

Oracle Database 11g: Develop PL/SQL Program Units(한글판)

학생용 • 볼륨 1

D49986KR20

Edition 2.0

2009년 12월

D64312

ORACLE®

저자

Lauran Serhal

기술 제공자 및 검토자

Anjulapponni
 Azhangulekshmi
 Christian Bauwens
 Christoph Burandt
 Zarko Cesljas
 Yanti Chang
 Salome Clement
 Laszlo Czinkoczki
 Ingrid DelaHaye
 Steve Friedberg
 Laura Garza
 Joel Goodman
 Nancy Greenberg
 Manish Pawar
 Brian Pottle
 Helen Robertson
 Tulika Srivastava
 Ted Witiuk

편집자

Arijit Ghosh
 Raj Kumar

발행인

Pavithran Adka
 Sheryl Domingue

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.

목차

I 소개

- 단원 목표 I-2
- 단원 내용 I-3
- 과정 목표 I-4
- 권장 과정 일정 I-5
- 단원 내용 I-7
- 이 과정에서 사용되는 HR (Human Resources) 스키마 I-8
- 클래스 계정 정보 I-9
- 본 과정에 사용되는 부록 I-10
- PL/SQL 개발 환경 I-11
- Oracle SQL Developer 란? I-12
- SQL*Plus에서 PL/SQL 코딩 I-13
- Oracle JDeveloper에서 PL/SQL 코딩 I-14
- PL/SQL 블록의 출력 활성화 I-15
- 단원 내용 I-16
- Oracle 11g SQL 및 PL/SQL 설명서 I-17
- 추가 자료 I-18
- 요약 I-19
- 연습 | 개요: 시작하기 I-20

1 프로시저 생성

- 목표 1-2
- 단원 내용 1-3
- 모듈화된 서브 프로그램 설계 생성 1-4
- 계층화된 서브 프로그램 설계 생성 1-5
- PL/SQL 블록을 사용한 개발 모듈화 1-6
- 익명 블록: 개요 1-7
- PL/SQL 런타임 구조 1-8
- PL/SQL 서브 프로그램이란? 1-9
- PL/SQL 서브 프로그램 사용 시 이점 1-10
- 익명 블록과 서브 프로그램의 차이 1-11
- 단원 내용 1-12
- 프로시저란? 1-13
- 프로시저 생성: 개요 1-14
- SQL CREATE OR REPLACE 문으로 프로시저 생성 1-15
- SQL Developer를 사용하여 프로시저 생성 1-16
- SQL Developer에서 프로시저 컴파일 및 컴파일 오류 표시 1-17

SQL Developer에서 컴파일 오류 해결	1-18
이 과정에서 사용하는 PL/SQL 구조의 이름 지정 규칙	1-19
파라미터 및 파라미터 모드란?	1-20
형식 파라미터 및 실제 파라미터	1-21
프로시저 파라미터 모드	1-22
파라미터 모드 비교	1-23
IN 파라미터 모드 사용: 예제	1-24
OUT 파라미터 모드 사용: 예제	1-25
IN OUT 파라미터 모드 사용: 예제	1-26
OUT 파라미터 보기: DBMS_OUTPUT.PUT_LINE 서브 루틴 사용	1-27
OUT 파라미터 보기: SQL*Plus 호스트 변수 사용	1-28
실제 파라미터 전달 시 사용 가능한 표기법	1-29
실제 파라미터 전달: add_dept 프로시저 생성	1-30
실제 파라미터 전달: 예제	1-31
파라미터에 DEFAULT 옵션 사용	1-32
프로시저 호출	1-34
SQL Developer를 사용하여 프로시저 호출	1-35
단원 내용	1-36
처리된 예외	1-37
처리된 예외: 예제	1-38
처리되지 않은 예외	1-39
처리되지 않은 예외: 예제	1-40
프로시저 제거: DROP SQL 문 또는 SQL Developer 사용	1-41
데이터 덕셔너리 뷰를 사용하여 프로시저 정보 보기	1-42
SQL Developer를 사용하여 프로시저 정보 보기	1-43
퀴즈	1-44
요약	1-45
연습 1 개요: 프로시저 생성, 컴파일 및 호출	1-46

2 함수 생성 및 서브 프로그램 디버그

목표	2-2
단원 내용	2-3
내장 함수 개요	2-4
함수 생성	2-5
프로시저와 함수의 차이	2-6
함수 생성 및 실행: 개요	2-7
CREATE FUNCTION 문을 사용하여 내장 함수 생성 및 호출: 예제	2-8
서로 다른 방법을 사용하여 함수 실행	2-9
SQL Developer를 사용하여 함수 생성 및 컴파일	2-11
SQL Developer를 사용하여 함수 실행	2-12
SQL 문에서 유저 정의 함수를 사용하는 경우의 이점	2-13
SQL 표현식에 함수 사용: 예제	2-14

SQL 문에서 유저 정의 함수 호출	2-15
SQL 표현식에서 함수를 호출할 때의 제한 사항	2-16
SQL 표현식에서 함수를 호출할 때의 부작용 제어	2-17
SQL에서 함수를 호출할 때의 제한 사항: 예제	2-18
SQL의 이름 지정 및 혼합 표기법	2-19
SQL의 이름 지정 및 혼합 표기법: 예제	2-20
함수 제거: DROP SQL 문 또는 SQL Developer 사용	2-21
데이터 덕셔너리 뷰를 사용하여 함수 보기	2-22
SQL Developer를 사용하여 함수 정보 보기	2-23
퀴즈	2-24
연습 2-1: 개요	2-25
단원 내용	2-26
SQL Developer Debugger를 사용하여 PL/SQL 서브 프로그램 디버그	2-27
서브 프로그램 디버그: 개요	2-28
프로시저 또는 함수 코드 편집 탭	2-29
프로시저 또는 함수 탭 도구 모음	2-30
Debugging – Log 탭 도구 모음	2-31
추가 탭	2-33
프로시저 예제 디버그: 새 emp_list 프로시저 생성	2-34
프로시저 예제 디버그: 새 get_location 함수 생성	2-35
중단점 설정 및 디버그 모드용으로 emp_list 컴파일	2-36
디버그 모드용으로 get_location 함수 컴파일	2-37
emp_list 디버그 및 PMAXROWS 파라미터 값 입력	2-38
emp_list 디버그: 코드 Step Into(F7)	2-39
데이터 보기	2-41
코드 디버그 도중 변수 수정	2-42
emp_list 디버그: 코드 Step Over	2-43
emp_list 디버그: 코드 Step Out(Shift+F7)	2-44
emp_list 디버그: Run to Cursor(F4)	2-45
emp_list 디버그: Step to End of Method	2-46
서브 프로그램 원격 디버깅: 개요	2-47
연습 2-2 개요: SQL Developer Debugger 소개	2-48
요약	2-49

3 패키지 생성

목표	3-2
단원 내용	3-3
PL/SQL 패키지란?	3-4
패키지 사용 시 이점	3-5
PL/SQL 패키지 구성 요소	3-7
패키지 구성 요소의 내부 및 external 표시 여부	3-8
PL/SQL 패키지 개발: 개요	3-9

단원 내용 3-10
Package Spec 생성: CREATE PACKAGE 문 사용 3-11
Package Spec 생성: SQL Developer 사용 3-12
Package Body 생성: SQL Developer 사용 3-13
Package Spec 예제: comm_pkg 3-14
Package Body 작성 3-15
Package Body 예제: comm_pkg 3-16
패키지 서브 프로그램 호출: 예제 3-17
패키지 서브 프로그램 호출: SQL Developer 사용 3-18
본문 없는 패키지 생성 및 사용 3-19
패키지 제거: SQL Developer 또는 SQL DROP 문 사용 3-20
데이터 딕셔너리를 사용하여 패키지 보기 3-21
SQL Developer를 사용하여 패키지 보기 3-22
패키지 작성 지침 3-23
퀴즈 3-24
요약 3-25
연습 3 개요: 패키지 생성 및 사용 3-26

4 패키지 작업

목표 4-2
단원 내용 4-3
PL/SQL에서 서브 프로그램 오버로드 4-4
프로시저 오버로드 예제: Package Spec 생성 4-6
프로시저 오버로드 예제: Package Body 생성 4-7
오버로드 및 STANDARD 패키지 4-8
잘못된 프로시저 참조 4-9
사전 선언을 사용하여 잘못된 프로시저 참조 해결 4-10
패키지 초기화 4-11
SQL에서 패키지 함수 사용 4-12
PL/SQL 서브 프로그램의 부작용 제어 4-13
SQL의 패키지 함수: 예제 4-14
단원 내용 4-15
패키지의 지속 상태 4-16
패키지 변수의 지속 상태: 예제 4-18
패키지 커서의 지속 상태: 예제 4-19
CURS_PKG 패키지 실행 4-21
패키지에서 연관 배열 사용 4-22
퀴즈 4-23
요약 4-24
연습 4: 개요 4-25

5 응용 프로그램 개발 시 오라클 제공 패키지 사용

목표	5-2
단원 내용	5-3
오라클 제공 패키지 사용	5-4
몇 가지 오라클 제공 패키지 예제	5-5
단원 내용	5-7
DBMS_OUTPUT 패키지 작동 방식	5-8
UTL_FILE 패키지를 사용하여 운영 체제 파일과 상호 작용	5-9
몇 가지 UTL_FILE 프로시저 및 함수	5-10
UTL_FILE 패키지를 사용한 파일 처리: 개요	5-11
UTL_FILE 패키지에서 사용 가능한 선언된 예외 사용	5-13
FOPEN 및 IS_OPEN 함수: 예제	5-14
UTL_FILE 사용: 예제	5-16
UTL_MAIL 패키지란?	5-18
UTL_MAIL 설정 및 사용: 개요	5-20
UTL_MAIL 서브 프로그램 요약	5-21
UTL_MAIL 설치 및 사용	5-22
SEND 프로시저 구문	5-23
SEND_ATTACH_RAW 프로시저	5-24
Binary File 을 첨부하여 전자 메일 보내기: 예제	5-25
SEND_ATTACH_VARCHAR2 프로시저	5-27
텍스트 파일을 첨부하여 전자 메일 보내기: 예제	5-28
퀴즈	5-30
요약	5-31
연습 5: 개요	5-32

6 동적 SQL 사용

목표	6-2
단원 내용	6-3
SQL의 실행 흐름	6-4
동적 SQL 작업	6-5
동적 SQL 사용	6-6
NDS(Native Dynamic SQL)	6-7
EXECUTE IMMEDIATE 문 사용	6-8
NDS 사용 방법	6-9
DDL 문을 사용하는 동적 SQL: 예제	6-11
DML 문을 사용하는 동적 SQL	6-12
단일 행 query를 사용하는 동적 SQL: 예제	6-13
PL/SQL 익명 블록 동적 실행	6-14
Native Dynamic SQL을 사용하여 PL/SQL 코드 컴파일	6-15
단원 내용	6-16
DBMS_SQL 패키지 사용	6-17

DBMS_SQL 패키지 서브 프로그램 사용	6-18
DML 문과 함께 DBMS_SQL 사용: 행 삭제	6-20
Parameterized DML 문과 함께 DBMS_SQL 사용	6-21
퀴즈	6-22
요약	6-23
연습 6 개요: Native Dynamic SQL 사용	6-24

7 PL/SQL 코드 설계 고려 사항

목표	7-2
단원 내용	7-3
상수 및 예외 표준화	7-4
예외 표준화	7-5
예외 처리 표준화	7-6
상수 표준화	7-7
로컬 서브 프로그램	7-8
정의자 권한과 호출자 권한 비교	7-9
호출자 권한 지정: AUTHID 를 CURRENT_USER 로 설정	7-10
독립 트랜잭션(Autonomous Transaction)	7-11
독립 트랜잭션의 특징	7-12
독립 트랜잭션 사용: 예제	7-13
단원 내용	7-15
NOCOPY 힌트 사용	7-16
NOCOPY 힌트의 영향	7-17
PL/SQL 컴파일러가 NOCOPY 힌트를 무시하는 경우	7-18
PARALLEL_ENABLE 힌트 사용	7-19
세션간 PL/SQL 함수 결과 캐시 사용	7-20
함수의 결과 캐싱 활성화	7-21
결과 캐시된 함수 선언 및 정의: 예제	7-22
함수와 함께 DETERMINISTIC 절 사용	7-24
단원 내용	7-25
RETURNING 절 사용	7-26
대량 바인드 사용	7-27
대량 바인딩: 구문 및 키워드	7-28
FORALL 대량 바인드: 예제	7-30
Query 와 함께 BULK COLLECT INTO 사용	7-32
커서와 함께 BULK COLLECT INTO 사용	7-33
RETURNING 절과 함께 BULK COLLECT INTO 사용	7-34
Sparse Collection 에서 대량 바인드 사용	7-35
인덱스 배열에서 대량 바인드 사용	7-38
퀴즈	7-39
요약	7-40
연습 7: 개요	7-41

8 트리거 생성

- 목표 8-2
- 트리거란? 8-3
- 트리거 정의 8-4
- 트리거 이벤트 유형 8-5
 - 응용 프로그램 및 데이터베이스 트리거 8-6
 - 업무용 응용 프로그램의 트리거 구현 시나리오 8-7
- 사용 가능한 트리거 유형 8-8
- 트리거 이벤트 유형 및 본문 8-9
- CREATE TRIGGER 문을 사용하여 DML 트리거 생성 8-10
- 트리거 실행 지정(타이밍) 8-11
- 문장 레벨 트리거와 행 레벨 트리거의 비교 8-12
- SQL Developer를 사용하여 DML 트리거 생성 8-13
- 트리거 실행 시퀀스: 단일 행 조작 8-14
- 트리거 실행 시퀀스: 여러 행 조작 8-15
- DML 문장 트리거 생성 예제: SECURE_EMP 8-16
- 트리거 SECURE_EMP 테스트 8-17
- 조건부 술어 사용 8-18
- DML 행 트리거 생성 8-19
- OLD 및 NEW 수식자 사용 8-20
- OLD 및 NEW 수식자 사용: 예제 8-21
- WHEN 절을 사용하여 조건을 기준으로 행 트리거 실행 8-23
- 트리거 실행 모델 요약 8-24
- After 트리거를 사용하여 무결성 제약 조건 구현 8-25
- INSTEAD OF 트리거 8-26
- INSTEAD OF 트리거 생성: 예제 8-27
- INSTEAD OF 트리거를 생성하여 복합 뷰에서 DML 수행 8-28
- 트리거의 상태 8-30
- 비활성화된 트리거 생성 8-31
- ALTER 및 DROP SQL 문을 사용하여 트리거 관리 8-32
- SQL Developer를 사용하여 트리거 관리 8-33
- 트리거 테스트 8-34
- 트리거 정보 보기 8-35
- USER_TRIGGERS 사용 8-36
- 퀴즈 8-37
- 요약 8-38
- 연습 8 개요: 문장 및 행 트리거 생성 8-39

9 혼합, DDL 및 데이터베이스 이벤트 트리거 생성

목표 9-2

혼합 트리거란? 9-3

혼합 트리거 작업 9-4

혼합 트리거 사용 시의 이점 9-5

테이블 혼합 트리거의 타이밍 지점 섹션 9-6

테이블을 위한 혼합 트리거 구조 9-7

뷰를 위한 혼합 트리거 구조 9-8

혼합 트리거의 제한 사항 9-9

변경 테이블의 트리거 제한 사항 9-10

변경 테이블: 예제 9-11

혼합 트리거를 사용하여 변경 테이블 오류 해결 9-13

DDL 문에 트리거 생성 9-15

데이터베이스 이벤트 트리거 생성 9-16

시스템 이벤트에 트리거 생성 9-17

LOGON 및 LOGOFF 트리거: 예제 9-18

트리거의 CALL 문 9-19

데이터베이스 이벤트 트리거의 이점 9-20

트리거를 관리하는 데 필요한 시스템 권한 9-21

트리거 설계 지침 9-22

퀴즈 9-23

요약 9-24

연습 9: 개요 9-25

10 PL/SQL 컴파일러 사용

목표 10-2

단원 내용 10-3

PL/SQL 컴파일용 초기화 파라미터 10-4

PL/SQL 컴파일을 위해 초기화 파라미터 사용 10-5

컴파일러 설정 10-7

PL/SQL 초기화 파라미터 표시 10-8

PL/SQL 초기화 파라미터 표시 및 설정 10-9

PL/SQL 초기화 파라미터 변경: 예제 10-10

단원 내용 10-11

서브 프로그램용 PL/SQL 컴파일 타임 경고 개요 10-12

컴파일러 경고의 이점 10-14

PL/SQL 컴파일 타임 경고 메시지의 범주 10-15

경고 메시지 레벨 설정 10-16

컴파일러 경고 레벨 설정: PLSQL_WARNINGS 사용 10-17

컴파일러 경고 레벨 설정: PLSQL_WARNINGS 사용, 예제 10-18

컴파일러 경고 레벨 설정: SQL Developer에서 PLSQL_WARNINGS 사용 10-19

PLSQL_WARNINGS의 현재 설정 보기 10-20

컴파일러 경고 보기: SQL Developer, SQL*Plus 또는
데이터 딕셔너리 뷰 사용 10-21
SQL*Plus 경고 메시지: 예제 10-22
PLSQL_WARNINGS 사용 지침 10-23
단원 내용 10-24
컴파일러 경고 레벨 설정: DBMS_WARNING 패키지 사용 10-25
DBMS_WARNING 패키지 서브 프로그램 사용 10-27
DBMS_WARNING 프로시저: 구문, 파라미터 및 허용 값 10-28
DBMS_WARNING 프로시저: 예제 10-29
DBMS_WARNING 함수: 구문, 파라미터 및 허용값 10-30
DBMS_WARNING 함수: 예제 10-31
DBMS_WARNING 사용: 예제 10-32
PLW 06009 경고 메시지 사용 10-34
PLW 06009 경고: 예제 10-35
퀴즈 10-36
요약 10-37
연습 10: 개요 10-38

11 PL/SQL 코드 관리

목표 11-2
단원 내용 11-3
조건부 컴파일이란? 11-4
조건부 컴파일의 작동 방식 11-5
선택 지시어 사용 11-6
미리 정의된 조회 지시어 및 유저 정의 조회 지시어 사용 11-7
PLSQL_CCFLAGS 파라미터 및 조회 지시어 11-8
PLSQL_CCFLAGS 초기화 파라미터 설정 표시 11-9
PLSQL_CCFLAGS 파라미터 및 조회 지시어: 예제 11-10
조건부 컴파일 오류 지시어를 사용하여 유저 정의 오류 발생 11-11
조건부 컴파일에 정적 표현식 사용 11-12
DBMS_DB_VERSION 패키지: 부울 상수 11-13
DBMS_DB_VERSION 패키지 상수 11-14
데이터베이스 버전과 함께 조건부 컴파일 사용: 예제 11-15
DBMS_PREPROCESSOR 프로시저를 사용하여 소스
텍스트 인쇄 또는 검색 11-17
단원 내용 11-18
난독 처리란? 11-19
난독 처리의 이점 11-20
Oracle 10g 이후 동적 난독 처리의 새로운 기능 11-21
난독 처리가 적용되지 않은 PL/SQL 코드: 예제 11-22
난독 처리가 적용된 PL/SQL 코드: 예제 11-23
동적 난독 처리: 예제 11-24

PL/SQL 래퍼 유ти리티	11-25
래퍼 유ти리티 실행	11-26
래핑 결과	11-27
래핑 지침	11-28
DBMS_DDL 패키지와 Wrap 유ти리티 비교	11-29
퀴즈	11-30
요약	11-31
연습 11: 개요	11-32

12 종속성 관리

목표	12-2
스키마 객체 종속성 개요	12-3
종속성	12-4
직접 로컬 종속성	12-5
직접 객체 종속성 query: USER_DEPENDENCIES 뷰 사용	12-6
객체 상태 query	12-7
종속 객체 무효화	12-8
일부 종속 항목을 무효화하는 스키마 객체 변경: 예제	12-9
직접 및 간접 종속성 표시	12-11
DEPTREE 뷰를 사용하여 종속성 표시	12-12
Oracle Database 11g의 보다 정밀한 종속성 메타 데이터	12-13
Fine-Grained Dependency 관리	12-14
Fine-Grained Dependency 관리: 예제 1	12-15
Fine-Grained Dependency 관리: 예제 2	12-17
동의어 종속성의 변화	12-18
유효한 PL/SQL 프로그램 단위 및 뷰 유지 관리	12-19
로컬 종속성에 대한 또 다른 시나리오	12-20
무효화를 줄이기 위한 지침	12-21
객체 재검증	12-22
원격 종속성	12-23
원격 종속성의 개념	12-24
REMOTE_DEPENDENCIES_MODE 파라미터 설정	12-25
오전 8:00에 원격 프로시저 B 컴파일	12-26
오전 9:00에 로컬 프로시저 A 컴파일	12-27
프로시저 A 실행	12-28
오전 11:00에 원격 프로시저 B 재컴파일	12-29
프로시저 A 실행	12-30
서명 모드	12-31
PL/SQL 프로그램 단위 재컴파일	12-32
재컴파일 실패	12-33
재컴파일 성공	12-34
프로시저 재컴파일	12-35

- 패키지 및 종속성: 서브 프로그램에서 패키지 참조 12-36
- 패키지 및 종속성: 패키지 서브 프로그램에서 프로시저 참조 12-37
- 퀴즈 12-38
- 요약 12-39
- 연습 12 개요: 스키마에서 종속성 관리 12-40

부록 A: 연습 및 해답

부록 AP: 추가 연습 및 해답

부록 B: 테이블 설명

부록 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

부록 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
- SERVEROUTPUT 명령 D-15
- SQL*Plus SPOOL 명령 사용 D-16
- AUTOTRACE 명령 사용 D-17
- 요약 D-18

부록 E: JDeveloper 사용

- 목표 E-2
- Oracle JDeveloper E-3
- Database Navigator E-4
- 연결 생성 E-5
- 데이터베이스 객체 탐색 E-6
- SQL 문 실행 E-7
- 프로그램 단위 생성 E-8
- 컴파일 E-9
- 프로그램 단위 실행 E-10
- 프로그램 단위 삭제 E-11
- Structure window E-12
- Editor window E-13
- Application Navigator E-14
- Java 내장 프로시저 배치 E-15
- PL/SQL에 Java 게시(publishing) E-16
- JDeveloper 11g에 대해 자세히 배울 수 있는 방법 E-17
- 요약 E-18

부록 F: PL/SQL 검토

- 목표 F-2
- 익명 PL/SQL 블록의 블록 구조 F-3
- PL/SQL 변수 선언 F-4
- %TYPE 속성을 사용하여 변수 선언: 예제 F-5
- PL/SQL 레코드 생성 F-6
- %ROWTYPE 속성: 예제 F-7

PL/SQL 테이블 생성	F-8
PL/SQL 의 SELECT 문: 예제	F-9
데이터 삽입: 예제	F-10
데이터 갱신: 예제	F-11
데이터 삭제: 예제	F-12
COMMIT 및 ROLLBACK 문을 사용하여 트랜잭션 제어	F-13
IF, THEN 및 ELSIF 문: 예제	F-14
기본 루프: 예제	F-15
FOR 루프: 예제	F-16
WHILE 루프: 예제	F-17
SQL 암시적 커서 속성	F-18
명시적 커서 제어	F-19
명시적 커서 제어: 커서 선언	F-20
명시적 커서 제어: 커서 열기	F-21
명시적 커서 제어: 커서에서 데이터 패치(fetch)	F-22
명시적 커서 제어: 커서 닫기	F-23
명시적 커서 속성	F-24
커서 FOR 루프: 예제	F-25
FOR UPDATE 절: 예제	F-26
WHERE CURRENT OF 절: 예제	F-27
미리 정의된 Oracle 서버 오류 트랩	F-28
미리 정의된 Oracle 서버 오류 트랩: 예제	F-29
미리 정의되지 않은 오류	F-30
유저 정의 예외: 예제	F-31
RAISE_APPLICATION_ERROR 프로시저	F-32
요약	F-34

부록 G: 트리거 구현 학습

목표	G-2
서버 내에서 보안 제어	G-3
데이터베이스 트리거를 사용한 보안 제어	G-4
서버 내에서 데이터 무결성 적용	G-5
트리거를 사용한 데이터 무결성 보호	G-6
서버 내에서 참조 무결성 적용	G-7
트리거를 사용한 참조 무결성 보호	G-8
서버 내에서 테이블 복제(replication)	G-9
트리거를 사용한 테이블 복제(replication)	G-10
서버 내에서 파생된 데이터 계산	G-11
트리거를 사용하여 파생된 값 계산	G-12
트리거를 사용한 이벤트 로깅	G-13
요약	G-15

부록 H: DBMS_SCHEDULER 및 HTP 패키지 사용

목표 H-2

HTP 패키지를 사용하여 웹 페이지 생성 H-3

HTP 패키지 프로시저 사용 H-4

SQL*Plus 를 사용하여 HTML 파일 생성 H-5

DBMS_SCHEDULER 패키지 H-6

작업 생성 H-8

인라인 파라미터를 사용하여 작업 생성 H-9

프로그램을 사용하여 작업 생성 H-10

인수를 사용하여 프로그램에 대한 작업 생성 H-11

스케줄을 사용하여 작업 생성 H-12

작업 반복 간격 설정 H-13

명명된 프로그램과 스케줄을 사용하여 작업 생성 H-14

작업 관리 H-15

데이터 딕셔너리 뷰 H-17

요약 H-18

I

소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 과정 목표 설명
- 본 과정에서 사용할 수 있는 환경 식별
- 본 과정에 사용되는 데이터베이스 스키마 및 테이블 설명
- 사용 가능한 설명서 및 리소스 나열

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

PL/SQL은 여러 프로그램 생성자를 지원합니다. 이 단원에서는 익명 블록 형태의 프로그램 단위를 살펴보고 명명된 PL/SQL 블록에 대해 소개합니다. 명명된 PL/SQL 블록은 서브 프로그램이라고도 하며, 프로시저와 함수를 포함합니다.

HR (Human Resources) 스키마(본 과정의 연습에 사용됨)의 테이블에 대해 간략하게 설명하며 PL/SQL 작성, 테스트 및 디버깅을 위한 개발 도구를 나열합니다.

단원 내용

- 과정 목표 및 과정 내용
- 이 과정에서 사용된 스키마와 부록 및 이 과정에서 사용 가능한 PL/SQL 개발 환경
- Oracle 11g 설명서 및 추가 리소스

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

과정 목표

이 과정을 마치면 다음을 수행할 수 있습니다.

- 다음의 항목 생성, 실행 및 유지 관리:
 - OUT 파라미터를 사용하는 프로시저 및 함수
 - 패키지 생성자
 - 데이터베이스 트리거
- PL/SQL 서브 프로그램 및 트리거 관리
- 오라클 제공 패키지를 사용하여 화면 출력 및 파일 출력 생성
- PL/SQL 코드 설계 고려 사항에 영향을 주는 다양한 기법 식별
- PL/SQL 컴파일러 사용, PL/SQL 코드 관리 및 종속성 관리

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

과정 목표

다음과 같은 데이터베이스 객체를 사용하여 데이터베이스 프로시저로 모듈 방식의 응용 프로그램을 개발할 수 있습니다.

- 프로시저 및 함수
- 패키지
- 데이터베이스 트리거

모듈 방식의 응용 프로그램을 사용하면 다음이 향상됩니다.

- 기능
- 보안
- 전체적인 성능

권장 과정 일정

첫째 날:

- 단원 1: 소개
- 단원 1: 프로시저 생성
- 단원 2: 함수 생성
- 단원 3: 패키지 생성
- 단원 4: 패키지 작업

둘째 날:

- 단원 5: 응용 프로그램 개발 시 오라클 제공 패키지 사용
- 단원 6: 동적 SQL 사용
- 단원 7: PL/SQL 코드 설계 고려 사항
- 단원 8: 트리거 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

권장 과정 일정

셋째 날:

- 단원 9: 혼합, DDL 및 이벤트 데이터베이스 트리거 생성
- 단원 10: PL/SQL 컴파일러 사용
- 단원 11: PL/SQL 코드 관리
- 단원 12: 종속성 관리

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

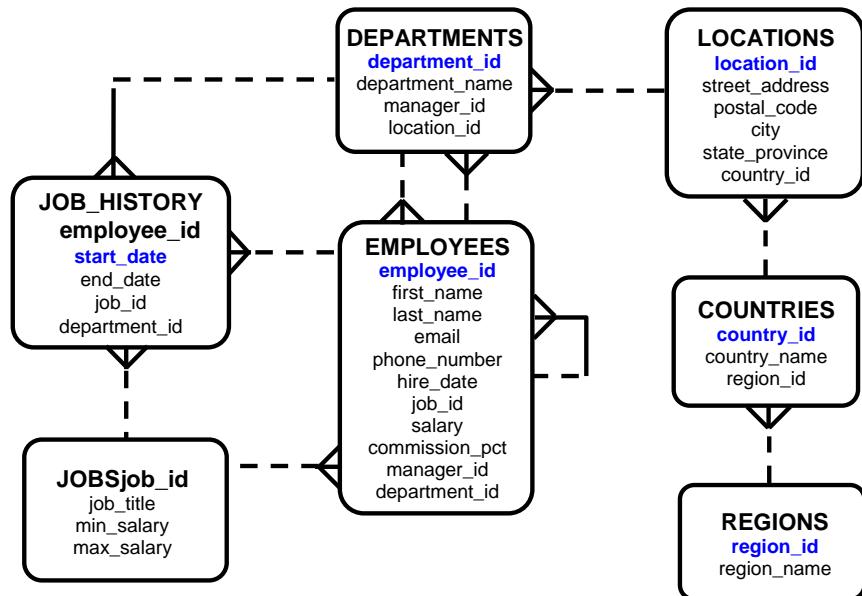
단원 내용

- 과정 목표 및 과정 내용
- 이 과정에서 사용된 스키마와 부록 및 이 과정에서 사용 가능한 PL/SQL 개발 환경
- Oracle 11g 설명서 및 추가 리소스

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

I | 과정에서 사용되는 HR (Human Resources) 스키마



ORACLE

Copyright © 2009, Oracle. All rights reserved.

HR (Human Resources) 스키마 설명

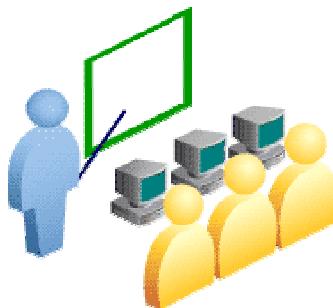
HR (Human Resources) 스키마는 오라클 데이터베이스에 설치할 수 있는 오라클 예제 스키마의 일부입니다. 본 과정의 연습 세션에서는 HR 스키마의 데이터를 사용합니다.

테이블 설명

- REGIONS에는 Americas, Asia 등의 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.

클래스 계정 정보

- 복제된 HR 계정 ID가 설정됩니다.
- 계정 ID는 ora61 또는 ora62입니다.
- 암호는 계정 ID와 일치합니다.
- 시스템마다 하나의 계정이 할당되어 있습니다.
- 강사는 별도의 ID를 가지고 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고: 계정 ID ora61 또는 ora62를 사용합니다.

본 과정에 사용되는 부록

- **부록 A: 연습 및 해답**
- **부록 AP: 추가 연습 및 해답**
- **부록 B: 테이블 설명**
- **부록 C: SQL Developer 사용**
- **부록 D: SQL*Plus 사용**
- **부록 E: JDeveloper 검토**
- **부록 F: PL/SQL 검토**
- **부록 G: 트리거 구현 학습**
- **부록 H: DBMS_SCHEDULER 및 HTP 패키지 사용**

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

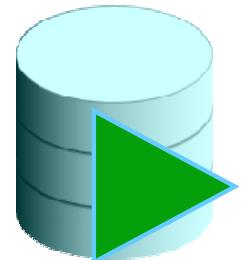
PL/SQL 코드 개발을 위한 환경을 제공하는 도구에는 여러 가지가 있습니다. 오라클은 PL/SQL 코드를 작성하는 데 사용할 수 있는 여러 도구를 제공합니다. 본 과정에서 사용할 수 있는 몇 가지 개발 도구는 다음과 같습니다.

- 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 제공되는 그래픽 도구입니다. 간단히 버튼을 몇 번 누르기만 하면 손쉽게 내장 프로시저를 생성 및 유지 관리하고, SQL 문을 테스트하고, 옵티마이저 계획을 볼 수 있습니다.

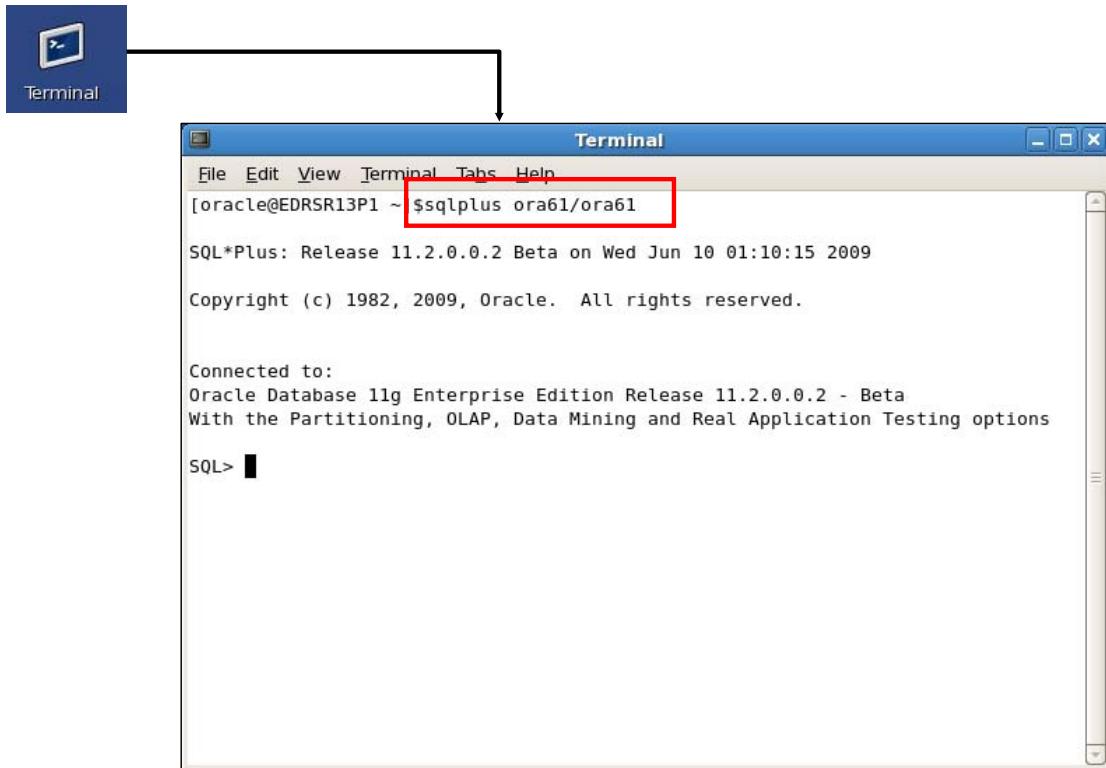
데이터베이스 개발을 위한 시각적 도구인 SQL Developer는 다음 작업을 단순화합니다.

- 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.

참고: 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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus에서 PL/SQL 코딩

Oracle SQL*Plus는 SQL 문과 PL/SQL 블록을 실행한 다음 그 결과를 응용 프로그램이나 명령 window에서 수신할 수 있는 GUI(graphical user interface) 또는 명령행 응용 프로그램입니다.

SQL*Plus는 다음 특징을 갖습니다.

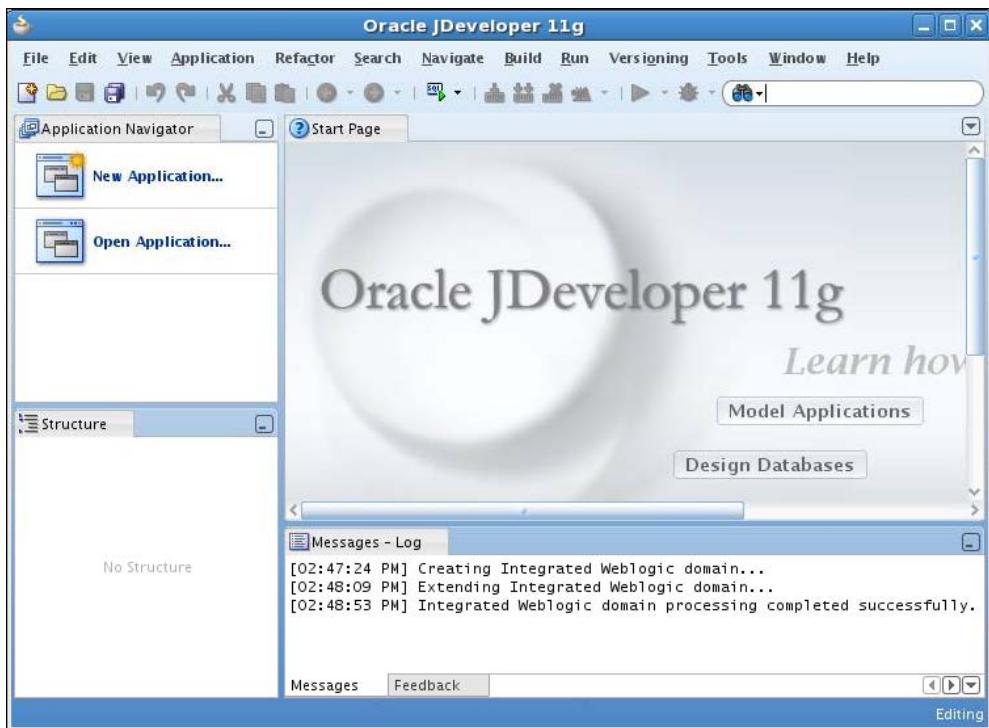
- 데이터베이스와 함께 제공됩니다.
- 클라이언트 및 데이터베이스 서버 시스템에 설치됩니다.
- 아이콘이나 명령행을 통해 액세스됩니다.

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: SQL*Plus를 사용하는 방법에 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 때는 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.
- JDeveloper를 사용하는 방법에 대한 자세한 내용은 부록 E를 참조하십시오.

PL/SQL 블록의 출력 활성화

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

```
SET SERVEROUTPUT ON;
```

- 다음과 같이 미리 정의된 DBMS_OUTPUT 오라클 패키지와 해당 프로시저를 사용하여 출력을 표시합니다.

- DBMS_OUTPUT.PUT_LINE

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

ORACLE

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 SQL 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.

단원 내용

- 과정 목표 및 과정 내용
- 이 과정에서 사용된 스키마 및 부록과 이 과정에서 사용 가능한 PL/SQL 개발
- Oracle 11g 설명서 및 추가 리소스

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

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.

Oracle 11g SQL 및 PL/SQL 설명서

<http://www.oracle.com/pls/db111/homepage>로 이동하여 왼쪽 프레임에서 Master Book List 링크를 누르십시오.

추가 자료

새로운 Oracle 11g SQL과 PL/SQL의 새로운 기능에 대한 자세한 내용은 다음 자료를 참조하십시오.

- **Oracle Database 11g: New Features eStudies**
- **Oracle by Example (OBE) series: Oracle Database 11g:**
 - http://www.oracle.com/technology/obe/11gr1_db/admin/11gr1db.html
- **OTN(Oracle Technology Network)에 게시된 What's New in PL/SQL in Oracle Database 11g:**
 - http://www.oracle.com/technology/tech/pl_sql/



Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 과정 목표 설명
- 본 과정에서 사용할 수 있는 환경 식별
- 본 과정에 사용되는 데이터베이스 스키마 및 테이블 설명
- 사용 가능한 설명서 및 리소스 나열

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

PL/SQL 언어는 재사용이 가능한 코드 블록에 여러 가지 프로그램 생성자를 제공합니다. 이름이 지정되지 않거나 익명인 PL/SQL 블록을 사용하여 SQL 및 PL/SQL 작업, 프로시저, 함수 및 패키지 구성 요소를 호출할 수 있습니다. 명명된 PL/SQL 블록(서브 프로그램이라고도 함)에는 다음이 포함됩니다.

- 프로시저
- 함수
- 패키지 프로시저 및 함수
- 트리거

오라클은 여러 가지의 PL/SQL 기능 개발 도구를 제공합니다. 오라클은 Oracle Forms 및 Oracle Reports에 클라이언트측 또는 Middle-tier PL/SQL 런타임 환경을 제공하며, 오라클 데이터베이스 내에 PL/SQL 런타임 엔진을 제공합니다. 데이터베이스 내에 있는 프로시저와 함수는 오라클 데이터베이스에 연결하여 PL/SQL 코드를 실행할 수 있는 임의의 응용 프로그램 코드에서 호출할 수 있습니다.

연습 I: 개요: 시작하기

이 연습에서는 다음 내용을 다룹니다.

- 사용 가능한 SQL Developer 리소스 검토
- SQL Developer를 시작하여 새 데이터베이스 연결 생성 및 스키마 테이블 탐색
- 일부 SQL Developer 환경 설정 구성
- SQL Worksheet를 사용하여 SQL 문 및 익명 PL/SQL 블록 실행
- Oracle Database 11g 설명서 및 기타 유용한 웹 사이트 액세스 및 책갈피 지정

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 I: 개요

이 연습에서는 SQL Developer에서 SQL 문을 실행하여 스키마의 데이터를 검사합니다. 또한 간단한 익명 블록을 생성합니다. 선택적으로 SQL*Plus에서 PL/SQL 코드를 생성한 다음 실행해 볼 수도 있습니다.

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

프로시저 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 모듈화되고 계층화된 서브 프로그램 설계의 이점 식별
- 프로시저 생성 및 호출
- 형식 및 실제 파라미터 사용
- 위치 지정, 이름 지정 또는 혼합 표기법을 사용하여 파라미터 전달
- 사용 가능한 파라미터 전달 모드 식별
- 프로시저에서 예외 처리
- 프로시저 제거
- 프로시저 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 파라미터를 사용하거나 사용하지 않고 프로시저를 생성, 실행 및 제거하는 방법에 대해 설명합니다. 프로시저는 PL/SQL에서 모듈 방식 프로그래밍의 기반입니다. 보다 융통성 있는 프로시저를 만들기 위해서는 입력 파라미터를 사용하여 다양한 데이터를 계산하거나 프로시저에 전달하는 것이 중요합니다. 계산 결과는 OUT 파라미터를 사용하여 프로시저 호출자에게 반환될 수 있습니다.

견고한 프로그램을 만들려면 PL/SQL의 예외 처리 기능을 사용하여 예외 상태를 항상 관리해야 합니다.

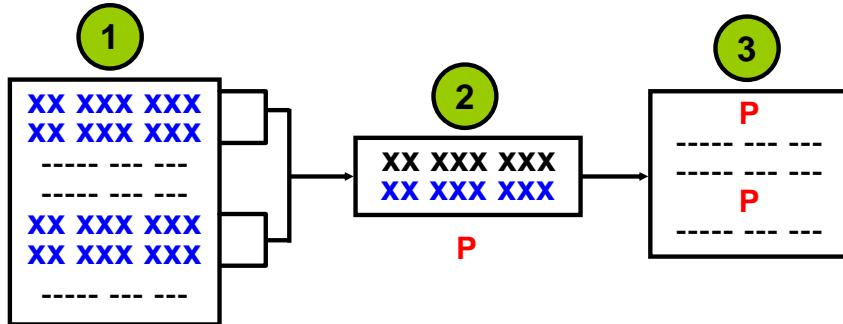
단원 내용

- 모듈화되고 계층화된 서브 프로그램 설계 사용 및 서브 프로그램의 이점 식별
- 프로시저 작업:
 - 프로시저 생성 및 호출
 - 사용 가능한 파라미터 전달 모드 식별
 - 형식 및 실제 파라미터 사용
 - 위치 지정, 이름 지정 또는 혼합 표기법 사용
- 프로시저에서 예외 처리, 프로시저 제거 및 프로시저 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

모듈화된 서브 프로그램 설계 생성



서브 프로그램으로 코드를 모듈화합니다.

1. 두 번 이상 반복되는 코드 시퀀스를 찾습니다.
2. 반복 코드가 포함된 서브 프로그램 P를 생성합니다.
3. 새 서브 프로그램을 호출하도록 원본 코드를 수정합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

모듈화되고 계층화된 서브 프로그램 설계 생성

위 도표는 서브 프로그램을 사용하는 모듈화 원칙, 즉 융통적이고 재사용이 가능한 코드를 보다 작은 관리 가능한 조각으로 생성하는 것을 보여줍니다. 서브 프로그램과 파라미터를 함께 사용하면 동일한 코드를 여러 입력 값에 재사용할 수 있는 융통성을 얻을 수 있습니다. 기존 코드를 모듈화하려면 다음 단계를 수행하십시오.

1. 반복되는 코드 시퀀스를 찾아 식별합니다.
2. 반복 코드를 PL/SQL 서브 프로그램으로 이동합니다.
3. 새 PL/SQL 서브 프로그램을 호출하도록 반복 코드 원본을 바꿉니다.

이와 같이 모듈화되고 계층화된 접근 방법을 따르면 특히 업무 규칙이 변경될 때 유지 관리가 쉬운 코드를 생성할 수 있습니다. 또한 SQL 문장을 원본 코드에서 호출될 수 있는 별도로 모듈화시켜 생성 관리하면 오라클 데이터베이스 옵티마이저 작업을 수행할 때 유용합니다. 즉, 서버측 리소스를 보다 잘 활용하기 위해 구문 분석된 SQL 문을 재사용할 수 있습니다.

계층화된 서브 프로그램 설계 생성

응용 프로그램에 대해 다음의 예처럼 서브 프로그램 층을 생성합니다.

- SQL 논리를 사용하는 데이터 액세스 서브 프로그램 층
- 업무 논리 서브 프로그램 층(데이터 액세스 층을 사용할 수도 있고 사용하지 않을 수도 있음)



Copyright © 2009, Oracle. All rights reserved.

계층화된 서브 프로그램 설계 생성

PL/SQL을 사용하면 SQL 문을 무리 없이 논리에 삽입할 수 있기 때문에 SQL 문을 코드 전체에 쉽게 사용할 수 있습니다. 그러나 오라클에서는 SQL 논리와 업무 논리를 구분해 둘 것을 권장합니다. 즉, 최소한 다음 두 층을 사용하여 계층화된 응용 프로그램 설계를 생성하는 것이 좋습니다.

- 데이터 액세스 계층: SQL 문을 사용하여 데이터에 액세스하는 서브 루틴용
- 업무 논리 계층: 업무 처리 규칙을 구현하는 서브 프로그램용(데이터 액세스 계층 루틴을 호출할 수도 있고 호출하지 않을 수도 있음)

PL/SQL 블록을 사용한 개발 모듈화

- PL/SQL은 블록 구조 언어입니다. PL/SQL 코드 블록은 다음을 사용하여 코드 모듈화를 지원합니다.
 - 익명 블록
 - 프로시저 및 함수
 - 패키지
 - 데이터베이스 트리거
- 모듈 방식 프로그램 생성자 사용 시 이점:
 - 손쉬운 유지 관리
 - 데이터 보안 및 무결성 향상
 - 성능 향상
 - 코드 명확성 향상

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록을 사용한 개발 모듈화

서브 프로그램은 표준 PL/SQL 구조를 기반으로 하며, 선언 섹션, 실행 섹션, 선택적으로 예외 처리 섹션을 포함합니다(예: 익명 블록, 프로시저, 함수, 패키지 및 트리거). 서브 프로그램은 데이터베이스에서 컴파일하고 저장할 수 있기 때문에 모듈화, 확장성, 재사용성 및 유지 관리 용이성을 제공합니다.

모듈화는 대용량 코드 블록을 모듈이라고 하는 보다 작은 코드 그룹으로 변환합니다. 모듈화한 후에는 모듈을 동일한 프로그램에서 재사용하거나 다른 프로그램과 공유할 수 있습니다. 작은 모듈로 구성된 코드를 유지 관리하고 디버그하는 것이 대용량의 단일 프로그램으로 된 코드를 유지 관리하는 것보다 쉽습니다. 필요할 경우 프로그램의 나머지 모듈에는 영향을 주지 않으면서 보다 많은 기능을 통합하여 커스터마이제이션을 위해 모듈을 쉽게 확장할 수 있습니다.

서브 프로그램은 코드가 한 곳에 위치하고 있어 서브 프로그램에 필요한 사항을 한 곳에서 수정할 수 있기 때문에 유지 관리가 쉬우며, 향상된 데이터 무결성과 보안을 제공합니다. 데이터 객체는 서브 프로그램을 통해 액세스되므로 적절한 액세스 권한을 가지고 있는 유저만 서브 프로그램을 호출할 수 있습니다.

참고: 본 과정을 수강하기 위해서는 익명 블록 개발 방법을 알고 있어야 합니다. 익명 블록에 대한 자세한 내용은 *Oracle 11g: PL/SQL Fundamentals* 과정을 참조하십시오.

익명 블록: 개요

익명 블록:

- 기본 PL/SQL 블록 구조를 구성함
- 응용 프로그램에서 PL/SQL 처리 작업을 시작함
- PL/SQL 블록의 실행 섹션 내에 중첩할 수 있음

```
[DECLARE      -- Declaration Section (Optional)
    variable declarations; ... ]
BEGIN        -- Executable Section (Mandatory)
    SQL or PL/SQL statements;
[EXCEPTION   -- Exception Section (Optional)
    WHEN exception THEN statements; ]
END;         -- End of Block (Mandatory)
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

익명 블록: 개요

익명 블록의 일반적인 용도는 다음과 같습니다.

- Oracle Forms 구성 요소에 대한 트리거 코드 생성
- 프로시저, 함수 및 패키지 생성자에 대한 호출 시작
- 코드 블록 내에서 예외 처리 분리
- 코드 흐름 제어를 관리하기 위해 다른 PL/SQL 블록 내에 중첩

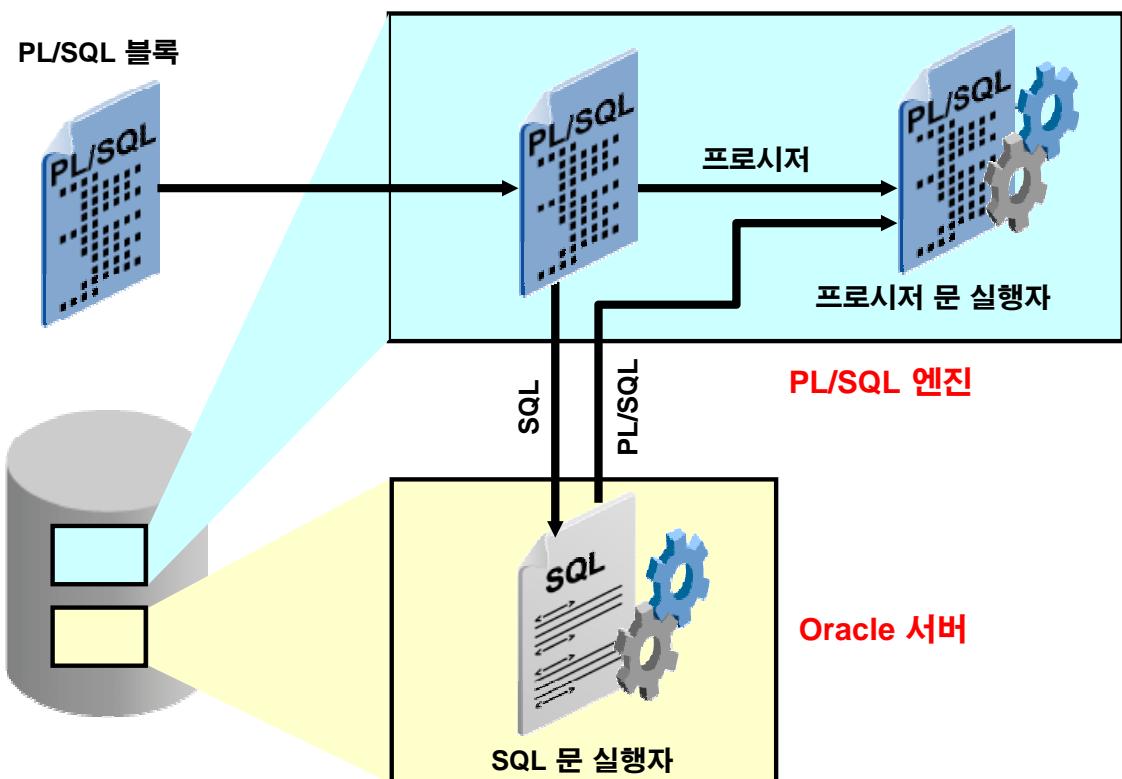
DECLARE 키워드는 선택 사항이지만, PL/SQL 블록 내에서 사용될 변수, 상수 및 예외를 선언할 경우에는 필수입니다.

BEGIN 및 END는 필수이며 두 키워드 사이에 명령문(SQL, PL/SQL 또는 둘 다)이 하나 이상 있어야 합니다.

예외 섹션은 선택적이며 PL/SQL 블록 범위 내에서 발생하는 오류를 처리하는 데 사용됩니다.

특정 예외에 대해 예외 처리기를 제외시킴으로써 처리되지 않은 예외를 생성하게 되면 익명 블록 호출자에게 예외를 전달할 수 있습니다.

PL/SQL 런타임 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 런타임 구조

위 도표는 PL/SQL 엔진에 의해 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. 예를 들어 select 문에 함수를 사용할 때 SQL 환경에서 PL/SQL 환경을 호출할 수도 있습니다.

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 전환 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/SQL 서브 프로그램이란?

- PL/SQL 서브 프로그램은 파라미터 집합을 사용하여 호출할 수 있는 명명된 PL/SQL 블록입니다.
- PL/SQL 블록 또는 다른 서브 프로그램 내에서 서브 프로그램을 선언하고 정의할 수 있습니다.
- 서브 프로그램은 Spec과 Body로 구성됩니다.
- 서브 프로그램은 프로시저이거나 함수일 수 있습니다.
- 일반적으로, 작업을 수행할 때는 프로시저를 사용하고 값을 계산하고 반환할 때는 함수를 사용합니다.
- 서브 프로그램은 PL/SQL 패키지로 그룹화될 수 있습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 서브 프로그램이란?

PL/SQL 서브 프로그램은 파라미터 집합을 사용하여 호출할 수 있는 명명된 PL/SQL 블록입니다.
PL/SQL 블록 또는 다른 서브 프로그램 내에서 서브 프로그램을 선언하고 정의할 수 있습니다.

서브 프로그램 부분

서브 프로그램은 Spec과 Body로 구성됩니다. 서브 프로그램을 선언하려면 파라미터 설명을 포함하는 Spec을 제공해야 합니다. 그리고 서브 프로그램을 정의하려면 Spec과 Body를 모두 제공해야 합니다. 서브 프로그램을 먼저 선언한 후에 같은 블록이나 서브 프로그램에서 나중에 정의할 수도 있고, 동시에 선언 및 정의할 수도 있습니다.

서브 프로그램 유형

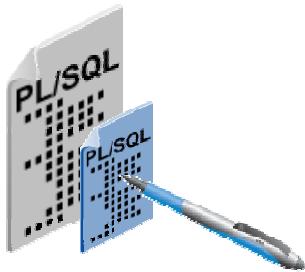
PL/SQL에는 두 가지 유형의 서브 프로그램인 프로시저와 함수가 있습니다. 일반적으로 작업을 수행할 때는 프로시저를 사용하고 값을 계산하고 반환할 때는 함수를 사용합니다.

함수에만 RETURN 절이나 RETURN 문 등의 추가 항목이 있다는 것을 제외하면 프로시저와 함수의 구조는 같습니다.

RETURN 절은 반환 값의 데이터 유형(필수)을 지정합니다. 그리고 RETURN 문은 반환 값(필수)을 지정합니다. 함수에 대해서는 다음 단원인 "함수 생성 및 서브 프로그램 디버그"에서 자세히 다룹니다.

서브 프로그램을 PL/SQL 패키지로 그룹화할 수 있습니다. 패키지를 사용하면 코드를 재사용하고 유지 관리하기가 훨씬 더 쉬워집니다. 패키지는 패키지 단원인 단원 3 및 4에서 다룹니다.

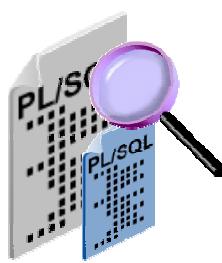
PL/SQL 서브 프로그램 사용 시 이점



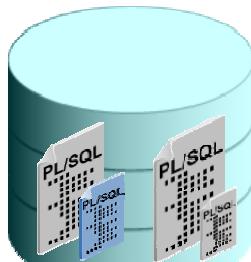
손쉬운 유지 관리



데이터 보안 및 무결성 향상



코드 명확성 향상



서브 프로그램: 내장 프로시저 및 함수



성능 향상

ORACLE

Copyright © 2009, Oracle. All rights reserved.

서브 프로그램의 이점

프로시저와 함수는 코드가 모듈화되어 있기 때문에 사용할 경우 여러 가지 이점이 있습니다.

- 서브 프로그램이 한 곳에 있기 때문에 유지 관리가 쉽습니다. 한 곳에서만 수정해도 여러 응용 프로그램에 영향을 미칠 수 있으며, 과도한 테스팅을 최소화할 수 있습니다.
- 보안 권한을 사용하여 권한이 없는 유저가 데이터베이스 객체에 간접 액세스하는 것을 통제함으로써 데이터 보안이 향상됩니다. 서브 프로그램은 기본적으로 정의자 권한으로 실행됩니다. 실행 권한으로는 호출한 유저가 객체에 대한 직접 액세스를 통해 서브 프로그램에 액세스할 수 없습니다.
- 관련 작업이 모두 함께 수행되도록 하거나 아예 수행되지 않도록 함으로써 데이터 무결성을 관리합니다.
- 서버의 공유 SQL 영역에서 사용할 수 있는 구문 분석된 PL/SQL 코드를 재사용할 수 있기 때문에 성능이 향상됩니다. 이후에 호출되는 서브 프로그램은 코드 구문을 다시 분석하지 않아도 됩니다. PL/SQL 코드의 구문은 컴파일 시에 분석되므로 런타임에 SQL 문의 구문 분석 오버헤드가 발생하는 것을 피할 수 있습니다. 데이터베이스에 대한 네트워크 호출 횟수가 감소하도록 코드를 작성할 수 있으므로 네트워크 트래픽이 감소합니다.
- 적절한 이름과 규약을 사용하여 루틴 작업을 설명하기 때문에 주석을 필요로 하는 경우가 감소하며 코드 명확성이 향상됩니다.

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.	함수라고 하는 서브 프로그램은 값을 반환해야 합니다.
파라미터를 사용할 수 없습니다.	파라미터를 사용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

익명 블록과 서브 프로그램의 차이

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

익명 블록은 영구적인 데이터베이스 객체가 아니며 오직 한 번 컴파일되고 실행됩니다. 또한 재사용할 수 있도록 데이터베이스에 저장되지 않습니다. 익명 블록을 재사용하려면 재컴파일과 실행이 수행되도록 익명 블록을 생성하는 스크립트를 다시 실행해야 합니다. 프로시저와 함수는 컴파일한 다음 컴파일된 형태로 데이터베이스에 저장할 수 있습니다. 프로시저와 함수는 변경된 경우에만 재컴파일됩니다. 데이터베이스에 저장되기 때문에 모든 응용 프로그램이 적절한 권한에 준하여 이러한 서브 프로그램을 사용할 수 있습니다. 프로시저가 파라미터를 받아들이도록 설계되었다면 호출하는 응용 프로그램이 파라미터를 프로시저로 전달할 수 있습니다. 마찬가지로 호출하는 응용 프로그램이 함수 또는 프로시저를 실행 중이라면 값을 검색할 수 있습니다.

단원 내용

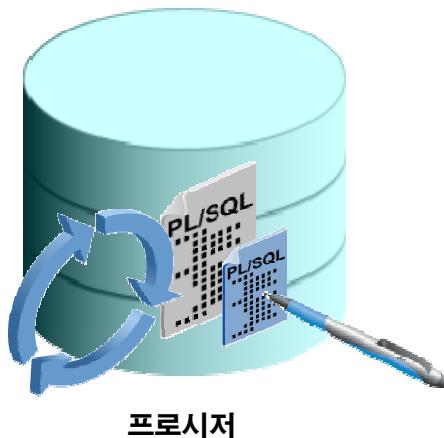
- 모듈화되고 계층화된 서브 프로그램 설계 사용 및 서브 프로그램의 이점 식별
- **프로시저 작업:**
 - 프로시저 생성 및 호출
 - 사용 가능한 파라미터 전달 모드 식별
 - 형식 및 실제 파라미터 사용
 - 위치 지정, 이름 지정 또는 혼합 표기법 사용
- 프로시저에서 예외 처리, 프로시저 제거 및 프로시저 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

프로시저란?

- 특정 작업을 수행하는 일종의 서브 프로그램입니다.
- 데이터베이스에 스키마 객체로 저장할 수 있습니다.
- 재사용성과 유지 관리 용이성이 증대됩니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

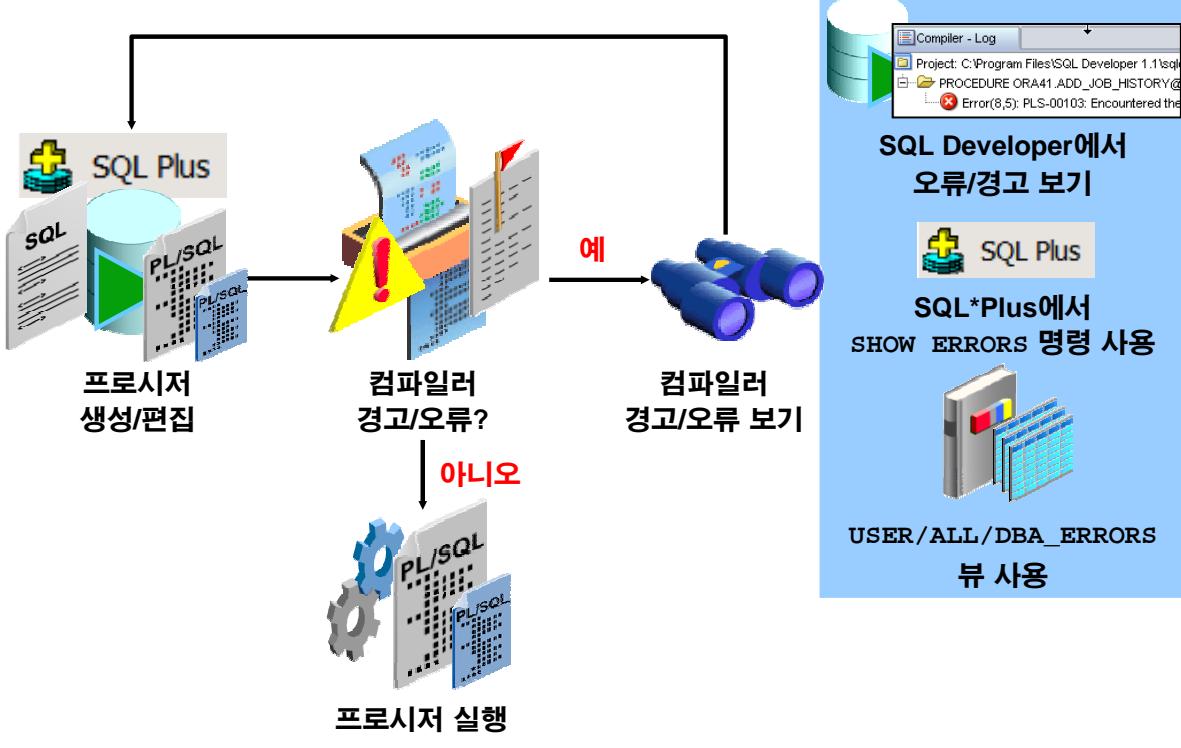
프로시저의 정의

프로시저는 파라미터(인수라고도 함)를 사용할 수 있는 명명된 PL/SQL 블록입니다. 일반적으로 프로시저를 사용하여 특정 작업을 수행할 수 있습니다. 프로시저는 헤더, 선언 섹션, 실행 섹션 및 선택적 예외 처리 섹션으로 구성되며 다른 PL/SQL 블록의 실행 섹션에 있는 프로시저 이름을 사용하여 호출됩니다.

프로시저는 컴파일한 다음 데이터베이스에 스키마 객체로 저장할 수 있습니다. Oracle Forms 및 Reports에서 프로시저를 사용할 경우 프로시저는 Oracle Forms 또는 Oracle Reports 실행 모듈 내에서 컴파일할 수 있습니다.

프로시저를 사용하면 재사용성과 유지 관리 용이성이 증대됩니다. 또한 검증되면 여러 응용 프로그램에 사용할 수 있습니다. 요구 사항이 변경될 경우 프로시저만 갱신하면 됩니다.

프로시저 생성: 개요



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 생성: 개요

SQL Developer 등의 도구를 사용하여 프로시저를 개발하려면 다음 단계를 따르십시오.

1. SQL Developer의 Object Navigator 트리 또는 SQL Worksheet 영역을 사용하여 프로시저를 생성합니다.
2. 프로시저를 컴파일합니다. 프로시저가 데이터베이스에서 생성되고 컴파일됩니다. CREATE PROCEDURE 문은 데이터베이스에서 소스 코드와 컴파일된 *m-code*를 생성하고 저장합니다. 프로시저를 컴파일하려면 Object Navigator 트리에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 Compile을 누릅니다.
3. 컴파일 오류가 발생하면 *m-code*가 저장되지 않으므로 소스 코드를 편집하여 수정해야 합니다. 컴파일 오류가 있는 프로시저는 호출할 수 없습니다. 슬라이드에 나와 있는 것처럼 SQL Developer, SQL*Plus 또는 해당하는 데이터 딕셔너리 뷰에서 컴파일 오류를 확인할 수 있습니다.
4. 성공적으로 컴파일되면 프로시저를 실행하여 원하는 작업을 수행합니다. SQL Developer를 사용하여 프로시저를 실행하거나 SQL*Plus에서 EXECUTE 명령을 사용할 수 있습니다.

참고: 컴파일 오류가 발생했는데 이미 이전에 CREATE PROCEDURE 문을 사용했다면 CREATE OR REPLACE PROCEDURE 문을 사용하여 기존 코드를 덮쳐 쓰십시오. DROP을 사용하여 프로시저를 먼저 삭제한 후에 CREATE PROCEDURE 문을 실행할 수도 있습니다.

SQL CREATE OR REPLACE 문으로 프로시저 생성

- CREATE 절을 사용하여 오라클 데이터베이스에 저장되는 독립형 프로시저를 생성합니다.
- OR REPLACE 옵션을 사용하여 기존 프로시저를 겹쳐씁니다.

```

CREATE [OR REPLACE] PROCEDURE procedure_name
  [(parameter1 [mode] datatype1,
    parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; ...]
BEGIN
  -- actions;
END [procedure_name];

```

PL/SQL block

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL CREATE OR REPLACE 문으로 프로시저 생성

CREATE PROCEDURE SQL 문을 사용하여 오라클 데이터베이스에 저장되는 독립형 프로시저를 생성할 수 있습니다. 프로시저는 소형 프로그램과 비슷하며 특정 작업을 수행합니다. 프로시저의 이름, 파라미터, 로컬 변수와 함께 코드를 포함하고 예외를 처리하는 BEGIN-END 블록을 지정합니다.

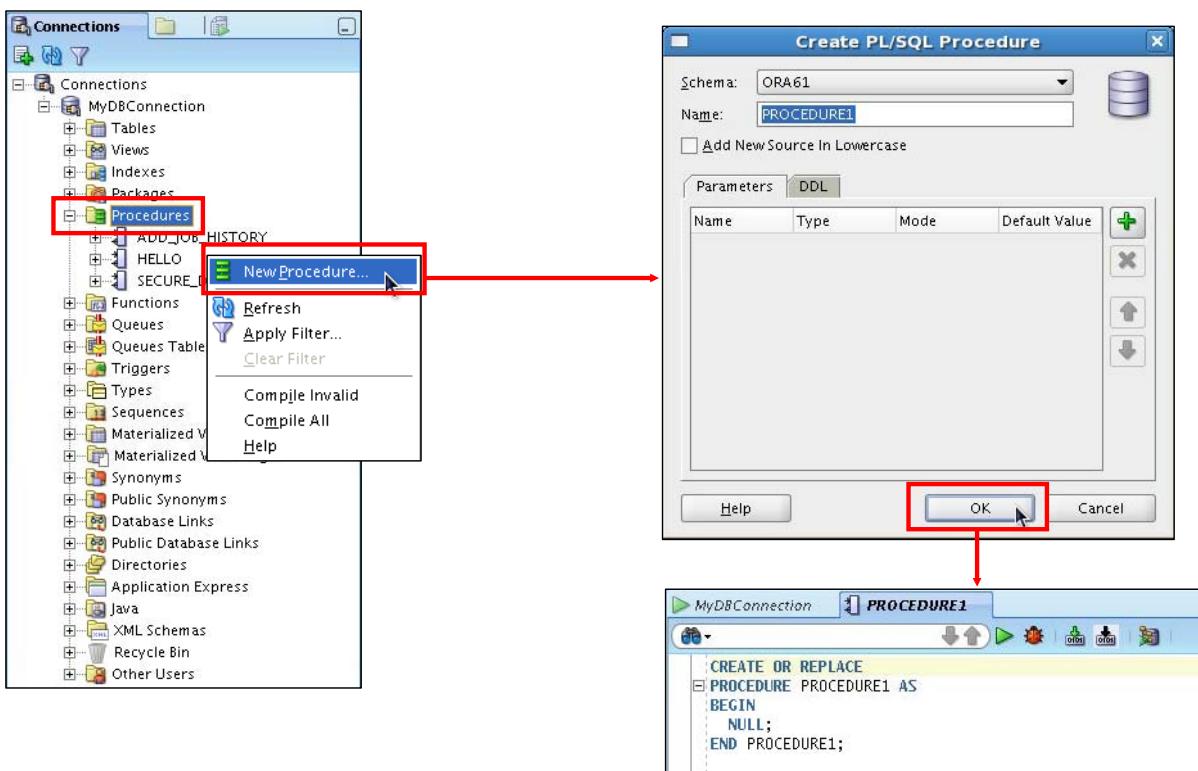
- PL/SQL 블록은 BEGIN으로 시작하여(경우에 따라 앞에 로컬 변수의 선언이 올 수도 있음) END 또는 END *procedure_name*으로 끝납니다.
- REPLACE 옵션은 기존 프로시저가 있을 경우 이를 삭제하고 명령문에 의해 생성되는 새 버전으로 바꾼다는 것을 나타냅니다. REPLACE 옵션을 사용해도 프로시저와 관련된 권한이 삭제되지는 않습니다.

기타 구문 요소

- parameter1*은 파라미터 이름을 나타냅니다.
- mode* 옵션은 파라미터가 사용되는 방식, 즉 IN(기본값), OUT 또는 IN OUT을 정의합니다.
- datatype1*은 자릿수 없이 파라미터 데이터 유형을 지정합니다.

참고: 파라미터는 로컬 변수로 간주할 수 있습니다. 치환 변수와 호스트(바인드) 변수는 PL/SQL 내장 프로시저 정의의 어느 부분에서도 참조될 수 없습니다. 자신이 객체 소유자이면서 CREATE [ANY] PROCEDURE 권한을 가지고 있다면 OR REPLACE 옵션을 사용할 때 객체 보안을 변경할 필요가 없습니다.

SQL Developer를 사용하여 프로시저 생성



Copyright © 2009, Oracle. All rights reserved.

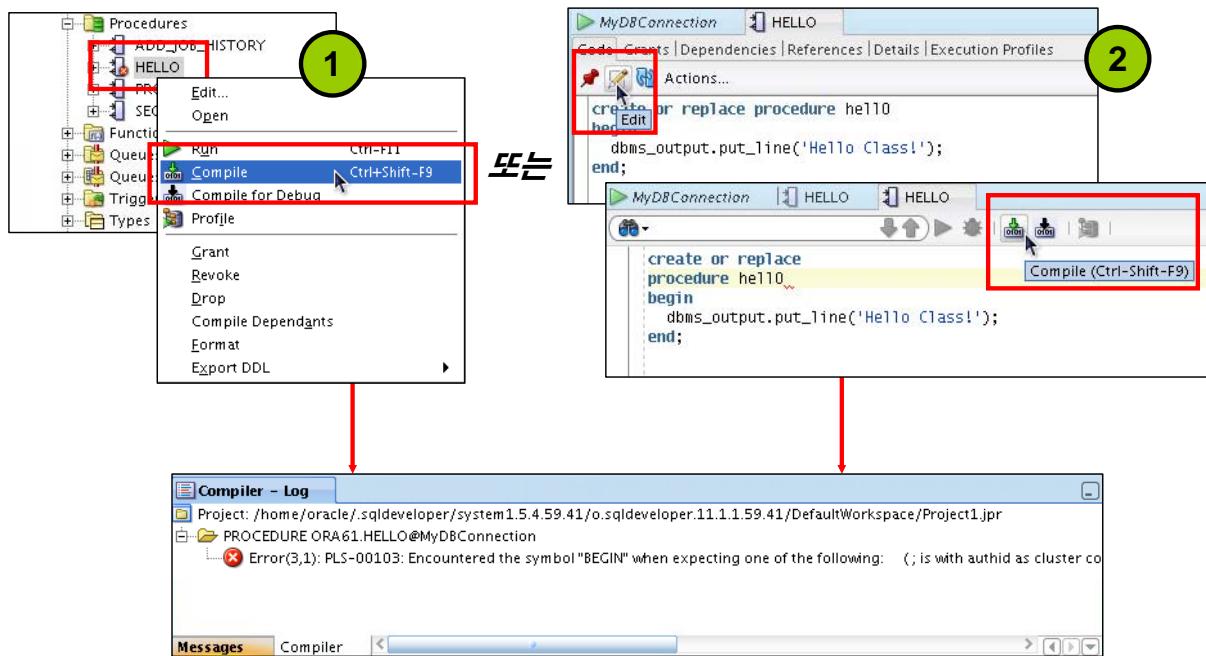
SQL Developer를 사용하여 프로시저 생성

1. **Connections** 템 페이지에서 **Procedures** 노드를 마우스 오른쪽 버튼으로 누릅니다.
2. 단축 메뉴에서 **New Procedure**를 선택합니다. **Create PL/SQL Procedure** 대화상자가 표시됩니다. 새 프로시저에 대해 정보를 지정하고 OK를 눌러 서브 프로그램을 생성한 다음 세부 정보를 입력할 수 있는 Editor window에 표시되도록 합니다.

Create PL/SQL Procedure 대화상자의 구성 요소는 다음과 같습니다.

- **Schema:** PL/SQL 서브 프로그램을 생성할 데이터베이스 스키마입니다.
- **Name:** 스키마 내에서 고유해야 하는 서브 프로그램의 이름입니다.
- **Add New Source in Lowercase:** 이 옵션을 선택하는 경우 새 텍스트는 입력 방식에 관계없이 모두 소문자로 표시됩니다. PL/SQL은 실행될 때 대소문자를 구분하지 않으므로 이 옵션은 코드의 모양에만 적용됩니다.
- **Parameters 템:** 파라미터를 추가하려면 Add (+) 아이콘을 누릅니다. 프로시저에서 각 파라미터를 생성하려면 파라미터 이름, 데이터 유형, 모드를 비롯하여 선택적으로 기본값을 지정합니다. Remove (X) 아이콘과 화살표 아이콘을 사용하면 리스트에서 파라미터를 삭제하거나 위 또는 아래로 이동할 수 있습니다.
- **DDL 템:** 이 템에는 서브 프로그램의 현재 정의를 반영하는 SQL 문의 읽기 전용 표시가 포함되어 있습니다.

SQL Developer에서 프로시저 컴파일 및 컴파일 오류 표시



ORACLE

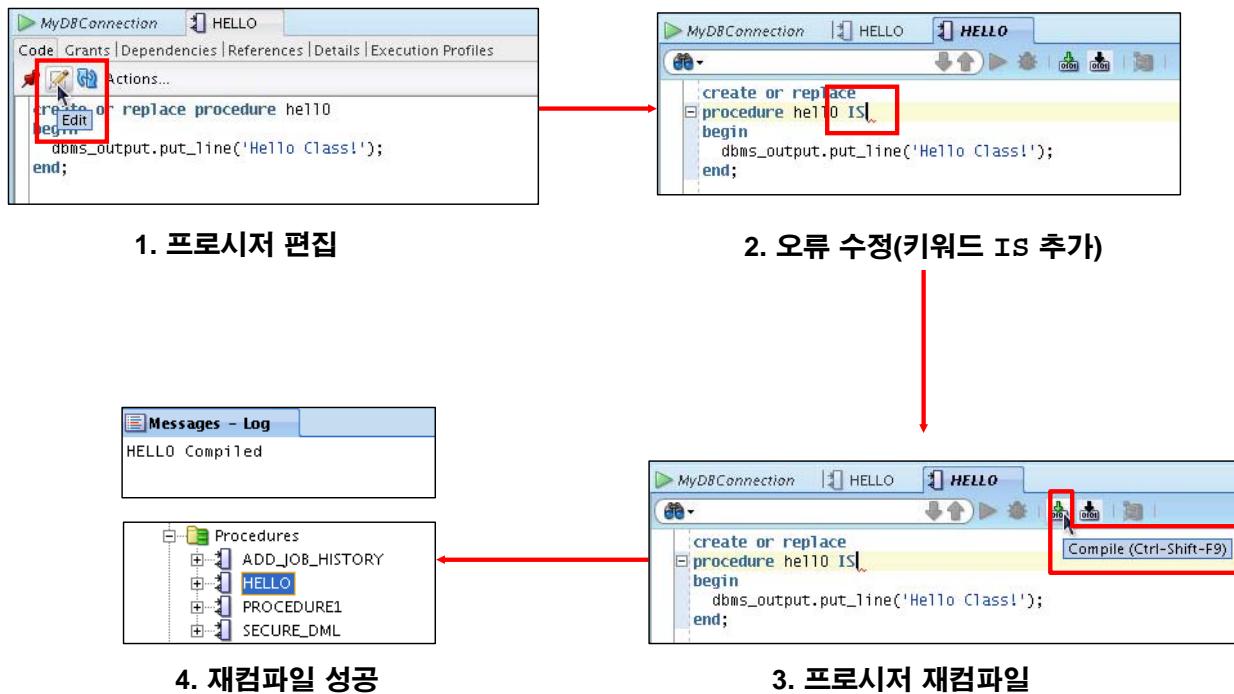
Copyright © 2009, Oracle. All rights reserved.

SQL Developer에서 프로시저 컴파일 및 컴파일 오류 표시

다음 두 가지 방법 중 하나를 사용하여 프로시저를 컴파일할 수 있습니다.

- Object Navigator 트리에서 Procedures 노드로 이동합니다. 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile을 선택합니다. 컴파일 메시지를 보려면 Compiler – Log 탭에서 Messages 하위 탭을 봅니다.
- 프로시저의 코드 도구 모음에 있는 Edit 아이콘을 사용하여 프로시저를 편집합니다. 필요한 내용을 편집한 후에 코드 도구 모음에서 Compile 아이콘을 누릅니다. 컴파일 메시지를 보려면 Compiler – Log 탭에서 Messages 하위 탭을 봅니다.

SQL Developer에서 컴파일 오류 해결



SQL Developer에서 컴파일 오류 해결

- 프로시저의 코드 도구 모음에 있는 Edit 아이콘을 사용하여 프로시저를 편집합니다. 새 프로시저 코드 탭이 읽기/쓰기 모드에서 열립니다.
- 필요한 내용을 수정합니다.
- 코드 도구 모음에서 Compile 아이콘을 누릅니다.
- 컴파일 메시지를 보려면 **Compiler - Log** 탭에서 Messages 하위 탭을 봅니다. 또한 프로시저가 성공적으로 컴파일되면 Object Navigator 트리에서 프로시저 이름에 있던 빨강색 X 표시가 없어집니다.

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.

파라미터 및 파라미터 모드란?

- PL/SQL 헤더에서 서브 프로그램 이름 뒤에 선언됩니다.
- 호출 환경과 서브 프로그램 간에 데이터를 전달합니다.
- 로컬 변수처럼 사용되지만 파라미터 전달 모드에 따라 달라집니다.
 - IN 파라미터 모드(기본값)는 처리할 값을 서브 프로그램에 제공합니다.
 - OUT 파라미터 모드는 호출자에게 값을 반환합니다.
 - IN OUT 파라미터 모드는 입력 값을 제공하며, 이 값은 수정된 값으로 반환(출력)될 수도 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

파라미터란?

파라미터는 데이터 값을 호출 환경에서 프로시저(또는 서브 프로그램)로 또는 프로시저(또는 서브 프로그램)에서 호출 환경으로 전송하는 데 사용되며, 서브 프로그램 헤더에서 이름과 로컬 변수 선언 섹션 사이에 선언됩니다.

파라미터는 세 가지 파라미터 전달 모드인 IN, OUT, IN OUT 중 하나를 따릅니다.

- IN 파라미터는 상수 값을 호출 환경에서 프로시저로 전달합니다.
- OUT 파라미터는 값을 프로시저에서 호출 환경으로 전달합니다.
- IN OUT 파라미터는 값을 호출 환경에서 프로시저로 전달하며, 동일한 파라미터를 사용하여 다른 값을 프로시저에서 다시 호출 환경으로 전달합니다.

파라미터는 특수한 형태의 로컬 변수로 간주할 수 있습니다. 해당 입력 값은 서브 프로그램이 호출될 때 호출 환경에 의해 초기화되며, 출력 값은 서브 프로그램이 제어를 호출자에게 반환할 때 호출 환경에 반환됩니다.

형식 파라미터 및 실제 파라미터

- **형식 파라미터:** 서브 프로그램 사양의 파라미터 리스트에 선언된 로컬 변수
- **실제 파라미터 또는 인수:** 호출하는 서브 프로그램의 파라미터 리스트에 사용되는 리터럴 값, 변수 및 표현식

```
-- Procedure definition, Formal parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
    . . .
END raise_sal;

-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_emp_id, 2000)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

형식 파라미터 및 실제 파라미터

형식 파라미터는 서브 프로그램 사양의 파라미터 리스트에 선언된 로컬 변수입니다. 첫번째 예제의 `raise_sal` 프로시저에서 `p_id` 변수 및 `p_ sal` 식별자는 형식 파라미터를 나타냅니다.

실제 파라미터는 호출하는 서브 프로그램의 파라미터 리스트에 제공되는 리터럴 값, 변수 및 표현식이 될 수 있습니다. 두번째 예제에서는 `raise_sal`을 호출합니다. 여기서 `v_emp_id` 변수는 형식 파라미터 `p_id`에 실제 파라미터 값을 제공하며, 2000은 `p_sal`에 실제 파라미터 값으로 제공됩니다. 실제 파라미터의 특징은 다음과 같습니다.

- 서브 프로그램이 호출되는 동안 형식 파라미터와 연관됩니다.
- `raise_sal(v_emp_id, raise+100);`과 같이 표현식이 될 수도 있습니다.

형식 파라미터와 실제 파라미터의 데이터 유형은 호환되어야 합니다. 필요할 경우 PL/SQL은 값을 할당하기 전에 실제 파라미터 값의 데이터 유형을 형식 파라미터의 데이터 유형으로 변환합니다.

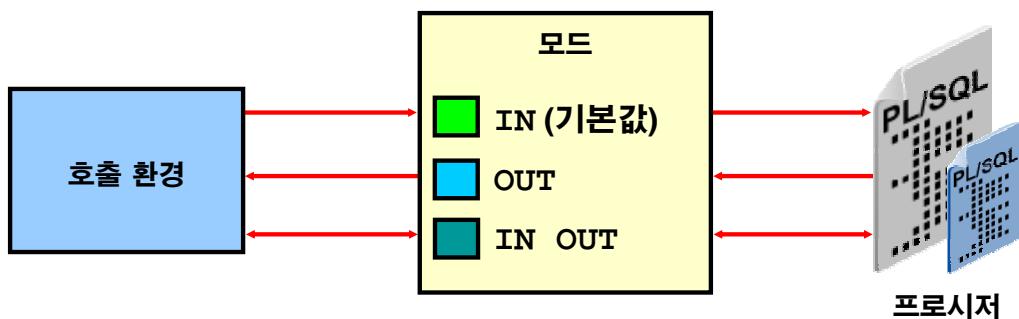
참고: 실제 파라미터를 실제 인수라고도 합니다.

프로시저 파라미터 모드

- 파라미터 모드는 파라미터 이름과 데이터 유형 사이인 형식
파라미터 선언 부분에 지정됩니다.
- 모드가 지정되어 있지 않을 경우 IN 모드가 기본값입니다.

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)
```

...



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 파라미터 모드

프로시저를 생성할 때 형식 파라미터는 변수 이름을 정의하며 이 변수 값은 PL/SQL 블록의 실행 섹션에 사용됩니다. 실제 파라미터는 입력 값을 제공하거나 출력 결과를 수신하기 위해 프로시저를 호출할 때 사용됩니다.

IN 파라미터 모드는 기본 전달 모드입니다. 즉, 파라미터 선언에서 모드를 지정하지 않으면 파라미터는 IN 파라미터로 간주됩니다. 파라미터 모드 OUT 및 IN OUT은 파라미터 선언 부분에 명시적으로 지정되어 있어야 합니다.

datatype 파라미터는 크기 사양 없이 지정되며 다음과 같이 지정될 수 있습니다.

- 명시적 데이터 유형
- %TYPE 정의 사용
- %ROWTYPE 정의 사용

참고: 여러 개의 형식 파라미터는 쉼표로 구분하여 선언할 수 있습니다.

파라미터 모드 비교

IN	OUT	IN OUT
기본 모드	지정해야 합니다.	지정해야 합니다.
값이 서브 프로그램에 전달됩니다.	값이 호출 환경에 반환됩니다.	서브 프로그램에 전달되는 값, 호출 환경에 반환되는 값
형식 파라미터가 상수로 동작합니다.	초기화되지 않은 변수	초기화된 변수
실제 파라미터가 리터럴, 표현식, 상수 또는 초기화된 변수가 될 수 있습니다.	변수여야 합니다.	변수여야 합니다.
기본값을 할당할 수 있습니다.	기본값을 할당할 수 없습니다.	기본값을 할당할 수 없습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

파라미터 모드 비교

선언에 모드가 지정되어 있지 않을 경우 IN 파라미터 모드가 기본 모드입니다. 파라미터 모드 OUT 및 IN OUT은 파라미터 선언 부분에 명시적으로 지정되어 있어야 합니다.

IN 모드에서는 형식 파라미터에 값을 할당할 수 없으며, 프로시저 본문에서 형식 파라미터를 수정할 수 없습니다. 기본적으로 IN 파라미터는 참조에 의해 전달됩니다. 형식 파라미터 선언 부분에서 IN 파라미터에 기본값을 할당할 수 있습니다. 이 경우 기본값을 적용하면 호출자가 파라미터에 값을 제공할 필요가 없습니다.

호출 환경에 반환하기 전에 OUT 또는 IN OUT 파라미터에 값을 할당해야 합니다. OUT 및 IN OUT 파라미터에는 기본값을 할당할 수 없습니다. OUT 및 IN OUT 파라미터를 사용하여 성능을 향상시키려면 NOCOPY 컴파일러 힌트를 사용하여 참조에 의해 전달되도록 요청할 수 있습니다.

참고: NOCOPY 사용법은 본 과정의 뒷부분에서 다릅니다.

IN 파라미터 모드 사용: 예제

```

CREATE OR REPLACE PROCEDURE raise_salary
  (p_id      IN employees.employee_id%TYPE,
   p_percent IN NUMBER)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * (1 + p_percent/100)
  WHERE employee_id = p_id;
END raise_salary;
/

```



```
EXECUTE raise_salary(176, 10)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IN 파라미터 사용: 예제

슬라이드의 예제는 두 개의 IN 파라미터를 사용하는 프로시저를 보여줍니다. 첫번째 슬라이드 예제를 실행하면 데이터베이스에 `raise_salary` 프로시저가 생성됩니다. 두번째 슬라이드 예제는 `raise_salary`를 호출하고 첫번째 파라미터 값으로 사원 ID 176을, 두번째 파라미터 값으로 10% 급여 인상을 제공합니다.

SQL Developer의 SQL Worksheet 또는 SQL*Plus를 사용하여 프로시저를 호출하려면 슬라이드의 두번째 코드 예제에 표시된 다음 `EXECUTE` 명령을 사용합니다.

다른 프로시저에서 프로시저를 호출하려면 호출 블록의 실행 섹션 내에 직접 호출을 사용하십시오. 새 프로시저를 호출하는 위치에 프로시저 이름과 실제 파라미터를 입력하십시오. 예를 들면 다음과 같습니다.

```

...
BEGIN
  raise_salary (176, 10);
END;

```

참고: IN 파라미터는 읽기 전용 값으로 호출 환경에서 프로시저로 전달됩니다. IN 파라미터 값을 변경하려고 하면 컴파일 타임 오류가 발생합니다.

OUT 파라미터 모드 사용: 예제

```

CREATE OR REPLACE PROCEDURE query_emp
(p_id      IN employees.employee_id%TYPE,
 p_name    OUT employees.last_name%TYPE,
 p_salary  OUT employees.salary%TYPE) IS
BEGIN
  SELECT last_name, salary INTO p_name, p_salary
  FROM   employees
  WHERE  employee_id = p_id;
END query_emp;
/

```

```

SET SERVEROUTPUT ON
DECLARE
  v_emp_name  employees.last_name%TYPE;
  v_emp_sal   employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name || ' earns ' ||
    to_char(v_emp_sal, '$999,999.00'));
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

OUT 파라미터 사용: 예제

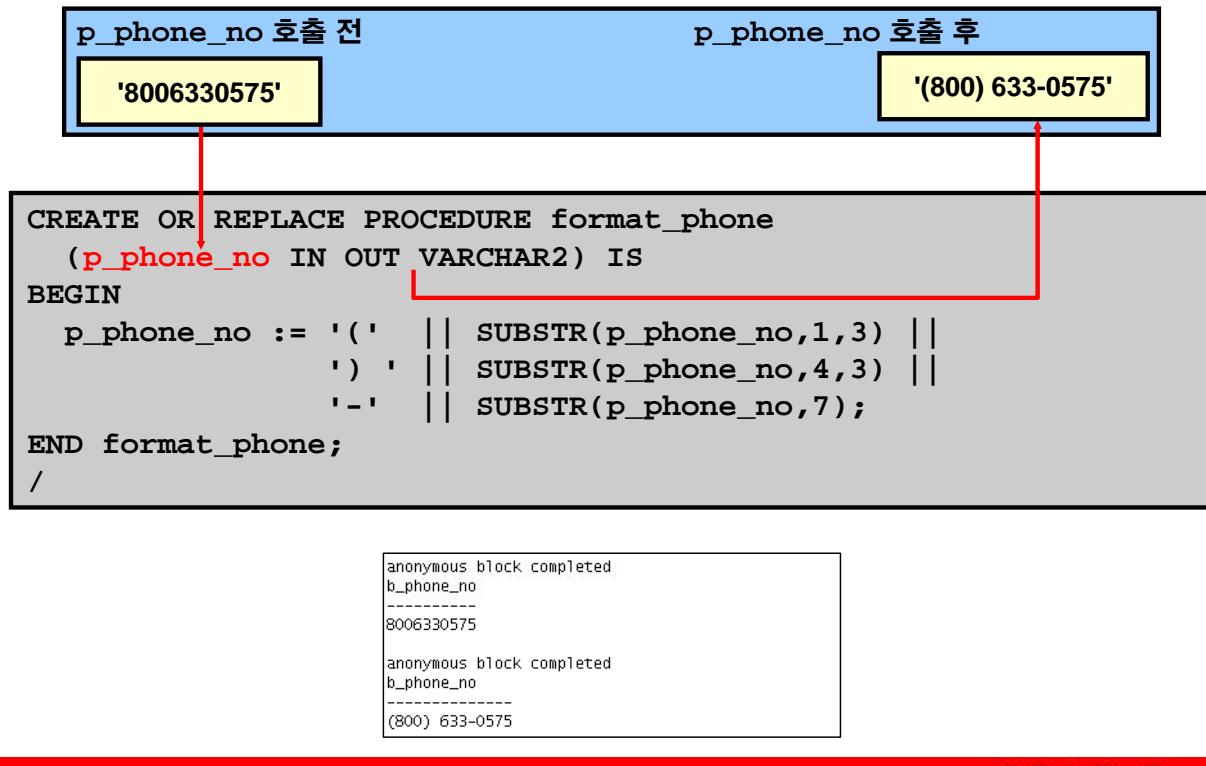
슬라이드의 예제에서는 OUT 파라미터를 사용하여 사원에 대한 정보를 검색하는 프로시저를 생성합니다. 이 프로시저에서는 값 171을 사원 ID로 사용하고 두 개의 OUT 파라미터로 ID가 171인 사원의 이름과 급여를 검색합니다. query_emp 프로시저에는 세 개의 형식 파라미터가 있습니다. 슬라이드의 두번째 코드 상자에 표시된 것과 같이, 이 중 두 개는 값을 호출 환경에 반환하는 OUT 파라미터입니다. 프로시저는 p_id 파라미터를 통해 사원 ID 값을 받아들입니다. v_emp_name 및 v_emp_salary 변수는 두 개의 해당 OUT 파라미터로 query에서 검색된 정보로 채워집니다. 다음은 슬라이드의 두번째 코드 예제에 있는 코드를 실행한 결과입니다. v_emp_name에 포함된 값은 Smith이고 v_emp_salary에 포함된 값은 7400입니다.



참고: OUT 파라미터에서 값을 검색하는 데 사용되는 실제 파라미터 변수의 데이터 유형이 반환되는 데이터 값을 포함할 수 있을 정도의 크기인지 확인하십시오.

IN OUT 파라미터 모드 사용: 예제

호출 환경



ORACLE

Copyright © 2009, Oracle. All rights reserved.

IN OUT 파라미터 사용: 예제

IN OUT 파라미터를 사용하면 개신 가능한 프로시저에 값을 전달할 수 있습니다. 호출 환경에서 제공되는 실제 파라미터 값은 변경되지 않은 원래 값이나 프로시저 내에서 설정되는 새 값을 반환할 수 있습니다.

참고: IN OUT 파라미터는 초기화된 변수로 동작합니다.

위 슬라이드에 표시된 예제는 IN OUT 파라미터를 사용하여 10개의 숫자로 된 문자열을 전화 번호로 사용하는 프로시저를 생성합니다. 프로시저는 처음 세 자는 괄호로 묶고 여섯번째 숫자 뒤에는 하이픈이 추가된 전화 번호를 반환합니다. 예를 들면, 전화 번호 문자열 8006330575는 (800) 633-0575로 반환됩니다.

다음 코드에서는 SQL*Plus의 `b_phone_no` 호스트 변수를 사용하여 `FORMAT_PHONE` 프로시저로 전달되는 입력 값을 제공합니다. 프로시저가 실행되고 `b_phone_no` 호스트 변수의 개신된 문자열이 반환됩니다. 다음 코드의 출력이 위 슬라이드에 표시됩니다.

```

VARIABLE b_phone_no VARCHAR2(15)
EXECUTE :b_phone_no := '8006330575'
PRINT b_phone_no
EXECUTE format_phone (:b_phone_no)
PRINT b_phone_no

```

OUT 파라미터 보기: DBMS_OUTPUT.PUT_LINE

서브 루틴 사용

DBMS_OUTPUT.PUT_LINE 프로시저를 호출하여 출력되는 PL/SQL 변수를 사용합니다.

```
SET SERVEROUTPUT ON

DECLARE
    v_emp_name employees.last_name%TYPE;
    v_emp_sal   employees.salary%TYPE;
BEGIN
    query_emp(171, v_emp_name, v_emp_sal);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
    DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_sal);
END;
```

```
anonymous block completed
Name: Smith
Salary: 7400
```

Copyright © 2009, Oracle. All rights reserved.

OUT 파라미터 보기: DBMS_OUTPUT 서브 루틴 사용

슬라이드의 예제에서는 SQL*Plus 또는 SQL Developer Worksheet의 OUT 파라미터에서 반환되는 값을 보는 방법을 보여줍니다.

익명 블록에서 PL/SQL 변수를 사용하여 OUT 파라미터 값을 검색할 수 있습니다.

DBMS_OUTPUT.PUT_LINE 프로시저는 PL/SQL 변수에 포함된 값을 인쇄하기 위해 호출됩니다. SET SERVEROUTPUT은 ON이어야 합니다.

OUT 파라미터 보기: SQL*Plus 호스트 변수 사용

1. SQL*Plus 호스트 변수를 사용합니다.
2. 호스트 변수를 사용하여 QUERY_EMP를 실행합니다.
3. 호스트 변수를 인쇄합니다.

```
VARIABLE b_name  VARCHAR2(25)
VARIABLE b_sal    NUMBER
EXECUTE query_emp(171, :b_name, :b_sal)
PRINT b_name b_sal
```

```
anonymous block completed
b_name
-----
Smith

b_sal
-----
7400
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

OUT 파라미터 보기: SQL*Plus 호스트 변수 사용

슬라이드의 예제에서는 VARIABLE 명령으로 생성하는 SQL*Plus 호스트 변수를 사용하는 방법을 보여줍니다. SQL*Plus 변수는 PL/SQL 블록 밖에 있으며 호스트 또는 바인드 변수라고도 합니다. PL/SQL 블록에서 호스트 변수를 참조하려면 변수 이름 앞에 콜론(:)을 붙여야 합니다. 호스트 변수에 저장된 값을 표시하려면 SQL*Plus PRINT 명령 뒤에 SQL*Plus 변수 이름(PL/SQL 명령이나 컨텍스트가 아니므로 콜론 없이 사용)을 사용해야 합니다.

참고: VARIABLE 명령에 대한 자세한 내용은 SQL*Plus Command Reference를 참조하십시오.

실제 파라미터 전달 시 사용 가능한 표기법

- 서브 프로그램을 호출할 때 다음 표기법을 사용하여 실제 파라미터를 작성할 수 있습니다.
 - 위치 지정: 실제 파라미터를 형식 파라미터와 동일한 순서로 나열합니다.
 - 이름 지정: 실제 파라미터를 임의의 순서로 나열하며 연관 연산자(=>)를 사용하여 이름 지정 형식 파라미터를 실제 파라미터와 연관시킵니다.
 - 혼합: 일부 실제 파라미터는 위치 지정으로 나열하고, 나머지는 이름 지정으로 나열합니다.
- Oracle Database 11g 이전에는 SQL의 호출에서 위치 지정 표기법만 지원되었습니다.
- Oracle Database 11g 부터는 이름 지정 및 혼합 표기법을 사용하여 SQL 문에서 PL/SQL 서브 루틴을 호출할 때 인수를 지정할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

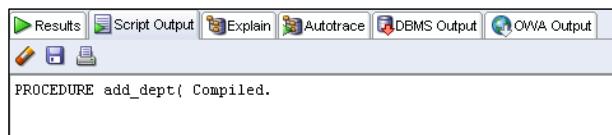
파라미터 전달 구문

서브 프로그램을 호출할 때 다음 표기법을 사용하여 실제 파라미터를 작성할 수 있습니다.

- 위치 지정: 형식 파라미터가 선언된 순서로 실제 파라미터 값을 나열합니다. 이 표기법은 간단하지만 파라미터(특히 리터럴)의 순서를 잘못 지정하면 오류를 검색하기가 어렵습니다. 프로시저의 파라미터 리스트가 변경되는 경우에는 코드를 변경해야 합니다.
- 이름 지정: 실제 값을 임의의 순서로 나열하며 연관 연산자를 사용하여 각각의 실제 파라미터를 이름별로 형식 파라미터와 연관시킵니다. PL/SQL 연관 연산자는 "등호" 다음에 "닫는 꺾쇠" 부호가 나오는 형태(=>)로 중간에 공백이 있으면 안됩니다. 파라미터의 순서는 중요하지 않습니다. 이 표기법은 길이가 길지만 코드를 보다 쉽게 읽고 유지 관리할 수 있습니다. 또한 파라미터의 순서를 재지정하거나 새 옵션 파라미터를 추가하는 등 프로시저의 파라미터 리스트가 변경되는 경우에 코드를 변경하지 않아도 될 수 있습니다.
- 혼합: 첫번째 파라미터 값은 위치별로 나열하고, 나머지는 이름 지정 메소드의 특수 구문을 사용하여 나열합니다. 이 표기법을 사용하면 일부 필수 파라미터 뒤에 일부 옵션 파라미터가 오는 프로시저를 호출할 수 있습니다.

실제 파라미터 전달: add_dept 프로시저 생성

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_name IN departments.department_name%TYPE,
    p_loc  IN departments.location_id%TYPE) IS
BEGIN
    INSERT INTO departments(department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name , p_loc );
END add_dept;
/
```



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

파라미터 전달: 예제

슬라이드의 예제에서 add_dept 프로시저는 두 개의 IN 형식 파라미터인 p_name과 p_loc를 선언합니다. 이 두 파라미터의 값은 INSERT 문에서 각각 department_name 및 location_id 열을 설정하는 데 사용됩니다.

실제 파라미터 전달: 예제

```
-- Passing parameters using the positional notation.
EXECUTE add_dept ('TRAINING', 2500)
```

anonymous block completed			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500
1 rows selected			

```
-- Passing parameters using the named notation.
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```

anonymous block completed			
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
290	EDUCATION		2400
1 rows selected			

ORACLE

Copyright © 2009, Oracle. All rights reserved.

실제 파라미터 전달: 예제

위치별로 실제 파라미터를 전달하는 작업은 슬라이드의 첫번째 코드 예제에서 add_dept를 실행하는 첫번째 호출에 나와 있습니다. 첫번째 실제 파라미터는 name 형식 파라미터에 대해 TRAINING 값을 제공합니다. 두번째 실제 파라미터 값 2500은 위치에 따라 loc 형식 파라미터에 할당됩니다.

이름 지정 표기법을 사용하여 파라미터를 전달하는 작업은 슬라이드의 두번째 코드 예제에 나와 있습니다. 두번째 형식 파라미터로 선언된 loc 실제 파라미터는 호출에서 이름별로 참조되며, 실제 값 2400과 연관됩니다. name 파라미터는 값 EDUCATION과 연관됩니다. 모든 파라미터 값이 지정되어 있을 경우 실제 파라미터의 순서는 아무 의미가 없습니다.

참고: 형식 파라미터에 기본값이 할당되어 있지 않을 경우 각 파라미터에 값을 제공해야 합니다. 형식 파라미터에 대해 기본값을 지정하는 방법은 다음 페이지에서 설명합니다.

파라미터에 DEFAULT 옵션 사용

- 파라미터에 대한 기본값 정의
- 위치 지정 및 이름 지정 파라미터 전달 구문을 함께 사용하여 융통성 제공

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_name departments.department_name%TYPE := 'Unknown',
    p_loc   departments.location_id%TYPE DEFAULT 1700)
IS
BEGIN
    INSERT INTO departments (department_id,
                           department_name, location_id)
    VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

```
EXECUTE add_dept
EXECUTE add_dept ('ADVERTISING', p_loc => 1200)
EXECUTE add_dept (p_loc => 1200)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

파라미터에 DEFAULT 옵션 사용

다음과 같이 IN 파라미터에 기본값을 할당할 수 있습니다.

- 할당 연산자(:=). 슬라이드의 name 파라미터에 대해 표시됨
- DEFAULT 옵션. 슬라이드의 p_loc 파라미터에 대해 표시됨

형식 파라미터에 기본값이 할당되면 파라미터에 실제 파라미터 값은 제공하지 않고도 프로시저를 호출할 수 있습니다. 따라서 기본값을 그대로 사용하거나 필요에 따라 재정의하여 서브 프로그램에 여러 개의 실제 파라미터를 전달할 수 있습니다. 먼저 기본값 없이 파라미터를 선언할 것을 권장합니다. 이후에는 프로시저에 대한 모든 호출을 변경할 필요 없이 기본값을 사용하여 형식 파라미터를 추가할 수 있습니다.

참고: OUT 및 IN OUT 파라미터에는 기본값을 할당할 수 없습니다.

슬라이드의 두번째 코드는 add_dept 프로시저를 호출하는 세 가지 방법을 보여줍니다.

- 첫번째 예제에서는 각 파라미터에 대해 기본값이 할당됩니다.
- 두번째 예제에서는 위치 및 이름 지정 표기법을 함께 사용하여 값이 할당됩니다. 이 경우 이름 지정 표기법을 사용하는 예제가 제시되었습니다.
- 마지막 예제에서 name 파라미터에는 기본값(Unknown)이 사용되고 p_loc 파라미터에는 제공된 값이 사용됩니다.

파라미터에 DEFAULT 옵션 사용(계속)

다음은 위 슬라이드에 있는 두번째 슬라이드 코드 예제의 결과입니다.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING	2500	
290	EDUCATION	2400	
300	Unknown	1700	
310	ADVERTISING	1200	
320	Unknown	1200	
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
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

32 rows selected

일반적으로 이름 지정 표기법을 사용하여 형식 파라미터의 기본값을 재정의할 수 있습니다.
그러나 형식 파라미터에 제공된 기본값이 없을 경우에는 실제 파라미터를 제공해야 합니다.

참고: 서브 프로그램 호출에서 모든 위치 지정 파라미터는 이름 지정 파라미터 앞에 와야 합니다.
그렇지 않을 경우 다음 예제에 표시된 것과 같이 오류 메시지를 수신하게 됩니다.

```
EXECUTE add_dept(p_name=>'new dept', 'new location')
```

```
Error starting at line 1 in command:
EXECUTE add_dept(p_name=>'new dept', 'new location')
Error report:
ORA-06550: line 1, column 36:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
06550. 00000 - "line %s, column %s:\n%s"
*Cause: Usually a PL/SQL compilation error.
*Action:
```

프로시저 호출

- 익명 블록, 다른 프로시저 또는 패키지를 사용하여 프로시저를 호출할 수 있습니다.
- 프로시저를 소유하거나 EXECUTE 권한을 가지고 있어야 합니다.

```
CREATE OR REPLACE PROCEDURE process_employees
IS
    CURSOR cur_emp_cursor IS
        SELECT employee_id
        FROM employees;
BEGIN
    FOR emp_rec IN cur_emp_cursor
    LOOP
        raise_salary(emp_rec.employee_id, 10);
    END LOOP;
    COMMIT;
END process_employees;
/
```

PROCEDURE process_employees Compiled.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 호출

프로시저는 다음을 사용하여 호출할 수 있습니다.

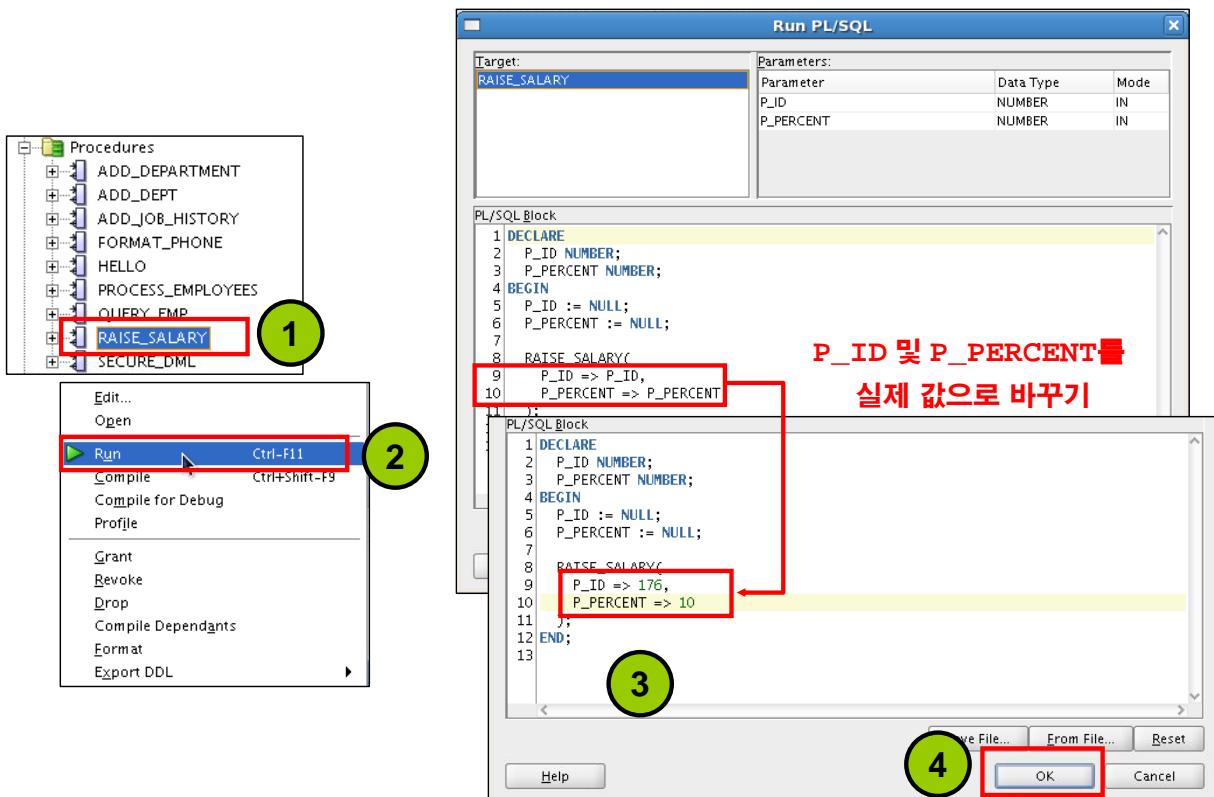
- 익명 블록
- 다른 프로시저 또는 PL/SQL 서브 프로그램

앞선 페이지의 예제는 익명 블록을 사용하는 방법 또는 SQL Developer나 SQL*Plus에서 EXECUTE 명령을 사용하는 방법을 보여주었습니다.

이 슬라이드의 예제는 다른 내장 프로시저에서 프로시저를 호출하는 방법을 보여줍니다. PROCESS_EMPLOYEES 내장 프로시저는 커서를 사용하여 EMPLOYEES 테이블의 모든 레코드를 처리한 다음 각 사원의 ID를 RAISE_SALARY 프로시저에 전달합니다. 그 결과 회사 전체적으로 급여가 10% 인상됩니다.

참고: 프로시저를 소유하거나 EXECUTE 권한을 가지고 있어야 합니다.

SQL Developer를 사용하여 프로시저 호출



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 프로시저 호출

슬라이드의 예제에서는 raise_salary 프로시저를 호출하여 다음과 같이 사원 176의 현재 급여(\$8,600)를 10% 인상합니다.

1. Procedures 노드에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 Run을 누릅니다. Run PL/SQL 대화상자가 표시됩니다.
2. PL/SQL 블록 섹션에서 연관 연산자 " $=>$ " 뒤에 표시되는 IN 및 IN/OUT 형식 파라미터 사양을 함수나 프로시저를 실행 또는 디버그하는 데 사용할 실제 값으로 변경합니다. 예를 들어 사원 176의 현재 급여를 8,600에서 10% 인상하려면 슬라이드에 나와 있는 것처럼 raise_salary 프로시저를 호출하면 됩니다. ID 및 PERCENT 입력 파라미터에 대해 값을 제공합니다. 이 두 파라미터는 각각 176과 10으로 지정됩니다. 이 작업은 표시되는 ID $=>$ ID를 ID $=>$ 176으로, 그리고 PERCENT $=>$ PERCENT를 PERCENT $=>$ 10으로 변경하여 수행합니다.
3. OK를 누릅니다. SQL Developer가 프로시저를 실행합니다. 다음과 같이 갱신된 급여인 9,460이 표시됩니다.

Results	Script Output	Explain	Autotrace	DBMS Output	OWA Output
Results:					
	EMPLOYEE_ID	SALARY			
1	176	9460			

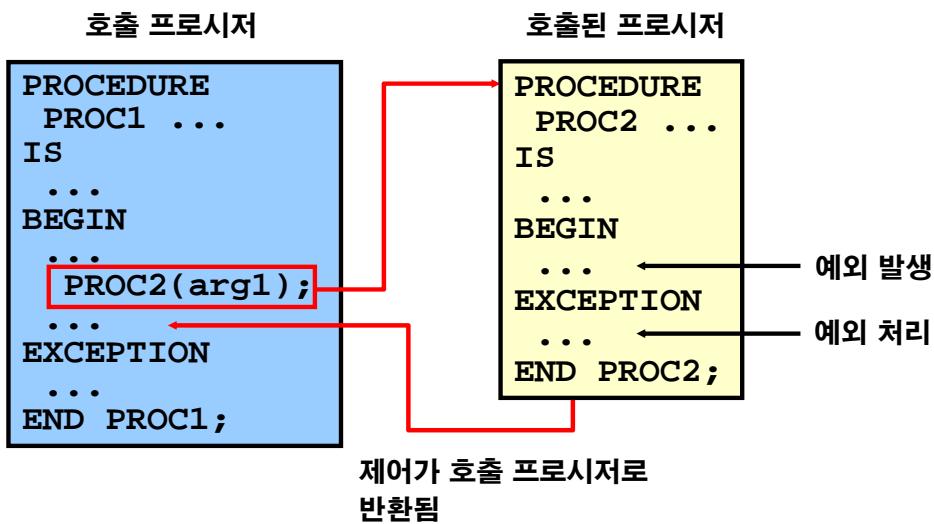
단원 내용

- 모듈화되고 계층화된 서브 프로그램 설계 사용 및 서브 프로그램의 이점 식별
- 프로시저 작업:
 - 프로시저 생성 및 호출
 - 사용 가능한 파라미터 전달 모드 식별
 - 형식 및 실제 파라미터 사용
 - 위치 지정, 이름 지정 또는 혼합 표기법 사용
- 프로시저에서 예외 처리, 프로시저 제거 및 프로시저 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

처리된 예외



ORACLE

Copyright © 2009, Oracle. All rights reserved.

처리된 예외

다른 프로시저에서 호출되는 프로시저를 개발할 경우, 처리된 예외와 처리되지 않은 예외가 트랜잭션과 호출 프로시저에 미치는 영향을 파악하고 있어야 합니다.

호출된 프로시저에서 예외가 발생할 경우 제어는 즉시 해당 블록의 예외 섹션으로 이동합니다. 발생한 예외에 대한 처리기를 예외 섹션에서 제공하면 처리된 예외로 간주됩니다.

예외가 발생하여 처리될 경우 코드 흐름은 다음과 같습니다.

1. 예외가 발생합니다.
2. 제어가 예외 처리기에 전달됩니다.
3. 블록이 종료됩니다.
4. 아무 문제도 발생하지 않은 것처럼 호출 프로그램/블록이 계속 실행됩니다.

트랜잭션이 시작된 경우, 즉 예외가 발생한 프로시저를 실행하기 전에 DML(데이터 조작어) 문이 실행된 경우 해당 트랜잭션은 영향을 받지 않습니다. 예외가 발생하기 전에 프로시저 내에서 수행된 DML 작업은 롤백됩니다.

참고: 예외 섹션에서 COMMIT 또는 ROLLBACK 작업을 실행하면 트랜잭션을 명시적으로 종료할 수 있습니다.

처리된 예외: 예제

```

CREATE PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Err: adding dept: '|| p_name);
END;

```

```

CREATE PROCEDURE create_departments IS
BEGIN
    add_department('Media', 100, 1800);
    add_department('Editing', 99, 1800); X
    add_department('Advertising', 101, 1800); ✓
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

처리된 예외: 예제

슬라이드에 나와 있는 두 프로시저는 다음과 같습니다.

- `add_department` 프로시저는 오라클 시퀀스에서 새 부서 멤버를 할당하여 새 부서 레코드를 생성하고 `p_name`, `p_mgr` 및 `p_loc` 파라미터를 사용하여 `department_name`, `manager_id` 및 `location_id` 열 값을 각각 설정합니다.
- `create_departments` 프로시저는 `add_department` 프로시저에 대한 호출을 사용하여 여러 개의 부서를 생성합니다.

`add_department` 프로시저는 발생한 예외를 모두 자체 처리기에서 처리합니다.

`create_departments`가 실행되면 다음과 같은 출력이 생성됩니다.

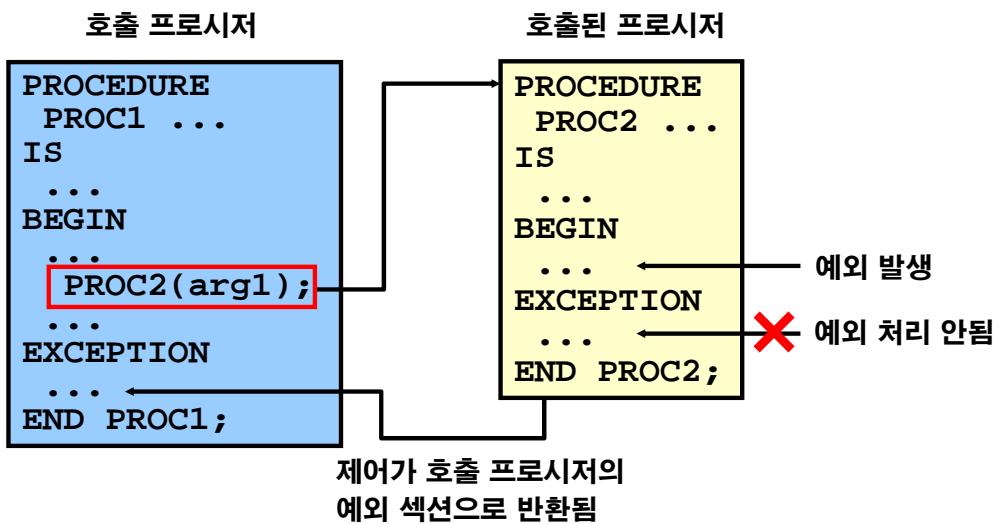
```

Added Dept: Media
Err: adding dept: Editing
Added Dept: Advertising

```

`manager_id`가 99인 `Editing` 부서의 경우 `manager_id`의 Foreign Key 무결성 제약 조건 위반으로 인해 관리자의 ID가 99일 수 없으므로 삽입되지 않습니다. 예외는 `add_department` 프로시저에서 처리되었기 때문에 `create_department` 프로시저는 계속 실행됩니다. `location_id`가 1800인 조건으로 `DEPARTMENTS` 테이블을 query하면 `Media` 및 `Advertising`은 추가되지만 `Editing` 레코드는 추가되지 않은 것으로 나타납니다.

처리되지 않은 예외



ORACLE

Copyright © 2009, Oracle. All rights reserved.

처리되지 않은 예외

앞서 설명했듯이 호출된 프로시저에서 예외가 발생할 경우 제어는 즉시 해당 블록의 예외 섹션으로 이동합니다. 발생한 예외에 대한 처리기를 예외 섹션에서 제공하지 않으면 예외는 처리되지 않습니다. 코드 흐름은 다음과 같습니다.

1. 예외가 발생합니다.
2. 예외 처리기가 없기 때문에 블록이 종료됩니다. 프로시저 내에서 수행된 DML 작업은 롤백됩니다.
3. 예외는 호출 프로시저의 예외 섹션으로 전달됩니다. 즉, 컨트롤이 호출 블록의 예외 섹션(있는 경우)으로 반환됩니다.

예외가 처리되지 않은 경우 호출 프로시저와 호출된 프로시저에 있는 DML 문은 호스트 변수 변경 사항과 함께 롤백됩니다. 처리되지 않은 예외가 발생한 PL/SQL 코드를 호출하기 전에 실행된 DML 문은 영향을 받지 않습니다.

처리되지 않은 예외: 예제

```
SET SERVEROUTPUT ON
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
    INSERT INTO DEPARTMENTS (department_id,
        department_name, manager_id, location_id)
    VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
    DBMS_OUTPUT.PUT_LINE('Added Dept: ' || p_name);
END;
```

```
CREATE PROCEDURE create_departments_noex IS
BEGIN
    add_department_noex('Media', 100, 1800);
    add_department_noex('Editing', 99, 1800);
    add_department_noex('Advertising', 101, 1800); X
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

처리되지 않은 예외: 예제

위 슬라이드에 표시된 코드 예제는 예외 셙션이 없는 `add_department_noex`를 보여줍니다. 이 경우 `Editing` 부서가 추가될 때 예외가 발생합니다. 서브 프로그램 중 하나에서 예외가 처리되지 않았기 때문에 `DEPARTMENTS` 테이블에 새 부서 레코드가 추가되지 않습니다. `create_departments_noex` 프로시저를 실행하면 다음과 유사한 결과가 생성됩니다.

```
Error starting at line 8 in command:
EXECUTE create_departments_noex
Error report:
ORA-02291: integrity constraint (ORA62.DEPT_MGR_FK) violated - parent key not found
ORA-06512: at "ORA62.ADD_DEPARTMENT_NOEX", line 4
ORA-06512: at "ORA62.CREATE_DEPARTMENTS_NOEX", line 4
ORA-06512: at line 1
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause: A foreign key value has no matching primary key value.
*Action: Delete the foreign key or add a matching primary key.
```

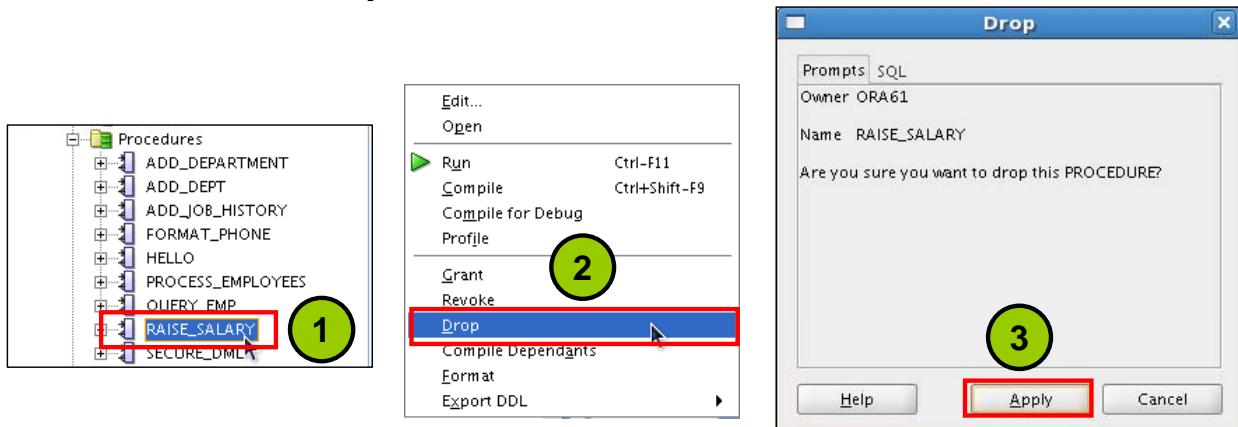
이 결과에서는 `Media` 부서가 추가된 것으로 나타나지만, 호출된 서브 프로그램 중 하나에서 예외가 처리되지 않았기 때문에 이 작업은 롤백됩니다.

프로시저 제거: DROP SQL 문 또는 SQL Developer 사용

- **DROP 문 사용:**

```
DROP PROCEDURE raise_salary;
```

- **SQL Developer 사용:**



Copyright © 2009, Oracle. All rights reserved.

ORACLE

프로시저 제거: DROP SQL 문 또는 SQL Developer 사용

내장 프로시저가 더 이상 필요하지 않은 경우 다음과 같이 DROP PROCEDURE SQL 문 다음에 프로시저 이름을 붙여 프로시저를 제거할 수 있습니다.

`DROP PROCEDURE procedure_name`

또한 SQL Developer를 사용하여 다음과 같이 내장 프로시저를 삭제할 수도 있습니다.

1. **Procedures** 노드에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 **Drop**을 누릅니다. **Drop** 대화상자가 표시됩니다.
2. **Apply**를 눌러 프로시저를 삭제합니다.

참고

- 성공 여부와 관계없이 DROP PROCEDURE와 같은 DDL (데이터 정의어) 명령을 실행하면 룰백될 수 없는 보류 중인 트랜잭션이 커밋됩니다.
- 삭제 작업의 결과를 확인하려면 **Procedures** 노드를 refresh해야 할 수도 있습니다. **Procedures** 노드를 refresh하려면 **Procedures** 노드에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 **Refresh**를 누릅니다.

데이터 딕셔너리 뷰를 사용하여 프로시저 정보 보기

```
DESCRIBE user_source
```

DESCRIBE user_source		
Name	Null	Type
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)

4 rows selected

```
SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```

TEXT
1 PROCEDURE add_dept(2 p_name IN departments.department_name%TYPE, 3 p_loc IN departments.location_id%TYPE) IS 4 5 BEGIN 6 INSERT INTO departments(department_id, department_name, location_id) 7 VALUES (departments_seq.NEXTVAL, p_name, p_loc); 8 END add_dept;

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리에서 프로시저 보기

PL/SQL 서브 프로그램의 소스 코드는 데이터 딕셔너리 테이블에 저장되어 있습니다. 소스 코드로 컴파일에 성공한 PL/SQL 프로시저나 컴파일에 실패한 PL/SQL 프로시저에 액세스할 수 있습니다. 데이터 딕셔너리에 저장된 PL/SQL 소스 코드를 보려면 다음 테이블에서 SELECT 문을 실행하십시오.

- 자신이 소유한 PL/SQL 코드를 표시하려면 USER_SOURCE 테이블 조회
- 서브 프로그램 코드 소유자로부터 EXECUTE 권한을 부여 받은 PL/SQL 코드를 표시하려면 ALL_SOURCE 테이블 조회

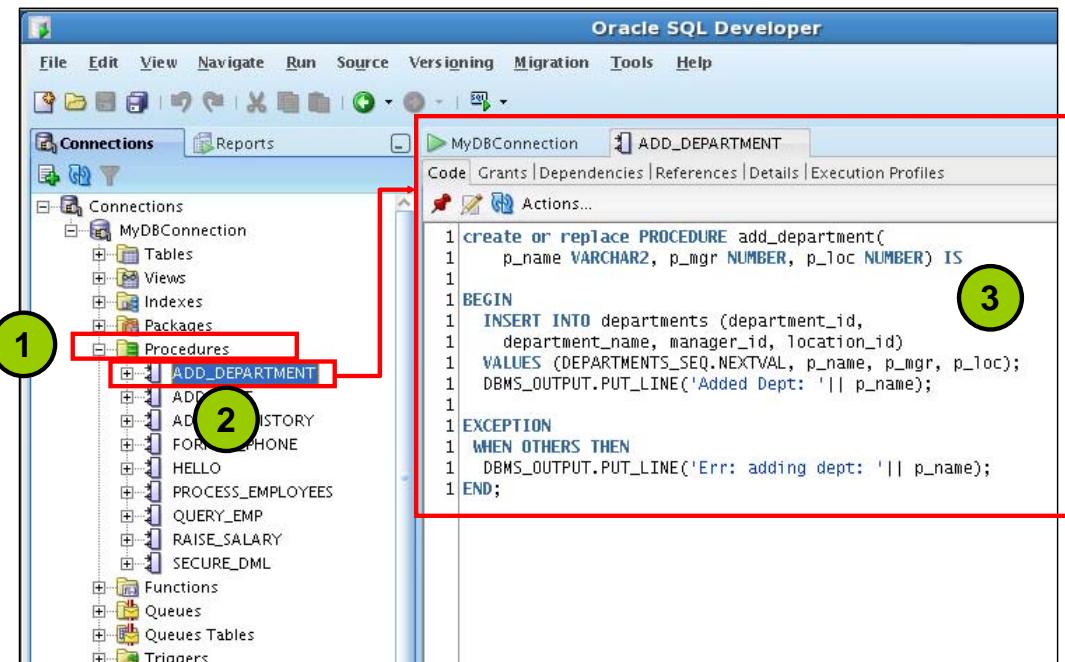
위의 query 예제는 USER_SOURCE 테이블에서 제공하는 모든 열을 보여줍니다.

- TEXT 열은 PL/SQL 소스 코드 행을 포함하고 있습니다.
- NAME 열은 서브 프로그램의 이름을 대문자로 포함하고 있습니다.
- TYPE 열은 PROCEDURE 또는 FUNCTION 같은 서브 프로그램 유형을 포함하고 있습니다.
- LINE 열은 각 소스 코드 행에 대한 행 번호를 저장합니다.

ALL_SOURCE 테이블은 위에 나열된 열 이외에도 OWNER 열을 제공합니다.

참고: Oracle PL/SQL 내장 패키지 또는 WRAP 유ти리티를 사용하여 소스 코드가 래핑된 PL/SQL의 경우에는 소스 코드를 표시할 수 없습니다. WRAP 유ти리티는 PL/SQL 소스 코드를 사람이 해독할 수 없는 형태로 변환합니다.

SQL Developer를 사용하여 프로시저 정보 보기



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 프로시저 정보 보기

SQL Developer에서 프로시저의 코드를 보려면 다음 단계를 따르십시오.

1. **Connections** 탭에서 **Procedures** 노드를 누릅니다.
2. 프로시저 이름을 누릅니다.
3. 슬라이드에 나와 있는 것처럼 프로시저 코드가 **Code** 탭에 표시됩니다.

퀴즈

형식 파라미터는 호출하는 서브 프로그램의 파라미터 리스트에 사용되는 리터럴 값, 변수 및 표현식입니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 2

형식 파라미터 및 실제(또는 인수) 파라미터

형식 파라미터: 서브 프로그램 사양의 파라미터 리스트에 선언된 로컬 변수

실제 파라미터: 호출하는 서브 프로그램의 파라미터 리스트에 사용되는 리터럴 값, 변수 및 표현식

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 모듈화되고 계층화된 서브 프로그램 설계의 이점 식별
- 프로시저 생성 및 호출
- 형식 및 실제 파라미터 사용
- 위치 지정, 이름 지정 또는 혼합 표기법을 사용하여 파라미터 전달
- 사용 가능한 파라미터 전달 모드 식별
- 프로시저에서 예외 처리
- 프로시저 제거
- 프로시저 정보 표시



Copyright © 2009, Oracle. All rights reserved.

연습 1 개요: 프로시저 생성, 컴파일 및 호출

이 연습에서는 다음 내용을 다룹니다.

- 내장 프로시저를 생성하여 다음 작업을 수행합니다.
 - 제공된 파라미터 값을 사용하여 테이블에 새 행 삽입
 - 테이블에서 제공된 파라미터 값과 일치하는 행에 대한 데이터 갱신
 - 테이블에서 제공된 파라미터 값과 일치하는 행 삭제
 - 제공된 파라미터 값을 기준으로 테이블 query 및 데이터 검색
- 프로시저에서 예외 처리
- 프로시저 컴파일 및 호출

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 1: 개요

이 연습에서는 DML 및 Query 명령을 실행하는 프로시저를 생성하고 컴파일하고 호출합니다. 프로시저에서 예외를 처리하는 방법도 배웁니다.

프로시저를 실행할 때 컴파일 오류가 발생하면 SQL Developer의 Compiler-Log 탭을 사용할 수 있습니다.

참고: 이 연습에서는 SQL Developer를 사용할 것을 권장합니다.

중요

이 과정의 모든 연습과 연습 해답에서는 SQL Developer의 SQL Worksheet 영역을 사용하여 프로시저, 함수 등의 객체를 생성한다고 가정합니다. SQL Worksheet 영역에서 객체를 생성할 때 새 객체가 Navigator 트리에 표시되도록 하려면 객체 노드를 refresh해야 합니다. 새로 생성된 객체를 컴파일하려면 Navigator 트리에서 객체 이름을 마우스 오른쪽 버튼으로 누른 다음 단축 메뉴에서 Compile을 선택합니다. 예를 들어 SQL Worksheet 영역에서 코드를 입력하여 프로시저를 생성한 후에 Run Script 아이콘을 누르거나 F5 키를 눌러 코드를 실행합니다. 이렇게 하면 프로시저가 생성되고 컴파일됩니다.

Navigator 트리에서 PROCEDURES 노드를 사용하여 프로시저 등의 객체를 생성한 다음 해당 프로시저를 컴파일할 수도 있습니다. Navigator 트리를 사용하여 객체를 생성하면 새로 생성된 객체가 자동으로 표시됩니다.

함수 생성 및 서브 프로그램 디버그

2

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 프로시저와 함수의 차이 설명
- 함수 사용법 설명
- 내장 함수 생성
- 함수 호출
- 함수 제거
- SQL Developer Debugger의 기본 기능 이해

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 함수를 생성, 호출 및 유지 관리하는 방법에 대해 설명합니다.

단원 내용

- **함수 작업:**
 - 프로시저와 함수의 차이 설명
 - 함수 사용법 설명
 - 내장 함수 생성, 호출 및 제거
- SQL Developer Debugger 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

내장 함수 개요

함수:

- 값을 반환하는 명명된 PL/SQL 블록입니다.
- 반복 실행을 위해 데이터베이스에 스키마 객체로 저장할 수 있습니다.
- 표현식의 일부로 호출되거나 또 다른 서브 프로그램에 파라미터 값을 제공하는 데 사용됩니다.
- PL/SQL 패키지로 그룹화할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

내장 함수 개요

함수는 파라미터를 사용하고, 호출되며, 값을 반환할 수 있는 명명된 PL/SQL 블록입니다. 일반적으로 함수를 사용하여 값을 계산할 수 있습니다. 함수와 프로시저의 구조는 유사합니다. 함수는 호출 환경에 값을 반환해야 하지만, 프로시저는 호출 환경에 값을 반환할 수도 있고 반환하지 않을 수도 있습니다. 프로시저처럼 함수는 헤더, 선언 섹션, 실행 섹션 및 옵션 예외 처리 섹션으로 구성됩니다. 헤더에는 RETURN 절이 있어야 하며 실행 섹션에는 적어도 한 개의 RETURN 문이 있어야 합니다.

함수는 반복 실행을 위해 데이터베이스에 스키마 객체로 저장할 수 있습니다. 데이터베이스에 저장되어 있는 함수를 내장 함수(Stored Function)라고 합니다. 함수를 클라이언트측 응용 프로그램에 생성할 수도 있습니다.

함수를 사용하면 재사용성과 유지 관리 용이성이 향상됩니다. 또한 겸중되면 여러 응용 프로그램에 사용할 수 있습니다. 처리 요구 사항이 변경될 경우 함수만 갱신하면 됩니다.

함수는 또한 SQL 표현식의 일부나 PL/SQL 표현식의 일부로 호출될 수 있습니다. SQL 표현식의 컨텍스트에서 부작용을 제어하려면 함수가 특정 규칙을 준수해야 합니다. PL/SQL 표현식에서 함수 식별자는 전달되는 파라미터에 따라 값이 달라지는 변수처럼 동작합니다.

함수 및 프로시저는 PL/SQL 패키지로 그룹화될 수 있습니다. 패키지를 사용하면 코드를 재사용하고 유지 관리하기가 훨씬 더 쉬워집니다. 패키지는 "패키지 생성" 및 "패키지 작업" 단원에서 다룹니다.

함수 생성

PL/SQL 블록에는 적어도 하나의 RETURN 문이 있어야 합니다.

```

CREATE [OR REPLACE] FUNCTION function_name
  [(parameter1 [mode1] datatype1, . . .)]
  RETURN datatype IS|AS
    [local_variable_declarations;
     . . .]
  BEGIN
    -- actions;
    RETURN expression;
  END [function_name];

```

PL/SQL 블록

ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 생성 구문

함수는 값을 반환하는 PL/SQL 블록입니다. 함수 선언과 일치하는 데이터 유형을 가진 값을 반환하려면 RETURN 문을 제공해야 합니다.

새 함수는 CREATE FUNCTION 문을 사용하여 생성되는데, 이 문은 파라미터 리스트를 선언할 수 있고 한 개의 값을 반환해야 하며 표준 PL/SQL 블록에 의해 수행되는 작업을 정의해야 합니다. CREATE FUNCTION 문에 대해 다음 사항을 고려해야 합니다.

- REPLACE 옵션은 기존 함수가 있을 경우 이를 삭제하고 명령문에 의해 생성되는 새 버전으로 바꾼다는 것을 나타냅니다.
- RETURN 데이터 유형은 크기 사양을 포함해서는 안됩니다.
- PL/SQL 블록은 로컬 변수 선언 뒤에서 BEGIN으로 시작되고 END로 끝나는데, 이 뒤에 *function_name*이 올 수도 있습니다.
- 적어도 하나의 RETURN *expression* 문이 있어야 합니다.
- 내장 함수의 PL/SQL 블록에서는 호스트나 바인드 변수를 참조할 수 없습니다.

참고: OUT 및 IN OUT 파라미터 모드를 함수와 함께 사용할 수는 있지만, 이러한 모드를 함수와 함께 사용하는 것은 바람직한 프로그래밍 방법이 아닙니다. 그러나 함수에서 여러 개의 값을 반환해야 할 경우에는 PL/SQL 레코드나 PL/SQL 테이블과 같은 복합 데이터 구조를 이용해서 값을 반환해 보십시오.

프로시저와 함수의 차이

프로시저	함수
PL/SQL 문으로 실행	표현식의 일부로 호출
헤더에 RETURN 절이 없음	헤더에 RETURN 절을 포함해야 함
출력 파라미터를 사용하여 값(있을 경우)을 전달할 수 있음	단일 값을 반환해야 함
값 없이 RETURN 문을 포함할 수 있음	적어도 하나의 RETURN 문을 포함해야 함

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

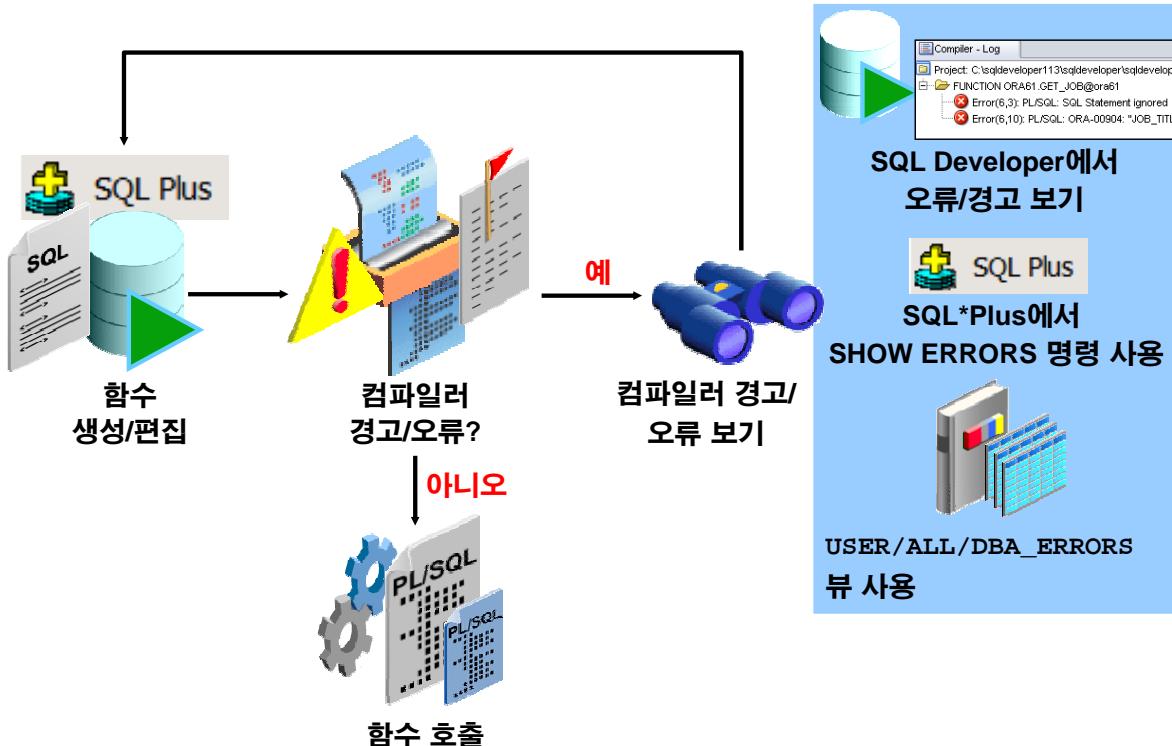
프로시저와 함수의 차이

나중에 연습에 사용할 수 있도록 일련의 작업을 저장하는 프로시저를 생성하십시오. 프로시저는 호출 환경에서 전달되거나 호출 환경으로 전달되는 파라미터를 여러 개 포함할 수도 있고 포함하지 않을 수도 있지만, 값을 반환할 필요는 없습니다. 프로시저는 프로시저 작업을 돋는 함수를 호출할 수 있습니다.

참고: 단일 OUT 파라미터를 포함하는 프로시저는 값을 반환하는 함수로 재작성하는 것이 좋습니다.

호출 환경에 반환되어야 하는 값을 계산하려는 경우 함수를 생성하십시오. 함수는 호출 환경에서 전달되는 파라미터를 여러 개 포함할 수도 있고 포함하지 않을 수도 있습니다. 함수는 일반적으로 단일 값만 반환하는데 같은 RETURN 문을 통해 반환됩니다. SQL 문에 사용되는 함수는 OUT 또는 IN OUT 모드 파라미터를 사용해서는 안됩니다. 출력 파라미터를 사용하는 함수를 PL/SQL 프로시저나 블록에 사용할 수는 있지만, SQL 문에 사용할 수는 없습니다.

함수 생성 및 실행: 개요



ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 생성 및 실행: 개요

슬라이드의 도표에서는 함수의 생성 및 실행을 위한 다음과 같은 기본 단계를 보여줍니다.

1. SQL Developer의 Object Navigator 트리 또는 SQL Worksheet 영역을 사용하여 함수를 생성합니다.
2. 함수를 컴파일합니다. 함수가 데이터베이스에 생성됩니다. CREATE FUNCTION 문은 데이터베이스에서 소스 코드와 컴파일된 *m-code*를 생성하고 저장합니다. 함수를 컴파일하려면 Object Navigator 트리에서 함수 이름을 마우스 오른쪽 버튼으로 누르고 Compile을 누릅니다.
3. 컴파일 경고나 오류가 있는 경우 다음 방법 중 하나를 사용하여 경고 또는 오류를 보고 해결할 수 있습니다.
 - a. SQL Developer 인터페이스 사용(Compiler – Log 탭)
 - b. SHOW ERRORS SQL*Plus 명령 사용
 - c. USER/ALL/DBA_ERRORS 뷰 사용
4. 컴파일이 성공하면 함수를 호출하여 원하는 값을 반환합니다.

CREATE FUNCTION 문을 사용하여

내장 함수 생성 및 호출: 예제

```
CREATE OR REPLACE FUNCTION get_sal
(p_id employees.employee_id%TYPE) RETURN NUMBER IS
v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO v_sal
  FROM employees
  WHERE employee_id = p_id;
  RETURN v_sal;
END get_sal;
/
```

FUNCTION get_sal Compiled.

```
-- Invoke the function as an expression or as
-- a parameter value.
```

```
EXECUTE dbms_output.put_line(get_sal(100))
```

anonymous block completed
24000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

내장 함수: 예제

단일 입력 파라미터를 사용하여 get_sal 함수가 생성되고 급여를 숫자로 반환합니다. 표시된 것과 같이 명령을 실행하거나 명령을 스크립트 파일로 저장한 다음 이 스크립트를 실행하여 get_sal 함수를 생성하십시오.

get_sal 함수는 로컬 변수에 할당된 값을 반환하는 단일 RETURN 문을 사용하는 일반적인 프로그래밍 방식을 따릅니다. 함수에 예외 셙션이 있으면 RETURN 문도 포함되어 있을 수 있습니다.

함수는 호출 환경에 값을 반환하므로 함수를 PL/SQL 표현식의 일부로 호출하십시오. 두 번째 코드 상자에서는 SQL*Plus EXECUTE 명령을 사용하여 DBMS_OUTPUT.PUT_LINE 프로시저를 호출하는데, 이 프로시저의 인수는 get_sal 함수에서 반환된 값입니다. 이 경우 ID가 100인 사원의 급여를 계산하기 위해 get_sal이 먼저 호출됩니다. 반환된 급여 값은 결과(SET SERVEROUTPUT ON을 실행한 경우)를 표시하는 DBMS_OUTPUT.PUT_LINE 파라미터의 값으로 제공됩니다.

참고: 함수는 항상 값을 반환해야 합니다. 위 예제에서는 제공된 ID의 행을 찾을 수 없을 경우 값을 반환하지 않습니다. 하지만 예외 처리기를 생성하여 값을 반환하는 것이 이상적입니다.

서로 다른 방법을 사용하여 함수 실행

```
-- As a PL/SQL expression, get the results using host variables
```

```
VARIABLE b_salary NUMBER
EXECUTE :b_salary := get_sal(100)
```

```
anonymous block completed
b_salary
-----
24000
```

```
-- As a PL/SQL expression, get the results using a local
-- variable
```

```
SET SERVEROUTPUT ON
DECLARE
    sal employees.salary%type;
BEGIN
    sal := get_sal(100);
    DBMS_OUTPUT.PUT_LINE('The salary is: ' || sal);
END;
/
```

```
anonymous block completed
The salary is: 24000
```

Copyright © 2009, Oracle. All rights reserved.

서로 다른 방법을 사용하여 함수 실행

심사숙고하여 설계된 함수는 강력한 생성자가 될 수 있습니다. 함수는 다음 방법으로 호출할 수 있습니다.

- **PL/SQL 표현식의 일부로 사용:** 호스트 또는 로컬 변수를 사용하여 함수에서 반환된 값을 보유할 수 있습니다. 위 슬라이드에 표시된 첫번째 예제에서는 호스트 변수를 사용하고, 두번째 예제에서는 익명 블록에서 로컬 변수를 사용합니다.

참고: SQL 문에서 함수를 사용할 때 함수에 적용되는 이점과 제한 사항은 다음 몇 페이지에서 다릅니다.

서로 다른 방법을 사용하여 함수 실행

```
-- Use as a parameter to another subprogram

EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

```
-- Use in a SQL statement (subject to restrictions)

SELECT job_id, get_sal(employee_id)
FROM employees;
```

JOB_ID	GET_SAL(EMPLOYEE_ID)
SH_CLERK	2600
SH_CLERK	2600
AD_ASST	4400
MK_MAN	13000

```
...
SH_CLERK 3100
SH_CLERK 3000
107 rows selected
```

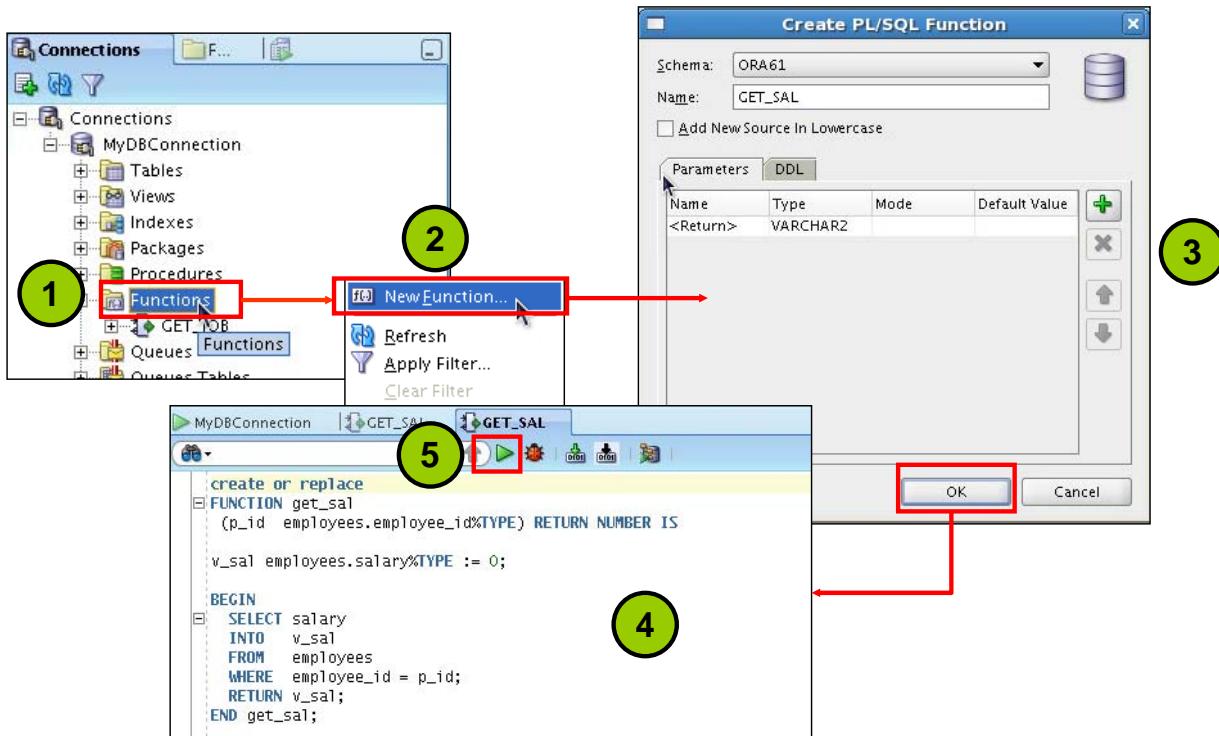
ORACLE

Copyright © 2009, Oracle. All rights reserved.

서로 다른 방법을 사용하여 함수 실행(계속)

- 다른 서브 프로그램에 대한 파라미터로 사용: 위 슬라이드에 표시된 첫번째 예제에서 이와 같이 사용되고 있습니다. `get_sal` 함수는 모든 해당 인수와 함께 `DBMS_OUTPUT.PUT_LINE` 프로시저에 필요한 파라미터에 중첩됩니다. 이것은 *Oracle Database 11g: SQL Fundamentals I* 과정에서 설명한 중첩 함수의 개념에서 유래된 것입니다.
- SQL 문의 표현식으로 사용: 슬라이드의 두번째 예제는 함수를 SQL 문에서 단일 행 함수로 사용하는 방법을 보여줍니다.

SQL Developer를 사용하여 함수 생성 및 컴파일



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 함수 생성 및 컴파일

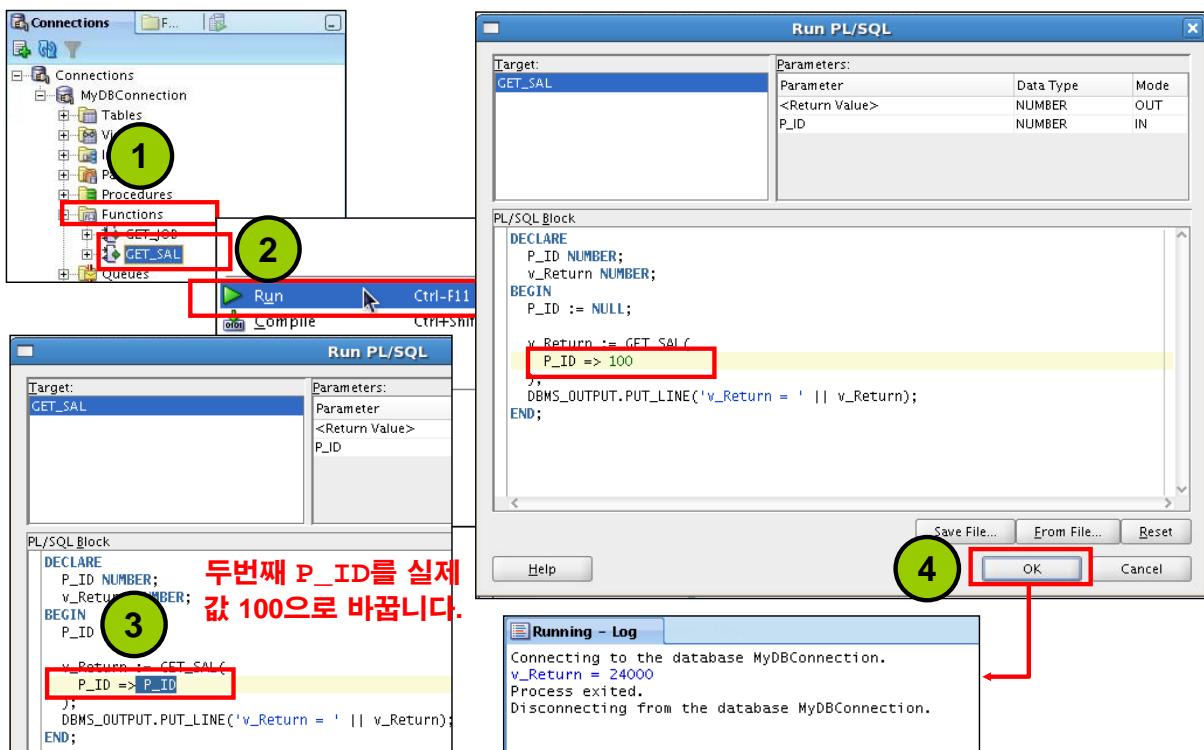
다음 단계를 수행하여 SQL Developer에서 새 함수를 만들 수 있습니다.

1. Functions 노드를 마우스 오른쪽 버튼으로 누릅니다.
2. 단축 메뉴에서 New Function을 선택합니다. Create PL/SQL Function 대화상자가 표시됩니다.
3. + 아이콘을 사용하여 스키마, 함수 이름 및 파라미터 리스트를 선택하고 OK를 누릅니다. 함수의 코드 편집기가 표시됩니다.
4. 함수 코드를 입력합니다.
5. 함수를 컴파일하려면 Compile 아이콘을 누릅니다.

참고

- SQL Developer에서 새 함수를 생성하려면 SQL Worksheet에서 코드를 입력한 후에 Run Script 아이콘을 눌러도 됩니다.
- SQL Developer에서 함수를 생성하는 방법에 대한 자세한 내용은 해당하는 온라인 도움말 항목 "Create PL/SQL Subprogram Function or Procedure"를 참조하십시오.

SQL Developer를 사용하여 함수 실행



Copyright © 2009, Oracle. All rights reserved.

ORACLE

SQL Developer를 사용하여 함수 실행

다음 단계를 수행하여 SQL Developer에서 함수를 실행할 수 있습니다.

1. **Functions** 노드를 누릅니다.
2. 함수 이름을 마우스 오른쪽 버튼으로 누르고 **Run**을 선택합니다. **Run PL/SQL** 대화상자가 표시됩니다.
3. 슬라이드 예제에 나와 있는 것처럼 두번째 파라미터 이름을 실제 파라미터 값으로 바꿉니다.
4. OK를 누릅니다.

참고: SQL Developer에서 함수를 실행하는 방법에 대한 자세한 내용은 온라인 도움말 항목 "Running and Debugging Functions and Procedures"를 참조하십시오.

SQL 문에서 유저 정의 함수를 사용하는 경우의 이점

- SQL만으로 표현하기에 복잡하거나 표현이 어려운 경우에 SQL의 확장 기능으로 함수로 표현하는 것이 도움이 될 수 있습니다.
- 응용 프로그램에서 데이터를 필터링하는 경우와 반대로, 데이터 필터링을 위해 WHERE 절에 사용할 경우 효율성을 높일 수 있습니다.
- 데이터 값을 조작할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 문에서 유저 정의 함수를 사용하는 경우의 이점

SQL 문은 SQL 표현식이 허용되는 곳이라면 어디든지 PL/SQL 유저 정의 함수를 참조할 수 있습니다. 예를 들면 유저 정의 함수는 UPPER()와 같은 내장 SQL 함수가 있을 수 있는 곳이라면 어디든지 사용할 수 있습니다.

이점

- 너무 복잡하거나, 부적절하거나, SQL을 사용할 수 없는 계산이 허용됩니다. 함수는 데이터를 응용 프로그램으로 읽어 들이는 대신 Oracle 서버 내에서 복잡한 데이터 분석을 처리하므로 데이터 독립성을 증대시킵니다.
- 응용 프로그램이 아닌 query에서 함수를 수행하므로 query 효율성이 증대됩니다.
- 문자열을 인코딩하고 문자열에서 작동하는 함수를 사용함으로써 새로운 유형의 데이터(예: 위도 및 경도)를 조작할 수 있습니다.

SQL 표현식에 함수 사용: 예제

```

CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
  RETURN NUMBER IS
BEGIN
  RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;

```

FUNCTION tax(value Compiled.			
EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarra	7700	616
112	Urman	7800	624
113	Popp	6900	552

6 rows selected

Copyright © 2009, Oracle. All rights reserved.

SQL 표현식 함수: 예제

위 슬라이드에 표시된 예제는 소득세를 계산하는 `tax` 함수를 생성하는 방법을 보여줍니다. 이 함수는 `NUMBER` 파라미터를 사용하여 간단한 균일 소득세율 8%를 토대로 계산된 소득세를 반환합니다.

슬라이드 예제의 코드를 SQL Developer에서 실행하려면 SQL Worksheet에 코드를 입력한 후 **Run Script** 아이콘을 누릅니다. `tax` 함수는 ID가 100인 부서에 소속된 사원들에 대한 사원 ID, 성 및 급여와 함께 `SELECT` 절에서 표현식으로 호출됩니다. `tax` 함수가 반환하는 결과는 query의 정규 출력을 사용하여 표시됩니다.

SQL 문에서 유저 정의 함수 호출

유저 정의 함수는 내장 단일 행 함수처럼 동작하며 다음 위치에서 사용할 수 있습니다.

- Query의 SELECT 리스트 또는 절
- WHERE 및 HAVING 절의 조건식
- Query의 CONNECT BY, START WITH, ORDER BY 및 GROUP BY 절
- INSERT 문의 VALUES 절
- UPDATE 문의 SET 절

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 문에서 유저 정의 함수 호출

PL/SQL 유저 정의 함수는 다음 예제와 같이 내장 단일 행 함수를 호출할 수 있는 모든 SQL 표현식에서 호출할 수 있습니다.

```
SELECT employee_id, tax(salary)FROM employees
WHERE tax(salary) > (SELECT MAX(tax(salary)))
                     FROM employees
WHERE department_id = 30)
ORDER BY tax(salary) DESC;
```

EMPLOYEE_ID	TAX(SALARY)
100	1920
101	1360
102	1360
145	1120
146	1080
201	1040
205	960
147	960
108	960
168	920

10 rows selected

SQL 표현식에서 함수를 호출할 때의 제한 사항

- **SQL 표현식에서 호출할 수 있는 유저 정의 함수의 경우:**
 - 데이터베이스에 저장해야 합니다.
 - PL/SQL 고유 유형이 아닌 적합한 SQL 데이터 유형을 가진 IN 파라미터만 사용해야 합니다.
 - PL/SQL 고유 유형이 아닌 적합한 SQL 데이터 유형을 반환해야 합니다.
- **SQL 문에서 함수를 호출할 경우:**
 - 함수를 소유하거나 EXECUTE 권한을 가지고 있어야 합니다.
 - SQL 문의 병렬 실행을 허용하려면 PARALLEL_ENABLE 키워드를 활성화해야 할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 표현식에서 함수를 호출할 때의 제한 사항

SQL 표현식에서 호출할 수 있는 유저 정의 PL/SQL 함수는 다음과 같은 요구 사항을 충족해야 합니다.

- 함수는 데이터베이스에 저장해야 합니다.
- 함수 파라미터는 IN이어야 하며 적합한 SQL 데이터 유형이어야 합니다.
- 함수는 적합한 SQL 데이터 유형을 반환해야 합니다. BOOLEAN, RECORD 또는 TABLE과 같은 PL/SQL 고유 데이터 유형일 수 없습니다. 함수 파라미터에도 동일한 제한 사항이 적용됩니다.

SQL 문에서 함수를 호출할 경우 다음과 같은 제한 사항이 적용됩니다.

- 파라미터는 위치 지정 표기법을 사용해야 합니다. 이름 지정 표기법은 지원되지 않습니다.
- 함수를 소유하고 있거나 함수에 대한 EXECUTE 권한을 가지고 있어야 합니다.
- 함수를 사용하여 SQL 문의 병렬 실행을 허용하려면 PARALLEL_ENABLE 키워드를 활성화해야 할 수 있습니다. 병렬 슬래이브마다 함수 로컬 변수의 전용(private) 복사본이 있습니다.

유저 정의 함수에 대한 기타 제한 사항은 다음과 같습니다. CREATE TABLE 또는 ALTER TABLE 문의 CHECK 제약 조건 절에서 호출할 수 없습니다. 또한 열 기본값을 지정하는 데 사용할 수 없습니다. SQL 문에서는 내장 함수만 호출할 수 있습니다. 내장 프로시저는 위의 요구 사항을 충족하는 함수에서 호출될 경우에만 호출할 수 있습니다.

SQL 표현식에서 함수를 호출할 때의 부작용 제어

제한 사항:

- SELECT 문에서 호출된 함수는 DML 문을 포함할 수 없습니다.
- 테이블 T의 UPDATE 또는 DELETE 문에서 호출된 함수는 같은 테이블 T의 DML을 query하거나 포함할 수 없습니다.
- SQL 문에서 호출된 함수는 트랜잭션을 종료할 수 없습니다.
즉, COMMIT 또는 ROLLBACK 작업을 실행할 수 없습니다.

참고: 이러한 제한 사항을 위반하는 서브 프로그램의 호출도 함수에서 허용되지 않습니다.

Copyright © 2009, Oracle. All rights reserved.

SQL 표현식에서 함수를 호출할 때의 부작용 제어

내장 함수를 호출하는 SQL 문을 실행하려면 Oracle 서버는 함수에 특정 부작용이 있는지 여부를 알아야 합니다. 부작용이란 데이터베이스 테이블이 부적절하게 변경되는 것입니다.

함수가 SQL 문의 표현식에서 호출될 경우 추가 제한 사항이 적용됩니다.

- SELECT 문 또는 병렬 UPDATE나 DELETE 문에서 호출하는 함수는 데이터베이스 테이블을 수정할 수 없습니다.
- UPDATE 또는 DELETE 문에서 호출하는 함수는 해당 명령문이 수정한 데이터베이스 테이블을 query하거나 수정할 수 없습니다.
- SELECT, INSERT, UPDATE 또는 DELETE 문에서 호출하는 함수는 다른 서브 프로그램이나 다음과 같은 SQL 트랜잭션 제어문을 통해 직접 또는 간접적으로 실행할 수 없습니다.
 - COMMIT 또는 ROLLBACK 문
 - 세션 제어문(예: SET ROLE)
 - 시스템 제어문(예: ALTER SYSTEM)
 - DDL 문(예: CREATE). DDL 문 다음에는 자동 커밋이 이어지기 때문입니다.

SQL에서 함수를 호출할 때의 제한 사항: 예제

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
  RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                        email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;
```

```
FUNCTION dml_call_sql(p_sal Compiled.

Error starting at line 1 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
 WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA62.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA62.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:  A trigger (or a user defined plsql function that is referenced in
        this statement) attempted to look at (or modify) a table that was
        in the middle of being modified by the statement which fired it.
*Action: Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL에서 함수를 호출할 때의 제한 사항: 예제

위 슬라이드에 표시된 `dml_call_sql` 함수는 `EMPLOYEES` 테이블에 새 레코드를 삽입한 다음 100씩 증가하는 입력 급여 값을 반환하는 `INSERT` 문을 포함합니다. 이 함수는 함수에서 반환된 금액으로 사원 170의 급여를 수정하는 `UPDATE` 문에서 호출됩니다. 이 `UPDATE` 문은 테이블이 변경되는 중임(즉, 동일한 테이블에서 변경이 이미 진행 중임)을 나타내는 오류로 인해 실패합니다. 다음 예제에서 `query_call_sql` 함수는 `EMPLOYEES` 테이블의 `SALARY` 열을 `query`합니다.

```
CREATE OR REPLACE FUNCTION query_call_sql(p_a NUMBER) RETURN
NUMBER IS
  v_s NUMBER;
BEGIN
  SELECT salary INTO v_s FROM employees
  WHERE employee_id = 170;
  RETURN (v_s + p_a);
END;
```

다음 `UPDATE` 문에서 호출하면 위 슬라이드에 표시된 것과 유사한 오류 메시지가 반환됩니다.

```
UPDATE employees SET salary = query_call_sql(100)
WHERE employee_id = 170;
```

SQL의 이름 지정 및 혼합 표기법

- PL/SQL에서는 숫자 자리, 이름 지정 또는 혼합 표기법을 사용하여 서브 루틴 호출에 인수를 지정할 수 있습니다.
- Oracle Database 11g 이전에는 SQL의 호출에서 숫자 자리 표기법만 지원되었습니다.
- Oracle Database 11g 부터는 SQL 문에서 PL/SQL 서브 루틴을 호출할 때 이름 지정 및 혼합 표기법을 사용하여 인수를 지정할 수 있습니다.
- 대부분 기본값이 포함되어 있는 긴 파라미터 리스트의 경우에는 옵션 파라미터의 값을 생략할 수 있습니다.
- 각 호출 사이트에서 옵션 파라미터의 기본값이 중복되지 않도록 할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL의 이름 지정 및 혼합 표기법: 예제

```
CREATE OR REPLACE FUNCTION f(
    p_parameter_1 IN NUMBER DEFAULT 1,
    p_parameter_5 IN NUMBER DEFAULT 5)
RETURN NUMBER
IS
    v_var number;
BEGIN
    v_var := p_parameter_1 + (p_parameter_5 * 2);
    RETURN v_var;
END f;
/
```

FUNCTION f(Compiled.

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

F(P_PARAMETER_5=>10)

21
1 rows selected

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문의 이름 지정 및 혼합 표기법 사용 예제

슬라이드 예제에서 SQL SELECT 문 내의 함수 f에 대한 호출에는 이름 지정 표기법이 사용됩니다. Oracle Database 11g 이전에는 이름 지정 표기법 또는 혼합 표기법을 사용하여 SQL 문 내에서 함수에 파라미터를 전달할 수 없었습니다. Oracle Database 11g 이전 버전에서 이러한 작업을 수행하면 다음과 같은 오류가 발생합니다.

```
SELECT f(p_parameter_5 => 10) FROM DUAL;
```

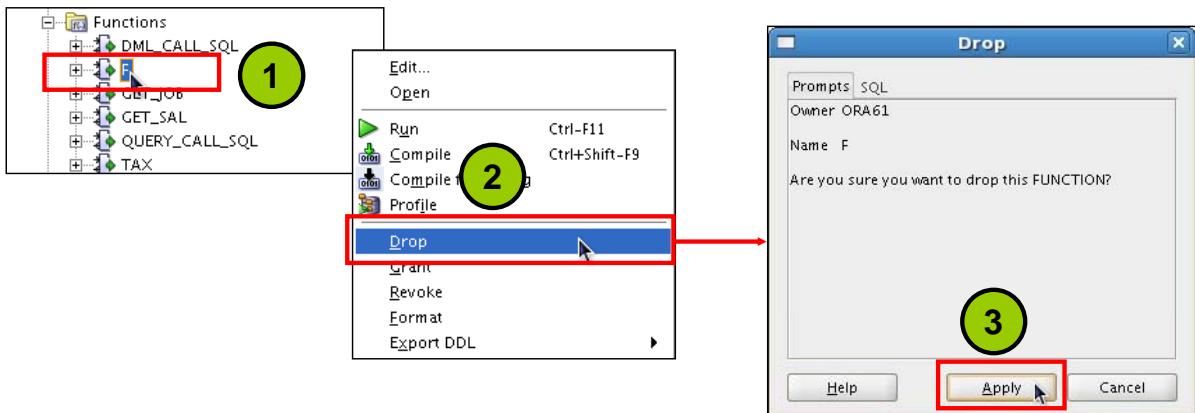
ORA-00907: missing right parenthesis

함수 제거: DROP SQL 문 또는 SQL Developer 사용

- **DROP 문 사용:**

```
DROP FUNCTION f;
```

- **SQL Developer 사용:**



Copyright © 2009, Oracle. All rights reserved.

함수 제거

DROP 문 사용

더 이상 필요하지 않은 내장 함수는 SQL*Plus에서 SQL 문을 사용하여 삭제할 수 있습니다. SQL*Plus를 사용하여 내장 함수를 제거하려면 **DROP FUNCTION SQL** 명령을 실행합니다.

CREATE OR REPLACE를 사용하는 경우와 **DROP** 및 **CREATE**를 사용하는 경우의 차이

CREATE OR REPLACE 구문의 **REPLACE** 절은 함수를 삭제한 다음 재생성하는 것과 같습니다. **CREATE OR REPLACE** 구문을 사용할 경우 다른 유저에게 부여된 이 객체에 대한 권한은 동일하게 유지됩니다. 그러나 함수를 삭제한 다음 재생성할 경우 이 함수에 부여된 모든 권한은 자동으로 취소됩니다.

SQL Developer 사용

SQL Developer에서 함수를 삭제하려면 **Functions** 노드에서 함수 이름을 마우스 오른쪽 버튼으로 누르고 **Drop**을 선택합니다. **Drop** 대화상자가 표시됩니다. 함수를 삭제하려면 **Apply**를 누릅니다.

데이터 딕셔너리 뷰를 사용하여 함수 보기

DESCRIBE USER_SOURCE

Name	Null	Type
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)

4 rows selected

```
SELECT text
FROM user_source
WHERE type = 'FUNCTION'
ORDER BY line;
```

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA
Results:
1 FUNCTION tax(p_value IN NUMBER)
2 FUNCTION query_call_sql(p_a NUMBER) RETURN NUMBER IS
3 FUNCTION get_sal
4 FUNCTION dml_call_sql(p_sal NUMBER)
5 RETURN NUMBER IS
6 RETURN NUMBER IS
7 (p_id employees.employee_id%TYPE) RETURN NUMBER IS
8 v_s NUMBER;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리 뷰를 사용하여 함수 보기

PL/SQL 함수의 소스 코드는 데이터 딕셔너리 테이블에 저장되어 있습니다. 소스 코드로 컴파일에 성공한 PL/SQL 함수나 컴파일에 실패한 PL/SQL 함수에 액세스할 수 있습니다. 데이터 딕셔너리에 저장된 PL/SQL 함수 코드를 보려면 TYPE 열 값이 FUNCTION인 다음 테이블에 대해 SELECT 문을 실행하십시오.

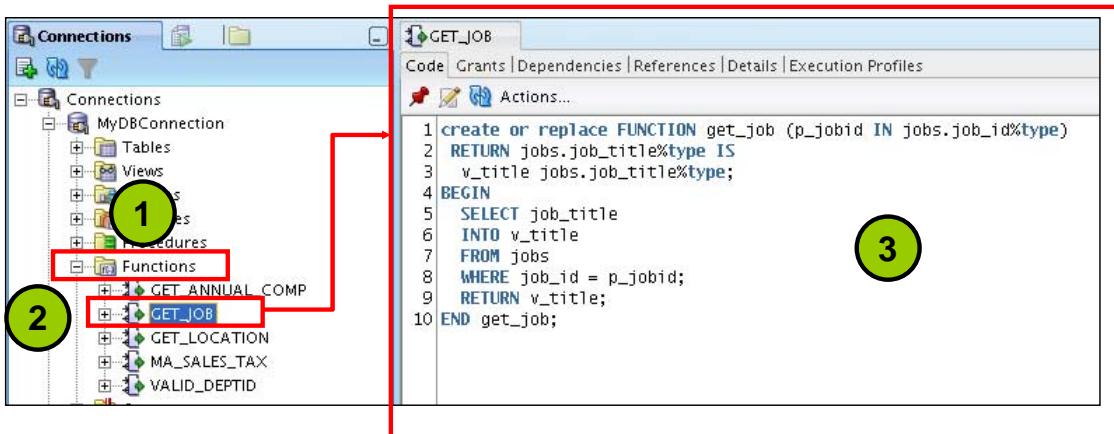
- 자신이 소유한 PL/SQL 코드를 표시하려면 USER_SOURCE 테이블 조회
- 서브 프로그램 코드 소유자로부터 EXECUTE 권한을 부여 받은 PL/SQL 코드를 표시하려면 ALL_SOURCE 테이블 조회

슬라이드의 두번째 예제에서는 USER_SOURCE 테이블을 사용하여 스키마에 있는 모든 함수의 소스 코드를 표시합니다.

USER_OBJECTS 데이터 딕셔너리 뷰를 사용하여 함수 이름 리스트를 표시할 수도 있습니다.

참고: 슬라이드의 두번째 코드 예제 출력은 보다 나은 형식의 출력을 제공하기 위해 도구 모음의 Execute Statement (F9) 아이콘을 사용하여 생성했습니다.

SQL Developer를 사용하여 함수 정보 보기



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 함수 정보 보기

SQL Developer에서 함수의 코드를 보려면 다음 단계를 따르십시오.

1. **Connections** 탭에서 **Functions** 노드를 누릅니다.
2. 함수 이름을 누릅니다.
3. 슬라이드에 나와 있는 것처럼 함수 코드가 **Code** 탭에 표시됩니다.

퀴즈

PL/SQL 내장 함수:

1. 표현식의 일부로 호출 가능
2. 헤더에 RETURN 절을 포함해야 함
3. 단일 값을 반환해야 함
4. 적어도 하나의 RETURN 문을 포함해야 함
5. 헤더에 RETURN 절이 없음

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 4

연습 2-1: 개요

이 연습에서는 다음 내용을 다룹니다.

- 내장 함수 생성:
 - 데이터베이스 테이블을 query하여 특정 값 반환
 - SQL 문에서 사용
 - 특정 파라미터 값을 사용하여 데이터베이스 테이블에 새 행 삽입
 - 기본 파라미터 값 사용
- SQL 문에서 내장 함수 호출
- 내장 프로시저에서 내장 함수 호출

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 2-1: 개요

이 연습에서는 SQL Developer를 사용할 것을 권장합니다.

단원 내용

- **함수 작업:**
 - 프로시저와 함수의 차이 설명
 - 함수 사용법 설명
 - 내장 함수 생성, 호출 및 제거
- **SQL Developer Debugger 소개**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer Debugger를 사용하여 PL/SQL 서브 프로그램 디버그

- 디버거를 사용하여 PL/SQL 프로그램의 실행을 제어할 수 있습니다.
- PL/SQL 서브 프로그램을 디버그하려면 보안 관리자가 다음 권한을 응용 프로그램 개발자에게 부여해야 합니다.
 - DEBUG ANY PROCEDURE
 - DEBUG CONNECT SESSION

```
GRANT DEBUG ANY PROCEDURE TO ora61;
GRANT DEBUG CONNECT SESSION TO ora61;
```

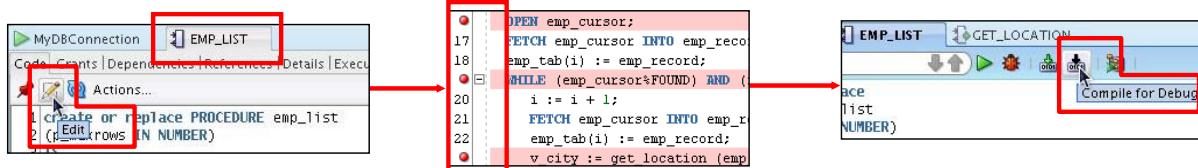
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

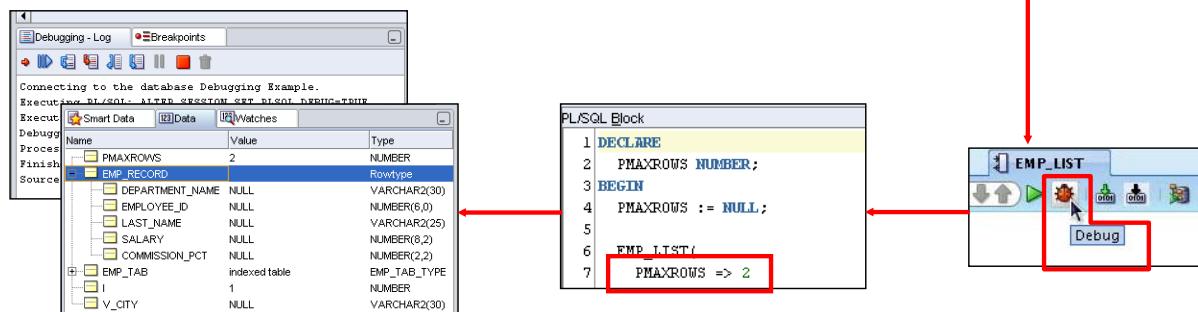
디버그 도중 코드 내 이동

SQL Developer Debugger를 사용하면 프로그램의 실행을 제어할 수 있습니다. 즉, 프로그램이 단일 코드 행을 실행하는지, 전체 서브 프로그램(프로시저 또는 함수)을 실행하는지 아니면 전체 프로그램 블록을 실행하는지를 제어할 수 있습니다. 프로그램을 실행할 시기와 일시 중지할 시기를 수동으로 제어하면 올바르게 작동하는 셋션은 빠르게 건너뛰고 문제가 발생하는 셋션에 대해 집중적으로 작업을 수행할 수 있습니다.

서브 프로그램 디버그: 개요



1. 프로시저 편집 2. 종단점 추가 3. Compile for Debug

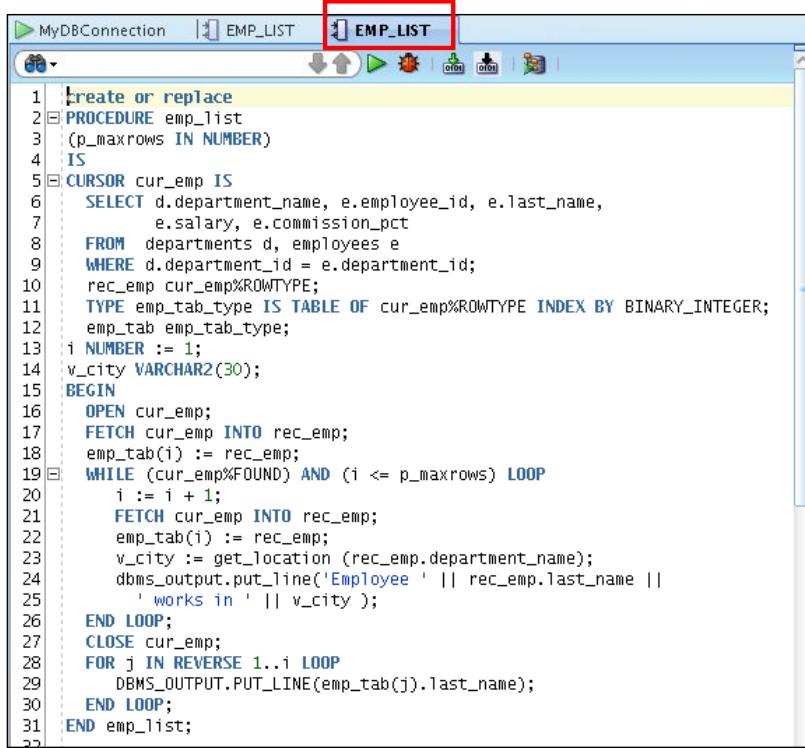


4. 디버그 도구를 선택하고
데이터 모니터 5. 파라미터 값 입력 6. 디버그

Copyright © 2009, Oracle. All rights reserved.

ORACLE

프로시저 또는 함수 코드 편집 탭



```

1  CREATE OR REPLACE
2  PROCEDURE emp_list
3  (p_maxrows IN NUMBER)
4  IS
5  CURSOR cur_emp IS
6    SELECT d.department_name, e.employee_id, e.last_name,
7           e.salary, e.commission_pct
8    FROM departments d, employees e
9   WHERE d.department_id = e.department_id;
10  RECURSIVE_TYPE cur_emp%ROWTYPE;
11  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
12  emp_tab emp_tab_type;
13  i NUMBER := 1;
14  v_city VARCHAR2(30);
15  BEGIN
16    OPEN cur_emp;
17    FETCH cur_emp INTO rec_emp;
18    emp_tab(i) := rec_emp;
19    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
20      i := i + 1;
21      FETCH cur_emp INTO rec_emp;
22      emp_tab(i) := rec_emp;
23      v_city := get_location (rec_emp.department_name);
24      dbms_output.put_line('Employee ' || rec_emp.last_name ||
25                            ' works in ' || v_city );
26    END LOOP;
27    CLOSE cur_emp;
28    FOR j IN REVERSE 1..i LOOP
29      DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
30    END LOOP;
31  END emp_list;
32

```

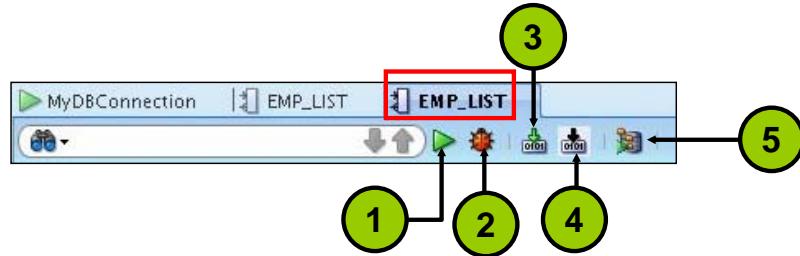
ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 또는 함수 코드 탭

이 탭에는 도구 모음 및 편집 가능한 서브 프로그램 텍스트가 표시됩니다. 중단점을 연결할 각 명령문 옆에 있는 가는 세로 줄 왼쪽을 누르면 디버그용으로 중단점을 설정하고 설정을 해제할 수 있습니다. 중단점이 설정되면 빨강색 원이 표시됩니다.

프로시저 또는 함수 탭 도구 모음



아이콘	설명
1. Run	함수 또는 프로시저 실행을 정상적으로 시작하고 결과를 Running – Log 탭에 표시합니다.
2. Debug	디버그 모드에서 서브 프로그램을 실행하고 실행 제어를 위한 디버깅 도구 모음이 포함된 Debugging – Log 탭을 표시합니다.
3. Compile	서브 프로그램을 컴파일합니다.
4. Compile for Debug	디버깅할 수 있도록 서브 프로그램을 컴파일합니다.
5. Profile	PL/SQL 함수 또는 프로시저 실행, 디버깅 또는 프로파일 생성을 위한 파라미터 값을 지정하는 데 사용하는 Profile window를 표시합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Debugging – Log 탭 도구 모음



아이콘	설명
1. Find Execution Point	다음 실행 지점으로 이동합니다.
2. Step Over	다음 서브 프로그램을 통과하고 서브 프로그램 뒤의 다음 명령문으로 이동합니다.
3. Step Into	한 번에 하나의 프로그램 명령문을 실행합니다. 실행 지점이 서브 프로그램 호출에 있는 경우에는 해당 서브 프로그램에서 첫번째 명령문을 Step Into합니다.
4. Step Out	현재 서브 프로그램을 그대로 두고 중단점이 포함된 다음 명령문으로 이동합니다.

ORACLE®

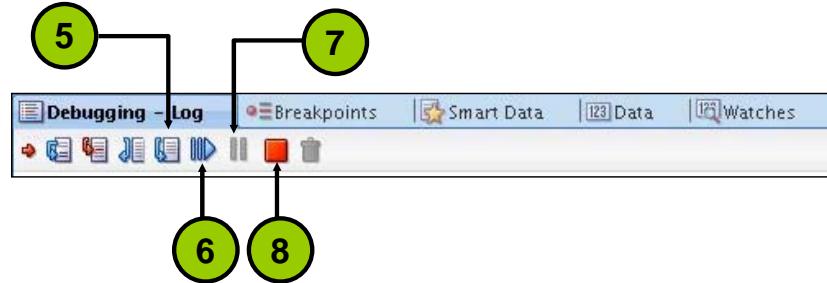
Copyright © 2009, Oracle. All rights reserved.

Debugging – Log 탭 도구 모음

Debugging - Log에는 디버깅 도구 모음과 정보 메시지가 포함되어 있습니다.

1. **Find Execution Point:** 실행 지점(디버거에서 실행할 소스 코드의 다음 행)으로 이동합니다.
2. **Step Over:** 서브 프로그램에 중단점이 없는 경우 다음 서브 프로그램을 통과하고 서브 프로그램 뒤의 다음 명령문으로 이동합니다. 실행 지점이 서브 프로그램 호출에 있는 경우 서브 프로그램을 Step Into하는 대신 정지되지 않고 해당 프로그램을 실행한 후 실행 지점을 호출 뒤에 이어지는 명령문에 놓습니다. 실행 지점이 서브 프로그램의 마지막 명령문에 있는 경우 Step Over가 서브 프로그램에서 반환되며, 실행 지점은 반환될 서브 프로그램 호출 뒤에 이어지는 코드 행에 배치됩니다.
3. **Step Into:** 한 번에 하나의 프로그램 명령문을 실행합니다. 실행 지점이 서브 프로그램 호출에 있는 경우 Step Into는 해당 서브 프로그램을 Step Into하여 실행 지점을 첫번째 명령문에 놓습니다. 실행 지점이 서브 프로그램의 마지막 명령문에 있는 경우 Step Into가 서브 프로그램에서 반환되며, 실행 지점은 반환될 서브 프로그램 호출 뒤에 이어지는 코드 행에 배치됩니다.
4. **Step Out:** 현재 서브 프로그램에서 나와 다음 명령문으로 이동합니다.

Debugging – Log 탭 도구 모음



아이콘	설명
5. Step to End of Method	현재 서브 프로그램의 마지막 명령문으로 이동합니다.
6. Resume	실행을 계속합니다.
7. Pause	실행을 중지하지만 종료하지는 않습니다.
8. Terminate	실행을 중지 및 종료합니다.

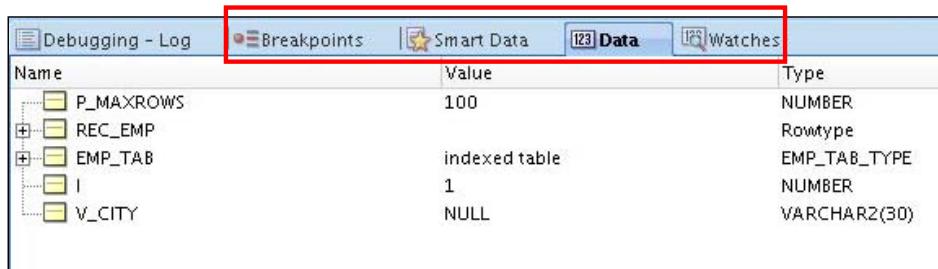
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Debugging – Log 탭 도구 모음(계속)

- 5. **Step to End of Method:** 현재 서브 프로그램의 마지막 명령문으로 이동합니다.
- 6. **Resume:** 실행이 계속됩니다.
- 7. **Pause:** 실행이 중지되지만 종료되지는 않으므로 나중에 실행을 재개할 수 있습니다.
- 8. **Terminate:** 실행이 중지 및 종료됩니다. 이 지점에서는 실행을 재개할 수 없습니다. 대신 서브 프로그램의 시작 부분부터 실행 또는 디버깅을 시작하려면 Source 탭 도구 모음의 Run 또는 Debug 아이콘을 누르십시오.

추가 탭



탭	설명
Breakpoints	시스템 정의 중단점과 유저 정의 중단점을 모두 표시합니다.
Smart Data	변수에 대한 정보를 표시합니다. Smart Data window를 마우스 오른쪽 버튼으로 누르고 Preferences를 선택하여 이러한 환경 설정을 지정할 수 있습니다.
Data	코드 텍스트 영역 아래에 있으며 모든 변수에 대한 정보를 표시합니다.
Watches	코드 텍스트 영역 아래에 있으며 Watches에 대한 정보를 표시합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

추가 탭

표현식 Watch 설정

Watch를 사용하면 프로그램이 실행됨에 따라 변경되는 변수 또는 표현식의 값을 모니터할 수 있습니다. Watch 표현식을 입력하면 Watches window에 표현식의 현재 값이 표시됩니다. 프로그램이 실행됨에 따라 프로그램이 Watch 표현식의 변수 값을 갱신하면서 Watch 값이 변경됩니다.

Watch는 Stack window의 선택 사항으로 제어되는 현재 컨텍스트에 따라 표현식을 평가합니다. 새 컨텍스트로 이동하면 표현식이 다음 컨텍스트에 대해 재평가됩니다. 실행 지점이 watch 표현식의 변수가 정의되지 않은 위치로 이동하면 전체 watch 표현식이 미정 상태가 됩니다. 실행 지점이 watch 표현식을 평가할 수 있는 위치로 돌아가면 Watches window에 watch 표현식의 값이 다시 표시됩니다.

Watches window를 열려면 View, Debugger, Watches를 차례로 누릅니다.

Watch를 추가하려면 Watches window를 마우스 오른쪽 버튼으로 누르고 Add Watch를 선택합니다. Watch를 편집하려면 Watches window를 마우스 오른쪽 버튼으로 누르고 Edit Watch를 선택합니다.

참고: 이 단원에서 설명하는 디버깅 탭 중 일부가 표시되지 않는 경우에는 View > Debugger 메뉴 옵션을 사용하여 해당 탭을 다시 표시할 수 있습니다.

프로시저 예제 디버그: 새 emp_list 프로시저 생성

```

1 CREATE OR REPLACE PROCEDURE emp_list(pmaxrows IN NUMBER) AS
2 CURSOR emp_cursor IS
3 SELECT d.department_name,
4 e.employee_id,
5 e.last_name,
6 e.salary,
7 e.commission_pct
8 FROM departments d,
9 employees e
10 WHERE d.department_id = e.department_id;
11 emp_record emp_cursor % rowtype;
12 type emp_tab_type is TABLE OF emp_cursor % rowtype INDEX BY binary_integer;
13 emp_tab emp_tab_type;
14 i NUMBER := 1;
15 v_city VARCHAR2(30);
16 BEGIN
17
18 OPEN emp_cursor;
19 FETCH emp_cursor
20 INTO emp_record;
21 emp_tab(i) := emp_record;
22 WHILE (emp_cursor % FOUND)
23 AND (i <= pmaxrows)
24 LOOP
25 i := i + 1;
26 FETCH emp_cursor
27 INTO emp_record;
28 emp_tab(i) := emp_record;
29 v_city := get_location(emp_record.department_name);
30 DEMS_OUTPUT.PUT_LINE('Employee ' || emp_record.last_name || ' works in ' || v_city);
31 END LOOP;
32
33 CLOSE emp_cursor;
34 FOR j IN REVERSE 1 .. i
35 LOOP
36 DEMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
37 END LOOP;
38 END emp_list;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 예제 디버그: 새 emp_list 프로시저 생성

emp_list 프로시저는 사원의 부서 이름, ID, 이름, 급여 및 커미션 비율과 같은 사원 정보를 수집하며 이러한 사원 정보를 저장하기 위한 레코드를 생성합니다. 또한 이 프로시저는 다중 사원 레코드를 보관할 수 있는 테이블도 생성합니다. "i"는 카운터입니다.

이 코드는 커서를 열고 사원 레코드를 패치(fetch)하며, 더 패치(fetch)할 레코드가 있는지 여부와 지금까지 패치(fetch)한 레코드 수가 지정한 레코드 수보다 작은지 여부도 확인합니다. 그리고 마지막으로 사원 정보를 인쇄합니다. 이 프로시저는 사원이 근무하는 도시 이름을 반환하는 get_location 함수도 호출합니다.

참고: 프로시저 코드는 편집 모드에서 표시해야 합니다. 프로시저 코드를 편집하려면 프로시저 도구 모음에서 Edit 아이콘을 누르십시오.

프로시저 예제 디버그: 새 get_location 함수 생성

```
1 | CREATE OR REPLACE FUNCTION get_location(p_deptname IN VARCHAR2) RETURN VARCHAR2 AS
2 |   v_loc_id NUMBER;
3 |   v_city VARCHAR2(30);
4 | BEGIN
5 |   SELECT d.location_id,
6 |         l.city
7 |     INTO v_loc_id,
8 |           v_city
9 |    FROM departments d,
10 |          locations l
11 |   WHERE UPPER(department_name) = UPPER(p_deptname)
12 |     AND d.location_id = l.location_id;
13 |   RETURN v_city;
14 | END get_location;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

프로시저 예제 디버그: 새 프로시저 생성

이 함수는 사원이 근무하는 도시를 반환하며 emp_list 프로시저에서 호출됩니다.

중단점 설정 및 디버그 모드용으로 emp_list 컴파일

```

1  create or replace
2  PROCEDURE emp_list
3  (p_maxrows IN NUMBER)
4  IS
5  CURSOR cur_emp IS
6      SELECT d.department_name, e.employee_id, e.last_name,
7          e.salary, e.commission_pct
8      FROM departments d, employees e
9      WHERE d.department_id = e.department_id;
10     rec_emp cur_emp%ROWTYPE;
11     TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
12     emp_tab emp_tab_type;
13     i NUMBER := 1;
14     v_city VARCHAR2(30);
15 BEGIN
16     OPEN cur_emp;
17     FETCH cur_emp INTO rec_emp;
18     emp_tab(i) := rec_emp;
19     WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
20         i := i + 1;
21         FETCH cur_emp INTO rec_emp;
22         emp_tab(i) := rec_emp;
23         v_city := get_location(rec_emp.department_name);
24         dbms_output.put_line('Employee ' || rec_emp.last_name ||
25                             ' works in ' || v_city );
26     END LOOP;
27     CLOSE cur_emp;
28     FOR i IN REVERSE 1 .. LOOP
29         DBMS_OUTPUT.PUT_LINE(emp_tab(i).last_name);
30     END LOOP;
31 END emp_list;
32

```

Messages - Log
EMP_LIST Compiled

ORACLE

Copyright © 2009, Oracle. All rights reserved.

중단점 설정 및 디버그 모드용으로 emp_list 컴파일

슬라이드의 예제에서는 emp_list 프로시저가 편집 모드에서 표시되며 네 개의 중단점이 코드의 여러 위치에 추가되었습니다. 디버그용으로 프로시저를 컴파일하려면 코드를 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile for Debug를 선택합니다. 그러면 Messages – Log 탭에 프로시저가 컴파일되었다는 메시지가 표시됩니다.

디버그 모드용으로 get_location 함수 컴파일

```

1  create or replace
2  FUNCTION get_location
3  ( p_deptname IN VARCHAR2) RETURN VARCHAR2
4  AS
5      v_loc_id NUMBER;
6      v_city    VARCHAR2(30);
7  BEGIN
8      SELECT d.location_id, l.city INTO v_loc_id, v_city
9      FROM departments d, locations l
10     WHERE upper(department_name) = upper(p_deptname)
11       and d.location_id = l.location_id;
12     RETURN v_city;
13 END GET_LOCATION;
14

```

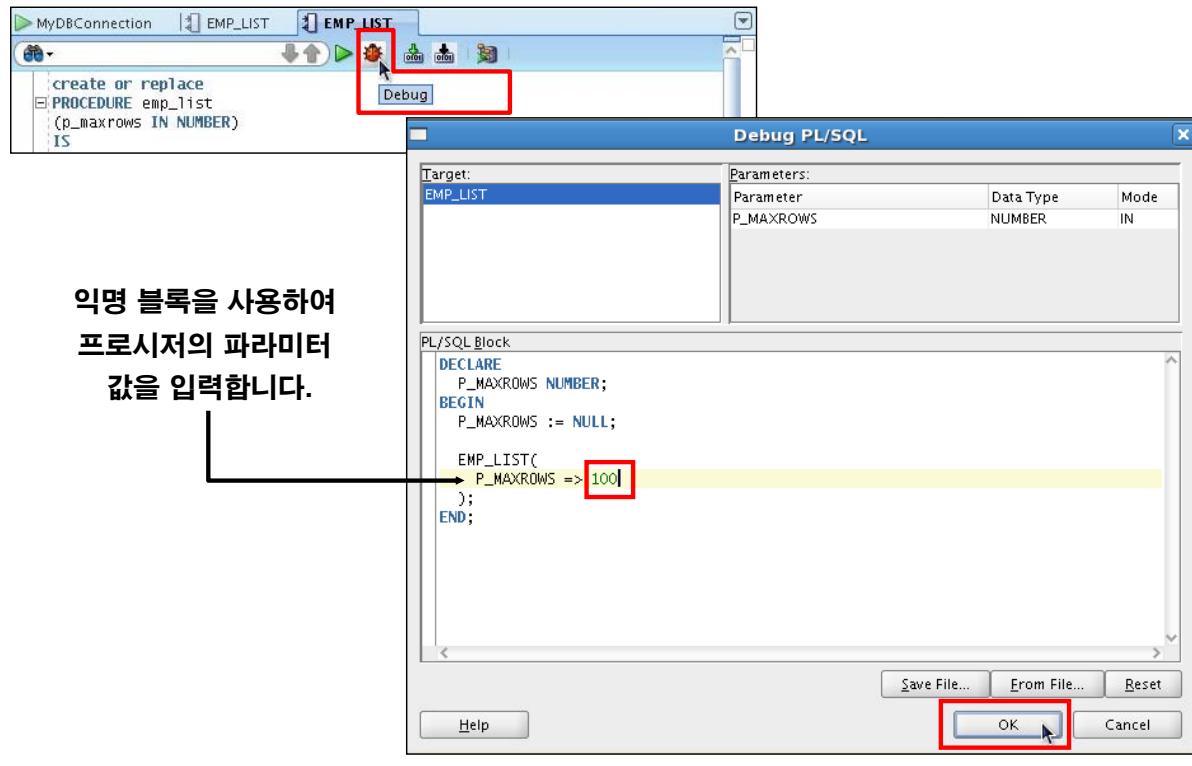
ORACLE

Copyright © 2009, Oracle. All rights reserved.

디버그 모드용으로 get_location 함수 컴파일

슬라이드의 예제에서는 get_location 함수가 편집 모드에서 표시됩니다. 디버그용으로 함수를 컴파일하려면 코드를 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile for Debug를 선택합니다. 그러면 Messages – Log 탭에 함수가 컴파일되었다는 메시지가 표시됩니다.

emp_list 디버그 및 PMAXROWS 파라미터 값 입력



Copyright © 2009, Oracle. All rights reserved.

ORACLE

emp_list 디버그 및 PMAXROWS 파라미터 값 입력

디버깅 프로세스의 다음 단계는 프로시저 도구 모음에서 Debug 아이콘을 누르는 등 앞서 설명했던 다양한 사용 가능 메소드 중 하나를 사용하여 프로시저를 디버그하는 것입니다. 익명 블록이 표시되고 해당 프로시저에 대한 파라미터를 입력하라는 메시지가 표시됩니다. emp_list에는 반환할 레코드 수를 지정하는 PMAXROWS 파라미터가 하나 있습니다. 두 번째 PMAXROWS를 숫자(예: 100)로 바꾸고 OK를 누릅니다.

emp_list 디버그: 코드 Step Into (F7)

프로그램 제어는
첫번째 중단점에서
정지됩니다.

```

1 CREATE OR REPLACE PROCEDURE emp_list
2   (p_maxrows IN NUMBER)
3   IS
4   CURSOR cur_emp IS
5     SELECT d.department_name, e.employee_id, e.last_name,
6           e.salary, e.commission_pct
7     FROM departments d, employees e
8    WHERE d.department_id = e.department_id;
9   rec_emp cur_emp%ROWTYPE;
10  TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
11  emp_tab emp_tab_type;
12  i NUMBER := 1;
13  v_city VARCHAR2(30);
14  BEGIN
15    OPEN cur_emp;
16    FETCH cur_emp INTO rec_emp;
17    emp_tab(i) := rec_emp;
18    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19      i := i + 1;
20      FETCH cur_emp INTO rec_emp;
21      emp_tab(i) := rec_emp;
22      v_city := get_location (rec_emp.department_name);
23      dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );
25    END LOOP;
26    CLOSE cur_emp;
27    FOR j IN REVERSE 1..i LOOP
28      DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
29    END LOOP;
30  END emp_list;
31

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

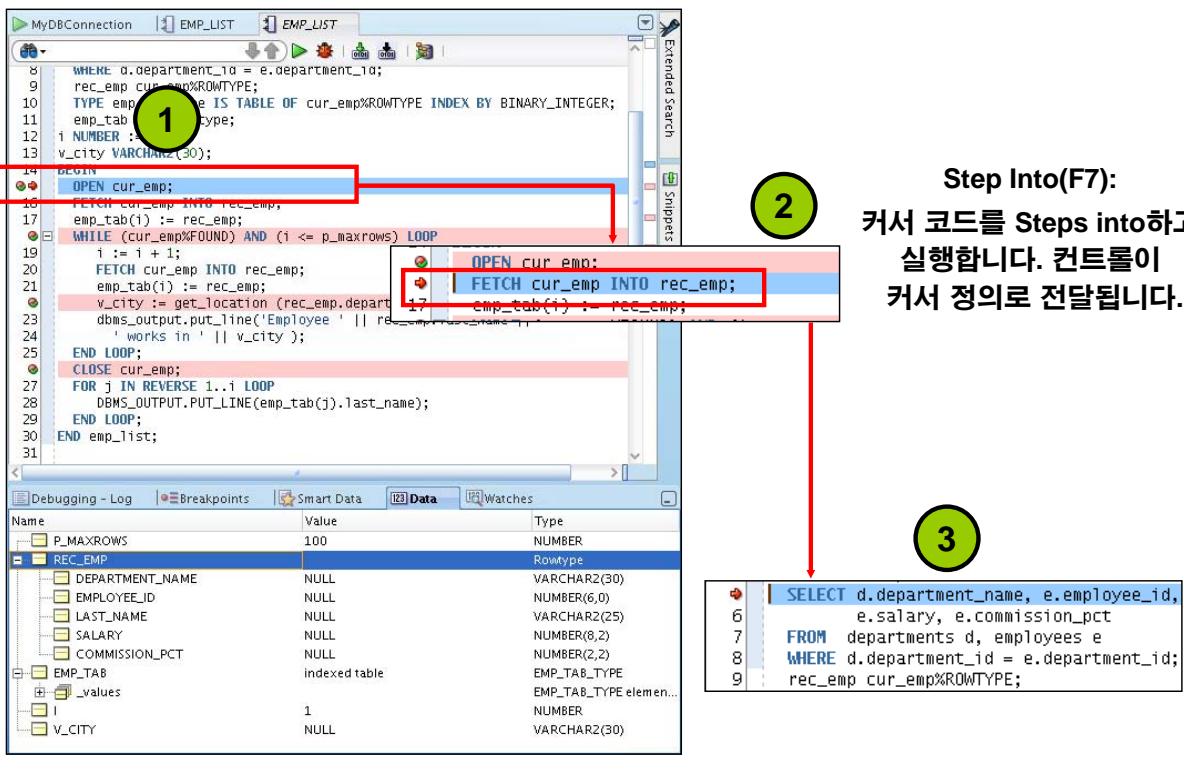
Step Into 디버깅 도구

Step Into 명령은 한 번에 하나의 프로그램 명령문을 실행합니다. 실행 지점이 서브 프로그램 호출에 있는 경우 Step Into 명령은 해당 서브 프로그램을 Step Into하여 실행 지점을 서브 프로그램의 첫번째 명령문에 놓습니다. 실행 지점이 서브 프로그램의 마지막 명령문에 있는 경우 Step Into를 선택하면 디버거가 서브 프로그램에서 반환되며, 실행 지점은 반환될 서브 프로그램 호출 뒤에 이어지는 코드 행에 배치됩니다. *Single Stepping*은 Step Into를 사용하여 프로그램 코드의 명령문을 순차적으로 실행하는 것을 나타냅니다. Debug > Step Into를 선택하거나, F7 키를 누르거나, Debugging - Log 도구 모음에서 Step Into 아이콘을 눌러 서브 프로그램을 Step Into할 수 있습니다.

슬라이드의 예제에서는 프로그램 제어가 코드의 첫번째 중단점에서 정지됩니다. 중단점 옆에 있는 화살표는 해당 코드 행이 다음에 실행됨을 나타냅니다. window 하단에 있는 여러 탭을 확인하십시오.

참고: Step Into 및 Step Over 명령을 사용하면 프로그램 코드 내에서 쉽게 이동할 수 있습니다. 이 두 명령은 매우 비슷하지만 서로 다른 방식으로 코드 실행을 제어합니다.

emp_list 디버그: 코드 Step Into(F7)



emp_list 디버그: 코드 Step Into

Step Into 명령을 선택하면 한 번에 하나의 프로그램 명령문이 실행됩니다. 실행 지점이 서브 프로그램 호출에 있는 경우 Step Into 명령은 해당 서브 프로그램을 Step Into하여 실행을 서브 프로그램의 첫번째 명령문에 놓습니다.

슬라이드 예제에서는 F7 키를 누르면 첫번째 중단점에서 코드 행이 실행됩니다. 이 경우 프로그램 제어는 커서가 정의되는 섹션으로 전달됩니다.

데이터 보기

The screenshot shows the Oracle Database 11g PL/SQL Developer interface. At the top, there are two code snippets:

```
18 OPEN emp_cursor;
20 FETCH emp_cursor
```

```
16 OPEN cur_emp;
17   FETCH cur_emp INTO rec_emp;
18   emp_tab(1) := rec_emp;
19   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
```

Below the code, a red arrow points from the first snippet to a 'Data' tab in the Debugger window. The 'Data' tab displays variable values:

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	NULL	VARCHAR2(30)
EMPLOYEE_ID	NULL	NUMBER(6,0)
LAST_NAME	NULL	VARCHAR2(25)
SALARY	NULL	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE elem...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

A second red arrow points from the second snippet to another 'Data' tab in the Debugger window. This tab shows the state of the variables after the first iteration of the loop:

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP	Rowtype	
DEPARTMENT_NAME	'Executive'	VARCHAR2(30)
EMPLOYEE_ID	100	NUMBER(6,0)
LAST_NAME	'King'	VARCHAR2(25)
SALARY	26400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	Indexed table	EMP_TAB_TYPE
_values		EMP_TAB_TYPE elem...
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

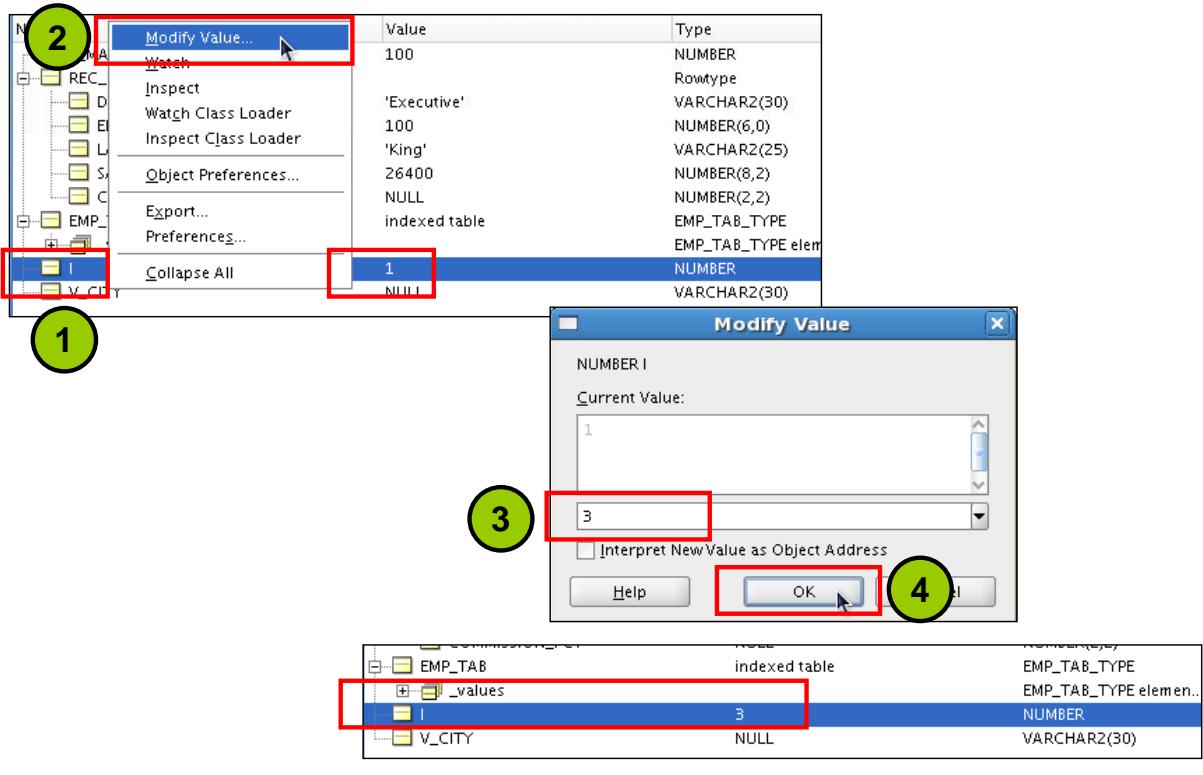
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 보기

코드를 디버그하는 동안 Data 탭을 사용하여 변수를 표시 및 수정할 수 있습니다. 또한 감시점(Watch)을 설정하여 Data 탭에 표시되는 변수의 하위 집합을 모니터할 수 있습니다. Data, Smart Data 및 Watch 탭을 표시하거나 숨기려면 View > Debugger를 선택한 다음 해당 탭을 선택합니다.

코드 디버그 중 변수 수정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

코드 디버그 도중 변수 수정

Data 탭에서 변수 값을 수정하려면 변수 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 `Modify Value`를 선택합니다. `Modify Value` window가 표시됩니다. 변수의 현재 값이 표시됩니다. 두 번째 텍스트 상자에 새 값을 입력하고 `OK`를 누르면 됩니다.

emp_list 디버그: 코드 Step Over

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

Step Over(F8):
커서를 실행합니다.
F7 키를 누르는 것과 같지만
컨트롤이 Open Cursor 코드로
전달되지 않습니다.

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

```

14 BEGIN
15   OPEN cur_emp;
16   FETCH cur_emp INTO rec_emp;
17   emp_tab(i) := rec_emp;
18   WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
19     i := i + 1;
20     FETCH cur_emp INTO rec_emp;
21     emp_tab(i) := rec_emp;
22     v_city := get_location (rec_emp.department_name);
23     dbms_output.put_line('Employee ' || rec_emp.last_name ||
24                           ' works in ' || v_city );

```

ORACLE

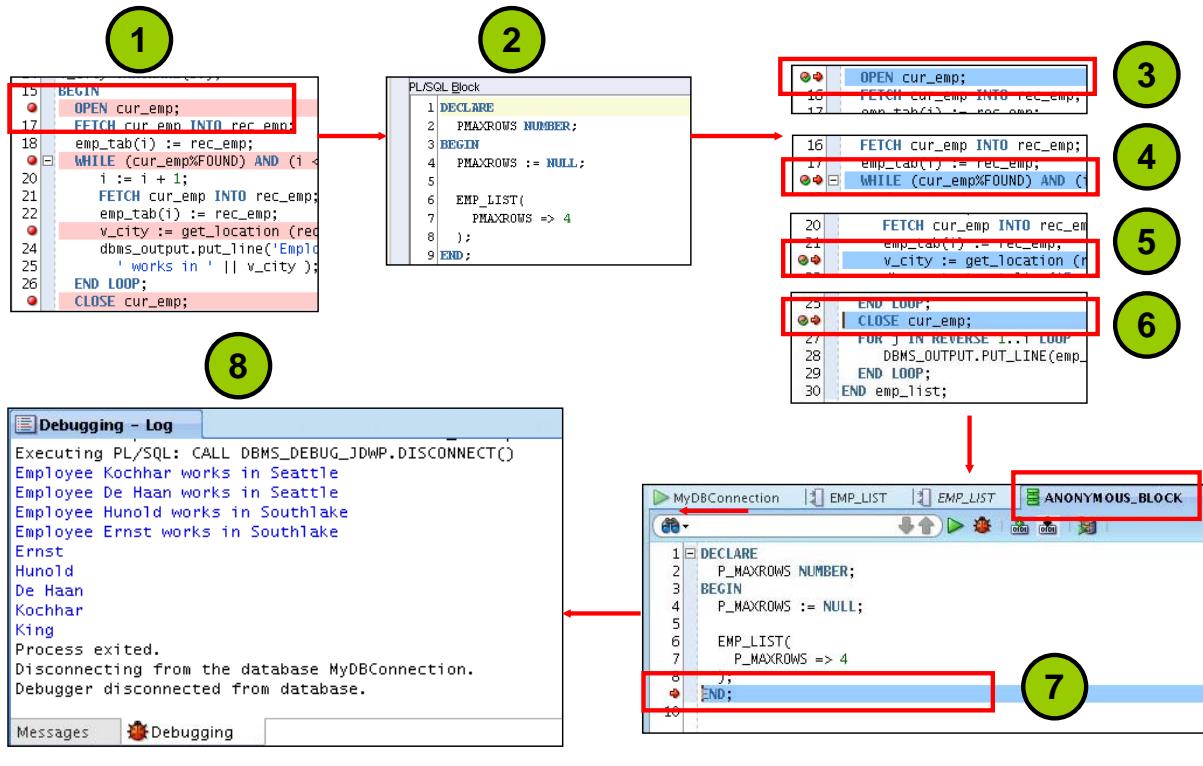
Copyright © 2009, Oracle. All rights reserved.

Step Over 디버깅 도구

Step Over 명령을 사용하면 Step Into와 마찬가지로 프로그램 명령문을 한 번에 하나씩 실행할 수 있습니다. 그러나 실행 지점이 서브 프로그램 호출에 있을 때 Step Over 명령을 실행하면 디버거가 서브 프로그램을 Step Into하는 대신 정지되지 않고 해당 프로그램을 실행한 다음 실행 지점을 서브 프로그램 호출 뒤에 이어지는 명령문에 놓습니다. 실행 지점이 서브 프로그램의 마지막 명령문에 있는 경우 Step Over를 선택하면 디버거가 서브 프로그램에서 반환되며, 실행 지점은 반환될 서브 프로그램 호출 뒤에 이어지는 코드 행에 배치됩니다. Debug > Step Over를 선택하거나, F8 키를 누르거나, Debugging - Log 도구 모음에서 Step Over 아이콘을 눌러 서브 프로그램을 Step Over할 수 있습니다.

슬라이드의 예제에서 Step Over를 선택하면 프로그램 제어를 커서 정의로 전달한 Step Into 옵션 예제와 달리 그대로 open cursor 행이 실행됩니다.

emp_list 디버그: 코드 Step Out(Shift+F7)



ORACLE

Copyright © 2009, Oracle. All rights reserved.

emp_list 디버그: 코드 Step Out (Shift+F7)

코드의 Step Out 옵션을 선택하면 현재 서브 프로그램을 그대로 두고 다음 명령문 서브 프로그램으로 이동합니다. 슬라이드의 예제(단계 3부터 시작)에서 Shift+F7을 누르면 프로그램 제어가 프로시저에서 나와 익명 블록의 다음 명령문으로 이동합니다. Shift+F7을 계속 누르고 있으면 SQL 버퍼의 내용을 인쇄하는 다음 익명 블록으로 이동합니다.

emp_list 디버그: Run to Cursor(F4)

```

10  emp_record emp_cursor%ROWTYPE;
11  TYPE emp_tab_type IS TABLE OF emp_cursor%ROWTYPE INDEX BY BINARY_INTEGER;
12  emp_tab emp_tab_type;
13  i NUMBER := 1;
14  v_city VARCHAR2(30);
15  BEGIN
16      OPEN emp_cursor;
17      FETCH emp_cursor INTO emp_record;
18      emp_tab(i) := emp_record;
19      WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
20          i := i + 1;
21          FETCH emp_cursor INTO emp_record;
22          emp_tab(i) := emp_record;
23          v_city := get_location (emp_record.department_name);
24          dbms_output.put_line('Employee ' || emp_record.last_name ||
25                                ' works in ' || v_city );
26      END LOOP;
27      CLOSE emp_cursor;
28
29      FOR j IN REVERSE 1..1 LOOP
30          DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
31      END LOOP;
32  END emp_list;

```

Run to Cursor F4:
Single Step을 수행하거나
중단점을 설정하지 않고
커서 위치에서 실행합니다.

Name	Type
PMAXROWS	NUMBER
EMP_RECORD	Rowtype
DEPARTMENT_NAME	VARCHAR2...
EMPLOYEE_ID	NUMBER(6,0)
LAST_NAME	VARCHAR2...
SALARY	NUMBER(8,2)
COMMISSION_PCT	NUMBER(2,2)
EMP_TAB	indexed table
I	NUMBER
V_CITY	VARCHAR2...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

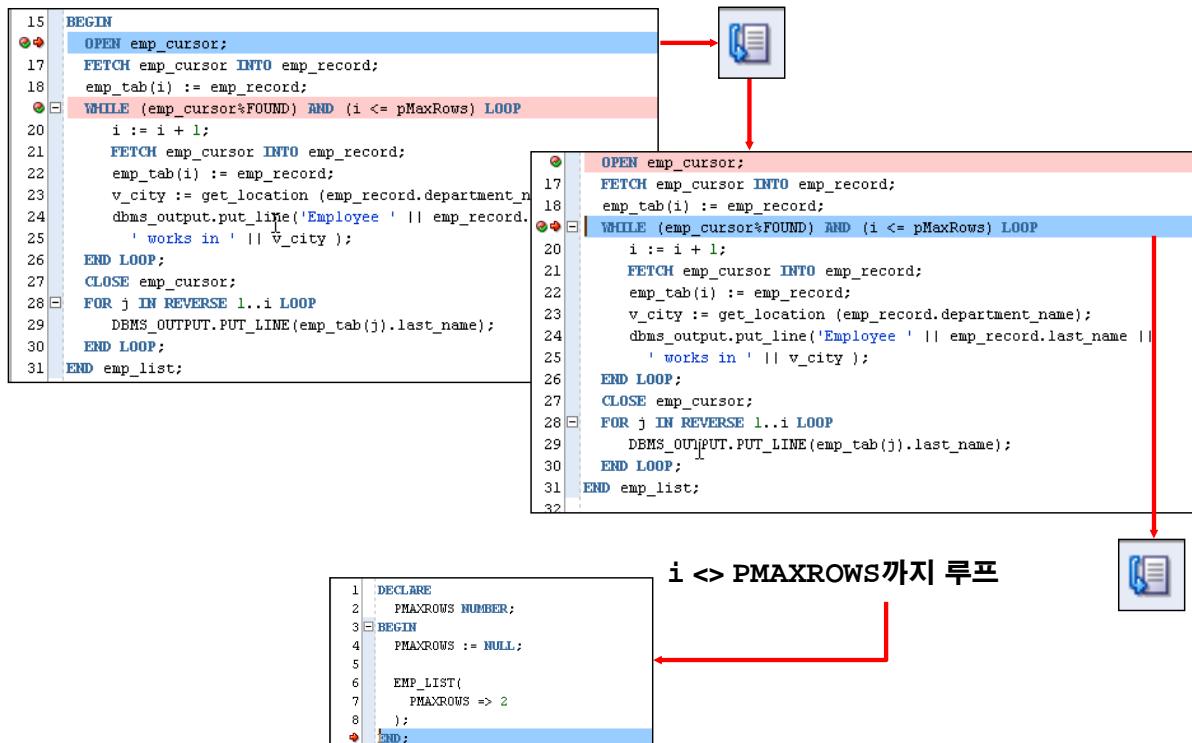
커서 위치로 실행

디버거에서 응용 프로그램 코드를 단계적으로 검사할 때 Single Step을 수행하거나 중단점을 설정하지 않고 특정 위치로 실행하려는 경우가 있습니다. 특정 프로그램 위치에서 실행하려면 서브 프로그램 편집기에서 텍스트 커서를 디버거가 정지되도록 할 코드 행에 놓습니다. 프로시저 편집기에서 마우스 오른쪽 버튼을 누르고 Run to Cursor를 선택하거나, 주 메뉴에서 Debug > Run to Cursor 옵션을 선택하거나, F4 키를 눌러 커서 위치에서 실행할 수 있습니다.

이때 다음과 같은 상황이 발생할 수 있습니다.

- 커서로 실행할 때 프로그램은 소스 편집기의 텍스트 커서가 표시하는 위치에 도달할 때까지 정지되지 않고 실행됩니다.
- 프로그램이 텍스트 커서가 있는 코드 행을 실제로 실행하지 않는 경우 Run to Cursor 명령을 선택하면 프로그램이 중단점에 도달하거나 완료될 때까지 계속 실행됩니다.

emp_list 디버그: Step to End of Method



ORACLE

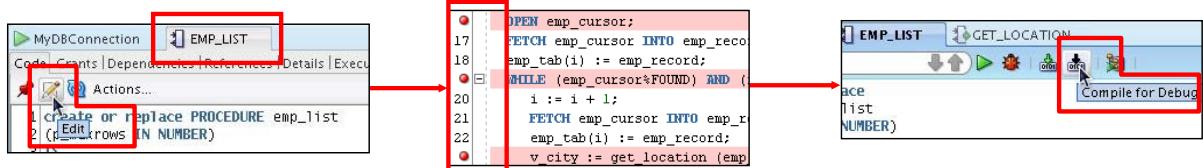
Copyright © 2009, Oracle. All rights reserved.

emp_list 디버그: Step to End of Method

Step to End of Method를 선택하면 현재 서브 프로그램의 마지막 명령문이나 현재 서브 프로그램에 중단점이 있는 경우 다음 중단점으로 이동합니다.

슬라이드의 예제에서는 Step to End of Method 디버깅 도구가 사용됩니다. 두번째 중단점이 있기 때문에 Step to End of Method를 선택하면 컨트롤이 해당 중단점으로 전달됩니다. Step to End of Method를 다시 선택하면 먼저 while 루프를 반복한 후 프로그램 제어를 익명 블록의 다음 실행문에 전달합니다.

서브 프로그램 원격 디버깅: 개요



1. 프로시저 편집

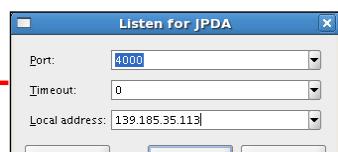
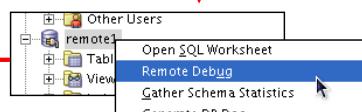
2. 중단점 추가

3. Compile for Debug

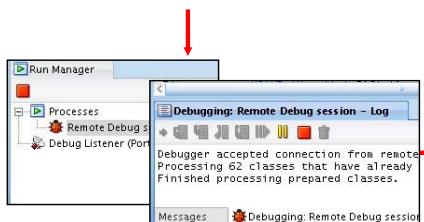
```
SQL> EXEC DBMS_DEBUG_JDWP.CONNECT_TCP('139.185.35.113', '4000');
PL/SQL procedure successfully completed.

SQL> EXEC emp_list(100);
```

6. 디버거 연결 명령을 실행하고
SQL*Plus 등의 다른 세션에서
프로시저 호출

5. 로컬 시스템의 IP 주소 및
디버깅 포트 입력

4. Remote Debug 선택

7. 중단점에 도달하면 컨트롤이
SQL Developer에 전달됨8. 디버깅 도구를 사용하여
디버그 및 모니터

ORACLE

Copyright © 2009, Oracle. All rights reserved.

원격 디버깅

데이터베이스에 대한 액세스 권한이 있고 데이터베이스에 연결된 경우 데이터베이스 위치에 관계없이 원격 디버깅을 사용해서 데이터베이스의 PL/SQL 코드에 연결하여 코드를 디버깅할 수 있습니다. 원격 디버깅을 사용하면 개발자로서의 유저가 아닌 다른 주체에 의해 시작된 프로시저를 유저가 디버그할 수 있습니다. 원격 디버깅과 로컬 디버깅에는 공통된 단계가 많습니다. 원격으로 디버깅하려면 다음 단계를 수행합니다.

1. 디버깅 할 서브 프로그램을 편집합니다.
2. 서브 프로그램에서 중단점을 추가합니다.
3. 도구 모음에서 **Compile for Debug** 아이콘을 누릅니다.
4. 원격 데이터베이스에 대한 연결을 마우스 오른쪽 버튼으로 누르고 **Remote Debug**를 선택합니다.
5. **Debugger - Attach to JPDA** 대화상자에서 로컬 시스템의 IP 주소와 포트 번호(예: 4000)를 입력한 다음 **OK**를 누릅니다.
6. SQL*Plus 등의 다른 세션을 사용하여 디버거 연결 명령을 실행한 다음 해당 세션에서 프로시저를 호출합니다.
7. 중단점에 도달하면 컨트롤이 원래의 SQL Developer 세션에 다시 전달되고 디버거 도구 모음이 표시됩니다.
8. 앞에서 설명한 디버깅 도구를 사용하여 서브 프로그램을 디버깅하고 데이터를 모니터합니다.

연습 2-2 개요: SQL Developer Debugger 소개

이 연습에서는 다음 내용을 다룹니다.

- 프로시저 및 함수 생성
- 프로시저에 중단점 삽입
- 디버그 모드용으로 프로시저 및 함수 컴파일
- 프로시저 디버깅 및 코드 Step Into
- 서브 프로그램의 변수 표시 및 수정

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 2-2: 개요

이 연습을 수행하려면 SQL Developer를 사용해야 합니다. 이 연습에서는 SQL Developer Debugger의 기본 기능에 대해 소개합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 프로시저와 함수의 차이 설명
- 함수 사용법 설명
- 내장 함수 생성
- 함수 호출
- 함수 제거
- SQL Developer Debugger의 기본 기능 이해

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

함수는 값을 반환해야 하는 명명된 PL/SQL 블록입니다. 일반적으로 값을 계산한 다음 반환하려면 함수를 생성하고, 작업을 수행하려면 프로시저를 생성하십시오.

함수는 생성하거나 삭제할 수 있습니다.

함수는 표현식의 일부로 호출됩니다.

3 패키지 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 패키지 설명 및 패키지 구성 요소 나열
- 관련 변수, 커서, 상수, 예외, 프로시저 및 함수를 함께 그룹화하는 패키지 생성
- 패키지 생성자를 공용(public) 또는 전용(private)으로 지정
- 패키지 생성자 호출
- 본문이 없는 패키지 사용법 설명

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 패키지에 대한 정의와 패키지 구성 요소에 대해 설명하며, 패키지를 생성하여 사용하는 방법에 대해서도 설명합니다.

단원 내용

- 패키지의 이점 및 구성 요소 식별
- 패키지 작업:
 - Package Spec 및 Body 생성
 - 패키지 서브 프로그램 호출
 - 패키지 제거
 - 패키지 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 패키지란?

- 패키지는 논리적으로 관련된 PL/SQL 유형, 변수 및 서브 프로그램을 그룹화하는 스키마 객체입니다.
- 패키지는 대개 다음과 같은 두 부분으로 구성됩니다.
 - Spec
 - Body
- Spec은 패키지의 인터페이스입니다. 패키지 외부에서 참조할 수 있는 유형, 변수, 상수, 예외, 커서 및 서브 프로그램을 선언합니다.
- Body는 커서에 대한 Query와 서브 프로그램에 대한 코드를 정의합니다.
- Oracle 서버가 동시에 다중 객체를 메모리 안으로 읽을 수 있도록 합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 패키지: 개요

PL/SQL 패키지를 사용하여 관련 PL/SQL 유형, 변수, 데이터 구조, 예외 및 서브 프로그램을 하나의 컨테이너로 번들화할 수 있습니다. 예를 들어 Human Resources 패키지는 고용 및 해고 프로시저, 커미션 및 보너스 함수, 면세 변수 등을 포함할 수 있습니다.

패키지는 일반적으로 데이터베이스에 개별적으로 저장되는 다음 두 부분으로 구성됩니다.

- Spec
- Body(선택 사항)

패키지 자체는 호출, 파라미터화, 중첩이 불가능합니다. 작성하고 컴파일한 후 내용을 여러 응용 프로그램과 공유할 수 있습니다.

PL/SQL 패키지 생성자가 처음으로 참조될 때 전체 패키지가 메모리에 로드됩니다. 이후에 동일한 패키지에서 생성자에 액세스할 때에는 디스크 I/O(입/출력)가 필요하지 않습니다.

패키지 사용 시 이점

- **모듈화:** 관련 생성자를 캡슐화합니다.
- **손쉬운 유지 관리:** 논리적으로 관련된 기능을 함께 보관합니다.
- **쉬운 응용 프로그램 설계:** Spec과 Body를 별도로 코딩 및 컴파일합니다.
- **정보 숨기기:**
 - Package Spec에 있는 선언만 응용 프로그램에서 볼 수 있고 액세스할 수 있습니다.
 - Package Body의 전용(private) 생성자는 표시되지 않으며 액세스할 수 없습니다.
 - 모든 코딩은 Package Body에서 표시되지 않습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

패키지 사용 시 이점

패키지는 프로시저와 함수 생성을 대신하여 독립형 스키마 객체로 사용할 수 있으며, 사용할 경우 여러 가지 이점이 있습니다.

모듈화 및 손쉬운 유지 관리: 논리적으로 관련된 프로그래밍 구조를 명명된 모듈에 캡슐화할 수 있습니다. 각 패키지는 쉽게 이해할 수 있으며, 패키지 간의 인터페이스는 간단하고 명확하며 잘 정의되어 있습니다.

쉬운 응용 프로그램 설계: 처음에는 Package Spec에 인터페이스 정보만 있으면 됩니다. Spec은 Body 없이 코딩 및 컴파일할 수 있습니다. 그런 다음 해당 패키지를 참조하는 내장 서브 프로그램도 컴파일할 수 있습니다. Package Body는 응용 프로그램을 완성한 후에 완전히 정의할 수 있습니다.

정보 숨기기: 공용(public)(보기 및 액세스 가능) 생성자와 전용(private)(보기 및 액세스 불가능) 생성자를 결정하십시오. Package Spec에 있는 선언은 응용 프로그램에서 볼 수 있고 액세스할 수 있습니다. Package Body에는 전용 생성자의 정의가 표시되지 않으므로, 정의가 변경될 경우 응용 프로그램이나 호출 프로그램은 영향을 받지 않고 패키지만 영향을 받습니다. 따라서 호출 프로그램을 재컴파일하지 않고도 구현을 변경할 수 있습니다. 또한 유저가 보지 못하도록 구현 세부 정보를 숨겨 패키지 무결성을 보호할 수 있습니다.

패키지 사용 시 이점

- **추가된 기능: 공용(public) 변수 및 커서 지속성**
- **응용 프로그램 성능 향상:**
 - 패키지가 처음으로 참조될 때 전체 패키지가 메모리에 로드됩니다.
 - 메모리에는 모든 유저에 대해 한 개의 복사본만 있습니다.
 - 종속 계층이 단순합니다.
- **오버로드: 동일한 이름의 다중 서브 프로그램**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

패키지 사용 시 이점(계속)

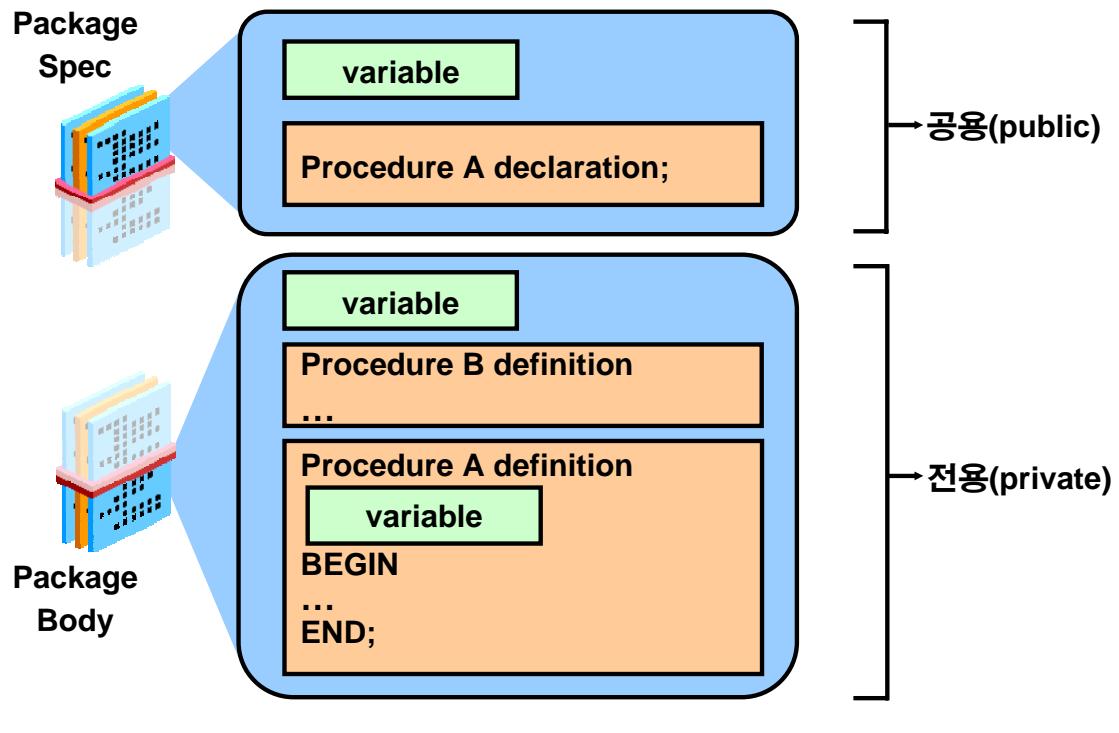
추가된 기능: 패키지화된 공용(public) 변수와 커서는 세션 기간 동안 지속됩니다. 따라서 해당 환경에서 실행되는 모든 서브 프로그램에서 이를 공유할 수 있습니다. 또한 그에 따라 데이터를 데이터베이스에 저장할 필요 없이 트랜잭션 전체에서 유지 관리할 수 있습니다. 전용(private) 생성자도 세션 기간 동안 지속되지만 패키지 내에서만 액세스할 수 있습니다.

성능 향상: 패키지화된 서브 프로그램을 처음으로 호출할 때 전체 패키지가 메모리에 로드됩니다. 따라서 이후에 패키지에서 관련 서브 프로그램을 호출할 때 더 이상 디스크 I/O가 필요하지 않습니다. 또한 패키지화된 서브 프로그램이 다른 서브 프로그램에 종속되지 않으므로 불필요한 컴파일을 피할 수 있습니다.

오버로드: 패키지를 사용하면 프로시저와 함수를 오버로드할 수 있습니다. 즉, 파라미터의 개수나 데이터 유형을 달리하여 동일한 패키지에 동일한 이름의 다중 서브 프로그램을 생성할 수 있습니다.

참고: 종속성은 "종속성 관리" 단원에서 자세히 다룹니다.

PL/SQL 패키지 구성 요소



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 패키지 구성 요소

패키지는 다음 두 부분으로 생성할 수 있습니다.

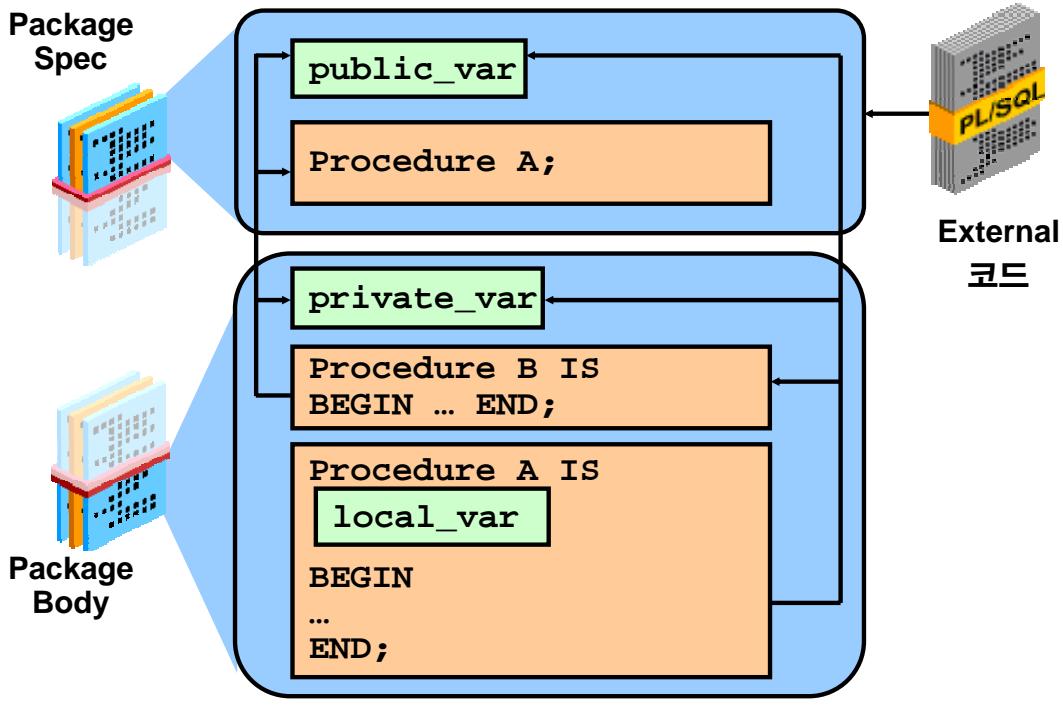
- **Package Spec**은 응용 프로그램에 대한 인터페이스로, 사용할 수 있는 공용(public) 유형, 변수, 상수, 예외, 커서 및 서브 프로그램을 선언합니다. Package Spec은 컴파일러에 대한 지시어인 PRAGMA를 포함할 수도 있습니다.
- **Package Body**는 자체 서브 프로그램을 정의하며 Spec 부분에 선언된 서브 프로그램을 완전히 구현해야 합니다. 또한 유형, 변수, 상수, 예외, 커서 등의 PL/SQL 생성자를 정의할 수도 있습니다.

공용(public) 구성 요소는 Package Spec에서 선언됩니다. Spec은 패키지 기능 유저를 위한 공용 API(Application Programming Interface)를 정의합니다. 즉, 공용 구성 요소는 패키지 external에 있는 모든 Oracle 서버 환경에서 참조할 수 있습니다.

전용(private) 구성 요소는 Package Body에 있으며 동일한 Package Body 내에 있는 다른 생성자만 참조할 수 있습니다. 전용 구성 요소는 패키지의 공용(public) 구성 요소를 참조할 수 있습니다.

참고: Package Spec에 서브 프로그램 선언이 없을 경우 Package Body가 필요하지 않습니다.

패키지 구성 요소의 내부 및 external 표시 여부



ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 구성 요소의 내부 및 external 표시 여부

구성 요소의 표시 여부란 구성 요소를 볼 수 있는지 여부, 즉 다른 구성 요소나 객체에 의해 참조되고 사용되는지 여부를 의미합니다. 구성 요소의 표시 여부는 구성 요소가 로컬(Local)에서 선언되는지 아니면 전역적(Global)으로 선언되는지에 따라 달라집니다.

로컬 구성 요소는 다음과 같이 구성 요소가 선언된 구조 내에서 볼 수 있습니다.

- 서브 프로그램에 정의된 변수는 해당 서브 프로그램 내에서 참조할 수 있으며, external 구성 요소에서는 볼 수 없습니다. 예를 들어, `local_var`은 프로시저 A에 사용할 수 있습니다.
- Package Body에 선언된 전용(private) 패키지 변수는 동일한 Package Body 내에 있는 다른 구성 요소가 참조할 수 있습니다. 패키지 외부에 있는 서브 프로그램이나 객체에서는 전용 패키지 변수를 볼 수 없습니다. 예를 들어, `private_var`은 Package Body 내에서 프로시저 A 및 B에 사용될 수 있지만 패키지 외부에서는 사용될 수 없습니다.

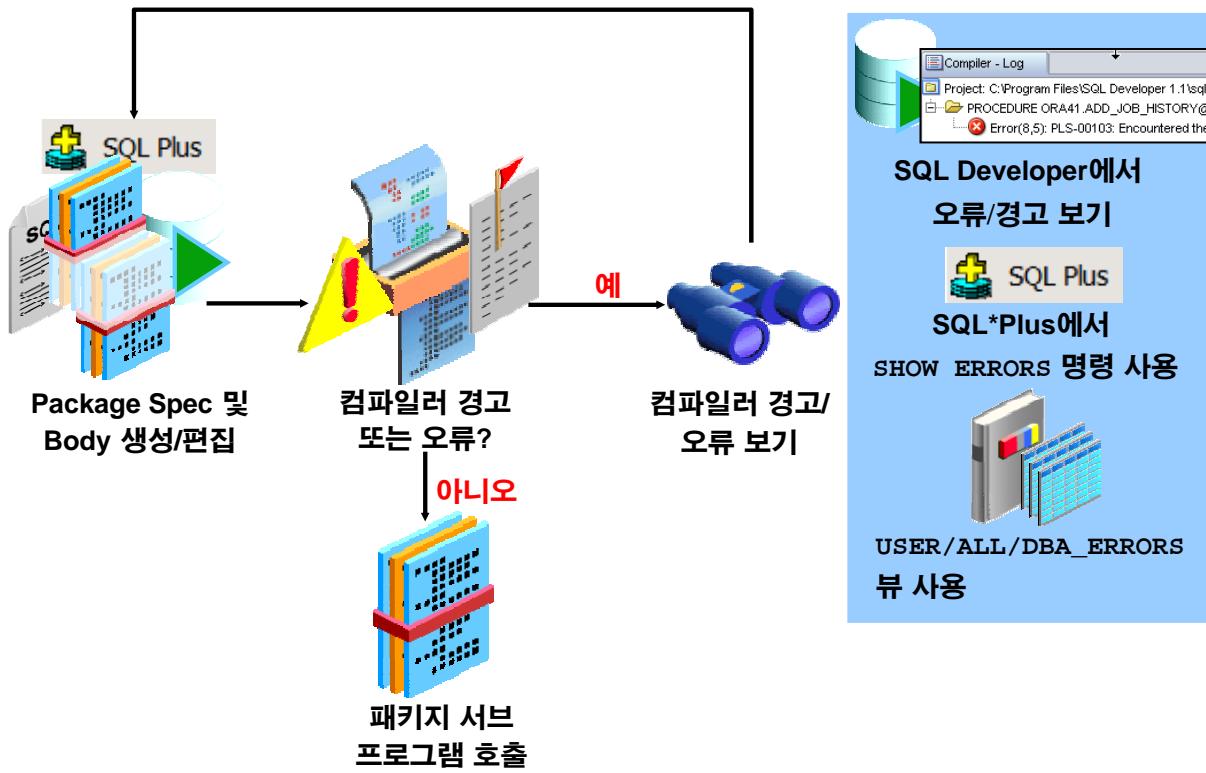
전역적으로 선언된 구성 요소는 다음과 같이 패키지 내부와 external에서 볼 수 있습니다.

Package Spec에 선언된 공용(public) 변수는 해당 패키지의 외부에서 참조하고 변경할 수 있습니다. 예를 들어, `public_var`은 외부에서 참조할 수 있습니다.

- Spec에 있는 패키지 서브 프로그램은 external 코드 소스에서 호출할 수 있습니다. 예를 들어, 프로시저 A는 패키지 external에 있는 환경에서 호출할 수 있습니다.

참고: B 프로시저와 같은 전용(private) 서브 프로그램은 A 프로시저와 같은 공용(public) 서브 프로그램이나 전용 패키지 생성자를 통해서만 호출할 수 있습니다. Package Spec에 선언된 공용 변수는 global 변수입니다.

PL/SQL 패키지 개발: 개요



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 패키지 개발

슬라이드의 그래픽은 패키지를 개발하고 사용하는 기본 단계를 설명합니다.

1. SQL Developer의 Object Navigator 트리 또는 SQL Worksheet 영역을 사용하여 프로시저를 생성합니다.
2. 패키지를 컴파일합니다. 패키지가 데이터베이스에 생성됩니다. CREATE PACKAGE 문은 데이터베이스에서 소스 코드와 컴파일된 *m-code*를 생성하고 저장합니다. 패키지를 컴파일하려면 Object Navigator 트리에서 패키지 이름을 마우스 오른쪽 버튼으로 누르고 Compile을 누릅니다.
3. 컴파일 경고나 오류가 없으면 Oracle 서버 환경의 Package Spec 내에서 공용(public) 생성자를 실행합니다.
4. 컴파일 경고나 오류가 있는 경우 다음 방법 중 하나를 사용하여 경고 또는 오류를 보고 해결할 수 있습니다.
 - SQL Developer 인터페이스 사용(Compiler - Log 탭)
 - SHOW ERRORS SQL*Plus 명령 사용
 - USER/ALL/DBA_ERRORS 뷰 사용

단원 내용

- 패키지의 이점 및 구성 요소 식별
- 패키지 작업:
 - Package Spec 및 Body 생성
 - 패키지 서브 프로그램 호출
 - 패키지 제거
 - 패키지 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Package Spec 생성: CREATE PACKAGE 문 사용

```
CREATE [OR REPLACE] PACKAGE package_name IS|AS
    public type and variable declarations
    subprogram specifications
END [package_name];
```

- OR REPLACE 옵션은 Package Spec을 삭제한 후 다시 생성합니다.
- Package Spec에 선언된 변수는 기본적으로 NULL로 초기화됩니다.
- 패키지에 대한 권한을 부여 받은 유저는 Package Spec에 선언된 모든 생성자를 볼 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Package Spec 작성

패키지를 작성하려면 Package Spec 내에서 모든 공용(public) 생성자를 선언하십시오.

- 기존 Package Spec을 덮쳐쓰려면 OR REPLACE 옵션을 지정합니다.
- 필요할 경우 선언 내에서 상수 값 또는 공식으로 변수를 초기화합니다. 그렇지 않으면 변수는 암시적으로 NULL로 초기화됩니다.

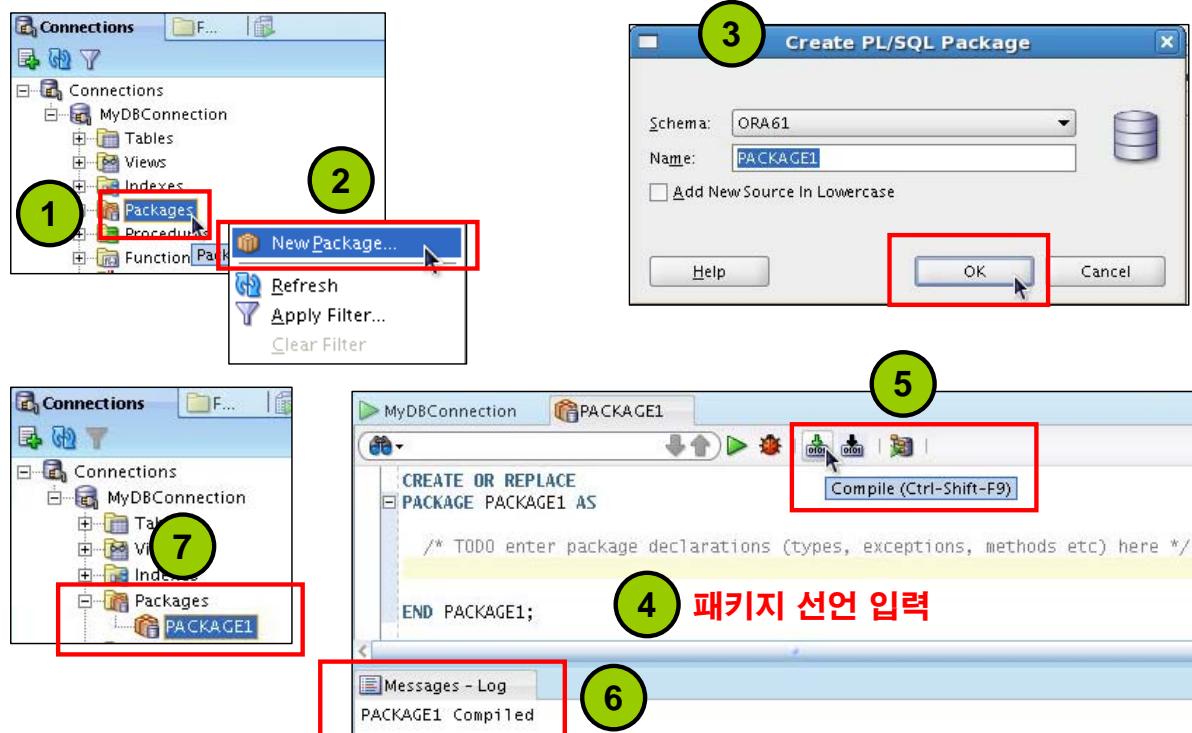
다음은 패키지 구문 항목에 대한 정의입니다.

- **package_name**은 패키지의 이름을 지정합니다. 이 이름은 소유 스키마 내에 있는 객체 간에 고유해야 합니다. END 키워드 뒤에 패키지 이름이 올 수도 있습니다.
- **public type and variable declarations**는 공용(public) 변수, 상수, 커서, 예외, 유저 정의 유형 및 서브타입을 선언합니다.
- **subprogram specification**은 공용(public) 프로시저나 함수 선언을 지정합니다.

Package Spec은 IS(또는 AS) 키워드와 PL/SQL 블록 없이 세미콜론으로 끝나는 프로시저 및 함수 머리글을 포함해야 합니다. Package Spec에 선언된 프로시저나 함수는 Package Body에서 구현됩니다.

오라클 데이터베이스는 Package Spec과 Body를 따로 저장합니다. 따라서 프로그램 생성자를 호출하거나 참조하는 다른 스키마 객체를 무효화하지 않고도 Package Body에서 프로그램 생성자 구현을 변경할 수 있습니다.

Package Spec 생성: SQL Developer 사용



ORACLE®

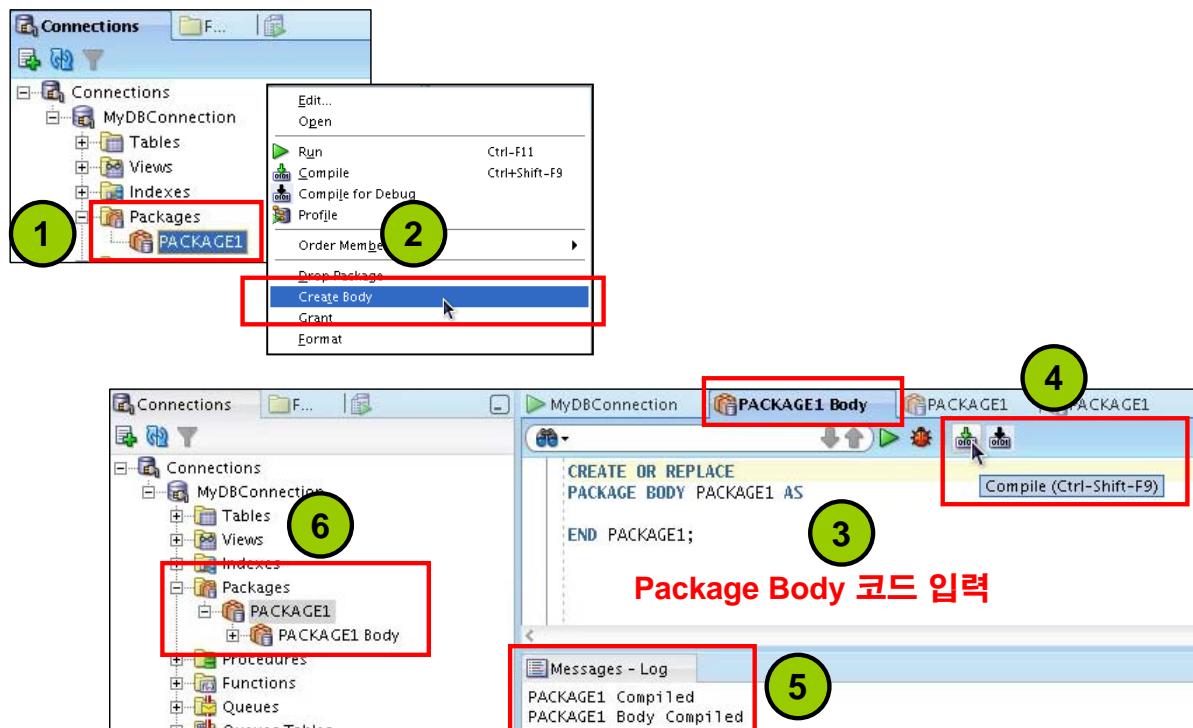
Copyright © 2009, Oracle. All rights reserved.

Package Spec 생성: SQL Developer 사용

다음과 같이 SQL Developer를 사용하여 Package Spec을 생성할 수 있습니다.

1. Connections 탐색 트리에서 **Packages** 노드를 마우스 오른쪽 버튼으로 누릅니다.
2. 단축 메뉴에서 **New Package**를 선택합니다.
3. **Create PL/SQL Package** window에서 스키마 이름을 선택하고 새 패키지의 이름을 입력한 다음 OK를 누릅니다. 새 패키지의 템이 새 패키지의 셀과 함께 표시됩니다.
4. 새 패키지에 대한 코드를 입력합니다.
5. 새 패키지를 컴파일하거나 저장(기본 도구 모음의 Save 아이콘 사용)합니다.
6. **Messages - Log** 템은 컴파일의 성공 여부를 표시합니다.
7. 새로 생성된 패키지는 Connections 탐색 트리의 **Packages** 노드 아래에 표시됩니다.

Package Body 생성: SQL Developer 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Package Body 생성: SQL Developer 사용

다음과 같이 SQL Developer를 사용하여 Package Spec을 생성할 수 있습니다.

1. **Connections** 탐색 트리의 **Packages** 노드에서 **Body**를 생성할 패키지 이름을 마우스 오른쪽 버튼으로 누릅니다.
2. 단축 메뉴에서 **Create Body**를 선택합니다. 새 **Package Body**의 템이 새 **Package Body**의 셀과 함께 표시됩니다.
3. 새 **Package Body**에 대한 코드를 입력합니다.
4. 새 **Package Body**를 컴파일하거나 저장합니다.
5. **Messages - Log** 템은 컴파일의 성공 여부를 표시합니다.
6. 새로 생성된 **Package Body**는 **Connections** 탐색 트리의 **Packages** 노드 아래에 표시됩니다.

Package Spec 예제: comm_pkg

```
-- The package spec with a public variable and a
-- public procedure that are accessible from
-- outside the package.

CREATE OR REPLACE PACKAGE comm_pkg IS
    v_std_comm NUMBER := 0.10; --initialized to 0.10
    PROCEDURE reset_comm(p_new_comm NUMBER);
END comm_pkg;
/
```

- **V_STD_COMM**은 0.10으로 초기화된 공용 (public) global 변수입니다.
- **RESET_COMM**은 몇 가지 업무 규칙을 기준으로 표준 커미션을 재설정하는 데 사용되는 공용 (public) 프로시저이며, Package Body에서 구현됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Package Spec 예제: comm_pkg

슬라이드의 예제에서는 커미션 계산을 위해 업무 처리 규칙을 관리하는 comm_pkg라는 패키지를 생성합니다.

v_std_comm 공용(public) Global 변수는 유저 세션에 대해 최대로 허용할 수 있는 커미션 비율을 수용하도록 선언되며, 0.10(10%)으로 초기화됩니다.

reset_comm 공용(public) 프로시저는 커미션 검증 규칙이 허용될 경우 표준 커미션 비율을 갱신하는 새 커미션 비율을 허용하도록 선언됩니다. 커미션을 재설정하는 검증 규칙은 공용으로 설정되지 않으므로 Package Spec에 표시되지 않습니다. 검증 규칙은 Package Body에서 전용(private) 함수를 사용하여 관리됩니다.

Package Body 작성

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS | AS
    private type and variable declarations
    subprogram bodies
[BEGIN initialization statements]
END [package_name];
```

- OR REPLACE 옵션은 Package Body를 삭제한 후 재생성합니다.
- Package Body에 정의된 식별자는 전용(private)이므로 Package Body 외부에서 볼 수 없습니다.
- 모든 전용(private) 생성자는 참조되기 전에 선언되어야 합니다.
- 공용(public) 생성자는 Package Body에 표시됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Package Body 작성

Package Body를 작성하여 모든 공용(public) 서브 프로그램 및 지원되는 전용(private) 생성자를 정의하고 구현할 수 있습니다. Package Body를 생성할 때는 다음 단계를 따르십시오.

- 기존 Package Body를複쳐 쓰려면 OR REPLACE 옵션을 지정합니다.
- 서브 프로그램을 적절한 순서로 정의합니다. 기본 원칙은 동일한 Package Body에 있는 다른 구성 요소가 참조하기 전에 변수나 서브 프로그램을 선언하는 것입니다. 일반적으로 Package Body에서 모든 전용(private) 변수와 서브 프로그램이 제일 먼저 정의되고, 공용(public) 서브 프로그램이 제일 나중에 정의됩니다.
- Package Spec에서 선언한 모든 프로시저 또는 함수의 구현을 Package Body 내에서 완료합니다.

다음은 Package Body 구문 항목에 대한 정의입니다.

- **package_name**은 패키지의 이름을 지정합니다. 이 이름은 Package Spec과 동일해야 합니다. END 키워드 뒤에 패키지 이름이 올 수도 있습니다.
- **private type and variable declarations**는 전용(private) 변수, 상수, 커서, 예외, 유저 정의 유형 및 서브타입을 선언합니다.
- **subprogram specification**은 전용(private)/공용(public) 프로시저나 함수의 전체 구현을 지정합니다.
- **[BEGIN initialization statements]**는 패키지가 처음 참조될 때 실행되는 선택적 초기화 코드 블록입니다.

Package Body 예제: comm_pkg

```

CREATE OR REPLACE PACKAGE BODY comm_pkg IS
  FUNCTION validate(p_comm NUMBER) RETURN BOOLEAN IS
    v_max_comm employees.commission_pct%type;
  BEGIN
    SELECT MAX(commission_pct) INTO v_max_comm
    FROM employees;
    RETURN (p_comm BETWEEN 0.0 AND v_max_comm);
  END validate;

  PROCEDURE reset_comm (p_new_comm NUMBER) IS
  BEGIN
    IF validate(p_new_comm) THEN
      v_std_comm := p_new_comm; -- reset public var
    ELSE RAISE_APPLICATION_ERROR(
      -20210, 'Bad Commission');
    END IF;
  END reset_comm;
END comm_pkg;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Package Body 예제: comm_pkg

슬라이드는 커미션이 적합한지 검사하는 validate라는 전용(private) 함수가 포함된 comm_pkg의 전체 Package Body를 보여줍니다. 검증 시 커미션은 양수여야 하며 기존 사원의 최고 커미션보다 작아야 합니다. reset_comm 프로시저는 v_std_comm에서 표준 커미션을 변경하기 전에 전용 검증 함수를 호출합니다. 위 예제에서 다음을 살펴보십시오.

- reset_comm 프로시저에서 참조하는 v_std_comm 변수는 공용(public) 변수입니다. Package Spec에 선언된 v_std_comm과 같은 변수는 조건 없이 직접 참조될 수 있습니다.
- reset_comm 프로시저는 Spec에서 공용(public) 정의를 구현합니다.
- comm_pkg Body에서 validate 함수는 전용(private) 함수이며 조건 없이 reset_comm 프로시저에서 직접 참조됩니다.

참고: reset_comm 프로시저가 validate 함수를 참조하기 때문에 validate 함수는 reset_comm 프로시저 앞에 표시됩니다. 서브 프로그램이 나타나는 순서를 변경해야 할 경우 Package Body에 서브 프로그램의 사전 선언을 생성할 수 있습니다. Package Spec이 서브 프로그램 사양 없이 유형, 상수, 변수, 예외만 선언할 경우 Package Body는 필요하지 않습니다. 그러나 Package Spec에 선언된 항목을 초기화하는 데 Body를 사용할 수 있습니다.

패키지 서브 프로그램 호출: 예제

```
-- Invoke a function within the same package:
CREATE OR REPLACE PACKAGE BODY comm_pkg IS ...
  PROCEDURE reset_comm(p_new_comm NUMBER) IS
    BEGIN
      IF validate(p_new_comm) THEN
        v_std_comm := p_new_comm;
      ELSE ...
      END IF;
    END reset_comm;
  END comm_pkg;
```

```
-- Invoke a package procedure from SQL*Plus:
EXECUTE comm_pkg.reset_comm(0.15)
```

```
-- Invoke a package procedure in a different schema:
EXECUTE scott.comm_pkg.reset_comm(0.15)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 서브 프로그램 호출

데이터베이스에 패키지를 저장한 후에는 동일한 패키지 내에 공용(public) 또는 전용(private) 서브 프로그램을 호출할 수도 있고, 패키지 external에서 공용 서브 프로그램을 호출할 수 있습니다. 패키지 external에서 호출될 경우 패키지 이름을 사용하여 서브 프로그램을 패키지 이름과 같이 호출하십시오. package_name.subprogram 구문을 사용하십시오.

동일한 패키지 내에서 호출될 때 서브 프로그램을 패키지 이름과 같이 호출하는 것은 선택 사항입니다.

예제 1: 동일한 패키지 내에 있는 reset_comm 프로시저에서 validate 함수를 호출합니다. 패키지 이름 접두어는 선택 사항입니다.

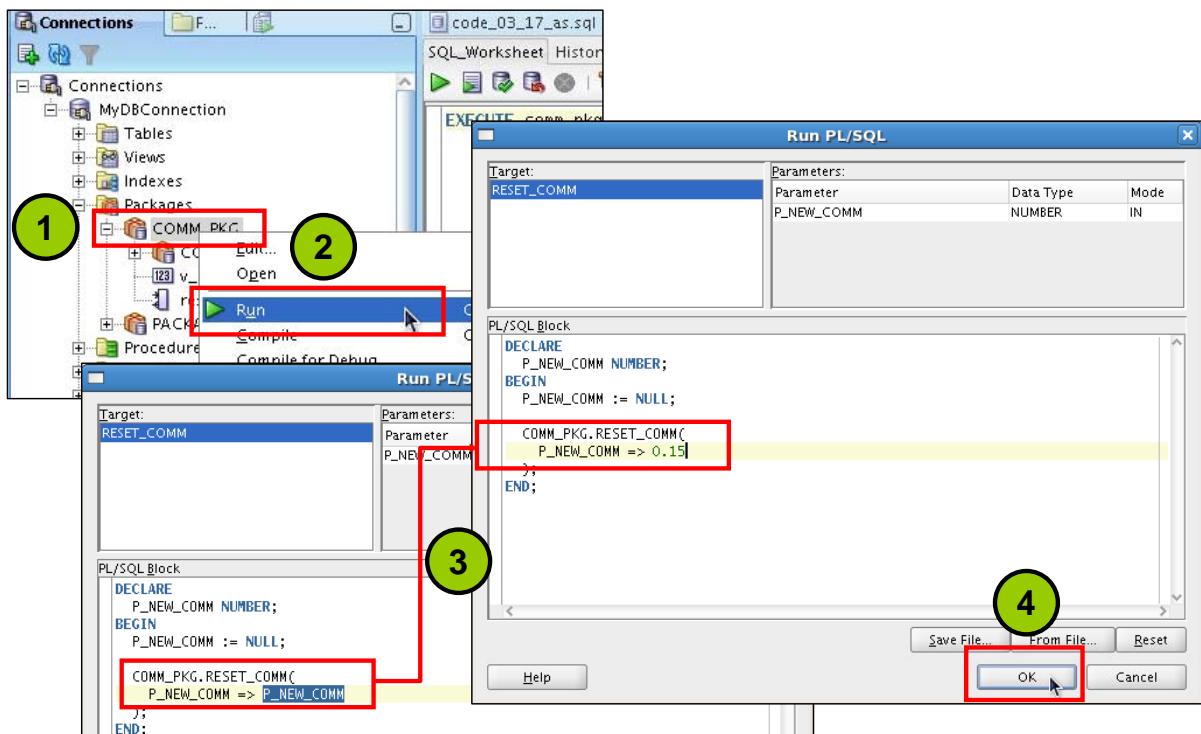
예제 2: SQL*Plus(패키지 external에 있는 환경)에서 reset_comm 프로시저를 호출하여 유저 세션의 일반적인 커미션을 0.15로 재설정합니다.

예제 3: SCOTT이라는 스키마 유저가 소유한 reset_comm 프로시저를 호출합니다. SQL*Plus를 사용할 경우 한정된 패키지 프로시저 앞에 스키마 이름이 붙습니다. 이 작업은 schema.package_name을 참조하는 동의어를 사용하여 간단하게 수행할 수 있습니다.

reset_comm 패키지 프로시저가 생성된 원격 데이터베이스에 대해 NY라는 데이터베이스 링크가 생성되었다고 가정합시다. 원격 프로시저를 호출하려면 다음을 사용하십시오.

```
EXECUTE comm_pkg.reset_comm@NY(0.15)
```

패키지 서브 프로그램 호출: SQL Developer 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 서브 프로그램 호출: SQL Developer 사용

다음과 같이 SQL Developer를 사용하여 패키지 서브 프로그램을 호출할 수 있습니다.

1. Navigation 트리의 Packages 노드에서 패키지 이름을 마우스 오른쪽 버튼으로 누릅니다.
2. 이동 가능한 메뉴에서 Run을 선택합니다. Run PL/SQL window가 표시됩니다. Run PL/SQL window를 사용하여 PL/SQL 함수 또는 프로시저를 실행하기 위한 파라미터 값을 지정할 수 있습니다. 패키지를 지정하는 경우 패키지에서 함수 또는 프로시저를 선택하십시오. 다음을 지정합니다.
 - a. **Target:** 실행할 함수 또는 프로시저의 이름을 선택합니다.
 - b. **Parameters:** 이 섹션에는 지정된 대상의 각 파라미터가 나열됩니다. 각 파라미터의 모드는 IN(값이 전달됨), OUT(값이 반환됨) 또는 IN/OUT(값이 전달되고 함수 또는 프로시저 작업의 결과가 파라미터에 저장됨)일 수 있습니다.
3. PL/SQL 블록 섹션에서 이 블록의 형식 IN 및 IN/OUT 파라미터 사양을 함수 또는 프로시저를 실행하는 데 사용할 실제 값으로 변경합니다. 예를 들어, P_NEW_COMM이라는 입력 파라미터 값으로 0.15을 지정하려면 P_NEW_COMM => P_NEW_COMM을 P_NEW_COMM => 0.15로 변경합니다.
4. OK를 누릅니다. SQL Developer가 함수 또는 프로시저를 실행합니다.

본문 없는 패키지 생성 및 사용

```
CREATE OR REPLACE PACKAGE global_consts IS
  c_mile_2_kilo CONSTANT NUMBER := 1.6093;
  c_kilo_2_mile CONSTANT NUMBER := 0.6214;
  c_yard_2_meter CONSTANT NUMBER := 0.9144;
  c_meter_2_yard CONSTANT NUMBER := 1.0936;
END global_consts;
```

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('20 miles = ' ||
    20 * global_consts.c_mile_2_kilo || ' km');
END;
```

```
SET SERVEROUTPUT ON
CREATE FUNCTION mtr2yrd(p_m NUMBER) RETURN NUMBER IS
BEGIN
  RETURN (p_m * global_consts.c_meter_2_yard);
END mtr2yrd;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(mtr2yrd(1))
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

본문 없는 패키지 생성 및 사용

독립형 서브 프로그램 내에 선언된 변수와 상수는 서브 프로그램이 실행되는 동안에만 존재합니다. 유저 세션 기간 동안 존재하는 데이터를 제공하려면 공용(public) Global 변수와 상수 선언을 포함하는 Package Spec을 생성하십시오. 이 경우 Package Body가 없는 Package Spec이 생성되는데 이를 본문 없는 패키지(Bodiless Package)라고 합니다. 이 단원의 앞 부분에서 설명했듯이 Spec이 유형, 상수, 변수, 예외만 선언할 경우 Package Body는 필요하지 않습니다.

예제

위 슬라이드에 표시된 첫번째 코드 상자에서는 변환율에 사용될 몇 가지 상수를 사용하여 본문이 없는 Package Spec이 생성됩니다. Package Body는 이 Package Spec을 지원하는 데 필요하지 않습니다. 슬라이드의 코드 예제를 실행하기 전에 SET SERVEROUTPUT ON 문을 실행했다고 가정합니다.

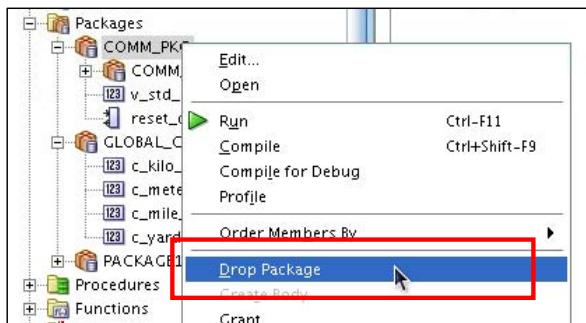
두번째 코드 상자에서는 상수 식별자 앞에 패키지 이름을 삽입하여 global_consts 패키지에서 c_mile_2_kilo 상수를 참조합니다.

세번째 예제에서는 미터를 야드로 변환하는 독립형 함수 c_mtr2yrd를 생성하고 global_consts 패키지에 선언된 상수 변환율 c_meter_2_yard를 사용합니다. 함수는 DBMS_OUTPUT.PUT_LINE 파라미터에서 호출됩니다.

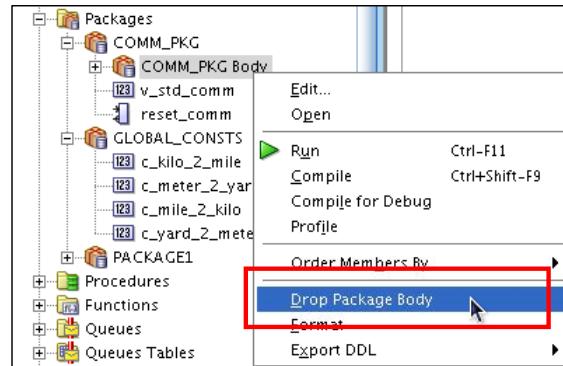
따라야 할 규칙: 패키지 외부에서 변수, 커서, 상수 또는 예외를 참조할 때 이를 패키지 이름으로 한정해야 합니다.

패키지 제거: SQL Developer 또는 SQL DROP 문 사용

Package Spec 및 Body 삭제



Package Body만 삭제



```
-- Remove the package specification and body  
DROP PACKAGE package_name;
```

```
-- Remove the package body only  
DROP PACKAGE BODY package_name;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 제거

패키지가 더 이상 필요하지 않으면 SQL Developer에서 SQL 문을 사용하여 제거할 수 있습니다. 패키지는 두 부분으로 구성됩니다. 따라서 전체 패키지를 제거할 수도 있고, Package Body만 제거하고 Package Spec은 보존할 수도 있습니다.

데이터 딕셔너리를 사용하여 패키지 보기

```
-- View the package specification.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE'
ORDER BY LINE;
```

TEXT
1 PACKAGE comm_pkg IS
2 std_comm NUMBER := 0.10; --initialized to 0.10
3 PROCEDURE reset_comm(new_comm NUMBER);
4 END comm_pkg;

```
-- View the package body.
SELECT text
FROM user_source
WHERE name = 'COMM_PKG' AND type = 'PACKAGE BODY'
ORDER BY LINE;
```

TEXT
1 PACKAGE BODY comm_pkg IS
2 FUNCTION validate(comm NUMBER) RETURN BOOLEAN IS
3 max_comm employees.commission_pct%type;
4 BEGIN
5 SELECT MAX(commission_pct) INTO max_comm
6 FROM employees;
7 RETURN (comm BETWEEN 0.0 AND max_comm);

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리에서 패키지 보기

PL/SQL 패키지의 소스 코드는 USER_SOURCE 및 ALL_SOURCE 데이터 딕셔너리 뷰에도 저장됩니다. USER_SOURCE 테이블은 소유하고 있는 PL/SQL 코드를 표시하는 데 사용됩니다. ALL_SOURCE 테이블은 해당 서브 프로그램 코드의 소유자로부터 EXECUTE 권한을 부여 받은 PL/SQL 코드를 표시하는 데 사용되며, 위의 열 외에도 OWNER 열을 제공합니다.

패키지를 query하는 경우 다음 조건을 사용하십시오.

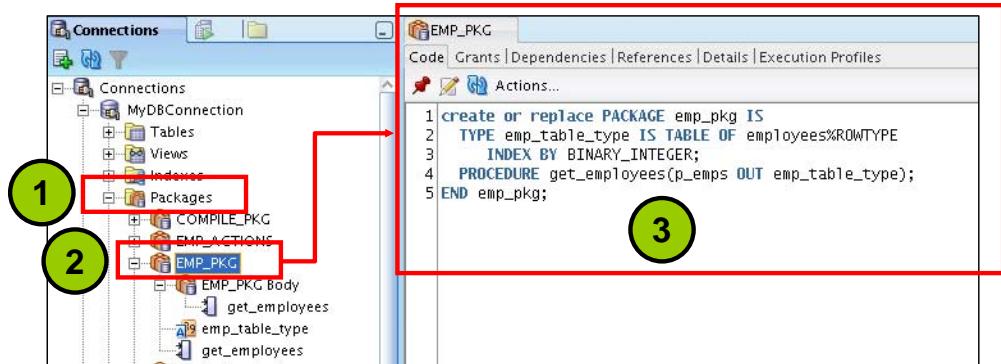
- Package Spec의 소스 코드를 표시하려면 TYPE 열이 'PACKAGE'와 일치
- Package Body의 소스 코드를 표시하려면 TYPE 열이 'PACKAGE BODY'와 일치

Packages 노드에 있는 패키지 이름을 사용하여 SQL Developer에서 Package Spec 및 Body를 볼 수도 있습니다.

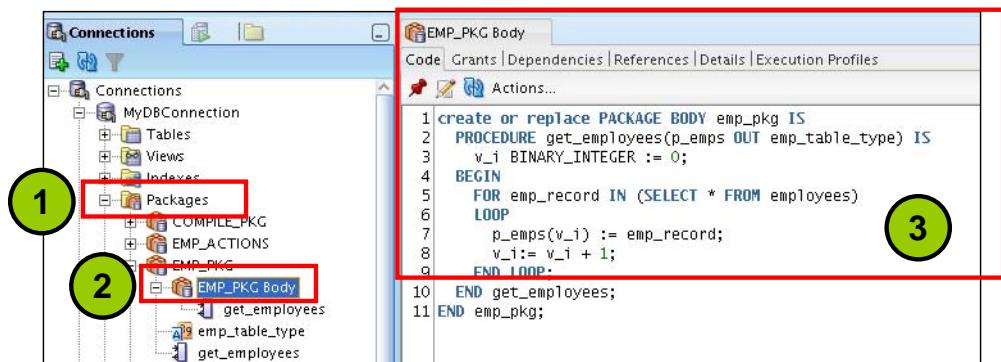
참고: Oracle PL/SQL 내장 패키지 또는 WRAP 유ти리티나 난독 처리(Obfuscation)를 사용하여 소스 코드가 래핑된 PL/SQL의 경우에는 소스 코드를 표시할 수 없습니다. PL/SQL 소스 코드를 난독 처리하거나 래핑하는 것은 이후의 단원에서 다룹니다. 때로는 슬라이드 예제와 같이 SQL Worksheet 도구 모음에서 Run Script 아이콘 대신 Execute Statement(F9) 아이콘을 누를 때 더 좋은 형식의 출력이 Results 탭에 표시될 수 있습니다.

SQL Developer를 사용하여 패키지 보기

Package Spec을 보려면 패키지 이름을 누릅니다.



Package Body를 보려면 Package Body를 누릅니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 패키지 보기

SQL Developer에서 Package Spec을 보려면 다음 단계를 따르십시오.

1. Connections 탭에서 Packages 노드를 누릅니다.
2. 패키지 이름을 누릅니다.
3. 슬라이드에 나와 있는 것처럼 Package Spec 코드가 Code 탭에 표시됩니다.

SQL Developer에서 Package Body를 보려면 다음 단계를 따르십시오.

1. Connections 탭에서 Packages 노드를 누릅니다.
2. Package Body를 누릅니다.
3. 슬라이드에 나와 있는 것처럼 Package Body 코드가 Code 탭에 표시됩니다.

패키지 작성 지침

- 패키지를 일반적인 용도로 개발합니다.
- Package Spec은 Body 앞에 정의합니다.
- Package Spec은 공용(public) 생성자만 포함해야 합니다.
- 항목을 세션이나 트랜잭션 전체에서 유지 관리해야 할 경우 Package Body 선언 부분에 해당 항목을 배치합니다.
- Fine-Grain Dependency 관리는 Package Spec이 변경될 때 참조 서브 프로그램을 재컴파일할 필요성을 줄입니다.
- Package Spec은 생성자를 가능한 한 적게 포함해야 합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

패키지 작성 지침

이후 응용 프로그램에 다시 사용할 수 있도록 패키지는 가능한 한 일반적으로 유지하십시오. 또한 패키지를 작성할 때 Oracle 서버에서 제공하는 기능과 중복되지 않도록 하십시오.

Package Spec은 응용 프로그램 설계를 반영하므로 Package Body를 정의하기 전에 Package Spec을 정의하십시오. Package Spec은 패키지 유저가 볼 수 있는 생성자만 포함해야 합니다. 따라서 다른 개발자가 관련 없는 세부 정보를 토대로 코드를 작성하여 패키지를 잘못 사용할 가능성이 없어집니다.

항목을 세션이나 트랜잭션 전체에서 유지 관리해야 할 경우 Package Body 선언 부분에 해당 항목을 배치합니다. 예를 들어, NUMBER_EMPLOYED라는 변수를 사용하는 프로시저에 대한 각각의 호출을 유지 관리해야 할 경우 해당 변수를 전용(private) 변수로 선언하십시오. Package Spec에서 Global 변수로 선언될 경우 해당 Global 변수의 값은 패키지의 생성자가 처음으로 호출된 세션에서 초기화됩니다.

Oracle Database 11g 이전에는 Package Body를 변경할 경우 종속 생성자를 재컴파일할 필요가 없지만, Package Spec을 변경할 경우 해당 패키지를 참조하는 모든 내장 서브 프로그램을 재컴파일해야 했습니다. Oracle Database 11g에서는 이러한 종속성을 줄여 이제 단위 내의 요소 레벨에서 종속성이 추적됩니다. Fine-Grain Dependency 관리는 이후의 단원에서 다룹니다.

퀴즈

Package Spec은 응용 프로그램에 대한 인터페이스로, 사용할 수 있는 공용(public) 유형, 변수, 상수, 예외, 커서 및 서브 프로그램을 선언합니다. Package Spec은 컴파일러에 대한 지시어인 PRAGMA를 포함할 수도 있습니다.

1. 맞습니다.
2. 틀립니다.



Copyright © 2009, Oracle. All rights reserved.

정답: 1

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 패키지 설명 및 패키지 구성 요소 나열
- 관련 변수, 커서, 상수, 예외, 프로시저 및 함수를 그룹화하는 패키지 생성
- 패키지 생성자를 공용(public) 또는 전용(private)으로 지정
- 패키지 생성자 호출
- 본문이 없는 패키지 사용법 설명

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

패키지에 관련 프로시저와 함수를 그룹화할 수 있습니다. 패키지를 사용하면 구성, 관리, 보안 및 성능이 향상됩니다.

패키지는 Package Spec과 Package Body로 구성됩니다. Package Spec에 영향을 주지 않고 Package Body를 변경할 수 있습니다.

패키지를 사용하면 유저가 보지 못하도록 소스 코드를 숨길 수 있습니다. 패키지를 처음으로 호출할 때 전체 패키지가 메모리에 로드됩니다. 따라서 이후 호출에 대한 디스크 액세스가 감소합니다.

연습 3 개요: 패키지 생성 및 사용

이 연습에서는 다음 내용을 다룹니다.

- 패키지 생성
- 패키지 프로그램 단위 호출



Copyright © 2009, Oracle. All rights reserved.

연습 3: 개요

이 연습에서는 Package Spec과 Package Body를 생성한 다음 예제 데이터를 사용하여 패키지에서 생성자를 호출합니다.

참고: SQL Developer를 사용하는 경우에는 컴파일 타임 오류가 Message Log 탭에 표시됩니다.
SQL*Plus를 사용하여 저장된 코드를 생성하는 경우에는 SHOW ERRORS를 사용하여 컴파일 오류를 보십시오.

4 패키지 작업

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 패키지 프로시저 및 함수 오버로드
- 사전 선언 사용
- Package Body에 초기화 블록 생성
- 세션 기간 동안 지속되는 패키지 데이터 상태 관리
- 패키지에서 연관 배열(인덱스화된 테이블) 및 레코드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 오버로드, 사전 참조, 1회 실행 프로시저 및 변수, 상수, 예외, 커서의 지속성을 비롯한 PL/SQL의 고급 기능에 대해 소개하며 SQL 문에 사용되는 함수를 패키지화한 결과에 대해 설명합니다.

단원 내용

- 패키지 서브 프로그램 오버로드, 사전 선언 사용,
Package Body에서 초기화 블록 생성
- 세션 기간 동안 지속되는 패키지 데이터 상태 관리 및
패키지에서 연관 배열(인덱스화된 테이블)과 레코드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에서 서브 프로그램 오버로드

- 동일한 이름의 서브 프로그램을 두 개 이상 생성할 수 있습니다.
- 서브 프로그램에서 사용하는 형식 파라미터의 개수, 순서 또는 데이터 유형 계열이 달라야 합니다.
- 서로 다른 데이터를 처리하는 서브 프로그램을 여러 가지 방법으로 호출할 수 있습니다.
- 기존 코드의 손실 없이 기능을 확장하는 방법을 제공합니다. 즉, 기존 서브 프로그램에 새 파라미터를 추가합니다.
- 로컬 서브 프로그램, 패키지 서브 프로그램 및 유형 메소드는 오버로드하지만 독립형 서브 프로그램은 오버로드하지 않습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

서브 프로그램 오버로드

PL/SQL에서 오버로드 기능을 사용하면 동일한 이름의 패키지화된 서브 프로그램을 두 개 이상 개발할 수 있습니다. 오버로드는 서브 프로그램에 데이터 유형이 다른 유사한 파라미터 집합을 사용하고자 할 때 유용합니다. 예를 들면, 숫자나 날짜를 문자열로 변환하도록 하는 TO_CHAR 함수를 여러 가지 방법으로 호출할 수 있습니다.

PL/SQL에서는 패키지 서브 프로그램 이름과 객체 유형 메소드를 오버로드할 수 있습니다.

여기에서 중요한 규칙은 형식 파라미터의 개수, 순서 또는 데이터 유형 계열이 다를 경우 동일한 이름을 여러 서브 프로그램에 사용할 수 있다는 점입니다.

다음과 같은 경우에 오버로드를 사용하십시오.

- 둘 이상의 서브 프로그램에 대해 처리 규칙은 유사하지만 사용되는 파라미터의 개수나 유형이 서로 다를 경우.
- 검색 조건을 다르게 하여 다른 데이터를 찾는 대체 방법을 제공하려는 경우. 예를 들면, 사원 ID로 사원을 찾을 수도 있고 성으로 사원을 찾을 수도 있습니다. 논리는 본질적으로 같지만 파라미터나 검색 조건이 다릅니다.
- 기존 코드를 바꾸고 싶지 않을 때 기능을 확장하려는 경우.

참고: 독립형 서브 프로그램은 오버로드할 수 없습니다. 로컬 서브 프로그램을 객체 유형 메소드로 생성하는 방법은 본 과정에서 다루지 않습니다.

서브 프로그램 오버로드(계속)

제한 사항

오버로드 할 수 없는 경우

- 두 서브 프로그램이 형식 파라미터의 데이터 유형만 다르고, 이 데이터 유형이 동일한 계열에 속하는 경우(NUMBER와 DECIMAL은 동일한 계열에 속함).
- 두 서브 프로그램이 형식 파라미터의 서브타입만 다르고, 이 서브타입이 동일한 계열의 유형을 기반으로 하는 경우(VARCHAR 및 STRING은 VARCHAR2의 PL/SQL 서브타입임).
- 두 함수가 반환 유형만 다른 경우(해당 유형이 다른 계열에 속하더라도 적용됨).

위와 같은 특징을 가진 서브 프로그램을 오버로드하면 런타임 오류가 발생합니다.

참고: 파라미터 이름이 동일한 경우에도 위와 같은 제한 사항이 적용됩니다. 서로 다른 파라미터 이름을 사용할 경우 파라미터에 대해 이름 지정 표기법을 사용하여 서브 프로그램을 호출할 수 있습니다.

호출 분석

컴파일러는 호출과 일치하는 선언을 찾으려고 시도합니다. 먼저 현재 범위에서 검색한 다음 필요한 경우 이어지는 포함 범위에서 검색합니다. 호출된 서브 프로그램의 이름과 이름이 일치하는 서브 프로그램 선언을 한 개 이상 찾으면 컴파일러는 검색을 정지합니다. 동일한 범위 레벨에서 서브 프로그램의 이름이 유사하게 지정된 경우 컴파일러를 사용하려면 실제 파라미터와 형식 파라미터 간의 개수, 순서, 데이터 유형이 정확히 일치해야 합니다.

프로시저 오버로드 예제: Package Spec 생성

```

CREATE OR REPLACE PACKAGE dept_pkg IS
    PROCEDURE add_department
        (p_deptno departments.department_id%TYPE,
         p_name   departments.department_name%TYPE := 'unknown',
         p_loc    departments.location_id%TYPE := 1700);

    PROCEDURE add_department
        (p_name   departments.department_name%TYPE := 'unknown',
         p_loc    departments.location_id%TYPE := 1700);
END dept_pkg;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

오버로드: 예제

슬라이드는 add_department라는 프로시저가 오버로드된 dept_pkg Package Spec을 보여줍니다. 첫번째 선언에서는 department 테이블에 삽입되는 새 부서 레코드의 데이터를 제공하기 위해 세 개의 파라미터를 사용합니다. 두번째 선언에서는 두 개의 파라미터만 사용하는데, 이 버전에서는 오라클 시퀀스를 통해 부서 ID를 내부적으로 생성하기 때문입니다. 슬라이드의 예제와 같이 데이터베이스 테이블에서 열을 채우는 데 사용되는 변수의 %TYPE 속성을 사용하여 데이터 유형을 지정하는 것이 좋지만 다음과 같이 데이터 유형을 지정할 수도 있습니다.

```

CREATE OR REPLACE PACKAGE dept_pkg_method2 IS
    PROCEDURE add_department(p_deptno NUMBER,
                             p_name   VARCHAR2 := 'unknown', p_loc NUMBER := 1700);
    ...

```

프로시저 오버로드 예제: Package Body 생성

```
-- Package body of package defined on previous slide.
CREATE OR REPLACE PACKAGE BODY dept_pkg  IS
PROCEDURE add_department -- First procedure's declaration
(p_deptno departments.department_id%TYPE,
 p_name   departments.department_name%TYPE := 'unknown',
 p_loc    departments.location_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments(department_id,
    department_name, location_id)
  VALUES  (p_deptno, p_name, p_loc);
END add_department;
PROCEDURE add_department -- Second procedure's declaration
(p_name   departments.department_name%TYPE := 'unknown',
 p_loc    departments.location_id%TYPE := 1700) IS
BEGIN
  INSERT INTO departments (department_id,
    department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_department;
END dept_pkg; /
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

오버로드: 예제(계속)

add_department를 명시적으로 제공된 부서 ID와 함께 호출하면 PL/SQL은 첫번째 버전의 프로시저를 사용합니다. 다음 예제를 살펴보십시오.

```
EXECUTE dept_pkg.add_department(980, 'Education', 2500)
SELECT * FROM departments
WHERE department_id = 980;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
980	Education		2500
1 rows selected			

부서 ID 없이 add_department를 호출하면 PL/SQL은 두번째 버전을 사용합니다.

```
EXECUTE dept_pkg.add_department ('Training', 2400)
SELECT * FROM departments
WHERE department_name = 'Training';
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	Training		2400
1 rows selected			

오버로드 및 STANDARD 패키지

- STANDARD 패키지는 PL/SQL 환경과 내장 함수를 정의합니다.
- 대부분의 내장 함수는 오버로드됩니다. TO_CHAR 함수를 예로 들 수 있습니다.

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN
    VARCHAR2;
. . .
```

- 패키지 이름으로 한정하지 않는 한, 내장 서브 프로그램과 이름이 동일한 PL/SQL 서브 프로그램은 로컬 컨텍스트에서 표준 선언을 재정의합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

오버로드 및 STANDARD 패키지

STANDARD 패키지는 PL/SQL 환경을 정의하며, PL/SQL 프로그램에 자동으로 제공되는 유형, 예외 및 서브 프로그램을 전역적(global)으로 선언합니다. STANDARD 패키지에 있는 대부분의 내장 함수는 오버로드됩니다. 예를 들어 TO_CHAR 함수에는 슬라이드에 표시된 것처럼 네 개의 선언이 있습니다. TO_CHAR 함수는 DATE 또는 NUMBER 데이터 유형을 가져와서 이를 문자 데이터 유형으로 변환합니다. 날짜 또는 숫자가 변환되어야 하는 형식을 함수 호출에 지정할 수도 있습니다.

다른 PL/SQL 프로그램에서 내장 서브 프로그램을 재선언할 경우 로컬 선언이 표준 또는 내장 서브 프로그램을 재정의합니다. 내장 서브 프로그램에 액세스하려면 이를 패키지 이름으로 한정해야 합니다. 예를 들어, TO_CHAR 함수를 재선언하여 내장 함수에 액세스하려는 경우에는 STANDARD.TO_CHAR로 참조합니다.

내장 서브 프로그램을 독립형 서브 프로그램으로 다시 선언할 경우 서브 프로그램에 액세스하려면 이를 스키마 이름으로 한정(예: SCOTT.TO_CHAR)해야 합니다.

슬라이드의 예제에서 PL/SQL은 형식 및 실제 파라미터의 숫자와 데이터 유형을 일치시킴으로써 TO_CHAR에 대한 호출을 분석합니다.

잘못된 프로시저 참조

- **블록 구조 언어(예: PL/SQL)는 식별자를 참조하기 전에 선언해야 합니다.**
- **참조 관련 문제 예제:**

```

CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE award_bonus(. . .) IS
    BEGIN
      calc_rating(. . .);      --illegal reference
    END;

  PROCEDURE calc_rating(. . .) IS
    BEGIN
      ...
    END;
END forward_pkg;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

사전 선언 사용

일반적으로 PL/SQL은 다른 블록 구조 언어와 유사하므로 사전 참조가 허용되지 않습니다. 식별자를 사용하려면 먼저 선언해야 합니다. 예를 들어, 서브 프로그램을 호출하려면 먼저 선언해야 합니다.

코딩 표준에 따르면 대개 서브 프로그램은 문자순으로 보관해야 쉽게 찾을 수 있습니다. 이 경우 위 슬라이드에 표시된 것과 같이 문제가 발생할 수 있습니다. 여기서 calc_rating 프로시저는 아직 선언되지 않았기 때문에 참조될 수 없습니다.

두 프로시저의 순서를 바꾸면 잘못된 참조 문제를 해결할 수 있습니다. 그러나 코딩 규칙이 서브 프로그램을 문자순으로 선언할 것을 요구하는 경우 이와 같이 쉬운 방법으로는 문제가 해결되지 않습니다.

이 경우의 해결 방법은 PL/SQL에서 제공하는 사전 선언을 사용하는 것입니다. 사전 선언을 사용하면 서브 프로그램 머리글을 선언할 수 있습니다. 즉, 서브 프로그램 사양이 세미콜론으로 끝납니다.

참고: calc_rating이 전용(private) 패키지 프로시저일 경우에만 calc_rating에 대해 컴파일 오류가 발생합니다. Package Spec에서 calc_rating을 선언할 경우 사전 선언처럼 이미 선언되어 있으므로 컴파일러가 참조를 분석할 수 있습니다.

사전 선언을 사용하여 잘못된 프로시저 참조 해결

Package Body에서 사전 선언은 세미콜론으로 끝나는 전용(private) 서브 프로그램 사양입니다.

```

CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE calc_rating (...) ; -- forward declaration
  -- Subprograms defined in alphabetical order

  PROCEDURE award_bonus (...) IS
    BEGIN
      calc_rating (...);           -- reference resolved!
      . . .
    END;

  PROCEDURE calc_rating (...) IS -- implementation
    BEGIN
      . . .
    END;
END forward_pkg;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

사전 선언 사용(계속)

앞에서 언급한 바와 같이, PL/SQL을 사용하면 사전 선언(Forward Declaration)^o라는 특별한 서브 프로그램 선언을 생성할 수 있습니다. 사전 선언은 Package Body의 전용(private) 서브 프로그램에 필요하며, 세미콜론으로 끝나는 서브 프로그램 사양으로 구성됩니다. 사전 선언은 다음 작업을 수행할 때 유용합니다.

- 문자순 또는 논리순으로 서브 프로그램 정의.
- 상호 Recursive 서브 프로그램 정의. 상호 Recursive 프로그램이란 서로를 직접 또는 간접적으로 호출하는 프로그램입니다.
- Package Body에서 서브 프로그램 그룹화 및 논리적으로 구성.

사전 선언을 생성할 때는 다음 사항에 유의하십시오.

- 형식 파라미터는 사전 선언과 서브 프로그램 본문 모두에 나타나야 합니다.
- 서브 프로그램 본문은 사전 선언 뒤라면 어디에든지 나타날 수 있지만 사전 선언과 서브 프로그램 본문 모두 동일한 프로그램 단위(예: 같은 패키지)에 나타나야 합니다.

사전 선언 및 패키지

일반적으로 서브 프로그램 사양은 Package Spec에 속하고, 서브 프로그램 본문은 Package Body에 속합니다. Package Spec에 있는 공용(public) 서브 프로그램 선언의 경우 사전 선언이 필요하지 않습니다.

패키지 초기화

**Package Body 끝에 있는 블록은 한 번 실행되며 공용
(public)/전용(private) 패키지 변수를 초기화하는 데 사용됩니다.**

```

CREATE OR REPLACE PACKAGE taxes IS
    v_tax    NUMBER;
    ... -- declare all public procedures/functions
END taxes;
/
CREATE OR REPLACE PACKAGE BODY taxes IS
    ... -- declare all private variables
    ... -- define public/private procedures/functions
BEGIN
    SELECT    rate_value INTO v_tax
    FROM      tax_rates
    WHERE     rate_name = 'TAX';
END taxes;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 초기화 블록

패키지 구성 요소가 처음으로 참조될 때 전체 패키지가 유저 세션 동안 메모리에 로드됩니다. 기본적으로 변수 초기값은 NULL입니다(명시적으로 초기화되지 않은 경우). 패키지 변수를 초기화하기 위해 다음을 수행할 수 있습니다.

- 간단한 초기화 작업의 경우 선언에 할당 연산 사용
- 보다 복잡한 초기화 작업의 경우 Package Body 끝에 코드 블록 추가

Package Body 끝에 있는 코드 블록은 유저 세션 내에서 패키지가 처음 호출될 때 한 번만 실행되는 패키지 초기화 블록으로 간주하십시오.

슬라이드의 예제에서는 taxes 패키지가 처음으로 참조될 때 공용(public) 변수 v_tax가 tax_rates 테이블에 있는 값으로 초기화되는 것을 보여줍니다.

참고: 할당 연산을 사용하여 선언에서 변수를 초기화할 경우 Package Body 끝에 있는 초기화 블록 코드가 이 변수를 덮쳐씁니다. Package Body에 대한 초기화 블록은 END 키워드로 끝납니다.

SQL에서 패키지 함수 사용

- SQL 문에서 패키지 함수를 사용합니다.
- 멤버 함수를 호출하는 SQL 문을 실행하려면 오라클 데이터베이스가 함수의 Purity 레벨을 알고 있어야 합니다.
- Purity 레벨은 함수가 부작용으로부터 얼마나 안전한지를 나타냅니다. 부작용이란 읽거나 쓰기 위해 데이터베이스 테이블, 패키지 변수 등에 액세스하는 것을 의미합니다.
- 부작용은 다음과 같은 영향을 주기 때문에 반드시 제어해야 합니다.
 - Query의 적절한 병렬화를 방해합니다.
 - 순서에 따라 다른 불명확한 결과가 야기됩니다.
 - 유저 세션 간의 패키지 상태 유지 관리와 같이 허용되지 않는 작업을 필요로 합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL의 패키지 함수 사용 및 제한 사항

내장 함수를 호출하는 SQL 문을 실행하려면 Oracle 서버에서 함수의 Purity 레벨을 알거나 함수가 부작용으로부터 얼마나 안전한지를 알고 있어야 합니다. 부작용이라는 용어는 읽거나 쓰기 위해 데이터베이스 테이블, 패키지 변수 등에 액세스하는 것을 의미합니다. 부작용은 Query의 적절한 병렬화를 방해하거나, 순서에 따라 다른 불명확한 결과를 야기하거나, 유저 세션 간의 패키지 상태 유지 관리와 같이 허용되지 않는 작업을 요구하기 때문에 반드시 제어해야 합니다.

일반적으로 제한 사항이란 데이터베이스 테이블 또는 공용(public) 패키지 변수(Package Spec에 선언된 변수)가 변경되는 것입니다. 제한 사항으로 인해 query 실행이 지연되거나, 순서에 따라 다른(불명확한) 결과가 야기되거나, 유저 세션 간에 패키지 상태 변수를 유지 관리해야 할 수 있습니다. 함수가 SQL query 또는 DML 문에서 호출될 때는 여러 제한 사항이 허용되지 않습니다.

PL/SQL 서브 프로그램의 부작용 제어

내장 함수가 다음 Purity 규칙을 따라 부작용을 제어할 경우에만 SQL 문에서 호출할 수 있습니다.

- **SELECT 문 또는 병렬화된 DML 문에서 호출될 경우 함수는 어떠한 데이터베이스 테이블도 수정할 수 없습니다.**
- **DML 문에서 호출될 경우 함수는 해당 명령문에 의해 수정된 어떠한 데이터베이스 테이블도 query하거나 수정할 수 없습니다.**
- **SELECT 또는 DML 문에서 호출될 경우 함수는 SQL 트랜잭션 제어, 세션 제어 또는 시스템 제어 명령문을 실행할 수 없습니다.**



Copyright © 2009, Oracle. All rights reserved.

PL/SQL 서브 프로그램의 부작용 제어

함수의 부작용이 적을수록 query 내에서의 최적화 효과가 높아질 수 있으며, 특히 PARALLEL_ENABLE 또는 DETERMINISTIC 힌트가 사용될 때는 더욱 그렇습니다.

내장 함수(그리고 내장 함수가 호출하는 모든 서브 프로그램)는 슬라이드에 나열된 Purity 규칙을 따르는 경우에만 SQL 문에서 호출될 수 있습니다. 해당 규칙의 목적은 부작용을 제어하는 것입니다.

함수 본문 내의 SQL 문이 규칙을 위반하는 경우에는 런타임(명령문이 구문 분석될 때)에 오류가 발생합니다.

컴파일 시 Purity 규칙 위반을 점검하려면 RESTRICT_REFERENCES 프래그마를 사용하여 함수가 데이터베이스 테이블 또는 패키지 변수를 읽거나 쓰지 않음을 검증하십시오.

참고

- 슬라이드에서 DML 문은 INSERT, UPDATE 또는 DELETE 문을 나타냅니다.
- RESTRICT_REFERENCES 프래그마 사용에 대한 자세한 내용은 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)*을 참조하십시오.

SQL의 패키지 함수: 예제

```

CREATE OR REPLACE PACKAGE taxes_pkg IS
  FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pkg;
/
CREATE OR REPLACE PACKAGE BODY taxes_pkg IS
  FUNCTION tax (p_value IN NUMBER) RETURN NUMBER IS
    v_rate NUMBER := 0.08;
  BEGIN
    RETURN (p_value * v_rate);
  END tax;
END taxes_pkg;
/

```

```

SELECT taxes_pkg.tax(salary), salary, last_name
FROM employees;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL의 패키지 함수: 예제

슬라이드 쇼에서 첫번째 코드 예제는 taxes_pkg 패키지에서 tax 함수를 캡슐화하는 Package Spec 및 Body를 생성하는 방법을 보여줍니다. 두번째 코드 예제는 SELECT 문에서 패키지화된 tax 함수를 호출하는 방법을 보여줍니다. 결과는 다음과 같습니다.

TAXES_PKG.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
384	4800	Austin
384	4800	Pataballa
336	4200	Lorentz
960	12000	Greenberg

...

107 rows selected

단원 내용

- 패키지 서브 프로그램 오버로드, 사전 선언 사용,
Package Body에서 초기화 블록 생성
- 세션 기간 동안 지속되는 패키지 데이터 상태 관리 및
패키지에서 연관 배열(인덱스화된 테이블)과 레코드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

패키지의 지속 상태

패키지 변수 및 값 모음은 패키지 상태를 정의합니다. 패키지 상태의 특징은 다음과 같습니다.

- **패키지가 처음으로 로드될 때 초기화됨**
- **세션 기간 동안 지속됨(기본값):**
 - UGA(User Global Area)에 저장됨
 - 세션마다 고유함
 - 패키지 서브 프로그램이 호출되거나 공용(public) 변수가 수정될 때 변경될 수 있음
- **Package Spec에서 PRAGMA SERIALLY_REUSABLE을 사용할 경우 세션 동안 지속되는 대신 서브 프로그램이 호출되는 동안 지속됨**



Copyright © 2009, Oracle. All rights reserved.

패키지의 지속 상태

공용(public) 및 전용(private) 패키지 변수 모음은 유저 세션 동안 패키지 상태를 나타냅니다. 즉, 패키지 상태는 특정 시점에 모든 패키지 변수에 저장된 값 집합입니다. 일반적으로 패키지 상태는 유저 세션 기간 동안 존재합니다.

패키지 변수는 유저 세션 동안 패키지가 처음으로 메모리에 로드될 때 초기화됩니다. 패키지 변수는 기본적으로 세션마다 고유하며 유저 세션이 종료될 때까지 변수 값을 보유합니다. 즉, 변수는 각 유저 세션 동안 데이터베이스에 의해 할당되는 UGA(User Global Area) 메모리에 저장됩니다. 패키지 상태는 패키지 서브 프로그램이 호출되어 서브 프로그램 논리가 변수 상태를 수정할 때 변경됩니다. 공용(public) 패키지 상태는 해당 유형에 적합한 작업에 의해 직접 수정될 수 있습니다.

PRAGMA는 명령문이 컴파일러 지시어임을 의미합니다. PRAGMA는 런타임에 처리되지 않고 컴파일 시 처리됩니다. 프로그램의 의미에는 영향을 주지 않으며 컴파일러에 정보를 전달할 뿐입니다. Package Spec에 PRAGMA SERIALLY_RESUABLE을 추가할 경우 데이터베이스는 유저 세션 간에 공유되는 SGA(System Global Area)에 패키지 변수를 저장합니다. 이 경우 패키지 상태는 서브 프로그램이 호출되는 동안이나 패키지 생성자를 한 번 참조하는 동안 유지 관리됩니다. SERIALLY_RESUABLE 지시어는 메모리를 절약하려는 경우와 각 유저 세션 동안 패키지 상태를 지속할 필요가 없을 경우에 유용합니다.

패키지의 지속 상태(계속)

이 PRAGMA는 한 번 사용되며 같은 세션의 이후 데이터베이스 호출 시 필요하지 않은 큰 임시 작업 영역을 선언하는 패키지에 적합합니다.

본문 없는 패키지를 직렬로 재사용할 수 있도록 표시할 수 있습니다. 패키지에 Spec과 Body가 있으면 둘 다 표시해야 하며 Body만 표시할 수는 없습니다.

직렬로 재사용할 수 있는 패키지의 Global 메모리는 SGA(System Global Area)에 풀링되며 UGA(User Global Area)의 개별 유저에게 할당되지 않습니다. 즉, 패키지 작업 영역은 재사용할 수 있습니다. 서버에 대한 호출이 끝나면 메모리가 풀로 돌아갑니다. 패키지가 재사용될 때마다 공용(public) 변수는 기본값 또는 NULL로 초기화됩니다.

참고: 직렬로 재사용할 수 있는 패키지는 데이터베이스 트리거 또는 SQL 문에서 호출되는 기타 PL/SQL 서브 프로그램으로부터 액세스할 수 없습니다. 액세스를 시도하면 Oracle 서버에서 오류를 생성합니다.

패키지 변수의 지속 상태: 예제

시간	이벤트	v_std_comm [variable]	MAX (commission_pct) [column]	v_std_comm [variable]	MAX (commission_pct) [Column]
9:00	Scott> EXECUTE comm_pkg.reset_comm(0.25)	0.10 0.25	0.4	-	0.4
9:30	Jones> INSERT INTO employees(last_name, commission_pct) VALUES('Madonna', 0.8);	0.25	0.4		0.8
9:35	Jones> EXECUTE comm_pkg.reset_comm (0.5)	0.25	0.4	0.10 0.5	0.8
10:00	Scott> EXECUTE comm_pkg.reset_comm (0.6) Err -20210 'Bad Commission'	0.25	0.4	0.5	0.8
11:00 11:01 12:00	Jones> ROLLBACK; EXIT ... EXEC comm_pkg.reset_comm(0.2)	0.25 0.25 0.25	0.4 0.4 0.4	0.5 - 0.2	0.4 0.4 0.4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 변수의 지속 상태: 예제

위의 슬라이드 시퀀스는 comm_pkg("패키지 생성" 단원에서 다룸)를 실행하는 Scott과 Jones라는 두 명의 유저를 기반으로 하는데, 여기서 reset_comm 프로시저는 validate 함수를 호출하여 새 커미션을 검사합니다. 예제는 v_std_comm 패키지 변수의 지속 상태가 각 유저 세션에서 유지 관리되는 방식을 보여줍니다.

9시: Scott이 새 커미션 값 0.25로 reset_comm을 호출합니다. v_std_comm의 패키지의 상태가 0.10으로 초기화된 다음 0.25로 설정되는데, 이 값은 데이터베이스 최대값 0.4보다 작기 때문에 유효합니다.

9시 30분: Jones가 EMPLOYEES 테이블에 v_commission_pct의 새 최대값 0.8을 포함하는 새 행을 추가합니다. 이것은 커밋되지 않았으므로 Jones만 볼 수 있습니다. Scott의 상태는 영향을 받지 않습니다.

9시 35분: Jones가 새 커미션 값 0.5로 reset_comm을 호출합니다. Jones의 v_std_comm 상태가 0.10으로 초기화된 다음 새 값 0.5로 설정되는데, 이 값은 데이터베이스 최대값이 0.8인 Jones의 세션에 대해 유효합니다.

10시: Scott이 새 커미션 값 0.6으로 reset_comm을 호출하는데, 이 값은 Scott의 세션에 표시되는 최대 데이터베이스 커미션(0.4)보다 큽니다. (Jones는 값 0.8을 커밋하지 않았습니다.)

11시와 12시 사이: Jones가 트랜잭션을 롤백하고(INSERT 문) 세션을 종료합니다. Jones가 11시 45분에 로그인하여 자신의 상태를 0.2로 설정하는 프로시저를 성공적으로 실행합니다.

패키지 커서의 지속 상태: 예제

```
CREATE OR REPLACE PACKAGE curs_pkg IS -- Package spec
  PROCEDURE open;
  FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close;
END curs_pkg;

CREATE OR REPLACE PACKAGE BODY curs_pkg IS
-- Package body
  CURSOR cur_c IS
    SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT cur_c%ISOPEN THEN
      OPEN cur_c;
    END IF;
  END open;
  . . . -- code continued on next slide
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 커서의 지속 상태: 예제

슬라이드의 예제는 CURS_PKG Package Spec 및 Body를 보여줍니다. Body 선언은 다음 슬라이드에서 계속됩니다.

이 패키지를 사용하려면 다음 단계를 수행하여 행을 처리하십시오.

1. 커서를 여는 open 프로시저를 호출합니다.

패키지 커서의 지속 상태: 예제

```

. . .
FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN IS
    v_emp_id employees.employee_id%TYPE;
BEGIN
    FOR count IN 1 .. p_n LOOP
        FETCH cur_c INTO v_emp_id;
        EXIT WHEN cur_c%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Id: ' || (v_emp_id));
    END LOOP;
    RETURN cur_c%FOUND;
END next;
PROCEDURE close IS
BEGIN
    IF cur_c%ISOPEN THEN
        CLOSE cur_c;
    END IF;
END close;
END curs_pkg;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지 커서의 지속 상태: 예제(계속)

2. 한 개의 행이나 지정된 개수의 행을 페치(fetch)하는 next 프로시저를 호출합니다. 실제 존재하는 것보다 많은 행을 요청할 경우 프로시저는 성공적으로 종료됩니다. 더 많은 행을 처리해야 할 경우에는 TRUE가 반환되고, 그렇지 않은 경우에는 FALSE가 반환됩니다.
3. 행 처리 이전이나 끝에서 커서를 닫는 close 프로시저를 호출합니다.

참고: 커서 선언은 패키지 전용(private)입니다. 따라서 슬라이드에 나열된 패키지 프로시저와 함수를 호출하면 cursor state가 영향을 받을 수 있습니다.

CURS_PKG 패키지 실행

```

1 SET SERVEROUTPUT ON
2
3 EXECUTE curs_pkg.open
4 DECLARE
5   v_more BOOLEAN := curs_pkg.next(3);
6 BEGIN
7   IF NOT v_more THEN
8     curs_pkg.close;
9   END IF;
10 END;
11 /

```

anonymous block completed
anonymous block completed
Id: 100
Id: 101
Id: 102

anonymous block completed
anonymous block completed
Id: 103
Id: 104
Id: 105

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CURS_PKG 실행

패키지 변수나 커서의 상태는 세션 내의 전체 트랜잭션에서 지속됩니다. 그러나 동일한 유저의 다른 세션에서는 상태가 지속되지 않습니다. 데이터베이스 테이블은 세션과 유저 전체에서 지속되는 데이터를 보유합니다. curs_pkg.open을 호출하면 커서가 열리는데, 이 커서는 세션이 종료되거나 커서가 명시적으로 닫힐 때까지 열려 있습니다. 익명 블록은 선언 셋션에서 next 함수를 실행하는데 EMPLOYEES 테이블의 행이 세 개를 초과하므로 BOOLEAN 변수 b_more가 TRUE로 초기화됩니다. 블록은 결과 집합의 끝 부분이 적합한지 검사한 다음 적합하면 커서를 닫습니다. 블록이 실행되면 처음 세 개의 행이 표시됩니다.

```

Id :100
Id :101
Id :102

```

Run Script(F5) 아이콘을 다시 누르면 다음 세 개의 행이 표시됩니다.

```

Id :103
Id :104
Id :105

```

커서를 닫으려면 다음 명령을 실행하여 패키지에서 커서를 닫거나 세션을 종료하면 됩니다.

```
EXECUTE curs_pkg.close
```

패키지에서 연관 배열 사용

```
CREATE OR REPLACE PACKAGE emp_pkg IS
    TYPE emp_table_type IS TABLE OF employees%ROWTYPE
        INDEX BY BINARY_INTEGER;
    PROCEDURE get_employees(p_emps OUT emp_table_type);
END emp_pkg;
```

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    PROCEDURE get_employees(p_emps OUT emp_table_type) IS
        v_i BINARY_INTEGER := 0;
    BEGIN
        FOR emp_record IN (SELECT * FROM employees)
        LOOP
            emps(v_i) := emp_record;
            v_i:= v_i + 1;
        END LOOP;
    END get_employees;
END emp_pkg;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

패키지에서 연관 배열 사용

인덱스화된 테이블이라고 알리는 데 사용되는 연관 배열

emp_pkg 패키지는 EMPLOYEES 테이블에서 행을 읽어 연관 배열(PL/SQL 레코드 테이블)인 OUT 파라미터를 사용하는 행을 반환하는 get_employees 프로시저를 포함합니다. 주요 사항은 다음과 같습니다.

- employee_table_type은 공용(public) 유형으로 선언됩니다.
- employee_table_type은 프로시저에서 형식 출력 파라미터에 사용되고 호출 블록에서 employees 변수에 사용됩니다(아래 참조).

다음 예제 및 출력과 같이 SQL Developer에서는 v_employees 변수를 사용하여 익명 PL/SQL 블록에서 get_employees 프로시저를 호출할 수 있습니다.

```
SET SERVEROUTPUT ON
DECLARE
    v_employees    emp_pkg.emp_table_type;
BEGIN
    emp_pkg.get_employees(v_employees);
    DBMS_OUTPUT.PUT_LINE('Emp 5: ' || v_employees(4).last_name);
END;
anonymous block completed
Emp 5: Ernst
```

퀴즈

PL/SQL에서 서브 프로그램 오버로드:

- 1. 동일한 이름의 서브 프로그램을 두 개 이상 생성할 수 있습니다.**
- 2. 서브 프로그램에서 사용하는 형식 파라미터의 개수, 순서 또는 데이터 유형 계열이 달라야 합니다.**
- 3. 서로 다른 데이터를 처리하는 서브 프로그램을 여러 가지 방법으로 호출할 수 있습니다.**
- 4. 기존 코드의 손실 없이 기능을 확장하는 방법을 제공합니다.
즉, 기존 서브 프로그램에 새 파라미터를 추가합니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 4

PL/SQL에서 오버로드 기능을 사용하면 동일한 이름의 패키지화된 서브 프로그램을 두 개 이상 개발할 수 있습니다. 오버로드는 서브 프로그램에 데이터 유형이 다른 유사한 파라미터 집합을 사용하고자 할 때 유용합니다. 예를 들면, 숫자나 날짜를 문자열로 변환하도록 하는 TO_CHAR 함수를 여러 가지 방법으로 호출할 수 있습니다.

PL/SQL에서는 패키지 서브 프로그램 이름과 객체 유형 메소드를 오버로드할 수 있습니다.

여기에서 중요한 규칙은 형식 파라미터의 개수, 순서 또는 데이터 유형 계열이 다를 경우 동일한 이름을 여러 서브 프로그램에 사용할 수 있다는 점입니다.

다음과 같은 경우에 오버로드를 사용하십시오.

- 둘 이상의 서브 프로그램에 대해 처리 규칙은 유사하지만 사용되는 파라미터의 개수나 유형이 서로 다를 경우.
- 검색 조건을 다르게 하여 다른 데이터를 찾는 대체 방법을 제공하려는 경우. 예를 들면, 사원 ID로 사원을 찾을 수도 있고 성으로 사원을 찾을 수도 있습니다. 논리는 본질적으로 같지만 파라미터나 검색 조건이 다릅니다.
- 기존 코드를 바꾸고 싶지 않을 때 기능을 확장하려는 경우.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 패키지 프로시저 및 함수 오버로드
- 사전 선언 사용
- Package Body에 초기화 블록 생성
- 세션 기간 동안 지속되는 패키지 데이터 상태 관리
- 패키지에서 연관 배열(인덱스화된 테이블) 및 레코드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

오버로드 기능을 사용하면 여러 서브 프로그램을 동일한 이름으로 정의할 수 있습니다. 두 서브 프로그램의 프로세스는 동일하지만 전달되는 파라미터가 다를 경우 두 서브 프로그램에 동일한 이름을 지정하십시오.

PL/SQL에서는 사전 선언이라는 특별한 서브 프로그램 선언이 허용됩니다. 사전 선언을 사용하면 서브 프로그램을 논리순 또는 문자순으로 정의하거나, 상호 Recursive 서브 프로그램을 정의하거나, 패키지에서 서브 프로그램을 그룹화할 수 있습니다.

패키지 초기화 블록은 다른 유저 세션 내에서 처음으로 호출될 때 한 번만 실행됩니다. 이 기능을 사용하여 세션당 한 번만 변수를 초기화할 수 있습니다.

유저가 변수나 커서를 처음으로 참조할 때부터 유저 연결이 해제될 때까지 유저 세션 동안 지속되는 패키지 변수나 커서의 상태를 추적할 수 있습니다.

PL/SQL 래퍼를 사용하면 데이터베이스에 저장된 소스 코드를 숨겨 지적 소유권을 보호할 수 있습니다.

연습 4: 개요

이 연습에서는 다음 내용을 다룹니다.

- 오버로드된 서브 프로그램 사용
- 패키지 초기화 블록 생성
- 사전 선언 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 4: 개요

이 연습에서는 오버로드된 서브 프로그램을 포함하도록 기존 패키지를 수정하고 사전 선언을 사용합니다. 또한 Package Body 내에 패키지 초기화 블록을 생성하여 PL/SQL 테이블을 채웁니다.

5 응용 프로그램 개발 시 오라클 제공 패키지 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- DBMS_OUTPUT 패키지 작동 방식 설명
- UTL_FILE을 사용하여 출력을 운영 체제 파일로 지정
- UTL_MAIL의 주요 기능 설명

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 몇 가지 오라클 제공 패키지와 이들의 기능을 사용하는 방법에 대해 설명합니다.

단원 내용

- 오라클 제공 패키지 사용 시 이점 파악 및 해당 패키지 일부 나열
- 다음과 같은 오라클 제공 패키지 사용:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

오라클 제공 패키지 사용

- **오라클 제공 패키지:**
 - Oracle 서버에서 제공합니다.
 - 데이터베이스 기능을 확장합니다.
 - 일반적으로 PL/SQL에서는 제한되는 특정 SQL 기능에 액세스할 수 있습니다.
- 예를 들어 DBMS_OUTPUT 패키지는 원래 PL/SQL 프로그램을 디버그하도록 설계되었습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

오라클 제공 패키지 사용

Oracle 서버에서 제공하는 패키지를 사용하면 다음 중 하나를 수행할 수 있습니다.

- 특정 SQL 기능에 대한 PL/SQL 액세스
- 데이터베이스 기능 확장

응용 프로그램을 생성할 때 이러한 패키지에서 제공하는 기능을 사용할 수도 있고, 자체 내장 프로시저를 작성할 때 이러한 패키지를 단지 아이디어로만 활용할 수도 있습니다.

대부분의 표준 패키지는 `catproc.sql`을 실행하여 생성됩니다. DBMS_OUTPUT 패키지는 본 과정에서 가장 많이 다룰 패키지 중 하나입니다. *Oracle Database 11g: PL/SQL Fundamentals* 과정을 수강했다면 이 패키지에 대해 잘 알고 있어야 합니다.

몇 가지 오라클 제공 패키지 예제

다음은 몇 가지 오라클 제공 패키지의 요약 리스트입니다.

- DBMS_OUTPUT
- UTL_FILE
- UTL_MAIL
- DBMS_ALERT
- DBMS_LOCK
- DBMS_SESSION
- DBMS_APPLICATION_INFO
- HTP
- DBMS_SCHEDULER

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

몇 가지 오라클 제공 패키지 리스트

새 버전의 오라클 데이터베이스가 계속 출시되면서 제공되는 PL/SQL 패키지 리스트도 늘어나고 있습니다. 이 단원에서는 슬라이드에 있는 처음 세 개의 패키지를 다룹니다. 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)*를 참조하십시오.

다음은 슬라이드에 나열된 나머지 패키지에 대한 간단한 설명입니다.

- DBMS_OUTPUT은 텍스트 데이터의 디버깅과 버퍼링을 제공합니다.
- UTL_FILE을 사용하면 운영 체제 텍스트 파일을 읽고 쓸 수 있습니다.
- UTL_MAIL을 사용하면 전자 메일 메시지를 작성하고 보낼 수 있습니다.
- DBMS_ALERT는 데이터베이스 이벤트의 비동기 통지를 지원합니다.
메시지나 경고는 COMMIT 명령으로 전송됩니다.
- DBMS_LOCK은 Oracle Lock Management 서비스를 통해 잠금을 요청, 변환 및 해제하는 데 사용됩니다.
- DBMS_SESSION을 사용하면 ALTER SESSION SQL 문과 기타 세션 레벨 명령을 프로그래밍 방식으로 사용할 수 있습니다.

몇 가지 오라클 제공 패키지 리스트(계속)

- Oracle Trace 및 SQL Trace 기능에서 DBMS_APPLICATION_INFO를 사용하여 데이터베이스에서 실행 중인 모듈 또는 트랜잭션의 이름을 기록할 수 있습니다. 나중에 다양한 모듈 및 디버깅의 성능을 추적하는 데 이 기록을 사용할 수 있습니다.
- HTP 패키지는 데이터베이스 버퍼에 HTML 태그 데이터를 작성합니다.
- DBMS_SCHEDULER를 사용하면 PL/SQL 블록, 내장 프로시저, external 프로시저 및 실행 파일을 스케줄링(scheduling)하고 자동으로 실행할 수 있습니다(부록 G에서 다룸).

단원 내용

- 오라클 제공 패키지 사용 시 이점 파악 및 해당 패키지 일부 나열
- 다음과 같은 오라클 제공 패키지 사용:
 - DBMS_OUTPUT
 - UTL_FILE
 - UTL_MAIL

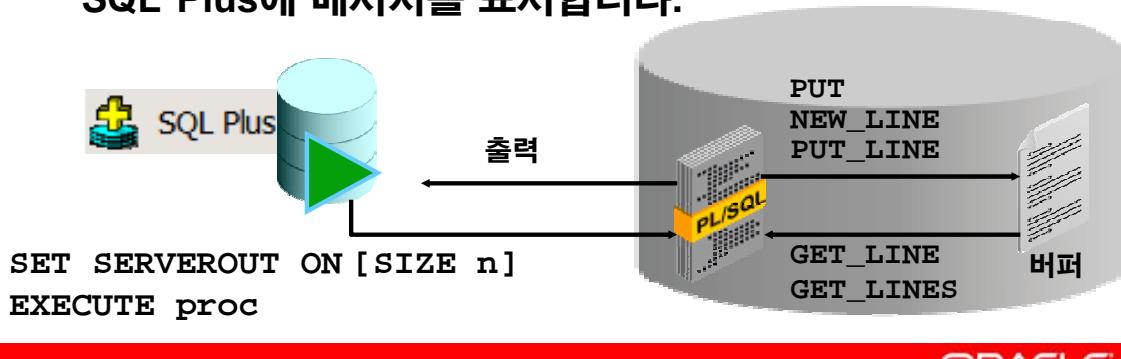
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DBMS_OUTPUT 패키지 작동 방식

DBMS_OUTPUT 패키지를 사용하면 내장 서브 프로그램 및 트리거에서 메시지를 보낼 수 있습니다.

- PUT 및 PUT_LINE은 텍스트를 버퍼에 삽입합니다.
- GET_LINE 및 GET_LINES는 버퍼를 읽습니다.
- 보내는 서브 프로그램 또는 트리거가 완료되기 전에는 메시지를 보내지 않습니다.
- SET SERVEROUTPUT ON을 사용하여 SQL Developer 및 SQL*Plus에 메시지를 표시합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_OUTPUT 패키지 작동 방식

DBMS_OUTPUT 패키지는 텍스트 메시지를 임의의 PL/SQL 블록에서 데이터베이스 버퍼로 보냅니다. 패키지에서 제공하는 프로시저는 다음과 같습니다.

- PUT - 프로시저의 텍스트를 행 출력 버퍼의 현재 행에 추가합니다.
- NEW_LINE - 행 끝 표시자를 출력 버퍼에 삽입합니다.
- PUT_LINE - PUT 및 NEW_LINE의 작업을 결합합니다(선행 공백을 자름).
- GET_LINE - 버퍼의 현재 행을 프로시저 변수로 읽어 들입니다.
- GET_LINES - 행 배열을 프로시저 배열 변수로 읽어 들입니다.
- ENABLE/DISABLE - DBMS_OUTPUT 프로시저에 대한 호출을 활성화하거나 비활성화합니다.

버퍼 크기는 다음을 사용하여 설정할 수 있습니다.

- SET SERVEROUTPUT ON 명령에 추가된 SIZE n 옵션. 여기서 n은 문자 수입니다. 최소값은 2,000이고 최대값은 무제한입니다. 기본값은 20,000입니다.
- ENABLE 프로시저에 있는 2,000 - 1,000,000 범위의 정수 파라미터

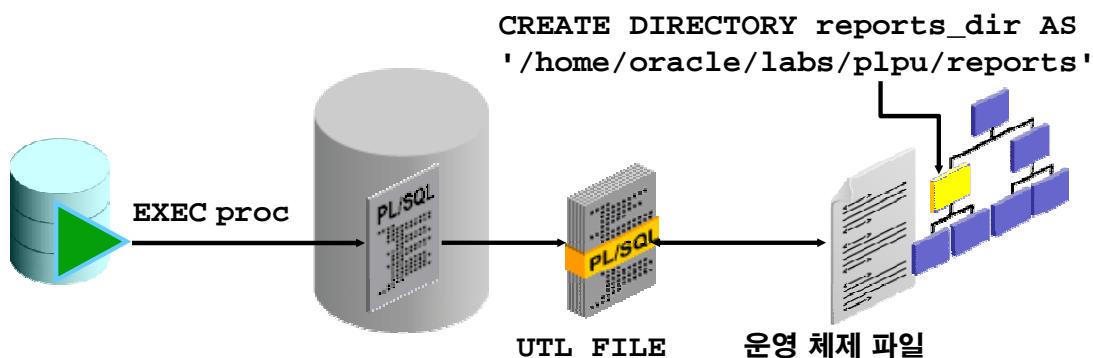
디버깅을 위해 결과를 window에 출력할 수 있습니다. 함수나 프로시저의 코드 실행 경로를 trace할 수 있습니다. 서브 프로그램과 트리거 간에 메시지를 보낼 수 있습니다.

참고: 프로시저가 실행되는 동안 출력 내용을 버퍼에서 비울 수 있는 메커니즘은 없습니다.

UTL_FILE 패키지를 사용하여 운영 체제 파일과 상호 작용

UTL_FILE 패키지는 운영 체제 텍스트 파일을 읽고 쓸 수 있도록 PL/SQL 프로그램 기능을 제공합니다.

- 텍스트 파일에 대해 제한된 버전의 운영 체제 스트림 파일 I/O를 제공합니다.
- CREATE DIRECTORY 문에 의해 정의된 운영 체제 디렉토리의 파일에 액세스할 수 있습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

운영 체제 파일과 상호 작용

오라클에서 제공하는 UTL_FILE 패키지는 데이터베이스 서버 운영 체제에 있는 텍스트 파일에 액세스하는 데 사용됩니다. 데이터베이스는 다음을 사용하여 특정 운영 체제 디렉토리에 대한 읽기/쓰기 액세스 권한을 제공합니다.

- alias를 운영 체제 디렉토리와 연관시키는 CREATE DIRECTORY 문. 데이터베이스 디렉토리 alias에 READ 및 WRITE 권한을 부여하여 운영 체제 파일에 대한 액세스 유형을 제어 할 수 있습니다. 예를 들면 다음과 같습니다.


```
CREATE DIRECTORY my_dir AS '/temp/my_files';
GRANT READ, WRITE ON DIRECTORY my_dir TO public.
```
- 경로가 utl_file_dir 데이터베이스 초기화 파라미터에 지정되었습니다.

UTL_FILE_DIR 대신 CREATE DIRECTORY 기능을 사용하여 디렉토리 액세스를 검증할 것을 권장합니다. 디렉토리 객체는 UTL_FILE 응용 프로그램 관리자에게 보다 유연하고 세밀한 제어 능력을 제공할 뿐 아니라 데이터베이스를 종료하지 않고 동적으로 관리할 수 있으며 다른 오라클 도구에서도 일관되게 사용할 수 있습니다. CREATE DIRECTORY 권한은 기본적으로 SYS 및 SYSTEM에만 부여됩니다.

이러한 방법 중 하나를 사용하여 지정된 운영 체제 디렉토리는 데이터베이스 서버 프로세스와 동일한 시스템에(서) 액세스할 수 있어야 합니다. 일부 운영 체제는 경로(디렉토리) 이름에서 대소문자를 구분할 수도 있습니다.

몇 가지 UTL_FILE 프로시저 및 함수

서브 프로그램	설명
ISOPEN 함수	파일 핸들이 열려 있는 파일을 참조하는지 여부를 결정합니다.
FOPEN 함수	입력 또는 출력을 위한 파일을 엽니다.
FCLOSE 함수	열려 있는 모든 파일 핸들을 닫습니다.
FCOPY 프로시저	연속하는 파일 부분을 새로 생성된 파일에 복사합니다.
FGETATTR 프로시저	디스크 파일의 속성을 읽고 반환합니다.
GET_LINE 프로시저	열려 있는 파일에서 텍스트를 읽습니다.
FREMOVE 프로시저	디스크 파일을 삭제합니다(충분한 권한이 있는 경우).
FRENAME 프로시저	기존 파일의 이름을 새 이름으로 바꿉니다.
PUT 프로시저	파일에 문자열을 씁니다.
PUT_LINE 프로시저	파일에 행을 기록하여 운영 체제별 행 종료자를 추가합니다.

ORACLE®

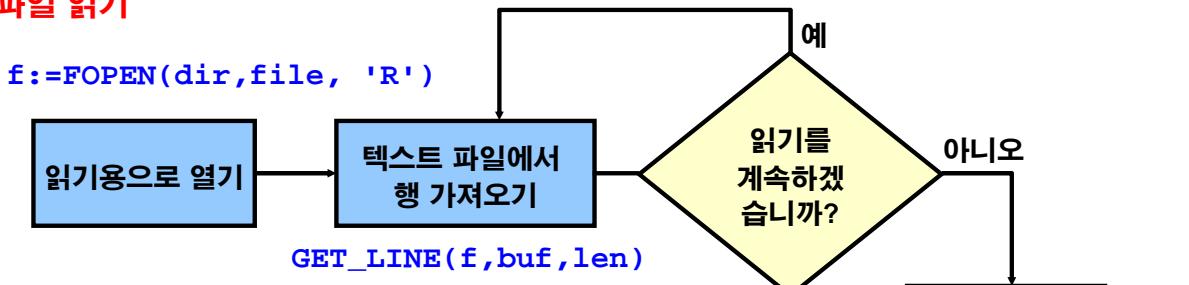
Copyright © 2009, Oracle. All rights reserved.

몇 가지 UTL_FILE 프로시저 및 함수

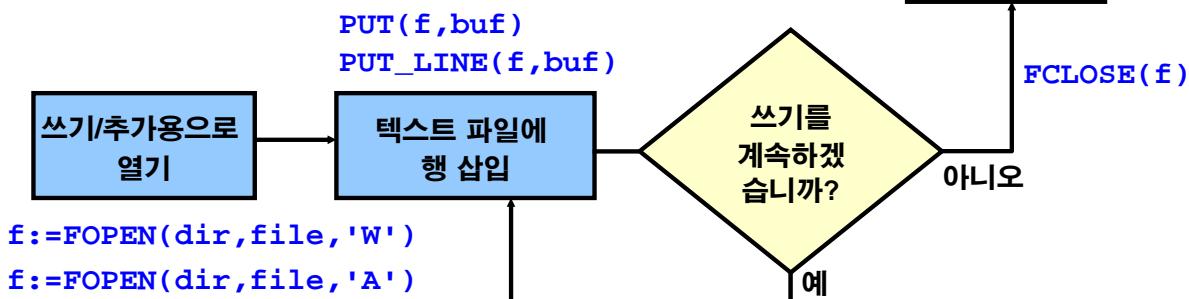
슬라이드의 표에는 몇 가지 UTL_FILE 패키지 서브 프로그램이 나열되어 있습니다. 패키지 서브 프로그램의 전체 리스트는 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

UTL_FILE 패키지를 사용한 파일 처리: 개요

파일 읽기



파일에 쓰기 또는 추가



ORACLE

Copyright © 2009, Oracle. All rights reserved.

UTL_FILE 패키지를 사용하여 파일 처리

UTL_FILE 패키지에 있는 프로시저 및 함수를 사용하여 FOPEN 함수로 파일을 열 수 있습니다. 그런 다음 파일 처리가 완료될 때까지 파일에서 읽어 들이거나, 파일에 쓰거나 추가할 수 있습니다. 파일 처리를 완료한 후 FCLOSE 프로시저를 사용하여 파일을 닫으십시오. 서브 프로그램은 다음과 같습니다.

- FOPEN 함수는 I/O(입/출력)를 위해 지정된 디렉토리에서 파일을 열고 후속 I/O 작업에 사용되는 파일 핸들을 반환합니다.
- IS_OPEN 함수는 파일 핸들이 열려 있는 파일을 참조할 때마다 부울 값을 반환합니다. IS_OPEN을 사용하여 파일을 열기 전에 해당 파일이 이미 열려 있는지 여부를 확인하십시오.
- GET_LINE 프로시저는 파일의 텍스트 행을 출력 버퍼 파라미터로 읽어 들입니다. FOPEN의 오버로드 버전에서 더 큰 크기를 지정하지 않는 한 입력 레코드의 최대 크기는 1,023바이트입니다.
- PUT 및 PUT_LINE 프로시저는 열려 있는 파일에 텍스트를 씁니다.
- PUTF 프로시저는 두 서식 지정자, 즉 출력 문자열에 나타나는 값을 치환하는 %s와 줄바꿈 문자 \n을 사용하여 출력에 서식을 지정합니다.
- NEW_LINE 프로시저는 출력 파일에서 행을 종료합니다.

UTL_FILE 패키지를 사용하여 파일 처리(계속)

- FFLUSH 프로시저는 메모리에 버퍼된 모든 데이터를 파일에 씁니다.
- FCLOSE 프로시저는 열려 있는 파일을 닫습니다.
- FCLOSE_ALL 프로시저는 세션에 대해 열려 있는 파일 핸들을 모두 닫습니다.

UTL_FILE 패키지에서 사용 가능한 선언된 예외 사용

예외 이름	설명
INVALID_PATH	잘못된 파일 위치
INVALID_MODE	FOPEN에서 open_mode 파라미터가 잘못됨
INVALID_FILEHANDLE	잘못된 파일 핸들
INVALID_OPERATION	파일을 열 수 없거나 요청대로 작동할 수 없음
READ_ERROR	읽기 작업 도중 운영 체제 오류 발생
WRITE_ERROR	쓰기 작업 도중 운영 체제 오류 발생
INTERNAL_ERROR	지정되지 않은 PL/SQL 오류

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

UTL_FILE 패키지의 예외

UTL_FILE 패키지는 운영 체제 파일 처리에서 오류 조건을 나타내는 15가지 예외를 선언합니다. UTL_FILE 서브 프로그램을 사용할 때는 이러한 예외 중 하나를 처리해야 할 수 있습니다.

슬라이드에는 예외의 하위 집합이 표시되어 있습니다. 나머지 예외에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

참고: 이러한 예외 앞에는 항상 패키지 이름이 접두어로 붙습니다. UTL_FILE 프로시저는 NO_DATA_FOUND 또는 VALUE_ERROR와 같은 미리 정의된 PL/SQL 예외를 발생시킬 수도 있습니다.

NO_DATA_FOUND 예외는 UTL_FILE.GET_LINE 또는 UTL_FILE.GET_LINES를 사용하여 파일 끝을 지나서 읽을 때 발생합니다.

FOPEN 및 IS_OPEN 함수: 예제

- ① FOPEN 함수는 입력 또는 출력을 위한 파일을 엽니다.

```
FUNCTION FOPEN (p_location  IN VARCHAR2,
               p_filename   IN VARCHAR2,
               p_open_mode  IN VARCHAR2)
RETURN UTL_FILE.FILE_TYPE;
```

- IS_OPEN 함수는 파일 핸들이 열려 있는 파일을 참조하는지 여부를 결정합니다.

```
FUNCTION IS_OPEN (p_file IN FILE_TYPE)
RETURN BOOLEAN;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FOPEN 및 IS_OPEN 함수 파라미터: 예제

파라미터는 다음과 같습니다.

- p_location 파라미터: CREATE DIRECTORY 문에 의해 정의되는 디렉토리 alias의 이름을 지정하거나, utl_file_dir 데이터베이스 파라미터를 사용하여 지정되는 운영 체제별 경로를 지정합니다.
- p_filename 파라미터: 확장자를 포함한 파일 이름을 경로 정보 없이 지정합니다.
- open_mode 문자열: 파일을 여는 방식을 지정합니다. 다음과 같은 값을 지정할 수 있습니다.
 'R': 텍스트 읽기(GET_LINE 사용)
 'W': 텍스트 쓰기(PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)
 'A': 텍스트 추가(PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)

FOPEN에서 반환되는 값은 유형이 UTL_FILE.FILE_TYPE인 파일 핸들입니다.

핸들은 열려 있는 파일에서 작동하는 루틴에 대한 후속 호출에 사용해야 합니다.

IS_OPEN 함수 파라미터는 파일 핸들입니다. IS_OPEN 함수는 파일 핸들이 열려 있는 파일을 식별하는지 여부를 테스트합니다. 파일이 열려 있을 경우에는 부울 값 TRUE가 반환되고, 그렇지 않을 경우에는 파일이 열려 있지 않음을 나타내는 FALSE 값이 반환됩니다. 위의 슬라이드 예제는 두 개의 서브 프로그램을 함께 사용하는 방법을 보여줍니다. 전체 구문은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

FOPEN 및 IS_OPEN 함수 파라미터: 예제(계속)

```

CREATE OR REPLACE PROCEDURE read_file(p_dir VARCHAR2,
p_filename VARCHAR2) IS
  f_file UTL_FILE.FILE_TYPE;
  v_buffer VARCHAR2(200);
  v_lines PLS_INTEGER := 0;
BEGIN
  DBMS_OUTPUT.PUT_LINE(' Start ');
  IF NOT UTL_FILE.IS_OPEN(f_file) THEN
    DBMS_OUTPUT.PUT_LINE(' Open ');
    f_file := UTL_FILE.FOPEN (p_dir, p_filename, 'R');
    DBMS_OUTPUT.PUT_LINE(' Opened ');
  BEGIN
    LOOP
      UTL_FILE.GET_LINE(f_file, v_buffer);
      v_lines := v_lines + 1;
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_lines, '099') || |
                           ' || buffer);
    END LOOP;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE(' ** End of File **');
  END; -- ends Begin
  DBMS_OUTPUT.PUT_LINE(v_lines || ' lines read from file');
  UTL_FILE.FCLOSE(f_file);
END IF;
END read_file;
/
SHOW ERRORS
EXECUTE read_file('REPORTS_DIR', 'instructor.txt')

```

다음은 위 코드에 대한 출력의 일부입니다.

```

PROCEDURE read_file(dir Compiled.
No Errors.
line 27: SQLPLUS Command Skipped: set serveroutput on
anonymous block completed
Start
Open
Opened
001 SALARY REPORT: GENERATED ON
002                         08-MAR-01
003
004 DEPARTMENT: 10
005   EMPLOYEE: Whalen earns: 4400

```

...

```

120 DEPARTMENT: 110
121   EMPLOYEE: Higgins earns: 12000
122   EMPLOYEE: Gietz earns: 8300
123   EMPLOYEE: Grant earns: 7000
124 *** END OF REPORT ***
** End of File **
124 lines read from file

```

UTL_FILE 사용: 예제

```

CREATE OR REPLACE PROCEDURE sal_status(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    CURSOR cur_emp IS
        SELECT last_name, salary, department_id
        FROM employees ORDER BY department_id;
    v_newdeptno employees.department_id%TYPE;
    v_olddeptno employees.department_id%TYPE := 0;
BEGIN
    f_file:= UTL_FILE.FOPEN(p_dir, p_filename, 'W');
    UTL_FILE.PUT_LINE(f_file,
        'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE (f_file);
    . . .

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UTL_FILE 사용: 예제

슬라이드 예제에서 `sal_status` 프로시저는 급여 정보가 포함된 각 부서의 사원 보고서를 생성합니다. 데이터는 `UTL_FILE` 패키지를 사용하여 텍스트 파일에 기록됩니다. 코드 예제에서 `file` 변수는 `BINARY_INTEGER` 데이터 유형의 ID 필드가 포함된 레코드인 `UTL_FILE.FILE_TYPE` 패키지 유형으로 선언됩니다. 예를 들면 다음과 같습니다.

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

`FILE_TYPE` 레코드 필드는 `UTL_FILE` 패키지 전용(private)이므로 참조되거나 변경되어서는 안됩니다. `sal_status` 프로시저는 다음 두 파라미터를 사용합니다.

- `p_dir` 파라미터 - 텍스트 파일이 기록되는 디렉토리의 이름
- `p_filename` 파라미터 - 파일 이름 지정

예를 들어 프로시저를 호출하려면 다음과 같이 사용하십시오.

```
EXECUTE sal_status('REPORTS_DIR', 'salreport2.txt')
```

참고: 사용된 디렉토리 위치(`REPORTS_DIR`)는 `CREATE DIRECTORY` 문에 의해 생성된 디렉토리 alias일 경우 대문자여야 합니다. 루프에서 파일을 읽을 때 `NO_DATA_FOUND` 예외가 발견되면 루프가 종료되어야 합니다. `UTL_FILE` 출력은 동기화 상태로 보내집니다. `DBMS_OUTPUT` 프로시저는 완료될 때까지 출력을 생성하지 않습니다.

UTL_FILE 사용: 예제

```

. . .
FOR emp_rec IN cur_emp LOOP
    IF emp_rec.department_id <> v_olddeptno THEN
        UTL_FILE.PUT_LINE (f_file,
            'DEPARTMENT: ' || emp_rec.department_id);
        UTL_FILE.NEW_LINE (f_file);
    END IF;
    UTL_FILE.PUT_LINE (f_file,
        ' EMPLOYEE: ' || emp_rec.last_name ||
        ' earns: ' || emp_rec.salary);
    v_olddeptno := emp_rec.department_id;
    UTL_FILE.NEW_LINE (f_file);
END LOOP;
UTL_FILE.PUT_LINE(f_file,'*** END OF REPORT ***');
UTL_FILE.FCLOSE (f_file);
EXCEPTION
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
        RAISE_APPLICATION_ERROR(-20001,'Invalid File.');
    WHEN UTL_FILE.WRITE_ERROR THEN
        RAISE_APPLICATION_ERROR (-20002, 'Unable to write to file');
END sal_status;/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UTL_FILE 사용: 예제(계속)

다음은 salreport2.txt 출력 파일의 샘플입니다.

```

salreport2.txt x
REPORT: GENERATED ON 07-AUG-09

DEPARTMENT: 10

EMPLOYEE: Whalen earns: 4400

DEPARTMENT: 20

EMPLOYEE: Hartstein earns: 13000
EMPLOYEE: Fay earns: 6000

DEPARTMENT: 30

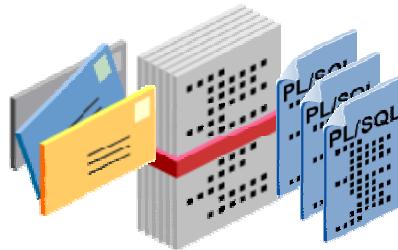
EMPLOYEE: Raphaely earns: 11000
EMPLOYEE: Khoo earns: 3100
EMPLOYEE: Baida earns: 2900

...

```

UTL_MAIL 패키지란?

- 전자 메일을 관리하기 위한 유틸리티
- SMTP_OUT_SERVER 데이터베이스 초기화 파라미터의 설정이 필요함
- 다음 프로시저를 제공함
 - SEND – 첨부 파일이 없는 메시지의 경우
 - SEND_ATTACH_RAW – Binary File이 첨부된 메시지의 경우
 - SEND_ATTACH_VARCHAR2 – 텍스트 파일이 첨부된 메시지의 경우



ORACLE

Copyright © 2009, Oracle. All rights reserved.

UTL_MAIL 사용

UTL_MAIL 패키지는 첨부 파일, 참조, 숨은 참조, 수신 확인 메일과 같이 일반적으로 사용되는 전자 메일 기능을 포함하는 전자 메일 관리용 유틸리티입니다.

SMTP_OUT_SERVER 구성 요구 사항과 여기에 수반되는 보안 노출 때문에 UTL_MAIL 패키지는 기본적으로 설치되어 있지 않습니다. UTL_MAIL을 설치할 때는 SMTP_OUT_SERVER에 의해 정의된 포트에 데이터 전송이 너무 많이 발생하지 않도록 조치를 취해야 합니다. UTL_MAIL을 설치하려면 SQL*Plus에 DBA 유저로 로그인하고 다음 스크립트를 실행하십시오.

```
@$ORACLE_HOME/rdbms/admin/utlmail.sql
@$ORACLE_HOME/rdbms/admin/prvtmail.p1b
```

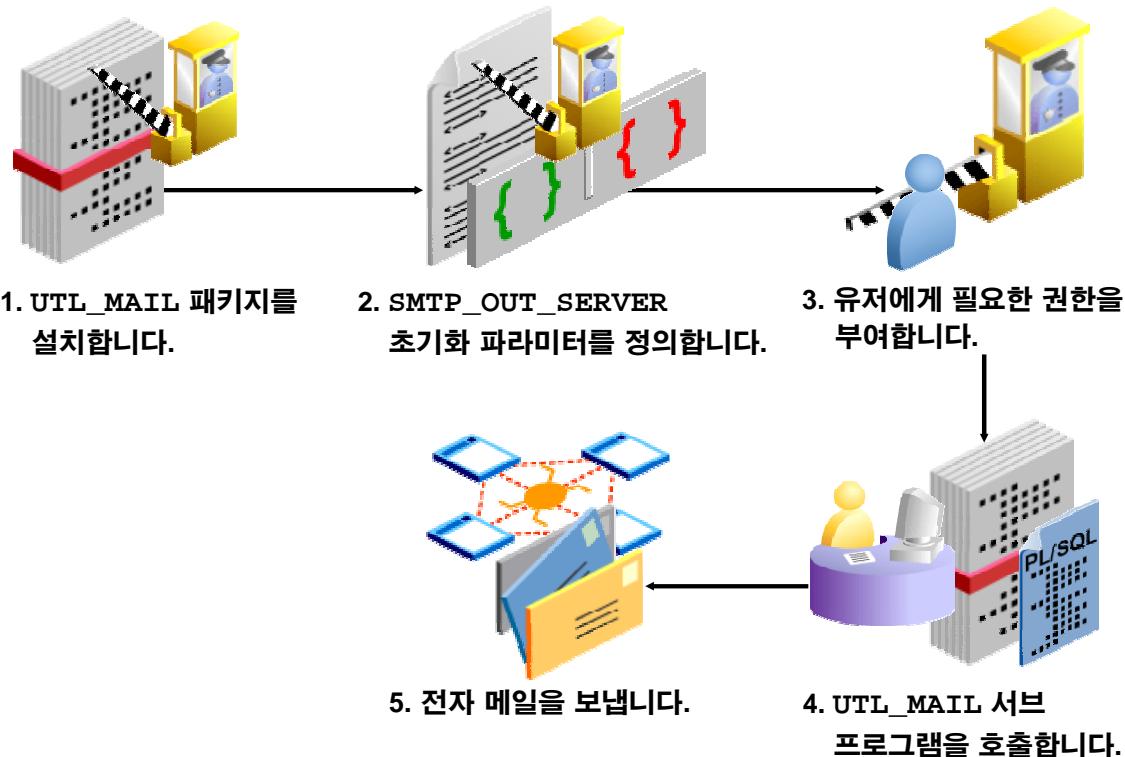
init.ora 데이터베이스 초기화 파일에 SMTP_OUT_SERVER 파라미터를 정의해야 합니다.
SMTP_OUT_SERVER=mystmpserver.mydomain.com

UTL_MAIL 사용(계속)

SMTP_OUT_SERVER 파라미터는 UTL_MAIL이 발송 전자 메일을 배달하는 SMTP 호스트 및 포트를 지정합니다. 서버가 여러 개일 경우 쉼표로 구분하여 지정할 수 있습니다. UTL_MAIL은 리스트에 있는 첫번째 서버를 사용할 수 없을 경우 두번째 서버를 시도하는 식으로 계속 그 다음 서버를 시도합니다. SMTP_OUT_SERVER가 정의되어 있지 않을 경우 네트워크 구조 내에서 데이터베이스의 논리적 위치를 지정하는 데이터베이스 초기화 파라미터 DB_DOMAIN에서 파생된 기본 설정이 호출됩니다. 예를 들면 다음과 같습니다.

```
db_domain=mydomain.com
```

UTL_MAIL 설정 및 사용: 개요



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

UTL_MAIL 설정 및 사용: 개요

Oracle Database 11g에서 UTL_MAIL 패키지는 이제 호출자의 권한 패키지이며, 호출하는 유저는 연결할 원격 네트워크 호스트에 할당된 액세스 제어 리스트에서 연결 권한을 부여 받아야 합니다. 이 작업은 보안 관리자가 수행합니다.

참고

- SYSDBA 권한이 있는 유저가 이 패키지를 사용하는 데 필요한 Fine-Grained Privilege를 유저에게 부여하는 방법에 대한 자세한 내용은 *Oracle Database Security Guide 11g Release 2 (11.2)* 설명서의 "Managing Fine-Grained Access to External Network Services" 항목과 *Oracle Database 11g Advanced PL/SQL* 장사 주도형 과정을 참조하십시오.
- Firewall 제한으로 인해 이 단원의 UTL_MAIL 예제는 보여줄 수 없습니다.
따라서 UTL_MAIL 사용에 대한 연습은 없습니다.

UTL_MAIL 서브 프로그램 요약

서브 프로그램	설명
SEND 프로시저	전자 메일 메시지를 패키지화하고, SMTP 정보를 찾고, 메시지를 SMTP 서버로 배달하여 수신자에게 전달함
SEND_ATTACH_RAW 프로시저	RAW 첨부 파일에 대해 오버로드된 SEND 프로시저를 나타냄
SEND_ATTACH_VARCHAR2 프로시저	VARCHAR2 첨부 파일에 대해 오버로드된 SEND 프로시저를 나타냄



Copyright © 2009, Oracle. All rights reserved.

UTL_MAIL 설치 및 사용

- SYSDBA로서 SQL Developer 또는 SQL*Plus 사용:

- UTL_MAIL 패키지 설치

```
@?/rdbms/admin/utlmail.sql
@?/rdbms/admin/prvtmail.plb
```

- SMTP_OUT_SERVER 설정

```
ALTER SYSTEM SET SMTP_OUT_SERVER='smtp.server.com'
SCOPE=SPFILE
```

- 개발자로서 UTL_MAIL 프로시저 호출:

```
BEGIN
  UTL_MAIL.SEND('otn@oracle.com','user@oracle.com',
    message => 'For latest downloads visit OTN',
    subject => 'OTN Newsletter');
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UTL_MAIL 설치 및 사용

슬라이드는 네트워크에서 SMTP 호스트 이름에 대해 SMTP_OUT_SERVER 파라미터를 구성하는 방법과 기본적으로 설치되어 있지 않은 UTL_MAIL 패키지를 설치하는 방법을 보여줍니다.

SMTP_OUT_SERVER 파라미터를 변경하면 데이터베이스 Instance를 재시작해야 합니다.

이러한 작업은 SYSDBA 권한을 가진 유저가 수행할 수 있습니다.

슬라이드의 마지막 예제는 UTL_MAIL.SEND 프로시저를 사용하여 최소한 제목과 메시지가 포함된 텍스트 메시지를 보내는 가장 간단한 방법을 보여줍니다. 처음 두 개의 필수 파라미터는 다음과 같습니다.

- **sender** 전자 메일 주소(이 경우 otn@oracle.com)
- **recipients** 전자 메일 주소(예: user@oracle.com). 쉼표로 구분된 주소 리스트를 값으로 사용할 수 있습니다.

UTL_MAIL.SEND 프로시저는 cc, bcc, priority 등의 몇 가지 기타 파라미터를 제공합니다. 이러한 파라미터는 값이 지정되지 않을 경우 기본값이 사용됩니다. 예제에서 message 파라미터는 전자 메일 텍스트를 지정하고, subject 파라미터는 제목 행 텍스트를 포함합니다. HTML 태그가 있는 HTML 메시지를 보내려면 mime_type 파라미터를 추가하십시오(예: mime_type=>'text/html').

참고: 모든 UTL_MAIL 프로시저 파라미터에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

SEND 프로시저 구문

전자 메일 메시지를 적절한 형식으로 패키지화하고, SMTP 정보를 찾고, 메시지를 SMTP 서버로 배달하여 수신자에게 전달합니다.

```
UTL_MAIL.SEND (
    sender      IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients  IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc          IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    bcc         IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    subject     IN      VARCHAR2 CHARACTER SET ANY_CS
                      DEFAULT NULL,
    message     IN      VARCHAR2 CHARACTER SET ANY_CS,
    mime_type   IN      VARCHAR2
                      DEFAULT 'text/plain; charset=us-ascii',
    priority    IN      PLS_INTEGER DEFAULT NULL);
```

Copyright © 2009, Oracle. All rights reserved.

SEND 프로시저

이 프로시저는 전자 메일 메시지를 적절한 형식으로 패키지화하고, SMTP 정보를 찾고, 메시지를 SMTP 서버로 배달하여 수신자에게 전달합니다. 또한 SMTP API를 숨기고 사용하기 쉬운 한 행 전자 메일 기능을 노출합니다.

SEND 프로시저 파라미터

- **sender:** 송신자의 전자 메일 주소입니다.
- **recipients:** 수신자의 전자 메일 주소로서 쉼표로 구분됩니다.
- **cc:** 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **bcc:** 숨은 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **subject:** 전자 메일 제목 문자열에 포함될 문자열입니다. 기본값은 NULL입니다.
- **message:** 텍스트 메시지 본문입니다.
- **mime_type:** 메시지의 mime 형식입니다. 기본값은 'text/plain; charset=us-ascii'.
- **priority:** 메시지 우선 순위입니다. 기본값은 NULL입니다.

SEND_ATTACH_RAW 프로시저

이 프로시저는 RAW 첨부 파일에 대해 오버로드된 SEND 프로시저입니다.

```
UTL_MAIL.SEND_ATTACH_RAW (
    sender          IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients     IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN      VARCHAR2 DEFAULT CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    priority        IN      PLS_INTEGER DEFAULT 3,
    attachment     IN      RAW,
    att_inline      IN      BOOLEAN DEFAULT TRUE,
    att_mime_type   IN      VARCHAR2 CHARACTER SET ANY_CS
                            DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL);
```

Copyright © 2009, Oracle. All rights reserved.

SEND_ATTACH_RAW 프로시저 파라미터

- **sender:** 송신자의 전자 메일 주소입니다.
- **recipients:** 수신자의 전자 메일 주소로서 쉼표로 구분됩니다.
- **cc:** 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **bcc:** 숨은 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **subject:** 전자 메일 제목 문자열에 포함될 문자열입니다. 기본값은 NULL입니다.
- **message:** 텍스트 메시지 본문입니다.
- **mime_type:** 메시지의 mime 형식입니다. 기본값은 'text/plain; charset=us-ascii'입니다.
- **priority:** 메시지 우선 순위입니다. 기본값은 NULL입니다.
- **attachment:** RAW 첨부 파일입니다.
- **att_inline:** 첨부 파일을 메시지 본문과 함께 인라인으로 볼 수 있는지 여부를 지정합니다. 기본값은 TRUE입니다.
- **att_mime_type:** 첨부 파일의 mime 형식입니다. 기본값은 'application/octet'입니다.
- **att_filename:** 첨부 파일이 포함된 파일 이름을 지정하는 문자열입니다. 기본값은 NULL입니다.

Binary File을 첨부하여 전자 메일 보내기: 예제

```

CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
    UTL_MAIL.SEND_ATTACH_RAW(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Logo',
        mime_type => 'text/html'
        attachment => get_image('oracle.gif'),
        att_inline => true,
        att_mime_type => 'image/gif',
        att_filename => 'oralogo.gif');
END;
/

```



Copyright © 2009, Oracle. All rights reserved.

Binary File을 첨부하여 전자 메일 보내기: 예제

슬라이드는 Binary File이 첨부된 텍스트 또는 HTML 메시지를 보내는 UTL_MAIL.SEND_ATTACH_RAW 프로시저를 호출하는 프로시저를 보여줍니다. SEND_ATTACH_RAW 프로시저는 전자 메일 메시지의 주요 부분에 값을 제공하는 sender, recipients, message, subject, mime_type 파라미터 이외에도 다음과 같은 파라미터(위에 강조 표시됨)를 포함합니다.

- attachment 파라미터(필수)는 RAW 데이터 유형을 사용하여 최대 크기는 32,767 이진 문자입니다.
- att_inline 파라미터(선택 사항)는 첨부 파일을 메시지 본문과 함께 볼 수 있음을 나타내는 부울(기본값 TRUE)입니다.
- att_mime_type 파라미터(선택 사항)는 첨부 파일의 형식을 지정합니다. 제공되지 않으면 application/octet으로 설정됩니다.
- att_filename 파라미터(선택 사항)는 첨부 파일에 파일 이름을 할당합니다. 기본적으로 NULL이며, 이 경우 기본 이름이 할당됩니다.

예제의 get_image 함수는 BFILE을 사용하여 이미지 데이터를 읽습니다. BFILE을 사용하려면 CREATE DIRECTORY 문을 사용하여 데이터베이스에 논리적 디렉토리 이름을 생성해야 합니다. get_image의 코드는 다음 페이지에 나옵니다.

Binary File을 첨부하여 전자 메일 보내기: 예제(계속)

get_image 함수는 DBMS_LOB 패키지를 사용하여 운영 체제에서 Binary File을 읽습니다.

```
CREATE OR REPLACE FUNCTION get_image(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP')
RETURN RAW IS
    image RAW(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    image := DBMS_LOB.SUBSTR(file);
    DBMS_LOB.CLOSE(file);
    RETURN image;
END;
/
```

TEMP라는 디렉토리를 생성하려면 SQL Developer 또는 SQL*Plus에서 다음 명령문을 실행하십시오.

```
CREATE DIRECTORY temp AS 'd:\temp';
```

참고

- 이 명령문을 실행하려면 CREATE ANY DIRECTORY 시스템 권한이 필요합니다.
- Oracle Education Center의 firewall 제한으로 인해 이 페이지와 이전 페이지에 있는 예제는 데모로 사용할 수 없습니다.

SEND_ATTACH_VARCHAR2 프로시저

이 프로시저는 VARCHAR2 첨부 파일에 대해 오버로드된 SEND 프로시저입니다.

```
UTL_MAIL.SEND_ATTACH_VARCHAR2 (
    sender          IN      VARCHAR2 CHARACTER SET ANY_CS,
    recipients     IN      VARCHAR2 CHARACTER SET ANY_CS,
    cc              IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    bcc             IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    subject         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    message         IN      VARCHAR2 CHARACTER SET ANY_CS DEFAULT NULL,
    mime_type       IN      VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    priority        IN      PLS_INTEGER DEFAULT 3,
    attachment     IN      VARCHAR2 CHARACTER SET ANY_CS,
    att_inline      IN      BOOLEAN DEFAULT TRUE,
    att_mime_type   IN      VARCHAR2 CHARACTER SET ANY_CS
                           DEFAULT 'text/plain; charset=us-ascii',
    att_filename    IN      VARCHAR2CHARACTER SET ANY_CS DEFAULT NULL);
```

Copyright © 2009, Oracle. All rights reserved.

SEND_ATTACH_VARCHAR2 프로시저 파라미터

- **sender:** 송신자의 전자 메일 주소입니다.
- **recipients:** 수신자의 전자 메일 주소로서 쉼표로 구분됩니다.
- **cc:** 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **bcc:** 숨은 참조 수신자의 전자 메일 주소로서 쉼표로 구분됩니다. 기본값은 NULL입니다.
- **subject:** 전자 메일 제목 문자열에 포함될 문자열입니다. 기본값은 NULL입니다.
- **Message:** 텍스트 메시지 본문입니다.
- **mime_type:** 메시지의 mime 형식입니다. 기본값은
'text/plain; charset=us-ascii'입니다.
- **priority:** 메시지 우선 순위입니다. 기본값은 NULL입니다.
- **attachment:** 텍스트 첨부 파일입니다.
- **att_inline:** 첨부 파일의 인라인 여부를 지정합니다. 기본값은 TRUE입니다.
- **att_mime_type:** 첨부 파일의 mime 형식입니다. 기본값은
'text/plain; charset=us-ascii'입니다.
- **att_filename:** 첨부 파일이 포함된 파일 이름을 지정하는 문자열입니다.
기본값은 NULL입니다.

텍스트 파일을 첨부하여 전자 메일 보내기: 예제

```

CREATE OR REPLACE PROCEDURE send_mail_file IS
BEGIN
    UTL_MAIL.SEND_ATTACH_VARCHAR2(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Notes',
        mime_type => 'text/html'
        attachment => get_file('notes.txt'),
        att_inline => false,
        att_mime_type => 'text/plain',
        att_filename => 'notes.txt');
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

텍스트 파일을 첨부하여 전자 메일 보내기

슬라이드는 텍스트 파일이 첨부된 텍스트 또는 HTML 메시지를 보내는 UTL_MAIL.SEND_ATTACH_VARCHAR2 프로시저를 호출하는 프로시저를 보여줍니다. SEND_ATTACH_VARCHAR2 프로시저는 전자 메일 메시지의 주요 부분에 값을 제공하는 sender, recipients, message, subject, mime_type 파라미터 이외에도 다음과 같은 파라미터(위에 강조 표시됨)를 포함합니다.

- attachment 파라미터(필수)는 VARCHAR2 데이터 유형을 사용하며 최대 크기는 32,767 이진 문자입니다.
- att_inline 파라미터(선택 사항)는 첨부 파일을 메시지 본문과 함께 볼 수 있음을 나타내는 부울(기본값 TRUE)입니다.
- att_mime_type 파라미터(선택 사항)는 첨부 파일의 형식을 지정합니다. 제공되지 않으면 application/octet으로 설정됩니다.
- att_filename 파라미터(선택 사항)는 첨부 파일에 파일 이름을 할당합니다. 기본적으로 NULL이며, 이 경우 기본 이름이 할당됩니다.

예제의 get_file 함수는 attachment 파라미터 값을 얻기 위해 BFILE을 사용하여 운영 체제 디렉토리에서 텍스트 파일을 읽습니다. 이렇게 하면 간단하게 VARCHAR2 변수에서 파라미터 값을 채울 수 있습니다. get_file의 코드는 다음 페이지에 나옵니다.

텍스트 파일을 첨부하여 전자 메일 보내기(계속)

`get_file` 함수는 DBMS_LOB 패키지를 사용하여 운영 체제에서 Binary File을 읽고, UTL_RAW 패키지를 사용하여 RAW 이진 데이터를 VARCHAR2 데이터 유형 형식의 읽기 가능한 텍스트 데이터로 변환합니다.

```
CREATE OR REPLACE FUNCTION get_file(
    filename VARCHAR2, dir VARCHAR2 := 'TEMP' )
RETURN VARCHAR2 IS
    contents VARCHAR2(32767);
    file BFILE := BFILENAME(dir, filename);
BEGIN
    DBMS_LOB.FILEOPEN(file, DBMS_LOB.FILE_READONLY);
    contents := UTL_RAW.CAST_TO_VARCHAR2(
        DBMS_LOB.SUBSTR(file));
    DBMS_LOB CLOSE(file);
    RETURN contents;
END;
/
```

참고: 또는 UTL_FILE 패키지 기능을 사용하여 텍스트 파일의 내용을 VARCHAR2 변수로 읽을 수 있습니다.

위의 예제를 사용하려면 SQL*Plus에서 다음 명령문과 유사한 방식으로 TEMP 디렉토리를 생성해야 합니다.

```
CREATE DIRECTORY temp AS '/temp';
```

참고

- 이 명령문을 실행하려면 CREATE ANY DIRECTORY 시스템 권한이 필요합니다.
- Oracle Education Center의 firewall 제한으로 인해 이 페이지와 이전 페이지에 있는 예제는 데모로 사용할 수 없습니다.

퀴즈

오라클에서 제공하는 UTL_FILE 패키지는 데이터베이스 서버 운영 체제에 있는 텍스트 파일에 액세스하는 데 사용됩니다. 데이터베이스에서는 디렉토리 객체를 통해 특정 운영 체제 디렉토리에 액세스할 수 있는 기능을 제공합니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1

오라클에서 제공하는 UTL_FILE 패키지는 데이터베이스 서버 운영 체제에 있는 텍스트 파일에 액세스하는 데 사용됩니다. 데이터베이스는 다음을 사용하여 특정 운영 체제 디렉토리에 대한 읽기/쓰기 액세스 권한을 제공합니다.

- alias를 운영 체제 디렉토리와 연관시키는 CREATE DIRECTORY 문. 데이터베이스 디렉토리 alias에 READ 및 WRITE 권한을 부여하여 운영 체제 파일에 대한 액세스 유형을 제어 할 수 있습니다.
- 경로가 utl_file_dir 데이터베이스 초기화 파라미터에 지정되었습니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- DBMS_OUTPUT 패키지 작동 방식
- UTL_FILE을 사용하여 출력을 운영 체제 파일로 지정하는 방법
- UTL_MAIL의 주요 기능

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 오라클 데이터베이스와 함께 제공되는 패키지의 작은 하위 집합에 대해 설명합니다. DBMS_OUTPUT은 디버깅 용도 그리고 프로시저에 따라 생성된 정보를 SQL*Plus의 화면에 표시하는 데 광범위하게 사용되었습니다.

이 단원에서는 UTL_FILE을 사용하여 운영 체제에서 텍스트 파일을 생성할 수 있도록 데이터베이스가 제공하는 고급 기능을 사용하는 방법에 대해 배웠습니다. UTL_MAIL 패키지를 사용하여 첨부 파일 없이 전자 메일을 보내거나 Binary File 또는 텍스트 파일이 첨부된 전자 메일을 보내는 방법도 배웠습니다.

참고: 모든 PL/SQL 패키지 및 유형에 대한 자세한 내용은 *PL/SQL Packages and Types Reference*를 참조하십시오.

연습 5: 개요

이 연습에서는 UTL_FILE을 사용하여 텍스트 보고서를 작성하는 방법을 살펴봅니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 5: 개요

이 연습에서는 UTL_FILE을 사용하여 각 부서의 사원에 대한 텍스트 파일 보고서를 작성합니다.



동적 SQL 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- SQL 문의 실행 흐름 설명
- NDS(Native Dynamic SQL)를 사용하여 SQL 문을 동적으로 생성 및 실행
- NDS 대신 DBMS_SQL 패키지를 사용하여 SQL 문을 동적으로 생성하고 실행해야 할 상황 식별

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 SQL 문을 동적으로 생성하고 실행하는 방법, 즉 런타임에 PL/SQL에서 Native Dynamic SQL 문을 사용하는 방법에 대해 설명합니다.

단원 내용

- **NDS(Native Dynamic SQL) 사용**
- **DBMS_SQL 패키지 사용**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL의 실행 흐름

- 모든 SQL 문은 다음 단계 중 일부 또는 전부를 거칩니다.
 - 구문 분석
 - 바인드
 - 실행
 - 패치(fetch)
- 일부 단계는 일부 명령문에만 관련될 수도 있습니다.
 - 패치(fetch) 단계는 query에 적용될 수 있습니다.
 - SELECT, DML, MERGE, COMMIT, SAVEPOINT, ROLLBACK 같은 내장된 SQL 문에 대해서는 구문 분석 및 바인드 단계가 컴파일 시 수행됩니다.
 - 동적 SQL 문의 경우 모든 단계가 런타임에 수행됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문 처리 단계

모든 SQL 문은 여러 단계를 거쳐야 합니다. 그러나 일부 단계는 일부 명령문에만 관련될 수 있습니다. 주요 단계는 다음과 같습니다.

- **구문 분석:** 모든 SQL 문은 구문을 분석해야 합니다. 명령문 구문 분석에는 명령문의 구문을 검사하고 명령문을 검증함으로써 객체에 대한 참조가 모두 올바른지 확인하고 해당 객체와 관련된 권한이 존재하는지를 확인하는 과정이 포함됩니다.
- **바인드:** 구문 분석 후 Oracle 서버는 명령문의 바인드 변수에 대한 값을 필요로 할 수 있습니다. 이러한 값을 얻는 과정을 변수 바인딩(Binding Variables)이라고 합니다. 명령문이 바인드 변수를 포함하지 않을 경우 이 단계를 생략해도 됩니다.
- **실행:** 이 시점에서 Oracle 서버는 필요한 정보와 자원을 모두 가지고 있으므로 명령문이 실행됩니다. Query가 아닌 명령문의 경우 이 단계가 마지막 단계입니다.
- **패치(fetch):** 패치(fetch) 단계(query에만 해당됨)에서는 행이 선택되어 정렬(query에서 요청하는 경우)되며 이어지는 각각의 패치(fetch)는 마지막 행이 패치(fetch)될 때까지 다른 결과 행을 읽어 들입니다.

동적 SQL 작업

동적 SQL을 사용하여 런타임에 구조를 변경할 수 있는 SQL 문을 생성할 수 있습니다. 동적 SQL의 특징은 다음과 같습니다.

- **응용 프로그램 내에서 문자열, 문자열 변수 또는 문자열 표현식으로 구성되고 저장됩니다.**
- **다양한 열 데이터 또는 위치 표시자(바인드 변수)를 사용하거나 사용하지 않는 다양한 조건을 가진 SQL 문입니다.**
- **PL/SQL에서 DDL, DCL 또는 세션 제어문을 작성하고 실행할 수 있도록 합니다.**
- **Native Dynamic SQL 문 또는 DBMS_SQL 패키지를 사용하여 실행됩니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

동적 SQL

PL/SQL에서 사용할 수 있는 내장된 SQL 문은 SELECT, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK으로 제한되는데, 이들은 모두 컴파일 시 구문이 분석되므로 고정된 구조를 가집니다. 다음과 같은 경우 동적 SQL 기능을 사용해야 합니다.

- 런타임에 SQL 문의 구조를 변경해야 할 경우
- PL/SQL에서 DDL(데이터 정의) 문 및 기타 SQL 기능에 액세스해야 할 경우

PL/SQL에서 이러한 종류의 작업을 수행하려면 동적으로 SQL 문을 문자열로 생성한 후 다음 중 하나를 사용하여 실행해야 합니다.

- EXECUTE IMMEDIATE가 있는 Native Dynamic SQL 문
- DBMS_SQL 패키지

"동적 SQL"이란 소스 프로그램에 포함되어 있지는 않지만 문자열로 생성되어 런타임에 실행되는 SQL 문을 사용하는 과정을 말합니다. SQL 문은 런타임에 동적으로 생성되며 PL/SQL 변수에 액세스하고 PL/SQL 변수를 사용할 수 있습니다. 예를 들면, 동적 SQL을 사용하여 런타임까지 이름을 알 수 없는 테이블에 대해 작동하는 프로시저를 생성하거나, DDL 문(예: CREATE TABLE), 데이터 제어문(예: GRANT) 또는 세션 제어문(예: ALTER SESSION)을 실행합니다.

동적 SQL 사용

- **동적 SQL 문의 전체 텍스트를 런타임까지 알 수 없는 경우에 동적 SQL을 사용합니다. 따라서 구문은 컴파일 시 검사되지 않고 런타임에 검사됩니다.**
- **선행 컴파일 시 다음 항목 중 하나를 알 수 없는 경우 동적 SQL을 사용합니다.**
 - SQL 문의 텍스트(명령, 절 등)
 - 호스트 변수의 수와 데이터 유형
 - 데이터베이스 객체에 대한 참조(예: 테이블, 열, 인덱스, 시퀀스, Username, 뷰)
- **동적 SQL을 사용하여 PL/SQL 프로그램을 보다 일반적이고 융통적으로 만들 수 있습니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

동적 SQL 사용

PL/SQL에서 컴파일 시 전체 텍스트를 알 수 없는 경우 다음 SQL 문을 실행하려면 동적 SQL이 필요합니다.

- 컴파일 시 알 수 없는 식별자(예: 테이블 이름)가 포함된 SELECT 문
- 컴파일 시 열 이름을 알 수 없는 WHERE 절

참고

동적 SQL에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- *Pro*C/C++ Programmer's Guide 11g Release 2 (11.2)* 설명서
 - 단원 13, *Oracle Dynamic SQL*에서는 동적 SQL 문을 정의하는 데 사용할 수 있는 네 가지 방법에 대해 설명합니다. 또한 각 방법의 기능과 제한 사항에 대해 설명하고 올바른 방법을 선택하는 지침을 제공합니다. 설명서의 후반부에서는 각 방법을 어떻게 사용하는지 보여주고 연습에 사용할 수 있는 예제 프로그램을 제공합니다.
 - 단원 15, *Oracle Dynamic SQL: Method 4*에는 동적 SQL 문을 정의하는 Method 4에 대한 자세한 정보가 포함되어 있습니다.
- *Oracle PL/SQL Programming* book(Steven Feuerstein 및 Bill Pribyl 저). 단원 16, *Dynamic SQL and Dynamic PL/SQL*에는 동적 SQL에 대한 자세한 정보가 포함되어 있습니다.

NDS(Native Dynamic SQL)

- PL/SQL 언어에서 동적 SQL에 대한 기본 지원을 직접 제공합니다.
- 실행 시까지 구조를 알 수 없는 SQL 문을 실행할 수 있는 기능을 제공합니다.
- 동적 SQL 문이 여러 행을 반환하는 SELECT 문인 경우에는 NDS에서 다음 중 하나를 수행할 수 있습니다.
 - EXECUTE IMMEDIATE 문을 BULK COLLECT INTO 절과 함께 사용
 - OPEN-FOR, FETCH 및 CLOSE 문 사용
- Oracle Database 11g에서 NDS는 CLOB 인수를 받아들임으로써 32KB보다 큰 명령문을 지원합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Native Dynamic SQL

Native Dynamic SQL은 실행 시에 구조가 생성되는 SQL 문을 동적으로 실행할 수 있는 기능을 제공합니다. Native Dynamic SQL을 지원하기 위해 PL/SQL에서 확장되거나 추가된 명령문은 다음과 같습니다.

- **EXECUTE IMMEDIATE:** 명령문을 준비하고 실행하며 변수를 반환한 다음 자원 할당을 해제합니다.
- **OPEN-FOR:** 커서 변수를 사용하여 명령문을 준비하고 실행합니다.
- **FETCH:** 커서 변수를 사용하여 열려 있는 명령문의 결과를 읽어 들입니다.
- **CLOSE:** 커서 변수가 사용하는 커서를 닫고 자원 할당을 해제합니다.

EXECUTE IMMEDIATE 및 OPEN 문의 Dynamic Parameter에 바인드 변수를 사용할 수 있습니다. Native Dynamic SQL의 기능은 다음과 같습니다.

- 동적 SQL 문 정의
- 이름이 아닌 위치에 따라 바인드되는 IN, IN OUT, OUT 바인드 변수 처리

EXECUTE IMMEDIATE 문 사용

NDS 또는 PL/SQL 익명 블록에 대해 EXECUTE IMMEDIATE 문 사용:

```
EXECUTE IMMEDIATE dynamic_string
[ INTO {define_variable
[, define_variable] ... | record} ]
[ USING [ IN|OUT|IN OUT] bind_argument
[, [ IN|OUT|IN OUT] bind_argument] ... ];
```

- INTO는 단일 행 query에 사용되며 읽어 들인 열 값을 적용할 변수나 레코드를 지정합니다.
- USING은 모든 바인드 인수를 포함하는 데 사용됩니다. 기본 파라미터 모드는 IN입니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

EXECUTE IMMEDIATE 문 사용

EXECUTE IMMEDIATE 문은 SQL 문 또는 PL/SQL 익명 블록을 실행하는 데 사용할 수 있습니다. 구문 요소는 다음과 같습니다.

- dynamic_string은 동적 SQL 문(종료자 사용 안함)이나 PL/SQL 블록(종료자 사용)을 나타내는 문자열 표현식입니다.
- define_variable은 선택된 열 값을 저장하는 PL/SQL 변수입니다.
- record는 선택된 행을 저장하는 유저 정의 또는 %ROWTYPE 레코드입니다.
- bind_argument는 동적 SQL 문 또는 PL/SQL 블록으로 값이 전달되는 표현식입니다.
- INTO 절은 읽어 들인 열 값을 적용할 변수나 레코드를 지정하며, 단일 행 query에 사용됩니다. query에 의해 읽어 들인 각 값의 경우 INTO 절에 호환 가능한 유형의 해당 변수나 필드가 있어야 합니다.
- USING 절은 모든 바인드 인수를 포함하고 있습니다. 기본 파라미터 모드는 IN입니다.

숫자, 문자, 문자열 리터럴은 바인드 인수로 사용할 수 있지만 부울 리터럴(TRUE, FALSE, NULL)은 사용할 수 없습니다.

참고: 여러 행 query의 경우 OPEN-FOR, FETCH 및 CLOSE를 사용하십시오. 대량 처리 작업에 대한 지원이 있기 때문에 슬라이드에 표시된 구문은 완전하지 않습니다. 대량 처리 작업에 대해서는 본 과정에서 다루지 않습니다.

NDS 사용 방법

방법 #	SQL 문 유형	사용된 NDS SQL 문
방법 1	호스트 변수가 없는 query가 아닌 명령문	USING 및 INTO 절이 없는 EXECUTE IMMEDIATE
방법 2	입력 호스트 변수의 수가 알려진 query가 아닌 명령문	USING 절이 있는 EXECUTE IMMEDIATE
방법 3	선택 리스트 항목과 입력 호스트 변수의 수가 알려진 query	USING 및 INTO 절이 있는 EXECUTE IMMEDIATE
방법 4	선택 리스트 항목 또는 입력 호스트 변수의 수를 알 수 없는 query	DBMS_SQL 패키지 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

NDS 사용 방법

슬라이드에 나열되는 네 가지 NDS 사용 방법은 점차적으로 일반화됩니다. 즉, 방법 2는 방법 1을 포함하고, 방법 3은 방법 1과 2를 포함하고, 방법 4는 방법 1, 2 및 3을 포함합니다. 그러나 다음과 같이 특정 종류의 SQL 문을 처리하는 데 가장 유용한 방법이 있습니다.

방법 1:

이 방법을 사용하면 프로그램에서 동적 SQL 문을 그대로 적용하거나 생성한 다음 EXECUTE IMMEDIATE 명령을 사용하여 즉시 실행할 수 있습니다. SQL 문은 query(SELECT 문)가 아니어야 하며 입력 호스트 변수에 대한 자리 표시자를 포함하지 않아야 합니다. 예를 들어 다음과 같은 호스트 문자열이 여기에 해당합니다.

- DELETE FROM EMPLOYEES WHERE DEPTNO = 20
- GRANT SELECT ON EMPLOYEES TO scott

방법 1에서 SQL 문은 실행될 때마다 구문 분석됩니다.

참고

- Query가 아닌 명령문의 예로는 DDL(데이터 정의) 문, UPDATE, INSERT 또는 DELETE가 있습니다.
- 선택 리스트 항목이라는 용어에는 SAL * 1.10 및 MAX(SAL)와 같은 열 이름과 표현식이 포함됩니다.

NDS 사용 방법(계속)

방법 2:

이 방법을 사용하면 프로그램에서 동적 SQL 문을 그대로 적용하거나 생성한 다음 PREPARE 및 EXECUTE 명령을 사용하여 처리할 수 있습니다. SQL 문은 query가 아니어야 합니다. 또한 선행 컴파일 시 입력 호스트 변수에 대한 자리 표시자 수와 데이터 유형을 알고 있어야 합니다. 예를 들어 다음과 같은 호스트 문자열이 이 범주에 해당합니다.

- `INSERT INTO EMPLOYEES (FIRST_NAME, LAST_NAME, JOB_ID) VALUES (:emp_first_name, :emp_last_name, :job_id)`
- `DELETE FROM EMPLOYEES WHERE EMPLOYEE_ID = :emp_number`

방법 2에서는 SQL 문이 한 번만 구문 분석되지만 다른 호스트 변수 값으로 여러 번 실행될 수 있습니다. CREATE 및 GRANT와 같은 SQL 데이터 정의문은 준비가 되면 실행됩니다.

방법 3:

이 방법을 사용하면 프로그램에서 동적 query를 그대로 적용하거나 생성한 다음 DECLARE, OPEN, FETCH 및 CLOSE 커서 명령과 함께 PREPARE 명령을 사용하여 처리할 수 있습니다.

선행 컴파일 시 선택 리스트 항목 수 및 입력 호스트 변수에 대한 자리 표시자 수와 데이터 유형을 알고 있어야 합니다. 예를 들어 다음과 같은 호스트 문자열이 여기에 해당합니다.

- `SELECT DEPARTMENT_ID, MIN(SALARY), MAX(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID`
- `SELECT LAST_NAME, EMPLOYEE_ID
FROM EMPLOYEES
WHERE DEPARTMENT_ID = :dept_number`

방법 4:

이 방법을 사용하면 프로그램에서 동적 SQL 문을 그대로 적용하거나 생성한 다음 descriptor를 사용하여 처리할 수 있습니다. descriptor는 프로그램과 오라클에서 동적 SQL 문의 변수에 대한 전체 설명을 보유하는 데 사용되는 메모리 영역입니다. 선택 리스트 항목 수 및 입력 호스트 변수에 대한 자리 표시자 수와 데이터 유형을 런타임까지 알 수 없습니다. 예를 들어 다음과 같은 호스트 문자열이 이 범주에 해당합니다.

- `INSERT INTO EMPLOYEES (<unknown>) VALUES (<unknown>)`
- `SELECT <unknown> FROM EMPLOYEES WHERE DEPARTMENT_ID = 20`

방법 4는 알 수 없는 수의 선택 리스트 항목 또는 입력 호스트 변수를 포함하는 동적 SQL 문에 필요합니다. 이 방법에서는 이 단원의 후반부에서 설명하는 DBMS_SQL 패키지를 사용합니다. 방법 4를 사용해야 하는 경우는 거의 없습니다.

참고

네 가지 동적 SQL 사용 방법에 대한 자세한 내용은 *Pro*C/C++ Programmer's Guide 11g Release 2 (11.2)* 설명서의 다음 단원을 참조하십시오.

- *단원 13, Oracle Dynamic SQL*
- *단원 15, Oracle Dynamic SQL: Method 4*

DDL 문을 사용하는 동적 SQL: 예제

```
-- Create a table using PL/SQL

CREATE OR REPLACE PROCEDURE create_table(
    p_table_name VARCHAR2, p_col_specs VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE ' || p_table_name ||
        ' (' || p_col_specs || ')';
END;
/
```

```
-- Call the procedure

BEGIN
    create_table('EMPLOYEE_NAMES',
        'id NUMBER(4) PRIMARY KEY, name VARCHAR2(40)');
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DDL 문을 사용하는 동적 SQL

코드 예제는 테이블 이름과 열 정의(사양)를 파라미터로 사용하는 `create_table` 프로시저를 생성하는 방법을 보여줍니다.

프로시저 호출은 다음 두 개의 열을 사용하여 `EMPLOYEE_NAMES`라는 테이블을 생성하는 방법을 보여줍니다.

- `NUMBER` 데이터 유형을 Primary Key로 사용하는 ID 열
- 최대 40자까지 허용되는, 사원 이름에 대한 name 열

명령문이 동적으로 생성되는지 아니면 리터럴 문자열로 지정되는지에 관계없이 위 슬라이드에 표시된 구문을 사용하여 DDL 문을 실행할 수 있습니다. 다음 예제와 같이 PL/SQL 문자열 변수에 저장되어 있는 명령문을 생성하고 실행할 수 있습니다.

```
CREATE OR REPLACE PROCEDURE add_col(p_table_name VARCHAR2,
                                    p_col_spec VARCHAR2) IS
    v_stmt VARCHAR2(100) := 'ALTER TABLE ' || p_table_name ||
        ' ADD ' || p_col_spec;
BEGIN
    EXECUTE IMMEDIATE v_stmt;
END;
/
```

테이블에 새 열을 추가하려면 다음을 입력하십시오.

```
EXECUTE add_col('employee_names', 'salary number(8,2)')
```

DML 문을 사용하는 동적 SQL

```
-- Delete rows from any table:
CREATE FUNCTION del_rows(p_table_name VARCHAR2)
RETURN NUMBER IS
BEGIN
    EXECUTE IMMEDIATE 'DELETE FROM ' || p_table_name;
    RETURN SQL%ROWCOUNT;
END;
/
BEGIN DBMS_OUTPUT.PUT_LINE(
    del_rows('EMPLOYEE_NAMES') || ' rows deleted.');
END;
/
```

```
-- Insert a row into a table with two columns:
CREATE PROCEDURE add_row(p_table_name VARCHAR2,
    p_id NUMBER, p_name VARCHAR2) IS
BEGIN
    EXECUTE IMMEDIATE 'INSERT INTO ' || p_table_name ||
        ' VALUES (:1, :2)' USING p_id, p_name;
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 문을 사용하는 동적 SQL

슬라이드의 첫번째 코드 예제에서는 방법 1을 사용하는 동적 SQL 문, 즉 호스트 변수가 없는 query가 아닌 명령문을 정의합니다. 위 슬라이드 예제는 다음을 보여줍니다.

- `del_rows` 함수는 지정된 테이블에서 행을 삭제한 다음 암시적 SQL 커서의 `%ROWCOUNT` 속성을 사용하여 삭제된 행 수를 반환합니다. 함수 실행에 대해서는 다음 페이지의 함수 생성 예제를 살펴보십시오.
- `add_row` 프로시저는 `USING` 절을 사용하는 동적 SQL 문에 입력 값을 제공하는 방법을 보여줍니다. 바인드 변수 이름 `:1`과 `:2`는 중요하지 않습니다. 단, `USING` 절에 사용된 파라미터 이름(`p_id` 및 `p_name`)의 순서는 각 변수가 나타난 순서 그대로 위치별로 바인드 변수와 연관됩니다. 따라서 PL/SQL 파라미터 `p_id`는 `:1` 위치 표시자에 할당되고, `p_name` 파라미터는 `:2` 위치 표시자에 할당됩니다. 위치 표시자 또는 바인드 변수 이름은 문자로 지정할 수 있지만 이름 앞에 반드시 콜론이 있어야 합니다.

참고: `EXECUTE IMMEDIATE` 문은 구문 분석을 준비한 다음 동적 SQL 문을 즉시 실행합니다. 동적 SQL 문은 항상 구문 분석이 수행됩니다.

또한 예제에서는 `COMMIT` 작업이 수행되지 않았습니다. 따라서 `ROLLBACK` 문을 사용하여 작업을 취소할 수 있습니다.

단일 행 query를 사용하는 동적 SQL: 예제

```

CREATE FUNCTION get_emp( p_emp_id NUMBER )
RETURN employees%ROWTYPE IS
    v_stmt VARCHAR2(200);
    v_emprec employees%ROWTYPE;
BEGIN
    v_stmt := 'SELECT * FROM employees ' ||
              'WHERE employee_id = :p_emp_id';
    EXECUTE IMMEDIATE v_stmt INTO v_emprec USING p_emp_id;
    RETURN v_emprec;
END;
/
DECLARE
    v_emprec employees%ROWTYPE := get_emp(100);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Emp: ' || v_emprec.last_name);
END;
/

```

```

FUNCTION get_emp(p_emp_id Compiled.
anonymous block completed
Emp: King

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단일 행 query를 사용하는 동적 SQL

슬라이드의 코드 예제에서는 단일 행 query, 즉 선택 리스트 항목 및 입력 호스트 변수의 수를 알 수 있는 query와 함께 방법 3을 사용하여 동적 SQL 문을 정의합니다.

단일 행 query 예제는 INTO 절에 지정된 변수로 EMPLOYEES 레코드를 읽어 들이는 get_emp 함수를 보여줍니다. 또한 WHERE 절에 입력 값을 제공하는 방법을 보여줍니다.

익명 블록은 get_emp 함수를 실행하는 데 사용되며 로컬 EMPLOYEES 레코드 변수에 결과를 반환합니다.

입력 파라미터 값에 따라 다른 WHERE 절을 제공하도록 예제를 보강하면 동적 SQL 처리에 보다 적합해질 수 있습니다.

참고:

- REF CURSORS를 사용한 "여러 행 query가 포함된 동적 SQL: 예제"는 /home/oracle/labs/plpu/demo 폴더의 demo_06_13_a를 참조하십시오.
- REF CURSORS 사용 예제는 /home/oracle/labs/plpu/demo 폴더의 demo_06_13_b를 참조하십시오.
- REF CURSORS에 대해서는 *Oracle Database 11g: Advanced PL/SQL 3-day* 과정에서 다룹니다.

PL/SQL 익명 블록 동적 실행

```

CREATE FUNCTION annual_sal( p_emp_id NUMBER)
RETURN NUMBER IS
    v_plsql varchar2(200) :=
        'DECLARE ' ||
        '    rec_emp employees%ROWTYPE; ' ||
        'BEGIN ' ||
        '    rec_emp := get_emp(:empid); ' ||
        '    :res := rec_emp.salary * 12; ' ||
        'END;';
    v_result NUMBER;
BEGIN
    EXECUTE IMMEDIATE v_plsql
        USING IN p_emp_id, OUT v_result;
    RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 블록 동적 실행

annual_sal 함수는 익명 PL/SQL 블록을 동적으로 생성합니다. PL/SQL 블록에는 다음에 대한 바인드 변수가 포함되어 있습니다.

- :empid 위치 표시자를 사용하는 사원 ID 입력
- :res 위치 표시자를 사용하여 사원의 연봉을 계산하는 출력 결과

참고: 이 예제에서는 EXECUTE IMMEDIATE 문의 USING 절에 있는 OUT 결과 구문을 사용하여 PL/SQL 블록에서 계산한 결과를 얻는 방법을 보여줍니다. 프로시저 출력 변수와 함수 반환 값은 동적으로 실행되는 익명 PL/SQL 블록에서 유사한 방법으로 얻을 수 있습니다. 슬라이드 예제의 출력은 다음과 같습니다.

```

FUNCTION annual_sal(emp_id Compiled.
anonymous block completed
316800

```

Native Dynamic SQL을 사용하여 PL/SQL 코드 컴파일

ALTER 문을 사용하여 PL/SQL 코드 컴파일:

- ALTER PROCEDURE name COMPILE
- ALTER FUNCTION name COMPILE
- ALTER PACKAGE name COMPILE SPECIFICATION
- ALTER PACKAGE name COMPILE BODY

```
CREATE PROCEDURE compile_plsql(p_name VARCHAR2,  
    p_plsql_type VARCHAR2, p_options VARCHAR2 := NULL) IS  
    v_stmt varchar2(200) := 'ALTER '|| p_plsql_type ||  
                           ' '|| p_name || ' COMPILE';  
  
BEGIN  
    IF p_options IS NOT NULL THEN  
        v_stmt := v_stmt || ' ' || p_options;  
    END IF;  
    EXECUTE IMMEDIATE v_stmt;  
END;  
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Native Dynamic SQL을 사용하여 PL/SQL 코드 컴파일

예제의 compile_plsql 프로시저는 ALTER DDL 문을 사용하여 여러 가지 PL/SQL 코드를 컴파일하는 데 사용할 수 있습니다. 다음 항목을 컴파일하는 네 가지 기본 ALTER 문 형식이 표시되어 있습니다.

- 프로시저
- 함수
- Package Spec
- Package Body

참고: ALTER PACKAGE 문에서 SPECIFICATION이나 BODY 키워드를 생략하면 Spec과 Body가 모두 컴파일됩니다.

다음은 위 슬라이드에 표시된 네 가지 경우 각각에 대한 프로시저를 호출하는 예제입니다.

```
EXEC compile_plsql ('list_employees', 'procedure')  
EXEC compile_plsql ('get_emp', 'function')  
EXEC compile_plsql ('mypack', 'package', 'specification')  
EXEC compile_plsql ('mypack', 'package', 'body')
```

다음은 get_emp 함수에 대해 debug를 활성화하여 컴파일하는 예제입니다.

```
EXEC compile_plsql ('get_emp', 'function', 'debug')
```

단원 내용

- NDS(Native Dynamic SQL) 사용
- DBMS_SQL 패키지 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DBMS_SQL 패키지 사용

- DBMS_SQL 패키지는 내장 프로시저에서 동적 SQL을 작성하고 DDL 문의 구문을 분석하는 데 사용됩니다.
- 입력 또는 출력 변수의 수를 알 수 없는 동적 SQL 문(방법 4라고도 함)은 DBMS_SQL 패키지를 사용하여 실행해야 합니다.
- 대부분의 경우 NDS는 DBMS_SQL보다 쉽게 사용하고 수행할 수 있지만 방법 4를 처리할 때는 예외입니다.
- 예를 들어 다음과 같은 경우에는 DBMS_SQL 패키지를 사용해야 합니다.
 - 컴파일 시 SELECT 리스트를 모르는 경우
 - SELECT 문이 반환할 열 수 또는 해당 데이터 유형을 모르는 경우

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_SQL 패키지 사용

DBMS_SQL을 사용하면 PL/SQL에서 DDL 문을 실행(예: DROP TABLE 문 실행)하는 것과 같이 동적 SQL을 사용하는 내장 프로시저와 익명 PL/SQL 블록을 작성할 수 있습니다. 이 패키지에서 제공하는 작업은 패키지 소유자 SYS가 아닌 현재 유저에 의해 수행됩니다.

방법 4: 방법 4는 동적 SQL 문에서 query에 대해 선택된 열 수 또는 바인드 변수의 수를 런타임까지 알 수 없는 상황을 나타냅니다. 이 경우에는 DBMS_SQL 패키지를 사용해야 합니다. 동적 SQL을 생성하는 경우 DBMS_SQL 제공 패키지를 사용하거나(방법 4 상황을 처리할 때) Native Dynamic SQL을 사용할 수 있습니다. Oracle Database 11g 이전에는 각 방식마다 기능적인 제한이 있었습니다. Oracle Database 11g에서는 이러한 방식을 보다 완전하게 만들기 위해 두 방식 모두에 기능성이 추가되었습니다.

PL/SQL에서 동적 SQL을 실행하는 기능은 Oracle Database 10g에서 몇 가지 제한이 있었습니다. DBMS_SQL은 방법 4 시나리오에 필요했지만 모든 범위의 데이터 유형을 처리할 수 없었으며 해당 커서 표현을 클라이언트가 데이터베이스에 사용할 수 없었습니다. Native Dynamic SQL은 방법 4가 아닌 시나리오에서 더욱 편리했지만 32KB를 초과하는 명령문을 지원하지 않았습니다. Oracle Database 11g는 PL/SQL에서 동적 SQL이 완전하게 지원되도록 이러한 제한과 기타 제한을 제거합니다.

DBMS_SQL 패키지 서브 프로그램 사용

패키지 프로시저 및 함수의 예:

- **OPEN_CURSOR**
- **PARSE**
- **BIND_VARIABLE**
- **EXECUTE**
- **FETCH_ROWS**
- **CLOSE_CURSOR**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_SQL 패키지 서브 프로그램 사용

DBMS_SQL 패키지는 동적 SQL을 실행하는 다음 서브 프로그램을 제공합니다.

- OPEN_CURSOR - 새 커서를 열고 커서 ID 번호를 반환합니다.
- PARSE - SQL 문을 구문 분석합니다. 모든 SQL 문은 PARSE 프로시저를 호출하여 구문 분석되어야 합니다. 명령문을 구문 분석하면 해당 명령문의 구문을 검사하고 프로그램에 있는 커서와 연결합니다. 모든 DML 또는 DDL 문을 구문 분석할 수 있습니다. 구문이 분석되면 DDL 문이 즉시 실행됩니다.
- BIND_VARIABLE - 제공된 값을 구문 분석 중인 명령문에서 이름으로 식별된 바인드 변수에 바인드합니다. 명령문에 바인드 변수가 없으면 이 서브 프로그램이 필요하지 않습니다.
- EXECUTE - SQL 문을 실행한 다음 처리된 행 수를 반환합니다.
- FETCH_ROWS - 다음 query 행을 읽어 들입니다(여러 행 루프에 사용).
- CLOSE_CURSOR - 지정된 커서를 닫습니다.

참고: DBMS_SQL 패키지를 사용하여 DDL 문을 실행하면 deadlock이 발생할 수 있습니다. 이 패키지가 아직 사용 중인 프로시저를 삭제하는 데 사용되고 있기 때문일 수 있습니다.

DBMS_SQL 패키지 서브 프로그램 사용(계속)

PARSE 프로시저 파라미터

PARSE 프로시저의 LANGUAGE_FLAG 파라미터는 특정 오라클 데이터베이스 버전과 연관된 동작을 사용하여 오라클에서 SQL 문을 처리하는 방법을 결정합니다. 이 파라미터에 NATIVE (또는 1)를 사용하면 프로그램이 연결된 데이터베이스와 연관된 기본 동작을 사용하도록 지정됩니다.

LANGUAGE_FLAG 파라미터가 V6(또는 0)으로 설정되어 있으면 버전 6 동작을 지정합니다. LANGUAGE_FLAG 파라미터가 V7(또는 2)로 설정되어 있으면 오라클 데이터베이스 버전 7 동작을 지정합니다.

참고: 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

DML 문과 함께 DBMS_SQL 사용: 행 삭제

```

CREATE OR REPLACE FUNCTION delete_all_rows
  (p_table_name VARCHAR2) RETURN NUMBER IS
    v_cur_id INTEGER;
    v_rows_del NUMBER;
BEGIN
  v_cur_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQLPARSE(v_cur_id, 'DELETE FROM ' ||
p_table_name, DBMS_SQL.NATIVE);
  v_rows_del := DBMS_SQL.EXECUTE (v_cur_id);
  DBMS_SQL.CLOSE_CURSOR(v_cur_id);
  RETURN v_rows_del;
END;
/

```

```

CREATE TABLE temp_emp AS SELECT * FROM employees;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Rows Deleted: ' ||
delete_all_rows('temp_emp'));
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 문과 함께 DBMS_SQL 사용

슬라이드에서는 테이블 이름이 `delete_all_rows` 함수에 전달됩니다. 이 함수는 동적 SQL을 사용하여 지정된 테이블에서 행을 삭제한 다음 명령문을 성공적으로 실행한 이후로 삭제된 행 수를 반환합니다.

DML 문을 동적으로 처리하려면 다음 단계를 수행하십시오.

1. OPEN_CURSOR를 사용하여 SQL 문을 처리할 영역을 메모리에 설정합니다.
2. PARSE를 사용하여 SQL 문의 유효성을 확인합니다.
3. EXECUTE 함수를 사용하여 SQL 문을 실행합니다. 이 함수는 처리된 행 수를 반환합니다.
4. CLOSE_CURSOR를 사용하여 커서를 닫습니다.

DDL 문 실행 단계도 이와 유사합니다. 그러나 DDL 문은 PARSE가 성공적으로 실행되면, 즉 명령문의 구문과 의미가 올바르면 즉시 실행되기 때문에 단계 3은 선택 사항입니다. DDL 문과 함께 EXECUTE 함수를 사용할 경우 아무 작업도 수행되지 않으며, DDL 문이 행을 처리하지 않기 때문에 처리된 행 수에 대해 값 0을 반환합니다.

```

CREATE TABLE succeeded.
anonymous block completed
Rows Deleted: 107

DROP TABLE temp_emp succeeded.

```

Parameterized DML 문과 함께 DBMS_SQL 사용

```

CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
    p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
    v_cur_id INTEGER;
    v_stmt VARCHAR2(200);
    v_rows_added NUMBER;
BEGIN
    v_stmt := 'INSERT INTO '|| p_table_name || 
        ' VALUES (:cid, :cname, :rid)';
    v_cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQLPARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
    v_rows_added := DBMS_SQL.EXECUTE(v_cur_id);
    DBMS_SQL.CLOSE_CURSOR(v_cur_id);
    DBMS_OUTPUT.PUT_LINE(v_rows_added||' row added');
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Parameterized DML 문과 함께 DBMS_SQL 사용

위 슬라이드 예제는 지정된 테이블에 행을 삽입하는 DML 작업을 수행하며, SQL 문에 있는 바인드 변수와 값을 연관시키는 데 필요한 추가 단계를 보여줍니다. 예를 들어 슬라이드에 표시된 프로시저 호출은 다음과 같습니다.

```
EXECUTE insert_row('countries', 'LB', 'Lebanon', 4)
```

명령문 구문이 분석된 후에는 DBMS_SQL.BIND_VARIABLE 프로시저를 호출하여 명령문에 있는 각 바인드 변수에 대한 값을 할당해야 합니다. 값 바인딩은 코드를 실행하기 전에 수행해야 합니다. SELECT 문을 동적으로 처리하려면 커서를 열고 나서 닫기 전에 다음 단계를 수행하십시오.

1. 선택된 각 열에 대해 DBMS_SQL.DEFINE_COLUMN을 실행합니다.
2. query의 각 바인드 변수에 대해 DBMS_SQL.BIND_VARIABLE을 실행합니다.
3. 각 행마다 다음 작업을 수행합니다.
 - a. DBMS_SQL.FETCH_ROWS를 실행하여 행을 읽어 들인 다음 패치(fetch)된 행 수를 반환합니다. 값 0이 반환될 경우 추가 처리를 중단합니다.
 - b. DBMS_SQL.COLUMN_VALUE를 실행하여 선택한 각 열 값을 처리할 각 PL/SQL 변수로 읽어 들입니다.

이와 같은 코딩 프로세스의 경우 복잡하지는 않지만, Native Dynamic SQL 접근 방법을 사용하는 경우와 비교했을 때 작성하는 데 많은 시간이 걸리고 오류가 발생하기 쉽습니다.

퀴즈

동적 SQL 문의 전체 텍스트를 런타임까지 알지 못할 수도 있습니다.
따라서 구문은 컴파일 시 검사되지 않고 런타임에 검사됩니다.

1. 맞습니다.
2. 틀립니다.



Copyright © 2009, Oracle. All rights reserved.

정답: 1

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- SQL 문의 실행 흐름 설명
- NDS(Native Dynamic SQL)를 사용하여 SQL 문을 동적으로 생성 및 실행
- NDS 대신 DBMS_SQL 패키지를 사용하여 SQL 문을 동적으로 생성하고 실행해야 할 상황 식별

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 SQL 문을 동적으로 생성한 다음 Native Dynamic SQL 문을 사용하여 실행하는 방법에 대해 설명했습니다. SQL 및 PL/SQL 코드를 동적으로 실행하면 PL/SQL의 기능이 query 및 트랜잭션 작업 이상으로 확장됩니다. 이전 데이터베이스 버전의 경우 DBMS_SQL 패키지를 사용하여 유사한 결과를 얻을 수 있었습니다.

연습 6 개요: Native Dynamic SQL 사용

이 연습에서는 다음 내용을 다룹니다.

- Native Dynamic SQL을 사용하여 테이블을 생성 또는 삭제하고 테이블에서 행을 수정, 삭제 및 채우는 패키지 생성
- 스키마에서 PL/SQL 코드를 컴파일하는 패키지 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 6: 개요

이 연습에서는 다음 작업을 수행하는 코드를 작성합니다.

- Native Dynamic SQL을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 수정, 삭제 및 채우는 패키지를 생성합니다.
- 스키마에서 PL/SQL 코드(모든 PL/SQL 코드 또는 USER_OBJECTS 테이블에서 상태가 INVALID인 코드만)를 컴파일하는 패키지를 생성합니다.

PL/SQL 코드 설계 고려 사항

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 표준 상수 및 예외 생성
- 로컬 서브 프로그램 작성 및 호출
- 서브 프로그램의 런타임 권한 제어
- 독립 트랜잭션 수행
- NOCOPY 힌트를 사용하여 파라미터를 참조로 전달
- 최적화를 위해 PARALLEL_ENABLE 힌트 사용
- 세션간 PL/SQL 함수 결과 캐시 사용
- 함수와 함께 DETERMINISTIC 절 사용
- DML과 함께 RETURNING 절 및 대량 바인드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 Package Spec을 사용하여 상수 값과 예외의 이름을 표준화하는 방법에 대해 설명하며, PL/SQL 블록에서 로컬로 사용할 수 있도록 PL/SQL 블록의 선언 섹션에 서브 프로그램을 생성하는 방법에 대해 설명합니다. PL/SQL 코드의 런타임 권한을 관리하는 방법과 서브 프로그램에 AUTONOMOUS TRANSACTION 지시어를 사용하여 독립 트랜잭션을 생성하는 방법을 보여주기 위해 AUTHID 컴파일러 지시어를 다룹니다.

이 단원에서는 또한 단일 SQL 문을 사용한 대량 바인드 작업, RETURNING 절, NOCOPY 및 PARALLEL_ENABLE 힌트와 같이 PL/SQL 응용 프로그램에만 적용할 수 있는 몇 가지 성능 고려 사항에 대해서도 설명합니다.

단원 내용

- 상수 및 예외 표준화, 로컬 서브 프로그램 사용, 서브 프로그램의 런타임 권한 제어, 독립 트랜잭션 수행
- NOCOPY 및 PARALLEL ENABLE 힌트, 세션간 PL/SQL 함수 결과 캐시, DETERMINISTIC 절 사용
- DML과 함께 RETURNING 절 및 대량 바인드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

상수 및 예외 표준화

상수와 예외는 일반적으로 본문이 없는 패키지, 즉 Package Spec을 사용하여 구현됩니다.

- **표준화의 이점:**
 - 일관성 있는 프로그램 개발
 - 보다 높은 수준으로 코드 재사용 증진
 - 간편한 코드 유지 관리
 - 전체 응용 프로그램에서 회사 표준 구현
- **표준화 대상:**
 - 예외 이름
 - 상수 정의



Copyright © 2009, Oracle. All rights reserved.

상수 및 예외 표준화

여러 명의 개발자가 한 응용 프로그램에 각각 따로 예외를 작성할 경우 오류 상황에 대한 처리가 일관되지 않을 수 있습니다. 정해진 표준을 따르지 않으면 동일한 오류를 처리하는 데 서로 다른 접근 방법을 사용하거나 유저에게 혼동을 줄 수 있는 상반되는 오류 메시지가 표시될 수 있기 때문에 혼란스러운 상황이 발생할 수 있습니다. 이러한 상황을 해결할 수 있는 방법은 다음과 같습니다.

- 전체 응용 프로그램에서 일관된 오류 처리 접근 방법을 사용하는 회사 표준 구현
- 응용 프로그램에서 일관성을 제공하는 미리 정의된 일반 예외 처리기 생성
- 일관된 오류 메시지를 생성하는 프로그램 작성 및 호출

훌륭한 프로그래밍 환경은 모두 이름 지정 및 코딩 표준을 지원합니다. PL/SQL에서 이름 지정 및 코딩 표준 구현은 응용 프로그램 도메인에서 일반적으로 사용되는 상수와 예외부터 시작하는 것이 좋습니다.

PL/SQL Package Spec에 선언되는 식별자는 모두 공용(public)이므로 PL/SQL Package Spec 생성자는 표준화를 지원하는 뛰어난 구성 요소입니다. 이러한 생성자는 Package Spec에 대해 EXECUTE 권한을 가진 패키지 및 전체 코드의 소유자가 개발한 서브 프로그램에서 볼 수 있습니다.

예외 표준화

응용 프로그램에 사용될 명명된 예외 및 프로그래머 정의 예외가 모두 포함된 표준화된 오류 처리 패키지를 생성합니다.

```
CREATE OR REPLACE PACKAGE error_pkg IS
    e_fk_err          EXCEPTION;
    e_seq_nbr_err    EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_fk_err, -2292);
    PRAGMA EXCEPTION_INIT (e_seq_nbr_err, -2277);
    ...
END error_pkg;
/
```

Copyright © 2009, Oracle. All rights reserved.

예외 표준화

슬라이드의 예제에서 error_pkg 패키지는 표준화된 예외 패키지로서, 프로그래머가 정의한 일련의 예외 식별자를 선언합니다. 오라클 데이터베이스의 미리 정의된 예외에는 대부분 식별 이름이 없기 때문에 슬라이드에 표시된 예제 패키지는 PRAGMA EXCEPTION_INIT 지시어를 사용하여 선택한 예외 이름을 오라클 데이터베이스 오류 번호와 연관시킵니다. 그러면 다음 예제와 같이 응용 프로그램에서 표준 방식으로 예외를 참조할 수 있습니다.

```
BEGIN
    DELETE FROM departments
    WHERE department_id = deptno;
    ...
EXCEPTION
    WHEN error_pkg.e_fk_err THEN
        ...
    WHEN OTHERS THEN
        ...
END;
/
```

예외 처리 표준화

다음을 수행하는 일반적인 예외 처리에 대한 서브 프로그램을 작성하십시오.

- 예외의 SQLCODE 및 SQLERRM 값을 기준으로 오류 표시
- 코드에 포함된 파라미터를 사용하여 다음 사항을 식별함으로써 쉽게 런타임 오류 추적
 - 오류가 발생한 프로시저
 - 오류 위치(행 번호)
 - 스택 trace 기능을 사용하는 RAISE_APPLICATION_ERROR (세번째 인수가 TRUE로 설정됨)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

예외 처리 표준화

표준화된 예외 처리는 독립형 서브 프로그램 또는 표준 예외를 정의하는 패키지에 추가된 서브 프로그램으로 구현할 수 있습니다. 다음을 사용하여 패키지를 생성해 보십시오.

- 응용 프로그램에 사용될 명명된 예외 전체
- 응용 프로그램에 사용되는 명명되지 않은 프로그래머 정의 예외 전체. 오류 번호 -20000에서 -20999까지입니다.
- 패키지 예외를 기준으로 RAISE_APPLICATION_ERROR를 호출하는 프로그램
- SQLCODE 및 SQLERRM의 값을 기준으로 오류를 표시하는 프로그램
- 오류 로그 테이블과 같은 추가 객체 및 이 테이블에 액세스하는 추가 프로그램

일반적인 방법은 오류가 발생한 프로시저 이름과 오류 위치를 식별하는 파라미터를 사용하는 것입니다. 그러면 런타임 오류를 훨씬 더 쉽게 추적할 수 있습니다. 또 다른 방법은 RAISE_APPLICATION_ERROR 내장 프로시저를 사용하여 오류를 발생시킨 호출 시퀀스 추적에 사용할 수 있는 예외 스택 trace를 보존하는 것입니다. 이를 위해서는 세번째 선택적 인수를 TRUE로 설정하십시오. 예를 들면 다음과 같습니다.

```
RAISE_APPLICATION_ERROR(-20001, 'My first error', TRUE);
```

이것은 둘 이상의 예외가 이와 같은 방식으로 발생할 경우에 의미가 있습니다.

상수 표준화

값을 변경하면 안되는 로컬 변수를 사용하는 프로그램의 경우:

- **변수를 상수로 변환하여 유지 관리와 디버깅 작업을 줄입니다.**
- **하나의 중앙 Package Spec을 생성한 후 이 안에 모든 상수를 배치합니다.**

```
CREATE OR REPLACE PACKAGE constant_pkg IS
    c_order_received CONSTANT VARCHAR(2) := 'OR';
    c_order_shipped CONSTANT VARCHAR(2) := 'OS';
    c_min_sal CONSTANT NUMBER(3) := 900;
END constant_pkg;
```

Copyright © 2009, Oracle. All rights reserved.

상수 표준화

원칙적으로 변수 값은 변경되지만 상수 값은 변경할 수 없습니다. 값을 변경하면 안되거나 값이 변경되지 않는 로컬 변수를 사용하는 프로그램의 경우 변수를 상수로 변환하십시오. 이렇게 하면 코드의 유지 관리 및 디버깅 작업을 줄일 수 있습니다.

모든 상수가 포함된 단일 공유 패키지를 생성해 보십시오. 그러면 상수를 훨씬 쉽게 유지 관리하고 변경할 수 있습니다. 이 프로시저나 패키지는 성능 향상을 위해 시스템을 시작할 때 로드할 수 있습니다.

슬라이드의 예제는 몇 가지 상수를 포함하는 constant_pkg 패키지를 보여줍니다. 필요에 따라 응용 프로그램에서 패키지 상수를 참조하십시오. 예를 들면 다음과 같습니다.

```
BEGIN
    UPDATE employees
        SET salary = salary + 200
    WHERE salary <= constant_pkg.c_min_sal;
END;
/
```

로컬 서브 프로그램

로컬 서브 프로그램은 서브 프로그램의 선언 섹션 끝에 정의된 PROCEDURE 또는 FUNCTION입니다.

```
CREATE PROCEDURE employee_sal(p_id NUMBER) IS
    v_emp employees%ROWTYPE;
    FUNCTION tax(p_salary VARCHAR2) RETURN NUMBER IS
    BEGIN
        RETURN p_salary * 0.825;
    END tax;
BEGIN
    SELECT * INTO v_emp
    FROM EMPLOYEES WHERE employee_id = p_id;
    DBMS_OUTPUT.PUT_LINE('Tax: ' || tax(v_emp.salary));
END;
/
EXECUTE employee_sal(100)
```

```
PROCEDURE employee_sal(p_id Compiled.
anonymous block completed
Tax: 19800
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

로컬 서브 프로그램

로컬 서브 프로그램은 하향식 설계를 유도할 수 있습니다. 이 프로그램은 중복되는 코드를 제거함으로써 모듈 크기를 줄여줍니다. 이것이 로컬 서브 프로그램을 생성하는 주요 이유 중 하나입니다. 모듈에 동일한 루틴이 여러 번 필요하지만 이 모듈에만 해당 루틴이 필요한 경우 이 루틴을 로컬 서브 프로그램으로 정의하십시오.

명명된 PL/SQL 블록은 PL/SQL 프로그램, 프로시저, 함수 또는 익명 블록의 선언 섹션에서 정의할 수 있습니다. 단, Declaration 섹션 끝에 선언되어야 합니다. 로컬 서브 프로그램의 특징은

다음과 같습니다.

- 해당 서브 프로그램이 정의되어 있는 블록에서만 액세스할 수 있습니다.
- 해당 서브 프로그램을 포함하는 블록의 일부로 컴파일됩니다.

로컬 서브 프로그램의 이점은 다음과 같습니다.

- 반복되는 코드를 줄일 수 있습니다.
- 코드 가독성이 향상되고 유지 관리가 쉽습니다.
- 유지 관리할 프로그램이 두 개가 아닌 한 개이기 때문에 관리 작업을 줄일 수 있습니다.

개념은 간단합니다. 위 슬라이드에 표시된 예제는 사원 급여에 대한 소득세를 계산하는 기본 예제와 함께 로컬 서브 프로그램의 개념을 보여줍니다.

정의자 권한과 호출자 권한 비교

정의자 권한:

- 프로그램은 생성자의 권한으로 실행됩니다.
- 유저에게는 프로시저에서 액세스하는 기본 객체에 대한 권한이 필요 없습니다.
프로시저 실행 권한만 필요합니다.

호출자 권한:

- 프로그램은 호출자의 권한으로 실행됩니다.
- 유저에게는 프로시저에서 액세스하는 기본 객체에 대한 권한이 필요합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정의자 권한과 호출자 권한 비교

정의자 권한 모델

기본적으로 모든 프로그램이 서브 프로그램을 생성한 유저의 권한으로 실행되었습니다. 이것을 정의자 권한 모델이라고 하며, 다음과 같은 특징이 있습니다.

- 프로그램 호출자에게 프로시저 실행 권한이 부여되지만, 프로시저에서 액세스하는 기본 객체에 대한 권한은 부여되지 않습니다.
- 소유자에게는 프로시저에서 참조하는 객체에 필요한 모든 객체 권한이 있어야 합니다.

예를 들어 유저 Scott이 나중에 Sarah에 의해 호출되는 PL/SQL 서브 프로그램 get_employees를 생성할 경우 get_employees 프로시저는 정의자 Scott의 권한으로 실행됩니다.

호출자 권한 모델

Oracle8i에서 처음 사용된 호출자 권한 모델에서 프로그램은 호출자 권한으로 실행됩니다. 호출자 권한으로 실행되는 프로시저의 유저에게는 프로시저에서 참조하는 기본 객체에 대한 권한이 필요합니다.

예를 들어 Scott의 PL/SQL 서브 프로그램 get_employees를 Sarah가 호출할 경우 get_employees 프로시저는 호출자 Sarah의 권한으로 실행됩니다.

호출자 권한 지정: AUTHID를 CURRENT_USER로 설정

```
CREATE OR REPLACE PROCEDURE add_dept(
    p_id NUMBER, p_name VARCHAR2) AUTHID CURRENT_USER IS
BEGIN
    INSERT INTO departments
    VALUES (p_id, p_name, NULL, NULL);
END;
```

독립형 함수, 프로시저 또는 패키지와 함께 사용할 경우:

- Query, DML, Native Dynamic SQL 및 DBMS_SQL 패키지에 사용되는 이름은 호출자의 스키마에서 분석됩니다.
- 기타 패키지, 함수 및 프로시저 호출은 정의자의 스키마에서 분석됩니다.



Copyright © 2009, Oracle. All rights reserved.

호출자 권한 지정

다음과 같이 서로 다른 PL/SQL 서브 프로그램 생성자에 대해 호출자 권한을 설정할 수 있습니다.

```
CREATE FUNCTION name RETURN type AUTHID CURRENT_USER IS...
CREATE PROCEDURE name AUTHID CURRENT_USER IS...
CREATE PACKAGE name AUTHID CURRENT_USER IS...
CREATE TYPE name AUTHID CURRENT_USER IS OBJECT...
```

기본값은 소유자의 권한으로 서브 프로그램이 실행되도록 지정하는 AUTHID DEFINER입니다. 가장 많이 제공되는 DBMS_LOB, DBMS_ROWID 등의 PL/SQL 패키지는 호출자 권한 패키지입니다.

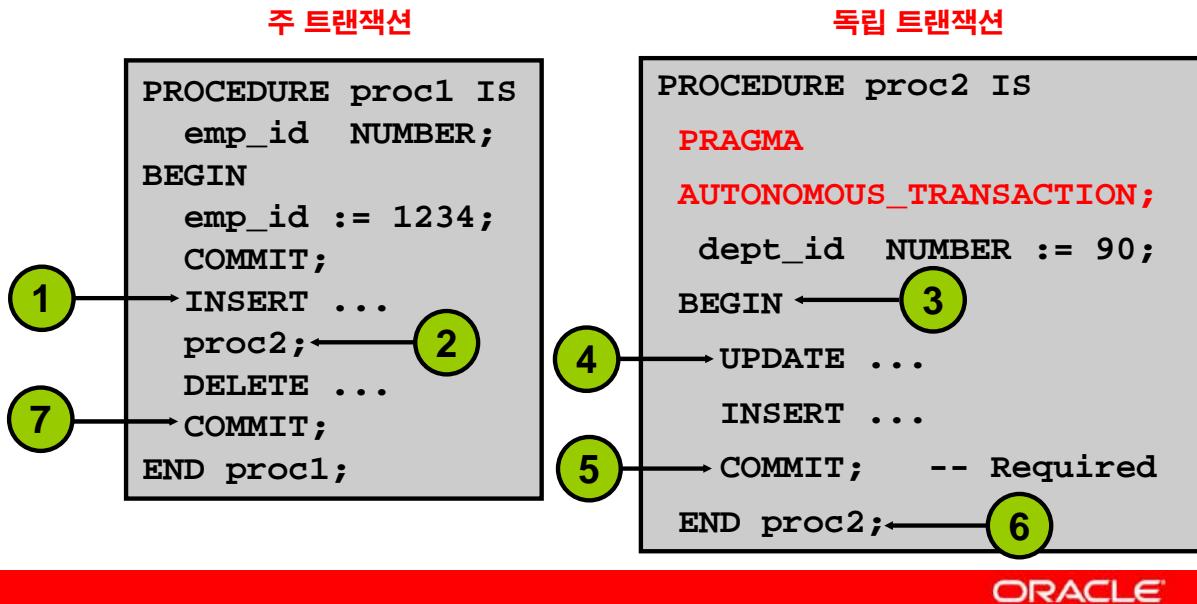
이름 분석

정의자 권한 프로시저의 경우 모든 external 참조는 정의자의 스키마에서 분석됩니다. 호출자 권한 프로시저의 경우 external 참조에 대한 분석은 해당 참조가 나타나는 명령문의 종류에 따라 달라집니다.

- Query, DML(데이터 조작어) 문, 동적 SQL 및 DBMS_SQL에 사용되는 이름은 호출자의 스키마에서 분석됩니다.
- 패키지, 함수 및 프로시저 호출 등의 다른 명령문은 모두 정의자의 스키마에서 분석됩니다.

독립 트랜잭션(Autonomous Transaction)

- 다른 주 트랜잭션에 의해 시작되는 독립적인 트랜잭션입니다.
- PRAGMA AUTONOMOUS_TRANSACTION으로 지정됩니다.



Copyright © 2009, Oracle. All rights reserved.

독립 트랜잭션(Autonomous Transaction)

트랜잭션은 통합된 단위로 완료되거나 실패하는 논리적 단위의 작업을 수행하는 일련의 명령문입니다. 대개 한 트랜잭션에 의해 다른 트랜잭션이 시작되는데, 후자는 시작 트랜잭션의 범위 밖에서 작동해야 합니다. 즉, 기존 트랜잭션에서 중요한 독립 트랜잭션은 시작 트랜잭션의 결과에 영향을 주지 않으면서 변경 사항을 커밋하거나 롤백해야 합니다. 예를 들면, 주식 구매 트랜잭션에서 전체적인 주식 구매가 완료되었는지 여부에 관계없이 고객의 정보를 커밋해야 합니다. 또는 동일한 트랜잭션을 실행하는 동안에는 전체 트랜잭션이 롤백된 경우에도 테이블에 메시지를 기록할 수 있습니다.

Oracle8i부터는 독립 트랜잭션이 추가되어 독립적인 트랜잭션 생성이 가능해졌습니다. AT(Autonomous Transaction, 독립 트랜잭션)는 다른 MT(Main Transaction, 주 트랜잭션)에 의해 시작되는 독립적인 트랜잭션입니다. 위 슬라이드는 AT의 동작을 보여줍니다.

1. 주 트랜잭션이 시작됩니다.
2. 독립 트랜잭션을 시작하기 위해 proc2 프로시저가 호출됩니다.
3. 주 트랜잭션이 일시 중지됩니다.
4. 독립 트랜잭션의 작업이 시작됩니다.
5. 독립 트랜잭션이 커밋 또는 롤백 작업으로 종료됩니다.
6. 주 트랜잭션이 재개됩니다.
7. 주 트랜잭션이 종료됩니다.

독립 트랜잭션의 특징

- 주 트랜잭션과 별개입니다.
- 독립 트랜잭션이 완료될 때까지 호출 트랜잭션을 일시 중지합니다.
- 중첩 트랜잭션이 아닙니다.
- 주 트랜잭션이 롤백되어도 롤백되지 않습니다.
- 커밋 시 다른 트랜잭션에서 변경 사항을 볼 수 있습니다.
- 중첩 또는 익명 PL/SQL 블록 단위가 아닌 개별 서브 프로그램 단위로 시작되고 종료됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

독립 트랜잭션의 특징

독립 트랜잭션은 다음과 같은 특징을 가집니다.

- 트랜잭션 내에서 호출한 경우에도 독립 트랜잭션은 해당 트랜잭션과 별개입니다.
즉, 중첩 트랜잭션이 아닙니다.
- 주 트랜잭션이 롤백되어도 독립 트랜잭션은 롤백되지 않습니다.
- 독립 트랜잭션이 커밋되면 독립 트랜잭션의 변경 사항을 다른 트랜잭션에서 볼 수 있습니다.
- 스택과 유사한 기능을 사용하여 "최상위" 트랜잭션만 지정된 시간에 액세스할 수 있습니다.
완료되면 독립 트랜잭션은 인출되고 호출 중인 트랜잭션이 재개됩니다.
- 자원이 부족한 경우를 제외하고 독립 트랜잭션을 반복적으로 호출할 수 있는 횟수에는 제한이 없습니다.
- 독립 트랜잭션은 명시적으로 커밋 또는 롤백되어야 하며 그렇지 않을 경우 독립 블록에서 반환하려고 하면 오류가 반환됩니다.
- PRAGMA를 사용하여 패키지의 모든 서브 프로그램을 독립으로 표시할 수는 없으며,
개별 루틴만 독립으로 표시할 수 있습니다.
- 중첩 또는 익명 PL/SQL 블록은 독립으로 표시할 수 없습니다.

독립 트랜잭션 사용: 예제

```

CREATE TABLE usage (card_id NUMBER, loc NUMBER)
/
CREATE TABLE txn (acc_id NUMBER, amount NUMBER)
/
CREATE OR REPLACE PROCEDURE log_usage (p_card_id NUMBER, p_loc NUMBER)
IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO usage
    VALUES (p_card_id, p_loc);
    COMMIT;
END log_usage;
/
CREATE OR REPLACE PROCEDURE bank_trans(p_cardnbr NUMBER,p_loc NUMBER) IS
BEGIN
    INSERT INTO txn VALUES (9001, 1000);
    log_usage (p_cardnbr, p_loc);
END bank_trans;
/
EXECUTE bank_trans(50, 2000)

```



Copyright © 2009, Oracle. All rights reserved.

독립 트랜잭션 사용

독립 트랜잭션을 정의하려면 PRAGMA AUTONOMOUS_TRANSACTION을 사용하십시오. PRAGMA는 루틴을 독립으로 표시하도록 PL/SQL 컴파일러에 지시합니다. 이 컨텍스트에서 용어 "루틴"에는 최상위 레벨의(중첩되지 않음) 익명 PL/SQL 블록, 로컬, 독립형 및 패키지 함수와 프로시저, SQL 객체 유형의 메소드, 데이터베이스 트리거가 포함됩니다. PRAGMA는 루틴의 선언 섹션의 어느 위치에나 코딩할 수 있습니다. 그러나 Declaration 섹션 맨 위에 배치하는 것이 가장 쉽습니다.

슬라이드의 예제에서는 트랜잭션 성공 여부와 관계없이 은행 카드의 사용처를 추적합니다. 다음은 독립 트랜잭션의 이점입니다.

- 독립 트랜잭션은 시작되면 완전히 독립적입니다. 주 트랜잭션과 잠금, 리소스 또는 커밋 종속성을 공유하지 않으므로 주 트랜잭션이 롤백된 경우에도 이벤트를 기록하고, 재시도 카운터를 높이는 등의 작업을 수행할 수 있습니다.
- 무엇보다도 독립 트랜잭션은 모듈 방식의 재사용 가능한 소프트웨어 구성 요소를 작성하는데 도움이 됩니다. 예를 들어 내장 프로시저는 자체적으로 독립 트랜잭션을 시작하고 완료할 수 있습니다. 호출 중인 응용 프로그램은 프로시저의 독립적인 작업에 대해 알 필요가 없으며 프로시저는 응용 프로그램의 트랜잭션 컨텍스트에 대해 알 필요가 없습니다. 따라서 독립 트랜잭션이 일반 트랜잭션보다 오류 발생 가능성이 낮으며 사용하기 더 쉽습니다.

독립 트랜잭션 사용(계속)

이전 슬라이드 예제에서 TXN 및 USAGE 테이블의 출력은 다음과 같습니다.

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
CREATE TABLE succeeded.
CREATE TABLE succeeded.
PROCEDURE log_usage Compiled.
PROCEDURE bank_trans(p_cardnbr Compiled.
anonymous block completed

```

이전 페이지의 코드를 실행하려면 다음 코드를 입력합니다.

```
EXECUTE bank_trans(50, 2000)
```

다음과 같이 Object Navigator 트리의 Tables 노드에서 Data 탭을 사용하여 TXN 및 USAGE 테이블의 값을 표시합니다.

TXN		
Columns	Data	Constraints
	ACC_ID	AMOUNT
1	9001	1000

USAGE		
Columns	Data	Constraints
	CARD_ID	LOC
1	50	2000

단원 내용

- 상수 및 예외 표준화, 로컬 서브 프로그램 사용, 서브 프로그램의 런타임 권한 제어, 독립 트랜잭션 수행
- NOCOPY 및 PARALLEL ENABLE 힌트, 세션간 PL/SQL 함수 결과 캐시, DETERMINISTIC 절 사용
- DML과 함께 RETURNING 절 및 대량 바인드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

NOCOPY 힌트 사용

- PL/SQL 컴파일러가 OUT 및 IN OUT 파라미터를 값이 아닌 참조로 전달하도록 허용합니다.
- 파라미터를 전달할 때 오버헤드를 줄임으로써 성능을 높입니다.

```

DECLARE
    TYPE      rec_emp_type IS TABLE OF employees%ROWTYPE;
    rec_emp  rec_emp_type;
    PROCEDURE populate(p_tab IN OUT NOCOPY emptabtype)IS
        BEGIN
            . . .
        END;
    BEGIN
        populate(rec_emp);
    END;
/

```



Copyright © 2009, Oracle. All rights reserved.

NOCOPY 힌트 사용

PL/SQL 서브 프로그램은 IN, OUT, IN OUT이라는 세 가지 파라미터 전달 모드를 지원합니다. 기본 사항은 다음과 같습니다.

- IN 파라미터는 참조로 전달됩니다. 즉, IN 실제 파라미터에 대한 포인터가 해당 형식 파라미터에 전달됩니다. 따라서 두 파라미터는 실제 파라미터 값은 보유하고 있는 동일한 메모리 위치를 참조합니다.
- OUT 및 IN OUT 파라미터는 값으로 전달됩니다. 즉, OUT 또는 IN OUT 실제 파라미터의 값이 해당 형식 파라미터에 복사됩니다. 그런 다음 서브 프로그램이 정상적으로 종료되면 OUT 및 IN OUT 형식 파라미터에 지정된 값이 해당하는 실제 파라미터에 복사됩니다.

큰 데이터 구조(예: 컬렉션, 레코드, 객체 유형 instance)를 나타내는 파라미터를 OUT 및 IN OUT 파라미터와 함께 복사하면 실행 속도가 느려지고 메모리 사용량이 늘어납니다. 이러한 오버헤드를 막기 위해 PL/SQL 컴파일러에서 OUT 및 IN OUT 파라미터를 참조로 전달할 수 있도록 NOCOPY 힌트를 지정할 수 있습니다.

위 슬라이드는 NOCOPY 힌트를 사용하여 IN OUT 파라미터를 선언하는 예제를 보여줍니다.

NOCOPY 힌트의 영향

- 처리되지 않은 예외가 발생하면서 서브 프로그램이 종료될 경우:
 - NOCOPY 파라미터에 전달된 실제 파라미터의 값을 신뢰할 수 없습니다.
 - 완료되지 않은 수정 사항은 "롤백"되지 않습니다.
- RPC(원격 프로시저 호출) 프로토콜을 사용하면 파라미터를 값으로만 전달할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

NOCOPY 힌트의 영향

NOCOPY 힌트를 사용하면 성능은 나아지지만 잘 정의된 예외 의미가 손상될 수 있습니다. 이 힌트를 사용하면 예외 처리에 다음과 같은 영향이 있습니다.

- NOCOPY는 지시어가 아니라 힌트이므로 컴파일러는 서브 프로그램에 NOCOPY 파라미터를 값이나 참조로 전달할 수 있습니다. 따라서 처리되지 않은 예외가 발생하면서 서브 프로그램이 종료될 경우 NOCOPY 실제 파라미터의 값을 신뢰할 수 없습니다.
- 처리되지 않은 예외가 발생하면서 서브 프로그램이 종료될 경우 기본적으로 서브 프로그램의 OUT 및 IN OUT 형식 파라미터에 지정된 값은 해당하는 실제 파라미터에 복사되지 않으며 변경 사항은 롤백되는 것으로 나타납니다. 그러나 NOCOPY를 지정하면 형식 파라미터에 지정된 사항이 실제 파라미터에도 즉시 적용됩니다. 따라서 처리되지 않은 예외가 발생하면서 서브 프로그램이 종료될 경우 (완료되지 않았을 수 있는) 변경 사항이 "롤백"되지 않습니다.
- 현재 RPC 프로토콜은 파라미터를 값으로만 전달할 수 있도록 합니다. 따라서 응용 프로그램을 분할할 때 예외 의미가 통지 없이 변경될 수 있습니다. 예를 들어 NOCOPY 파라미터가 있는 로컬 프로시저를 원격 사이트로 이동하는 경우 해당 파라미터는 더 이상 참조로 전달되지 않습니다.

PL/SQL 컴파일러가 NOCOPY 힌트를 무시하는 경우

다음의 경우 NOCOPY 힌트의 영향을 받지 않습니다.

- **다음 조건의 실제 파라미터:**
 - 연관 배열(인덱스화된 테이블)의 요소일 경우
 - 제약 조건이 있을 경우(예: 크기 제한 또는 NOT NULL)
 - 형식 파라미터가 레코드인데, 여기서 한 레코드 또는 두 레코드 모두 %ROWTYPE 또는 %TYPE을 사용하여 선언되었고 레코드의 해당 필드에 대한 제약 조건이 서로 다름
 - 암시적 데이터 유형 변환이 필요할 경우
- **서브 프로그램이 external 또는 원격 프로시저 호출에 포함될 경우**

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 컴파일러가 NOCOPY 힌트를 무시하는 경우

다음 경우에 PL/SQL 컴파일러는 NOCOPY 힌트를 무시하고 값에 의한 파라미터 전달 방법을 사용합니다(오류는 생성되지 않음).

- 실제 파라미터는 연관 배열(인덱스화된 테이블)의 요소입니다. 이 제한 사항이 연관 배열 전체에 적용되는 것은 아닙니다.
- 실제 파라미터에 제약 조건이 있습니다(예: 크기 제한 또는 NOT NULL). 이 제한 사항은 제약 조건이 있는 요소나 속성으로 확장되지 않습니다. 또한 크기가 제한된 문자열에는 적용되지 않습니다.
- 실제 및 형식 파라미터가 레코드인데, %ROWTYPE 또는 %TYPE을 사용하여 하나 또는 두 레코드가 모두 선언되었으며 레코드의 해당 필드에 대한 제약 조건이 다릅니다.
- 실제 및 형식 파라미터가 레코드인데, 여기서 실제 파라미터가 커서 FOR 루프의 인덱스로 선언되었으며(암시적), 레코드의 해당 필드에 대한 제약 조건이 서로 다릅니다.
- 실제 파라미터를 전달하려면 암시적으로 데이터 유형을 변환해야 합니다.
- 서브 프로그램은 external 또는 원격 프로시저 호출에 포함됩니다.

PARALLEL_ENABLE 힌트 사용

- 최적화 힌트로 함수에 사용할 수 있습니다.
- 함수를 Parallelized Query 또는 Parallelized DML 문에 사용할 수 있음을 나타냅니다.

```
CREATE OR REPLACE FUNCTION f2 (p_p1 NUMBER)
  RETURN NUMBER PARALLEL_ENABLE IS
BEGIN
  RETURN p_p1 * 2;
END f2;
```

FUNCTION f2 Compiled.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PARALLEL_ENABLE 힌트 사용

PARALLEL_ENABLE 키워드는 함수 선언 구문에 사용할 수 있으며, 함수를 Parallelized Query 또는 Parallelized DML 문에 사용할 수 있음을 나타내는 최적화 힌트입니다. 오라클의 병렬 실행 기능은 SQL 문 실행 작업을 다중 프로세스로 나눕니다. 병렬로 실행되는 SQL 문에서 함수를 호출하는 경우 이러한 각 프로세스에서 별도의 복사본이 실행될 수 있으며, 각 복사본은 해당 프로세스에 의해 처리되는 일부 행에 대해서만 호출됩니다.

Oracle8i 이전에는 DML 문에서 병렬 최적화는 PRAGMA RESTRICT_REFERENCES 선언에 RNDS, WNDS, RNPS 및 WNPS가 네 개 모두 지정된 것으로 함수가 인식되는지 확인했습니다. 데이터베이스 또는 패키지 변수에 대해 읽기나 쓰기로 표시되지 않은 함수는 병렬로 실행될 수 있었습니다. 다시 말해 CREATE FUNCTION 문을 사용하여 정의된 이러한 함수는 해당 코드가 실제로 충분히 순수한지 여부를 암시적으로 검사했습니다. 따라서 이러한 함수에 PRAGMA를 지정할 수 없는 경우에도 병렬 실행이 발생할 수 있었습니다.

슬라이드의 예제와 같이 PARALLEL_ENABLE 키워드는 함수 선언의 반환 값 유형 뒤에 배치됩니다.

참고: 패키지 변수는 병렬 실행 서버 간에 공유되지 않을 수도 있으므로 함수는 패키지 변수와 같은 세션 상태를 사용하면 안됩니다.

세션간 PL/SQL 함수 결과 캐시 사용

- 서로 다른 파라미터 값으로 결과 캐시된 PL/SQL 함수를 호출할 때마다 해당 파라미터 및 결과가 캐시에 저장됩니다.
- 함수 결과 캐시는 SGA에 저장되므로 응용 프로그램을 실행하는 모든 세션에서 사용할 수 있습니다.
- 같은 파라미터를 사용하는 동일한 함수에 대한 후속 호출은 캐시로부터 결과를 사용합니다.
- 성능 및 확장성이 향상됩니다.
- 이 기능은 거의 변경되지 않는 정보에 종속되며 자주 호출되는 함수에 사용합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

세션간 PL/SQL 함수 결과 캐시

Oracle Database 11g부터는 PL/SQL 세션간 함수 결과 캐싱 메커니즘을 사용할 수 있습니다. 이 캐싱 메커니즘은 PL/SQL 함수의 결과를 SGA(Shared Global Area)에 저장할 수 있는 언어 지원 및 시스템 관리 수단을 제공합니다. SGA는 응용 프로그램을 실행하는 모든 세션에서 사용할 수 있습니다. 캐시 메커니즘은 효율적이면서 사용하기 쉽고 캐시 및 캐시 관리 정책을 따로 설계하고 개발해야 하는 번거로움을 없애줍니다.

서로 다른 파라미터 값으로 결과 캐시된 PL/SQL 함수를 호출할 때마다 해당 파라미터 및 결과가 캐시에 저장됩니다. 이후에는 동일한 파라미터 값으로 같은 함수를 호출할 때 결과를 다시 계산하는 대신 캐시에서 읽어 들입니다. 캐시된 결과를 계산하는 데 사용된 데이터베이스 객체가 갱신될 경우 캐시된 결과는 무효가 되며 다시 계산되어야 합니다.

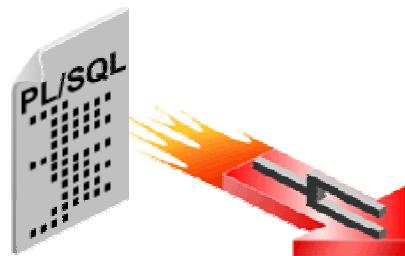
결과 캐싱 기능은 절대(또는 거의) 변경되지 않는 정보에 종속되며 자주 호출되는 함수에 사용하십시오.

참고: 세션간 PL/SQL 함수 결과 캐시에 대한 자세한 내용은 *Oracle Database 11g Advanced PL/SQL* 과정, *Oracle Database 11g SQL and PL/SQL New Features* 과정 또는 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

함수의 결과 캐싱 활성화

다음과 같이 함수를 결과 캐시된 함수로 만들 수 있습니다.

- 다음에 RESULT_CACHE 절을 포함합니다.
 - 함수 선언
 - 함수 정의
- 옵션 RELIES_ON 절을 포함하여 함수 결과가 종속된 테이블 또는 뷰를 지정합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

함수의 결과 캐싱 활성화

PL/SQL 함수의 결과 캐싱을 활성화하려면 RESULT_CACHE 절을 사용하십시오. 결과 캐시된 함수가 호출되면 시스템에서 함수 결과 캐시를 검사합니다. 동일한 파라미터 값을 사용한 이전 함수 호출의 결과가 캐시에 포함되어 있는 경우에는 캐시된 결과가 호출자에게 반환되고 함수 본문이 재실행되지 않습니다. 캐시에 결과가 포함되어 있지 않으면 함수 본문이 실행되고 결과(해당 파라미터 값에 대한 결과)가 캐시에 추가됩니다. 이 작업이 수행된 이후에는 호출자가 다시 제어할 수 있습니다.

캐시에는 여러 결과가 누적될 수 있습니다. 각 결과 캐시된 함수를 호출할 때 사용된 모든 파라미터 값의 고유한 조합에 대해 결과가 하나씩 있습니다. 시스템에 메모리가 더 필요한 경우에는 하나 이상의 캐시된 결과를 Age Out(삭제)합니다.

참고: 함수 실행이 처리되지 않은 예외로 끝날 경우 예외 결과는 캐시에 저장되지 않습니다.

결과 캐시된 함수 선언 및 정의: 예제

```

CREATE OR REPLACE FUNCTION emp_hire_date (p_emp_id
    NUMBER) RETURN VARCHAR
RESULT_CACHE RELIES_ON (employees) IS
    v_date_hired DATE;
BEGIN
    SELECT hire_date INTO v_date_hired
    FROM HR.Employees
    WHERE Employee_ID = p_emp_ID;
    RETURN to_char(v_date_hired);
END;

```

FUNCTION emp_hire_date Compiled.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

결과 캐시된 함수 선언 및 정의: 예제

NLS_DATE_FORMAT 및 TIME_ZONE과 같이 세션마다 달라질 수 있는 설정에 함수가 종속되어 있는 경우에는 함수를 수정하여 다양한 설정을 처리할 수 있는 경우에만 함수를 결과 캐시된 함수로 만드십시오.

슬라이드의 예제에서 emp_hire_date 함수는 to_char 함수를 사용하여 DATE 항목을 VARCHAR 항목으로 변환합니다. emp_hire_date는 형식 마스크를 지정하지 않으므로 NLS_DATE_FORMAT이 지정하는 형식 마스크가 기본 형식 마스크입니다. emp_hire_date를 호출하는 세션의 NLS_DATE_FORMAT 설정이 서로 다를 경우 캐시된 결과의 형식이 서로 다를 수 있습니다. 한 세션에서 계산한 캐시된 결과가 Age Out되어 다른 세션에서 이를 다시 계산하는 경우에는 동일한 파라미터 값에 대해서도 형식이 다를 수 있습니다. 세션의 형식과 다른 형식의 캐시된 결과를 받을 경우 해당 결과는 잘못된 것일 수 있습니다.

이 문제에 대해 적용 가능한 몇 가지 해결책은 다음과 같습니다.

- emp_hire_date의 반환 유형을 DATE로 변경하고 각 세션이 to_char 함수를 호출하도록 합니다.
- 모든 세션에 공통 형식을 그대로 적용할 수 있는 경우 형식 마스크를 지정하고 NLS_DATE_FORMAT에 대한 종속성을 제거합니다(예: to_char(date_hired, 'mm/dd/yy'))

결과 캐시된 함수 선언 및 정의: 예제(계속)

- 다음과 같이 HireDate에 형식 마스크 파라미터를 추가합니다.

```
CREATE OR REPLACE FUNCTION emp_hire_date (p_emp_id NUMBER,  
                                         fmt VARCHAR) RETURN VARCHAR  
RESULT_CACHE RELIES_ON (employees) IS  
    v_date_hired DATE;  
BEGIN  
    SELECT hire_date INTO v_date_hired  
    FROM employees  
    WHERE employee_id = p_emp_id;  
    RETURN to_char(v_date_hired, fmt);  
END;
```

함수와 함께 DETERMINISTIC 절 사용

- DETERMINISTIC을 지정하여 동일한 인수 값으로 함수를 호출할 때마다 동일한 결과 값이 반환됨을 나타냅니다.
- 이를 통해 옵티마이저가 중복된 함수 호출을 피할 수 있습니다.
- 이전에 동일한 인수를 사용하여 함수를 호출한 경우에는 옵티마이저가 이전 결과를 사용할 수도 있습니다.
- 세션 변수 또는 스키마 객체의 상태에 결과가 종속되는 함수에 대해서는 DETERMINISTIC을 지정하지 마십시오.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

함수와 함께 DETERMINISTIC 절 사용

DETERMINISTIC 함수 절을 사용하여 동일한 인수 값으로 함수를 호출할 때마다 같은 결과 값이 반환됨을 나타낼 수 있습니다.

함수 기반 인덱스의 표현식에서 함수를 호출하거나 REFRESH FAST 또는 ENABLE QUERY REWRITE라고 표시된 Materialized view의 query로부터 함수를 호출하려면 이 키워드를 지정해야 합니다. 오라클 데이터베이스는 이러한 쿼리스트 중 하나에서 Deterministic 함수를 발견할 때 가능하면 함수를 재실행하기보다는 이전에 계산한 결과를 사용하려고 합니다. 나중에 함수의 의미를 변경하는 경우에는 모든 종속 함수 기반 인덱스와 Materialized view를 수동으로 재작성해야 합니다.

함수의 반환 결과에 영향을 미치는 방식으로 패키지 변수를 사용하거나 데이터베이스에 액세스하는 함수의 경우에는 이 절을 지정하여 정의하지 마십시오. 오라클 데이터베이스에서 함수를 재실행하지 않기로 결정할 경우 해당 결과가 캡처되지 않습니다.

참고

- 세션 변수 또는 스키마 객체의 상태에 결과가 종속되는 함수에 대해서는 DETERMINISTIC을 지정하지 마십시오. 호출할 때마다 결과가 달라질 수 있기 때문입니다. 대신 함수를 결과 캐시된 함수로 만드십시오.
- DETERMINISTIC 절에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g Release 1 (11.1)* 설명서를 참조하십시오.

단원 내용

- 상수 및 예외 표준화, 로컬 서브 프로그램 사용, 서브 프로그램의 런타임 권한 제어, 독립 트랜잭션 수행
- NOCOPY 및 PARALLEL ENABLE 힌트, 세션간 PL/SQL 함수 결과 캐시, DETERMINISTIC 절 사용
- DML과 함께 RETURNING 절 및 대량 바인드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

RETURNING 절 사용

- INSERT, UPDATE 및 DELETE 문을 사용하여 열 값을 반환하므로 성능이 향상됩니다.
- SELECT 문을 사용할 필요가 없습니다.

```

CREATE OR REPLACE PROCEDURE update_salary(p_emp_id
    NUMBER) IS
    v_name employees.last_name%TYPE;
    v_new_sal employees.salary%TYPE;
BEGIN
    UPDATE employees
        SET salary = salary * 1.1
    WHERE employee_id = p_emp_id
    RETURNING last_name, salary INTO v_name, v_new_sal;
    DBMS_OUTPUT.PUT_LINE(v_name || ' new salary is ' ||
        v_new_sal);
END update_salary;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

RETURNING 절 사용

보고서를 생성하거나 후속 작업을 수행하기 위해 응용 프로그램은 종종 SQL 작업의 영향을 받는 행에 대한 정보가 필요할 수 있습니다. INSERT, UPDATE 및 DELETE 문은 영향을 받는 행의 열 값을 PL/SQL 변수나 호스트 변수로 반환하는 RETURNING 절을 포함할 수 있습니다. 그러면 INSERT 또는 UPDATE 후 또는 DELETE 전에 행에 대해 SELECT를 수행할 필요가 없습니다. 결과적으로 네트워크 왕복, 서버 CPU 시간, 커서 및 서버 메모리에 대한 요구가 줄어듭니다. 슬라이드에 표시된 예제는 사원의 급여를 갱신하는 동시에 사원의 성과 새로운 급여를 로컬 PL/SQL 변수로 읽어 들이는 방법을 보여줍니다. 코드 예제를 테스트하려면 업데이트하기 전에 employee_id 108의 급여를 확인하는 다음 명령을 실행합니다. 그러면 employee_id 108을 파라미터에 전달하는 프로시저가 호출됩니다.

```

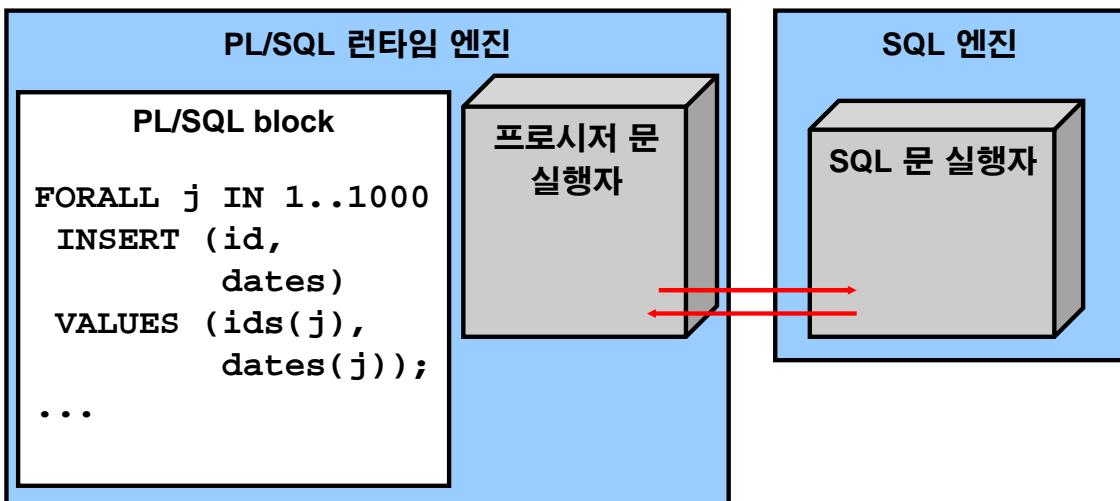
1 SET SERVEROUTPUT ON
2 /
3 SELECT last_name, salary
4 FROM employees
5 WHERE employee_id = 108;
6 /
7 EXECUTE update_salary(108)

```

LAST_NAME	SALARY
Greenberg	12000
1 rows selected	
anonymous block completed	
Greenberg new salary is 13200	

대량 바인드 사용

루프를 사용하여 FETCH, INSERT, UPDATE 및 DELETE 작업을 여러 번 수행하는 대신, 한 번의 작업으로 전체 값 배열을 바인드합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

대량 바인드 사용

Oracle 서버는 다음 두 개의 엔진을 사용하여 PL/SQL 블록과 서브 프로그램을 실행합니다.

- PL/SQL 런타임 엔진 - 프로시저 문을 실행하지만 SQL 문을 SQL 엔진에 전달합니다.
- SQL 엔진 - SQL 문을 구문 분석 및 실행하고, 경우에 따라 PL/SQL 엔진에 데이터를 반환합니다.

실행 중 모든 SQL 문은 두 엔진 간에 컨텍스트 전환을 유발하여 과도한 SQL 처리량으로 인한 성능 저하를 가져옵니다. 이것은 인덱스화된 컬렉션의 값을 사용하는 루프에 SQL 문이 포함된 응용 프로그램에서 일반적으로 나타나는 현상입니다. 컬렉션에는 중첩 테이블, 가변 배열, 인덱스화된 테이블 및 호스트 배열이 포함됩니다.

대량 바인드를 사용하여 컨텍스트 전환 횟수를 최소화하면 성능을 크게 향상시킬 수 있습니다.

대량 바인드는 한 번의 호출, 즉 한 번의 컨텍스트 전환으로 전체 컬렉션이 SQL 엔진에 바인드되도록 합니다. 즉, 루프 반복에서는 컬렉션 요소마다 컨텍스트 전환이 발생하는 것에 비해 대량 바인드 프로세스는 한 번의 컨텍스트 전환으로 두 엔진 간에 전체 값 모음을 전달합니다. 대량 바인드의 경우 SQL 문의 영향을 받는 행이 많을수록 성능이 더 향상됩니다.

대량 바인딩: 구문 및 키워드

- **FORALL 키워드는 입력 컬렉션을 SQL 엔진에 보내기 전에 PL/SQL 엔진에서 이를 대량으로 바인드하도록 합니다.**

```
FORALL index IN lower_bound .. upper_bound
[ SAVE EXCEPTIONS ]
sql_statement;
```

- **BULK COLLECT 키워드는 출력 컬렉션을 PL/SQL 엔진에 반환하기 전에 SQL 엔진에서 이를 대량으로 바인드할 수 있도록 합니다.**

```
... BULK COLLECT INTO
collection_name[,collection_name] ...
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

대량 바인딩: 구문 및 키워드

대량 바인드를 사용하여 다음의 성능을 높일 수 있습니다.

- 컬렉션 요소를 참조하는 DML 문
- 컬렉션 요소를 참조하는 SELECT 문
- 컬렉션 및 RETURNING INTO 절을 참조하는 커서 FOR 루프

FORALL 키워드는 입력 컬렉션을 SQL 엔진에 보내기 전에 PL/SQL 엔진에서 이를 대량으로 바인드하도록 합니다. FORALL 문은 반복 체계를 포함하고 있지만 FOR 루프가 아닙니다.

BULK COLLECT 키워드는 출력 컬렉션을 PL/SQL 엔진에 반환하기 전에 SQL 엔진에서 이를 대량으로 바인드하도록 합니다. 이 키워드를 사용하면 읽어 들인 값을 SQL이 대량으로 반환할 수 있는 위치를 바인드할 수 있습니다. 따라서 SELECT INTO, FETCH INTO 및 RETURNING INTO 절에서 이 키워드를 사용할 수 있습니다.

SAVE EXCEPTIONS 키워드는 선택적입니다. 그러나 일부 DML 작업은 성공하고 일부는 실패하는 경우 실패하는 작업을 추적하고 보고할 수 있습니다. 이 경우 SAVE EXCEPTIONS 키워드를 사용하면 %BULK_EXCEPTIONS라는 커서 속성에 실패한 작업이 저장됩니다. 이 속성은 대량 DML 반복 작업 수와 해당 오류 코드를 나타내는 레코드 모음입니다.

대량 바인딩: 구문 및 키워드(계속)

%BULK_EXCEPTIONS 속성을 사용하여 FORALL 예외 처리

예외를 관리하고 오류가 발생하더라도 대량 바인드를 완료하려면 FORALL 문에서 bound와 DML 문 사이에 SAVE EXCEPTIONS 키워드를 추가하십시오.

실행 중에 발생한 모든 예외는 레코드 모음을 저장하는 커서 속성 %BULK_EXCEPTIONS에 저장됩니다. 각 레코드에는 두 개의 필드가 있습니다.

%BULK_EXCEPTIONS(i).ERROR_INDEX는 예외가 발생하는 동안 FORALL 문의 "반복"을 포함하고 %BULK_EXCEPTIONS(i).ERROR_CODE는 해당하는 오라클 오류 코드를 포함합니다.

%BULK_EXCEPTIONS에 저장된 값은 가장 최근에 실행된 FORALL 문을 참조합니다.

해당 첨자 범위는 1부터 %BULK_EXCEPTIONS.COUNT까지입니다.

참고: 대량 바인드 및 대량 바인드 예외 처리에 대한 자세한 내용은 *Oracle Database PL/SQL User's Guide and Reference 11g Release 2 (11.2)*를 참조하십시오.

FORALL 대량 바인드: 예제

```

CREATE PROCEDURE raise_salary(p_percent NUMBER) IS
    TYPE numlist_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_id  numlist_type; -- collection
BEGIN
    v_id(1):= 100; v_id(2):= 102; v_id(3):= 104; v_id(4) := 110;
    -- bulk-bind the PL/SQL table
    FORALL i IN v_id.FIRST .. v_id.LAST
        UPDATE employees
            SET salary = (1 + p_percent/100) * salary
            WHERE employee_id = v_id(i);
END;
/

```

```
EXECUTE raise_salary(10)
```

anonymous block completed

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FORALL 대량 바인드: 예제

참고: 슬라이드의 예제를 실행하기 전에 다음과 같이 update_job_history 트리거를 비활성화해야 합니다.

```
ALTER TRIGGER update_job_history DISABLE;
```

슬라이드의 예제에서 PL/SQL 블록은 관리자 ID가 100, 102, 104 또는 110인 사원의 급여를 인상하며, FORALL 키워드를 사용하여 컬렉션을 대량으로 바인드합니다. 대량 바인드를 사용하지 않았다면 PL/SQL 블록은 개신되는 사원 레코드마다 SQL 엔진에 SQL 문을 보냈을 것입니다. 개신할 사원 레코드가 많을 경우 PL/SQL 엔진과 SQL 엔진 간의 컨텍스트 전환도 많아지므로 SQL 엔진이 성능에 영향을 줄 수 있습니다. 그러나 FORALL 키워드는 컬렉션을 대량으로 바인드하여 성능을 높입니다.

참고: 이 기능을 사용할 경우 루프 생성은 더 이상 필요하지 않습니다.

FORALL 대량 바인드: 예제(계속)

DML 작업의 추가 커서 속성

대량의 작업을 지원하기 위해 추가된 또 다른 커서 속성은 %BULK_ROWCOUNT입니다.

%BULK_ROWCOUNT 속성은 FORALL 문과 함께 사용하도록 설계된 조합 구조입니다. 이 속성은 인덱스화된 테이블처럼 작동합니다. 이 속성의 *i*번째 요소는 UPDATE 또는 DELETE 문을 *i*번째 실행할 때 처리되는 행 수를 저장합니다. *i*번째 실행할 때 처리되는 행이 없을 경우 %BULK_ROWCOUNT(*i*)는 0을 반환합니다.

예를 들면 다음과 같습니다.

```

CREATE TABLE num_table (n NUMBER);
DECLARE
    TYPE num_list_type IS TABLE OF NUMBER
        INDEX BY BINARY_INTEGER;
    v_nums num_list_type;
BEGIN
    v_nums(1) := 1;
    v_nums(2) := 3;
    v_nums(3) := 5;
    v_nums(4) := 7;
    v_nums(5) := 11;
    FORALL i IN v_nums.FIRST .. v_nums.LAST
        INSERT INTO v_num_table (n) VALUES (v_nums(i));
    FOR i IN v_nums.FIRST .. v_nums.LAST
        LOOP
            dbms_output.put_line('Inserted ' ||
                SQL%BULK_ROWCOUNT(i) || ' row(s)' ||
                ' on iteration ' || i);
        END LOOP;
    END;
/
DROP TABLE num_table;

```

이 예제의 결과는 다음과 같습니다.

```

CREATE TABLE succeeded.
anonymous block completed
Inserted 1 row(s) on iteration 1
Inserted 1 row(s) on iteration 2
Inserted 1 row(s) on iteration 3
Inserted 1 row(s) on iteration 4
Inserted 1 row(s) on iteration 5

DROP TABLE num_table succeeded.

```

Query와 함께 BULK COLLECT INTO 사용

SELECT 문은 BULK COLLECT INTO 구문을 지원합니다.

```

CREATE PROCEDURE get_departments(p_loc NUMBER) IS
  TYPE dept_tab_type IS
    TABLE OF departments%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  SELECT * BULK COLLECT INTO v_depts
  FROM departments
  WHERE location_id = p_loc;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      || ' ' || v_depts(i).department_name);
  END LOOP;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Query와 함께 BULK COLLECT INTO 사용

Oracle Database 10g부터는 PL/SQL에서 SELECT 문을 사용할 때 슬라이드 예제에 표시된 대량 컬렉션 구문을 사용할 수 있습니다. 따라서 커서 방식을 사용하지 않고도 행 집합을 신속하게 얻을 수 있습니다.

예제는 지정된 지역의 모든 부서 행을 PL/SQL 테이블로 읽어 들입니다. PL/SQL 테이블의 내용은 SELECT 문 뒤에 나오는 FOR 루프를 사용하여 표시됩니다.

커서와 함께 BULK COLLECT INTO 사용

FETCH 문은 BULK COLLECT INTO 구문을 지원하도록 향상되었습니다.

```

CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER) IS
  CURSOR cur_dept IS
    SELECT * FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tab_type IS TABLE OF cur_dept%ROWTYPE;
  v_depts dept_tab_type;
BEGIN
  OPEN cur_dept;
  FETCH cur_dept BULK COLLECT INTO v_depts;
  CLOSE cur_dept;
  FOR i IN 1 .. v_depts.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_depts(i).department_id
      || ' ' || v_depts(i).department_name);
  END LOOP;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

커서와 함께 BULK COLLECT INTO 사용

Oracle Database 10g에서는 PL/SQL에서 커서를 사용할 때 슬라이드 예제에 표시된 대량 컬렉션 구문을 지원하는 FETCH 문 형식을 사용할 수 있습니다.

이 예제는 BULK COLLECT INTO를 커서와 함께 사용하는 방법을 보여줍니다.

또한 LIMIT 절을 추가하여 각 작업에서 패치(fetch)된 행 수를 제어할 수 있습니다.

위 슬라이드에 표시된 코드 예제는 다음과 같이 수정할 수 있습니다.

```

CREATE OR REPLACE PROCEDURE get_departments(p_loc NUMBER,
  p_nrows NUMBER) IS
  CURSOR dept_csr IS SELECT *
    FROM departments
    WHERE location_id = p_loc;
  TYPE dept_tabtype IS TABLE OF dept_csr%ROWTYPE;
  depts dept_tabtype;
BEGIN
  OPEN dept_csr;
  FETCH dept_csr BULK COLLECT INTO depts LIMIT nrows;
  CLOSE dept_csr;
  DBMS_OUTPUT.PUT_LINE(depts.COUNT || ' rows read');
END;

```

RETURNING 절과 함께 BULK COLLECT INTO 사용

```

CREATE OR REPLACE PROCEDURE raise_salary(p_rate NUMBER)
IS
  TYPE emplist_type IS TABLE OF NUMBER;
  TYPE numlist_type IS TABLE OF employees.salary%TYPE
    INDEX BY BINARY_INTEGER;
  v_emp_ids emplist_type :=
    emplist_type(100,101,102,104);
  v_new_sals numlist_type;
BEGIN
  FORALL i IN v_emp_ids.FIRST .. v_emp_ids.LAST
    UPDATE employees
      SET commission_pct = p_rate * salary
      WHERE employee_id = v_emp_ids(i)
    RETURNING salary BULK COLLECT INTO v_new_sals;
  FOR i IN 1 .. v_new_sals.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(v_new_sals(i));
  END LOOP;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

RETURNING 절과 함께 BULK COLLECT INTO 사용

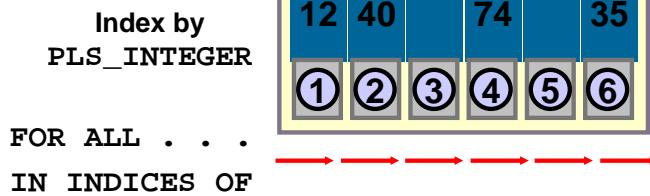
대량 바인드를 사용하여 컬렉션을 참조하고 DML을 반환하는 FOR 루프의 성능을 높일 수 있습니다. 이러한 역할을 하는 PL/SQL 코드가 있거나 앞으로 추가될 예정이면 FORALL 키워드를 RETURNING 및 BULK COLLECT INTO 키워드와 함께 사용하여 성능을 높일 수 있습니다.

슬라이드에 표시된 예제에서는 salary 정보를 EMPLOYEES 테이블에서 가져오고 new_sals 배열로 수집합니다. new_sals 컬렉션은 PL/SQL 엔진에 대량으로 반환됩니다.

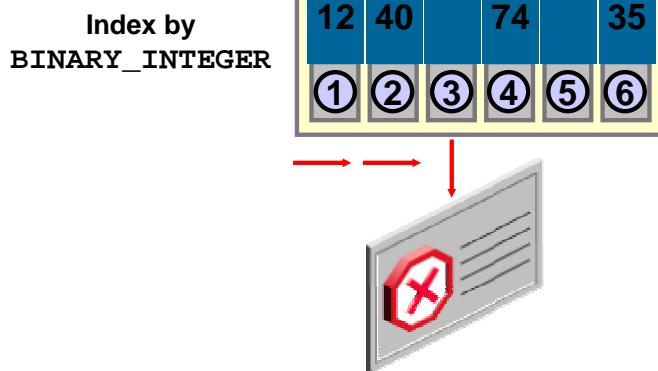
슬라이드에 표시된 예제는 UPDATE 작업에서 받은 새 급여 데이터를 반복 실행하고 그 결과를 처리하는 데 사용되는 불완전한 FOR 루프를 보여줍니다.

Sparse Collection에서 대량 바인드 사용

현재 버전:



이전 버전:



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Sparse Collection에 대한 FORALL 지원

- 이전 버전에서는 PL/SQL의 FORALL 문에서 Sparse Collection을 사용하도록 허용하지 않았습니다.
- SAVE EXCEPTIONS 키워드가 지정되어 있지 않으면 삭제된 첫번째 요소가 발견될 때 명령문이 종료되었습니다. SAVE EXCEPTION이 사용되더라도 PL/SQL 엔진은 모든 요소(존재하는 요소와 존재하지 않는 요소)를 반복하려고 했습니다. 이로 인해 삭제된 요소의 비율이 비교적 높은 경우에는 DML 작업의 성능이 크게 저하되었습니다.
- Oracle Database 10g 이상에서 사용할 수 있는 키워드 INDICES를 사용하면 Sparse Collection을 FORALL 문과 함께 반복 실행할 수 있습니다. 이 구문은 Sparse Collection을 더 효율적으로 바인드하며 컬렉션을 반복하도록 인덱스 배열을 지정할 수 있는 보다 일반적인 접근 방식을 지원합니다.
- 대량 작업에서 Sparse Collection 및 인덱스 배열을 사용하면 성능이 향상됩니다.

Sparse Collection에서 대량 바인드 사용

```
-- The INDICES OF syntax allows the bound arrays
-- themselves to be sparse.

FORALL index_name IN INDICES OF sparse_array_name
    BETWEEN LOWER_BOUND AND UPPER_BOUND -- optional
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        sparse_array(index_name);
    . . .
```

```
-- The VALUES OF syntax lets you indicate a subset
-- of the binding arrays.

FORALL index_name IN VALUES OF index_array_name
    SAVE EXCEPTIONS -- optional, but recommended
    INSERT INTO table_name VALUES
        binding_array_name(index_name);
    . . .
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Sparse Collection에 대한 FORALL 지원(계속)

- INDICES OF 및 VALUES OF 구문을 FORALL 문과 함께 사용할 수 있습니다.
- Sparse Array 구문의 대량 바인드는 모든 DML 구문에서 사용될 수 있습니다.
- 구문에서 인덱스 배열은 조밀(dense)해야 하고, 바인딩 배열은 조밀하거나 산재(sparse)될 수 있으며 표시하는 요소가 존재해야 합니다.

Sparse Collection에서 대량 바인드 사용

이 기능은 일반적으로 주문 입력 및 주문 처리 시스템에서 다음과 같은 때 적용됩니다.

- 유저가 웹을 통해 주문을 입력할 때
- 주문이 검증 이전에 스테이지 처리 테이블에 배치될 때
- 나중에 복잡한 업무 규칙(대개 PL/SQL을 사용하여 프로그래밍 방식으로 구현됨)을 기준으로 데이터를 검증할 때
- 무효 주문을 유효 주문과 분리할 때
- 유효 주문을 처리될 영구 테이블에 삽입할 때

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Sparse Collection에서 대량 바인드 사용

대량 수집을 사용하여 채운 조밀한 PL/SQL 레코드 테이블 또는 스칼라 테이블로 시작되는 응용 프로그램에서 이 기능을 사용할 수 있습니다. 이 기능은 바인딩 배열로 사용됩니다. 배열의 요소가 바인딩 배열의 인덱스를 표시하는 Dense Array(포인터)는 응용 프로그램 논리를 기준으로 Sparse Array가 됩니다. 그런 다음 FORALL 문에서 이 포인터 배열을 사용하여 대량 DML을 바인딩 배열과 함께 수행합니다. 발생하는 모든 예외를 저장한 다음 다른 FORALL 문을 사용하여 예외 처리 섹션에서 처리할 수 있습니다.

인덱스 배열에서 대량 바인드 사용

```

CREATE OR REPLACE PROCEDURE ins_emp2 AS
  TYPE emptab_type IS TABLE OF employees%ROWTYPE;
  v_emp emptab_type;
  TYPE values_of_tab_type IS TABLE OF PLS_INTEGER
    INDEX BY PLS_INTEGER;
  v_num values_of_tab_type;
  . . .
BEGIN
  . . .
  FORALL k IN VALUES OF v_num
    INSERT INTO new_employees VALUES v_emp(k);
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

인덱스 배열에서 대량 바인드 사용

PLS_INTEGER 또는 BINARY_INTEGER(또는 해당 서브타입 중 하나)의 인덱스 모음 값을 FORALL을 사용하는 대량 바인드 DML 작업과 관련된 컬렉션의 인덱스로 사용할 수 있습니다. 이러한 인덱스 컬렉션은 FORALL 문에서 VALUES OF 절을 사용하는 대량 DML을 처리하는데 사용될 수 있습니다.

위 예제에서 V_NUM은 유형이 PLS_INTEGER인 컬렉션입니다. 예제에서는 성의 첫 글자가 나타날 때마다 한 명의 사원만 식별하는 INS_EMP2 프로시저를 생성합니다. 그런 다음 앞에서 FORALL..IN VALUES OF 구문을 사용하여 생성한 NEW_EMPLOYEES 테이블에 이 프로시저를 삽입합니다.

퀴즈

NOCOPY 힌트를 사용하면 PL/SQL 컴파일러에서 OUT 및 IN OUT 파라미터를 값이 아닌 참조로 전달할 수 있습니다. 그러면 파라미터를 전달할 때 오버헤드가 줄어 성능이 향상됩니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1

PL/SQL 서브 프로그램은 IN, OUT, IN OUT이라는 세 가지 파라미터 전달 모드를 지원합니다. 기본 사항은 다음과 같습니다.

- IN 파라미터는 참조로 전달됩니다. 즉, IN 실제 파라미터에 대한 포인터가 해당 형식 파라미터에 전달됩니다. 따라서 두 파라미터는 실제 파라미터 값은 보유하고 있는 동일한 메모리 위치를 참조합니다.
- OUT 및 IN OUT 파라미터는 값으로 전달됩니다. 즉, OUT 또는 IN OUT 실제 파라미터의 값이 해당 형식 파라미터에 복사됩니다. 그런 다음 서브 프로그램이 정상적으로 종료되면 OUT 및 IN OUT 형식 파라미터에 지정된 값이 해당하는 실제 파라미터에 복사됩니다.

큰 데이터 구조(예: 컬렉션, 레코드, 객체 유형 instance)를 나타내는 파라미터를 OUT 및 IN OUT 파라미터와 함께 복사하면 실행 속도가 느려지고 메모리 사용량이 늘어납니다. 이러한 오버헤드를 막기 위해 PL/SQL 컴파일러에서 OUT 및 IN OUT 파라미터를 참조로 전달할 수 있도록 NOCOPY 힌트를 지정할 수 있습니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 표준 상수 및 예외 생성
- 로컬 서브 프로그램 작성 및 호출
- 서브 프로그램의 런타임 권한 제어
- 독립 트랜잭션 수행
- NOCOPY 힌트를 사용하여 파라미터를 참조로 전달
- 최적화를 위해 PARALLEL_ENABLE 힌트 사용
- 세션간 PL/SQL 함수 결과 캐시 사용
- 함수와 함께 DETERMINISTIC 절 사용
- DML과 함께 RETURNING 절 및 대량 바인드 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 Package Spec에서 상수와 예외를 정의하여 PL/SQL 코드 관리 방법을 살펴봤습니다. 따라서 보다 높은 수준으로 코드를 재사용하고 표준화할 수 있습니다.

로컬 서브 프로그램은 로컬 블록에서 서브 프로그램 기능이 반복적으로 사용되는 코드 블록을 단순화하고 모듈화하는 데 사용할 수 있습니다.

서브 프로그램의 런타임 보안 권한은 정의자 권한이나 호출자 권한을 사용하여 변경할 수 있습니다.

독립 트랜잭션은 기존 주 트랜잭션에 영향을 주지 않으면서 실행될 수 있습니다.

SQL 문에서 NOCOPY 힌트, 대량 바인드 및 RETURNING 절을 사용하고 함수 최적화를 위해 PARALLEL_ENABLE 힌트를 사용하여 성능을 높이는 방법을 이해해야 합니다.

연습 7: 개요

이 연습에서는 다음 내용을 다룹니다.

- 대량 패치(fetch) 작업을 사용하는 패키지 생성
- 업무 운영을 감사(audit)하도록 독립 트랜잭션을 수행하는 로컬 서브 프로그램 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 7: 개요

이 연습에서는 지정된 부서의 사원을 대량으로 패치(fetch)하는 패키지를 작성합니다. 데이터는 패키지의 PL/SQL 테이블에 저장됩니다. 또한 테이블 내용을 표시하는 프로시저를 제공합니다.

패키지에 새로운 사원을 삽입하는 add_employee 프로시저를 추가합니다. 이 프로시저는 레코드가 성공적으로 추가되었는지 여부와 관계없이 add_employee 프로시저가 호출될 때마다 로컬 독립 서브 프로그램을 사용하여 로그 레코드를 기록합니다.

트리거 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 데이터베이스 트리거 및 사용법 설명
- 여러 유형의 트리거 설명
- 데이터베이스 트리거 생성
- 데이터베이스 트리거 실행 규칙 설명
- 데이터베이스 트리거 제거
- 트리거 정보 표시

ORACLE®

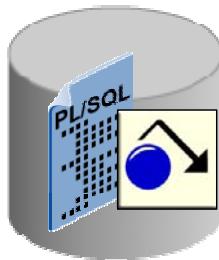
Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 데이터베이스 트리거를 생성하고 사용하는 방법에 대해 설명합니다.

트리거란?

- 트리거는 데이터베이스에 저장되고 지정된 이벤트에 대한 응답으로 실행되는 PL/SQL 블록입니다.
- 오라클 데이터베이스는 지정된 조건이 발생할 때 트리거를 자동으로 실행합니다.



ORACLE®

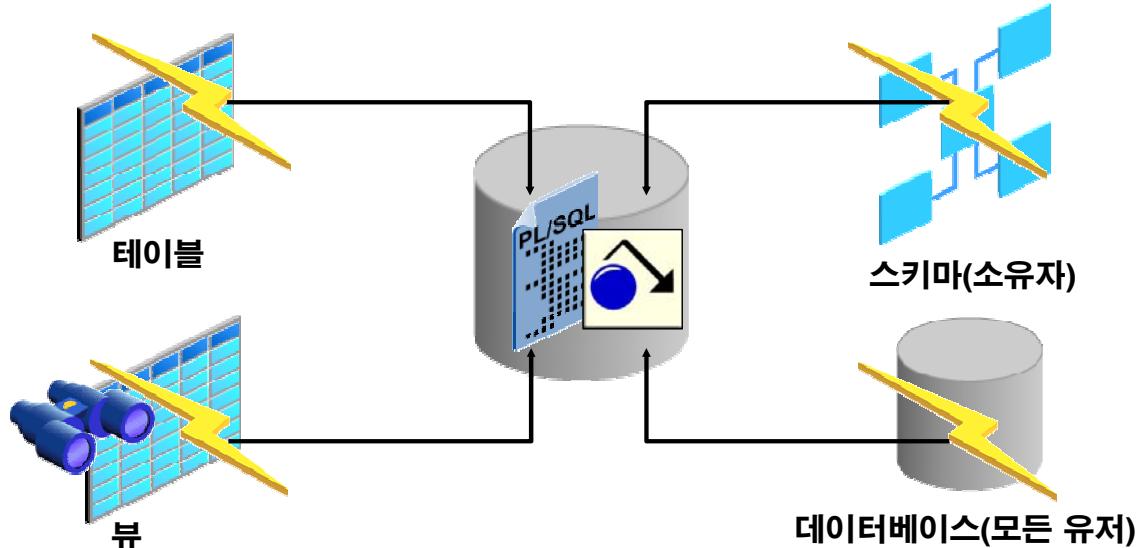
Copyright © 2009, Oracle. All rights reserved.

트리거 작업: 개요

트리거는 내장 프로시저와 비슷합니다. 데이터베이스에 저장된 트리거에는 PL/SQL이 익명 블록, 호출 문 또는 혼합 트리거 블록의 형태로 포함되어 있습니다. 그러나 프로시저와 트리거는 호출 방법이 다릅니다. 프로시저는 유저, 응용 프로그램 또는 트리거에 의해 명시적으로 실행됩니다. 트리거는 연결된 유저와 사용 중인 응용 프로그램에 관계없이 트리거 이벤트가 발생할 때 오라클 데이터베이스에서 암시적으로 실행됩니다.

트리거 정의

트리거는 테이블, 뷰, 스키마(스키마 소유자) 또는 데이터베이스(모든 유저)에 정의될 수 있습니다.



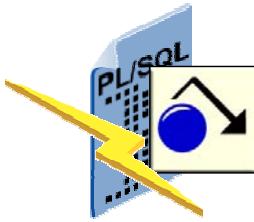
Copyright © 2009, Oracle. All rights reserved.

ORACLE

트리거 이벤트 유형

데이터베이스에서 다음 작업 중 하나가 발생할 때마다 실행되는 트리거를 작성할 수 있습니다.

- DML(데이터베이스 조작) 문(DELETE, INSERT 또는 UPDATE)
- DDL(데이터베이스 정의) 문(CREATE, ALTER 또는 DROP)
- SERVERERROR, LOGON, LOGOFF, STARTUP 또는 SHUTDOWN과 같은 데이터베이스 작업



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

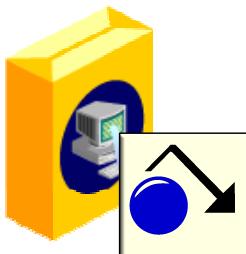
트리거 이벤트 또는 트리거 문

트리거 이벤트 또는 트리거 문은 트리거가 실행되도록 만드는 SQL 문, 데이터베이스 이벤트 또는 유저 이벤트입니다. 트리거 이벤트는 다음 중 하나 이상일 수 있습니다.

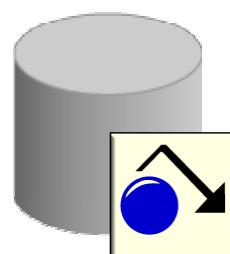
- 특정 테이블(또는 일부 경우 뷰)에 있는 INSERT, UPDATE 또는 DELETE 문
- 스키마 객체에 있는 CREATE, ALTER 또는 DROP 문
- 데이터베이스 시작 또는 Instance 종료
- 특정 오류 메시지 또는 임의의 오류 메시지
- 유저 로그온 또는 로그오프

응용 프로그램 및 데이터베이스 트리거

- **데이터베이스 트리거(이 과정에서 다룹):**
 - 스키마 또는 데이터베이스에서 DML, DDL 또는 시스템 이벤트가 발생할 때마다 실행됨
- **응용 프로그램 트리거:**
 - 특정 응용 프로그램 내에서 이벤트가 발생할 때마다 실행됨



응용 프로그램 트리거



데이터베이스 트리거

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거 유형

응용 프로그램 트리거는 응용 프로그램 내에서 특정 DML(데이터 조작어) 이벤트가 발생할 때마다 암시적으로 실행됩니다. 트리거를 광범위하게 사용하는 응용 프로그램에 대한 예로 Oracle Forms Developer를 사용하여 개발된 응용 프로그램을 들 수 있습니다.

데이터베이스 트리거는 다음 이벤트가 발생할 경우에 암시적으로 실행됩니다.

- 테이블에 대한 DML 작업
- INSTEAD OF 트리거를 사용하는 뷰에 대한 DML 작업
- DDL 문(예: CREATE 및 ALTER)

이러한 데이터베이스 트리거는 연결된 유저나 사용 중인 응용 프로그램의 영향을 받지 않습니다. 일부 유저 작업이나 데이터베이스 시스템 작업이 발생한 경우(예를 들어 유저가 로그온하거나 DBA가 데이터베이스를 종료한 경우)에도 데이터베이스 트리거가 암시적으로 실행됩니다.

데이터베이스 트리거는 데이터베이스 또는 스키마에 있는 시스템 트리거일 수 있습니다. 이는 다음 단원에서 다릅니다. 데이터베이스의 경우 트리거가 모든 유저의 이벤트마다 실행되고, 스키마의 경우 트리거가 특정 유저의 이벤트마다 실행됩니다. Oracle Forms는 다양한 종류의 트리거를 정의, 저장 및 실행할 수 있습니다. 그러나 Oracle Forms 트리거를 이 단원에서 설명하는 트리거와 혼동해서는 안됩니다.

업무용 응용 프로그램의 트리거 구현 시나리오

다음 작업에 트리거를 사용할 수 있습니다.

- 보안
- 감사(audit)
- 데이터 무결성
- 참조 무결성
- 테이블 복제(replication)
- 파생된 데이터 자동 계산
- 이벤트 로깅

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

업무용 응용 프로그램의 트리거 구현 시나리오

Oracle 서버에서 구현할 수 있는 기능을 강화하기 위해 또는 Oracle 서버에서 제공하는 기능의 대체 기능으로 데이터베이스 트리거를 개발하십시오.

- **보안:** Oracle 서버는 유저나 룰에 대한 테이블 액세스를 허용합니다. 트리거는 데이터 값에 따라 테이블 액세스를 허용합니다.
- **감사(audit):** Oracle 서버는 테이블에 대한 데이터 작업을 추적합니다. 트리거는 테이블에 대한 데이터 작업 값을 추적합니다.
- **데이터 무결성:** Oracle 서버는 무결성 제약 조건을 적용합니다. 트리거는 복잡한 무결성 규칙을 구현합니다.
- **참조 무결성:** Oracle 서버는 표준 참조 무결성 규칙을 적용합니다. 트리거는 비표준 기능을 구현합니다.
- **테이블 복제(replication):** Oracle 서버는 비동기적으로 스냅샷에 테이블을 복사합니다. 트리거는 동기적으로 복제본에 테이블을 복사합니다.
- **파생된 데이터:** Oracle 서버는 파생된 데이터 값을 수동으로 계산합니다. 트리거는 파생된 데이터 값을 자동으로 계산합니다.
- **이벤트 로깅:** Oracle 서버는 이벤트를 명시적으로 기록합니다. 트리거는 이벤트를 투명하게 기록합니다.

사용 가능한 트리거 유형

- 단순 DML 트리거
 - BEFORE
 - AFTER
 - INSTEAD OF
- 혼합 트리거
- DML이 아닌 트리거
 - DDL 이벤트 트리거
 - 데이터베이스 이벤트 트리거

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고

이 단원에서는 BEFORE, AFTER 및 INSTEAD OF 트리거에 대해 설명합니다. 다른 트리거 유형은 "혼합, DDL 및 데이터베이스 이벤트 트리거 생성" 단원에서 설명합니다.

트리거 이벤트 유형 및 본문

- 트리거 이벤트 유형에 따라 트리거를 실행시키는 DML 문이 결정됩니다. 가능한 이벤트는 다음과 같습니다.
 - INSERT
 - UPDATE [OF column]
 - DELETE
- 트리거 본문은 수행될 작업을 결정하며 PL/SQL 블록 또는 프로시저에 대한 CALL입니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 이벤트 유형

트리거 이벤트 또는 트리거 문은 테이블에 있는 INSERT, UPDATE 또는 DELETE 문일 수 있습니다.

- 트리거 이벤트가 UPDATE 문일 경우 트리거 실행을 위해 변경되어야 하는 열을 식별하는 열 리스트를 포함할 수 있습니다. INSERT 문이나 DELETE 문은 항상 전체 행에 영향을 주기 때문에 이들에 대한 열 리스트는 지정할 수 없습니다.


```
... UPDATE OF salary ...
```
- 트리거 이벤트는 이러한 DML 작업 중 일부만 포함하거나 세 개 모두 포함할 수 있습니다.


```
... INSERT or UPDATE or DELETE
```

```
... INSERT or UPDATE OF job_id ...
```

트리거 본문은 트리거 이벤트가 실행될 때 수행되어야 할 작업을 정의합니다. PL/SQL 블록은 SQL 및 PL/SQL 문을 포함할 수 있으며 변수, 커서, 예외 등의 PL/SQL 생성자를 정의할 수 있습니다. 또한 PL/SQL 프로시저나 Java 프로시저를 호출할 수도 있습니다.

참고: 트리거의 크기는 32KB를 초과할 수 없습니다.

CREATE TRIGGER 문을 사용하여 DML 트리거 생성

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing -- when to fire the trigger
  event1 [OR event2 OR event3]
  ON object_name
  [REFERENCING OLD AS old | NEW AS new]
  FOR EACH ROW -- default is statement level trigger
  WHEN (condition)])
  DECLARE]
  BEGIN
    ... trigger_body -- executable statements
  [EXCEPTION . . .]
  END [trigger_name];
```

timing = BEFORE | AFTER | INSTEAD OF

event = INSERT | DELETE | UPDATE | UPDATE OF column_list

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 트리거 생성

트리거 구문의 구성 요소는 다음과 같습니다.

- trigger_name - 트리거를 고유하게 식별합니다.
- timing - 트리거 이벤트와 관련한 트리거 실행 시기를 나타냅니다. 값은 BEFORE, AFTER 및 INSTEAD OF입니다.
- event - 트리거가 실행되도록 하는 DML 작업을 식별합니다. 값은 INSERT, UPDATE [OF column] 및 DELETE입니다.
- object_name - 트리거와 연관된 테이블이나 뷰를 나타냅니다.
- 행 트리거의 경우 다음을 지정할 수 있습니다.
 - REFERENCING 절 - 현재 행의 이전 값과 새 값을 참조하기 위한 상관 관계 이름을 선택하며, 기본값은 OLD 및 NEW입니다.
 - FOR EACH ROW - 트리거가 행 트리거가 되도록 지정합니다.
 - WHEN 절 - 트리거 본문의 실행 여부를 결정하기 위해 각 행에 대해 평가되는 조건부 술어(괄호로 묶음)를 적용합니다.
- trigger_body는 트리거에 의해 수행되는 작업으로, 다음 중 하나로 구현됩니다.
 - DECLARE 또는 BEGIN과 END가 포함된 익명 블록
 - 다음과 같이 독립형 또는 패키지화된 내장 프로시저를 호출하는 CALL 절
CALL my_procedure;

트리거 실행 지정(타이밍)

트리거 타이밍을 지정하여 트리거의 작업이 트리거 문 이전 또는 이후에 실행되도록 할 것인지 여부를 결정할 수 있습니다.

- **BEFORE:** 테이블에서 DML 이벤트를 트리거하기 전에 트리거 본문을 실행합니다.
- **AFTER:** 테이블에서 DML 이벤트를 트리거한 후에 트리거 본문을 실행합니다.
- **INSTEAD OF:** 트리거 문 대신 트리거 본문을 실행하며, 다른 방법으로는 수정이 불가능한 뷰에 사용됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 타이밍

BEFORE 트리거 타이밍은 다음 상황에서 자주 사용됩니다.

- 트리거 문의 완료를 허용해야 할지 여부를 결정하려는 경우(이렇게 하면 불필요한 처리 과정이 없어지며 트리거 작업에 예외가 발생할 경우 롤백이 가능합니다.)
- INSERT 또는 UPDATE 문을 완료하기 전에 열 값을 파생하려는 경우
- Global 변수나 플래그를 초기화하거나 복잡한 업무 규칙을 검증하려는 경우

AFTER 트리거는 다음 상황에서 자주 사용됩니다.

- 트리거 작업을 실행하기 전에 트리거 문을 완료하려는 경우
- BEFORE 트리거가 이미 존재할 때 동일한 트리거 문에서 다른 작업을 수행하려는 경우

뷰를 항상 수정할 수 있는 것은 아니기 때문에 **INSTEAD OF** 트리거는 SQL DML 문을 통해 직접 수정할 수 없는 뷰를 수정하는 투명한 방법을 제공합니다. INSTEAD OF 트리거 본문 내에 적절한 DML 문을 작성하여 뷰의 기본 테이블에서 직접 작업을 수행할 수 있습니다.

타이밍 지점이 서로 다른 여러 개별 트리거를 원하는 순서대로 작업을 명시적으로 코딩하는 하나의 혼합 트리거로 대체하는 것이 실용적인 경우도 있습니다. 동일한 타이밍 지점에 두 개 이상의 트리거가 정의되어 있고 실행 시퀀스가 중요한 경우에는 FOLLOWS 및 PRECEDES 절을 사용하여 실행 시퀀스를 제어할 수 있습니다.

문장 레벨 트리거와 행 레벨 트리거 비교

문장 레벨 트리거	행 레벨 트리거
트리거를 생성할 때 기본값입니다.	트리거를 생성할 때 FOR EACH ROW 절을 사용합니다.
트리거 이벤트에 대해 한 번 실행됩니다.	트리거 이벤트의 영향을 받는 각 행에 대해 한 번 실행됩니다.
영향을 받는 행이 전혀 없더라도 한 번 실행됩니다.	트리거 이벤트의 영향을 받는 행이 없을 경우에는 실행되지 않습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DML 트리거 유형

트리거를 트리거 문(예: 여러 행 UPDATE)의 영향을 받는 행마다 한 번씩 실행할지 아니면 영향을 받는 행 수에 관계없이 트리거 문에 대해 한 번만 실행할지를 지정할 수 있습니다.

문장 트리거

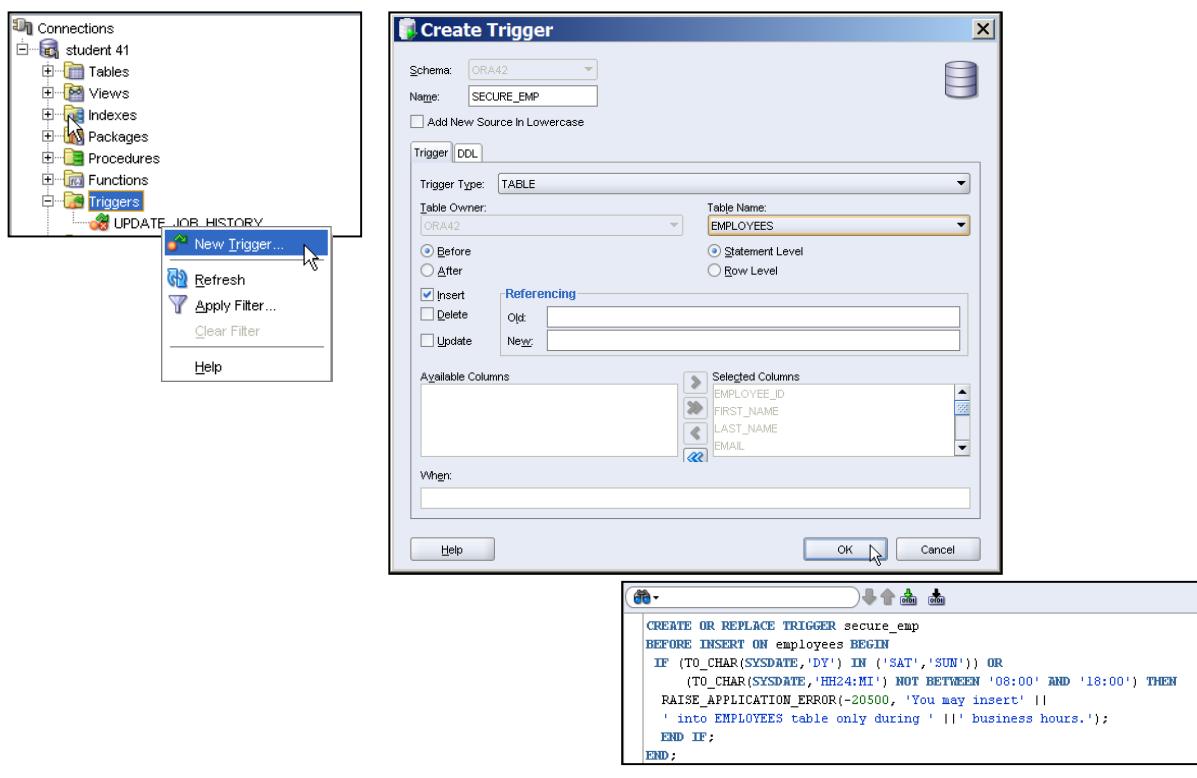
문장 트리거는 영향을 받는 행이 전혀 없더라도 트리거 이벤트 대신 한 번은 실행됩니다. 문장 트리거는 트리거 작업이 영향을 받는 행의 데이터 또는 트리거 이벤트 자체(예: 현재 유저에 대해 복잡한 보안 검사를 수행하는 트리거)에서 제공하는 데이터에 종속되지 않은 경우에 유용합니다.

행 트리거

행 트리거는 테이블이 트리거 이벤트의 영향을 받을 때마다 실행되고, 트리거 이벤트의 영향을 받는 행이 없을 경우에는 실행되지 않습니다. 행 트리거는 영향을 받는 행의 데이터나 트리거 이벤트 자체에서 제공하는 데이터에 트리거 작업이 종속될 경우에 유용합니다.

참고: 행 트리거는 상관 관계 이름을 사용하여 트리거에 의해 처리되는 행의 이전 열 값과 새 열 값에 액세스합니다.

SQL Developer를 사용하여 DML 트리거 생성



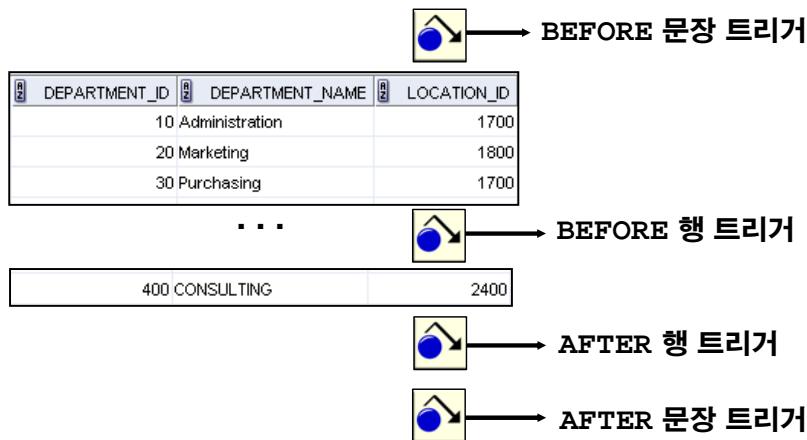
Copyright © 2009, Oracle. All rights reserved.

ORACLE

트리거 실행 시퀀스: 단일 행 조작

단일 행을 조작할 경우 테이블에서 다음 트리거 실행 시퀀스를 사용하십시오.

```
INSERT INTO departments
  (department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거 실행 시퀀스: 단일 행 조작

요구 사항, 즉 트리거가 트리거 문의 영향을 받는 행마다 한 번씩 실행될지 아니면 영향을 받는 행 수에 관계없이 트리거 문에 대해 한 번만 실행될지에 따라 문장 트리거를 생성하거나, 행 트리거를 생성합니다.

트리거 DML 문이 단일 행에만 영향을 줄 경우 문장 트리거와 행 트리거 모두 정확히 한 번 실행됩니다.

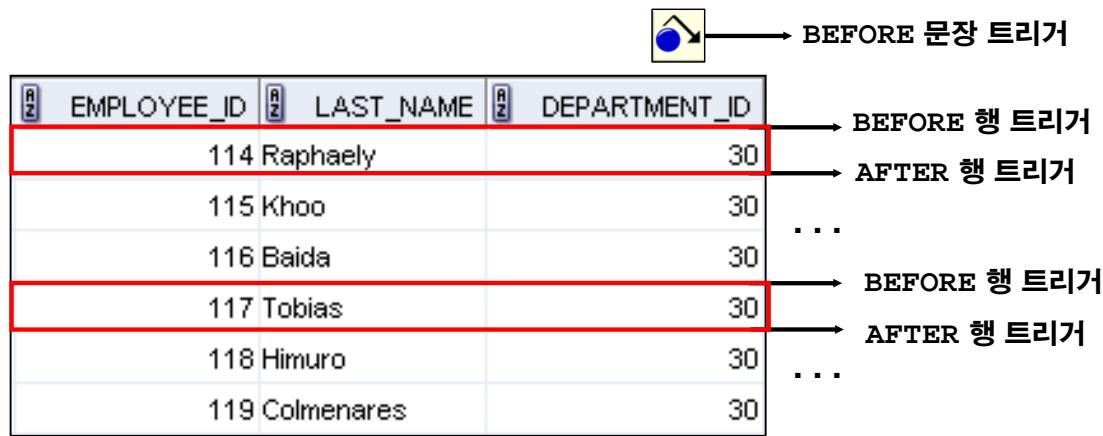
예제

슬라이드에 있는 SQL 문의 경우 표시된 `INSERT` 문의 구문을 사용하여 테이블에 정확히 한 개의 행이 삽입되었기 때문에 문장 트리거와 행 트리거가 구분되지 않습니다.

트리거 실행 시퀀스: 여러 행 조작

여러 행을 조작할 경우 테이블에서 다음 트리거 실행 시퀀스를 사용하십시오.

```
UPDATE employees
  SET salary = salary * 1.1
 WHERE department_id = 30;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

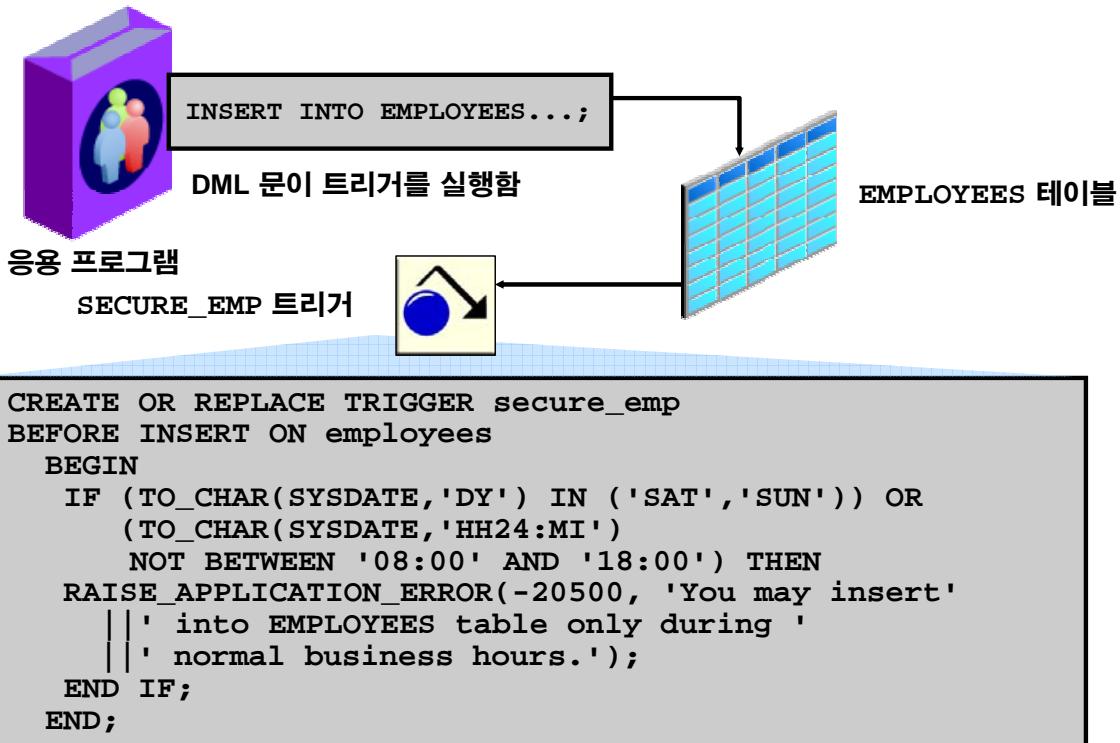
트리거 실행 시퀀스: 여러 행 조작

트리거 DML 문이 여러 행에 영향을 줄 경우 문장 트리거는 한 번만 실행되고, 행 트리거는 명령문의 영향을 받는 모든 행에 대해 한 번씩 실행됩니다.

예제

슬라이드에 있는 SQL 문의 경우 행 레벨 트리거가 WHERE 절을 만족하는 행 수, 즉 부서 30에 보고하는 사원 수와 동일한 횟수만큼 실행됩니다.

DML 문장 트리거 생성 예제: SECURE_EMP



ORACLE

Copyright © 2009, Oracle. All rights reserved.

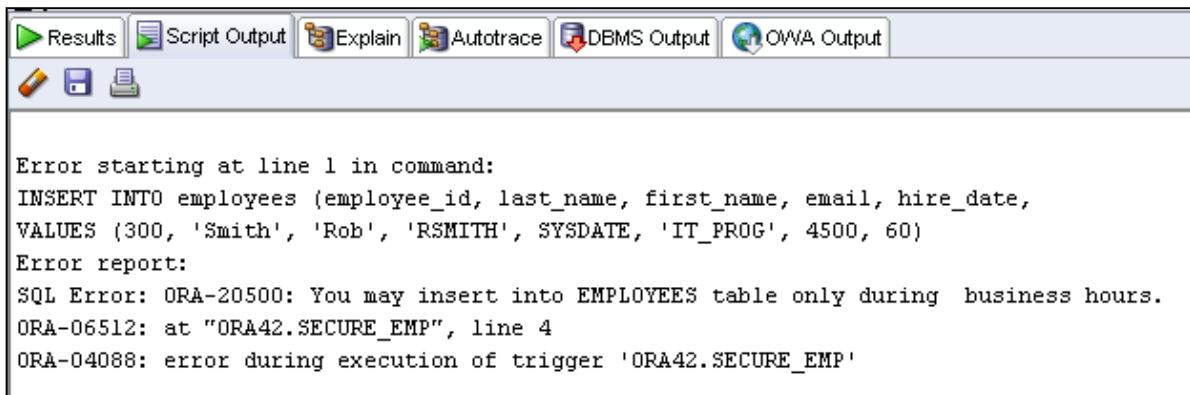
DML 문장 트리거 생성

슬라이드의 예제에서 SECURE_EMP 데이터베이스 트리거는 업무 조건을 위반할 경우 INSERT 작업이 실패하도록 하는 BEFORE 문장 트리거입니다. 이 경우 트리거는 EMPLOYEES 테이블에 대한 삽입을 특정 업무 시간(월 ~ 금) 동안으로 제한합니다.

토요일에 유저가 EMPLOYEES 테이블에 행을 삽입하려고 하면 오류 메시지가 표시되고 트리거가 실패하며 트리거 문이 롤백됩니다. RAISE_APPLICATION_ERROR는 유저에게 오류를 반환하며 PL/SQL 블록을 실패하게 하는 서버측 내장 프로시저입니다. 데이터베이스 트리거가 실패하면 트리거 문은 Oracle 서버에 의해 자동으로 롤백됩니다.

트리거 SECURE_EMP 테스트

```
INSERT INTO employees (employee_id, last_name,
    first_name, email, hire_date, job_id, salary,
    department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
    'IT_PROG', 4500, 60);
```



The screenshot shows the Oracle SQL Developer interface. The top menu bar includes 'File', 'Edit', 'Tools', 'Help', and a 'Database' dropdown. Below the menu is a toolbar with icons for 'Run', 'Script', 'Autotrace', 'DBMS Output', and 'OWA Output'. The main area displays an error message:

```
Error starting at line 1 in command:
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date,
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
Error report:
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "ORA42.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORA42.SECURE_EMP'
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SECURE_EMP 테스트

트리거를 테스트하려면 업무 시간이 아닐 때 EMPLOYEES 테이블에 행을 삽입하십시오. 날짜와 시간이 트리거에 지정된 업무 시간을 벗어나면 슬라이드에 표시된 오류 메시지가 나타납니다.

조건부 술어 사용

```

CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
BEGIN
    IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
       (TO_CHAR(SYSDATE,'HH24'))
       NOT BETWEEN '08' AND '18' ) THEN
        IF DELETING THEN RAISE_APPLICATION_ERROR(
            -20502,'You may delete from EMPLOYEES table'|||
            'only during normal business hours.');
        ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
            -20500,'You may insert into EMPLOYEES table'|||
            'only during normal business hours.');
        ELSIF UPDATING ('SALARY') THEN
            RAISE_APPLICATION_ERROR(-20503, 'You may '|||
            'update SALARY only normal during business hours.');
        ELSE RAISE_APPLICATION_ERROR(-20504,'You may'|||
            ' update EMPLOYEES table only during'|||
            ' normal business hours.');
        END IF;
    END IF;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거를 실행한 DML 작업 감지

트리거를 실행할 수 있는 DML 작업 유형이 여러 개 있을 경우(예: ON INSERT OR DELETE OR UPDATE OF Emp_tab) 트리거 본문은 조건부 술어 INSERTING, DELETING 및 UPDATING을 사용하여 트리거를 실행한 문장 유형을 확인할 수 있습니다.

트리거 본문 내에서 특수 조건부 술어 INSERTING, UPDATING 및 DELETING을 사용하여 여러 트리거 이벤트를 하나로 결합할 수 있습니다.

예제

EMPLOYEES 테이블에 대한 모든 데이터 조작 이벤트를 특정 업무 시간(월 ~ 금, 오전 8시 ~ 오후 6시)으로 제한하는 트리거를 한 개 생성하십시오.

DML 행 트리거 생성

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
        AND :NEW.salary > 15000 THEN
        RAISE_APPLICATION_ERROR (-20202,
        'Employee cannot earn more than $15,000.');
    END IF;
END;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 행 트리거 생성

특정 조건을 위반할 경우 트리거 작업이 실패하도록 하는 BEFORE 행 트리거를 생성할 수 있습니다.

첫번째 슬라이드 예제에서는 직무 ID가 AD_PRES 또는 AD_VP인 사원만 15,000이 넘는 급여를 받을 수 있도록 하는 트리거가 생성됩니다. 사원 ID가 SA_MAN인 사원 Russell의 급여를 갱신하려고 하면 트리거가 슬라이드에 표시된 예외를 발생시킵니다.

참고: 슬라이드의 첫번째 코드 예제를 실행하기 전에 secure_emp 및 secure_employees 트리거를 비활성화해야 합니다.

OLD 및 NEW 수식자 사용

- 행 레벨 트리거가 실행될 때 PL/SQL 런타임 엔진은 두 개의 데이터 구조를 생성하고 채웁니다.
 - OLD: 트리거가 처리한 레코드의 원래 값을 저장합니다.
 - NEW: 새 값을 포함합니다.
- NEW 및 OLD는 트리거가 연결된 테이블에서 %ROWTYPE을 사용하여 선언한 레코드와 구조가 동일합니다.

데이터 작업	이전 값	새 값
INSERT	NULL	삽입된 값
UPDATE	갱신 전 값	갱신 후 값
DELETE	삭제 전 값	NULL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

OLD 및 NEW 수식자 사용

ROW 트리거 내에서 데이터 앞에 OLD 및 NEW 수식자를 붙여 데이터 변경 전후의 열 값을 참조할 수 있습니다.

참고

- OLD 및 NEW 수식자는 ROW 트리거에서만 사용할 수 있습니다.
- 모든 SQL 및 PL/SQL 문에서는 이러한 수식자 앞에 콜론(:)을 붙입니다.
- WHEN 제한 조건에서 수식자가 참조될 경우에는 콜론(:) 접두어가 없습니다.
- 큰 테이블에서 여러 번 갱신하는 경우에는 행 트리거가 성능을 저하시킬 수 있습니다.

OLD 및 NEW 수식자 사용: 예제

```

CREATE TABLE audit_emp (
    user_name      VARCHAR2(30),
    time_stamp     date,
    id             NUMBER(6),
    old_last_name VARCHAR2(25),
    new_last_name VARCHAR2(25),
    old_title      VARCHAR2(10),
    new_title      VARCHAR2(10),
    old_salary     NUMBER(8,2),
    new_salary     NUMBER(8,2) )
/
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_emp(user_name, time_stamp, id,
        old_last_name, new_last_name, old_title,
        new_title, old_salary, new_salary)
    VALUES (USER, SYSDATE, :OLD.employee_id,
        :OLD.last_name, :NEW.last_name, :OLD.job_id,
        :NEW.job_id, :OLD.salary, :NEW.salary);
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

OLD 및 NEW 수식자 사용: 예제

슬라이드의 예제에서는 AUDIT_EMP_VALUES 트리거가 EMPLOYEES 테이블에 생성됩니다. 트리거는 AUDIT_EMP 유저 테이블에 행을 추가하고 EMPLOYEES 테이블에 대한 유저 작업을 기록합니다. 트리거는 OLD 및 NEW 수식자를 각각의 열 이름과 함께 사용하여 데이터 변경 전후의 여러 열 값을 기록합니다.

OLD 및 NEW 수식자 사용: 예제

```

INSERT INTO employees (employee_id, last_name, job_id,
salary, email, hire_date)
VALUES (999, 'Temp emp', 'SA_REP', 6000, 'TEMPEMP',
TRUNC(SYSDATE))
/
UPDATE employees
SET salary = 7000, last_name = 'Smith'
WHERE employee_id = 999
/
SELECT *
FROM audit_emp;

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	USER_NAME	TIME_STAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
1	ORA61	04-JUN-09	(null)	(null)	Temp emp	(null)	SA_REP	(null)	6000
2	ORA61	04-JUN-09	999	Temp emp	Smith	SA_REP	SA_REP	6000	7000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

OLD 및 NEW 수식자 사용: AUDIT_EMP 테이블 사용 예제

EMPLOYEES 테이블에 대해 유저 작업을 기록하는 유저 테이블 AUDIT_EMP에 행을 추가하는 트리거를 EMPLOYEES 테이블에 생성하십시오. 트리거는 OLD 및 NEW 수식자를 각각의 열 이름과 함께 사용하여 데이터 변경 전후의 여러 열 값을 기록합니다.

다음은 EMPLOYEES 테이블에 사원 레코드를 삽입한 결과입니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)
300	Rob	RSMITH	(null)	04-JUN-09	IT_PROG	4500	(null)	(null)	(null)	60
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110

...

다음은 사원 "Smith"의 급여를 갱신한 결과입니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999 (null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)	(null)

...

WHEN 절을 사용하여 조건을 기준으로 행 트리거 실행

```

CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
  IF INSERTING THEN
    :NEW.commission_pct := 0;
  ELSIF :OLD.commission_pct IS NULL THEN
    :NEW.commission_pct := 0;
  ELSE
    :NEW.commission_pct := :OLD.commission_pct+0.05;
  END IF;
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

행 트리거 제한: 예제

선택적으로 WHEN 절에서 부울 SQL 표현식을 지정하여 행 트리거의 정의에 트리거 제한을 포함할 수 있습니다. 트리거에 WHEN 절을 포함하면 트리거의 영향을 받는 각 행에서 WHEN 절의 표현식이 평가됩니다.

표현식이 행에 대해 TRUE로 평가되는 경우에는 트리거 본문이 해당 행에 대해 실행됩니다. 그러나 표현식이 행에 대해 FALSE 또는 NOT TRUE로 평가되는 경우(알 수 없음, null)에는 트리거 본문이 해당 행에 대해 실행되지 않습니다. WHEN 절의 평가는 트리거 SQL 문의 실행에 영향을 주지 않습니다. 다시 말해, WHEN 절의 표현식이 FALSE로 평가되면 트리거 문이 룰백되지 않습니다.

참고: 문장 트리거의 정의에 WHEN 절을 포함할 수 없습니다.

슬라이드의 예제에서는 EMPLOYEES 테이블에 행이 추가된 경우나 사원의 급여가 수정된 경우 사원의 커미션을 계산하는 트리거를 EMPLOYEES 테이블에 생성합니다.

WHEN 절이 PL/SQL 블록의 밖에 있기 때문에 NEW 수식자는 WHEN 절에서 콜론 접두어가 붙을 수 없습니다.

트리거 실행 모델 요약

- 1. 모든 BEFORE STATEMENT 트리거를 실행합니다.**
- 2. SQL 문의 영향을 받는 각 행에 대해 루프합니다.**
 - a. 해당 행에 대해 모든 BEFORE ROW 트리거를 실행합니다.**
 - b. 해당 행에 대해 DML 문을 실행하고 무결성 제약 조건 검사를 수행합니다.**
 - c. 해당 행에 대해 모든 AFTER ROW 트리거를 실행합니다.**
- 3. 모든 AFTER STATEMENT 트리거를 실행합니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 실행 모델

단일 DML 문은 잠재적으로 최대 네 가지 유형의 트리거를 실행할 수 있습니다.

- BEFORE 및 AFTER 문장 트리거
- BEFORE 및 AFTER 행 트리거

트리거 이벤트 또는 트리거 내의 명령문에서 무결성 제약 조건을 한 개 이상 검사할 수 있습니다. 그러나 COMMIT 작업이 수행될 때까지 제약 조건 검사를 연기할 수 있습니다.

트리거는 또한 다른 트리거를 실행할 수도 있는데 이를 계단식 트리거(Cascading Trigger)라고 합니다.

SQL 문의 결과로 수행된 모든 작업과 검사는 성공해야 합니다. 트리거 내에서 예외가 발생했는데 해당 예외가 명시적으로 처리되지 않은 경우 트리거를 실행하여 수행된 작업을 비롯하여 원래 SQL 문 때문에 수행된 모든 작업은 롤백됩니다. 이렇게 하면 무결성 제약 조건이 절대로 트리거에 의해 손상되지 않습니다.

트리거가 실행되면 트리거 작업에서 참조된 테이블은 다른 유저의 트랜잭션에 의해 변경될 수 있습니다. 모든 경우에 트리거가 읽기(query) 또는 쓰기(갱신)를 수행해야 하는 수정된 값에 대해 읽기 일관성 이미지가 보장됩니다.

참고: 무결성 검사는 COMMIT 작업이 수행될 때까지 연기할 수 있습니다.

After 트리거를 사용하여 무결성 제약 조건 구현

```
-- Integrity constraint violation error -2991 raised.
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
    INSERT INTO departments VALUES(:new.department_id,
        'Dept ' || :new.department_id, NULL, NULL);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        NULL; -- mask exception if department exists
END;
/
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
WHERE employee_id = 170;
```

1 rows updated

ORACLE

Copyright © 2009, Oracle. All rights reserved.

After 트리거를 사용하여 무결성 제약 조건 구현

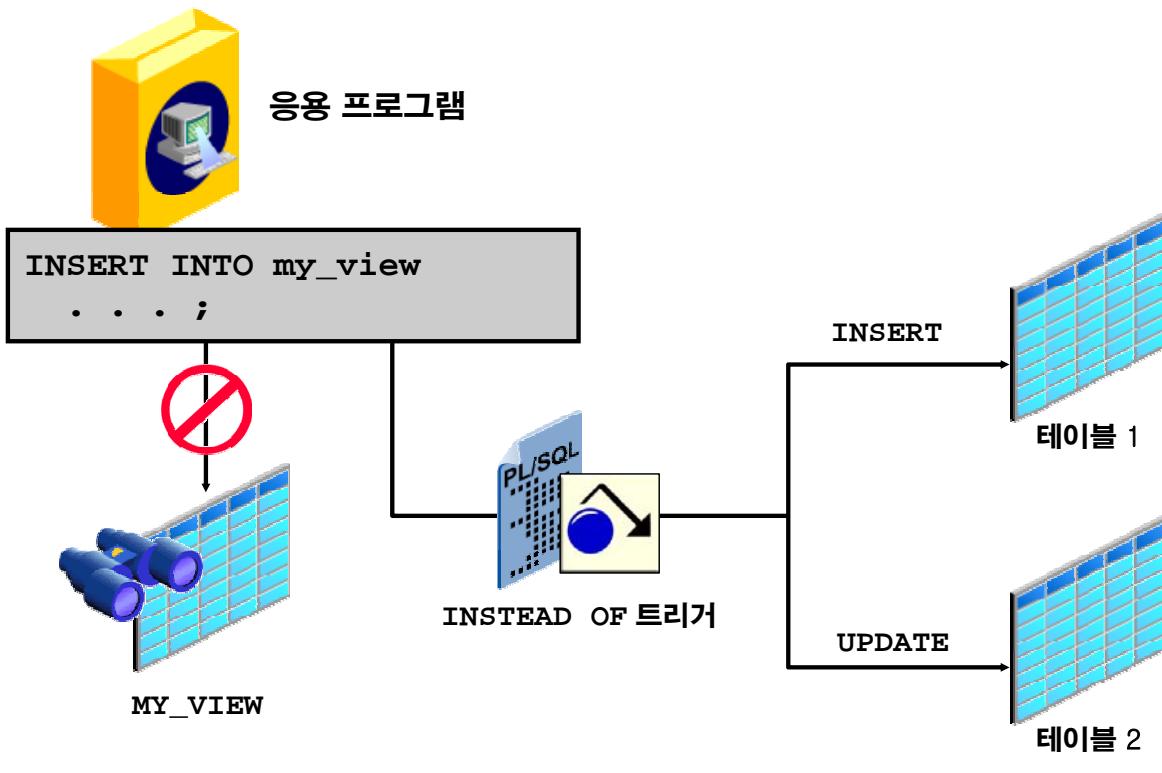
슬라이드의 예제는 AFTER 트리거를 사용하여 무결성 제약 조건을 처리할 수 있는 상황에 대해 설명합니다. EMPLOYEES 테이블에는 DEPARTMENTS 테이블의 DEPARTMENT_ID 열에 대한 Foreign Key 제약 조건이 있습니다.

첫번째 SQL 문에서 사원 170의 DEPARTMENT_ID는 999로 수정됩니다. DEPARTMENTS 테이블에는 부서 999가 없으므로 명령문은 무결성 제약 조건 위반에 해당하는 예외 -2291을 발생시킵니다.

새 부서의 DEPARTMENT_ID 값으로 :NEW.DEPARTMENT_ID를 사용하여 DEPARTMENTS 테이블에 새 행을 삽입하는 EMPLOYEE_DEPT_FK_TRG 트리거가 생성됩니다. UPDATE 문이 사원 170의 DEPARTMENT_ID를 999로 수정하면 트리거가 실행됩니다. 트리거가 DEPARTMENTS 테이블에 부서 999를 삽입했기 때문에 foreign key 제약 조건에 대한 검사가 성공합니다. 즉, 트리거가 새 행을 삽입하려고 할 때 부서가 이미 존재하지 않는 한 예외가 발생하지 않습니다. 그러나 EXCEPTION 처리기는 작업이 성공하도록 예외를 트랩하고 마스크합니다.

참고: 슬라이드에 표시된 예제는 HR 스키마의 제한된 데이터로 인해 다소 인위적이기는 하지만, 커밋될 때까지 제약 조건 검사를 지연할 경우 제약 조건 failure를 감지하여 커밋 작업 전에 복구하도록 트리거를 설계할 수 있다는 데 초점을 맞추고 있습니다.

INSTEAD OF 트리거



Copyright © 2009, Oracle. All rights reserved.

ORACLE

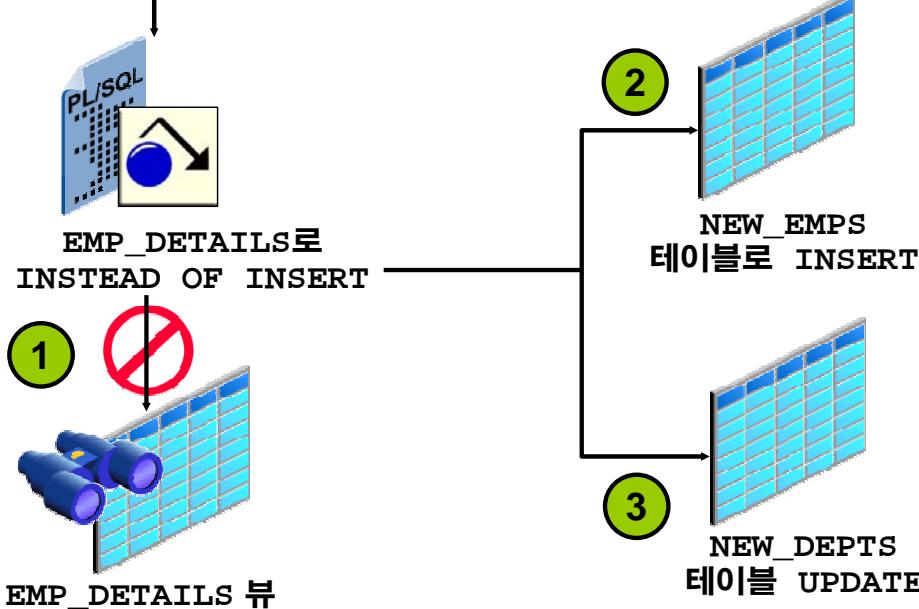
INSTEAD OF 트리거

INSTEAD OF 트리거를 사용하면 원래 생성할 수 없는 뷰에 대해 실행된 DML 문에 포함된 데이터를 수정할 수 있습니다. 다른 트리거와 달리 Oracle 서버가 트리거 문을 실행하는 대신 트리거를 실행하기 때문에 이러한 트리거를 INSTEAD OF 트리거라고 합니다. 이러한 트리거는 기본 테이블에서 INSERT, UPDATE 및 DELETE 작업을 직접 수행하는 데 사용됩니다. 뷰에 대해 INSERT, UPDATE 및 DELETE 문을 작성할 수 있습니다. 그러면 INSTEAD OF 트리거가 백그라운드에서 보이지 않게 작동하여 적절한 작업이 발생하도록 합니다. 뷰 query에 Set 연산자, 그룹 함수, GROUP BY, CONNECT BY, START와 같은 절, DISTINCT 연산자 또는 조인이 포함되어 있을 경우 일반적인 DML 문으로는 뷰를 수정할 수 없습니다. 예를 들어 뷰가 여러 개의 테이블로 구성된 경우 뷔에 대한 삽입이 한 테이블에 대한 삽입과 다른 테이블에 대한 생성으로 이어질 수 있습니다. 그러므로 뷔에 대한 삽입을 작성할 때 실행되는 INSTEAD OF 트리거를 작성하십시오. 원래의 삽입 대신 트리거 본문이 실행되고 이로 인해 한 테이블에는 데이터가 삽입되고 또 다른 테이블은 생성됩니다.

참고: 원래부터 생성 가능한 뷔에 INSTEAD OF 트리거가 있을 경우 트리거가 우선하므로, INSTEAD OF 트리거는 행 트리거입니다. INSTEAD OF 트리거를 사용하여 뷔에 대한 삽입이나 생성을 수행할 경우 뷔에 대한 CHECK 옵션은 시행되지 않습니다. INSTEAD OF 트리거 본문은 검사를 시행해야 합니다.

INSTEAD OF 트리거 생성: 예제

```
INSERT INTO emp_details
VALUES (9001,'ABBOTT',3000, 10, 'Administration');
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

INSTEAD OF 트리거 생성

뷰가 기반으로 하는 기본 테이블을 유지 관리하는 INSTEAD OF 트리거를 생성할 수 있습니다. 슬라이드의 예제는 EMPLOYEES 및 DEPARTMENTS 테이블을 기반으로 하는 query를 통해 EMP_DETAILS 뷰에 사원이 삽입되는 과정을 보여줍니다. NEW_EMP_DEPT (INSTEAD OF) 트리거는 트리거를 실행하는 INSERT 작업 대신 실행됩니다. 그런 다음 INSTEAD OF 트리거는 EMP_DETAILS 뷰에서 사용되는 기본 테이블에 적절한 INSERT 및 UPDATE를 실행합니다. 따라서 EMPLOYEES 테이블에 새로운 사원 레코드가 삽입되는 대신 다음과 같은 작업이 수행됩니다.

1. NEW_EMP_DEPT INSTEAD OF 트리거가 실행됩니다.
2. NEW_EMPS 테이블에 행이 삽입됩니다.
3. NEW_DEPTS 테이블의 DEPT_SAL 열이 갱신됩니다. 새로운 사원에 대해 제공된 급여 값이 이 사원이 소속된 부서의 기존 총 급여에 추가됩니다.

참고: 슬라이드의 예제를 실행하기 전에 다음 두 페이지에 표시된 필수 구조를 생성해야 합니다.

INSTEAD OF 트리거를 생성하여 복합 뷰에서 DML 수행

```

CREATE TABLE new_emps AS
SELECT employee_id, last_name, salary, department_id
FROM employees;

CREATE TABLE new_depts AS
SELECT d.department_id, d.department_name,
       sum(e.salary) dept_sal
  FROM employees e, departments d
 WHERE e.department_id = d.department_id;

CREATE VIEW emp_details AS
SELECT e.employee_id, e.last_name, e.salary,
       e.department_id, d.department_name
  FROM employees e, departments d
 WHERE e.department_id = d.department_id
 GROUP BY d.department_id, d.department_name;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INSTEAD OF 트리거 생성(계속)

슬라이드의 예제에서는 각각 EMPLOYEES 및 DEPARTMENTS 테이블을 기반으로 하는 NEW_EMPS 및 NEW_DEPTS라는 두 개의 새 테이블을 생성합니다. EMPLOYEES 및 DEPARTMENTS 테이블에서 EMP_DETAILS 뷰도 생성합니다.

뷰의 query 구조가 복잡할 경우 뷰에서 직접 DML을 수행해도 기본 테이블에 영향을 주지 못할 수 있습니다. 예제의 경우 다음 페이지에 표시된 NEW_EMP_DEPT라는 INSTEAD OF 트리거를 생성해야 합니다. NEW_DEPT_EMP 트리거는 DML을 다음과 같은 방식으로 처리합니다.

- EMP_DETAILS 뷰에 행이 삽입될 경우 해당 뷰에 행이 직접 삽입되는 대신 INSERT 문과 함께 제공되는 데이터 값을 사용하여 NEW_EMPS 및 NEW_DEPTS 테이블에 행이 추가됩니다.
- EMP_DETAILS 뷰를 통해 행이 수정되거나 삭제되면 NEW_EMPS 및 NEW_DEPTS 테이블에 있는 대응하는 행이 영향을 받습니다.

참고: INSTEAD OF 트리거는 뷰에 대해서만 작성할 수 있으므로 BEFORE 및 AFTER 타이밍 옵션은 유효하지 않습니다.

INSTEAD OF 트리거 생성(계속)

```

CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emps
        VALUES (:NEW.employee_id, :NEW.last_name,
                :NEW.salary, :NEW.department_id);
        UPDATE new_depts
        SET dept_sal = dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emps
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal - :OLD.salary
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('salary') THEN
        UPDATE new_emps
        SET salary = :NEW.salary
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal +
                      (:NEW.salary - :OLD.salary)
        WHERE department_id = :OLD.department_id;
    ELSIF UPDATING ('department_id') THEN
        UPDATE new_emps
        SET department_id = :NEW.department_id
        WHERE employee_id = :OLD.employee_id;
        UPDATE new_depts
        SET dept_sal = dept_sal - :OLD.salary
        WHERE department_id = :OLD.department_id;
        UPDATE new_depts
        SET dept_sal = dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    END IF;
END;

```

DEPARTMENT_ID	DEPARTMENT_NAME	DEPT_SAL
10	Administration	7400

1 rows selected

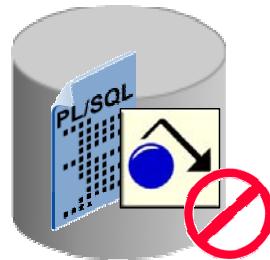
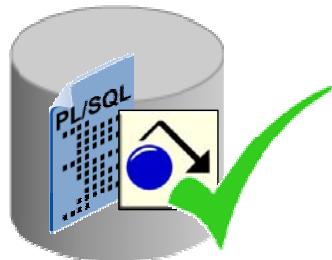
EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10
9001	ABBOTT	3000	10

2 rows selected

트리거의 상태

트리거는 두 가지 모드 중 하나입니다.

- Enabled: 트리거 문이 실행되고 트리거 제한(있는 경우)이 true(기본값)로 평가될 경우 트리거가 해당 트리거 작업을 실행합니다.
- Disabled: 트리거 문이 실행되고 트리거 제한(있는 경우)이 true로 평가될 경우에도 트리거가 해당 트리거 작업을 실행하지 않습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

비활성화된 트리거 생성

- Oracle Database 11g 이전에는 본문에 PL/SQL 컴파일 오류가 있는 트리거를 생성하면 테이블에 대한 DML이 실패했습니다.
- Oracle Database 11g에서는 비활성화된 트리거를 생성한 후 성공적으로 컴파일되는 것이 확인된 경우에만 활성화할 수 있습니다.

```
CREATE OR REPLACE TRIGGER mytrg
  BEFORE INSERT ON mytable FOR EACH ROW
  DISABLE
BEGIN
  :New.ID := my_seq.Nextval;
  . . .
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

비활성화된 트리거 생성

Oracle Database 11g 이전에는 본문에 PL/SQL 컴파일 오류가 있는 트리거를 생성하면 테이블에 대한 DML이 실패했습니다. 다음 오류 메시지가 표시됩니다.

ORA-04098: trigger 'TRG' is invalid and failed re-validation

Oracle Database 11g에서는 비활성화된 트리거를 생성한 후 성공적으로 컴파일되는 것이 확인된 경우에만 활성화할 수 있습니다.

다음과 같은 경우에 트리거를 일시적으로 비활성화할 수도 있습니다.

- 참조하는 객체를 사용할 수 없는 경우
- 큰 데이터 로드를 수행해야 하는데 트리거를 실행하지 않고 빠르게 진행하려는 경우
- 데이터를 다시 로드하는 경우

참고: 슬라이드의 코드 예제에서는 my_seq라는 기존 시퀀스가 있다고 가정합니다.

ALTER 및 DROP SQL 문을 사용하여 트리거 관리

```
-- Disable or reenable a database trigger:
```

```
ALTER TRIGGER trigger_name DISABLE | ENABLE;
```

```
-- Disable or reenable all triggers for a table:
```

```
ALTER TABLE table_name DISABLE | ENABLE ALL TRIGGERS;
```

```
-- Recompile a trigger for a table:
```

```
ALTER TRIGGER trigger_name COMPILE;
```

```
-- Remove a trigger from the database:
```

```
DROP TRIGGER trigger_name;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거 관리

트리거에는 ENABLED 및 DISABLED라는 두 가지 모드 또는 상태가 있습니다. 트리거는 처음 생성될 때 기본적으로 활성화됩니다. Oracle 서버는 트리거를 활성화하기 위해 무결성 제약 조건을 검사하여 트리거가 이 제약 조건에 위배되지 않도록 보장합니다. 또한 Oracle 서버는 query와 제약 조건에 읽기 일관성 뷰를 제공하고, 종속성을 관리하며, 트리거가 분산 데이터베이스의 원격 테이블을 갱신할 경우 2PC(2 Phase Commit) 프로세스를 제공합니다.

트리거 비활성화

ALTER TRIGGER 명령을 사용하여 트리거를 비활성화합니다. ALTER TABLE 명령을 사용하여 테이블에서 모든 트리거를 비활성화할 수도 있습니다. SQL*Loader와 같은 유ти리티를 사용하여 데이터를 대량으로 로드할 때 성능을 향상시키거나 데이터 무결성 검사를 피하기 위해 트리거를 비활성화할 수 있습니다. 네트워크 연결 실패, 디스크 고장, 오프라인 데이터 파일 또는 오프라인 테이블스페이스로 인해 현재 사용할 수 없는 데이터베이스 객체를 트리거가 참조하는 경우에도 트리거를 비활성화할 수 있습니다.

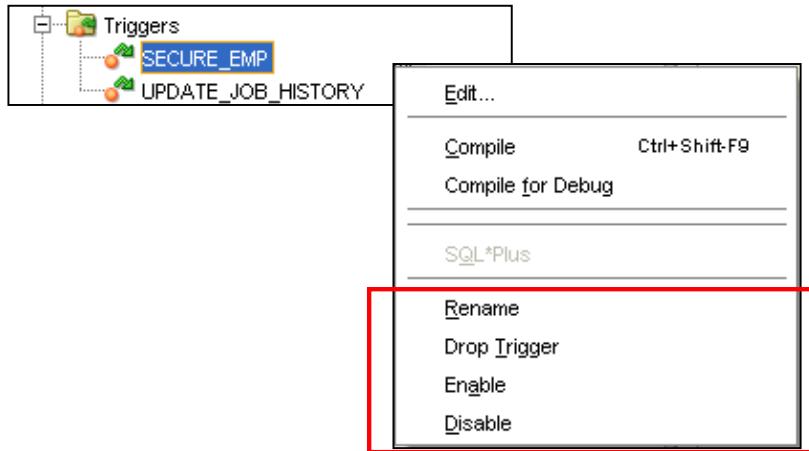
트리거 재컴파일

유효하지 않은 트리거를 명시적으로 재컴파일하려면 ALTER TRIGGER 명령을 사용합니다.

트리거 제거

트리거가 더 이상 필요하지 않으면 SQL Developer 또는 SQL*Plus에서 SQL 문을 사용하여 제거하십시오. 테이블을 제거하면 해당 테이블에 있는 모든 트리거도 제거됩니다.

SQL Developer를 사용하여 트리거 관리



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer를 사용하여 트리거 관리

Connections 탐색 트리의 Triggers 노드를 사용하여 트리거를 관리할 수 있습니다. 트리거 이름을 마우스 오른쪽 버튼으로 누르고 다음 옵션 중 하나를 선택하십시오.

- Edit
- Compile
- Compile for Debug
- Rename
- Drop Trigger
- Enable
- Disable

트리거 테스트

- 각 트리거 데이터 작업은 물론 비트리거 데이터 작업도 테스트합니다.
- WHEN 절에 대한 각각의 경우를 테스트합니다.
- 트리거는 기본 데이터 작업에서 직접 실행되도록 하고 프로시저에서도 간접적으로 실행되도록 합니다.
- 다른 트리거에서 해당 트리거의 영향을 테스트합니다.
- 해당 트리거에서 다른 트리거의 영향을 테스트합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 테스트

코드를 테스트하는 데는 많은 시간이 소모될 수 있습니다. 트리거를 테스트할 때 다음을 수행하십시오.

- 여러 가지 경우를 개별적으로 테스트하여 트리거가 적절하게 작동하는지 확인합니다.
 - 먼저 가장 일반적인 성공 시나리오를 테스트합니다.
 - 가장 일반적인 failure 상황이 적절히 관리되는지 테스트합니다.
- 트리거가 복잡해질수록 테스트 내용은 보다 세분화됩니다. 예를 들어 WHEN 절이 지정된 행 트리거가 있을 경우 조건을 만족하면 트리거가 실행되는지 확인해야 합니다. 또는 계단식 트리거가 있을 경우 한 트리거가 다른 트리거에 미치는 영향을 테스트하고 원하는 결과가 나타나는지 확인해야 합니다.
- DBMS_OUTPUT 패키지를 사용하여 트리거를 디버그합니다.

트리거 정보 보기

다음 트리거 정보를 볼 수 있습니다.

데이터 딕셔너리 뷰	설명
USER_OBJECTS	객체 정보 표시
USER/ALL/DBA_TRIGGERS	트리거 정보 표시
USER_ERRORS	트리거에 대한 PL/SQL 구문 오류 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 정보 보기

위 슬라이드는 트리거 관련 정보를 보기 위해 액세스할 수 있는 데이터 딕셔너리 뷰를 보여줍니다.

USER_OBJECTS 뷰는 트리거의 이름과 상태 및 트리거가 생성된 날짜와 시간을 포함합니다.

USER_ERRORS 뷰는 트리거가 컴파일되는 동안에 발생한 컴파일 오류의 세부 정보를 포함합니다. 이러한 뷰의 내용은 서브 프로그램의 내용과 유사합니다.

USER_TRIGGERS 뷰는 이름, 유형, 트리거 이벤트 및 트리거가 생성된 테이블, 트리거 본문과 같은 세부 정보를 포함합니다.

SELECT Username FROM USER_USERS; 문은 테이블을 생성하는 유저의 이름이 아닌 트리거의 소유자 이름을 제공합니다.

USER_TRIGGERS 사용

DESCRIBE user_triggers

Name	Null	Type
TRIGGER_NAME		VARCHAR2(30)
TRIGGER_TYPE		VARCHAR2(16)
TRIGGERING_EVENT		VARCHAR2(227)
TABLE_OWNER		VARCHAR2(30)
BASE_OBJECT_TYPE		VARCHAR2(16)
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
REFERENCING_NAMES		VARCHAR2(128)
WHEN_CLAUSE		VARCHAR2(4000)
STATUS		VARCHAR2(8)
DESCRIPTION		VARCHAR2(4000)
ACTION_TYPE		VARCHAR2(11)
TRIGGER_BODY		LONG()
CROSSEDITION		VARCHAR2(7)
BEFORE_STATEMENT		VARCHAR2(3)
BEFORE_ROW		VARCHAR2(3)
AFTER_ROW		VARCHAR2(3)
AFTER_STATEMENT		VARCHAR2(3)
INSTEAD_OF_ROW		VARCHAR2(3)
FIRE_ONCE		VARCHAR2(3)
APPLY_SERVER_ONLY		VARCHAR2(3)
21 rows selected		

```
SELECT trigger_type, trigger_body
FROM user_triggers
WHERE trigger_name = 'SECURE_EMP';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

USER_TRIGGERS 사용

소스 파일을 사용할 수 없는 경우에는 SQL Developer 또는 SQL*Plus에서 SQL Worksheet를 사용하여 USER_TRIGGERS에서 재생성 할 수 있습니다. 각각 객체의 소유자에 대해 추가 열 OWNER를 포함하는 ALL_TRIGGERS 및 DBA_TRIGGERS 뷰를 조사할 수도 있습니다. 두 번째 슬라이드 예제의 결과는 다음과 같습니다.

TRIGGER_TYPE	TRIGGER_BODY
BEFORE STATEMENT BEGIN	<pre>IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN IF DELETING THEN RAISE_APPLICATION_ERROR(-20502, 'You may delete from EMPLOYEES table only during normal business hours.'); ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(-20500, 'You may insert into EMPLOYEES table only during normal business hours.'); ELSIF UPDATING('SALARY') THEN RAISE_APPLICATION_ERROR(-20503, 'You may update SALARY only during normal business hours.'); ELSE RAISE_APPLICATION_ERROR(-20504, 'You may update EMPLOYEES table only during normal business hours.'); END IF; END IF;</pre>

퀴즈

트리거 이벤트는 다음 중 하나 이상일 수 있습니다.

- 1. 특정 테이블(또는 일부 경우 뷰)에 있는 INSERT, UPDATE 또는 DELETE 문**
- 2. 스키마 객체에 있는 CREATE, ALTER 또는 DROP 문**
- 3. 데이터베이스 시작 또는 Instance 종료**
- 4. 특정 오류 메시지 또는 임의의 오류 메시지**
- 5. 유저 로그온 또는 로그오프**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 4, 5

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- DML 작업에 의해 호출되는 데이터베이스 트리거 생성
- 문장 트리거 및 행 트리거 유형 생성
- 데이터베이스 트리거 실행 규칙 사용
- 데이터베이스 트리거 활성화, 비활성화 및 관리
- 트리거 테스트 전략 마련
- 데이터베이스 트리거 제거

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 지정된 DML 작업을 수행하기 이전이나 이후 또는 지정된 DML 작업 대신 실행되는 데이터베이스 트리거를 생성하는 방법에 대해 설명했습니다. 트리거는 데이터베이스 테이블이나 뷰와 연관됩니다. BEFORE 및 AFTER 타이밍은 테이블에 대한 DML 작업에 적용됩니다. INSTEAD OF 트리거는 뷰에 대한 DML 작업을 데이터베이스에 있는 다른 테이블에 대해 적절한 DML 문으로 대체하는 방법입니다.

트리거는 기본적으로 활성화되지만 다시 활성화될 때까지 트리거 작업이 실행되지 않도록 비활성화할 수 있습니다. 업무 규칙이 변경될 경우 필요에 따라 트리거를 제거하거나 변경할 수 있습니다.

연습 8 개요: 문장 및 행 트리거 생성

이 연습에서는 다음 내용을 다룹니다.

- 행 트리거 생성
- 문장 트리거 생성
- 트리거에서 프로시저 호출

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 8: 개요

이 연습에서는 문장 및 행 트리거를 생성합니다. 또한 트리거 내에서 호출되는 프로시저도 생성합니다.

혼합, DDL 및 데이터베이스 이벤트 트리거 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 혼합 트리거 설명
- 변경 테이블 설명
- DDL 문에 트리거 생성
- 시스템 이벤트에 트리거 생성
- 트리거에 대한 정보 표시

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 데이터베이스 트리거를 생성하고 사용하는 방법에 대해 설명합니다.

혼합 트리거란?

다음과 같은 네 개의 타이밍 지점에 대한 작업을 각각 지정하는 데 사용할 수 있는 테이블의 단일 트리거입니다.

- 실행되는 문장 앞에
- 실행되는 문장이 영향을 주는 각 행 앞에
- 실행되는 문장이 영향을 주는 각 행 뒤에
- 실행되는 문장 뒤에

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

혼합 트리거란?

Oracle Database 11g부터는 혼합 트리거를 사용할 수 있습니다. 혼합 트리거는 다음과 같은 네 개의 트리거 타이밍 지점에 대한 작업을 각각 지정하는 데 사용할 수 있는 테이블의 단일 트리거입니다.

- 실행되는 문장 앞에
- 실행되는 문장이 영향을 주는 각 행 앞에
- 실행되는 문장이 영향을 주는 각 행 뒤에
- 실행되는 문장 뒤에

참고: 트리거에 대한 자세한 내용은 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)*를 참조하십시오.

혼합 트리거 작업

- 혼합 트리거 본문은 각 타이밍 지점의 코드가 액세스할 수 있는 공통 PL/SQL 상태를 지원합니다.
- 혼합 트리거의 공통 상태는 다음과 같습니다.
 - 트리거 문이 시작되면 설정됨
 - 트리거 문이 완료되면 삭제됨
- 혼합 트리거에는 선언 섹션과 각 타이밍 포인트에 대한 섹션이 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

혼합 트리거 작업

혼합 트리거 본문은 각 타이밍 지점의 코드가 액세스할 수 있는 공통 PL/SQL 상태를 지원합니다. 공통 상태는 실행 문이 완료되면 오류가 발생하더라도 자동으로 삭제됩니다. 기록 테이블이나 감사(audit) 테이블 등의 두번째 테이블에 사용되는 행이 누적되도록 허용한 다음 해당 행을 대량 삽입하여 응용 프로그램에서 변경 테이블 오류가 발생하지 않도록 할 수 있습니다.

Oracle Database 11g Release 1 (11.1) 이전에는 부속 패키지를 사용하여 공통 상태를 모델링해야 했습니다. 그러나 이 방법은 프로그래밍하기 번거로울 뿐 아니라 실행 문에 오류가 발생하여 after-statement 트리거가 실행되지 않는 경우에는 메모리 누수의 원인이 되었습니다. 혼합 트리거를 사용하면 PL/SQL을 보다 간편하게 사용할 수 있으며 런타임 성능과 확장성이 개선됩니다.

혼합 트리거 사용 시의 이점

혼합 트리거를 사용하여 다음을 수행할 수 있습니다.

- 다양한 타이밍 지점에 대해 구현하는 여러 작업에서 공통되는 데이터를 공유하도록 하는 접근 방법을 프로그래밍합니다.
- 두번째 테이블에 사용되는 행을 누적하여 주기적으로 대량 삽입합니다.
- 두번째 테이블에 사용되는 행이 누적되도록 허용한 후 대량 삽입하여 변경 테이블 오류(ORA-04091)를 방지합니다.



Copyright © 2009, Oracle. All rights reserved.

테이블 혼합 트리거의 타이밍 지점 섹션

테이블에 대해 정의된 혼합 트리거에는 다음 타이밍 지점 섹션이 하나 이상 포함됩니다. 타이밍 지점 섹션은 테이블에 나와 있는 순서대로 표시되어야 합니다.

타이밍 지점	혼합 트리거 섹션
트리거 문 실행 앞에	BEFORE 문장
트리거 문 실행 뒤에	AFTER 문장
트리거 문이 영향을 주는 각 행 앞에	BEFORE EACH ROW
트리거 문이 영향을 주는 각 행 뒤에	AFTER EACH ROW

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고

타이밍 지점 섹션은 슬라이드에 나와 있는 순서대로 표시되어야 합니다. 타이밍 지점 섹션이 없으면 타이밍 지점에서 아무 작업도 수행되지 않습니다.

테이블을 위한 혼합 트리거 구조

```
CREATE OR REPLACE TRIGGER schema.trigger
FOR dml_event_clause ON schema.table
COMPOUND TRIGGER
```

```
-- Initial section
-- Declarations
-- Subprograms
```

1

```
-- Optional section
BEFORE STATEMENT IS ...;
```

2

```
-- Optional section
AFTER STATEMENT IS ...;
```

```
-- Optional section
BEFORE EACH ROW IS ...;
```

```
-- Optional section
AFTER EACH ROW IS ...;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블을 위한 혼합 트리거 구조

혼합 트리거에는 다음과 같은 두 개의 주 섹션이 있습니다.

- 변수 및 서브 프로그램이 선언되는 초기 섹션. 이 섹션의 코드는 옵션 섹션의 모든 코드보다 먼저 실행됩니다.
- 사용 가능한 각 트리거 지점에 대해 코드를 정의하는 옵션 섹션. 이러한 트리거 지점은 혼합 트리거를 테이블에 대해 정의하는지, 뷰에 대해 정의하는지에 따라 달라집니다. 테이블과 뷰 각각에 대한 구조는 위 페이지와 다음 페이지의 이미지에 나와 있습니다. 트리거 지점용 코드는 위에 나와 있는 순서를 따라야 합니다.

참고: 혼합 트리거에 대한 자세한 내용은 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

뷰를 위한 혼합 트리거 구조

```
CREATE OR REPLACE TRIGGER
schema.trigger
FOR dml_event_clause ON schema.view
COMPOUND TRIGGER
    -- Initial section
    -- Declarations
    -- Subprograms
    -- Optional section (exclusive)
    INSTEAD OF EACH ROW IS
    ...;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

뷰를 위한 혼합 트리거 구조

뷰의 경우 허용되는 섹션은 INSTEAD OF EACH ROW 절뿐입니다.

혼합 트리거의 제한 사항

- 혼합 트리거는 DML 트리거여야 하며 테이블이나 뷰에 정의해야 합니다.
- 혼합 트리거의 본문은 PL/SQL에서 작성한 혼합 트리거 블록이어야 합니다.
- 혼합 트리거 본문에는 초기화 블록이 포함될 수 없으므로 예외 �セ션이 있을 수 없습니다.
- 한 셜션에서 발생하는 예외는 해당 셜션에서 처리되어야 합니다. 다른 셜션에서 처리하도록 권한을 이전할 수 없습니다.
- :OLD 및 :NEW는 선언, BEFORE STATEMENT 또는 AFTER STATEMENT 셜션에 나타날 수 없습니다.
- BEFORE EACH ROW 셜션만 :NEW 값을 변경할 수 있습니다.
- FOLLOWS 절을 사용하지 않으면 혼합 트리거의 실행 순서가 일정하지 않습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

혼합 트리거의 제한 사항

다음은 혼합 트리거를 사용할 때의 몇 가지 제한 사항입니다.

- 혼합 트리거의 본문은 PL/SQL에서 작성한 혼합 트리거 블록이어야 합니다.
- 혼합 트리거는 DML 트리거여야 합니다.
- 혼합 트리거는 테이블이나 뷰에 정의해야 합니다.
- 혼합 트리거 본문에는 초기화 블록이 포함될 수 없으므로 예외 셜션이 있을 수 없습니다. BEFORE STATEMENT 셜션은 다른 모든 타이밍 지점 셜션이 실행되기 전에 한 번만 실행되므로 이는 문제가 되지 않습니다.
- 한 셜션에서 발생하는 예외는 해당 셜션에서 처리되어야 합니다. 다른 셜션에서 처리하도록 권한을 이전할 수 없습니다.
- :OLD, :NEW 및 :PARENT는 선언 셜션, BEFORE STATEMENT 셜션 또는 AFTER STATEMENT 셜션에 나타날 수 없습니다.
- FOLLOWS 절을 사용하지 않으면 혼합 트리거의 실행 순서가 일정하지 않습니다.

변경 테이블의 트리거 제한 사항

- **변경 테이블은 다음과 같은 테이블입니다.**
 - UPDATE, DELETE 또는 INSERT 문을 사용하여 수정하는 테이블 또는
 - DELETE CASCADE 제약 조건의 결과로 인해 생성될 수 있는 테이블
- **트리거 문을 실행한 세션은 변경 테이블을 query하거나 수정할 수 없습니다.**
- **이 제한 사항은 트리거에 일관성 없는 데이터 집합이 표시되지 않도록 합니다.**
- **이 제한 사항은 FOR EACH ROW 절을 사용하는 모든 트리거에 적용됩니다.**
- **INSTEAD OF 트리거에서 수정되는 뷰는 변경 테이블로 간주되지 않습니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거 제어 규칙

트리거를 사용하여 데이터를 읽고 쓰기 위해서는 특정 규칙을 따라야 합니다. 문장 트리거가 ON DELETE CASCADE의 결과로 실행되는 경우를 제외하고, 제한 사항은 행 트리거에만 적용됩니다.

Mutating Table

변경 테이블은 UPDATE, DELETE 또는 INSERT 문에 의해 현재 수정되고 있는 테이블이거나, 선언적 DELETE CASCADE 참조 무결성 작업의 결과에 따라 생성되어야 하는 테이블입니다. STATEMENT 트리거의 경우 테이블을 변경 테이블로 간주하지 않습니다.

변경 테이블 오류(ORA-4091)는 행 레벨 트리거가 이미 DML 문을 통해 변경 중인 테이블을 변경하거나 검사하려는 경우에 발생합니다.

트리거된 테이블 자체는 변경 테이블이며, FOREIGN KEY 제약 조건을 사용하여 이 테이블을 참조하는 테이블도 변경 테이블입니다. 이 제한 사항은 행 트리거에 일관성 없는 데이터 집합이 표시되지 않도록 합니다.

변경 테이블: 예제

```
CREATE OR REPLACE TRIGGER check_salary
BEFORE INSERT OR UPDATE OF salary, job_id
ON employees
FOR EACH ROW
WHEN (NEW.job_id <> 'AD_PRES')
DECLARE
v_minsalary employees.salary%TYPE;
v_maxsalary employees.salary%TYPE;
BEGIN
SELECT MIN(salary), MAX(salary)
INTO v_minsalary, v_maxsalary
FROM employees
WHERE job_id = :NEW.job_id;
IF :NEW.salary < v_minsalary OR :NEW.salary > v_maxsalary THEN
RAISE_APPLICATION_ERROR(-20505,'Out of range');
END IF;
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변경 테이블: 예제

슬라이드 예제에서 CHECK_SALARY 트리거는 EMPLOYEES 테이블에 새로운 사원이 추가되거나 기존 사원의 급여나 직무 ID가 변경될 때마다 사원의 급여가 해당 직무에 대해 설정된 급여 범위 내에 있는지 확인합니다.

사원 레코드 생성 시 생성되는 행마다 CHECK_SALARY 트리거가 실행됩니다. 트리거 코드는 생성 중인 동일한 테이블을 query합니다. 따라서 EMPLOYEES 테이블이 변경 테이블입니다.

변경 테이블: 예제

```
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
```

```
TRIGGER check_salary Compiled.

Error starting at line 1 in command:
UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles'
Error report:
SQL Error: ORA-04091: table ORA42.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA42.CHECK_SALARY", line 5
ORA-04088: error during execution of trigger 'ORA42.CHECK_SALARY'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
          this statement) attempted to look at (or modify) a table that was
          in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

변경 테이블: 예제(계속)

슬라이드의 예제에서 트리거 코드는 변경 테이블에서 데이터를 읽거나 선택하려고 합니다. 급여를 기존 최소값과 기존 최대값 사이의 범위로 제한하면 런타임 오류가 발생합니다. EMPLOYEES 테이블은 변경 중이거나 변경 상태이므로 트리거가 해당 테이블에서 읽을 수 없습니다.

함수가 DML 문에서 호출된 경우에도 함수에 변경 테이블 오류가 발생할 수 있습니다.

해결 방법

이와 같은 변경 테이블 문제를 해결하는 방법은 다음과 같습니다.

- 이 단원 앞부분에서 설명한 대로 혼합 트리거를 사용합니다.
- 다른 DML 트리거를 사용하여 최신 상태를 유지하는 다른 요약 테이블에 요약 데이터(최소 급여 및 최대 급여)를 저장합니다.
- 요약 데이터를 PL/SQL 패키지에 저장한 다음 해당 패키지의 데이터에 액세스합니다. 이 작업은 BEFORE 문장 트리거에서 수행할 수 있습니다.

문제의 특성에 따라 해결 방법이 보다 복잡해질 수도 있고 해결하기 어려울 수도 있습니다.

이 경우 응용 프로그램이나 Middle-tier에서 규칙을 구현하고 데이터베이스 트리거를 사용하여 지나치게 복잡한 업무 규칙을 수행하지 않도록 하십시오. 슬라이드 코드 예제의 insert 문은 변경 테이블 예제를 생성하지 않습니다.

혼합 트리거를 사용하여 변경 테이블 오류 해결

```

CREATE OR REPLACE TRIGGER check_salary
FOR INSERT OR UPDATE OF salary, job_id
ON employees
WHEN (NEW.job_id <> 'AD_PRES')
COMPOUND TRIGGER

    TYPE salaries_t          IS TABLE OF employees.salary%TYPE;
    min_salaries             salaries_t;
    max_salaries             salaries_t;

    TYPE department_ids_t    IS TABLE OF employees.department_id%TYPE;
    department_ids           department_ids_t;

    TYPE department_salaries_t IS TABLE OF employees.salary%TYPE
                                INDEX BY VARCHAR2(80);
    department_min_salaries  department_salaries_t;
    department_max_salaries  department_salaries_t;

-- example continues on next slide

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

혼합 트리거를 사용하여 변경 테이블 오류 해결

CHECK_SALARY 혼합 트리거는 앞서 소개한 예제에서 발생하는 변경 테이블 오류를 해결합니다. 이 작업을 수행하려면 PL/SQL 컬렉션에 값을 저장한 후 혼합 트리거의 "before statement" 섹션에서 대량 삽입/갱신을 수행하면 됩니다. 슬라이드의 예제에서는 PL/SQL 컬렉션을 사용합니다. 사용되는 요소 유형은 EMPLOYEES 테이블의 SALARY 및 DEPARTMENT_ID 열을 기준으로 합니다. 컬렉션을 생성하려면 컬렉션 유형을 정의한 후 해당 유형의 변수를 선언합니다. 컬렉션은 블록이나 서브 프로그램을 시작하면 인스턴스화되며 블록이나 서브 프로그램을 종료하면 삭제됩니다. min_salaries는 각 부서의 최소 급여를 보관하는 데 사용되며 max_salaries는 각 부서의 최대 급여를 보관하는 데 사용됩니다. department_ids는 부서 ID를 보관하는 데 사용됩니다. 최소 또는 최대 급여를 받는 사원에 대해 부서가 할당되어 있지 않은 경우에는 NVL 함수를 사용하여 부서 ID에 대해 NULL 대신 -1을 저장합니다. 다음으로는 대량 삽입을 사용하여 부서 ID별로 각각 그룹화되어 있는 min_salaries, max_salaries 및 department_ids에 최소 급여, 최대 급여 및 부서 ID를 수집합니다. select 문은 13개의 행을 반환합니다. department_ids의 값은 department_min_salaries 및 department_max_salaries 테이블의 인덱스로 사용됩니다. 그러므로 이 두 테이블의 인덱스(VARCHAR2)는 실제 department_ids를 나타냅니다.

혼합 트리거를 사용하여 변경 테이블 오류 해결

```

. . .
BEFORE STATEMENT IS
BEGIN
    SELECT MIN(salary), MAX(salary), NVL(department_id, -1)
    BULK COLLECT INTO min_Salaries, max_salaries, department_ids
    FROM employees
    GROUP BY department_id;
    FOR j IN 1..department_ids.COUNT() LOOP
        department_min_salaries(department_ids(j)) := min_salaries(j);
        department_max_salaries(department_ids(j)) := max_salaries(j);
    END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
    IF :NEW.salary < department_min_salaries(:NEW.department_id)
    OR :NEW.salary > department_max_salaries(:NEW.department_id) THEN
        RAISE_APPLICATION_ERROR(-20505,'New Salary is out of acceptable
                                         range');
    END IF;
END AFTER EACH ROW;
END check_salary;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

혼합 트리거를 사용하여 변경 테이블 오류 해결(계속)

각 행을 추가한 후 새 급여가 해당 부서의 최소 급여보다 적거나 최대 급여보다 많은 경우에는 오류 메시지가 표시됩니다.

새로 생성한 혼합 트리거를 테스트하려면 다음 명령문을 실행합니다.

```

UPDATE employees
SET salary = 3400
WHERE last_name = 'Stiles';
1 rows updated

```

사원 Stiles의 급여가 갱신되었는지 확인하려면 SQL Developer에서 F9 키를 사용하여 다음 query를 실행합니다.

```

SELECT employee_id, first_name, last_name, job_id, department_id,
       salary
  FROM employees
 WHERE last_name = 'Stiles';

```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID	DEPARTMENT_ID	SALARY
1	138	Stephen	Stiles	ST_CLERK	50	3400

DDL 문에 트리거 생성

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER -- Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

예제 DDL 이벤트	실행 시기
CREATE	임의 데이터베이스 객체가 CREATE 명령을 사용하여 생성됩니다.
ALTER	임의 데이터베이스 객체가 ALTER 명령을 사용하여 변경됩니다.
DROP	임의 데이터베이스 객체가 DROP 명령을 사용하여 삭제됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DDL 문에 트리거 생성

트리거가 실행되도록 하는 DDL 문의 유형을 하나 이상 지정할 수 있습니다. 다른 설명이 없는 경우에는 DATABASE 또는 SCHEMA에 이러한 이벤트에 대한 트리거를 생성할 수 있습니다. 또한 트리거 타이밍에 대해 BEFORE 및 AFTER를 지정할 수도 있습니다. 오라클 데이터베이스는 기존 유저 트랜잭션에서 트리거를 실행합니다.

PL/SQL 프로시저를 통해 수행하는 DDL 작업은 트리거 이벤트로 지정할 수 없습니다.

슬라이드에서 구문의 트리거 본문은 전체 PL/SQL 블록을 나타냅니다.

생성되는 객체가 클러스터, 함수, 인덱스, 패키지, 프로시저, 룰, 시퀀스, 동의어, 테이블, 테이블스페이스, 트리거, 유형, 뷰 또는 유저일 경우에만 DDL 트리거가 실행됩니다.

데이터베이스 이벤트 트리거 생성

- **유저 이벤트 트리거:**
 - CREATE, ALTER 또는 DROP
 - 로그온 또는 로그오프
- **데이터베이스 또는 시스템 이벤트 트리거:**
 - 데이터베이스 종료 또는 시작
 - 발생한 특정 오류 (또는 임의의 오류)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 트리거 생성

트리거 본문을 코딩하려면 먼저 트리거 구성 요소를 결정하십시오.

시스템 이벤트 트리거는 데이터베이스 또는 스키마 레벨에서 정의될 수 있습니다. 예를 들면, 데이터베이스 종료 트리거는 데이터베이스 레벨에서 정의됩니다. DDL (데이터 정의어) 문장 트리거나 유저 로그온 또는 로그오프 트리거도 데이터베이스 레벨이나 스키마 레벨에서 정의될 수 있습니다. DML (데이터 조작어) 문장 트리거는 특정 테이블이나 뷰에서 정의됩니다.

데이터베이스 레벨에서 정의된 트리거는 모든 유저에 대해 실행되고, 스키마 또는 테이블 레벨에서 정의된 트리거는 트리거 이벤트가 해당 스키마나 테이블을 포함할 경우에만 실행됩니다.

트리거 실행을 유발하는 트리거 이벤트는 다음과 같습니다.

- 데이터베이스나 스키마에 있는 객체에 대한 데이터 정의문
- 특정 유저(임의의 유저) 로그온 또는 로그오프
- 데이터베이스 종료 또는 시작
- 발생한 오류

시스템 이벤트에 트리거 생성

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE | AFTER -- timing
[database_event1 [OR database_event2 OR ...]]
ON {DATABASE | SCHEMA}
trigger_body
```

데이터베이스 이벤트	트리거 실행 시기
AFTER SERVERERROR	오라클 오류가 발생합니다.
AFTER LOGON	유저가 데이터베이스에 로그온합니다.
BEFORE LOGOFF	유저가 데이터베이스를 로그오프합니다.
AFTER STARTUP	데이터베이스가 열립니다.
BEFORE SHUTDOWN	데이터베이스가 정상적으로 종료됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시스템 이벤트에 트리거 생성

DATABASE 또는 SCHEMA에 대해 슬라이드의 표에 나와 있는 이벤트의 트리거를 생성할 수 있습니다. 단, DATABASE에만 적용되는 SHUTDOWN 및 STARTUP은 예외입니다.

LOGON 및 LOGOFF 트리거: 예제

```
-- Create the log_trig_table shown in the notes page
-- first

CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LOGON 및 LOGOFF 트리거: 예제

로그온 및 로그오프 횟수를 모니터하는 LOGON 및 LOGOFF 트리거를 생성하거나 로그온 시간 길이를 모니터하는 보고서를 작성할 수도 있습니다. ON SCHEMA를 지정하면 트리거가 특정 유저에 대해 실행되고, ON DATABASE를 지정하면 트리거가 모든 유저에 대해 실행됩니다.

슬라이드 예제에 사용되는 log_trig_table의 정의는 다음과 같습니다.

```
CREATE TABLE log_trig_table(
    user_id VARCHAR2(30),
    log_date DATE,
    action VARCHAR2(40))
/
```

트리거의 CALL 문

```
CREATE [OR REPLACE] TRIGGER trigger_name  
timing  
event1 [OR event2 OR event3]  
ON table_name  
[REFERENCING OLD AS old | NEW AS new]  
[FOR EACH ROW]  
[WHEN condition]  
CALL procedure_name  
/
```

```
CREATE OR REPLACE PROCEDURE log_execution IS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE('log_execution: Employee Inserted');  
END;  
/  
CREATE OR REPLACE TRIGGER log_employee  
BEFORE INSERT ON EMPLOYEES  
CALL log_execution -- no semicolon needed  
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거의 CALL 문

CALL 문을 사용하면 트리거 자체에서 PL/SQL 본문을 코딩하는 대신 내장 프로시저를 호출할 수 있습니다. 프로시저는 PL/SQL, C 또는 Java로 구현할 수 있습니다.

다음 예제에서와 같이 호출은 트리거 속성 :NEW 및 :OLD를 파라미터로 참조할 수 있습니다.

```
CREATE OR REPLACE TRIGGER salary_check  
BEFORE UPDATE OF salary, job_id ON employees  
FOR EACH ROW  
WHEN (NEW.job_id <> 'AD_PRES')  
CALL check_salary(:NEW.job_id, :NEW.salary)
```

참고: CALL 문 끝에는 세미콜론(;)이 없습니다.

위 예제에서 트리거는 check_salary 프로시저를 호출합니다. 이 프로시저는 JOBS 테이블에 있는 새 직무 ID의 급여 범위와 새 급여를 비교합니다.

데이터베이스 이벤트 트리거의 이점

- 데이터 보안 향상:
 - 복합적인 고급 보안 검사를 제공합니다.
 - 복합적인 고급 감사를 제공합니다.
- 데이터 무결성 향상:
 - 동적 데이터 무결성 제약 조건을 적용합니다.
 - 복합적인 참조 무결성 제약 조건을 적용합니다.
 - 관련 작업이 암시적으로 함께 수행되도록 합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 이벤트 트리거의 이점

데이터베이스 트리거는 다음과 같은 경우에 사용할 수 있습니다.

- Oracle 서버에서 제공하는 기능의 대체 기능으로
- 작업 요구 사항이 Oracle 서버에서 제공하는 요구 사항보다 복잡하거나 간단할 경우
- 작업 요구 사항을 Oracle 서버에서 제공하지 않을 경우

트리거를 관리하는 데 필요한 시스템 권한

트리거를 관리하려면 다음과 같은 시스템 권한이 필요합니다.

- **스키마에서 트리거를 생성, 변경 및 삭제할 수 있는 권한:**
 - GRANT CREATE TRIGGER TO ora61
 - GRANT ALTER ANY TRIGGER TO ora61
 - GRANT DROP ANY TRIGGER TO ora61
- **데이터베이스에서 트리거를 생성할 수 있는 권한:**
 - GRANT ADMINISTER DATABASE TRIGGER TO ora61
- **EXECUTE 권한 (트리거가 스키마에 포함되지 않은 객체를 참조하는 경우)**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거를 관리하는 데 필요한 시스템 권한

자신의 스키마에서 트리거를 생성하려면 CREATE TRIGGER 시스템 권한이 필요하며 트리거 문에 지정된 테이블을 소유하거나, 트리거 문의 테이블에 대한 ALTER 권한을 가지고 있거나, ALTER ANY TABLE 시스템 권한을 가지고 있어야 합니다. 자신의 트리거는 추가 권한 없이 변경하거나 삭제할 수 있습니다.

ANY 키워드를 사용하면 자신의 트리거와 다른 스키마의 트리거를 생성, 변경 또는 삭제할 수 있으며, 다른 유저의 테이블과 연관시킬 수 있습니다.

자신의 스키마에서 트리거를 호출할 때는 권한이 필요하지 않습니다. 트리거는 자신이 실행한 DML 문에 의해 호출됩니다. 그러나 트리거가 자신의 스키마에 포함되지 않은 객체를 참조할 경우 트리거를 생성하는 유저는 참조된 프로시저, 함수 또는 패키지에 대한 EXECUTE 권한(롤이 아님)을 가지고 있어야 합니다.

DATABASE에 트리거를 생성하려면 ADMINISTER DATABASE TRIGGER 권한이 있어야 합니다. 나중에 이 권한을 취소하면 트리거를 삭제할 수 있지만 변경할 수는 없습니다.

참고: 내장 프로시저와 마찬가지로, 트리거 본문에 있는 명령문은 트리거를 실행하는 유저의 권한이 아닌 트리거 소유자의 권한을 사용합니다.

트리거 설계 지침

- **트리거를 설계하여 다음 작업을 수행할 수 있습니다.**
 - 관련 작업 수행
 - 전역(global) 작업 중앙화
- **다음과 같은 경우에는 트리거를 설계하면 안됩니다.**
 - Oracle 서버에 이미 기능이 내장되어 있는 경우
 - 다른 트리거와 중복되는 경우
- **PL/SQL 코드가 매우 길 경우 내장 프로시저를 생성한 다음 이를 트리거에서 호출할 수 있습니다.**
- **트리거를 지나치게 많이 사용하면 상호 종속성이 복잡해져서 대용량 응용 프로그램에서 유지 관리하기 어려울 수 있습니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거 설계 지침

- 트리거를 사용하면 명령문을 실행하는 유저 또는 응용 프로그램에 관계없이 트리거 문에 대해 실행되어야 하는 특정 작업 및 중앙의 전역(global) 작업에 대해 관련 작업이 수행되도록 할 수 있습니다.
- 오라클 데이터베이스에 이미 내장되어 있는 기능과 중복되거나 이 기능을 대체하도록 트리거를 정의해서는 안됩니다. 예를 들어 트리거 대신 선언 제약 조건을 사용하여 무결성 규칙을 구현하십시오. 업무 규칙을 구현하려면 다음의 설계 순서를 따르십시오.
 - Primary Key와 같은 Oracle 서버에 내장된 제약 조건을 사용합니다.
 - Middle-tier에서 서블릿이나 EJB(Enterprise JavaBeans)와 같은 응용 프로그램 또는 데이터베이스 트리거를 개발합니다.
 - 데이터 표시 규칙에 대해서는 Oracle Forms, HTML, JSP(JavaServer Pages) 등의 표시 인터페이스를 사용합니다.
- 트리거를 지나치게 많이 사용하면 상호 종속성이 복잡해져서 유지 관리하기 힘들 수 있습니다. 반드시 필요한 경우에만 트리거를 사용하고, 연쇄적인 recursive 결과에 유의하십시오.
- 트리거 본문에서 호출되는 내장 프로시저나 패키지화된 프로시저를 생성하면 트리거 본문 논리가 길어지는 것을 피할 수 있습니다.
- 데이터베이스 트리거는 생성된 트리거에서 이벤트가 발생할 때마다 모든 유저에 대해 실행됩니다.

퀴즈

1. 트리거가 CREATE TRIGGER 문을 사용하여 정의됩니다.
2. 트리거의 소스 코드가 USER_TRIGGERS 데이터 딕셔너리에 포함됩니다.
3. 트리거가 명시적으로 호출됩니다.
4. 트리거가 DML에 의해 암시적으로 호출됩니다.
5. 트리거를 사용할 때 COMMIT, SAVEPOINT 및 ROLLBACK이 허용되지 않습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 4, 5

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 혼합 트리거 설명
- 변경 테이블 설명
- DDL 문에 트리거 생성
- 시스템 이벤트에 트리거 생성
- 트리거에 대한 정보 표시



Copyright © 2009, Oracle. All rights reserved.

연습 9: 개요

이 연습에서는 다음 내용을 다룹니다.

- 데이터 무결성 규칙을 관리하는 고급 트리거 생성
- 변경 테이블 예외를 유발하는 트리거 생성
- 패키지 상태를 사용하여 변경 테이블 문제를 해결하는 트리거 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 9: 개요

이 연습에서는 직무에 유효한 급여 범위와 관련하여 사원 급여의 데이터 무결성을 보장하는 간단한 업무 규칙을 구현하고, 이 규칙에 대한 트리거를 생성하는 과정을 다룹니다.

이 프로세스 동안 새 트리거가 이전 단원의 연습 섹션에서 생성한 트리거와 함께 연쇄적인 결과를 일으킵니다. 이 연쇄적인 결과로 인해 JOBS 테이블에 대한 변경 테이블 예외가 발생합니다. 그러면 PL/SQL 패키지와 추가 트리거를 생성하여 변경 테이블 문제를 해결합니다.

10

PL/SQL 컴파일러 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- PL/SQL 컴파일러 초기화 파라미터 사용
- PL/SQL 컴파일 타임 경고 사용



Copyright © 2009, Oracle. All rights reserved.

단원 내용

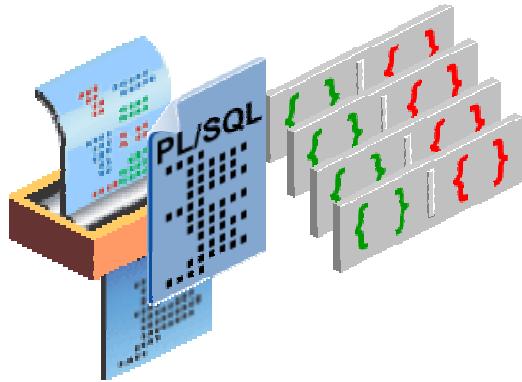
- **PLSQL_CODE_TYPE 및 PLSQL_OPTIMIZE_LEVEL**
PL/SQL 컴파일 초기화 파라미터 사용
- **PL/SQL 컴파일 타임 경고 사용:**
 - **PLSQL_WARNING 초기화 파라미터 사용**
 - **DBMS_WARNING 패키지 서브 프로그램 사용**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 컴파일용 초기화 파라미터

- **PLSQL_CODE_TYPE**
- **PLSQL_OPTIMIZE_LEVEL**
- **PLSQL_CCFLAGS**
- **PLSQL_WARNINGS**



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 컴파일용 초기화 파라미터

Oracle Database 10g 이전 버전에서 PL/SQL 컴파일러는 성능을 향상시키기 위한 다수의 변경 사항을 적용하지 않고 코드를 기계 코드로 변환했습니다. 그러나 이제 PL/SQL에서는 코드를 재배열하여 성능을 향상시킬 수 있는 최적화 컴파일러를 사용합니다. 이 새로운 옵티マイ저는 기본적으로 사용되며 때문에 해당 기능을 사용하기 위해 추가 작업을 수행할 필요가 없습니다.

참고

- PLSQL_CCFLAGS 초기화 파라미터는 "PL/SQL 코드 관리" 단원에서 다룹니다.
- PLSQL_WARNINGS 초기화 파라미터는 이 단원 뒷부분에서 다룹니다.

PL/SQL 컴파일을 위해 초기화 파라미터 사용

- **PLSQL_CODE_TYPE:** PL/SQL 라이브러리 단위에 대해 컴파일 모드를 지정합니다.

```
PLSQL_CODE_TYPE = { INTERPRETED | NATIVE }
```

- **PLSQL_OPTIMIZE_LEVEL:** PL/SQL 라이브러리 단위를 컴파일하는 데 사용할 최적화 레벨을 지정합니다.

```
PLSQL_OPTIMIZE_LEVEL = { 0 | 1 | 2 | 3 }
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 컴파일을 위해 초기화 파라미터 사용

PLSQL_CODE_TYPE 파라미터

이 파라미터는 PL/SQL 라이브러리 단위에 대해 컴파일 모드를 지정합니다. INTERPRETED를 선택하는 경우 PL/SQL 라이브러리 단위는 PL/SQL 바이트 코드 형식으로 컴파일됩니다. 이러한 모듈은 PL/SQL 인터프리터 엔진에서 실행합니다. NATIVE를 선택하는 경우 PL/SQL 라이브러리 단위(최상위 익명 PL/SQL 블록은 제외될 수 있음)는 Native(기계) 코드로 컴파일됩니다. 이러한 모듈은 인터프리터 오버헤드를 발생시키지 않고 Native하게 실행됩니다. 이 파라미터의 값이 변경되어도 이미 컴파일된 PL/SQL 라이브러리 단위에는 아무런 영향이 없습니다. 이 파라미터의 값은 각 라이브러리 단위에 영구적으로 저장됩니다. PL/SQL 라이브러리 단위가 Native하게 컴파일되는 경우 해당 라이브러리 단위의 모든 후속 자동 재컴파일에서도 Native Compile이 사용됩니다. Oracle Database 11g에서는 Native Compile을 통합 방식으로 보다 쉽게 수행할 수 있으며 설정해야 하는 초기화 파라미터의 수도 더 적습니다.

드문 경우이기는 하지만 옵티마이저의 오버헤드로 인해 매우 큰 응용 프로그램의 컴파일 타임이 너무 오래 걸리는 경우에는 초기화 파라미터 PLSQL_OPTIMIZE_LEVEL의 값을 기본값인 2 대신 1로 설정하여 최적화 레벨을 낮출 수 있습니다. 또한 그러한 경우는 거의 없지만 예외 동작이 변경될 수도 있습니다. 즉, 예외가 전혀 발생하지 않거나 예상보다 일찍 발생할 수 있습니다. PLSQL_OPTIMIZE_LEVEL을 0으로 설정하여 코드가 재배열되지 않도록 할 수 있습니다.

PL/SQL 컴파일을 위해 초기화 파라미터 사용(계속)

PLSQL_OPTIMIZE_LEVEL 파라미터

이 파라미터는 PL/SQL 라이브러리 단위를 컴파일하는 데 사용할 최적화 레벨을 지정합니다. 이 파라미터의 설정 값이 높을수록 컴파일러가 PL/SQL 라이브러리 단위를 최적화하기 위해 보다 많은 작업을 수행합니다. 사용 가능한 값은 다음과 같습니다(0, 1 및 2는 Oracle 10g 버전 2부터 사용 가능).

0: 평가 순서를 유지하여 Oracle9i 이전 버전의 부작용, 예외 및 패키지 초기화 패턴을 그대로 유지합니다. 또한 BINARY_INTEGER 및 PLS_INTEGER의 새로운 의미 ID를 제거하며 정수 표현식 평가에 대해 이전 규칙을 복원합니다. 코드는 Oracle9i에서 실행할 때보다 빠르게 실행되기는 하지만 레벨을 0으로 설정하면 Oracle Database 10g부터 제공되는 PL/SQL의 성능 이점을 활용할 수 없게 됩니다.

1: 불필요한 계산 및 예외 제거 등 PL/SQL 프로그램에 대해 광범위한 최적화 기능을 적용하지만 일반적으로 원본 소스 순서를 벗어나 소스 코드를 이동하지는 않습니다.

2: 소스 코드를 원래 위치에서 상대적으로 멀리 이동하는 변경 사항을 포함하여 레벨 1 이상의 최신 최적화 기술을 광범위하게 적용합니다.

3: Oracle Database 11g에서 새롭게 제공되는 레벨 값으로, 구체적으로 요청하지 않은 기술을 자동으로 포함하여 레벨 2 이상의 최적화 기술을 광범위하게 적용합니다. 이를 통해 호출할 프로시저 본문의 복사본으로 프로시저 호출을 바꾸는 최적화 프로세스인 프로시저 인라이닝을 수행할 수 있습니다. 복사된 프로시저는 거의 모든 경우에 원래 호출보다 빠른 속도로 실행됩니다. 서브 프로그램 인라이닝을 허용하려면 PLSQL_OPTIMIZE_LEVEL 초기화 파라미터의 기본값(2)을 그대로 사용하거나 값을 3으로 설정합니다. PLSQL_OPTIMIZE_LEVEL = 2인 경우에는 인라이닝 할 각 서브 프로그램을 지정해야 합니다. PLSQL_OPTIMIZE_LEVEL = 3인 경우 PL/SQL 컴파일러는 유저가 지정하는 범위를 벗어나는 위치에서도 서브 프로그램 인라이닝 가능성을 검색합니다.

참고: 인라이닝에 대한 자세한 내용은 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* 설명서 및 *Oracle Database 11g Advanced PL/SQL* 강사 주도형 과정을 참조하십시오.

일반적으로 이 파라미터를 2로 설정하면 실행 성능이 향상됩니다. 그러나 컴파일러가 특정 소스 모듈에서 느리게 실행되는 경우나 고속 개발 등과 같이 최적화를 수행할 수 없는 경우 이 파라미터를 1로 설정하면 컴파일 타임 리소스를 보다 적게 사용하므로 파라미터를 2로 설정할 때와 거의 비슷한 컴파일 성능을 유지할 수 있습니다. 이 파라미터의 값은 라이브러리 단위에 영구적으로 저장됩니다.

참고

Oracle Database 10g에서는 다음과 같은 파라미터가 폐기되었으며 새로운 PLSQL_CODE_TYPE 파라미터가 사용됩니다.

- PLSQL_NATIVE_C_COMPILER
- PLSQL_NATIVE_MAKE_FILE_NAME
- PLSQL_NATIVE_C_COMPILER
- PLSQL_NATIVE_MAKE.Utility
- PLSQL_NATIVE_LINKER

Oracle Database 11g에서 PLSQL_DEBUG 파라미터 지원이 중단되었습니다. PLSQL_DEBUG 파라미터는 더 이상 PL/SQL 컴파일러의 디버깅 정보 생성 동작을 제어하지 않습니다. 디버깅 정보는 항상 생성되며 특별한 파라미터를 사용할 필요가 없습니다.

컴파일러 설정

컴파일러 옵션	설명
PLSQL_CODE_TYPE	PL/SQL 라이브러리 단위에 대해 컴파일 모드를 지정합니다.
PLSQL_OPTIMIZE_LEVEL	PL/SQL 라이브러리 단위를 컴파일하는 데 사용할 최적화 레벨을 지정합니다.
PLSQL_WARNINGS	PL/SQL 컴파일러에서 생성되는 경고 메시지의 보고를 활성화하거나 비활성화합니다.
PLSQL_CCFLAGS	각 PL/SQL 라이브러리 단위의 조건부 컴파일을 독립적으로 제어합니다.

일반적으로 컴파일 속도를 최대로 높이려면 다음과 같은 설정을 사용합니다.

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

Copyright © 2009, Oracle. All rights reserved.

컴파일러 설정

새로운 컴파일러는 PL/SQL 코드의 성능을 향상시키며 코드 실행 속도를 Oracle8i 데이터베이스에 비해 2배 그리고 Oracle9i 데이터베이스 버전 2에 비해 1.5배 ~ 1.75배 높입니다.

성능을 높이려면 컴파일러 설정을 다음과 같이 지정해야 합니다.

```
PLSQL_CODE_TYPE = NATIVE
PLSQL_OPTIMIZE_LEVEL = 2
```

PL/SQL 초기화 파라미터 표시

**USER | ALL | DBA_PLSQL_OBJECT_SETTINGS 데이터
딕셔너리 뷰를 사용하여 PL/SQL 객체의 설정 표시**

DESCRIBE USER_PLSQL_OBJECT_SETTINGS

Name	Null	Type
NAME	NOT NULL	VARCHAR2(30)
TYPE		VARCHAR2(12)
PLSQL_OPTIMIZE_LEVEL		NUMBER
PLSQL_CODE_TYPE		VARCHAR2(4000)
PLSQL_DEBUG		VARCHAR2(4000)
PLSQL_WARNINGS		VARCHAR2(4000)
NLS_LENGTH_SEMANTICS		VARCHAR2(4000)
PLSQL_CCFLAGS		VARCHAR2(4000)
PLSCOPE_SETTINGS		VARCHAR2(4000)
9 rows selected		

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 초기화 파라미터 표시

USER_PLSQL_OBJECTS_SETTINGS 데이터 딕셔너리 뷰에는 다음과 같은 열이 있습니다.

Owner: 객체의 소유자입니다. 이 열은 USER_PLSQL_OBJECTS_SETTINGS 뷰에 표시되지 않습니다.

Name: 객체의 이름입니다.

Type: PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, TYPE, TYPE BODY 중에서 선택할 수 있습니다.

PLSQL_OPTIMIZE_LEVEL: 객체를 컴파일하는 데 사용한 최적화 레벨입니다.

PLSQL_CODE_TYPE: 객체의 컴파일 모드입니다.

PLSQL_DEBUG: 객체를 디버깅용으로 컴파일했는지 여부를 지정합니다.

PLSQL_WARNINGS: 객체를 컴파일하는 데 사용하는 컴파일러 경고 설정입니다.

NLS_LENGTH_SEMANTICS: 객체를 컴파일하는 데 사용하는 NLS 길이 의미입니다.

PLSQL_CCFLAGS: 객체를 컴파일하는 데 사용하는 조건부 컴파일 플래그입니다.

PLSCOPE_SETTINGS: Oracle Database 11g의 새로운 기능으로, 컴파일 타임 컬렉션, 상호 참조 및 PL/SQL 소스 코드 식별자 데이터의 저장 영역을 제어합니다.

PL/SQL 초기화 파라미터 표시 및 설정

```
SELECT name, type, plsql_code_type AS CODE_TYPE,  
       plsql_optimize_level AS OPT_LVL  
  FROM user_plsql_object_settings;
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	2
2 ADD_JOB_HIST	PROCEDURE	INTERPRETED	2
3 ADD_JOB_HISTORY	PROCEDURE	INTERPRETED	2
4 BANK_TRANS	PROCEDURE	INTERPRETED	2
5 CHECK_SALARY	PROCEDURE	INTERPRETED	2

- **ALTER SYSTEM 또는 ALTER SESSION 문을 사용하여 컴파일러 초기화 파라미터의 값을 설정합니다.**
- **CREATE OR REPLACE 문을 실행할 때 파라미터의 값에 액세스합니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고

- ALTER SYSTEM 또는 ALTER SESSION 문에 대한 자세한 내용은 *Oracle Database SQL Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.
- Oracle Database 10g에서 DBA_STORED_SETTINGS 데이터 딕셔너리 뷰 계열의 지원이 중단되었으며 DBA_PLSQL_OBJECT_SETTINGS 데이터 딕셔너리 뷰 계열로 바뀌었습니다.

PL/SQL 초기화 파라미터 변경: 예제

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL = 1;
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE';
```

```
ALTER SESSION SET succeeded.
ALTER SESSION SET succeeded.
```

```
-- code displayed in the notes page
CREATE OR REPLACE PROCEDURE add_job_history
. . .
```

```
@code_10_10_s.sql
```

NAME	TYPE	CODE_TYPE	OPT_LVL
1 ADD_COL	PROCEDURE	INTERPRETED	2
2 ADD_EMPLOYEE	PROCEDURE	INTERPRETED	2
3 ADD_JOB_HIST	PROCEDURE	INTERPRETED	2
4 ADD_JOB_HISTORY	PROCEDURE	NATIVE	1

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 초기화 파라미터 변경: 예제

컴파일된 PL/SQL 객체를 Interpreted 코드 유형에서 Native 코드 유형으로 변경하려면 먼저 PLSQL_CODE_TYPE 파라미터를 NATIVE로 설정한 후 필요한 경우 다른 파라미터를 설정하고 프로그램을 재컴파일합니다. 모든 PL/SQL 코드에 Native Compile을 적용하려면 각 코드를 재컴파일해야 합니다. rdbms/admin 디렉토리의 스크립트는 전체 Native Compile(dbmsupgnv.sql) 또는 전체 Interpreted 컴파일(dbmsupgin.sql)로 변환할 수 있도록 제공됩니다. add_job_history 프로시저는 다음과 같이 생성됩니다.

```
CREATE OR REPLACE PROCEDURE add_job_history
( p_emp_id          job_history.employee_id%type
, p_start_date      job_history.start_date%type
, p_end_date        job_history.end_date%type
, p_job_id          job_history.job_id%type
, p_department_id   job_history.department_id%type )
IS
BEGIN
    INSERT INTO job_history (employee_id, start_date,
                           end_date, job_id, department_id)
    VALUES(p_emp_id, p_start_date, p_end_date,
           p_job_id, p_department_id);
END add_job_history;
/
```

단원 내용

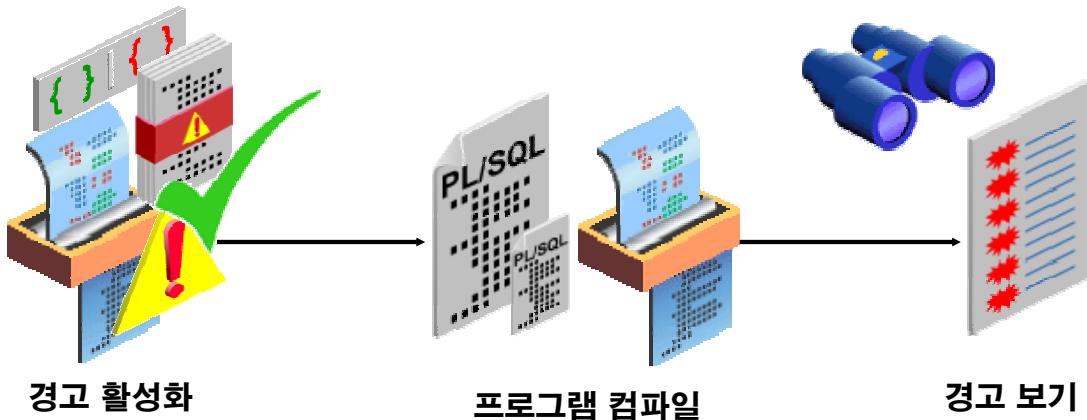
- PLSQL_CODE_TYPE 및 PLSQL_OPTIMIZE_LEVEL
PL/SQL 컴파일 초기화 파라미터 사용
- PL/SQL 컴파일 타임 경고 사용:
 - PLSQL_WARNING 초기화 파라미터 사용
 - DBMS_WARNING 패키지 서브 프로그램 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

서브 프로그램용 PL/SQL 컴파일 타임 경고 개요

Oracle 10g부터는 서브 프로그램에 대한 경고를 생성하도록 PL/SQL 컴파일러의 기능이 향상되었습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

서브 프로그램용 PL/SQL 컴파일 타임 경고 개요

프로그램을 보다 강력하게 만들고 런타임에 문제가 발생하지 않도록 하려면 특정 경고 조건 확인 기능을 설정하면 됩니다. 이러한 조건은 오류를 발생시킬 정도로 심각한 것은 아니지만 서브 프로그램을 컴파일할 수 없게 되는 원인이 되며, 정의되지 않은 결과를 만들어내거나 성능 문제를 발생시킬 수 있는 서브 프로그램의 특정 요소를 나타낼 수도 있습니다.

Oracle Database 10g 이전 버전에서는 PL/SQL 프로그램을 컴파일하면 다음과 같은 두 가지 결과가 발생했습니다.

- 성공(올바른 컴파일 단위 생성)
- Failure (프로그램에 구문 또는 의미 오류가 있음을 나타내는 컴파일 오류 발생)

그러나 프로그램이 성공적으로 컴파일된 경우에도 권장되는 최적의 사용법을 위반했거나 효율성이 떨어지는 방식으로 코딩되었을 수 있습니다. Oracle Database 10g에는 이러한 경우 PL/SQL 컴파일러가 경고 메시지를 제공할 수 있도록 하는 간편한 새 기능이 도입되었습니다. 개발자는 컴파일러 경고를 통해 일반적인 코딩 오류를 방지하여 생산성을 높일 수 있습니다.

서브 프로그램용 PL/SQL 컴파일 타임 경고 개요(계속)

PL/SQL에서는 NOCOPY 컴파일러 힌트를 통해 IN OUT 및 OUT 파라미터를 값이나 참조로 전달할 수 있습니다. 파라미터를 값으로 전달하는 경우 데이터의 복사본이 여러 개 만들어지므로 결과적으로는 효율성이 떨어집니다. Oracle Database 11g에서는 컴파일러가 NOCOPY 힌트를 자동으로 감지하여 사용하도록 권장합니다. 여기서 파라미터 유형은 대형 객체, 레코드 또는 컬렉션 유형입니다.

PL/SQL 컴파일러 경고 기능을 사용하는 경우 PL/SQL 프로그램을 컴파일하면 다음과 같은 결과가 추가로 발생할 수 있습니다.

- 성공(컴파일 경고 생성)
- Failure(컴파일 오류 및 컴파일 경고 생성)

컴파일러는 컴파일이 성공하는 경우에도 경고 메시지를 발행할 수 있습니다. 컴파일 오류는 내장 프로시저를 사용하려면 수정해야 하는 반면 경고는 정보용으로만 사용됩니다.

경고 메시지 예제

SP2-0804: Procedure created with compilation warnings

PLW-07203: Parameter 'IO_TBL' may benefit from use of the NOCOPY compiler hint

컴파일러 경고의 이점

- 보다 강력한 프로그램을 만들고 런타임에 발생하는 문제 방지
- 잠재적인 성능 문제 식별
- 정의되지 않은 결과를 생성하는 요인 식별



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고의 이점

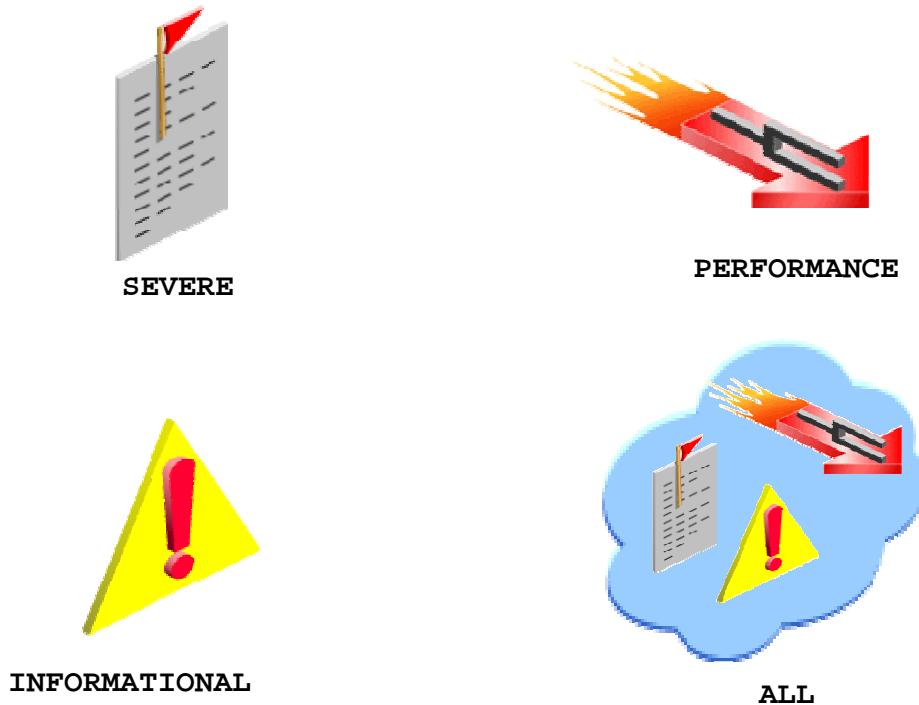
컴파일러 경고를 사용하면 다음과 같은 이점이 있습니다.

- 보다 강력한 프로그램을 만들고 런타임에 발생하는 문제 방지
- 잠재적인 성능 문제 식별
- 정의되지 않은 결과를 생성하는 요인 식별

참고

- 특정 경고 조건이 오류를 일으킬 정도로 심각하지 않은데 서브 프로그램이 컴파일되지 않는 경우 이러한 경고 조건을 검사할 수 있습니다.
- 경고 메시지는 PL/SQL 서브 프로그램 컴파일 중에 발생될 수 있으며 익명 블록은 경고를 생성하지 않습니다.
- 모든 PL/SQL 경고 메시지에는 접두어 PLW가 사용됩니다.

PL/SQL 컴파일 타임 경고 메시지의 범주



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 컴파일 타임 경고 메시지의 범주

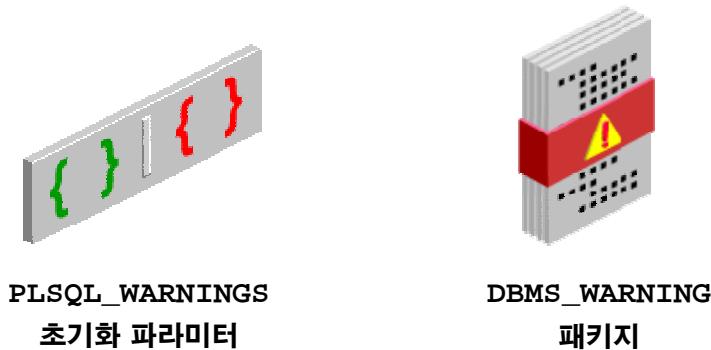
PL/SQL 경고 메시지는 여러 범주로 구분되므로 컴파일 중에 비슷한 경고로 구성되는 그룹을 표시하지 않거나 표시할 수 있습니다. 다음과 같은 범주가 있습니다.

- **SEVERE:** 파라미터로 인한 Alias 지정 문제 등 예기치 않은 동작이나 잘못된 결과를 만들어내는 원인이 될 수 있는 조건에 대한 메시지입니다.
- **PERFORMANCE:** INSERT 문에서 VARCHAR2 값을 NUMBER 열로 전달하는 등 성능 문제를 일으킬 수 있는 조건에 대한 메시지입니다.
- **INFORMATIONAL:** 실행할 수 없는 연결 불가 코드 등 성능에 영향을 주거나 수정을 필요로 하지는 않지만 코드를 보다 쉽게 유지 관리하기 위해 변경할 수 있는 조건에 대한 메시지입니다.

경고 메시지 레벨 설정

다음 방법 중 하나를 사용하여 경고 레벨을 설정할 수 있습니다.

- **선언적으로:**
 - PLSQL_WARNINGS 초기화 파라미터 사용
- **프로그래밍 방식으로:**
 - DBMS_WARNING 패키지 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

경고 메시지 레벨 설정

다음 방법 중 하나를 사용하여 컴파일러 경고 메시지 레벨을 설정할 수 있습니다.

PLSQL_WARNINGS 초기화 파라미터 사용

PLSQL_WARNINGS 설정은 PL/SQL 컴파일러에 의한 경고 메시지 보고를 활성화하거나 비활성화하며, 오류로 표시할 경고 메시지를 지정합니다. PLSQL_WARNINGS 파라미터에 대한 설정은 컴파일된 각 서브 프로그램과 함께 저장됩니다. PLSQL_WARNINGS 초기화 파라미터를 사용하여 다음을 수행할 수 있습니다.

- 모든 경고, 선택한 범주의 경고 또는 특정 경고 메시지의 보고를 활성화하거나 비활성화합니다.
- 모든 경고, 선택한 경고 범주 또는 특정 경고 메시지를 오류로 취급합니다.
- 앞의 항목을 적절하게 조합하여 적용합니다.

ALL 키워드를 사용하면 모든 경고 메시지(SEVERE, PERFORMANCE 및 INFORMATIONAL)를 간편하게 참조할 수 있습니다.

DBMS_WARNING 패키지 사용

DBMS_WARNING 패키지는 PL/SQL 경고 메시지의 동작을 조작하는 방법을 제공하는데, 특히 PLSQL_WARNINGS 초기화 파라미터의 설정을 읽고 변경하여 숨기거나 표시하거나 오류로 처리할 경고의 종류를 제어합니다. 이 패키지는 현재 시스템이나 세션 설정을 query, 수정 및 삭제할 수 있는 인터페이스를 제공합니다. 이 패키지에 대해서는 이 단원의 뒷부분에서 다룹니다.

컴파일러 경고 레벨 설정: PLSQL_WARNINGS 사용

```
ALTER [SESSION|SYSTEM]
PLSQL_WARNINGS = 'value_clause1'[ , 'value_clause2' ]...
```

```
value_clause = Qualifier Value : Modifier Value
```

```
Qualifier Value = { ENABLE | DISABLE | ERROR }
```

```
Modifier Value =
{ ALL | SEVERE | INFORMATIONAL | PERFORMANCE |
{ integer | (integer [, integer ] ...) } }
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 설정 수정

파라미터 값은 따옴표로 묶은 수식자와 수정자 키워드의 쉼표로 구분된 리스트로 구성됩니다. 여기서 키워드는 콜론으로 구분됩니다. 수식자 값은 ENABLE, DISABLE 및 ERROR입니다. 수정자 값 ALL은 모든 경고 메시지에 적용됩니다. SEVERE, INFORMATIONAL 및 PERFORMANCE는 해당 범주에 있는 메시지 및 특정 경고 메시지의 정수 리스트에 적용됩니다. ENABLE, DISABLE 및 ERROR에는 다음 값을 사용할 수 있습니다.

- ALL
- SEVERE
- INFORMATIONAL
- PERFORMANCE
- numeric_value

numeric_value의 값은 다음 범위에 있습니다.

- SEVERE의 경우 5000-5999 사이
- INFORMATIONAL의 경우 6000-6249 사이
- PERFORMANCE의 경우 7000-7249 사이

컴파일러 경고 레벨 설정: PLSQL_WARNINGS 사용, 예제

```
ALTER SESSION
SET plsql_warnings = 'enable:severe',
    'enable:performance',
    'disable:informational';
```

ALTER SESSION succeeded.

```
ALTER SESSION
SET plsql_warnings = 'enable:severe';
```

ALTER SESSION succeeded.

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:SEVERE',
    'DISABLE:PERFORMANCE' , 'ERROR:05003';
```

ALTER SESSION SET succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 레벨 설정: PLSQL_WARNINGS 사용, 예제

ALTER SESSION 또는 ALTER SYSTEM 명령을 사용하여 PLSQL_WARNINGS 초기화 파라미터를 변경할 수 있습니다. 슬라이드의 그림에는 다양한 컴파일러 경고 활성화 및 비활성화 예제가 나와 있습니다.

예제 1

이 예제에서는 SEVERE 및 PERFORMANCE 경고를 활성화하고 INFORMATIONAL 경고를 비활성화합니다.

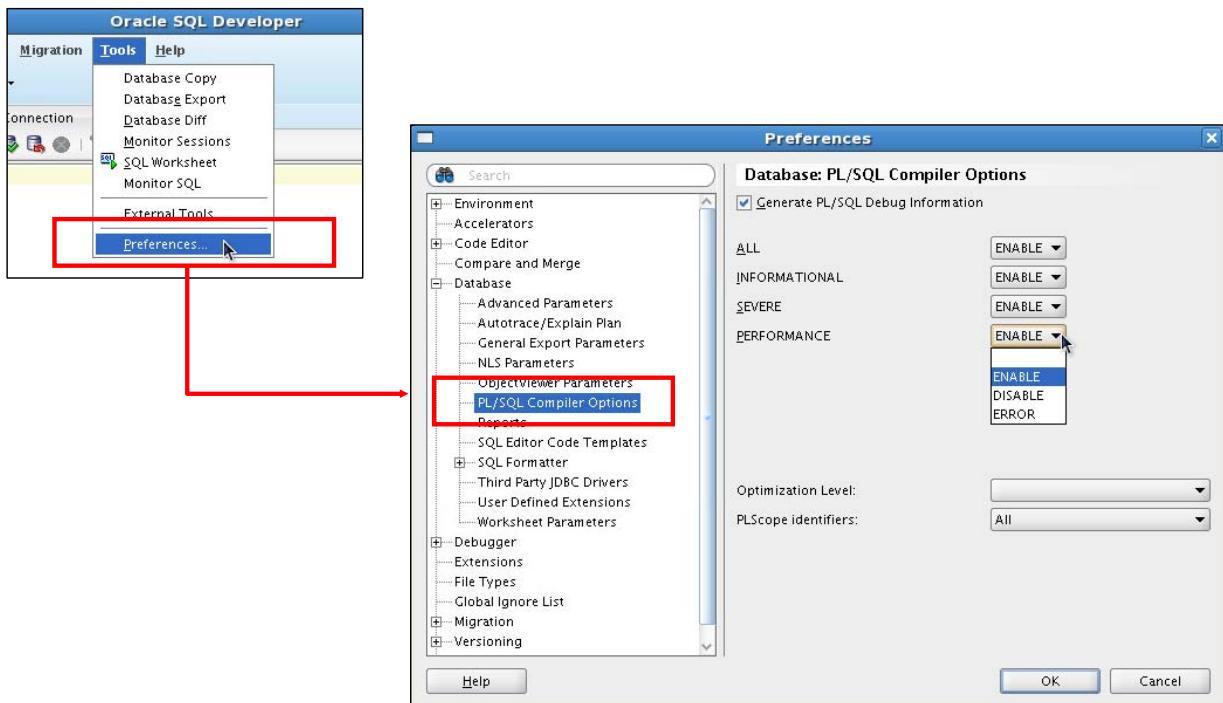
예제 2

이 예제에서는 SEVERE 경고만 활성화합니다.

예제 3

특정 메시지를 경고가 아닌 오류로 취급할 수도 있습니다. 이 예제에서 경고 메시지 PLW-05003이 코드의 심각한 문제를 나타내는 경우 'ERROR:05003'을 PLSQL_WARNINGS 설정에 포함하면 해당 조건이 경고 메시지가 아닌 오류 메시지(PLS_05003)를 트리거하게 됩니다. 오류 메시지가 발생하면 컴파일이 실패합니다. 이 예제에서는 PERFORMANCE 경고도 비활성화합니다.

컴파일러 경고 레벨 설정: SQL Developer에서 PLSQL_WARNINGS 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 레벨 설정: SQL Developer에서 PLSQL_WARNINGS 사용

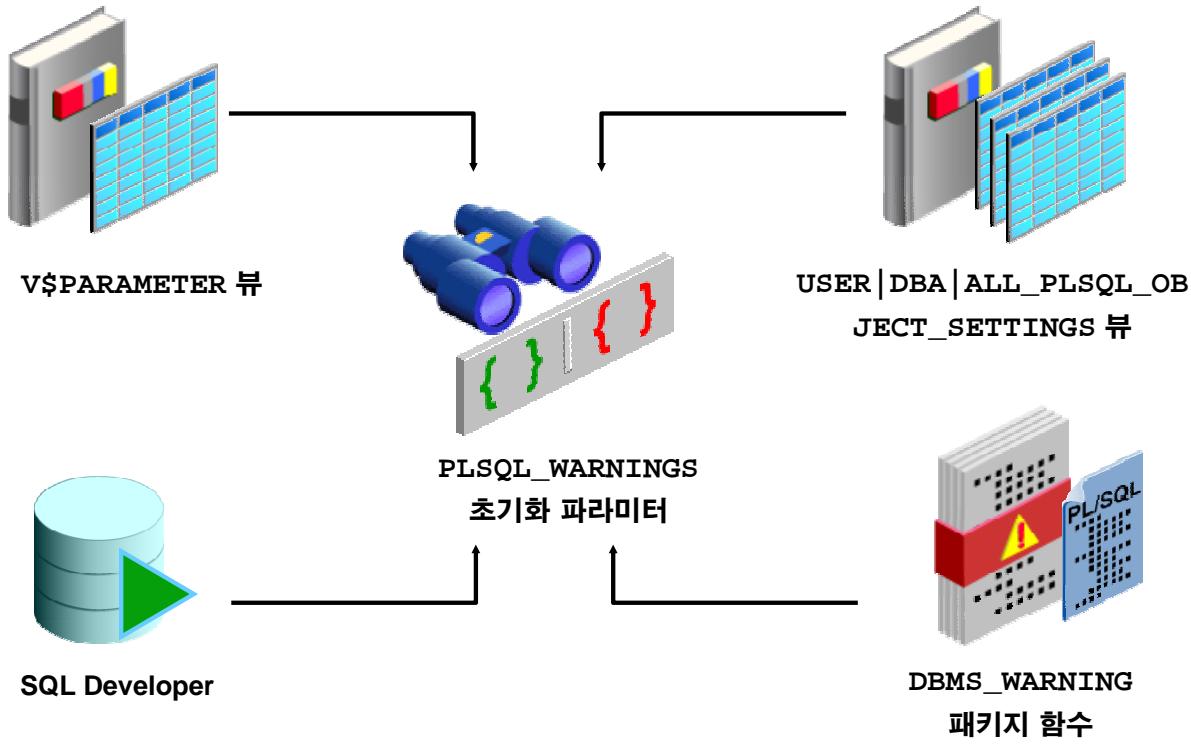
PL/SQL Compiler 창에서는 PL/SQL 서브 프로그램 컴파일에 대한 옵션을 지정합니다. **Generate PL/SQL Debug Information** 체크 박스를 선택하면 PL/SQL 디버그 정보가 컴파일된 코드에 포함되고, 이 옵션을 선택하지 않으면 이 디버그 정보가 포함되지 않습니다. 개별 코드 행에서 컴파일을 정지하고 디버거가 변수에 액세스하도록 하는 기능은 생성된 디버그 정보를 포함하여 컴파일된 코드에서만 사용할 수 있습니다.

SQL Developer에서 PL/SQL 컴파일 타임 경고 메시지 범주 설정 및 보기

INFORMATIONAL, SEVERE 및 PERFORMANCE 관련 메시지 표시를 제어할 수 있습니다. **ALL** 유형을 선택하면 다른 메시지 유형에 대해 개별적으로 지정한 옵션이 재정의됩니다. 각 메시지 유형에 대해 다음을 지정할 수 있습니다.

- **항목 없음(공란):** ALL에 대해 지정한 모든 값을 사용합니다. 지정한 값이 없으면 오라클 기본값이 사용됩니다.
- **Enable:** 해당 범주의 모든 메시지 표시를 활성화합니다.
- **Disable:** 해당 범주의 모든 메시지 표시를 비활성화합니다.
- **Error:** 해당 범주의 오류 메시지에 대해서만 표시를 활성화합니다.

PLSQL_WARNINGS의 현재 설정 보기



Copyright © 2009, Oracle. All rights reserved.

ORACLE

PLSQL_WARNINGS 파라미터의 현재 값 보기

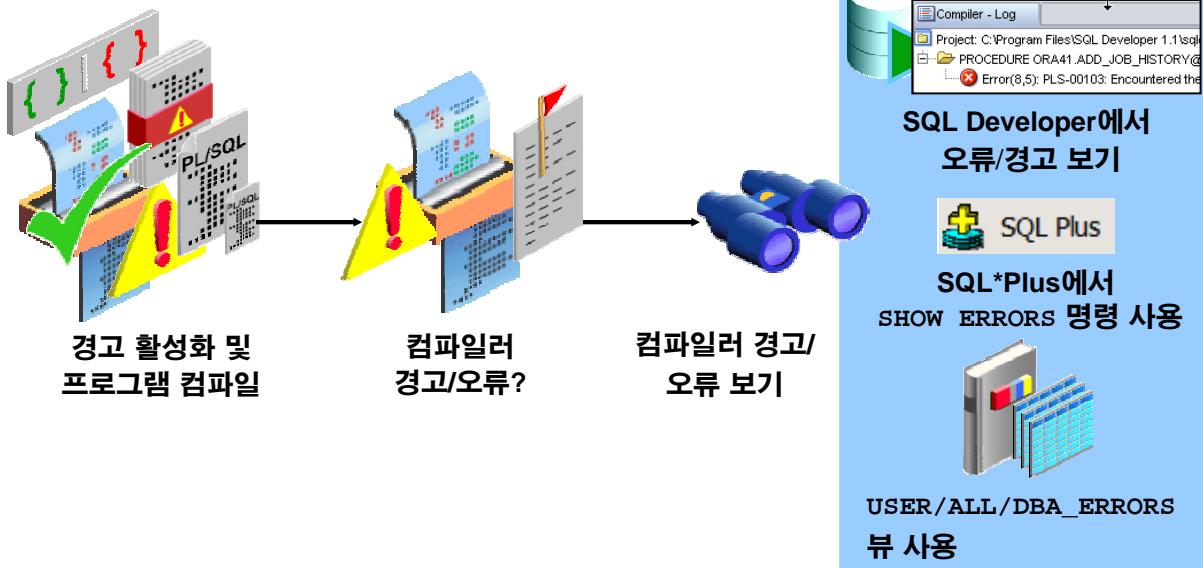
V\$PARAMETER 뷰에서 SELECT 문을 실행하여 PLSQL_WARNINGS 파라미터에 대한 현재 설정을 확인할 수 있습니다. 예를 들면 다음과 같습니다.

```
ALTER SESSION SET plsql_warnings = 'enable:severe',
               'enable:performance', 'enable:informational';
Session altered.
SELECT value FROM v$parameter WHERE name='plsql_warnings';
VALUE
-----
ENABLE:ALL
```

또한 DBMS_WARNING.GET_WARNING_SETTING_STRING 패키지와 프로시저를 사용하여 PLSQL_WARNINGS 파라미터의 현재 설정을 검색할 수도 있습니다.

```
DECLARE s VARCHAR2(1000);
BEGIN
    s := dbms_warning.get_warning_setting_string();
    dbms_output.put_line (s);
END;
/
anonymous block completed
ENABLE:ALL
```

컴파일러 경고 보기: SQL Developer, SQL*Plus 또는 데이터 딕셔너리 뷰 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 보기

SQL*Plus를 사용하면 PL/SQL 블록을 컴파일한 결과로 발생하는 모든 경고를 볼 수 있습니다. SQL*Plus에서는 컴파일 경고가 발생했음을 표시합니다. "*SP2-08xx: <object> created with compilation warnings.*" 메시지는 PERFORMANCE, INFORMATIONAL 또는 SEVERE 수정자를 사용하여 컴파일한 객체에 대해 표시됩니다. 세 수정자에 대해 모두 동일한 메시지가 표시됩니다. 프로그램을 컴파일하기 전에 컴파일러 경고를 활성화해야 합니다. 다음 방법 중 하나를 사용하여 컴파일러 경고 메시지를 표시할 수 있습니다.

SQL*Plus SHOW ERRORS 명령 사용

이 명령은 새 컴파일러 경고 및 정보 메시지를 포함하여 모든 컴파일러 오류를 표시합니다. 이 명령은 CREATE [PROCEDURE | FUNCTION | PACKAGE] 명령을 사용한 후에 즉시 호출됩니다. SHOW ERRORS 명령은 경고 및 컴파일러 오류를 표시합니다. SHOW ERRORS를 호출하면 새 컴파일러 경고 및 정보 메시지가 컴파일러 오류와 "함께 표시"됩니다.

데이터 딕셔너리 뷰 사용

USER_|ALL_|DBA_ERRORS 데이터 딕셔너리 뷰 중에서 PL/SQL 컴파일러 경고를 표시할 뷰를 선택할 수 있습니다. 이 뷰의 ATTRIBUTES 열에는 WARNING이라는 새로운 속성이 있으며, 경고 메시지는 TEXT 열에 표시됩니다.

SQL*Plus 경고 메시지: 예제

```
CREATE OR REPLACE PROCEDURE bad_proc(p_out ...) IS
BEGIN
  . . .;
END;
/
```

SP2-0804: Procedure created with compilation warnings.

```
SHOW ERRORS;
Errors for PROCEDURE BAD_PROC:

LINE/COL      ERROR
-----  
6/24          PLW-07203: parameter 'p_out' may benefit
                  from use of the NOCOPY compiler hint
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 경고 메시지: 예제

SQL*Plus에서 SHOW ERRORS 명령을 사용하여 내장 프로시저의 컴파일 오류를 표시합니다. 이 옵션을 인수 없이 지정하면 SQL*Plus에서는 가장 최근에 생성되었거나 변경된 내장 프로시저에 대한 컴파일 오류를 표시합니다. 내장 프로시저를 생성하거나 변경한 후 SQL*Plus에 컴파일 경고 메시지가 표시되면 SHOW ERRORS 명령을 사용하여 자세한 정보를 확인할 수 있습니다.

이제는 PL/SQL 경고가 지원되기 때문에 피드백 메시지의 범위가 확장되어 다음과 같은 세번째 메시지도 포함됩니다.

SP2-08xx: <object> created with compilation warnings.

그러므로 컴파일 경고와 컴파일 오류를 구분할 수 있습니다. 내장 프로시저를 사용하려면 오류는 수정해야 하는 반면 경고는 정보용으로만 제공됩니다.

경고 메시지에 포함된 SP2 접두어를 사용하면 *SQL*Plus User's Guide and Reference*에서 해당하는 메시지 번호를 조회하여 특정 메시지의 원인과 필요한 조치를 확인할 수 있습니다.

참고: SHOW SQL*Plus 명령은 이 클래스에서 사용되는 SQL Developer 1.5.4 버전에서 지원되지 않습니다. USER_ | ALL_ | DBA_ERRORS 데이터 덕셔너리 뷰를 사용하면 컴파일러 오류 및 경고를 볼 수 있습니다.

PLSQL_WARNINGS 사용 지침

- PLSQL_WARNINGS 파라미터에 대한 설정은 컴파일된 각 서브 프로그램과 함께 저장됩니다.
- 다음 명령문 중 하나를 사용하여 서브 프로그램을 재컴파일하면 해당 세션의 현재 설정이 사용됩니다.
 - CREATE OR REPLACE
 - ALTER ... COMPILE
- ALTER ...COMPILE 문을 REUSE SETTINGS 절과 함께 사용하여 서브 프로그램을 재컴파일하는 경우 프로그램과 함께 저장된 원래 설정이 사용됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PLSQL_WARNINGS 사용 지침

앞에서 설명한 바와 같이 PLSQL_WARNINGS 파라미터는 세션 레벨이나 시스템 레벨에서 설정할 수 있습니다.

PLSQL_WARNINGS 파라미터에 대한 설정은 컴파일된 각 서브 프로그램과 함께 저장됩니다. CREATE OR REPLACE 문을 사용하여 서브 프로그램을 재컴파일하는 경우 해당 세션에 대한 현재 설정이 사용됩니다. ALTER...COMPILE 문을 사용하여 서브 프로그램을 재컴파일하는 경우 명령문에 REUSE SETTINGS 절(서브 프로그램과 함께 저장된 원래 설정을 사용함)이 지정되어 있지 않으면 현재 세션 설정이 사용됩니다.

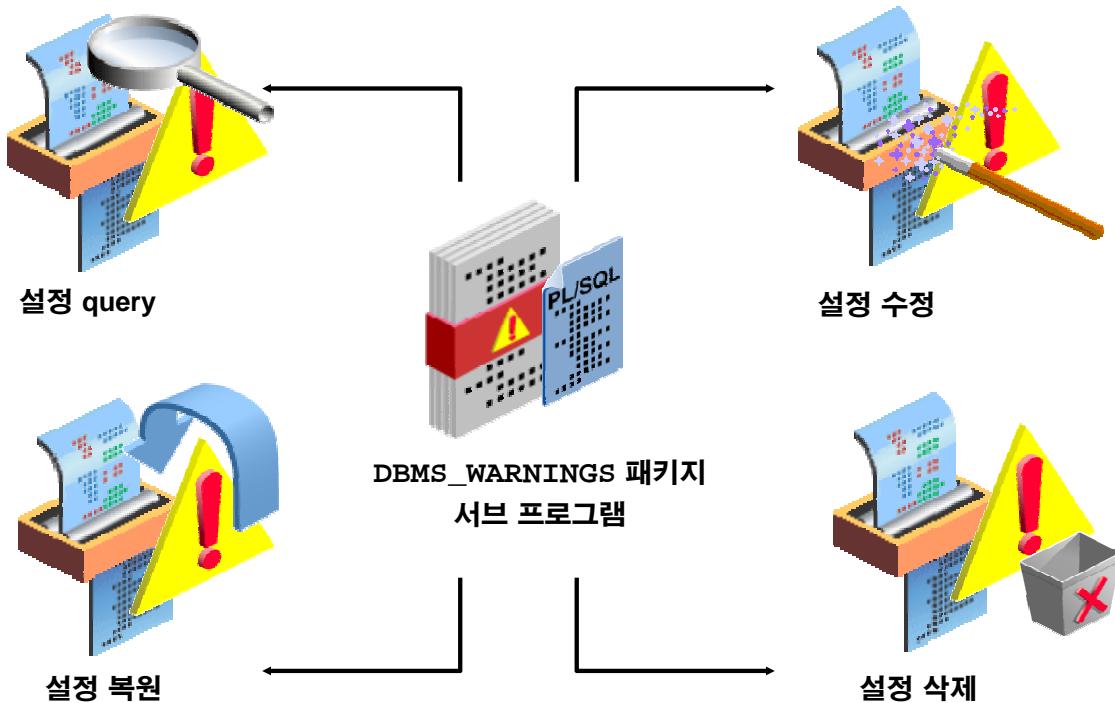
단원 내용

- PLSQL_CODE_TYPE 및 PLSQL_OPTIMIZE_LEVEL PL/SQL 컴파일 초기화 파라미터 사용
- PL/SQL 컴파일 타임 경고 사용:
 - PLSQL_WARNING 초기화 파라미터 사용
 - DBMS_WARNING 패키지 서브 프로그램 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 레벨 설정: DBMS_WARNING 패키지 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일러 경고 레벨 설정: DBMS_WARNING 패키지 사용

DBMS_WARNING 패키지를 사용하여 현재 시스템이나 세션의 PL/SQL 경고 설정 동작을 프로그래밍 방식으로 조작합니다. DBMS_WARNING 패키지는 PL/SQL 경고 메시지의 동작을 조작하는 방법을 제공하는데, 특히 PLSQL_WARNINGS 초기화 파라미터의 설정을 읽고 변경하여 숨기거나 표시하거나 오류로 처리할 경고의 종류를 제어합니다. 이 패키지는 현재 시스템이나 세션 설정을 query, 수정 및 삭제할 수 있는 인터페이스를 제공합니다.

DBMS_WARNING 패키지는 PL/SQL 서브 프로그램을 컴파일하는 개발 환경을 작성하는데 유용합니다. 패키지 인터페이스 루틴을 사용하면 자신의 필요에 맞게 PL/SQL 경고 메시지를 프로그래밍 방식으로 제어 할 수 있습니다.

컴파일러 경고 레벨 설정: DBMS_WARNING 패키지 사용(계속)

서브 프로그램용 PL/SQL 컴파일 타임 경고 개요: 예제

PL/SQL 코드를 컴파일할 코드를 작성한다고 가정합니다. 아시다시피 NOCOPY 힌트를 지정하지 않고 컬렉션 변수를 OUT 또는 IN OUT 파라미터로 전달할 경우 컴파일러는 PERFORMANCE 경고를 표시하게 됩니다. 컴파일 유ти리티를 호출하는 일반적인 환경에는 적절한 경고 레벨 설정이 있을 수도 있고 없을 수도 있습니다. 어떤 경우든지 업무 규칙에 따라 호출 환경 설정은 보존되어야 하며, 컴파일 프로세스는 경고를 표시하지 않아야 합니다. DBMS_WARNING 패키지에서 서브 프로그램을 호출하면 현재 경고 설정을 감지하고, 해당 설정을 업무 요구 사항에 맞게 변경하며, 처리가 완료되었을 때 원래 설정을 복원할 수 있습니다.

ALTER SESSION 또는 ALTER SYSTEM 명령을 사용하여 PLSQL_WARNINGS 파라미터를 설정하는 경우 지정되는 새 값이 이전 값을 완전하게 바꿉니다. 새로운 패키지인 DBMS_WARNING은 Oracle Database 10g에서 사용 가능합니다. 여기에는 PLSQL_WARNINGS 파라미터의 설정을 query하고 충분 방식으로 변경하여 요구 사항에 보다 적합하게 조정할 수 있는 인터페이스가 포함되어 있습니다.

DBMS_WARNING 패키지를 사용하면 PLSQL_WARNINGS 파라미터를 충분 방식으로 변경할 수 있으므로 직접적인 관계가 없는 모든 경고의 값을 보존할 방식을 지정하지 않고도 원하는 경고를 설정할 수 있습니다. 예를 들어 DBA가 초기화 파라미터 파일의 전체 데이터베이스에 대해 SEVERE 경고 설정만을 활성화하지만 새 코드를 테스트하는 개발자가 특정 PERFORMANCE 및 INFORMATIONAL 메시지를 보려는 경우가 있습니다. 이 경우 개발자는 DBMS_WARNING 패키지를 사용하여 보려는 특정 경고를 충분 방식으로 추가할 수 있습니다. 그러면 개발자는 DBA의 설정을 바꾸지 않고도 원하는 메시지를 볼 수 있습니다.

DBMS_WARNING 패키지 서브 프로그램 사용

시나리오	사용할 서브 프로그램
경고 설정	ADD_WARNING_SETTING_CAT (프로시저) ADD_WARNING_SETTING_NUM (프로시저)
경고 query	GET_WARNING_SETTING_CAT (함수) GET_WARNING_SETTING_NUM (함수) GET_WARNING_SETTING_STRING (함수)
경고 바꾸기	SET_WARNING_SETTING_STRING (프로시저)
경고 범주 이름 가져오기	GET_CATEGORY (함수)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 서브 프로그램 사용

다음은 DBMS_WARNING 서브 프로그램 리스트입니다.

- ADD_WARNING_SETTING_CAT: 이전에 제공된 warning_category의 현재 세션 또는 시스템 경고 설정을 수정합니다.
- ADD_WARNING_SETTING_NUM: 이전에 제공된 warning_number의 현재 세션 또는 시스템 경고 설정을 수정합니다.
- GET_CATEGORY: 메시지 번호가 제공되면 범주 이름을 반환합니다.
- GET_WARNING_SETTING_CAT: 세션의 특정 경고 범주를 반환합니다.
- GET_WARNING_SETTING_NUM: 세션의 특정 경고 번호를 반환합니다.
- GET_WARNING_SETTING_STRING: 현재 세션의 경고 문자열 전체를 반환합니다.
- SET_WARNING_SETTING_STRING: 이전 설정을 새 값으로 바꿉니다.

참고: 위의 서브 프로그램에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)*를 참조하십시오.

DBMS_WARNING 프로시저: 구문, 파라미터 및 허용값

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT (
    warning_category      IN      VARCHAR2,
    warning_value         IN      VARCHAR2,
    scope                 IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_NUM (
    warning_number        IN      NUMBER,
    warning_value          IN      VARCHAR2,
    scope                 IN      VARCAHR2);
```

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING (
    warning_value          IN      VARCHAR2,
    scope                 IN      VARCHAR2);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 프로시저: 구문, 파라미터 및 허용값

warning_category는 범주의 이름입니다. 허용값은 다음과 같습니다.

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

warning_value는 범주의 값입니다. 허용값은 다음과 같습니다.

- ENABLE
- DISABLE
- ERROR

warning_number는 경고 메시지 번호입니다. 허용값은 모든 유효한 경고 번호입니다.

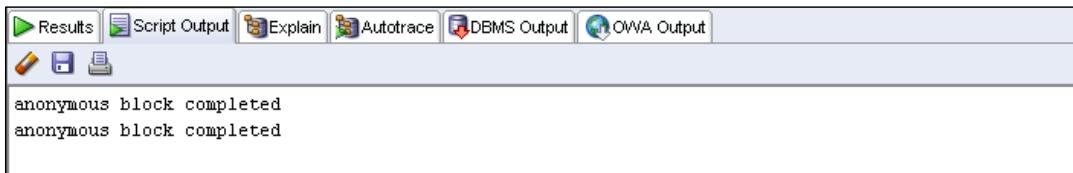
범위는 변경 사항이 세션 컨텍스트에서 수행되는지 아니면 시스템 컨텍스트에서 수행되는지를 지정합니다. 허용값은 SESSION 또는 SYSTEM입니다. SYSTEM을 사용하려면 ALTER SYSTEM 권한이 필요합니다.

DBMS_WARNING 프로시저: 예제

```
-- Establish the following warning setting string in the
-- current session:
-- ENABLE:INFORMATIONAL,
-- DISABLE:PERFORMANCE,
-- ENABLE:SEVERE

EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(
  'ENABLE:ALL', 'SESSION');

EXECUTE DBMS_WARNING.ADD_WARNING_SETTING_CAT(
  'PERFORMANCE', 'DISABLE', 'SESSION');
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 프로시저 사용: 예제

SET_WARNING_SETTING_STRING 프로시저를 사용하여 경고 설정을 하나 지정할 수 있습니다. 경고 설정이 여러 개일 경우 다음 단계를 수행해야 합니다.

1. SET_WARNING_SETTING_STRING을 호출하여 초기 경고 설정 문자열을 설정합니다.
2. ADD_WARNING_SETTING_CAT 또는 ADD_WARNING_SETTING_NUM을 반복 호출하여 초기 문자열에 설정을 추가합니다.

슬라이드의 예제에서는 현재 세션에서 다음과 같은 경고 설정 문자열을 지정합니다.

ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE

DBMS_WARNING 함수: 구문, 파라미터 및 허용값

```
DBMS_WARNING.GET_WARNING_SETTING_CAT (
    warning_category IN VARCHAR2) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_NUM (
    warning_number IN NUMBER) RETURN warning_value;
```

```
DBMS_WARNING.GET_WARNING_SETTING_STRING
RETURN pls_integer;
```

```
DBMS_WARNING.GET_CATEGORY (
    warning_number IN pls_integer) RETURN VARCHAR2;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 함수: 구문, 파라미터 및 허용값

warning_category는 범주의 이름입니다. 허용값은 다음과 같습니다.

- ALL
- INFORMATIONAL
- SEVERE
- PERFORMANCE

warning_number는 경고 메시지 번호입니다. 허용값은 모든 유효한 경고 번호입니다.

scope는 변경 사항이 세션 컨텍스트에서 수행되는지 아니면 시스템 컨텍스트에서 수행되는지를 지정합니다. 허용값은 SESSION 또는 SYSTEM입니다. SYSTEM을 사용하려면 ALTER SYSTEM 권한이 필요합니다.

참고: v\$parameter 또는 v\$parameter2 고정 테이블(Fixed Table)에 대한 SELECT 권한이 없거나 경고 문자열을 직접 구문 분석한 후 SET_WARNING_SETTING_STRING을 사용하여 새 값을 수정 및 설정하려는 경우에는 GET_WARNING_SETTING_STRING 함수를 사용하십시오.

DBMS_WARNING 함수: 예제

```
-- Determine the current session warning settings
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_WARNING_SETTING_STRING);
```

```
anonymous block completed
ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
```

```
-- Determine the category for warning message number
-- PLW-07203
SET SERVEROUTPUT ON
EXECUTE DBMS_OUTPUT.PUT_LINE( -
DBMS_WARNING.GET_CATEGORY(7203));
```

```
anonymous block completed
PERFORMANCE
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고

GET_CATEGORY 파라미터의 데이터 유형이 PLS_INTEGER(양의 정수 값 허용)이므로 메시지 번호는 양의 정수로 지정해야 합니다.

DBMS_WARNING 사용: 예제

```

CREATE OR REPLACE PROCEDURE compile_code(p_pkg_name VARCHAR2) IS
  v_warn_value    VARCHAR2(200);
  v_compile_stmt VARCHAR2(200) := 
    'ALTER PACKAGE ' || p_pkg_name || ' COMPILE';

BEGIN
  v_warn_value := DBMS_WARNING.GET_WARNING_SETTING_STRING;
  DBMS_OUTPUT.PUT_LINE('Current warning settings: ' ||
    v_warn_value);
  DBMS_WARNING.ADD_WARNING_SETTING_CAT(
    'PERFORMANCE', 'DISABLE', 'SESSION');
  DBMS_OUTPUT.PUT_LINE('Modified warning settings: ' ||
    DBMS_WARNING.GET_WARNING_SETTING_STRING);
  EXECUTE IMMEDIATE v_compile_stmt;
  DBMS_WARNING.SET_WARNING_SETTING_STRING(v_warn_value,
    'SESSION');
  DBMS_OUTPUT.PUT_LINE('Restored warning settings: ' ||
    DBMS_WARNING.GET_WARNING_SETTING_STRING);
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 사용: 예제

참고: 슬라이드의 예제에 제공되어 있는 코드를 실행하기 전에 demo_10_33.sql의 MY_PKG 스크립트를 생성해야 합니다. 이 데모 스크립트는 MY_PKG 패키지를 생성합니다.

위 슬라이드에 표시된 compile_code 프로시저 예제는 명명된 PL/SQL 패키지를 컴파일하기 위해 작성되었습니다. 코드에서는 PERFORMANCE 범주 경고를 표시하지 않습니다. 호출 환경의 경고 설정은 컴파일이 수행된 후에 복원해야 합니다. 위 코드에서는 호출 환경의 경고 설정이 무엇인지 알 수 없고, 단지 GET_WARNING_SETTING_STRING 함수를 사용하여 현재 설정을 저장합니다. 이 값은 위 예제 코드의 마지막 행에서

DBMS_WARNING.SET_WARNING_SETTING_STRING 프로시저를 사용하여 호출 환경 설정을 복원하는 데 사용됩니다. Native Dynamic SQL을 사용하여 패키지를 컴파일하기 전에 compile_code 프로시저는 PERFORMANCE 범주의 경고를 비활성화하여 현재 세션 경고 레벨을 변경합니다. 또한 이 코드는 원래 경고 설정, 수정된 경고 설정 및 복원된 경고 설정을 인쇄합니다.

DBMS_WARNING 사용: 예제

```
EXECUTE DBMS_WARNING.SET_WARNING_SETTING_STRING(
  'ENABLE:ALL', 'SESSION');
```

anonymous block completed

@code_10_32_s.sql

```
PROCEDURE compile_code(p_pkg_name Compiled.
```

@code_10_33_cs.sql -- compiles the DEPT_PKG package

anonymous block completed
 Current warning settings: ENABLE:ALL
 Modified warning settings: ENABLE:INFORMATIONAL,DISABLE:PERFORMANCE,ENABLE:SEVERE
 Restored warning settings: ENABLE:ALL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_WARNING 사용: 예제(계속)

슬라이드의 예제에서는 이전 슬라이드에서 제공되었던 예제를 테스트합니다. 먼저 모든 컴파일러 경고를 활성화합니다. 그런 다음 이전 페이지의 스크립트를 실행합니다. 마지막으로 compile_code 프로시저를 호출하고 기존 패키지 이름(DEPT_PKG)을 파라미터로 전달합니다.

PLW 06009 경고 메시지 사용

- PLW 경고는 PL/SQL 서브 루틴의 OTHERS 처리기가 다음을 실행하지 않고 종료될 수 있음을 나타냅니다.
 - 일부 RAISE 형식 또는
 - 표준 프로시저 RAISE_APPLICATION_ERROR에 대한 호출
- 적절한 방식으로 프로그래밍하려면 OTHERS 처리기가 항상 예외를 위쪽으로 전달하도록 하는 것이 좋습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

새로운 PLW 06009 경고 사용

적절한 방식으로 프로그래밍하려면 OTHERS 예외 처리기가 호출 서브 루틴 위쪽으로 예외를 전달하도록 해야 합니다. 이 기능을 추가하지 않으면 예외가 인식되지 않을 수도 있습니다. 코드에 이러한 결함이 없도록 하려면 세션에 대해 경고 기능을 설정하고 확인하려는 코드를 재컴파일하면 됩니다. OTHERS 처리기가 예외를 처리하지 않으면 PLW 06009 경고가 이를 알려줍니다.

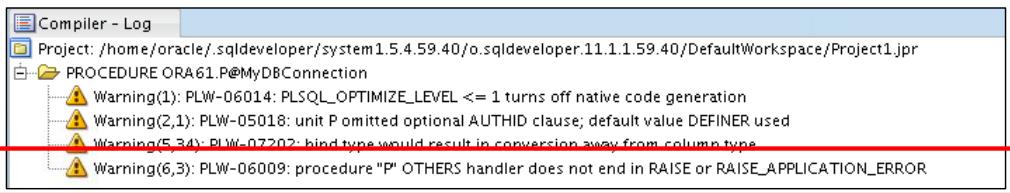
참고: PLW 06009 외에도 Oracle Database 11g에는 새로운 경고 메시지가 더 있습니다. 전체 PLW 경고 리스트를 보려면 *Oracle Database Error Messages 11g Release 2 (11.2)* 설명서를 참조하십시오.

PLW 06009 경고: 예제

```

CREATE OR REPLACE PROCEDURE p(i IN VARCHAR2)
IS
BEGIN
  INSERT INTO t(col_a) VALUES (i);
EXCEPTION
  WHEN OTHERS THEN null;
END p;
/
ALTER PROCEDURE P COMPILE
PLSQL_warnings = 'enable:all' REUSE SETTINGS;

```



```

SELECT *
FROM user_errors
WHERE name = 'P'

```

NAME	TYPE	SEQUEN...	LINE	POSITION	TEXT	ATTR...	MESSAGE_NUMBER
1 P	PROCEDURE	1	4	34	PLW-07202: bind type would result in conversion away from column type	WARNING	7202
2 P	PROCEDURE	2	1	1	PLW-05018: unit P omitted optional AUTHID clause; default value DEFINER used	WARNING	5018
3 P	PROCEDURE	3	0	0	PLW-06014: PLSQL_OPTIMIZE_LEVEL <= 1 turns off native code generation	WARNING	6014
4 P	PROCEDURE	4	5	3	PLW-06009: procedure "P" OTHERS handler does not end in RAISE or RAISE_APPLICATION... WARNING	WARNING	6009

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PLW 06009 경고: 예제

슬라이드의 첫번째 코드를 실행하고 Object Navigation 트리를 사용하여 프로시저를 컴파일하고 나면 **Compiler – Log** 탭에 PLW-06009 경고가 표시됩니다.

`user_error` 데이터 덕셔너리 뷰를 사용하여 오류를 표시할 수도 있습니다. 슬라이드 예제에서 사용되는 테이블 `t`의 정의는 다음과 같습니다.

```
CREATE TABLE t (col_a NUMBER);
```

퀴즈

PL/SQL 컴파일 타임 경고 메시지의 범주:

- 1. SEVERE**
- 2. PERFORMANCE**
- 3. INFORMATIONAL**
- 4. ALL**
- 5. CRITICAL**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 4

PL/SQL 경고 메시지는 여러 범주로 구분되므로 컴파일 중에 비슷한 경고로 구성되는 그룹을 표시하지 않거나 표시할 수 있습니다. 다음과 같은 범주가 있습니다.

- **SEVERE:** 파라미터로 인한 Alias 지정 문제 등 예기치 않은 동작이나 잘못된 결과를 야기할 수 있는 조건에 대한 메시지입니다.
- **PERFORMANCE:** INSERT 문에서 VARCHAR2 값을 NUMBER 열로 전달하는 등 성능 문제를 일으킬 수 있는 조건에 대한 메시지입니다.
- **INFORMATIONAL:** 실행할 수 없는 연결 불가 코드 등 성능에 영향을 주거나 수정을 필요로 하지는 않지만 코드를 보다 쉽게 유지 관리하기 위해 변경할 수 있는 조건에 대한 메시지입니다.
- **ALL:** 모든 범주를 표시합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- PL/SQL 컴파일러 초기화 파라미터 사용
- PL/SQL 컴파일 타임 경고 사용



Copyright © 2009, Oracle. All rights reserved.

연습 10: 개요

이 연습에서는 다음 내용을 다룹니다.

- 컴파일러 초기화 파라미터 표시
- 세션에 대한 Native Compile 활성화 및 프로시저 컴파일
- 컴파일러 경고를 비활성화한 다음 원래 세션 경고 설정 복원
- 일부 컴파일러 경고 메시지 번호의 범주 식별

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 10: 개요

이 연습에서는 컴파일러 초기화 파라미터를 표시합니다. 그런 다음 세션에 대한 Native Compile을 활성화하고 프로시저를 컴파일합니다. 그리고 모든 컴파일러 경고 범주를 숨기고 원래 세션 경고 설정을 복원합니다. 마지막으로 일부 컴파일러 경고 메시지 번호의 범주를 식별합니다.

11

PL/SQL 코드 관리

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 조건부 컴파일 설명 및 사용
- 동적 난독 처리 및 Wrap 유ти리티를 사용하여 PL/SQL 소스 코드 숨기기

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 조건부 컴파일과 PL/SQL 코드 난독 처리(Obfuscation), 즉 래핑을 소개합니다.

단원 내용

- 조건부 컴파일 사용
- PL/SQL 코드에 난독 처리 적용

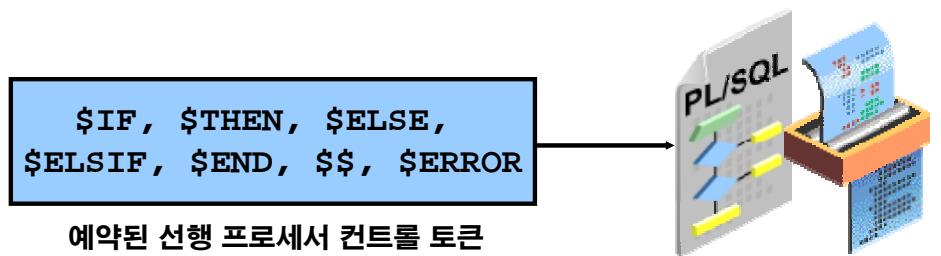
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

조건부 컴파일이란?

**소스 코드를 제거하지 않고 PL/SQL 응용 프로그램의 기능
커스터마이즈 가능:**

- 최신 데이터베이스 버전의 최신 기능을 활용하거나 새로운 기능을 비활성화하여 응용 프로그램을 이전 버전의 데이터베이스에 대해 실행
- 개발 환경에서 디버깅 또는 추적 기능을 활성화하고 응용 프로그램이 운영 사이트에서 실행 중인 동안 해당 기능 숨기기



ORACLE

Copyright © 2009, Oracle. All rights reserved.

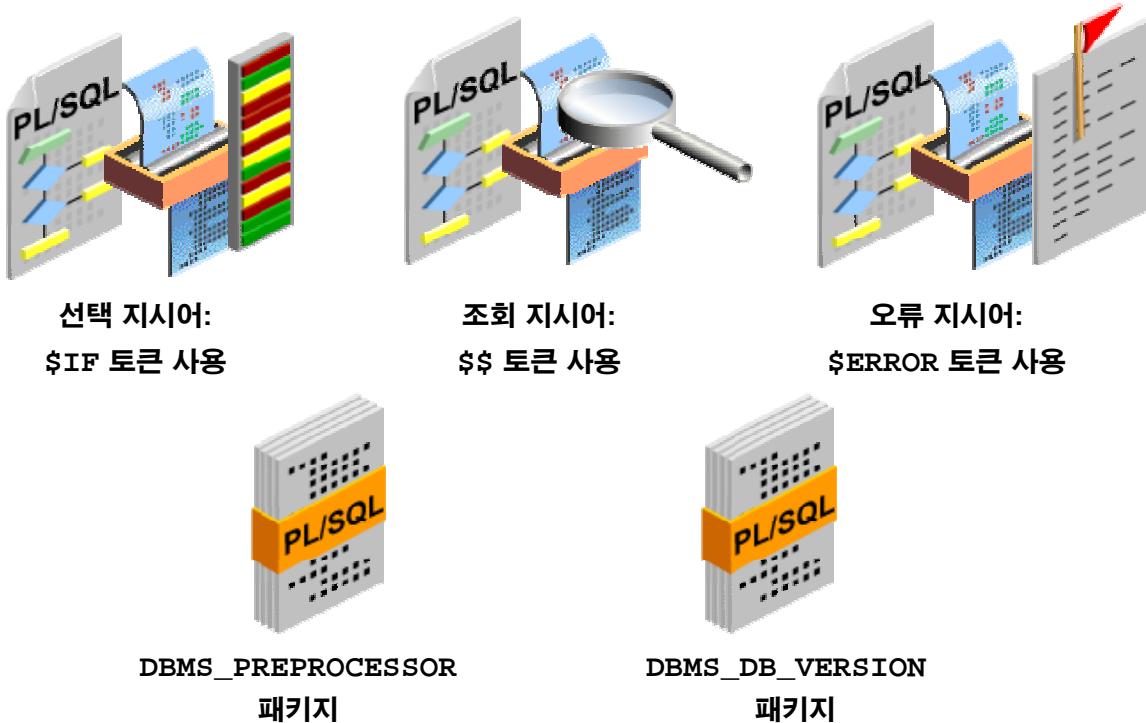
조건부 컴파일이란?

조건부 컴파일을 사용하면 컴파일 중에 평가된 조건의 값에 따라 선택적으로 코드를 포함할 수 있습니다. 예를 들어 조건부 컴파일을 통해 PL/SQL 응용 프로그램에서 특정 데이터베이스 버전에 사용되는 PL/SQL 기능을 확인할 수 있습니다. 응용 프로그램의 최신 PL/SQL 기능을 새 데이터베이스 버전에서 실행할 수 있을 뿐 아니라 조건부로 설정할 수 있으므로 동일한 응용 프로그램이 이전 데이터베이스 버전과 호환됩니다. 조건부 컴파일은 개발 환경에서 디버깅 프로시저를 실행하고 운영 환경에서는 디버깅 루틴을 해제하려는 경우에도 유용합니다.

조건부 컴파일의 이점

- 하나의 소스 코드로 동일 프로그램의 여러 버전 지원
- 손쉬운 코드 유지 관리 및 디버그
- 다른 버전의 데이터베이스로 코드를 간편하게 이전

조건부 컴파일의 작동 방식



ORACLE

Copyright © 2009, Oracle. All rights reserved.

조건부 컴파일의 작동 방식

조건부 컴파일은 PL/SQL 소스 프로그램의 지시어에 포함하여 사용할 수 있습니다. PL/SQL 프로그램을 컴파일용으로 실행하면 실행 프로세서가 이러한 지시어를 평가하고 프로그램의 컴파일 할 부분을 선택합니다. 그러면 선택된 프로그램 소스가 컴파일용으로 컴파일러에 핸드오프됩니다.

조회 지시어는 \$\$ 토큰을 사용하여 컴파일되는 단위에 대한 PL/SQL 컴파일러 초기화 파라미터인 PLSQL_CCFLAGS 또는 PLSQL_OPTIMIZE_LEVEL 값 등의 컴파일 환경을 조회합니다. 이 지시어를 조건부 선택 지시어와 함께 사용하여 프로그램의 컴파일 할 부분을 선택할 수 있습니다.

선택 지시어는 \$IF 생성자를 통해 조건이 충족되는 경우 수행 가능한 컴파일에 대해 코드의 분기 셋션을 생성하여 조회 지시어 또는 정적 패키지 상수를 테스트할 수 있습니다.

오류 지시어는 \$ERROR 토큰을 사용하여 조건부 컴파일 중 예기치 않은 조건이 나타나면 컴파일 오류를 발생시킵니다.

DBMS_DB_VERSION 패키지는 조건부 컴파일에 사용할 수 있는 데이터베이스 버전 상수를 제공합니다.

DBMS_PREPROCESSOR 패키지는 PL/SQL 단위에서 조건부 컴파일 지시어에 의해 선택되는 포스트 처리된 소스 텍스트에 액세스하기 위한 서브 프로그램을 제공합니다.

선택 지시어 사용

```
$IF <Boolean-expression> $THEN Text
$ELSEIF <Boolean-expression> $THEN Text
. . .
$ELSE Text
$END
```

```
DECLARE
CURSOR cur IS SELECT employee_id FROM
employees WHERE
$IF myapp_tax_package.new_tax_code $THEN
    salary > 20000;
$ELSE
    salary > 50000;
$END
BEGIN
    OPEN cur;
. . .
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

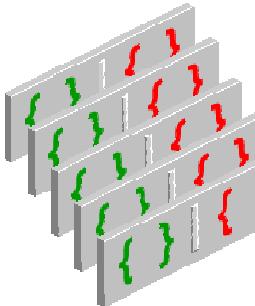
선택 지시어 사용

조건부 선택 지시어는 PL/SQL 적시 프로세스의 IF-THEN-ELSE 메커니즘과 모양 및 작동 방식이 비슷합니다. 선행 프로세서는 \$THEN을 발견하면 \$IF와 \$THEN 사이의 텍스트가 정적 표현식인지 여부를 확인합니다. 확인이 성공하고 평가 결과가 TRUE이면 \$THEN과 \$ELSE 또는 \$ELSIF 사이의 PL/SQL 프로그램 텍스트가 컴파일을 위해 선택됩니다.

선택 조건(\$IF 및 \$THEN 사이의 표현식)은 다른 패키지에 정의되어 있는 상수나 조회 지시어 또는 이 두 항목의 조합을 참조하여 생성할 수 있습니다.

슬라이드에 나와 있는 예제에서 조건부 선택 지시어는 MYAPP_TAX_PACKAGE.NEW_TAX_CODE 값을 기준으로 커서의 두 버전 중 하나를 선택합니다. 값이 TRUE이면 급여가 20000보다 큰 사원이 선택되고, 그렇지 않으면 50000보다 큰 사원이 선택됩니다.

미리 정의된 조회 지시어 및 유저 정의 조회 지시어 사용



PLSQL_CCFLAGS
PLSQL_CODE_TYPE
PLSQL_OPTIMIZE_LEVEL
PLSQL_WARNINGS
NLS_LENGTH_SEMANTICS
PLSQL_LINE
PLSQL_UNIT

미리 정의된 조회 지시어

```
PLSQL_CCFLAGS = 'plsql_ccflags:true,debug:true,debug:0';
```

유저 정의 조회 지시어

ORACLE

Copyright © 2009, Oracle. All rights reserved.

미리 정의된 조회 지시어 및 유저 정의 조회 지시어 사용

조회 지시어는 미리 정의되어 있을 수도 있고 유저가 정의할 수도 있습니다. 다음 항목에서는 조건부 컴파일 과정에서 조회 지시어를 분석하려 할 때의 처리 흐름 순서를 설명합니다.

1. ID는 검색 키에 대해 `$$id` 형식의 조회 지시어로 사용됩니다.
2. 다음과 같은 two-pass 알고리즘이 진행됩니다.
 - a. `PLSQL_CCFLAGS` 초기화 파라미터의 문자열이 오른쪽에서 왼쪽으로 스캔되어 이름이 일치하는 ID(대소문자 구분 안 함)를 검색합니다. 해당하는 ID가 발견되면 작업은 완료됩니다.
 - b. 미리 정의된 조회 지시어를 검색합니다. 해당하는 항목이 발견되면 작업이 완료됩니다.
3. `$$ID`를 값으로 분석할 수 없는 경우 소스 텍스트가 래핑되지 않으면 PLW-6003 경고 메시지가 보고됩니다. NULL 리터럴이 정의되지 않은 조회 지시어 값으로 치환됩니다. PL/SQL 코드를 래핑하는 경우에는 정의되지 않은 조회 지시어가 표시되지 않도록 경고 메시지가 비활성화됩니다.

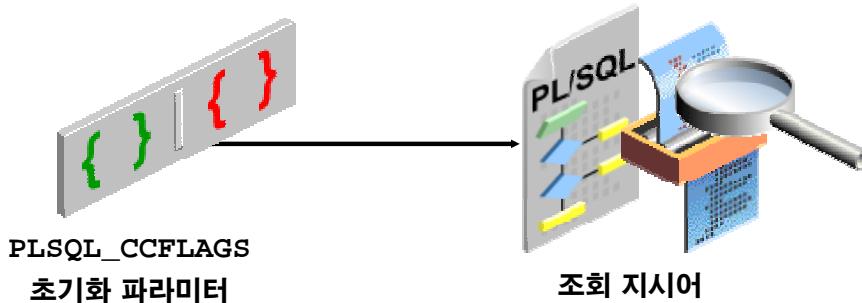
슬라이드의 예제에서는 `$$debug`의 값이 0이고 `$$plsql_ccflags`의 값은 TRUE입니다. `$$plsql_ccflags` 값은 `PLSQL_CCFLAGS` 컴파일러 파라미터 값 내에서 유저 정의 `plsql_ccflags`로 분석되는데, 이는 유저 정의 지시어가 미리 정의된 지시어를 재정의하기 때문입니다.

PLSQL_CCFLAGS 파라미터 및 조회 지시어

PLSQL_CCFLAGS 파라미터를 사용하여 각 PL/SQL 라이브러리 단위의 조건부 컴파일을 개별적으로 제어

```
PLSQL_CCFLAGS = '<v1>:<c1>,<v2>:<c2>,...,<vn>:<cn>'
```

```
ALTER SESSION SET
PLSQL_CCFLAGS = 'plsql_ccflags:true, debug:true, debug:0';
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

PLSQL_CCFLAGS 파라미터 및 조회 지시어

Oracle Database 10g Release 2에는 조건부 컴파일에 사용하기 위한 새로운 오라클 초기화 파라미터인 PLSQL_CCFLAGS가 도입되었습니다. 이 Dynamic Parameter를 사용하면 이름-값 쌍을 설정할 수 있습니다. 그러면 이름(플래그 이름)을 조회 지시어에서 참조할 수 있습니다. PL/SQL 프로그래머는 PLSQL_CCFLAGS를 사용하여 각 PL/SQL 라이브러리 단위의 조건부 컴파일을 개별적으로 제어할 수 있습니다.

값

- vi: 따옴표가 없는 PL/SQL 식별자 형태입니다. 제한이 없으며 예약어 또는 키워드일 수 있습니다. 텍스트는 대소문자를 구분하지 않습니다. 각 항목은 플래그 또는 플래그 이름으로 알려져 있습니다. 개별 vi가 문자열에서 여러 번 나타날 수 있으며 각 항목의 플래그 값과 플래그 값의 종류는 서로 다를 수 있습니다.
- ci: 다음 중 하나일 수 있습니다.
 - PL/SQL 부울 리터럴
 - PLS_INTEGER 리터럴
 - 리터럴 NULL(기본값). 텍스트는 대소문자를 구분하지 않습니다. 각 항목은 플래그 값으로 알려져 있으며 플래그 이름에 대응됩니다.

PLSQL_CCFLAGS 초기화 파라미터 설정 표시

```
SELECT name, type, plsql_ccflags
FROM user_plsql_object_settings
```

Results:

NAME	TYPE	PLSQL_CCFLAGS
1 DEPT_PKG	PACKAGE	(null)
2 DEPT_PKG	PACKAGE BODY	(null)
3 TAXES_PKG	PACKAGE	(null)
4 TAXES_PKG	PACKAGE BODY	(null)
5 EMP_PKG	PACKAGE	(null)
6 EMP_PKG	PACKAGE BODY	(null)
7 SECURE_DML	PROCEDURE	(null)
8 SECURE_EMPLOYEES	TRIGGER	(null)
9 ADD_JOB_HISTORY	PROCEDURE	plsql_ccflags:true, debug:true, debug:0
10 UPDATE_JOB_HISTORY	TRIGGER	(null)

...

Copyright © 2009, Oracle. All rights reserved.

PLSQL_CCFLAGS 초기화 파라미터 설정 표시

USER | ALL | DBA_PLSQL_OBJECT_SETTINGS 데이터 딕셔너리 뷰를 사용하여 PL/SQL 객체의 설정을 표시합니다.

PLSQL_CCFLAGS에 대해 허용 가능한 모든 값을 정의할 수 있습니다. 그러나 오라클에서는 이 파라미터를 디버깅 또는 추적 코드의 조건부 컴파일을 제어하는 데 사용할 것을 권장합니다. 플래그 이름은 예약어 및 키워드를 포함하여 어떤 식별자로든 설정할 수 있습니다. 이 값은 TRUE, FALSE, NULL 또는 PLS_INTEGER 리터럴이거나 PLS_INTEGER 리터럴이어야 합니다. 플래그 이름 및 값은 대소문자를 구분하지 않습니다. PLSQL_CCFLAGS 파라미터는 다른 컴파일러 파라미터와 같이 PL/SQL 컴파일러 파라미터이며 PL/SQL 프로그램 단위와 함께 저장됩니다. 그러므로 이후에 REUSE SETTINGS 절(예: ALTER PACKAGE ...REUSE SETTINGS)을 사용하여 PL/SQL 프로그램을 재컴파일하면 동일한 값의 PLSQL_CCFLAGS가 재컴파일에 사용됩니다. PLSQL_CCFLAGS 파라미터는 각 PL/SQL 단위에 대해 다른 값으로 설정할 수 있으므로 조건부 컴파일을 단위별로 편리하게 제어할 수 있습니다.

PLSQL_CCFLAGS 파라미터 및 조회 지시어: 예제

```

ALTER SESSION SET PLSQL_CCFLAGS = 'Tracing:true';
CREATE OR REPLACE PROCEDURE P IS
BEGIN
  $IF $$tracing $THEN
    DBMS_OUTPUT.PUT_LINE ('TRACING');
  $END
END P;

```

ALTER SESSION SET succeeded.
PROCEDURE P Compiled.

```

SELECT name, plsql_ccflags
FROM USER_PLSQL_OBJECT_SETTINGS
WHERE name = 'P';

```

Results:

	NAME	PLSQL_CCFLAGS
1	P	Tracing:true

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PLSQL_CCFLAGS 파라미터 및 조회 지시어: 예제

위 슬라이드의 예제에서는 파라미터를 설정한 후에 프로시저를 생성합니다. 설정은 각 PL/SQL 단위와 함께 저장됩니다.

조건부 컴파일 오류 지시어를 사용하여 유저 정의 오류 발생

```
$ERROR varchar2_static_expression $END

ALTER SESSION SET Plsql_CCFlags = ' Trace_Level:3 '
/ CREATE PROCEDURE P IS
BEGIN
  $IF    $$Trace_Level = 0 $THEN ...
  $ELSIF $$Trace_Level = 1 $THEN ...
  $ELSIF $$Trace_Level = 2 $THEN ...
  $else $error 'Bad: '||$$Trace_Level $END -- error
                                -- directive
$END -- selection directive ends
END P;
```

```
SHOW ERRORS
Errors for PROCEDURE P:
LINE/COL  ERROR
-----
6 /9       PLS-00179: $ERROR: Bad: 3
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조건부 컴파일 오류 지시어를 사용하여 유저 정의 오류 발생

\$ERROR 오류 지시어는 유저 정의 오류를 발생시키며 다음과 같은 형태로 이루어집니다.

```
$ERROR varchar2_static_expression $END
```

참고: varchar2_static_expression은 VARCHAR2 정적 표현식이어야 합니다.

조건부 컴파일에 정적 표현식 사용

- **Boolean 정적 표현식:**
 - TRUE, FALSE, NULL, IS NULL, IS NOT NULL
 - >, <, >=, <=, =, <>, NOT, AND, OR
- **PLS_INTEGER 정적 표현식:**
 - -2147483648 ~ 2147483647, NULL
- **VARCHAR2 정적 표현식에 포함되는 항목:**
 - ||, NULL, TO_CHAR
- **정적 상수:**

```
static_constant CONSTANT datatype := static_expression;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조건부 컴파일에 정적 표현식 사용

앞에서 설명한 것처럼 선행 프로세서는 적시 컴파일이 시작되기 전에 조건부 지시어를 처리합니다. 따라서 조건부 컴파일 지시어에는 컴파일 시 완전히 평가할 수 있는 표현식만 사용할 수 있습니다. PL/SQL을 실행해야 하는 변수 또는 함수에 대한 참조를 포함하는 표현식은 컴파일 중에 사용할 수 없으며 평가할 수도 없습니다.

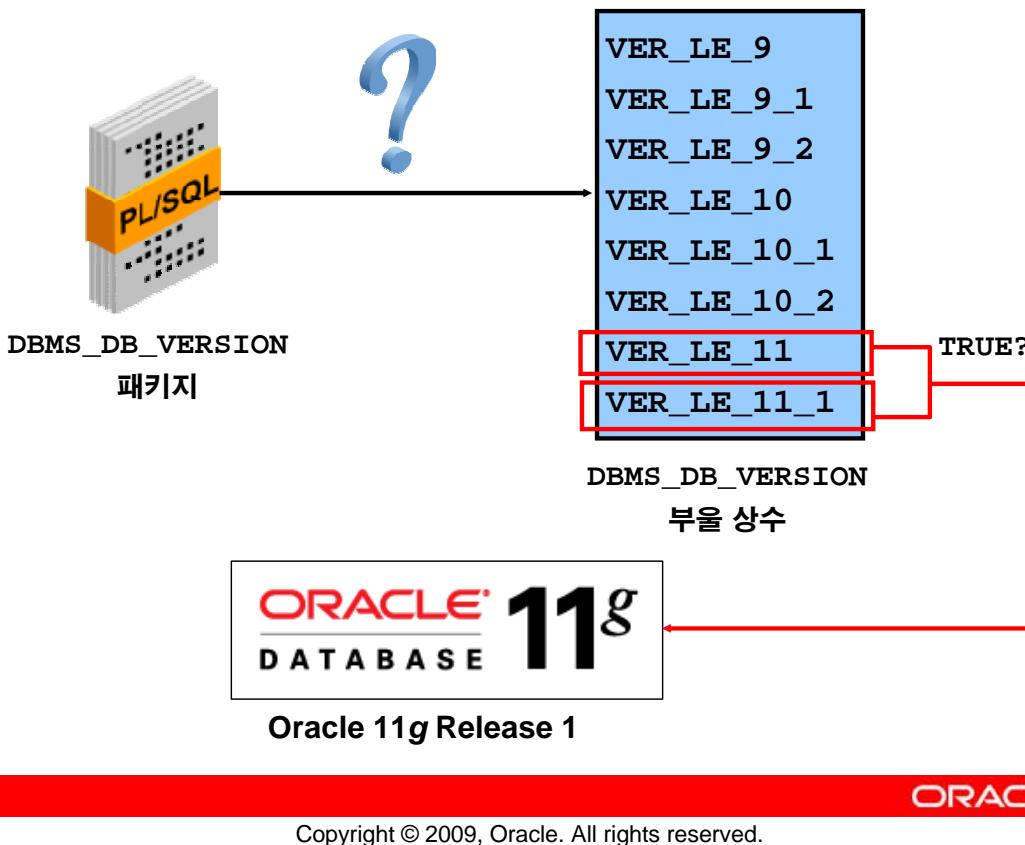
조건부 컴파일 지시어에서 사용할 수 있는 이러한 PL/SQL 표현식 하위 집합을 정적 표현식(Static Expression)이라고 합니다. 정적 표현식은 단위에서 사용하는 값이 변경되지 않은 상태로 해당 단위가 자동으로 재컴파일되는 경우 표현식이 같은 방식으로 평가되고 동일한 소스가 컴파일되도록 주의해서 정의해야 합니다.

일반적으로 정적 표현식은 다음과 같은 세 가지 소스로 구성됩니다.

- \$\$로 표시되는 조회 지시어
- DBMS_DB_VERSION 등의 PL/SQL 패키지에서 정의되는 상수. 이러한 값은 PL/SQL의 일반 연산을 사용하여 결합 및 비교할 수 있습니다.
- TRUE, FALSE, 'CA', 123, NULL 등의 리터럴

정적 표현식은 비교, 논리 부울 연산(OR 및 AND 등) 또는 정적 문자 표현식 연결 등이 들어 있는 연산을 포함할 수도 있습니다.

DBMS_DB_VERSION 패키지: 부울 상수



DBMS_DB_VERSION 패키지

Oracle Database 10g Release 2에는 `DBMS_DB_VERSION` 패키지가 도입되었습니다. 이 패키지는 조건부 컴파일에 대해 간단한 선택을 할 때 유용한 오라클 데이터베이스 버전 번호를 지정합니다. 상수는 버전보다 작거나 같은 부울 조건(있는 경우)을 나타냅니다.

예제

`VER_LE_11`은 데이터베이스 버전이 ≤ 11 임을 나타냅니다. 상수의 값은 TRUE 또는 FALSE입니다. 예를 들어 Oracle Database 11g Release 1 데이터베이스에서 `VER_LE_11` 및 `VER_LE_11_1`은 TRUE이고 다른 상수는 모두 FALSE입니다.

DBMS_DB_VERSION 패키지 상수

이름	값	설명
VER_LE_9	FALSE	Version <= 9.
VER_LE_9_1	FALSE	Version <= 9 and release <= 1.
VER_LE_9_2	FALSE	Version <= 9 and release <= 2.
VER_LE_10	TRUE	Version <= 10.
VER_LE_10_1	FALSE	Version <= 10 and release <= 1.
VER_LE_10_2	TRUE	Version <= 10 and release <= 2.
VER_LE_11	FALSE	Version <= 11.
VER_LE_11_1	TRUE	Version <= 11 and release <= 1.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_DB_VERSION 패키지

다음은 Oracle Database 11g Release 1 버전의 패키지입니다.

```
PACKAGE DBMS_DB_VERSION IS
    VERSION CONSTANT PLS_INTEGER := 11; -- RDBMS version
                                         -- number
    RELEASE CONSTANT PLS_INTEGER := 1;   -- RDBMS release
                                         -- number
    ver_le_9_1      CONSTANT BOOLEAN := FALSE;
    ver_le_9_2      CONSTANT BOOLEAN := FALSE;
    ver_le_9        CONSTANT BOOLEAN := FALSE;
    ver_le_10_1     CONSTANT BOOLEAN := FALSE;
    ver_le_10_2     CONSTANT BOOLEAN := FALSE;
    ver_le_10       CONSTANT BOOLEAN := FALSE;
    ver_le_11_1     CONSTANT BOOLEAN := TRUE;
    ver_le_11       CONSTANT BOOLEAN := TRUE;
END DBMS_DB_VERSION;
```

DBMS_DB_VERSION 패키지에는 Oracle Database 버전마다 다른 상수가 포함되어 있습니다.

Oracle Database 11g Release 1 버전의 DBMS_DB_VERSION 패키지에서는 슬라이드에 나와 있는 상수를 사용합니다.

데이터베이스 버전과 함께 조건부 컴파일 사용: 예제

```

ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE, my_tracing:FALSE';
CREATE PACKAGE my_pkg AS
  SUBTYPE my_real IS
    -- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
    -- else use NUMBER data type
    $IF DBMS_DB_VERSION.VERSION < 10 $THEN    NUMBER;
    $ELSE    BINARY_DOUBLE;
    $END
    my_pi my_real; my_e my_real;
  END my_pkg;
/
CREATE PACKAGE BODY my_pkg AS
BEGIN
  $IF DBMS_DB_VERSION.VERSION < 10 $THEN
    my_pi := 3.14016408289008292431940027343666863227;
    my_e  := 2.71828182845904523536028747135266249775;
  $ELSE
    my_pi := 3.14016408289008292431940027343666863227d;
    my_e  := 2.71828182845904523536028747135266249775d;
  $END
END my_pkg;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 버전과 함께 조건부 컴파일 사용: 예제

이 예제에서는 PLSQL_CCFLAGS 파라미터를 사용하는 방법도 보여줍니다. 먼저 디버깅 코드 및 추적 정보를 표시하기 위해 PLSQL_CCFLAGS 파라미터 플래그를 설정합니다.

이 페이지와 다음 페이지의 슬라이드 예제에서 조건부 컴파일은 데이터베이스 버전의 코드를 지정하는 데 사용됩니다. 조건부 컴파일을 사용하여 BINARY_DOUBLE 데이터 유형을 데이터베이스의 PL/SQL 단위 계산에 사용할 수 있는지 여부를 결정합니다. BINARY_DOUBLE 데이터 유형은 Oracle Database 10g 이상에서만 사용할 수 있습니다. Oracle Database 10g를 사용하는 경우 my_real의 데이터 유형은 BINARY_DOUBLE이고 그렇지 않은 경우 my_real의 데이터 유형은 NUMBER입니다.

새 Package Spec(my_pkg)에서 조건부 컴파일은 데이터베이스 버전을 확인하는 데 사용됩니다. 그리고 Package Body 정의에서 조건부 컴파일을 재사용하여 데이터베이스 버전을 기준으로 이후 계산에 사용할 my_pi 및 my_e 값을 설정합니다.

슬라이드 예제 코드의 결과는 다음과 같습니다.

```

ALTER SESSION SET succeeded.
PACKAGE my_pkg Compiled.
PACKAGE BODY my_pkg Compiled.

```

데이터베이스 버전과 함께 조건부 컴파일 사용: 예제

```

CREATE OR REPLACE PROCEDURE circle_area(p_radius my_pkg.my_real) IS
  v_my_area my_pkg.my_real;
  v_my_datatype VARCHAR2(30);
BEGIN
  v_my_area := my_pkg.my_pi * p_radius * p_radius;
  DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(p_radius)
    || ' Area: ' || TO_CHAR(v_my_area));
  $IF $$my_debug $THEN -- if my_debug is TRUE, run some debugging code
    SELECT DATA_TYPE INTO v_my_datatype FROM USER_ARGUMENTS
    WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME = 'P_RADIUS';
    DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument is: ' ||
      v_my_datatype);
  $END
END; /

```

PROCEDURE circle_area(p_radius Compiled.

CALL circle_area(50); -- Using Oracle Database 11g Release 2

CALL circle_area(50) succeeded.
Radius: 5.0E+001 Area: 7.8504102072252062E+003

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 버전과 함께 조건부 컴파일 사용: 예제(계속)

슬라이드의 예제에는 `circle_area`라는 새 프로시저가 정의되어 있습니다. 이 프로시저는 이전 페이지에서 정의한 `my_pkg` 패키지의 변수 `pi`를 기준으로 원 면적을 계산합니다. 이 프로시저에는 IN 형식 파라미터인 `radius`가 하나 있습니다.

이 프로시저는 `my_area`와 `my_datatype` 두 변수를 선언합니다. `my_area`는 `my_pkg`의 `my_real`과 데이터 유형이 같고 `my_datatype`은 `VARCHAR2(30)` 형식입니다.

프로시저 본문에서 `my_area`는 `my_pkg`에서 설정하는 `my_pi` 값에 반지름으로 프로시저에 전달되는 값을 곱한 값과 같아집니다. 슬라이드의 두번째 코드 예제에서처럼 반지름과 원 면적을 표시하는 메시지가 인쇄됩니다.

참고: `my_debug`를 `TRUE`로 설정하려는 경우에는 다음과 같이 `REUSE SETTINGS` 절을 사용하여 `circle_area` 프로시저만 이렇게 변경하면 됩니다.

```

ALTER PROCEDURE circle_area COMPILE PLSQL_CCFLAGS =
  'my_debug:TRUE' REUSE SETTINGS;

```

DBMS_PREPROCESSOR 프로시저를 사용하여 소스 텍스트 인쇄 또는 검색

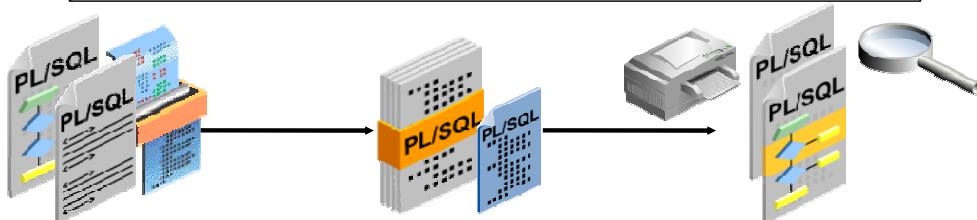
CALL

```
DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE'
, 'ORA61', 'MY_PKG');
```

```

CALL DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE('PACKAGE', succeeded.
PACKAGE my_pkg AS
SUBTYPE my_real IS
-- Check the database version, if >= 10g, use BINARY_DOUBLE data type,
-- else use NUMBER data type
BINARY_DOUBLE;
my_pi my_real; my_e my_real;
END my_pkg;

```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_PREPROCESSOR 프로시저를 사용하여 소스 텍스트 인쇄 또는 검색

DBMS_PREPROCESSOR 서브 프로그램은 조건부 컴파일 지시어를 처리한 후에 PL/SQL 단위의 포스트 처리된 소스 텍스트를 인쇄하거나 검색합니다. 이 포스트 처리된 텍스트는 올바른 PL/SQL 단위를 컴파일하는 데 사용되는 실제 소스입니다. 슬라이드 쇼의 예제에서는 PRINT_POST_PROCESSED_SOURCE 프로시저를 사용하여 my_pkg의 포스트 처리된 품을 인쇄하는 방법을 보여줍니다.

Oracle Database 10g 버전 이상 데이터베이스에서 HR 계정을 사용하여 my_pkg를 컴파일하는 경우 슬라이드에서 첫번째 예제의 출력이 슬라이드의 예제에 나와 있습니다.

PRINT_POST_PROCESSED_SOURCE는 선택하지 않은 텍스트를 제거합니다. 즉, 포스트 처리된 텍스트에 포함되지 않은 코드 행은 제거됩니다. PRINT_POST_PROCESSED_SOURCE 프로시저의 인수는 객체 유형, 스키마 이름(수강생 계정 ORA61 사용) 및 객체 이름입니다.

참고: DBMS_PREPROCESSOR 패키지에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

단원 내용

- 조건부 컴파일 사용
- PL/SQL 코드에 난독 처리 적용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

난독 처리란?

- PL/SQL 단위의 난독 처리, 즉 래핑은 PL/SQL 소스 코드를 숨기는 프로세스입니다.
- 래핑은 Wrap 유ти리티 및 DBMS_DDL 서브 프로그램을 사용하여 수행할 수 있습니다.
- Wrap 유ти리티는 명령행에서 실행하며 SQL*Plus 설치 스크립트 등의 입력 SQL 파일을 처리합니다.
- DBMS_DDL 서브 프로그램은 동적으로 생성된 단일 CREATE PROCEDURE 명령과 같은 단일 PL/SQL 단위를 래핑합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

참고

난독 처리에 대한 자세한 내용은 *Oracle Database PL/SQL Language Reference 11g Release 2 (11.2)* 설명서를 참조하십시오.

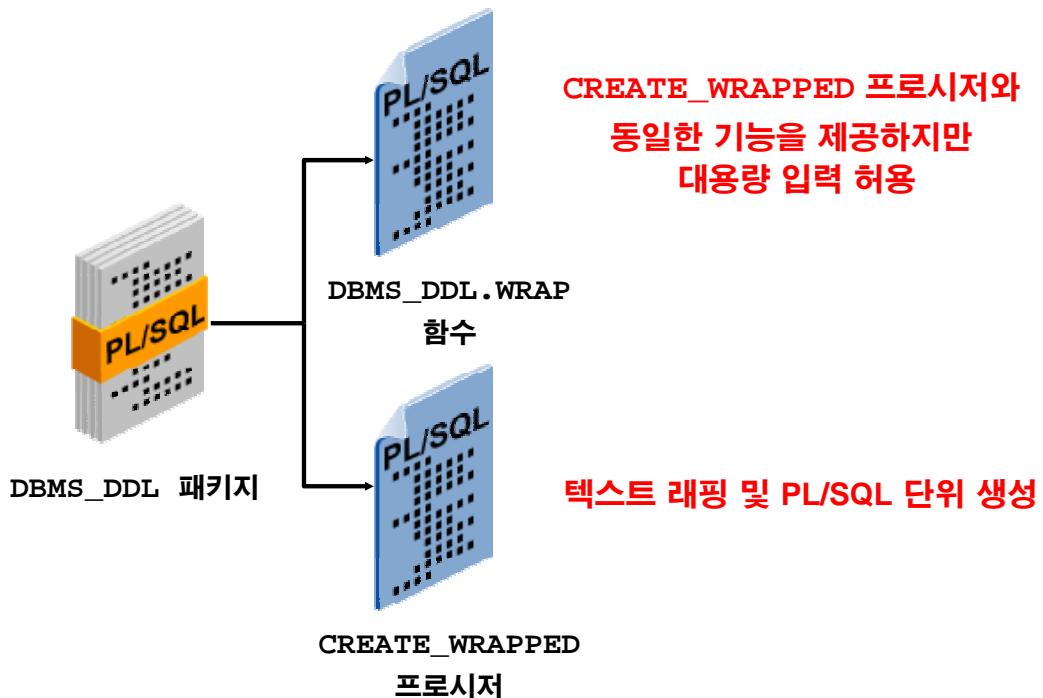
난독 처리의 이점

- 다른 사람이 소스 코드를 볼 수 없습니다.
- 소스 코드가 USER_SOURCE, ALL_SOURCE 또는 DBA_SOURCE 데이터 딕셔너리 뷰를 통해 표시되지 않습니다.
- SQL*Plus가 난독 처리된 소스 파일을 처리할 수 있습니다.
- Import 및 Export 유ти리티가 래핑된 파일을 그대로 사용합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Oracle 10g 이후 동적 난독 처리의 새로운 기능



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle 10g 이후 동적 난독 처리의 새로운 기능

CREATE_WWRAPPED 프로시저

이 프로시저는 PL/SQL Package Spec, Package Body, 함수, 프로시저, Type Spec 또는 Type Body 생성을 지정하는 단일 CREATE OR REPLACE 문을 입력으로 가져와 난독 처리된 PL/SQL 소스 텍스트로 CREATE OR REPLACE 문을 생성한 다음 생성된 명령문을 실행합니다.

WRAP 함수

이 함수는 PL/SQL Package Spec, Package Body, 함수, 프로시저, Type Spec 또는 Type Body 생성을 지정하는 CREATE OR REPLACE 문을 입력으로 가져와 CREATE OR REPLACE 문을 반환합니다. 여기서 PL/SQL 단위의 텍스트가 난독 처리됩니다.

난독 처리가 적용되지 않은 PL/SQL 코드: 예제

```
SET SERVEROUTPUT ON
BEGIN -- The ALL_SOURCE view family shows source code
EXECUTE IMMEDIATE '
CREATE OR REPLACE PROCEDURE P1 IS
BEGIN
    DBMS_OUTPUT.PUT_LINE ('I am not wrapped');
END P1;
';
END;
/
CALL p1();
```

```
anonymous block completed
CALL p1() succeeded.
I'm not wrapped
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

	TEXT
1	PROCEDURE P1 IS
2	BEGIN
3	DBMS_OUTPUT.PUT_LINE ('I am not wrapped');
4	END P1;

ORACLE

Copyright © 2009, Oracle. All rights reserved.

난독 처리가 적용되지 않은 PL/SQL 코드: 예제

슬라이드의 첫번째 예제에서 EXECUTE IMMEDIATE 문은 P1 프로시저를 생성하는 데 사용됩니다. 생성된 프로시저의 코드는 래핑되지 않습니다. 이 코드는 ALL_SOURCE 뷰 계열의 뷰를 사용하여 슬라이드에서처럼 프로시저의 코드를 표시할 때 숨겨지지 않습니다.

난독 처리가 적용된 PL/SQL 코드: 예제

```
BEGIN -- ALL_SOURCE view family obfuscates source code
  DBMS_DDL.CREATE_WRAPPED (
    CREATE OR REPLACE PROCEDURE P1 IS
      BEGIN
        DBMS_OUTPUT.PUT_LINE (''I am wrapped now'');
      END P1;
    );
  END;
/
CALL p1();
```

```
anonymous block completed
call pl() succeeded.
I am wrapped now
```

```
SELECT text FROM user_source
WHERE name = 'P1' ORDER BY line;
```

```
TEXT
-----
PROCEDURE P1 wrapped
a000000
b2
abcd
```

...

Copyright © 2009, Oracle. All rights reserved.

난독 처리가 적용된 PL/SQL 코드: 예제

슬라이드의 예제에서 DBMS_DDL.CREATE_WRAPPED 패키지 프로시저는 P1 프로시저를 생성하는 데 사용됩니다.

이 코드는 다음 페이지와 같이 ALL_SOURCE 뷰 계열의 뷰를 사용하여 프로시저의 코드를 표시할 때 난독 처리됩니다. *_SOURCE 뷰를 확인하면 슬라이드의 명령 출력에 나와 있는 것처럼 다른 사람이 코드 상세 정보를 볼 수 없도록 소스가 래핑, 즉 숨겨짐을 알 수 있습니다.

동적 난독 처리: 예제

```

SET SERVEROUTPUT ON

DECLARE
  c_code CONSTANT VARCHAR2(32767) :=
' CREATE OR REPLACE PROCEDURE new_proc AS
  v_VDATE DATE;
BEGIN
  v_VDATE := SYSDATE;
  DBMS_OUTPUT.PUT_LINE(v_VDATE) ;
END; '
BEGIN
  DBMS_DDL.CREATE_WRAPPED (c_CODE);
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

동적 난독 처리: 예제

슬라이드의 예제에서는 NEW_PROC라는 동적으로 난독 처리된 프로시저의 생성을 보여줍니다. NEW_PROC의 코드가 난독 처리되는지 여부를 확인하려면 다음과 같이 DBA|ALL|USER_SOURCE 텁셔너리 뷰에서 query를 수행하면 됩니다.

```

SELECT text FROM user_source
WHERE name = 'NEW_PROC';

```

TEXT

```

-----
PROCEDURE new_proc wrapped
a000000
-----
```

7

71 9e

```

hBWmpGeSsd58b4jCP3/0d04rof0wg5nnm7+fMr2ywFyFDGLQlhaXriu4dCuPCWnnx1J0U1xp
pvc8nsr7Seq/riQvHRsXAQovdoh0K6ZvM1Kbskr+KLK957KzHQYwLK4k6rJLC$5EyJ7qJB/2
RDmm3j79Uw==
```

1 rows selected

PL/SQL 래퍼 유ти리티

- **PL/SQL 래퍼는 PL/SQL 소스 코드를 이식 가능한 객체 코드로 변환하여 응용 프로그램 내부 사항을 숨기는 독립형 유ти리티입니다.**
- **래핑에는 다음 기능이 있습니다.**
 - 플랫폼 독립성
 - 동적 로드
 - 동적 바인드
 - 종속성 검사
 - 호출 시 정상적인 임포트 및 엑스포트

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 래퍼

PL/SQL 래퍼는 PL/SQL 소스 코드를 이식 가능한 객체 코드로 변환하는 독립형 유ти리티입니다. 래퍼를 사용하면 소스 코드를 공개하지 않고도 고유 알고리즘 및 데이터 구조를 포함할 수 있는 PL/SQL 응용 프로그램을 제공할 수 있습니다. 래퍼는 읽을 수 있는 소스 코드를 읽을 수 없는 코드로 변환합니다. 응용 프로그램 내부 사항을 숨김으로써 응용 프로그램이 잘못 사용되는 경우를 방지할 수 있습니다.

PL/SQL 내장 프로그램과 같은 래핑된 코드는 다음과 같은 몇 가지 기능을 갖습니다.

- 플랫폼 독립적이므로 동일한 컴파일 단위의 여러 버전을 제공할 필요가 없습니다.
- 동적 로드를 허용하므로 유저가 새 기능을 추가하기 위해 종료했다가 다시 시작할 필요가 없습니다.
- 동적 바인드를 허용하므로 로드 시에 external 참조가 분석됩니다.
- 엄격한 종속성 검사 기능을 제공하므로 무효화된 프로그램 단위는 호출될 때 자동으로 재컴파일됩니다.
- 정상적인 임포트 및 엑스포트를 지원하므로 Import/Export 유ти리티로 래핑된 파일을 처리할 수 있습니다.

래퍼 유ти리티 실행

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

- 등호 양쪽에는 공백을 넣지 마십시오.
- INAME 인수는 필수입니다.
- 이름과 함께 지정하지 않는 한 입력 파일의 기본 확장자는 .sql입니다.
- ONAME 인수는 선택 사항입니다.
- ONAME 인수와 함께 지정하지 않는 한 출력 파일의 기본 확장자는 .plb입니다.

예제

```
WRAP INAME=demo_04_hello.sql
WRAP INAME=demo_04_hello
WRAP INAME=demo_04_hello.sql ONAME=demo_04_hello.plb
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

래퍼 실행

래퍼는 WRAP이라는 운영 체제 실행 파일입니다. 래퍼를 실행하려면 운영 체제 프롬프트에 다음 명령을 입력합니다.

```
WRAP INAME=input_file_name [ONAME=output_file_name]
```

슬라이드에 표시된 각 예제는 demo_04_hello.sql을 입력 파일로 사용하고 demo_04_hello.plb를 출력 파일로 생성합니다.

래핑된 파일이 생성되면 SQL 스크립트 파일을 실행할 때처럼 SQL*Plus에서 .plb 파일을 실행하여 소스 코드의 래핑된 버전을 컴파일한 후 저장하십시오.

참고

- INAME 인수만 필수 인수입니다. ONAME 인수를 지정하지 않으면 출력 파일은 입력 파일과 동일한 이름에 확장자 .plb가 붙습니다.
- 입력 파일은 임의 확장자를 가질 수 있지만 기본 확장자는 .sql입니다.
- 운영 체제에 따라 INAME 및 ONAME 값에 대해 대소문자를 구분하는 경우도 있습니다.
- 일반적으로 출력 파일이 입력 파일보다 훨씬 더 큽니다.
- INAME 및 ONAME 인수와 값의 등호 앞뒤에 공백을 넣지 마십시오.

래핑 결과

```
-- Original PL/SQL source code in input file:

CREATE PACKAGE banking IS
    min_bal := 100;
    no_funds EXCEPTION;
    ...
END banking;
/
```

```
-- Wrapped code in output file:

CREATE PACKAGE banking
    wrapped
012abc463e ...
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

래핑 결과

래핑된 객체 유형, 패키지 또는 서브 프로그램은 헤더, 단어 wrapped 및 암호화된(encrypted) 본문으로 구성됩니다.

입력 파일은 임의 SQL 문 조합을 포함할 수 있습니다. 그러나 PL/SQL 래퍼는 다음 CREATE 문만을 래핑합니다.

- CREATE [OR REPLACE] TYPE
- CREATE [OR REPLACE] TYPE BODY
- CREATE [OR REPLACE] PACKAGE
- CREATE [OR REPLACE] PACKAGE BODY
- CREATE [OR REPLACE] FUNCTION
- CREATE [OR REPLACE] PROCEDURE

다른 모든 SQL CREATE 문은 변경되지 않은 상태로 출력 파일에 전달됩니다.

래핑 지침

- **Package Spec이 아닌 Package Body만 래핑해야 합니다.**
- **래퍼는 구문 오류는 감지할 수 있지만 의미 오류는 감지할 수 없습니다.**
- **출력 파일은 편집하면 안됩니다. 원본 소스 코드를 유지 관리해 두었다가 필요에 따라 다시 래핑합니다.**
- **소스 코드의 중요 부분이 모두 난독 처리되도록 하려면 래핑된 파일을 배포하기 전 텍스트 편집기에서 확인하십시오.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

래핑 지침

지침은 다음과 같습니다.

- 패키지 또는 객체 유형을 래핑할 때는 Spec이 아닌 Body만 래핑합니다. 따라서 다른 개발자에게 패키지 구현을 공개하지 않으면서 패키지 사용에 필요한 정보를 제공할 수 있습니다.
- 입력 파일에 구문 오류가 있으면 PL/SQL 래퍼는 이 오류를 감지하고 보고합니다. 그러나 래퍼는 external 참조를 분석하지 않으므로 의미 오류는 감지할 수 없습니다. 예를 들어 amp 뷰 또는 테이블이 없어도 래퍼는 오류를 보고하지 않습니다.

```
CREATE PROCEDURE raise_salary (emp_id INTEGER, amount NUMBER)
AS
BEGIN
    UPDATE amp -- should be emp
        SET sal = sal + amount WHERE empno = emp_id;
END;
```

그러나 PL/SQL 컴파일러는 external 참조를 분석합니다. 따라서 래퍼 출력 파일(.p1b 파일)이 컴파일될 때 의미 오류가 보고됩니다.

- 출력 파일의 내용은 읽을 수 없으므로 출력 파일을 편집해서는 안됩니다. 래핑된 객체를 변경하려면 원래의 소스 코드를 수정한 다음 코드를 다시 래핑해야 합니다.

DBMS_DDL 패키지와 Wrap 유ти리티 비교

기능	DBMS_DDL	Wrap 유ти리티
코드 난독 처리	있음	있음
동적 난독 처리	있음	없음
여러 프로그램을 동시에 난독 처리	없음	있음

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DBMS_DDL과 Wrap 유ти리티 비교

Wrap 유ти리티와 DBMS_DDL 패키지는 사용 용도가 서로 다릅니다.

Wrap 유ти리티는 한 번 실행하여 여러 프로그램을 난독 처리할 수 있습니다. 실제로 전체 응용 프로그램을 래핑할 수 있습니다. 그러나 Wrap 유ти리티를 사용하여 런타임에 동적으로 생성된 코드를 난독 처리할 수는 없습니다. Wrap 유ти리티는 입력 SQL 파일을 처리하며 파일에 있는 다음과 같은 PL/SQL 단위만을 난독 처리합니다.

- Package Spec 및 Package Body
- 함수 및 프로시저
- Type Spec 및 Type Body

Wrap 유ти리티는 다음 항목에 포함된 PL/SQL 내용은 난독 처리하지 않습니다.

- 익명 블록
- 트리거
- PL/SQL이 아닌 코드

DBMS_DDL 패키지는 다른 프로그램 단위 내에서 동적으로 생성된 프로그램 단위를 난독 처리하기 위한 것입니다. DBMS_DDL 패키지 메소드는 한 번의 실행으로 여러 프로그램 단위를 난독 처리할 수 없습니다. 이러한 메소드를 실행할 때는 한 번에 하나의 CREATE OR REPLACE 문만 인수로 사용할 수 있습니다.

퀴즈

조건부 컴파일을 사용하면 소스 코드를 제거하지 않고 PL/SQL 응용 프로그램의 기능을 커스터마이즈할 수 있습니다.

- 1. 맞습니다.**
- 2. 틀립니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1

조건부 컴파일

조건부 컴파일을 사용하면 소스 코드를 제거하지 않고 PL/SQL 응용 프로그램의 기능 커스터마이즈할 수 있습니다.

최신 데이터베이스 버전의 새로운 기능을 활용하거나 새로운 기능을 비활성화하여 응용 프로그램을 이전 버전의 데이터베이스에 대해 실행

개발 환경에서 디버깅 또는 추적 기능을 활성화하고 응용 프로그램이 운영 사이트에서 실행 중인 동안 해당 기능 숨기기

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 조건부 컴파일 설명 및 사용
- 동적 난독 처리 및 Wrap 유ти리티를 사용하여 PL/SQL 소스 코드 숨기기

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 조건부 컴파일 및 PL/SQL 코드의 난독 처리, 즉 래핑에 대해 소개했습니다.

연습 11: 개요

이 연습에서는 다음 내용을 다룹니다.

- 조건부 컴파일을 사용하는 패키지 및 프로시저 생성
- 적절한 패키지를 사용하여 PL/SQL 단위의 포스트 처리된 소스 텍스트 검색
- 일부 PL/SQL 코드에 난독 처리 적용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 11: 개요

이 연습에서는 조건부 컴파일을 사용하는 패키지 및 프로시저를 생성합니다. 또한 적절한 패키지를 사용하여 PL/SQL 단위의 포스트 처리된 소스 텍스트를 검색합니다. 일부 PL/SQL 코드에 난독 처리도 적용합니다.

12

종속성 관리

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **프로시저 종속성 추적**
- **프로시저 및 함수에서 데이터베이스 객체를 변경한 결과 예측**
- **프로시저 종속성 관리**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 객체 종속성 및 유효하지 않은 객체의 명시적/암시적 재컴파일에 대해 소개합니다.

스키마 객체 종속성 개요

객체 유형	종속되거나 참조될 수 있음
Package Body	종속 전용
Package Spec	둘 다
시퀀스	참조 전용
서브 프로그램	둘 다
동의어	둘 다
테이블	둘 다
트리거	둘 다
유저 정의 객체	둘 다
유저 정의 컬렉션	둘 다
뷰	둘 다

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

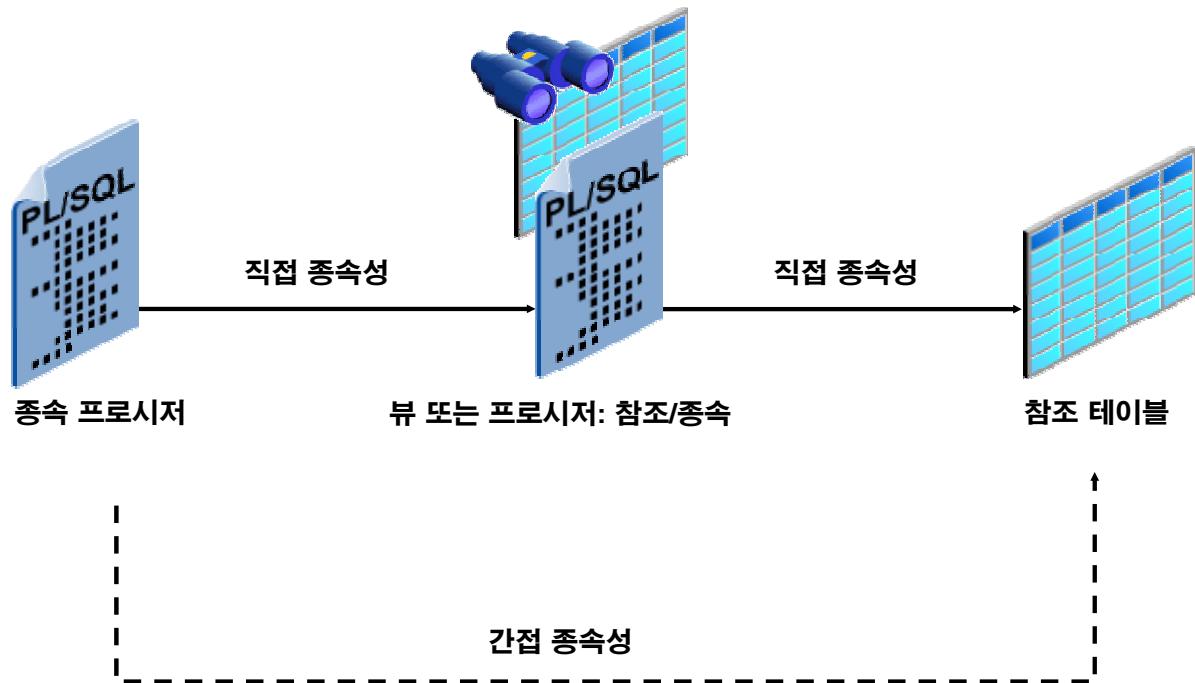
종속 객체 및 참조된 객체

일부 스키마 객체 유형은 해당 정의에 있는 다른 객체를 참조할 수 있습니다. 예를 들어 뷰는 테이블 또는 다른 뷰를 참조하는 query에 의해 정의되며 서브 프로그램 본문은 다른 객체를 참조하는 SQL 문을 포함할 수 있습니다. 객체 A의 정의가 객체 B를 참조하는 경우 A는 B에 대한 종속 객체이고 B는 A에 의해 참조된 객체입니다.

종속성 문제

- 참조된 객체의 정의를 변경하면 종속 객체가 계속 제대로 작동할 수도 있고 그렇지 않을 수도 있습니다. 예를 들어 테이블 정의가 변경되면 프로시저가 오류 없이 계속 작동할 수도 있고 그렇지 않을 수도 있습니다.
- Oracle 서버는 객체 간의 종속성을 자동으로 기록합니다. 종속성을 관리하기 위해 모든 스키마 객체는 상태(유효하거나 유효하지 않음)를 가지는데, 이 상태는 데이터 딕셔너리에 기록되어 USER_OBJECTS 데이터 딕셔너리 뷰에서 볼 수 있습니다.
- 스키마 객체의 상태가 VALID인 경우에는 객체를 이미 컴파일한 것이며 참조 시 바로 사용할 수 있습니다.
- 스키마 객체의 상태가 INVALID인 경우에는 스키마 객체를 컴파일한 후에야 사용할 수 있습니다.

종속성



ORACLE®

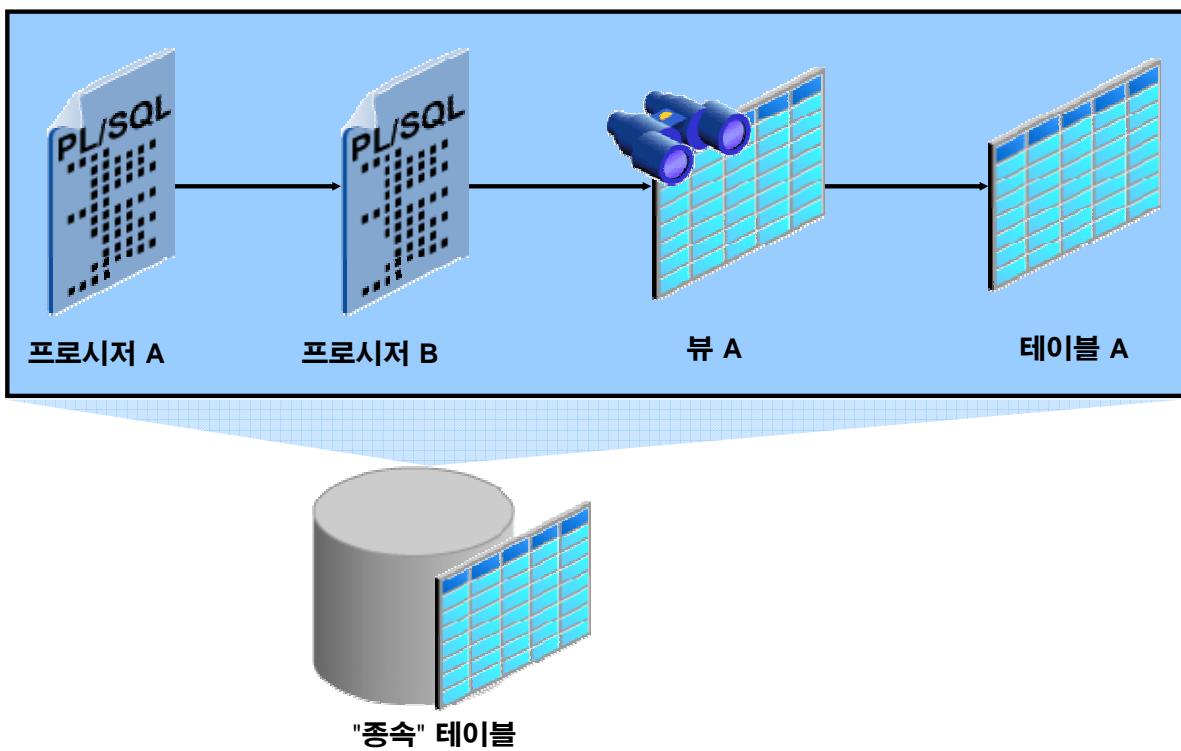
Copyright © 2009, Oracle. All rights reserved.

종속 객체 및 참조된 객체(계속)

프로시저나 함수는 중간 뷰, 프로시저, 함수 또는 패키지화된 프로시저나 함수를 통해 다음 객체를 직접 또는 간접적으로 참조할 수 있습니다.

- 테이블
- 뷰
- 시퀀스
- 프로시저
- 함수
- 패키지화된 프로시저 또는 함수

직접 로컬 종속성



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

로컬 종속성 관리

로컬 종속성의 경우 객체는 동일한 데이터베이스의 동일한 노드에 있습니다. Oracle 서버는 데이터베이스의 내부 "종속" 테이블을 사용하여 모든 로컬 종속성을 자동으로 관리합니다. 참조된 객체가 수정될 경우 종속 객체는 간혹 무효화됩니다. 다음 번에 무효화된 객체가 호출되면 Oracle 서버가 이를 자동으로 재컴파일합니다.

참조된 객체의 정의를 변경하는 경우 변경 유형에 따라 종속 객체가 오류 없이 계속 작동할 수도 있고 그렇지 않을 수도 있습니다. 예를 들어 테이블을 삭제하면 해당 테이블을 기반으로 하는 뷰를 사용할 수 없습니다.

Oracle Database 10g부터 CREATE OR REPLACE SYNONYM 명령은 이를 참조하는 종속 PL/SQL 프로그램 단위 및 뷰에 대한 무효화를 최소화하도록 개선되었습니다. 이 내용은 이 단원의 뒷부분에서 다룹니다.

Oracle Database 11g부터 종속성은 단위 내의 요소 레벨에서 추적됩니다. 이를 Fine-Grained Dependency라고 합니다. Fine-Grained Dependency는 이 단원의 뒷부분에서 다릅니다.

직접 객체 종속성 query: USER_DEPENDENCIES 뷰 사용

```
DESC User_dependencies
Name          Null    Type
-----        ----   -----
NAME          NOT NULL VARCHAR2(30)
TYPE          VARCHAR2(17)
REFERENCED_OWNER  VARCHAR2(30)
REFERENCED_NAME  VARCHAR2(64)
REFERENCED_TYPE  VARCHAR2(17)
REFERENCED_LINK_NAME  VARCHAR2(128)
SCHEMADID     NUMBER
DEPENDENCY_TYPE  VARCHAR2(4)

8 rows selected
```

```
SELECT name, type, referenced_name, referenced_type
FROM user_dependencies
WHERE referenced_name IN ('EMPLOYEES', 'EMP_VW');
```

NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE
1 QUERY_EMP	PROCEDURE	EMPLOYEES	TABLE
2 DML_CALL_SQL	FUNCTION	EMPLOYEES	TABLE
3 QUERY_CALL_SQL	FUNCTION	EMPLOYEES	TABLE
4 COMM_PKG	PACKAGE BODY	EMPLOYEES	TABLE
5 CURS_PKG	PACKAGE BODY	EMPLOYEES	TABLE
6 EMP_PKG	PACKAGE	EMPLOYEES	TABLE
7 EMP_PKG	PACKAGE BODY	EMPLOYEES	TABLE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

직접 객체 종속성 query: USER_DEPENDENCIES 뷰 사용

USER_DEPENDENCIES 데이터 딕셔너리 뷰에서 직접 종속성을 표시하여 수동으로 재컴파일될 데이터베이스 객체를 확인할 수 있습니다.

ALL_DEPENDENCIES 및 DBA_DEPENDENCIES 뷰에는 객체의 소유자를 참조하는 OWNER 열이 추가로 포함되어 있습니다.

USER_DEPENDENCIES 데이터 딕셔너리 뷰 열

USER_DEPENDENCIES 데이터 딕셔너리 뷰의 열은 다음과 같습니다.

- NAME: 종속 객체의 이름
- TYPE: 종속 객체의 유형(PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER 또는 VIEW)
- REFERENCED_OWNER: 참조된 객체의 스키마
- REFERENCED_NAME: 참조된 객체의 이름
- REFERENCED_TYPE: 참조된 객체의 유형
- REFERENCED_LINK_NAME: 참조된 객체에 액세스하는 데 사용되는 데이터베이스 링크

객체 상태 query

모든 데이터베이스 객체에는 다음 상태 값 중 하나가 지정되어 있습니다.

상태	설명
VALID	데이터 딕셔너리에 있는 현재 정의를 사용하여 객체가 성공적으로 컴파일되었습니다.
COMPILED WITH ERRORS	객체를 컴파일하려는 가장 최근의 시도로 인해 오류가 발생했습니다.
INVALID	참조하는 객체가 변경되었기 때문에 이 객체가 무효로 표시됩니다. (종속 객체만 무효일 수 있습니다.)
UNAUTHORIZED	참조된 객체의 액세스 권한이 취소되었습니다. (종속 객체만 권한이 취소될 수 있습니다.)

ORACLE®

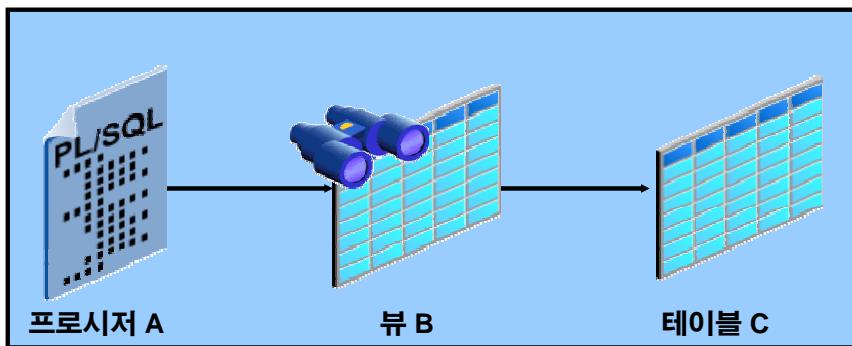
Copyright © 2009, Oracle. All rights reserved.

객체 상태 query

모든 데이터베이스 객체에는 슬라이드의 표에 나타난 상태 값 중 하나가 지정되어 있습니다.

참고: USER_OBJECTS, ALL_OBJECTS 및 DBA_OBJECTS 정적 데이터 딕셔너리 뷰는 Invalid 및 Unauthorized 컴파일 오류를 구분하지 않는 대신 이들 오류를 모두 INVALID로 설명합니다.

종속 객체 무효화



- **프로시저 A는 뷰 B에 직접적으로 종속되며 뷰 B는 테이블 C에 직접적으로 종속됩니다. 또한 프로시저 A는 테이블 C에 간접적으로 종속됩니다.**
- **직접 종속 항목은 영향을 주는 참조된 객체가 변경되는 경우에만 무효화됩니다.**
- **간접 종속 항목은 영향을 주지 않는 참조 객체의 변경으로 인해 무효화될 수 있습니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

종속 객체 무효화

객체 A가 객체 B에 종속되어 있고, 객체 B가 객체 C에 종속되어 있고, A가 B에 대한 직접 종속이고, B가 C에 대한 직접 종속인 경우 A는 C에 간접적으로 종속됩니다.

Oracle Database 11g에서 직접 종속 항목은 영향을 주는 참조된 객체가 변경(참조된 객체의 서명 변경)될 경우에만 무효화됩니다.

간접 종속 항목은 영향을 주지 않는 참조 객체의 변경으로 인해 무효화될 수 있습니다. 테이블 C의 변경으로 인해 뷰 B가 무효화될 경우에는 프로시저 A와 뷰 B의 다른 직접 및 간접 종속도 모두 무효화합니다. 이를 연쇄적인 무효화라고 합니다.

뷰의 기반이 되는 기본 테이블의 구조가 수정된다고 가정하겠습니다. SQL*Plus DESCRIBE 명령을 사용하여 뷰를 설명하면 이 객체가 설명에 유효하지 않다는 오류 메시지가 나타납니다. 이것은 명령이 SQL 명령이 아니기 때문입니다. 이 시점에 뷰의 기본 테이블 구조가 변경되었기 때문에 뷰가 유효하지 않습니다. 지금 뷰를 query하면 뷰가 자동으로 재컴파일되며 재컴파일이 성공할 경우 그 결과를 볼 수 있습니다.

일부 종속 항목을 무효화하는 스키마 객체 변경: 예제

```
CREATE VIEW commissioned AS
SELECT first_name, last_name, commission_pct FROM employees
WHERE commission_pct > 0.00;
```

```
CREATE VIEW six_figure_salary AS
SELECT * FROM employees
WHERE salary >= 100000;
```

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

Results:

OBJECT_NAME	STATUS
1 EMP_DETAILS	VALID
2 COMMISSIONED	VALID
3 SIX FIGURE SALARY	VALID
4 EMP DETAILS VIEW	VALID

Copyright © 2009, Oracle. All rights reserved.

일부 종속 항목을 무효화하는 스키마 객체 변경: 예제

슬라이드의 예제에서는 일부 종속 항목만 무효화하고 다른 종속 항목은 무효화하지 않는 스키마 객체 변경 사항의 예를 보여줍니다. 새로 생성된 두 뷰는 HR 스키마의 EMPLOYEES 테이블을 기반으로 합니다. 새로 생성된 뷰의 상태는 VALID입니다.

일부 종속 항목을 무효화하는 스키마 객체 변경: 예제

```
ALTER TABLE employees MODIFY email VARCHAR2(50);

SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW';
```

Results:

OBJECT_NAME	STATUS
1 EMP_DETAILS	VALID
2 COMMISSIONED	VALID
3 SIX FIGURE_SALARY	INVALID
4 EMP_DETAILS_VIEW	VALID

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

일부 종속 항목을 무효화하는 스키마 객체 변경: 예제(계속)

이번에는 EMPLOYEES 테이블의 EMAIL 열을 25에서 50으로 늘려야 하기 때문에 위 슬라이드에 표시된 것과 같이 테이블을 변경한다고 가정하겠습니다.

COMMISSIONED 뷰는 선택 리스트에 EMAIL 열을 포함하지 않기 때문에 무효화되지 않습니다. 그러나 SIXFIGURES 뷰는 테이블에 있는 모든 열이 선택되기 때문에 무효화됩니다.

직접 및 간접 종속성 표시

- 직접 및 간접 종속성을 표시할 수 있게 해주는 객체를 생성하는 `utldtree.sql` 스크립트를 실행합니다.

```
@/home/oracle/labs/plpu/labs/utldtree.sql
```

- `DEPTREE_FILL` 프로시저를 실행합니다.

```
EXECUTE deptree_fill('TABLE', 'ORA61', 'EMPLOYEES')
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

오라클에서 제공하는 뷰를 사용하여 직접 및 간접 종속성 표시

`DEPTREE` 및 `IDEPTREE`라는 추가 유저 뷰에서 직접 및 간접 종속성을 표시합니다. 이러한 뷰는 오라클에서 제공합니다.

예제

- `utldtree.sql` 스크립트가 실행되었는지 확인합니다. 이 스크립트는 `$ORACLE_HOME/labs/plpu/labs` 폴더에 있습니다. 다음과 같이 스크립트를 실행할 수 있습니다.

```
@?/labs/plpu/labs/utldtree.sql
```

참고: 이 과정에서는 이 스크립트가 클래스 파일의 `labs` 폴더에 있습니다. 위의 코드 예제에는 수강생 계정 `ORA61`이 사용됩니다.

- `DEPTREE_FILL` 프로시저를 호출하여 참조된 특정 객체에 대한 정보로 `DEPTREE_TEMPTAB` 테이블을 채웁니다. 이 프로시저에 대한 파라미터는 다음 세 가지입니다.

<code>object_type</code>	참조된 객체의 유형
<code>object_owner</code>	참조된 객체의 스키마
<code>object_name</code>	참조된 객체의 이름

DEPTREE 뷰를 사용하여 종속성 표시

```
SELECT    nested_level, type, name
FROM      deptree
ORDER BY  seq#;
```

NESTED_LEVEL	TYPE	NAME
0	TABLE	EMPLOYEES
1	VIEW	EMP_DETAILS_VIEW
1	trigger	SECURE_EMPLOYEES
1	trigger	UPDATE_JOB_HISTORY
1	PROCEDURE	RAISE_SALARY
2	PROCEDURE	PROCESS_EMPLOYEES
1	PROCEDURE	QUERY_EMP
1	PROCEDURE	PROCESS_EMPLOYEES
1	FUNCTION	DML_CALL_SQL
1	FUNCTION	QUERY_CALL_SQL
1	PACKAGE BODY	COMM_PKG
1	PACKAGE BODY	CURS_PKG
1	PACKAGE	EMP_PKG
2	PACKAGE BODY	EMP_PKG
1	PACKAGE BODY	EMP_PKG
1	PROCEDURE	SAL_STATUS

...

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DEPTREE 뷰를 사용하여 종속성 표시

DEPTREE 뷰를 query하여 모든 종속 객체를 테이블 형식으로 표시합니다. 다음과 같이 DEPENDENCIES라는 단일 열로 구성된 IDEPTREE 뷰를 query하여 동일한 정보를 들여써서 표시합니다.

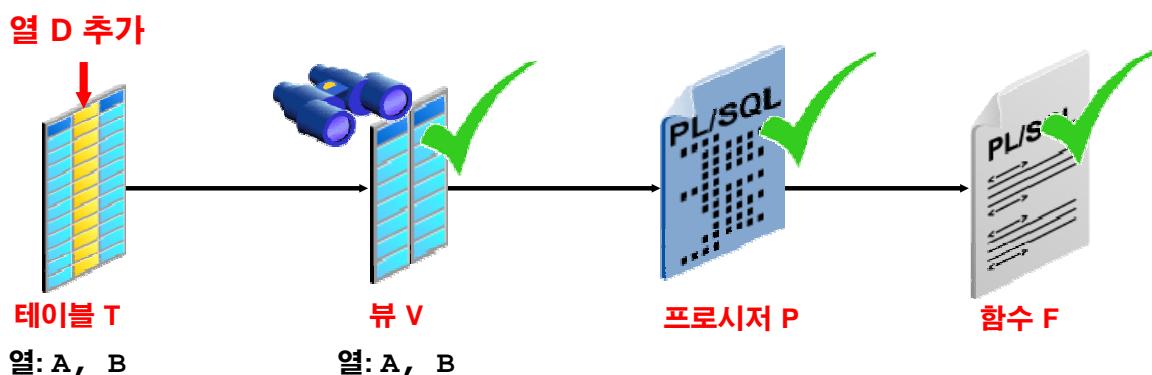
```
SELECT *
FROM  ideptree;
DEPENDENCIES
```

TRIGGER ORA61.SECURE_EMPLOYEES
PROCEDURE ORA61.UPDATE_SALARY
TRIGGER ORA61.AUDIT_EMP_VALUES
PROCEDURE ORA61.EMP_LIST
PROCEDURE ORA61.PROCESS_EMPLOYEES
PACKAGE BODY ORA61.CURS_PKG
PACKAGE BODY ORA61.EMP_PKG
VIEW ORA61.EMP_DETAILS
TRIGGER ORA61.NEW_EMP_DEPT
TRIGGER ORA61.UPDATE_JOB_HISTORY
PACKAGE BODY ORA61.EMP_PKG
FUNCTION ORA61.EMP_HIRE_DATE
PROCEDURE ORA61.SAL_STATUS
TRIGGER ORA61.DERIVE_COMMISSION_PCT
TRIGGER ORA61.CHECK_SALARY
PROCEDURE ORA61.PROCESS_EMPLOYEES
PROCEDURE ORA61.QUERY_EMP
TRIGGER ORA61.RESTRICT_SALARY
VIEW ORA61.COMMISSIONED

...

Oracle Database 11g의 보다 정밀한 종속성 메타 데이터

- 11g 이전에는 열 D를 테이블 T에 추가하면 종속 객체가 무효화되었습니다.
- Oracle Database 11g는 추가적인 Fine-Grained Dependency 관리를 기록합니다.
 - 열 D를 테이블 T에 추가해도 뷰 V에는 영향을 주지 않고 종속 객체가 무효화되지 않습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Fine-Grained Dependency

Oracle Database 11g부터는 보다 정밀한 종속성 메타 데이터를 설명하는 레코드에 액세스할 수 있습니다. 이를 Fine-Grained Dependency라고 하며, 이를 통해 논리적 요구 사항 없이 종속 객체가 무효화되지 않을 때를 확인할 수 있습니다.

초기 Oracle Database 버전은 종속성 메타 데이터(예: PL/SQL 단위 P가 PL/SQL 단위 F에 종속되거나 뷰 V가 테이블 T에 종속됨)를 전체 객체의 정밀도와 함께 기록합니다. 즉, 논리적 요구 사항 없이도 간혹 종속 객체가 무효화됩니다. 예를 들어 뷰 V가 테이블 T의 열 A 및 B에만 종속되어 있는데 열 D가 테이블 T에 추가될 경우 뷰 V의 유효성은 논리적으로 영향을 받지 않습니다. 그러나 Oracle Database Release 11.1 이전에는 열 D를 테이블 T에 추가하면 뷰 V가 무효화됩니다. Oracle Database Release 11.1에서는 열 D를 테이블 T에 추가해도 뷰 V가 무효화되지 않습니다. 마찬가지로 프로시저 P가 패키지 내의 요소 E1 및 E2에만 종속되어 있으면 요소 E99를 패키지에 추가해도 프로시저 P가 무효화되지 않습니다.

종속 관계에 있는 상위 객체가 변경될 때 종속 객체가 무효화되는 것을 줄임으로써 온라인 응용 프로그램 업그레이드 시에는 물론 개발 환경에서도 응용 프로그램의 가용성이 향상됩니다.

이 항목에 대한 자세한 내용은 *11g: Infrastructure Grid – High Availability PI eStudy*를 참조하십시오.

Fine-Grained Dependency 관리

- Oracle Database 11g에서는 이제 종속성이 단위 내의 요소 레벨에서 추적됩니다.
- 요소 기반 종속성 추적은 다음을 다룹니다.
 - 기본 테이블에서 단일 테이블 뷰의 종속성
 - 다음에서 PL/SQL 프로그램 단위(Package Spec, Package Body 또는 서브 프로그램)의 종속성
 - 기타 PL/SQL 프로그램 단위
 - 테이블
 - 뷰



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Fine-Grained Dependency 관리: 예제 1

```
CREATE TABLE t2 (col_a NUMBER, col_b NUMBER, col_c NUMBER);
CREATE VIEW v AS SELECT col_a, col_b FROM t2;
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:					
	NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	VIEW	T2	TABLE	VALID

```
ALTER TABLE t2 ADD (col_d VARCHAR2(20));
```

```
SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:					
	NAME	TYPE	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	VIEW	T2	TABLE	VALID

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Fine-Grained Dependency 관리: 예제 1

기본 테이블에서 단일 테이블 뷰의 종속성 예제

슬라이드의 첫번째 예제에서는 테이블 T2가 COL_A, COL_B 및 COL_C의 세 가지 열로 생성됩니다.

테이블 T2의 COL_A 및 COL_B 열을 기반으로 V라는 뷰가 생성됩니다. 덕셔너리 뷰가 query되고 뷰 V는 테이블 T에 종속되며 유효 상태입니다.

세번째 예제에서는 테이블 T2가 변경되고 COL_D라는 새 열이 추가됩니다. 덕셔너리 뷰는 뷰 V를 계속 종속 상태로 보고하는데, 이는 요소 기반 종속성 추적이 열 COL_A 및 COL_B가 수정되지 않은 것을 확인함에 따라 뷰를 무효화할 필요가 없기 때문입니다.

Fine-Grained Dependency 관리: 예제 1

```
ALTER TABLE t2 MODIFY (col_a VARCHAR2(20));
SELECT ud.name, ud.referenced_name, ud.referenced_type,
       uo.status
  FROM user_dependencies ud, user_objects uo
 WHERE ud.name = uo.object_name AND ud.name = 'V';
```

Results:

	NAME	REFERENCED_NAME	REFERENCED_TYPE	STATUS
1	V	T2	TABLE	INVALID

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Fine-Grained Dependency 관리: 예제 1(계속)

슬라이드 예제에서는 뷰가 종속된 테이블에서 요소(COL_A)가 수정되었기 때문에 뷰가 무효화됩니다.

Fine-Grained Dependency 관리: 예제 2

```

CREATE PACKAGE pkg IS
  PROCEDURE proc_1;
END pkg;
/
CREATE OR REPLACE PROCEDURE p IS
BEGIN
  pkg.proc_1();
END p;
/
CREATE OR REPLACE PACKAGE pkg
IS
  PROCEDURE proc_1;
  PROCEDURE unheard_of;
END pkg;
/

```

```

PACKAGE pkg Compiled.
PROCEDURE p Compiled.
PACKAGE pkg Compiled.

```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Fine-Grained Dependency 관리: 예제 2

슬라이드의 예제에서는 프로시저 PROC_1이 선언되는 PKG라는 패키지를 생성합니다.

P라는 프로시저가 PKG.PROC_1을 호출합니다.

PKG 패키지의 정의가 수정되고 패키지 선언에 또 다른 서브 루틴이 추가됩니다.

USER_OBJECTS 덕셔너리 뷰에서 P 패키지의 상태를 query하면 아래와 같이 계속 유효한 것으로 표시되는데, 이는 PKG의 정의에 추가된 요소가 프로시저 P를 통해 참조되지 않기 때문입니다.

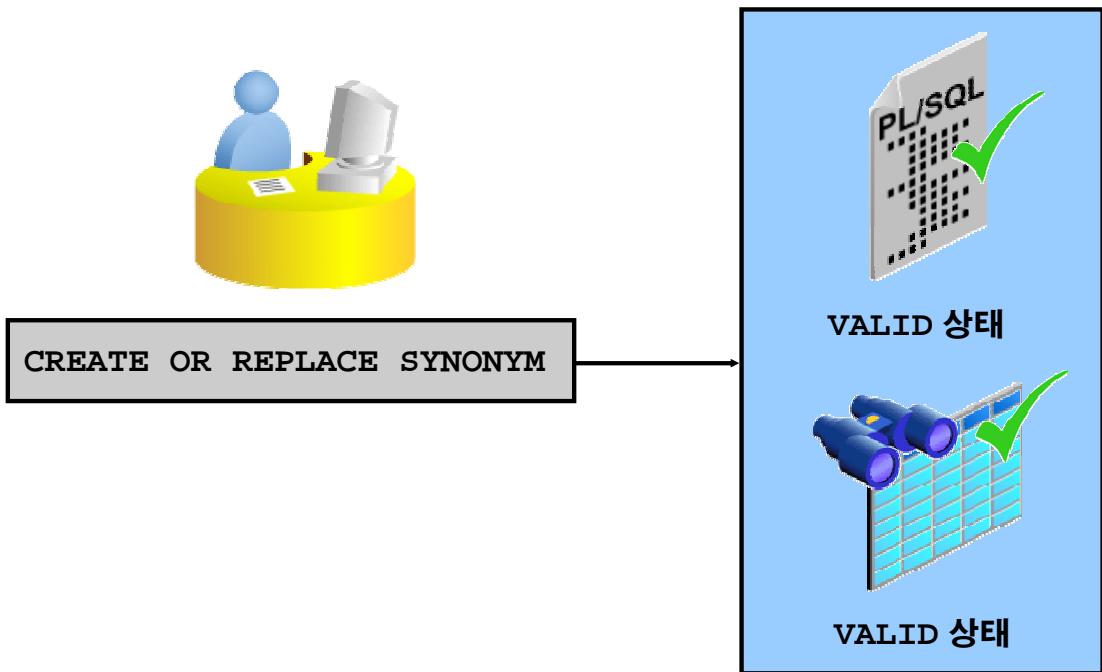
```

SELECT status FROM user_objects
WHERE object_name = 'P';

```

Results:	
	STATUS
1	VALID

동의어 종속성의 변화



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

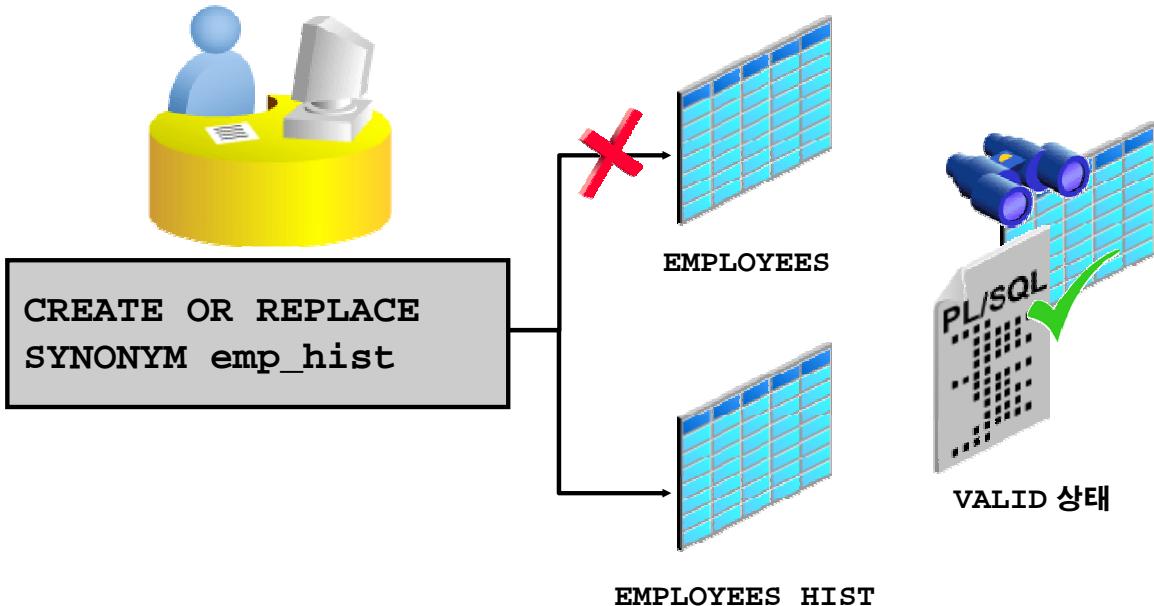
동의어 종속성의 변화

Oracle Database는 코드 업그레이드 및 스키마 병합으로 인해 발생하는 다운타임을 최소화합니다. 열, 권한, Partition 등에서 특정 조건이 충족될 경우 테이블 또는 객체 유형은 동등한 것으로 간주되고 더 이상 종속 객체가 무효화되지 않습니다.

Oracle Database 10g에서 CREATE OR REPLACE SYNONYM 명령은 이를 참조하는 종속 PL/SQL 프로그램 단위 및 뷰에 대한 무효화를 최소화하도록 개선되었습니다. 따라서 동의어를 재정의한 후에나 실행하는 동안 프로그램 단위를 재컴파일하는 데 시간을 낭비하지 않아도 됩니다. 이 기능을 활성화하기 위해 파라미터를 설정하거나 특별한 명령을 실행할 필요가 없으며, 무효화는 자동으로 최소화됩니다.

참고: 이 향상된 기능은 테이블을 가리키는 동의어에만 적용됩니다.

유효한 PL/SQL 프로그램 단위 및 뷰 유지 관리



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

유효한 PL/SQL 프로그램 단위 유지 관리

Oracle Database 10g Release 2부터는 동의어의 정의를 변경해도 다음과 같은 조건에서는 종속 PL/SQL 프로그램 단위가 무효화되지 않습니다.

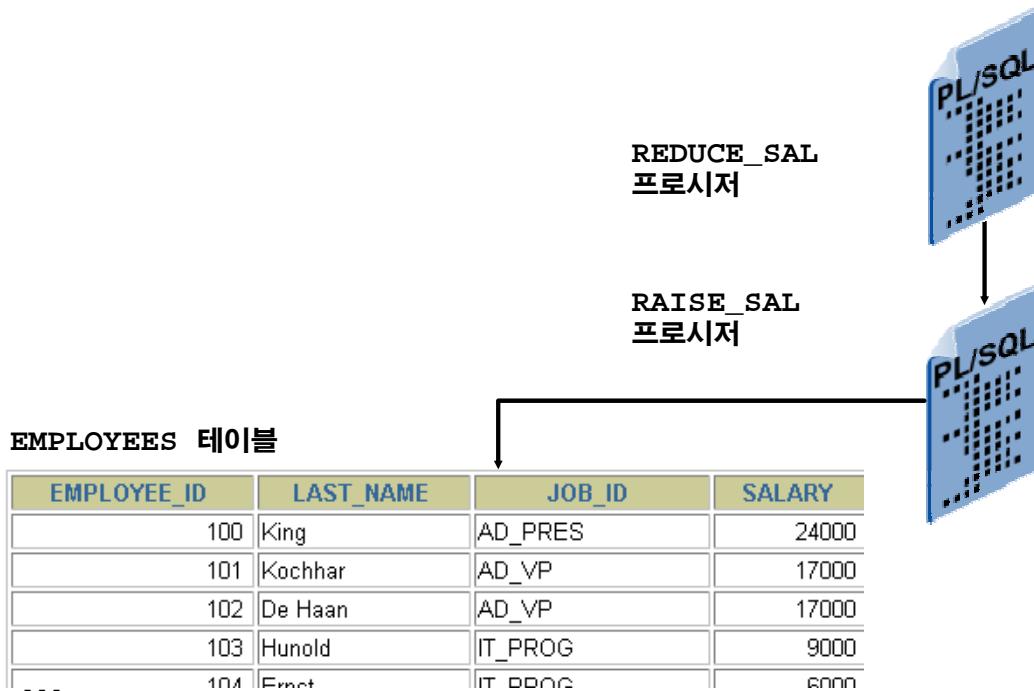
- 테이블의 열 순서, 열 이름 및 열 데이터 유형이 동일합니다.
- 새로 참조된 테이블 및 해당 열의 권한이 원래 테이블의 권한보다 높습니다. 이러한 권한이 룰을 통해서만 파생된 것이어서는 안됩니다.
- Partition 및 Subpartition의 이름과 유형이 동일합니다.
- 테이블의 구성 유형이 동일합니다.
- 객체 유형 열이 동일한 유형입니다.

유효한 뷰 유지 관리

종속 PL/SQL 프로그램 단위와 마찬가지로 동의어의 정의를 변경해도 위의 단락에 나열된 조건에서는 종속 뷰가 무효화되지 않습니다. 또한 다음 조건은 동의어를 재정의할 때 종속 뷰의 VALID 상태를 유지하기 위해 필요한 조건일 뿐이며 종속 PL/SQL 프로그램 단위의 경우에는 필요하지 않습니다.

- Primary Key 및 고유 인덱스에 정의된 열 및 열 순서, NOT NULL 제약 조건, Primary Key 및 고유 제약 조건이 동일해야 합니다.
- 종속 뷰에는 참조 제약 조건이 있을 수 없습니다.

로컬 종속성에 대한 또 다른 시나리오



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

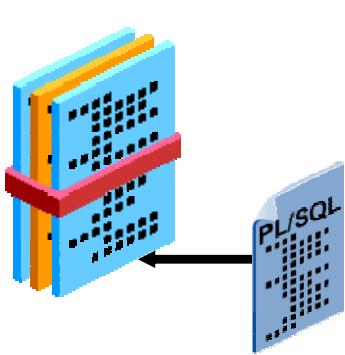
로컬 종속성에 대한 또 다른 시나리오

슬라이드의 예제에서는 프로시저 정의 변경이 종속 프로시저의 재컴파일에 미치는 영향을 보여줍니다. RAISE_SAL 프로시저가 EMPLOYEES 테이블을 직접 생성하고 REDUCE_SAL 프로시저가 RAISE_SAL을 통해 EMPLOYEES 테이블을 간접적으로 생성한다고 가정하겠습니다.

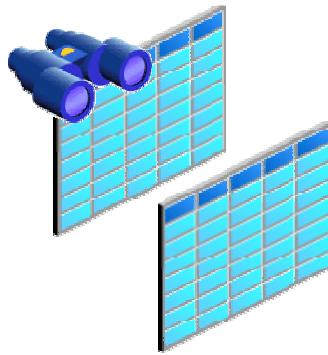
- RAISE_SAL 프로시저의 내부 논리가 수정된 경우 RAISE_SAL이 성공적으로 컴파일되면 REDUCE_SAL이 재컴파일됩니다.
- RAISE_SAL 프로시저의 형식 파라미터가 제거되면 REDUCE_SAL이 재컴파일되지 않습니다.

무효화를 줄이기 위한 지침

종속 객체의 무효화를 줄이려면



새 항목을 패키지의 끝에 추가



뷰를 통해 각 테이블 참조

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

무효화를 줄이기 위한 지침

새 항목을 패키지 끝에 추가

새 항목을 패키지에 추가할 때 패키지의 맨 끝에 추가하십시오. 그러면 기존 최상위 패키지 항목의 슬롯 번호와 시작점 번호가 그대로 유지되므로 무효화되는 것을 방지할 수 있습니다. 예를 들어 다음 패키지를 살펴 보십시오.

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
```

항목을 pkg1의 끝에 추가하면 get_var를 참조하는 종속 항목이 무효화되지 않습니다. 항목을 get_var 함수와 set_var 프로시저 사이에 삽입하면 set_var 함수를 참조하는 종속 항목이 무효화됩니다.

뷰를 통해 각 테이블 참조

뷰를 사용하여 테이블을 간접적으로 참조합니다. 그러면 다음이 가능합니다.

- 종속 뷰 또는 종속 PL/SQL 객체를 무효화하지 않고 테이블에 열을 추가할 수 있습니다.
- 종속 객체를 무효화하지 않고 뷰에서 참조하지 않는 열을 수정하고 삭제할 수 있습니다.

객체 재검증

- 참조될 때 유효하지 않은 객체는 사용되기 전에 검증을 받아야 합니다.
- 검증은 객체가 참조될 때 자동으로 수행되므로 유저가 따로 개입할 필요가 없습니다.
- 객체가 유효화되지 않으면 상태가 **COMPILED WITH ERRORS**, **UNAUTHORIZED** 또는 **INVALID**가 됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

객체 재검증

컴파일러는 오류와 함께 컴파일된 객체를 자동으로 재검증할 수 없습니다. 컴파일러가 객체를 재컴파일할 때 오류 없이 재컴파일되면 재검증되며, 그렇지 않으면 무효 상태가 계속 유지됩니다.

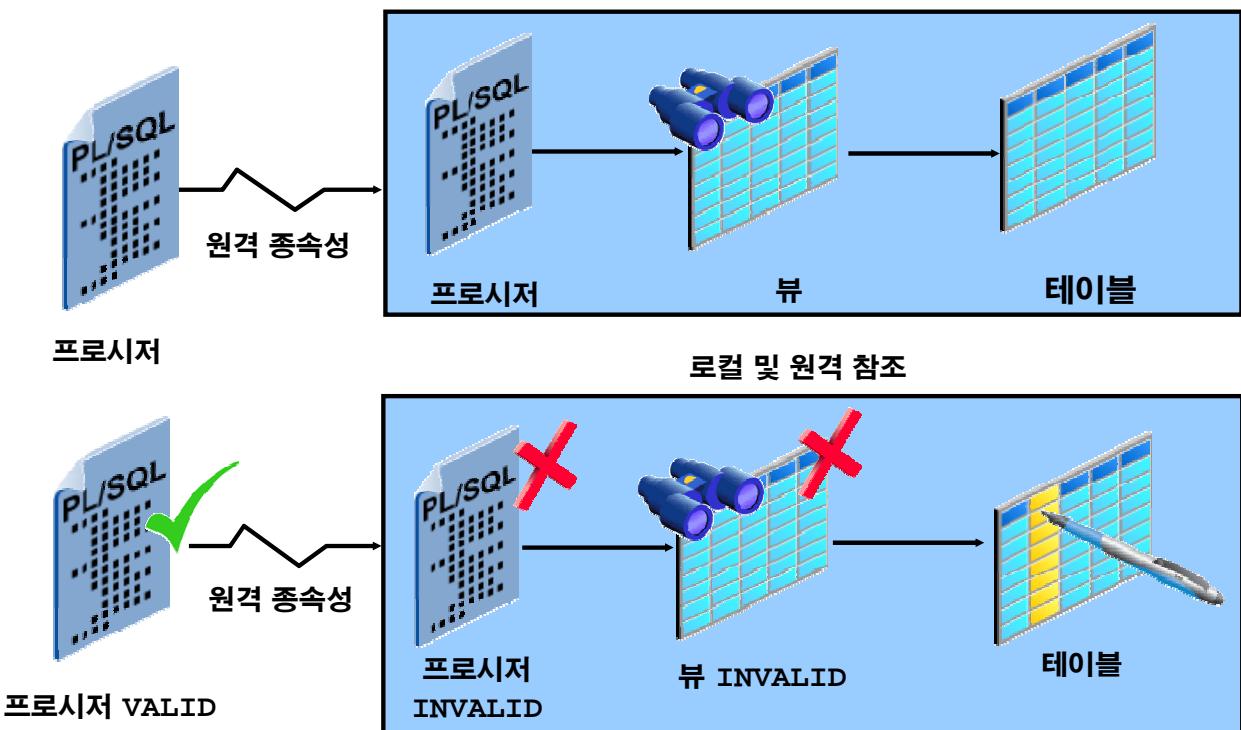
컴파일러는 권한 없는 객체가 모든 참조된 객체에 액세스할 수 있는지를 검사합니다. 그럴 경우 컴파일러는 권한 없는 객체를 재컴파일하지 않고 재검증합니다. 그렇지 않을 경우에는 컴파일러가 적절한 오류 메시지를 내보냅니다.

SQL 컴파일러는 유효하지 않은 객체를 재컴파일합니다. 객체가 오류 없이 재컴파일되면 재검증됩니다. 그렇지 않은 경우에는 무효 상태를 유지합니다.

유효하지 않은 PL/SQL 프로그램 단위(프로시저, 함수 또는 패키지)의 경우 PL/SQL 컴파일러는 참조된 객체가 유효하지 않은 객체에 영향을 주는 방법으로 변경되었는지 확인합니다.

- 그럴 경우 컴파일러는 유효하지 않은 객체를 재컴파일합니다. 객체가 오류 없이 재컴파일되면 재검증됩니다. 그렇지 않은 경우에는 무효 상태를 유지합니다. 그렇지 않은 경우 컴파일러는 유효하지 않은 객체를 재컴파일하지 않고 재검증합니다.
- 그렇지 않은 경우 컴파일러는 유효하지 않은 객체를 재컴파일하지 않고 재검증합니다. 연쇄 무효화로 인해 무효화된 객체는 일반적으로 신속하게 재검증됩니다.

원격 종속성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

원격 종속성

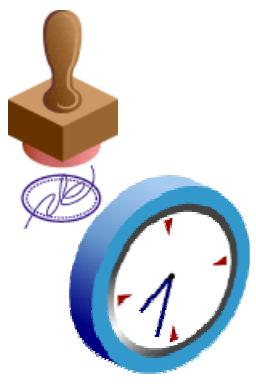
원격 종속성의 경우 객체는 별도의 노드에 있습니다. Oracle 서버는 로컬 프로시저와 원격 프로시저 간의 종속성(함수, 패키지, 트리거 포함)만 관리하고, 원격 스키마 객체 간의 종속성은 관리하지 않습니다. 로컬 내장 프로시저와 모든 종속 객체는 무효화되지만 이러한 객체는 처음으로 호출될 때 자동으로 재컴파일되지는 않습니다.

종속 객체 재컴파일: 로컬 및 원격

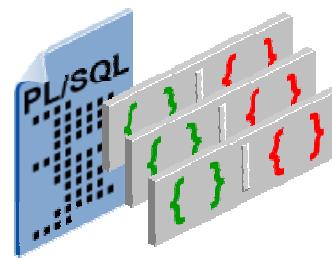
- USER_OBJECTS 뷰 내에서 프로시저의 상태를 점검하여 종속 원격 프로시저의 명시적 재컴파일과 종속 로컬 프로시저의 암시적 재컴파일이 성공했는지 확인하십시오.
- 종속 로컬 프로시저의 자동 암시적 재컴파일이 실패하면 프로시저가 유효하지 않은 상태로 남아 있고 Oracle 서버가 런타임 오류를 표시합니다. 따라서 작업 실행을 방해하지 않으려면 자동 방식에 의존하지 말고 로컬 종속 객체를 수동으로 재컴파일할 것을 적극 권장합니다.

원격 종속성의 개념

원격 종속성은 유저가 선택한 모드에 의해 결정됩니다.



TIMESTAMP 검사



SIGNATURE 검사

ORACLE

Copyright © 2009, Oracle. All rights reserved.

원격 종속성의 개념

TIMESTAMP 검사

각 PL/SQL 프로그램 단위에는 생성되거나 재컴파일될 때 설정된 시간 기록이 있습니다. PL/SQL 프로그램 단위나 관련 스키마 객체를 변경할 때마다 모든 종속 프로그램 단위는 유효하지 않은 상태로 표시되므로 실행하기 전에 재컴파일해야 합니다. 로컬 프로시저 본문에 있는 명령문이 원격 프로시저를 호출할 때 실제 시간 기록 비교가 수행됩니다.

SIGNATURE 검사

PL/SQL 프로그램 단위마다 시간 기록과 서명이 모두 기록됩니다. PL/SQL 생성자의 서명에는 다음에 대한 정보가 포함되어 있습니다.

- 생성자(프로시저, 함수 또는 패키지) 이름
- 생성자의 기본 파라미터 유형
- 파라미터 모드(IN, OUT 또는 IN OUT)
- 파라미터 수

호출 프로그램 단위에 기록된 시간 기록은 호출된 원격 프로그램 단위의 현재 시간 기록과 비교됩니다. 시간 기록이 일치하면 호출이 진행됩니다. 일치하지 않으면 RPC(원격 프로시저 호출) 계층에서 간단한 서명 비교가 수행되어 호출의 안전 여부를 확인합니다. 서명이 서로 호환되도록 변경되었다면 실행이 계속되지만, 그렇지 않을 경우 오류가 반환됩니다.

REMOTE_DEPENDENCIES_MODE 파라미터 설정

- **init.ora** 파라미터로 설정:

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP |  
SIGNATURE
```

- 시스템 레벨의 경우:

```
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

- 세션 레벨의 경우:

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE =  
TIMESTAMP | SIGNATURE
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REMOTE_DEPENDENCIES_MODE 파라미터

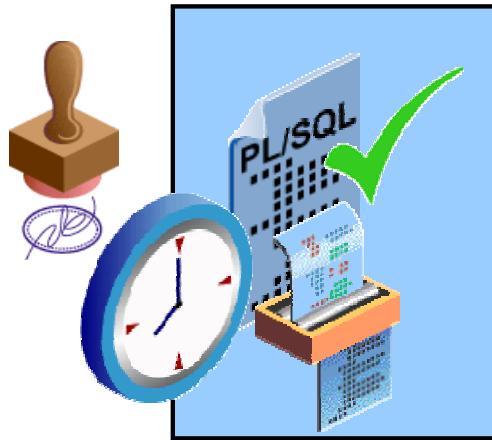
REMOTE_DEPENDENCIES_MODE 설정

value	TIMESTAMP
	SIGNATURE

위 슬라이드에 설명된 세 가지 방법 중 하나를 사용하여 REMOTE_DEPENDENCIES_MODE 파라미터 값을 지정하십시오.

참고: 호출 사이트에 따라 종속성 모델이 결정됩니다.

오전 8:00시에 원격 프로시저 B 컴파일



원격 프로시저 B:
오전 8:00에
컴파일 및 VALID

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

원격 프로시저를 참조하는 로컬 프로시저

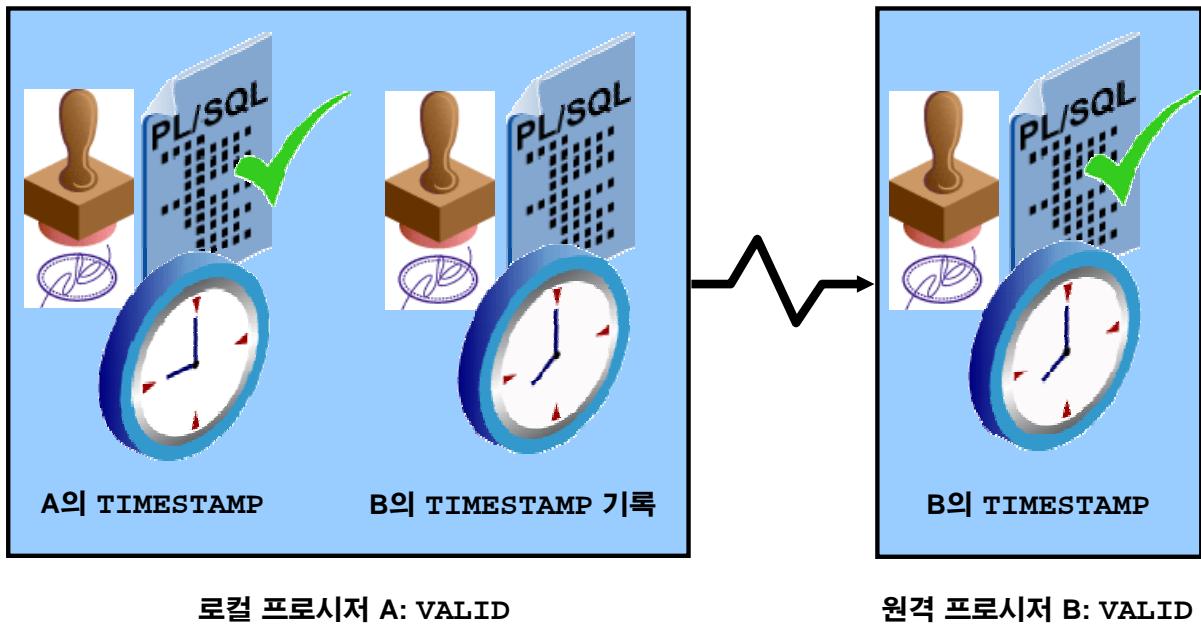
로컬 프로시저가 컴파일된 후 원격 프로시저가 재컴파일될 경우 원격 프로시저를 참조하는 로컬 프로시저는 Oracle 서버에 의해 무효화됩니다.

자동 원격 종속성 방식

프로시저가 컴파일되면 Oracle 서버는 프로시저의 P code 내에 해당 컴파일의 시간 기록을 기록합니다.

위 슬라이드에서 원격 프로시저 B가 오전 8:00에 성공적으로 컴파일되면 이 시간이 해당 프로시저의 시간 기록으로 기록됩니다.

오전 9:00에 로컬 프로시저 A 컴파일



Copyright © 2009, Oracle. All rights reserved.

ORACLE

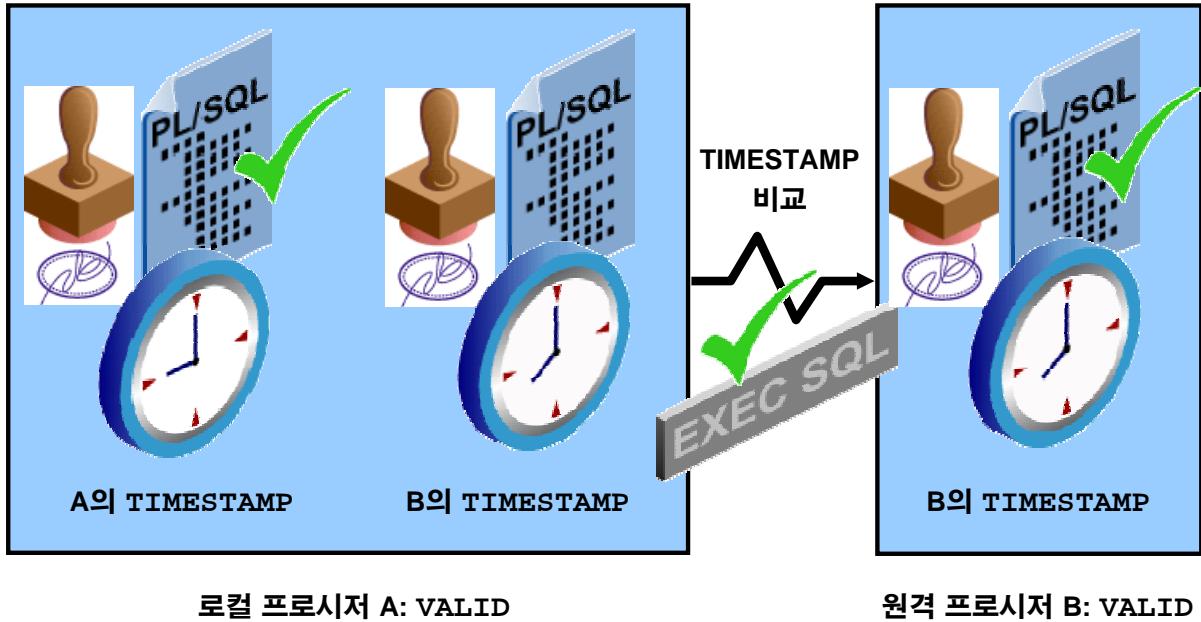
원격 프로시저를 참조하는 로컬 프로시저(계속)

자동 원격 종속성 방식(계속)

원격 프로시저를 참조하는 로컬 프로시저가 컴파일될 때 Oracle 서버는 로컬 프로시저의 P code에 원격 프로시저의 시간 기록도 기록합니다.

슬라이드에서 로컬 프로시저 A(원격 프로시저 B에 종속됨)는 오전 9:00에 컴파일됩니다. 프로시저 A와 원격 프로시저 B의 시간 기록이 프로시저 A의 P code에 기록됩니다.

프로시저 A 실행



자동 원격 종속성

런타임에 로컬 프로시저가 호출될 때 Oracle 서버는 참조된 원격 프로시저의 두 시간 기록을 비교합니다.

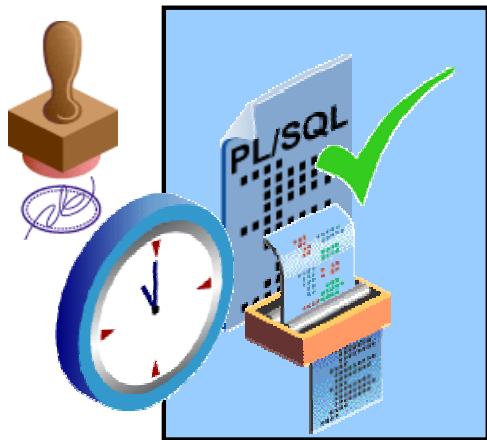
시간 기록이 같으면(원격 프로시저가 재컴파일되지 않았음을 나타냄) Oracle 서버는 로컬 프로시저를 실행합니다.

슬라이드 예제에서 원격 프로시저 B의 P code에 기록된 시간 기록이 로컬 프로시저 A에 기록된 시간 기록과 동일합니다. 따라서 로컬 프로시저 A는 유효합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

오전 11:00에 원격 프로시저 B 재컴파일



원격 프로시저 B:
오전 11:00에
재컴파일 및 VALID

ORACLE®

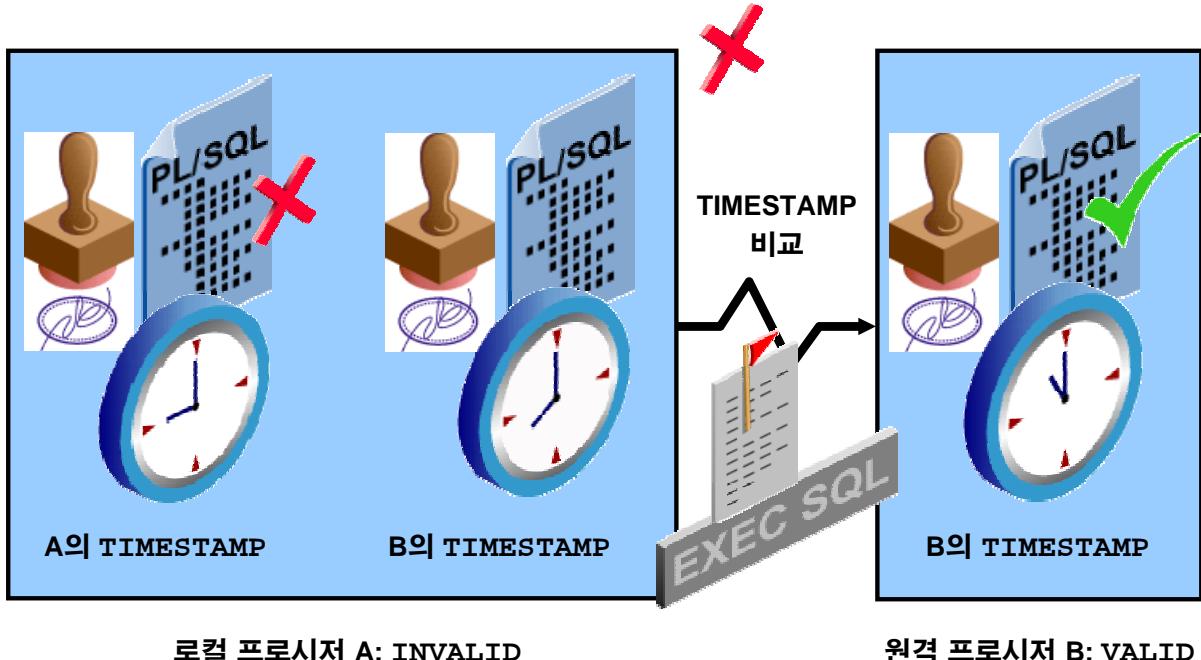
Copyright © 2009, Oracle. All rights reserved.

원격 프로시저를 참조하는 로컬 프로시저

원격 프로시저 B가 오전 11:00에 성공적으로 재컴파일되었다고 가정합시다. 새 시간 기록이 해당 프로시저의 P code와 함께 기록됩니다.

프로시저 A 실행

B의 저장된 TIMESTAMP != B의 컴파일 시간



ORACLE

Copyright © 2009, Oracle. All rights reserved.

자동 원격 종속성

시간 기록이 같지 않으면(원격 프로시저가 재컴파일되었음을 나타냄) Oracle 서버는 로컬 프로시저를 무효화한 다음 런타임 오류를 반환합니다. 로컬 프로시저(현재 유효하지 않은 상태로 표시됨)가 두번째로 호출되면 자동 로컬 종속성 방식에 따라 Oracle 서버는 이 프로시저를 실행하기 전에 재컴파일합니다.

참고: 로컬 프로시저가 처음으로 호출될 때 런타임 오류가 반환될 경우(원격 프로시저의 시간 기록이 변경되었음을 나타냄) 로컬 프로시저를 다시 호출하는 전략을 마련해야 합니다. 슬라이드 예제에서 원격 프로시저는 오전 11:00에 재컴파일되며 이 시간이 P code에 시간 기록으로 기록됩니다. 로컬 프로시저 A의 P code에서 원격 프로시저 B의 시간 기록은 여전히 오전 8:00로 남아 있습니다. 로컬 프로시저 A의 P code에 기록된 시간 기록이 원격 프로시저 B에 기록된 시간 기록과 다르기 때문에 로컬 프로시저는 유효하지 않은 상태로 표시됩니다. 로컬 프로시저는 두번째로 호출될 때 성공적으로 컴파일된 다음 유효한 상태로 표시될 수 있습니다.

시간 기록 모드의 단점은 불필요하게 제한적이라는 점입니다. 네트워크에서 종속 객체의 재컴파일이 꼭 필요하지 않은 경우에도 종종 수행되어 성능 저하를 일으킵니다.

서명 모드

- **프로시저의 서명에는 다음이 포함됩니다.**
 - 프로시저의 이름
 - 파라미터의 데이터 유형
 - 파라미터 모드
- **원격 프로시저의 서명은 로컬 프로시저에 저장됩니다.**
- **종속 프로시저를 실행할 때 참조된 원격 프로시저의 서명이 비교됩니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

서명

시간 기록 종속성 모델과 관련된 몇 가지 문제를 해결하는 데 서명 모델을 사용할 수 있습니다. 서명 모델을 사용하면 로컬 프로시저에 영향을 주지 않고 원격 프로시저를 재컴파일할 수 있습니다. 이는 분산 데이터베이스일 경우에 중요합니다.

서브 프로그램의 서명에는 다음 정보가 포함되어 있습니다.

- 서브 프로그램 이름
- 파라미터의 데이터 유형
- 파라미터 모드
- 파라미터 수
- 함수 반환 값의 데이터 유형

원격 프로그램은 변경되고 재컴파일되었지만 서명은 변경되지 않은 경우 로컬 프로시저는 원격 프로시저를 실행할 수 있습니다. 시간 기록 방식을 사용할 경우 시간 기록이 일치하지 않기 때문에 오류가 발생합니다.

PL/SQL 프로그램 단위 재컴파일

재컴파일:

- 암시적 런타임 재컴파일을 통해 자동으로 처리됩니다.
- ALTER 문을 사용한 명시적 재컴파일을 통해 처리됩니다.

```
ALTER PROCEDURE [ SCHEMA. ]procedure_name COMPILE;
```

```
ALTER FUNCTION [ SCHEMA. ]function_name COMPILE;
```

```
ALTER PACKAGE [ SCHEMA. ]package_name
    COMPILE [ PACKAGE | SPECIFICATION | BODY ];
```

```
ALTER TRIGGER trigger_name [ COMPILE[DEBUG] ];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 객체 재컴파일

재컴파일에 성공할 경우 객체가 유효해집니다. 성공하지 못할 경우 Oracle 서버는 오류를 반환하고 객체는 유효하지 않은 상태로 남아 있습니다. PL/SQL 객체를 재컴파일할 때 Oracle 서버는 먼저 유효하지 않은 종속 객체를 재컴파일합니다.

프로시저: 프로시저에 종속된 로컬 객체(예: 재컴파일된 프로시저를 호출하는 프로시저 또는 재컴파일된 프로시저를 호출하는 프로시저를 정의하는 Package Body)도 무효화됩니다.

패키지: COMPILE PACKAGE 옵션은 객체가 유효하지 않은지 여부와 관계없이 Package Spec과 Body를 모두 재컴파일합니다. COMPILE SPECIFICATION 옵션은 Package Spec을 재컴파일합니다. Package Spec을 재컴파일하면 해당 패키지를 사용하는 서브 프로그램과 같이 Spec에 종속된 로컬 객체가 무효화됩니다. Package Body도 Package Spec에 종속됩니다. COMPILE BODY 옵션은 Package Body만 재컴파일합니다.

트리거: 명시적 재컴파일을 통해 암시적 런타임 재컴파일이 필요하지 않게 되므로 연관된 런타임 컴파일 오류와 성능 오버헤드를 방지할 수 있습니다.

DEBUG 옵션을 사용하면 PL/SQL 컴파일러가 PL/SQL 디버거에 의해 사용될 코드를 생성하고 저장합니다.

재컴파일 실패

종속 프로시저와 함수의 재컴파일이 실패하는 경우:

- 참조된 객체가 삭제되거나 이름이 변경된 경우
- 참조된 열의 데이터 유형이 변경된 경우
- 참조된 열이 삭제된 경우
- 참조된 뷰가 다른 열을 가진 뷰로 바뀐 경우
- 참조된 프로시저의 파라미터 리스트가 수정된 경우

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

재컴파일 실패

참조된 테이블이 삭제되거나 이름이 변경되는 등의 경우가 발생하면 때때로 종속 프로시저의 재컴파일이 실패할 수 있습니다.

재컴파일의 성공 여부는 정확한 종속성에 달려 있습니다. 참조된 뷰가 재생성될 경우 해당 뷰에 종속된 객체를 재컴파일해야 합니다. 뷰에 현재 포함되어 있는 열과 종속 객체 실행을 위해 필요한 열에 따라 재컴파일의 성공 여부가 결정됩니다. 필요한 열이 새 뷰에 포함되어 있지 않을 경우 객체는 유효하지 않은 상태로 남아 있습니다.

재컴파일 성공

종속 프로시저와 함수의 재컴파일이 성공하는 경우:

- 참조된 테이블에 새 열이 있는 경우
- 참조된 열의 데이터 유형이 변경되지 않은 경우
- 전용(private) 테이블은 삭제되었지만 동일한 이름과 구조의 공용(public) 테이블이 존재하는 경우
- 참조된 프로시저의 PL/SQL 본문이 수정되어 성공적으로 재컴파일된 경우

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

재컴파일 성공

종속 객체의 재컴파일이 성공하는 경우는 다음과 같습니다.

- 참조된 테이블에 새 열이 추가되는 경우
- 모든 INSERT 문이 열 리스트를 포함하는 경우
- NOT NULL로 정의된 새 열이 없는 경우

종속 프로시저가 참조하는 전용(private) 테이블이 삭제된 경우 종속 프로시저는 유효하지 않은 상태가 됩니다. 프로시저가 명시적으로 또는 암시적으로 재컴파일되고 공용(public) 테이블이 있는 경우 프로시저는 성공적으로 재컴파일될 수 있지만 이제 공용 테이블에 종속됩니다. 프로시저에서 필요로 하는 열이 공용 테이블에 포함되어 있을 경우에만 재컴파일이 성공하고, 그렇지 않을 경우 프로시저는 유효하지 않은 상태로 남아 있습니다.

프로시저 재컴파일

다음 방법으로 종속성 failure를 최소화합니다.

- **%ROWTYPE 속성으로 레코드 선언**
- **%TYPE 속성으로 변수 선언**
- **SELECT * 표기법으로 query**
- **INSERT 문을 사용하여 열 리스트 삽입**

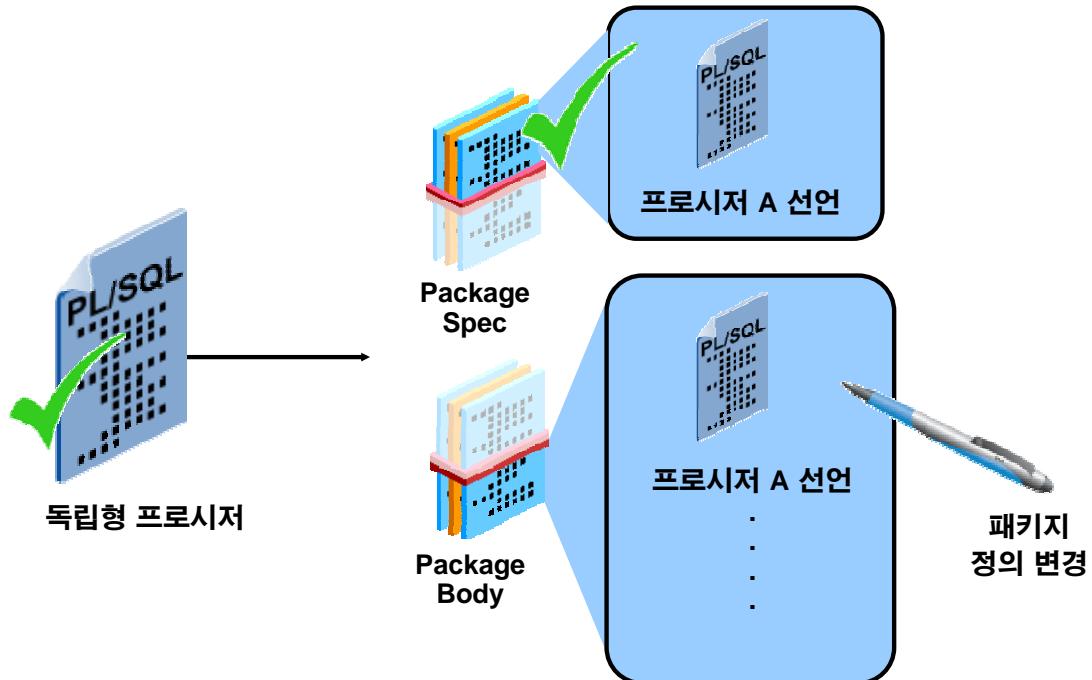
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

프로시저 재컴파일

위 슬라이드에 표시된 지침을 따르면 재컴파일 failure를 최소화할 수 있습니다.

패키지 및 종속성: 서브 프로그램에서 패키지 참조



ORACLE

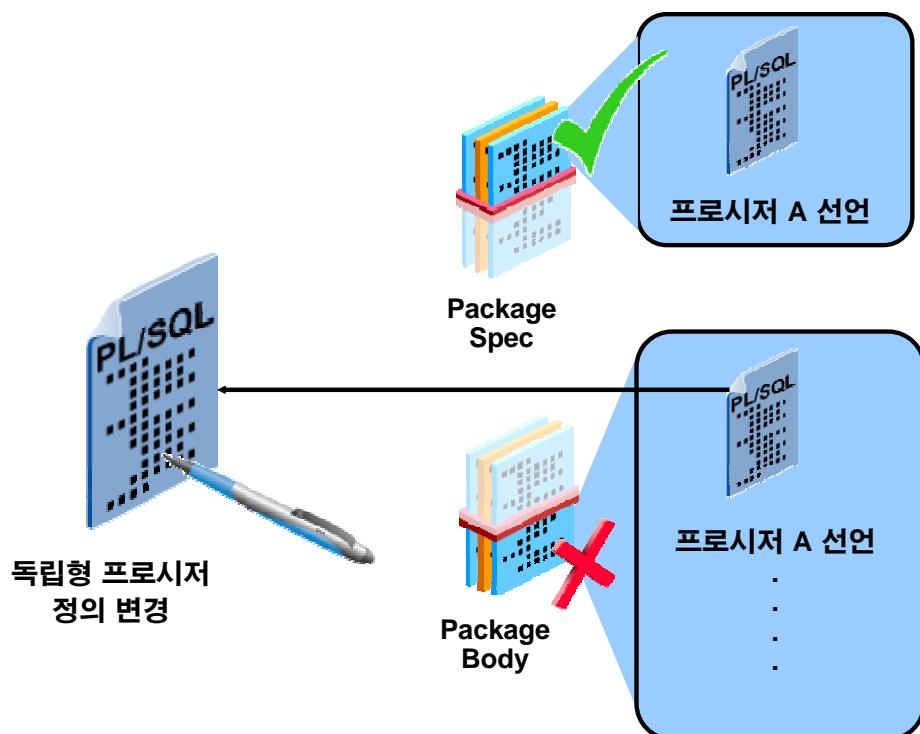
Copyright © 2009, Oracle. All rights reserved.

패키지 및 종속성: 서브 프로그램에서 패키지 참조

독립형 프로시저나 함수에서 패키지 프로시저나 함수를 참조할 때 패키지를 사용하여 종속성 관리를 간소화할 수 있습니다.

- Package Body는 변경되었지만 Package Spec이 변경되지 않은 경우 패키지 생성자를 참조하는 독립형 프로시저는 유효한 상태로 남아 있습니다.
- Package Spec이 변경된 경우 패키지 생성자를 참조하는 외부 프로시저는 무효화되며, Package Body도 마찬가지입니다.

패키지 및 종속성: 패키지 서브 프로그램에서 프로시저 참조



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

패키지 및 종속성: 패키지 서브 프로그램에서 프로시저 참조

패키지 내에서 참조된 독립형 프로시저가 변경된 경우 전체 Package Body는 무효화되지만 Package Spec은 유효한 상태로 남아 있습니다. 따라서 프로시저를 패키지에 포함시킬 것을 권장합니다.

퀴즈

utldtree.sql 스크립트를 실행하고 참조된 특정 객체에 대한 정보로 DEPTREE_TEMP TAB 테이블을 채운 다음 DEPTREE 또는 IDEPTREE 뷰를 query하여 직접 및 간접 종속성을 표시할 수 있습니다.

- 1. 맞습니다.**
- 2. 틀립니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1

직접 및 간접 종속성 표시

다음과 같이 직접 및 간접 종속성을 표시할 수 있습니다.

1. 직접 및 간접 종속성을 표시할 수 있는 객체를 생성하는 **utldtree.sql** 스크립트를 실행합니다.
2. DEPTREE_FILL 프로시저를 실행하여 참조된 특정 객체에 대한 정보로 DEPTREE_TEMP TAB 테이블을 채웁니다.
3. DEPTREE 또는 IDEPTREE 뷰를 query합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- **프로시저 종속성 추적**
- **프로시저 및 함수에서 데이터베이스 객체를 변경한 결과 예측**
- **프로시저 종속성 관리**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

작업 실행을 방해하지 않으려면 종속 프로시저를 추적하고 있다가 데이터베이스 객체 정의가 변경된 후 가능한 한 빨리 수동으로 해당 프로시저를 재컴파일하십시오.

연습 12 개요: 스키마에서 종속성 관리

이 연습에서는 다음 내용을 다룹니다.

- DEPTREE_FILL 및 IDEPTREE를 사용하여 종속성 확인
- 프로시저, 함수 및 패키지 재컴파일

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 12: 개요

이 연습에서는 DEPTREE_FILL 프로시저와 IDEPTREE 뷰를 사용하여 스키마에서 종속성을 조사하며 유효하지 않은 프로시저, 함수, 패키지 및 뷰를 재컴파일합니다.