
    MAXSALARY    NUMBER( 7 , 2 ) = 5000;
```
- 4)

```
DECLARE  
    JOINDATE     BOOLEAN := SYSDATE;
```

연습 2: 표현식 평가

다음 각 할당에서 결과 표현식의 데이터 유형을 파악합니다.

- 1) `email := firstname || to_char(empno);`
- 2) `confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');`
- 3) `sal := (1000*12) + 500`
- 4) `test := FALSE;`
- 5) `temp := templ < (temp2/ 3);`
- 6) `var := sysdate;`

해답 1: 선언 평가

다음에 나열된 선언을 검토하여 그 중 잘못된 선언을 판별하고 그 이유를 설명합니다.

1) DECLARE

```
name ,dept      VARCHAR2(14);
```

각 선언마다 하나의 식별자만 허용되기 때문에 잘못된 선언입니다.

2) DECLARE

```
test          NUMBER( 5 );
```

올바른 선언입니다.

3) DECLARE

```
MAXSALARY     NUMBER( 7 , 2 ) = 5000;
```

할당 연산자가 틀리기 때문에 잘못된 선언입니다. 연산자는 =여야 합니다.

4) DECLARE

```
JOINDATE      BOOLEAN := SYSDATE;
```

데이터 유형이 일치하지 않기 때문에 잘못된 선언입니다. Boolean 데이터 유형에는 날짜 값을 할당할 수 없습니다. Date 데이터 유형을 사용해야 합니다.

해답 2: 표현식 평가

다음 각 할당에서 결과 표현식의 데이터 유형을 파악합니다.

1) email := firstname || to_char(empno);

Character string

2) confirm := to_date('20-JAN-1999', 'DD-MON-YYYY');

Date

3) sal := (1000*12) + 500

Number

4) test := FALSE;

Boolean

5) temp := temp1 < (temp2/ 3);

Boolean

6) var := sysdate;

Date

추가 연습 및 해답: 단원 3

연습 3: 실행문 평가

이 필기형 연습에서는 PL/SQL 블록을 평가한 다음 범위 지정 규칙에 따라 데이터 유형과 각 변수의 값을 결정하여 다음 문제에 답합니다.

```
DECLARE
    v_custid      NUMBER(4) := 1600;
    v_custname    VARCHAR2(300) := 'Women Sports Club';
    v_new_custid  NUMBER(3) := 500;
BEGIN
    DECLARE
        v_custid      NUMBER(4) := 0;
        v_custname    VARCHAR2(300) := 'Shape up Sports Club';
        v_new_custid  NUMBER(3) := 300;
        v_new_custname VARCHAR2(300) := 'Jansports Club';
    BEGIN
        v_custid := v_new_custid;
        v_custname := v_custname || ' ' || v_new_custname;
1   →
        END;
        v_custid := (v_custid *12) / 10;
2   →
        END;
```

위의 PL/SQL 블록을 평가하고 범위 지정 규칙에 따라 다음 각 변수의 값과 데이터 유형을 결정합니다.

- 1) 위치 1의 v_custid:
- 2) 위치 1의 v_custname:
- 3) 위치 1의 v_new_custid:
- 4) 위치 1의 v_new_custname:
- 5) 위치 2의 v_custid:
- 6) 위치 2의 v_custname:

해답 3: 실행문 평가

다음 PL/SQL 블록을 평가합니다. 그런 다음 범위 지정 규칙에 따라 다음 각 변수의 데이터 유형과 값을 결정하여 다음 문제에 답합니다.

```

DECLARE
    v_custid      NUMBER(4) := 1600;
    v_custname   VARCHAR2(300) := 'Women Sports Club';
    v_new_custid  NUMBER(3) := 500;
BEGIN
    DECLARE
        v_custid      NUMBER(4) := 0;
        v_custname   VARCHAR2(300) := 'Shape up Sports Club';
        v_new_custid NUMBER(3) := 300;
        v_new_custname VARCHAR2(300) := 'Jansports Club';
    BEGIN
        v_custid := v_new_custid;
        v_custname := v_custname || ' ' || v_new_custname;
1   →
        END;
        v_custid := (v_custid *12) / 10;
2   →
        END;
    
```

위의 PL/SQL 블록을 평가하고 범위 지정 규칙에 따라 다음 각 변수의 값과 데이터 유형을 결정합니다.

1) 위치 1의 v_custid:

300이고, 데이터 유형은 **NUMBER**입니다.

2) 위치 1의 v_custname:

Shape up Sports Club Jansports Club이고, 데이터 유형은 **VARCHAR2**입니다.

3) 위치 1의 v_new_custid:

500이고, 데이터 유형은 **NUMBER(또는 INTEGER)**입니다.

4) 위치 1의 v_new_custname:

Jansports Club이고, 데이터 유형은 **VARCHAR2**입니다.

5) 위치 2의 v_custid:

1920이고, 데이터 유형은 **NUMBER**입니다.

6) 위치 2의 v_custname:

Women Sports Club이고, 데이터 유형은 **VARCHAR2**입니다.

단원 4 의 추가 연습 및 해답

연습 4-1: Oracle 서버와 상호 작용

이 연습에는 결과를 저장할 임시 테이블(Temporary Table)이 필요합니다.

- 여기에서 설명한 테이블을 생성하는 lab_ap_04.sql 스크립트를 실행합니다.

Column Name	NUM_STORE	CHAR_STORE	DATE_STORE
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Data Type	Number	VARCHAR2	Date
Length	7,2	35	

- 다음을 수행하는 PL/SQL 블록을 작성합니다.

- 변수를 두 개 선언하고 다음 값을 할당합니다.

변수	데이터 유형	내용
V_MESSAGE	VARCHAR2 (35)	This is my first PL/SQL program
V_DATE_WRITTEN	DATE	Current date

- 이 변수의 값을 해당 TEMP 테이블 열에 저장합니다.

- TEMP 테이블을 query하여 결과를 확인합니다. 출력 결과는 다음과 같아 나타나야 합니다.

```
anonymous block completed
+-----+-----+-----+
| NUM_STORE | CHAR_STORE | DATE_STORE |
+-----+-----+-----+
| This is my first PLSQL Program | 15-JUL-09 |
+-----+-----+-----+
1 rows selected
```

연습 4-2: Oracle 서버와 상호 작용

이 연습에서는 employees 테이블의 데이터를 사용합니다.

- 1) 지정된 부서에 근무하는 사원 수를 결정하는 PL/SQL 블록을 작성합니다. PL/SQL 블록은 다음을 수행해야 합니다.
 - 치환 변수를 사용하여 부서 번호를 저장합니다.
 - 지정된 부서에서 근무하는 사원 수를 출력합니다.
- 2) 블록이 실행되면 치환 변수 window 가 나타납니다. 적합한 부서 번호를 입력하고 OK 를 누릅니다. 출력 결과는 다음과 유사해야 합니다.

```

anonymous block completed
6 employee(s) work for department number 30
  
```

해답 4-1: Oracle 서버와 상호 작용

이 연습에는 결과를 저장할 임시 테이블(Temporary Table)이 필요합니다.

- 1) 여기에서 설명한 테이블을 생성하는 lab_ap_04.sql 스크립트를 실행합니다.

Column Name	NUM_STORE	CHAR_STORE	DATE_STORE
Key Type			
Nulls/Unique			
FK Table			
FK Column			
Data Type	Number	VARCHAR2	Date
Length	7,2	35	

2) 다음을 수행하는 PL/SQL 블록을 작성합니다.

c) 변수를 두 개 선언하고 다음 값을 할당합니다.

변수	데이터 유형	내용
V_MESSAGE	VARCHAR2(35)	This is my first PL/SQL program
V_DATE_WRITTEN	DATE	Current date

d) 이 변수의 값을 해당 TEMP 테이블 열에 저장합니다.

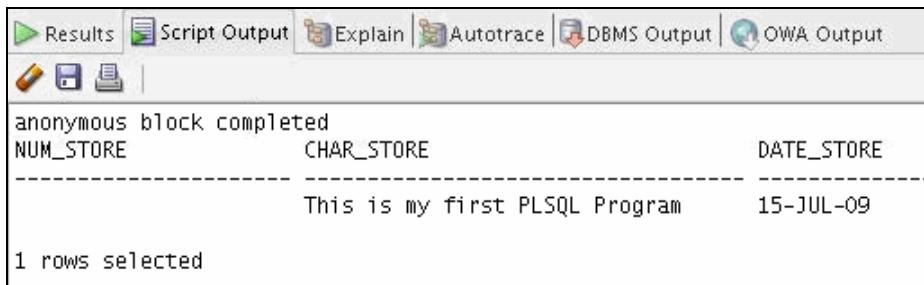
```

DECLARE
    V_MESSAGE VARCHAR2(35);
    V_DATE_WRITTEN DATE;
BEGIN
    V_MESSAGE := 'This is my first PLSQL Program';
    V_DATE_WRITTEN := SYSDATE;
    INSERT INTO temp(CHAR_STORE,DATE_STORE)
        VALUES (V_MESSAGE,V_DATE_WRITTEN);
END;
/

```

3) TEMP 테이블을 query하여 결과를 확인합니다. 출력 결과는 다음과 유사해야 합니다.

```
SELECT * FROM TEMP;
```



The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output window displays the results of the executed anonymous block:

```

anonymous block completed
NUM_STORE      CHAR_STORE          DATE_STORE
-----          -----
This is my first PLSQL Program   15-JUL-09
1 rows selected

```

해답 4-2: Oracle 서버와 상호 작용

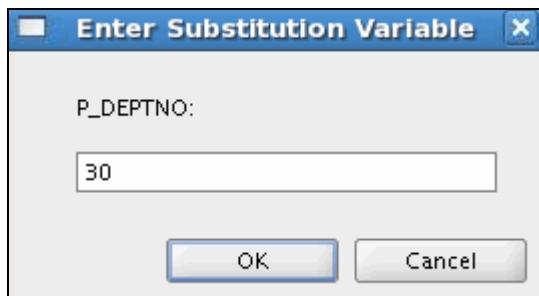
이 연습에서는 employees 테이블의 데이터를 사용합니다.

- 1) 지정된 부서에 근무하는 사원 수를 결정하는 PL/SQL 블록을 작성합니다. PL/SQL 블록은 다음을 수행해야 합니다.

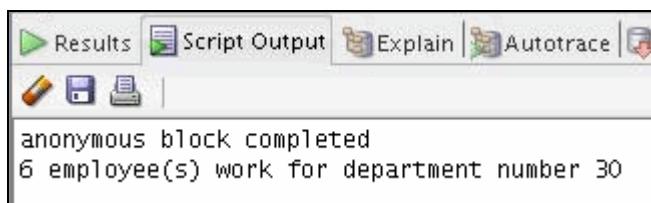
- 치환 변수를 사용하여 부서 번호를 저장합니다.
- 지정된 부서에서 근무하는 사원 수를 출력합니다.

```
SET SERVEROUTPUT ON;
DECLARE
    V_HOWMANY NUMBER(3);
    V_DEPTNO DEPARTMENTS.department_id%TYPE := &P_DEPTNO;
BEGIN
    SELECT COUNT(*) INTO V_HOWMANY FROM employees
    WHERE department_id = V_DEPTNO;
    DBMS_OUTPUT.PUT_LINE (V_HOWMANY || ' employee(s)'
        work for department number ' || V_DEPTNO);
END;
/
```

- 2) 블록이 실행되면 치환 변수 window 가 나타납니다. 적합한 부서 번호를 입력하고 OK 를 누릅니다.



출력 결과는 다음과 유사해야 합니다.



단원 5의 추가 연습 및 해답

이 연습에서는 제어 구조를 사용하여 프로그램 흐름의 논리를 지정합니다.

연습 5-1: 제어 구조 작성

- 연도 입력을 받아들이는 PL/SQL 블록을 작성하고 연도가 윤년인지 여부를 확인합니다. **힌트:** 윤년은 4로 정확히 나누어 떨어지며 100의 배수는 아닙니다. 그렇지만 400의 배수는 윤년입니다.
- 다음 표를 사용하여 해답을 테스트합니다. 예를 들어, 입력한 연도가 1990이면 "1990 is not a leap year"가 출력되어야 합니다.

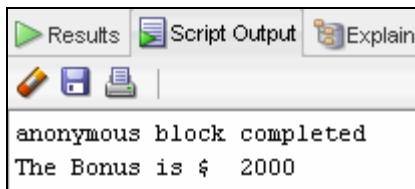
1990	Not a leap year
2000	Leap year
1996	Leap year
1886	Not a leap year
1992	Leap year
1824	Leap year

연습 5-2: 제어 구조 작성

- 사원의 월급을 치환 변수에 저장하는 PL/SQL 블록을 작성합니다. PL/SQL 블록은 다음을 수행해야 합니다.
 - 급여에 12를 곱하여 연봉을 계산합니다.
 - 다음 표와 같이 상여금을 계산합니다.

연봉	상여금
>= 20,000	2,000
19,999–10,000	1,000
<= 9,999	500

- 다음과 같은 형식으로 상여금 액수를 Script Output window에 표시합니다.



- 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

월급	상여금
3000	2000
1200	1000
800	500

해답 5-1: 제어 구조 작성

- 1) 연도 입력을 받아들이는 PL/SQL 블록을 작성하고 연도가 윤년인지 여부를 확인합니다. 힌트: 윤년은 4로 정확히 나누어 떨어지며 100의 배수는 아닙니다. 그렇지만 400의 배수는 윤년입니다.

```
SET SERVEROUTPUT ON;
DECLARE
    v_YEAR NUMBER(4) := &P_YEAR;
    v_REMAINDER1 NUMBER(5,2);
    v_REMAINDER2 NUMBER(5,2);
    v_REMAINDER3 NUMBER(5,2);
BEGIN
    v_REMAINDER1 := MOD(v_YEAR,4);
    v_REMAINDER2 := MOD(v_YEAR,100);
    v_REMAINDER3 := MOD(v_YEAR,400);
    IF ((v_REMAINDER1 = 0 AND v_REMAINDER2 <> 0 ) OR
        v_REMAINDER3 = 0) THEN
        DBMS_OUTPUT.PUT_LINE(v_YEAR || ' is a leap year');
    ELSE
        DBMS_OUTPUT.PUT_LINE(v_YEAR || ' is not a leap
year');
    END IF;
END;
/
```

- 2) 다음 표를 사용하여 해답을 테스트합니다. 예를 들어, 입력한 연도가 1990이면 "1990 is not a leap year"가 출력되어야 합니다.

1990	Not a leap year
2000	Leap year
1996	Leap year
1886	Not a leap year
1992	Leap year
1824	Leap year

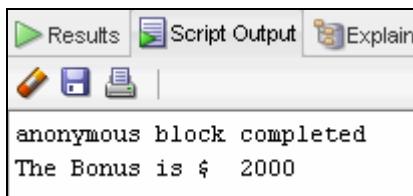
해답 5-2: 제어 구조 작성

1) 사원의 월급을 치환 변수에 저장하는 PL/SQL 블록을 작성합니다. PL/SQL 블록은 다음을 수행해야 합니다.

- 급여에 12를 곱하여 연봉을 계산합니다.
- 다음 표와 같이 상여금을 계산합니다.

연봉	상여금
>= 20,000	2,000
19,999–10,000	1,000
<= 9,999	500

- 다음과 같은 형식으로 상여금 액수를 Script Output window에 표시합니다.



```

SET SERVEROUTPUT ON;
DECLARE
    V_SAL          NUMBER(7,2) := &M_SALARY;
    V_BONUS        NUMBER(7,2);
    V_ANN_SALARY  NUMBER(15,2);
BEGIN
    V_ANN_SALARY := V_SAL * 12;
    IF V_ANN_SALARY >= 20000 THEN
        V_BONUS := 2000;
    ELSIF V_ANN_SALARY <= 19999 AND V_ANN_SALARY >=10000 THEN
        V_BONUS := 1000;
    ELSE
        V_BONUS := 500;
    END IF;
    DBMS_OUTPUT.PUT_LINE ('The Bonus is $ ' ||
                          TO_CHAR(V_BONUS));
END;
/

```

2) 다음과 같은 내용으로 PL/SQL 블록을 테스트합니다.

월급	상여금
3000	2000
1200	1000
800	500

단원 6 및 7의 추가 연습 및 해답

다음 연습에서는 연관 배열(단원 6에서 다룸)과 명시적 커서(단원 7에서 다룸) 사용을 연습합니다. 첫번째 연습에서는 명시적 커서를 사용하여 데이터를 패치(fetch)합니다. 두번째 연습에서는 연관 배열의 사용을 명시적 커서와 결합하여 특정 기준과 일치하는 데이터를 출력합니다.

연습 6/7-1: 명시적 커서를 사용하여 데이터 패치

이 연습에서는 다음을 수행하는 PL/SQL 블록을 생성합니다.

- 1) EMPLOYEES 테이블에서 사원의 성, 급여 및 채용 날짜를 선택하도록 EMP_CUR이라는 커서를 선언합니다.
- 2) 커서에서 각 행을 처리한 다음, 급여가 15,000 보다 많고 채용 날짜가 01-FEB-1988 이후인 경우 다음 예제 출력에 표시된 형식으로 사원 이름, 급여 및 채용 날짜를 표시합니다.

The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output window displays the following text:

```
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
```

연습 6/7-2: 연관 배열 및 명시적 커서 사용

이 연습에서는 EMPLOYEES 테이블에서 EMPLOYEE_ID 가 115 보다 작은 각 사원의 성과 부서 ID를 검색하고 출력하는 PL/SQL 블록을 생성합니다.

PL/SQL 블록에서 이전 연습에 사용한 OPEN / FETCH / CLOSE 커서 메소드 대신 커서 FOR 루프 전략을 사용합니다.

1) 선언 섹션에서 다음을 수행합니다.

- 연관 배열을 두 개 만듭니다. 이들 배열의 Unique Key 열은 모두 BINARY_INTEGER 데이터 유형이어야 합니다. 한 배열은 사원의 성을 보유하고 다른 한 배열은 부서 ID를 보유합니다.
- ID가 115 보다 작은 사원의 성과 부서 ID를 선택하는 커서를 선언합니다.
- 실행 섹션에서 사용할 적절한 카운터 변수를 선언합니다.

2) 실행 섹션에서 커서 FOR 루프(7 단원에서 다룸)를 사용하여 커서 값에 액세스하고 이 값을 적절한 연관 배열에 할당한 다음 배열에서 출력합니다. 올바른 출력은 다음 형식과 같이 15 행을 반환해야 합니다.

```

anonymous block completed
Employee: King is in department number: 90
Employee: Kochhar is in department number: 90
Employee: De Haan is in department number: 80

```

해답 6/7-1: 명시적 커서를 사용하여 데이터 패치

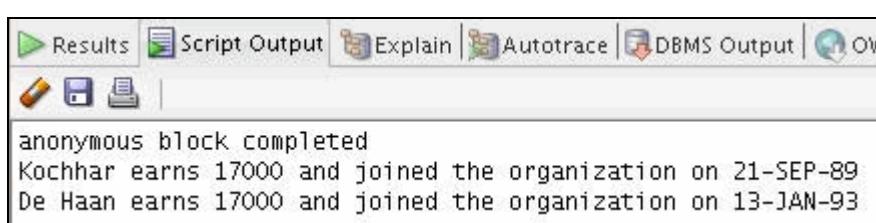
이 연습에서는 다음을 수행하는 PL/SQL 블록을 생성합니다.

- EMPLOYEES 테이블에서 사원의 성, 급여 및 채용 날짜를 선택하도록 EMP_CUR이라는 커서를 선언합니다.

```
SET SERVEROUTPUT ON;
DECLARE
    CURSOR C_EMP_CUR IS
        SELECT last_name,salary,hire_date FROM EMPLOYEES;
    V_ENAME VARCHAR2(25);
    V_SAL    NUMBER(7,2);
    V_HIREDATE DATE;
```

- 커서에서 각 행을 처리한 다음, 급여가 15,000 보다 많고 채용 날짜가 01-FEB-1988 이후인 경우 다음 예제 출력에 표시된 형식으로 사원 이름, 급여 및 채용 날짜를 표시합니다.

```
BEGIN
    OPEN C_EMP_CUR;
    FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
    WHILE C_EMP_CUR%FOUND
    LOOP
        IF V_SAL > 15000 AND V_HIREDATE >=
            TO_DATE('01-FEB-1988','DD-MON-YYYY') THEN
            DBMS_OUTPUT.PUT_LINE (V_ENAME || ' earns '
            || TO_CHAR(V_SAL)|| ' and joined the organization on '
            || TO_DATE(V_HIREDATE,'DD-Mon-YYYY'));
        END IF;
        FETCH C_EMP_CUR INTO V_ENAME,V_SAL,V_HIREDATE;
    END LOOP;
    CLOSE C_EMP_CUR;
END;
/
```



The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output window displays the following text:

```
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
```

해답 6/7-2: 연관 배열 및 명시적 커서 사용

이 연습에서는 EMPLOYEES 테이블에서 EMPLOYEE_ID 가 115 보다 작은 각 사원의 성과 부서 ID를 검색하고 출력하는 PL/SQL 블록을 생성합니다.

PL/SQL 블록에서 이전 연습에 사용한 OPEN / FETCH / CLOSE 커서 메소드 대신 커서 FOR 루프 전략을 사용합니다.

1. 선언 섹션에서 다음을 수행합니다.

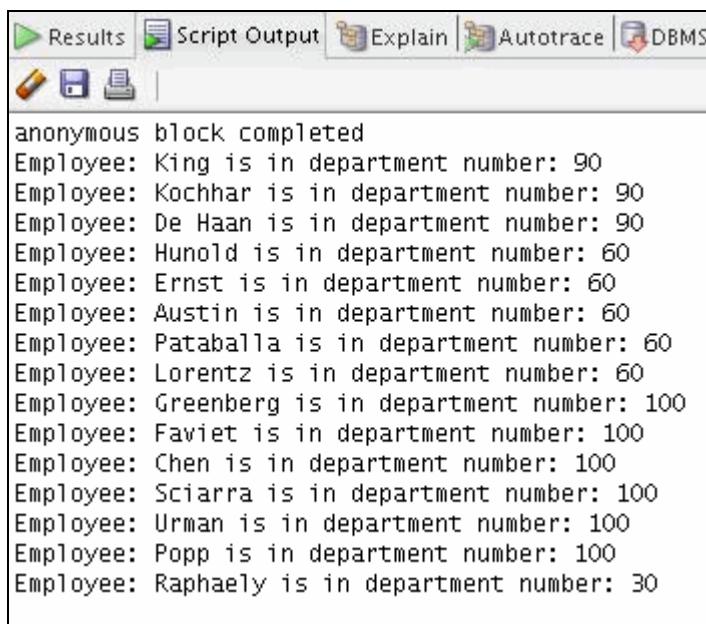
- 연관 배열을 두 개 만듭니다. 이들 배열의 Unique Key 열은 모두 BINARY_INTEGER 데이터 유형이어야 합니다. 한 배열은 사원의 성을 보유하고 다른 한 배열은 부서 ID를 보유합니다.
- 실행 섹션에서 사용할 카운터 변수를 선언합니다.
- ID 가 115 보다 작은 사원의 성과 부서 ID를 선택하는 커서를 선언합니다.

```
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;
```

2. 실행 섹션에서 커서 FOR 루프(7 단원에서 다룸)를 사용하여 커서 값에 액세스하고 이 값을 적절한 연관 배열에 할당한 다음 배열에서 출력합니다.

```
BEGIN
    FOR emprec in Namedept
    LOOP
        i := i +1;
        Tename(i) := emprec.last_name;
        Tdept(i) := emprec.department_id;
        DBMS_OUTPUT.PUT_LINE ('Employee: ' || Tename(i) ||
            ' is in department number: ' || Tdept(i));
    END LOOP;
END;
/
```

올바른 출력은 다음과 같이 15 행을 반환해야 합니다.



The screenshot shows a software interface with a toolbar at the top containing icons for Results, Script Output, Explain, Autotrace, and DBMS. Below the toolbar is a menu bar with options like File, Edit, View, Tools, Database, Help, and a Log Out icon. The main area displays the output of an anonymous block. The output consists of 15 lines, each starting with 'Employee:' followed by an employee's name and their department number. The names listed are King, Kochhar, De Haan, Hunold, Ernst, Austin, Pataballa, Lorentz, Greenberg, Faviet, Chen, Sciarra, Urman, Popp, and Raphaely. The department numbers for these employees are 90, 90, 90, 60, 60, 60, 60, 60, 100, 100, 100, 100, 100, 100, 100, and 30 respectively.

```
anonymous block completed
Employee: King is in department number: 90
Employee: Kochhar is in department number: 90
Employee: De Haan is in department number: 90
Employee: Hunold is in department number: 60
Employee: Ernst is in department number: 60
Employee: Austin is in department number: 60
Employee: Pataballa is in department number: 60
Employee: Lorentz is in department number: 60
Employee: Greenberg is in department number: 100
Employee: Faviet is in department number: 100
Employee: Chen is in department number: 100
Employee: Sciarra is in department number: 100
Employee: Urman is in department number: 100
Employee: Popp is in department number: 100
Employee: Raphaely is in department number: 30
```

단원 8의 추가 연습 및 해답

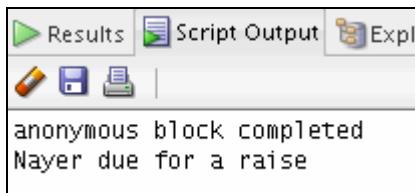
연습 8-1: 예외 처리

이 연습에서는 먼저 몇 가지 결과를 저장하는 테이블을 생성해야 합니다. 테이블을 자동으로 생성하는 `lab_ap_08.sql` 스크립트를 실행합니다. 스크립트는 다음과 같아야 합니다.

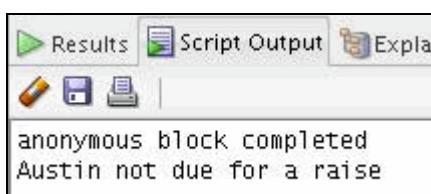
```
CREATE TABLE analysis
  (ename Varchar2(20), years Number(2), sal Number(8,2)
  );
```

이 연습에서는 다음과 같이 예외를 처리하는 PL/SQL 블록을 작성합니다.

- 1) 사원의 성, 급여 및 채용 날짜에 대한 변수를 선언합니다. 사원의 성에는 치환 변수를 사용합니다. 그런 다음 `employees` 테이블에서 지정된 사원의 `last_name`, `salary` 및 `hire_date`를 query 합니다.
- 2) 사원이 5년 이상 조직에 근무했고 사원의 급여가 3,500 보다 작은 경우 예외가 발생합니다. 예외 처리기에서 다음을 수행합니다.
 - 다음과 같이 사원의 성과 "due for a raise" 메시지로 구성된 정보를 출력합니다.



- 성, 근무 연수 및 급여를 `analysis` 테이블에 삽입합니다.
- 3) 예외가 없는 경우 다음과 같이 사원의 성과 "not due for a raise" 메시지를 출력합니다.



- 4) `analysis` 테이블을 query 하여 결과를 확인합니다. 다음 테스트를 사용하여 PL/SQL 블록을 테스트합니다.

LAST_NAME	메시지
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

해답 8-1: 예외 처리

이 연습에서는 먼저 몇 가지 결과를 저장하는 테이블을 생성해야 합니다. 테이블을 자동으로 생성하는 lab_ap_08.sql 스크립트를 실행합니다. 스크립트는 다음과 유사해야 합니다.

```
CREATE TABLE analysis
  (ename Varchar2(20), years Number(2), sal Number(8,2)
  );
```

이 연습에서는 다음과 같이 예외를 처리하는 PL/SQL 블록을 작성합니다.

- 1) 사원의 성, 급여 및 채용 날짜에 대한 변수를 선언합니다. 사원의 성에는 치환 변수를 사용합니다. 그런 다음 employees 테이블에서 지정된 사원의 last_name, salary 및 hire_date 를 query 합니다.
- 2) 사원이 5년 이상 조직에 근무했고 사원의 급여가 3,500 보다 작은 경우 예외가 발생합니다. 예외 처리기에서 다음을 수행합니다.
 - 사원의 성과 "due for a raise" 메시지로 구성된 정보를 출력합니다.
 - 사원의 이름, 근무 연수 및 급여를 analysis 테이블에 삽입합니다.
- 3) 예외가 없는 경우 사원의 성과 "not due for a raise" 메시지를 출력합니다.

```
SET SERVEROUTPUT ON;
DECLARE
  E_DUE_FOR_RAISE EXCEPTION;
  V_HIREDATE EMPLOYEES.HIRE_DATE%TYPE;
  V_ENAME EMPLOYEES.LAST_NAME%TYPE := INITCAP(' & B_ENAME');
  V_SAL EMPLOYEES.SALARY%TYPE;
  V_YEARS NUMBER(2);
BEGIN
  SELECT LAST_NAME, SALARY, HIRE_DATE
  INTO V_ENAME, V_SAL, V_HIREDATE
  FROM employees WHERE last_name = V_ENAME;
  V_YEARS := MONTHS_BETWEEN(SYSDATE, V_HIREDATE)/12;
  IF V_SAL < 3500 AND V_YEARS > 5 THEN
    RAISE E_DUE_FOR_RAISE;
  ELSE
    DBMS_OUTPUT.PUT_LINE (' not due for a raise');
  END IF;
EXCEPTION
  WHEN E_DUE_FOR_RAISE THEN
    BEGIN
      DBMS_OUTPUT.PUT_LINE (V_NAME || ' due for a raise');
      INSERT INTO ANALYSIS(ENAME, YEARS, SAL)
        VALUES (V_ENAME, V_YEARS, V_SAL);
    END;
END;
/
```

- 4) analysis 테이블을 query 하여 결과를 확인합니다. 다음 테스트를 사용하여 PL/SQL 블록을 테스트합니다.

LAST_NAME	메시지
Austin	Not due for a raise
Nayer	Due for a raise
Fripp	Not due for a raise
Khoo	Due for a raise

```
SELECT * FROM analysis;
```