

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

학생용 • 볼륨 2

D49986KR20

Edition 2.0

2009년 12월

D64313

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

부록 A

연습 및 해답

목차

단원 I 의 연습 및 해답	4
연습 I-1: 사용 가능한 SQL Developer 리소스 식별	5
연습 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용	6
연습 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행.....	7
연습 I-4: 일부 SQL Developer 환경 설정 구성	8
연습 I-5: Oracle Database 11g Release 2 온라인 설명서 라이브러리에 액세스.....	9
연습 해답 I-1: 사용 가능한 SQL Developer 리소스 식별	10
연습 해답 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용	12
연습 해답 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행	15
연습 해답 I-4: 일부 SQL Developer 환경 설정 구성	19
연습 해답 I-5: Oracle Database 11g Release 2 온라인 설명서 라이브러리에 액세스.....	23
단원 1 의 연습 및 해답.....	24
연습 1-1: 프로시저 생성, 컴파일 및 호출	25
연습 해답 1-1: 프로시저 생성, 컴파일 및 호출.....	27
단원 2 의 연습 및 해답.....	38
연습 2-1: 함수 생성	39
연습 2-2: SQL Developer 디버거 소개.....	41
연습 해답 2-1: 함수 생성	42
연습 해답 2-2: SQL Developer 디버거 소개	48
단원 3 의 연습 및 해답.....	58
연습 3-1: 패키지 생성 및 사용.....	59
연습 해답 3-1: 패키지 생성 및 사용	61
단원 4 의 연습 및 해답.....	68
연습 4-1: 패키지 작업	69
연습 해답 4-1: 패키지 작업	73
단원 5 의 연습 및 해답.....	103
연습 5-1: UTL_FILE 패키지 사용	104
연습 해답 5-1: UTL_FILE 패키지 사용	105
단원 6 의 연습 및 해답.....	109
연습 6-1: Native Dynamic SQL 사용.....	110
연습 해답 6-1: Native Dynamic SQL 사용.....	112
단원 7 의 연습 및 해답.....	122
연습 7-1: 대량 바인드 및 독립 트랜잭션 사용	123
연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용	125
단원 8 의 연습 및 해답.....	145
연습 8-1: 문장 및 행 트리거 생성	146
연습 해답 8-1: 문장 및 행 트리거 생성	148
단원 9 의 연습 및 해답.....	156
연습 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리	157
연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리	160
단원 10 의 연습 및 해답.....	173
연습 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용	174
연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용	176

단원 11 의 연습 및 해답	184
연습 11-1: 조건부 컴파일 사용	185
연습 해답 11-1: 조건부 컴파일 사용	187
단원 12 의 연습 및 해답	193
연습 12-1: 스키마에서 종속성 관리	194
연습 해답 12-1: 스키마에서 종속성 관리	195

단원 I 의 연습 및 해답

본 과정에 나오는 여러 연습 중 첫번째입니다. 필요한 경우 "부록 A: 연습 및 해답"에서 해답을 찾아볼 수 있습니다. 연습에서는 해당 단원에 나오는 대부분의 주제를 다룹니다.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

이 연습에서는 사용 가능한 SQL Developer 리소스를 검토합니다. 또한 이 과정에서 사용하게 될 유저 계정에 대해서도 살펴봅니다. 그런 다음 SQL Developer를 시작하고, 새 데이터베이스 연결을 생성하고, 스키마 테이블을 탐색하고, 간단한 익명 블록을 생성하여 실행합니다. 또한 SQL Developer 환경 설정을 몇 개 설정하고 SQL 문을 실행하며 SQL Worksheet를 사용하여 익명 PL/SQL 블록을 실행합니다. 마지막으로 Oracle Database 11g 설명서와 이 과정에서 사용할 수 있는 기타 유용한 웹 사이트에 액세스하고 책갈피를 지정합니다.

연습 I-1: 사용 가능한 SQL Developer 리소스 식별

이 연습에서는 사용 가능한 SQL Developer 리소스를 검토합니다.

- 1) 필요한 경우 부록 C: SQL Developer 사용을 참조하여 Oracle SQL Developer에 익숙해지도록 합니다.
- 2) 다음 위치의 SQL Developer 홈 페이지에 온라인으로 액세스합니다.
http://www.oracle.com/technology/products/database/sql_developer/index.html
- 3) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.
- 4) 다음 위치의 SQL Developer 자습서에 온라인으로 액세스합니다.
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>
- 5) 필요한 경우 자습서에서 사용 가능한 링크 및 데모, 특히 "Creating a Database Connection" 및 "Accessing Data" 링크를 미리 보고 경험합니다.

연습 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용

이 연습에서는 연결 정보를 사용하여 SQL Developer 를 시작하고 새 데이터베이스 연결을 생성합니다.

- 1) 강사가 알려준 유저 ID 및 암호(예: ora61)를 사용하여 SQL Developer 를 시작합니다.
- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.
 - a) Connection Name: MyDBConnection
 - b) Username: ora61
 - c) Password: ora61
 - d) Hostname: PC 의 호스트 이름을 입력합니다.
 - e) Port: 1521
 - f) SID: ORCL
- 3) 새 연결을 테스트합니다. Status 가 Success 이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.
 - a) Connections 탭 페이지에서 MyDBConnection 아이콘을 두 번 누릅니다.
 - b) New>Select Database Connection window 의 Test 버튼을 누릅니다. 상태가 Success 이면 Connect 버튼을 누릅니다.

연습 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행

이 연습에서는 스키마 테이블을 탐색하고 간단한 익명 블록을 생성하여 실행합니다.

- 1) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.
 - a) 옆에 있는 더하기 기호를 눌러 MyDBConnection 연결을 확장합니다.
 - b) 옆에 있는 더하기(+) 기호를 눌러 Tables 아이콘을 확장합니다.
 - c) EMPLOYEES 테이블의 구조를 표시합니다.
 - 2) EMPLOYEES 테이블을 탐색하고 해당 데이터를 표시합니다.
 - 3) SQL Worksheet 를 사용하여 연봉이 \$10,000 가 넘는 모든 사원의 성 및 급여를 선택합니다. Execute Statement (F9) 및 Run Script (F5) 아이콘을 모두 사용하여 SELECT 문을 실행합니다. 해당 템에서 SELECT 문을 실행하여 두 메소드 모두의 결과를 검토합니다.
- 참고:** 몇 분 동안 데이터를 살펴보거나, 이 과정에서 사용할 HR 스키마의 모든 테이블에 대한 설명과 데이터가 제공된 부록 B를 참조하십시오.
- 4) "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다.
 - a) SET SERVEROUTPUT ON 을 활성화하여 DBMS_OUTPUT 패키지 문의 출력을 표시합니다.
 - b) SQL Worksheet 영역을 사용하여 익명 블록의 코드를 입력합니다.
 - c) Run Script (F5) 아이콘을 눌러 익명 블록을 실행합니다.

연습 I-4: 일부 SQL Developer 환경 설정 구성

이 연습에서는 일부 SQL Developer 환경 설정을 구성합니다.

- 1) SQL Developer 메뉴에서 Tools > Preferences로 이동합니다.
Preferences window가 표시됩니다.
- 2) Code Editor 옵션을 확장하고 Display 옵션을 눌러 "Code Editor: Display" 섹션을 표시합니다. "Code Editor: Display" 섹션에는 코드 편집기의 모양과 동작에 대한 일반 옵션이 포함되어 있습니다.
 - a) Show Visible Right Margin 섹션에서 Right Margin Column 텍스트 상자에 100을 입력합니다. 그러면 코드 행의 길이를 제어하기 위해 설정할 수 있는 오른쪽 여백이 렌더링됩니다.
 - b) Line Gutter 옵션을 누릅니다. Line Gutter 옵션은 Line Gutter(코드 편집기의 왼쪽 여백)의 옵션을 지정합니다. Show Line Numbers 체크 박스를 선택하여 코드 행 번호를 표시합니다.
- 3) Database 옵션 아래에서 Worksheet Parameters 옵션을 누릅니다. "Select default path to look for scripts" 텍스트 상자에서 /home/oracle/labs/plpu 폴더를 지정합니다. 이 과정에서 사용하는 솔루션 스크립트, 코드 예제 스크립트, 실습 또는 데모가 이 폴더에 있습니다.
- 4) OK를 눌러 변경 사항을 그대로 적용하고 Preferences window를 종료합니다.
- 5) /home/oracle/labs/plpu 폴더를 익힙니다.
 - a) Connections 탭 옆에 있는 Files 탭을 누릅니다.
 - b) /home/oracle/labs/plpu 폴더로 이동합니다. labs 폴더에 하위 폴더가 몇 개 보입니까?
 - c) 폴더를 탐색한 후 코드를 실행하지 않고 스크립트 파일을 엽니다.
 - d) SQL Worksheet 영역에서 표시된 코드를 지웁니다.

연습 I-5: Oracle Database 11g Release 2 온라인 설명서 라이브러리에 액세스

이 연습에서는 이 과정에서 사용할 일부 Oracle Database 11g Release 2 참고 설명서에 액세스하고 책갈피를 지정합니다.

- 1) Oracle Database 11g Release 2 설명서 웹
페이지(<http://www.oracle.com/pls/db111/homepage>)에 액세스합니다.
- 2) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.
- 3) Oracle Database 11g Release 2에 대한 전체 설명서 리스트를 표시합니다.
- 4) 이 과정에서 필요에 따라 사용할 다음 참고 설명서를 메모해 둡니다.
 - a) *Advanced Application Developer's Guide*
 - b) *New Features Guide*
 - c) *PL/SQL Language Reference*
 - d) *Oracle Database Reference*
 - e) *Oracle Database Concepts*
 - f) *SQL Developer User's Guide*
 - g) *SQL Language Reference Guide*
 - h) *SQL*Plus User's Guide and Reference*

연습 해답 I-1: 사용 가능한 SQL Developer 리소스 식별

이 연습에서는 사용 가능한 SQL Developer 리소스를 검토합니다.

- 1) 필요한 경우 부록 C: SQL Developer 사용을 참조하여 Oracle SQL Developer에 익숙해지도록 합니다.
- 2) 다음 위치의 SQL Developer 흄 페이지에 온라인으로 액세스합니다.

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

SQL Developer 흄 페이지가 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Developer page on Oracle Technology Network. The URL in the browser is http://www.oracle.com/technology/products/database/sql_developer/index.html. The page features a sidebar with links to various Oracle products and technologies like Database, Middleware, Developer Tools, etc. The main content area has a heading 'Oracle SQL Developer' with a brief description: 'Oracle SQL Developer is a **free** and **fully supported** graphical tool for database development. With SQL Developer, you can browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements. You can also run any number of provided reports, as well as create and save your own. SQL Developer enhances productivity and simplifies your database development tasks.' Below this, there are several links: 'What is Oracle SQL Developer?', 'SQL Developer 1.5', 'SQL Developer OTN Forum', 'White papers and Supporting Documents', 'SQL Developer Exchange', 'Online Demonstrations', 'SQL Developer Third Party Extensions', 'Tutorials and Oracle by Example (OBE)', 'Blogs, Magazine Articles and Podcasts', and 'SQL Developer Documentation'. To the right, there's a 'FREE DOWNLOAD' button with a link to 'Download (July '09) Oracle SQL Developer 1.5.5'. A 'Events' section lists conferences like NoCOUG, VOUG, Oracle OpenWorld, AUSOUG, and UKOUG. A 'Related Technologies' section includes links to Oracle SQL Developer Data Modeler, Oracle Database 11g, and Oracle Database 10g Express.

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

정해진 해답은 없습니다. 다음과 같이 연결 도구 모음에 링크가 추가됩니다.

연습 해답 I-1: 사용 가능한 SQL Developer 리소스 식별 (계속)

- 4) 다음 위치의 SQL Developer 자습서에 온라인으로 액세스합니다.

<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>

위의 URL 을 사용하여 SQL Developer 자습서에 액세스합니다. 다음 페이지가 표시됩니다.

The screenshot shows the Oracle SQL Developer Tutorial website in Mozilla Firefox 3. The title bar says "Oracle SQL Developer Tutorial - Mozilla Firefox 3". The menu bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The toolbar has icons for Back, Forward, Stop, Home, and Search. The address bar shows the URL: http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm. Below the address bar are several bookmarks: Smart Bookmarks, Oracle SQL Developer, Enterprise Linux, Linux Technology C..., Oracle University, and Feature: Faster. The main content area has a sidebar on the left with a navigation tree:

- Oracle SQL Developer Tutorial
 - Using this Tutorial
 - Product Overview
 - Installation
 - What to Do First
 - Creating a Database Connection
 - Working with Database Objects
 - Accessing Data
 - Building Simple Queries
 - Accessing Multiple Tables
 - Using Functions to Customize Output
 - Working With Groups of Data
 - Manipulating Data
 - Adding PL/SQL Components
 - Building Reports
 - Using Source Code Control
 - Additional Learning Resources
 - Definitions
 - Summary

The main content area displays the "Welcome to the Oracle SQL Developer Tutorial!" page. It includes a brief description of the tutorial's purpose, learning objectives, and a "Start Tutorial" button.

- 5) 필요한 경우 자습서에서 사용 가능한 링크 및 데모, 특히 "Creating a Database Connection" 및 "Accessing Data" 링크를 미리 보고 경험합니다.

데이터베이스 연결 생성에 대한 섹션을 검토하려면 "What to Do First" 링크 옆에 있는 더하기 "+" 기호를 눌러 "Creating a Database Connection" 링크를 표시합니다.

"Creating a Database Connection" 항목을 검토하려면 항목의 링크를 누릅니다.

데이터 액세스에 대한 섹션을 검토하려면 "Accessing Data" 링크 옆의 더하기 "+" 기호를 눌러 사용 가능한 항목 리스트를 표시합니다. 항목을 검토하려면 해당 항목의 링크를 누릅니다.

연습 해답 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용

이 연습에서는 연결 정보를 사용하여 SQL Developer를 시작하고 새 데이터베이스 연결을 생성합니다.

- 1) 강사가 알려준 유저 ID 및 암호(예: ora61)를 사용하여 SQL Developer를 시작합니다.

바탕 화면에서 SQL Developer 아이콘을 누릅니다.



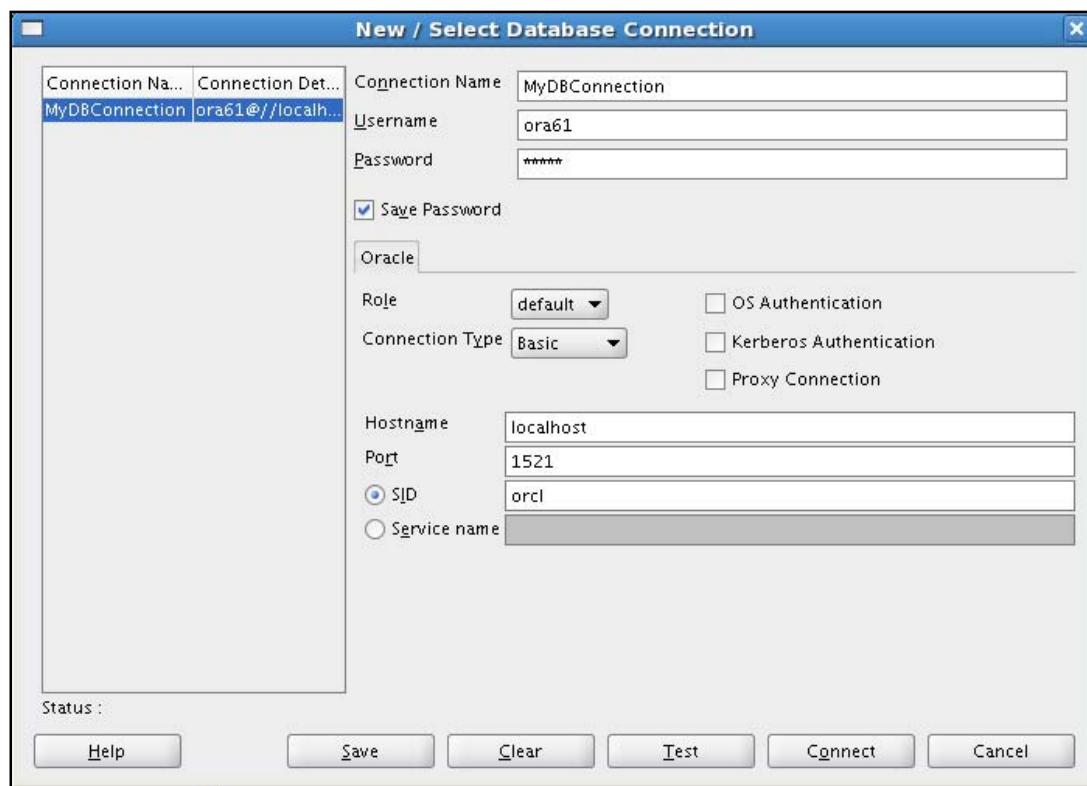
- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

- a) Connection Name: MyDBConnection
- b) Username: ora61
- c) Password: ora61
- d) Hostname: PC의 호스트 이름을 입력합니다.
- e) Port: 1521
- f) SID: ORCL

연습 해답 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용 (계속)

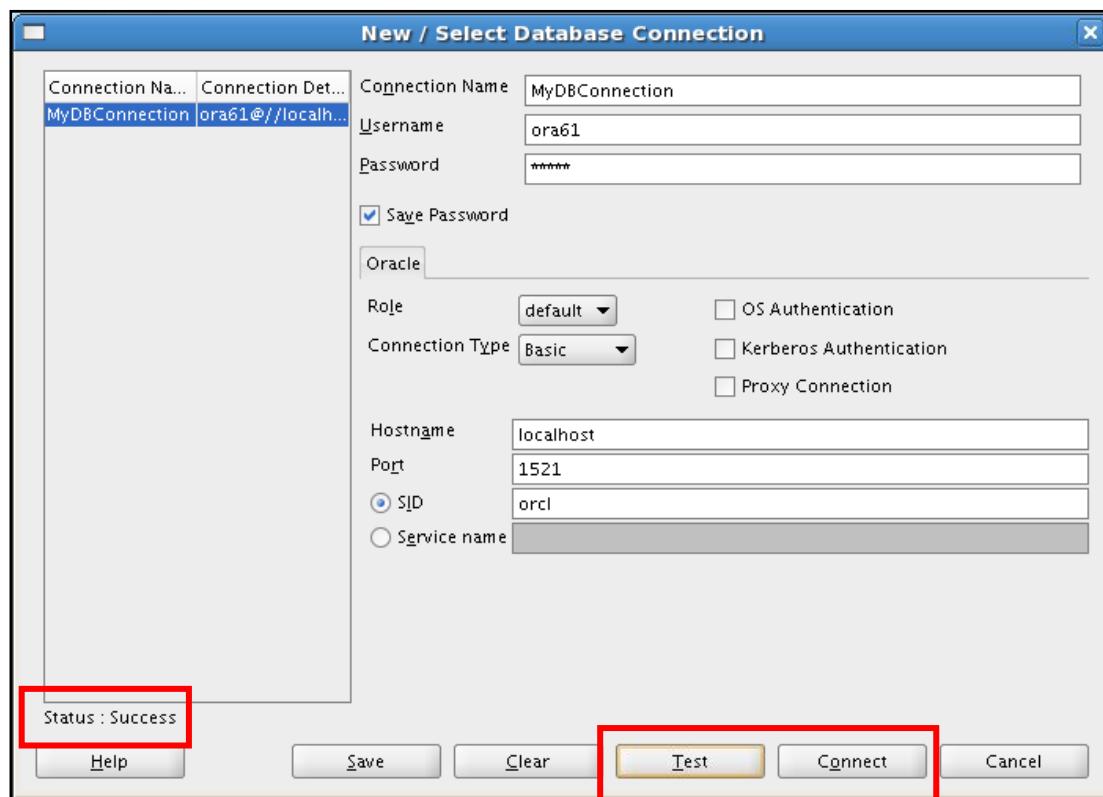
Connections 탭 페이지의 Connections 아이콘을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 New Connection 옵션을 선택합니다. New>Select Database Connection window 가 표시됩니다. 제공된 이전 정보를 사용하여 새 데이터베이스 연결을 생성합니다.

참고: 새로 생성된 연결의 속성을 표시하려면 연결 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Properties 를 선택합니다. 유저 이름, 암호, 호스트 이름 및 서비스 이름을 강사가 제공한 적절한 정보로 대체합니다. 다음은 ora61 수강생을 위해 새로 생성한 데이터베이스 연결의 예제입니다.



연습 해답 I-2: 새 SQL Developer 데이터베이스 연결 생성 및 사용 (계속)

- 3) 새 연결을 테스트합니다. Status 가 Success 이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.
- Connections 탭 페이지에서 MyDBConnection 아이콘을 두 번 누릅니다.
 - New>Select Database Connection window 의 Test 버튼을 누릅니다. 상태가 Success 이면 Connect 버튼을 누릅니다.



연습 해답 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행

이 연습에서는 스키마 테이블을 탐색하고 간단한 익명 블록을 생성하여 실행합니다.

- 1) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.
 - a) 옆에 있는 더하기(+) 기호를 눌러 MyDBConnection 연결을 확장합니다.
 - b) 옆에 있는 더하기(+) 기호를 눌러 Tables 아이콘을 확장합니다.
 - c) EMPLOYEES 테이블의 구조를 표시합니다.

EMPLOYEES 테이블을 두 번 누릅니다. **Columns** 탭에 **EMPLOYEES** 테이블의 열이 다음과 같이 표시됩니다.

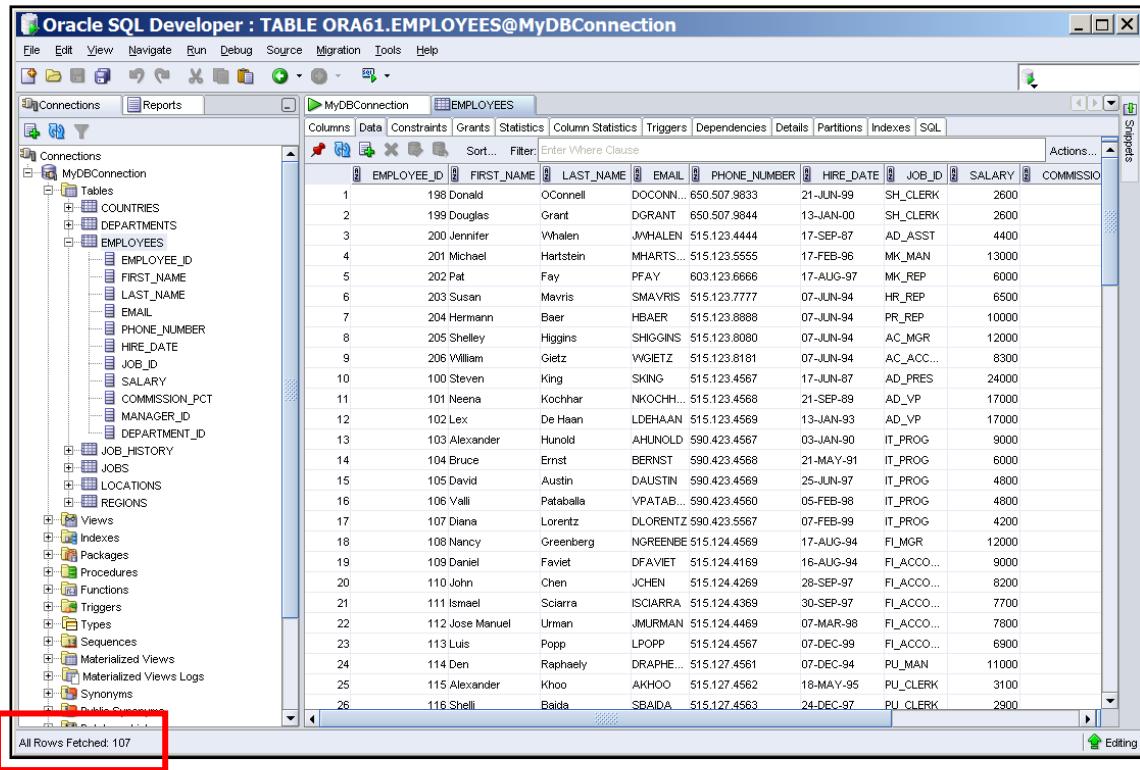
The screenshot shows the Oracle SQL Developer interface. On the left, the Connections pane shows a connection named 'MyDBConnection' expanded, revealing tables like COUNTRIES, DEPARTMENTS, and EMPLOYEES. The EMPLOYEES table is selected. The main workspace displays the 'EMPLOYEES' table structure in the 'Columns' tab. The columns listed are:

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	Comments
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1	Primary key of employees table.
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2		(null) First name of the employee. A not null column.
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3		(null) Last name of the employee. A not null column.
EMAIL	VARCHAR2(25 BYTE)	No	(null)	4		(null) Email id of the employee.
PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5		(null) Phone number of the employee; includes country code and area code.
HIRE_DATE	DATE	No	(null)	6		(null) Date when the employee started on this job. A not null column.
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7		(null) Current job of the employee; foreign key to job_id column.
SALARY	NUMBER(8,2)	Yes	(null)	8		(null) Monthly salary of the employee. Must be greater than zero.
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9		(null) Commission percentage of the employee. Only employees with non-null commission_pct receive commissions.
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10		(null) Manager id of the employee; has same domain as manager_id column.
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11		(null) Department id where employee works; foreign key to department_id column.

연습 해답 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행 (계속)

- 2) EMPLOYEES 테이블을 탐색하고 해당 데이터를 표시합니다.

사원의 데이터를 표시하려면 Data 탭을 누릅니다. EMPLOYEES 테이블 데이터가 다음과 같이 표시됩니다.



The screenshot shows the Oracle SQL Developer interface with the title bar "Oracle SQL Developer : TABLE ORA61.EMPLOYEES@MyDBConnection". The left sidebar shows the database structure under "MyDBConnection" with the "EMPLOYEES" table selected. The main pane displays the data from the EMPLOYEES table, with 107 rows fetched. The columns shown are: EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, and COMMISSION_PCT. The data includes various employees like Donald O'Connell, Douglas Grant, Jennifer Whalen, Michael Hartstein, Pat Fay, Susan Mavris, Hermann Baer, Shelley Higgins, William Gietz, Steven King, Neena Kochhar, Lex De Haan, Alexander Hunold, Bruce Ernst, David Austin, Valli Pataballa, Diana Lorentz, Nancy Greenberg, Daniel Faviet, John Chen, Ismael Sciarra, Jose Manuel Urman, Luis Popp, Den Raphaely, Alexander Khoo, and Shelli Baida. The salary column ranges from 2600 to 11000.

- 3) SQL Worksheet 를 사용하여 연봉이 \$10,000 가 넘는 모든 사원의 성 및 급여를 선택합니다. Execute Statement (F9) 및 Run Script (F5) 아이콘을 모두 사용하여 SELECT 문을 실행합니다. 해당 탭에서 SELECT 문을 실행하여 두 메소드 모두의 결과를 검토합니다.

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

다음 두 가지 방법 중 하나를 사용하여 SQL Worksheet 를 표시합니다.

1. Tools > SQL Worksheet 를 선택하거나 Open SQL Worksheet 아이콘을 누릅니다. Select Connection window 가 표시됩니다.
2. 아직 선택하지 않은 경우 Connection drop-down list 에서 새 MyDBConnection 을 선택하고 OK 를 누릅니다.

연습 해답 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행 (계속)

다음 두 가지 방법 중 하나를 사용하여 /home/oracle/labs/plpu/solns 폴더에서 sol_I_03.sql 파일을 다음과 같이 엽니다.

1. Files 탭에서 열고자 하는 스크립트 파일을 선택합니다. 또는 해당 파일이 있는 위치로 이동합니다.
2. 열려는 파일의 이름을 두 번 누릅니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
3. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다.

또는 다음과 같이 할 수도 있습니다.

1. File 메뉴에서 Open 을 선택합니다. Open 대화상자가 표시됩니다.
2. Open 대화상자에서 열고자 하는 스크립트 파일을 선택합니다. 또는 해당 파일이 있는 위치로 이동합니다.
3. Open 을 누릅니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
4. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다.

단일 SELECT 문을 실행하려면 해당 행에 커서를 놓고 SQL Worksheet 도구 모음에서 Execute Statement (F9) 아이콘을 눌러 SELECT 문을 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;
```

The screenshot shows the Oracle SQL Worksheet interface with the 'Results' tab selected. The query executed was:

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY > 10000;
```

The results are displayed in a grid:

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Greenberg	12000
Raphaely	11000
Russell	14000
Partners	13500
Errazuriz	12000
Cambrault	11000
Zlotkey	10500
Vishney	10500
Ozer	11500
Abel	11000
Hartstein	13000
Higgins	12000

At the bottom of the results area, it says "15 rows selected".

연습 해답 I-3: 스키마 테이블 탐색 및 간단한 익명 블록 생성 및 실행 (계속)

4) "Hello World"를 출력하는 간단한 익명 블록을 생성하여 실행합니다.

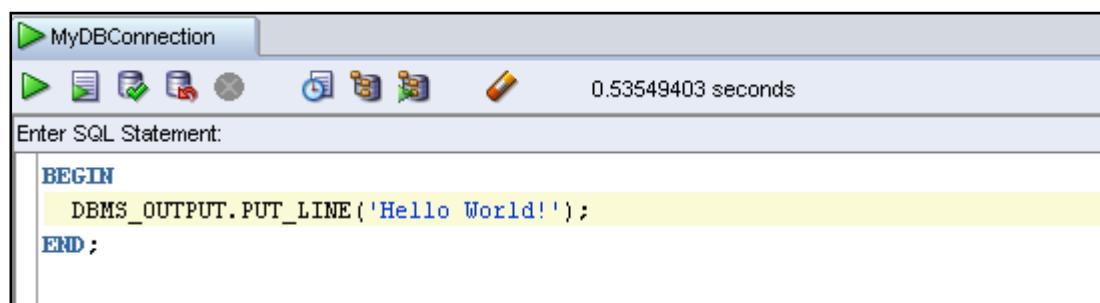
- a) SET SERVEROUTPUT ON 을 활성화하여 DBMS_OUTPUT 패키지 문의 출력을 표시합니다.

SQL Worksheet 영역에 다음 명령을 입력한 다음 Run Script (F5) 아이콘을 누릅니다.

```
SET SERVEROUTPUT ON
```

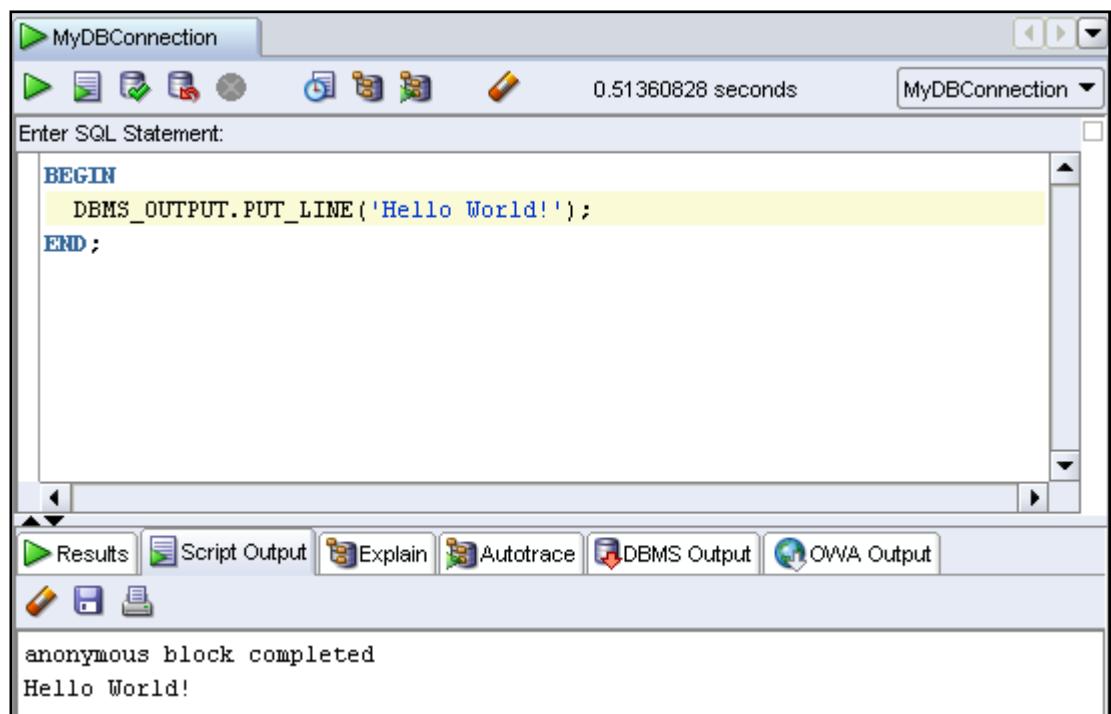
- b) SQL Worksheet 영역을 사용하여 익명 블록의 코드를 입력합니다.

SQL Worksheet 영역에 다음 코드를 아래와 같이 입력합니다. 또는 /home/oracle/labs/plpu/solns 폴더의 sol_I_04.sql 파일을 엽니다. 코드가 다음과 같이 표시됩니다.



- c) Run Script (F5) 아이콘을 눌러 익명 블록을 실행합니다.

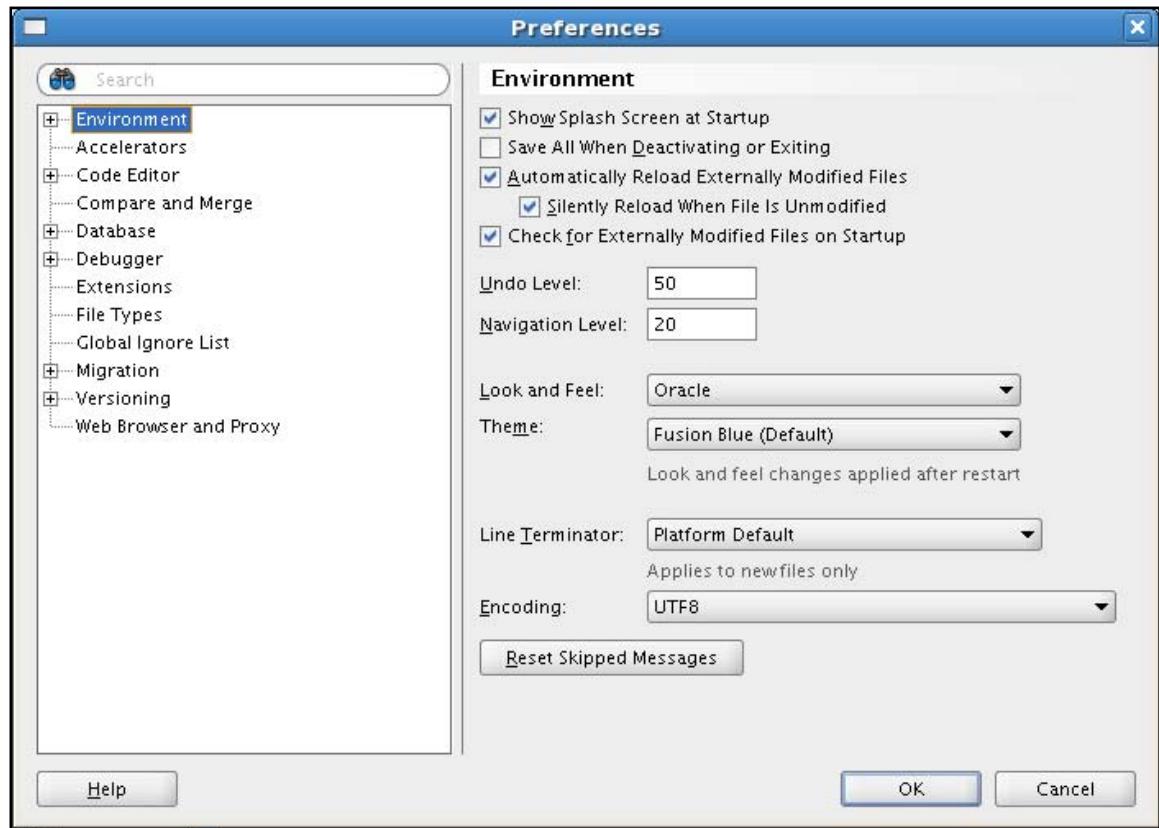
다음과 같이 Script Output 탭에 익명 블록의 출력이 표시됩니다.



연습 해답 I-4: 일부 SQL Developer 환경 설정 구성

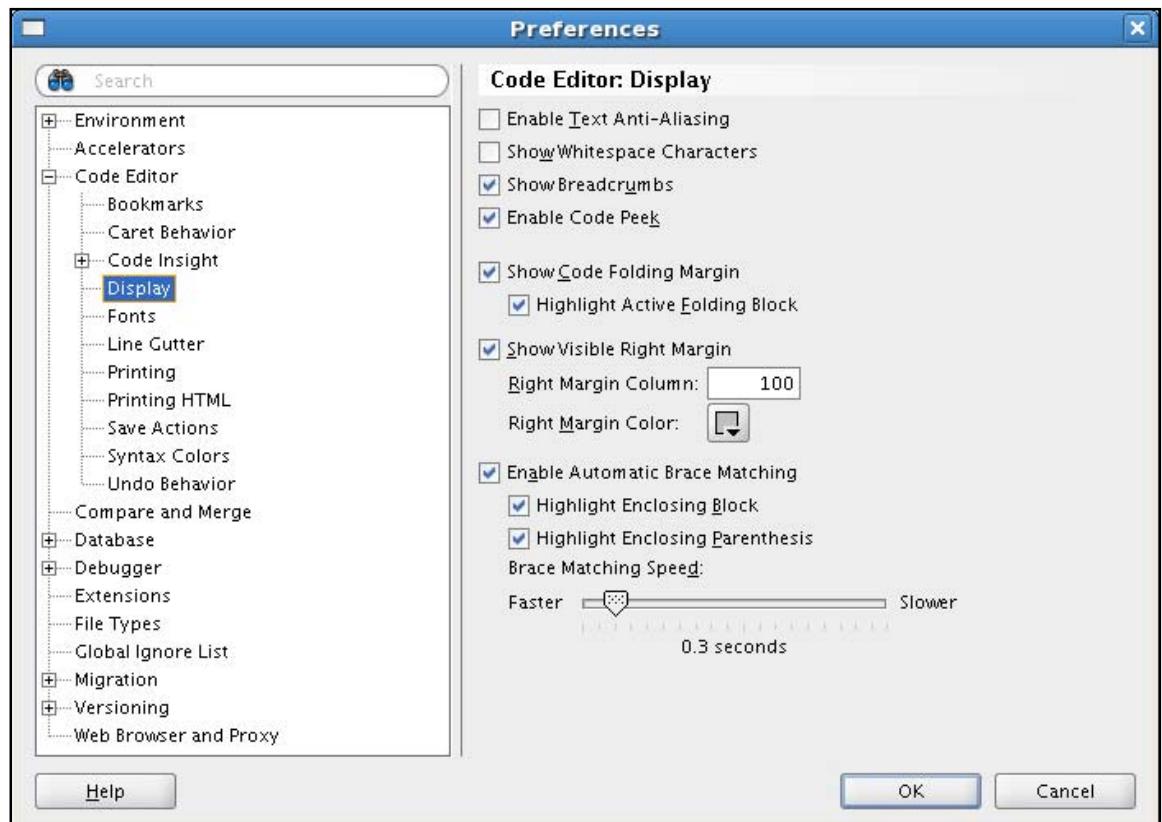
이 연습에서는 일부 SQL Developer 환경 설정을 구성합니다.

- 1) SQL Developer 메뉴에서 Tools > Preferences로 이동합니다. Preferences window 가 표시됩니다.



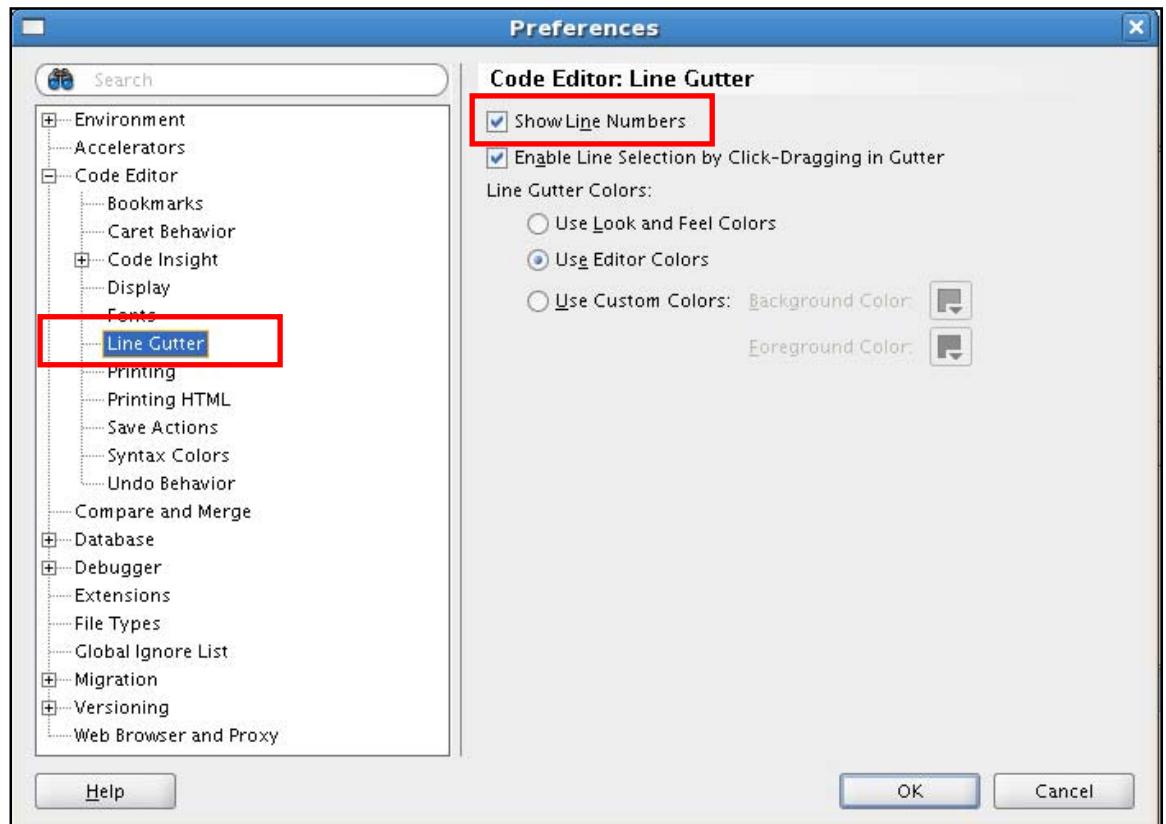
연습 해답 I-4: 일부 SQL Developer 환경 설정 구성 (계속)

- 2) Code Editor 옵션을 확장하고 Display 옵션을 눌러 "Code Editor: Display" 섹션을 표시합니다. "Code Editor: Display" 섹션에는 코드 편집기의 모양과 동작에 대한 일반 옵션이 포함되어 있습니다.
- a) Show Visible Right Margin 섹션에서 Right Margin Column 텍스트 상자에 100 을 입력합니다. 그러면 코드 행의 길이를 제어하기 위해 설정할 수 있는 오른쪽 여백이 렌더링됩니다.



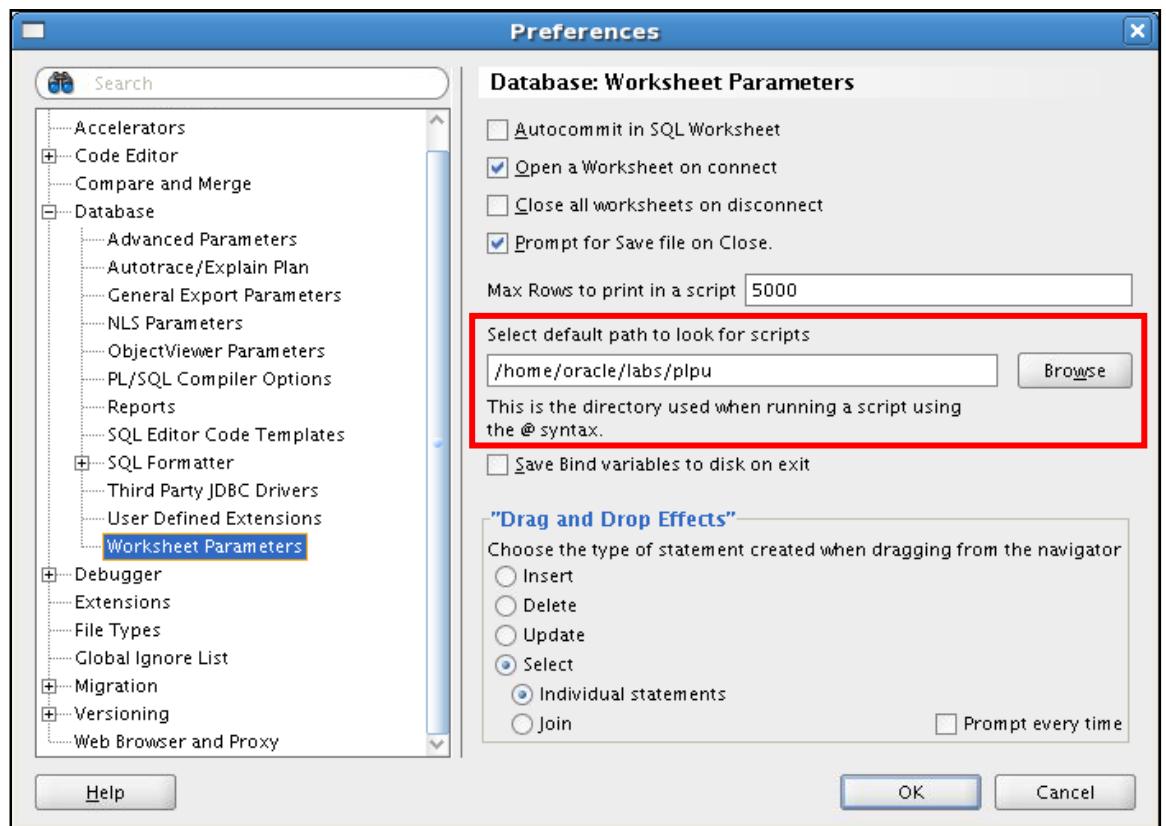
연습 해답 I-4: 일부 SQL Developer 환경 설정 구성 (계속)

- b) Line Gutter 옵션을 누릅니다. Line Gutter 옵션은 Line Gutter(코드 편집기의 왼쪽 여백)의 옵션을 지정합니다. Show Line Numbers 체크 박스를 선택하여 코드 행 번호를 표시합니다.



연습 해답 I-4: 일부 SQL Developer 환경 설정 구성(계속)

- 3) Database 옵션 아래에서 Worksheet Parameters 옵션을 누릅니다. "Select default path to look for scripts" 텍스트 상자에서 /home/oracle/labs/plpu 폴더를 지정합니다. 이 과정에서 사용하는 솔루션 스크립트, 코드 예제 스크립트, 실습 또는 데모가 이 폴더에 있습니다.



- 4) OK 를 눌러 변경 사항을 그대로 적용하고 Preferences window 를 종료합니다.
- 5) /home/oracle/labs/plpu 폴더에 있는 labs 폴더를 충분히 살펴봅니다.
- Connections 탭 옆에 있는 Files 탭을 누릅니다.
 - /home/oracle/labs/plpu 폴더로 이동합니다.
 - labs 폴더에 하위 폴더가 몇 개 보입니까?
 - 폴더를 탐색한 후 코드를 실행하지 않고 스크립트 파일을 엽니다.
 - SQL Worksheet 영역에서 표시된 코드를 지웁니다. SQL Developer 메뉴에서 Tools > Preferences 로 이동합니다.

연습 해답 I-5: Oracle Database 11g Release 2 온라인 설명서 라이브러리에 액세스

이 연습에서는 이 과정에서 사용할 일부 Oracle Database 11g Release 2 참고 설명서에 액세스하고 책갈피를 지정합니다.

- 1) Oracle Database 11g Release 2 설명서 웹
페이지(<http://www.oracle.com/pls/db111/homepage>)에 액세스합니다.
- 2) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.
- 3) Oracle Database 11g Release 2에 대한 전체 설명서 리스트를 표시합니다.
- 4) 이 과정에서 필요에 따라 사용할 다음 참고 설명서를 메모해 둡니다.
 - a) *Advanced Application Developer's Guide*
 - b) *New Features Guide*
 - c) *PL/SQL Language Reference*
 - d) *Oracle Database Reference*
 - e) *Oracle Database Concepts*
 - f) *SQL Developer User's Guide*
 - g) *SQL Language Reference Guide*
 - h) *SQL*Plus User's Guide and Reference*

단원 1 의 연습 및 해답

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

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한
후 다음 단계나 연습으로 진행하십시오.

연습 1-1: 프로시저 생성, 컴파일 및 호출

이 연습에서는 ADD_JOB 프로시저를 생성하여 호출하고 결과를 검토합니다. 또한 UPD_JOB이라는 프로시저를 생성한 후 호출하여 JOBS 테이블에서 직무를 수정하고 DEL_JOB이라는 프로시저를 생성한 후 호출하여 JOBS 테이블에서 직무를 삭제합니다. 마지막으로 사원 ID를 제공할 경우 사원의 급여와 직무 ID를 검색하는 GET_EMPLOYEE라는 프로시저를 생성하여 EMPLOYEES 테이블을 query 합니다.

- 1) ADD_JOB 프로시저를 생성, 컴파일 및 호출하고 결과를 검토합니다.
 - a) JOBS 테이블에 새로운 직무를 삽입하는 ADD_JOB이라는 프로시저를 생성합니다. 두 개의 파라미터를 사용하여 ID 및 직위를 입력합니다.

참고: SQL Worksheet 영역에 코드를 입력하여 프로시저 및 기타 객체를 생성한 다음 Run Script (F5) 아이콘을 누를 수 있습니다. 이렇게 하면 프로시저가 생성되고 컴파일됩니다. 프로시저에 오류가 있는지 알아보려면 프로시저 노드에서 프로시저 이름을 누르고 팝업 메뉴에서 Compile 을 선택합니다.
 - b) IT_DBA 를 직무 ID로 사용하고 Database Administrator 를 직위로 사용하여 프로시저를 호출합니다. JOBS 테이블을 query 하고 결과를 확인합니다.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		
1 rows selected			

- c) 프로시저를 다시 호출하여 직무 ID ST_MAN 과 직위 Stock Manager 를 전달합니다. 어떤 결과가 발생하며, 그 이유는 무엇입니까?
- 2) JOBS 테이블의 직무를 수정하는 UPD_JOB이라는 프로시저를 생성합니다.
 - a) 직위를 갱신하는 UPD_JOB이라는 프로시저를 생성합니다. 두 개의 파라미터를 사용하여 직무 ID 와 새로운 직위를 제공합니다. 갱신이 수행되지 않을 경우에 필요한 예외 처리를 추가합니다.
 - b) 직무 ID IT_DBA 의 직위를 Data Administrator 로 변경하는 프로시저를 호출합니다. JOBS 테이블을 query 하고 결과를 확인합니다.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		
1 rows selected			

연습 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- c) 존재하지 않는 직무의 갱신을 시도하여 프로시저의 예외 처리 섹션을 테스트합니다. 직무 ID IT_WEB 과 직위 Web Master 를 사용할 수 있습니다.

```
Error starting at line 1 in command:
EXECUTE upd_job ('IT_WEB', 'Web Master')
Error report:
ORA-20202: No job updated.
ORA-06512: at "ORA80.UPD_JOB", line 9
ORA-06512: at line 1
```

- 3) JOBS 테이블에서 직무를 삭제하는 DEL_JOB 이라는 프로시저를 생성합니다.

- a) 직무를 삭제하는 DEL_JOB 이라는 프로시저를 생성합니다. 직무가 삭제되지 않을 경우에 필요한 예외 처리를 포함합니다.
- b) 직무 ID IT_DBA 를 사용하여 프로시저를 호출합니다. JOBS 테이블을 query 하고 결과를 확인합니다.

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
<hr/>			
0 rows selected			

- c) 존재하지 않는 직무의 삭제를 시도하여 프로시저의 예외 처리 섹션을 테스트합니다. IT_WEB 을 직무 ID 로 사용합니다. 프로시저의 예외 처리 섹션에 포함했던 메시지가 출력되어야 합니다.

```
Error starting at line 1 in command:
EXECUTE del_job ('IT_WEB')
Error report:
ORA-20203: No jobs deleted.
ORA-06512: at "ORA80.DEL_JOB", line 6
ORA-06512: at line 1
```

- 4) 사원 ID 를 제공할 경우 사원의 급여와 직무 ID 를 검색하는 GET_EMPLOYEE 라는 프로시저를 생성하여 EMPLOYEES 테이블을 query 합니다.

- a) 지정된 사원 ID 에 대한 SALARY 및 JOB_ID 열에서 값을 반환하는 프로시저를 생성합니다. 구문 오류가 있으면 제거한 다음 코드를 재컴파일합니다.
- b) 두 개의 OUT 파라미터, 즉 급여에 대한 파라미터와 직무 ID 에 대한 파라미터에 호스트 변수를 사용하여 프로시저를 실행합니다. 사원 ID 120 에 대한 급여와 직무 ID 를 표시합니다.

```
v_salary
-----
8000

v_job
-----
ST_MAN
```

- c) 프로시저를 다시 호출하고 값이 300 인 EMPLOYEE_ID 를 전달합니다. 어떤 결과가 발생하며, 그 이유는 무엇입니까?

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출

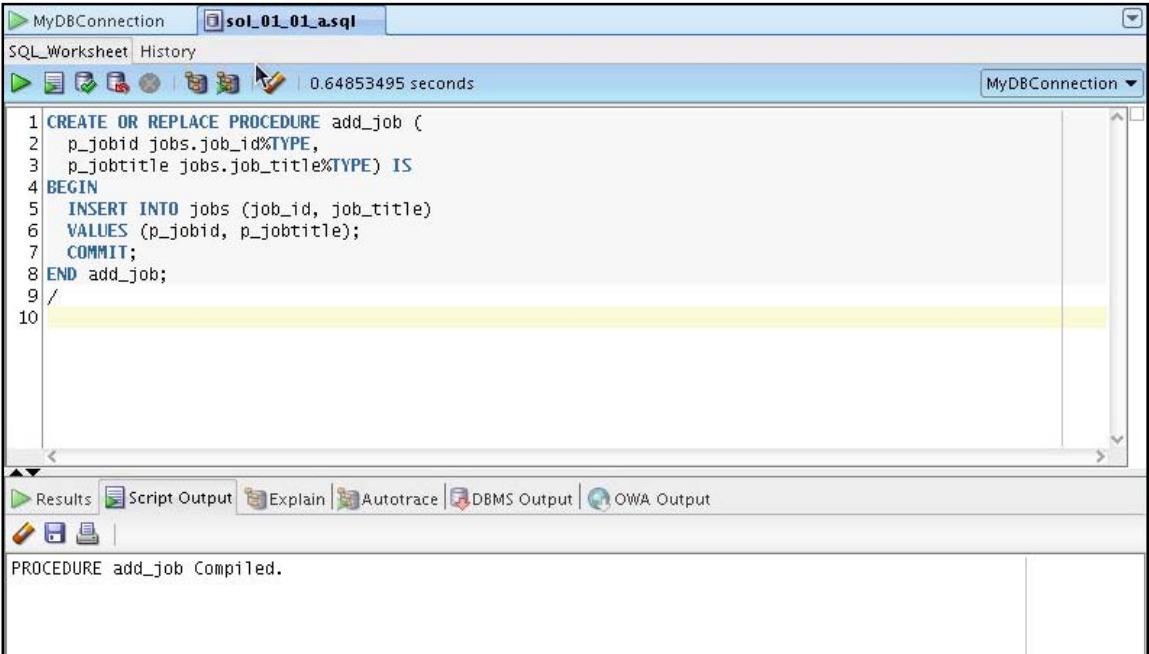
이 연습에서는 ADD_JOB 프로시저를 생성하여 호출하고 결과를 검토합니다. 또한 UPD_JOB이라는 프로시저를 생성한 후 호출하여 JOBS 테이블에서 직무를 수정하고 DEL_JOB이라는 프로시저를 생성한 후 호출하여 JOBS 테이블에서 직무를 삭제합니다. 마지막으로 사원 ID를 제공할 경우 사원의 급여와 직무 ID를 검색하는 GET_EMPLOYEE라는 프로시저를 생성하여 EMPLOYEES 테이블을 query 합니다.

1) ADD_JOB 프로시저를 생성, 컴파일 및 호출하고 결과를 검토합니다.

- JOBS 테이블에 새로운 직무를 삽입하는 ADD_JOB이라는 프로시저를 생성합니다. 두 개의 파라미터를 사용하여 ID 및 직위를 입력합니다.

참고: SQL Worksheet 영역에 코드를 입력하여 프로시저 및 기타 객체를 생성한 다음 Run Script (F5) 아이콘을 누를 수 있습니다. 이렇게 하면 프로시저가 생성되고 컴파일됩니다. 프로시저를 생성할 때 오류 메시지가 생성되면 프로시저 노드에서 해당 프로시저 이름을 누르고 프로시저를 편집한 다음 팝업 메뉴에서 Compile 을 선택합니다.

/home/oracle/labs/plpu/solns 폴더에서 sol_01_01_a.sql 파일을 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.



The screenshot shows the Oracle SQL Worksheet interface. The top window displays the PL/SQL code for the 'add_job' procedure:

```

1 CREATE OR REPLACE PROCEDURE add_job (
2   p_jobid jobs.job_id%TYPE,
3   p_jobtitle jobs.job_title%TYPE) IS
4 BEGIN
5   INSERT INTO jobs (job_id, job_title)
6   VALUES (p_jobid, p_jobtitle);
7   COMMIT;
8 END add_job;
9 /
10

```

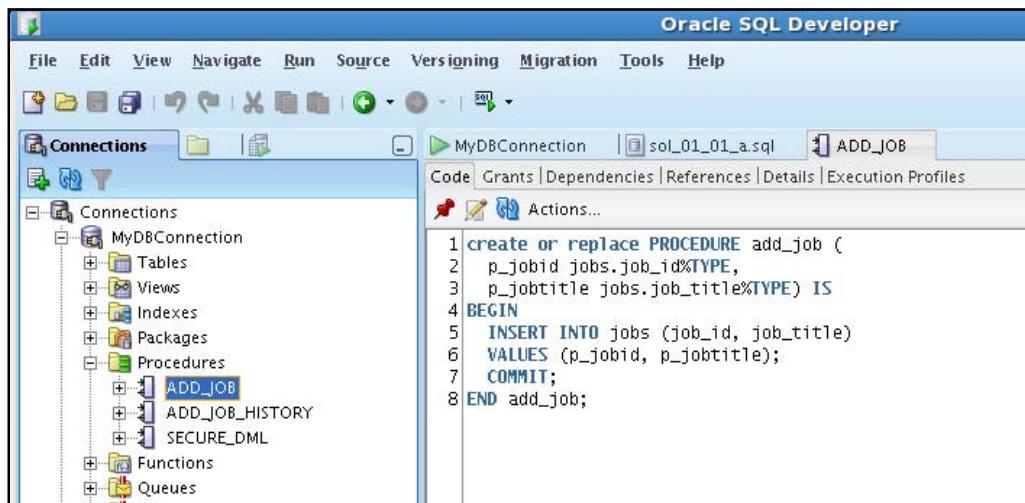
The bottom window shows the results of the compilation:

RESULTS

PROCEDURE add_job Compiled.

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

새로 생성된 프로시저를 보려면 Object Navigator에서 Procedures 노드를 누릅니다. 새로 생성된 프로시저가 표시되지 않으면 Procedures 노드를 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Refresh를 선택합니다. 새 프로시저가 다음과 같이 표시됩니다.



- b) IT_DBA를 직무 ID로 사용하고 Database Administrator를 직위로 사용하여 프로시저를 호출합니다. JOBS 테이블을 query하고 결과를 확인합니다.

/home/oracle/labs/plpu/soln/sol_01_01_b.sql 스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.

```

EXECUTE add_job ('IT_DBA', 'Database Administrator')
SELECT * FROM jobs WHERE job_id = 'IT_DBA';

```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Database Administrator		

1 rows selected

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- c) 프로시저를 다시 호출하여 직무 ID ST_MAN 과 직위 Stock Manager 를 전달합니다. 어떤 결과가 발생하며, 그 이유는 무엇입니까?

JOB_ID 열에 대한 Unique Key 무결성 제약 조건이 있기 때문에 예외가 발생합니다.

The screenshot shows an Oracle SQL Worksheet window titled 'sol_01_01.cspl'. In the main pane, the command 'EXECUTE add_job ('ST_MAN', 'Stock Manager')' is highlighted in yellow. Below the command, an error message is displayed:

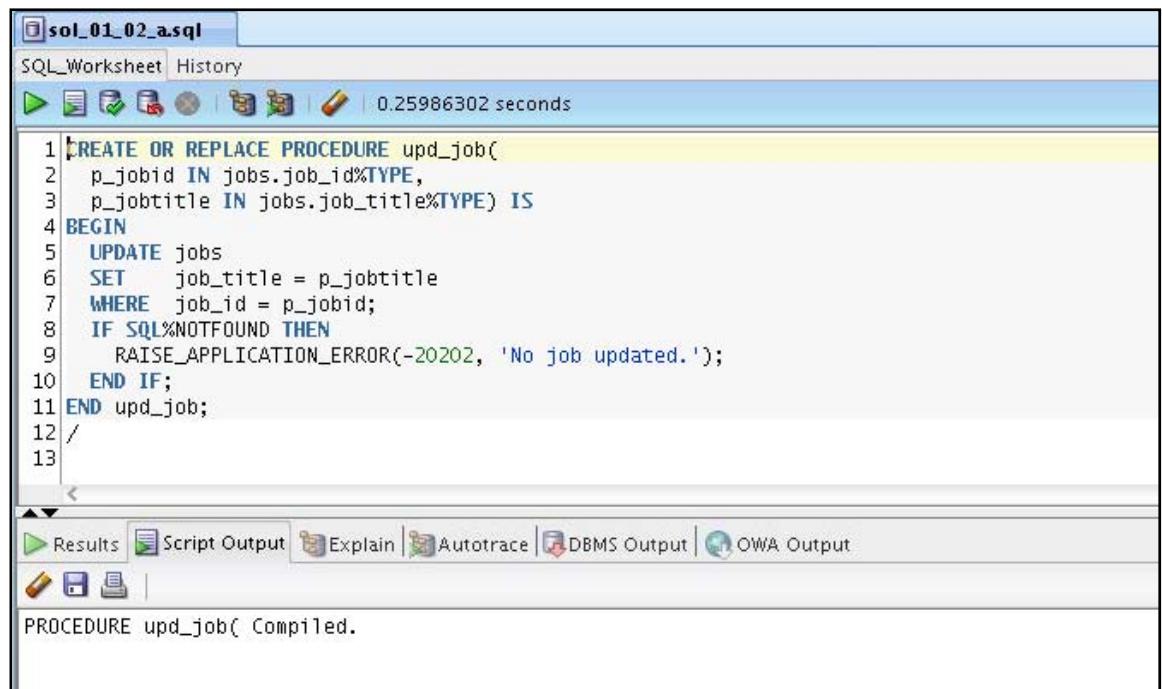
```
Error starting at line 1 in command:
EXECUTE add_job ('ST_MAN', 'Stock Manager')
Error report:
ORA-00001: unique constraint (ORA61.JOB_ID_PK) violated
ORA-06512: at "ORA61.ADD_JOB", line 5
ORA-06512: at line 1
00001. 00000 - "unique constraint (%s.%s) violated"
*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.
        For Trusted Oracle configured in DBMS MAC mode, you may see
        this message if a duplicate entry exists at a different level.
*Action: Either remove the unique restriction or do not insert the key.
```

- 2) JOBS 테이블의 직무를 수정하는 UPD_JOB 이라는 프로시저를 생성합니다.

- a) 직위를 갱신하는 UPD_JOB 이라는 프로시저를 생성합니다. 두 개의 파라미터를 사용하여 직무 ID 와 새로운 직위를 제공합니다. 갱신이 수행되지 않을 경우에 필요한 예외 처리를 추가합니다.

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

/home/oracle/labs/plpu/soln/sol_01_02_a.sql 스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.



```

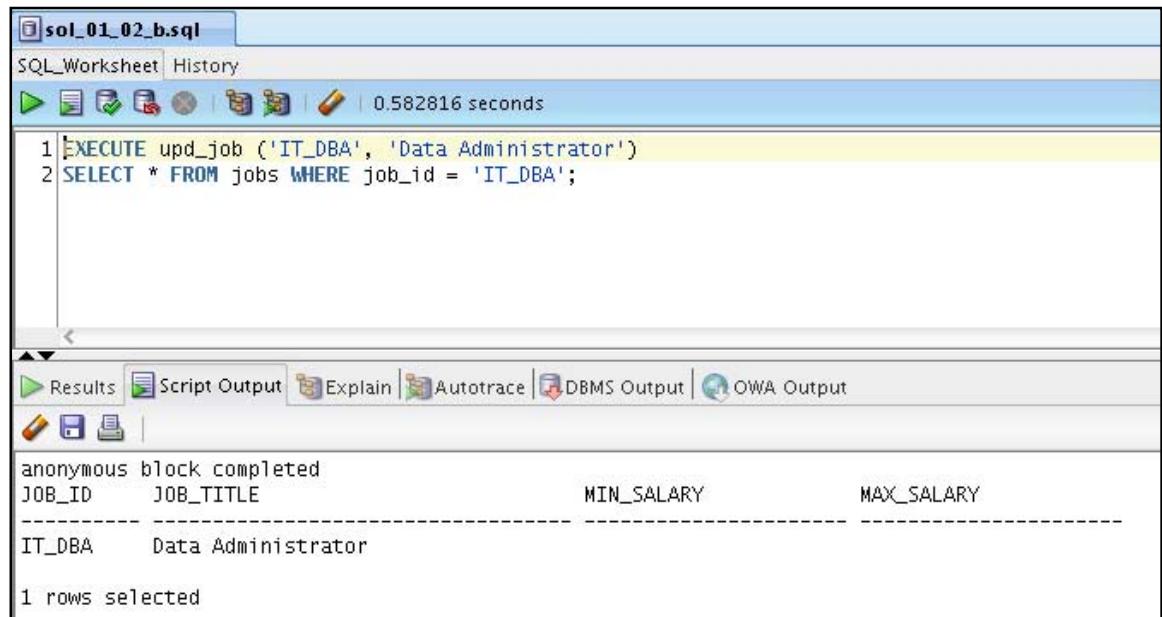
CREATE OR REPLACE PROCEDURE upd_job(
  p_jobid IN jobs.job_id%TYPE,
  p_jobtitle IN jobs.job_title%TYPE) IS
BEGIN
  UPDATE jobs
  SET job_title = p_jobtitle
  WHERE job_id = p_jobid;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202, 'No job updated.');
  END IF;
END upd_job;
/

```

PROCEDURE upd_job(Compiled.

- b) 직무 ID IT_DBA 의 직위를 Data Administrator 로 변경하는 프로시저를 호출합니다. JOBS 테이블을 query 하고 결과를 확인합니다.

/home/oracle/labs/plpu/soln/sol_01_02_b.sql 스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.



```

EXECUTE upd_job ('IT_DBA', 'Data Administrator')
SELECT * FROM jobs WHERE job_id = 'IT_DBA';

```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_DBA	Data Administrator		

1 rows selected

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- c) 존재하지 않는 직무의 갱신을 시도하여 프로시저의 예외 처리 셕션을 테스트합니다. 직무 ID IT_WEB 과 직위 Web Master 를 사용할 수 있습니다.

The screenshot shows an Oracle SQL Worksheet window titled 'sol_01_02_Csql'. The SQL tab is active, displaying the following code:

```
1 EXECUTE upd_job ('IT_WEB', 'Web Master')
2 SELECT * FROM jobs WHERE job_id = 'IT_WEB';
```

The execution time is listed as 0.79948604 seconds. Below the code, the Results tab is selected, showing the following output:

```
Error starting at line 1 in command:
EXECUTE upd_job ('IT_WEB', 'Web Master')
Error report:
ORA-20202: No job updated.
ORA-06512: at "ORA61.UPD_JOB", line 9
ORA-06512: at line 1
```

Below the error message, there is a table output:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
0 rows selected			

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

3) JOBS 테이블에서 직무를 삭제하는 DEL_JOB이라는 프로시저를 생성합니다.

- a) 직무를 삭제하는 DEL_JOB이라는 프로시저를 생성합니다. 직무가 삭제되지 않을 경우에 필요한 예외 처리를 포함합니다.

/home/oracle/labs/plpu/soln/sol_01_03_a.sql 스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Developer interface with a SQL Worksheet tab active. The code area contains the following PL/SQL procedure:

```
1 CREATE OR REPLACE PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
2 BEGIN
3   DELETE FROM jobs
4   WHERE job_id = p_jobid;
5   IF SQL%NOTFOUND THEN
6     RAISE_APPLICATION_ERROR(-20203, 'No jobs deleted.');
7   END IF;
8 END del_job;
9 /
```

Below the code, the results pane shows the message: "PROCEDURE del_job Compiled."

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- b) 프로시저를 호출한 다음 JOBS 테이블을 query 하려면

/home/oracle/labs/plpu/solns 폴더에서 sol_01_03_b.sql 파일을 로드합니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'sol_01_03_b.sql'. The main area contains the following code:

```
1 EXECUTE del_job ('IT_DBA')
2 SELECT * FROM jobs WHERE job_id = 'IT_DBA';|
```

Below the code, the results pane shows the output of the anonymous block:

```
anonymous block completed
JOB_ID      JOB_TITLE          MIN_SALARY      MAX_SALARY
-----  -----
0 rows selected
```

The 'Results' tab is selected at the bottom. Other tabs include 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'.

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- c) 존재하지 않는 직무의 삭제를 시도하여 프로시저의 예외 처리 섹션을 테스트합니다. IT_WEB 을 직무 ID 로 사용합니다. 프로시저의 예외 처리 섹션에 포함했던 메시지가 출력되어야 합니다.

프로시저를 호출한 다음 JOBS 테이블을 query 하려면

/home/oracle/labs/plpu/solns 폴더에서 sol_01_03_c.sql 파일을 로드합니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'sol_01_03_c.sql'. The main area contains the following SQL command:

```
1 EXECUTE del_job ('IT_WEB')
```

Below the command, the results pane displays an error message:

```
Error starting at line 1 in command:  
EXECUTE del_job ('IT_WEB')  
Error report:  
ORA-20203: No jobs deleted.  
ORA-06512: at "ORA61.DEL_JOB", line 6  
ORA-06512: at line 1
```

The interface includes tabs for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. There are also toolbar icons for running scripts, saving, and other functions.

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- 4) 사원 ID를 제공할 경우 사원의 급여와 직무 ID를 검색하는 GET_EMPLOYEE라는 프로시저를 생성하여 EMPLOYEES 테이블을 query 합니다.
- a) 지정된 사원 ID에 대한 SALARY 및 JOB_ID 열에서 값을 반환하는 프로시저를 생성합니다. 구문 오류가 있으면 제거한 다음 코드를 재컴파일합니다.

/home/oracle/labs/plpu/solns/sol_01_04_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

sol_01_04_a.sql
SQL Worksheet History
0.66626805 seconds

CREATE OR REPLACE PROCEDURE get_employee
(p_empid IN employees.employee_id%TYPE,
 p_sal   OUT employees.salary%TYPE,
 p_job   OUT employees.job_id%TYPE) IS
BEGIN
  SELECT salary, job_id
  INTO   p_sal, p_job
  FROM   employees
  WHERE  employee_id = p_empid;
END get_employee;
/

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

PROCEDURE get_employee Compiled.

참고

새로 생성된 프로시저가 Object Navigator에 표시되지 않으면 Object Navigator에서 Procedures 노드를 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Refresh를 선택합니다. Object Navigator에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile을 선택합니다. 프로시저가 컴파일됩니다.

Messages - Log

GET_EMPLOYEE Compiled

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- b) 두 개의 OUT 파라미터 즉, 급여에 대한 파라미터와 직무 ID에 대한 파라미터에 호스트 변수를 사용하여 프로시저를 실행합니다. 사원 ID 120에 대한 급여와 직무 ID를 표시합니다.

/home/oracle/labs/plpu/solns/sol_01_04_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows the Oracle SQL Worksheet interface. The title bar says 'sol_01_04_b.sql'. The main area contains the following PL/SQL code:

```
1 VARIABLE v_salary NUMBER
2 VARIABLE v_job      VARCHAR2(15)
3 EXECUTE get_employee(120, :v_salary, :v_job)
4 PRINT v_salary v_job
5
```

Below the code, the results pane shows the output:

```
anonymous block completed
v_salary
-----
8000

v_job
-----
ST_MAN
```

연습 해답 1-1: 프로시저 생성, 컴파일 및 호출 (계속)

- c) 프로시저를 다시 호출하고 값이 300 인 EMPLOYEE_ID 를 전달합니다. 어떤 결과가 발생하며, 그 이유는 무엇입니까?

EMPLOYEES 테이블에는 EMPLOYEE_ID 가 300 인 사원이 없습니다.

SELECT 문이 데이터베이스에서 데이터를 읽어 들일 수 없으므로 다음과 같이 치명적인 PL/SQL 오류 NO_DATA_FOUND 가 발생합니다.

The screenshot shows the Oracle SQL Developer interface. The top window is titled "SQL Worksheet" and contains the following PL/SQL code:

```
1 VARIABLE v_salary NUMBER
2 VARIABLE v_job    VARCHAR2(15)
3 EXECUTE get_employee(300, :v_salary, :v_job)
```

The bottom window is titled "Results" and displays the output of the executed anonymous block. It shows the values assigned to the variables and then an error report:

```
anonymous block completed
v_salary
-----
8000

v_job
-----
ST_MAN

Error starting at line 1 in command:
EXECUTE get_employee(300, :v_salary, :v_job)
Error report:
ORA-01403: no data found
ORA-06512: at "ORA61.GET_EMPLOYEE", line 6
ORA-06512: at line 1
01403. 00000 -  "no data found"
*Cause:
*Action:
```

단원 2 의 연습 및 해답

연습 2-1 에서는 다음을 생성, 컴파일 및 사용합니다.

- 직위를 반환하는 GET_JOB 함수
- 파라미터로 전달된 사원의 월급과 커미션에서 계산된 연봉을 반환하는 GET_ANNUAL_COMP 함수
- EMPLOYEES 테이블에 새로운 사원을 삽입하는 ADD_EMPLOYEE 프로시저

연습 2-2 에서는 SQL Developer 디버거의 기본 기능에 대해 소개합니다.

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

참고: 연습에서 겪너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 2-1: 함수 생성

이 연습에서는 내장 함수와 프로시저를 생성, 컴파일 및 사용합니다.

- 1) 직위를 반환하는 GET_JOB 함수를 생성하여 호출합니다.
 - a) 직위를 반환하는 GET_JOB 함수를 생성하여 컴파일합니다.
 - b) 최대 35 자가 허용되는 b_title이라는 VARCHAR2 호스트 변수를 생성합니다. 직무 ID가 SA_REP 인 함수를 호출하여 값을 호스트 변수에 반환하고 호스트 변수를 인쇄하여 결과를 봅니다.

```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
b_title
-----
Sales Representative
  
```

- 2) 파라미터로 전달된 사원의 월급과 커미션에서 계산된 연봉을 반환하는 GET_ANNUAL_COMP라는 함수를 생성합니다.
 - a) 월급 및 커미션의 파라미터 값을 받아들이는 GET_ANNUAL_COMP 함수를 생성합니다. 두 값 중 하나 또는 둘 다 NULL 일 수는 있지만 함수가 반환하는 연봉은 NULL 이 아니어야 합니다. 다음과 같은 기본 공식을 사용하여 연봉을 계산합니다.

$$(salary * 12) + (commission_pct * salary * 12)$$
 - b) EMPLOYEES 테이블에 대한 SELECT 문에서 부서 30에 근무하는 사원에 대해 이 함수를 사용합니다.

EMPLOYEE_ID	LAST_NAME	Annual Compensation
114	Raphaely	132000
115	Khoo	37200
116	Baida	34800
117	Tobias	33600
118	Himuro	31200
119	Colmenares	30000

6 rows selected

연습 2-1: 함수 생성(계속)

- 3) EMPLOYEES 테이블에 새로운 사원을 삽입하는 ADD_EMPLOYEE라는 프로시저를 생성합니다. 이 프로시저는 VALID_DEPTID 함수를 호출하여 새로운 사원에게 지정된 부서 ID 가 DEPARTMENTS 테이블에 있는지 여부를 확인해야 합니다.
- 지정된 부서 ID 를 검증한 다음 해당 부서가 존재할 경우 BOOLEAN 값 TRUE 를 반환하는 VALID_DEPTID 함수를 생성합니다.
 - EMPLOYEES 테이블에 사원을 추가하는 ADD_EMPLOYEE 프로시저를 생성합니다. VALID_DEPTID 함수가 TRUE 를 반환할 경우 EMPLOYEES 테이블에 행이 추가되어야 하며, 그렇지 않을 경우 적절한 메시지를 표시하여 유저에게 경고해야 합니다. 다음 파라미터를 입력합니다.
 - first_name
 - last_name
 - email
 - job: 'SA_REP' 를 기본값으로 사용합니다.
 - mgr: 145 를 기본값으로 사용합니다.
 - sal: 1000 을 기본값으로 사용합니다.
 - comm: 0 을 기본값으로 사용합니다.
 - dept_id: 30 을 기본값으로 사용합니다.
 - EMPLOYEES_SEQ 시퀀스를 사용하여 employee_id 열을 설정합니다.
 - hire_date 열을 TRUNC(SYSDATE)로 설정합니다.
 - 부서 15 에 근무하는 'Jane Harris' 에 대해 ADD_EMPLOYEE 를 호출하되, 다른 파라미터는 기본값을 그대로 적용합니다. 어떤 결과가 발생합니까?
 - 부서 80 에 Joe Harris 라는 또 다른 사원을 추가하되, 나머지 파라미터는 기본값을 그대로 적용합니다. 어떤 결과가 발생합니까?

연습 2-2: SQL Developer 디버거 소개

이 연습에서는 SQL Developer 디버거의 기본 기능을 실행해 봅니다.

- 1) SERVEROUTPUT 을 활성화합니다.
- 2) sol_02_02_02.sql 스크립트를 실행하여 emp_list 프로시저를 생성합니다. 프로시저의 코드를 검사하고 프로시저를 컴파일합니다. 왜 컴파일 오류가 발생합니까?
- 3) sol_02_02_03.sql 스크립트를 실행하여 get_location 함수를 생성합니다. 함수의 코드를 검사하고 함수를 컴파일한 후 오류가 있는 경우 수정합니다.
- 4) emp_list 프로시저를 재컴파일합니다. 프로시저가 성공적으로 컴파일되어야 합니다.
- 5) emp_list 프로시저 및 get_location 함수를 편집합니다.
- 6) 코드의 다음 행에 emp_list 프로시저에 대한 중단점을 네 개 추가합니다.
 - a) OPEN emp_cursor;
 - b) WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
 - c) v_city := get_location (emp_record.department_name);
 - d) CLOSE emp_cursor;
- 7) 디버깅을 위해 emp_list 프로시저를 컴파일합니다.
- 8) 프로시저를 디버깅합니다.
- 9) PMAXROWS 파라미터 값으로 100을 입력합니다.
- 10) Data 탭에서 변수 값을 검사합니다. REC_EMP 및 EMP_TAB에 지정된 값은 무엇입니까? 그 이유는 무엇입니까?
- 11) Step Into 디버그 옵션을 사용하여 emp_list 의 각 코드 행을 Step Into 하고 while 루프를 한 번만 진행합니다.
- 12) Data 탭에서 변수 값을 검사합니다. REC_EMP에 지정된 값은 무엇입니까?
- 13) emp_tab(i) := rec_emp; 행이 실행될 때까지 F7 키를 계속 누릅니다. Data 탭에서 변수 값을 검사합니다. EMP_TAB에 지정된 값은 무엇입니까?
- 14) Data 탭을 사용하여 카운터 i의 값을 98로 수정합니다.
- 15) F7 키를 계속 눌러 Debugging - Log 탭에 표시된 사원 리스트를 확인합니다. 표시되는 사원은 몇 명입니까?
- 16) Step Over 디버거 옵션을 사용하여 코드를 진행하는 경우 get_location 함수를 진행합니까? 그 이유는 무엇입니까?

연습 해답 2-1: 함수 생성

이 연습에서는 내장 함수와 프로시저를 생성, 컴파일 및 사용합니다.

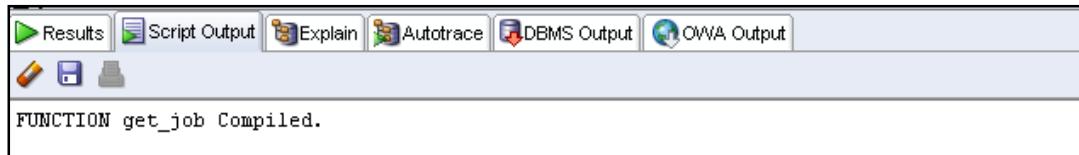
- 직위를 반환하는 GET_JOB 함수를 생성하여 호출합니다.

- 직위를 반환하는 GET_JOB 함수를 생성하여 컴파일합니다.

```
/home/oracle/labs/plpu/solns/sol_02_01_01_a.sql
```

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 함수를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE FUNCTION get_job (p_jobid IN
jobs.job_id%type)
RETURN jobs.job_title%type IS
v_title jobs.job_title%type;
BEGIN
SELECT job_title
INTO v_title
FROM jobs
WHERE job_id = p_jobid;
RETURN v_title;
END get_job;
/
```



연습 해답 2-1: 함수 생성 (계속)

- b) 최대 35 자가 허용되는 b_title 이라는 VARCHAR2 호스트 변수를 생성합니다. 직무 ID 가 SA_REP 인 함수를 호출하여 값을 호스트 변수에 반환하고 호스트 변수를 인쇄하여 결과를 봅니다.

`/home/oracle/labs/plpu/solns/sol_02_01_01_b.sql`
스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 함수를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
VARIABLE b_title VARCHAR2(35)
EXECUTE :b_title := get_job ('SA_REP');
PRINT b_title
```

```
anonymous block completed
b_title
-----
Sales Representative
```

- 2) 파라미터로 전달된 사원의 월급과 커미션에서 계산된 연봉을 반환하는 GET_ANNUAL_COMP 라는 함수를 생성합니다.

- a) 월급 및 커미션의 파라미터 값을 받아들이는 GET_ANNUAL_COMP 함수를 생성합니다. 두 값 중 하나 또는 둘 다 NULL 일 수는 있지만 함수가 반환하는 연봉은 NULL 이 아니어야 합니다. 다음과 같은 기본 공식을 사용하여 연봉을 계산합니다.

$$(\text{salary} * 12) + (\text{commission_pct} * \text{salary} * 12)$$

`/home/oracle/labs/plpu/solns/sol_02_01_02_a.sql`
스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 함수를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE FUNCTION get_annual_comp(
    p_sal IN employees.salary%TYPE,
    p_comm IN employees.commission_pct%TYPE)
RETURN NUMBER IS
BEGIN
    RETURN (NVL(p_sal,0) * 12 + (NVL(p_comm,0) *
nvl(p_sal,0) * 12));
END get_annual_comp;
/
```

```
FUNCTION get_annual_comp( Compiled.
```

연습 해답 2-1: 함수 생성(계속)

- b) EMPLOYEES 테이블에 대한 SELECT 문에서 부서 30에 근무하는 사원에 대해 이 함수를 사용합니다.

/home/oracle/labs/plpu/solns/sol_02_01_02_b.sql

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 함수를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT employee_id, last_name,
       get_annual_comp(salary,commission_pct) "Annual
Compensation"
FROM   employees
WHERE  department_id=30
/
```

EMPLOYEE_ID	LAST_NAME	Annual Compensation
114	Raphaely	132000
115	Khoo	37200
116	Baida	34800
117	Tobias	33600
118	Himuro	31200
119	Colmenares	30000

6 rows selected

- 3) EMPLOYEES 테이블에 새로운 사원을 삽입하는 ADD_EMPLOYEE라는 프로시저를 생성합니다. 이 프로시저는 VALID_DEPTID 함수를 호출하여 새로운 사원에게 지정된 부서 ID 가 DEPARTMENTS 테이블에 있는지 여부를 확인해야 합니다.

연습 해답 2-1: 함수 생성(계속)

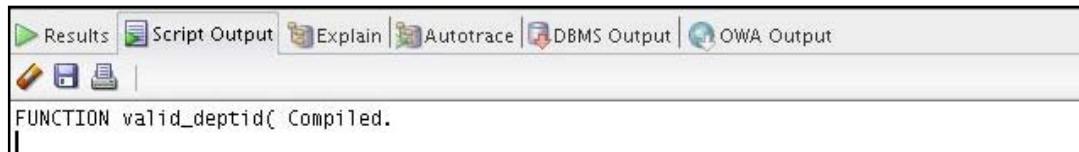
- a) 지정된 부서 ID를 검증한 다음 해당 부서가 존재할 경우 BOOLEAN 값 TRUE를 반환하는 VALID_DEPTID 함수를 생성합니다.

/home/oracle/labs/plpu/solns/sol_02_01_03_a.sql

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 함수를 생성합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE FUNCTION valid_deptid(
    p_deptid IN departments.department_id%TYPE)
RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;

BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;
/
```



- b) EMPLOYEES 테이블에 사원을 추가하는 ADD_EMPLOYEE 프로시저를 생성합니다. VALID_DEPTID 함수가 TRUE를 반환할 경우 EMPLOYEES 테이블에 행이 추가되어야 하며, 그렇지 않을 경우 적절한 메시지를 표시하여 유저에게 경고해야 합니다. 다음 파라미터를 입력합니다.

- first_name
- last_name
- email
- job: 'SA_REP'를 기본값으로 사용합니다.
- mgr: 145를 기본값으로 사용합니다.
- sal: 1000을 기본값으로 사용합니다.
- comm: 0을 기본값으로 사용합니다.

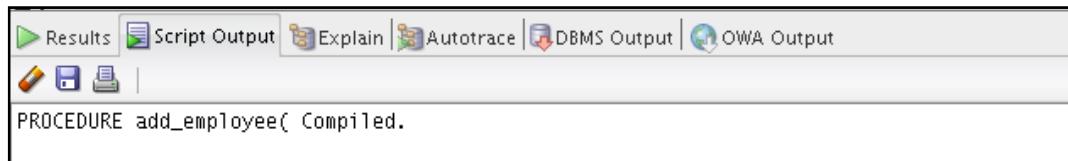
연습 해답 2-1: 함수 생성(계속)

- deptid: 30 을 기본값으로 사용합니다.
- EMPLOYEES_SEQ 시퀀스를 사용하여 employee_id 열을 설정합니다.
- hire_date 열을 TRUNC(SYSDATE)로 설정합니다.

/home/oracle/labs/plpu/solns/sol_02_01_03_b.sql

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name  employees.last_name%TYPE,
    p_email      employees.email%TYPE,
    p_job        employees.job_id%TYPE          DEFAULT
    'SA_REP',
    p_mgr        employees.manager_id%TYPE      DEFAULT 145,
    p_sal        employees.salary%TYPE          DEFAULT 1000,
    p_comm       employees.commission_pct%TYPE   DEFAULT 0,
    p_deptid     employees.department_id%TYPE   DEFAULT 30)
IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name,
last_name, email,
        job_id, manager_id, hire_date, salary,
commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
        p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
    END IF;
END add_employee;
/
```



연습 해답 2-1: 함수 생성(계속)

- c) 부서 15에 근무하는 'Jane Harris'에 대해 ADD_EMPLOYEE를 호출하되, 다른 파라미터는 기본값을 그대로 적용합니다. 어떤 결과가 발생합니까?

`/home/oracle/labs/plpu/solns/sol_02_01_03_c.sql`

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE add_employee ('Jane', 'Harris', 'JAHARRIS', p_deptid=> 15)
```

The screenshot shows the Oracle SQL Worksheet interface with the 'Results' tab selected. The output window displays the following error message:

```
Error starting at line 1 in command:
EXECUTE add_employee('Jane', 'Harris', 'JAHARRIS', p_deptid=> 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.ADD_EMPLOYEE", line 17
ORA-06512: at line 1
```

- d) 부서 80에 Joe Harris라는 또 다른 사원을 추가하되, 나머지 파라미터는 기본값을 그대로 적용합니다. 어떤 결과가 발생합니까?

`/home/oracle/labs/plpu/solns/sol_02_01_03_d.sql`

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE add_employee('Joe', 'Harris', 'JAHARRIS',
p_deptid=> 80)
```

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

```
anonymous block completed
```

연습 해답 2-2: SQL Developer 디버거 소개

이 연습에서는 SQL Developer 디버거의 기본 기능을 실행해 봅니다.

- 1) SERVEROUTPUT 을 활성화합니다.

SQL Worksheet 영역에 다음 명령을 입력한 다음 **Run Script (F5)**를 누릅니다.
SQL Worksheet 도구 모음에서 아이콘을 누릅니다.

```
SET SERVEROUTPUT ON
```

- 2) sol_02_02_02.sql 스크립트를 실행하여 emp_list 프로시저를 생성합니다. 프로시저의 코드를 검사하고 프로시저를 컴파일합니다. 왜 컴파일 오류가 발생합니까?

/home/oracle/labs/plpu/sols/sol_02_02_02.sql 스크립트를 엽니다. **SQL Worksheet** 도구 모음에서 **Run Script (F5)** 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. codex 및 결과가 다음과 같이 표시됩니다.

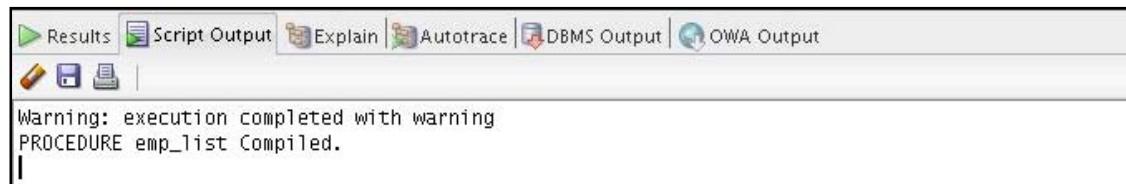
```
CREATE OR REPLACE PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur_emp IS
    SELECT d.department_name, e.employee_id, e.last_name,
           e.salary, e.commission_pct
      FROM departments d, employees e
     WHERE d.department_id = e.department_id;
rec_emp cur_emp%ROWTYPE;
TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY
BINARY_INTEGER;
emp_tab emp_tab_type;
i NUMBER := 1;
v_city VARCHAR2(30);
BEGIN
    OPEN cur_emp;
    FETCH cur_emp INTO rec_emp;
    emp_tab(i) := rec_emp;
    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
        i := i + 1;
        FETCH cur_emp INTO rec_emp;
        emp_tab(i) := rec_emp;
        v_city := get_location (rec_emp.department_name);
        dbms_output.put_line('Employee ' || rec_emp.last_name ||
```

연습 해답 2-2: SQL Developer 퍼거 소개 (계속)

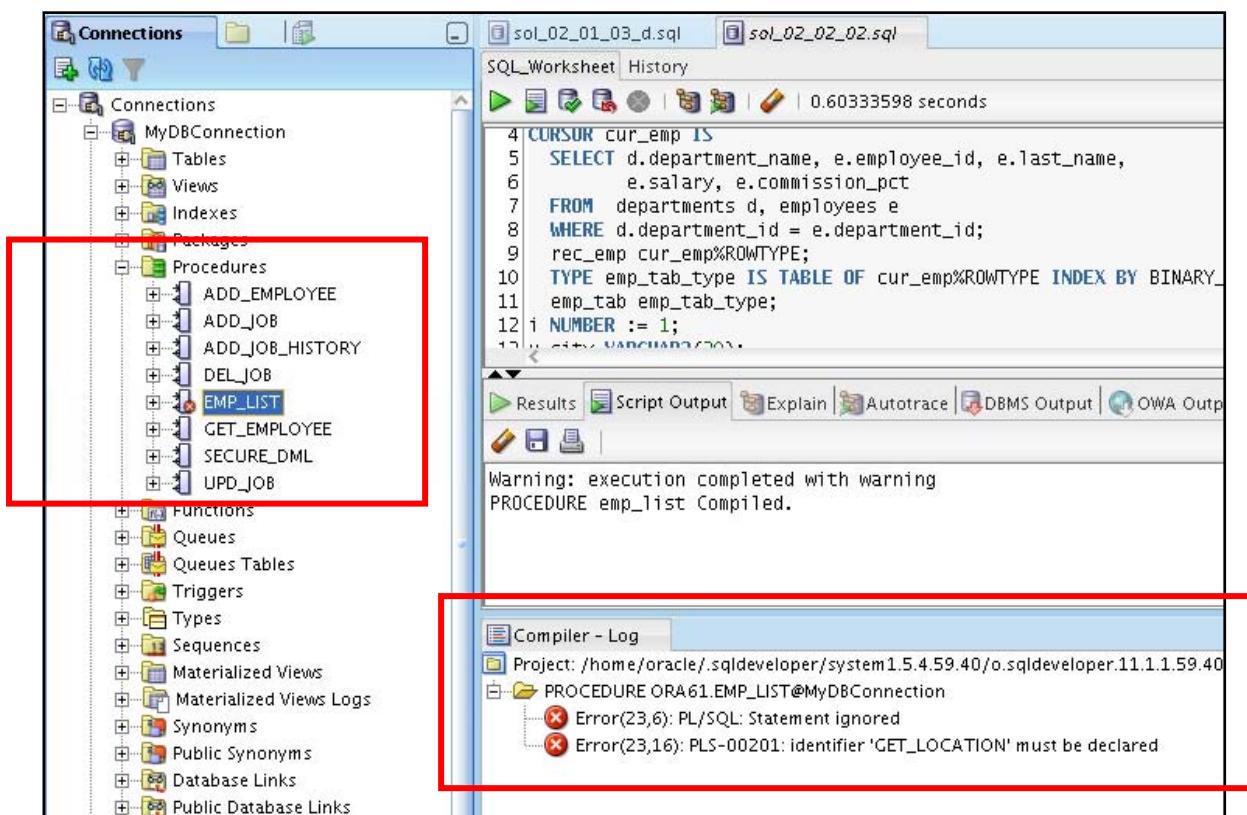
```

        ' works in ' || v_city );
END LOOP;
CLOSE cur_emp;
FOR j IN REVERSE 1..i LOOP
    DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
END LOOP;
END emp_list;
/

```



`get_location` 함수가 아직 선언되지 않았기 때문에 컴파일 경고가 표시됩니다. 컴파일 오류를 자세히 표시하려면 Procedures 노드에서 `EMP_LIST` 프로시저를 마우스 오른쪽 버튼으로 누르고(새로 생성된 `EMP_LIST` 프로시저를 보려면 프로시저 리스트를 Refresh 해야 할 수 있음) 팝업 메뉴에서 Compile을 선택합니다. 다음과 같이 자세한 경고 메시지가 Compiler-Log 탭에 표시됩니다.

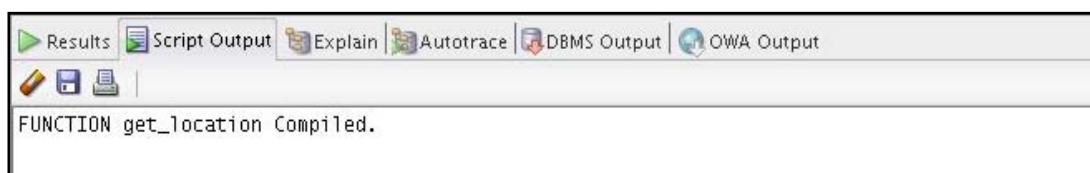


연습 해답 2-2: SQL Developer 툴 소개 (계속)

- 3) sol_02_02_03.sql 스크립트를 실행하여 get_location 함수를 생성합니다. 함수의 코드를 검사하고 함수를 컴파일한 후 오류가 있는 경우 수정합니다.

/home/oracle/labs/plpu/sols/sol_02_02_03.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. codex 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE FUNCTION get_location
( p_deptname IN VARCHAR2 ) RETURN VARCHAR2
AS
    v_loc_id NUMBER;
    v_city    VARCHAR2( 30 );
BEGIN
    SELECT d.location_id, l.city INTO v_loc_id, v_city
    FROM departments d, locations l
    WHERE upper(department_name) = upper(p_deptname)
    and d.location_id = l.location_id;
    RETURN v_city;
END GET_LOCATION;
/
```



- 4) emp_list 프로시저를 재컴파일합니다. 프로시저가 성공적으로 컴파일되어야 합니다.

프로시저를 재컴파일하려면 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile 을 선택합니다.



- 5) emp_list 프로시저 및 get_location 함수를 편집합니다.
- Object Navigator에서 emp_list 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 Edit 를 선택합니다. emp_list 프로시저가 편집 모드로 열립니다. 프로시저가 SQL Worksheet 영역에 이미 표시되어 있지만 읽기 전용 모드인 경우 Code 탭에서 Edit 아이콘(연필 아이콘)을 누릅니다.
- Object Navigator에서 get_location 함수 이름을 마우스 오른쪽 버튼으로 누르고 Edit 를 선택합니다. get_location 함수가 편집 모드로 열립니다. 함수가 SQL Worksheet 영역에 이미 표시되어 있지만 읽기 전용 모드인 경우 Code 탭에서 Edit 아이콘(연필 아이콘)을 누릅니다.

연습 해답 2-2: SQL Developer 편집기 소개 (계속)

6) 코드의 다음 행에 emp_list 프로시저에 대한 중단점을 네 개 추가합니다.

- OPEN emp_cursor;
- WHILE (emp_cursor%FOUND) AND (i <= pMaxRows) LOOP
- v_city := get_location (emp_record.department_name);
- CLOSE emp_cursor;

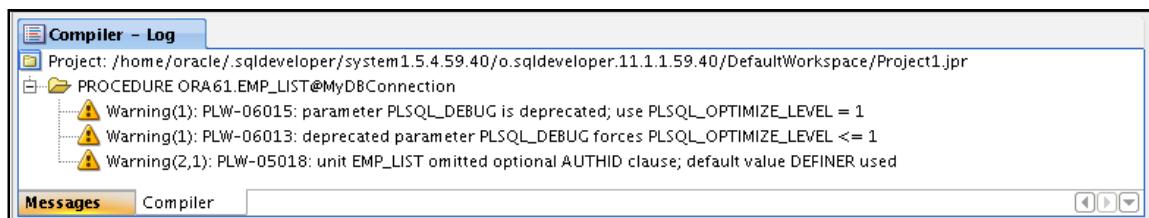
중단점을 추가하려면 아래와 같이 위에 나열된 각 행의 옆에 있는 Line Gutter를 누릅니다.

```
create or replace
PROCEDURE emp_list
(p_maxrows IN NUMBER)
IS
CURSOR cur_emp IS
    SELECT d.department_name, e.employee_id, e.last_name,
           e.salary, e.commission_pct
      FROM departments d, employees e
     WHERE d.department_id = e.department_id;
rec_emp cur_emp%ROWTYPE;
TYPE emp_tab_type IS TABLE OF cur_emp%ROWTYPE INDEX BY BINARY_INTEGER;
emp_tab emp_tab_type;
i NUMBER := 1;
v_city VARCHAR2(30);
BEGIN
    OPEN cur_emp;
    FETCH cur_emp INTO rec_emp;
    emp_tab(i) := rec_emp;
    WHILE (cur_emp%FOUND) AND (i <= p_maxrows) LOOP
        i := i + 1;
        FETCH cur_emp INTO rec_emp;
        emp_tab(i) := rec_emp;
        v_city := get_location (rec_emp.department_name);
        dbms_output.put_line('Employee ' || rec_emp.last_name ||
                           ' works in ' || v_city );
    END LOOP;
    CLOSE cur_emp;
    FOR j IN REVERSE 1..i LOOP
        DBMS_OUTPUT.PUT_LINE(emp_tab(j).last_name);
    END LOOP;
END emp_list;
```

연습 해답 2-2: SQL Developer 디버거 소개 (계속)

7) 디버깅을 위해 emp_list 프로시저를 컴파일합니다.

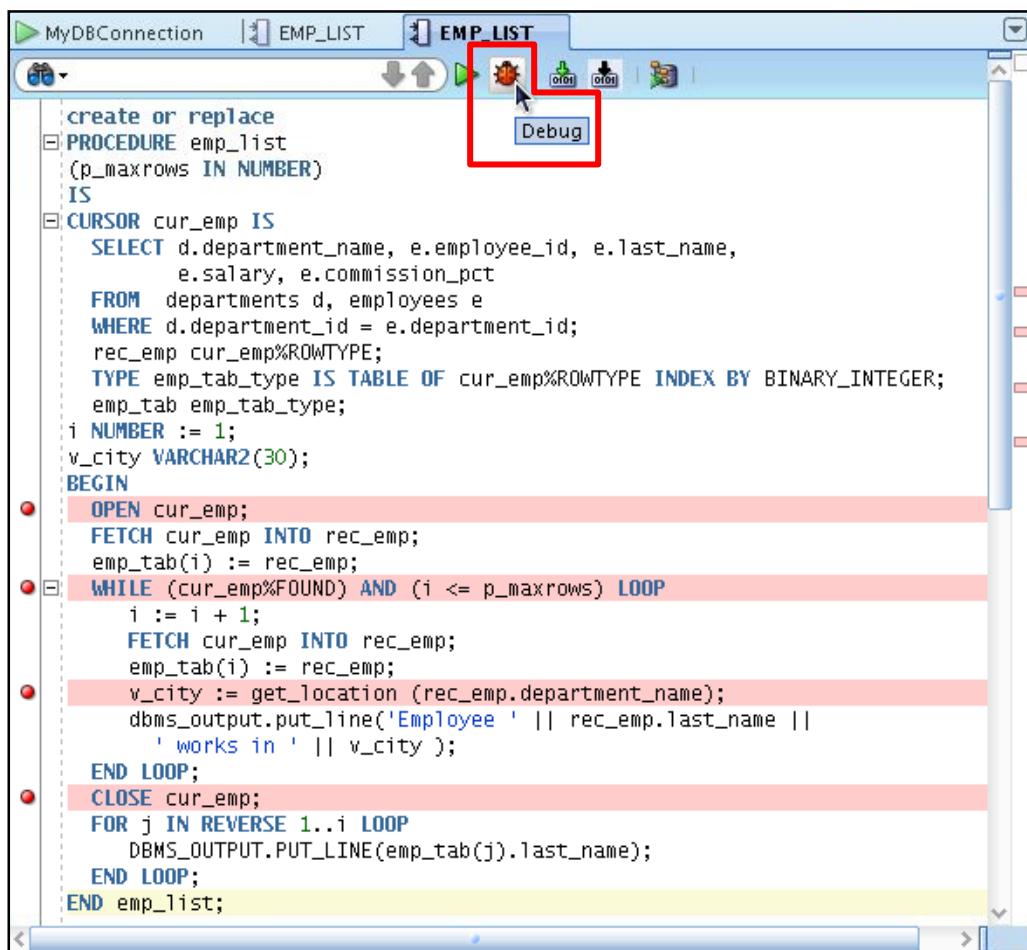
아래와 같이 프로시저의 도구 모음에서 "Compile for Debug" 아이콘을 누릅니다.



참고: 위의 경고가 표시되는 경우 이는 예상된 상황입니다. 처음 두 경고는 Oracle Database 11g에서 PLSQL_DEBUG 파라미터 지원이 중단되었지만 SQL Developer에서 해당 파라미터를 계속 사용하기 때문에 표시됩니다. 마지막 경고는 프로시저에서 AUTHID 절을 사용하는 것과 관련이 있습니다. 이 절은 나중에 다루겠습니다.

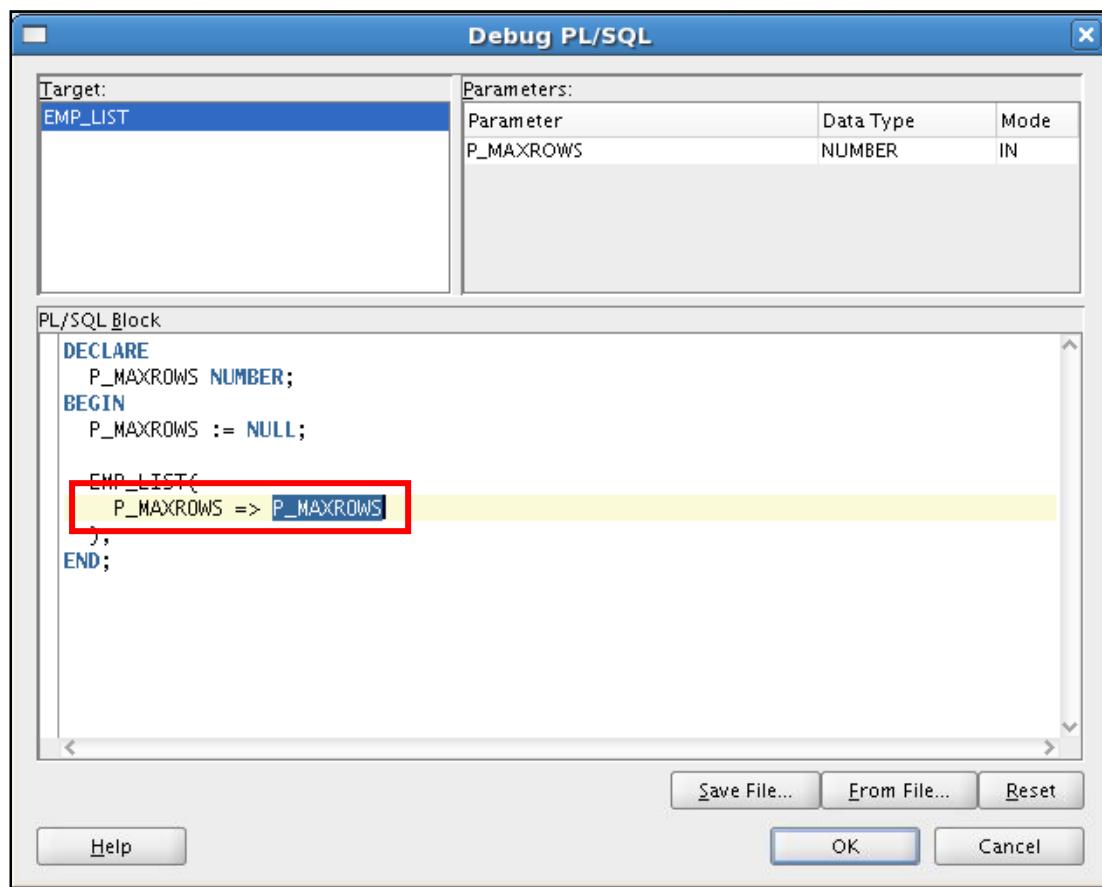
8) 프로시저를 디버깅합니다.

아래와 같이 프로시저의 도구 모음에서 Debug 아이콘을 누릅니다.



연습 해답 2-2: SQL Developer 디버거 소개 (계속)

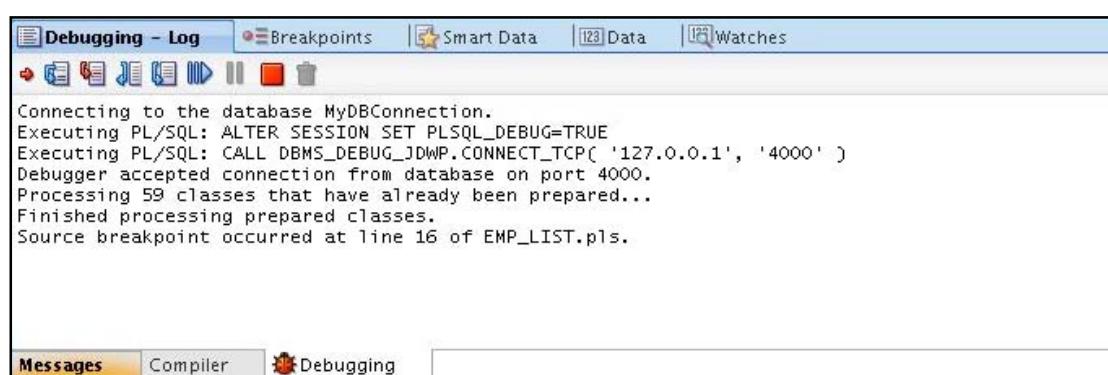
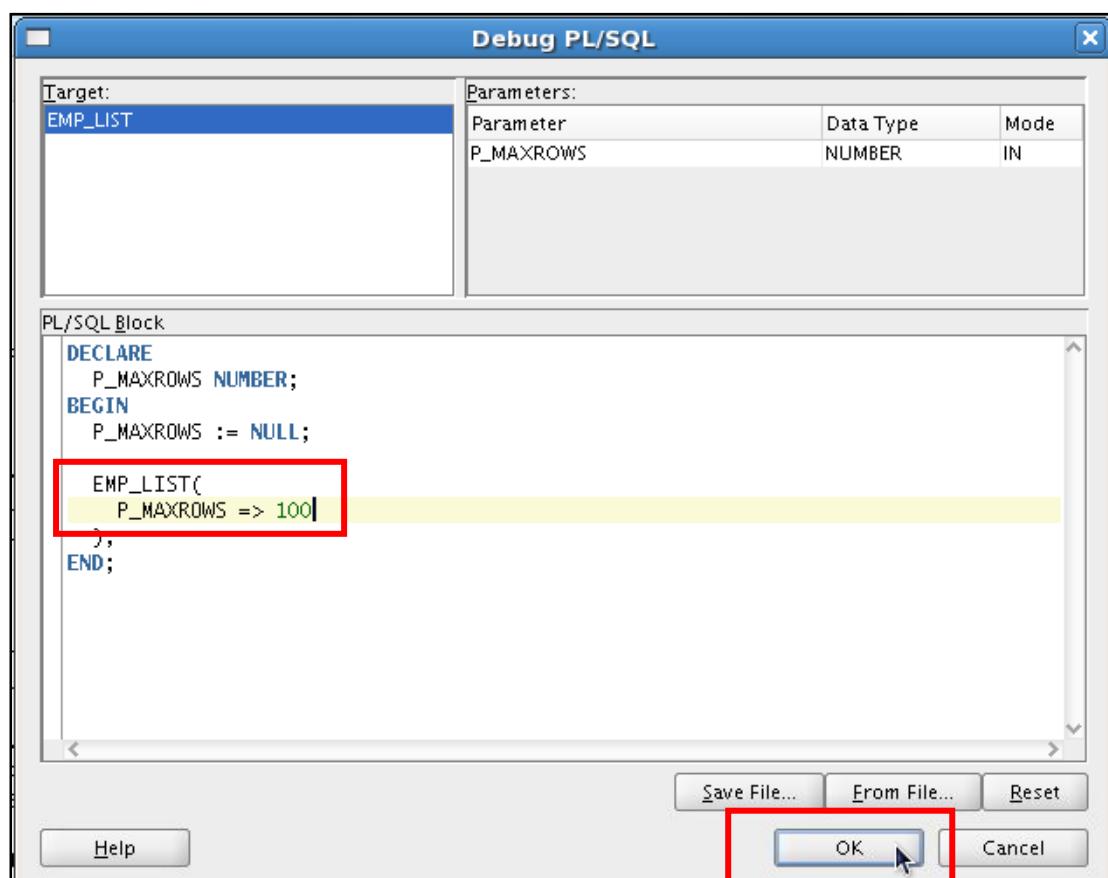
Debug PL/SQL window 가 다음과 같이 표시됩니다.



연습 해답 2-2: SQL Developer 디버거 소개 (계속)

- 9) P_MAXROWS 파라미터 값으로 100 을 입력합니다.

두번째 P_MAXROWS 를 100 으로 바꾼 다음 OK 를 누릅니다. 프로그램 제어가 프로시저에서 해당 코드 행을 가리키는 빨강색 화살표와 파랑색의 강조 색으로 표시된 첫번째 중단점에서 어떻게 정지하는지 확인합니다. 추가 디버깅 템이 페이지 아래쪽에 표시됩니다.



연습 해답 2-2: SQL Developer 디버거 소개 (계속)

- 10) Data 탭에서 변수 값을 검사합니다. REC_EMP 및 EMP_TAB에 지정된 값은 무엇입니까? 그 이유는 무엇입니까?

데이터가 커서에 아직 패치(fetch)되지 않았기 때문에 두 값이 모두 NULL로 설정됩니다.

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 element[0]
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

- 11) Step Into 디버그 옵션을 사용하여 emp_list의 각 코드 행을 Step Into하고 while 루프를 한 번만 진행합니다.

[F7] 키를 눌러 코드를 한 번만 Step Into 합니다.

- 12) Data 탭에서 변수 값을 검사합니다. REC_EMP 및 EMP_TAB에 지정된 값은 무엇입니까?

`FETCH cur_emp INTO rec_emp;` 행이 실행되면 `rec_emp` 가 아래와 같이 초기화됩니다.

Name	Value	Type
P_MAXROWS	100	NUMBER
REC_EMP		Rowtype
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whalen'	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
/values		EMP_TAB_TYPE element[0]
I	1	NUMBER
V_CITY	NULL	VARCHAR2(30)

연습 해답 2-2: SQL Developer 디버거 소개 (계속)

13) `emp_tab(i) := rec_emp;` 행이 실행될 때까지 F7 키를 계속 누릅니다.

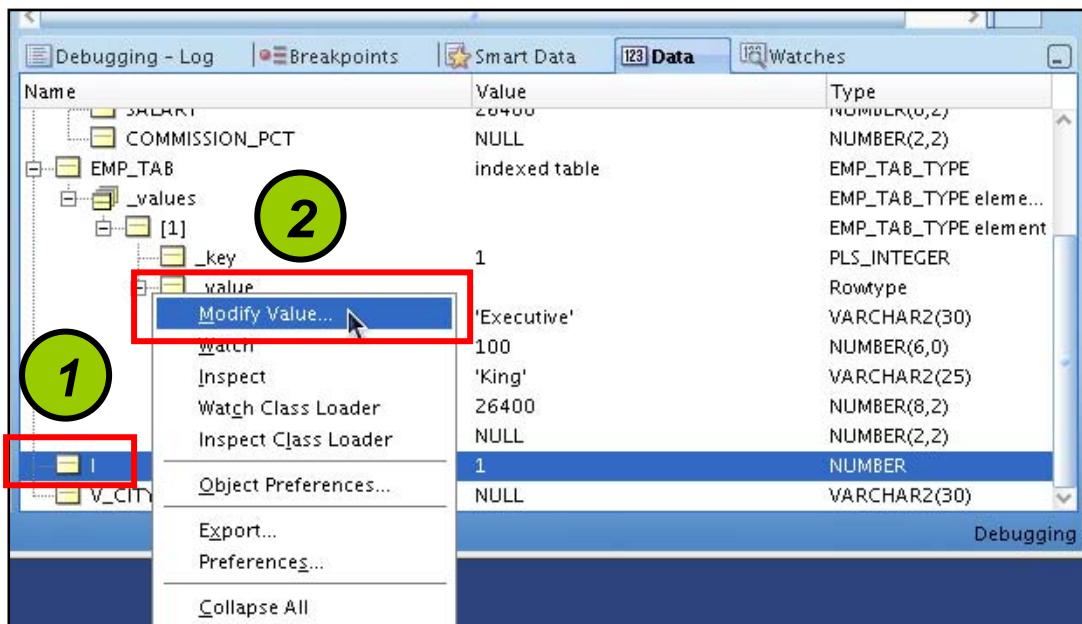
Data 탭에서 변수 값을 검사합니다. EMP_TAB에 지정된 값은 무엇입니까?

`emp_tab(i) := rec_emp;` 행이 실행되면 `emp_tab`이 아래와 같이 `rec_emp`로 초기화됩니다.

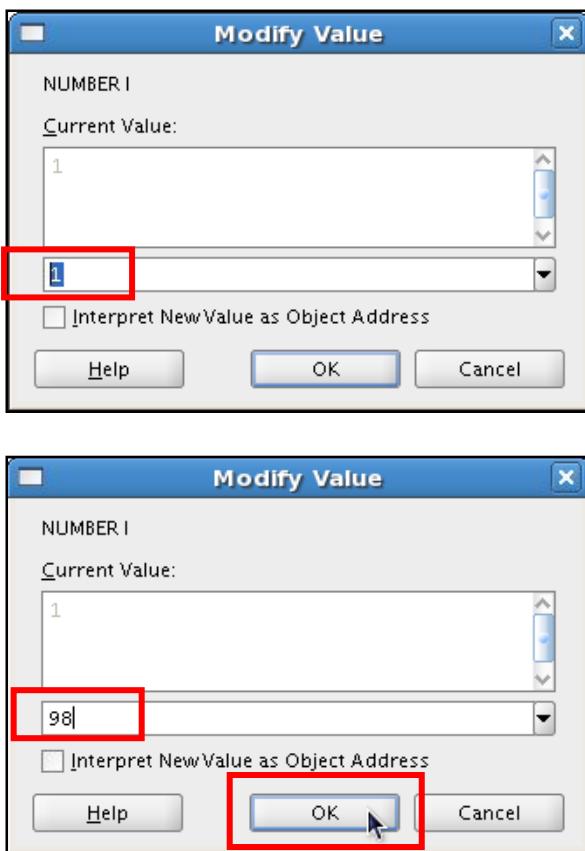
Name	Value	Type
J_DEPART	7400	NUMBER(5,2)
COMMISSION_PCT	NULL	NUMBER(2,2)
EMP_TAB	indexed table	EMP_TAB_TYPE
/values		EMP_TAB_TYPE element[1]
[1]		EMP_TAB_TYPE element
/key	1	PLS_INTEGER
/_value		Rowtype
DEPARTMENT_NAME	'Administration'	VARCHAR2(30)
EMPLOYEE_ID	200	NUMBER(6,0)
LAST_NAME	'Whalen'	VARCHAR2(25)
SALARY	4400	NUMBER(8,2)
COMMISSION_PCT	NULL	NUMBER(2,2)

14) Data 탭을 사용하여 카운터 i의 값을 98로 수정합니다.

Data 탭에서 I를 마우스 오른쪽 버튼으로 누른 다음 단축 메뉴에서 `Modify Value`를 선택합니다. `Modify Value window`가 표시됩니다. 아래와 같이 텍스트 상자에서 값 1을 98로 바꾸고 OK를 누릅니다.



연습 해답 2-2: SQL Developer 디버거 소개 (계속)



- 15) F7 키를 계속 눌러 Debugging – Log 탭에 표시된 사원 리스트를 확인합니다.
표시되는 사원은 몇 명입니까?

디버깅 세션의 끝에 출력이 표시되고 그 아래에 3명의 사원이 표시됩니다.

```
Debugging - Log
Exception breakpoint occurred at line 29 of EMP_LIST.pls.
$0racle.EXCEPTION_ORA_1403:
ORA-01403: no data found
ORA-06512: at "0RA61.EMP_LIST", line 28
ORA-06512: at line 6
Executing PL/SQL: CALL DBMS_DEBUG_JDWP.DISCONNECT()
Employee Fay works in Toronto
Employee Hartstein works in Toronto
Employee Colmenares works in Seattle
Colmenares
Hartstein
Fay
Process exited.
Disconnecting from the database MyDBConnection.
Debugger disconnected from database.

Messages Compiler Debugging
```

- 16) Step Over 디버거 옵션을 사용하여 코드를 진행하는 경우 get_location 함수를 진행합니까? 그 이유는 무엇입니까?

세번째 중단점이 설정된 코드 행에 get_location 함수 호출이 포함되어 있더라도 Step Over (F8)는 코드 행을 실행하고 반환되는 함수 값([F7] 키와 동일)을 검색합니다. 그러나 컨트롤은 get_location 함수로 전달되지 않습니다.

단원 3 의 연습 및 해답

이 연습에서는 ADD_JOB, UPD_JOB 및 DEL_JOB 프로시저의 복사본과 GET_JOB 함수가 포함된 JOB_PKG라는 Package Spec 및 Body를 생성합니다. 또한 예제 데이터를 사용하여 전용(private) 생성자와 공용(public) 생성자가 포함된 패키지를 생성하고 호출합니다.

참고: 연습에서 건너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

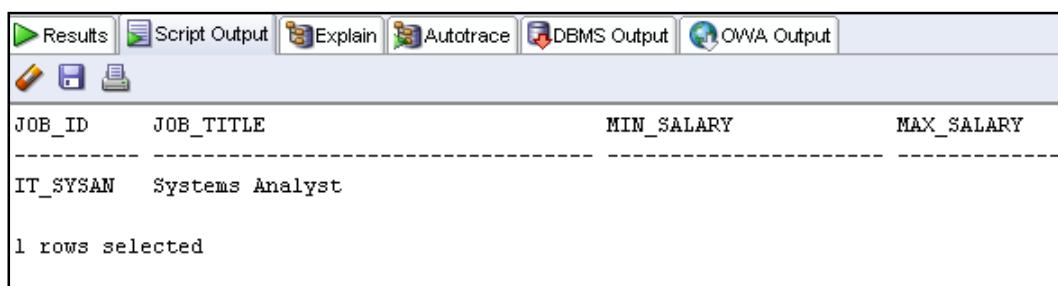
연습 3-1: 패키지 생성 및 사용

이 연습에서는 Package Spec 과 Package Body 를 생성한 다음 예제 데이터를 사용하여 패키지에서 생성자를 호출합니다.

- 1) ADD_JOB, UPD_JOB 및 DEL_JOB 프로시저의 복사본과 GET_JOB 함수가 포함된 JOB_PKG라는 Package Spec 및 Body 를 생성합니다.

참고: 패키지를 생성할 때는 이전에 저장한 프로시저 및 함수의 코드를 사용하십시오. 프로시저 또는 함수에서 코드를 복사하여 패키지의 적절한 부분에 붙여넣을 수 있습니다.

- a) 프로시저와 함수 머리글이 공용(public) 생성자로 포함된 Package Spec 을 생성합니다.
- b) 각 서브 프로그램을 구현하여 Package Body 를 생성합니다.
- c) Object Navigation 트리에서 Procedures 및 Functions 노드를 사용하여 방금 패키지화한 다음의 독립형 프로시저 및 함수를 삭제합니다.
 - i) ADD_JOB, UPD_JOB 및 DEL_JOB 프로시저
 - ii) GET_JOB 함수
- d) IT_SYSAN 및 SYSTEMS ANALYST 값을 파라미터로 전달하여 ADD_JOB 패키지 프로시저를 호출합니다.
- e) JOBS 테이블을 query 하고 결과를 확인합니다.



The screenshot shows the Oracle SQL developer interface with the 'Results' tab selected. The query executed was:

```
SELECT * FROM JOBS WHERE JOB_TITLE = 'SYSTEMS ANALYST';
```

The results are displayed in a grid:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		

Below the grid, it says "1 rows selected".

연습 3-1: 패키지 생성 및 사용 (계속)

- 2) 전용(private) 생성자와 공용(public) 생성자가 포함된 패키지를 생성하여 호출합니다.
- a) 앞에서 생성한 다음 프로시저 및 함수가 포함된 EMP_PKG라는 Package Spec과 Package Body를 생성합니다.
 - i) ADD_EMPLOYEE 프로시저를 공용(public) 생성자로 설정
 - ii) GET_EMPLOYEE 프로시저를 공용(public) 생성자로 설정
 - iii) VALID_DEPTID 함수를 전용(private) 생성자로 설정
 - b) 전자 메일 ID가 JAHARRIS인 사원 Jane Harris의 부서 ID 15를 사용하여 EMP_PKG.ADD_EMPLOYEE 프로시저를 호출합니다. 부서 ID 15는 존재하지 않으므로 프로시저의 예외 처리기에 지정된 대로 오류 메시지가 표시되어야 합니다.
 - c) 전자 메일 ID가 DASMITH인 사원 David Smith의 부서 ID 80을 사용하여 ADD_EMPLOYEE 패키지 프로시저를 호출합니다.
 - d) EMPLOYEES 테이블을 query하여 새 사원이 추가되었는지 확인합니다.

연습 해답 3-1: 패키지 생성 및 사용

이 연습에서는 Package Spec 과 Package Body 를 생성한 다음 예제 데이터를 사용하여 패키지에서 생성자를 호출합니다.

- 1) ADD_JOB, UPD_JOB 및 DEL_JOB 프로시저의 복사본과 GET_JOB 함수가 포함된 JOB_PKG라는 Package Spec 및 Body 를 생성합니다.

참고: 패키지를 생성할 때는 이전에 저장한 프로시저 및 함수의 코드를 사용하십시오. 프로시저 또는 함수에서 코드를 복사하여 패키지의 적절한 부분에 붙여넣을 수 있습니다.

- a) 프로시저와 함수 머리글이 공용(public) 생성자로 포함된 Package Spec 을 생성합니다.

/home/oracle/labs/plpu/solns/sol_03_01_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Spec 을 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE job_pkg IS
    PROCEDURE add_job (p_jobid jobs.job_id%TYPE, p_jobtitle
jobs.job_title%TYPE);
    PROCEDURE del_job (p_jobid jobs.job_id%TYPE);
    FUNCTION get_job (p_jobid IN jobs.job_id%type) RETURN
jobs.job_title%type;
    PROCEDURE upd_job(p_jobid IN jobs.job_id%TYPE,
p_jobtitle IN jobs.job_title%TYPE);
END job_pkg;
/
SHOW ERRORS
```



연습 해답 3-1: 패키지 생성 및 사용 (계속)

- b) 각 서브 프로그램을 구현하여 Package Body 를 생성합니다.

```
/home/oracle/labs/plpu/solns/sol_03_01_b.sql 스크립트를
엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러
Package Body 를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이
표시됩니다.

CREATE OR REPLACE PACKAGE BODY job_pkg IS
    PROCEDURE add_job (
        p_jobid jobs.job_id%TYPE,
        p_jobtitle jobs.job_title%TYPE) IS
    BEGIN
        INSERT INTO jobs (job_id, job_title)
        VALUES (p_jobid, p_jobtitle);
        COMMIT;
    END add_job;

    PROCEDURE del_job (p_jobid jobs.job_id%TYPE) IS
    BEGIN
        DELETE FROM jobs
        WHERE job_id = p_jobid;
        IF SQL%NOTFOUND THEN
            RAISE_APPLICATION_ERROR(-20203, 'No jobs
deleted.');
        END IF;
    END DEL_JOB;

    FUNCTION get_job (p_jobid IN jobs.job_id%type)
    RETURN jobs.job_title%type IS
        v_title jobs.job_title%type;
    BEGIN
        SELECT job_title
        INTO v_title
        FROM jobs
        WHERE job_id = p_jobid;
        RETURN v_title;
    END get_job;

    PROCEDURE upd_job(
        p_jobid IN jobs.job_id%TYPE,
        p_jobtitle IN jobs.job_title%TYPE) IS
    BEGIN
        UPDATE jobs
        SET job_title = p_jobtitle
        WHERE job_id = p_jobid;
```

연습 해답 3-1: 패키지 생성 및 사용 (계속)

```

    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20202, 'No job
updated.');
    END IF;
END upd_job;

END job_pkg;
/
SHOW ERRORS

```

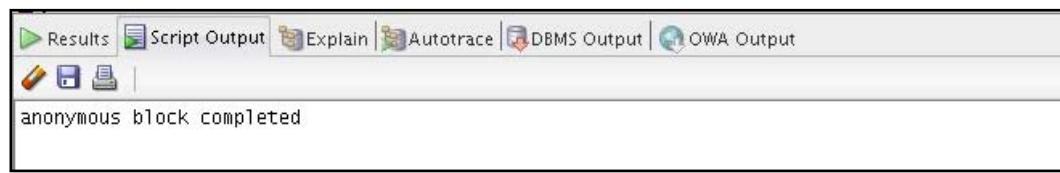


- c) Object Navigation 트리에서 Procedures 및 Functions 노드를 사용하여 방금 패키지화한 다음의 독립형 프로시저 및 함수를 삭제합니다.
 - i) ADD_JOB, UPD_JOB 및 DEL_JOB 프로시저
 - ii) GET_JOB 함수

프로시저나 함수를 삭제하려면 Object Navigation 트리에서 프로시저 또는 함수의 이름을 마우스 오른쪽 버튼으로 누르고 팝업 메뉴에서 Drop 을 선택합니다. Drop window 가 표시됩니다. Apply 를 눌러 프로시저 또는 함수를 삭제합니다. 확인 window 가 표시되면 OK 를 누릅니다.
- d) IT_SYSAN 및 SYSTEMS ANALYST 값을 파라미터로 전달하여 ADD_JOB 패키지 프로시저를 호출합니다.

/home/oracle/labs/plpu/solns/sol_03_01_d.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE job_pkg.add_job('IT_SYSAN', 'Systems Analyst')
```



연습 해답 3-1: 패키지 생성 및 사용 (계속)

- e) JOBS 테이블을 query 하고 결과를 확인합니다.

/home/oracle/labs/plpu/solns/sol_03_01_e.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘 또는 Execute Statement (F9)를 눌러 JOBS 테이블을 query 합니다. 코드 및 결과(Run Script 아이콘 사용)가 다음과 같이 표시됩니다.

```
SELECT *
FROM jobs
WHERE job_id = 'IT_SYSAN';
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
IT_SYSAN	Systems Analyst		
1 rows selected			

- 2) 전용(private) 생성자와 공용(public) 생성자가 포함된 패키지를 생성하여 호출합니다.

- a) 앞에서 생성한 다음 프로시저 및 함수가 포함된 EMP_PKG라는 Package Spec 과 Package Body 를 생성합니다.

- i) ADD_EMPLOYEE 프로시저를 공용(public) 생성자로 설정
- ii) GET_EMPLOYEE 프로시저를 공용(public) 생성자로 설정
- iii) VALID_DEPTID 함수를 전용(private) 생성자로 설정

/home/oracle/labs/plpu/solns/sol_03_02_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);
  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
```

연습 해답 3-1: 패키지 생성 및 사용 (계속)

```

END emp_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
    END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30)
IS
    BEGIN
        IF valid_deptid(p_deptid) THEN
            INSERT INTO employees(employee_id, first_name,
last_name, email,
                job_id, manager_id, hire_date, salary,
commission_pct, department_id)
            VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204, 'Invalid
department ID. Try again.');
        END IF;
    END add_employee;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
    BEGIN

```

연습 해답 3-1: 패키지 생성 및 사용 (계속)

```

SELECT salary, job_id
INTO p_sal, p_job
FROM employees
WHERE employee_id = p_empid;
END get_employee;
END emp_pkg;
/
SHOW ERRORS

```

The screenshot shows the SQL Worksheet interface with the 'Script Output' tab selected. The output window displays the following message:

```

PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

```

- b) 전자 메일 ID 가 JAHARRIS 인 사원 Jane Harris 의 부서 ID 15 를 사용하여 EMP_PKG.ADD_EMPLOYEE 프로시저를 호출합니다. 부서 ID 15 는 존재하지 않으므로 프로시저의 예외 처리기에 지정된 대로 오류 메시지가 표시되어야 합니다.

/home/oracle/labs/plpu/solns/sol_03_02_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

참고: 이 단계를 수행하기 전에 단계 3-2-a 를 완료해야 합니다. 단계 3-2-a 를 완료하지 않은 경우 sol_03_02_a.sql 스크립트를 먼저 실행합니다.

```

EXECUTE emp_pkg.add_employee( 'Jane' , 'Harris' , 'JAHARRIS' ,
p_deptid => 15 )

```

The screenshot shows the SQL Worksheet interface with the 'Script Output' tab selected. The output window displays the following error message:

```

Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Jane', 'Harris','JAHARRIS', p_deptid => 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 31
ORA-06512: at line 1

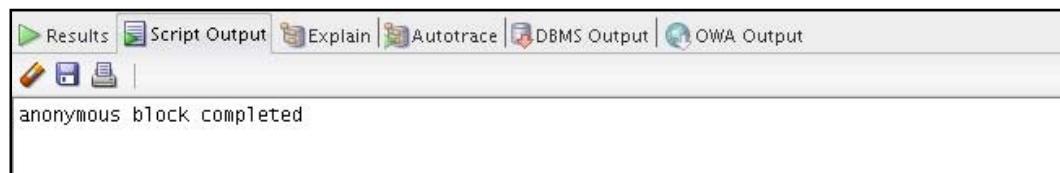
```

연습 해답 3-1: 패키지 생성 및 사용 (계속)

- c) 전자 메일 ID 가 DASMITH 인 사원 David Smith 의 부서 ID 80 을 사용하여 ADD_EMPLOYEE 패키지 프로시저를 호출합니다.

/home/oracle/labs/plpu/solns/sol_03_02_c.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee( 'David' , 'Smith' , 'DASMITH' ,
p_deptid => 80 )
```



- d) EMPLOYEES 테이블을 query 하여 새 사원이 추가되었는지 확인합니다.

/home/oracle/labs/plpu/solns/sol_03_02_d.sql 스크립트를 엽니다. SELECT 문 코드에 커서를 놓고 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘 또는 Execute Statement (F9) 아이콘을 눌러 EMPLOYEES 테이블을 query 합니다. 코드 및 결과(Execute Statement 아이콘)가 다음과 같이 표시됩니다.

```
SELECT *
FROM employees
WHERE last_name = 'Smith' ;
```

F9 아이콘을 사용하여 코드를 실행했기 때문에 출력이 Results 탭에 다음과 같이 표시됩니다.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	208	David	Smith	DASMITH	(null)	19-AUG-09	SA_REP	1000	0	145	80
2	159	Lindsey	Smith	LSMITH	011.44.1345.729268	10-MAR-97	SA_REP	8000	0.3	146	80
3	171	William	Smith	WSMITH	011.44.1343.629268	23-FEB-99	SA_REP	7400	0.15	148	80

단원 4 의 연습 및 해답

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

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 4-1: 패키지 작업

이 연습에서는 앞에서 생성한 EMP_PKG 패키지에 대한 코드를 수정하고 ADD_EMPLOYEE 프로시저를 오버로드합니다. 그런 다음 EMP_PKG 패키지에서 GET_EMPLOYEE라는 두 개의 오버로드된 함수를 생성합니다. 또한 유효한 부서 ID의 전용(private) PL/SQL 테이블을 채우도록 공용(public) 프로시저를 EMP_PKG에 추가하고, 전용(private) PL/SQL 테이블 내용을 사용하여 부서 ID 값을 검증하도록 VALID_DEPTID 함수를 수정합니다. 부서 ID의 전용(private) PL/SQL 테이블을 사용하도록 VALID_DEPTID 검증 처리 함수를 변경합니다. 마지막으로 Package Spec과 Package Body에 있는 서브 프로그램을 문자순으로 정렬되도록 재구성합니다.

- 1) 연습 4의 단계 2에서 생성한 EMP_PKG 패키지의 코드를 수정하고 ADD_EMPLOYEE 프로시저를 오버로드합니다.
 - a) Package Spec에서 다음 세 개의 파라미터를 받아들이는 ADD_EMPLOYEE라는 새 프로시저를 추가합니다.
 - i) First name
 - ii) Last name
 - iii) Department ID
 - b) Run Script(F5)를 눌러 패키지를 생성하고 컴파일합니다.
 - c) 다음과 같이 새 ADD_EMPLOYEE 프로시저를 Package Body에 구현합니다.
 - i) 이름의 첫 글자와 성의 첫 일곱 글자를 연결하여 만든 전자 메일 주소를 대문자 형식으로 지정합니다.
 - ii) 이 프로시저는 기존의 ADD_EMPLOYEE 프로시저를 호출하여 해당 파라미터와 형식이 지정된 전자 메일을 통해 값을 제공하여 실제 INSERT 작업을 수행해야 합니다.
 - iii) Run Script를 눌러 패키지를 생성합니다. 패키지를 컴파일합니다.
 - d) 부서 30에 추가될 Samuel Joplin이라는 이름을 사용하여 새 ADD_EMPLOYEE 프로시저를 호출합니다.
 - e) EMPLOYEES 테이블에 새 사원이 추가되었는지 확인합니다.

연습 4-1: 패키지 작업 (계속)

- 2) EMP_PKG 패키지에서 GET_EMPLOYEE라는 두 개의 오버로드된 함수를 생성합니다.
- Package Spec에서 다음 함수를 추가합니다.
 - `employees.employee_id%TYPE` 유형을 기준으로 `p_emp_id`라는 파라미터를 받아들이는 GET_EMPLOYEE 함수. 이 함수는 EMPLOYEES%ROWTYPE을 반환해야 합니다.
 - `employees.last_name%TYPE` 유형의 `p_family_name`이라는 파라미터를 받아들이는 GET_EMPLOYEE 함수. 이 함수는 EMPLOYEES%ROWTYPE을 반환해야 합니다.
 - Run Script를 눌러 패키지를 재생성하고 컴파일합니다.
 - Package Body에서 다음을 수행합니다.
 - 사원의 ID를 사용하여 사원을 query하기 위한 첫번째 GET_EMPLOYEE 함수를 구현합니다.
 - `p_family_name` 파라미터에서 제공되는 값에 등가 연산자를 사용하기 위한 두번째 GET_EMPLOYEE 함수를 구현합니다.
 - Run Script를 눌러 패키지를 재생성하고 컴파일합니다.
 - 다음과 같이 유ти리티 프로시저 PRINT_EMPLOYEE를 EMP_PKG 패키지에 추가합니다.
 - 프로시저가 EMPLOYEES%ROWTYPE을 파라미터로 받아들입니다.
 - 프로시저가 DBMS_OUTPUT 패키지를 사용하여 사원에 대한 다음 항목을 한 행에 표시합니다.
 - department_id
 - employee_id
 - first_name
 - last_name
 - job_id
 - salary
 - Run Script(F5)를 눌러 패키지를 생성하고 컴파일합니다.
 - 익명 블록을 사용하여 사원 ID 100과 성 'Joplin'으로 EMP_PKG.GET_EMPLOYEE 함수를 호출합니다. PRINT_EMPLOYEE 프로시저를 사용하여 각 행에 대해 반환된 결과를 표시합니다.
- 3) 회사에서는 부서 데이터를 자주 변경하지 않으므로, 공용(public) 프로시저 INIT_DEPARTMENTS를 사용하여 유효한 부서 ID의 전용(private) PL/SQL 테이블을 채움으로써 EMP_PKG의 성능을 향상시킬 수 있습니다. 전용(private) PL/SQL 테이블의 내용을 사용하여 부서 ID 값을 검증하도록 VALID_DEPTID 함수를 수정합니다.
- 참고: sol_04_03.sql 솔루션 파일 스크립트에는 단계 a, b 및 c를 위한 코드가 포함되어 있습니다.

연습 4-1: 패키지 작업 (계속)

- a) Package Spec에서 파라미터가 없는 INIT_DEPARTMENTS라는 프로시저를 생성합니다. PRINT_EMPLOYEES Spec 앞의 Package Spec 섹션에 다음을 추가하면 됩니다.

```
PROCEDURE init_departments;
```

- b) Package Body에서 BOOLEAN 값을 포함하는 valid_departments라는 전용(private) PL/SQL 인덱스화된 테이블에 모든 부서 ID를 저장하는 INIT_DEPARTMENTS 프로시저를 구현합니다.

- i) Package Body의 모든 프로시저 앞에 valid_departments 변수와 유형 정의 boolean_tab_type을 선언합니다. Package Body의 맨 앞에 다음을 입력합니다.

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

- ii) department_id 열 값을 인덱스로 사용하여 인덱스화된 테이블에 부서 ID의 존재 상태를 표시하기 위한 항목을 생성하고 이 항목에 TRUE 값을 지정합니다. 다음과 같이 Package Body 끝에 INIT_DEPARTMENTS 프로시저 선언을 입력합니다(print_employees 프로시저 바로 뒤).

```
PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;
```

- c) Body에서 다음과 같이 INIT_DEPARTMENTS 프로시저를 호출하여 테이블을 초기화하는 초기화 블록을 생성합니다.

```
BEGIN
    init_departments;
END;
```

- d) Run Script(F5)를 눌러 패키지를 생성하고 컴파일합니다.

- 4) 부서 ID의 전용(private) PL/SQL 테이블을 사용하도록 VALID_DEPTID 검증 처리 함수를 변경합니다.

- a) 부서 ID 값의 PL/SQL 테이블을 사용하여 검증을 수행하도록 VALID_DEPTID 함수를 수정합니다. Run Script(F5)를 눌러 패키지를 생성합니다. 패키지를 컴파일합니다.

- b) 부서 15에 있는 James Bond라는 이름을 사용하여 ADD_EMPLOYEE를 호출함으로써 코드를 테스트합니다. 어떤 결과가 나타납니까?

- c) 새 부서를 삽입합니다. 부서 ID에 15, 부서 이름에 'Security'를 지정합니다. 변경 사항을 커밋하고 확인합니다.

연습 4-1: 패키지 작업 (계속)

- d) 부서 15에 있는 James Bond라는 이름을 사용하여 ADD_EMPLOYEE를호출함으로써 코드를 다시 테스트합니다. 어떤 결과가 나타납니까?
 - e) EMP_PKG.INIT_DEPARTMENTS 프로시저를 실행하여 최근 부서 데이터로내부 PL/SQL 테이블을 갱신합니다.
 - f) 부서 15에 근무하는 James Bond라는 사원 이름을 사용하여ADD_EMPLOYEE를 호출함으로써 코드를 테스트합니다. 어떤 결과가나타납니까?
 - g) 각 테이블에서 사원 James Bond와 부서 15를 삭제하고 변경사항을 커밋한 다음 EMP_PKG.INIT_DEPARTMENTS 프로시저를호출하여 부서 데이터를 Refresh합니다. 먼저 SET SERVEROUTPUTON을 입력해야 합니다.
- 5) Package Spec과 Package Body에 있는 서브 프로그램을 문자순으로 정렬되도록재구성합니다.
- a) Package Spec을 편집하고 서브 프로그램을 문자순으로 재구성합니다. Run Script를 눌러 Package Spec을 재생성합니다. Package Spec을 컴파일합니다. 어떤 결과가 나타납니까?
 - b) Package Body를 편집하고 모든 서브 프로그램을 문자순으로 재구성합니다. Run Script를 눌러 Package Spec을 재생성합니다. Package Spec을재컴파일합니다. 어떤 결과가 나타납니까?
 - c) 오류가 발생한 서브 프로그램 참조에 대해 Body에서 사전 선언을 사용하여컴파일 오류를 수정합니다. Run Script를 눌러 패키지를 재생성하고 패키지를재컴파일합니다. 어떤 결과가 나타납니까?

연습 해답 4-1: 패키지 작업

이 연습에서는 앞에서 생성한 EMP_PKG 패키지에 대한 코드를 수정하고 ADD_EMPLOYEE 프로시저를 오버로드합니다. 그런 다음 EMP_PKG 패키지에서 GET_EMPLOYEE라는 두 개의 오버로드된 함수를 생성합니다. 또한 유효한 부서 ID의 전용(private) PL/SQL 테이블을 채우도록 공용(public) 프로시저를 EMP_PKG에 추가하고 전용(private) PL/SQL 테이블 내용을 사용하여 부서 ID 값을 검증하도록 VALID_DEPTID 함수를 수정합니다. 부서 ID의 전용(private) PL/SQL 테이블을 사용하도록 VALID_DEPTID 검증 처리 함수를 변경합니다. 마지막으로 Package Spec과 Package Body에 있는 서브 프로그램을 문자순으로 정렬되도록 재구성합니다.

- 1) 연습 4의 단계 2에서 생성한 EMP_PKG 패키지의 코드를 수정하고 ADD_EMPLOYEE 프로시저를 오버로드합니다.

- a) Package Spec에서 다음 세 개의 파라미터를 받아들이는 ADD_EMPLOYEE라는 새 프로시저를 추가합니다.
 - i) First name
 - ii) Last name
 - iii) Department ID

/home/oracle/labs/plpu/solns/sol_04_01_a.sql 파일을 엽니다. 코드가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  /* New overloaded add_employee */

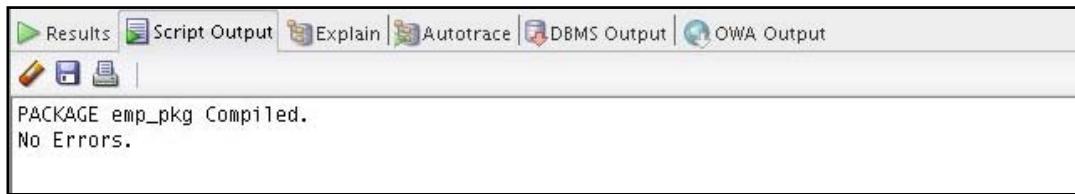
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);
END emp_pkg;
/
SHOW ERRORS

```

연습 해답 4-1: 패키지 작업 (계속)

- b) SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지를 생성하고 컴파일합니다.



- c) 다음과 같이 새 ADD_EMPLOYEE 프로시저를 Package Body에 구현합니다.

- i) 이름의 첫 글자와 성의 첫 일곱 글자를 연결하여 만든 전자 메일 주소를 대문자 형식으로 지정합니다.
- ii) 이 프로시저는 기존의 ADD_EMPLOYEE 프로시저를 호출하여 해당 파라미터와 형식이 지정된 전자 메일을 통해 값을 제공하여 실제 INSERT 작업을 수행해야 합니다.
- iii) Run Script 를 눌러 패키지를 생성합니다. 패키지를 컴파일합니다.

/home/oracle/labs/plpu/solns/sol_04_01_c.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드와 결과가 다음과 같이 표시됩니다. 아래 코드 상자에서 새로 추가된 코드가 굵은체 텍스트로 강조 표시됩니다.

```

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
                           departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END valid_deptid;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
    
```

연습 해답 4-1: 패키지 작업 (계속)

```

p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
  IF valid_deptid(p_deptid) THEN
    INSERT INTO employees(employee_id, first_name, last_name,
                           email, job_id, manager_id, hire_date, salary,
                           commission_pct, department_id)
    VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
            p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
            p_deptid);
  ELSE
    RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID. Try
                               again.');
  END IF;
END add_employee;

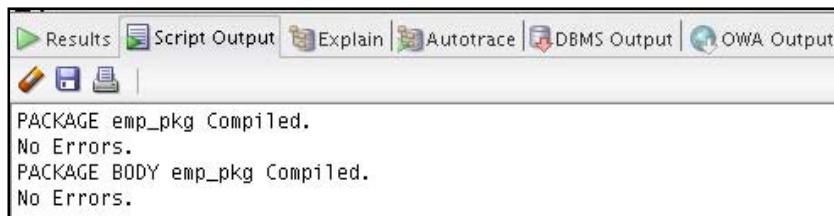
/* New overloaded add_employee procedure */

PROCEDURE add_employee(
  p_first_name employees.first_name%TYPE,
  p_last_name employees.last_name%TYPE,
  p_deptid employees.department_id%TYPE) IS
  p_email employees.email%type;
BEGIN
  p_email := UPPER(SUBSTR(p_first_name, 1,
                          1)||SUBSTR(p_last_name, 1, 7));
  add_employee(p_first_name, p_last_name, p_email, p_deptid =>
                p_deptid);
END;

/* End declaration of the overloaded add_employee procedure */

PROCEDURE get_employee(
  p.empid IN employees.employee_id%TYPE,
  p_sal OUT employees.salary%TYPE,
  p_job OUT employees.job_id%TYPE) IS
BEGIN
  SELECT salary, job_id
  INTO p_sal, p_job
  FROM employees
  WHERE employee_id = p.empid;
END get_employee;
END emp_pkg;
/
SHOW ERRORS

```

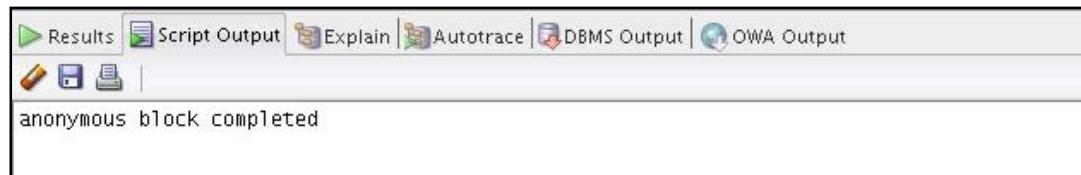


연습 해답 4-1: 패키지 작업 (계속)

- d) 부서 30에 추가될 Samuel Joplin이라는 이름을 사용하여 새 ADD_EMPLOYEE 프로시저를 호출합니다.

/home/oracle/labs/plpu/solns/sol_04_01_d.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee('Samuel', 'Joplin', 30)
```



- e) EMPLOYEES 테이블에 새 사원이 추가되었는지 확인합니다.

/home/oracle/labs/plpu/solns/sol_04_01_e.sql 스크립트를 엽니다. SELECT 문의 아무 곳이나 누른 다음 SQL Worksheet 도구 모음에서 Execute Statement (F9) 아이콘을 눌러 query를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT *
FROM employees
WHERE last_name = 'Joplin';
```

Results												
Results:												
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID	
1	209	Samuel	Joplin	SJOPLIN	(null)	17-JUN-09	SA_REP	1000	0	145	30	

- 2) EMP_PKG 패키지에서 GET_EMPLOYEE라는 두 개의 오버로드된 함수를 생성합니다.

- a) Package Spec에서 다음 함수를 추가합니다.

- i) employees.employee_id%TYPE 유형을 기준으로 p_emp_id라는 파라미터를 받아들이는 GET_EMPLOYEE 함수. 이 함수는 EMPLOYEES%ROWTYPE을 반환해야 합니다.

연습 해답 4-1: 패키지 작업 (계속)

- ii) employees.last_name%TYPE 유형의 p_family_name이라는
파라미터를 받아들이는 GET_EMPLOYEE 함수. 이 함수는
EMPLOYEES%ROWTYPE 을 반환해야 합니다.

/home/oracle/labs/plpu/solns/sol_04_02_a.sql
스크립트를 엽니다.

```

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

  PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

  /* New overloaded get_employees functions specs starts here: */

  FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

  FUNCTION get_employee(p.family_name employees.last_name%type)
    return employees%rowtype;

  /* New overloaded get_employees functions specs ends here. */

END emp_pkg;
/
SHOW ERRORS

```

연습 해답 4-1: 패키지 작업 (계속)

- b) Run Script 를 눌러 Package Spec 을 재생성하고 컴파일합니다.

SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Spec 을 재생성하고 컴파일합니다. 결과가 다음과 같이 표시됩니다.

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
PACKAGE emp_pkg Compiled.
No Errors.
```

참고: 앞에서 설명한 것처럼 코드에 오류 메시지가 포함된 경우 다음 프로시저로 코드를 재컴파일하여 Compiler - Log 탭에서 오류 또는 경고에 대한 세부 정보를 볼 수 있습니다. Package Spec 을 컴파일하려면 Object Navigator 트리에서 Package Spec 또는 전체 패키지 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Compile 을 선택합니다. 경고는 예상된 것이며 참고용일 뿐입니다.

```
Compiler - Log
Project: C:\Program Files\SQL Developer 1.2\sqldeveloper\system\oracle.sqldeveloper.1.2.0.2998\DefaultWorkspaces\ORA61.EMP_PKG@MyDBConnection
PACKAGE ORA61.EMP_PKG@MyDBConnection
Warning(21,5): PLW-07203: parameter 'P_JOB' may benefit from use of the NOCOPY compiler hint
```

- c) Package Body 에서 다음을 수행합니다.

- 사원의 ID 를 사용하여 사원을 query 하기 위한 첫번째 GET_EMPLOYEE 함수를 구현합니다.
- p_family_name 파라미터에서 제공되는 값에 등가 연산자를 사용하기 위한 두번째 GET_EMPLOYEE 함수를 구현합니다.

/home/oracle/labs/plpu/solns/sol_04_02_c.sql
스크립트를 엽니다. 새로 추가된 함수가 다음 코드 상자에서 강조 표시됩니다.

```
CREATE OR REPLACE PACKAGE emp_pkg IS
PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
```

연습 해답 4-1: 패키지 작업 (계속)

```

p_last_name employees.last_name%TYPE,
p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p.sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE);

/* New overloaded get_employees functions specs starts here: */

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype;

/* New overloaded get_employees functions specs ends here. */

END emp_pkg;
/
SHOW ERRORS

-- package body

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END valid_deptid;

    PROCEDURE add_employee(
        p.first_name employees.first_name%TYPE,
        p.last_name employees.last_name%TYPE,
        p.email employees.email%TYPE,
        p.job employees.job_id%TYPE DEFAULT 'SA_REP',
        p.mgr employees.manager_id%TYPE DEFAULT 145,
        p.sal employees.salary%TYPE DEFAULT 1000,
        p.comm employees.commission_pct%TYPE DEFAULT 0,
        p.deptid employees.department_id%TYPE DEFAULT 30) IS
    BEGIN
        IF valid_deptid(p.deptid) THEN
            INSERT INTO employees(employee_id, first_name, last_name,

```

연습 해답 4-1: 패키지 작업 (계속)

```

        email, job_id, manager_id, hire_date, salary,
        commission_pct, department_id)

    VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
            p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
            p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department ID.
                                         Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

/* New get_employee function declaration starts here */

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p.family_name;

```

연습 해답 4-1: 패키지 작업 (계속)

```

    RETURN rec_emp;
END;

/* New overloaded get_employee function declaration ends here */

END emp_pkg;
/
SHOW ERRORS

```

- d) Run Script 를 눌러 패키지를 재생성합니다. 패키지를 컴파일합니다.

SQL Worksheet 도구 모음에서 **Run Script (F5)** 아이콘을 눌러 패키지를 재생성하고 컴파일합니다. 결과가 다음과 같이 표시됩니다.

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

```

- e) 다음과 같이 유ти리티 프로시저 PRINT_EMPLOYEE 를 EMP_PKG 패키지에 추가합니다.
- 프로시저가 EMPLOYEES%ROWTYPE 을 파라미터로 받아들입니다.
 - 프로시저가 DBMS_OUTPUT 패키지를 사용하여 사원에 대한 다음 항목을 한 행에 표시합니다.
 - department_id
 - employee_id
 - first_name
 - last_name
 - job_id
 - salary

/home/oracle/labs/plpu/solns/sol_04_02_e.sql 스크립트를 엽니다. 새로 추가된 코드가 다음 코드 상자에서 강조 표시됩니다.

```

-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

```

연습 해답 4-1: 패키지 작업 (계속)

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p_empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;

/* New print_employee print_employee procedure spec */

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        SELECT 1
        INTO v_dummy
        FROM departments
        WHERE department_id = p_deptid;
        RETURN TRUE;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
    END valid_deptid;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30) IS
    BEGIN
        IF valid_deptid(p_deptid) THEN

```

연습 해답 4-1: 패키지 작업 (계속)

```

        INSERT INTO employees(employee_id, first_name,
last_name, email,
                job_id, manager_id, hire_date, salary,
commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
        END IF;
    END add_employee;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE) IS
        p_email employees.email%type;
    BEGIN
        p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
        add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
    END;

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p.job OUT employees.job_id%TYPE) IS
    BEGIN
        SELECT salary, job_id
        INTO p_sal, p.job
        FROM employees
        WHERE employee_id = p.empid;
    END get_employee;

    FUNCTION get_employee(p.emp_id employees.employee_id%type)
        return employees%rowtype IS
        rec_emp employees%rowtype;
    BEGIN
        SELECT * INTO rec_emp
        FROM employees
        WHERE employee_id = p.emp_id;
        RETURN rec_emp;
    END;

    FUNCTION get_employee(p.family_name
employees.last_name%type)
        return employees%rowtype IS
        rec_emp employees%rowtype;
    BEGIN
        SELECT * INTO rec_emp
        FROM employees
        WHERE last_name = p.family_name;
    END;

```

연습 해답 4-1: 패키지 작업 (계속)

```

    RETURN rec_emp;
END;

/* New print_employees procedure declaration. */

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

END emp_pkg;
/
SHOW ERRORS

```

- f) Run Script (F5)를 눌러 패키지를 생성하고 컴파일합니다.

SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지를 재생성하고 컴파일합니다.



- g) 임명 블록을 사용하여 사원 ID 100 과 성 'Joplin'으로 EMP_PKG.GET_EMPLOYEE 함수를 호출합니다. PRINT_EMPLOYEE 프로시저를 사용하여 각 행에 대해 반환된 결과를 표시합니다. 먼저 SET SERVEROUTPUT ON 을 입력해야 합니다.

/home/oracle/labs/plpu/solns/sol_04_02_g.sql
스크립트를 엽니다.

```

SET SERVEROUTPUT ON
BEGIN
    emp_pkg.print_employee(emp_pkg.get_employee(100));
    emp_pkg.print_employee(emp_pkg.get_employee('Joplin'));
END;
/

```

연습 해답 4-1: 패키지 작업 (계속)

```

anonymous block completed
90 100 Steven King AD_PRES 24000
30 209 Samuel Joplin SA_REP 1000
  
```

- 3) 회사에서는 부서 데이터를 자주 변경하지 않으므로, 공용(public) 프로시저 INIT_DEPARTMENTS 를 사용하여 유효한 부서 ID 의 전용(private) PL/SQL 테이블을 채움으로써 EMP_PKG 의 성능을 향상시킬 수 있습니다. 전용(private) PL/SQL 테이블의 내용을 사용하여 부서 ID 값을 검증하도록 VALID_DEPTID 함수를 수정합니다.

참고: sol_04_03.sql 솔루션 파일 스크립트에는 단계 a, b 및 c 를 위한 코드가 포함되어 있습니다.

- a) Package Spec에서 파라미터가 없는 INIT_DEPARTMENTS 라는 프로시저를 생성합니다. PRINT_EMPLOYEES Spec 앞의 Package Spec 섹션에 다음을 추가하면 됩니다.

```
PROCEDURE init_departments;
```

- b) Package Body에서 BOOLEAN 값을 포함하는 valid_departments 라는 전용(private) PL/SQL 인덱스화된 테이블에 모든 부서 ID를 저장하는 INIT_DEPARTMENTS 프로시저를 구현합니다.

- i) Package Body의 모든 프로시저 앞에 valid_departments 변수와 유형 정의 boolean_tab_type 을 선언합니다. Package Body의 맨 앞에 다음을 입력합니다.

```
TYPE boolean_tab_type IS TABLE OF BOOLEAN
INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;
```

- ii) department_id 열 값을 인덱스로 사용하여 인덱스화된 테이블에 부서 ID의 존재 상태를 표시하기 위한 항목을 생성하고 이 항목에 TRUE 값을 지정합니다. 다음과 같이 Package Body 끝에 INIT_DEPARTMENTS 프로시저 선언을 입력합니다(print_employees 프로시저 바로 뒤).

```
PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;
```

연습 해답 4-1: 패키지 작업 (계속)

- c) Body에서 다음과 같이 INIT_DEPARTMENTS 프로시저를 호출하여 테이블을 초기화하는 초기화 블록을 생성합니다.

```
BEGIN
    init_departments;
END;
```

/home/oracle/labs/plpu/solns/sol_04_03.sql 스크립트를 업니다. 새로 추가된 코드가 다음 코드 상자에서 강조 표시됩니다.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p_empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p_emp_id employees.employee_id%type)
        return employees%rowtype;

    FUNCTION get_employee(p_family_name
employees.last_name%type)
        return employees%rowtype;

    /* New procedure init_departments spec */

    PROCEDURE init_departments;

    PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY
```

연습 해답 4-1: 패키지 작업 (계속)

```

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

/* New type */

TYPE boolean_tab_type IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    SELECT 1
    INTO v_dummy
    FROM departments
    WHERE department_id = p_deptid;
    RETURN TRUE;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN

        INSERT INTO employees(employee_id, first_name, last_name,
            email, job_id, manager_id, hire_date, salary,
            commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name, p_last_name,
            p_email, p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
            p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID.
                                         Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%TYPE;
BEGIN

```

연습 해답 4-1: 패키지 작업 (계속)

```

    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
    p_rec_emp.employee_id|| ' ' ||
    p_rec_emp.first_name|| ' ' ||
    p_rec_emp.last_name|| ' ' ||
    p_rec_emp.job_id|| ' ' ||
    p_rec_emp.salary);
END;

/* New init_departments procedure declaration. */

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

```

연습 해답 4-1: 패키지 작업 (계속)

```

    END LOOP;
END;

/* call the new init_departments procedure. */

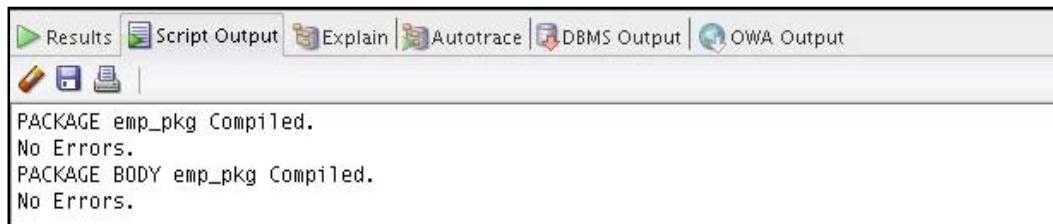
BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

- d) Run Script (F5)를 눌러 패키지를 재생성하고 컴파일합니다.

SQL Worksheet 도구 모음에서 **Run Script (F5)** 아이콘을 눌러 패키지를 재생성하고 컴파일합니다.



- 4) 부서 ID의 전용(private) PL/SQL 테이블을 사용하도록 VALID_DEPTID 검증 처리 함수를 변경합니다.
- 부서 ID 값의 PL/SQL 테이블을 사용하여 검증을 수행하도록 VALID_DEPTID 함수를 수정합니다. Run Script (F5)를 눌러 패키지를 생성하고 컴파일합니다.

/home/oracle/labs/plpu/solns/sol_04_04_a.sql 스크립트를 엽니다. Run Script (F5)를 눌러 패키지를 생성하고 컴파일합니다. 새로 추가된 코드가 다음 코드 상자에서 강조 표시됩니다.

```

-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

```

연습 해답 4-1: 패키지 작업 (계속)

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
    employees.last_name%type)
    return employees%rowtype;

/* New procedure init_departments spec */

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS

TYPE boolean_tab_type IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;
valid_departments boolean_tab_type;

    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
        v_dummy PLS_INTEGER;
    BEGIN
        RETURN valid_departments.exists(p_deptid);
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN FALSE;
    END valid_deptid;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,

```

연습 해답 4-1: 패키지 작업 (계속)

```

    p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name,
                           last_name, email, job_id, manager_id, hire_date,
                           salary, commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
                p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal,
                p_com, p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID.
                                         Try again.' );
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN

```

연습 해답 4-1: 패키지 작업 (계속)

```
SELECT * INTO rec_emp
FROM employees
WHERE last_name = p_family_name;
RETURN rec_emp;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

/* New init_departments procedure declaration. */

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

/* call the new init_departments procedure. */

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS
```



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

```
PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.
```

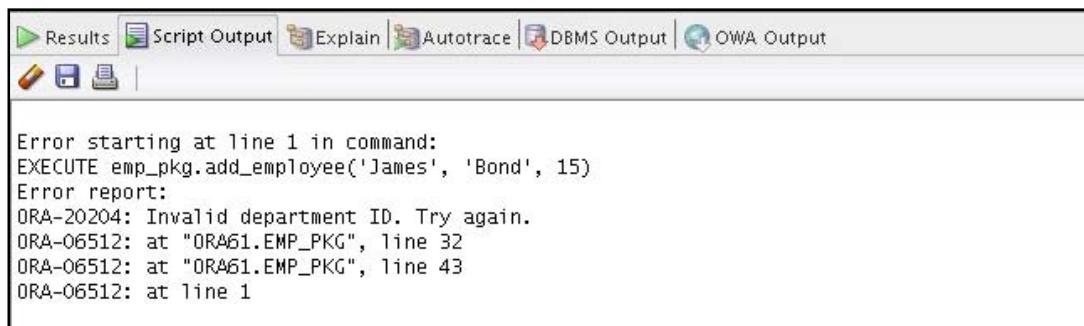
연습 해답 4-1: 패키지 작업 (계속)

- b) 부서 15에 있는 James Bond라는 이름을 사용하여 ADD_EMPLOYEE를 호출함으로써 코드를 테스트합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_04_b.sql
스크립트를 엽니다.

```
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
```

새 사원 삽입을 테스트하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다. 부서 15는 존재하지 않으므로 사원을 추가하는 삽입 작업이 예외와 함께 실패합니다.



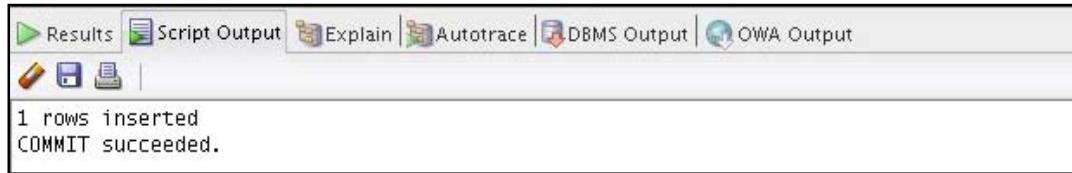
The screenshot shows the Oracle SQL Worksheet interface with the 'Script Output' tab selected. The output window displays the following error message:

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 32
ORA-06512: at "ORA61.EMP_PKG", line 43
ORA-06512: at line 1
```

- c) 새 부서를 삽입합니다. 부서 ID에 15, 부서 이름에 'Security'를 지정합니다. 변경 사항을 커밋하고 확인합니다.

/home/oracle/labs/plpu/solns/sol_04_04_c.sql 스크립트를 엽니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
INSERT INTO departments (department_id, department_name)
VALUES (15, 'Security');
COMMIT;
```



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

```
1 rows inserted
COMMIT succeeded.
```

연습 해답 4-1: 패키지 작업 (계속)

- d) 부서 15에 있는 James Bond라는 이름을 사용하여 ADD_EMPLOYEE를 호출함으로써 코드를 다시 테스트합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_04_d.sql 스크립트를 엽니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee( 'James' , 'Bond' , 15 )
```

The screenshot shows the SQL Developer interface with the 'Results' tab selected. The output window displays the following error message:

```
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('James', 'Bond', 15)
Error report:
ORA-20204: Invalid department ID. Try again.
ORA-06512: at "ORA61.EMP_PKG", line 32
ORA-06512: at "ORA61.EMP_PKG", line 43
ORA-06512: at line 1
```

사원을 추가하는 삽입 작업이 예외와 함께 실패합니다. 부서 15는 PL/SQL 연관 배열(인덱스화된 테이블) 패키지 상태 변수에서 항목으로 존재하지 않습니다.

- e) EMP_PKG.INIT_DEPARTMENTS 프로시저를 실행하여 최근 부서 데이터로 내부 PL/SQL 테이블을 갱신합니다.

/home/oracle/labs/plpu/solns/sol_04_04_e.sql 스크립트를 엽니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```

The screenshot shows the SQL Developer interface with the 'Results' tab selected. The output window displays the message:

```
anonymous block completed
```

- f) 부서 15에 근무하는 James Bond라는 사원을 사용하여 ADD_EMPLOYEE를 호출함으로써 코드를 테스트합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_04_f.sql 스크립트를 엽니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee( 'James' , 'Bond' , 15 )
```

연습 해답 4-1: 패키지 작업 (계속)

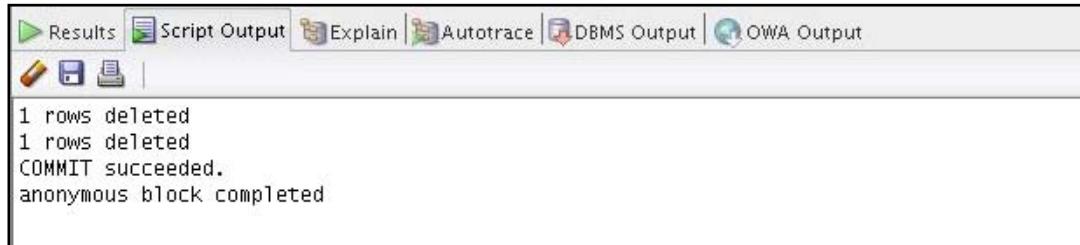
데이터베이스와 패키지의 PL/SQL 인덱스화된 테이블에 부서 15 레코드가 존재하므로 행이 최종적으로 삽입됩니다. 이는 패키지 상태 데이터를 Refresh 하는 **EMP_PKG.INIT_DEPARTMENTS** 를 호출했기 때문입니다.



- g) 각 테이블에서 사원 James Bond 와 부서 15 를 삭제하고 변경 사항을 커밋한 다음 EMP_PKG.INIT_DEPARTMENTS 프로시저를 호출하여 부서 데이터를 Refresh 합니다.

/home/oracle/labs/plpu/solns/sol_04_04_g.sql 스크립트를 엽니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
DELETE FROM employees
WHERE first_name = 'James' AND last_name = 'Bond';
DELETE FROM departments WHERE department_id = 15;
COMMIT;
EXECUTE EMP_PKG.INIT_DEPARTMENTS
```



- 5) Package Spec 과 Package Body 에 있는 서브 프로그램을 문자순으로 정렬되도록 재구성합니다.

- a) Package Spec 을 편집하고 서브 프로그램을 문자순으로 재구성합니다. Run Script 를 눌러 Package Spec 을 재생성합니다. Package Spec 을 컴파일합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_05_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지를 재생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다. Package Spec 서브 프로그램은 이미 문자순으로 되어 있습니다.

연습 해답 4-1: 패키지 작업 (계속)

```

CREATE OR REPLACE PACKAGE emp_pkg IS

/* the package spec is already in an alphabetical order. */

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id
employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype;

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

PACKAGE emp_pkg Compiled.
No Errors.

연습 해답 4-1: 패키지 작업 (계속)

- b) Package Body 를 편집하고 모든 서브 프로그램을 문자순으로 재구성합니다.

Run Script 를 눌러 Package Spec 을 재생성합니다. Package Spec 을 재컴파일합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_05_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지를 재생성합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
-- Package BODY
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30) IS
    BEGIN
        IF valid_deptid(p_deptid) THEN
            INSERT INTO employees(employee_id, first_name,
last_name, email,
                job_id, manager_id, hire_date, salary,
                commission_pct, department_id)
            VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204, 'Invalid
department ID. Try again.');
        END IF;
    END add_employee;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE) IS
        p_email employees.email%TYPE;
    BEGIN
        p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
        add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
    END;

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
```

연습 해답 4-1: 패키지 작업 (계속)

```

    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p_emp_id
employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p.family_name;
    RETURN rec_emp;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                           p_rec_emp.employee_id|| ' ' ||
                           p_rec_emp.first_name|| ' ' ||
                           p_rec_emp.last_name|| ' ' ||
                           p_rec_emp.job_id|| ' ' ||
                           p_rec_emp.salary);
END;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN

```

연습 해답 4-1: 패키지 작업 (계속)

```

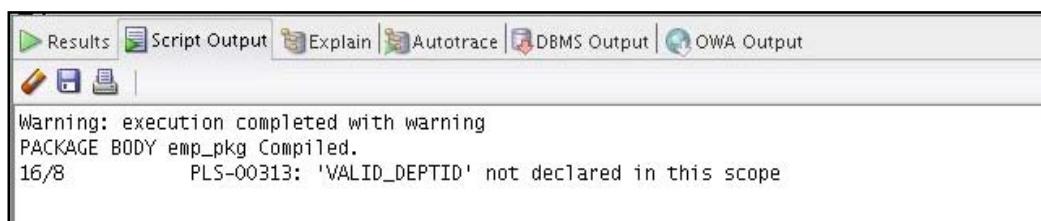
    RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

VALID_DEPTID 함수가 선언되기 전에 참조되므로 패키지가 성공적으로 컴파일되지 않습니다.



- c) 오류가 발생한 서브 프로그램 참조에 대해 Body에서 사전 선언을 사용하여 컴파일 오류를 수정합니다. Run Script 를 눌러 패키지를 재생성하고 패키지를 재컴파일합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_04_05_c.sql 스크립트를 엽니다. 아래의 코드 상자에서 함수의 사전 선언이 강조 표시됩니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지를 재생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;

/* forward declaration of valid_deptid */

FUNCTION valid_deptid(p_deptid IN
    departments.department_id%TYPE)
RETURN BOOLEAN;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,

```

연습 해답 4-1: 패키지 작업 (계속)

```

p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN /* valid_deptid function
referenced */
        INSERT INTO employees(employee_id, first_name,
last_name, email,
            job_id, manager_id, hire_date, salary,
commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
            p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype IS

```

연습 해답 4-1: 패키지 작업 (계속)

```

rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

/* New alphabetical location of function init_departments.
 */

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                          p_rec_emp.employee_id|| ' ' ||
                          p_rec_emp.first_name|| ' ' ||
                          p_rec_emp.last_name|| ' ' ||
                          p_rec_emp.job_id|| ' ' ||
                          p_rec_emp.salary);
END;

/* New alphabetical location of function valid_deptid. */

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

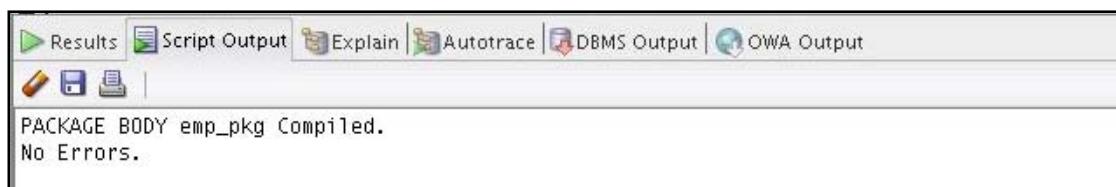
BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

연습 해답 4-1: 패키지 작업 (계속)

아래 표시된 것과 같이 VALID_DEPTID 함수에 대한 사전 선언으로 인해
Package Body 가 성공적으로 컴파일됩니다.



The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the message: 'PACKAGE BODY emp_pkg Compiled. No Errors.' This indicates that the package body was successfully compiled without any errors.

단원 5의 연습 및 해답

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

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 5-1: UTL_FILE 패키지 사용

이 연습에서는 UTL_FILE 패키지를 사용하여 각 부서의 사원에 대한 텍스트 파일 보고서를 작성합니다. 먼저 UTL_FILE 패키지를 사용하여 사원 보고서를 운영 체제에 파일로 작성하는 EMPLOYEE_REPORT 라는 프로시저를 생성하여 실행합니다. 보고서에서는 소속 부서의 평균 급여를 초과하는 사원 리스트를 생성해야 합니다. 마지막으로 생성된 출력 텍스트 파일을 봅니다.

- 1) UTL_FILE 패키지를 사용하여 사원 보고서를 운영 체제에 파일로 작성하는 EMPLOYEE_REPORT 라는 프로시저를 생성합니다. 보고서에서는 소속 부서의 평균 급여를 초과하는 사원 리스트를 생성해야 합니다.

- a) 프로그램은 두 가지 파라미터를 사용해야 합니다. 첫번째 파라미터는 출력 디렉토리입니다. 두번째 파라미터는 작성하려는 텍스트 파일의 이름입니다.

참고: 디렉토리 위치 값 UTL_FILE 을 사용하십시오. UTL_FILE 패키지를 사용할 때 발생할 수 있는 오류를 처리하는 예외 처리 섹션을 추가하십시오.

- b) SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다.

- 2) 다음 두 인수를 사용하여 프로시저를 호출합니다.

- a) 디렉토리 객체에 대한 별칭인 REPORTS_DIR 을 첫번째 파라미터로 사용합니다.

- b) sal_rpt61.txt 를 두번째 파라미터로 사용합니다.

- 3) 다음과 같이 생성된 출력 텍스트 파일을 봅니다.

- a) 바탕 화면에서 **Terminal** 아이콘을 두 번 누릅니다. **Terminal window** 가 표시됩니다.
 - b) \$ 프롬프트에서 cd 명령을 사용하여 생성된 출력 파일 sal_rpt61.txt 가 들어 있는 **/home/oracle/labs/plpu/reports** 폴더로 변경합니다.

참고: pwd 명령을 사용하여 현재 작업 디렉토리를 나열할 수 있습니다.

- c) ls 명령을 사용하여 현재 디렉토리의 내용을 나열합니다.
 - d) 선택한 편집기 또는 gedit 를 사용하여 전송된 sal_rpt61.txt 파일을 엽니다.

연습 해답 5-1: UTL_FILE 패키지 사용

이 연습에서는 UTL_FILE 패키지를 사용하여 각 부서의 사원에 대한 텍스트 파일 보고서를 작성합니다. 먼저 UTL_FILE 패키지를 사용하여 사원 보고서를 운영 체제에 파일로 작성하는 EMPLOYEE_REPORT라는 프로시저를 생성하여 실행합니다. 보고서에서는 소속 부서의 평균 급여를 초과하는 사원 리스트를 생성해야 합니다. 마지막으로 생성된 출력 텍스트 파일을 봅니다.

- 1) UTL_FILE 패키지를 사용하여 사원 보고서를 운영 체제에 파일로 작성하는 EMPLOYEE_REPORT라는 프로시저를 생성합니다. 보고서에서는 소속 부서의 평균 급여를 초과하는 사원 리스트를 생성해야 합니다.

- a) 프로그램은 두 가지 파라미터를 사용해야 합니다. 첫번째 파라미터는 출력 디렉토리입니다. 두번째 파라미터는 작성하려는 텍스트 파일의 이름입니다.

참고: 디렉토리 위치 값 UTL_FILE 을 사용하십시오. UTL_FILE 패키지를 사용할 때 발생할 수 있는 오류를 처리하는 예외 처리 섹션을 추가하십시오.

/home/oracle/labs/plpu/solns/sol_05_01.sql 스크립트에서 파일을 엽니다.

```
-- Verify with your instructor that the database initSID.ora
-- file has the directory path you are going to use with
this -- procedure.
-- For example, there should be an entry such as:
-- UTL_FILE_DIR = /home1/teachX/UTL_FILE in your initSID.ora
-- (or the SPFILE)
-- HOWEVER: The course has a directory alias provided called
-- "REPORTS_DIR" that is associated with an appropriate
-- directory. Use the directory alias name in quotes for the
-- first parameter to create a file in the appropriate
-- directory.

CREATE OR REPLACE PROCEDURE employee_report(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    f UTL_FILE.FILE_TYPE;
    CURSOR cur_avg IS
        SELECT last_name, department_id, salary
        FROM employees outer
        WHERE salary > (SELECT AVG(salary)
                         FROM employees inner
                         GROUP BY outer.department_id)
        ORDER BY department_id;
BEGIN
    f := UTL_FILE.FOPEN(p_dir, p_filename, 'W');

    UTL_FILE.PUT_LINE(f, 'Employees who earn more than average
salary: '');
```

연습 해답 5-1: UTL_FILE 패키지 사용 (계속)

```

UTL_FILE.PUT_LINE(f, 'REPORT GENERATED ON ' || SYSDATE);
UTL_FILE.NEW_LINE(f);
FOR emp IN cur_avg
LOOP

    UTL_FILE.PUT_LINE(f,
    RPAD(emp.last_name, 30) || ' ' ||
    LPAD(NVL(TO_CHAR(emp.department_id, '9999'), '-'), 5) || '
    ' ||
    LPAD(TO_CHAR(emp.salary, '$99,999.00'), 12));
END LOOP;
UTL_FILE.NEW_LINE(f);
UTL_FILE.PUT_LINE(f, '*** END OF REPORT ***');
UTL_FILE.FCLOSE(f);
END employee_report;
/

```

- b) SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다.

SQL Worksheet 도구 모음에서 **Run Script (F5)** 아이콘을 눌러 프로시저를 생성하고 컴파일합니다.



- 2) 다음을 인수로 사용하여 프로시저를 호출합니다.

- a) 디렉토리 객체에 대한 별칭인 REPORTS_DIR 을 첫번째 파라미터로 사용합니다.
- b) sal_rpt61.txt 를 두번째 파라미터로 사용합니다.

/home/oracle/labs/plpu/solns/sol_05_02.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 실행합니다. 결과가 다음과 같이 표시됩니다.

```
-- For example, if you are student ora61, use 61 as a prefix
EXECUTE employee_report('REPORTS_DIR', 'sal_rpt61.txt')
```

연습 해답 5-1: UTL_FILE 패키지 사용 (계속)

- 3) 다음과 같이 생성된 출력 텍스트 파일을 봅니다.
- 바탕 화면에서 **Terminal** 아이콘을 두 번 누릅니다. **Terminal** window 가 표시됩니다.
 - 다음과 같이 \$ 프롬프트에서 cd 명령을 사용하여 생성된 출력 파일 **sal_rpt61.txt** 가 들어 있는 **/home/oracle/labs/plpu/reports** 폴더로 변경합니다.

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$
```

참고: 위 스크린샷에 표시된 것처럼 pwd 명령을 사용하여 현재 작업 디렉토리를 나열할 수 있습니다.

- 다음과 같이 ls 명령을 사용하여 현재 디렉토리의 내용을 나열됩니다.

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$ls
salary_report.sql sal_rpt61.txt
[oracle@EDRSR13P1 reports]$
```

생성된 출력 파일 **sal_rpt61.txt** 를 적어 두십시오.

- 선택한 편집기 또는 gedit 를 사용하여 전송된 sal_rpt61.txt 파일을 엽니다. 보고서가 다음과 같이 표시됩니다.

```
[oracle@EDRSR13P1 ~]$cd labs/plpu/reports
[oracle@EDRSR13P1 reports]$pwd
/home/oracle/labs/plpu/reports
[oracle@EDRSR13P1 reports]$ls
salary_report.sql sal_rpt61.txt
[oracle@EDRSR13P1 reports]$gedit sal_rpt61.txt
```

연습 해답 5-1: UTL_FILE 패키지 사용 (계속)

```
sal_rpt61.txt *  
Employees who earn more than average salary:  
REPORT GENERATED ON 18-JUN-09  
  
Hartstein          20   $13,000.00  
Raphaely           30   $11,000.00  
Mavris             40   $6,500.00  
Weiss              50   $8,000.00  
Fripp               50   $8,200.00  
Kaufling            50   $7,900.00  
Vollman             50   $6,500.00  
Hunold              60   $9,000.00  
Baer                70   $10,000.00  
Cambrault            80   $11,000.00  
Zlotkey              80   $10,500.00  
Tucker              80   $10,000.00  
Bernstein            80   $9,500.00  
Hall                 80   $9,000.00  
Olsen                80   $8,000.00  
Cambrault            80   $7,500.00  
Tuvault              80   $7,000.00  
King                 80   $10,000.00  
Sully                80   $9,500.00  
McEwen               80   $9,000.00  
Smith                80   $8,000.00  
Doran                80   $7,500.00  
Sewall               80   $7,000.00  
Vishney              80   $10,500.00  
Greene               80   $9,500.00  
Marvins              80   $7,200.00  
Lee                  80   $6,800.00  
  
...  
  
Kochhar              90   $17,000.00  
Urman                100  $7,800.00  
Sciarra              100  $7,700.00  
Chen                 100  $8,200.00  
Faviet                100  $9,000.00  
Greenberg             100  $12,000.00  
Popp                  100  $6,900.00  
Higgins               110  $12,000.00  
Gietz                  -   $8,300.00  
Grant                  -   $7,000.00  
  
*** END OF REPORT ***
```

단원 6 의 연습 및 해답

이 연습에서는 Native Dynamic SQL 을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 채우고 수정하고 삭제하는 패키지를 생성합니다. 또한 스키마에서 PL/SQL 코드(모든 PL/SQL 코드 또는 USER_OBJECTS 테이블에서 상태가 INVALID 인 코드만)를 컴파일하는 패키지를 생성합니다.

참고: 연습에서 건너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 6-1: Native Dynamic SQL 사용

이 연습에서는 Native Dynamic SQL 을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 채우고 수정하고 삭제하는 패키지를 생성합니다. 또한 스키마에서 PL/SQL 코드(모든 PL/SQL 코드 또는 USER_OBJECTS 테이블에서 상태가 INVALID 인 코드만)를 컴파일하는 패키지를 생성합니다.

- 1) Native Dynamic SQL 을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 수정, 삭제 및 채우는 TABLE_PKG 라는 패키지를 생성합니다. 서브 프로그램이 옵션 기본 파라미터를 NULL 값으로 관리해야 합니다.

- a) 다음 프로시저를 사용하여 Package Spec 을 생성합니다.

```
PROCEDURE make(p_table_name VARCHAR2,
p_col_specs VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

- b) 파라미터를 받아들이고 Native Dynamic SQL 을 사용하여 실행되는 적절한 SQL 문을 동적으로 생성하는 Package Body 를 생성합니다(remove 프로시저 제외). 이 프로시저는 DBMS_SQL 패키지를 사용하여 작성해야 합니다.
- c) MAKE 패키지 프로시저를 실행하여 다음과 같이 테이블을 생성합니다.

```
make('my_contacts', 'id number(4), name
varchar2(40)');
```

- d) MY_CONTACTS 테이블 구조에 대해 설명합니다.
- e) ADD_ROW 패키지 프로시저를 실행하여 다음 행을 추가합니다.
SERVOROUTPUT 을 활성화합니다.

```
add_row('my_contacts','1','Lauran Serhal','id, name');
add_row('my_contacts','2','Nancy','id, name');
add_row('my_contacts','3','Sunitha Patel','id, name');
add_row('my_contacts','4','Valli Pataballa','id, name');
```

- f) MY_CONTACTS 테이블 내용을 query 하여 추가 내용을 확인합니다.
- g) DEL_ROW 패키지 프로시저를 실행하여 ID 값이 3 인 연락처를 삭제합니다.
- h) 다음 행 데이터를 사용하여 UPD_ROW 프로시저를 실행합니다.
- i) MY_CONTACTS 테이블 내용을 query 하여 변경 내용을 확인합니다.
- j) remove 프로시저를 사용하여 테이블을 삭제하고 MY_CONTACTS 테이블에 대해 설명합니다.

연습 6-1: Native Dynamic SQL 사용 (계속)

- 2) 스키마에서 PL/SQL 코드를 컴파일하는 COMPILE_PKG 패키지를 생성합니다.
- Spec에서 컴파일될 PL/SQL 프로그램 단위의 이름을 사용하는 MAKE라는 패키지 프로시저를 생성합니다.
 - Package Body에 다음을 포함합니다.
 - i) 이 연습의 단계 1에 있는 TABLE_PKG 프로시저에 사용된 EXECUTE 프로시저
 - ii) 데이터 딕셔너리에서 PL/SQL 객체 유형을 확인하기 위한 GET_TYPE이라는 전용(private) 함수
 - 이 함수는 객체가 존재할 경우 유형 이름을 반환하고(Body가 있는 패키지에는 PACKAGE 사용), 그렇지 않을 경우에는 NULL을 반환합니다.
 - WHERE 절 조건에서 조건에 다음을 추가하여 이름이 PACKAGE(PACKAGE BODY가 포함될 수도 있음)를 나타낼 경우 한 행만 반환되도록 합니다. 이 경우 전체 패키지만 컴파일할 수 있으며 Spec 또는 Body를 별도의 구성 요소로 컴파일할 수는 없습니다.

```
rownum = 1
```
 - iii) 다음 정보를 사용하여 MAKE 프로시저를 생성합니다.
 - MAKE 프로시저는 객체 이름을 나타내는 name이라는 인수 하나만 받아들입니다.
 - MAKE 프로시저는 GET_TYPE 함수를 호출해야 합니다. 객체가 있을 경우 MAKE는 ALTER 문을 사용하여 객체를 동적으로 컴파일합니다.
 - c) COMPILE_PKG.MAKE 프로시저를 사용하여 다음을 컴파일합니다.
 - i) EMPLOYEE_REPORT 프로시저
 - ii) EMP_PKG 패키지
 - iii) EMP_DATA라는 존재하지 않는 객체

연습 해답 6-1: Native Dynamic SQL 사용

이 연습에서는 Native Dynamic SQL을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 채우고 수정하고 삭제하는 패키지를 생성합니다. 또한 스키마에서 PL/SQL 코드(모든 PL/SQL 코드 또는 USER_OBJECTS 테이블에서 상태가 INVALID인 코드만)를 컴파일하는 패키지를 생성합니다.

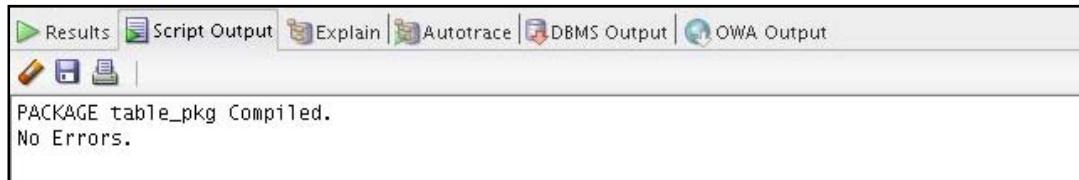
- 1) Native Dynamic SQL을 사용하여 테이블을 생성하거나 삭제하고 테이블에서 행을 수정, 삭제 및 채우는 TABLE_PKG라는 패키지를 생성합니다. 서브 프로그램이 옵션 기본 파라미터를 NULL 값으로 관리해야 합니다.

- a) 다음 프로시저를 사용하여 Package Spec을 생성합니다.

```
PROCEDURE make(p_table_name VARCHAR2, p_col_specs
               VARCHAR2)
PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                  VARCHAR2, p_cols VARCHAR2 := NULL)
PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                  VARCHAR2, p_conditions VARCHAR2 := NULL)
PROCEDURE del_row(p_table_name VARCHAR2,
                  p_conditions VARCHAR2 := NULL);
PROCEDURE remove(p_table_name VARCHAR2)
```

/home/oracle/labs/plpu/solns/sol_06_01_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Spec을 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE table_pkg IS
  PROCEDURE make(p_table_name VARCHAR2, p_col_specs
                 VARCHAR2);
  PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                    VARCHAR2, p_cols VARCHAR2 := NULL);
  PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                    VARCHAR2, p_conditions VARCHAR2 := NULL);
  PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
                    VARCHAR2 := NULL);
  PROCEDURE remove(p_table_name VARCHAR2);
END table_pkg;
/
SHOW ERRORS
```



연습 해답 6-1: Native Dynamic SQL 사용 (계속)

- b) 파라미터를 받아들이고 Native Dynamic SQL 을 사용하여 실행되는 적절한 SQL 문을 동적으로 생성하는 Package Body 를 생성합니다(remove 프로시저 제외). 이 프로시저는 DBMS_SQL 패키지를 사용하여 작성해야 합니다.

/home/oracle/labs/plpu/solns/sol_06_01_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Spec 을 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PACKAGE BODY table_pkg IS
    PROCEDURE execute(p_stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(p_stmt);
        EXECUTE IMMEDIATE p_stmt;
    END;

    PROCEDURE make(p_table_name VARCHAR2, p_col_specs VARCHAR2)
    IS
        v_stmt VARCHAR2(200) := 'CREATE TABLE || p_table_name ||
                                (' || p_col_specs || ')';
    BEGIN
        execute(v_stmt);
    END;

    PROCEDURE add_row(p_table_name VARCHAR2, p_col_values
                      VARCHAR2, p_cols VARCHAR2 := NULL) IS
        v_stmt VARCHAR2(200) := 'INSERT INTO || p_table_name';
    BEGIN
        IF p_cols IS NOT NULL THEN
            v_stmt := v_stmt || ' (' || p_cols || ')';
        END IF;
        v_stmt := v_stmt || ' VALUES (' || p_col_values || ')';
        execute(v_stmt);
    END;

    PROCEDURE upd_row(p_table_name VARCHAR2, p_set_values
                      VARCHAR2, p_conditions VARCHAR2 := NULL)
    IS

        v_stmt VARCHAR2(200) := 'UPDATE || p_table_name || ' SET
        ' || p_set_values;
    BEGIN
        IF p_conditions IS NOT NULL THEN
            v_stmt := v_stmt || ' WHERE ' || p_conditions;
        END IF;
        execute(v_stmt);
    END;

```

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

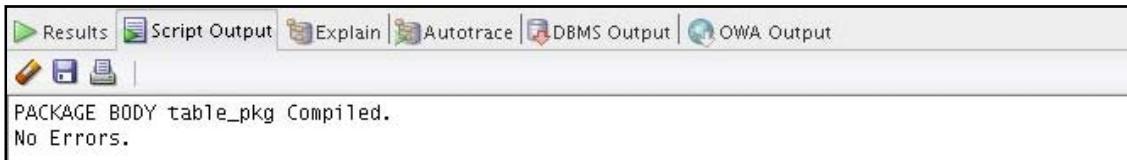
```

PROCEDURE del_row(p_table_name VARCHAR2, p_conditions
                  VARCHAR2 := NULL) IS
    v_stmt VARCHAR2(200) := 'DELETE FROM ' || p_table_name;
BEGIN
    IF p_conditions IS NOT NULL THEN
        v_stmt := v_stmt || ' WHERE ' || p_conditions;
    END IF;
    execute(v_stmt);
END;

PROCEDURE remove(p_table_name VARCHAR2) IS
    cur_id INTEGER;
    v_stmt VARCHAR2(100) := 'DROP TABLE ' || p_table_name;
BEGIN
    cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_OUTPUT.PUT_LINE(v_stmt);
    DBMS_SQLPARSE(cur_id, v_stmt, DBMS_SQL.NATIVE);
    -- Parse executes DDL statements,no EXECUTE is required.
    DBMS_SQL CLOSE_CURSOR(cur_id);
END;

END table_pkg;
/
SHOW ERRORS

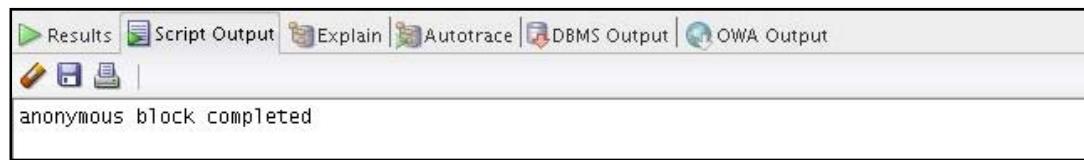
```



- c) MAKE 패키지 프로시저를 실행하여 다음과 같이 테이블을 생성합니다.

make('my_contacts', 'id number(4), name
varchar2(40)');
 /home/oracle/labs/plpu/solns/sol_06_01_c.sql 스크립트를
 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러
 Package Spec 을 생성합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE table_pkg.make('my_contacts', 'id number(4), name  
varchar2(40)')
```



연습 해답 6-1: Native Dynamic SQL 사용 (계속)

- d) MY_CONTACTS 테이블 구조에 대해 설명합니다.

코드 및 결과가 다음과 같이 표시됩니다.

```
MyDBConnection
1 DESCRIBE my_contacts
Results Script Output Explain Autotrace DBMS Output OWA Output
DESCRIBE my_contacts
Name Null Type
ID NUMBER(4)
NAME VARCHAR2(40)
2 rows selected
```

- e) ADD_ROW 패키지 프로시저를 실행하여 다음 행을 추가합니다.

SERVERTOUTPUT 을 활성화합니다.

```
add_row('my_contacts', '1', 'Lauran Serhal', 'id, name');
add_row('my_contacts', '2', 'Nancy', 'id, name');
add_row('my_contacts', '3', 'Sunitha Patel', 'id, name');
add_row('my_contacts', '4', 'Valli Pataballa', 'id, name');
```

/home/oracle/labs/plpu/solns/sol_06_01_e.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다.

```
SET SERVEROUTPUT ON
BEGIN
  table_pkg.add_row('my_contacts', '1', 'Lauran Serhal', 'id, name');
  table_pkg.add_row('my_contacts', '2', 'Nancy', 'id, name');
  table_pkg.add_row('my_contacts', '3', 'Sunitha Patel', 'id, name');
  table_pkg.add_row('my_contacts', '4', 'Valli Pataballa', 'id, name');
END;
/
```

```
anonymous block completed
INSERT INTO my_contacts (id, name) VALUES (1,'Lauran Serhal')
INSERT INTO my_contacts (id, name) VALUES (2,'Nancy')
INSERT INTO my_contacts (id,name) VALUES (3,'Sunitha Patel')
INSERT INTO my_contacts (id,name) VALUES (4,'Valli Pataballa')
```

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

- f) MY_CONTACTS 테이블 내용을 query하여 추가 내용을 확인합니다.

코드 및 결과가 다음과 같이 표시됩니다.

```
1 SELECT *
2 FROM my_contacts;
3
```

Results Script Output Explain Autotrace DBMS Output OWA Output

ID	NAME
1	Lauran Serhal
2	Nancy
3	Sunitha Patel
4	Valli Pataballa

4 rows selected

- g) DEL_ROW 패키지 프로시저를 실행하여 ID 값이 3인 연락처를 삭제합니다.

코드 및 결과가 다음과 같이 표시됩니다.

```
1 SET SERVEROUTPUT ON
2 EXECUTE table_pkg.del_row('my_contacts', 'id=3')
3
```

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

- h) 다음 행 데이터를 사용하여 UPD_ROW 프로시저를 실행합니다.

```
upd_row( 'my_contacts' , 'name= ''Nancy Greenberg''' , 'id=2' );
```

코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows an Oracle SQL Worksheet window. The code area contains:

```
1 SET SERVEROUTPUT ON
2 EXEC table_pkg.upd_row('my_contacts', 'name= ''Nancy Greenberg''' , 'id=2')
3
```

The results area shows the output of the anonymous block:

```
anonymous block completed
UPDATE my_contacts SET name='Nancy Greenberg' WHERE id=2
```

- i) MY_CONTACTS 테이블 내용을 query 하여 변경 내용을 확인합니다.

코드 및 결과가 다음과 같이 표시됩니다.

The screenshot shows an Oracle SQL Worksheet window. The code area contains:

```
1 SELECT *
2 FROM my_contacts;
3
```

The results area shows the output of the SELECT query:

ID	NAME
1	Lauran Serhal
2	Nancy Greenberg
4	Valli Pataballa

3 rows selected

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

- j) remove 프로시저를 사용하여 테이블을 삭제하고 MY_CONTACTS 테이블에 대해 설명합니다.

코드 및 결과가 다음과 같이 표시됩니다.

```

1 EXECUTE table_pkg.remove('my_contacts')
2 DESCRIBE my_contacts
3

anonymous block completed
DESCRIBE my_contacts
ERROR:
-----
ERROR: object MY_CONTACTS does not exist

1 rows selected

```

- 2) 스키마에서 PL/SQL 코드를 컴파일하는 COMPILE_PKG 패키지를 생성합니다.

- a) Spec에서 컴파일될 PL/SQL 프로그램 단위의 이름을 사용하는 MAKE라는 패키지 프로시저를 생성합니다.

/home/oracle/labs/plpu/solns/sol_06_02_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Spec을 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PACKAGE compile_pkg IS
  PROCEDURE make(p_name VARCHAR2);
END compile_pkg;
/
SHOW ERRORS

```

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

The screenshot shows the Oracle SQL Developer interface. In the main editor window, the following PL/SQL code is displayed:

```

1 CREATE OR REPLACE PACKAGE compile_pkg IS
2   PROCEDURE make(p_name VARCHAR2);
3 END compile_pkg;
4 /
5 SHOW ERRORS
6

```

Below the code, the results pane shows the output of the compilation:

```

PACKAGE compile_pkg Compiled.
No Errors.

```

b) Package Body에 다음을 포함합니다.

- i) 이 연습의 단계 1에 있는 TABLE_PKG 프로시저에 사용된 EXECUTE 프로시저
- ii) 데이터 덕셔너리에서 PL/SQL 객체 유형을 확인하기 위한 GET_TYPE이라는 전용(private) 함수
 - 이 함수는 객체가 존재할 경우 유형 이름을 반환하고(Body 가 있는 패키지에는 PACKAGE 사용), 그렇지 않을 경우에는 NULL 을 반환합니다.
 - WHERE 절 조건에서 조건에 다음을 추가하여 이름이 PACKAGE(PACKAGE BODY 가 포함될 수도 있음)를 나타낼 경우 한 행만 반환되도록 합니다. 이 경우 전체 패키지만 컴파일할 수 있으며 Spec 또는 Body 를 별도의 구성 요소로 컴파일할 수는 없습니다.
- iii) 다음 정보를 사용하여 MAKE 프로시저를 생성합니다.
 - MAKE 프로시저는 객체 이름을 나타내는 name 이라는 인수 하나만 받아들입니다.
 - MAKE 프로시저는 GET_TYPE 함수를 호출해야 합니다. 객체가 있을 경우 MAKE 는 ALTER 문을 사용하여 객체를 동적으로 컴파일합니다.

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

/home/oracle/labs/plpu/solns/sol_06_02_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Body 를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PACKAGE BODY compile_pkg IS

PROCEDURE execute(p_stmt VARCHAR2) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_stmt);
    EXECUTE IMMEDIATE p_stmt;
END;

FUNCTION get_type(p_name VARCHAR2) RETURN VARCHAR2 IS
    v_proc_type VARCHAR2(30) := NULL;
BEGIN
    /*
     * The ROWNUM = 1 is added to the condition
     * to ensure only one row is returned if the
     * name represents a PACKAGE, which may also
     * have a PACKAGE BODY. In this case, we can
     * only compile the complete package, but not
     * the specification or body as separate
     * components.
    */
    SELECT object_type INTO v_proc_type
    FROM user_objects
    WHERE object_name = UPPER(p_name)
    AND ROWNUM = 1;
    RETURN v_proc_type;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN NULL;
END;

PROCEDURE make(p_name VARCHAR2) IS
    v_stmt          VARCHAR2(100);
    v_proc_type    VARCHAR2(30) := get_type(p_name);
BEGIN
    IF v_proc_type IS NOT NULL THEN
        v_stmt := 'ALTER ' || v_proc_type || ' ' || p_name || '
COMPILE';
        execute(v_stmt);
    ELSE
        RAISE_APPLICATION_ERROR(-20001,
            'Subprogram '''|| p_name ||''' does not exist');
    END IF;
    END make;
END compile_pkg;
/
SHOW ERRORS

```

연습 해답 6-1: Native Dynamic SQL 사용 (계속)

```
Results Script Output Explain Autotrace DBMS Output OWA Output
PACKAGE BODY compile_pkg Compiled.
No Errors.
```

c) COMPILE_PKG.MAKE 프로시저를 사용하여 다음을 컴파일합니다.

- i) EMPLOYEE_REPORT 프로시저
- ii) EMP_PKG 패키지
- iii) EMP_DATA라는 존재하지 않는 객체

/home/oracle/labs/plpu/solns/sol_06_02_c.sql
 스크립트에서 파일을 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5)
 아이콘을 눌러 패키지의 프로시저를 실행합니다. 코드 및 결과가 다음과
 같이 표시됩니다.

```
SET SERVEROUTPUT ON
EXECUTE compile_pkg.make('employee_report')
EXECUTE compile_pkg.make('emp_pkg')
EXECUTE compile_pkg.make('emp_data')
```

```
1 SET SERVEROUTPUT ON
2
3 EXECUTE compile_pkg.make('employee_report')
4 EXECUTE compile_pkg.make('emp_pkg')
5 EXECUTE compile_pkg.make('emp_data')
6

anonymous block completed
ALTER PROCEDURE employee_report COMPILE

anonymous block completed
ALTER PACKAGE emp_pkg COMPILE

Error starting at line 5 in command:
EXECUTE compile_pkg.make('emp_data')
Error report:
ORA-20001: Subprogram 'emp_data' does not exist
ORA-06512: at "ORA61.COMPILE_PKG", line 39
ORA-06512: at line 1
```

단원 7 의 연습 및 해답

이 연습에서는 지정된 부서의 사원을 대량으로 패치(fetch)하는 패키지를 작성합니다. 데이터는 패키지의 PL/SQL 테이블에 저장됩니다. 또한 테이블 내용을 표시하는 프로시저를 제공합니다. 아울러 새 사원을 삽입하는 add_employee 프로시저를 생성합니다. 이 프로시저는 레코드가 성공적으로 추가되었는지 여부와 관계없이 add_employee 프로시저가 호출될 때마다 로컬 독립 서브 프로그램을 사용하여 로그 레코드를 기록합니다.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 7-1: 대량 바인드 및 독립 트랜잭션 사용

이 연습에서는 지정된 부서의 사원을 대량으로 패치(fetch)하는 패키지를 작성합니다. 데이터는 패키지의 PL/SQL 테이블에 저장됩니다. 또한 테이블 내용을 표시하는 프로시저를 제공합니다. 아울러 새 사원을 삽입하는 add_employee 프로시저를 생성합니다. 이 프로시저는 레코드가 성공적으로 추가되었는지 여부와 관계없이 add_employee 프로시저가 호출될 때마다 로컬 독립 서브 프로그램을 사용하여 로그 레코드를 기록합니다.

- 1) 지정한 부서의 사원을 query 하는 새 프로시저로 EMP_PKG 패키지를 개신합니다.
 - a) Package Spec에서 다음을 수행합니다.
 - i) dept_id라는 파라미터를 사용하여 employees.department_id 열 유형을 기준으로 하는 get_employees 프로시저를 선언합니다.
 - ii) 인덱스화된 PL/SQL 유형을 TABLE OF EMPLOYEES%ROWTYPE 으로 정의합니다.
 - b) Package Body에서 다음을 수행합니다.
 - i) 사원 레코드를 보관하기 위해 Spec에 정의된 유형을 기준으로 emp_table이라는 전용(private) 변수를 정의합니다.
 - ii) 테이블로 데이터를 대량 패치(fetch)하는 get_employees 프로시저를 구현합니다.
 - c) Spec 및 Body에서 인수가 없는 show_employees라는 새 프로시저를 생성합니다. 프로시저는 전용(private) PL/SQL 테이블 변수의 내용을 표시합니다(데이터가 있는 경우). 이 경우 이전 연습에서 생성한 print_employee 프로시저를 사용하십시오. 결과를 보려면 SQL Developer의 DBMS Output 탭에서 Enable DBMS Output 아이콘을 누릅니다(아직 누르지 않은 경우).
 - d) SERVEROUTPUT 을 활성화합니다. 부서 30에 대해 emp_pkg.get_employees 프로시저를 호출한 다음 emp_pkg.show_employees를 호출합니다. 부서 60에 대해서도 이 작업을 반복합니다.

연습 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

- 2) 상사는 패키지에 포함된 add_employee 프로시저가 호출되어 EMPLOYEES 테이블에 새로운 사원이 삽입될 때마다 로그를 보존하고자 할 수도 있습니다.
- 먼저 /home/oracle/labs/plpu/solns/sol_07_02_a.sql 스크립트를 로드하고 실행하여 LOG_NEWEMP 로그 테이블과 log_newemp_seq 시퀀스를 생성합니다.
 - EMP_PKG Package Body에서 실제 INSERT 작업을 수행하는 add_employee 프로시저를 수정합니다. 다음과 같이 audit_newemp라는 로컬 프로시저를 추가합니다.
 - audit_newemp 프로시저는 독립 트랜잭션을 사용하여 LOG_NEWEMP 테이블에 로그 레코드를 삽입해야 합니다.
 - 로그 테이블 행에 USER, 현재 시간 및 새로운 사원의 이름을 저장합니다.
 - log_newemp_seq를 사용하여 entry_id 열을 설정합니다.

참고: COMMIT 작업은 독립 트랜잭션을 사용하는 프로시저에서 수행하십시오.

- 삽입 작업을 수행하기 전에 audit_emp를 호출하도록 add_employee 프로시저를 수정합니다.
- 부서 20에 근무하는 Max Smart와 부서 10에 근무하는 Clark Kent에 대해 add_employee 프로시저를 호출합니다. 어떤 결과가 나타납니까?
- 추가된 두 EMPLOYEES 레코드와 LOG_NEWEMP 테이블의 레코드를 query합니다. 몇 개의 로그 레코드가 있습니까?
- ROLLBACK 문을 실행하여 커밋되지 않은 삽입 작업을 언두합니다. 다음과 같이 단계 2e에서와 같은 query를 사용합니다.
 - 첫번째 query를 사용하여 Smart 및 Kent에 대한 사원 행이 제거되었는지를 검사합니다.
 - 두번째 query를 사용하여 LOG_NEWEMP 테이블에서 로그 레코드를 검사합니다. 몇 개의 로그 레코드가 있습니까? 그 이유는 무엇입니까?

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용

이 연습에서는 지정된 부서의 사원을 대량으로 패치(fetch)하는 패키지를 작성합니다. 데이터는 패키지의 PL/SQL 테이블에 저장됩니다. 또한 테이블 내용을 표시하는 프로시저를 제공합니다. 아울러 새 사원을 삽입하는 add_employee 프로시저를 생성합니다. 이 프로시저는 레코드가 성공적으로 추가되었는지 여부와 관계없이 add_employee 프로시저가 호출될 때마다 로컬 독립 서브 프로그램을 사용하여 로그 레코드를 기록합니다.

1) 지정한 부서의 사원을 query 하는 새 프로시저로 EMP_PKG 패키지를 생성합니다.

a) Package Spec에서 다음을 수행합니다.

- i) dept_id라는 파라미터를 사용하여 employees.department_id 열 유형을 기준으로 하는 get_employees 프로시저를 선언합니다.
- ii) 인덱스화된 PL/SQL 유형을 TABLE OF EMPLOYEES%ROWTYPE으로 정의합니다.

/home/oracle/labs/plpu/solns/sol_07_01_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Spec 을 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다. 새로 추가된 코드는 아래 코드 상자에서 굵은체 글자로 강조 표시됩니다.

```

CREATE OR REPLACE PACKAGE emp_pkg IS

    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

END emp_pkg;
/
SHOW ERRORS

```



b) Package Body에서 다음을 수행합니다.

- i) 사원 레코드를 보관하기 위해 Spec에 정의된 유형을 기준으로 emp_table이라는 전용(private) 변수를 정의합니다.
- ii) 테이블로 데이터를 대량 패치(fetch)하는 get_employees 프로시저를 구현합니다.

/home/oracle/labs/plpu/solns/sol_07_01_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 Package Body를 생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다. 새로 추가된 코드는 아래 코드 상자에서 굵은체 글자로 강조 표시됩니다.

```

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
    valid_departments boolean_tab_type;
emp_table          emp_tab_type;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

p_comm employees.commission_pct%TYPE DEFAULT 0,
p_deptid employees.department_id%TYPE DEFAULT 30) IS
BEGIN
    IF valid_deptid(p_deptid) THEN

        INSERT INTO employees(employee_id, first_name,
last_name, email,
                job_id, manager_id, hire_date, salary,
commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%type;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p.job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype IS

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

/* New get_employees procedure. */

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||
                         p_rec_emp.salary);
END;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE) RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

The screenshot shows the Oracle SQL Developer interface with the 'Results' tab selected. The output window displays the message: 'PACKAGE BODY emp_pkg Compiled. No Errors.'.

- c) Spec 및 Body에서 인수가 없는 show_employees라는 새 프로시저를 생성합니다. 프로시저는 전용(private) PL/SQL 테이블 변수의 내용을 표시합니다(데이터가 있는 경우). 이 경우 이전 연습에서 생성한 print_employee 프로시저를 사용하십시오. 결과를 보려면 SQL Developer의 DBMS Output 탭에서 Enable DBMS Output 아이콘을 누릅니다(아직 누르지 않은 경우).

/home/oracle/labs/plpu/solns/sol_07_01_c.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 새 프로시저로 패키지를 재생성하고 컴파일합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS
    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p_emp_id employees.employee_id%type)
        return employees%rowtype;

    FUNCTION get_employee(p_family_name
        employees.last_name%type)
        return employees%rowtype;
```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  TYPE boolean_tab_type IS TABLE OF BOOLEAN
    INDEX BY BINARY_INTEGER;

  valid_departments boolean_tab_type;
  emp_table          emp_tab_type;
  FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email      employees.email%TYPE,
    p_job        employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr        employees.manager_id%TYPE DEFAULT 145,
    p_sal        employees.salary%TYPE DEFAULT 1000,
    p_comm       employees.commission_pct%TYPE DEFAULT 0,
    p_deptid     employees.department_id%TYPE DEFAULT 30) IS
BEGIN
  IF valid_deptid(p_deptid) THEN
    INSERT INTO employees(employee_id, first_name,
last_name, email,
           job_id, manager_id, hire_date, salary,
commission_pct, department_id)
      VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
           p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
      RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
    END IF;
  END add_employee;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid   employees.department_id%TYPE) IS
    p_email employees.email%type;

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS
BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p.empid;
END get_employee;

FUNCTION get_employee(p.emp_id employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p.emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p.family_name;
    RETURN rec_emp;
END;

PROCEDURE get_employees(p.dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p.dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id || ' ' ||
                         p_rec_emp.first_name || ' ' ||
                         p_rec_emp.last_name || ' ' ||
                         p_rec_emp.job_id || ' ' ||
                         p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```



연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

- d) SERVEROUTPUT 을 활성화합니다. 부서 30에 대해 emp_pkg.get_employees 프로시저를 호출한 다음 emp_pkg.show_employees 를 호출합니다. 부서 60에 대해서도 이 작업을 반복합니다.

/home/oracle/labs/plpu/solns/sol_07_01_d.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 패키지의 프로시저를 호출합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SET SERVEROUTPUT ON

EXECUTE emp_pkg.get_employees(30)
EXECUTE emp_pkg.show_employees

EXECUTE emp_pkg.get_employees(60)
EXECUTE emp_pkg.show_employees
```

```
anonymous block completed
Employees in Package table
30 114 Den Raphaely PU_MAN 11000
30 115 Alexander Khoo PU_CLERK 3100
30 116 Shelli Baida PU_CLERK 2900
30 117 Sigal Tobias PU_CLERK 2800
30 118 Guy Himuro PU_CLERK 2600
30 119 Karen Colmenares PU_CLERK 2500
30 209 Samuel Joplin SA_REP 1000

anonymous block completed
anonymous block completed
Employees in Package table
60 103 Alexander Hunold IT_PROG 9000
60 104 Bruce Ernst IT_PROG 6000
60 105 David Austin IT_PROG 4800
60 106 Valli Pataballa IT_PROG 4800
60 107 Diana Lorentz IT_PROG 4200
```

- 2) 상사는 패키지에 포함된 add_employee 프로시저가 호출되어 EMPLOYEES 테이블에 새로운 사원이 삽입될 때마다 로그를 보존하고자 할 수도 있습니다.
 - a) 먼저, /home/oracle/labs/plpu/solns/sol_07_02_a.sql 스크립트를 로드하고 실행하여 LOG_NEWEMP 로그 테이블과 log_newemp_seq 시퀀스를 생성합니다.

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

/home/oracle/labs/plpu/solns/sol_07_02_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE TABLE log_newemp (
    entry_id NUMBER(6) CONSTRAINT log_newemp_pk PRIMARY KEY,
    user_id   VARCHAR2(30),
    log_time  DATE,
    name      VARCHAR2(60)
);

CREATE SEQUENCE log_newemp_seq;
```



- b) EMP_PKG Package Body에서 실제 INSERT 작업을 수행하는 add_employee 프로시저를 수정합니다. 다음과 같이 audit_newemp라는 로컬 프로시저를 추가합니다.
- audit_newemp 프로시저는 독립 트랜잭션을 사용하여 LOG_NEWEMP 테이블에 로그 레코드를 삽입해야 합니다.
 - 로그 테이블 행에 USER, 현재 시간 및 새로운 사원의 이름을 저장합니다.
 - log_newemp_seq를 사용하여 entry_id 열을 설정합니다.

참고: COMMIT 작업은 독립 트랜잭션을 사용하는 프로시저에서 수행하십시오.

/home/oracle/labs/plpu/solns/sol_07_02_b.sql 스크립트를 엽니다. 새로 추가된 코드는 다음 코드 상자에서 굵은체 글자로 강조 표시됩니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;
```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p.emp_id
employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p.family_name
employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p.dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;

    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
        RETURN BOOLEAN;

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30) IS

-- New local procedure

PROCEDURE audit_newemp IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    user_id VARCHAR2(30) := USER;
BEGIN
    INSERT INTO log_newemp (entry_id, user_id, log_time,
                           name)
    VALUES (log_newemp_seq.NEXTVAL, user_id,
            sysdate, p_first_name || ' ' || p_last_name);
    COMMIT;
END audit_newemp;

BEGIN -- add_employee
    IF valid_deptid(p_deptid) THEN
        INSERT INTO employees(employee_id, first_name,
last_name, email,
                     job_id, manager_id, hire_date, salary,
commission_pct, department_id)
        VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                 p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
    ELSE
        RAISE_APPLICATION_ERROR (-20204, 'Invalid
department ID. Try again.');
    END IF;
END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE) IS
    p_email employees.email%TYPE;
BEGIN
    p_email := UPPER(SUBSTR(p_first_name, 1,
1) || SUBSTR(p_last_name, 1, 7));
    add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
END;

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE) IS

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

BEGIN
    SELECT salary, job_id
    INTO p_sal, p_job
    FROM employees
    WHERE employee_id = p_empid;
END get_employee;

FUNCTION get_employee(p_emp_id
employees.employee_id%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

/* New get_employees procedure. */

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype)
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id|| ' ' ||
                         p_rec_emp.first_name|| ' ' ||
                         p_rec_emp.last_name|| ' ' ||
                         p_rec_emp.job_id|| ' ' ||

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

        p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

```

The screenshot shows the Oracle SQL Worksheet interface with the following details:

- Toolbar icons: Results, Script Output, Explain, Autotrace, DBMS Output, OWA Output.
- Script Editor area: Contains the PL/SQL code for the package and its body.
- Output Area: Displays the compilation results:
 - PACKAGE emp_pkg Compiled.
 - No Errors.
 - PACKAGE BODY emp_pkg Compiled.
 - No Errors.

- c) 삽입 작업을 수행하기 전에 audit_emp 를 호출하도록 add_employee 프로시저를 수정합니다.

/home/oracle/labs/plpu/solns/sol_07_02_c.sql 스크립트를 엽니다. 새로 추가된 코드는 다음 코드 상자에서 굵은체 글자로 강조 표시됩니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

    TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA_REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_deptid employees.department_id%TYPE DEFAULT 30);

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE);

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE);

    FUNCTION get_employee(p_emp_id employees.employee_id%type)
        return employees%rowtype;

    FUNCTION get_employee(p_family_name
employees.last_name%type)
        return employees%rowtype;

    PROCEDURE get_employees(p_dept_id
employees.department_id%type);

    PROCEDURE init_departments;

    PROCEDURE print_employee(p_rec_emp employees%rowtype);

    PROCEDURE show_employees;

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;
```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

valid_departments boolean_tab_type;
emp_table          emp_tab_type;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email     employees.email%TYPE,
    p_job       employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr       employees.manager_id%TYPE DEFAULT 145,
    p_sal       employees.salary%TYPE DEFAULT 1000,
    p_comm      employees.commission_pct%TYPE DEFAULT 0,
    p_deptid   employees.department_id%TYPE DEFAULT 30) IS

PROCEDURE audit_newemp IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    user_id VARCHAR2(30) := USER;
BEGIN
    INSERT INTO log_newemp (entry_id, user_id, log_time,
name)
        VALUES (log_newemp_seq.NEXTVAL, user_id,
sysdate,p_first_name||' '||p_last_name);
    COMMIT;
END audit_newemp;

BEGIN -- add_employee
    IF valid_deptid(p_deptid) THEN
        audit_newemp;
        INSERT INTO employees(employee_id, first_name,
last_name, email,
        job_id, manager_id, hire_date, salary,
commission_pct, department_id)
            VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
            p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
        END IF;
    END add_employee;

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid   employees.department_id%TYPE) IS
    p_email     employees.email%TYPE;
BEGIN

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

        p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
        add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
    END;

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p_job OUT employees.job_id%TYPE) IS
    BEGIN
        SELECT salary, job_id
        INTO p_sal, p_job
        FROM employees
        WHERE employee_id = p.empid;
    END get_employee;

    FUNCTION get_employee(p_emp_id employees.employee_id%type)
        return employees%rowtype IS
        rec_emp employees%rowtype;
    BEGIN
        SELECT * INTO rec_emp
        FROM employees
        WHERE employee_id = p_emp_id;
        RETURN rec_emp;
    END;

    FUNCTION get_employee(p_family_name
employees.last_name%type)
        return employees%rowtype IS
        rec_emp employees%rowtype;
    BEGIN
        SELECT * INTO rec_emp
        FROM employees
        WHERE last_name = p_family_name;
        RETURN rec_emp;
    END;

    PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
    BEGIN
        SELECT * BULK COLLECT INTO emp_table
        FROM EMPLOYEES
        WHERE department_id = p_dept_id;
    END;

    PROCEDURE init_departments IS
    BEGIN
        FOR rec IN (SELECT department_id FROM departments)
        LOOP
            valid_departments(rec.department_id) := TRUE;
        END LOOP;
    END;

```

연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

```

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
                         p_rec_emp.employee_id || ' ' ||
                         p_rec_emp.first_name || ' ' ||
                         p_rec_emp.last_name || ' ' ||
                         p_rec_emp.job_id || ' ' ||
                         p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;
BEGIN
    init_departments;
END emp_pkg;
/
SHOW ERRORS

```

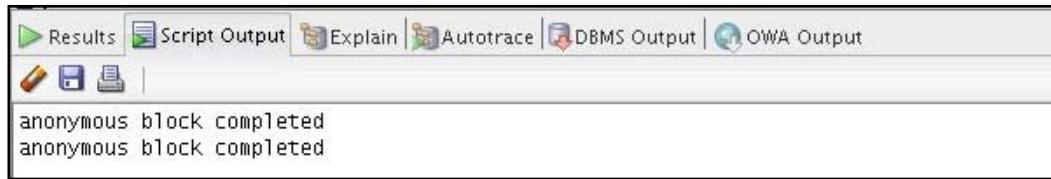


연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

- d) 부서 20에 근무하는 Max Smart 와 부서 10에 근무하는 Clark Kent에 대해 add_employee 프로시저를 호출합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_07_02_d.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과는 다음과 같습니다.

```
EXECUTE emp_pkg.add_employee('Max', 'Smart', 20)
EXECUTE emp_pkg.add_employee('Clark', 'Kent', 10)
```



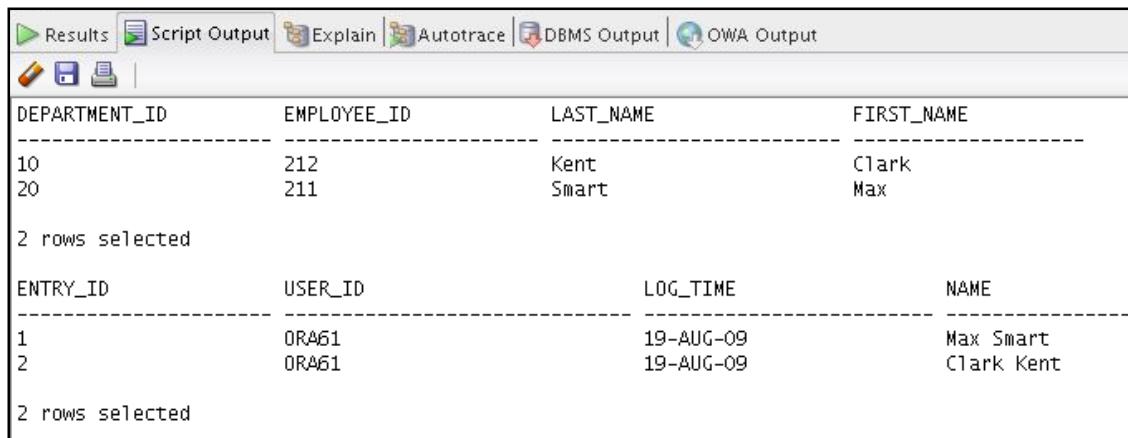
두 insert 문 모두 성공적으로 완료됩니다. 다음 단계에서처럼 로그 테이블에 두 개의 로그 레코드가 있습니다.

- e) 추가된 두 EMPLOYEES 레코드와 LOG_NEWEMP 테이블의 레코드를 query 합니다. 몇 개의 로그 레코드가 있습니까?

/home/oracle/labs/plpu/solns/sol_07_02_e.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
select department_id, employee_id, last_name, first_name
from employees
where last_name in ('Kent', 'Smart');

select * from log_newemp;
```



연습 해답 7-1: 대량 바인드 및 독립 트랜잭션 사용 (계속)

두 개의 로그 레코드가 있는데, 하나는 Smart 에 대한 레코드이고 다른 하나는 Kent 에 대한 레코드입니다.

- f) ROLLBACK 문을 실행하여 커밋되지 않은 삽입 작업을 언두합니다. 다음과 같이 단계 2 e에서와 같은 query 를 사용합니다.
- 첫번째 query 를 사용하여 Smart 및 Kent 에 대한 사원 행이 제거되었는지를 검사합니다.
 - 두번째 query 를 사용하여 LOG_NEWEMP 테이블에서 로그 레코드를 검사합니다. 몇 개의 로그 레코드가 있습니까? 그 이유는 무엇입니까?

ROLLBACK ;



```

1 select department_id, employee_id, last_name, first_name
2 from employees
3 where last_name in ('Kent', 'Smart');
4
5 select * from log_newemp;
6

```

DEPARTMENT_ID	EMPLOYEE_ID	LAST_NAME	FIRST_NAME

0 rows selected

ENTRY_ID	USER_ID	LOG_TIME	NAME
1	ORA61	18-JUN-09	Max Smart
2	ORA61	18-JUN-09	Clark Kent

2 rows selected

두 개의 사원 레코드가 제거(롤백)되었습니다. 그러나 두 로그 레코드는 주 트랜잭션에서 수행된 롤백 작업의 영향을 받지 않는 독립 트랜잭션을 사용하여 삽입되었기 때문에 로그 테이블에 남아 있습니다.

단원 8 의 연습 및 해답

이 연습에서는 문장 및 행 트리거를 생성합니다. 또한 트리거 내에서 호출되는 프로시저도 생성합니다.

참고: 연습에서 건너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 8-1: 문장 및 행 트리거 생성

이 연습에서는 문장 및 행 트리거를 생성합니다. 또한 트리거 내에서 호출되는 프로시저도 생성합니다.

- 1) JOBS 테이블의 행에는 여러 JOB_ID 값에 허용되는 최대 급여와 최소 급여가 저장됩니다. 삽입과 갱신 작업을 위해 사원의 급여가 그들의 직무 유형에 허용되는 범위 내에 있도록 하는 코드를 작성하라는 요청을 받았습니다.
 - a) 다음과 같이 CHECK_SALARY 라는 프로시저를 생성합니다.
 - i) 프로시저는 두 개의 파라미터, 즉 사원의 직무 ID 문자열에 대한 파라미터와 급여에 대한 파라미터를 그대로 적용합니다.
 - ii) 프로시저는 직무 ID를 사용하여 지정된 직무의 최소 급여와 최대 급여를 결정합니다.
 - iii) 급여 파라미터가 직무의 급여 범위(최소 급여 및 최대 급여 포함)에 속하지 않을 경우 "Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>"(이)라는 메시지와 함께 응용 프로그램 예외가 발생해야 합니다. 메시지의 여러 항목을 query에 의해 채워지는 파라미터와 변수가 제공하는 값으로 바꿉니다. 파일을 저장합니다.
 - b) 각 행에 대한 INSERT 또는 UPDATE 작업 이전에 실행되는 CHECK_SALARY_TRG 라는 트리거를 EMPLOYEES 테이블에 생성합니다.
 - i) 트리거는 CHECK_SALARY 프로시저를 호출하여 업무 논리를 수행해야 합니다.
 - ii) 트리거는 새로운 직무 ID 와 급여를 프로시저 파라미터에 전달해야 합니다.
- 2) 다음과 같은 경우를 통해 CHECK_SAL_TRG 트리거를 테스트합니다.
 - a) EMP_PKG.ADD_EMPLOYEE 프로시저를 사용하여 부서 30에 Eleanor Beh Beh라는 사원을 추가합니다. 어떤 결과가 발생하며 그 이유는 무엇입니까?
 - b) 사원 115의 급여를 \$2,000로 갱신합니다. 별도의 갱신 작업에서 사원의 직무 ID를 HR_REP로 변경합니다. 각 경우에 어떤 결과가 나타납니까?
 - c) 사원 115의 급여를 \$2,800로 갱신합니다. 어떤 결과가 나타납니까?

연습 8-1: 문장 및 행 트리거 생성 (계속)

- 3) 직무 ID 나 급여 값이 실제로 변경된 경우에만 실행되도록 CHECK_SALARY_TRG 트리거를 생성합니다.
- JOB_ID 또는 SALARY 값이 변경되었는지 여부를 확인하도록 WHEN 절을 사용하여 업무 규칙을 구현합니다.
- 참고:** INSERT 작업이 수행될 경우 조건이 OLD.column_name 값의 NULL을 처리하도록 지정하십시오. 그렇지 않으면 삽입 작업이 실패하게 됩니다.
- 다음 파라미터 값으로 EMP_PKG.ADD_EMPLOYEE 프로시저를 실행하여 트리거를 테스트합니다.
 - p_first_name: 'Eleanor'
 - p_last_name: 'Beh'
 - p_Email: 'EBEH'
 - p_Job: 'IT_PROG'
 - p_Sal: 5000
 - 직무가 IT_PROG 인 사원의 급여를 \$2,000 씩 인상하여 생성합니다.
어떤 결과가 나타납니까?
 - Eleanor Beh 의 급여를 \$9,000 로 생성합니다.

힌트: WHERE 절의 subquery 와 함께 UPDATE 문을 사용하십시오. 어떤 결과가 나타납니까?

- subquery 를 포함한 다른 UPDATE 문을 사용하여 Eleanor Beh 의 직무를 ST_MAN 으로 변경합니다. 어떤 결과가 나타납니까?
- 4) 업무 시간 동안 사원이 삭제되지 않게 하라는 요청을 받았습니다.
- 평일 업무 시간(오전 9:00 - 오후 6:00) 중에는 행이 삭제되지 않도록 하는 DELETE_EMP_TRG 라는 문장 트리거를 EMPLOYEES 테이블에 작성합니다.
 - 소속 부서가 없는 JOB_ID 가 SA REP 인 사원을 삭제해 봅니다.

힌트: 이는 ID 가 178 인 Grant라는 사원입니다.

연습 해답 8-1: 문장 및 행 트리거 생성

이 연습에서는 문장 및 행 트리거를 생성합니다. 또한 트리거 내에서 호출되는 프로시저도 생성합니다.

- 1) JOBS 테이블의 행에는 여러 JOB_ID 값에 허용되는 최대 급여와 최소 급여가 저장됩니다. 삽입과 갱신 작업을 위해 사원의 급여가 그들의 직무 유형에 허용되는 범위 내에 있도록 하는 코드를 작성하라는 요청을 받았습니다.
 - a) 다음과 같이 CHECK_SALARY라는 프로시저를 생성합니다.
 - i) 프로시저는 두 개의 파라미터, 즉 사원의 직무 ID 문자열에 대한 파라미터와 급여에 대한 파라미터를 그대로 적용합니다.
 - ii) 프로시저는 직무 ID를 사용하여 지정된 직무의 최소 급여와 최대 급여를 결정합니다.
 - iii) 급여 파라미터가 직무의 급여 범위(최소 급여 및 최대 급여 포함)에 속하지 않을 경우 "Invalid salary <sal>. Salaries for job <jobid> must be between <min> and <max>"(이)라는 메시지와 함께 응용 프로그램 예외가 발생해야 합니다. 메시지의 여러 항목을 query에 의해 채워지는 파라미터와 변수가 제공하는 값으로 바꿉니다. 파일을 저장합니다.

/home/oracle/labs/plpu/solns/sol_08_01_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PROCEDURE check_salary (p_the_job
VARCHAR2, p_the_salary NUMBER) IS
    v_minsal jobs.min_salary%type;
    v_maxsal jobs.max_salary%type;
BEGIN
    SELECT min_salary, max_salary INTO v_minsal, v_maxsal
    FROM jobs
    WHERE job_id = UPPER(p_the_job);
    IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
        RAISE_APPLICATION_ERROR(-20100,
        'Invalid salary $' || p_the_salary || '. ' ||
        'Salaries for job ' || p_the_job ||
        ' must be between $' || v_minsal || ' and $' ||
        v_maxsal);
    END IF;
END;
/
SHOW ERRORS

```

연습 해답 8-1: 문장 및 행 트리거 생성 (계속)

```
PROCEDURE check_salary Compiled.
No Errors.
```

- b) 각 행에 대한 INSERT 또는 UPDATE 작업 이전에 실행되는 CHECK_SALARY_TRG라는 트리거를 EMPLOYEES 테이블에 생성합니다.
- 트리거는 CHECK_SALARY 프로시저를 호출하여 업무 끝리를 수행해야 합니다.
 - 트리거는 새로운 직무 ID 와 급여를 프로시저 파라미터에 전달해야 합니다.

/home/oracle/labs/plpu/solns/sol_08_01_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees
FOR EACH ROW
BEGIN
    check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

```
TRIGGER check_salary_trg Compiled.
No Errors.
```

- 2) 다음과 같은 경우를 통해 CHECK_SAL_TRG 트리거를 테스트합니다.

- a) EMP_PKG.ADD_EMPLOYEE 프로시저를 사용하여 부서 30에 Eleanor Beh Beh라는 사원을 추가합니다. 어떤 결과가 발생하며 그 이유는 무엇입니까?

/home/oracle/labs/plpu/solns/sol_08_02_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
```

연습 해답 8-1: 문장 및 행 트리거 생성 (계속)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | | | | |
Error starting at line 1 in command:
EXECUTE emp_pkg.add_employee('Eleanor', 'Beh', 30)
Error report:
ORA-20100: Invalid salary $1000. Salaries for job SA_REP must be between $6000 and $12000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 35
ORA-06512: at "ORA61.EMP_PKG", line 51
ORA-06512: at line 1

```

EMP_PKG.ADD_EMPLOYEE 프로시저는 기본 급여 \$1,000 와 기본 직무 ID SA_REP 를 사용하는 오버로드 버전을 호출하기 때문에 트리거에 예외가 발생합니다. 그러나 JOBS 테이블에는 SA_REP 유형에 대한 최소 급여 \$6,000 가 저장됩니다.

- b) 사원 115 의 급여를 \$2,000 로 갱신합니다. 별도의 갱신 작업에서 사원의 직무 ID 를 HR_REP 로 변경합니다. 각 경우에 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_08_02_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

UPDATE employees
  SET salary = 2000
 WHERE employee_id = 115;

UPDATE employees
  SET job_id = 'HR_REP'
 WHERE employee_id = 115;

```

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | | | | |
Error starting at line 1 in command:
UPDATE employees
  SET salary = 2000
 WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $2000. Salaries for job PU_CLERK must be between $2500 and $5500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'

Error starting at line 5 in command:
UPDATE employees
  SET job_id = 'HR_REP'
 WHERE employee_id = 115
Error report:
SQL Error: ORA-20100: Invalid salary $3100. Salaries for job HR_REP must be between $4000 and $9000
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'

```

연습 해답 8-1: 문장 및 행 트리거 생성 (계속)

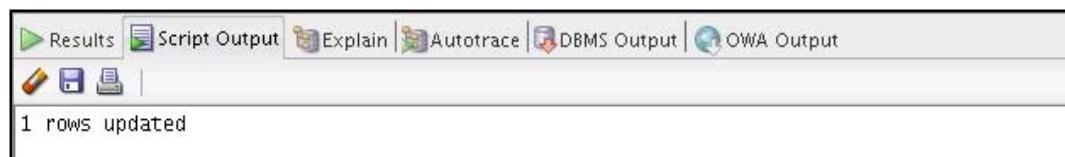
첫번째 update 문에서는 급여가 \$2,000로 설정되지 않습니다. 사원 115의 새로운 급여가 PU_CLERK 직무 ID에 허용되는 최소 급여보다 적기 때문에 급여 검사 트리거 규칙이 갱신 작업에 실패합니다.

두번째 update 문에서는 현재 사원의 급여 \$3,100가 새 HR_REP 직무 ID의 최소 급여보다 적기 때문에 사원의 직무가 변경되지 않습니다.

- 사원 115의 급여를 \$2,800로 갱신합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_08_02_c.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
UPDATE employees
   SET salary = 2800
 WHERE employee_id = 115;
```



새 급여가 현재 직무 ID의 허용 범위 내에 있기 때문에 갱신 작업이 성공합니다.

- 직무 ID나 급여 값이 실제로 변경된 경우에만 실행되도록 CHECK_SALARY_TRG 트리거를 갱신합니다.
- JOB_ID 또는 SALARY 값이 변경되었는지 여부를 확인하도록 WHEN 절을 사용하여 업무 규칙을 구현합니다.

참고: INSERT 작업이 수행될 경우 조건이 OLD.column_name
값의 NULL을 처리하도록 지정하십시오. 그렇지 않으면 삽입 작업이 실패하게 됩니다.

연습 해답 8-1: 문장 및 행 트리거 생성(계속)

/home/oracle/labs/plpu/solns/sol_08_03_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE TRIGGER check_salary_trg
BEFORE INSERT OR UPDATE OF job_id, salary
ON employees FOR EACH ROW
WHEN (new.job_id <> NVL(old.job_id,'?') OR
      new.salary <> NVL(old.salary,0))
BEGIN
    check_salary(:new.job_id, :new.salary);
END;
/
SHOW ERRORS
```

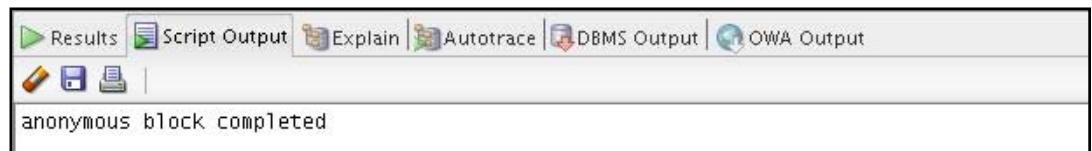


- b) 다음 파라미터 값으로 EMP_PKG.ADD_EMPLOYEE 프로시저를 실행하여 트리거를 테스트합니다.

- p_first_name: 'Eleanor'
- p_last_name: 'Beh'
- p_Email: 'EBEH'
- p_Job: 'IT_PROG'
- p_Sal: 5000

/home/oracle/labs/plpu/solns/sol_08_03_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
BEGIN
    emp_pkg.add_employee('Eleanor', 'Beh', 'EBEH',
                          job => 'IT_PROG', sal => 5000);
END;
/
```



연습 해답 8-1: 문장 및 행 트리거 생성 (계속)

- c) 직무가 IT_PROG 인 사원의 급여를 \$2,000 씩 인상하여 갱신합니다.
어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_08_03_c.sql 스크립트를
엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를
누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
UPDATE employees
  SET salary = salary + 2000
 WHERE job_id = 'IT_PROG';
```

The screenshot shows the Oracle SQL Worksheet interface with the 'Script Output' tab selected. The output window displays the following error message:

```
Error starting at line 1 in command:
UPDATE employees
  SET salary = salary + 2000
 WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-20100: Invalid salary $11000. Salaries for job IT_PROG must be between $4000 and $10000
ORA-06512: at "ORAG1.CHECK_SALARY", line 9
ORA-06512: at "ORAG1.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORAG1.CHECK_SALARY_TRG'
```

지정된 직무 유형에 속하는 사원의 급여가 해당 직무 유형의 최대 급여를
초과합니다. IT_PROG 직무 유형에 속하는 사원의 급여가 갱신되지 않습니다.

- d) Eleanor Beh 의 급여를 \$9,000 로 갱신합니다.

/home/oracle/labs/plpu/solns/sol_08_03_d.sql 스크립트를
엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를
누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
UPDATE employees
  SET salary = 9000
 WHERE employee_id = (SELECT employee_id
                        FROM employees
                       WHERE last_name = 'Beh');
```

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

```
1 rows updated
```

힌트: WHERE 절의 subquery 와 함께 UPDATE 문을 사용하십시오.
어떤 결과가 나타납니까?

연습 해답 8-1: 문장 및 행 트리거 생성 (계속)

- e) subquery 를 포함한 다른 UPDATE 문을 사용하여 Eleanor Beh 의 직무를 ST_MAN 으로 변경합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_08_03_e.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                      FROM employees
                     WHERE last_name = 'Beh');
```

The screenshot shows the SQL Worksheet interface with the 'Results' tab selected. The error message is displayed:

```
Error starting at line 1 in command:
UPDATE employees
  set job_id = 'ST_MAN'
WHERE employee_id = (SELECT employee_id
                      FROM employees
                     WHERE last_name = 'Beh')
Error report:
SQL Error: ORA-20100: Invalid salary $9000. Salaries for job ST_MAN must be between $5500 and $8500
ORA-06512: at "ORA61.CHECK_SALARY", line 9
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
```

새 직무 유형의 최대 급여가 이 사원의 현재 급여보다 적기 때문에 개신 작업이 실패합니다.

- 4) 업무 시간 동안 사원이 삭제되지 않게 하라는 요청을 받았습니다.

- a) 평일 업무 시간(오전 9:00 - 오후 6:00) 중에는 행이 삭제되지 않도록 하는 DELETE_EMP_TRG 라는 문장 트리거를 EMPLOYEES 테이블에 작성합니다.

/home/oracle/labs/plpu/solns/sol_08_04_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE TRIGGER delete_emp_trg
BEFORE DELETE ON employees
DECLARE
    the_day VARCHAR2(3) := TO_CHAR(SYSDATE, 'DY');
    the_hour PLS_INTEGER := TO_NUMBER(TO_CHAR(SYSDATE,
'HH24'));
BEGIN
    IF (the_hour BETWEEN 9 AND 18) AND (the_day NOT IN
('SAT','SUN')) THEN
        RAISE_APPLICATION_ERROR(-20150,
        'Employee records cannot be deleted during the
business')
```

연습 해답 8-1: 문장 및 행 트리거 생성(계속)

```

        hours of 9AM and 6PM' );
    END IF;
END;
/
SHOW ERRORS

```

The screenshot shows the Oracle SQL Worksheet interface. The tabs at the top are Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the tabs, there are three small icons: a pencil, a floppy disk, and a printer. The main area displays the following text:
TRIGGER delete_emp_trg Compiled.
No Errors.

- b) 소속 부서가 없는 JOB_ID 가 SA_REP 인 사원을 삭제해 봅니다.

힌트: 이는 ID 가 178 인 Grant 라는 사원입니다.

/home/oracle/labs/plpu/solns/sol_08_04_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

DELETE FROM employees
WHERE job_id = 'SA_REP'
    AND department_id IS NULL;

```

The screenshot shows the Oracle SQL Worksheet interface. The tabs at the top are Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the tabs, there are three small icons: a pencil, a floppy disk, and a printer. The main area displays the following text:
Error starting at line 1 in command:
DELETE FROM employees
WHERE job_id = 'SA_REP'
 AND department_id IS NULL
Error report:
SQL Error: ORA-20150: Employee records cannot be deleted during the business hours of 9AM and 6PM
ORA-06512: at "ORA61.DELETE_EMP_TRG", line 6
ORA-04088: error during execution of trigger 'ORA61.DELETE_EMP_TRG'

참고: 강의실에 있는 호스트 시스템의 현재 시간에 따라 삭제 작업을 수행할 수 있는지 여부가 결정됩니다. 예를 들어, 위 화면 캡처에서는 삭제 작업이 호스트 시스템 시간을 기준으로 허용된 업무 시간을 벗어나서 수행되었기 때문에 실패했습니다.

단원 9 의 연습 및 해답

이 연습에서는 직무에 유효한 급여 범위와 관련하여 사원 급여의 데이터 무결성을 보장하는 간단한 업무 규칙을 구현하고, 이 규칙에 대한 트리거를 생성하는 과정을 다릅니다. 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.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리

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.

- 1) 특정 직무의 최소 급여가 현재 급여보다 높은 값으로 인상될 경우 사원의 급여가 자동으로 인상됩니다. 트리거로 호출되는 패키지 프로시저를 통해 이 요구 사항을 JOBS 테이블에 구현합니다. JOBS 테이블에서 최소 급여를 갱신하고 사원 급여를 갱신하려고 하면 CHECK_SALARY 트리거가 JOBS 테이블(변경될 수 있음)을 읽으려고 하므로 변경 테이블 예외가 발생합니다. 이 문제는 새 패키지와 추가 트리거를 생성하여 변경 테이블 문제를 해결할 수 있습니다.
 - a. 다음과 같이 EMP_PKG 패키지(연습 8에서 마지막으로 갱신한 버전)를 갱신합니다.
 - i. 사원의 급여를 갱신하는 SET_SALARY라는 프로시저를 추가합니다.
 - ii. SET_SALARY 프로시저는 다음 두 파라미터를 받아들입니다. 하나는 갱신될 수 있는 급여에 대한 직무 ID이고, 다른 하나는 직무 ID에 대한 새 최소 급여입니다.
 - b. 지정된 직무 ID에 대해 JOBS 테이블에 있는 최소 급여가 갱신되면 EMP_PKG.SET_SALARY 프로시저를 호출하는 UPD_MINSALARY_TRG라는 행 트리거를 JOBS 테이블에 생성합니다.
 - c. 프로그래머(JOB_ID: 'IT_PROG')인 사원의 ID, 성, 직무 ID, 현재 급여, 최소 급여를 표시하는 query를 작성합니다. 그런 다음 JOBS 테이블에서 최소 급여가 \$1000 인상되도록 갱신합니다. 어떤 결과가 나타납니까?
- 2) 변경 테이블 문제를 해결하려면 JOBS 테이블의 행 복사본을 메모리에 유지하는 JOBS_PKG 패키지를 생성합니다. 그런 다음 예외를 피하기 위해 변경되는 테이블에 대해 query를 실행하는 대신 패키지 데이터를 사용하도록 CHECK_SALARY 프로시저를 수정합니다. 그러나 BEFORE INSERT OR UPDATE 문장 트리거를 EMPLOYEES 테이블에 생성하여 JOBS_PKG 패키지 상태를 초기화한 후 CHECK_SALARY 행 트리거를 실행해야 합니다.
 - a. 다음 Spec이 포함된 JOBS_PKG라는 새 패키지를 생성합니다.


```
PROCEDURE initialize;
FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
```

연습 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리 (계속)

```

PROCEDURE set_minsalary(jobid VARCHAR2,min_salary
                        NUMBER);
PROCEDURE set_maxsalary(jobid VARCHAR2,max_salary
                        NUMBER);

```

- b. 다음과 같이 JOBS_PKG 의 Body 를 구현합니다.
- i. JOBS.JOB_ID%TYPE 을 기준으로 문자열 유형에 의해 인덱스화되는 jobs_tab_type 이라는 전용(private) PL/SQL 인덱스화된 테이블을 선언합니다.
 - ii. jobs_tab_type 을 기반으로 하는 jobstab 이라는 전용(private) 변수를 선언합니다.
 - iii. INITIALIZE 프로시저는 커서 루프를 사용하여 JOBS 테이블의 행을 읽어 들이며, 해당 행이 할당된 jobstab 인덱스에 JOB_ID 값을 사용합니다.
 - iv. GET_MINSalary 함수는 p_jobid 파라미터를 jobstab 에 대한 인덱스로 사용하여 해당 요소의 min_salary 를 반환합니다.
 - v. GET_MAXSalary 함수는 p_jobid 파라미터를 jobstab 에 대한 인덱스로 사용하여 해당 요소의 max_salary 를 반환합니다.
 - vi. SET_MINSalary 프로시저는 p_jobid 를 jobstab 에 대한 인덱스로 사용하여 해당 요소의 min_salary 필드를 min_salary 파라미터의 값으로 설정합니다.
 - vii. SET_MAXSalary 프로시저는 p_jobid 를 jobstab 에 대한 인덱스로 사용하여 해당 요소의 max_salary 필드를 max_salary 파라미터의 값으로 설정합니다.
- c. 연습 10 의 연습 문제 1a 에서 사용한 CHECK_SALARY 프로시저를 복사한 다음 해당 GET_*SALARY 함수를 호출하여 로컬 minsal 및 maxsal 변수를 JOBS_PKG 데이터의 값으로 설정하는 명령문으로 JOBS 테이블에 대한 query 를 바꾸어 코드를 수정합니다. 이 단계에서는 변경 트리거 예외가 발생하면 안됩니다.
- d. DML 작업을 수행하기 전에 패키지가 최신 상태인지 확인하기 위해 CALL 구문을 사용하여 JOBS_PKG.INITIALIZE 프로시저를 호출하는 INIT_JOBPKG_TRG 라는 BEFORE INSERT OR UPDATE 문장 트리거를 구현합니다.
- e. 프로그래머인 사원을 표시하는 query 를 실행하고 IT_PROG 직무 유형의 최소 급여를 \$1,000 인상하는 update 문을 JOBS 테이블에서 실행하여 코드 변경 사항을 테스트합니다. 그런 다음 IT_PROG 직무 유형의 사원에 대한 query 를 수행하여 어떻게 변경되었는지 확인합니다. 어떤 사원의 급여가 해당 직무의 최소 급여로 설정되었습니까?

연습 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리 (계속)

- 3) CHECK_SALARY 프로시저는 사원을 삽입하거나 갱신하기 전에 CHECK_SALARY_TRG 에 의해 실행되므로 이 프로시저가 여전히 예상대로 작동하는지 확인해야 합니다.
- a. 이를 테스트하려면 EMP_PKG.ADD_EMPLOYEE 에 ('Steve', 'Morse', 'SMORSE', and sal => 6500) 파라미터를 사용하여 새로운 사원을 추가합니다. 어떤 결과가 나타납니까?
 - b. 사원을 추가하거나 갱신하는 동안 발생한 문제를 해결하려면
 - i. JOBS_PKG.INITIALIZE 프로시저를 호출하는 EMPLOYEE_INITJOBS_TRG 라는 BEFORE INSERT OR UPDATE 문장 트리거를 EMPLOYEES 테이블에서 생성합니다.
 - ii. 트리거 본문에 CALL 구문을 사용합니다.
 - c. 사원 Steve Morse 를 다시 추가하여 트리거를 테스트합니다. 사원 ID, 이름과 성, 급여, 직무 ID, 부서 ID 를 표시하여 EMPLOYEES 테이블에서 삽입된 레코드를 확인합니다.

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리

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.

- 1) 특정 직무의 최소 급여가 현재 급여보다 높은 값으로 인상될 경우 사원의 급여가 자동으로 인상됩니다. 트리거로 호출되는 패키지 프로시저를 통해 이 요구 사항을 JOBS 테이블에 구현합니다. JOBS 테이블에서 최소 급여를 갱신하고 사원 급여를 갱신하려고 하면 CHECK_SALARY 트리거가 JOBS 테이블(변경될 수 있음)을 읽으려고 하므로 변경 테이블 예외가 발생합니다. 이 문제는 새 패키지와 추가 트리거를 생성하여 해결할 수 있습니다.
 - a) 다음과 같이 EMP_PKG 패키지(연습 8에서 마지막으로 갱신한 버전)를 갱신합니다.
 - i. 사원의 급여를 갱신하는 SET_SALARY라는 프로시저를 추가합니다.
 - ii. SET_SALARY 프로시저는 다음 두 파라미터를 받아들입니다. 하나는 갱신될 수 있는 급여에 대한 직무 ID이고, 다른 하나는 직무 ID에 대한 새 최소 급여입니다.

/home/oracle/labs/plpu/solns/sol_09_01_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다. 새로 추가된 코드는 다음 코드 상자에서 굵은체 글자로 강조 표시됩니다.

```
-- Package SPECIFICATION

CREATE OR REPLACE PACKAGE emp_pkg IS

  TYPE emp_tab_type IS TABLE OF employees%ROWTYPE;

  PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_email employees.email%TYPE,
    p_job employees.job_id%TYPE DEFAULT 'SA_REP',
    p_mgr employees.manager_id%TYPE DEFAULT 145,
    p_sal employees.salary%TYPE DEFAULT 1000,
    p_comm employees.commission_pct%TYPE DEFAULT 0,
    p_deptid employees.department_id%TYPE DEFAULT 30);


```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```

PROCEDURE add_employee(
    p_first_name employees.first_name%TYPE,
    p_last_name employees.last_name%TYPE,
    p_deptid employees.department_id%TYPE);

PROCEDURE get_employee(
    p.empid IN employees.employee_id%TYPE,
    p_sal OUT employees.salary%TYPE,
    p_job OUT employees.job_id%TYPE);

FUNCTION get_employee(p_emp_id employees.employee_id%type)
    return employees%rowtype;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype;

PROCEDURE get_employees(p_dept_id
employees.department_id%type);

PROCEDURE init_departments;

PROCEDURE print_employee(p_rec_emp employees%rowtype);

PROCEDURE show_employees;

/* New set_salary procedure */

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER);

END emp_pkg;
/
SHOW ERRORS

-- Package BODY

CREATE OR REPLACE PACKAGE BODY emp_pkg IS
    TYPE boolean_tab_type IS TABLE OF BOOLEAN
        INDEX BY BINARY_INTEGER;

    valid_departments boolean_tab_type;
    emp_table          emp_tab_type;

    FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
        RETURN BOOLEAN;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_email employees.email%TYPE,
        p_job employees.job_id%TYPE DEFAULT 'SA REP',
        p_mgr employees.manager_id%TYPE DEFAULT 145,
        p_sal employees.salary%TYPE DEFAULT 1000,
        p_comm employees.commission_pct%TYPE DEFAULT 0,
        p_hiredate employees.hire_date%TYPE DEFAULT SYSDATE);

```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```

p_deptid employees.department_id%TYPE DEFAULT 30) IS

    PROCEDURE audit_newemp IS
        PRAGMA AUTONOMOUS_TRANSACTION;
        user_id VARCHAR2(30) := USER;
    BEGIN
        INSERT INTO log_newemp (entry_id, user_id, log_time,
name)
            VALUES (log_newemp_seq.NEXTVAL, user_id,
sysdate,p_first_name||' '||p_last_name);
        COMMIT;
    END audit_newemp;

    BEGIN -- add_employee
        IF valid_deptid(p_deptid) THEN
            audit_newemp;
            INSERT INTO employees(employee_id, first_name,
last_name, email,
                job_id, manager_id, hire_date, salary,
commission_pct, department_id)
                VALUES (employees_seq.NEXTVAL, p_first_name,
p_last_name, p_email,
                p_job, p_mgr, TRUNC(SYSDATE), p_sal, p_comm,
p_deptid);
        ELSE
            RAISE_APPLICATION_ERROR (-20204, 'Invalid department
ID. Try again.');
        END IF;
    END add_employee;

    PROCEDURE add_employee(
        p_first_name employees.first_name%TYPE,
        p_last_name employees.last_name%TYPE,
        p_deptid employees.department_id%TYPE) IS
        p_email employees.email%type;
    BEGIN
        p_email := UPPER(SUBSTR(p_first_name, 1,
1)||SUBSTR(p_last_name, 1, 7));
        add_employee(p_first_name, p_last_name, p_email,
p_deptid => p_deptid);
    END;

    PROCEDURE get_employee(
        p.empid IN employees.employee_id%TYPE,
        p_sal OUT employees.salary%TYPE,
        p.job OUT employees.job_id%TYPE) IS
    BEGIN
        SELECT salary, job_id
        INTO p_sal, p_job
        FROM employees
        WHERE employee_id = p.empid;
    END get_employee;

    FUNCTION get_employee(p.emp_id employees.employee_id%type)
        return employees%rowtype IS

```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```

rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE employee_id = p_emp_id;
    RETURN rec_emp;
END;

FUNCTION get_employee(p_family_name
employees.last_name%type)
    return employees%rowtype IS
    rec_emp employees%rowtype;
BEGIN
    SELECT * INTO rec_emp
    FROM employees
    WHERE last_name = p_family_name;
    RETURN rec_emp;
END;

PROCEDURE get_employees(p_dept_id
employees.department_id%type) IS
BEGIN
    SELECT * BULK COLLECT INTO emp_table
    FROM EMPLOYEES
    WHERE department_id = p_dept_id;
END;

PROCEDURE init_departments IS
BEGIN
    FOR rec IN (SELECT department_id FROM departments)
    LOOP
        valid_departments(rec.department_id) := TRUE;
    END LOOP;
END;

PROCEDURE print_employee(p_rec_emp employees%rowtype) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(p_rec_emp.department_id || ' ' ||
    p_rec_emp.employee_id|| ' ' ||
    p_rec_emp.first_name|| ' ' ||
    p_rec_emp.last_name|| ' ' ||
    p_rec_emp.job_id|| ' ' ||
    p_rec_emp.salary);
END;

PROCEDURE show_employees IS
BEGIN
    IF emp_table IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('Employees in Package table');
        FOR i IN 1 .. emp_table.COUNT
        LOOP
            print_employee(emp_table(i));
        END LOOP;
    END IF;
END show_employees;

```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```

FUNCTION valid_deptid(p_deptid IN
departments.department_id%TYPE)
    RETURN BOOLEAN IS
    v_dummy PLS_INTEGER;
BEGIN
    RETURN valid_departments.exists(p_deptid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN FALSE;
END valid_deptid;

/* New set_salary procedure */

PROCEDURE set_salary(p_jobid VARCHAR2, p_min_salary NUMBER) IS
    CURSOR cur_emp IS
        SELECT employee_id
        FROM employees
        WHERE job_id = p_jobid AND salary < p_min_salary;
BEGIN
    FOR rec_emp IN cur_emp
    LOOP
        UPDATE employees
        SET salary = p_min_salary
        WHERE employee_id = rec_emp.employee_id;
    END LOOP;
END set_salary;

BEGIN
    init_departments;
END emp_pkg;

/
SHOW ERRORS

```

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

```

PACKAGE emp_pkg Compiled.
No Errors.
PACKAGE BODY emp_pkg Compiled.
No Errors.

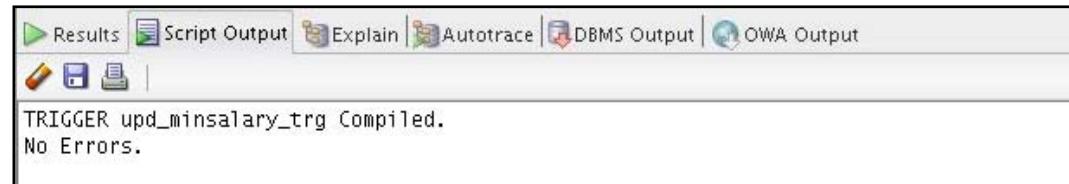
```

- b) 지정된 직무 ID에 대해 JOBS 테이블에 있는 최소 급여가갱신되면
`EMP_PKG.SET_SALARY` 프로시저를 호출하는
`UPD_MINSLARY_TRG`라는 행 트리거를 JOBS 테이블에 생성합니다.

`/home/oracle/labs/plpu/solns/sol_09_01_b.sql` 스크립트를
 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를
 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```
CREATE OR REPLACE TRIGGER upd_minsalary_trg
AFTER UPDATE OF min_salary ON JOBS
FOR EACH ROW
BEGIN
    emp_pkg.set_salary(:new.job_id, :new.min_salary);
END;
/
SHOW ERRORS
```



- c) 프로그래머(JOB_ID: 'IT_PROG')인 사원의 ID, 성, 직무 ID, 현재 급여, 최소 급여를 표시하는 query 를 작성합니다. 그런 다음 JOBS 테이블에서 최소 급여가 \$1000 인상되도록 갱신합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_09_01_c.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
    SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';
```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

The screenshot shows an Oracle SQL Developer session window. The 'Results' tab is active, displaying a query output:

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
105	Austin	4800
106	Pataballa	4800
107	Lorentz	4200
214	Beh	9000

Below the table, the message "6 rows selected" is displayed.

Following the table, an error message is shown:

```
Error starting at line 5 in command:
UPDATE jobs
  SET min_salary = min_salary + 1000
 WHERE job_id = 'IT_PROG'
Error report:
SQL Error: ORA-04091: table ORA61.JOBS is mutating, trigger/function may not see it
ORA-06512: at "ORA61.CHECK_SALARY", line 5
ORA-06512: at "ORA61.CHECK_SALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.CHECK_SALARY_TRG'
ORA-06512: at "ORA61.EMP_PKG", line 143
ORA-06512: at "ORA61.UPD_MINSALARY_TRG", line 2
ORA-04088: error during execution of trigger 'ORA61.UPD_MINSALARY_TRG'
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.
```

JOBS 테이블에 대한 UPD_MINSALARY_TRG 트리거가

EMP_PKG.SET_SALARY 프로시저를 호출하여 사원의 급여를 갱신하려고 하기 때문에 'IT_PROG' 직무의 min_salary 열이 갱신되지 않습니다.

SET_SALARY 프로시저로 인해 CHECK_SALARY_TRG 트리거가

실행됩니다(연쇄적인 결과). CHECK_SALARY_TRG 는 JOBS 테이블 데이터를 읽으려고 하는 CHECK_SALARY 프로시저를 호출합니다. JOBS 테이블을 읽는 동안 CHECK_SALARY 프로시저에서 변경 테이블 예외가 발생합니다.

- 2) 변경 테이블 문제를 해결하려면 JOBS 테이블의 행 복사본을 메모리에 유지하는 JOBS_PKG 패키지를 생성합니다. 그런 다음 예외를 피하기 위해 변경되는 테이블에 대해 query를 실행하는 대신 패키지 데이터를 사용하도록 CHECK_SALARY 프로시저를 수정합니다. 그러나 BEFORE INSERT OR UPDATE 문장 트리거를 EMPLOYEES 테이블에 생성하여 JOBS_PKG 패키지 상태를 초기화한 후 CHECK_SALARY 행 트리거를 실행해야 합니다.

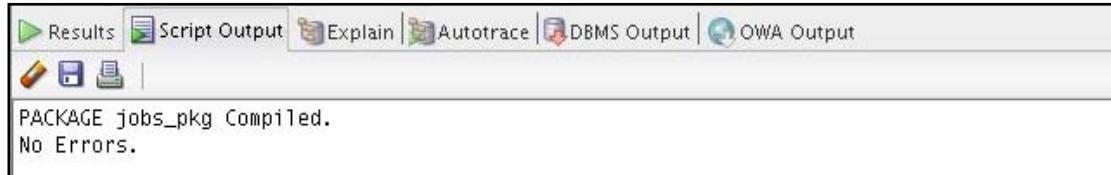
- a) 다음 Spec이 포함된 JOBS_PKG라는 새 패키지를 생성합니다.

```
PROCEDURE initialize;
FUNCTION get_minsalary(jobid VARCHAR2) RETURN NUMBER;
FUNCTION get_maxsalary(jobid VARCHAR2) RETURN NUMBER;
PROCEDURE set_minsalary(jobid VARCHAR2,min_salary
                       NUMBER);
PROCEDURE set_maxsalary(jobid VARCHAR2,max_salary
                       NUMBER);
```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

/home/oracle/labs/plpu/solns 폴더에서 sol_09_02_a.sql 파일을 열거나 SQL Worksheet 영역에서 다음 코드를 복사하여 붙여넣습니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE jobs_pkg IS
  PROCEDURE initialize;
  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER;
  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER;
  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary NUMBER);
  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary NUMBER);
END jobs_pkg;
/
SHOW ERRORS
```



b) 다음과 같이 JOBS_PKG 의 Body 를 구현합니다.

- i. JOBS.JOB_ID%TYPE 을 기준으로 문자열 유형에 의해 인덱스화되는 jobs_tab_type 이라는 전용(private) PL/SQL 인덱스화된 테이블을 선언합니다.
- ii. jobs_tab_type 을 기반으로 하는 jobstab이라는 전용(private) 변수를 선언합니다.
- iii. INITIALIZE 프로시저는 커서 루프를 사용하여 JOBS 테이블의 행을 읽어 들이며, 해당 행이 할당된 jobstab 인덱스에 JOB_ID 값을 사용합니다.
- iv. GET_MINSalary 함수는 p_jobid 파라미터를 jobstab에 대한 인덱스로 사용하여 해당 요소의 min_salary 를 반환합니다.
- v. GET_MAXSalary 함수는 p_jobid 파라미터를 jobstab에 대한 인덱스로 사용하여 해당 요소의 max_salary 를 반환합니다.
- vi. SET_MINSalary 프로시저는 p_jobid 를 jobstab에 대한 인덱스로 사용하여 해당 요소의 min_salary 필드를 min_salary 파라미터의 값으로 설정합니다.

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

- vii. SET_MAXSALARY 프로시저는 p_jobid 를 jobstab 에 대한 인덱스로 사용하여 해당 요소의 max_salary 필드를 max_salary 파라미터의 값으로 설정합니다.

/home/oracle/labs/plpu/solns/sol_09_02_b.sql

스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서

Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

Package Body 를 컴파일하려면 Object Navigator 트리에서 패키지 이름 또는 Body 를 마우스 오른쪽 버튼으로 누르고 Compile 을 선택합니다.

```

CREATE OR REPLACE PACKAGE BODY jobs_pkg IS
  TYPE jobs_tab_type IS TABLE OF jobs%rowtype
    INDEX BY jobs.job_id%type;
  jobstab jobs_tab_type;

  PROCEDURE initialize IS
  BEGIN
    FOR rec_job IN (SELECT * FROM jobs)
    LOOP
      jobstab(rec_job.job_id) := rec_job;
    END LOOP;
  END initialize;

  FUNCTION get_minsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).min_salary;
  END get_minsalary;

  FUNCTION get_maxsalary(p_jobid VARCHAR2) RETURN NUMBER IS
  BEGIN
    RETURN jobstab(p_jobid).max_salary;
  END get_maxsalary;

  PROCEDURE set_minsalary(p_jobid VARCHAR2, p_min_salary
NUMBER) IS
  BEGIN
    jobstab(p_jobid).min_salary := p_min_salary;
  END set_minsalary;

  PROCEDURE set_maxsalary(p_jobid VARCHAR2, p_max_salary
NUMBER) IS
  BEGIN
    jobstab(p_jobid).max_salary := p_max_salary;
  END set_maxsalary;

END jobs_pkg;
/
SHOW ERRORS

```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```
Results Script Output Explain Autotrace DBMS Output OWA Output
PACKAGE BODY jobs_pkg Compiled.
No Errors.
```

- c) 연습 8의 연습 문제 1a에서 사용한 CHECK_SALARY 프로시저를 복사한 다음 해당 GET_*SALARY 함수를 호출하여 로컬 minsal 및 maxsal 변수를 JOBS_PKG 데이터의 값으로 설정하는 명령문으로 JOBS 테이블에 대한 query를 바꾸어 코드를 수정합니다. 이 단계에서는 변경 트리거 예외가 발생하면 안됩니다.

/home/oracle/labs/plpu/solns/sol_09_02_c.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PROCEDURE check_salary (p_the_job
VARCHAR2, p_the_salary NUMBER) IS
v_minsal jobs.min_salary%type;
v_maxsal jobs.max_salary%type;
BEGIN
/*
 ** Commented out to avoid mutating trigger exception on
the JOBS table
SELECT min_salary, max_salary INTO v_minsal, v_maxsal
FROM jobs
WHERE job_id = UPPER(p_the_job);
*/
v_minsal := jobs_pkg.get_minsalary(UPPER(p_the_job));
v_maxsal := jobs_pkg.get_maxsalary(UPPER(p_the_job));
IF p_the_salary NOT BETWEEN v_minsal AND v_maxsal THEN
RAISE_APPLICATION_ERROR(-20100,
'Invalid salary $'||p_the_salary||'. ||
'SALARIES FOR JOB '|| p_the_job ||
' MUST BE BETWEEN $'|| v_minsal ||' AND $' ||
v_maxsal);
END IF;
END;
/
SHOW ERRORS
```

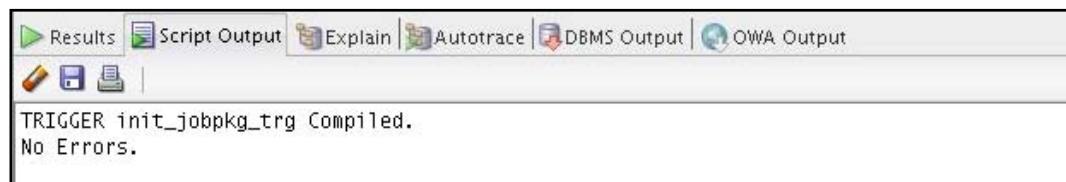
```
Results Script Output Explain Autotrace DBMS Output OWA Output
PROCEDURE check_salary Compiled.
No Errors.
```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리 (계속)

- d) DML 작업을 수행하기 전에 패키지가 최신 상태인지 확인하기 위해 CALL 구문을 사용하여 JOBS_PKG.INITIALIZE 프로시저를 호출하는 INIT_JOBPKG_TRG라는 BEFORE INSERT OR UPDATE 문장 트리거를 구현합니다.

/home/oracle/labs/plpu/solns/sol_09_02_d.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE TRIGGER init_jobpkg_trg
BEFORE INSERT OR UPDATE ON jobs
CALL jobs_pkg.initialize
/
SHOW ERRORS
```



- e) 프로그래머인 사원을 표시하는 query를 실행하고 IT_PROG 직무 유형의 최소 급여를 \$1,000 인상하는 update 문을 JOBS 테이블에서 실행하여 코드 변경 사항을 테스트합니다. 그런 다음 IT_PROG 직무 유형의 사원에 대한 query를 수행하여 어떻게 변경되었는지 확인합니다. 어떤 사원의 급여가 해당 직무의 최소 급여로 설정되었습니까?

/home/oracle/labs/plpu/solns/sol_09_02_e.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';

UPDATE jobs
SET min_salary = min_salary + 1000
WHERE job_id = 'IT_PROG';

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'IT_PROG';
```

연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리(계속)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | |
EMPLOYEE_ID LAST_NAME SALARY
-----|-----|-----
103      Hunold     9000
104      Ernst      6000
105      Austin     4800
106      Pataballa  4800
107      Lorentz    4200
214      Beh        9000

6 rows selected

1 rows updated
EMPLOYEE_ID LAST_NAME SALARY
-----|-----|-----
103      Hunold     9000
104      Ernst      6000
105      Austin     5000
106      Pataballa  5000
107      Lorentz    5000
214      Beh        9000

6 rows selected

```

성이 Austin, Pataballa 및 Lorentz 인 사원의 급여가 모두
갱신되었습니다. 이 프로세스 동안에는 예외가 발생하지 않으므로 변경
테이블 트리거 예외에 대한 해결 방법을 구현한 것입니다.

- 3) CHECK_SALARY 프로시저는 사원을 삽입하거나 갱신하기 전에
CHECK_SALARY_TRG 에 의해 실행되므로 이 프로시저가 여전히
예상대로 작동하는지 확인해야 합니다.
 - a) 이를 테스트하려면 EMP_PKG.ADD_EMPLOYEE 에 ('Steve',
'Morse', 'SMORSE', and sal => 6500) 파라미터를
사용하여 새로운 사원을 추가합니다. 어떤 결과가 나타납니까?

/home/oracle/labs/plpu/solns/sol_09_03_a.sql 스크립트를
입니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를
누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE emp_pkg.add_employee('Steve', 'Morse', 'SMORSE',
p_sal => 6500)
```

```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | |
anonymous block completed

```

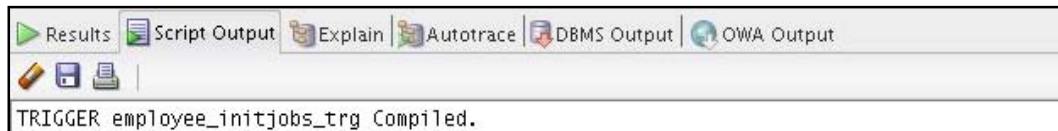
연습 해답 9-1: 데이터 무결성 규칙 및 변경 테이블 예외 관리 (계속)

b) 사원을 추가하거나 갱신하는 동안 발생한 문제를 해결하려면

- JOBS_PKG. INITIALIZE 프로시저를 호출하는 EMPLOYEE_INITJOBS_TRG라는 BEFORE INSERT OR UPDATE 문장 트리거를 EMPLOYEES 테이블에서 생성합니다.
- 트리거 본문에 CALL 구문을 사용합니다.

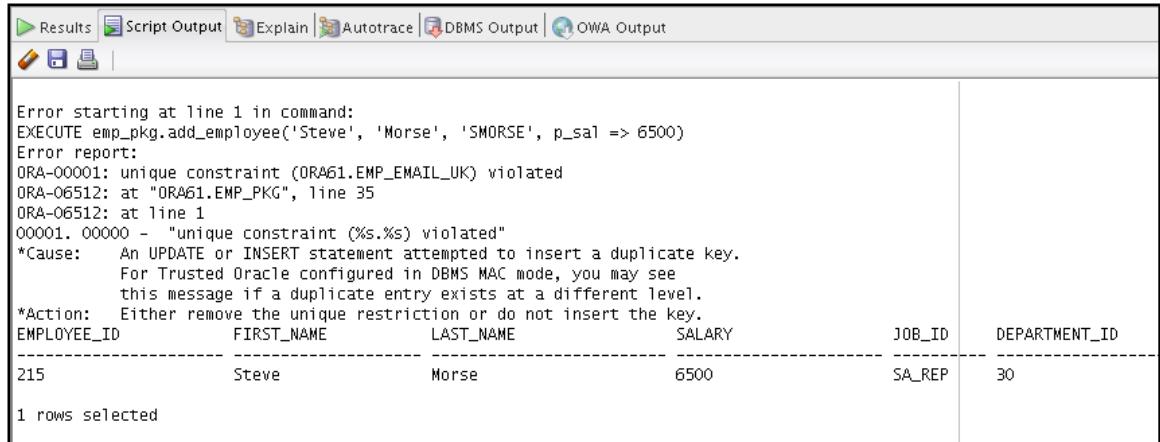
/home/oracle/labs/plpu/solns/sol_09_03_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE TRIGGER employee_initjobs_trg
BEFORE INSERT OR UPDATE OF job_id, salary ON employees
CALL jobs_pkg.initialize
/
```



c) 사원 Steve Morse를 다시 추가하여 트리거를 테스트합니다. 사원 ID, 이름과 성, 급여, 직무 ID, 부서 ID를 표시하여 EMPLOYEES 테이블에서 삽입된 레코드를 확인합니다.

/home/oracle/labs/plpu/solns/sol_09_03_c.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.



단원 10 의 연습 및 해답

이 연습에서는 컴파일러 초기화 파라미터를 표시합니다. 그런 다음 세션에 대한 Native Compile 을 활성화하고 프로시저를 컴파일합니다. 그리고 모든 컴파일러 경고 범주를 제거하고 원래 세션 경고 설정을 복원합니다. 마지막으로 일부 컴파일러 경고 메시지 번호의 범주를 식별합니다.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용

이 연습에서는 컴파일러 초기화 파라미터를 표시합니다. 그런 다음 세션에 대한 Native Compile 을 활성화하고 프로시저를 컴파일합니다. 그리고 모든 컴파일러 경고 범주를 제거하고 원래 세션 경고 설정을 복원합니다. 마지막으로 일부 컴파일러 경고 메시지 번호의 범주를 식별합니다.

- 1) USER_PLSQL_OBJECT_SETTINGS 데이터 딕셔너리 뷰를 사용하여 컴파일러 초기화 파라미터에 대한 다음 정보를 표시하기 위해 lab_10_01 스크립트를 생성하여 실행합니다. ADD_JOB_HISTORY 객체에 대한 설정을 확인합니다.
참고: Results 탭에 결과를 표시하려면 Execute Statement (F9) 아이콘을 사용하십시오.
 - a) 객체 이름
 - b) 객체 유형
 - c) 객체의 컴파일 모드
 - d) 컴파일 최적화 레벨
- 2) PLSQL_CODE_TYPE 파라미터를 변경하여 세션에 대해 Native Compile 을 활성화한 다음 ADD_JOB_HISTORY 를 컴파일합니다.
 - a) ALTER SESSION 명령을 실행하여 세션에 대한 Native Compile 을 활성화합니다.
 - b) ADD_JOB_HISTORY 프로시저를 컴파일합니다.
 - c) sol_10_01 스크립트를 재실행합니다. PLSQL_CODE_TYPE 파라미터를 확인합니다.
 - d) 다음과 같이 INTERPRETED 컴파일 모드를 사용하도록 컴파일을 전환합니다.
- 3) Tools > Preferences > PL/SQL Compiler Options 영역을 사용하여 모든 컴파일러 경고 범주를 비활성화합니다.
- 4) lab_10_04.sql 스크립트를 편집, 검사 및 실행하여 UNREACHABLE_CODE 프로시저를 생성합니다. Run Script 아이콘 (F5)을 눌러 프로시저를 생성합니다. Navigation 트리에 있는 프로시저 이름을 사용하여 프로시저를 컴파일합니다.
- 5) Compiler – Log 탭에 컴파일러 경고가 표시될 경우 어떤 것이 표시됩니까?
- 6) Preferences window 를 사용하여 이 세션의 모든 컴파일러 경고 메시지를 활성화합니다.

연습 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

- 7) Object Navigation 트리를 사용하여 UNREACHABLE_CODE 프로시저를 재컴파일합니다. 어떤 컴파일러 경고가 표시됩니까(있을 경우)?
- 8) 다음과 같이 USER_ERRORS 데이터 딕셔너리 뷰를 사용하여 컴파일러 경고 메시지 상세 내역을 표시합니다.
- 9) EXECUTE DBMS_OUTPUT 및 DBMS_WARNING 패키지를 사용하여 컴파일러 경고 메시지 번호 5050, 6075 및 7100에 대한 범주를 식별하는 warning_msgs라는 스크립트를 생성합니다. 스크립트를 실행하기 전에 SERVEROUTPUT 을 활성화합니다.

연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용

이 연습에서는 컴파일러 초기화 파라미터를 표시합니다. 그런 다음 세션에 대한 Native Compile 을 활성화하고 프로시저를 컴파일합니다. 그리고 모든 컴파일러 경고 범주를 제거하고 원래 세션 경고 설정을 복원합니다. 마지막으로 일부 컴파일러 경고 메시지 번호의 범주를 식별합니다.

- 1) USER_PLSQL_OBJECT_SETTINGS 데이터 딕셔너리 뷰를 사용하여 컴파일러 초기화 파라미터에 대한 다음 정보를 표시하기 위해 lab_10_01 스크립트를 생성하여 실행합니다. ADD_JOB_HISTORY 객체에 대한 설정을 확인합니다.
- 참고:** Results 탭에 결과를 표시 하려면 Execute Statement (F9) 아이콘을 사용하십시오.

- a) 객체 이름
- b) 객체 유형
- c) 객체의 컴파일 모드
- d) 컴파일 최적화 레벨

/home/oracle/labs/plpu/solns/sol_10_01.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Execute Statement (F9) 아이콘을 눌러 query 를 실행합니다. 코드 및 결과 예제가 다음과 같이 표시됩니다.

```
SELECT name, type, plsql_code_type as code_type,
       plsql_optimize_level as opt_lvl
  FROM user_plsql_object_settings;
```

NAME	TYPE	CODE_TYPE
ADD_EMPLOYEE	PROCEDURE	INTERPRETED
ADD_JOB	PROCEDURE	INTERPRETED
ADD_JOB_HISTORY	PROCEDURE	INTERPRETED
CHECK_SALARY	PROCEDURE	INTERPRETED
CHECK_SALARY_TRG	TRIGGER	INTERPRETED
COMPILE_PKG	PACKAGE	INTERPRETED
COMPILE_PKG	PACKAGE BODY	INTERPRETED
DELETE_EMP_TRG	TRIGGER	INTERPRETED
DEL_JOB	PROCEDURE	INTERPRETED
EMPLOYEE_INITJOBS_TRG	TRIGGER	INTERPRETED
EMPLOYEE_REPORT	PROCEDURE	INTERPRETED
EMP_LIST	PROCEDURE	INTERPRETED
EMP_PKG	PACKAGE	INTERPRETED
EMP_PKG	PACKAGE BODY	INTERPRETED
GET_ANNUAL_COMP	FUNCTION	INTERPRETED
GET_EMPLOYEE	PROCEDURE	INTERPRETED
GET_JOB	FUNCTION	INTERPRETED
GET_LOCATION	FUNCTION	INTERPRETED
INIT_JOBPKG_TRG	TRIGGER	INTERPRETED
JOB_PKG	PACKAGE	INTERPRETED
JOB_PKG	PACKAGE BODY	INTERPRETED
JOB_PKG	PACKAGE	INTERPRETED

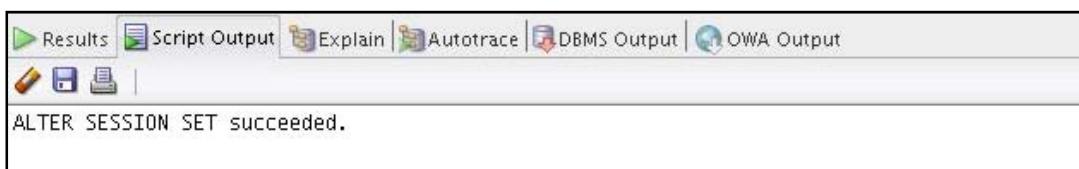
연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

- 2) PLSQL_CODE_TYPE 파라미터를 변경하여 세션에 대해 Native Compile 을 활성화한 다음 ADD_JOB_HISTORY 를 컴파일합니다.

- a) ALTER SESSION 명령을 실행하여 세션에 대한 Native Compile 을 활성화합니다.

/home/oracle/labs/plpu/solns/sol_10_02_a.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 query 를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

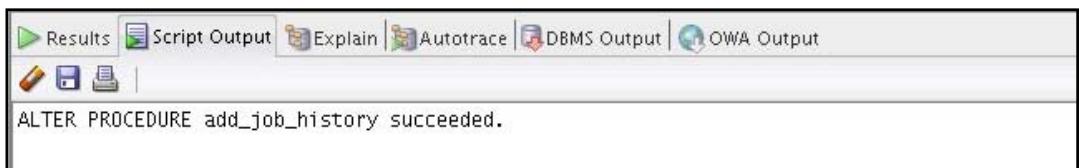
```
ALTER SESSION SET PLSQL_CODE_TYPE = 'NATIVE' ;
```



- b) ADD_JOB_HISTORY 프로시저를 컴파일합니다.

/home/oracle/labs/plpu/solns/sol_10_02_b.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 query 를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
ALTER PROCEDURE add_job_history COMPILE;
```



- c) sol_10_01.sql 스크립트를 재실행합니다. PLSQL_CODE_TYPE 파라미터를 확인합니다.

```
SELECT name, type, plsql_code_type as code_type,
plsql_optimize_level as opt_lvl
FROM user_plsql_object_settings;
```

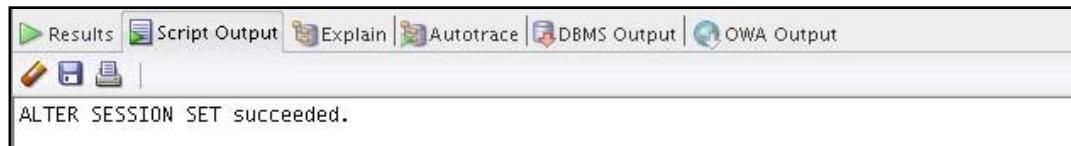
NAME	TYPE	CODE_TYPE
ADD_EMPLOYEE	PROCEDURE	INTERPRETED
ADD_JOB	PROCEDURE	INTERPRETED
ADD_JOB_HISTORY	PROCEDURE	NATIVE
CHECK_SALARY	PROCEDURE	INTERPRETED
CHECK_SALARY TRG	TRIGGER	INTERPRETED

...

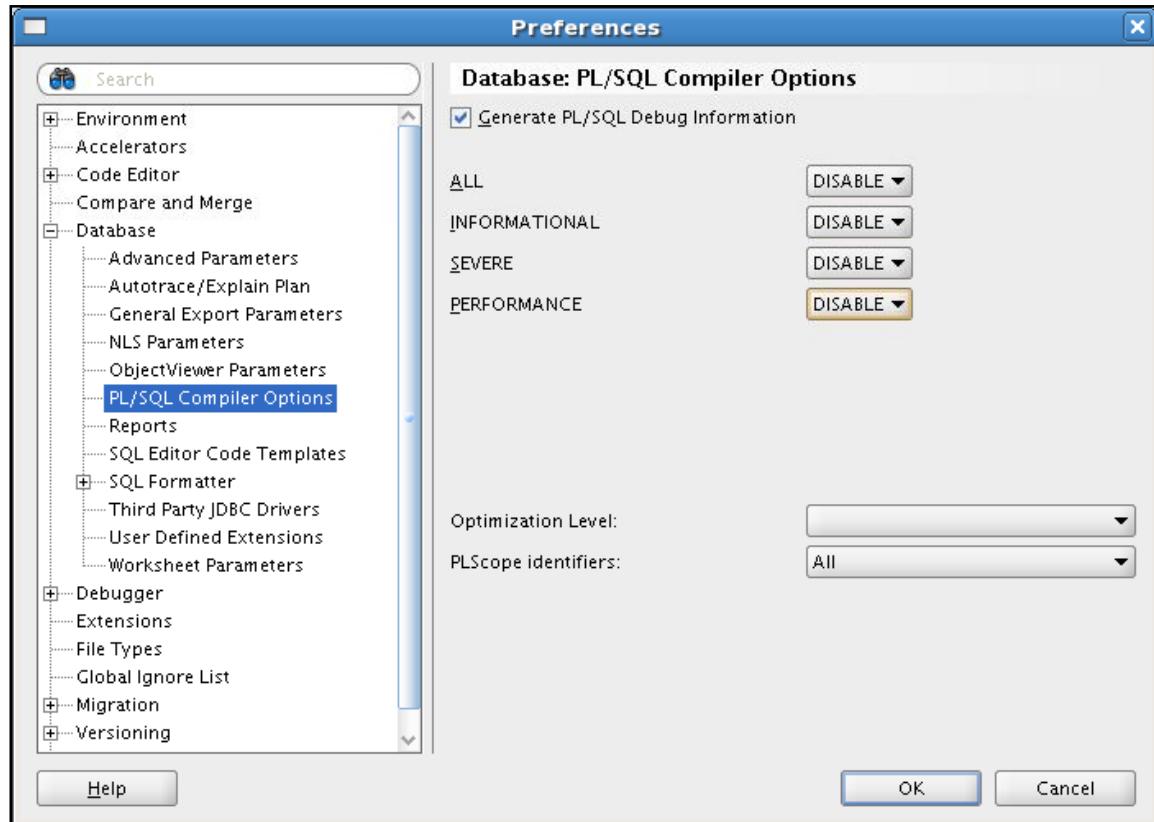
연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

- d) 다음과 같이 INTERPRETED 컴파일 모드를 사용하도록 컴파일을 전환합니다.

```
ALTER SESSION SET PLSQL_CODE_TYPE = 'INTERPRETED';
```



- 3) Tools > Preferences > PL/SQL Compiler Options 영역을 사용하여 모든 컴파일러 경고 범주를 비활성화합니다.



네 가지 PL/SQL 컴파일러 경고 범주 모두에 대해 DISABLE 을 선택하고 OK 를 누릅니다.

- 4) lab_10_04.sql 스크립트를 편집, 검사 및 실행하여 UNREACHABLE_CODE 프로시저를 생성합니다. Run Script 아이콘(F5)을 눌러 프로시저를 생성하고 컴파일합니다.

/home/oracle/labs/plpu/solns/sol_10_04.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 query 를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

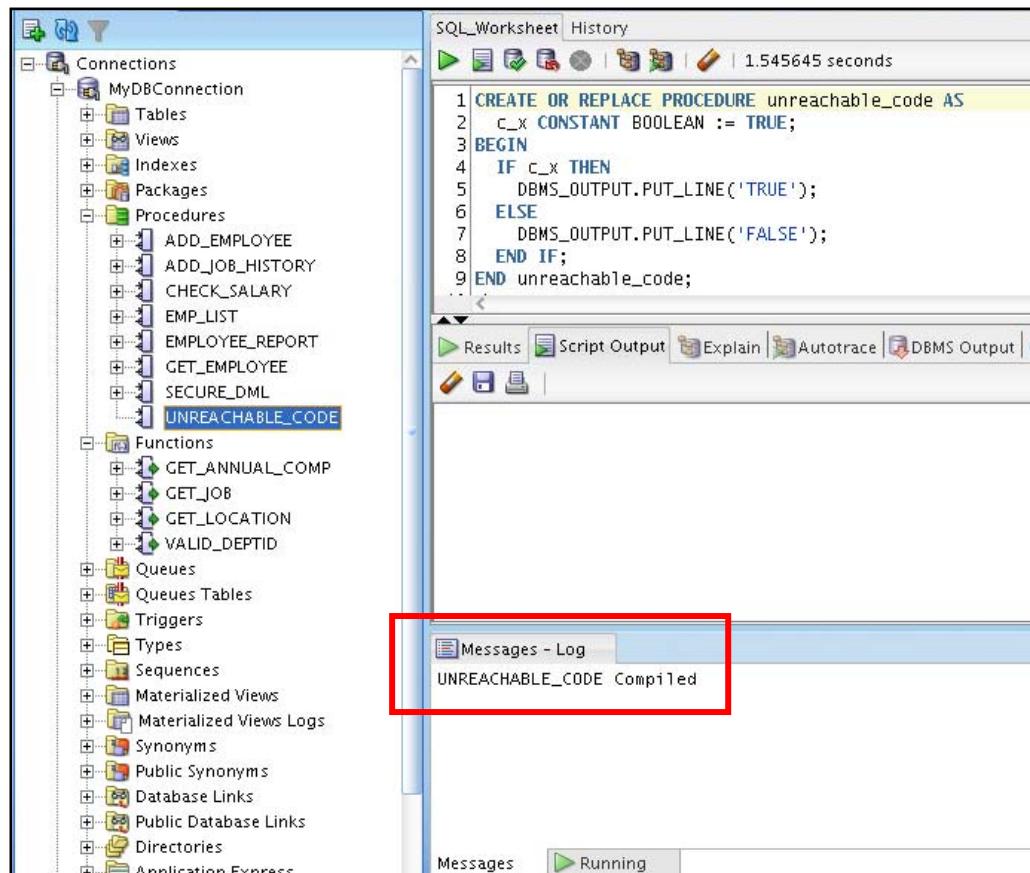
```
CREATE OR REPLACE PROCEDURE unreachable_code AS
  c_x CONSTANT BOOLEAN := TRUE;
```

연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

```
BEGIN
  IF c_x THEN
    DBMS_OUTPUT.PUT_LINE( 'TRUE' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'FALSE' );
  END IF;
END unreachable_code;
/
```



컴파일러 경고 오류를 보려면 Navigation 트리의 Procedures 노드에서 프로시저 이름을 마우스 오른쪽 버튼으로 누른 다음 Compile 을 누릅니다.



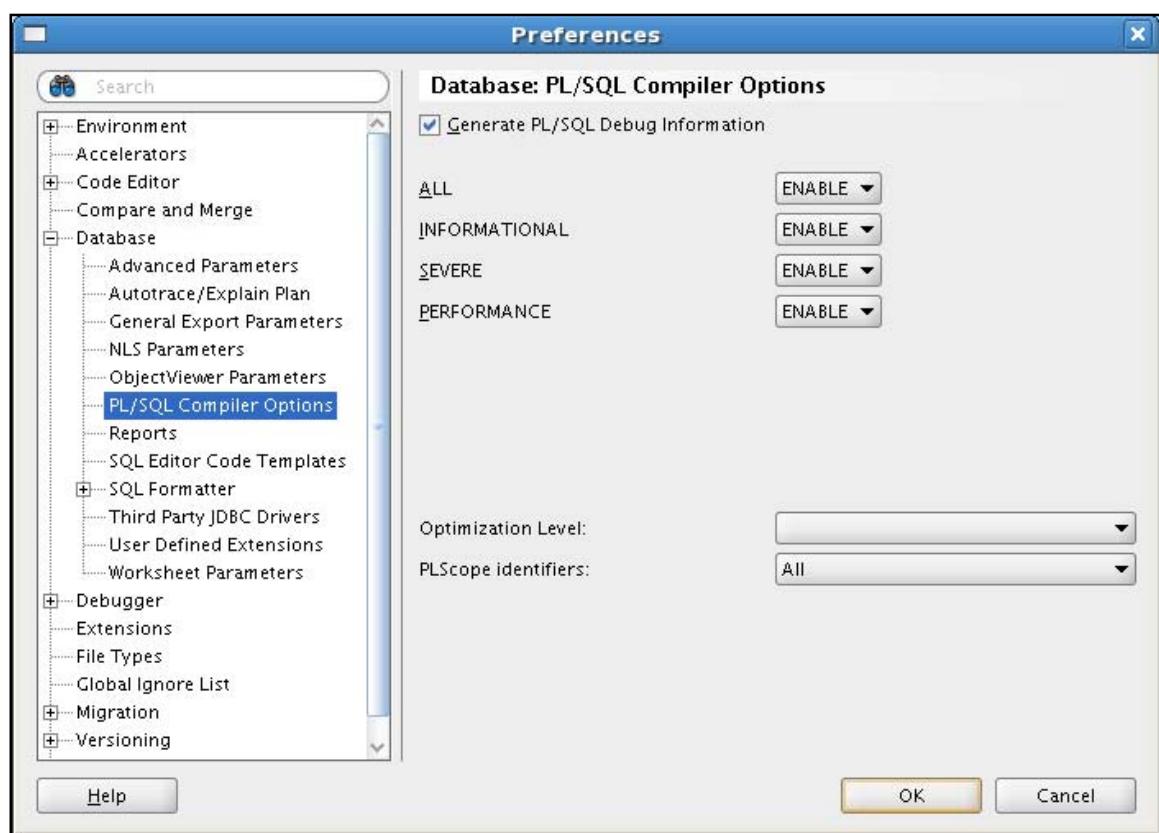
참고

- 프로시저가 Navigation 트리에 표시되지 않는 경우 Connections 탭에서 Refresh 아이콘을 누릅니다.
- Messages – Log 탭이 표시되는지 확인합니다(메뉴 모음에서 View > Log 선택).

연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

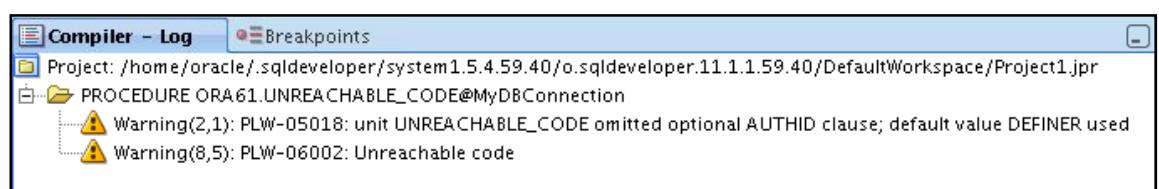
- 5) Compiler – Log 탭에 컴파일러 경고가 표시될 경우 어떤 것이 표시됩니까?
 단계 3에서 컴파일러 경고를 비활성화했기 때문에 Messages – Log 탭에 "UNREACHABLE_CODE Compiled"라는 메시지가 표시되고 경고 메시지는 표시되지 않습니다.
- 6) Preferences window 를 사용하여 이 세션의 모든 컴파일러 경고 메시지를 활성화합니다.

네 가지 PL/SQL 컴파일러 경고 모두에 대해 ENABLE 을 선택하고 OK 를 누릅니다.



- 7) Object Navigation 트리를 사용하여 UNREACHABLE_CODE 프로시저를 재컴파일합니다. 어떤 컴파일러 경고가 표시됩니까(있을 경우)?

Object Navigation 트리에서 프로시저 이름을 마우스 오른쪽 버튼으로 누르고 Compile 을 선택합니다. Compiler – Log 탭에서 Messages 및 Compiler 하위 탭에 표시되는 메시지를 확인합니다.



연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

참고: SQL Developer 에 다음과 같은 두 경고가 표시되는 것은 일부 SQL Developer 버전에서 예상된 상황입니다. 다음 경고가 표시될 경우 SQL Developer 버전에서 Oracle 11g 데이터베이스에 사용되지 않는 PLSQL_DEBUG 파라미터를 아직 사용하고 있기 때문입니다.

```
Warning(1):PLW-06015:parameter PLSQL_DEBUG is
deprecated ; use PLSQL_OPTIMIZE_LEVEL=1
```

```
Warning(1):PLW-06013:deprecated parameter PLSQL_DEBUG
forces PLSQL_OPTIMIZE_LEVEL<=1
```

- 8) 다음과 같이 USER_ERRORS 데이터 덕셔너리 뷰를 사용하여 컴파일러 경고 메시지 상세 내역을 표시합니다.

```
DESCRIBE user_errors
```

```
DESCRIBE user_errors
Name          Null    Type
-----        -----
NAME          NOT NULL VARCHAR2(30)
TYPE          VARCHAR2(12)
SEQUENCE      NOT NULL NUMBER
LINE          NOT NULL NUMBER
POSITION      NOT NULL NUMBER
TEXT          NOT NULL VARCHAR2(4000)
ATTRIBUTE     VARCHAR2(9)
MESSAGE_NUMBER NUMBER

8 rows selected
```

```
SELECT *
FROM user_errors;
```

Results								
	NAME	TYPE	SEQUENCE	LINE	POSITION	TEXT	ATTRIBUTE	MESSAGE_NUMBER
1	UNREACHABLE_CODE PROCEDURE		1	1	1	PLW-05018: unit UNREACHABLE_CODE omitted optional AUTHID clause; def...	WARNING	5018
2	UNREACHABLE_CODE PROCEDURE		2	0	0	PLW-06015: parameter PLSQL_DEBUG is deprecated; use PLSQL_OPTIMIZE_L...	WARNING	6015
3	UNREACHABLE_CODE PROCEDURE		3	0	0	PLW-06013: deprecated parameter PLSQL_DEBUG forces PLSQL_OPTIMIZE_L...	WARNING	6013
4	UNREACHABLE_CODE PROCEDURE		4	7	5	PLW-06002: Unreachable code	WARNING	6002

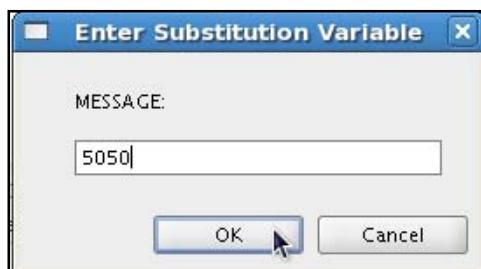
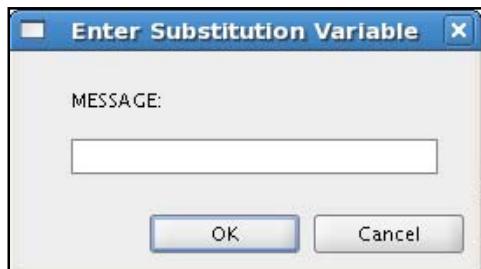
참고: F9 키를 사용하여 SELECT 문을 실행했기 때문에 Results 탭에 출력이 표시되었습니다. SELECT 문의 결과는 세션에서 발생한 오류의 양에 따라 다를 수 있습니다.

연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)

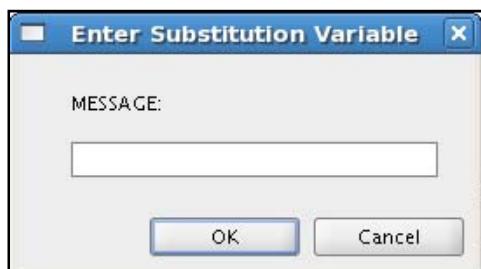
- 9) EXECUTE DBMS_OUTPUT 및 DBMS_WARNING 패키지를 사용하여 컴파일러 경고 메시지 번호 5050, 6075 및 7100에 대한 범주를 식별하는 warning_msgs라는 스크립트를 생성합니다. 스크립트를 실행하기 전에 SERVEROUTPUT 을 활성화합니다.

/home/oracle/labs/plpu/solns/sol_10_09.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 query 를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

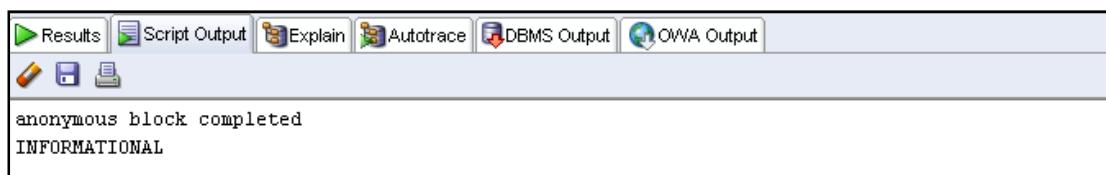
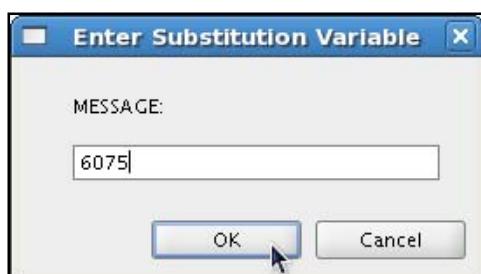
```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



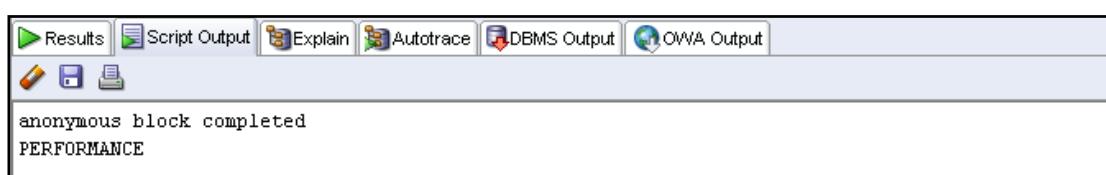
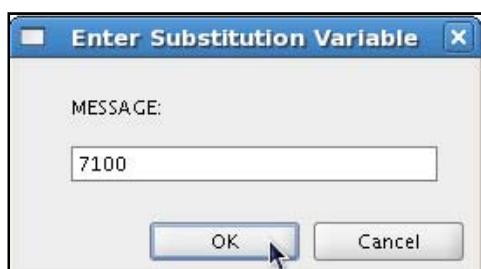
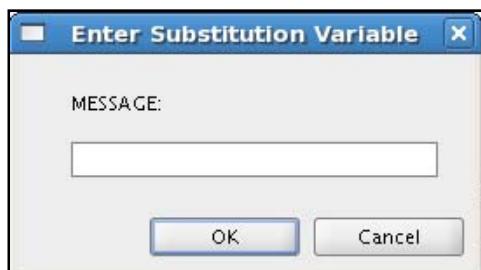
```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



연습 해답 10-1: PL/SQL 컴파일러 파라미터 및 경고 사용 (계속)



```
EXECUTE
DBMS_OUTPUT.PUT_LINE(DBMS_WARNING.GET_CATEGORY(&message));
```



단원 11 의 연습 및 해답

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 처리(Obfuscation)도 적용합니다.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 11-1: 조건부 컴파일 사용

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.

- 1) lab_11_01.sql 스크립트를 검토한 다음 실행합니다. 이 스크립트는 디버깅 코드 및 추적 정보를 표시하기 위한 플래그를 설정합니다. 이 스크립트는 my_pkg 패키지와 circle_area 프로시저도 생성합니다.
- 2) lab_11_01에서 조건부 컴파일 지시어를 처리한 후에 DBMS_PREPROCESSOR 서브 프로그램을 사용하여 PL/SQL 단위의 포스트 처리된 소스 텍스트를 읽어 들입니다. SERVEROUTPUT을 활성화합니다.
- 3) 조건부 컴파일에 DBMS_DB_VERSION 상수를 사용하는 PL/SQL 스크립트를 생성합니다. 코드는 다음의 오라클 데이터베이스 버전에 대해 테스트해야 합니다.
 - a) 데이터베이스 버전이 10.1 이하일 경우에는 다음 오류 메시지가 표시되어야 합니다.
Unsupported database release.
 - b) 데이터베이스 버전이 11.1 이상일 경우에는 다음 메시지가 표시되어야 합니다.
Release 11.1 is supported.
- 4) CREATE_WWRAPPED를 사용하여 데이터베이스에서 Package Spec 및 Package Body를 동적으로 생성하고 래핑하는 다음 코드를 lab_11_04.sql 스크립트에 사용해 봅니다. lab_11_04.sql 스크립트를 편집하여 PL/SQL 코드를 난독 처리하는 데 필요한 코드를 추가합니다. 스크립트를 저장하고 실행합니다.

```

DECLARE
  -- the package_text variable contains the text to create
  -- the package spec and body
  package_text VARCHAR2(32767);
  FUNCTION generate_spec (pkgname VARCHAR2) RETURN
  VARCHAR2 AS
  BEGIN
    RETURN 'CREATE PACKAGE ' || pkgname || ' AS
      PROCEDURE raise_salary (emp_id NUMBER, amount
      NUMBER);
      PROCEDURE fire_employee (emp_id NUMBER);
    END ' || pkgname || ';';
  END generate_spec;
  FUNCTION generate_body (pkgname VARCHAR2) RETURN
  VARCHAR2 AS
  BEGIN
    RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
      PROCEDURE raise_salary (emp_id NUMBER, amount
      NUMBER) IS

```

연습 11-1: 조건부 컴파일 사용 (계속)

```

BEGIN
    UPDATE employees SET salary = salary + amount
WHERE employee_id = emp_id;
    END raise_salary;

PROCEDURE fire_employee (emp_id NUMBER) IS
BEGIN
    DELETE FROM employees WHERE employee_id = emp_id;
END fire_employee;
END ' || pkgname || ';' ;
END generate_body;

a) emp_actions 파라미터를 전달하면서 Package Spec 을 생성합니다.
b) Package Spec 을 생성하고 래핑합니다.
c) Package Body 를 생성합니다.
d) Package Body 를 생성하고 래핑합니다.
e) 다음과 같이 래핑된 패키지에서 프로시저를 호출합니다.
    CALL emp_actions.raise_salary(120, 100);
f) 다음과 같이 USER_SOURCE 데이터 딕셔너리 뷰를 사용하여 코드가 숨겨져
    있는지 확인합니다.
    SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';

```

연습 해답 11-1: 조건부 컴파일 사용

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.

- 1) lab_11_01.sql 스크립트를 검토한 다음 실행합니다. 이 스크립트는 디버깅 코드 및 추적 정보를 표시하기 위한 플래그를 설정합니다. 이 스크립트는 my_pkg 패키지와 circle_area 프로시저도 생성합니다.

/home/oracle/labs/plpu/solns/sol_11_01.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
ALTER SESSION SET PLSQL_CCFLAGS = 'my_debug:FALSE,
my_tracing:FALSE';

CREATE OR REPLACE PACKAGE my_pkg AS
    SUBTYPE my_real IS
        $IF DBMS_DB_VERSION.VERSION < 10 $THEN NUMBER;
        -- check database version
        $ELSE                                BINARY_DOUBLE;
        $END
    my_pi my_real; my_e my_real;
END my_pkg;
/
CREATE OR REPLACE 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;
/

CREATE OR REPLACE PROCEDURE circle_area(radius
my_pkg.my_real) IS
    my_area my_pkg.my_real;
    my_datatype VARCHAR2(30);
BEGIN
    my_area := my_pkg.my_pi * radius * radius;
    DBMS_OUTPUT.PUT_LINE('Radius: ' || TO_CHAR(radius)
                         || ' Area: ' || TO_CHAR(my_area));
    $IF $$my_debug $THEN
        -- if my_debug is TRUE, run some debugging code
    END IF;
END;
```

연습 해답 11-1: 조건부 컴파일 사용 (계속)

```

SELECT DATA_TYPE INTO my_datatype FROM USER_ARGUMENTS
    WHERE OBJECT_NAME = 'CIRCLE_AREA' AND ARGUMENT_NAME =
'RADIUS';
    DBMS_OUTPUT.PUT_LINE('Datatype of the RADIUS argument
is: ' || my_datatype);
$END
END;
/

```

```

ALTER SESSION SET succeeded.
PACKAGE my_pkg Compiled.
PACKAGE BODY my_pkg Compiled.
PROCEDURE circle_area(radius Compiled.

```

- 2) lab_11_01에서 조건부 컴파일 지시어를 처리한 후에 DBMS_PREPROCESSOR 서브 프로그램을 사용하여 PL/SQL 단위의 포스트 처리된 소스 텍스트를 읽어 들입니다. SERVEROUTPUT 을 활성화합니다.

/home/oracle/labs/plpu/solns/sol_11_02.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

-- The code example assumes you are the student with the
-- account ora70. Substitute ora70 with your account
-- information.
SET SERVEROUTPUT ON

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
    BINARY_DOUBLE;
my_pi my_real; my_e my_real;
END my_pkg;

```

연습 해답 11-1: 조건부 컴파일 사용 (계속)

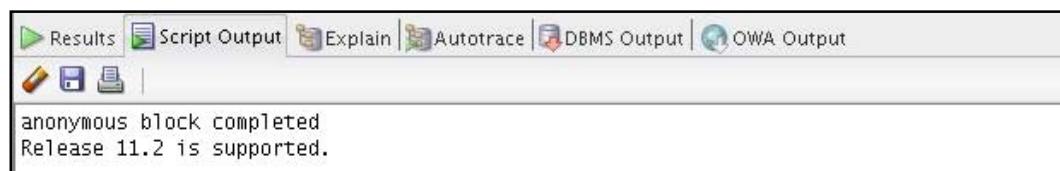
- 3) 조건부 컴파일에 DBMS_DB_VERSION 상수를 사용하는 PL/SQL 스크립트를 생성합니다. 코드는 다음의 오라클 데이터베이스 버전에 대해 테스트해야 합니다.
- 데이터베이스 버전이 10.1 이하일 경우에는 다음 오류 메시지가 표시되어야 합니다.
Unsupported database release.
 - 데이터베이스 버전이 11.1 이상일 경우에는 다음 메시지가 표시되어야 합니다.
Release 11.2 is supported.

/home/oracle/labs/plpu/solns/sol_11_03.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
BEGIN
$IF DBMS_DB_VERSION.VER_LE_10_1 $THEN
$ERROR 'unsupported database release.' $END

$ELSE
    DBMS_OUTPUT.PUT_LINE ('Release ' ||
DBMS_DB_VERSION.VERSION || '.' ||
DBMS_DB_VERSION.RELEASE || ' is
supported.');
    -- Note that this COMMIT syntax is newly supported in
10.2
    COMMIT WRITE IMMEDIATE NOWAIT;
$END
END;
/

```



- 4) CREATE_WWRAPPED 를 사용하여 데이터베이스에서 Package Spec 및 Package Body를 동적으로 생성하고 래핑하는 다음 코드를 lab_11_04.sql 스크립트에 사용해 봅니다. lab_11_04.sql 스크립트를 편집하여 PL/SQL 코드를 난독 처리하는 데 필요한 코드를 추가합니다. 스크립트를 저장하고 실행합니다.

```
DECLARE
-- the package_text variable contains the text to create
-- the package spec and body
  package_text VARCHAR2(32767);
  FUNCTION generate_spec (pkgname VARCHAR2) RETURN
  VARCHAR2 AS
BEGIN
  RETURN 'CREATE PACKAGE ' || pkgname || ' AS
```

연습 해답 11-1: 조건부 컴파일 사용 (계속)

```

PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER);
PROCEDURE fire_employee (emp_id NUMBER);
END ' || pkgname || ';'
END generate_spec;
FUNCTION generate_body (pkgname VARCHAR2) RETURN
VARCHAR2 AS
BEGIN
    RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
        PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER) IS
        BEGIN
            UPDATE employees SET salary = salary + amount
WHERE employee_id = emp_id;
        END raise_salary;

        PROCEDURE fire_employee (emp_id NUMBER) IS
        BEGIN
            DELETE FROM employees WHERE employee_id = emp_id;
        END fire_employee;
    END ' || pkgname || ';'
END generate_body;

```

- a) emp_actions 파라미터를 전달하면서 Package Spec 을 생성합니다.
- b) Package Spec 을 생성하고 래핑합니다.
- c) Package Body 를 생성합니다.
- d) Package Body 를 생성하고 래핑합니다.
- e) 다음과 같이 래핑된 패키지에서 프로시저를 호출합니다.

```
CALL emp_actions.raise_salary(120, 100);
```

- f) 다음과 같이 USER_SOURCE 데이터 딕셔너리 뷰를 사용하여 코드가 숨겨져 있는지 확인합니다.

```
SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';
```

/home/oracle/labs/plpu/solns/soln_11_04.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

DECLARE
-- the package_text variable contains the text to create
the package spec and body
    package_text VARCHAR2(32767);
    FUNCTION generate_spec (pkgname VARCHAR2) RETURN
VARCHAR2 AS
BEGIN
    RETURN 'CREATE PACKAGE ' || pkgname || ' AS
        PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER);
        PROCEDURE fire_employee (emp_id NUMBER);
    END';

```

연습 해답 11-1: 조건부 컴파일 사용 (계속)

```

        END ' || pkgname || ';';
END generate_spec;
FUNCTION generate_body (pkgname VARCHAR2) RETURN
VARCHAR2 AS
BEGIN
    RETURN 'CREATE PACKAGE BODY ' || pkgname || ' AS
        PROCEDURE raise_salary (emp_id NUMBER, amount
NUMBER) IS
            BEGIN
                UPDATE employees SET salary = salary + amount WHERE
employee_id = emp_id;
            END raise_salary;
        PROCEDURE fire_employee (emp_id NUMBER) IS
            BEGIN
                DELETE FROM employees WHERE employee_id =
emp_id;
            END fire_employee;
        END ' || pkgname || ';';
END generate_body;

BEGIN

-- generate package spec
package_text := generate_spec('emp_actions');

-- create and wrap the package spec
SYS.DBMS_DDL.CREATE_WWRAPPED(package_text);

-- generate package body
package_text := generate_body('emp_actions');

-- create and wrap the package body
SYS.DBMS_DDL.CREATE_WWRAPPED(package_text);
END;
/
-- call a procedure from the wrapped package

CALL emp_actions.raise_salary(120, 100);

-- Use the USER_SOURCE data dictionary view to verify
that -- the code is hidden as follows:

SELECT text FROM USER_SOURCE WHERE name = 'EMP_ACTIONS';

```

연습 해답 11-1: 조건부 컴파일 사용 (계속)

단원 12 의 연습 및 해답

이 연습에서는 DEPTREE_FILL 프로시저와 IDEPTREE 뷰를 사용하여 스키마에서 종속성을 조사하며, 유효하지 않은 프로시저, 함수, 패키지 및 뷰를 재컴파일합니다.

참고: 연습에서 견너뛴 단계가 있는 경우 해당 연습 단계의 해답 스크립트를 실행한 후 다음 단계나 연습으로 진행하십시오.

연습 12-1: 스키마에서 종속성 관리

이 연습에서는 DEPTREE_FILL 프로시저와 IDEPTREE 뷰를 사용하여 스키마에서 종속성을 조사하며, 유효하지 않은 프로시저, 함수, 패키지 및 뷰를 재컴파일합니다.

- 1) add_employee 프로시저와 valid_deptid 함수를 포함하는 모든 종속성을 보여 주는 트리 구조를 생성합니다.

참고: add_employee 및 valid_deptid 는 "함수 생성" 단원에서 생성했습니다. 프로시저와 함수를 생성해야 할 경우 연습 3의 해답 스크립트를 실행할 수 있습니다.

- a) /home/oracle/labs/plpu/labs 폴더에 있는 utldtree.sql 스크립트를 로드하고 실행합니다.
- b) add_employee 프로시저에 대해 deptree_fill 프로시저를 실행합니다.
- c) IDEPTREE 뷰를 query 하여 결과를 확인합니다.
- d) valid_deptid 함수에 대해 deptree_fill 프로시저를 실행합니다.
- e) IDEPTREE 뷰를 query 하여 결과를 확인합니다.

시간 여유가 있을 경우 다음 연습을 완료합니다.

- 2) 유효하지 않은 객체를 동적으로 검증합니다.
 - a) EMPLOYEES 테이블의 복사본을 만들고 이름을 EMPS로 지정합니다.
 - b) EMPLOYEES 테이블을 변경하고 데이터 유형이 NUMBER(9,2)인 TOTSAL 열을 추가합니다.
 - c) 모두 유효하지 않은 객체의 이름, 유형, 상태를 표시하는 query를 생성하여 저장합니다.
 - d) compile_pkg("동적 SQL 사용" 단원의 연습 7에서 생성함)에서 유효하지 않은 프로시저, 함수 및 패키지를 모두 재컴파일하는 recompile이라는 프로시저를 스키마에 추가합니다. Native Dynamic SQL을 사용하여 유효하지 않은 객체 유형을 변경하고 컴파일합니다.
 - e) compile_pkg.recompile 프로시저를 실행합니다.
 - f) 단계 3 c.에서 생성한 스크립트 파일을 실행하여 STATUS 열 값을 확인합니다. 여전히 INVALID 상태의 객체가 있습니까?

연습 해답 12-1: 스키마에서 종속성 관리

이 연습에서는 DEPTREE_FILL 프로시저와 IDEPTREE 뷰를 사용하여 스키마에서 종속성을 조사하며, 유효하지 않은 프로시저, 함수, 패키지 및 뷰를 재컴파일합니다.

- 1) add_employee 프로시저와 valid_deptid 함수를 포함하는 모든 종속성을 보여 주는 트리 구조를 생성합니다.

참고: add_employee 및 valid_deptid 는 "함수 생성" 단원에서 생성했습니다. 프로시저와 함수를 생성해야 할 경우 연습 3의 해답 스크립트를 실행할 수 있습니다.

- a) /home/oracle/labs/plpu/labs 폴더에 있는 utldtree.sql 스크립트를 로드하고 실행합니다.

/home/oracle/labs/plpu/solns/utldtree.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```

Rem
Rem $Header: utldtree.sql,v 1.2 1992/10/26 16:24:44 RKOOI
Stab $
Rem
Rem Copyright (c) 1991 by Oracle Corporation
Rem NAME
Rem deptree.sql - Show objects recursively dependent on
Rem given object
Rem DESCRIPTION
Rem This procedure, view and temp table will allow you to
Rem see all objects that are (recursively) dependent on
the given Rem object.
Rem Note: you will only see objects for which you have
Rem permission.
Rem Examples:
Rem execute deptree_fill('procedure', 'scott', 'billing');
Rem select * from deptree order by seq#;
Rem
Rem execute deptree_fill('table', 'scott', 'emp');
Rem select * from deptree order by seq#;
Rem

Rem execute deptree_fill('package body', 'scott',
Rem 'accts_payable');
Rem select * from deptree order by seq#;
Rem
Rem A prettier way to display this information than
Rem select * from deptree order by seq#;
Rem
Rem select * from ideptree;

```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

```

Rem   This shows the dependency relationship via indenting.
Rem   Notice that no order by clause is needed with
ideptree.
Rem   RETURNS
Rem
Rem   NOTES
Rem   Run this script once for each schema that needs this
Rem   utility.
Rem   MODIFIED      (MM/DD/YY)
Rem   rkooi         10/26/92 - owner -> schema for SQL2
Rem   glumpkin      10/20/92 - Renamed from DEPTREE.SQL
Rem   rkooi         09/02/92 - change ORU errors
Rem   rkooi         06/10/92 - add rae errors
Rem   rkooi         01/13/92 - update for sys vs. regular user
Rem   rkooi         01/10/92 - fix ideptree
Rem   rkooi         01/10/92 - Better formatting, add ideptree
view
Rem   rkooi         12/02/91 - deal with cursors
Rem   rkooi         10/19/91 - Creation

DROP SEQUENCE deptree_seq
/
CREATE SEQUENCE deptree_seq cache 200
/* cache 200 to make sequence faster */

/
DROP TABLE deptree_temptab
/
CREATE TABLE deptree_temptab
(
    object_id          number,
    referenced_object_id number,
    nest_level         number,
    seq#               number
)
/
CREATE OR REPLACE PROCEDURE deptree_fill (type char, schema
char, name char) IS
    obj_id number;
BEGIN
    DELETE FROM deptree_temptab;
    COMMIT;
    SELECT object_id INTO obj_id FROM all_objects
        WHERE owner = upper(deptree_fill.schema)

    AND    object_name  = upper(deptree_fill.name)
        AND    object_type = upper(deptree_fill.type);
    INSERT INTO deptree_temptab
        VALUES(obj_id, 0, 0, 0);
    INSERT INTO deptree_temptab
        SELECT object_id, referenced_object_id,
            level, deptree_seq.nextval
        FROM public_dependency
        CONNECT BY PRIOR object_id = referenced_object_id

```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

```

        START WITH referenced_object_id = deptree_fill.obj_id;
EXCEPTION
    WHEN no_data_found then
        raise_application_error(-20000, 'ORU-10013: ' ||
            type || ' ' || schema || '.' || name || ' was not
found.');
END;
/

DROP VIEW deptree
/

SET ECHO ON

REM This view will succeed if current user is sys. This view
REM shows which shared cursors depend on the given object.
REM If the current user is not sys, then this view get an
REM error
REM either about lack of privileges or about the non-
REM existence of REM table x$kglxss.

SET ECHO OFF
CREATE VIEW sys.deptree
    (nested_level, type, schema, name, seq#)
AS
    SELECT d.nest_level, o.object_type, o.owner,
o.object_name, d.seq#
    FROM deptree temptab d, dba_objects o
    WHERE d.object_id = o.object_id (+)
UNION ALL
    SELECT d.nest_level+1, 'CURSOR', '<shared>',
'''||c.kglnaobj||''', d.seq#+.5
    FROM deptree temptab d, x$kgldp k, x$kglob g, obj$ o,
user$ u, x$kglob c,
x$kglxss a
    WHERE d.object_id = o.obj#
    AND o.name = g.kglnaobj
    AND o.owner# = u.user#
    AND u.name = g.kglnaown
    AND g.kglhdadr = k.kglrfhdl
    AND k.kglhdadr = a.kglhdadr /* make sure it is not a
transitive */
    AND k.kgldepno = a.kglxsdep /* reference, but a
direct one */
    AND k.kglhdadr = c.kglhdadr
    AND c.kglhdnsps = 0 /* a cursor */
/
SET ECHO ON

REM This view will succeed if current user is not sys. This
view
REM does *not* show which shared cursors depend on the given
REM object.

```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

```

REM If the current user is sys then this view will get an
error
REM indicating that the view already exists (since prior
view
REM create will have succeeded).

SET ECHO OFF
CREATE VIEW deptree
  (nested_level, type, schema, name, seq#)
AS
  select d.nest_level, o.object_type, o.owner,
o.object_name, d.seq#
  FROM deptree temptab d, all_objects o
  WHERE d.object_id = o.object_id (+)
/
DROP VIEW ideptree
/
CREATE VIEW ideptree (dependencies)
AS
  SELECT lpad(' ',3*(max(nested_level))) || max(nvl(type,
'<no permission>')
  || ' ' || schema || decode(type, NULL, '', '.') || name)
  FROM deptree
  GROUP BY seq# /* So user can omit sort-by when selecting
from ideptree */
/

```

The screenshot shows the Oracle SQL Developer interface with the 'Script Output' tab selected. The output window displays the execution of the provided PL/SQL script, showing errors at various points where the user does not have the necessary privileges.

```

Results Script Output Explain Autotrace DBMS Output OWA Output
[Edit] [Run] [Stop]

Error starting at line 47 in command:
DROP SEQUENCE deptree_seq
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.
CREATE SEQUENCE succeeded.

Error starting at line 53 in command:
DROP TABLE deptree temptab
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
CREATE TABLE succeeded.
Warning: execution completed with warning
PROCEDURE deptree_fill Compiled.

Error starting at line 88 in command:
DROP VIEW deptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
REM This view will succeed if current user is sys. This view

REM shows which shared cursors depend on the given object. If REM the current user is not sys, then this view get an error
REM either about lack of privileges or about the non-existence of REM table x$kglix.

SET ECHO OFF

```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

```
Error starting at line 98 in command:
CREATE VIEW sys.deptree
  (nested_level, type, schema, name, seq#)
AS
  SELECT d.nest_level, o.object_type, o.owner, o.object_name, d.seq#
  FROM deptree temptab d, dba_objects o
 WHERE d.object_id = o.object_id (+)
UNION ALL
  SELECT d.nest_level+1, 'CURSOR', '<shared>', ''||c.kglnaobj||'', d.seq#+.5
  FROM deptree temptab d, x$kgldp k, x$kglob g, obj$ o, user$ u, x$kglob c,
       x$kglx$ a
 WHERE d.object_id = o.obj#
   AND o.name = g.kglnaobj
   AND o.owner# = u.user#
   AND u.name = g.kglnaown
   AND g.kglhdadr = k.kglrfhdl /* make sure it is not a transitive */
   AND k.kglhdadr = a.kglhdadr /* reference, but a direct one */
   AND k.kglhdadr = c.kglhdadr
   AND c.kglhdnsp = 0 /* a cursor */
Error at Command Line:102 Column:7
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
REM This view will succeed if current user is not sys. This view

REM does *not* show which shared cursors depend on the given

REM object.

REM If the current user is sys then this view will get an error

REM indicating that the view already exists (since prior view
```

```
REM create will have succeeded).

SET ECHO OFF

CREATE VIEW succeeded.

Error starting at line 136 in command:
DROP VIEW ideptree
Error report:
SQL Error: ORA-00942: table or view does not exist
00942. 00000 -  "table or view does not exist"
*Cause:
*Action:
CREATE VIEW succeeded.
```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

- b) add_employee 프로시저에 대해 deptree_fill 프로시저를 실행합니다.

/home/oracle/labs/plpu/solns/sol_12_01_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

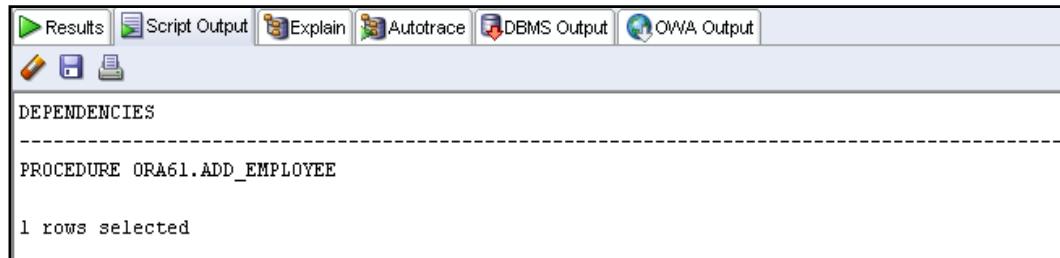
```
EXECUTE deptree_fill('PROCEDURE', USER, 'add_employee')
```



- c) IDEPTREE 뷰를 query 하여 결과를 확인합니다.

/home/oracle/labs/plpu/solns/sol_12_01_c.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

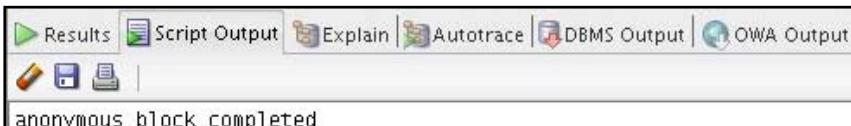
```
SELECT * FROM IDEPTREE;
```



- d) valid_deptid 함수에 대해 deptree_fill 프로시저를 실행합니다.

/home/oracle/labs/plpu/solns/sol_12_01_d.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE deptree_fill('FUNCTION', USER, 'valid_deptid')
```



연습 해답 12-1: 스키마에서 종속성 관리 (계속)

- e) IDEPTREE 뷰를 query 하여 결과를 확인합니다.

/home/oracle/labs/plpu/solns/sol_12_01_e.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Execute Statement (F9) 아이콘을 눌러 스크립트를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT * FROM IDEPTREE;
```

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
|-----|
DEPENDENCIES
-----
PROCEDURE ORA61.ADD_EMPLOYEE
FUNCTION ORA61.VALID_DEPTID
2 rows selected
```

시간 여유가 있을 경우 다음 연습을 완료합니다.

- 2) 유효하지 않은 객체를 동적으로 검증합니다.

- a) EMPLOYEES 테이블의 복사본을 만들고 이름을 EMPS로 지정합니다.

/home/oracle/labs/plpu/solns/sol_12_02_a.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
CREATE TABLE emps AS
SELECT * FROM employees;
```

```
Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
|-----|
CREATE TABLE succeeded.
```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

- b) EMPLOYEES 테이블을 변경하고 데이터 유형이 NUMBER(9,2)인 TOTSAL 열을 추가합니다.

/home/oracle/labs/plpu/solns/sol_12_02_b.sql 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
ALTER TABLE employees
ADD (totsal NUMBER(9,2));
```

The screenshot shows the SQL Worksheet interface with the 'Results' tab selected. The output window displays the message: 'ALTER TABLE employees succeeded.'

- c) 모두 유효하지 않은 객체의 이름, 유형, 상태를 표시하는 query를 생성하여 저장합니다.

/home/oracle/labs/plpu/solns/sol_12_02_c.sql 스크립트를 엽니다. SQL Worksheet 도구 모음에서 Execute Statement (F9) 아이콘을 눌러 스크립트를 실행합니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT object_name, object_type, status
FROM USER_OBJECTS
WHERE status = 'INVALID';
```

The screenshot shows the SQL Worksheet interface with the 'Results' tab selected. The output window displays a table of invalid objects:

OBJECT_NAME	OBJECT_TYPE	STATUS
EMP_ACTIONS	PACKAGE BODY	INVALID
UPD_MINSalary_TRG	TRIGGER	INVALID
DELETE_EMP_TRG	TRIGGER	INVALID
EMP_PKG	PACKAGE BODY	INVALID
EMP_PKG	PACKAGE	INVALID
GET_EMPLOYEE	PROCEDURE	INVALID
UPDATE_JOB_HISTORY	TRIGGER	INVALID
SECURE_EMPLOYEES	TRIGGER	INVALID

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

- d) `compile_pkg`("동적 SQL 사용" 단원의 연습 6에서 생성함)에서 모든 유효하지 않은 프로시저, 함수 및 패키지를 재컴파일하는 `recompile`이라는 프로시저를 스키마에 추가합니다. Native Dynamic SQL을 사용하여 유효하지 않은 객체 유형을 변경하고 컴파일합니다.

`/home/oracle/labs/plpu/solns/sol_12_02_d.sql` 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다. 새로 추가된 코드는 다음 코드 상자에서 굵은체 글자로 강조 표시됩니다.

```

CREATE OR REPLACE PACKAGE compile_pkg IS
    PROCEDURE make(name VARCHAR2);
    PROCEDURE recompile;
END compile_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY compile_pkg IS

    PROCEDURE execute(stmt VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE(stmt);
        EXECUTE IMMEDIATE stmt;
    END;

    FUNCTION get_type(name VARCHAR2) RETURN VARCHAR2 IS
        proc_type VARCHAR2(30) := NULL;
    BEGIN
        /*
         * The ROWNUM = 1 is added to the condition
         * to ensure only one row is returned if the
         * name represents a PACKAGE, which may also
         * have a PACKAGE BODY. In this case, we can
         * only compile the complete package, but not
         * the specification or body as separate
         * components.
        */
        SELECT object_type INTO proc_type
        FROM user_objects
        WHERE object_name = UPPER(name)
        AND ROWNUM = 1;
        RETURN proc_type;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN NULL;
    END;

    PROCEDURE make(name VARCHAR2) IS
        stmt      VARCHAR2(100);
        proc_type VARCHAR2(30) := get_type(name);
    BEGIN

```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

```
IF proc_type IS NOT NULL THEN
    stmt := 'ALTER ||| proc_type ||' '||| name |||
COMPILE';

execute(stmt);
ELSE
    RAISE_APPLICATION_ERROR(-20001,
        'Subprogram ''||| name ||''' does not exist');
END IF;
END make;

PROCEDURE recompile IS
    stmt VARCHAR2(200);
    obj_name user_objects.object_name%type;
    obj_type user_objects.object_type%type;
BEGIN
    FOR objrec IN (SELECT object_name, object_type
                    FROM user_objects
                   WHERE status = 'INVALID'
                     AND object_type <> 'PACKAGE BODY')
    LOOP
        stmt := 'ALTER ||| objrec.object_type ||' '|||
objrec.object_name ||' ' COMPILE';
        execute(stmt);
    END LOOP;
END recompile;

END compile_pkg;
/
SHOW ERRORS
```

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

```
PACKAGE compile_pkg Compiled.
No Errors.
PACKAGE BODY compile_pkg Compiled.
No Errors.
```

연습 해답 12-1: 스키마에서 종속성 관리 (계속)

- e) `compile_pkg.recompile` 프로시저를 실행합니다.

`/home/oracle/labs/plpu/solns/sol_12_02_e.sql` 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE compile_pkg.recompile
```

- f) 단계 3 c.에서 생성한 스크립트 파일을 실행하여 STATUS 열 값을 확인합니다. 여전히 INVALID 상태의 객체가 있습니까?

`/home/oracle/labs/plpu/solns/sol_12_02_f.sql` 스크립트를 엽니다. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5)를 누릅니다. 코드 및 결과가 다음과 같이 표시됩니다.

```
SELECT object_name, object_type, status
FROM USER_OBJECTS
WHERE status = 'INVALID';
```

	OBJECT_NAME	OBJECT_TYPE	STATUS
1	EMP_ACTIONS	PACKAGE BODY	INVALID

부록 AP
추가 연습 및 해답

목차

연습 1	3
연습 1-1: 새 SQL Developer 데이터베이스 연결 생성	4
연습 1-2: JOBS 테이블에 새 직무 추가	5
연습 1-3: JOB_HISTORY 테이블에 새 행 추가	6
연습 1-4: 직무에 대한 최소 급여 및 최대 급여 개선	7
연습 1-5: 사원 급여 모니터	8
연습 1-6: 사원의 총 근무 연수 검색	9
연습 1-7: 사원이 종사했던 다른 직무의 총 수 검색	10
연습 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성	11
연습 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성	12
연습 해답: 1-1: 새 SQL Developer 데이터베이스 연결 생성	13
연습 해답 1-2: JOBS 테이블에 새 직무 추가	15
연습 해답 1-3: JOB_HISTORY 테이블에 새 행 추가	18
연습 해답 1-4: 직무에 대한 최소 급여 및 최대 급여 개선	22
연습 해답 1-5: 사원 급여 모니터	26
연습 해답 1-6: 사원의 총 근무 연수 검색	31
연습 해답 1-7: 사원이 종사했던 다른 직무의 총 수 검색	35
연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성	38
연습 해답 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성	45
연습 2	48
연습 2-1: VIDEO_PKG 패키지 생성	49
연습 해답 2-1: VIDEO_PKG 패키지 생성	51

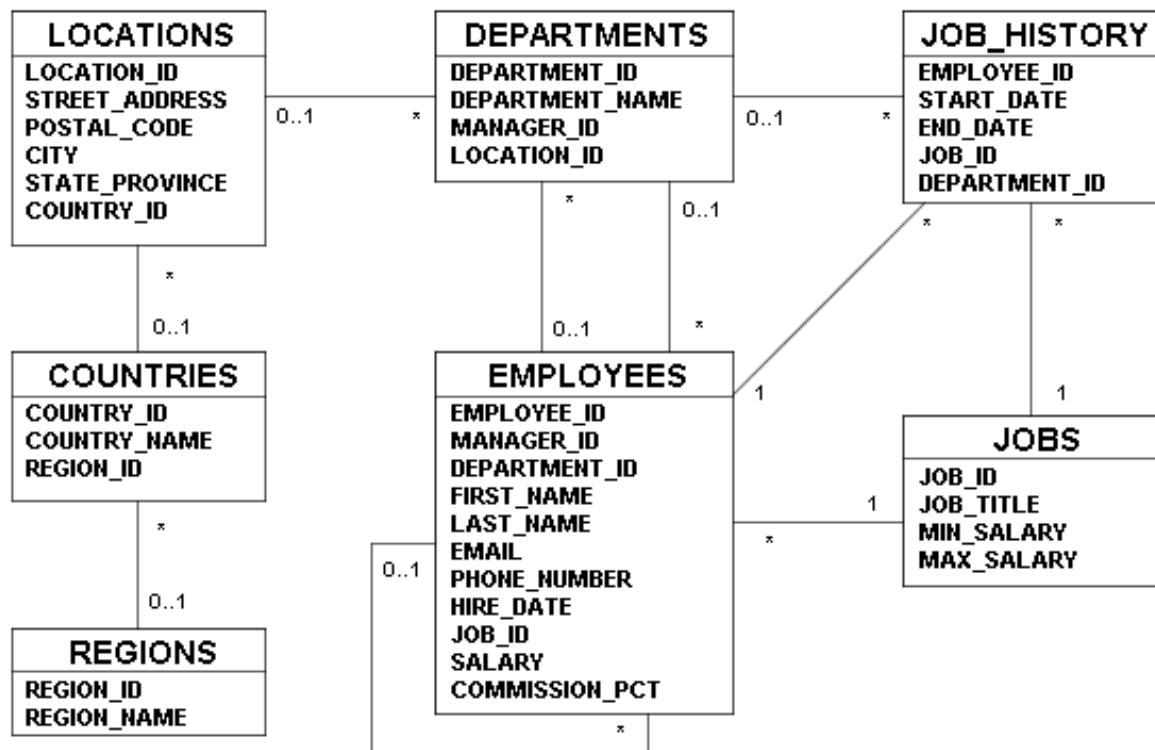
연습 1

추가 연습은 *Oracle Database 11g: Develop PL/SQL Program Units* 과정의 보충 자료로 제공됩니다. 이 연습에서는 본 과정에서 설명한 개념을 활용합니다. 추가 연습은 두 단원으로 구성됩니다.

단원 1은 내장 프로시저, 함수, 패키지 및 트리거를 생성하고 SQL Developer 또는 SQL*Plus를 개발 환경으로 하여 오라클 제공 패키지를 사용하는 추가 연습 문제입니다. 추가 연습의 이 부분에 사용되는 테이블은 EMPLOYEES, JOBS, JOB_HISTORY 및 DEPARTMENTS입니다.

각 연습의 시작 부분에는 엔티티 관계 도표가 나와 있습니다. 각 엔티티 관계 도표는 테이블 엔티티와 이들의 관계를 보여줍니다. 여기에 포함된 테이블과 데이터에 대한 보다 자세한 정의는 부록 "추가 연습: 테이블 설명 및 데이터"를 참조하십시오.

HR (Human Resources) 스키마 엔티티 관계 도표



연습 1-1: SQL Developer 데이터베이스 연결 생성

이 연습에서는 연결 정보를 사용하여 SQL Developer를 시작하고 새 데이터베이스 연결을 생성합니다.

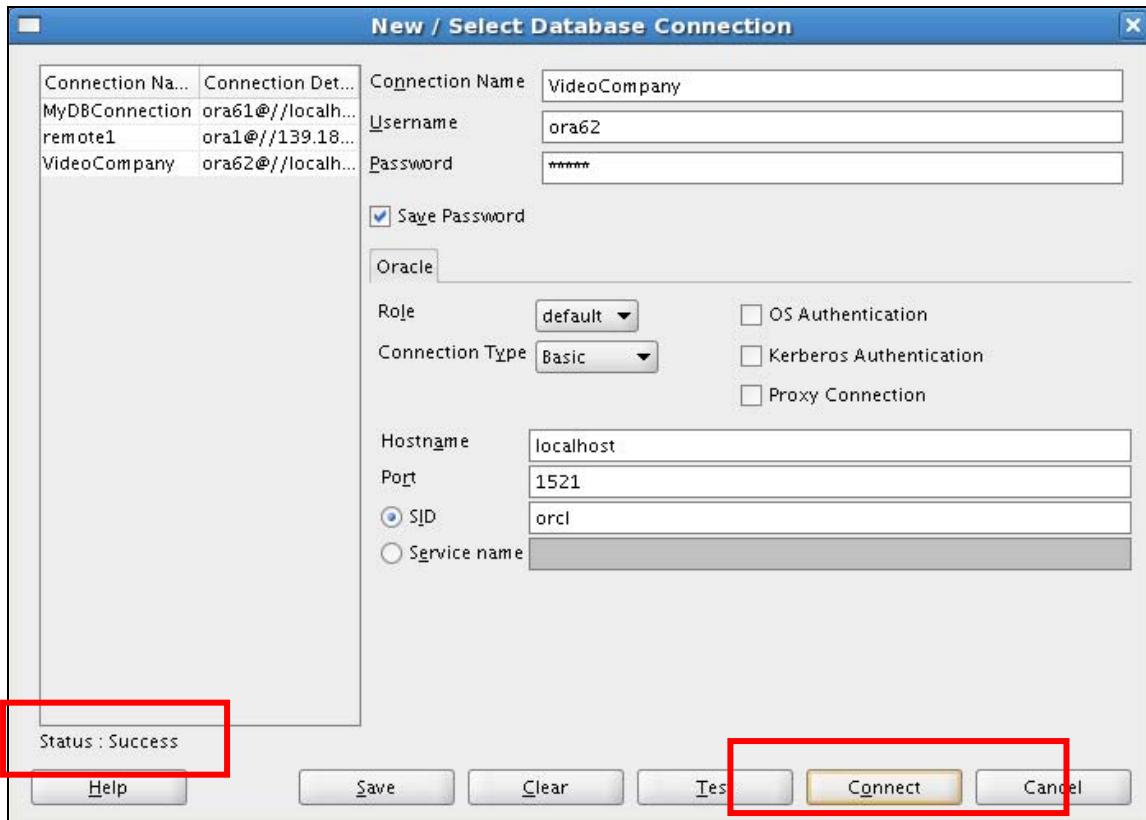
강사가 알려준 유저 ID 및 암호(예: ora62)를 사용하여 SQL Developer를 시작합니다.

1) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

- a) Connection Name: VideoCompany
- b) Username: ora62
- c) Password: ora62
- d) Hostname: PC의 호스트 이름을 입력합니다.
- e) Port: 1521
- f) SID: ORCL

2) 새 연결을 테스트합니다. Status 가 Success 이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.

- a) Connections 탭 페이지에서 VideoCompany 아이콘을 두 번 누릅니다.
- b) New>Select Database Connection window 의 Test 버튼을 누릅니다. 상태가 Success 이면 Connect 버튼을 누릅니다.



연습 1-2: JOBS 테이블에 새 직무 추가

이 연습에서는 JOBS 테이블에 새로운 직무를 추가하는 서브 프로그램을 생성합니다.

- 1) JOBS 테이블에 새 주문을 입력하는 NEW_JOB이라는 내장 프로시저를 생성합니다. 이 프로시저는 세 가지 파라미터를 사용해야 합니다. 첫번째 파라미터와 두번째 파라미터는 직무 ID 와 직위를 제공하고, 세번째 파라미터는 최소 급여를 제공합니다. 새로운 직무의 최대 급여를 직무 ID 에 대해 제공된 최소 급여의 두 배로 설정합니다.
- 2) SERVEROUTPUT 을 활성화한 다음 직무 ID 가 'SY_ANAL'이고 직위가 'System Analyst'이고 최소 급여가 6000 인 새로운 직무를 추가하는 프로시저를 호출합니다.
- 3) 행이 추가되었는지 확인하고 다음 연습에서 사용할 새 직무 ID 를 기록해둡니다. 변경 사항을 커밋합니다.

연습 1-3: JOB_HISTORY 테이블에 새 행 추가

이 연습에서는 JOB_HISTORY 테이블에 기존 사원에 대한 새 행을 추가합니다.

- 1) JOB_HISTORY 테이블에 연습 1 b에서 생성한 새 직무 ID ('SY_ANAL')로 자신의 직무를 변경하는 사원에 대한 새 행을 추가하는 ADD_JOB_HIST라는 내장 프로시저를 생성합니다.
 - a) 이 프로시저는 두 가지 파라미터를 제공해야 합니다. 하나는 직무를 변경하는 사원 ID에 대한 파라미터이고, 다른 하나는 새로운 직무 ID에 대한 파라미터입니다.
 - b) EMPLOYEES 테이블에서 사원 ID를 읽고 JOB_HISTORY 테이블에 삽입합니다.
 - c) JOB_HISTORY 테이블의 이 행에 대해 이 사원의 입사일을 시작 날짜로, 오늘 날짜를 종료 날짜로 설정합니다.
 - d) EMPLOYEES 테이블에서 이 사원의 채용 날짜를 오늘 날짜로 변경합니다.
 - e) 이 사원의 직무 ID를 파라미터로 전달된 직무 ID ('SY_ANAL' 직무 ID 사용)로 갱신하고 최소 급여와 동일한 급여를 해당 직무 ID + 500으로 갱신합니다.

참고: 존재하지 않는 사원을 삽입하려는 시도를 처리하려면 예외 처리를 추가합니다.

- 2) ADD_JOB_HIST 프로시저를 호출하기 전에 EMPLOYEES, JOBS 및 JOB_HISTORY 테이블에 대한 모든 트리거를 비활성화합니다.
- 3) SERVEROUTPUT 을 활성화한 다음 사원 ID 106 및 직무 ID 'SY_ANAL'을 파라미터로 사용하여 프로시저를 실행합니다.
- 4) JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 사원 106의 변경 사항을 확인하고 커밋합니다.
- 5) EMPLOYEES, JOBS 및 JOB_HISTORY 테이블에 대한 트리거를 다시 활성화합니다.

연습 1-4: 직무에 대한 최소 급여 및 최대 급여 갱신

이 연습에서는 JOBS 테이블에서 직무에 대한 최소 급여와 최대 급여를 갱신하는 프로그램을 생성합니다.

- 1) JOBS 테이블의 특정 직무 ID에 대해 최소 급여와 최대 급여를 갱신하는 UPD_JOBSAL이라는 내장 프로시저를 생성합니다. 이 프로시저는 직무 ID, 새로운 최소 급여, 새로운 최대 급여 세 가지 파라미터를 제공해야 합니다. JOBS 테이블의 직무 ID가 유효하지 않을 경우에 대한 예외 처리를 추가합니다. 제공된 최대 급여가 최소 급여보다 적을 경우 예외를 발생시키고 JOBS 테이블의 행이 잠겨 있을 경우 표시되는 메시지를 제공합니다.
힌트: 리소스 잠김/사용 중 오류 번호는 -54입니다.
- 2) SERVEROUTPUT 을 활성화한 다음 직무 ID 'SY_ANAL', 최소 급여 7000, 최대 급여 140을 사용하여 UPD_JOBSAL 프로시저를 실행합니다.
참고: 이 경우 예외 메시지가 나타나야 합니다.
- 3) EMPLOYEES 및 JOBS 테이블에 대한 트리거를 비활성화합니다.
- 4) 직무 ID 'SY_ANAL', 최소 급여 7000, 최대 급여 14000을 사용하여 UPD_JOBSAL 프로시저를 실행합니다.
- 5) JOBS 테이블을 query 하여 변경 사항을 확인한 다음 커밋합니다.
- 6) EMPLOYEES 및 JOBS 테이블에 대한 트리거를 활성화합니다.

연습 1-5: 사원 급여 모니터

이 연습에서는 직무 유형에 따라 사원이 평균 급여를 초과했는지 여부를 모니터하는 프로시저를 생성합니다.

- 1) SECURE_EMPLOYEES 트리거를 비활성화합니다.
- 2) EMPLOYEES 테이블에서 최대 세 자와 기본값 NO 를 저장할 EXCEED_AVGSAL 열을 추가합니다. YES 또는 NO 값을 허용하려면 CHECK 제약 조건을 사용합니다.
- 3) 각 사원의 급여가 JOB_ID 의 평균 급여를 초과하는지 여부를 확인하는 CHECK_AVGSAL 이라는 내장 프로시저를 생성합니다.
 - a) 직무에 대한 평균 급여는 JOBS 테이블의 정보를 사용하여 계산됩니다.
 - b) 사원의 급여가 해당 직무의 평균을 초과할 경우 EMPLOYEES 테이블의 EXCEED_AVGSAL 열 값을 YES 로 갱신하고, 그렇지 않은 경우에는 값을 NO 로 설정합니다.
 - c) 커서를 사용하여 query 에서 FOR UPDATE 옵션을 통해 사원의 행을 선택합니다.
 - d) 레코드가 잠겨 있을 경우에 대한 예외 처리를 추가합니다.
힌트: 리소스 잠김/사용 중 오류 번호는 -54 입니다.
 - e) 지정된 직무 ID 의 평균 급여를 파라미터로 결정하는 GET_JOB_AVGSAL 이라는 로컬 함수를 생성하여 사용하십시오.
- 4) CHECK_AVGSAL 프로시저를 실행합니다. 수정한 결과를 확인하려면 사원 ID, 직무, 해당 직무의 평균 급여, 사원 급여 및 급여가 해당 직무의 평균을 초과하는 사원의 exceed_avgsal 표시자 열을 표시하는 query 를 작성합니다. 끝으로 변경 사항을 커밋합니다.

참고: 다음 연습은 함수 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

연습 1-6: 사원의 총 근무 연수 검색

이 연습에서는 특정 사원의 근무 연수를 검색하는 서브 프로그램을 생성합니다.

- 1) 특정 사원의 총 근무 연수를 검색하는 GET_YEARS_SERVICE 라는 내장 함수를 생성합니다. 함수는 사원 ID를 파라미터로 사용하여 근무 연수를 반환해야 합니다. 사원 ID가 유효하지 않을 경우에 대한 오류 처리를 추가합니다.
- 2) ID 가 999 인 사원에 대한 DBMS_OUTPUT.PUT_LINE 호출에서 GET_YEARS_SERVICE 함수를 호출합니다.
- 3) GET_YEARS_SERVICE 함수를 호출하는 DBMS_OUTPUT.PUT_LINE 을 사용하여 사원 106 의 근무 연수를 표시합니다. SERVEROUTPUT 을 활성화해야 합니다.
- 4) 지정된 사원의 JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 수정 사항이 정확한지 확인합니다. 이 페이지에 결과로 표시된 값과 실제 query 를 수행하여 얻은 결과는 다를 수 있습니다.

연습 1-7: 사원이 종사했던 다른 직무의 총 수 검색

이 연습에서는 한 사원이 재직 기간 동안에 수행했던 다른 직무 수를 검색하는 프로그램을 생성합니다.

- 1) 사원이 종사했던 다른 직무의 근무 연수를 검색하는 GET_JOB_COUNT 라는 내장 함수를 생성합니다.
 - a) 함수는 사원 ID를 파라미터로 사용하고 현재 직무를 비롯하여 사원이 지금까지 수행했던 다른 직무 수를 반환해야 합니다.
 - b) 사원 ID가 유효하지 않을 경우에 대한 예외 처리를 추가합니다.
힌트: JOB_HISTORY 테이블에서 고유한 직무 ID를 사용하되, 이 ID가 사원이 이미 종사했던 직무 ID 중 하나일 경우 현재 직무 ID를 제외시키십시오.
 - c) 두 query에 대한 UNION을 작성하여 PL/SQL 테이블로 가져온 행 수를 계산합니다.
 - d) FETCH와 BULK COLLECT INTO를 함께 사용하여 사원의 고유 직무를 확인합니다.
- 2) ID가 176인 사원에 대해 함수를 호출합니다. SERVEROUTPUT을 활성화해야 합니다.

참고: 다음 연습은 패키지 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

연습 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성

이 연습에서는 NEW_JOB, ADD_JOB_HIST, UPD_JOBSAL 프로시저와 GET_YEARS_SERVICE 및 GET_JOB_COUNT 함수를 포함하는 EMPJOB_PKG라는 패키지를 생성합니다.

- 1) 서브 프로그램 생성자가 모두 공용(public)인 Package Spec 을 생성합니다. 서브 프로그램의 로컬 정의 유형을 Package Spec 으로 이동합니다.
- 2) 서브 프로그램을 구현하여 Package Body 를 생성하되, Package Spec 으로 이동한 유형은 서브 프로그램 구현에서 제거합니다.
- 3) EMPJOB_PKG.NEW_JOB 프로시저를 호출하여 ID 가 PR_MAN, 직위가 Public Relations Manager 이고, 급여가 6250 인 새로운 직무를 생성합니다.
SERVEROUTPUT 을 활성화해야 합니다.
- 4) 사원 ID 110 의 직무를 ID PR_MAN 으로 수정하는
EMPJOB_PKG.ADD_JOB_HIST 프로시저를 호출합니다.
참고: UPDATE_JOB_HISTORY 트리거는 ADD_JOB_HIST 프로시저를 실행하기 전에 비활성화했다가 이 프로시저를 실행한 후에 다시 활성화해야 합니다.
- 5) JOBS, JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 결과를 확인합니다.
참고: 다음 연습은 데이터베이스 트리거 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

연습 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성

이 연습에서는 특정 직무의 최소 급여 및 최대 급여가 수정되지 않도록 하여 해당 직무 ID를 가진 기존 사원의 급여가 해당 직무에 대해 지정된 새 범위를 벗어나도록 하는 트리거를 생성합니다.

- 1) JOBS 테이블의 MIN_SALARY 및 MAX_SALARY 열에서 갱신되는 모든 행 앞에 실행되는 CHECK_SAL_RANGE라는 트리거를 생성합니다.
 - a) 최소 급여 값이나 최대 급여 값이 변경된 경우 EMPLOYEES 테이블에서 해당 직무 ID를 가진 기존 사원의 급여가 이 직무 ID에 대해 지정된 새 급여 범위 내에 있는지 검사합니다.
 - b) 기존 사원의 레코드에 영향을 미치는 급여 범위 변경을 다루는 예외 처리를 추가합니다.
- 2) 새로운 최소 급여와 최대 급여를 5000과 7000으로 각각 설정하여 SY_ANAL 직무로 트리거를 테스트합니다. 필요한 사항을 변경하기 전에 직무 ID SY_ANAL의 현재 급여 범위를 표시하는 query와 동일한 직무 ID의 사원 ID, 성 및 급여를 표시하는 query를 작성합니다. 갱신이 끝나면 지정된 직무 ID에 대한 JOBS 테이블의 변경 사항(있는 경우)을 query 합니다.
- 3) SY_ANAL 직무를 사용하여 새로운 최소 급여를 7000으로, 새로운 최대 급여를 18000으로 설정하고 결과를 설명합니다.

연습 해답: 1-1: 새 SQL Developer 데이터베이스 연결 생성

이 연습에서는 연결 정보를 사용하여 SQL Developer 를 시작하고 새 데이터베이스 연결을 생성합니다.

- 1) 강사가 알려준 유저 ID 및 암호(예: ora62)를 사용하여 SQL Developer 를 시작합니다.

바탕 화면에서 SQL Developer 아이콘을 누릅니다.



- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

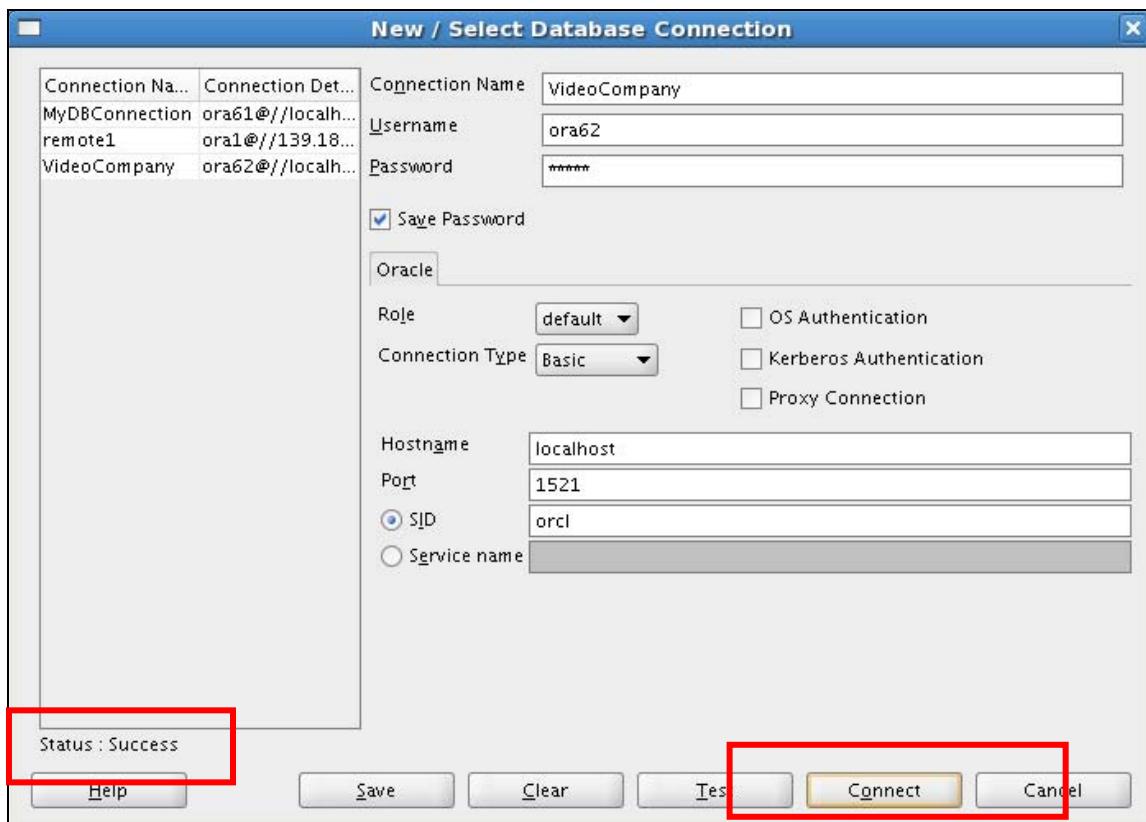
- a) Connection Name: VideoCompany
- b) Username: ora62
- c) Password: ora62
- d) Hostname: PC 의 호스트 이름을 입력합니다.
- e) Port: 1521
- f) SID: ORCL

Connections 템 페이지의 Connections 아이콘을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 New Connection 옵션을 선택합니다. New>Select Database Connection window 가 표시됩니다. 제공된 이전 정보를 사용하여 새 데이터베이스 연결을 생성합니다.

참고: 새로 생성된 연결의 속성을 표시하려면 연결 이름을 마우스 오른쪽 버튼으로 누르고 단축 메뉴에서 Properties 를 선택합니다. 유저 이름, 암호, 호스트 이름 및 서비스 이름을 강사가 제공한 적절한 정보로 대체합니다. 다음은 수강생 ora62 를 위해 새로 생성한 데이터베이스 연결의 예제입니다.

- 3) 새 연결을 테스트합니다. 상태가 Success 이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.
 - a) Connections 템 페이지에서 VideoCompany 아이콘을 두 번 누릅니다.
 - b) New>Select Database Connection window 의 Test 버튼을 누릅니다. 상태가 Success 이면 Connect 버튼을 누릅니다.

연습 해답: 1-1: SQL Developer Database 연결 생성 (계속)



연습 해답 1-2: JOBS 테이블에 새 직무 추가

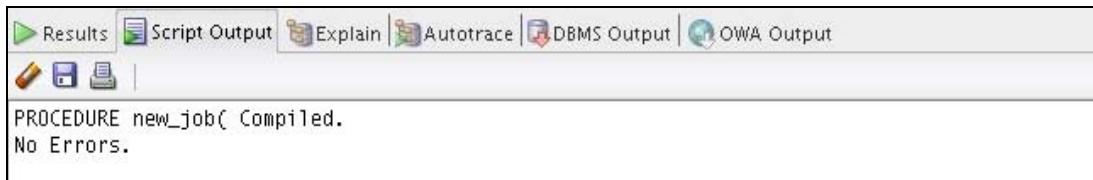
이 연습에서는 JOBS 테이블에 새로운 직무를 추가하는 서브 프로그램을 생성합니다.

- JOBS 테이블에 새 주문을 입력하는 NEW_JOB이라는 내장 프로시저를 생성합니다. 이 프로시저는 세 가지 파라미터를 사용해야 합니다. 첫번째 파라미터와 두번째 파라미터는 직무 ID 와 직위를 제공하고, 세번째 파라미터는 최소 급여를 제공합니다. 새로운 직무의 최대 급여를 직무 ID 에 대해 제공된 최소 급여의 두 배로 설정합니다.

```
/home/oracle/labs/plpu/solns/sol_ap_01_02_01.sql
```

스크립트를 엽니다. SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 눌러 프로시저를 생성하고 컴파일합니다. 연결을 선택하라는 메시지가 표시되면 새 VideoCompany 연결을 선택합니다. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PROCEDURE new_job(
    p_jobid    IN jobs.job_id%TYPE,
    p_title     IN jobs.job_title%TYPE,
    v_minsal   IN jobs.min_salary%TYPE) IS
    v_maxsal   jobs.max_salary%TYPE := 2 * v_minsal;
BEGIN
    INSERT INTO jobs(job_id, job_title, min_salary, max_salary)
    VALUES (p_jobid, p_title, v_minsal, v_maxsal);
    DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
    DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
                          v_minsal || ' ' || v_maxsal);
END new_job;
/
SHOW ERRORS
```



연습 해답 1-2: JOBS 테이블에 새 직무 추가 (계속)

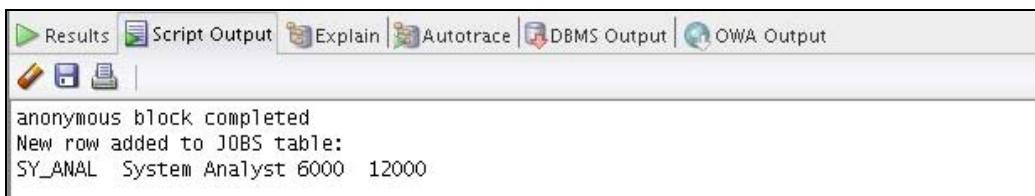
- 2) SERVEROUTPUT 을 활성화한 다음 직무 ID 가 'SY_ANAL'이고 직위가 'System Analyst'이고 최소 급여가 6000 인 새로운 직무를 추가하는 프로시저를 호출합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_02_02.sql`

스크립트를 실행하십시오. 연결을 선택하라는 메시지가 표시되면 새 VideoCompany 연결을 선택합니다. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SET SERVEROUTPUT ON

EXECUTE new_job ('SY_ANAL', 'System Analyst', 6000)
```



- 3) 행이 추가되었는지 확인하고 다음 연습에서 사용할 새 직무 ID를 기록해 둡니다. 변경 사항을 커밋합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_02_03.sql`

스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SELECT *
FROM   jobs
WHERE  job_id = 'SY_ANAL';
COMMIT;
```

연습 해답 1-2: JOBS 테이블에 새 직무 추가 (계속)

The screenshot shows two windows from Oracle SQL Developer.

The top window is titled "Select Connection" and displays a dropdown menu set to "VideoCompany". It includes buttons for "Help", "OK", and "Cancel", along with a "+" button for creating new connections.

The bottom window is titled "Results" and displays the output of a SQL query. The query results are as follows:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	6000	12000

Below the results, the message "1 rows selected" is displayed, followed by "COMMIT succeeded."

연습 해답 1-3: JOB_HISTORY 테이블에 새 행 추가

이 연습에서는 JOB_HISTORY 테이블에 기존 사원에 대한 새 행을 추가합니다.

- 1) JOB_HISTORY 테이블에 연습 1 b에서 생성한 새 직무 ID ('SY_ANAL')로 자신의 직무를 변경하는 사원에 대한 새 행을 추가하는 ADD_JOB_HIST라는 내장 프로시저를 생성합니다.
 - a) 이 프로시저는 두 가지 파라미터를 제공해야 합니다. 하나는 직무를 변경하는 사원 ID에 대한 파라미터이고, 다른 하나는 새로운 직무 ID에 대한 파라미터입니다.
 - b) EMPLOYEES 테이블에서 사원 ID를 읽고 JOB_HISTORY 테이블에 삽입합니다.
 - c) JOB_HISTORY 테이블의 이 행에 대해 이 사원의 입사일을 시작 날짜로, 오늘 날짜를 종료 날짜로 설정합니다.
 - d) EMPLOYEES 테이블에서 이 사원의 채용 날짜를 오늘 날짜로 변경합니다.
 - e) 이 사원의 직무 ID를 파라미터로 전달된 직무 ID ('SY_ANAL' 직무 ID 사용)로 갱신하고 최소 급여와 동일한 급여를 해당 직무 ID + 500으로 갱신합니다.

참고: 존재하지 않는 사원을 삽입하려는 시도를 처리하려면 예외 처리를 추가합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_03_01.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PROCEDURE add_job_hist(
    p_emp_id      IN employees.employee_id%TYPE,
    p_new_jobid   IN jobs.job_id%TYPE) IS
BEGIN
    INSERT INTO job_history
        SELECT employee_id, hire_date, SYSDATE, job_id,
               department_id
        FROM   employees
        WHERE  employee_id = p_emp_id;
    UPDATE employees
        SET    hire_date = SYSDATE,
              job_id = p_new_jobid,
              salary = (SELECT min_salary + 500
                         FROM   jobs
                         WHERE  job_id = p_new_jobid)
        WHERE employee_id = p_emp_id;
    DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id || '

```

연습 해답 1-3: JOB_HISTORY 테이블에 새 행 추가 (계속)

```

        ' details to the JOB_HISTORY
table');
DBMS_OUTPUT.PUT_LINE ('Updated current job of employee
' ||
p_emp_id|| ' to '|| p_new_jobid);

EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE_APPLICATION_ERROR (-20001, 'Employee does not
exist!');
END add_job_hist;
/
SHOW ERRORS

```



- 2) ADD_JOB_HIST 프로시저를 호출하기 전에 EMPLOYEES, JOBS 및 JOB_HISTORY 테이블에 대한 모든 트리거를 비활성화합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_03_02.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이
표시됩니다.

```

ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;
ALTER TABLE job_history DISABLE ALL TRIGGERS;

```



연습 해답 1-3: JOB_HISTORY 테이블에 새 행 추가 (계속)

```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| + | - | | | | |
ALTER TABLE employees succeeded.
ALTER TABLE jobs succeeded.
ALTER TABLE job_history succeeded.

```

- 3) SERVEROUTPUT 을 활성화한 다음 사원 ID 106 및 직무 ID 'SY_ANAL'을 파라미터로 사용하여 프로시저를 실행합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_03_03.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET SERVEROUTPUT ON
EXECUTE add_job_hist(106, 'SY_ANAL')

```



```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output
| + | - | | | | |
anonymous block completed
Added employee 106 details to the JOB_HISTORY table
Updated current job of employee 106 to SY_ANAL

```

- 4) JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 사원 106의 변경 사항을 확인하고 커밋합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_03_04.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SELECT * FROM job_history
WHERE employee_id = 106;

SELECT job_id, salary FROM employees
WHERE employee_id = 106;

COMMIT;

```

연습 해답 1-3: JOB_HISTORY 테이블에 새 행 추가 (계속)

```

Results Script Output Explain Autotrace DBMS Output OWA Output
EMPLOYEE_ID START_DATE END_DATE JOB_ID DEPARTMENT_ID
106 05-FEB-98 26-JUN-09 IT_PROG 60
1 rows selected
JOB_ID      SALARY
SY_ANAL     6500
1 rows selected
COMMIT succeeded.

```

- 5) EMPLOYEES, JOBS 및 JOB_HISTORY 테이블에 대한 트리거를 다시 활성화합니다.

`/home/oracle/labs/plpu/sols/sol_ap_01_03_05.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
ALTER TABLE job_history ENABLE ALL TRIGGERS;

```



```

Results Script Output Explain Autotrace DBMS Output OWA Output
EMPLOYEE_ID START_DATE END_DATE JOB_ID DEPARTMENT_ID
106 05-FEB-98 26-JUN-09 IT_PROG 60
1 rows selected
JOB_ID      SALARY
SY_ANAL     6500
1 rows selected
COMMIT succeeded.

```

연습 해답 1-4: 직무에 대한 최소 급여 및 최대 급여 갱신

이 연습에서는 JOBS 테이블에서 직무에 대한 최소 급여와 최대 급여를 갱신하는 프로그램을 생성합니다.

- 1) JOBS 테이블의 특정 직무 ID에 대해 최소 급여와 최대 급여를 갱신하는 UPD_JOBSAL이라는 내장 프로시저를 생성합니다. 이 프로시저는 직무 ID, 새로운 최소 급여, 새로운 최대 급여 세 가지 파라미터를 제공해야 합니다. JOBS 테이블의 직무 ID가 유효하지 않을 경우에 대한 예외 처리를 추가합니다. 제공된 최대 급여가 최소 급여보다 적을 경우 예외를 발생시키고 JOBS 테이블의 행이 잠겨 있을 경우 표시되는 메시지를 제공합니다.

힌트: 리소스 잠김/사용 중 오류 번호는 -54입니다.

/home/oracle/labs/plpu/sols/sol_ap_01_04_01.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE PROCEDURE upd_jobsal(
    p_jobid      IN jobs.job_id%type,
    p_new_minsal  IN jobs.min_salary%type,
    p_new_maxsal  IN jobs.max_salary%type) IS
    v_dummy          PLS_INTEGER;
    e_resource_busy  EXCEPTION;
    e_sal_error      EXCEPTION;
    PRAGMA           EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
    IF (p_new_maxsal < p_new_minsal) THEN
        RAISE e_sal_error;
    END IF;
    SELECT 1 INTO v_dummy
        FROM jobs
        WHERE job_id = p_jobid
        FOR UPDATE OF min_salary NOWAIT;
    UPDATE jobs
        SET min_salary =  p_new_minsal,
            max_salary =  p_new_maxsal
        WHERE job_id   = p_jobid;
EXCEPTION
    WHEN e_resource_busy THEN
        RAISE_APPLICATION_ERROR (-20001,
            'Job information is currently locked, try later.');
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');
    WHEN e_sal_error THEN
        RAISE_APPLICATION_ERROR(-20001,
            'Data error: Max salary should be more than min
salary');
END upd_jobsal;
/
SHOW ERRORS

```

연습 해답 1-4: 직무에 대한 최소 급여 및 최대 급여 갱신 (계속)



```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | |
PROCEDURE upd_jobsal( Compiled.
No Errors.

```

- 2) SERVEROUTPUT 을 활성화한 다음 직무 ID 'SY_ANAL', 최소 급여 7000, 최대 급여 140 을 사용하여 UPD_JOBSAL 프로시저를 실행합니다.
- 참고: 이 경우 예외 메시지가 나타나야 합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_04_02.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET SERVEROUTPUT ON
EXECUTE upd_jobsal('SY_ANAL', 7000, 140)

```



```

Results Script Output Explain Autotrace DBMS Output OWA Output
| | |
line 1: SQLPLUS Command Skipped: SET SEREVEROUTPUT ON
Error starting at line 3 in command:
EXECUTE upd_jobsal('SY_ANAL', 7000, 140)
Error report:
ORA-20001: Data error: Max salary should be more than min salary
ORA-06512: at "ORA62.UPD_JOBSAL", line 28
ORA-06512: at line 1

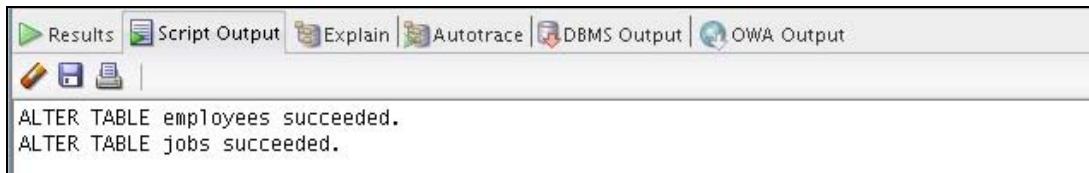
```

연습 해답 1-4: 직무에 대한 최소 급여 및 최대 급여 갱신 (계속)

- 3) EMPLOYEES 및 JOBS 테이블에 대한 트리거를 비활성화합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_04_03.sql
 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

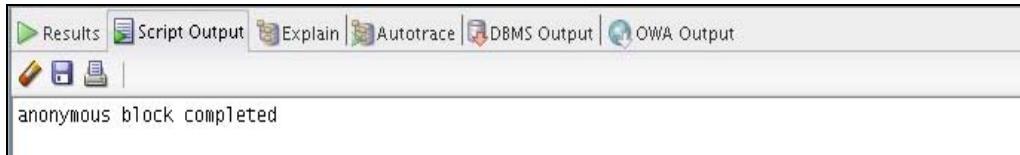
```
ALTER TABLE employees DISABLE ALL TRIGGERS;
ALTER TABLE jobs DISABLE ALL TRIGGERS;
```



- 4) 직무 ID 'SY_ANAL', 최소 급여 7000, 최대 급여 14000 을 사용하여 UPD_JOBSAL 프로시저를 실행합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_04_04.sql
 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
EXECUTE upd_jobsal('SY_ANAL', 7000, 14000)
```



연습 해답 1-4: 직무에 대한 최소 급여 및 최대 급여 갱신 (계속)

- 5) JOBS 테이블을 query하여 변경 사항을 확인한 다음 커밋합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_04_05.sql 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SELECT *
FROM   jobs
WHERE  job_id = 'SY_ANAL';
```



JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	7000	14000
1 rows selected			

- 6) EMPLOYEES 및 JOBS 테이블에 대한 트리거를 활성화합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_04_06.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
ALTER TABLE jobs ENABLE ALL TRIGGERS;
```



```
ALTER TABLE employees succeeded.
ALTER TABLE jobs succeeded.
```

연습 해답 1-5: 사원 급여 모니터

이 연습에서는 직무 유형에 따라 사원이 평균 급여를 초과했는지 여부를 모니터하는 프로시저를 생성합니다.

- SECURE_EMPLOYEES 트리거를 비활성화합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_05_01.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
ALTER TRIGGER secure_employees DISABLE;
```

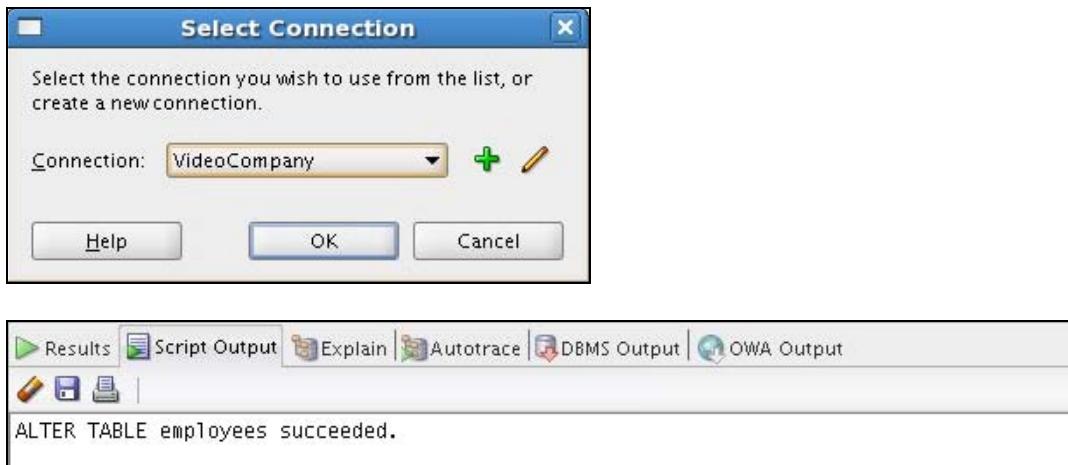


- EMPLOYEES 테이블에서 최대 세 자와 기본값 NO 를 저장할 EXCEED_AVGSAL 열을 추가합니다. YES 또는 NO 값을 허용하려면 CHECK 제약 조건을 사용합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_05_02.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
ALTER TABLE employees (
    ADD (exceed_avgsal VARCHAR2(3) DEFAULT 'NO'
        CONSTRAINT employees_exceed_avgsal_ck
        CHECK (exceed_avgsal IN ('YES', 'NO')));
```

연습 해답 1-5: 사원 급여 모니터 (계속)



- 3) 각 사원의 급여가 JOB_ID 의 평균 급여를 초과하는지 여부를 확인하는 CHECK_AVGSAL 이라는 내장 프로시저를 생성합니다.
- 직무에 대한 평균 급여는 JOBS 테이블의 정보를 사용하여 계산됩니다.
 - 사원의 급여가 해당 직무의 평균을 초과할 경우 EMPLOYEES 테이블의 EXCEED_AVGSAL 열 값을 YES 로 갱신하고, 그렇지 않은 경우에는 값을 NO 로 설정합니다.
 - 커서를 사용하여 query에서 FOR UPDATE 옵션을 통해 사원의 행을 선택합니다.
 - 레코드가 잠겨 있을 경우에 대한 예외 처리를 추가합니다.
힌트: 리소스 잠김/사용 중 오류 번호는 -54 입니다.
 - 지정된 직무 ID 의 평균 급여를 파라미터로 결정하는 GET_JOB_AVGSAL 이라는 로컬 함수를 생성하여 사용하십시오.

`/home/oracle/labs/plpu/solns/sol_ap_01_05_03.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PROCEDURE check_avgsal IS
    emp_exceed_avgsal_type employees.exceed_avgsal%type;
    CURSOR c_emp_csr IS
        SELECT employee_id, job_id, salary
        FROM employees
        FOR UPDATE;
    e_resource_busy EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_resource_busy, -54);
    FUNCTION get_job_avgsal (jobid VARCHAR2) RETURN NUMBER IS
        avg_sal employees.salary%type;
    BEGIN
        SELECT (max_salary + min_salary)/2 INTO avg_sal
        FROM jobs
```

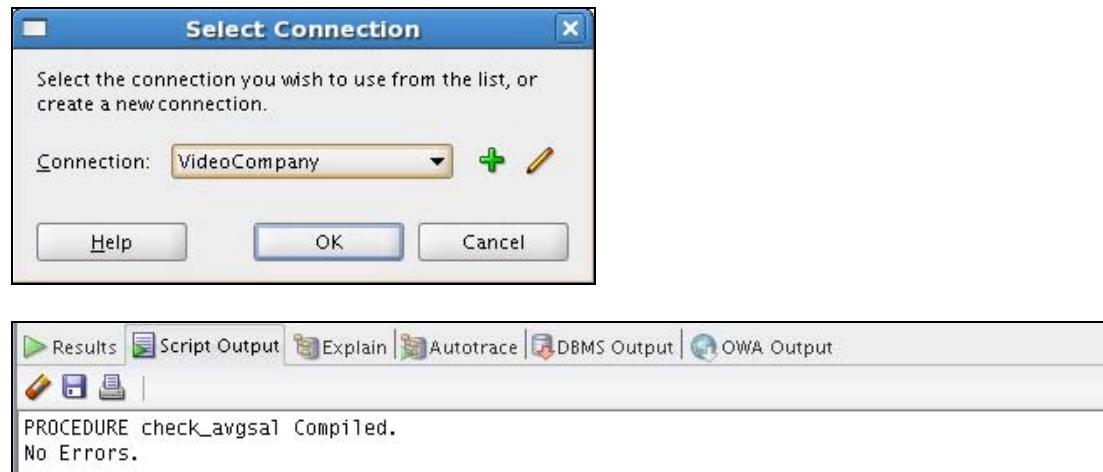
연습 해답 1-5: 사원 급여 모니터(계속)

```

    WHERE job_id = jobid;
    RETURN avg_sal;
END;

BEGIN
FOR emprec IN c_emp_csr
LOOP
    emp_exceed_avgsal_type := 'NO';
    IF emprec.salary >= get_job_avgsal(emprec.job_id) THEN
        emp_exceed_avgsal_type := 'YES';
    END IF;
    UPDATE employees
        SET exceed_avgsal = emp_exceed_avgsal_type
        WHERE CURRENT OF c_emp_csr;
END LOOP;
EXCEPTION
    WHEN e_resource_busy THEN
        ROLLBACK;
        RAISE_APPLICATION_ERROR (-20001, 'Record is busy, try
later.');
END check_avgsal;
/
SHOW ERRORS

```



- 4) CHECK_AVGSAL 프로시저를 실행합니다. 수정한 결과를 확인하려면 사원 ID, 직무, 해당 직무의 평균 급여, 사원 급여 및 급여가 해당 직무의 평균을 초과하는 사원의 exceed_avgsal 표시자 열을 표시하는 query를 작성합니다. 끝으로 변경 사항을 커밋합니다.

참고: 다음 연습은 함수 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

연습 해답 1-5: 사원 급여 모니터(계속)

/home/oracle/labs/plpu/solns/sol_ap_01_05_04.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이
표시됩니다.

```
EXECUTE check_avgsal

SELECT e.employee_id, e.job_id, (j.max_salary-
j.min_salary/2) job_avgsal,
       e.salary, e.exceed_avgsal avg_exceeded
FROM   employees e, jobs j
WHERE  e.job_id = j.job_id
and   e.exceed_avgsal = 'YES';

COMMIT;
```



연습 해답 1-5: 사원 급여 모니터(계속)

The screenshot shows the execution results of an anonymous block. The results are displayed in a table with the following columns: EMPLOYEE_ID, JOB_ID, JOB_AVGSAL, SALARY, and AVG_EXCEEDED. The table contains 30 rows of data, with the last row being a summary row for the SH_CLERK job.

anonymous block completed				
EMPLOYEE_ID	JOB_ID	JOB_AVGSAL	SALARY	AVG_EXCEEDED
103	IT_PROG	8000	9000	YES
109	FI_ACCOUNT	6900	9000	YES
110	FI_ACCOUNT	6900	8200	YES
111	FI_ACCOUNT	6900	7700	YES
112	FI_ACCOUNT	6900	7800	YES
113	FI_ACCOUNT	6900	6900	YES
120	ST_MAN	5750	8000	YES
121	ST_MAN	5750	8200	YES
122	ST_MAN	5750	7900	YES
137	ST_CLERK	4000	3600	YES
141	ST_CLERK	4000	3500	YES
150	SA_REP	9000	10000	YES
151	SA_REP	9000	9500	YES
152	SA_REP	9000	9000	YES
156	SA_REP	9000	10000	YES
157	SA_REP	9000	9500	YES
158	SA_REP	9000	9000	YES
162	SA_REP	9000	10500	YES
163	SA_REP	9000	9500	YES
168	SA_REP	9000	11500	YES
169	SA_REP	9000	10000	YES
170	SA_REP	9000	9600	YES
174	SA_REP	9000	11000	YES
184	SH_CLERK	4250	4200	YES
185	SH_CLERK	4250	4100	YES
192	SH_CLERK	4250	4000	YES
201	MK_MAN	10500	13000	YES
203	HR REP	7000	6500	YES
204	PR REP	8250	10000	YES
206	AC_ACCOUNT	6900	8300	YES

30 rows selected
COMMIT succeeded.

연습 해답 1-6: 사원의 총 근무 연수 검색

이 연습에서는 특정 사원의 근무 연수를 검색하는 서브 프로그램을 생성합니다.

- 특정 사원의 총 근무 연수를 검색하는 GET_YEARS_SERVICE라는 내장 함수를 생성합니다. 함수는 사원 ID를 파라미터로 사용하여 근무 연수를 반환해야 합니다. 사원 ID가 유효하지 않을 경우에 대한 오류 처리를 추가합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_06_01.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE FUNCTION get_years_service(
    p_emp.empid_type IN employees.employee_id%TYPE) RETURN
NUMBER IS
    CURSOR c_jobh_csr IS
        SELECT MONTHS_BETWEEN(end_date, start_date)/12
v_years_in_job
        FROM job_history
        WHERE employee_id = p_emp.empid_type;
    v_years_service NUMBER(2) := 0;
    v_years_in_job NUMBER(2) := 0;
BEGIN
    FOR jobh_rec IN c_jobh_csr
    LOOP
        EXIT WHEN c_jobh_csr%NOTFOUND;
        v_years_service := v_years_service +
jobh_rec.v_years_in_job;
    END LOOP;
    SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
v_years_in_job
        FROM employees
        WHERE employee_id = p_emp.empid_type;
    v_years_service := v_years_service + v_years_in_job;
    RETURN ROUND(v_years_service);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID '|| p_emp.empid_type || ' does not
exist.');
        RETURN NULL;
END get_years_service;
/
SHOW ERRORS
```

연습 해답 1-6: 사원의 총 근무 연수 검색(계속)

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains a dropdown menu labeled 'Connection: VideoCompany' and a toolbar with a green plus sign icon for creating new connections. Below the dialog are three buttons: 'Help', 'OK', and 'Cancel'. In the main workspace, there is a toolbar with several tabs: 'Results' (highlighted), 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Underneath the toolbar, the text 'FUNCTION get_years_service(Compiled.' and 'No Errors.' is displayed.

- 2) ID 가 999 인 사원에 대한 DBMS_OUTPUT.PUT_LINE 호출에서
GET_YEARS_SERVICE 함수를 호출합니다.

/home/oracle/labs/plpu/sols/sol_ap_01_06_02.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이
표시됩니다.

```
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service (999))
```

The screenshot shows the Oracle SQL Developer interface. At the top, there is a 'Select Connection' dialog box with the title 'Select Connection'. It contains a dropdown menu labeled 'Connection: VideoCompany' and a toolbar with a green plus sign icon for creating new connections. Below the dialog are three buttons: 'Help', 'OK', and 'Cancel'. In the main workspace, there is a toolbar with several tabs: 'Results' (highlighted), 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. Underneath the toolbar, the text 'Error starting at line 3 in command:
EXECUTE DBMS_OUTPUT.PUT_LINE(get_years_service (999))
Error report:
ORA-20348: Employee with ID 999 does not exist.
ORA-06512: at "ORA62.GET_YEARS_SERVICE", line 22
ORA-06512: at line 1' is displayed.

연습 해답 1-6: 사원의 총 근무 연수 검색(계속)

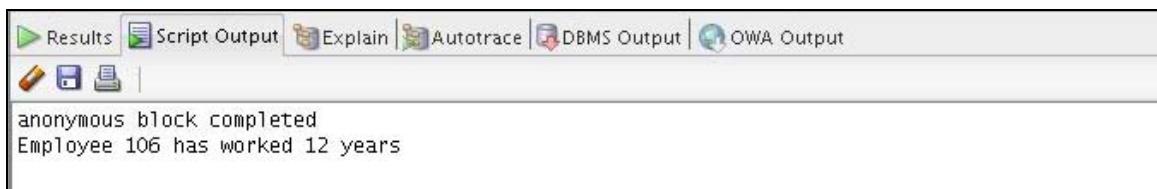
- 3) GET_YEARS_SERVICE 함수를 호출하는 DBMS_OUTPUT.PUT_LINE 을 사용하여 사원 106의 근무 연수를 표시합니다. SERVEROUTPUT 을 활성화해야 합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_06_03.sql`

스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SET SERVEROUTPUT ON

BEGIN
    DBMS_OUTPUT.PUT_LINE (
        'Employee 106 has worked ' || get_years_service(106) ||
        ' years');
END;
/
```



- 4) 지정된 사원의 JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 수정 사항이 정확한지 확인합니다. 이 페이지에 결과로 표시된 값과 실제 query 를 수행하여 얻은 결과는 다를 수 있습니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_06_04.sql`

스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SELECT employee_id, job_id,
       MONTHS_BETWEEN(end_date, start_date)/12 duration
  FROM job_history;

SELECT job_id, MONTHS_BETWEEN(SYSDATE, hire_date)/12 duration
  FROM employees
 WHERE employee_id = 106;
```

연습 해답 1-6: 사원의 총 근무 연수 검색(계속)



Screenshot of an Oracle Database SQL Developer interface showing the results of a query.

The query results are displayed in two sections:

EMPLOYEE_ID	JOB_ID	DURATION
102	IT_PROG	5.52956989247311827956989247311827956989
101	AC_ACCOUNT	4.09946236559139784946236559139784946237
101	AC_MGR	3.38172043010752688172043010752688172043
201	MK_REP	3.83870967741935483870967741935483870968
114	ST_CLERK	1.7688172043010752688172043010752688172
122	ST_CLERK	0.9973118279569892473118279569892473118283
200	AD_ASST	5.75
176	SA REP	0.7688172043010752688172043010752688172042
176	SA MAN	0.9973118279569892473118279569892473118283
200	AC_ACCOUNT	4.49731182795698924731182795698924731183
106	IT_PROG	11.53968678439864595778574273197929111908

11 rows selected

JOB_ID	DURATION
SY_ANAL	0

1 rows selected

연습 해답 1-7: 사원이 종사했던 다른 직무의 총 수 검색

이 연습에서는 한 사원이 재직 기간 동안에 수행했던 다른 직무 수를 검색하는 프로그램을 생성합니다.

- 1) 사원이 종사했던 다른 직무의 근무 연수를 검색하는 GET_JOB_COUNT 라는 내장 함수를 생성합니다.
 - a) 함수는 사원 ID를 파라미터로 사용하고 현재 직무를 비롯하여 사원이 지금까지 수행했던 다른 직무 수를 반환해야 합니다.
 - b) 사원 ID가 유효하지 않을 경우에 대한 예외 처리를 추가합니다.
힌트: JOB_HISTORY 테이블에서 고유한 직무 ID를 사용하되, 이 ID가 사원이 이미 종사했던 직무 ID 중 하나일 경우 현재 직무 ID를 제외시키십시오.
 - c) 두 query에 대한 UNION을 작성하여 PL/SQL 테이블로 가져온 행 수를 계산합니다.
 - d) FETCH와 BULK COLLECT INTO를 함께 사용하여 사원의 고유 직무를 확인합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_07_01.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE FUNCTION get_job_count(
    p_emp.empid_type IN employees.employee_id%TYPE) RETURN
NUMBER IS
    TYPE jobs_table_type IS TABLE OF jobs.job_id%type;
    v_jobtab jobs_table_type;
    CURSOR c_empjob_csr IS
        SELECT job_id
        FROM job_history
        WHERE employee_id = p_emp.empid_type
        UNION
        SELECT job_id
        FROM employees
        WHERE employee_id = p_emp.empid_type;
BEGIN
    OPEN c_empjob_csr;
    FETCH c_empjob_csr BULK COLLECT INTO v_jobtab;
    CLOSE c_empjob_csr;
    RETURN v_jobtab.count;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,

```

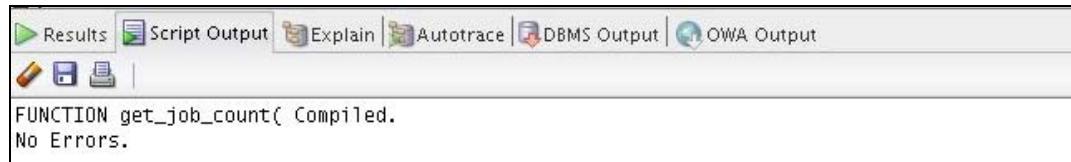
연습 해답 1-7: 사원이 종사했던 다른 직무의 총 수 검색(계속)

```

'Employee with ID ' || p_emp.empid_type || ' does not
exist! ');
      RETURN NULL;
END get_job_count;
/
SHOW ERRORS

FUNCTION get_job_count( Compiled.
No Errors.

```



- 2) ID 가 176 인 사원에 대해 함수를 호출합니다. SERVEROUTPUT 을 활성화해야 합니다.

참고: 다음 연습은 패키지 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

/home/oracle/labs/plpu/sols/sol_ap_01_07_02.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET SERVEROUTPUT ON

BEGIN
  DBMS_OUTPUT.PUT_LINE('Employee 176 worked on ' ||
    get_job_count(176) || ' different jobs.');
END;
/

```

연습 해답 1-7: 사원이 종사했던 다른 직무의 총 수 검색(계속)

The screenshot shows the Oracle SQL Developer interface. At the top, a 'Select Connection' dialog box is open, prompting the user to choose a connection from a list or create a new one. The selected connection is 'VideoCompany'. Below the dialog, the main SQL developer window is visible, featuring a toolbar with various icons and tabs for 'Results', 'Script Output', 'Explain', 'Autotrace', 'DBMS Output', and 'OWA Output'. The 'Results' tab is currently active. The output area displays the following text:
anonymous block completed
Employee 176 worked on 2 different jobs.

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성

이 연습에서는 NEW_JOB, ADD_JOB_HIST, UPD_JOBSAL 프로시저와 GET_YEARS_SERVICE 및 GET_JOB_COUNT 함수를 포함하는 EMPJOB_PKG라는 패키지를 생성합니다.

- 1) 서브 프로그램 생성자가 모두 공용(public)인 Package Spec 을 생성합니다. 서브 프로그램의 로컬 정의 유형을 Package Spec 으로 이동합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_08_01.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE empjob_pkg IS
    TYPE jobs_table_type IS TABLE OF jobs.job_id%TYPE;

    PROCEDURE add_job_hist(
        p_emp_id IN employees.employee_id%TYPE,
        p_new_jobid IN jobs.job_id%TYPE);

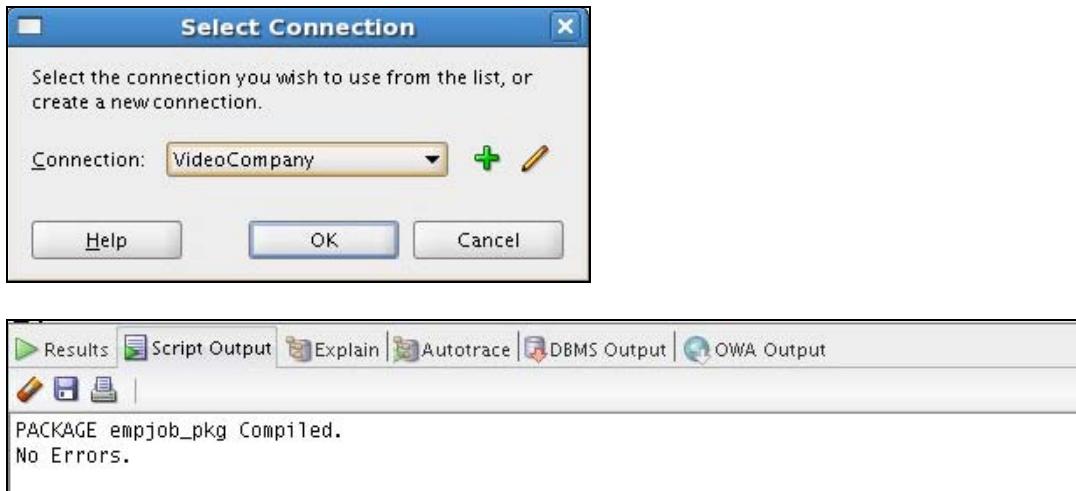
    FUNCTION get_job_count(
        p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

    FUNCTION get_years_service(
        p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER;

    PROCEDURE new_job(
        p_jobid IN jobs.job_id%TYPE,
        p_title IN jobs.job_title%TYPE,
        p_minsal IN jobs.min_salary%TYPE);

    PROCEDURE upd_jobsal(
        p_jobid IN jobs.job_id%type,
        p_new_minsal IN jobs.min_salary%type,
        p_new_maxsal IN jobs.max_salary%type);
END empjob_pkg;
/
SHOW ERRORS
```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)



- 2) 서브 프로그램을 구현하여 Package Body 를 생성하되, Package Spec 으로 이동한 유형은 서브 프로그램 구현에서 제거합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_08_02.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
CREATE OR REPLACE PACKAGE BODY empjob_pkg IS
  PROCEDURE add_job_hist(
    p_emp_id IN employees.employee_id%TYPE,
    p_new_jobid IN jobs.job_id%TYPE) IS
  BEGIN
    INSERT INTO job_history
      SELECT employee_id, hire_date, SYSDATE, job_id,
      department_id
      FROM employees
      WHERE employee_id = p_emp_id;
    UPDATE employees
      SET hire_date = SYSDATE,
          job_id = p_new_jobid,
          salary = (SELECT min_salary + 500
                     FROM jobs
                     WHERE job_id = p_new_jobid)
      WHERE employee_id = p_emp_id;
    DBMS_OUTPUT.PUT_LINE ('Added employee ' || p_emp_id ||
      ' details to the JOB_HISTORY table');
    DBMS_OUTPUT.PUT_LINE ('Updated current job of employee ' ||
      p_emp_id || ' to ' || p_new_jobid);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR (-20001, 'Employee does not
exist!');
  END;
```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)

```

END add_job_hist;

FUNCTION get_job_count(
    p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
    v_jobtab jobs_table_type;
    CURSOR c_empjob_csr IS
        SELECT job_id
        FROM job_history
        WHERE employee_id = p_emp_id
        UNION
        SELECT job_id
        FROM employees
        WHERE employee_id = p_emp_id;
BEGIN
    OPEN c_empjob_csr;
    FETCH c_empjob_csr BULK COLLECT INTO v_jobtab;
    CLOSE c_empjob_csr;
    RETURN v_jobtab.count;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID '|| p_emp_id || ' does not
exist!');
        RETURN 0;
END get_job_count;

FUNCTION get_years_service(
    p_emp_id IN employees.employee_id%TYPE) RETURN NUMBER IS
    CURSOR c_jobh_csr IS
        SELECT MONTHS_BETWEEN(end_date, start_date)/12
v_years_in_job
        FROM job_history
        WHERE employee_id = p_emp_id;
    v_years_service NUMBER(2) := 0;
    v_years_in_job NUMBER(2) := 0;
BEGIN
    FOR jobh_rec IN c_jobh_csr
    LOOP
        EXIT WHEN c_jobh_csr%NOTFOUND;
        v_years_service := v_years_service +
jobh_rec.v_years_in_job;
    END LOOP;
    SELECT MONTHS_BETWEEN(SYSDATE, hire_date)/12 INTO
v_years_in_job
        FROM employees
        WHERE employee_id = p_emp_id;
    v_years_service := v_years_service + v_years_in_job;
    RETURN ROUND(v_years_service);
END;

```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)

```

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR(-20348,
            'Employee with ID ' || p_emp_id || ' does not
exist.');
    RETURN 0;
END get_years_service;

PROCEDURE new_job(
    p_jobid IN jobs.job_id%TYPE,
    p_title IN jobs.job_title%TYPE,
    p_minsal IN jobs.min_salary%TYPE) IS
    v_maxsal jobs.max_salary%TYPE := 2 * p_minsal;
BEGIN
    INSERT INTO jobs(job_id, job_title, min_salary,
max_salary)
    VALUES (p_jobid, p_title, p_minsal, v_maxsal);
    DBMS_OUTPUT.PUT_LINE ('New row added to JOBS table:');
    DBMS_OUTPUT.PUT_LINE (p_jobid || ' ' || p_title || ' ' ||
p_minsal || ' ' || v_maxsal);
END new_job;

PROCEDURE upd_jobsal(
    p_jobid IN jobs.job_id%type,
    p_new_minsal IN jobs.min_salary%type,
    p_new_maxsal IN jobs.max_salary%type) IS
    v_dummy PLS_INTEGER;
    e_resource_busy EXCEPTION;
    e_sal_error EXCEPTION;
    PRAGMA EXCEPTION_INIT (e_resource_busy , -54);
BEGIN
    IF (p_new_maxsal < p_new_minsal) THEN
        RAISE e_sal_error;
    END IF;
    SELECT 1 INTO v_dummy
    FROM jobs
    WHERE job_id = p_jobid
    FOR UPDATE OF min_salary NOWAIT;
    UPDATE jobs
        SET min_salary = p_new_minsal,
            max_salary = p_new_maxsal
        WHERE job_id = p_jobid;
EXCEPTION
    WHEN e_resource_busy THEN
        RAISE_APPLICATION_ERROR (-20001,
            'Job information is currently locked, try later.');
    WHEN NO_DATA_FOUND THEN

```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)

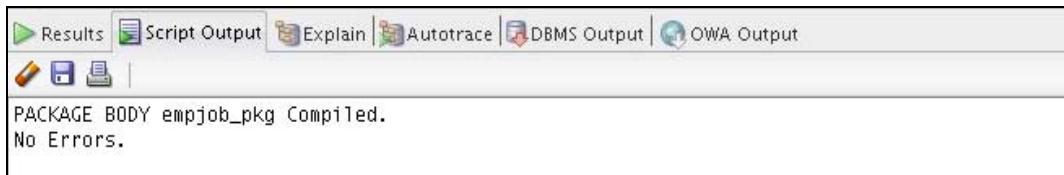
```

        RAISE_APPLICATION_ERROR(-20001, 'This job ID does not
exist');

        WHEN e_sal_error THEN
            RAISE_APPLICATION_ERROR(-20001,
                'Data error: Max salary should be more than min
salary');

        END upd_jobsal;
    END empjob_pkg;
/
SHOW ERRORS

```



- 3) EMPJOB_PKG.NEW_JOB 프로시저를 호출하여 ID 가 PR_MAN, 직위가 Public Relations Manager 이고, 급여가 6250 인 새로운 직무를 생성합니다.
SERVEROUTPUT 을 활성화해야 합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_08_03.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET SERVEROUTPUT ON

EXECUTE empjob_pkg.new_job('PR_MAN', 'Public Relations
Manager', 6250)

```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)



```

Results Script Output Explain Autotrace DBMS Output OWA Output
anonymous block completed
New row added to JOBS table:
PR_MAN Public Relations Manager 6250 12500

```

- 4) 사원 ID 110의 직무를 ID PR_MAN으로 수정하는

EMPJOB_PKG.ADD_JOB_HIST 프로시저를 호출합니다.

참고: UPDATE_JOB_HISTORY 트리거는 ADD_JOB_HIST 프로시저를 실행하기 전에 비활성화했다가 이 프로시저를 실행한 후에 다시 활성화해야 합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_08_04.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

ALTER TRIGGER update_job_history DISABLE;
EXECUTE empjob_pkg.add_job_hist(110, 'PR_MAN')
ALTER TRIGGER update_job_history ENABLE;

```



```

ALTER TRIGGER update_job_history succeeded.
anonymous block completed
Added employee 110 details to the JOB_HISTORY table
Updated current job of employee 110 to PR_MAN
ALTER TRIGGER update_job_history succeeded.

```

연습 해답 1-8: 새로 생성한 프로시저와 함수를 포함하는 새 패키지 생성 (계속)

- 5) JOBS, JOB_HISTORY 및 EMPLOYEES 테이블을 query 하여 결과를 확인합니다.

참고: 다음 연습은 데이터베이스 트리거 생성 방법을 설명할 때 추가 연습으로 사용할 수 있습니다.

/home/oracle/labs/plpu/solns/sol_ap_01_08_05.sql

스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SELECT * FROM jobs WHERE job_id = 'PR_MAN';
SELECT * FROM job_history WHERE employee_id = 110;
SELECT job_id, salary FROM employees WHERE employee_id = 110;
```



JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
PR_MAN	Public Relations Manager	6250	12500

1 rows selected

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
110	28-SEP-97	19-AUG-09	FI_ACCOUNT	100

1 rows selected

JOB_ID	SALARY
PR_MAN	6750

1 rows selected

연습 해답 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성

이 연습에서는 특정 직무의 최소 급여 및 최대 급여가 수정되지 않도록 하여 해당 직무 ID를 가진 기존 사원의 급여가 해당 직무에 대해 지정된 새 범위를 벗어나도록 하는 트리거를 생성합니다.

- 1) JOBS 테이블의 MIN_SALARY 및 MAX_SALARY 열에서 갱신되는 모든 행 앞에 실행되는 CHECK_SAL_RANGE라는 트리거를 생성합니다.
 - a) 최소 급여 값이나 최대 급여 값이 변경된 경우 EMPLOYEES 테이블에서 해당 직무 ID를 가진 기존 사원의 급여가 이 직무 ID에 대해 지정된 새 급여 범위 내에 있는지 검사합니다.
 - b) 기존 사원의 레코드에 영향을 미치는 급여 범위 변경을 다루는 예외 처리를 추가합니다.

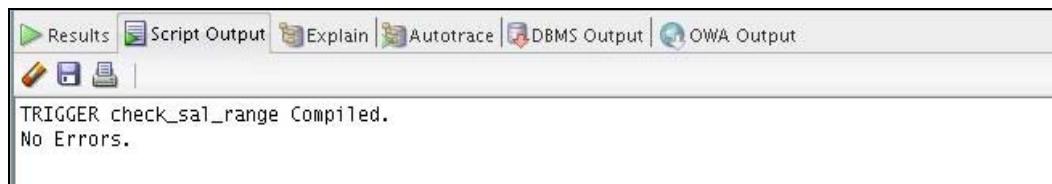
`/home/oracle/labs/plpu/solns/sol_ap_01_09_01.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE TRIGGER check_sal_range
BEFORE UPDATE OF min_salary, max_salary ON jobs
FOR EACH ROW
DECLARE
  v_minsal employees.salary%TYPE;
  v_maxsal employees.salary%TYPE;
  e_invalid_salrange EXCEPTION;
BEGIN
  SELECT MIN(salary), MAX(salary) INTO v_minsal, v_maxsal
  FROM employees
  WHERE job_id = :NEW.job_id;
  IF (v_minsal < :NEW.min_salary) OR (v_maxsal
  > :NEW.max_salary) THEN
    RAISE e_invalid_salrange;
  END IF;
EXCEPTION
  WHEN e_invalid_salrange THEN
    RAISE_APPLICATION_ERROR(-20550,
      'Employees exist whose salary is out of the
      specified range. ' ||
      'Therefore the specified salary range cannot be
      updated.');
END check_sal_range;
/
SHOW ERRORS

```

연습 해답 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성 (계속)



- 2) 새로운 최소 급여와 최대 급여를 5000 과 7000 으로 각각 설정하여 SY_ANAL 직무로 트리거를 테스트합니다. 필요한 사항을 변경하기 전에 직무 ID SY_ANAL 의 현재 급여 범위를 표시하는 query 와 동일한 직무 ID 의 사원 ID, 성 및 급여를 표시하는 query 를 작성합니다. 쟁신이 끝나면 지정된 직무 ID 에 대한 JOBS 테이블의 변경 사항(있는 경우)을 query 합니다.

/home/oracle/labs/plpu/solns/sol_ap_01_09_02.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';

SELECT employee_id, last_name, salary
FROM employees
WHERE job_id = 'SY_ANAL';

UPDATE jobs
SET min_salary = 5000, max_salary = 7000
WHERE job_id = 'SY_ANAL';

SELECT * FROM jobs
WHERE job_id = 'SY_ANAL';
```



연습 해답 1-9: 사원의 급여가 허용 범위 내에 있는지 확인하는 트리거 생성 (계속)

The screenshot shows three separate SQL queries:

```

    SELECT * FROM JOBS WHERE JOB_ID = 'SY_ANAL';
    SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID = 106;
    UPDATE JOBS SET MIN_SALARY = 5000, MAX_SALARY = 7000 WHERE JOB_ID = 'SY_ANAL';
  
```

Output from the first query:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	7000	14000

Output from the second query:

EMPLOYEE_ID	LAST_NAME	SALARY
106	Pataballa	6500

Output from the third query:

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
SY_ANAL	System Analyst	5000	7000

- 3) SY_ANAL 직무를 사용하여 새로운 최소 급여를 7000 으로, 새로운 최대 급여를 18000 으로 설정하고 결과를 설명합니다.

`/home/oracle/labs/plpu/solns/sol_ap_01_09_03.sql`
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
UPDATE jobs
  SET min_salary = 7000, max_salary = 18000
 WHERE job_id = 'SY_ANAL';
```



The output window shows the following error message:

```
Error starting at line 1 in command:
UPDATE jobs
  SET min_salary = 7000, max_salary = 18000
 WHERE job_id = 'SY_ANAL'
Error report:
SQL Error: ORA-20550: Employees exist whose salary is out of the specified range. Therefore the specified salary range cannot be updated.
ORA-06512: at "ORA62.CHECK_SAL_RANGE", line 14
ORA-04088: error during execution of trigger 'ORA62.CHECK_SAL_RANGE'
```

직무 ID 가 SY_ANAL 인 사원 106 의 급여가 6500 인데 이는 UPDATE 문에 지정된 새로운 급여 범위의 최소 급여보다 적기 때문에, CHECK_SAL_RANGE 트리거에서 제공하는 기능으로 인해 급여 범위를 변경할 수 없습니다.

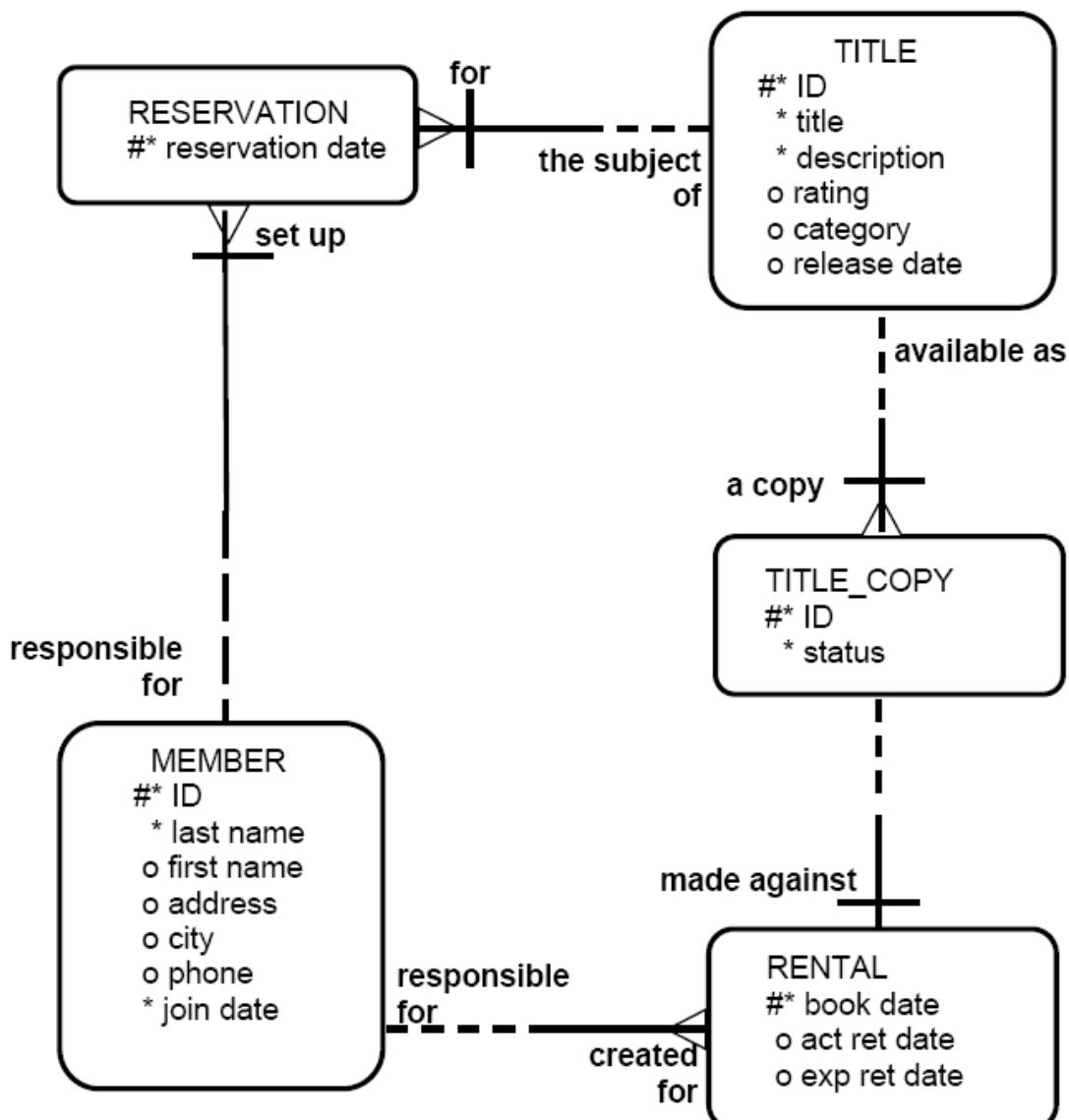
연습 2

이 사례 연구에서는 비디오 대여점 응용 프로그램에 대한 프로시저와 함수를 포함하는 VIDEO_PKG 라는 패키지를 생성합니다. 이 응용 프로그램을 사용하여 고객은 비디오 대여점의 회원이 될 수 있습니다. 회원이 되면 영화 대여, 대여한 영화 반납, 영화 예약 등이 가능합니다. 또한 비디오 테이블에 있는 데이터를 영업 시간에만 수정할 수 있도록 하는 트리거를 생성합니다.

SQL*Plus를 사용하여 패키지를 생성하고 오라클 제공 패키지 DBMS_OUTPUT을 사용하여 메시지를 표시합니다.

비디오 대여점 데이터베이스에는 TITLE, TITLE_COPY, RENTAL, RESERVATION, MEMBER 등의 테이블이 있습니다.

비디오 대여점 데이터베이스 엔티티 관계 도표



연습 2-1: VIDEO_PKG 패키지 생성

이 연습에서는 비디오 대여점 응용 프로그램에 대한 프로시저와 함수를 포함하는 VIDEO_PKG라는 패키지를 생성합니다.

- 1) /home/oracle/labs/plpu/labs/buildvid1.sql 스크립트를 로드한 다음 실행하여 이 연습에 필요한 필수 테이블과 시퀀스를 모두 생성합니다.
- 2) /home/oracle/labs/plpu/labs/buildvid2.sql 스크립트를 로드하고 실행하여 buildvid1.sql 스크립트로 생성한 모든 테이블을 채웁니다.
- 3) 다음 프로시저와 함수를 사용하여 VIDEO_PKG라는 패키지를 생성합니다.
 - a) **NEW_MEMBER:** MEMBER 테이블에 새로운 회원을 추가하는 공용(public) 프로시저입니다. 회원 ID 번호에는 MEMBER_ID_SEQ 시퀀스를 사용하고, 가입 날짜에는 SYSDATE 를 사용합니다. 새 행에 삽입될 다른 모든 값은 파라미터로 전달합니다.
 - b) **NEW_RENTAL:** 새로운 대여 내용을 기록하는 오버로드 공용(public) 함수입니다. 고객의 성이나 회원 ID 번호 중 하나와 고객이 대여하려고 하는 비디오의 제목 ID 번호를 함수에 전달합니다. 함수는 비디오 반납 기한을 반환해야 합니다. 반납 기한은 비디오를 대여한 날짜로부터 3일입니다. TITLE_COPY 테이블에 요청한 영화의 한 사본에 대한 상태가 AVAILABLE 로 표시될 경우 이 TITLE_COPY 테이블을 갱신한 다음 상태를 RENTED 로 설정합니다. 대여 가능한 비디오가 없을 경우 함수는 NULL 을 반환해야 합니다. 그런 다음 RENTAL 테이블에 오늘 날짜로 된 예약 날짜, 사본 ID 번호, 회원 ID 번호, 제목 ID 번호, 예상 반납 일자 등을 식별하는 새 레코드를 삽입합니다. 성이 같은 고객이 여러 명일 수 있다는 점을 유념합니다. 이 경우 함수가 NULL 을 반환하면서 일치하는 고객 이름과 ID 번호 리스트를 표시합니다.
 - c) **RETURN_MOVIE:** 비디오 상태(Available, Rented, Damaged)를 갱신하고 반납 일자를 설정하는 공용(public) 프로시저입니다. 이 프로시저에 제목 ID, 사본 ID 및 상태를 전달합니다. 해당 제목에 대해 예약된 사항이 있는지 확인한 다음 예약되어 있을 경우 메시지를 표시합니다. RENTAL 테이블을 갱신한 다음 실제 반납 일자를 오늘 날짜로 설정합니다. 프로시저에 전달된 상태 파라미터에 따라 TITLE_COPY 테이블의 상태를 갱신합니다.
 - d) **RESERVE_MOVIE:** NEW_RENTAL 프로시저에 요청된 모든 비디오 사본의 상태가 RENTED 일 경우에만 실행되는 전용(private) 프로시저입니다. 이 프로시저에 회원 ID 번호와 제목 ID 번호를 전달합니다. RESERVATION 테이블에 새 레코드를 삽입한 다음 예약 날짜, 회원 ID 번호 및 제목 ID 번호를 기록합니다. 영화가 예약된 상태임과 예상 반납 일자를 나타내는 메시지를 출력합니다.

연습 2-1: VIDEO_PKG 패키지 생성 (계속)

- e) **EXCEPTION_HANDLER:** 공용(public) 프로그램의 예외 처리기에서 호출되는 전용(private) 프로시저입니다. 이 프로시저에 SQLCODE 번호와 오류가 발생한 프로그램 이름(텍스트 문자열)을 전달합니다.
- RAISE_APPLICATION_ERROR 를 사용하여 유저가 정의한 오류를 발생시킵니다. Unique Key 위반(-1) 및 Foreign Key 위반(-2292)으로 시작합니다. 다른 오류에 대해서는 예외 처리기가 일반 오류를 발생시키도록 허용합니다.
- 4) /home/oracle/labs/plpu/soln 디렉토리에 있는 다음 스크립트를 사용하여 루틴을 테스트합니다.
- `sol_ap_02_01_04_a.sql` 을 사용하여 회원 두 명을 추가합니다.
 - `sol_ap_02_01_04_b.sql` 을 사용하여 새로운 비디오 대여 기록을 추가합니다.
 - `sol_ap_02_01_04_c.sql` 을 사용하여 영화를 반납합니다.
- 5) 비디오 대여점의 영업 시간은 일요일에서 금요일까지는 오전 8:00 - 오후 10:00이고 토요일은 오전 8:00 - 오후 12:00입니다. 이 시간 동안에만 테이블을 수정할 수 있도록 트리거에 의해 호출되는 내장 프로시저를 테이블에 생성합니다.
- 현재 시간이 영업 시간인지 확인하는 TIME_CHECK 라는 내장 프로시저를 생성합니다. 현재 시간이 영업 시간이 아닐 경우 RAISE_APPLICATION_ERROR 프로시저를 사용하여 해당 메시지를 제공합니다.
 - 다섯 개의 테이블 각각에 대한 트리거를 생성합니다. 데이터가 테이블에서 삽입, 갱신 및 삭제되기 전에 트리거를 실행합니다. 이러한 트리거 각각에서 TIME_CHECK 프로시저를 호출합니다.
 - 트리거를 테스트합니다.
- 참고:** 트리거가 실패하게 하려면 시간을 클래스의 현재 시간 범위 밖으로 변경해야 합니다. 예를 들면, 테스트하는 동안 트리거의 적합한 비디오 영업 시간을 오후 6:00에서 오전 8:00로 설정할 수 있습니다.

연습 해답 2-1: VIDEO_PKG 패키지 생성

이 연습에서는 비디오 대여점 응용 프로그램에 대한 프로시저와 함수를 포함하는 VIDEO_PKG라는 패키지를 생성합니다.

- 1) /home/oracle/labs/plpu/labs/buildvid1.sql 스크립트를 로드한 다음 실행하여 이 연습에 필요한 필수 테이블과 시퀀스를 모두 생성합니다.

/home/oracle/labs/plpu/labs/buildvid1.sql 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET ECHO OFF
/* Script to build the Video Application (Part 1 -
buildvid1.sql)
   for the Oracle Introduction to Oracle with Procedure
Builder course.
   Created by: Debby Kramer Creation date: 12/10/95
   Last updated: 2/13/96
   Modified by Nagavalli Pataballa on 26-APR-2001
   For the course Introduction to Oracle9i: PL/SQL
   This part of the script creates tables and sequences
   that are used
   by Part B of the Additional Practices of the course.
   Ignore the errors which appear due to dropping of table.
*/
DROP TABLE rental CASCADE CONSTRAINTS;
DROP TABLE reservation CASCADE CONSTRAINTS;
DROP TABLE title_copy CASCADE CONSTRAINTS;
DROP TABLE title CASCADE CONSTRAINTS;
DROP TABLE member CASCADE CONSTRAINTS;

PROMPT Please wait while tables are created.....

CREATE TABLE MEMBER
  (member_id  NUMBER (10)          CONSTRAINT member_id_pk
PRIMARY KEY
  , last_name  VARCHAR2(25)
    CONSTRAINT member_last_nn NOT NULL
  , first_name VARCHAR2(25)
  , address    VARCHAR2(100)
  , city       VARCHAR2(30)
  , phone      VARCHAR2(25)
  , join_date  DATE DEFAULT SYSDATE
    CONSTRAINT join_date_nn NOT NULL)
/
CREATE TABLE TITLE
  (title_id   NUMBER(10)
    CONSTRAINT title_id_pk PRIMARY KEY
  , title      VARCHAR2(60)
    CONSTRAINT title_nn NOT NULL
  , description VARCHAR2(400)
  
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

        CONSTRAINT title_desc_nn NOT NULL
    , rating      VARCHAR2(4)
        CONSTRAINT title_rating_ck CHECK (rating IN
    ('G','PG','R','NC17','NR'))
    , category    VARCHAR2(20) DEFAULT 'DRAMA'
        CONSTRAINT title_categ_ck CHECK (category IN
    ('DRAMA','COMEDY','ACTION','CHILD','SCIFI','DOCUMENTARY'))
    , release_date DATE
/
CREATE TABLE TITLE_COPY
    (copy_id      NUMBER(10)
    , title_id    NUMBER(10)
        CONSTRAINT copy_title_id_fk
            REFERENCES title(title_id)
    , status       VARCHAR2(15)
        CONSTRAINT copy_status_nn NOT NULL
        CONSTRAINT copy_status_ck CHECK (status IN
    ('AVAILABLE', 'DESTROYED',
        'RENTED', 'RESERVED'))
    , CONSTRAINT copy_title_id_pk PRIMARY KEY(copy_id,
title_id))
/
CREATE TABLE RENTAL
    (book_date    DATE DEFAULT SYSDATE
    , copy_id     NUMBER(10)
    , member_id   NUMBER(10)
        CONSTRAINT rental_mbr_id_fk REFERENCES member(member_id)
    , title_id    NUMBER(10)
    , act_ret_date DATE
    , exp_ret_date DATE DEFAULT SYSDATE+2
    , CONSTRAINT rental_copy_title_id_fk FOREIGN KEY (copy_id,
title_id)
            REFERENCES title_copy(copy_id,title_id)
    , CONSTRAINT rental_id_pk PRIMARY KEY(book_date, copy_id,
title_id, member_id))
/
CREATE TABLE RESERVATION
    (res_date     DATE
    , member_id   NUMBER(10)
    , title_id    NUMBER(10)
        CONSTRAINT res_id_pk PRIMARY KEY(res_date, member_id,
title_id))
/
PROMPT Tables created.
DROP SEQUENCE title_id_seq;
DROP SEQUENCE member_id_seq;

PROMPT Creating Sequences...
CREATE SEQUENCE member_id_seq
    START WITH 101
    NOCACHE

CREATE SEQUENCE title_id_seq
    START WITH 92

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```
NOCACHE  
/  
  
PROMPT Sequences created.  
  
PROMPT Run buildvid2.sql now to populate the above tables.
```



```
Results Script Output Explain Autotrace DBMS Output OWA Output  
|  
  
Error starting at line 12 in command:  
DROP TABLE rental CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
*Cause:  
*Action:  
  
Error starting at line 13 in command:  
DROP TABLE reservation CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
*Cause:  
*Action:  
  
Error starting at line 14 in command:  
DROP TABLE title_copy CASCADE CONSTRAINTS  
Error report:  
  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
*Cause:  
*Action:  
  
Error starting at line 15 in command:  
DROP TABLE title CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"  
*Cause:  
*Action:  
  
Error starting at line 16 in command:  
DROP TABLE member CASCADE CONSTRAINTS  
Error report:  
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - "table or view does not exist"
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```
*Cause:
*Action:
Please wait while tables are created....
CREATE TABLE succeeded.
Tables created.

Error starting at line 80 in command:
DROP SEQUENCE title_id_seq
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Error starting at line 81 in command:
DROP SEQUENCE member_id_seq
Error report:
Error report:
SQL Error: ORA-02289: sequence does not exist
02289. 00000 - "sequence does not exist"
*Cause: The specified sequence does not exist, or the user does
not have the required privilege to perform this operation.
*Action: Make sure the sequence name is correct, and that you have
the right to perform the desired operation on this sequence.

Creating Sequences...
CREATE SEQUENCE succeeded.
CREATE SEQUENCE succeeded.
Sequences created.
Run buildvid2.sql now to populate the above tables.
```

- 2) /home/oracle/labs/plpu/labs/buildvid2.sql 스크립트를 로드하고 실행하여 buildvid1.sql 스크립트로 생성한 모든 테이블을 채웁니다.

/home/oracle/labs/plpu/labs/buildvid2.sql 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
/* Script to build the Video Application (Part 2 -
buildvid2.sql)
   This part of the script populates the tables that are
   created using
      buildvid1.sql
   These are used by Part B of the Additional Practices of
   the course.
   You should run the script buildvid1.sql before running
   this script to
      create the above tables.
*/
INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL, 'Velasquez', 'Carmen',
           '283 King Street', 'Seattle', '587-99-6666', '03-MAR-90');
INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL, 'Ngao', 'LaDoris',
           '5 Modrany', 'Bratislava', '586-355-8882', '08-MAR-90');
INSERT INTO member
  VALUES  (member_id_seq.NEXTVAL, 'Nagayama', 'Midori',
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

'68 Via Centrale', 'Sao Paolo', '254-852-5764', '17-JUN-
91');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Quick-To-See', 'Mark',
'6921 King Way', 'Lagos', '63-559-777', '07-APR-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Ropeburn', 'Audry',
'86 Chu Street', 'Hong Kong', '41-559-87', '04-MAR-
90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Urguhart', 'Molly',
'3035 Laurier Blvd.', 'Quebec', '418-542-9988', '18-JAN-
91');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Menchu', 'Roberta',
'Boulevard de Waterloo 41', 'Brussels', '322-504-2228',
'14-MAY-90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Biri', 'Ben',
'398 High St.', 'Columbus', '614-455-9863', '07-APR-
90');
INSERT INTO member
VALUES (member_id_seq.NEXTVAL, 'Catchpole', 'Antoinette',
'88 Alfred St.', 'Brisbane', '616-399-1411', '09-FEB-
92');

COMMIT;
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Willie and Christmas Too',
'All of Willie''s friends made a Christmas list for
Santa, but Willie has yet to create his own wish list.',
'G', 'CHILD', '05-OCT-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Alien Again', 'Another
installment of science fiction history. Can the heroine save
the planet from the alien life form?', 'R', 'SCIFI',
'19-MAY-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'The Glob', 'A meteor
crashes near a small American town and unleashes carnivorous
goo in this classic.', 'NR', 'SCIFI', '12-AUG-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'My Day Off', 'With a little
luck and a lot of ingenuity, a teenager skips school for a
day in New York.', 'PG', 'COMEDY', '12-JUL-95');
INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Miracles on Ice', 'A six-
year-old has doubts about Santa Claus. But she discovers
that miracles really do exist.', 'PG', 'DRAMA', '12-SEP-
95');

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

INSERT INTO TITLE (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Soda Gang', 'After
discovering a cache of drugs, a young couple find
themselves pitted against a vicious gang.', 'NR', 'ACTION',
'01-JUN-95');
INSERT INTO title (title_id, title, description, rating,
category, release_date)
VALUES (TITLE_ID_SEQ.NEXTVAL, 'Interstellar Wars',
'Futuristic interstellar action movie. Can the rebels save
the humans from the evil Empire?', 'PG', 'SCIFI', '07-JUL-77');

COMMIT;

INSERT INTO title_copy VALUES (1,92, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,93, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,93, 'RENTED');
INSERT INTO title_copy VALUES (1,94, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (2,95, 'AVAILABLE');
INSERT INTO title_copy VALUES (3,95, 'RENTED');
INSERT INTO title_copy VALUES (1,96, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,97, 'AVAILABLE');
INSERT INTO title_copy VALUES (1,98, 'RENTED');
INSERT INTO title_copy VALUES (2,98, 'AVAILABLE');

COMMIT;
INSERT INTO reservation VALUES (sysdate-1, 101, 93);
INSERT INTO reservation VALUES (sysdate-2, 106, 102);

COMMIT;

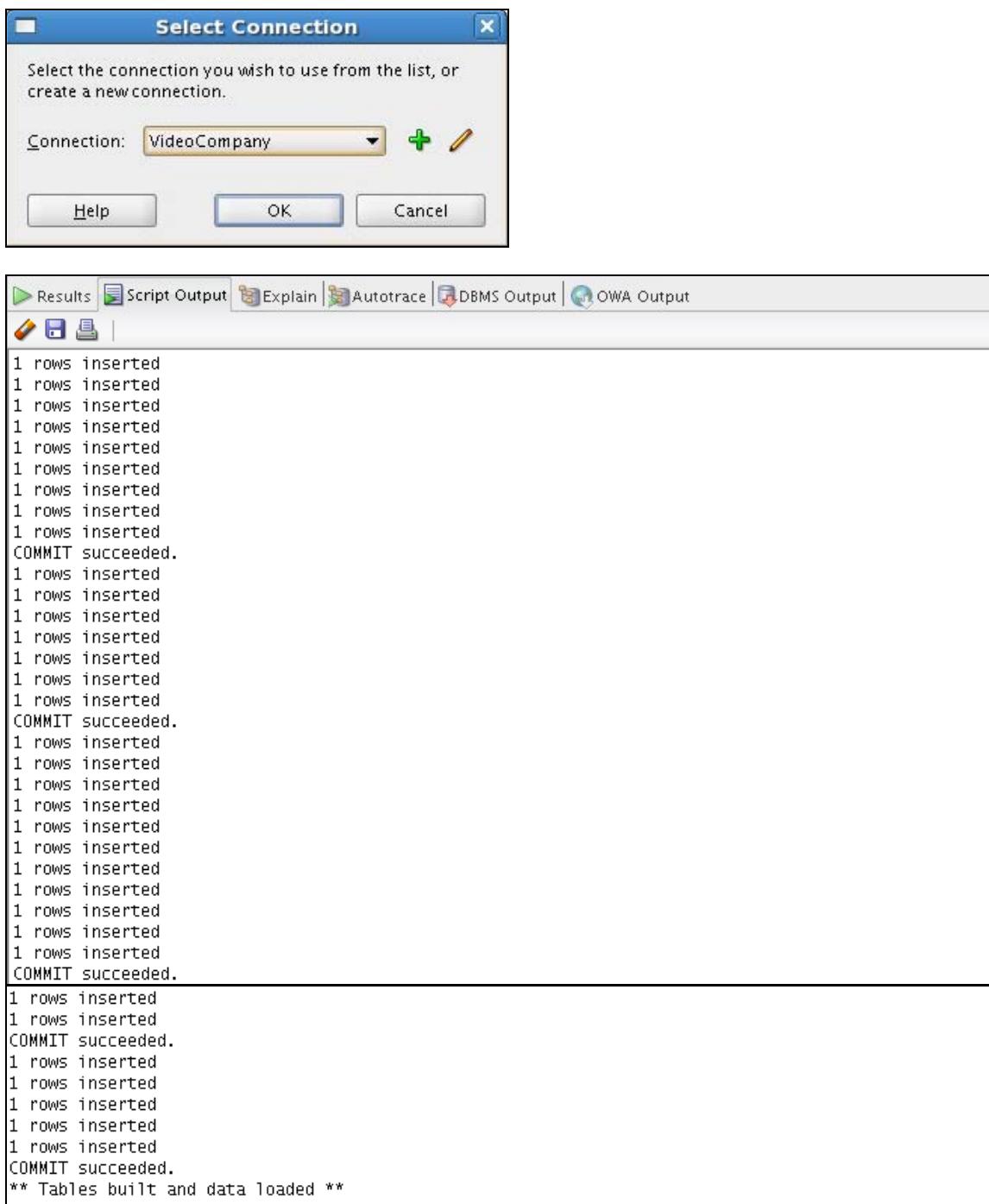
INSERT INTO rental VALUES (sysdate-1, 2, 101, 93, null,
sysdate+1);
INSERT INTO rental VALUES (sysdate-2, 3, 102, 95, null,
sysdate);
INSERT INTO rental VALUES (sysdate-3, 1, 101, 98, null,
sysdate-1);
INSERT INTO rental VALUES (sysdate-4, 1, 106, 97, sysdate-2,
sysdate-2);
INSERT INTO rental VALUES (sysdate-3, 1, 101, 92, sysdate-2,
sysdate-1);

COMMIT;

PROMPT ** Tables built and data loaded **

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)



연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

- 3) 다음 프로시저와 함수를 사용하여 VIDEO_PKG 라는 패키지를 생성합니다.
- NEW_MEMBER:** MEMBER 테이블에 새로운 회원을 추가하는 공용(public) 프로시저입니다. 회원 ID 번호에는 MEMBER_ID_SEQ 시퀀스를 사용하고, 가입 날짜에는 SYSDATE 를 사용합니다. 새 행에 삽입될 다른 모든 값은 파라미터로 전달합니다.
 - NEW_RENTAL:** 새로운 대여 내용을 기록하는 오버로드 공용(public) 함수입니다. 고객의 성이나 회원 ID 번호 중 하나와 고객이 대여하려고 하는 비디오의 제목 ID 번호를 함수에 전달합니다. 함수는 비디오 반납 기한을 반환해야 합니다. 반납 기한은 비디오를 대여한 날짜로부터 3 일입니다. TITLE_COPY 테이블에 요청한 영화의 한 사본에 대한 상태가 AVAILABLE 로 표시될 경우 이 TITLE_COPY 테이블을 갱신한 다음 상태를 RENTED 로 설정합니다. 대여 가능한 비디오가 없을 경우 함수는 NULL 을 반환해야 합니다. 그런 다음 RENTAL 테이블에 오늘 날짜로 된 예약 날짜, 사본 ID 번호, 회원 ID 번호, 제목 ID 번호, 예상 반납 일자 등을 식별하는 새 레코드를 삽입합니다. 성이 같은 고객이 여러 명일 수 있다는 점을 유념합니다. 이 경우 함수가 NULL 을 반환하면서 일치하는 고객 이름과 ID 번호 리스트를 표시합니다.
 - RETURN_MOVIE:** 비디오 상태(Available, Rented, Damaged)를 갱신하고 반납 일자를 설정하는 공용(public) 프로시저입니다. 이 프로시저에 제목 ID, 사본 ID 및 상태를 전달합니다. 해당 제목에 대해 예약된 사항이 있는지 확인한 다음 예약되어 있을 경우 메시지를 표시합니다. RENTAL 테이블을 갱신한 다음 실제 반납 일자를 오늘 날짜로 설정합니다. 프로시저에 전달된 상태 파라미터에 따라 TITLE_COPY 테이블의 상태를 갱신합니다.
 - RESERVE_MOVIE:** NEW_RENTAL 프로시저에 요청된 모든 비디오 사본의 상태가 RENTED 일 경우에만 실행되는 전용(private) 프로시저입니다. 이 프로시저에 회원 ID 번호와 제목 ID 번호를 전달합니다. RESERVATION 테이블에 새 레코드를 삽입한 다음 예약 날짜, 회원 ID 번호 및 제목 ID 번호를 기록합니다. 영화가 예약된 상태임과 예상 반납 일자를 나타내는 메시지를 출력합니다.
 - EXCEPTION_HANDLER:** 공용(public) 프로그램의 예외 처리기에서 호출되는 전용(private) 프로시저입니다. 이 프로시저에 SQLCODE 번호와 오류가 발생한 프로그램 이름(텍스트 문자열)을 전달합니다. RAISE_APPLICATION_ERROR 를 사용하여 유저가 정의한 오류를 발생시킵니다. Unique Key 위반(-1) 및 Foreign Key 위반(-2292)으로 시작합니다. 다른 오류에 대해서는 예외 처리기가 일반 오류를 발생시키도록 허용합니다.
- /home/oracle/labs/plpu/solns/sol_ap_02_01_03.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

VIDEO_PKG Package Spec

```

CREATE OR REPLACE PACKAGE video_pkg IS
  PROCEDURE new_member
    (p_lname      IN member.last_name%TYPE,
     p_fname      IN member.first_name%TYPE      DEFAULT NULL,
     p_address    IN member.address%TYPE         DEFAULT NULL,
     p_city       IN member.city%TYPE            DEFAULT NULL,
     p_phone      IN member.phone%TYPE          DEFAULT NULL
    );
  FUNCTION new_rental
    (p_memberid   IN rental.member_id%TYPE,
     p_titleid    IN rental.title_id%TYPE)
    RETURN DATE;
  FUNCTION new_rental
    (p_membername IN member.last_name%TYPE,
     p_titleid    IN rental.title_id%TYPE)
    RETURN DATE;
  PROCEDURE return_movie
    (p_titleid    IN rental.title_id%TYPE,
     p_copyid     IN rental.copy_id%TYPE,
     p_sts        IN title_copy.status%TYPE);
END video_pkg;
/
SHOW ERRORS

CREATE OR REPLACE PACKAGE BODY video_pkg IS
  PROCEDURE exception_handler(errcode IN NUMBER, context IN VARCHAR2) IS
  BEGIN
    IF errcode = -1 THEN
      RAISE_APPLICATION_ERROR(-20001,
        'The number is assigned to this member is already in use, ' ||
        'try again.');
    ELSIF errcode = -2291 THEN
      RAISE_APPLICATION_ERROR(-20002, context ||
        ' has attempted to use a foreign key value that is invalid');
    ELSE
      RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in ' ||
        context || '. Please contact your application ' ||
        'administrator with the following information: ' ||
        CHR(13) || SQLERRM);
    END IF;
  END exception_handler;
  PROCEDURE reserve_movie
    (memberid    IN reservation.member_id%TYPE,
     titleid     IN reservation.title_id%TYPE) IS
  
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

CURSOR rented_csr IS
    SELECT exp_ret_date
      FROM rental
     WHERE title_id = titleid
       AND act_ret_date IS NULL;
BEGIN
    INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, memberid, titleid);
    COMMIT;
    FOR rented_rec IN rented_csr LOOP
        DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back on: '
        || rented_rec.exp_ret_date);
        EXIT WHEN rented_csr%found;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'RESERVE_MOVIE');
END reserve_movie;

PROCEDURE return_movie(
    titleid IN rental.title_id%TYPE,
    copyid IN rental.copy_id%TYPE,
    sts IN title_copy.status%TYPE) IS
    v_dummy VARCHAR2(1);
    CURSOR res_csr IS
        SELECT *
          FROM reservation
         WHERE title_id = titleid;
BEGIN
    SELECT '' INTO v_dummy
      FROM title
     WHERE title_id = titleid;
    UPDATE rental
        SET act_ret_date = SYSDATE
      WHERE title_id = titleid
        AND copy_id = copyid AND act_ret_date IS NULL;
    UPDATE title_copy
        SET status = UPPER(sts)
      WHERE title_id = titleid AND copy_id = copyid;
    FOR res_rec IN res_csr LOOP
        IF res_csr%FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- ' ||
            'reserved by member #' || res_rec.member_id);
        END IF;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'RETURN_MOVIE');
END return_movie;
FUNCTION new_rental(
    memberid IN rental.member_id%TYPE,
    titleid IN rental.title_id%TYPE) RETURN DATE IS
    CURSOR copy_csr IS
        SELECT * FROM title_copy

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

        WHERE title_id = titleid
        FOR UPDATE;
    flag    BOOLEAN  := FALSE;
BEGIN

    FOR copy_rec IN copy_csr LOOP
        IF copy_rec.status = 'AVAILABLE' THEN
            UPDATE title_copy
                SET status = 'RENTED'
                WHERE CURRENT OF copy_csr;
            INSERT INTO rental(book_date, copy_id, member_id,
                               title_id, exp_ret_date)
            VALUES (SYSDATE, copy_rec.copy_id, memberid,
                    titleid, SYSDATE + 3);
            flag := TRUE;
            EXIT;
        END IF;
    END LOOP;
    COMMIT;
    IF flag THEN
        RETURN (SYSDATE + 3);
    ELSE
        reserve_movie(memberid, titleid);
        RETURN NULL;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_RENTAL');
END new_rental;

FUNCTION new_rental(
    membername IN member.last_name%TYPE,
    titleid    IN rental.title_id%TYPE) RETURN DATE IS
CURSOR copy_csr IS
    SELECT * FROM title_copy
        WHERE title_id = titleid
        FOR UPDATE;
flag    BOOLEAN  := FALSE;
memberid  member.member_id%TYPE;
CURSOR member_csr IS
    SELECT member_id, last_name, first_name
        FROM member
        WHERE LOWER(last_name) = LOWER(membername)
        ORDER BY last_name, first_name;
BEGIN
    SELECT member_id INTO memberid
        FROM member
        WHERE lower(last_name) = lower(membername);
    FOR copy_rec IN copy_csr LOOP
        IF copy_rec.status = 'AVAILABLE' THEN
            UPDATE title_copy
                SET status = 'RENTED'
                WHERE CURRENT OF copy_csr;
            INSERT INTO rental (book_date, copy_id, member_id,
                               title_id, exp_ret_date)

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

        VALUES (SYSDATE, copy_rec.copy_id, memberid,
                titleid, SYSDATE + 3);
        flag := TRUE;
        EXIT;
    END IF;
END LOOP;
COMMIT;
IF flag THEN
    RETURN(SYSDATE + 3);
ELSE
    reserve_movie(memberid, titleid);
    RETURN NULL;
END IF;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(
            'Warning! More than one member by this name.');
        FOR member_rec IN member_csr LOOP
            DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
                member_rec.last_name || ', ' ||
                member_rec.first_name);
        END LOOP;
        RETURN NULL;
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_RENTAL');
END new_rental;

PROCEDURE new_member(
    lname      IN member.last_name%TYPE,
    fname      IN member.first_name%TYPE      DEFAULT NULL,
    address    IN member.address%TYPE         DEFAULT NULL,
    city       IN member.city%TYPE           DEFAULT NULL,
    phone      IN member.phone%TYPE          DEFAULT NULL) IS
BEGIN
    INSERT INTO member(member_id, last_name, first_name,
                       address, city, phone, join_date)
        VALUES(member_id_seq.NEXTVAL, lname, fname,
               address, city, phone, SYSDATE);
    COMMIT;
CREATE OR REPLACE PACKAGE BODY video_pkg IS
    PROCEDURE exception_handler(errcode IN NUMBER, p_context
IN VARCHAR2) IS
    BEGIN
        IF errcode = -1 THEN
            RAISE_APPLICATION_ERROR(-20001,
                'The number is assigned to this member is already in
use, ' ||
                'try again.');
        ELSIF errcode = -2291 THEN
            RAISE_APPLICATION_ERROR(-20002, p_context ||
                ' has attempted to use a foreign key value that is
invalid');
        ELSE

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

RAISE_APPLICATION_ERROR(-20999, 'Unhandled error in '
|| p_context || '. Please contact your application ' ||
'administrator with the following information: ' ||
|| CHR(13) || SQLERRM);
END IF;
END exception_handler;

PROCEDURE reserve_movie
  (p_memberid  IN reservation.member_id%TYPE,
   p_titleid   IN reservation.title_id%TYPE) IS
CURSOR c_rented_csr IS
  SELECT exp_ret_date
    FROM rental
   WHERE title_id = p_titleid
     AND act_ret_date IS NULL;
BEGIN
  INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, p_memberid, p_titleid);
  COMMIT;
  FOR rented_rec IN c_rented_csr LOOP
    DBMS_OUTPUT.PUT_LINE('Movie reserved. Expected back
on: ' || rented_rec.exp_ret_date);
    EXIT WHEN c_rented_csr%found;
  END LOOP;
EXCEPTION
  WHEN OTHERS THEN
    exception_handler(SQLCODE, 'RESERVE_MOVIE');
END reserve_movie;

PROCEDURE return_movie(
  p_titleid IN rental.title_id%TYPE,
  p_copyid  IN rental.copy_id%TYPE,
  p_sts     IN title_copy.status%TYPE) IS
v_dummy VARCHAR2(1);
CURSOR c_res_csr IS
  SELECT *
    FROM reservation
   WHERE title_id = p_titleid;
BEGIN
  SELECT '' INTO v_dummy
    FROM title
   WHERE title_id = p_titleid;
  UPDATE rental
    SET act_ret_date = SYSDATE
      WHERE title_id = p_titleid
        AND copy_id = p_copyid AND act_ret_date IS NULL;
  UPDATE title_copy
    SET status = UPPER(p_sts)
      WHERE title_id = p_titleid AND copy_id = p_copyid;
  FOR res_rec IN c_res_csr LOOP
    IF c_res_csr%FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Put this movie on hold -- ' ||
      'reserved by member #' || res_rec.member_id);
    END IF;
  END LOOP;
END return_movie;

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

        END IF;
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'RETURN_MOVIE');
END return_movie;

FUNCTION new_rental(
    p_memberid  IN rental.member_id%TYPE,
    p_titleid   IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
    SELECT * FROM title_copy
    WHERE title_id = p_titleid
    FOR UPDATE;
    v_flag      BOOLEAN := FALSE;
BEGIN
    FOR copy_rec IN c_copy_csr LOOP
        IF copy_rec.status = 'AVAILABLE' THEN
            UPDATE title_copy
                SET status = 'RENTED'
                WHERE CURRENT OF c_copy_csr;
            INSERT INTO rental(book_date, copy_id, member_id,
                               title_id, exp_ret_date)
            VALUES (SYSDATE, copy_rec.copy_id, p_memberid,
                    p_titleid, SYSDATE + 3);
            v_flag := TRUE;
            EXIT;
        END IF;
    END LOOP;
    COMMIT;
    IF v_flag THEN
        RETURN (SYSDATE + 3);
    ELSE
        reserve_movie(p_memberid, p_titleid);
        RETURN NULL;
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_RENTAL');
        RETURN NULL;
END new_rental;

FUNCTION new_rental(
    p_membername IN member.last_name%TYPE,
    p_titleid   IN rental.title_id%TYPE) RETURN DATE IS
CURSOR c_copy_csr IS
    SELECT * FROM title_copy
    WHERE title_id = p_titleid
    FOR UPDATE;
    v_flag      BOOLEAN := FALSE;
    v_memberid  member.member_id%TYPE;
CURSOR c_member_csr IS
    SELECT member_id, last_name, first_name
    FROM member
    WHERE LOWER(last_name) = LOWER(p_membername)

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

        ORDER BY last_name, first_name;
BEGIN
    SELECT member_id INTO v_memberid
      FROM member
     WHERE lower(last_name) = lower(p_membername);
FOR copy_rec IN c_copy_csr LOOP
    IF copy_rec.status = 'AVAILABLE' THEN
        UPDATE title_copy
          SET status = 'RENTED'
         WHERE CURRENT OF c_copy_csr;
        INSERT INTO rental (book_date, copy_id, member_id,
                           title_id, exp_ret_date)
              VALUES (SYSDATE, copy_rec.copy_id, v_memberid,
                      p_titleid, SYSDATE + 3);
        v_flag := TRUE;
        EXIT;
    END IF;
END LOOP;
COMMIT;
IF v_flag THEN
    RETURN(SYSDATE + 3);
ELSE
    reserve_movie(v_memberid, p_titleid);
    RETURN NULL;
END IF;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(
            'Warning! More than one member by this name.');
        FOR member_rec IN c_member_csr LOOP
            DBMS_OUTPUT.PUT_LINE(member_rec.member_id || CHR(9) ||
                member_rec.last_name || ', ' ||
                member_rec.first_name);
        END LOOP;
        RETURN NULL;
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_RENTAL');
        RETURN NULL;
END new_rental;

PROCEDURE new_member(
    p_lname      IN member.last_name%TYPE,
    p_fname      IN member.first_name%TYPE      DEFAULT NULL,
    p_address    IN member.address%TYPE         DEFAULT NULL,
    p_city       IN member.city%TYPE            DEFAULT NULL,
    p_phone      IN member.phone%TYPE           DEFAULT NULL) IS
BEGIN
    INSERT INTO member(member_id, last_name, first_name,
                      address, city, phone, join_date)
        VALUES(member_id_seq.NEXTVAL, p_lname, p_fname,
               p_address, p_city, p_phone, SYSDATE);
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        exception_handler(SQLCODE, 'NEW_MEMBER');

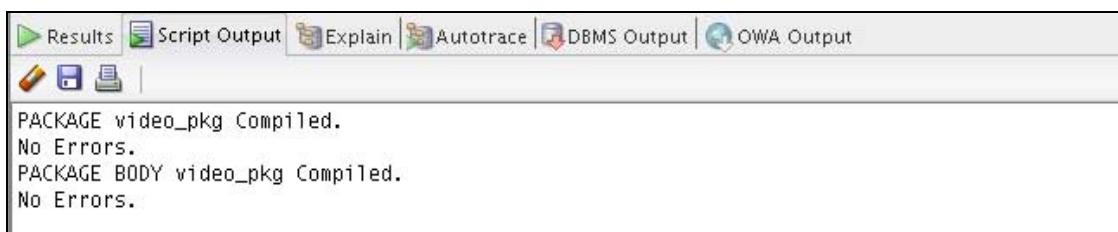
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

END new_member;
END video_pkg;
/
SHOW ERRORS

```



- 4) /home/oracle/labs/plpu/soln 디렉토리에 있는 다음 스크립트를 사용하여 루틴을 테스트합니다. SERVEROUTPUT 을 활성화해야 합니다.
- sol_ap_02_01_04_a.sql 을 사용하여 회원 두 명을 추가합니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_04_a.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```

SET SERVEROUTPUT ON
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut
Street', 'Boston', '617-123-4567')
EXECUTE video_pkg.new_member('Biri', 'Allan', 'Hiawatha
Drive', 'New York', '516-123-4567')

```



연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```
anonymous block completed
anonymous block completed
```

- b) sol_ap_02_01_04_b.sql 을 사용하여 새로운 비디오 대여 기록을 추가합니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_04_b.sql
스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(107, 98))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97))
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))
```



```
anonymous block completed
02-JUL-09

anonymous block completed
02-JUL-09

anonymous block completed
Movie reserved. Expected back on: 28-JUN-09

anonymous block completed
Warning! More than one member by this name.
111 Biri, Allan
108 Biri, Ben

Error starting at line 7 in command:
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))
Error report:
ORA-20002: NEW_RENTAL has attempted to use a foreign key value that is invalid
ORA-06512: at "ORA62.VIDEO_PKG", line 9
ORA-06512: at "ORA62.VIDEO_PKG", line 103
ORA-06512: at line 1
```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

- c) sol_ap_02_01_04_c.sql 스크립트를 사용하여 영화를 반납합니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_04_c.sql
 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

```
SET SERVEROUTPUT ON

EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE')
EXECUTE video_pkg.return_movie(95, 3, 'AVAILABLE')
EXECUTE video_pkg.return_movie(111, 1, 'RENTED')
```



```
anonymous block completed
Put this movie on hold -- reserved by member #107
anonymous block completed
Error starting at line 5 in command:
EXECUTE video_pkg.return_movie(111, 1, 'RENTED')
Error report:
ORA-20999: Unhandled error in RETURN_MOVIE. Please contact your application administrator with the following information:
ORA-01403: no data found
ORA-06512: at "ORA62.VIDEO_PKG", line 12
ORA-06512: at "ORA62.VIDEO_PKG", line 69
ORA-06512: at line 1
```

- 5) 비디오 대여점의 영업 시간은 일요일에서 금요일까지는 오전 8:00 - 오후 10:00이고 토요일은 오전 8:00 - 오후 12:00입니다. 이 시간 동안에만 테이블을 수정할 수 있도록 트리거에 의해 호출되는 내장 프로시저를 테이블에 생성합니다.

- a) 현재 시간이 영업 시간인지 확인하는 TIME_CHECK라는 내장 프로시저를 생성합니다. 현재 시간이 영업 시간이 아닐 경우 RAISE_APPLICATION_ERROR 프로시저를 사용하여 해당 메시지를 제공합니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_05_a.sql
 스크립트를 실행하십시오. 코드, 연결 프롬프트 및 결과가 다음과 같이 표시됩니다.

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```

CREATE OR REPLACE PROCEDURE time_check IS
BEGIN
    IF ((TO_CHAR(SYSDATE, 'D') BETWEEN 1 AND 6) AND
        (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT BETWEEN
         TO_DATE('08:00', 'hh24:mi') AND TO_DATE('22:00',
         'hh24:mi'))) OR ((TO_CHAR(SYSDATE, 'D') = 7) AND
        (TO_DATE(TO_CHAR(SYSDATE, 'hh24:mi'), 'hh24:mi') NOT BETWEEN
         TO_DATE('08:00', 'hh24:mi') AND TO_DATE('24:00',
         'hh24:mi')))) THEN
        RAISE_APPLICATION_ERROR(-20999,
        'Data changes restricted to office hours.');
    END IF;
END time_check;
/
SHOW ERRORS

PROCEDURE time_check Compiled.
No Errors.

```



- b) 다섯 개의 테이블 각각에 대한 트리거를 생성합니다. 데이터가 테이블에서 삽입, 갱신 및 삭제되기 전에 트리거를 실행합니다. 이러한 트리거 각각에서 TIME_CHECK 프로시저를 호출합니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_05_b.sql
스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.

```

CREATE OR REPLACE TRIGGER member_trig
    BEFORE INSERT OR UPDATE OR DELETE ON member
    CALL time_check
/

CREATE OR REPLACE TRIGGER rental_trig
    BEFORE INSERT OR UPDATE OR DELETE ON rental
    CALL time_check
/

```

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

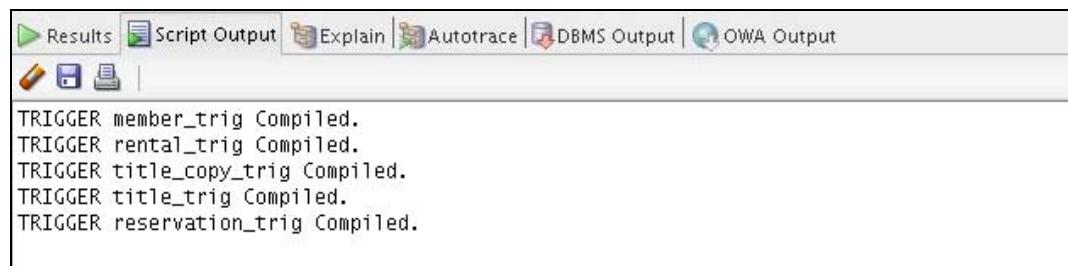
```

CREATE OR REPLACE TRIGGER title_copy_trig
  BEFORE INSERT OR UPDATE OR DELETE ON title_copy
CALL time_check
/

CREATE OR REPLACE TRIGGER title_trig
  BEFORE INSERT OR UPDATE OR DELETE ON title
CALL time_check
/
CREATE OR REPLACE TRIGGER reservation_trig
  BEFORE INSERT OR UPDATE OR DELETE ON reservation
CALL time_check
/

TRIGGER member_trig Compiled.
TRIGGER rental_trig Compiled.
TRIGGER title_copy_trig Compiled.
TRIGGER title_trig Compiled.
TRIGGER reservation_trig Compiled.

```



- c) 트리거를 테스트합니다.

참고: 트리거가 실패하게 하려면 시간을 클래스의 현재 시간 범위 밖으로 변경해야 합니다. 예를 들면, 테스트하는 동안 트리거의 적합한 비디오 영업 시간을 오후 6:00에서 오전 8:00로 설정할 수 있습니다.

/home/oracle/labs/plpu/solns/sol_ap_02_01_05_C.sql
스크립트를 실행하십시오. 코드 및 결과가 다음과 같이 표시됩니다.

연습 해답 2-1: VIDEO_PKG 패키지 생성 (계속)

```
-- First determine current timezone and time
SELECT SESSIONTIMEZONE,
       TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY HH24:MI')
  CURR_DATE
  FROM DUAL;

-- Change your time zone usinge [+|-]HH:MI format such
-- that -- the current time returns a time between 6pm and
8am

ALTER SESSION SET TIME_ZONE='-07:00';

-- Add a new member (for a sample test)

EXECUTE video_pkg.new_member('Elias', 'Elliane', 'Vine
Street', 'California', '789-123-4567')

BEGIN video_pkg.new_member('Elias', 'Elliane', 'Vine
Street', 'California', '789-123-4567'); END;

-- Restore the original time zone for your session.

ALTER SESSION SET TIME_ZONE='-00:00';
```

SESSIONTIMEZONE	CURR_DATE
Etc/GMT-7	29-JUN-2009 21:51

1 rows selected

ALTER SESSION SET succeeded.
anonymous block completed
ALTER SESSION SET succeeded.

B

테이블 설명

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

스키마 설명

전반적인 설명

오라클 데이터베이스 예제 스키마는 전세계에 사업체를 두고 여러 다양한 제품에 대한 주문을 처리하는 예제 회사를 보여줍니다. 이 회사에는 다음과 같은 3개의 부서가 있습니다.

- **Human Resources:** 사원 및 회사 시설에 대한 정보를 추적합니다.
- **Order Entry:** 다양한 통로를 통해 제품 재고 및 판매량을 추적합니다.
- **Sales History:** 용이한 업무 결정을 위해 업무 통계를 추적합니다.

이러한 부서는 각각 스키마로 표현됩니다. 본 과정에서 수강생들은 이러한 모든 스키마의 객체에 액세스할 수 있습니다. 그러나 예제, 데모 및 연습에서는 HR(Human Resources) 스키마를 중점적으로 다룹니다.

예제 스키마를 생성하는 데 필요한 모든 스크립트는 \$ORACLE_HOME/demo/schema/ 폴더에 있습니다.

HR(Human Resources)

이 과정에서 사용되는 스키마입니다. 회사의 HR(Human Resource) 레코드에는 각 사원의 식별 번호, 전자 메일 주소, 업무 식별 코드, 급여 및 관리자가 기록되어 있습니다. 급여 외에 커미션을 받는 사원도 있습니다.

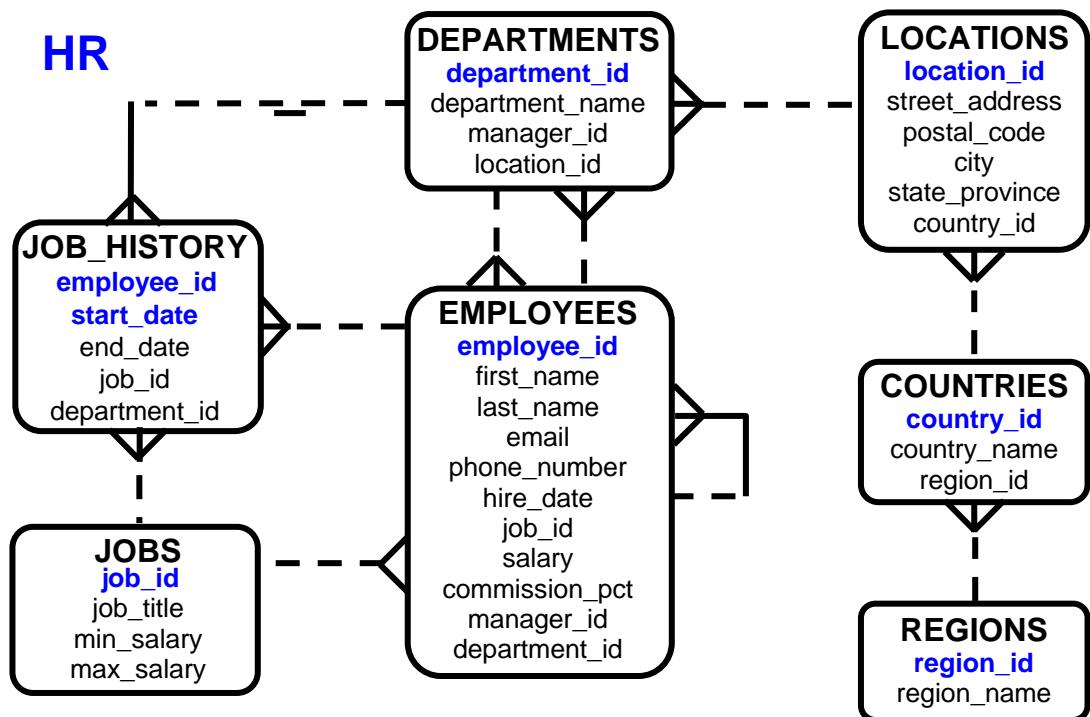
또한 조직 내에서 업무에 대한 정보를 추적합니다. 업무마다 해당 업무에 대한 식별 코드, 직위, 최소 및 최대 급여 범위가 있습니다. 일부 사원은 오랫동안 회사에 근무하며 사내에서 다양한 업무를 맡았습니다. 사원이 퇴직하면 해당 사원이 근무한 기간, 업무 식별 번호 및 근무 부서가 기록됩니다.

예제 회사는 여러 지역에 분포되어 있으므로 회사의 물류 창고 및 부서 위치를 추적합니다.

각 사원은 부서에 배정되고 각 부서는 고유한 부서 번호나 단축 이름으로 식별됩니다. 각 부서는 하나의 위치와 연관되고 각 위치는 번지, 우편 번호, 시, 도 및 국가 코드를 포함한 전체 주소를 가집니다.

회사는 부서 및 물류 창고가 위치한 장소에서 국가 이름, 통화 기호, 통화 이름을 비롯하여 해당 국가가 지리적으로 위치하고 있는 지역 등의 세부 사항을 기록합니다.

HR 엔티티 관계 다이어그램



HR (Human Resources) 테이블 설명

DESCRIBE countries

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT * FROM countries

COUNTRY_ID	COUNTRY_NAME	REGION_ID
1 AR	Argentina	2
2 AU	Australia	3
3 BE	Belgium	1
4 BR	Brazil	2
5 CA	Canada	2
6 CH	Switzerland	1
7 CN	China	3
8 DE	Germany	1
9 DK	Denmark	1
10 EG	Egypt	4
11 FR	France	1
12 HK	HongKong	3
13 IL	Israel	4
14 IN	India	3
15 IT	Italy	1
16 JP	Japan	3
17 KW	Kuwait	4
18 MX	Mexico	2
19 NG	Nigeria	4
20 NL	Netherlands	1
21 SG	Singapore	3
22 UK	United Kingdom	1
23 US	United States of ...	2
24 ZM	Zambia	4
25 ZW	Zimbabwe	4

HR (Human Resources) 테이블 설명(계속)

DESCRIBE departments

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

SELECT * FROM departments

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10 Administration	200	1700
2	20 Marketing	201	1800
3	30 Purchasing	114	1700
4	40 Human Resources	203	2400
5	50 Shipping	121	1500
6	60 IT	103	1400
7	70 Public Relations	204	2700
8	80 Sales	145	2500
9	90 Executive	100	1700
10	100 Finance	108	1700
11	110 Accounting	205	1700
12	120 Treasury	(null)	1700
13	130 Corporate Tax	(null)	1700
14	140 Control And Credit	(null)	1700
15	150 Shareholder Services	(null)	1700
16	160 Benefits	(null)	1700
17	170 Manufacturing	(null)	1700
18	180 Construction	(null)	1700
19	190 Contracting	(null)	1700
20	200 Operations	(null)	1700
21	210 IT Support	(null)	1700
22	220 NOC	(null)	1700
23	230 IT Helpdesk	(null)	1700
24	240 Government Sales	(null)	1700
25	250 Retail Sales	(null)	1700
26	260 Recruiting	(null)	1700
27	270 Payroll	(null)	1700
28	980 Education	(null)	2500
29	280 Training	(null)	2400

HR (Human Resources) 테이블 설명(계속)

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

SELECT * FROM employees

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100 Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90
1	100 Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90
3	102 Lex	De Haan	LDEHAN	515.123.4569	13-JAN-93	AD_VP	17000	(null)	100	90
4	103 Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	(null)	102	60
5	104 Bruce	Ernst	BBERNST	590.423.4568	21-MAY-91	IT_PROG	6000	(null)	103	60
6	105 David	Austin	DAUSTIN	590.423.4569	25-JUN-97	IT_PROG	4800	(null)	103	60
7	106 Valli	Pataballa	VPATABALLA	590.423.4560	05-FEB-98	IT_PROG	4800	(null)	103	60
8	107 Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	(null)	103	60
9	108 Nancy	Greenberg	NGRGRN	515.124.4569	17-AUG-94	FI_MGR	12000	(null)	101	100
10	109 Daniel	Faviet	DFAFVIE	515.124.4169	16-AUG-94	FI_ACCOUNT	9000	(null)	108	100
11	110 John	Chen	JCHEN	515.124.4269	28-SEP-97	FI_ACCOUNT	8200	(null)	108	100
12	111 Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-97	FI_ACCOUNT	7700	(null)	108	100
13	112 Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-98	FI_ACCOUNT	7800	(null)	108	100
14	113 Luis	Popp	LPOPP	515.124.4567	07-DEC-99	FI_ACCOUNT	6900	(null)	108	100
15	114 Den	Raphaely	DRAPEL	515.127.4561	07-DEC-94	PU_MAN	11000	(null)	100	30
16	115 Alexander	Khoo	AKHOO	515.127.4562	18-MAY-95	PU_CLERK	3100	(null)	114	30
17	116 Shelli	Baida	SBAIL	515.127.4563	24-DEC-97	PU_CLERK	2900	(null)	114	30
18	117 Sigal	Tobias	STOBIA	515.127.4564	24-JUL-97	PU_CLERK	2800	(null)	114	30
19	118 Guy	Himuro	GHIMURO	515.127.4565	15-NOV-98	PU_CLERK	2600	(null)	114	30
20	119 Karen	Colmenares	KCOLUMNA	515.127.4566	10-AUG-99	PU_CLERK	2500	(null)	114	30
21	120 Matthew	Weiss	MWEWISS	650.123.1234	18-JUL-96	ST_MAN	8000	(null)	100	50
22	121 Adam	Fripp	AFRIPPI	650.123.2234	10-APR-97	ST_MAN	8200	(null)	100	50
23	122 Payam	Kaufling	PKAUF	650.123.3234	01-MAY-95	ST_MAN	7900	(null)	100	50
24	123 Shanta	Vollman	SVOLL	650.123.4234	10-OCT-97	ST_MAN	6500	(null)	100	50
25	124 Kevin	Mourgos	KMOURGO	650.123.5234	16-NOV-99	ST_MAN	5800	(null)	100	50
26	125 Julia	Nayer	JNAYER	650.124.1214	16-JUL-97	ST_CLERK	3200	(null)	120	50
27	126 Irene	Mikkilineni	IMIKKIL	650.124.1224	28-SEP-98	ST_CLERK	2700	(null)	120	50
28	127 James	Landry	JLA	650.124.1334	14-JAN-99	ST_CLERK	2400	(null)	120	50
29	128 Steven	Markle	SMA	650.124.1434	08-MAR-00	ST_CLERK	2200	(null)	120	50
30	129 Laura	Bissot	LBIS	650.124.5234	20-AUG-97	ST_CLERK	3300	(null)	121	50
31	130 Mozhe	Atkinson	MAT	650.124.6234	30-OCT-97	ST_CLERK	2800	(null)	121	50
32	131 James	Marlow	JAM	650.124.7234	16-FEB-97	ST_CLERK	2500	(null)	121	50
33	132 TJ	Olson	TJOL	650.124.8234	10-APR-99	ST_CLERK	2100	(null)	121	50
34	133 Jason	Mallin	JMA	650.127.1934	14-JUN-96	ST_CLERK	3300	(null)	122	50
35	134 Michael	Rogers	MRO	650.127.1834	26-AUG-98	ST_CLERK	2900	(null)	122	50
36	135 Ki	Gee	KGEE	650.127.1734	12-DEC-99	ST_CLERK	2400	(null)	122	50
37	136 Hazel	Philtanker	PHIL	650.127.1634	06-FEB-00	ST_CLERK	2200	(null)	122	50
38	137 Renske	Ladwig	RLADWIG	650.121.1234	14-JUL-95	ST_CLERK	3600	(null)	123	50

HR (Human Resources) 테이블 설명(계속)

Employees (계속)

39	138 Stephen	Stiles	SSTI...	650.121.2034	26-OCT-97	ST_CLERK	3200	(null)	123	50
40	139 John	Seo	JSEO	650.121.2019	12-FEB-98	ST_CLERK	2700	(null)	123	50
41	140 Joshua	Patel	JPAT...	650.121.1834	06-APR-98	ST_CLERK	2500	(null)	123	50
42	141 Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500	(null)	124	50
43	142 Curtis	Davies	CDA...	650.121.2994	29-JAN-97	ST_CLERK	3100	(null)	124	50
44	143 Randall	Matos	RMA...	650.121.2874	15-MAR-98	ST_CLERK	2600	(null)	124	50
45	144 Peter	Vargas	PVA...	650.121.2004	09-JUL-98	ST_CLERK	2500	(null)	124	50
46	145 John	Russell	JRU...	011.44.1344.42...	01-OCT-96	SA_MAN	14000	0.4	100	80
47	146 Karen	Partners	KPA...	011.44.1344.46...	05-JAN-97	SA_MAN	13500	0.3	100	80
48	147 Alberto	Errazuriz	AER...	011.44.1344.42...	10-MAR-97	SA_MAN	12000	0.3	100	80
49	148 Gerald	Cambrault	GCA...	011.44.1344.61...	15-OCT-99	SA_MAN	11000	0.3	100	80
50	149 Eleni	Zlotkey	EZL...	011.44.1344.42...	29-JAN-00	SA_MAN	10500	0.2	100	80
51	150 Peter	Tucker	PTU...	011.44.1344.12...	30-JAN-97	SA REP	10000	0.3	145	80
52	151 David	Bernstein	DBE...	011.44.1344.34...	24-MAR-97	SA REP	9500	0.25	145	80
53	152 Peter	Hall	PHALL	011.44.1344.47...	20-AUG-97	SA REP	9000	0.25	145	80
54	153 Christopher	Olsen	COL...	011.44.1344.49...	30-MAR-98	SA REP	8000	0.2	145	80
55	154 Nanette	Cambrault	NCA...	011.44.1344.98...	09-DEC-98	SA REP	7500	0.2	145	80
56	155 Oliver	Tuvault	OTU...	011.44.1344.48...	23-NOV-99	SA REP	7000	0.15	145	80
57	156 Janette	King	JKING	011.44.1345.42...	30-JAN-96	SA REP	10000	0.35	146	80
58	157 Patrick	Sully	PSU...	011.44.1345.92...	04-MAR-96	SA REP	9500	0.35	146	80
59	158 Allan	McEwen	AMC...	011.44.1345.82...	01-AUG-96	SA REP	9000	0.35	146	80
60	159 Lindsey	Smith	LSMIL...	011.44.1345.72...	10-MAR-97	SA REP	8000	0.3	146	80
61	160 Louise	Doran	LDO...	011.44.1345.62...	15-DEC-97	SA REP	7500	0.3	146	80
62	161 Sarath	Sewall	SSE...	011.44.1345.52...	03-NOV-98	SA REP	7000	0.25	146	80
63	162 Clara	Vishney	CVIS...	011.44.1346.12...	11-NOV-97	SA REP	10500	0.25	147	80
64	163 Danielle	Greene	DGR...	011.44.1346.22...	19-MAR-99	SA REP	9500	0.15	147	80
65	164 Mattea	Marvins	MMA...	011.44.1346.32...	24-JAN-00	SA REP	7200	0.1	147	80
66	165 David	Lee	DLEE	011.44.1346.52...	23-FEB-00	SA REP	6800	0.1	147	80
67	166 Sundar	Ande	SAN...	011.44.1346.62...	24-MAR-00	SA REP	6400	0.1	147	80
68	167 Amit	Banda	ABA...	011.44.1346.72...	21-APR-00	SA REP	6200	0.1	147	80
69	168 Lisa	Ozer	LOZER	011.44.1343.92...	11-MAR-97	SA REP	11500	0.25	148	80
70	169 Harrison	Bloom	HBL...	011.44.1343.82...	23-MAR-98	SA REP	10000	0.2	148	80
71	170 Tayler	Fox	TFOX	011.44.1343.72...	24-JAN-98	SA REP	9600	0.2	148	80
72	171 William	Smith	WSM...	011.44.1343.62...	23-FEB-99	SA REP	7400	0.15	148	80
73	172 Elizabeth	Bates	EBA...	011.44.1343.52...	24-MAR-99	SA REP	7300	0.15	148	80
74	173 Sundita	Kumar	SKU...	011.44.1343.32...	21-APR-00	SA REP	6100	0.1	148	80

HR (Human Resources) 테이블 설명(계속)

Employees (계속)

75	174 Ellen	Abel	EABEL	011.44.1644.42...	11-MAY-96	SA_REP	11000	0.3	149	80
76	175 Alyssa	Hutton	AHUTT	011.44.1644.42...	19-MAR-97	SA_REP	8800	0.25	149	80
77	176 Jonathon	Taylor	JTA...	011.44.1644.42...	24-MAR-98	SA_REP	8600	0.2	149	80
78	177 Jack	Livingston	JLIVI...	011.44.1644.42...	23-APR-98	SA_REP	8400	0.2	149	80
79	178 Kimberely	Grant	KGR...	011.44.1644.42...	24-MAY-99	SA_REP	7000	0.15	149	(null)
80	179 Charles	Johnson	CJO...	011.44.1644.42...	04-JAN-00	SA_REP	6200	0.1	149	80
81	180 Winston	Taylor	WTAYL	650.507.9876	24-JAN-98	SH_CLERK	3200	(null)	120	50
82	181 Jean	Fleaur	JFLEA...	650.507.9877	23-FEB-98	SH_CLERK	3100	(null)	120	50
83	182 Martha	Sullivan	MSU...	650.507.9878	21-JUN-99	SH_CLERK	2500	(null)	120	50
84	183 Girard	Geoni	GGE...	650.507.9879	03-FEB-00	SH_CLERK	2800	(null)	120	50
85	184 Nandita	Sarchand	NSA...	650.509.1876	27-JAN-96	SH_CLERK	4200	(null)	121	50
86	185 Alexis	Bull	ABULL	650.509.2876	20-FEB-97	SH_CLERK	4100	(null)	121	50
87	186 Julia	Dellinger	JDEL...	650.509.3876	24-JUN-98	SH_CLERK	3400	(null)	121	50
88	187 Anthony	Cabrio	ACA...	650.509.4876	07-FEB-99	SH_CLERK	3000	(null)	121	50
89	188 Kelly	Chung	KCH...	650.505.1876	14-JUN-97	SH_CLERK	3800	(null)	122	50
90	189 Jennifer	Dilly	JDILLY	650.505.2876	13-AUG-97	SH_CLERK	3600	(null)	122	50
91	190 Timothy	Gates	TGA...	650.505.3876	11-JUL-98	SH_CLERK	2900	(null)	122	50
92	191 Randall	Perkins	RPE...	650.505.4876	19-DEC-99	SH_CLERK	2500	(null)	122	50
93	192 Sarah	Bell	SBELL	650.501.1876	04-FEB-96	SH_CLERK	4000	(null)	123	50
94	193 Britney	Everett	BEV...	650.501.2876	03-MAR-97	SH_CLERK	3900	(null)	123	50
95	194 Samuel	McCain	SMC...	650.501.3876	01-JUL-98	SH_CLERK	3200	(null)	123	50
96	195 Vance	Jones	VJO...	650.501.4876	17-MAR-99	SH_CLERK	2800	(null)	123	50
97	196 Alana	Walsh	AW...	650.507.9811	24-APR-98	SH_CLERK	3100	(null)	124	50
98	197 Kevin	Feeney	KFEE...	650.507.9822	23-MAY-98	SH_CLERK	3000	(null)	124	50
99	198 Donald	OConnell	DOC...	650.507.9833	21-JUN-99	SH_CLERK	2600	(null)	124	50
100	199 Douglas	Grant	DGR...	650.507.9844	13-JAN-00	SH_CLERK	2600	(null)	124	50
101	200 Jennifer	Whalen	JWHL...	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10
102	201 Michael	Hartstein	MHA...	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20
103	202 Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000	(null)	201	20
104	203 Susan	Mavris	SMA...	515.123.7777	07-JUN-94	HR_REP	6500	(null)	101	40
105	204 Hermann	Baer	HBA...	515.123.8888	07-JUN-94	PR_REP	10000	(null)	101	70
106	205 Shelley	Higgins	SHIG...	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110
107	206 William	Gietz	WGI...	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110

HR (Human Resources) 테이블 설명(계속)

```
DESCRIBE job_history
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

```
SELECT * FROM job_history
```

EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102 13-JAN-93	24-JUL-98	IT_PROG	60
2	101 21-SEP-89	27-OCT-93	AC_ACCOUNT	110
3	101 28-OCT-93	15-MAR-97	AC_MGR	110
4	201 17-FEB-96	19-DEC-99	MK_REP	20
5	114 24-MAR-98	31-DEC-99	ST_CLERK	50
6	122 01-JAN-99	31-DEC-99	ST_CLERK	50
7	200 17-SEP-87	17-JUN-93	AD_ASST	90
8	176 24-MAR-98	31-DEC-98	SA REP	80
9	176 01-JAN-99	31-DEC-99	SA MAN	80
10	200 01-JUL-94	31-DEC-98	AC_ACCOUNT	90

HR (Human Resources) 테이블 설명(계속)

DESCRIBE jobs

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT * FROM jobs

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1 AD_PRES	President	20000	40000
2 AD_VP	Administration Vice President	15000	30000
3 AD_ASST	Administration Assistant	3000	6000
4 FI_MGR	Finance Manager	8200	16000
5 FI_ACCOUNT	Accountant	4200	9000
6 AC_MGR	Accounting Manager	8200	16000
7 AC_ACCOUNT	Public Accountant	4200	9000
8 SA_MAN	Sales Manager	10000	20000
9 SA_REP	Sales Representative	6000	12000
10 PU_MAN	Purchasing Manager	8000	15000
11 PU_CLERK	Purchasing Clerk	2500	5500
12 ST_MAN	Stock Manager	5500	8500
13 ST_CLERK	Stock Clerk	2000	5000
14 SH_CLERK	Shipping Clerk	2500	5500
15 IT_PROG	Programmer	4000	10000
16 MK_MAN	Marketing Manager	9000	15000
17 MK_REP	Marketing Representative	4000	9000
18 HR_REP	Human Resources Representative	4000	9000
19 PR_REP	Public Relations Representative	4500	10500

HR (Human Resources) 테이블 설명(계속)

DESCRIBE locations

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

SELECT * FROM locations

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1000 1297 Via Cola di Rie	00989	Roma	(null)	IT
2	1100 93091 Calle della Testa	10934	Venice	(null)	IT
3	1200 2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
4	1300 9450 Kamiya-cho	6823	Hiroshima	(null)	JP
5	1400 2014 Jabberwocky Rd	26192	Southlake	Texas	US
6	1500 2011 Interiors Blvd	99236	South San Francisco	California	US
7	1600 2007 Zagora St	50090	South Brunswick	New Jersey	US
8	1700 2004 Charade Rd	98199	Seattle	Washington	US
9	1800 147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
10	1900 6092 Boxwood St	YSW9T2	Whitehorse	Yukon	CA
11	2000 40-5-12 Laogianggen	190518	Beijing	(null)	CN
12	2100 1298 Vileparle (E)	490231	Bombay	Maharashtra	IN
13	2200 12-98 Victoria Street	2901	Sydney	New South Wales	AU
14	2300 198 Clementi North	540198	Singapore	(null)	SG
15	2400 8204 Arthur St	(null)	London	(null)	UK
16	2500 Magdalene Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK
17	2600 9702 Chester Road	09629850293	Stretford	Manchester	UK
18	2700 Schwanthalerstr. 7031	80925	Munich	Bavaria	DE
19	2800 Rua Frei Caneca 1360	01307-002	Sao Paulo	Sao Paulo	BR
20	2900 20 Rue des Corps-Saints	1730	Geneva	Geneve	CH
21	3000 Murtenstrasse 921	3095	Bern	BE	CH
22	3100 Pieter Breughelstraat 837	3029SK	Utrecht	Utrecht	NL
23	3200 Mariano Escobedo 9991	11932	Mexico City	Distrito Federal,	MX

HR (Human Resources) 테이블 설명(계속)

```
DESCRIBE regions
```

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM locations
```

REGION_ID	REGION_NAME
1	1 Europe
2	2 Americas
3	3 Asia
4	4 Middle East and Africa



SQL Developer 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- Oracle SQL Developer의 주요 기능 숙지
- Oracle SQL Developer의 메뉴 항목 식별
- 데이터베이스 연결 생성
- 데이터베이스 객체 관리
- SQL Worksheet 사용
- SQL 스크립트 저장 및 실행
- 보고서 생성 및 저장

ORACLE®

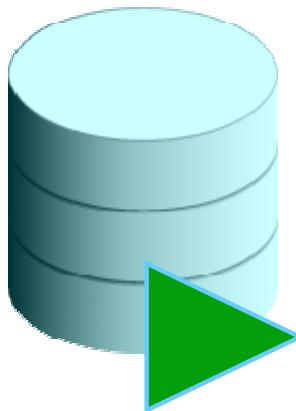
Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 SQL Developer라는 그래픽 도구에 대해 알아봅니다. 먼저, 데이터베이스 개발 작업에 SQL Developer를 사용하는 방법에 대해 알아봅니다. SQL Worksheet를 사용하여 SQL 문과 SQL 스크립트를 실행하는 방법에 대해 설명합니다.

Oracle SQL Developer란?

- Oracle SQL Developer는 생산성을 향상하고 데이터베이스 개발 작업을 단순화하는 그래픽 도구입니다.
- 표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다.



SQL Developer

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Oracle SQL Developer란?

Oracle SQL Developer는 생산성을 높이고 일상적인 데이터베이스 작업의 개발을 간소화하기 위해 무료로 제공되는 그래픽 도구입니다. 마우스를 몇 번 누르는 것만으로 간단하게 내장 프로시저를 생성 및 디버그하고, 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.

SQL Developer 1.2 버전은 유저가 단일 위치에서 third-party 데이터베이스의 데이터 및 데이터베이스 객체를 탐색하고 오라클에서 이러한 데이터베이스를 이전할 수 있도록 하는 *Developer Migration Workbench*와 긴밀하게 통합됩니다. 또한 MySQL, Microsoft SQL Server 및 Microsoft Access와 같은 일부 third-party 데이터베이스의 스키마에 연결할 수 있으며 이러한 데이터베이스의 메타 데이터와 데이터를 볼 수도 있습니다.

뿐만 아니라 SQL Developer에서는 Oracle Application Express 3.0.1(Oracle APEX)도 지원합니다.

SQL Developer 사양

- Oracle Database 11g Release 2와 함께 제공
- Java로 개발됨
- Windows, Linux 및 Mac OS X 플랫폼 지원
- JDBC(Java Database Connectivity) Thin 드라이버를 사용한 기본 연결
- Oracle Database 9.2.0.1 이상 버전에 연결
- 다음 링크에서 무료 다운로드 가능
 - http://www.oracle.com/technology/products/database/sql_developer/index.html

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 사양

Oracle SQL Developer 1.5는 Oracle Database 11g Release 2와 함께 제공됩니다. SQL Developer는 Oracle JDeveloper 통합 개발 환경(IDE)을 활용하여 Java로 개발되었습니다. 교차 플랫폼 도구로, 이 도구는 Windows, Linux 및 Mac OS(운영 체제) X 플랫폼에서 실행됩니다.

데이터베이스에 대한 기본 연결은 JDBC Thin 드라이버를 통해 이루어지므로 Oracle 품이 필요 없습니다. SQL Developer에는 Installer가 필요 없으며 다운로드한 파일의 압축을 풀기만 하면 됩니다. SQL Developer 유저는 Oracle Databases 9.2.0.1 이상 버전 및 Express Edition을 포함한 모든 Oracle Database Edition에 연결할 수 있습니다.

참고

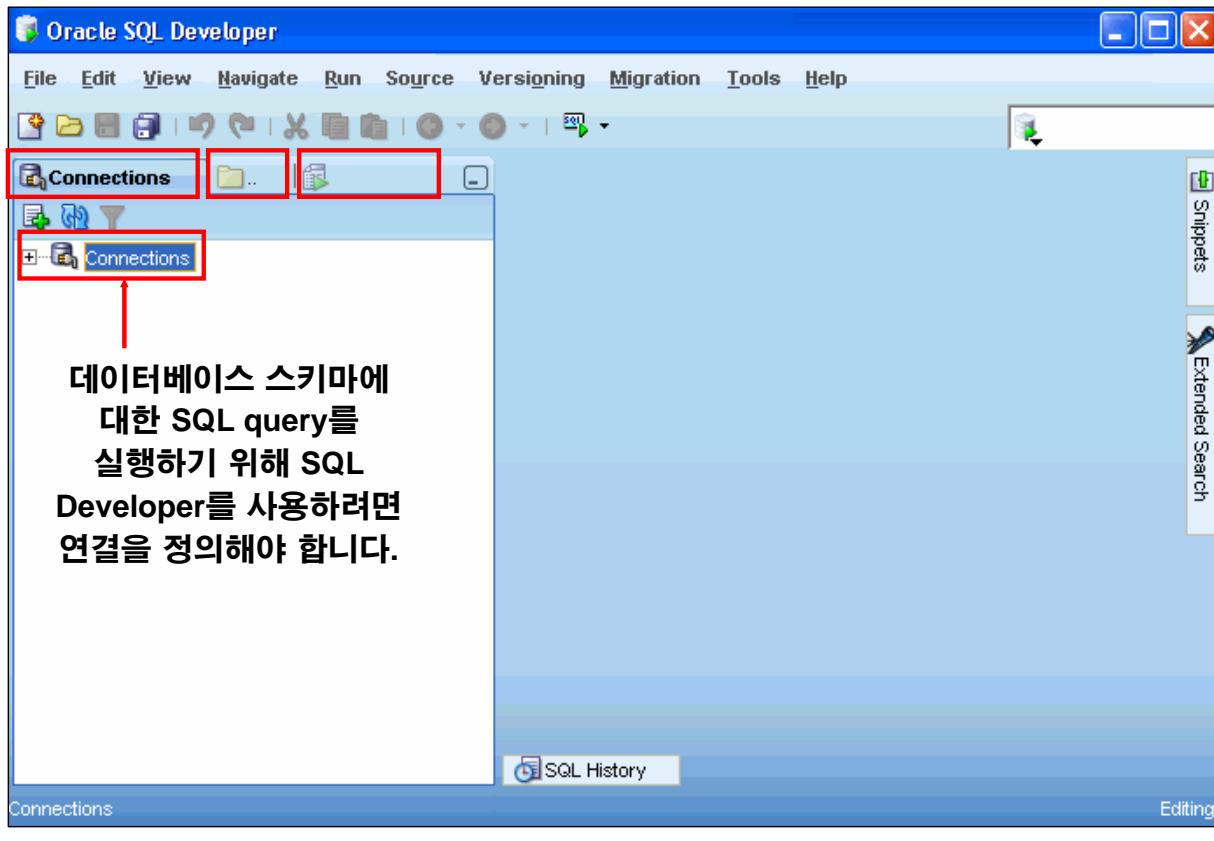
Oracle Database 11g Release 2 이전의 Oracle Database 버전을 사용하는 경우 SQL Developer를 다운로드하여 설치해야 합니다. SQL Developer 1.5는 다음 링크에서 무료로 다운로드할 수 있습니다.

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

SQL Developer를 설치하는 방법을 보려면 다음 링크를 방문하십시오.

http://download.oracle.com/docs/cd/E12151_01/index.htm

SQL Developer 1.5 인터페이스



데이터베이스 스키마에
대한 SQL query를
실행하기 위해 SQL
Developer를 사용하려면
연결을 정의해야 합니다.

SQL Developer 1.5 인터페이스

SQL Developer 1.5에는 다음과 같은 세 개의 기본 탐색 템(왼쪽부터)이 있습니다.

- **Connections 템:** 이 템을 사용하면 액세스할 수 있는 데이터베이스 객체 및 유저를 찾아볼 수 있습니다.
- **Files 템:** Files 폴더 아이콘으로 식별됩니다. 이 템을 사용하면 File > Open 메뉴를 사용하지 않고도 로컬 시스템에서 파일에 액세스할 수 있습니다.
- **Reports 템:** Reports 아이콘으로 식별됩니다. 이 템을 사용하면 미리 정의된 보고서를 실행하거나 사용자 보고서를 작성하고 추가할 수 있습니다.

일반 탐색 및 사용

SQL Developer는 왼쪽 탐색 영역에서 객체를 찾아 선택하면 오른쪽 탐색 영역에 선택한 객체에 대한 정보가 표시됩니다. 환경 설정을 통해 SQL Developer의 다양한 모양과 동작을 커스터마이즈할 수 있습니다.

참고: 데이터베이스 스키마에 연결하여 SQL Query를 실행하거나 프로시저/함수를 실행하려면 연결을 하나 이상 정의해야 합니다.

SQL Developer 1.5 인터페이스(계속)

메뉴

다음 메뉴에는 표준 항목과 SQL Developer 특정 기능에 대한 항목이 있습니다.

- **View:** SQL Developer 인터페이스에 표시되는 사항에 영향을 주는 옵션이 있습니다.
- **Navigate:** 다양한 창 이동 및 서브 프로그램 실행을 위한 옵션이 있습니다.
- **Run:** 함수나 프로시저가 선택될 때 연관되는 Run File 및 Execution Profile 옵션과 디버깅 옵션이 있습니다.
- **Source:** 함수와 프로시저를 편집할 때 사용하는 옵션이 있습니다.
- **Versioning:** CVS(Concurrent Versions System) 및 Subversion과 같은 버전 관리 및 소스 제어 시스템을 통합 지원합니다.
- **Migration:** Third-party 데이터베이스를 오라클로 이전할 때 관련되는 옵션이 있습니다.
- **Tools:** SQL*Plus, Preferences 및 SQL Worksheet 등의 SQL Developer 도구를 호출합니다.

참고: Run 메뉴에는 디버깅용으로 함수나 프로시저가 선택될 때 연관되는 옵션이 있습니다.

이러한 옵션은 1.2 버전의 Debug 메뉴에 있는 옵션과 동일합니다.

데이터베이스 연결 생성

- SQL Developer를 사용하려면 하나 이상의 데이터베이스 연결이 있어야 합니다.
- 다음과 같은 여러 대상에 대해 연결을 생성하고 테스트할 수 있습니다.
 - 데이터베이스
 - 스키마
- SQL Developer는 시스템의 `tnsnames.ora` 파일에 정의된 모든 연결을 자동으로 임포트합니다.
- XML(Extensible Markup Language) 파일로 연결을 эксп포트할 수 있습니다.
- 추가로 생성된 각각의 데이터베이스 연결은 Connections Navigator 계층에 나열됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 연결 생성

연결은 특정 데이터베이스에 해당 데이터베이스의 특정 유저로 연결하는 데 필요한 정보를 지정하는 SQL Developer 객체입니다. SQL Developer를 사용하려면 하나 이상의 데이터베이스 연결이 있어야 합니다. 기존 연결을 사용하거나 연결을 새로 생성 또는 임포트할 수도 있습니다. 여러 데이터베이스 및 여러 스키마에 대해 연결을 생성하고 테스트할 수 있습니다.

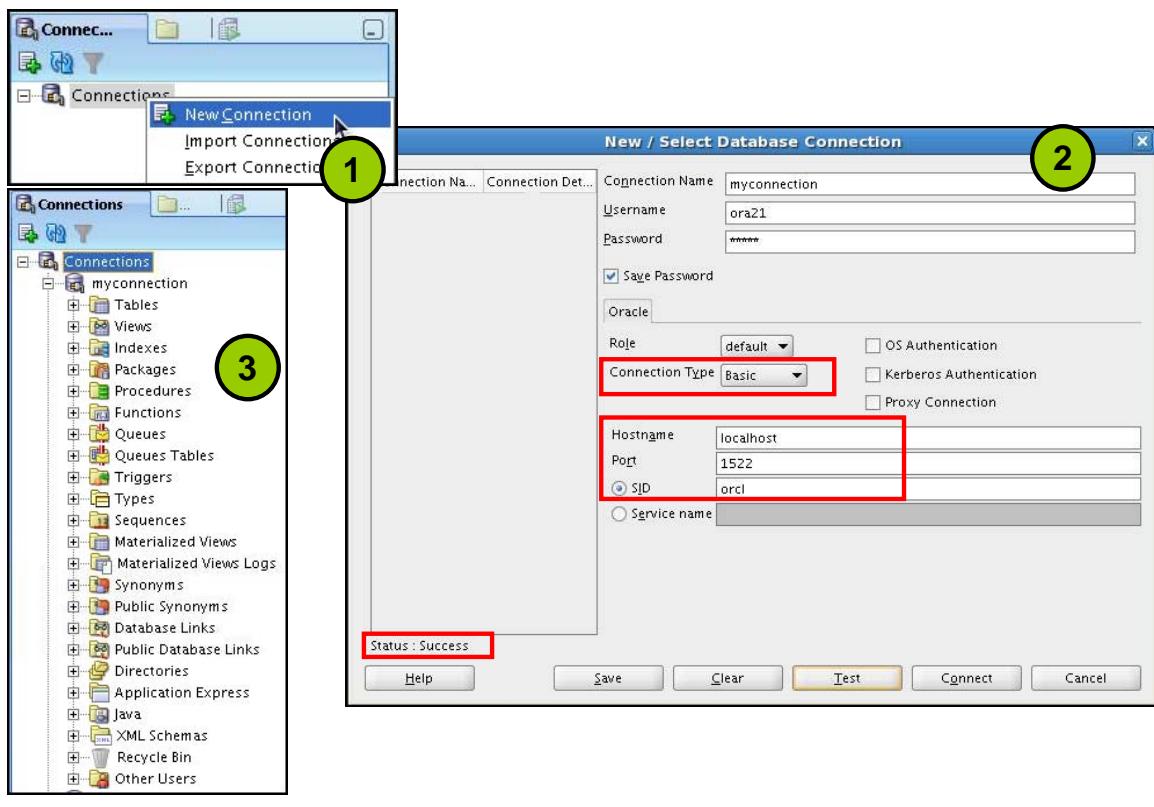
기본적으로 `tnsnames.ora` 파일은 `$ORACLE_HOME/network/admin` 디렉토리에 있지만, `TNS_ADMIN` 환경 변수 또는 레지스트리 값에 지정된 디렉토리에 있을 수도 있습니다. SQL Developer를 시작하고 Database Connections 대화상자를 표시하면 SQL Developer는 시스템의 `tnsnames.ora` 파일에 정의된 모든 연결을 자동으로 임포트합니다.

참고: Windows에서 `tnsnames.ora` 파일이 존재하지만 SQL Developer가 해당 연결을 사용하고 있지 않으면 `TNS_ADMIN`을 시스템 환경 변수로 정의하십시오.

연결을 XML 파일로 эксп포트하여 나중에 재사용할 수 있습니다.

동일한 데이터베이스에 다른 유저로 연결하거나 다른 데이터베이스에 연결하도록 추가 연결을 생성할 수 있습니다.

데이터베이스 연결 생성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 연결 생성(계속)

데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. Connections 탭 페이지에서 **Connections**를 마우스 오른쪽 버튼으로 누르고 **New Connection**을 선택합니다.
2. New>Select Database Connection window에 연결 이름을 입력합니다. 연결하려는 스키마의 username 및 암호를 입력합니다.
 - a) Role drop-down box에서 default 또는 SYSDBA를 선택할 수 있습니다. sys 유저 또는 데이터베이스 관리자 권한이 있는 모든 유저에 대해 SYSDBA를 선택하면 됩니다.
 - b) 연결 유형은 다음 중에서 선택하면 됩니다.
 - **Basic:** 이 유형의 경우 연결하려는 데이터베이스의 호스트 이름과 SID를 입력합니다. 포트는 이미 1521로 설정되어 있습니다. 원격 데이터베이스 연결을 사용하는 경우에는 서비스 이름을 직접 입력할 수도 있습니다.
 - **TNS:** tnsnames.ora 파일에서 임포트한 데이터베이스 alias 중 하나를 선택할 수 있습니다.
 - **LDAP:** Oracle Identity Management의 구성 요소인 Oracle Internet Directory에서 데이터베이스 서비스를 조회할 수 있습니다.
 - **Advanced:** 데이터베이스에 연결할 커스텀 JDBC URL을 정의할 수 있습니다.
 - c) Test를 눌러 연결이 올바르게 설정되었는지 확인합니다.
 - d) Connect를 누릅니다.

데이터베이스 연결 생성(계속)

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

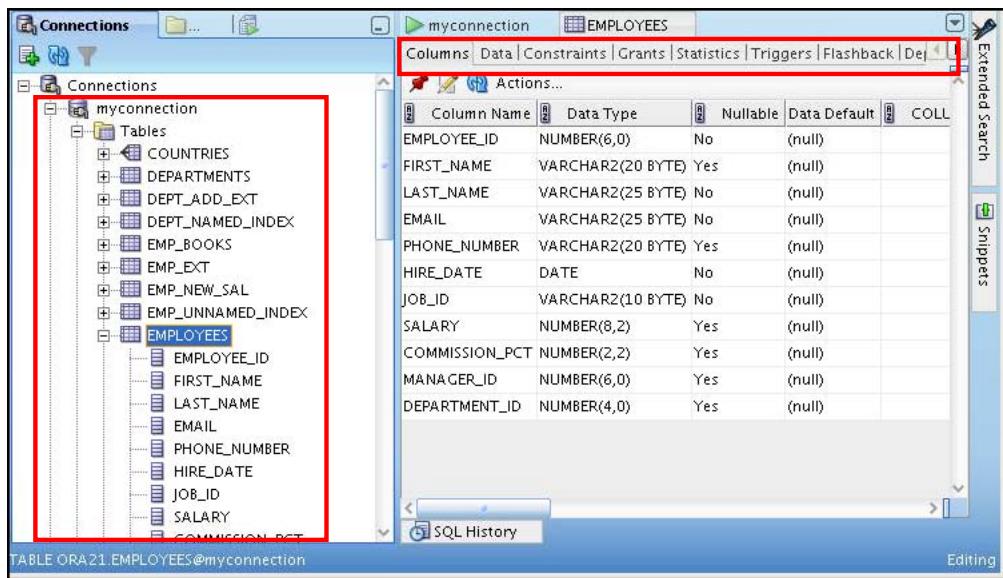
3. Connections Navigator에 연결이 추가됩니다. 연결을 확장하여 데이터베이스 객체를 볼 수 있으며 종속성, 상세 내역, 통계 등의 객체 정의를 볼 수 있습니다.

참고: 같은 New>Select Database Connection window에서 Access, MySQL 및 SQL Server 탭을 사용하여 third-party 데이터 소스에 대한 연결을 정의할 수 있습니다. 그러나 이러한 연결은 읽기 전용 연결이며 해당 데이터 소스에서 객체 및 데이터를 찾아볼 수만 있습니다.

데이터베이스 객체 탐색

Connections Navigator를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스 스키마에서 여러 객체 탐색
- 여러 객체의 정의를 한 번에 검토



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 객체 탐색

데이터베이스 연결을 생성한 후 Connections Navigator를 사용하여 데이터베이스 스키마에서 테이블, 뷰, 인덱스, 패키지, 프로시저, 트리거, 유형 등의 여러 객체를 탐색할 수 있습니다.

데이터 딕셔너리에서 추출된 객체 정의가 여러 정보 탭에 분할되어 있는 것을 볼 수 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 읽기 쉬운 탭 페이지에 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 세부 정보가 표시됩니다.

슬라이드에서처럼 EMPLOYEES 테이블의 정의를 보려면 다음 단계를 수행하십시오.

1. Connections Navigator에서 Connections 노드를 확장합니다.
2. Tables를 확장합니다.
3. EMPLOYEES를 누릅니다. 기본적으로 Columns 탭이 선택되며, 이 탭에는 해당 테이블의 열 설명이 표시됩니다. Data 탭을 통해 테이블 데이터를 볼 수 있으며 새 행을 입력하고, 데이터를 갱신하고, 해당 변경 내용을 데이터베이스로 커밋할 수 있습니다.

테이블 구조 표시

DESCRIBE 명령을 사용하여 테이블의 구조를 표시합니다.

The screenshot shows the Oracle SQL Developer interface. The title bar says "myconnection". The main window displays the results of the DESCRIBE EMPLOYEES command. The output is as follows:

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8, 2)
COMMISSION_PCT		NUMBER(2, 2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

11 rows selected

ORACLE®

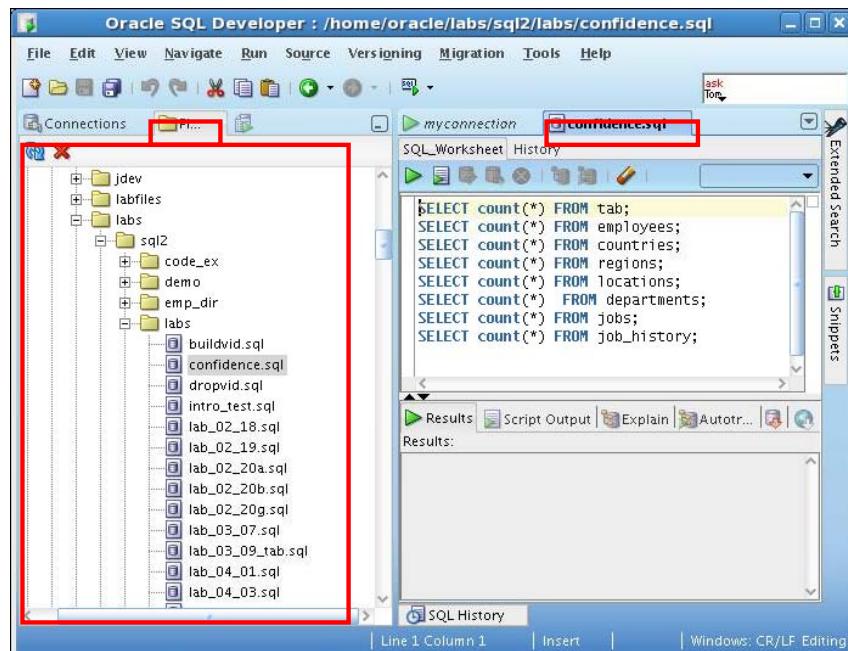
Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시

SQL Developer에서 DESCRIBE 명령을 사용하여 테이블 구조를 표시할 수도 있습니다.
이 명령의 결과로 열 이름, 데이터 유형 및 열에 반드시 데이터가 포함되어야 하는지
여부가 표시됩니다.

파일 탐색

File Navigator를 사용하여 파일 시스템을 탐색하고 시스템 파일을 열 수 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

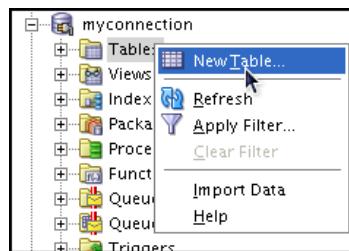
데이터베이스 객체 탐색

File Navigator를 사용하여 시스템 파일을 찾아서 열 수 있습니다.

- Files Navigator를 보려면 Files 탭을 누르거나 View > Files를 누릅니다.
- 파일 내용을 보려면 파일 이름을 두 번 눌러 SQL Worksheet 영역에 파일 내용을 표시합니다.

스키마 객체 생성

- SQL Developer는 다음 방법을 통한 스키마 객체 생성을 지원합니다.
 - SQL Worksheet에서 SQL 문 실행
 - 컨텍스트 메뉴 사용
- 편집 대화상자나 문맥에 따른 여러 가지 메뉴 중 하나를 사용하여 객체를 편집합니다.
- 새 객체 생성 또는 기존 스키마 객체 편집과 같은 조정을 위해 DDL(데이터 정의어)을 봅니다.



ORACLE

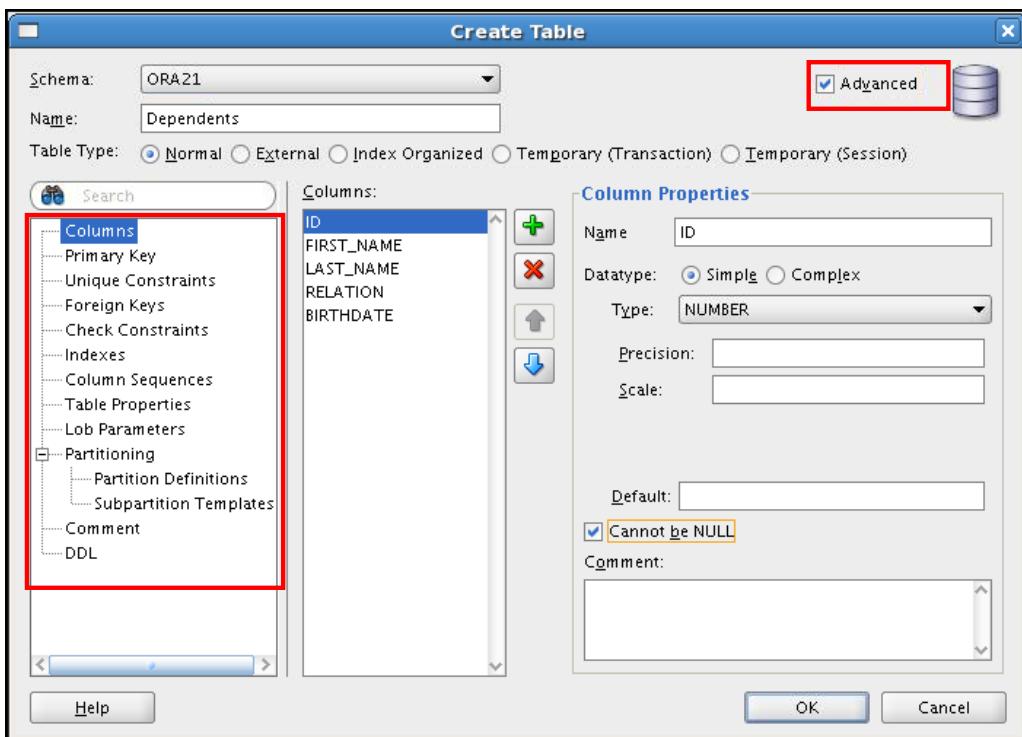
Copyright © 2009, Oracle. All rights reserved.

스키마 객체 생성

SQL Developer는 SQL Worksheet에서 SQL 문을 실행하여 스키마 객체 생성을 지원합니다. 또한 컨텍스트 메뉴를 사용하여 객체를 생성할 수도 있습니다. 객체를 생성한 후 편집 대화상자나 문맥에 따른 여러 메뉴 중 하나를 사용하여 객체를 편집할 수 있습니다. 새 객체를 생성하거나 기존 객체를 편집할 때 이러한 조정 작업을 위한 DDL을 확인할 수 있습니다. 스키마의 여러 객체에 대한 전체 DDL을 생성하려는 경우 Export DDL 옵션을 사용할 수 있습니다.

슬라이드에서는 컨텍스트 메뉴를 사용하여 테이블을 생성하는 방법을 보여줍니다. 새 테이블 생성 대화상자를 열려면 Tables를 마우스 오른쪽 버튼으로 누르고 New Table을 선택합니다. 데이터베이스 객체를 생성하고 편집하는 대화상자에는 여러 탭이 있으며 각 탭은 해당 객체 유형의 논리적 속성 그룹화를 반영합니다.

새 테이블 생성: 예제



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

새 테이블 생성: 예제

Create Table 대화상자에서 Advanced 체크 박스를 선택하지 않은 경우, 열과 자주 사용하는 일부 기능을 지정하여 빠르게 테이블을 생성할 수 있습니다.

Advanced 체크 박스를 선택한 경우에는 Create Table 대화상자에 여러 옵션이 표시되며, 여기에서 테이블을 생성하는 동안 확장된 기능을 지정할 수 있습니다.

슬라이드의 예제는 Advanced 체크 박스를 선택하여 DEPENDENTS 테이블을 생성하는 방법을 보여줍니다.

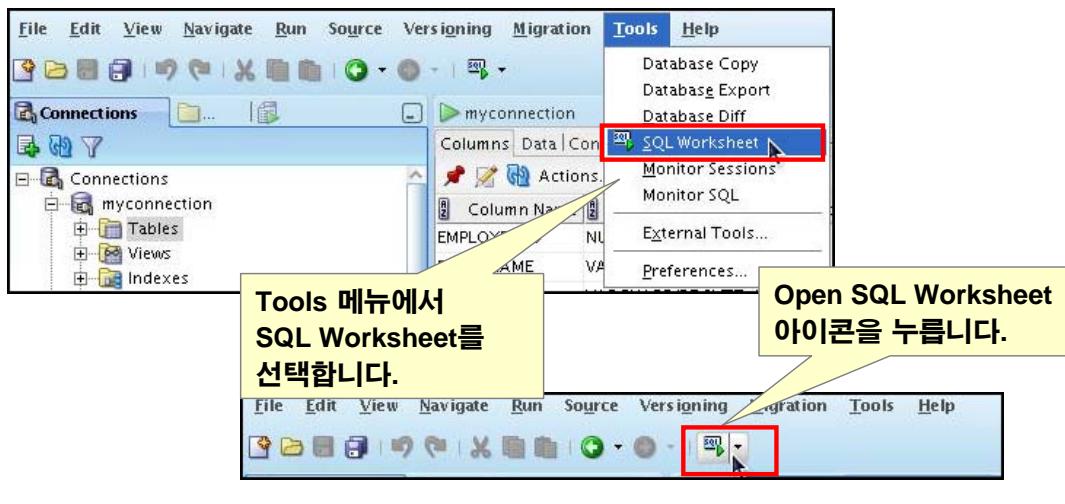
새 테이블을 생성하려면 다음 단계를 수행하십시오.

1. Connections Navigator에서 Tables를 마우스 오른쪽 버튼으로 누릅니다.
2. Create TABLE을 선택합니다.
3. Create Table 대화상자에서 Advanced를 선택합니다.
4. 열 정보를 지정합니다.
5. OK를 누릅니다.

또한 필수 사항은 아니지만 대화상자의 Primary Key 탭을 사용하여 Primary Key를 지정하는 것이 좋습니다. 생성한 테이블을 편집하려면 Connections Navigator에서 테이블을 마우스 오른쪽 버튼으로 누르고 Edit를 선택합니다.

SQL Worksheet 사용

- SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행합니다.
- 워크시트와 연관된 데이터베이스 연결로 처리할 수 있는 모든 작업을 지정합니다.



Copyright © 2009, Oracle. All rights reserved.

ORACLE

SQL Worksheet 사용

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet을 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. SQL Worksheet은 특정 범위까지 SQL*Plus 문을 지원합니다. SQL Worksheet에서 지원되지 않는 SQL*Plus 문은 무시되고 데이터베이스에 전달되지 않습니다.

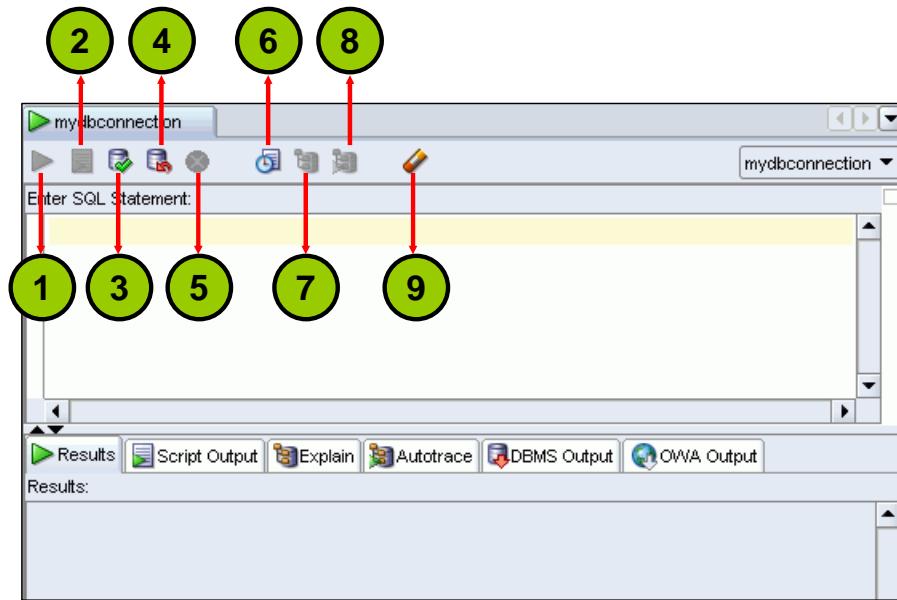
다음과 같이 워크시트와 연관된 데이터베이스 연결에 의해 처리될 수 있는 작업을 지정합니다.

- 테이블 생성
- 데이터 삽입
- 트리거 생성 및 편집
- 테이블에서 데이터 선택
- 파일에 선택한 데이터 저장

다음 중 하나를 사용하여 SQL Worksheet을 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- Open SQL Worksheet 아이콘을 누릅니다.

SQL Worksheet 사용



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

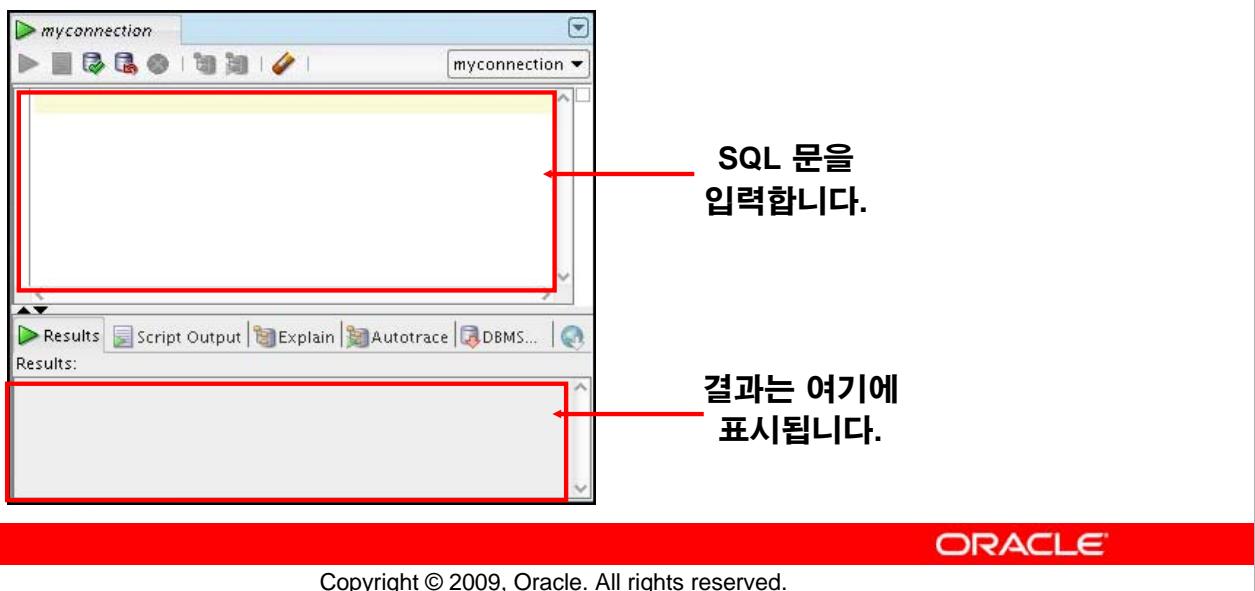
SQL Worksheet 사용(계속)

단축키 또는 아이콘을 사용하여 SQL 문 실행, 스크립트 실행, 실행한 SQL 문 기록 보기 등의 특정 작업을 수행하는 경우가 있습니다. 여러 아이콘이 있는 SQL Worksheet 도구 모음을 사용하여 다음 작업을 수행할 수 있습니다.

1. **Execute Statement:** Enter SQL Statement 상자에서 커서가 위치한 명령문을 실행합니다. SQL 문에 바인드 변수를 사용할 수 있지만 치환 변수는 사용할 수 없습니다.
2. **Run Script:** Script Runner를 사용하여 Enter SQL Statement 상자에 있는 모든 명령문을 실행합니다. SQL 문에 치환 변수를 사용할 수 있지만 바인드 변수는 사용할 수 없습니다.
3. **Commit:** 데이터베이스에 대한 모든 변경 사항을 기록하고 트랜잭션을 종료합니다.
4. **Rollback:** 데이터베이스에 대한 모든 변경 사항을 데이터베이스에 기록하지 않은 채 폐기하고 트랜잭션을 종료합니다.
5. **Cancel:** 현재 실행 중인 모든 명령문의 실행을 정지합니다.
6. **SQL History:** 실행한 SQL 문에 대한 정보가 있는 대화상자를 표시합니다.
7. **Execute Explain Plan:** Explain 탭을 눌러 볼 수 있는 실행 계획을 생성합니다.
8. **Autotrace:** 명령문에 대한 추적 정보를 생성합니다.
9. **Clear:** Enter SQL Statement 상자에서 명령문을 지웁니다.

SQL Worksheet 사용

- SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행합니다.
- 워크시트와 연관된 데이터베이스 연결로 처리할 수 있는 모든 작업을 지정합니다.



SQL Worksheet 사용(계속)

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. 모든 SQL 및 PL/SQL 명령이 지원되며 이러한 명령은 SQL Worksheet에서 오라클 데이터베이스로 직접 전달됩니다. SQL Developer에서 사용되는 SQL*Plus 명령은 데이터베이스로 전달하기 전에 SQL Worksheet에서 해석되어야 합니다.

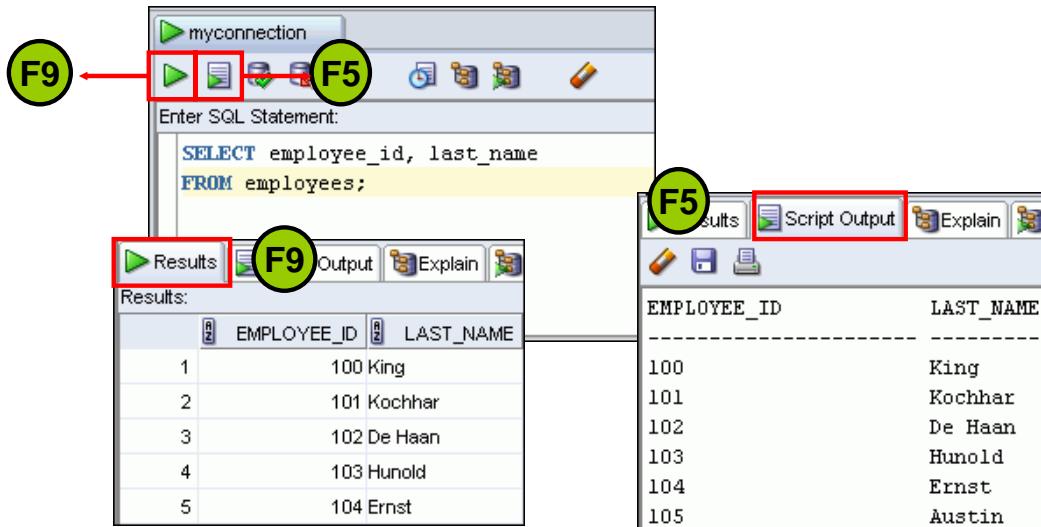
현재 SQL Worksheet에서는 다수의 SQL*Plus 명령을 지원합니다. SQL Worksheet에서 지원되지 않는 명령은 무시되며 오라클 데이터베이스로 보내지지 않습니다. SQL Worksheet를 통해 SQL 문을 실행하거나 SQL*Plus 명령 중 일부를 실행할 수 있습니다.

다음 두 가지 옵션 중 하나를 사용하여 SQL Worksheet를 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- Open SQL Worksheet 아이콘을 누릅니다.

SQL 문 실행

Enter SQL Statement 상자를 사용하여 SQL 문을 하나 또는 여러 개 입력합니다.



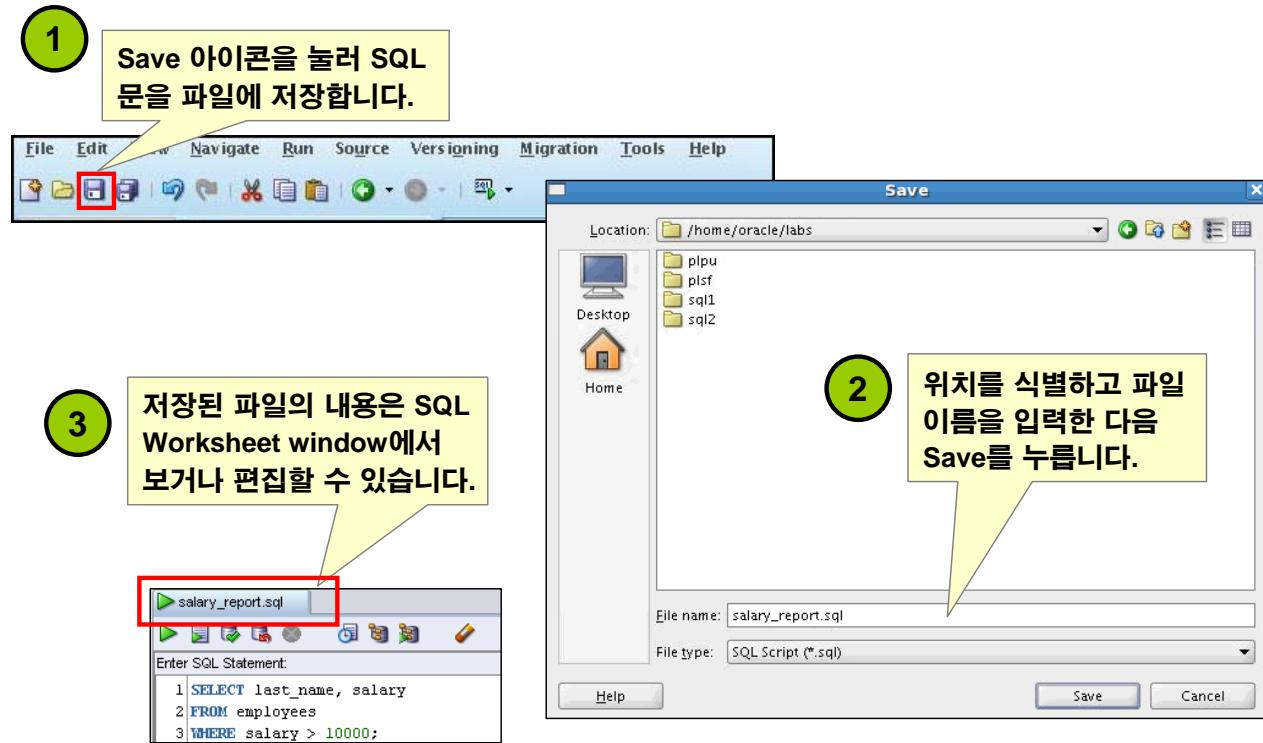
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 문 실행

슬라이드의 예제에서는 동일한 query에 대해 F9 키 또는 Execute Statement를 사용한 출력과 F5 키 또는 Run Script를 사용한 출력의 차이를 보여줍니다.

SQL 스크립트 저장



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 스크립트 저장

SQL 문을 SQL Worksheet에서 텍스트 파일로 저장할 수 있습니다. Enter SQL Statement 상자의 내용을 저장하려면 다음 단계를 따르십시오.

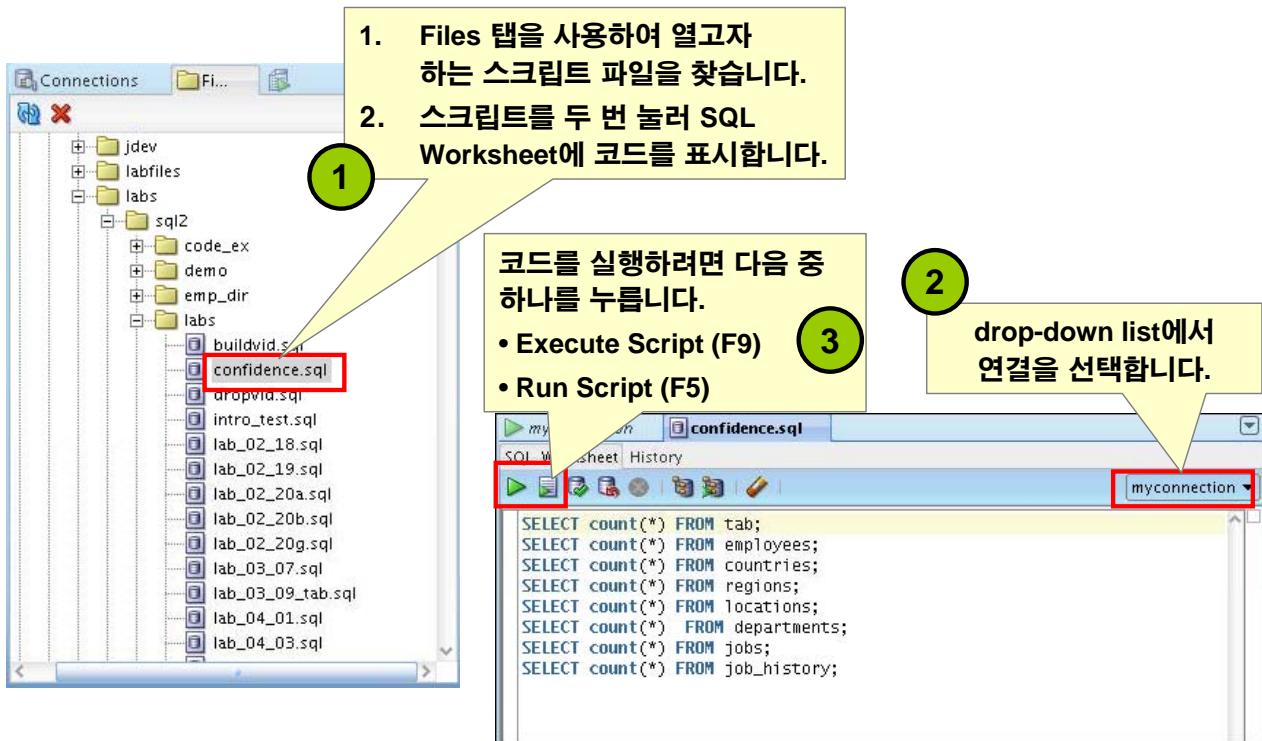
1. Save 아이콘을 누르거나 File > Save 메뉴 항목을 사용합니다.
2. Windows Save 대화상자에서 파일 이름과 파일을 저장할 위치를 입력합니다.
3. Save를 누릅니다.

내용을 파일에 저장하면 Enter SQL Statement window가 파일 내용의 탭 페이지를 표시합니다. 여러 개의 파일을 동시에 열 수 있으며 각 파일은 탭 페이지로 표시됩니다.

스크립트 경로

스크립트를 찾고 저장할 기본 경로를 선택할 수 있습니다. Tools > Preferences > Database > Worksheet Parameters 아래에서 "Select default path to look for scripts" 필드에 값을 입력하십시오.

저장된 SQL 스크립트 실행: 방법 1



ORACLE

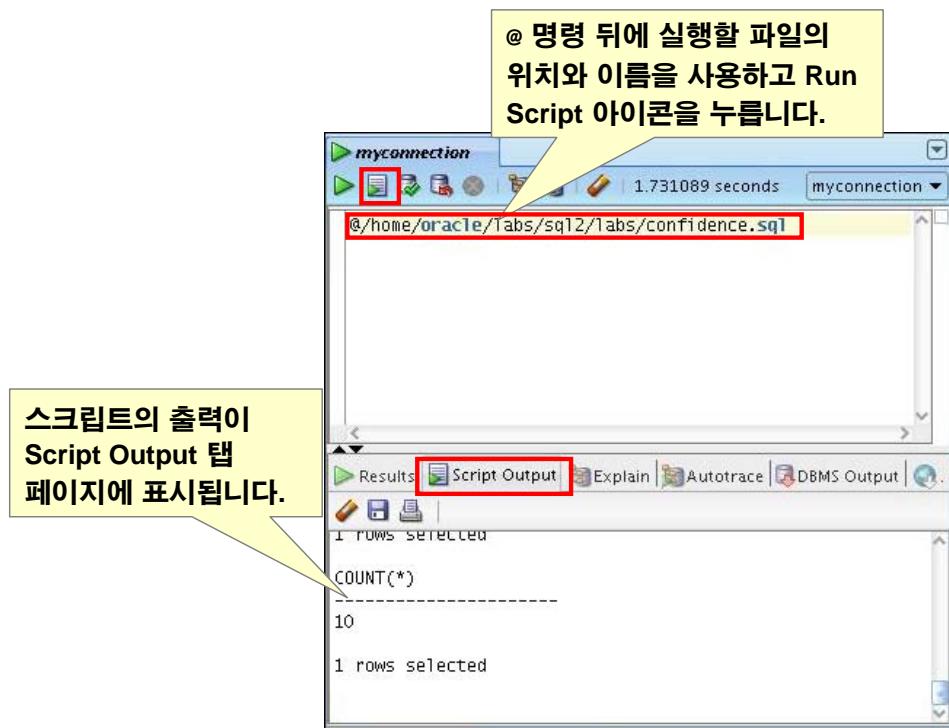
Copyright © 2009, Oracle. All rights reserved.

저장된 SQL 스크립트 실행: 방법 1

스크립트 파일을 열고 SQL Worksheet 영역에 코드를 표시하려면 다음을 수행합니다.

- Files Navigator에서 열고자 하는 스크립트 파일을 선택합니다. 또는 해당 파일이 있는 위치로 이동합니다.
- 두 번 눌러 엽니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
- connection drop-down list에서 연결을 선택합니다.
- 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다.
connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다.
스크립트 실행에 사용할 연결을 선택합니다.
또는 다음과 같이 할 수도 있습니다.
- File > Open을 선택합니다. Open 대화상자가 표시됩니다.
- Open 대화상자에서 열고자 하는 스크립트 파일을 선택합니다. (또는 해당 파일이 있는 위치로 이동합니다.)
- Open을 누릅니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
- connection drop-down list에서 연결을 선택합니다.
- 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다.
connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다.
스크립트 실행에 사용할 연결을 선택합니다.

저장된 SQL 스크립트 실행: 방법 2



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

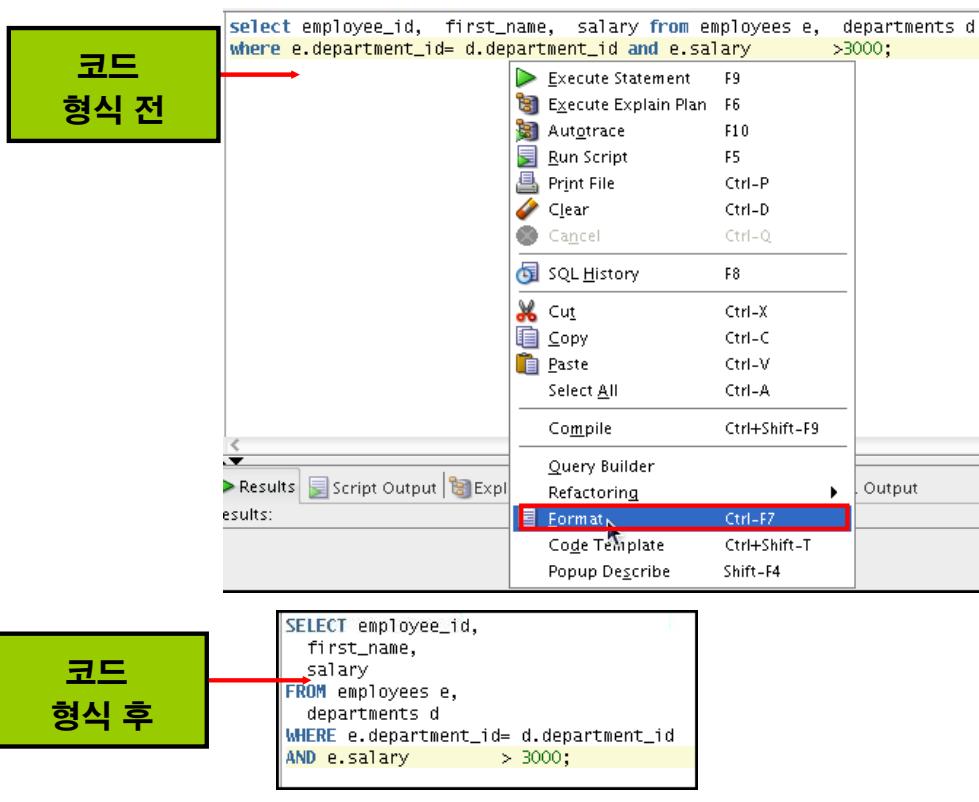
저장된 SQL 스크립트 실행: 방법 2

저장된 SQL 스크립트를 실행하려면 다음을 수행합니다.

1. Enter SQL Statement window에서 @ 명령을 사용하고 그 뒤에 실행할 파일의 위치와 이름을 입력합니다.
2. Run Script 아이콘을 누릅니다.

파일의 실행 결과가 Script Output 탭 페이지에 표시됩니다. 또한 Script Output 탭 페이지에서 Save 아이콘을 눌러 스크립트 출력을 저장할 수도 있습니다. Windows Save 대화상자가 나타나고 파일의 이름 및 위치를 선택할 수 있습니다.

SQL 코드 형식 지정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 코드 형식 지정

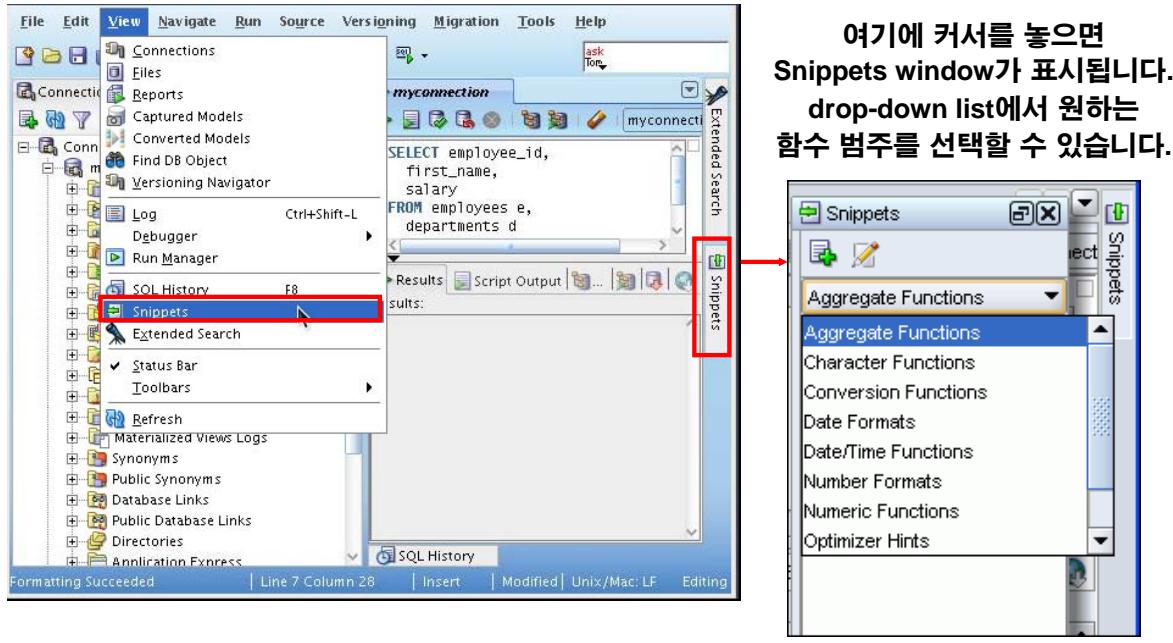
SQL 코드의 들여쓰기, 간격, 대소문자 및 줄 구분을 보기 좋게 지정해야 할 경우가 있습니다.
SQL Developer에는 SQL 코드의 형식을 지정하는 기능이 있습니다.

SQL 코드에 형식을 지정하려면 명령문 영역을 마우스 오른쪽 버튼으로 누른 다음 Format SQL을 선택합니다.

슬라이드의 예제에서 형식이 지정되기 전의 SQL 코드에는 키워드가 대문자로 표시되지 않고 명령문의 들여쓰기가 제대로 되어 있지 않습니다. 형식 지정 이후의 SQL 코드에서는 키워드가 대문자로 표시되고 명령문이 적절히 들여쓰기되어 보기 좋게 다듬어졌습니다.

Snippet 사용

Snippet은 구문이거나 예제일 수 있는 코드 부분입니다.



Copyright © 2009, Oracle. All rights reserved.

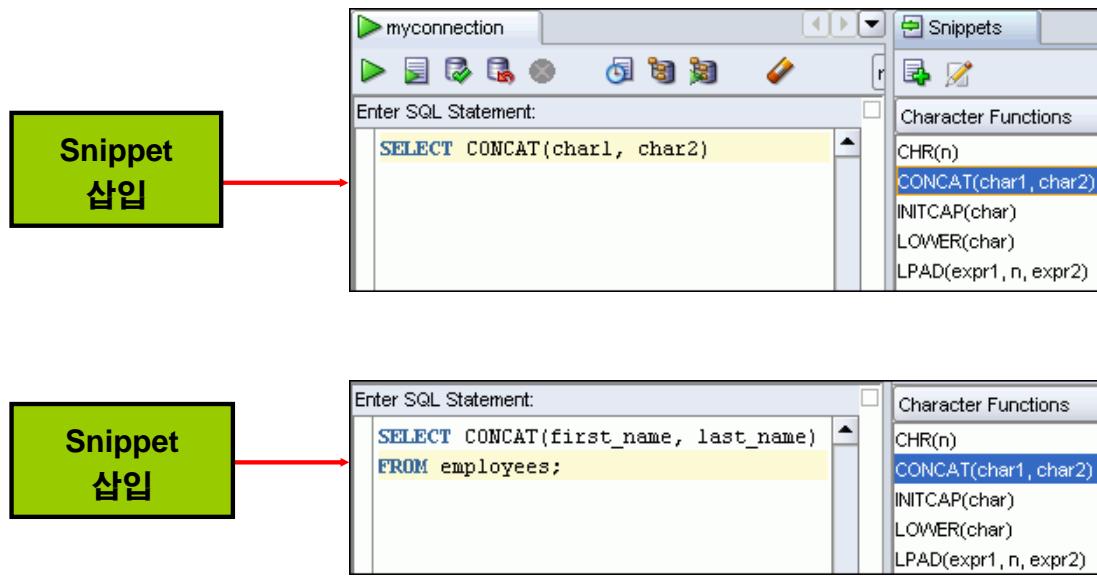
Snippet 사용

SQL Worksheet을 사용하거나 PL/SQL 함수 또는 프로시저를 생성하거나 편집할 때 특정 코드 부분을 사용해야 하는 경우가 있습니다. SQL Developer에는 Snippets, 즉 Snippet이라는 기능이 있습니다. Snippet은 SQL 함수, 옵티마이저 힌트 및 기타 PL/SQL 프로그래밍 기법과 같은 코드 부분입니다. Snippet을 Editor window로 끌어올 수 있습니다.

Snippet을 표시하려면 View > Snippets를 선택합니다.

그리면 Snippets window가 오른쪽에 표시됩니다. drop-down list를 사용하여 그룹을 선택할 수 있습니다. Snippets window가 숨겨진 경우 이를 표시할 수 있도록 오른쪽 window 여백에 Snippets 버튼이 표시됩니다.

Snippet 사용: 예제



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Snippet 사용: 예제

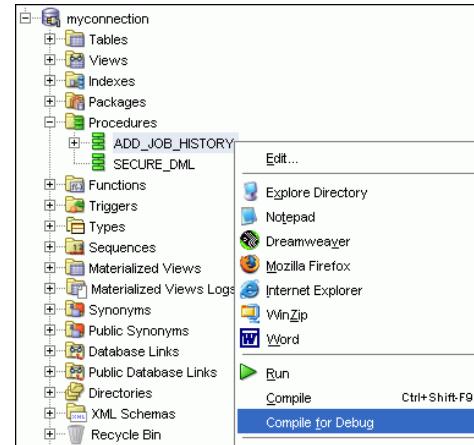
SQL Worksheet 또는 PL/SQL 함수나 프로시저의 코드에 Snippet을 삽입하려면 Snippets window에서 코드의 원하는 위치로 Snippet을 끌어옵니다. 그런 다음 SQL 함수가 현재 컨텍스트에서 유효하도록 구문을 편집할 수 있습니다. 도구 설명에서 SQL 함수에 대한 간단한 설명을 보려면 커서를 함수 이름 위에 둡니다.

슬라이드의 예제는 Snippets window의 Character Functions 그룹에서 `CONCAT(char1, char2)`를 끌어오는 과정을 보여줍니다. 이어서 다음과 같이 `CONCAT` 함수 구문을 편집하고 나머지 명령문을 추가합니다.

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

프로시저 및 함수 디버깅

- **SQL Developer를 사용하여 PL/SQL 함수 및 프로시저를 디버그합니다.**
- **프로시저를 디버그할 수 있도록 PL/SQL 컴파일을 수행하려면 "Compile for Debug" 옵션을 사용합니다.**
- **중단점을 설정하고 Step Into 및 Step Over 작업을 수행하려면 Debug 메뉴 옵션을 사용합니다.**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로시저 및 함수 디버깅

SQL Developer에서는 PL/SQL 프로시저 및 함수를 디버깅할 수 있습니다. Debug 메뉴 옵션을 사용하여 다음 디버깅 작업을 수행할 수 있습니다.

- **Find Execution Point**를 사용하면 다음 실행 지점으로 이동합니다.
- **Resume**을 사용하면 실행이 계속됩니다.
- **Step Over**를 사용하면 다음 메소드를 건너뛰고 해당 메소드 뒤의 다음 명령문으로 이동합니다.
- **Step Into**를 사용하면 다음 메소드의 첫번째 명령문으로 이동합니다.
- **Step Out**을 사용하면 현재 메소드에서 나와 다음 명령문으로 이동합니다.
- **Step to End of Method**를 사용하면 현재 메소드의 마지막 명령문으로 이동합니다.
- **Pause**를 사용하면 실행이 중지되지만 종료되지는 않으므로 나중에 실행을 재개할 수 있습니다.
- **Terminate**를 사용하면 실행이 중지 및 종료됩니다. 이 지점에서는 실행을 재개할 수 없습니다. 대신 함수나 프로시저 시작 부분부터 실행이나 디버깅을 시작하려면 Source 탭 도구 모음의 Run 또는 Debug 아이콘을 누르십시오.
- **Garbage Collection**을 사용하면 캐시에서 무효한 객체가 제거되고 자주 액세스하는 유효한 객체가 사용됩니다.

이러한 옵션은 디버깅 도구 모음에서 아이콘으로도 사용할 수 있습니다.

데이터베이스 보고

SQL Developer에서는 데이터베이스 및 해당 객체에 대한 여러 가지 미리 정의된 보고서가 제공됩니다.

The screenshot shows the SQL Developer interface with the 'Reports' tab selected in the top navigation bar. On the left, a tree view lists various report categories such as 'About Your Database', 'All Objects', 'Dependencies', and 'Data Dictionary'. The 'Dependencies' node is highlighted with a red box. On the right, a table displays dependency information with columns for Owner, Name, Type, Referenced Owner, and Referenced Name. The table contains 18 rows of data related to CTXSYS objects.

Owner	Name	Type	Referenced Owner	Referenced Name
CTXSYS	CTX_CLASSES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_CLS	PACKAGE	SYS	STANDARD
CTXSYS	CTX_DOC	PACKAGE	SYS	STANDARD
CTXSYS	CTX_INDEX_SETS	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SETS	VIEW	SYS	USER\$
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	CTXSYS	DR\$INDEX_SET_INDEX
CTXSYS	CTX_INDEX_SET_INDEXES	VIEW	SYS	USER\$
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECTS	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTES	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$CLASS
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE
CTXSYS	CTX_OBJECT_ATTRIBUTE_LOV	VIEW	CTXSYS	DR\$OBJECT_ATTRIBUTE_LOV
CTXSYS	CTX_PARAMETERS	VIEW	CTXSYS	DR\$PARAMETER

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 보고

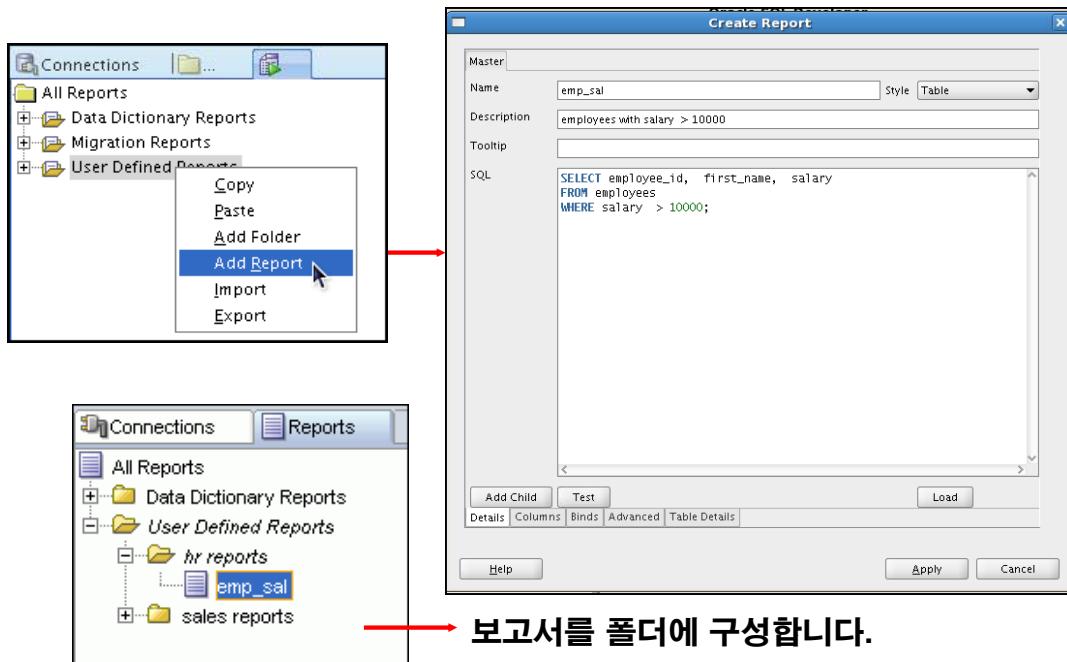
SQL Developer는 데이터베이스 및 객체에 대한 다양한 보고서를 제공합니다. 이러한 보고서는 다음 범주로 그룹화할 수 있습니다.

- About Your Database 보고서
- Database Administration 보고서
- Table 보고서
- PL/SQL 보고서
- Security 보고서
- XML 보고서
- Jobs 보고서
- Streams 보고서
- All Objects 보고서
- Data Dictionary 보고서
- User-Defined 보고서

보고서를 표시하려면 window 왼쪽의 Reports 탭을 누르십시오. 그러면 개별 보고서가 window 오른쪽의 탭 창에 표시되며 각 보고서에 대해 drop-down list를 사용하여 보고서를 표시할 데이터베이스 연결을 선택할 수 있습니다. 객체에 대한 보고서의 경우 선택한 데이터베이스 연결과 연관된 데이터베이스 유저가 볼 수 있는 객체만 표시되고 행은 일반적으로 Owner별로 정렬됩니다. 고유한 유저 정의 보고서를 생성할 수도 있습니다.

유저 정의 보고서 작성

반복적으로 사용할 수 있도록 유저 정의 보고서를 생성하고 저장합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

유저 정의 보고서 작성

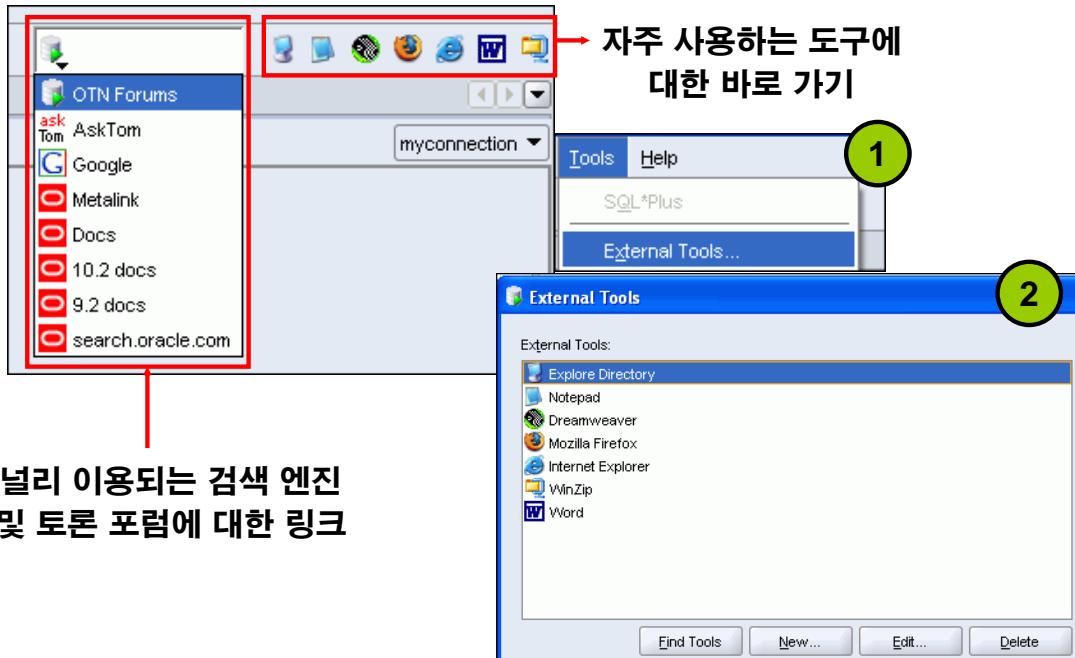
유저 정의 보고서는 SQL Developer 유저가 생성하는 보고서입니다. 유저 정의 보고서를 생성하려면 다음 단계를 수행하십시오.

1. Reports 아래에서 User Defined Reports 노드를 마우스 오른쪽 버튼으로 누르고 Add Report를 선택합니다.
2. Create Report 대화상자에서 보고서 이름을 지정하고 보고서 정보를 검색할 SQL query를 지정합니다. 그런 다음 Apply를 누릅니다.

슬라이드의 예제에서 보고서 이름은 emp_sal로 지정됩니다. 보고서에 급여 $>= 10000$ 인 사원에 대한 세부 정보가 포함되어 있음을 나타내는 선택적 설명이 제공됩니다. 유저 정의 보고서에 표시할 정보를 검색하기 위한 전체 SQL 문이 SQL 상자에서 지정됩니다. Reports Navigator 화면에서 보고서 이름 위에 잠시 커서를 두면 표시되는 선택적 도구 설명을 포함할 수도 있습니다.

유저 정의 보고서를 폴더에 구성하고 폴더와 하위 폴더의 계층을 생성할 수 있습니다. 유저 정의 보고서에 대한 폴더를 생성하려면 User Defined Reports 노드나 해당 노드 아래에 있는 임의의 폴더 이름을 마우스 오른쪽 버튼으로 누르고 Add Folder를 선택합니다. 이러한 보고서에 대한 폴더를 포함한 유저 정의 보고서에 대한 정보는 유저 특정 정보를 위한 디렉토리 아래 UserReports.xml이라는 파일에 저장됩니다.

검색 엔진 및 External 도구



널리 이용되는 검색 엔진
및 토론 포럼에 대한 링크

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

검색 엔진 및 External 도구

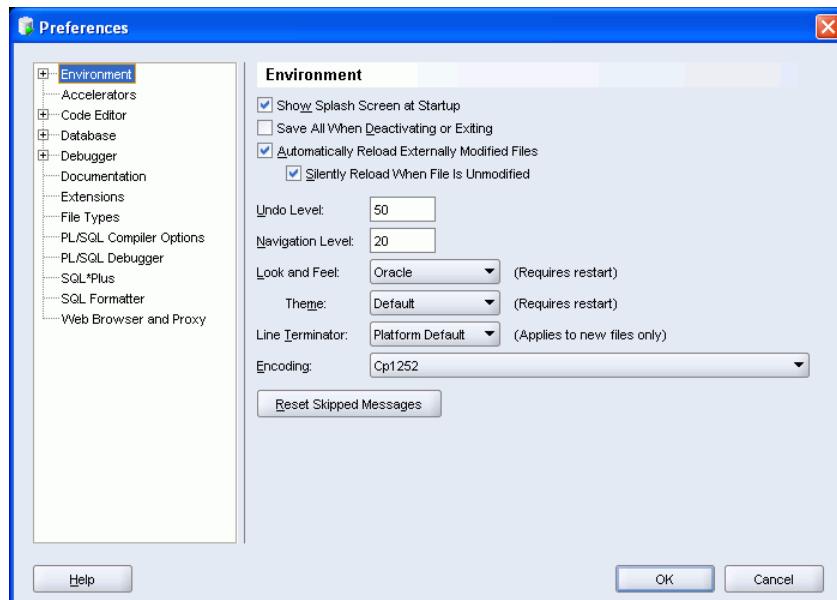
SQL 개발자의 생산성을 높이기 위해 SQL Developer에는 AskTom, Google 등의 유명 검색 엔진과 토의 포럼에 대한 빠른 링크가 포함되었습니다. 또한 메모장, Microsoft Word, Dreamweaver 등의 자주 사용하는 일부 도구에 대한 단축키 아이콘도 사용할 수 있습니다.

기존 리스트에 external 도구를 추가하거나 자주 사용하지 않는 도구에 대한 단축키를 삭제할 수도 있습니다. 이를 위해 다음을 수행하십시오.

1. Tools 메뉴에서 External Tools를 선택합니다.
2. 새 도구를 추가하려면 External Tools 대화상자에서 New를 선택합니다. 리스트에서 도구를 제거하려면 Delete를 선택합니다.

환경 설정

- SQL Developer 인터페이스와 환경을 커스터마이즈합니다.
- Tools 메뉴에서 Preferences를 선택합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

환경 설정

SQL Developer 환경 설정을 자신의 취향과 요구 사항에 맞게 수정함으로써 SQL Developer 인터페이스와 환경의 다양한 요소를 커스터마이즈할 수 있습니다. SQL Developer 환경 설정을 수정하려면 Tools와 Preferences를 차례로 선택합니다.

환경 설정은 다음 범주로 그룹화됩니다.

- Environment
- Accelerators (keyboard shortcuts)
- Code Editors
- Database
- Debugger
- Documentation
- Extensions
- File Types
- Migration
- PL/SQL Compilers
- PL/SQL Debugger 등

SQL Developer 레이아웃 재설정

```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$ locate windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.40/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
/home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48/windowinglayout.xml
[oracle@EDRSR5P1 ~]$ cd /home/oracle/.sqldeveloper/system1.5.4.59.41/o.ide.11.1.1.0.22.49.48
Debugging.layout Editing.layout projects windowinglayout.xml
dtcache.xml preferences.xml settings.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ rm windowinglayout.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 레이아웃 재설정

SQL Developer로 작업하는 동안 Connections Navigator가 사라지거나 Log window를 원래의 위치에 도킹할 수 없는 경우 다음 단계를 수행하여 문제를 해결합니다.

1. SQL Developer를 종료합니다.
2. 터미널 window를 열고 locate 명령을 사용하여 windowinglayout.xml의 위치를 찾습니다.
3. windowinglayout.xml이 있는 디렉토리로 이동하여 해당 파일을 삭제합니다.
4. SQL Developer를 재시작합니다.

요약

이 부록에서는 SQL Developer를 사용하여 다음 작업을 수행하는 방법을 배웠습니다.

- 데이터베이스 객체 탐색, 생성 및 편집
- SQL Worksheet에서 SQL 문 및 스크립트 실행
- 커스텀 보고서 생성 및 저장



Copyright © 2009, Oracle. All rights reserved.

요약

SQL Developer는 데이터베이스 개발 작업을 간편하게 수행할 수 있는 무료 그래픽 도구입니다. SQL Developer를 사용하여 데이터베이스 객체를 탐색, 생성 및 편집할 수 있습니다. 또한 SQL Worksheet를 사용하여 SQL 문 및 스크립트를 실행할 수 있습니다. SQL Developer를 통해 고유의 특수 보고서 집합을 생성하여 저장해 두었다가 반복해서 사용할 수 있습니다.

D SQL*Plus 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- SQL*Plus에 로그인
- SQL 명령 편집
- SQL*Plus 명령을 사용하여 출력 형식 지정
- 스크립트 파일과 상호 작용

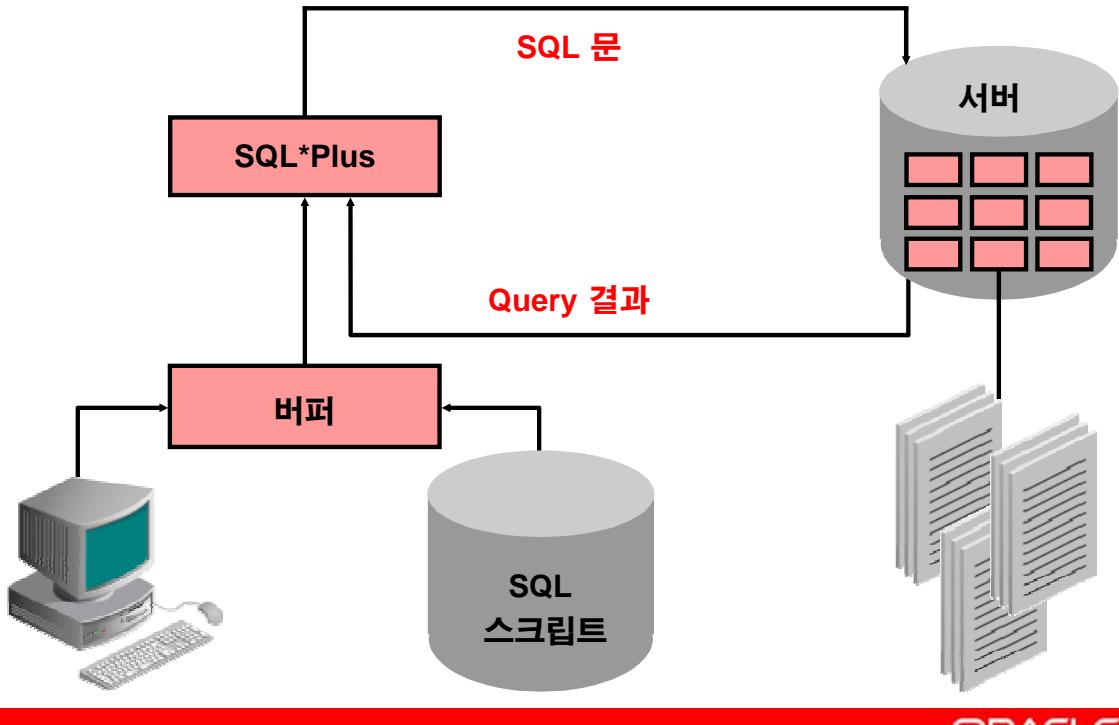
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

몇 번이고 사용할 수 있는 SELECT 문을 생성할 수 있습니다. 이 부록에서는 SQL*Plus 명령을 사용하여 SQL 문을 실행하는 과정도 다릅니다. SQL*Plus 명령을 사용하여 출력 형식을 지정하고, SQL 명령을 편집하고, SQL*Plus로 스크립트를 저장하는 방법을 배웁니다.

SQL과 SQL*Plus의 상호 작용



Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus

SQL은 임의의 도구 또는 응용 프로그램에서 Oracle 서버와 통신하는 데 사용하는 명령어입니다. Oracle SQL은 많은 확장을 포함합니다. SQL 문을 입력하면 SQL 버퍼(SQL Buffer)라고 하는 메모리 부분에 저장되어 새 SQL 문을 입력할 때까지 그대로 남아 있습니다. SQL*Plus는 SQL 문을 인식하여 실행을 위해 Oracle9i Server로 제출하는 도구입니다. SQL*Plus는 자체 명령어를 포함합니다.

SQL의 특성

- 프로그래밍의 경험이 별로 없거나 아예 없는 사용자를 포함하여 다양한 사용자가 사용할 수 있습니다.
- 비절차적 언어입니다.
- 시스템 생성 및 유지 관리에 필요한 시간을 단축합니다.
- 영어와 유사한 언어입니다.

SQL*Plus의 특성

- 명령문의 임시 항목을 받아들입니다.
- 파일을 사용하여 SQL을 입력할 수 있습니다.
- SQL 문 수정을 위한 행 편집기를 제공합니다.
- 환경 설정을 제어합니다.
- Query 결과를 기본 보고서 형식으로 만듭니다.
- 로컬 및 원격 데이터베이스에 액세스합니다.

SQL 문과 SQL*Plus 명령 비교

SQL

- 언어
 - ANSI 표준
 - 키워드를 약어로 표기할 수 없음
 - 명령문으로 데이터베이스의 데이터 및 테이블 정의를 조작함

SQL*Plus

- 환경
 - Oracle 고유
 - 키워드는 약어로 표시할 수 있음
 - 명령으로 데이터베이스의 값을 조작할 수 없음



Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus(계속)

다음 표에서는 SOL과 SOL*Plus를 비교합니다.

SQL	SQL*Plus
Oracle 서버와 통신하여 데이터에 액세스하기 위한 언어	SQL 문을 인식하고 서버로 보냄
ANSI(American National Standards Institute) 표준 SQL을 기반으로 함	SQL 문을 실행하기 위한 오라클 고유의 인터페이스
데이터베이스의 데이터 및 테이블을 조작함	데이터베이스의 값을 조작할 수 없음
SQL 버퍼에 하나 이상의 행이 입력됨	한 번에 한 행씩 입력되고 SQL 버퍼에 저장되지 않음
연속 문자가 없음	명령이 한 줄보다 긴 경우 연속 문자로 대시(-)를 사용
약어를 사용할 수 없음	약어를 사용할 수 있음
종료 문자를 사용하여 명령을 즉시 실행	종료 문자를 사용할 필요 없이 명령을 즉시 실행
함수를 사용하여 일부 형식 지정 수행	명령을 사용하여 데이터의 형식 지정

SQL*Plus 개요

- SQL*Plus에 로그인합니다.
- 테이블 구조를 설명합니다.
- SQL 문을 편집합니다.
- SQL*Plus에서 SQL을 실행합니다.
- SQL 문을 파일에 저장하고 첨부합니다.
- 저장된 파일을 실행합니다.
- 파일에서 버퍼로 명령을 로드하여 편집합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus

SQL*Plus는 다음 작업을 수행할 수 있는 환경입니다.

- 데이터베이스에서 데이터를 검색, 수정, 추가 및 제거하는 SQL 문을 실행합니다.
- query 결과의 형식을 지정하고 계산을 수행하고, 저장하고, 보고서 형식으로 인쇄합니다.
- 앞으로 반복 사용할 수 있도록 SQL 문을 저장하는 스크립트 파일을 생성합니다.

SQL*Plus 명령은 다음의 주요 범주로 나눌 수 있습니다.

범주	목적
환경	세션에 대한 SQL 문의 일반 동작에 영향을 줍니다.
형식	Query 결과의 형식을 지정합니다.
파일 조작	스크립트 파일을 저장, 로드, 실행합니다.
실행	SQL 문을 SQL 버퍼에서 Oracle 서버로 보냅니다.
편집	버퍼의 SQL 문을 수정합니다.
상호 작용	변수를 생성하여 SQL 문에 전달하고, 변수 값을 인쇄하고, 화면에 메시지를 출력합니다.
기타	데이터베이스에 연결하고, SQL*Plus 환경을 조작하며, 열 정의를 표시합니다.

SQL*Plus에 로그인

```
[oracle@EDRSR5P1 ~]$sqlplus
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:59:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Enter user-name: ora21@orcl
Enter password:
```

sqlplus [username[/password[@database]]]

```
[oracle@EDRSR5P1 ~]$sqlplus ora21/ora21@orcl
SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:58:06 2009
Copyright (c) 1982, 2009, Oracle. All rights reserved.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus에 로그인

SQL*Plus를 호출하는 방법은 Oracle Database를 실행 중인 운영 체제의 유형에 따라 다릅니다.

Linux 환경에서 로그인하려면 다음과 같이 하십시오.

1. Linux 바탕 화면을 마우스 오른쪽 버튼으로 누르고 terminal을 선택합니다.
2. 슬라이드에 표시된 sqlplus 명령을 입력합니다.
3. username, 암호 및 데이터베이스 이름을 입력합니다.

이 구문에서 다음이 적용됩니다.

username 데이터베이스 Username입니다.

password 데이터베이스 암호입니다. 여기에 암호를 입력하면 암호가 보입니다.

@database 데이터베이스 연결 문자열입니다.

참고: 암호의 무결성을 보장하려면 운영 체제 프롬프트에서 암호를 입력하지 마십시오. 대신 username만 입력하십시오. 암호는 암호 프롬프트에서 입력하십시오.

테이블 구조 표시

SQL*Plus DESCRIBE 명령을 사용하여 테이블의 구조를 표시합니다.

```
DESC[RE]B [tablename]
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시

SQL*Plus에서 DESCRIBE 명령을 사용하여 테이블의 구조를 표시할 수 있습니다. 이 명령의 결과로 열 이름, 데이터 유형 및 열에 반드시 데이터가 포함되어야 하는지 여부가 표시됩니다. 이 구문에서 다음이 적용됩니다.

tablename 유저가 액세스할 수 있는 기존의 테이블, 뷰 또는 동의어의 이름입니다.

DEPARTMENTS 테이블을 기술하려면 다음 명령을 사용합니다.

```
SQL> DESCRIBE DEPARTMENTS
```

```
Name Null? Type
```

```
-----  
DEPARTMENT_ID NOT NULL NUMBER(4)  
DEPARTMENT_NAME NOT NULL VARCHAR2(30)  
MANAGER_ID NUMBER(6)  
LOCATION_ID NUMBER(4)
```

테이블 구조 표시

```
DESCRIBE departments
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시(계속)

슬라이드의 예제는 DEPARTMENTS 테이블의 구조에 대한 정보를 표시합니다. 결과에서 다음이 적용됩니다.

Null?: 열에 데이터가 포함되어야 하는지 여부를 지정합니다. NOT NULL은 열에 데이터가 포함되어야 함을 나타냅니다.

Type: 열의 데이터 유형을 표시합니다.

SQL*Plus 편집 명령

- **A[PPEND] *text***
- **C[HANGE] / *old* / *new***
- **C[HANGE] / *text* /**
- **CL[EAR] BUFF[ER]**
- **DEL**
- **DEL *n***
- **DEL *m* *n***

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 편집 명령

SQL*Plus 명령은 한 번에 한 줄씩 입력되고 SQL 버퍼에 저장되지 않습니다.

명령	설명
A[PPEND] <i>text</i>	현재 행의 끝에 텍스트를 추가합니다.
C[HANGE] / <i>old</i> / <i>new</i>	현재 행에서 <i>old</i> 텍스트를 <i>new</i> 로 변경합니다.
C[HANGE] / <i>text</i> /	현재 행에서 <i>text</i> 를 삭제합니다.
CL[EAR] BUFF[ER]	SQL 버퍼에서 모든 행을 삭제합니다.
DEL	현재 행을 삭제합니다.
DEL <i>n</i>	<i>n</i> 행을 삭제합니다.
DEL <i>m</i> <i>n</i>	<i>m</i> 행부터 <i>n</i> 행까지 삭제합니다.

지침

- 명령을 완료하기 전에 Enter를 누르면 SQL*Plus에서 행 번호를 표시합니다.
- 종료 문자(세미콜론이나 슬래시) 중 하나를 입력하거나 Enter를 두 번 눌러 SQL 버퍼를 종료합니다. 그러면 SQL 프롬프트가 나타납니다.

SQL*Plus 편집 명령

- **I[NPUT]**
- **I[NPUT] *text***
- **L[IST]**
- **L[IST] *n***
- **L[IST] *m n***
- **R[UN]**
- ***n***
- ***n text***
- **0 *text***

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 편집 명령(계속)

명령	설명
I[NPUT]	임의 개수의 행을 삽입합니다.
I[NPUT] <i>text</i>	<i>text</i> 로 구성된 행을 삽입합니다.
L[IST]	SQL 버퍼에 있는 모든 행을 나열합니다.
L[IST] <i>n</i>	한 행(<i>n</i> 으로 지정된 행)을 나열합니다.
L[IST] <i>m n</i>	일정 범위(<i>m-n</i>)의 행을 나열합니다.
R[UN]	버퍼에 있는 현재 SQL 문을 표시하고 실행합니다.
<i>n</i>	<i>n</i> 행을 현재 행으로 지정합니다.
<i>n text</i>	<i>n</i> 행을 <i>text</i> 로 대체합니다.
0 <i>text</i>	1행 앞에 행을 삽입합니다.

참고: 각 SQL 프롬프트에서 하나의 SQL*Plus 명령만 입력할 수 있습니다. SQL*Plus 명령은 버퍼에 저장되지 않습니다. SQL*Plus 명령이 다음 행으로 이어지는 경우 첫 행의 끝에 하이픈(-)을 추가합니다.

LIST, n 및 APPEND 사용

```
LIST
1  SELECT last_name
2* FROM employees
```

```
1
1* SELECT last_name
```

```
A , job_id
1* SELECT last_name, job_id
```

```
LIST
1  SELECT last_name, job_id
2* FROM employees
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LIST, n 및 APPEND 사용

- L[IST] 명령을 사용하여 SQL 버퍼의 내용을 표시합니다. 버퍼에서 2행 옆의 별표(*)는 해당 행이 현재 행임을 나타냅니다. 편집한 내용은 현재 행에 적용됩니다.
- 편집하려는 행 번호(n)를 입력하여 현재 행의 번호를 바꿉니다. 새로운 현재 행이 표시됩니다.
- A[PPEND] 명령을 사용하여 현재 행에 텍스트를 추가합니다. 새로 편집한 행이 표시됩니다. LIST 명령을 사용하여 버퍼의 새 내용을 확인합니다.

참고: LIST 및 APPEND를 비롯한 대부분의 SQL*Plus 명령은 첫번째 문자만 사용하여 약어로 표기할 수 있습니다. LIST는 L, APPEND는 A라는 약어로 표기할 수 있습니다.

CHANGE 명령 사용

```
LIST
```

```
1* SELECT * from employees
```

```
c/employees/departments
```

```
1* SELECT * from departments
```

```
LIST
```

```
1* SELECT * from departments
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

CHANGE 명령 사용

- L[IST]를 사용하여 버퍼의 내용을 표시합니다.
- C[HANGE] 명령을 사용하여 SQL 버퍼에서 현재 행의 내용을 변경합니다. 이 경우 employees 테이블을 departments 테이블로 대체합니다. 새로운 현재 행이 표시됩니다.
- L[IST] 명령을 사용하여 버퍼의 새 내용을 확인합니다.

SQL*Plus 파일 명령

- **SAVE filename**
- **GET filename**
- **START filename**
- **@ filename**
- **EDIT filename**
- **SPOOL filename**
- **EXIT**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus 파일 명령

SQL 문은 Oracle 서버와 통신합니다. SQL*Plus 명령은 환경을 제어하고 query 결과의 서식을 지정하고 파일을 관리합니다. 다음 표에 설명된 명령을 사용할 수 있습니다.

명령	설명
SAV[E] <i>filename</i> [.ext] [REP[LACE]APP[END]]	SQL 버퍼의 현재 내용을 파일에 저장합니다. 기존 파일에 추가하려면 APPEND를 사용하고 기존 파일을 덮어 쓰려면 REPLACE를 사용합니다. 기본 확장자는 .sql입니다.
GET <i>filename</i> [.ext]	이전에 저장한 파일의 내용을 SQL 버퍼에 씁니다. 파일 이름의 기본 확장자는 sql입니다.
STA[RT] <i>filename</i> [.ext]	이전에 저장한 명령 파일을 실행합니다.
@ <i>filename</i>	이전에 저장한 명령 파일을 실행합니다(START와 동일).
ED[IT]	편집기를 호출하여 버퍼 내용을 afiedt.buf라는 파일에 저장합니다.
ED[IT] [<i>filename</i> [.ext]]	편집기를 호출하여 저장된 파일의 내용을 편집합니다.
SPO[OL] [<i>filename</i> [.ext]] OFF OUT	Query 결과를 파일에 저장합니다. OFF는 스풀 파일을 닫습니다. OUT은 스풀 파일을 닫고 파일 결과를 프린터로 보냅니다.
EXIT	SQL*Plus를 종료합니다.

SAVE 및 START 명령 사용

LIST

```
1  SELECT last_name, manager_id, department_id
2* FROM employees
```

SAVE my_query

Created file my_query

START my_query

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
107 rows selected.		

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SAVE 및 START 명령 사용

SAVE

SAVE 명령을 사용하여 버퍼의 현재 내용을 파일에 저장합니다. 이와 같은 방법으로 자주 사용되는 스크립트를 나중에 사용할 수 있도록 저장할 수 있습니다.

START

START 명령을 사용하여 SQL*Plus에서 스크립트를 실행합니다. 또는 기호 @을 사용하여 스크립트를 실행할 수도 있습니다.

`@my_query`

SERVEROUTPUT 명령

- SET SERVEROUT[PUT] 명령을 사용하여 내장 프로시저 또는 PL/SQL 블록의 출력을 SQL*Plus에 표시할지 여부를 제어할 수 있습니다.
- DBMS_OUTPUT 행 길이 제한이 255바이트에서 32767바이트로 늘어났습니다.
- 이제 기본 크기에 제한이 없습니다.
- SERVEROUTPUT를 설정하는 경우 리소스가 미리 할당되지 않습니다.
- Physical memory를 절약하려는 경우가 아니라면 UNLIMITED를 사용하십시오. 그래도 성능 저하가 발생하지 않습니다.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNLIMITED}]  
[FOR[MAT] {WRA[PPED] | WOR[D_WWRAPPED] | TRU[NCATED]}]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SERVEROUTPUT 명령

대부분의 PL/SQL 프로그램은 SQL 문을 통해 입/출력을 수행하여 데이터베이스 테이블에 데이터를 저장하거나 데이터베이스 테이블을 query합니다. 다른 모든 PL/SQL 입/출력은 다른 프로그램과 상호 작용하는 API를 통해 수행됩니다. 예를 들어, DBMS_OUTPUT 패키지에는 PUT_LINE과 같은 프로시저가 포함되어 있습니다. PL/SQL 외부의 결과를 보려면 DBMS_OUTPUT에 전달된 데이터를 읽고 표시하기 위한 SQL*Plus 등의 다른 프로그램이 필요합니다.

SQL*Plus에서 DBMS_OUTPUT 데이터를 표시하려면 먼저 다음과 같이 SQL*Plus 명령 SET SERVEROUTPUT ON을 실행해야 합니다.

```
SET SERVEROUTPUT ON
```

참고

- SIZE는 오라클 데이터베이스 서버 내에서 버퍼될 수 있는 출력 크기를 바이트 수로 나타낸 값입니다. 기본값은 UNLIMITED입니다. N은 2000보다 작거나 1,000,000보다 클 수 없습니다.
- SERVEROUTPUT에 대한 자세한 내용은 *Oracle Database PL/SQL User's Guide and Reference 11g*를 참조하십시오.

SQL*Plus SPOOL 명령 사용

```
SPO[OL] [file_name[.ext]] [CRE[A TE] | REP[L ACE] |
APP[END]] | OFF | OUT]
```

옵션	설명
file_name[.ext]	출력을 지정된 파일 이름으로 스팔합니다.
CRE[A TE]	지정한 이름으로 새 파일을 생성합니다.
REP[L ACE]	기존 파일의 내용을 바꿉니다. 파일이 존재하지 않을 경우 REPLACE를 사용하면 파일이 생성됩니다.
APP[END]	지정한 파일의 끝에 버퍼의 내용을 추가합니다.
OFF	스풀을 정지합니다.
OUT	스풀 작업을 정지하고 파일을 컴퓨터의 표준(기본) 프린터로 보냅니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus SPOOL 명령 사용

SPOOL 명령은 query 결과를 파일에 저장하거나 선택적으로 파일을 프린터로 보냅니다. SPOOL 명령이 향상되었습니다. 이전에는 SPOOL 명령을 사용하여 파일을 생성하거나 바꿀 수만 있었지만 이제는 기존 파일을 바꾸는 것뿐 아니라 기존 파일에 내용을 추가할 수도 있습니다. 기본값은 REPLACE입니다.

스크립트의 명령에 의해 생성된 출력을 화면에 표시하지 않고 스팔하려면 SET TERMOUT OFF를 사용합니다. SET TERMOUT OFF는 대화식으로 실행하는 명령의 출력에는 영향을 주지 않습니다.

공백이 포함된 파일 이름은 따옴표로 묶어야 합니다. SPOOL APPEND 명령을 사용하여 유효한 HTML 명령을 생성하려면 PROMPT 또는 유사한 명령을 사용하여 HTML 페이지 header와 footer를 생성해야 합니다. SPOOL APPEND 명령은 HTML 태그 구문을 분석하지 않습니다. CREATE, APPEND 및 SAVE 파라미터를 비활성화하려면 SQLPLUSCOMPAT[IBILITY]를 9.2 이하로 설정합니다.

AUTOTRACE 명령 사용

- **SELECT, INSERT, UPDATE, DELETE 등의 SQL DML(데이터 조작문) 문을 성공적으로 실행한 후에 보고서를 표시합니다.**
- **이제 보고서에 실행 통계 및 query 실행 경로가 포함될 수 있습니다.**

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]  
[STATISTICS]
```

```
SET AUTOTRACE ON  
-- The AUTOTRACE report includes both the optimizer  
-- execution path and the SQL statement execution  
-- statistics
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

AUTOTRACE 명령 사용

EXPLAIN은 EXPLAIN PLAN을 수행하여 query 실행 경로를 보여줍니다. STATISTICS는 SQL 문 통계를 표시합니다. AUTOTRACE 보고서의 형식은 연결되어 있는 서버의 버전과 서버의 구성에 따라 달라질 수 있습니다. DBMS_XPLAN 패키지를 사용하면 EXPLAIN PLAN 명령의 출력을 미리 정의된 여러 형식으로 간단히 표시할 수 있습니다.

참고

- 패키지 및 서브 프로그램에 대한 자세한 내용은 *Oracle Database PL/SQL Packages and Types Reference 11g* 설명서를 참조하십시오.
- EXPLAIN PLAN에 대한 자세한 내용은 *Oracle Database SQL Reference 11g*를 참조하십시오.
- 실행 계획 및 통계에 대한 자세한 내용은 *Oracle Database Performance Tuning Guide 11g*를 참조하십시오.

요약

이 부록에서는 다음 작업을 수행하기 위한 환경으로 SQL*Plus를 사용하는 방법을 배웠습니다.

- SQL 문 실행
- SQL 문 편집
- 출력 형식 지정
- 스크립트 파일과 상호 작용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

SQL*Plus는 SQL 명령을 데이터베이스 서버로 보내고 SQL 명령을 편집 및 저장하는 데 사용할 수 있는 실행 환경입니다. SQL 프롬프트 또는 스크립트 파일에서 명령을 실행할 수 있습니다.

JDeveloper 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- Oracle JDeveloper의 주요 기능 나열
- JDeveloper에서 데이터베이스 연결 생성
- JDeveloper에서 데이터베이스 객체 관리
- JDeveloper를 사용하여 SQL 명령 실행
- PL/SQL 프로그램 단위 생성 및 실행

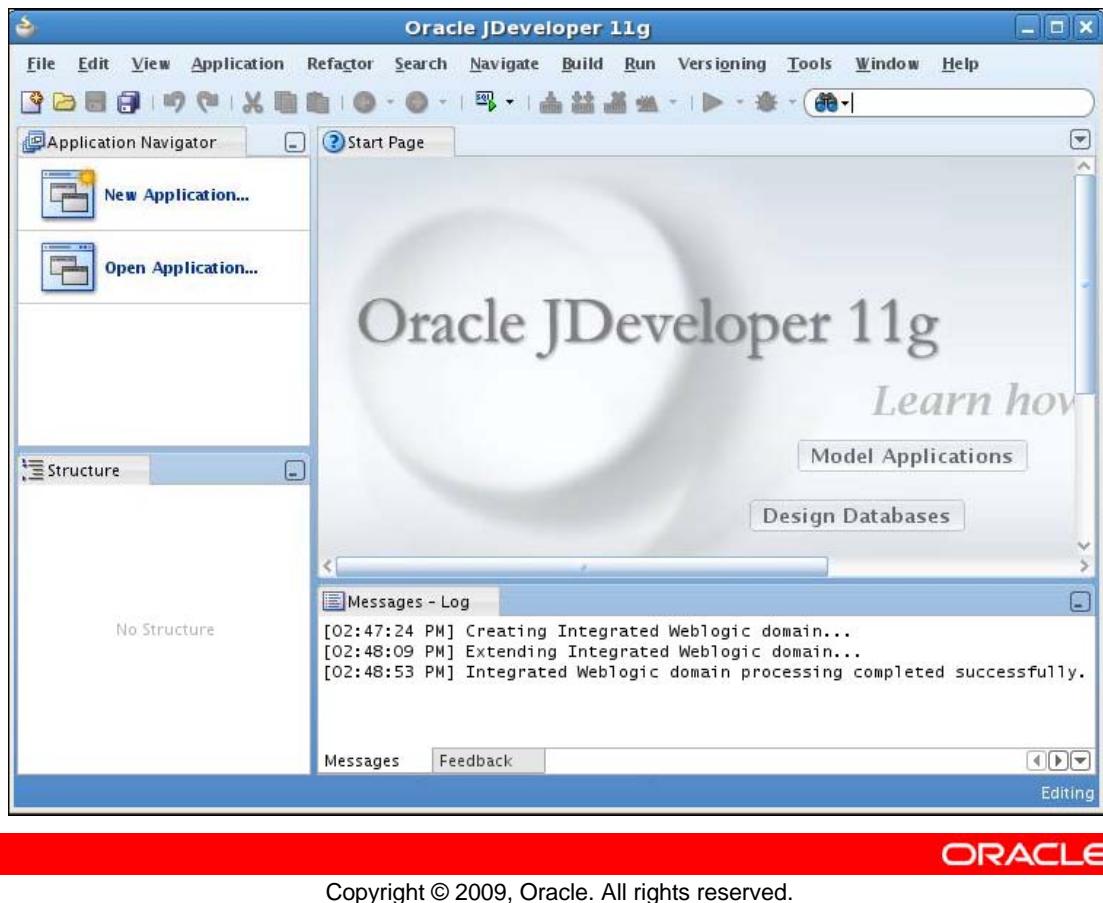
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 JDeveloper 도구를 소개합니다. 먼저, 데이터베이스 개발 작업에 JDeveloper를 사용하는 방법에 대해 알아봅니다.

Oracle JDeveloper

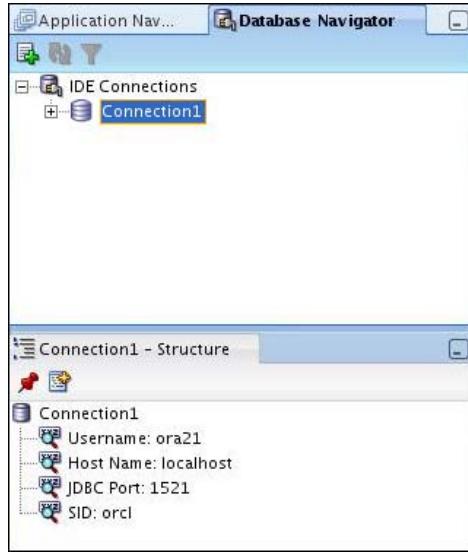


Oracle JDeveloper

Oracle JDeveloper는 Java 응용 프로그램과 웹 서비스를 개발하여 배치할 수 있는 IDE(통합 개발 환경)입니다. 모델링에서 배치까지 전단계의 SDLC(소프트웨어 개발 주기)를 지원합니다. 이 도구에는 응용 프로그램 개발 시 Java, XML 및 SQL에 대한 최신 산업 표준을 사용하는 기능이 있습니다.

Oracle JDeveloper 11g는 시각적/선언적 개발을 지원하는 기능으로 J2EE 개발에 대한 새로운 접근법을 시도합니다. 이 혁신적 접근은 J2EE 개발을 간단하고 효율적으로 만듭니다.

Database Navigator



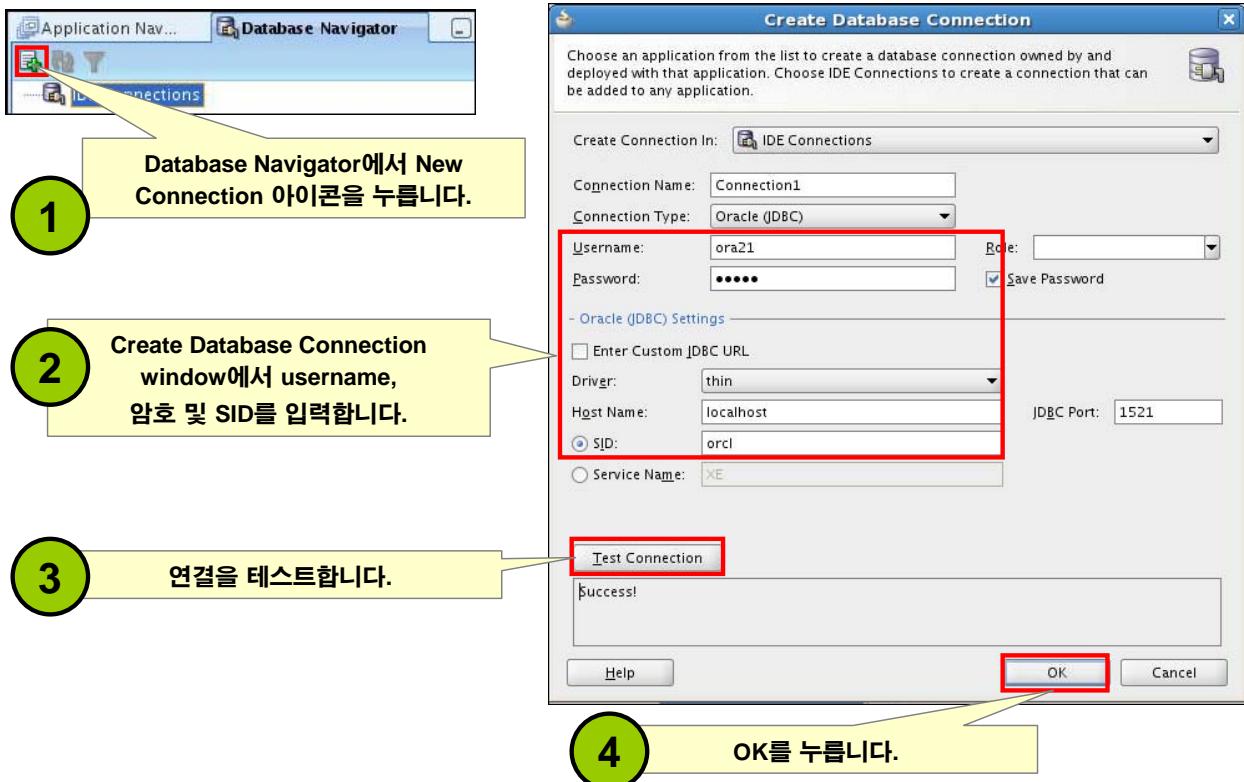
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Database Navigator

Oracle JDeveloper를 사용하면 데이터베이스에 연결하는 데 필요한 정보를 "connection"이라는 객체에 저장할 수 있습니다. 연결은 IDE 설정의 일부로 저장되며 유저 그룹 간에 공유하기 쉽도록 엑스포트하고 임포트할 수 있습니다. 또한 데이터베이스를 찾고 응용 프로그램을 구축하는 것에서부터 배치에 이르기까지 다양한 용도로 사용됩니다.

연결 생성



ORACLE

Copyright © 2009, Oracle. All rights reserved.

연결 생성

연결은 특정 데이터베이스에 해당 데이터베이스의 특정 유저로 연결하는 데 필요한 정보를 지정하는 객체입니다. 여러 데이터베이스 및 여러 스키마에 대해 연결을 생성하고 테스트할 수 있습니다.

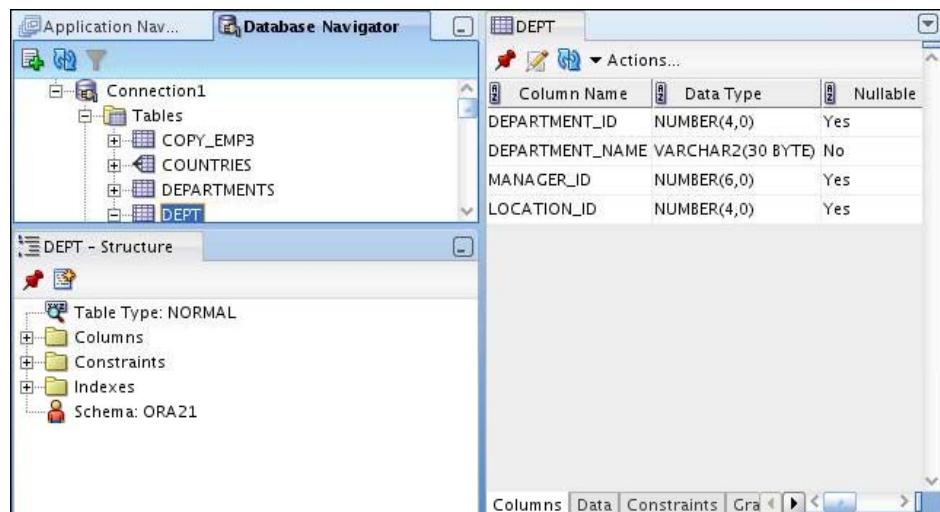
데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. Database Navigator에서 New Connection 아이콘을 누릅니다.
2. Create Database Connection window에 연결 이름을 입력합니다. 연결하려는 스키마의 username 및 암호를 입력합니다. 연결할 데이터베이스의 SID를 입력합니다.
3. Test를 눌러 연결이 올바르게 설정되었는지 확인합니다.
4. OK를 누릅니다.

데이터베이스 객체 탐색

Database Navigator를 사용하여 다음을 수행할 수 있습니다.

- 데이터베이스 스키마에서 여러 객체 탐색
- 여러 객체의 정의를 한 번에 검토



ORACLE

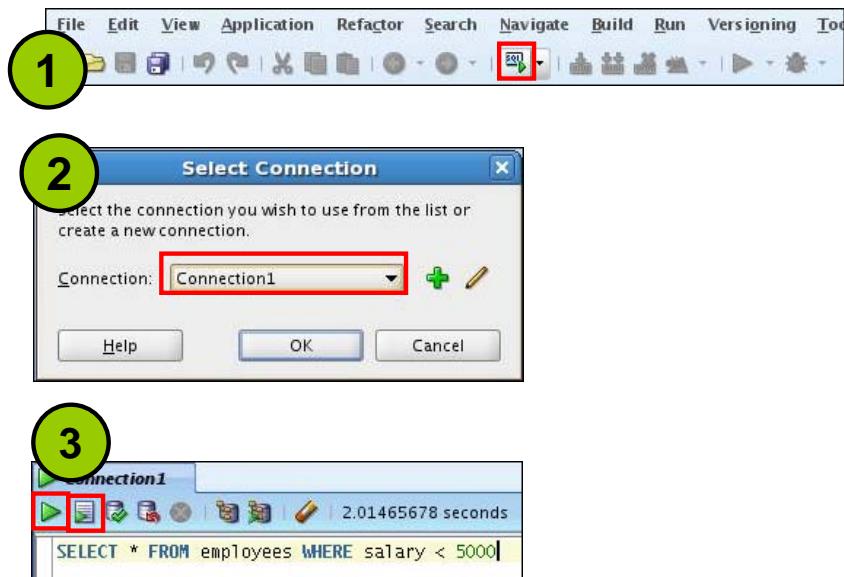
Copyright © 2009, Oracle. All rights reserved.

데이터베이스 객체 탐색

데이터베이스 연결을 생성한 후 Database Navigator를 사용하여 데이터베이스 스키마에서 테이블, 뷰, 인덱스, 패키지, 프로시저, 트리거, 유형 등의 여러 객체를 탐색할 수 있습니다.

데이터 딕셔너리에서 추출된 객체 정의가 여러 정보 탭에 분할되어 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 읽기 쉬운 탭 페이지에 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 세부 정보가 표시됩니다.

SQL 문 실행



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

SQL 문 실행

SQL 문을 실행하려면 다음 단계를 수행하십시오.

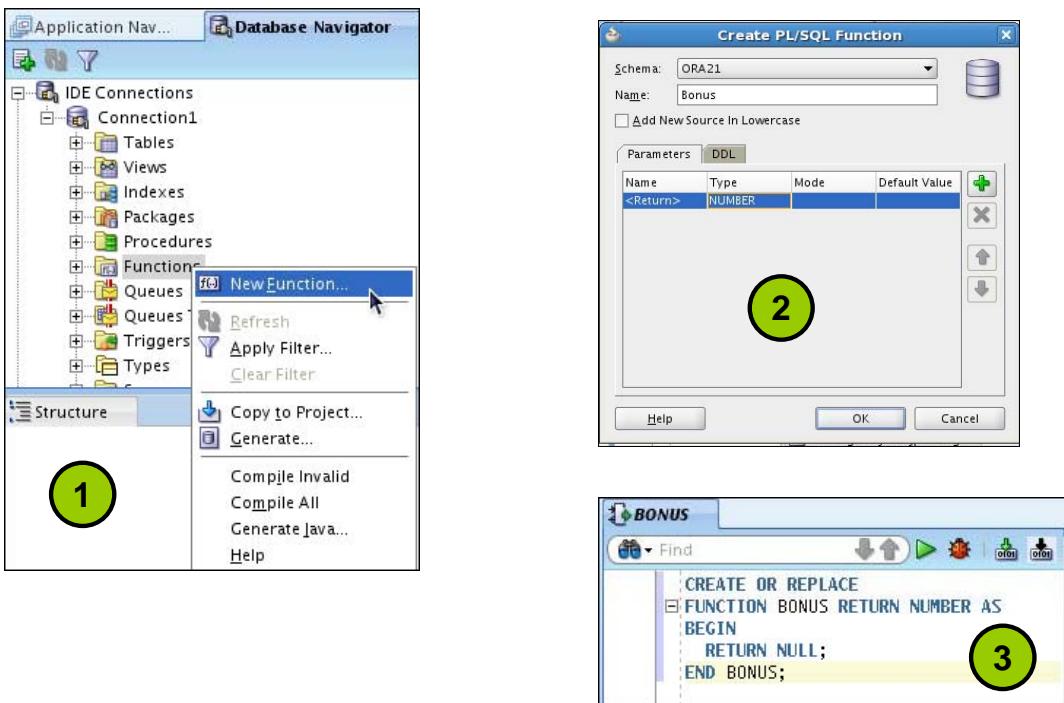
1. Open SQL Worksheet 아이콘을 누릅니다.
2. 연결을 선택합니다.
3. 다음을 눌러 SQL 명령을 실행합니다.
 - **Execute statement** 버튼 또는 F9 키를 누릅니다. 출력 결과는 다음과 같습니다.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME
1	100	Steven	King
2	101	Neena	Kochhar

- **Run Script** 버튼 또는 F5 키를 누릅니다. 출력 결과는 다음과 같습니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King

프로그램 단위 생성



함수의 기본 구조

ORACLE

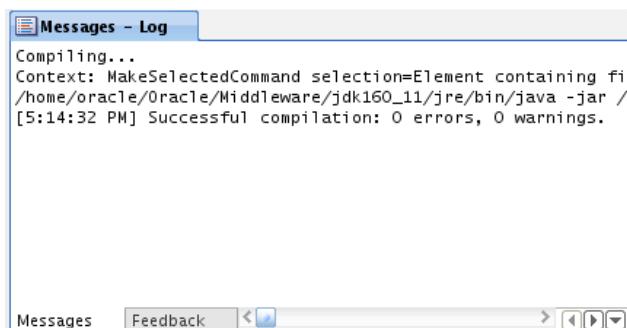
Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 생성

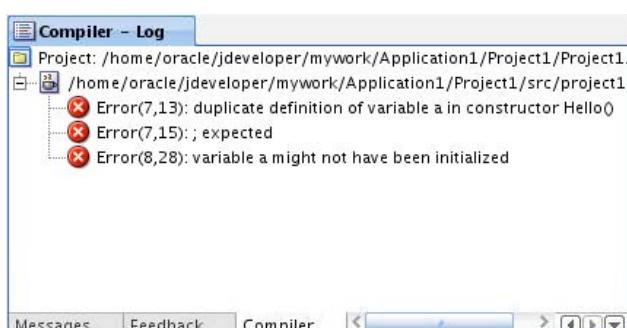
PL/SQL 프로그램 단위를 생성하려면 다음을 수행하십시오.

1. View > Database Navigator를 선택합니다. 데이터베이스 연결을 선택하고 확장합니다. 객체 유형에 해당하는 폴더(Procedures, Packages, Functions)를 마우스 오른쪽 버튼으로 누릅니다. "New [Procedures|Packages|Functions]"를 선택합니다.
2. 함수, 패키지 또는 프로시저에 대한 유효한 이름을 입력하고 OK를 누릅니다.
3. 기본 구조 정의가 생성되어 Code Editor에서 열립니다. 그런 다음 요구에 맞게 서브 프로그램을 편집할 수 있습니다.

컴파일



오류가 있는 컴파일



오류가 없는 컴파일

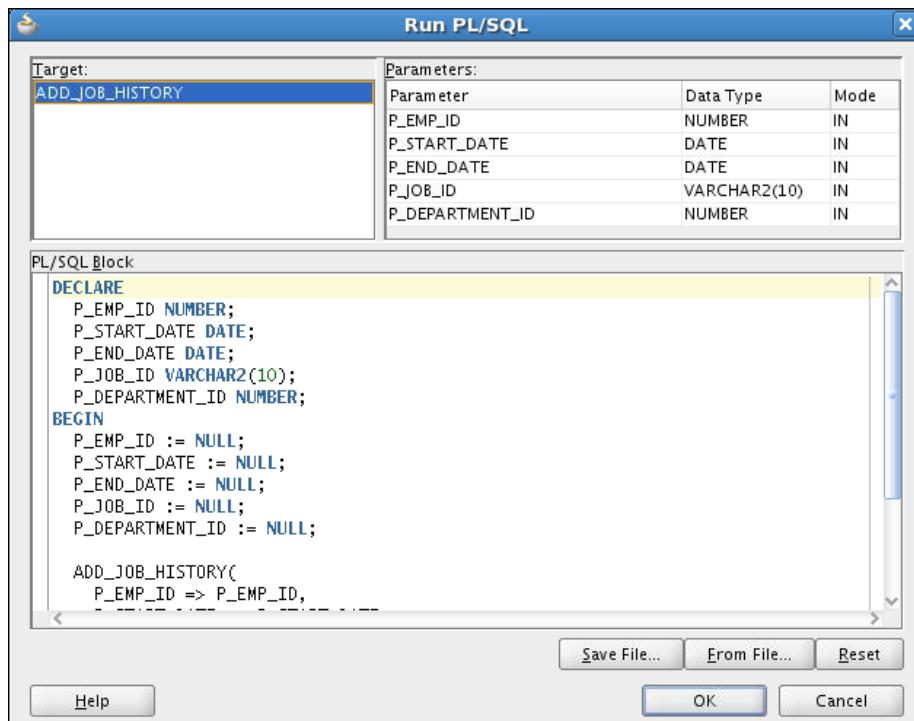
ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일

기본 구조 정의를 편집한 후 프로그램 단위를 컴파일해야 합니다. Connection Navigator에서 컴파일해야 하는 PL/SQL 객체를 마우스 오른쪽 버튼으로 누르고 Compile을 선택합니다. 또는 Ctrl + Shift + F9를 눌러 컴파일해도 됩니다.

프로그램 단위 실행



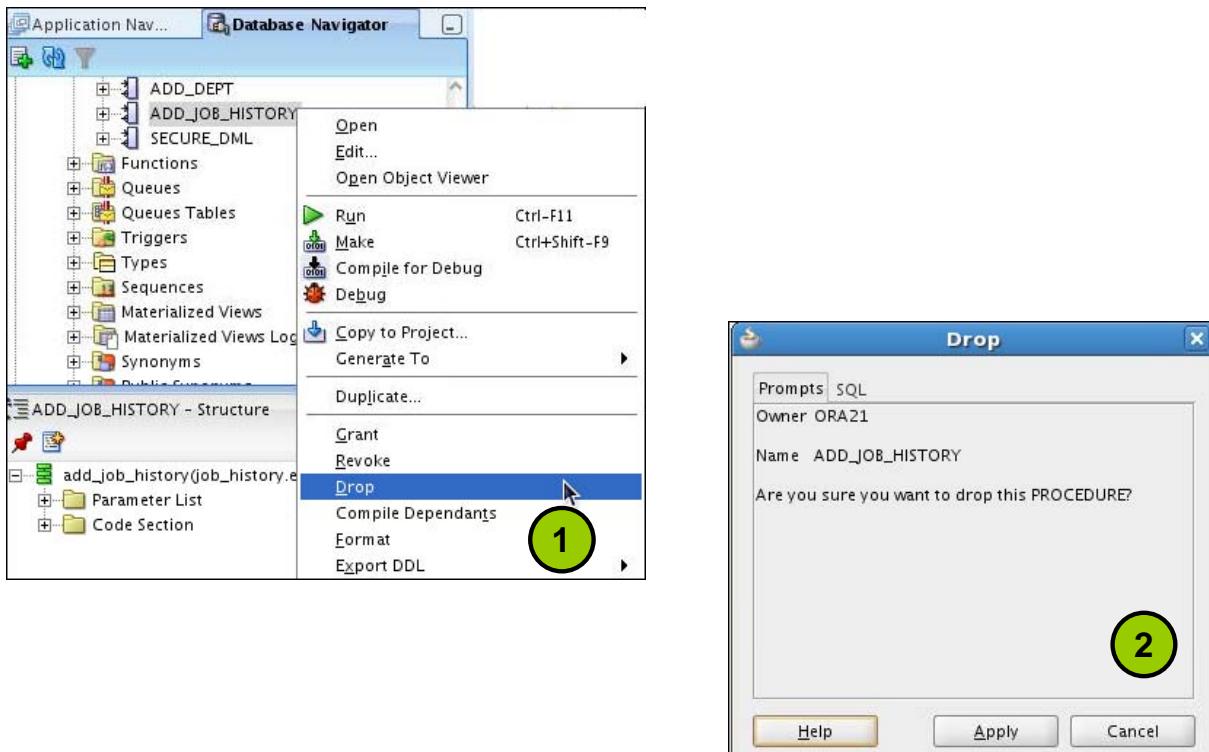
ORACLE®

Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 실행

프로그램 단위를 실행하려면 객체를 마우스 오른쪽 버튼으로 누르고 Run을 선택합니다. Run PL/SQL 대화상자가 나타납니다. NULL 값을 프로그램 단위로 전달하기에 적절한 값으로 변경해야 할 수도 있습니다. 해당 값을 변경한 후 OK를 누릅니다. Message-Log window에 출력 결과가 표시됩니다.

프로그램 단위 삭제



ORACLE®

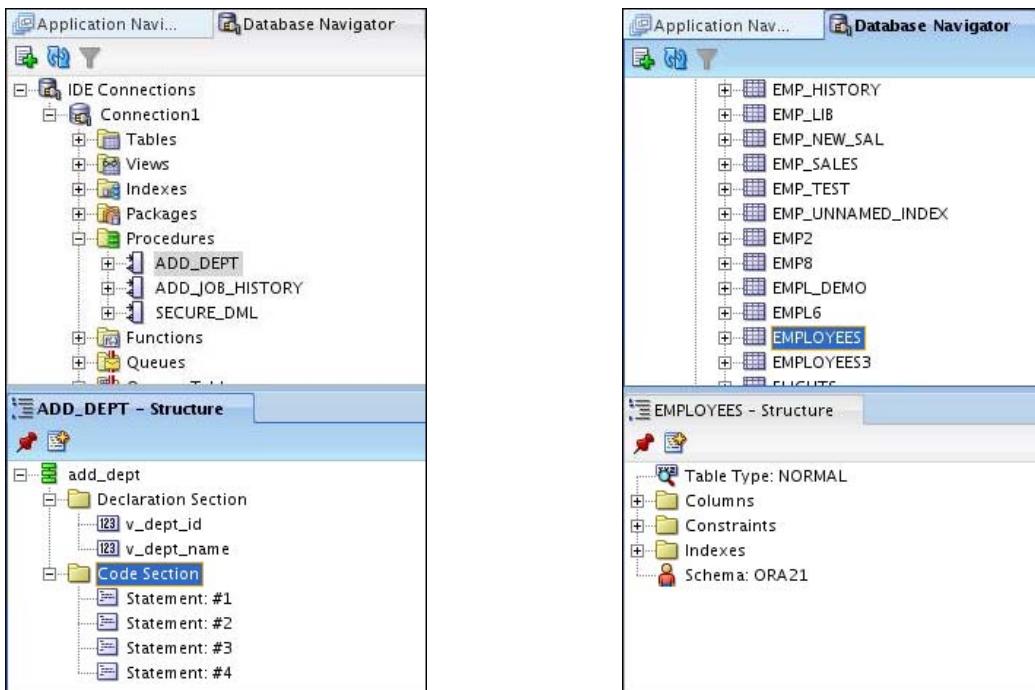
Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 삭제

프로그램 단위를 삭제하려면 다음을 수행하십시오.

1. 객체를 마우스 오른쪽 버튼으로 누르고 Drop을 선택합니다. Drop Confirmation 대화상자가 나타납니다.
2. Apply를 누릅니다. 객체가 데이터베이스에서 삭제됩니다.

Structure window



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Structure window

Structure window는 구조를 제공하는 데 관여하는 Navigator, Editor, Viewer, Property Inspector와 같은 window 중 활성 window에 현재 선택된 문서 데이터의 구조적 뷰를 제공합니다.

Structure window에서 다양한 방식으로 문서 데이터를 볼 수 있습니다. 표시할 수 있는 구조는 문서 유형에 따라 결정됩니다. Java 파일의 경우 코드 구조, UI 구조 또는 UI 모델 데이터를 볼 수 있습니다. XML 파일의 경우 XML 구조, 디자인 구조 또는 UI 모델 데이터를 볼 수 있습니다.

Structure window는 현재의 활성 Editor와 연관되어 특정 뷰에 window 내용을 고정하지 않는 한 항상 동적으로 활성 window의 현재 선택 사항을 추적합니다. 현재 선택 사항이 Navigator의 노드일 때를 기본 편집기로 간주합니다. 현재 선택 사항의 구조에 대한 뷰를 변경하려면 다른 구조 탭을 선택합니다.

Editor window



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

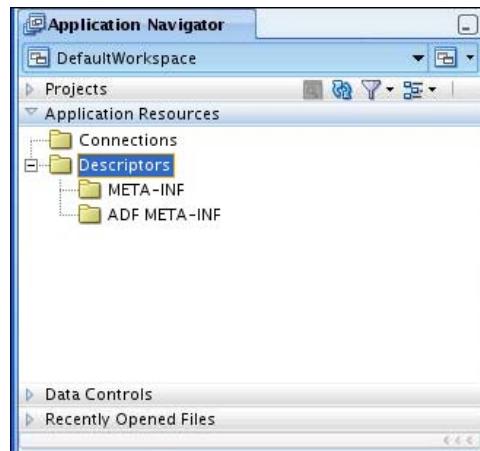
Editor window

단일 Editor window에서 프로젝트 파일을 모두 보거나, 한 파일을 여러 뷰로 열거나, 다양한 파일을 여러 뷰로 열 수 있습니다.

편집기 window의 상단에 있는 텁은 문서 텁입니다. 문서 텁을 선택하면 해당 파일을 현재 편집기의 window 맨 앞으로 가져와서 해당 파일에 집중시킵니다.

주어진 파일에 대해 편집기 window의 하단에 있는 탭은 편집기 탭입니다. 편집기 탭을 선택하면 해당 편집기에서 파일이 열립니다.

Application Navigator



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

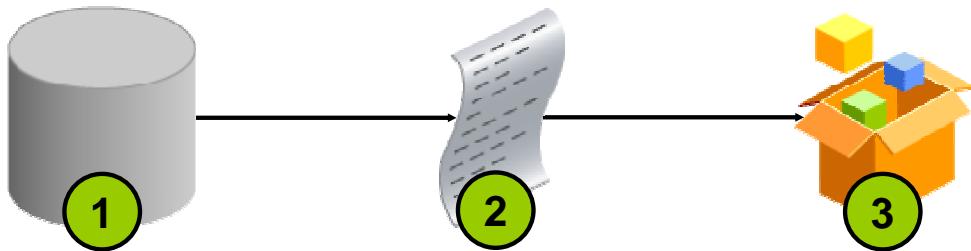
Application Navigator

Application Navigator는 응용 프로그램 및 이 응용 프로그램이 포함하는 데이터의 논리적 뷰를 제공합니다. Application Navigator는 다양한 확장을 플러그인하여 일관된 요약 방식으로 해당 데이터 및 메뉴를 구성하는 데 사용할 수 있는 Infrastructure를 제공합니다. Application Navigator는 Java 소스 파일과 같은 개별 파일을 포함할 수 있으며 이때 복잡한 데이터를 통합합니다. 엔티티 객체, UML (Unified Modeling Language) 다이어그램, EJB (Enterprise JavaBeans) 또는 웹 서비스와 같은 복잡한 데이터 유형이 Navigator에 단일 노드로 표시됩니다. 이러한 요약 노드를 구성하는 Raw File은 Structure window에 나타납니다.

Java 내장 프로시저 배치

Java 내장 프로시저를 배치하기 전에 다음 단계를 수행하십시오.

1. 데이터베이스 접속을 생성합니다.
2. 배치 프로파일을 생성합니다.
3. 객체를 배치합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Java 내장 프로시저 배치

Java 내장 프로시저에 대한 배치 프로파일을 생성한 다음 프로파일의 설정을 사용하여 JDeveloper에서 클래스를 배치하고 선택적으로 공용(public) 정적 메소드를 배치합니다.

데이터베이스에 배치할 때에는 Deployment Profile Wizard에 입력한 정보와 다음 두 개의 오라클 데이터베이스 유ти리티를 사용합니다.

- loadjava는 내장 프로시저를 포함하는 Java 클래스를 오라클 데이터베이스에 로드합니다.
- publish는 로드된 공용(public) 정적 메소드에 대해 PL/SQL 호출 특정 래퍼를 생성합니다. 게시(publishing) 후에는 Java 메소드를 PL/SQL 함수 또는 프로시저로 호출할 수 있습니다.

PL/SQL에 Java 게시(publishing)

The screenshot shows two windows side-by-side. The left window, titled 'TrimLob.java', contains Java code for a class named TrimLob with a main method. The right window, titled 'TRIMLOBPROC', contains PL/SQL code for creating or replacing a procedure named TRIMLOBPROC. A green circle labeled '1' is over the Java code area, and a green circle labeled '2' is over the PL/SQL code area.

```
public class TrimLob {  
    public static void main (String args [] ) throws SQLException {  
        Connection conn=null;  
        if (System.getProperty("oracle.jserver.version") != null)  
        {  
            conn = DriverManager.getConnection("jdbc:default:connection:");  
        }  
        else  
        {  
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());  
            conn = DriverManager.getConnection("jdbc:oracle:thin:scott/tiger");  
        }  
    }  
}  
  
CREATE OR REPLACE PROCEDURE TRIMLOBPROC  
as language java  
name 'TrimLob.main(java.lang.String[])';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에 Java 게시(publishing)

위 슬라이드는 Java 코드 및/oracle Java 코드를 PL/SQL 프로시저에 게시(publish)하는 방법을 보여줍니다.

JDeveloper 11g에 대해 자세히 배울 수 있는 방법

항목	웹 사이트
Oracle JDeveloper 제품 페이지	http://www.oracle.com/technology/products/jdev/index.html
Oracle JDeveloper 11g 학습서	http://www.oracle.com/technology/obe/obe11jdev/11/index.html
Oracle JDeveloper 11g 제품 설명서	http://www.oracle.com/technology/documentation/jdev.html
Oracle JDeveloper 11g 토의 포럼	http://forums.oracle.com/forums/forum.jspa?forumID=83

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 부록에서는 JDeveloper를 사용하여 다음 작업을 수행하는 방법을 배웠습니다.

- Oracle JDeveloper의 주요 기능 나열
- JDeveloper에서 데이터베이스 연결 생성
- JDeveloper에서 데이터베이스 객체 관리
- JDeveloper를 사용하여 SQL 명령 실행
- PL/SQL 프로그램 단위 생성 및 실행

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 JDeveloper 도구를 소개합니다. 먼저, 데이터베이스 개발 작업에 JDeveloper를 사용하는 방법에 대해 알아봅니다.

PL/SQL 검토

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- 익명 PL/SQL 블록의 블록 구조 검토
- PL/SQL 변수 선언
- PL/SQL 레코드 및 테이블 생성
- 데이터 삽입, 갱신 및 삭제
- IF, THEN 및 ELSIF 문 사용
- 기본 FOR 및 WHILE 루프 사용
- 파라미터가 포함된 커서 명시적 선언 및 사용
- 커서 FOR 루프 및 FOR UPDATE 및 WHERE CURRENT OF 절 사용
- 미리 정의된 예외 및 유저 정의 예외 트랩

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

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

익명 PL/SQL 블록의 블록 구조

- **DECLARE** (선택 사항)
 - 이 블록 내에서 사용될 PL/SQL 객체를 선언합니다.
- **BEGIN** (필수)
 - 실행문을 정의합니다.
- **EXCEPTION** (선택 사항)
 - 오류나 예외가 발생할 경우에 수행되는 작업을 정의합니다.
- **END ;** (필수)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

익명 블록

익명 블록은 이름이 없습니다. 익명 블록은 응용 프로그램에서 블록이 실행될 지점에 선언하십시오. 그러면 런타임에 실행을 위해 PL/SQL 엔진에 전달됩니다.

- DECLARE 키워드와 BEGIN 키워드 사이에 있는 섹션을 선언 섹션(Declare Section)이라고 합니다. 선언 섹션에는 블록 내에서 참조하고자 하는 변수, 상수, 커서 및 유저 정의 예외와 같은 PL/SQL 객체를 정의합니다. PL/SQL 객체를 선언하지 않는 경우 DECLARE 키워드는 선택 사항입니다.
- BEGIN 및 END 키워드는 필수이며 수행될 작업의 본문을 묶는 데 사용됩니다. 이 섹션을 블록의 실행 섹션(Executable Section)이라고 합니다.
- EXCEPTION과 END 사이에 있는 섹션을 예외 섹션(Exception Section)이라고 합니다. 예외 섹션은 오류 조건을 트랩합니다. 이 섹션에는 지정된 조건이 발생할 경우에 취할 조치를 정의합니다. 예외 섹션은 선택 사항입니다.

DECLARE, BEGIN 및 EXCEPTION 키워드 뒤에는 세미콜론이 오지 않지만 END를 비롯한 다른 모든 PL/SQL 문에는 세미콜론을 사용해야 합니다.

PL/SQL 변수 선언

- 구문:

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

- 예제:

```
Declare
    v_hiredate      DATE;
    v_deptno        NUMBER(2) NOT NULL := 10;
    v_location       VARCHAR2(13) := 'Atlanta';
    c_comm           CONSTANT NUMBER := 1400;
    v_count          BINARY_INTEGER := 0;
    v_valid          BOOLEAN NOT NULL := TRUE;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 변수 선언

PL/SQL 블록 내에서 모든 PL/SQL 식별자를 참조하려면 먼저 선언 섹션 내에 이들을 선언해야 합니다. 초기값을 할당할 수도 있습니다. 선언하려는 변수에는 값을 할당할 필요가 없습니다. 선언 섹션에서 다른 변수를 참조할 경우 이 변수가 이전 명령문에 별도로 선언되어 있어야 합니다. 이 구문에서 다음이 적용됩니다.

*Identifier*는 변수 이름입니다.

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

Datatype은 스칼라, 조합, 참조 또는 LOB 데이터 유형이 있습니다. (본 과정에서는 스칼라 및 조합 데이터 유형만 다룹니다.)

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

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

%TYPE 속성을 사용하여 변수 선언: 예제

```
...
  v_ename          employees.last_name%TYPE;
  v_balance        NUMBER(7,2);
  v_min_balance   v_balance%TYPE := 10;
...
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

%TYPE 속성을 사용하여 변수 선언

사원의 이름을 저장하도록 변수를 선언합니다.

```
...
  v_ename          employees.last_name%TYPE;
```

...

은행 계좌 잔액과 10으로 시작되는 최소 잔액을 저장하도록 변수를 선언합니다.

```
...
  v_balance        NUMBER(7,2);
  v_min_balance   v_balance%TYPE := 10;
...
```

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

PL/SQL 레코드 생성

새로운 사원의 이름, 직무, 급여를 저장하도록 변수를 선언합니다.

```
...
TYPE emp_record_type IS RECORD
(ename      VARCHAR2(25),
 job        VARCHAR2(10),
 sal         NUMBER(8,2));
emp_record  emp_record_type;
...
```

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 레코드 생성

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

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

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

참고: 필드 선언에 NOT NULL 제약 조건을 추가할 수 있으므로 해당 필드에 널이 할당되는 것을 방지할 수 있습니다. NOT NULL로 선언되는 필드는 반드시 초기화해야 합니다.

%ROWTYPE 속성: 예제

- DEPARTMENTS 테이블에 저장된 것과 동일한 부서 정보를 저장하도록 변수를 선언합니다.

```
dept_record    departments%ROWTYPE;
```

- EMPLOYEES 테이블에 저장된 것과 동일한 사원 정보를 저장하도록 변수를 선언합니다.

```
emp_record    employees%ROWTYPE;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

예제

위 슬라이드에 표시된 첫번째 선언은 DEPARTMENTS 테이블의 행과 동일한 필드 이름과 필드 데이터 유형을 가진 레코드를 생성합니다. 필드는 DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID 및 LOCATION_ID입니다.

위 슬라이드에 표시된 두번째 선언은 EMPLOYEES 테이블의 행과 동일한 필드 이름 및 필드 데이터 유형을 가진 레코드를 생성합니다. 필드는 EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID 및 DEPARTMENT_ID입니다.

다음 예제에서는 job_record 레코드에 대한 열 값을 선택합니다.

```
DECLARE
    job_record    jobs%ROWTYPE;
    ...
BEGIN
    SELECT * INTO job_record
    FROM   jobs
    WHERE   ...
```

PL/SQL 테이블 생성

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL 테이블 생성

스칼라 변수의 경우는 미리 정의된 데이터 유형이 있지만, PL/SQL 테이블의 경우는 미리 정의된 데이터 유형이 없습니다. 그러므로 데이터 유형을 먼저 생성한 다음 생성된 데이터 유형을 사용하여 식별자를 선언해야 합니다.

PL/SQL 테이블 참조

구문

```
pl/sql_table_name(primary_key_value)
```

이 구문에서 `primary_key_value`는 `BINARY_INTEGER` 유형에 속합니다.

PL/SQL 테이블 ENAME_TABLE의 세번째 행을 참조하십시오.

```
ename_table(3) ...
```

`BINARY_INTEGER`의 크기 범위는 -2,147,483,647 ~ 2,147,483,647입니다. 따라서 Primary Key 값은 음수일 수 있습니다. 인덱스는 1로 시작할 필요가 없습니다.

참고: 인덱스 `i`를 가진 행이 적어도 한 개 반환될 경우 `table.EXISTS(i)` 문은 TRUE를 반환합니다. 존재하지 않는 테이블 요소 참조 시 발생하는 오류를 방지하려면 EXISTS 문을 사용하십시오.

PL/SQL의 SELECT 문: 예제

INTO 절은 필수입니다.

```

DECLARE
    v_deptid NUMBER(4);
    v_loc NUMBER(4);
BEGIN
    SELECT department_id, location_id
    INTO v_deptid, v_loc
    FROM departments
    WHERE department_name = 'Sales';
    ...
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INTO 절

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

INTO 절을 사용하여 PL/SQL 변수나 호스트 변수를 채울 수 있습니다.

Query는 한 개의 행만 반환해야 함

PL/SQL 블록 내의 SELECT 문은 다음과 같은 규칙이 적용되는 내장 SQL의 ANSI 분류에 속합니다.

Query는 한 개의 행만 반환해야 합니다. 반환되는 행이 없거나 여러 개일 경우 오류가 발생합니다.

PL/SQL은 표준 예외를 발생시켜 이러한 오류를 처리하는데, 이 오류는 NO_DATA_FOUND 및 TOO_MANY_ROWS 예외를 사용하여 블록의 예외 섹션에서 트랩할 수 있습니다. 단일 행을 반환하려면 SELECT 문을 코딩해야 합니다.

데이터 삽입: 예제

EMPLOYEES 테이블에 새로운 사원 정보를 추가합니다.

```
DECLARE
    v_empid    employees.employee_id%TYPE;
BEGIN
    SELECT   employees_seq.NEXTVAL
    INTO      v_empno
    FROM      dual;
    INSERT INTO employees(employee_id, last_name,
                           job_id, department_id)
    VALUES(v_empid, 'HARDING', 'PU_CLERK', 30);
END;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 삽입

- USER 및 SYSDATE와 같은 SQL 함수를 사용합니다.
- 데이터베이스 시퀀스를 사용하여 Primary Key 값을 생성합니다.
- PL/SQL 블록에서 값을 파생시킵니다.
- 열 기본값을 추가합니다.

참고: INSERT 문에서는 식별자와 열 이름이 모호할 수가 없습니다. INSERT 절의 식별자는 데이터베이스 열 이름이어야 합니다.

데이터 갱신: 예제

EMPLOYEES 테이블에서 판매원인 모든 사원의 급여를 인상합니다.

```

DECLARE
    v_sal_increase    employees.salary%TYPE := 2000;
BEGIN
    UPDATE  employees
    SET      salary = salary + v_sal_increase
    WHERE    job_id = 'PU_CLERK';
END;

```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 갱신

UPDATE 문의 SET 절에 모호한 점이 있습니다. 할당 연산자 왼쪽에 있는 식별자는 항상 데이터베이스 열이지만 오른쪽에 있는 식별자는 데이터베이스 열일 수도 있고 PL/SQL 변수일 수도 있기 때문입니다.

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

참고: PL/SQL 변수 할당은 항상 `:=`를 사용하고, SQL 열 할당은 항상 `= .`을 사용합니다. WHERE 절에서 열 이름과 식별자 이름이 동일할 경우 Oracle 서버는 먼저 데이터베이스에서 이름을 검색합니다.

데이터 삭제: 예제

EMPLOYEES 테이블에서 부서 190에 속하는 행을 삭제합니다.

```
DECLARE
    v_deptid    employees.department_id%TYPE := 190;
BEGIN
    DELETE FROM employees
    WHERE department_id = v_deptid;
END;
```



Copyright © 2009, Oracle. All rights reserved.

데이터 삭제: 예제

특정 작업을 삭제합니다.

```
DECLARE
    v_jobid    jobs.job_id%TYPE := 'PR_REP';
BEGIN
    DELETE FROM jobs
    WHERE job_id = v_jobid;
END;
```

COMMIT 및 ROLLBACK 문을 사용하여 트랜잭션 제어

- 첫번째 DML 명령으로 트랜잭션을 시작하여 COMMIT 또는 ROLLBACK 문을 수행합니다.
- COMMIT 및 ROLLBACK SQL 문을 사용하여 트랜잭션을 명시적으로 종료합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

COMMIT 및 ROLLBACK 문을 사용하여 트랜잭션 제어

COMMIT 및 ROLLBACK SQL 문으로 트랜잭션 논리를 제어하여 일부 데이터베이스 변경 사항 그룹을 영구적으로 만들고 나머지 다른 그룹은 무시할 수 있습니다. Oracle 서버와 마찬가지로, DML(데이터 조작어) 트랜잭션은 첫번째 명령으로 시작되어 그 다음으로 COMMIT 또는 ROLLBACK이 수행된 다음 성공적인 COMMIT 또는 ROLLBACK으로 끝납니다. 이러한 작업은 PL/SQL 블록 내에서 수행되거나 호스트 환경에서 이벤트 결과로 수행될 수 있습니다. COMMIT은 보류 중인 모든 데이터베이스 변경 사항을 영구적으로 만들어 현재 트랜잭션을 종료합니다.

구문

```
COMMIT [ WORK ] ;
ROLLBACK [ WORK ] ;
```

이 구문에서 WORK는 ANSI 표준을 따르기 위한 것입니다.

참고: 호스트 환경에서 트랜잭션 제어 명령을 사용하는 데 약간의 제한이 있다고 하더라도 PL/SQL 내에서는 이러한 명령이 모두 유효합니다.

또한 블록에 명시적 잠금 명령(LOCK TABLE 및 SELECT ... FOR UPDATE)을 포함할 수도 있습니다. 이러한 명령은 트랜잭션이 끝날 때까지 유효합니다. 뿐만 아니라 하나의 PL/SQL 블록이 반드시 하나의 트랜잭션을 의미하는 것은 아닙니다.

IF, THEN 및 ELSIF 문: 예제

제공된 값을 입력할 경우 계산된 값이 반환됩니다.

```
.
.
.
IF v_start > 100 THEN
    v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
    v_start := 0.5 * v_start;
ELSE
    v_start := 0.1 * v_start;
END IF;
.
.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IF, THEN 및 ELSIF 문

가능하면 중첩 IF 문 대신 ELSIF 절을 사용하십시오. 그러면 코드를 보다 쉽게 읽고 이해할 수 있으며 논리도 명확하게 식별됩니다. ELSE 절의 동작이 순전히 다른 IF 문으로만 구성될 경우 ELSIF 절을 사용하는 것이 훨씬 편리합니다. 이렇게 하면 각각의 추가 조건 및 동작 집합 끝에 END IF를 중첩할 필요가 없어지므로 코드가 보다 명확해집니다.

예제

```
IF condition1 THEN
    statement1;
ELSIF condition2 THEN
    statement2;
ELSIF condition3 THEN
    statement3;
END IF;
```

위 슬라이드에 표시된 명령문은 다음과 같이 자세히 정의됩니다.

제공된 값을 입력할 경우 계산된 값이 반환됩니다. 입력한 값이 100보다 클 경우 계산된 값은 입력한 값의 두 배가 됩니다. 입력한 값이 50과 100 사이에 있을 경우 계산된 값은 시작 값의 50%가 됩니다. 입력한 값이 50 미만일 경우 계산된 값은 시작 값의 10%가 됩니다.

참고: 널 값을 포함하는 산술식은 널로 평가됩니다.

기본 루프: 예제

```
DECLARE
    v_ordid      order_items.order_id%TYPE := 101;
    v_counter    NUMBER(2) := 1;
BEGIN
    LOOP
        INSERT INTO order_items(order_id,line_item_id)
        VALUES(v_ordid, v_counter);
        v_counter := v_counter + 1;
        EXIT WHEN v_counter > 10;
    END LOOP;
END;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

기본 루프: 예제

위 슬라이드에 표시된 기본 루프 예제는 다음과 같이 정의됩니다.

주문 번호 101에 대해 처음 10개의 새 단일 품목을 삽입합니다.

참고: 기본 루프를 사용하면 루프 시작과 함께 조건이 충족되더라도 명령문을 적어도 한 번 실행할 수 있습니다.

FOR 루프: 예제

주문 번호 101에 대해 처음 10개의 새 단일 품목을 삽입합니다.

```
DECLARE
  v_ordid      order_items.order_id%TYPE := 101;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO order_items(order_id,line_item_id)
    VALUES(v_ordid, i);
  END LOOP;
END;
```



Copyright © 2009, Oracle. All rights reserved.

FOR 루프: 예제

위 슬라이드는 order_items 테이블에 10개의 행을 삽입하는 FOR 루프를 보여줍니다.

WHILE 루프: 예제

```
ACCEPT p_price PROMPT 'Enter the price of the item: '
ACCEPT p_itemtot -
  PROMPT 'Enter the maximum total for purchase of item: '
DECLARE
  ...
  v_qty          NUMBER(8) := 1;
  v_running_total NUMBER(7,2) := 0;

BEGIN
  ...
  WHILE v_running_total < &p_itemtot LOOP
    ...
    v_qty := v_qty + 1;
    v_running_total := v_qty * &p_price;
  END LOOP;
  ...

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WHILE 루프: 예제

위 슬라이드 예제에서는 수량이 해당 품목에 지출할 수 있는 최대 가격보다 같거나 높아질 때까지 루프가 반복될 때마다 수량이 늘어납니다.

SQL 암시적 커서 속성

SQL 커서 속성을 사용하면 SQL 문의 결과를 테스트할 수 있습니다.

SQL 커서 속성	설명
SQL%ROWCOUNT	가장 최근 SQL 문의 영향을 받는 행 수(정수 값)
SQL%FOUND	가장 최근 SQL 문이 여러 행에 영향을 줄 경우 TRUE로 평가되는 부울 속성
SQL%NOTFOUND	가장 최근 SQL 문이 어떤 행에도 영향을 주지 않을 경우 TRUE로 평가되는 부울 속성
SQL%ISOPEN	PL/SQL에서 암시적 커서가 실행된 후에 바로 닫히기 때문에 항상 FALSE로 평가되는 부울 속성

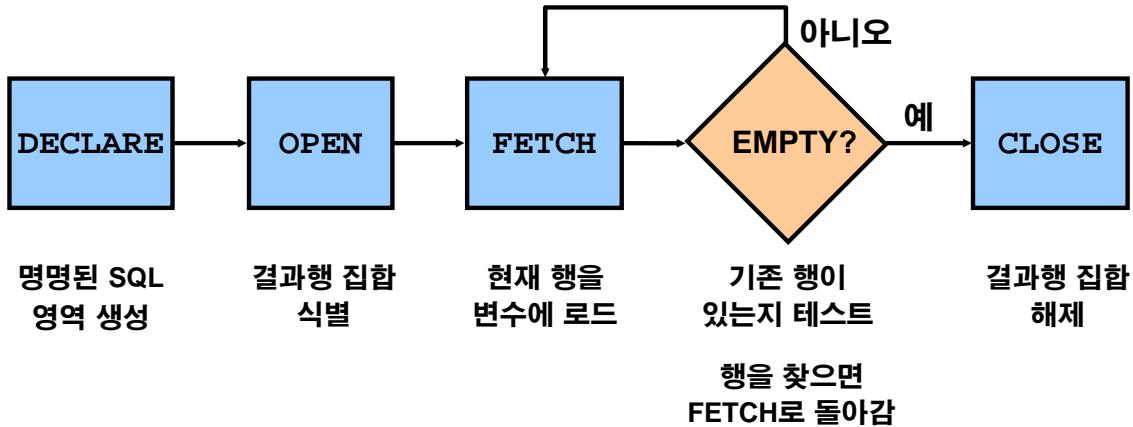
Copyright © 2009, Oracle. All rights reserved.

SQL 암시적 커서 속성

SQL 커서 속성을 사용하면 암시적 커서가 마지막으로 사용되었을 때 발생하는 작업을 평가할 수 있습니다. 이러한 속성은 함수와 같은 PL/SQL 문에서 사용하십시오. SQL 문에서는 사용할 수 없습니다.

블록의 예외 섹션에 SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND 및 SQL%ISOPEN 속성을 사용하면 DML 문 실행에 대한 정보를 수집할 수 있습니다. PL/SQL에서는 어떠한 행도 변경하지 않는 DML 문은 오류 조건으로 간주하지 않지만, SELECT 문의 경우 행을 찾을 수 없으면 예외를 반환합니다.

명시적 커서 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

명시적 커서

네 개의 명령을 사용하여 명시적 커서 제어

- 커서 이름을 지정하고 커서 내에서 수행될 query의 구조를 정의하여 커서를 선언합니다.
- 커서를 엽니다. OPEN 문은 query를 실행하고 참조된 모든 변수를 바인드합니다. query에 의해 식별된 행을 결과행 집합이라고 하며 현재 패치(fetch)할 수 있습니다.
- 커서에서 데이터를 패치(fetch)합니다. FETCH 문은 커서의 현재 행을 변수로 로드합니다. 패치할 때마다 커서가 커서 포인터를 결과행 집합의 다음 행으로 이동시킵니다. 따라서, 각각의 패치 시 query에 의해 반환된 다른 행에 액세스합니다. 위 슬라이드에 표시된 흐름도에서 패치할 때마다 기존 행이 있는지 커서를 테스트합니다. 행이 있으면 현재 행이 변수로 로드되고 그렇지 않으면 커서가 닫힙니다.
- 커서를 닫습니다. CLOSE 문은 결과행 집합을 해제합니다. 이제 커서를 다시 열어 새로운 결과행 집합을 설정할 수 있습니다.

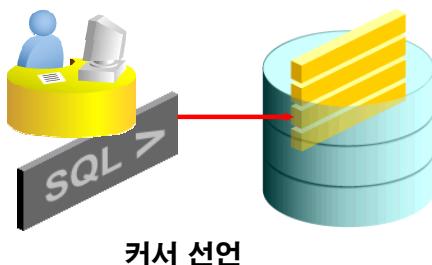
명시적 커서 제어: 커서 선언

```

DECLARE
    CURSOR c1 IS
        SELECT employee_id, last_name
        FROM   employees;

    CURSOR c2 IS
        SELECT *
        FROM   departments
        WHERE  department_id = 10;
BEGIN
    ...

```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

명시적 커서 선언

사원을 한 명씩 읽어 들입니다.

```

DECLARE
    v_empid  employees.employee_id%TYPE;
    v_ename   employees.last_name%TYPE;
    CURSOR c1 IS
        SELECT employee_id, last_name
        FROM   employees;
BEGIN
    ...

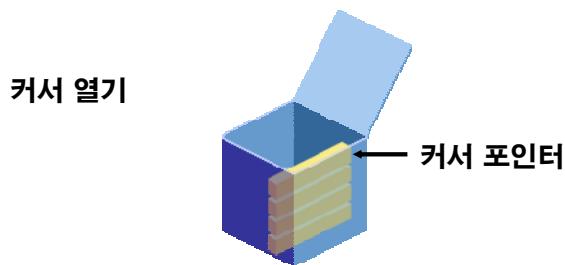
```

참고: 변수는 query에서 참조할 수 있지만 CURSOR 문 앞에 선언해야 합니다.

명시적 커서 제어: 커서 열기

```
OPEN cursor_name;
```

- 커서를 열어 query를 실행하고 결과행 집합을 식별합니다.
- query가 행을 반환하지 않으면 예외가 발생하지 않습니다.
- 커서 속성을 사용하여 패치(fetch) 후의 결과를 테스트합니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

OPEN 문

커서를 열어 query를 실행하고 query 검색 조건을 만족하는 모든 행으로 구성되는 결과 집합을 식별합니다. 이제 커서가 결과 집합의 첫번째 행을 가리킵니다.

구문에서 cursor_name은 이전에 선언한 커서의 이름입니다.

OPEN은 다음과 같은 작업을 수행하는 실행문입니다.

1. 결과적으로 중요한 처리 정보를 포함하게 될 컨텍스트 영역에 메모리를 동적으로 할당합니다.
2. SELECT 문의 구문을 분석합니다.
3. 입력 변수를 바인드합니다. 즉, 입력 변수의 메모리 주소를 확인하여 입력 변수 값을 설정합니다.
4. 결과 집합(검색 조건을 만족하는 행 집합)을 식별합니다. 결과 집합에 있는 행은 OPEN 문이 실행될 때 변수로 읽어 들여지지 않습니다. 대신 FETCH 문이 행을 읽어 들입니다.
5. 결과행 집합의 첫번째 행 바로 앞으로 포인터 위치를 지정합니다.

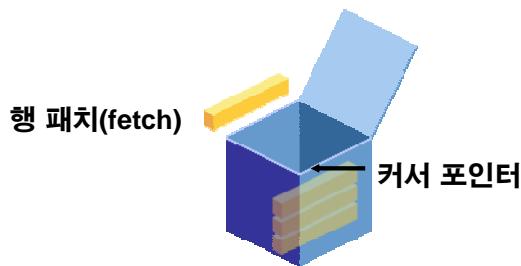
참고: 커서가 열려 있을 때 query가 행을 반환하지 않으면 PL/SQL에서 예외가 발생하지 않습니다. 그러나 패치(fetch) 이후의 커서 상태를 테스트할 수 있습니다.

FOR UPDATE 절을 사용하여 선언된 커서의 경우 OPEN 문으로도 이러한 행을 잠글 수 있습니다.

명시적 커서 제어: 커서에서 데이터 패치(fetch)

```
FETCH c1 INTO v.empid, v.ename;
```

```
...
OPEN defined_cursor;
LOOP
  FETCH defined_cursor INTO defined_variables
  EXIT WHEN ...;
  ...
  -- Process the retrieved data
  ...
END;
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

FETCH 문

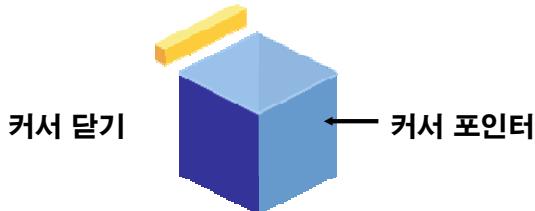
FETCH 문을 사용하여 현재 행 값을 출력 변수로 읽어 들일 수 있습니다. 패치(fetch) 후 추가 명령문으로 변수를 조작할 수 있습니다. 커서와 연관된 query를 통해 반환되는 열 값마다 INTO 리스트에 해당 변수가 있어야 합니다. 또한 데이터 유형은 호환되어야 합니다. 처음 10명의 사원을 한 명씩 읽어 들입니다.

```
DECLARE
  v_empid  employees.employee_id%TYPE;
  v_ename   employees.last_name%TYPE;
  i         NUMBER := 1;
  CURSOR c1 IS
    SELECT employee_id, last_name
    FROM   employees;
BEGIN
  OPEN c1;
  FOR i IN 1..10 LOOP
    FETCH c1 INTO v.empid, v.ename;
    ...
  END LOOP;
END;
```

명시적 커서 제어: 커서 닫기

```
CLOSE      cursor_name;
```

- 행 처리가 완료된 후에는 커서를 닫습니다.
- 필요한 경우 커서를 다시 엽니다.
- 커서가 닫힌 후에는 커서에서 데이터 패치(fetch)를 시도하지 마십시오.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

CLOSE 문

CLOSE 문은 커서를 비활성화하며 결과 집합은 미정의 상태가 됩니다. SELECT 문 처리가 완료된 후에는 커서를 닫으십시오. 필요한 경우 이 단계에서 커서를 다시 열 수 있습니다. 따라서 결과행 집합을 여러 번 설정할 수 있습니다.

구문에서 cursor_name은 이전에 선언한 커서의 이름입니다.

커서가 닫힌 후에는 커서에서 데이터 패치(fetch)를 시도하지 마십시오. 그렇지 않으면 INVALID_CURSOR 예외가 발생합니다.

참고: CLOSE 문은 컨텍스트 영역을 해제합니다. 커서를 닫지 않고 PL/SQL 블록을 종료할 수 있다 하더라도 리소스를 해제하려면 명시적으로 선언한 커서를 항상 닫아야 합니다. 유저당 최대 열린 커서 수 제한이 있는데, 이 제한은 데이터베이스 파라미터 펠드의 OPEN_CURSORS 파라미터에 의해 결정됩니다. 기본적으로 OPEN_CURSORS의 최대 수는 50입니다.

```
...
FOR i IN 1..10 LOOP
    FETCH c1 INTO v.empid, v.ename; ...
END LOOP;
CLOSE c1;
END;
```

명시적 커서 속성

커서에 대한 상태 정보를 획득합니다.

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

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

명시적 커서 속성

암시적 커서와 마찬가지로, 커서에 대한 상태 정보를 획득하기 위한 네 개의 속성이 있습니다. 이러한 속성을 커서나 커서 변수에 추가하면 DML 문 실행에 대한 유용한 정보가 반환됩니다.

참고: SQL 문에서 커서 속성을 직접 참조하지 마십시오.

커서 FOR 루프: 예제

남아 있는 사원이 없을 때까지 사원을 한 명씩 읽어 들입니다.

```
DECLARE
    CURSOR c1 IS
        SELECT employee_id, last_name
        FROM   employees;
BEGIN
    FOR emp_record IN c1 LOOP
        -- implicit open and implicit fetch occur
        IF emp_record.employee_id = 134 THEN
            ...
        END LOOP; -- implicit close occurs
END;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

커서 FOR 루프

커서 FOR 루프는 명시적 커서의 행을 처리합니다. 커서가 열리고, 행이 루프에서 반복될 때마다 한 번씩 패치(fetch)되며, 모든 행이 처리되면 자동으로 커서가 닫힙니다. 루프 자체는 마지막 행이 패치되어 반복 작업이 끝날 때 자동으로 종료됩니다. 슬라이드의 예제에서 커서 FOR 루프의 emp_record는 FOR LOOP 생성자에 사용되는 암시적으로 선언된 래코드입니다.

FOR UPDATE 절: 예제

오늘 처리된 주문 중 \$1,000를 초과한 주문을 읽어 들입니다.

```
DECLARE
  CURSOR c1 IS
    SELECT customer_id, order_id
    FROM   orders
    WHERE  order_date = SYSDATE
           AND order_total > 1000.00
    ORDER BY customer_id
    FOR UPDATE NOWAIT;
```

Copyright © 2009, Oracle. All rights reserved.

FOR UPDATE 절

데이터베이스 서버는 SELECT FOR UPDATE에 필요한 행에 대한 잠금을 확보할 수 없을 경우 무기한으로 기다립니다. SELECT FOR UPDATE 문에서 NOWAIT 절을 사용하여 루프에서 잠금을 확보하지 못해 반환되는 오류 코드가 있는지 테스트할 수 있습니다. 따라서 PL/SQL 블록이 종료될 때까지 커서 열기를 *n*번 재시도할 수 있습니다.

WHERE CURRENT OF 절을 사용하여 행을 생성하거나 삭제하려는 경우 FOR UPDATE OF 절에 열 이름을 지정해야 합니다.

대형 테이블이 있을 경우 LOCK TABLE 문을 사용하여 테이블의 모든 행을 잠그면 성능이 향상될 수 있습니다. 그러나 LOCK TABLE을 사용하는 경우에는 WHERE CURRENT OF 절을 사용할 수 없으며 WHERE *column* = *identifier* 표기법을 사용해야 합니다.

WHERE CURRENT OF 절: 예제

```

DECLARE
    CURSOR c1 IS
        SELECT salary FROM employees
        FOR UPDATE OF salary NOWAIT;
BEGIN
    ...
    FOR emp_record IN c1 LOOP
        UPDATE ...
            WHERE CURRENT OF c1;
    ...
    END LOOP;
    COMMIT;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WHERE CURRENT OF 절

커서 조건을 기준으로 행을 갱신할 수 있습니다.

또한 WHERE CURRENT OF cursor_name 절을 포함하도록 DELETE 또는 UPDATE 문을 작성하여 FETCH 문에 의해 처리된 제일 마지막 행을 참조할 수 있습니다. 이 절을 사용할 경우 참조하는 커서가 존재해야 하며 커서 query에 FOR UPDATE 절을 포함해야 합니다. 그렇지 않으면 오류가 발생합니다. 이 절을 사용하면 ROWID pseudocolumn을 명시적으로 참조할 필요 없이 현재 지정된 행에 갱신과 삭제를 적용할 수 있습니다.

미리 정의된 Oracle 서버 오류 트랩

- 예외 처리 루틴의 표준 이름을 참조합니다.
- 미리 정의된 예외 예제:
 - NO_DATA_FOUND
 - TOO_MANY_ROWS
 - INVALID_CURSOR
 - ZERO_DIVIDE
 - DUP_VAL_ON_INDEX

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

미리 정의된 Oracle 서버 오류 트랩

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

참고: PL/SQL은 미리 정의된 예외를 STANDARD 패키지에 선언합니다. 이 때 NO_DATA_FOUND 및 TOO_MANY_ROWS 예외 사용을 고려하는 것이 좋습니다. 이 두 가지가 가장 일반적인 미리 정의된 예외입니다.

미리 정의된 Oracle 서버 오류: 예제

```
BEGIN   SELECT ... COMMIT;  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
      statement1;  
      statement2;  
    WHEN TOO_MANY_ROWS THEN  
      statement1;  
    WHEN OTHERS THEN  
      statement1;  
      statement2;  
      statement3;  
END;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

미리 정의된 Oracle 서버 예외 트랩: 예제

위 슬라이드 예제에서는 각 예외에 대한 메시지가 유저에게 출력됩니다. 항상 한 개의 예외만 발생하고 처리됩니다.

미리 정의되지 않은 오류

Oracle 서버 오류 번호 -2292(무결성 제약 조건 위반)에 대한 트랩

```

DECLARE
    1 e_products_invalid EXCEPTION;
    PRAGMA EXCEPTION_INIT (
        2 e_products_invalid, -2292);
    v_message VARCHAR2(50);
BEGIN
    . . .
EXCEPTION
    3 WHEN e_products_invalid THEN
        :g_message := 'Product ID
specified is not valid.';
    . . .
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

미리 정의되지 않은 Oracle 서버 예외 트랩

- 선언 섹션 내에 예외에 대한 이름을 선언합니다.

구문

```
exception      EXCEPTION;
```

이 구문에서 *exception*은 예외의 이름입니다.

- PRAGMA EXCEPTION_INIT 문을 사용하여 선언된 예외를 표준 Oracle 서버 오류 번호와 연관시킵니다.

구문

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

구문 설명

<i>exception</i>	이 전에 선언된 예외
------------------	-------------

<i>error_number</i>	표준 Oracle 서버 오류 번호
---------------------	--------------------

- 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

슬라이드의 예제 설명: 제품에 재고가 있을 경우 프로세싱이 중지되고 유저에게 메시지가 출력됩니다.

유저 정의 예외: 예제

```
[DECLARE]
  e_amount_remaining EXCEPTION;
  .
  .
  .
BEGIN
  .
  .
  .
  RAISE e_amount_remaining;
  .
  .
  .
EXCEPTION
  WHEN e_amount_remaining THEN
    :g_message := 'There is still an amount
                   in stock.';
  .
  .
  .
END;
```

Copyright © 2009, Oracle. All rights reserved.

ORACLE

유저 정의 예외 트랩

유저 정의 예외를 선언한 다음 명시적으로 발생시켜 트랩합니다.

- 선언 섹션 내에 유저 정의 예외에 대한 이름을 선언합니다.

구문: exception EXCEPTION;

여기서 exception은 예외 이름

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

구문: RAISE exception;

여기서 exception은 이전에 선언된 예외

- 해당하는 예외 처리 루틴 내에 선언된 예외를 참조합니다.

슬라이드의 예제 설명: 이 고객은 재고가 남아 있는 제품은 데이터베이스에서 제거할 수 없다는 업무 규칙을 가지고 있습니다. 이 규칙을 적용하기 위한 제약 조건이 없기 때문에 개발자가 응용 프로그램에서 이를 명시적으로 처리합니다. PRODUCT_INFORMATION 테이블에 대한 DELETE를 수행하기 전에 블록은 문제가 되는 제품에 대한 재고가 있는지 확인하기 위해 INVENTORIES 테이블을 query합니다. 재고가 있으면 예외가 발생합니다.

참고: 예외 처리기 내에서 RAISE 문만 사용하여 호출 환경으로 다시 동일한 예외를 발생시킵니다.

RAISE_APPLICATION_ERROR 프로시저

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

- 내장 서브 프로그램에서 유저 정의 오류 메시지를 실행할 수 있습니다.
- 실행 중인 내장 서브 프로그램에서만 호출됩니다.



Copyright © 2009, Oracle. All rights reserved.

RAISE_APPLICATION_ERROR 프로시저

RAISE_APPLICATION_ERROR 프로시저를 사용하면 비표준 오류 코드 및 오류 메시지를 반환하여 미리 정의된 예외를 대화식으로 전달할 수 있습니다.

RAISE_APPLICATION_ERROR를 사용하여 응용 프로그램에 오류를 보고하고 처리되지 않은 예외가 반환되지 않도록 할 수 있습니다.

위 구문에서 *error_number*는 유저가 지정한 -20,000부터 -20,999까지 범위의 예외 번호입니다.

*Message*는 유저가 지정한 예외 메시지입니다. 메시지는 최대 2,048바이트의 문자열입니다.

TRUE | FALSE는 옵션 부울 파라미터입니다. TRUE일 경우 오류가 이전 오류의 스택에 위치하게 되고, FALSE(기본값)일 경우 이전 오류가 모두 해당 오류로 바뀝니다.

예제:

```
...  
EXCEPTION  
WHEN NO_DATA_FOUND THEN  
    RAISE_APPLICATION_ERROR (-20201,  
        'Manager is not a valid employee.' );  
END;
```

RAISE_APPLICATION_ERROR 프로시저

- 다음과 같은 서로 다른 두 위치에 사용됩니다.
 - Executable 섹션
 - Exception 섹션
- 다른 Oracle 서버 오류와 일치하는 방식으로 유저에게 오류 조건을 반환합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

RAISE_APPLICATION_ERROR 프로시저: 예제

```
...
DELETE FROM employees
WHERE manager_id = v_mgr;
IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20202,
        'This is not a valid manager');
END IF;
...
```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- **익명 PL/SQL 블록의 블록 구조 검토**
- **PL/SQL 변수 선언**
- **PL/SQL 레코드 및 테이블 생성**
- **데이터 삽입, 갱신 및 삭제**
- **IF, THEN 및 ELSIF 문 사용**
- **기본 FOR 및 WHILE 루프 사용**
- **파라미터가 포함된 커서 명시적 선언 및 사용**
- **커서 FOR 루프 및 FOR UPDATE 및 WHERE CURRENT OF 절 사용**
- **미리 정의된 예외 및 유저 정의 예외 트랩**



Copyright © 2009, Oracle. All rights reserved.

요약

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

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

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



트리거 구현 학습

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- 트리거를 사용하여 데이터베이스 보안 강화
- DML 트리거를 사용하여 데이터 무결성 적용
- 트리거를 사용하여 참조 무결성 유지 관리
- 트리거를 사용하여 테이블 간 데이터 복제(replication)
- 트리거를 사용하여 파생된 데이터 자동 계산
- 트리거를 사용하는 이벤트 로깅 기능 제공

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 Oracle 서버에서 구현할 수 없는 기능을 강화하기 위해 데이터베이스 트리거를 개발하는 방법에 대해 설명합니다. 경우에 따라서는 트리거를 사용하지 않고 Oracle 서버에서 제공하는 기능을 사용하는 것만으로도 충분할 수 있습니다.

이 단원에서는 다음과 같은 업무용 응용 프로그램 시나리오에 대해 다룹니다.

- 보안
- 감사(audit)
- 데이터 무결성
- 참조 무결성
- 테이블 복제(replication)
- 파생된 데이터 자동 계산
- 이벤트 로깅

서버 내에서 보안 제어

GRANT 문으로 데이터베이스 보안을 사용합니다.

```
GRANT SELECT, INSERT, UPDATE, DELETE  
  ON   employees  
  TO   clerk;          -- database role  
GRANT clerk TO scott;
```



Copyright © 2009, Oracle. All rights reserved.

서버 내에서 보안 제어

유저 ID에 따라 테이블에 대한 데이터 작업의 보안을 제어하도록 Oracle 서버 내에 스키마와 룰을 개발하십시오.

- 유저가 데이터베이스에 연결할 때 입력하는 username을 기준으로 권한 부여
- 테이블, 뷰, 동의어 및 시퀀스에 대한 액세스 권한 결정
- Query, 데이터 조작 및 데이터 정의 권한 결정

데이터베이스 트리거를 사용한 보안 제어

```
CREATE OR REPLACE TRIGGER secure_emp
  BEFORE INSERT OR UPDATE OR DELETE ON employees
DECLARE
  dummy PLS_INTEGER;
BEGIN
  IF (TO_CHAR (SYSDATE, 'DY') IN ('SAT','SUN')) THEN
    RAISE_APPLICATION_ERROR(-20506,'You may only
      change data during normal business hours.');
  END IF;
  SELECT COUNT(*) INTO dummy FROM holiday
  WHERE holiday_date = TRUNC (SYSDATE);
  IF dummy > 0 THEN
    RAISE_APPLICATION_ERROR(-20507,
      'You may not change data on a holiday.');
  END IF;
END;
/
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 트리거를 사용한 보안 제어

보다 복잡한 보안 요구 사항을 다루는 트리거를 개발하십시오.

- 시간, 요일과 같은 데이터베이스 값을 기준으로 권한 부여
- 테이블 액세스 권한만 결정
- 데이터 조작 권한만 결정

서버 내에서 데이터 무결성 적용

```
ALTER TABLE employees ADD  
CONSTRAINT ck_salary CHECK (salary >= 500);
```



Copyright © 2009, Oracle. All rights reserved.

서버 내에서 데이터 무결성 적용

Oracle 서버 내에서 데이터 무결성을 적용하고 보다 복잡한 데이터 무결성 규칙을 처리하는 트리거를 개발할 수 있습니다.

표준 데이터 무결성 규칙은 널이 아니면서 고유한 Primary Key 및 Foreign Key입니다.

이러한 규칙을 사용하여 다음 작업을 수행할 수 있습니다.

- 고정 기본값 제공
- 정적 제약 조건 적용
- 동적 활성화 및 비활성화

예제

위 슬라이드에 표시된 코드 예제에서는 최소 급여가 \$500가 되도록 설정합니다.

트리거를 사용한 데이터 무결성 보호

```
CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
  WHEN (NEW.salary < OLD.salary)
BEGIN
  RAISE_APPLICATION_ERROR (-20508,
    'Do not decrease salary.');
END;
/
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거를 사용한 데이터 무결성 보호

트리거를 사용하여 데이터 무결성을 보호하고 비표준 데이터 무결성 검사를 수행하십시오.

- 가변 기본값 제공
- 동적 제약 조건 적용
- 동적 활성화 및 비활성화
- 데이터 무결성을 보호하도록 테이블 정의 내에서 선언 제약 조건 통합

예제

위 슬라이드에 표시된 코드 예제에서는 급여가 감소하지 않도록 설정합니다.

서버 내에서 참조 무결성 적용

```
ALTER TABLE employees
  ADD CONSTRAINT emp_deptno_fk
    FOREIGN KEY (department_id)
      REFERENCES departments(department_id)
    ON DELETE CASCADE;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

서버 내에서 참조 무결성 적용

서버 내에서 데이터 불일치를 방지하고 참조 무결성을 적용하려면 테이블 정의 내에서 참조 무결성 제약 조건을 통합하십시오.

- 간접 및 삭제 제한
- 계단식 삭제
- 동적 활성화 및 비활성화

예제

DEPARTMENTS 상위 테이블에서 부서를 제거할 경우 EMPLOYEES 하위 테이블의 해당 행까지 연쇄적으로 삭제됩니다.

트리거를 사용한 참조 무결성 보호

```

CREATE OR REPLACE TRIGGER cascade_updates
AFTER UPDATE OF department_id ON departments
FOR EACH ROW
BEGIN
  UPDATE employees
    SET employees.department_id=:NEW.department_id
  WHERE employees.department_id=:OLD.department_id;
  UPDATE job_history
    SET department_id=:NEW.department_id
  WHERE department_id=:OLD.department_id;
END;
/

```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거를 사용한 참조 무결성 보호

다음의 참조 무결성 규칙은 선언 제약 조건에서 지원하지 않습니다.

- 계단식 갱신
- 갱신 및 삭제 시 NULL로 설정
- 갱신 및 삭제 시 기본값으로 설정
- 분산 시스템에서 참조 무결성 적용
- 동적 활성화 및 비활성화

이러한 무결성 규칙을 구현하는 트리거를 개발할 수 있습니다.

예제

트리거를 사용하여 참조 무결성을 적용할 수 있습니다. DEPARTMENTS 상위 테이블에서 DEPARTMENT_ID 값이 변경될 경우 EMPLOYEES 하위 테이블의 해당 행까지 연쇄적으로 갱신됩니다.

트리거를 사용하여 참조 무결성을 완전하게 보호하려면 단일 트리거만으로는 충분하지 않습니다.

서버 내에서 테이블 복제(replication)

```
CREATE MATERIALIZED VIEW emp_copy
NEXT sysdate + 7
AS SELECT * FROM employees@ny;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Materialized view 생성

Materialized view를 사용하면 복제(replication)를 위해 로컬 노드에서 원격 데이터의 복사본을 유지 관리할 수 있습니다. 일반 데이터베이스 테이블이나 뷰에서와 마찬가지로, Materialized view에서도 데이터를 선택할 수 있습니다. Materialized view는 query 결과 또는 query에 일부 데이터베이스 복사본을 포함하는 데이터베이스 객체입니다. Materialized view query의 FROM 절에서는 테이블, 뷰 및 기타 Materialized view의 이름을 지정할 수 있습니다.

Materialized view를 사용하면 복제(replication)가 Oracle 서버에 의해 암시적으로 수행됩니다. 이것은 유저가 정의한 PL/SQL 트리거를 사용하여 복제(replication)하는 것보다 성능이 우수합니다. Materialized view의 특징은 다음과 같습니다.

- 유저가 정의한 간격으로 로컬 및 원격 테이블에서 비동기적으로 데이터를 복사합니다.
- 다중 마스터 테이블을 기반으로 할 수 있습니다.
- Oracle Advanced Replication 기능을 사용하지 않는 한 기본적으로 읽기 전용입니다.
- 마스터 테이블에 대한 데이터 조작 성능이 향상됩니다.

또 다른 방법으로 트리거를 사용하여 테이블을 복제(replication)할 수 있습니다.

위 슬라이드에 표시된 예제에서는 New York의 원격 EMPLOYEES 테이블의 복사본을 생성합니다. NEXT 절은 자동 Refresh 간격에 대한 날짜 표현식을 지정합니다.

트리거를 사용한 테이블 복제(replication)

```

CREATE OR REPLACE TRIGGER emp_replica
BEFORE INSERT OR UPDATE ON employees FOR EACH ROW
BEGIN /* Proceed if user initiates data operation,
NOT through the cascading trigger.*/
    IF INSERTING THEN
        IF :NEW.flag IS NULL THEN
            INSERT INTO employees@sf
            VALUES(:new.employee_id,...,'B');
            :NEW.flag := 'A';
        END IF;
    ELSE /* Updating. */
        IF :NEW.flag = :OLD.flag THEN
            UPDATE employees@sf
            SET ename=:NEW.last_name,...,flag=:NEW.flag
            WHERE employee_id = :NEW.employee_id;
        END IF;
        IF :OLD.flag = 'A' THEN :NEW.flag := 'B';
            ELSE :NEW.flag := 'A';
        END IF;
    END IF;
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거를 사용한 테이블 복제(replication)

트리거를 사용하여 테이블을 복제(replication)할 수 있습니다. 테이블을 복제(replication)하면 다음 작업을 수행할 수 있습니다.

- 테이블을 동기적으로 실시간으로 복사
- 단일 마스터 테이블을 기반으로 복제본 생성
- 복제본에 대한 읽기 및 쓰기 작업

참고: 트리거를 지나치게 많이 사용하면 특히 네트워크 장애가 있을 경우 마스터 테이블에 대한 데이터 조작 성능이 떨어질 수 있습니다.

예제

New York의 로컬 EMPLOYEES 테이블을 San Francisco로 복제(replication)합니다.

서버 내에서 파생된 데이터 계산

```
UPDATE departments
  SET total_sal=(SELECT SUM(salary)
                  FROM employees
                 WHERE employees.department_id =
                       departments.department_id);
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

서버 내에서 파생된 데이터 계산

서버를 사용하면 다음 시나리오에 대해 일괄 처리 작업을 스케줄링하거나 데이터베이스 스케줄러를 사용할 수 있습니다.

- 유저가 정의한 간격으로 파생된 열 값을 비동기적으로 계산
- 데이터베이스 테이블에만 파생된 값 저장
- 데이터베이스에 대해 첫번째 패스에서 데이터를 수정한 다음 두번째 패스에서 파생된 데이터 계산

또 다른 방법으로 트리거를 사용하여 파생된 데이터를 계산할 수 있습니다.

예제

DEPARTMENTS 테이블의 특수 TOTAL_SALARY 열 내에 각 부서의 급여 총액을 저장합니다.

트리거를 사용하여 파생된 값 계산

```
CREATE PROCEDURE increment_salary
  (id NUMBER, new_sal NUMBER) IS
BEGIN
  UPDATE departments
  SET    total_sal = NVL (total_sal, 0)+ new_sal
  WHERE department_id = id;
END increment_salary;
```

```
CREATE OR REPLACE TRIGGER compute_salary
AFTER INSERT OR UPDATE OF salary OR DELETE
ON employees FOR EACH ROW
BEGIN
  IF DELETING THEN      increment_salary(
    :OLD.department_id,(-1*:OLD.salary));
  ELSIF UPDATING THEN   increment_salary(
    :NEW.department_id,:NEW.salary-:OLD.salary);
  ELSE                 increment_salary(
    :NEW.department_id,:NEW.salary); --INSERT
  END IF;
END;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

트리거를 사용하여 파생된 데이터 값 계산

트리거를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- 파생된 열을 동기적으로 실시간으로 계산
- 데이터베이스 테이블 또는 패키지 global 변수 내에 파생된 값 저장
- 데이터베이스에 대한 데이터 수정 및 파생된 데이터 계산 작업을 단일 패스로 수행

예제

DEPARTMENTS 테이블의 특수 TOTAL_SALARY 열에 각 부서의 급여에 대한 누계를 저장합니다.

트리거를 사용한 이벤트 로깅

```

CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF quantity_on_hand, reorder_point
ON inventories FOR EACH ROW
DECLARE
    dsc product_descriptions.product_description%TYPE;
    msg_text VARCHAR2(2000);
BEGIN
    IF :NEW.quantity_on_hand <=
        :NEW.reorder_point THEN
        SELECT product_description INTO dsc
        FROM product_descriptions
        WHERE product_id = :NEW.product_id;
        msg_text := 'ALERT: INVENTORY LOW ORDER:' ||
                    'Yours,' || CHR(10) || user || '.' || CHR(10);
    ELSIF :OLD.quantity_on_hand >=
        :NEW.quantity_on_hand THEN
        msg_text := 'Product #' || ... CHR(10);
    END IF;
    UTL_MAIL.SEND('inv@oracle.com','ord@oracle.com',
                  message=>msg_text, subject=>'Inventory Notice');
END;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리거를 사용한 이벤트 로깅

서버에서는 데이터를 query하고 작업을 수동으로 수행하여 이벤트를 기록할 수 있습니다. 이렇게 하면 특정 제품에 대한 재고가 허용 한도 아래로 떨어졌을 때 전자 메일 메시지가 전송됩니다. 이 트리거는 오라클에서 제공하는 UTL_MAIL 패키지를 사용하여 전자 메일 메시지를 보냅니다.

서버 내에서의 이벤트 로깅

- 데이터를 명시적으로 query하여 특정 작업이 필요한지 여부를 확인합니다.
- 메시지 보내기와 같은 작업을 수행합니다.

트리거를 사용한 이벤트 로깅

- 자동 전자 메모 해제와 같은 작업을 암시적으로 수행합니다.
- 한 번에 데이터를 수정하고 종속 작업을 수행합니다.
- 데이터가 변경되면 이벤트가 자동으로 기록됩니다.

트리거를 사용한 이벤트 로깅(계속)

투명한 이벤트 로깅

트리거 코드에서

- CHR(10)은 캐리지 리턴입니다.
- Reorder_point는 NULL이 아닙니다.
- 또 다른 트랜잭션이 파이프에서 메시지를 받고 읽을 수 있습니다.

예제

```

CREATE OR REPLACE TRIGGER notify_reorder_rep
BEFORE UPDATE OF amount_in_stock, reorder_point
ON inventory FOR EACH ROW
DECLARE
    dsc product.descrip%TYPE;
    msg_text VARCHAR2(2000);
BEGIN
    IF :NEW.amount_in_stock <= :NEW.reorder_point THEN
        SELECT descrip INTO dsc
        FROM PRODUCT WHERE prodid = :NEW.product_id;
        msg_text := 'ALERT: INVENTORY LOW ORDER:' || CHR(10) ||
                    'It has come to my personal attention that, due to recent' ||
                    CHR(10) || 'transactions, our inventory for product #' ||
                    TO_CHAR(:NEW.product_id) || '-- ' || dsc ||
                    ' -- has fallen below acceptable levels.' || CHR(10) ||
                    'Yours,' || CHR(10) || user || '.' || CHR(10) || CHR(10);
    ELSIF :OLD.amount_in_stock >= :NEW.amount_in_stock THEN
        msg_text := 'Product #' || TO_CHAR(:NEW.product_id)
                    || ' ordered. ' || CHR(10) || CHR(10);
    END IF;
    UTL_MAIL.SEND('inv@oracle.com', 'ord@oracle.com',
                  message => msg_text, subject => 'Inventory Notice');
END;

```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- 트리거를 사용하여 데이터베이스 보안 강화
- DML 트리거를 사용하여 데이터 무결성 적용
- 트리거를 사용하여 참조 무결성 유지 관리
- 트리거를 사용하여 테이블 간 데이터 복제(replication)
- 트리거를 사용하여 파생된 데이터 자동 계산
- 트리거를 사용하는 이벤트 로깅 기능 제공

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 오라클 데이터베이스 서버 기능을 사용하여 보안, 감사(audit), 데이터 무결성, 복제(replication) 및 로깅을 구현하는 것에 대해 상세히 비교했으며, 데이터베이스 트리거를 사용하여 동일한 기능을 구현하되, 데이터베이스 서버에서 제공하는 기능을 보다 강화하는 방법에 대해서도 설명했습니다. 일부 경우에 몇 가지 프로그래밍 작업을 수행하지 않으면 Oracle 서버가 이러한 종류의 업무 규칙을 구현하는 방법을 알 수 없기 때문에 트리거를 사용하여 몇 가지 작업(예: 파생된 데이터 계산)을 수행해야 합니다.

DBMS_SCHEDULER 및 HTP 패키지 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- HTP 패키지를 사용하여 간단한 웹 페이지 생성
- DBMS_SCHEDULER 패키지를 호출하여 실행할 PL/SQL 코드 스케줄링(scheduling)

ORACLE®

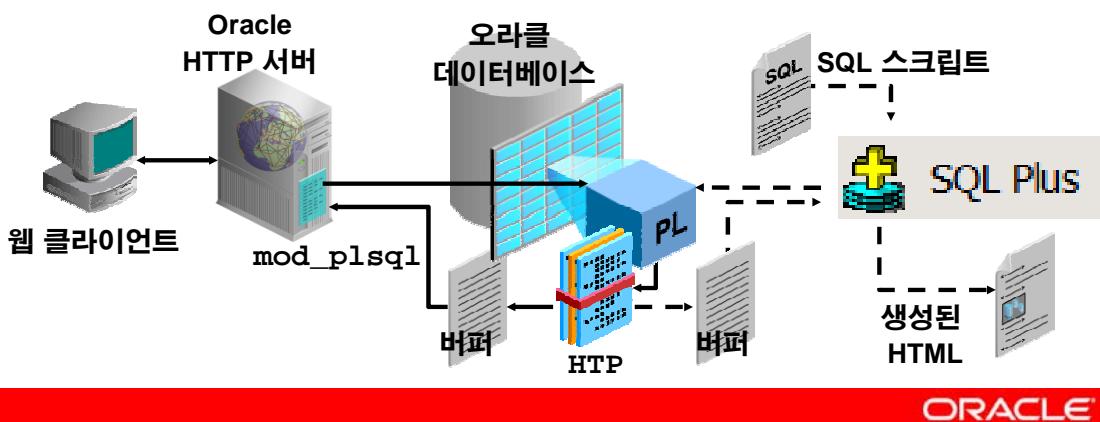
Copyright © 2009, Oracle. All rights reserved.

단원 목적

이 단원에서는 몇 가지 오라클 제공 패키지와 이들의 기능을 사용하는 방법에 대해 설명합니다. 웹 기반 출력을 생성하는 패키지와 제공되는 스케줄링(scheduling) 기능에 초점을 맞추고 있습니다.

HTP 패키지를 사용하여 웹 페이지 생성

- HTP 패키지 프로시저는 HTML 태그를 생성합니다.
- HTP 패키지는 HTML 문서를 동적으로 생성하는 데 사용되며 다음에서 호출할 수 있습니다.
 - Oracle HTTP Server 및 PL/SQL Gateway (`mod_plsql`) 서비스를 사용하는 브라우저
 - HTML 출력을 표시하는 SQL*Plus 스크립트



Copyright © 2009, Oracle. All rights reserved.

HTP 패키지를 사용하여 웹 페이지 생성

HTP 패키지는 HTML 태그를 생성하는 데 사용되는 프로시저를 포함합니다. 생성되는 HTML 태그는 일반적으로 여러 프로시저에 파라미터로 제공되는 데이터를 포함합니다. 슬라이드는 HTP 패키지를 사용하는 두 가지 방법을 보여줍니다.

- 대부분의 프로시저는 Oracle Application Server 제품의 일부인 Oracle HTTP Server에서 제공하는 `mod_plsql` 구성 요소를 통해 PL/SQL 게이트웨이 서비스에 의해 호출됩니다(그림에 실선으로 표시됨).
- 또 다른 방법(그림에 점선으로 표시됨)으로 프로시저를 SQL*Plus에서 호출할 수 있습니다. SQL*Plus에서는 생성된 HTML 출력을 표시하고, 이 출력을 복사하여 파일에 붙여넣을 수 있습니다. Oracle Application Server 소프트웨어가 과정 환경의 일부로 설치되어 있지 않으므로 본 과정에서는 이 방법을 사용합니다.

참고: HTP 프로시저는 데이터베이스 서버에 보관된 세션 버퍼에 정보를 출력합니다. Oracle HTTP Server 컨텍스트에서는 프로시저가 완료되면 `mod_plsql` 구성 요소가 자동으로 버퍼 내용을 받은 다음 이 내용을 브라우저에 HTTP 응답으로 반환합니다. SQL*Plus에서는 다음을 수동으로 실행해야 합니다.

- `SET SERVEROUTPUT ON` 명령
- 버퍼에 HTML을 생성하는 프로시저
- 버퍼 내용을 표시하기 위한 `OWA_UTIL.SHOWPAGE` 프로시저

HTP 패키지 프로시저 사용

- 여러 개의 HTML 태그를 생성합니다. 예를 들면 다음과 같습니다.

```
htp.bold('Hello');           -- <B>Hello</B>
htp.print('Hi <B>World</B>'); -- Hi <B>World</B>
```

- 잘 구성된 HTML 문서를 생성하는 데 사용됩니다.

<pre>BEGIN htp.htmlOpen; -----> htp.headOpen; -----> htp.title('Welcome'); --> htp.headClose; -----> htp.bodyOpen; -----> htp.print('My home page'); htp.bodyClose; -----> htp.htmlClose; -----> END;</pre>	-- Generates: <pre><HTML> <HEAD> <TITLE>Welcome</TITLE> </HEAD> <BODY> My home page </BODY> </HTML></pre>
--	--

ORACLE

Copyright © 2009, Oracle. All rights reserved.

HTP 패키지 프로시저 사용

HTP 패키지는 프로시저를 표준 HTML 태그에 일대일로 매핑하도록 구성됩니다. 예를 들어, 웹 페이지에 굵은체 텍스트를 표시하려면 해당 텍스트를 및 HTML 태그 쌍(pair) 사이에 삽입해야 합니다. 위 슬라이드에 표시된 첫번째 코드 상자는 해당하는 HTP 패키지 프로시저, 즉 HTP.BOLD를 사용하여 Hello라는 단어를 HTML 굵은체 텍스트로 생성하는 방법을 보여줍니다. HTP.BOLD 프로시저는 텍스트 파라미터를 사용하며, 텍스트 파라미터가 생성된 HTML 출력에서 해당 HTML 태그 사이에 오도록 합니다.

HTP.PRINT 프로시저는 텍스트 파라미터를 버퍼에 복사합니다. 슬라이드 쇼의 예제는 HTP.PRINT 프로시저에 제공된 파라미터가 어떻게 HTML 태그를 포함할 수 있는지를 보여줍니다. 이 방법은 HTP 패키지에서 제공하는 프로시저 집합을 사용하여 생성할 수 없는 HTML 태그를 사용해야 할 경우에만 권장됩니다.

위 슬라이드에 표시된 두번째 예제는 HTML 문서의 기본 형식을 생성하는 PL/SQL 블록을 제공하며, 오른쪽에 표시된 텍스트 상자에서 각 프로시저가 해당 HTML 행을 생성하는 방법을 보여줍니다.

HTP 패키지를 사용할 경우 얻게 되는 이점은 잘 구성된 HTML 문서가 작성되므로, HTML 태그를 각 데이터 주변에 수동으로 입력할 필요가 없다는 점입니다.

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

SQL*Plus를 사용하여 HTML 파일 생성

SQL*Plus를 사용하여 HTML 파일을 생성하려면 다음 단계를 수행하십시오.

1. 다음 명령으로 SQL 스크립트를 생성합니다.

```
SET SERVEROUTPUT ON
ACCEPT procname PROMPT "Procedure: "
EXECUTE &procname
EXECUTE owa_util.showpage
UNDEFINE proc
```

2. 치환 변수에 값을 제공하여 SQL*Plus에서 스크립트를 로드하고 실행합니다.
3. 브라우저에 생성된 HTML 텍스트를 선택하여 복사한 다음 HTML 파일에 붙여넣습니다.
4. 브라우저에서 HTML 파일을 엽니다.



Copyright © 2009, Oracle. All rights reserved.

SQL*Plus를 사용하여 HTML 파일 생성

슬라이드의 예제는 임의의 프로시저를 사용한 다음 그 출력을 HTML 파일에 저장하여 HTML을 생성하는 단계를 보여줍니다. 수행해야 할 단계는 다음과 같습니다.

1. SET SERVEROUTPUT ON 명령으로 서버 출력을 캡니다. 이렇게 하지 않으면 HTP 패키지에 대한 호출이 포함된 프로시저를 실행할 때 예외 메시지를 받게 됩니다.
2. HTP 패키지에 대한 호출이 포함된 프로시저를 실행합니다.

참고: 프로시저에 DBMS_OUTPUT 패키지에 대한 호출이 포함되어 있지 않으면 출력이 생성되지 않습니다.

3. OWA_UTIL.SHOWPAGE 프로시저를 실행하여 텍스트를 표시합니다. 이 호출은 실제로 베퍼에서 생성된 HTML 내용을 표시합니다.

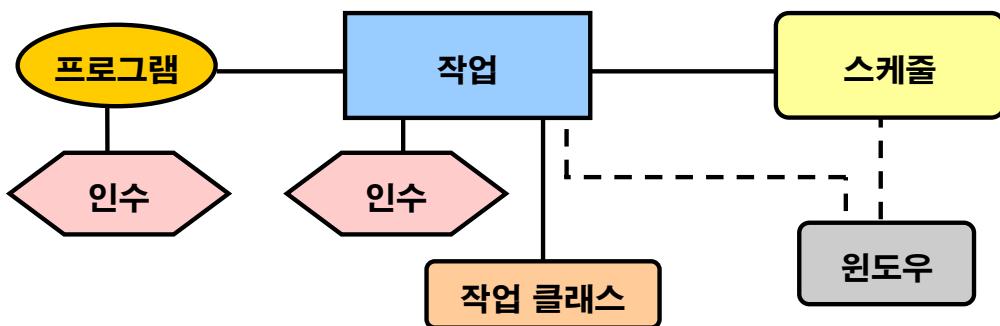
ACCEPT 명령은 실행할 프로시저 이름을 묻는 프롬프트를 표시합니다. OWA_UTIL.SHOWPAGE를 호출하면 브라우저 window에 HTML 태그가 표시됩니다. 따라서 생성된 HTML 태그를 브라우저 window에서 복사한 다음 HTML 파일(일반적으로 확장명이 .htm 또는 .html임)에 붙여넣을 수 있습니다.

참고: SQL*Plus를 사용할 경우 SPOOL 명령을 사용하여 HTML 출력을 직접 HTML 파일로 지정할 수 있습니다.

DBMS_SCHEDULER 패키지

데이터베이스 스케줄러는 작업 실행을 가능하게 하는 여러 구성 요소로 구성됩니다. DBMS_SCHEDULER 패키지를 사용하여 다음 요소를 가진 각각의 작업을 생성하십시오.

- 고유한 작업 이름
- 프로그램(실행될 "항목")
- 스케줄(실행 "시기")



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DBMS_SCHEDULER 패키지

오라클 데이터베이스에서 제공하는 DBMS_SCHEDULER 패키지의 서브 프로그램 모음은 관리를 간편하게 하고 복잡한 스케줄링(scheduling) 작업에 도움이 되는 풍부한 기능을 제공합니다. 이러한 서브 프로그램을 통틀어 스케줄러라고 하며 PL/SQL 프로그램에서 호출할 수 있습니다. 데이터베이스 관리자와 응용 프로그램 개발자는 스케줄러를 사용하여 여러 작업이 실행되는 시기와 위치를 제어할 수 있습니다. 수많은 루틴 데이터베이스 작업을 사용자 개입 없이 자동으로 실행하게 되면 운영비 절감, 보다 안정적인 루틴 구현, 사용자 오류 최소화 등의 효과를 얻을 수 있습니다.

위 도표는 다음과 같은 스케줄러 구성 요소를 보여줍니다.

- **작업**은 프로그램과 스케줄이 결합된 것입니다. 프로그램에서 요구하는 인수는 프로그램이나 작업에서 제공할 수 있습니다. 모든 작업 이름은 [schema.]name 형태입니다. 작업을 생성할 때는 작업 이름, 프로그램 및 스케줄을 지정하고 경우에 따라 **작업 클래스**를 통해 제공되는 작업 특성을 지정하십시오.
- **프로그램**은 실행되어야 할 항목을 결정합니다. 자동화된 모든 작업에는 특정 실행 파일이 포함되는데, 이 실행 파일은 PL/SQL 블록, 내장 프로시저, 고유 이진 실행 파일 또는 셀 스크립트일 수 있습니다. 프로그램은 특정 실행 파일에 대한 메타 데이터를 제공하며 인수 리스트를 필요로 할 수도 있습니다.
- **스케줄**은 작업의 실행 시기와 횟수를 지정합니다.

DBMS_SCHEDULER 패키지(계속)

- **작업 클래스**는 공통적인 리소스 사용 요구 사항과 기타 특성을 공유하는 작업의 범주를 정의합니다. 특정 시간에 각 작업은 하나의 작업 클래스에만 속할 수 있습니다. 작업 클래스는 다음과 같은 속성을 갖습니다.
 - 데이터베이스 서비스 이름. 작업 클래스의 작업은 지정된 특정 서비스에 대한 친화력이 있습니다. 즉, 지정된 서비스에 소속된 Instance에서 작업이 실행됩니다.
 - **Resource Consumer Group**. 공통적인 리소스 처리 요구 사항을 가진 유저 세션 집합을 분류한 것입니다. 특정 시간에 유저 세션이나 작업 클래스는 하나의 Resource Consumer Group에만 속할 수 있습니다. 작업 클래스와 연관된 Resource Consumer Group은 작업 클래스에 할당되는 리소스를 결정합니다.
- **윈도우**는 시작 시간과 종료 시간이 잘 정의된 시간 간격으로 표시되며, 각기 다른 시간에 각각의 Resource Plan을 활성화하는 데 사용됩니다.

위 슬라이드는 작업 구성 요소를 주요 엔티티로서 초점을 맞추고 있습니다. 그러나 프로그램, 스케줄, 윈도우, 작업 클래스도 개별 엔티티로 생성할 수 있는 구성 요소로서, 스케줄러에 의해 실행될 작업과 연관될 수 있습니다. 작업 생성 시에는 인라인, 즉 작업을 생성하는 호출에 필요한 모든 정보를 포함시킬 수도 있고, 아니면 이전에 정의한 프로그램이나 스케줄 구성 요소를 참조할 수도 있습니다. 이에 대한 예제는 다음 몇 페이지에 걸쳐 다릅니다.

스케줄러에 대한 자세한 내용은 *Oracle Database 11g: Configure and Manage Jobs with the Scheduler* 온라인 과정을 참조하십시오.

작업 생성

- **인라인 파라미터, 명명된 Programs 및 명명된 Schedules를 조합하여 여러 가지 방법으로 작업을 생성할 수 있습니다.**
- **다음 방법으로 CREATE_JOB 프로시저를 사용하는 작업을 생성할 수 있습니다.**
 - "실행될 프로그램"과 스케줄이 지정된 인라인 정보를 파라미터로 사용
 - 명명된(저장된) 프로그램 사용 및 인라인으로 스케줄 지정
 - 인라인으로 수행되어야 할 작업 지정 및 명명된 스케줄 사용
 - 명명된 프로그램 및 스케줄 구성 요소 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

작업 생성

지정된 시간에 특정 작업이 실행되도록 하는 구성 요소를 **작업**이라고 합니다. DBMS_SCHEDULER 패키지의 DBMS_SCHEDULER.CREATE_JOB 프로시저를 사용하여 작업을 생성하십시오.

이 작업은 기본적으로 비활성화된 상태입니다. 작업은 명시적으로 활성화될 때 활성 상태가 되고 스케줄링(scheduling)됩니다. 작업을 생성하려면 다음을 수행하십시오.

- [schema .]name 형태로 이름을 지정합니다.
- CREATE JOB 권한이 필요합니다.

참고: CREATE ANY JOB 권한을 가진 유저는 SYS 스키마를 제외한 모든 스키마에서 작업을 생성할 수 있습니다. 특정 클래스와 작업을 연관시키려면 해당 클래스에 대한 EXECUTE 권한이 필요합니다.

간단히 말해 CREATE_JOB 프로시저의 인수에 모든 작업 세부 정보(실행될 프로그램과 스케줄)를 지정하여 작업을 생성할 수 있습니다. 또는 미리 정의된 프로그램과 스케줄 구성 요소를 사용할 수도 있습니다. 명명된 프로그램과 스케줄이 있을 경우 이러한 구성 요소를 지정하거나 인라인 인수와 결합하여 작업 생성 방식의 유연성을 극대화할 수 있습니다.

스케줄 정보에 대해서는 간단한 논리 검사가 수행됩니다. 즉, 작업이 생성될 때 날짜 파라미터를 검사합니다. 데이터베이스는 종료 날짜가 시작 날짜보다 나중에 오는지 여부를 검사합니다. 시작 날짜가 이미 지나간 경우 시작 날짜는 현재 날짜로 변경됩니다.

인라인 파라미터를 사용하여 작업 생성

CREATE_JOB 프로시저의 인수에 실행할 작업의 코드 유형, 코드, 시작 시간 및 실행 빈도를 지정합니다.

```
-- Schedule a PL/SQL block every hour:

BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        job_name => 'JOB_NAME',
        job_type => 'PLSQL_BLOCK',
        job_action => 'BEGIN ...; END;',
        start_date => SYSTIMESTAMP,
        repeat_interval=>'FREQUENCY=HOURLY; INTERVAL=1',
        enabled => TRUE);
END;
/
```



Copyright © 2009, Oracle. All rights reserved.

인라인 파라미터를 사용하여 작업 생성

작업을 생성하면 DBMS_SCHEDULER.CREATE_JOB 프로시저를 사용하여 PL/SQL 블록, 내장 프로시저 또는 external 프로그램을 실행할 수 있습니다. CREATE_JOB 프로시저는 프로그램이나 스케줄 구성 요소를 생성할 필요 없이 직접 사용할 수 있습니다.

슬라이드 쇼의 예제는 모든 작업 세부 정보를 인라인으로 지정하는 방법을 보여줍니다.

CREATE_JOB 프로시저의 파라미터는 실행될 "항목", 스케줄 및 기타 작업 속성을 정의합니다.

실행될 항목을 정의하는 파라미터는 다음과 같습니다.

- job_type 파라미터는 다음 세 가지 값 중 하나일 수 있습니다.
 - PL/SQL 블록 또는 SQL 문의 PLSQL_BLOCK. 이 작업 유형은 인수를 사용할 수 없습니다.
 - 독립형 내장 프로시저 또는 패키지 프로시저의 STORED PROCEDURE. 프로시저는 작업에 제공되는 인수를 사용할 수 있습니다.
 - 실행 가능한 명령행 운영 체제 응용 프로그램의 EXECUTABLE
- 스케줄은 다음 파라미터를 사용하여 지정됩니다.
 - start_date는 시간 기록을 사용하고 repeat_interval은 달력 또는 PL/SQL 표현식을 사용하여 문자열로 지정됩니다. end_date는 지정할 수 있습니다.

참고: repeat_interval에 지정되는 문자열 식은 나중에 설명합니다. 위 예제는 작업이 1시간 간격으로 실행되도록 지정합니다.

프로그램을 사용하여 작업 생성

- **CREATE_PROGRAM을 사용하여 프로그램을 생성합니다.**

```
BEGIN
    DBMS_SCHEDULER.CREATE_PROGRAM(
        program_name => 'PROG_NAME',
        program_type => 'PLSQL_BLOCK',
        program_action => 'BEGIN ...; END;');
END;
```

- 오버로드된 CREATE_JOB 프로시저를 program_name 파라미터와 함께 사용합니다.

```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
        program_name => 'PROG_NAME',
        start_date => SYSTIMESTAMP,
        repeat_interval => 'FREQ=DAILY',
        enabled => TRUE);
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램을 사용하여 작업 생성

DBMS_SCHEDULER.CREATE_PROGRAM 프로시저는 고유한 이름을 지정해야 하는 프로그램을 정의합니다. 작업에 대한 프로그램을 별도로 생성하면 다음을 수행할 수 있습니다.

- 한 번 정의한 동작을 다른 작업에서 다시 사용
- PL/SQL 블록을 재생성할 필요 없이 작업 스케줄 변경
- 모든 작업을 변경하지 않고 실행된 프로그램 변경

프로그램 동작 문자열은 다음과 같은 program_type 파라미터 값에 따라 프로시저, 실행 파일 이름, PL/SQL 블록을 지정합니다.

- 익명 블록이나 SQL 문을 실행하기 위한 PLSQL_BLOCK
- PL/SQL, Java 또는 C와 같은 내장 프로시저를 실행하기 위한 STORED_PROCEDURE
- 운영 체제 명령행 프로그램을 실행하기 위한 EXECUTABLE

위 슬라이드에 표시된 예제는 익명 PL/SQL 블록을 호출하는 방법을 보여줍니다. 다음 예제에서와 같이 프로그램 내에서 external 프로시저를 호출할 수도 있습니다.

```
DBMS_SCHEDULER.CREATE_PROGRAM(program_name => 'GET_DATE',
    program_action => '/usr/local/bin/date',
    program_type => 'EXECUTABLE');
```

프로그램으로 작업을 생성하려면 위 슬라이드와 같이 DBMS_SCHEDULER.CREATE_JOB 프로시저에 대한 호출에서 program_name 인수에 프로그램 이름을 지정하십시오.

인수를 사용하여 프로그램에 대한 작업 생성

- **프로그램 생성:**

```
DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name => 'PROG_NAME',
    program_type => 'STORED_PROCEDURE',
    program_action => 'EMP_REPORT');
```

- **인수 정의:**

```
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
    program_name => 'PROG_NAME',
    argument_name => 'DEPT_ID',
    argument_position=> 1, argument_type=> 'NUMBER',
    default_value => '50');
```

- **인수 개수를 지정하여 작업 생성:**

```
DBMS_SCHEDULER.CREATE_JOB('JOB_NAME', program_name
    => 'PROG_NAME', start_date => SYSTIMESTAMP,
    repeat_interval => 'FREQ=DAILY',
    number_of_arguments => 1, enabled => TRUE);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

인수를 사용하여 프로그램에 대한 작업 생성

PL/SQL이나 external 프로시저 같은 프로그램은 입력 인수가 필요할 수도 있습니다.

DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT 프로시저를 사용하여 기존 프로그램에 대한 인수를 정의할 수 있습니다. DEFINE_PROGRAM_ARGUMENT 프로시저 파라미터는 다음과 같습니다.

- program_name은 변경될 기존 프로그램을 지정합니다.
- argument_name은 프로그램에 대한 고유 인수 이름을 지정합니다.
- argument_position은 프로그램이 호출될 때 인수가 전달되는 위치를 지정합니다.
- argument_type은 호출된 프로그램에 전달되는 인수 값의 데이터 유형을 지정합니다.
- default_value는 프로그램을 스케줄링(scheduling)하는 작업이 값을 제공하지 않을 경우 프로그램에 제공되는 기본값을 지정합니다.

위 슬라이드는 한 개의 인수를 사용하여 프로그램을 실행하는 작업을 생성하는 방법을 보여줍니다. 프로그램 인수의 기본값은 50입니다. 작업의 프로그램 인수 값을 변경하려면 다음과 같이 사용하십시오.

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE(
    job_name => 'JOB_NAME',
    argument_name => 'DEPT_ID', argument_value => '80');
```

스케줄을 사용하여 작업 생성

- CREATE_SCHEDULE을 사용하여 스케줄을 생성합니다.

```
BEGIN
    DBMS_SCHEDULER.CREATE_SCHEDULE('SCHED_NAME',
        start_date => SYSTIMESTAMP,
        repeat_interval => 'FREQ=DAILY',
        end_date => SYSTIMESTAMP +15);
END;
```

- schedule_name 파라미터에서 스케줄을 참조하여 CREATE_JOB을 사용합니다.

```
BEGIN
    DBMS_SCHEDULER.CREATE_JOB('JOB_NAME',
        schedule_name => 'SCHED_NAME',
        job_type => 'PLSQL_BLOCK',
        job_action => 'BEGIN ...; END;',
        enabled => TRUE);
END;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

스케줄을 사용하여 작업 생성

매번 스케줄 세부 정보를 지정할 필요 없이 여러 작업에 적용될 수 있는 공통된 스케줄을 생성할 수 있습니다. 다음은 스케줄을 생성하면 얻게 되는 이점입니다.

- 재사용이 가능하며 여러 작업에 할당할 수 있습니다.
- 스케줄을 변경하면 해당 스케줄을 사용하는 모든 작업이 영향을 받습니다. 작업 스케줄은 여러 번이 아닌 한 번만 변경됩니다.

스케줄은 가장 가까운 초 단위 시간까지만 정밀도를 유지합니다. TIMESTAMP 데이터 유형이 더 정확한 자릿수까지 표시하더라도 스케줄러가 초과된 자릿수를 반올림하여 가장 가까운 초 단위 시간으로 맞춥니다.

스케줄러의 시작 시간과 종료 시간은 TIMESTAMP 데이터 유형을 사용하여 지정됩니다. 저장된 스케줄의 end_date는 이 날짜가 경과하면 스케줄이 더 이상 유효하지 않게 되는 날짜입니다.

위 예제의 스케줄은 지정된 작업에서 해당 스케줄을 사용한 이후부터 15일 동안 유효합니다.

저장된 스케줄의 repeat_interval은 Calendaring 표현식을 사용하여 생성해야 합니다. repeat_interval의 NULL 값은 작업이 한 번만 실행되도록 지정합니다.

참고: PL/SQL 표현식은 저장된 스케줄의 반복 간격을 표현하는 데 사용할 수 없습니다.

작업 반복 간격 설정

- Calendaring 표현식 사용:

```
repeat_interval=> 'FREQ=HOURLY; INTERVAL=4'  
repeat_interval=> 'FREQ=DAILY'  
repeat_interval=> 'FREQ=MINUTELY; INTERVAL=15'  
repeat_interval=> 'FREQ=YEARLY;  
    BYMONTH=MAR,JUN,SEP,DEC;  
    BYMONTHDAY=15'
```

- PL/SQL 표현식 사용:

```
repeat_interval=> 'SYSDATE + 36/24'  
repeat_interval=> 'SYSDATE + 1'  
repeat_interval=> 'SYSDATE + 15/(24*60)'
```



Copyright © 2009, Oracle. All rights reserved.

명명된 프로그램과 스케줄을 사용하여 작업 생성

- **CREATE_PROGRAM** 프로시저를 사용하여 PROG_NAME이라는 명명된 프로그램을 생성합니다.
- **CREATE_SCHEDULE** 프로시저를 사용하여 SCHED_NAME이라는 명명된 스케줄을 생성합니다.
- 명명된 프로그램과 스케줄을 참조하는 작업을 생성합니다.

```

BEGIN
    DBMS_SCHEDULER.CREATE_JOB(
        'JOB_NAME',
        program_name => 'PROG_NAME',
        schedule_name => 'SCHED_NAME',
        enabled => TRUE);
END;
/

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

명명된 프로그램과 스케줄을 사용하여 작업 생성

슬라이드 쇼의 예제에서는 DBMS_SCHEDULER.CREATE_JOB 프로시저 사용의 마지막 형태를 보여줍니다. 이 예제에서 명명된 프로그램(PROG_NAME) 및 스케줄(SCHED_NAME)은 DBMS_SCHEDULER.CREATE_JOB 프로시저에 대한 호출에서 각각의 파라미터에 지정됩니다. 이 예제를 통해 미리 정의된 프로그램과 스케줄을 사용할 경우 얼마나 쉽게 작업을 생성할 수 있는지 확인할 수 있습니다.

일부 작업과 스케줄은 본 과정에서 다루기에 너무 복잡합니다. 예를 들어, 반복 시간 계획 윈도우를 생성한 다음 이 윈도우와 Resource Plan을 연관시킬 수 있습니다. Resource Plan은 실행 윈도우에 의해 정의된 기간 동안 필요한 리소스에 대한 속성을 정의합니다.

자세한 내용은 *Oracle Database 11g: Configure and Manage Jobs with the Scheduler* 온라인 과정을 참조하십시오.

작업 관리

- **작업 실행**

```
DBMS_SCHEDULER.RUN_JOB( 'SCHEMA.JOB_NAME' );
```

- **작업 정지**

```
DBMS_SCHEDULER.STOP_JOB( 'SCHEMA.JOB_NAME' );
```

- **작업 삭제(현재 실행 중일 경우에도 삭제되는 설정):**

```
DBMS_SCHEDULER.DROP_JOB( 'JOB_NAME' , TRUE );
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

작업 관리

작업을 생성한 후에는 다음을 수행할 수 있습니다.

- 작업 이름을 지정하는 RUN_JOB 프로시저를 호출하여 작업을 실행합니다. 작업이 현재 세션에서 즉시 실행됩니다.
- STOP_JOB 프로시저를 사용하여 작업을 정지합니다. 현재 실행 중인 작업이 즉시 정지됩니다. STOP_JOB 프로시저는 다음 두 가지 인수를 사용합니다.
 - **job_name:** 정지할 작업의 이름입니다.
 - **force:** 작업을 무리 없이 종료하려고 시도합니다. 이 시도가 실패하여 force가 TRUE로 설정될 경우 작업 슬레이브가 종료됩니다. 기본값은 FALSE입니다. force를 사용하려면 MANAGE_SCHEDULER 시스템 권한이 있어야 합니다.
- DROP_JOB 프로시저를 사용하여 작업을 삭제합니다. 이 프로시저는 다음 두 가지 인수를 사용합니다.
 - **job_name:** 삭제할 작업의 이름입니다.
 - **force:** 작업이 현재 실행 중일 경우 작업을 정지한 다음 삭제할 것인지 여부를 나타냅니다. 기본값은 FALSE입니다.

작업 관리(계속)

DROP_JOB 프로시저가 호출되었는데 지정된 작업이 현재 실행 중일 경우 force 옵션이 TRUE로 설정되어 있어야 명령이 성공합니다. force 옵션이 TRUE로 설정되어 있을 경우 실행 중인 작업의 Instance가 정지된 다음 작업이 삭제됩니다.

참고: 다른 유저가 소유한 작업을 실행, 정지 또는 삭제하려면 해당 작업에 대한 ALTER 권한 또는 CREATE ANY JOB 시스템 권한을 가지고 있어야 합니다.

데이터 딕셔너리 뷰

- [DBA | ALL | USER]_SCHEDULER_JOBS
- [DBA | ALL | USER]_SCHEDULER_RUNNING_JOBS
- [DBA | ALL]_SCHEDULER_JOB_CLASSES
- [DBA | ALL | USER]_SCHEDULER_JOB_LOG
- [DBA | ALL | USER]_SCHEDULER_JOB_RUN_DETAILS
- [DBA | ALL | USER]_SCHEDULER_PROGRAMS

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리 뷰

DBA_SCHEDULER_JOB_LOG 뷰는 성공 여부와 관계없이 완료된 작업 Instance를 모두 표시합니다.

작업 상태를 보려면 다음 Query를 사용하십시오.

```
SELECT job_name, program_name, job_type, state
  FROM USER_SCHEDULER_JOBS;
```

작업이 실행 중인 Instance를 확인하려면 다음 Query를 사용합니다.

```
SELECT owner, job_name, running_instance,
       resource_consumer_group
  FROM DBA_SCHEDULER_RUNNING_JOBS;
```

실행된 작업에 대한 정보를 확인하려면 다음 Query를 사용하십시오.

```
SELECT job_name, instance_id, req_start_date,
       actual_start_date, status
  FROM ALL_SCHEDULER_JOB_RUN_DETAILS;
```

작업 상태를 확인하려면 다음 Query를 사용하십시오.

```
SELECT job_name, status, error#, run_duration, cpu_used
  FROM USER_SCHEDULER_JOB_RUN_DETAILS;
```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- HTP 패키지를 사용하여 간단한 웹 페이지 생성
- DBMS_SCHEDULER 패키지를 호출하여 실행할 PL/SQL 코드 스케줄링(scheduling)

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 DBMS_SCHEDULER 패키지와 함께 실행되도록 PL/SQL 및 external 코드를 스케줄링(scheduling)하는 방법을 배웠습니다.

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