

Oracle Database 11g: SQL Fundamentals II(한글판)

블록 II • 학생용

D49994KR20

Edition 2.0

2009년 12월

D64455

ORACLE

저자

Chaitanya Koratamaddi

Brian Pottle

Tulika Srivastava

기술 제공자 및 검토자

Claire Bennett

Ken Cooper

Yanti Chang

Laszlo Czinkoczeki

Burt Demchick

Gerlinde Frenzen

Joel Goodman

Laura Garza

Richard Green

Nancy Greenberg

Akira Kinutani

Wendy Lo

Isabelle Marchand

Timothy Mcglue

Alan Paulson

Srinivas Putrevu

Bryan Roberts

Clinton Shaffer

Abhishek Singh

Jenny Tsai Smith

James Spiller

Lori Tritz

Lex van der Werff

Marcie Young

편집자

Amitha Narayan

Daniel Milne

그래픽 디자이너

Satish Bettegowda

발행인

Veena Narasimhan

Copyright © 2009, Oracle. All rights reserved.

Disclaimer

본 과정은 Release 11g에 예정된 기능과 향상된 부분을 개략적으로 설명합니다. 본 과정은 11g 업그레이드로 얻을 수 있는 업무상 혜택을 평가하여 IT 프로젝트를 계획하는 데 도움을 줄 목적으로만 제공됩니다.

과정 실습 및 인쇄 자료를 포함한 이 과정의 모든 형태는 오라클의 독점 자산인 독점적 정보를 포함하고 있습니다. 오라클의 사전 서면 동의 없이 본 과정과 여기에 포함된 정보를 오라클 외부의 다른 사람에게 공개, 복사, 재생산 또는 배포할 수 없습니다. 본 과정과 그 내용은 라이선스 계약에 포함되지 않으며 오라클 또는 그 자회사와의 어떠한 계약에도 편입될 수 없습니다.

본 과정은 오직 정보를 제공하기 위한 것이며 설명된 제품 기능의 구현 및 업그레이드를 계획하는 데 도움을 줄 목적으로만 제공됩니다. 이는 어떠한 자료, 코드 또는 기능을 제공할 것이라는 약속이 아니므로 구입 결정에 본 문서를 의지해서는 안 됩니다. 본 문서에 설명된 기능의 개발, 출시 및 시기에 대한 재량권은 전적으로 오라클에 있습니다.

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, 라이선스, 우편 발송, 전송 또는 배포할 수 없습니다.

본 문서의 내용은 사전 공지 없이 변경될 수 있습니다. 만일 본 문서의 내용상 문제점을 발견하면 서면으로 통지해 주시기 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
선수 과정	I-5
과정 일정	I-6
단원 내용	I-7
본 과정에 사용되는 테이블	I-8
본 과정에 사용되는 부록	I-9
개발 환경	I-10
단원 내용	I-11
데이터 제한 검토	I-12
데이터 정렬 검토	I-13
SQL 함수 검토	I-14
단일 행 함수 검토	I-15
그룹 함수의 유형 검토	I-16
Subquery 사용 검토	I-17
데이터 조작 검토	I-18
단원 내용	I-19
Oracle Database 11g SQL 설명서	I-20
추가 자료	I-21
요약	I-22
연습 I: 개요	I-23

1 유저 액세스 제어

목표	1-2
단원 내용	1-3
유저 액세스 제어	1-4
권한	1-5
시스템 권한	1-6
유저 생성	1-7
유저 시스템 권한	1-8
시스템 권한 부여	1-9
단원 내용	1-10

롤이란?	1-11
롤 생성 및 롤에 권한 부여	1-12
암호 변경	1-13
단원 내용	1-14
객체 권한	1-15
객체 권한 부여	1-17
권한 전달	1-18
부여된 권한 확인	1-19
단원 내용	1-20
객체 권한 취소	1-21
퀴즈	1-23
요약	1-24
연습 1: 개요	1-25

2 스키마 객체 관리

목표	2-2
단원 내용	2-3
ALTER TABLE 문	2-4
열 추가	2-6
열 수정	2-7
열 삭제	2-8
SET UNUSED 옵션	2-9
단원 내용	2-11
제약 조건 구문 추가	2-12
제약 조건 추가	2-13
ON DELETE 절	2-14
제약 조건 지연	2-15
INITIALLY DEFERRED 와 INITIALLY IMMEDIATE 의 차이	2-16
제약 조건 삭제	2-18
제약 조건 비활성화	2-19
제약 조건 활성화	2-20
계단식 제약 조건	2-22
테이블 열 및 제약 조건 이름 바꾸기	2-24
단원 내용	2-25
인덱스 개요	2-26
CREATE TABLE 문을 사용한 CREATE INDEX	2-27
함수 기반 인덱스	2-29
인덱스 제거	2-30
DROP TABLE ... PURGE	2-31

단원 내용	2-32
FLASHBACK TABLE 문	2-33
FLASHBACK TABLE 문 사용	2-35
단원 내용	2-36
임시 테이블	2-37
임시 테이블 생성	2-38
단원 내용	2-39
External Table	2-40
External Table 의 디렉토리 생성	2-42
External Table 생성	2-44
ORACLE_LOADER 를 사용하여 External Table 생성	2-46
External Table 조회	2-48
ORACLE_DATAPUMP 를 사용하여 External Table 생성: 예제	2-49
퀴즈	2-50
요약	2-52
연습 2: 개요	2-53

3 데이터 디렉터리 뷰를 사용하여 객체 관리

목표	3-2
단원 내용	3-3
데이터 디렉터리	3-4
데이터 디렉터리 구조	3-5
디렉터리 뷰 사용 방법	3-7
USER_OBJECTS 및 ALL_OBJECTS 뷰	3-8
USER_OBJECTS 뷰	3-9
단원 내용	3-10
테이블 정보	3-11
열 정보	3-12
제약 조건 정보	3-14
USER_CONSTRAINTS: 예제	3-15
USER_CONS_COLUMNS 조회	3-16
단원 내용	3-17
뷰 정보	3-18
시퀀스 정보	3-19
시퀀스 확인	3-20
인덱스 정보	3-21
USER_INDEXES: 예제	3-22
USER_IND_COLUMNS 조회	3-23
동의어 정보	3-24

단원 내용 3-25
 테이블에 주석 추가 3-26
 퀴즈 3-27
 요약 3-28
 연습 3: 개요 3-29

4 대형 데이터 집합 조작

목표 4-2
 단원 내용 4-3
 Subquery 를 사용하여 데이터 조작 4-4
 Subquery 를 소스로 사용하여 데이터 검색 4-5
 Subquery 를 대상으로 사용하여 삽입 4-7
 DML 문에 WITH CHECK OPTION 키워드 사용 4-9
 단원 내용 4-11
 명시적 기본값 기능의 개요 4-12
 명시적 기본값 사용 4-13
 다른 테이블에서 행 복사 4-14
 단원 내용 4-15
 다중 테이블 INSERT 문의 개요 4-16
 다중 테이블 INSERT 문의 유형 4-18
 다중 테이블 INSERT 문 4-19
 무조건 INSERT ALL 4-21
 조건부 INSERT ALL: 예제 4-23
 조건부 INSERT ALL 4-24
 조건부 INSERT FIRST: 예제 4-26
 조건부 INSERT FIRST 4-27
 피벗팅 INSERT 4-29
 단원 내용 4-32
 MERGE 문 4-33
 MERGE 문 구문 4-34
 행 병합: 예제 4-35
 단원 내용 4-38
 데이터 변경 사항 추적 4-39
 Flashback Version Query 의 예 4-40
 VERSIONS BETWEEN 절 4-42
 퀴즈 4-43
 요약 4-44
 연습 4: 개요 4-45

5 다른 시간대에서 데이터 관리

목표 5-2

단원 내용 5-3

시간대 5-4

TIME_ZONE 세션 파라미터 5-5

CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP 5-6

세션의 시간대에서 날짜와 시간 비교 5-7

DBTIMEZONE 및 SESSIONTIMEZONE 5-9

TIMESTAMP 데이터 유형 5-10

TIMESTAMP 필드 5-11

DATE 와 TIMESTAMP 의 차이 5-12

TIMESTAMP 데이터 유형 비교 5-13

단원 내용 5-14

INTERVAL 데이터 유형 5-15

INTERVAL 필드 5-17

INTERVAL YEAR TO MONTH: 예제 5-18

INTERVAL DAY TO SECOND 데이터 유형: 예제 5-20

단원 내용 5-21

EXTRACT 5-22

TZ_OFFSET 5-23

FROM_TZ 5-25

TO_TIMESTAMP 5-26

TO_YMINTERVAL 5-27

TO_DSINTERVAL 5-28

일광 절약 시간 5-29

퀴즈 5-31

요약 5-32

연습 5: 개요 5-33

6 Subquery 를 사용하여 데이터 검색

목표 6-2

단원 내용 6-3

여러 열 Subquery 6-4

열 비교 6-5

쌍 방식 비교 Subquery 6-6

비쌍 방식 비교 Subquery 6-8

단원 내용 6-10

스칼라 Subquery 표현식 6-11

스칼라 Subquery: 예제 6-12

단원 내용 6-14
 Correlated Subquery 6-15
 Correlated Subquery 사용 6-17
 단원 내용 6-19
 EXISTS 연산자 사용 6-20
 사원이 없는 부서 모두 찾기 6-22
 Correlated UPDATE 6-23
 Correlated UPDATE 사용 6-24
 Correlated DELETE 6-26
 Correlated DELETE 사용 6-27
 단원 내용 6-28
 WITH 절 6-29
 WITH 절: 예제 6-30
 Recursive WITH 절 6-32
 Recursive WITH 절: 예제 6-33
 퀴즈 6-34
 요약 6-35
 연습 6: 개요 6-37

7 정규식 지원

목표 7-2
 단원 내용 7-3
 정규식이란? 7-4
 정규식 사용 시 이점 7-5
 SQL 및 PL/SQL에서 정규식 함수 및 조건 사용 7-6
 단원 내용 7-7
 메타 문자란? 7-8
 정규식에서 메타 문자 사용 7-9
 단원 내용 7-11
 정규식 함수 및 조건: 구문 7-12
 REGEXP_LIKE 조건을 사용하여 기본 검색 수행 7-13
 REGEXP_REPLACE 함수를 사용하여 패턴 대체 7-14
 REGEXP_INSTR 함수를 사용하여 패턴 찾기 7-15
 REGEXP_SUBSTR 함수를 사용하여 부분 문자열 추출 7-16
 단원 내용 7-17
 하위식 7-18
 정규식을 지원하는 하위식 사용 7-19
 n 번째 하위식에 액세스하는 이유 7-20
 REGEXP_SUBSTR: 예제 7-21

단원 내용 7-22
REGEXP_COUNT 함수 사용 7-23
정규식 및 Check 제약 조건: 예제 7-24
퀴즈 7-25
요약 7-26
연습 7: 개요 7-27

부록 A: 연습 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/SQL 에 Java 게시(Publishing) E-16

JDeveloper 11g에 대해 자세히 배울 수 있는 방법 E-17

요약 E-18

부록 F: 관련 데이터를 그룹화하여 보고서 생성

목표 F-2

그룹 함수 검토 F-3

GROUP BY 절 검토 F-4

HAVING 절 검토 F-5

GROUP BY 에 ROLLUP 및 CUBE 연산자 사용 F-6

ROLLUP 연산자 F-7

ROLLUP 연산자: 예제 F-8

CUBE 연산자 F-9

CUBE 연산자: 예제 F-10

GROUPING 함수 F-11

GROUPING 함수: 예제 F-12

GROUPING SETS F-13

GROUPING SETS: 예제 F-15

조합 열 F-17

조합 열: 예제 F-19

연결된 그룹화 F-21

연결된 그룹화: 예제 F-22

요약 F-23

부록 G: 계층적 검색

목표 G-2

EMPLOYEES 테이블의 예제 데이터 G-3

일반 트리 구조 G-4

Hierarchical Query G-5

트리 탐색 G-6

트리 탐색: 상향식 G-8

트리 탐색: 하향식 G-9

LEVEL 의사 열로 행 순위 지정 G-10

LEVEL 및 LPAD 를 사용하여 계층 보고서 형식 지정 G-11

분기 제거 G-13

요약 G-14

부록 H: 고급 스크립트 작성

목표 H-2

SQL 을 사용하여 SQL 생성 H-3

기본 스크립트 작성 H-5

환경 제어 H-6

전체 그림 H-7

테이블 내용을 파일로 덤프 H-8

동적 술어 생성 H-10

요약 H-12

부록 I: 오라클 데이터베이스 구조 구성 요소

목표 I-2

오라클 데이터베이스 구조: 개요 I-3

오라클 데이터베이스 서버 구조 I-4

데이터베이스에 연결 I-5

오라클 데이터베이스와 상호 작용 I-6

Oracle 메모리 구조 I-8

프로세스 구조 I-10

데이터베이스 기록자 프로세스 I-12

로그 기록자 프로세스 I-13

체크포인트 프로세스 I-14

시스템 모니터 프로세스 I-15

프로세스 모니터 프로세스 I-16

오라클 데이터베이스 저장 영역 구조 I-17

논리적 및 물리적 데이터베이스 구조 I-19

SQL 문 처리 I-21

Query 처리 I-22

Shared Pool I-23

데이터베이스 버퍼 캐시 I-25

프로그램 글로벌 영역(PGA) I-26

DML 문 처리 I-27

리두 로그 버퍼 I-29

롤백 세그먼트 I-30

COMMIT 처리 I-31

오라클 데이터베이스 구조 요약 I-33

추가 연습**추가 연습 해답**

부록 A

연습 및 해답

목차

단원 I의 연습 및 해답	3
연습 I-1: SQL Developer 리소스에 액세스	4
연습 I-2: SQL Developer 사용	5
연습 문제 해답 I-1: SQL Developer 리소스에 액세스	7
연습 문제 해답 I-2: SQL Developer 사용	8
단원 1의 연습 및 해답	17
연습 1-1: 유저 액세스 제어	17
연습 문제 해답 1-1: 유저 액세스 제어	20
단원 2의 연습 및 해답	24
연습 2-1: 스키마 객체 관리	24
연습 문제 해답 2-1: 스키마 객체 관리	30
단원 3의 연습 및 해답	38
연습 3-1: 데이터 디셔너리 뷰로 객체 관리	38
연습 문제 해답 3-1: 데이터 디셔너리 뷰로 객체 관리	42
단원 4의 연습 및 해답	45
연습 4-1: 대형 객체 집합 조작	45
연습 문제 해답 4-1: 대형 객체 집합 조작	50
단원 5의 연습 및 해답	54
연습 5-1: 서로 다른 시간대에서의 데이터 관리	54
연습 문제 해답 5-1: 서로 다른 시간대에서의 데이터 관리	57
단원 6의 연습 및 해답	60
연습 6-1: Subquery 를 사용하여 데이터 검색	60
연습 문제 해답 6-1: Subquery 를 사용하여 데이터 검색	63
단원 7의 연습 및 해답	66
연습 7-1: 정규식 지원	66
연습 문제 해답 7-1: 정규식 지원	68

이 연습에서는 사용 가능한 SQL Developer 리소스를 검토합니다. 본 과정에서 사용할 유저 계정에 대해 배웁니다. 그런 다음 SQL Developer를 시작하여 새 데이터베이스 연결을 생성하고 HR 테이블을 탐색합니다. 또한 SQL Developer의 일부 환경 설정을 구성하고, SQL 문을 실행하고, SQL Worksheet를 사용하여 익명 PL/SQL 블록을 실행합니다. 끝으로 Oracle Database 11g 설명서와 본 과정에서 활용할 수 있는 기타 유용한 웹 사이트에 액세스하여 책갈피를 지정합니다.

연습 I-1: SQL Developer 리소스에 액세스

이 연습에서는 다음을 수행합니다.

- 1) SQL Developer 홈 페이지에 액세스합니다.
 - a) 다음 위치의 SQL Developer 홈 페이지에 온라인으로 액세스합니다.
http://www.oracle.com/technology/products/database/sql_developer/index.html
 - b) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.
- 2) 다음 위치의 SQL Developer 자습서에 온라인으로 액세스합니다.
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>. 그런 다음 아래 섹션 및 관련 데모를 검토합니다.
 - a) What to Do First
 - b) Working with Database Objects
 - c) Accessing Data

연습 I-2: SQL Developer 사용

- 1) 바탕 화면 아이콘을 사용하여 SQL Developer를 시작합니다.
- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.
 - a) Connection Name: myconnection
 - b) Username: oraxx(xx: PC 번호). ora21-ora40의 계정 범위 중에서 한 개의 ora 계정을 할당해 줄 것을 강사에게 요청하십시오.
 - c) Password: oraxx
 - d) Hostname: localhost
 - e) Port: 1521
 - f) SID: orcl(또는 강사가 알려준 값)
- 3) 새 연결을 테스트합니다. 상태가 Success이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.
 - a) New/Select Database Connection window의 Test 버튼을 누릅니다.
 - b) 상태가 Success이면 Connect 버튼을 누릅니다.
- 4) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.
 - a) 옆에 있는 더하기(+) 기호를 눌러 myconnection 연결을 확장합니다.
 - b) 옆에 있는 더하기(+) 기호를 눌러 Tables 아이콘을 확장합니다.
 - c) EMPLOYEES 테이블의 구조를 표시합니다.
 - d) DEPARTMENTS 테이블의 데이터를 확인합니다.
- 5) 몇 가지 기본 SELECT 문을 실행하여 SQL Worksheet 영역에 있는 EMPLOYEES 테이블의 데이터를 query합니다. Execute Statement(또는 F9)와 Run Script(또는 F5) 아이콘을 둘 다 사용하여 SELECT 문을 실행합니다. 해당 탭 페이지에서 SELECT 문을 실행하는 두 방법의 결과를 검토합니다.
 - a) 급여가 \$3,000 이하인 사원의 성과 급여를 선택하는 query를 작성합니다.
 - b) 커미션을 받을 자격이 없는 모든 사원의 성, 직무 ID 및 커미션을 표시하는 query를 작성합니다.
- 6) 스크립트 경로 환경 설정을 /home/oracle/labs/sql2로 설정합니다.
 - a) Tools > Preferences > Database > Worksheet Parameters를 선택합니다.
 - b) Select default path to look for scripts 필드에 값을 입력합니다.

연습 I-2: SQL Developer 사용(계속)

- 7) Enter SQL Statement 상자에 다음과 같이 입력합니다.

```
SELECT employee_id, first_name, last_name,  
       FROM employees;
```
- 8) File > Save As 메뉴 항목을 사용하여 SQL 문을 스크립트 파일로 저장합니다.
 - a) File > Save As를 선택합니다.
 - b) 파일 이름을 `intro_test.sql`로 지정합니다.
 - c) `/home/oracle/labs/sql2/labs` 폴더에 파일을 저장합니다.
- 9) `/home/oracle/labs/sql2/labs` 폴더에서 `confidence.sql`을 열어 실행한 다음 결과를 확인합니다.

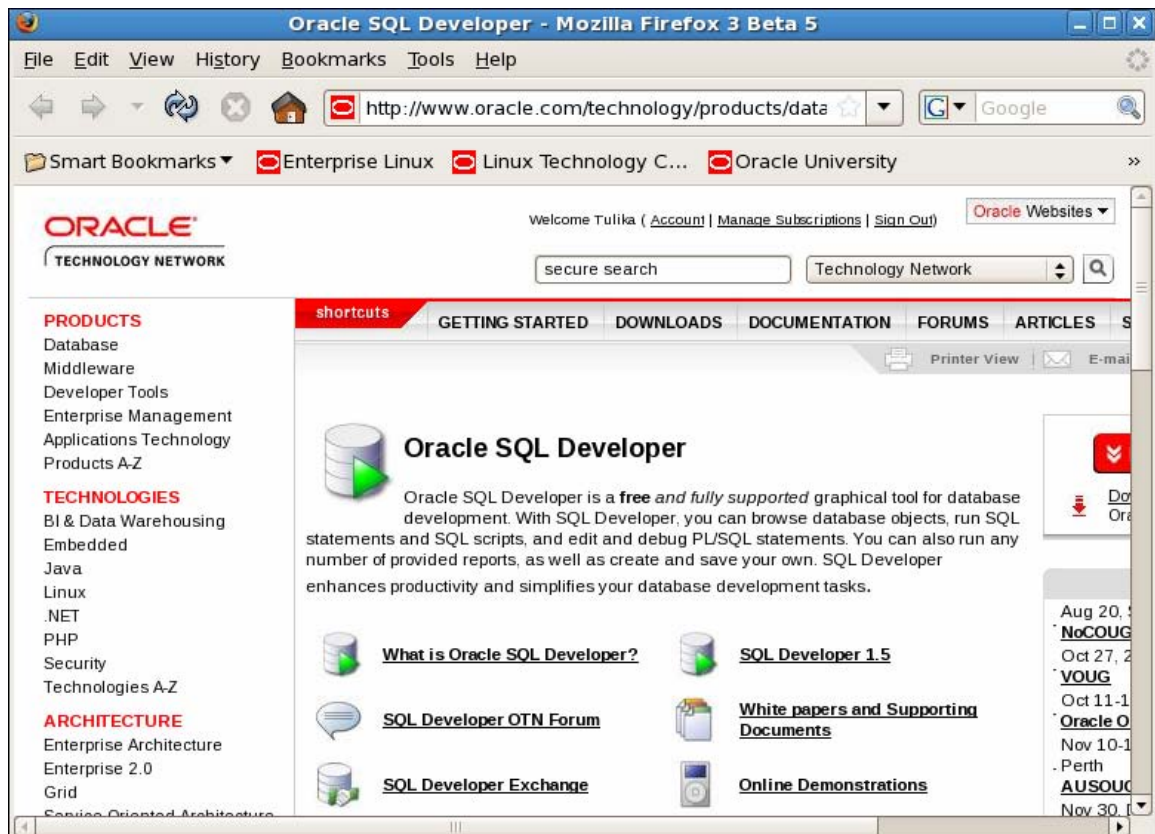
연습 문제 해답 I-1: SQL Developer 리소스에 액세스

1) SQL Developer 홈 페이지에 액세스합니다.

a) 다음 위치의 SQL Developer 홈 페이지에 온라인으로 액세스합니다.

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

다음과 같이 SQL Developer 홈 페이지가 표시됩니다.



b) 나중에 쉽게 액세스할 수 있도록 페이지에 책갈피를 지정합니다.

2) 다음 위치의 SQL Developer 홈 페이지에 온라인으로 액세스합니다.

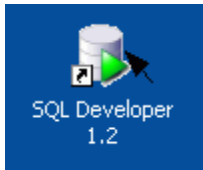
<http://st-curriculum.oracle.com/tutorial/SQLDeveloper/index.htm>

그런 다음 아래 섹션 및 관련 데모를 검토합니다.

- What to Do First
- Working with Database Objects
- Accessing Data

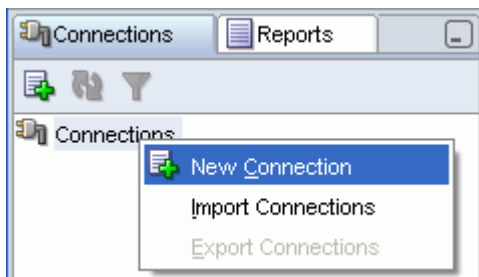
연습 문제 해답 1-2: SQL Developer 사용

1) 바탕 화면 아이콘을 사용하여 SQL Developer를 시작합니다.



2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

- a) Connection Name: myconnection
- b) Username: oraxx(ora21-ora40의 계정 범위 중에서 한 개의 ora 계정을 할당해 줄 것을 강사에게 요청하십시오.)
- c) Password: oraxx
- d) Hostname: localhost
- e) Port: 1521
- f) SID: orcl(또는 강사가 알려준 값)



연습 문제 해답 1-2: SQL Developer 사용(계속)

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

a) New/Select Database Connection window에서 Test 버튼을 누릅니다.

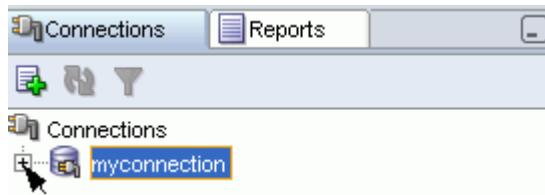
b) Status가 Success이면 Connect 버튼을 누릅니다.

연습 문제 해답 1-2: SQL Developer 사용(계속)

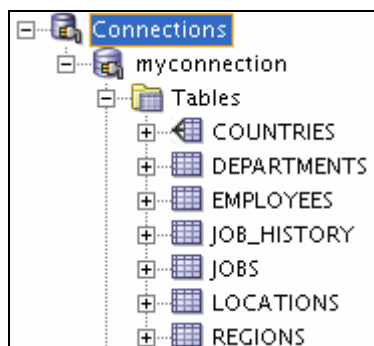
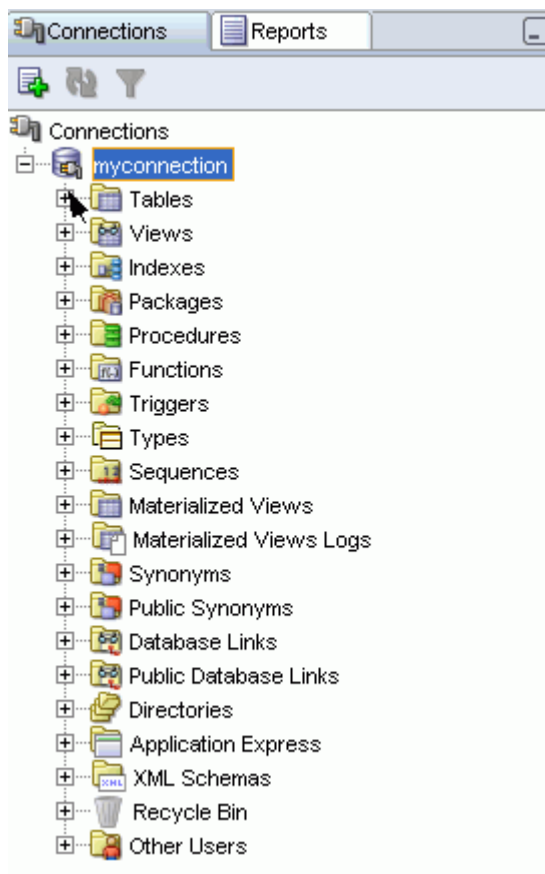
테이블 탐색

4) EMPLOYEES 테이블의 구조를 탐색하고 데이터를 표시합니다.

a) 옆에 있는 더하기 기호를 눌러 myconnection 연결을 확장합니다.



b) 옆에 있는 더하기 기호를 눌러 Tables 아이콘을 확장합니다.



연습 문제 해답 1-2: SQL Developer 사용(계속)

c) EMPLOYEES 테이블의 구조를 표시합니다.

EMPLOYEES 테이블을 누릅니다. Columns 탭에 EMPLOYEES 테이블의 열이 다음과 같이 표시됩니다.

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1	Primary key of employee
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)	First name of the employee
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3	(null)	Last name of the employee
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
HIRE_DATE	DATE	No	(null)	6	(null)	Date when the employee was hired
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7	(null)	Current job of the employee
SALARY	NUMBER(8,2)	Yes	(null)	8	(null)	Monthly salary of the employee
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9	(null)	Commission percentage of the employee
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10	(null)	Manager id of the employee
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11	(null)	Department id where employee works

d) DEPARTMENTS 테이블의 데이터를 확인합니다.

Connections navigator에서 **DEPARTMENTS** 테이블을 누릅니다. 그런 다음 Data 탭을 누릅니다.

	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

연습 문제 해답 1-2: SQL Developer 사용(계속)

- 5) 몇 가지 기본 SELECT 문을 실행하여 SQL Worksheet 영역에 있는 EMPLOYEES 테이블의 데이터를 query합니다. Execute Statement(또는 F9)와 Run Script 아이콘(또는 F5)을 둘 다 사용하여 SELECT 문을 실행합니다. 해당 탭 페이지에서 SELECT 문을 실행하는 두 방법의 결과를 검토합니다.

- a) 급여가 \$3,000 이하인 사원의 성 및 급여를 선택하는 query를 작성합니다.

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

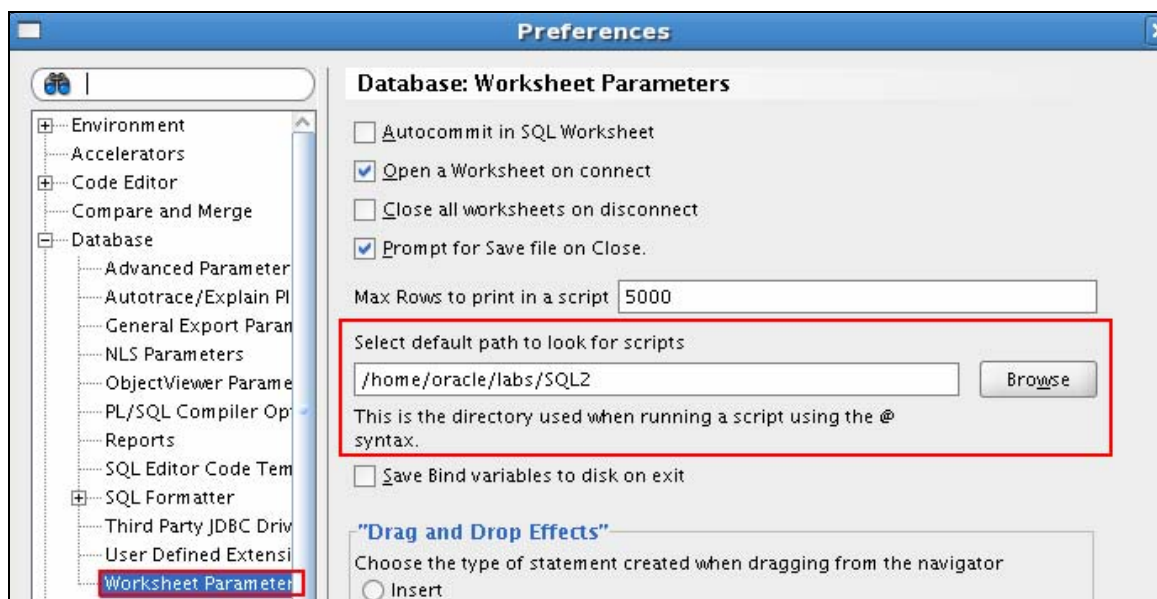
- b) 커미션을 받을 자격이 없는 모든 사원의 성, 직무 ID 및 커미션을 표시하는 query를 작성합니다.

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_pct IS NULL;
```

- 6) 스크립트 경로 환경 설정을 /home/oracle/labs/sql2로 설정합니다.

- a) Tools > Preferences > Database > Worksheet Parameters를 선택합니다.

- b) Select default path to look for scripts 필드에 값을 입력합니다. 그런 다음 OK를 누릅니다.



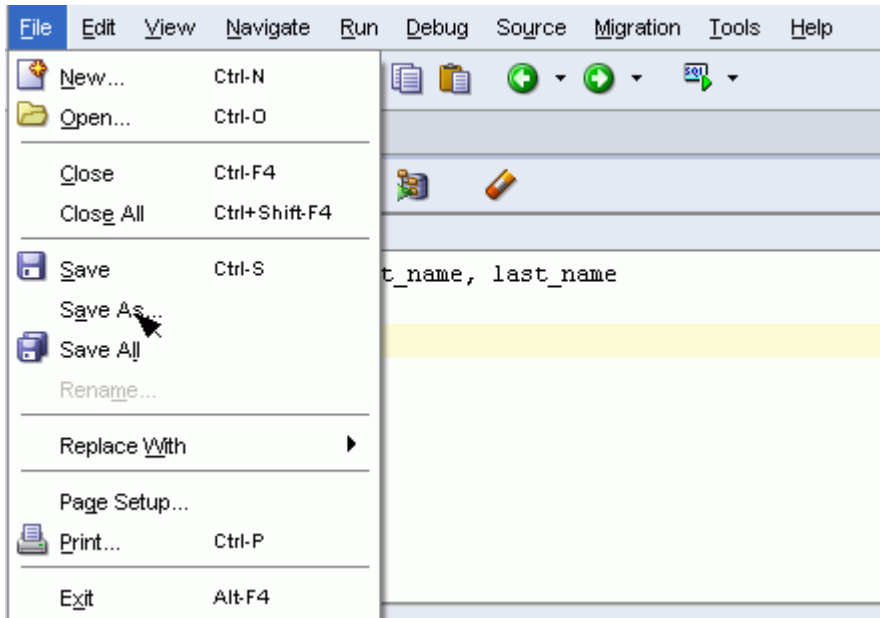
연습 문제 해답 I-2: SQL Developer 사용(계속)

7) 다음 SQL 문을 입력합니다.

```
SELECT employee_id, first_name, last_name
FROM employees;
```

8) File > Save As 메뉴 항목을 사용하여 SQL 문을 스크립트 파일로 저장합니다.

a) File > Save As를 선택합니다.

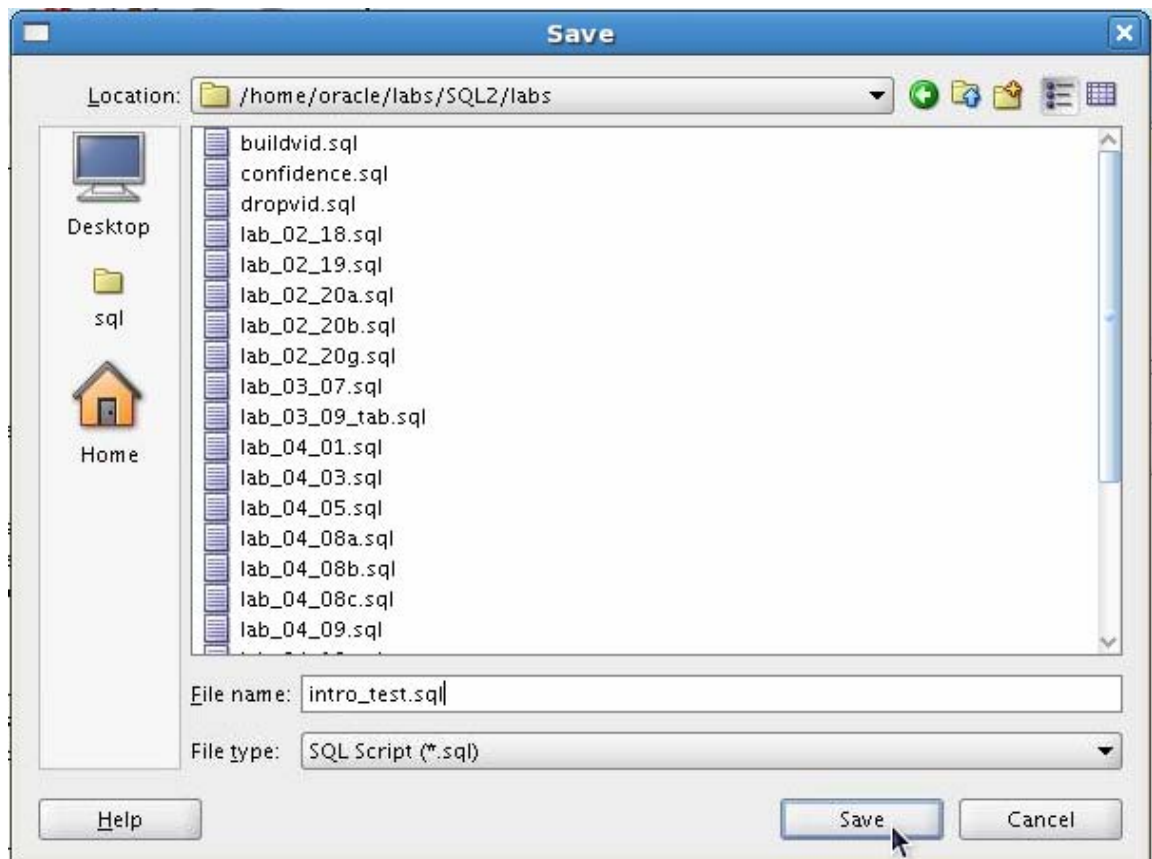


b) 파일 이름을 **intro_test.sql**로 지정합니다.

File_name 텍스트 상자에 **intro_test.sql**을 입력합니다.

연습 문제 해답 I-2: SQL Developer 사용(계속)

c) /home/oracle/labs/SQL2/labs 폴더에 파일을 저장합니다.

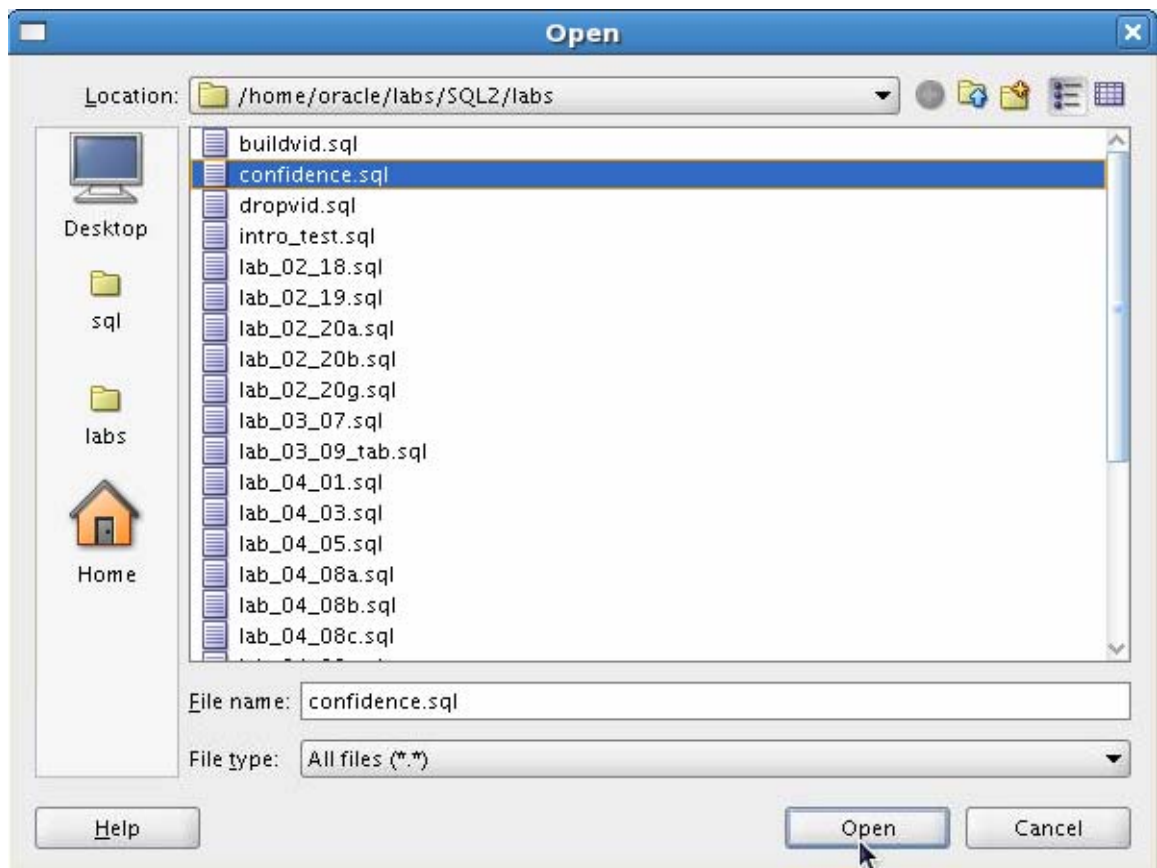


그런 다음 Save를 누릅니다.

- 9) /home/oracle/labs/SQL2/labs 폴더에서 confidence.sql을 열어 실행한 다음 결과를 확인합니다.

연습 문제 해답 I-2: SQL Developer 사용(계속)

File > Open 메뉴 항목을 사용하여 confidence.sql 스크립트 파일을 엽니다.



그런 다음 F5 키를 눌러 스크립트를 실행합니다.

예상되는 결과는 다음과 같습니다.

```
COUNT(*)
-----
8

1 rows selected

COUNT(*)
-----
107

1 rows selected

COUNT(*)
-----
25

1 rows selected
```

연습 문제 해답 I-2: SQL Developer 사용(계속)

```
COUNT(*)
-----
4

1 rows selected

COUNT(*)
-----
23

1 rows selected

COUNT(*)
-----
27

1 rows selected

COUNT(*)
-----
19

1 rows selected

COUNT(*)
-----
10

1 rows selected
```

연습 1-1: 유저 액세스 제어

- 1) Oracle 서버에 로그인하려면 유저에게 어떤 권한이 부여되어야 합니까? 시스템 권한입니까 아니면 객체 권한입니까?

- 2) 테이블 생성용으로 유저에게 부여되는 권한은 무엇입니까?

- 3) 테이블을 생성하는 경우 다른 유저에게 테이블에 대한 권한을 전달할 수 있는 사람은 누구입니까?

- 4) 여러분은 DBA입니다. 동일한 시스템 권한을 필요로 하는 많은 유저를 생성하고 있습니다.
작업을 보다 쉽게 수행하기 위해 무엇을 사용해야 합니까?

- 5) 암호를 변경하기 위해 어떤 명령을 사용합니까?

- 6) User21은 EMP 테이블의 소유자로서 WITH GRANT OPTION 절을 사용하여 DELETE 권한을 User22에게 부여합니다. 그러면 User22는 EMP에 대한 DELETE 권한을 User23에게 부여합니다. User21은 이제 User23에게 권한이 있음을 알게 되어 User22로부터 해당 권한을 취소합니다. 이제 EMP 테이블에서 데이터를 삭제할 수 있는 유저는 누구입니까?

- 7) DEPARTMENTS 테이블의 데이터를 갱신할 수 있는 권한을 SCOTT에게 부여하려고 합니다. 그리고 SCOTT이 다른 유저에게 이 권한을 부여할 수 있게 하려고 합니다. 어떤 명령을 사용해야 합니까?

문제 8과 이후 문제를 완료하려면 SQL Developer를 사용하여 데이터베이스에 연결해야 합니다. 연결되어 있지 않은 경우 다음을 수행하여 연결하십시오.

- 1) SQL Developer 바탕 화면 아이콘을 누릅니다.
- 2) Connections Navigator에서 강사가 알려준 **oraxx** 계정과 해당 암호를 사용하여 데이터베이스에 로그인합니다.

연습 1-1: 유저 액세스 제어(계속)

- 8) 다른 유저에게 테이블 대한 query 권한을 부여합니다. 그런 다음 해당 유저가 이 권한을 사용할 수 있는지 확인합니다.

참고: 이 연습에서는 다른 그룹과 팀을 이루십시오. 예를 들어 유저 ora21의 경우 다른 유저 ora22와 팀을 이루십시오.

- a) REGIONS 테이블의 레코드를 볼 수 있는 권한을 다른 유저에게 부여합니다. 이 권한을 다른 유저에게 추가로 부여할 수 있는 옵션을 이 유저에 대해 포함시킵니다.
- b) 유저가 REGIONS 테이블을 query하도록 합니다.
- c) 유저가 세번째 유저(예: ora23)에게 query 권한을 부여하도록 합니다.
- d) b단계를 수행하는 유저로부터 이 권한을 취소합니다.

참고: 각 팀은 연습 9와 10을 개별적으로 실행할 수 있습니다.

- 9) COUNTRIES 테이블에 대한 query 및 데이터 조작 권한을 다른 유저에게 부여합니다. 유저가 다른 유저에게 이러한 권한을 부여하지 못하도록 합니다.
- 10) 다른 유저에게 부여된 COUNTRIES 테이블에 대한 권한을 취소합니다.

참고: 연습 11 - 17에서는 다른 그룹과 팀을 이루십시오.

- 11) 다른 유저에게 DEPARTMENTS 테이블에 대한 액세스 권한을 부여합니다. 해당 유저가 본인의 DEPARTMENTS 테이블에 대한 query 액세스 권한을 여러분에게 부여하도록 합니다.
- 12) DEPARTMENTS 테이블의 모든 행을 query합니다.

	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

...

- 13) DEPARTMENTS 테이블에 새 행을 추가합니다. Team 1은 부서 번호가 500인 Education 부서를 추가해야 합니다. Team 2는 부서 번호가 510인 Human Resources 부서를 추가해야 합니다. 다른 팀의 테이블을 query합니다.
- 14) 다른 팀의 DEPARTMENTS 테이블에 대한 동의어를 생성합니다.

연습 1-1: 유저 액세스 제어(계속)

15) 동의어를 사용하여 다른 팀의 DEPARTMENTS 테이블에 있는 모든 행을 query합니다.

Team 1의 SELECT 문 결과:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
15	150	Marketing 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	510	Human Resources	(null)	(null)

Team 2의 SELECT 문 결과:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
15	150	Marketing 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	500	Education	(null)	(null)

16) 다른 팀으로부터 SELECT 권한을 취소합니다.

17) 13단계에서 DEPARTMENTS 테이블에 삽입했던 행을 제거하고 변경 사항을 저장합니다.

연습 문제 해답 1-1: 유저 액세스 제어

문제 8과 이후 문제를 완료하려면 SQL Developer를 사용하여 데이터베이스에 연결해야 합니다.

- 1) Oracle 서버에 로그인하려면 유저에게 어떤 권한이 부여되어야 합니까? 이것은 시스템 권한입니까, 객체 권한입니까?

CREATE SESSION 시스템 권한

- 2) 테이블 생성용으로 유저에게 부여되는 권한은 무엇입니까?

CREATE TABLE 권한

- 3) 테이블을 생성하는 경우 다른 유저에게 테이블에 대한 권한을 전달할 수 있는 사람은 누구입니까?

본인을 포함하여 본인이 WITH GRANT OPTION을 사용하여 권한을 부여한 모든 사람입니다.

- 4) 여러분은 DBA입니다. 동일한 시스템 권한을 필요로 하는 많은 유저를 생성하고 있습니다.

작업을 보다 쉽게 수행하기 위해 무엇을 사용해야 합니까?

시스템 권한을 포함하는 롤을 생성하고 이 롤을 유저에게 부여해야 합니다.

- 5) 암호를 변경하기 위해 어떤 명령을 사용합니까?

ALTER USER 문

- 6) User21은 EMP 테이블의 소유자로서 GRANT OPTION 절을 사용하여 DELETE 권한을 User22에게 부여합니다. 그러면 User22는 EMP에 대한 DELETE 권한을 User23에게 부여합니다. User21은 이제 User23에게 권한이 있음을 알게 되어 User22로부터 해당 권한을 취소합니다. 이 경우 EMP 테이블에서 데이터를 삭제할 수 있는 유저는 누구입니까?

User21만

- 7) DEPARTMENTS 테이블의 데이터를 갱신할 수 있는 권한을 SCOTT에게 부여하려고 합니다. 그리고 SCOTT이 다른 유저에게 이 권한을 부여할 수 있게 하려고 합니다. 어떤 명령을 사용해야 합니까?

GRANT UPDATE ON departments TO scott WITH GRANT OPTION;

연습 문제 해답 1-1: 유저 액세스 제어(계속)

- 8) 다른 유저에게 테이블 대한 query 권한을 부여합니다. 그런 다음 해당 유저가 이 권한을 사용할 수 있는지 확인합니다.

참고: 이 연습에서는 다른 그룹과 팀을 이루십시오. 예를 들어 유저 ora21의 경우 다른 유저 ora22와 팀을 이루십시오.

- a) REGIONS 테이블의 레코드를 볼 수 있는 권한을 다른 유저에게 부여합니다. 이 권한을 다른 유저에게 추가로 부여할 수 있는 옵션을 이 유저에 대해 포함시킵니다.

Team 1은 다음 명령문을 실행합니다.

```
GRANT select
ON regions
TO <team2_oraxx> WITH GRANT OPTION;
```

- b) 유저가 REGIONS 테이블을 query하도록 합니다.

Team 2는 다음 명령문을 실행합니다.

```
SELECT * FROM <team1_oraxx>.regions;
```

- c) 유저가 세번째 유저(예: ora23)에게 query 권한을 부여하도록 합니다.

Team 2는 다음 명령문을 실행합니다.

```
GRANT select
ON <team1_oraxx>.regions
TO <team3_oraxx>;
```

- d) b단계를 수행하는 유저로부터 이 권한을 취소합니다.

Team 1은 다음 명령문을 실행합니다.

```
REVOKE select
ON regions
FROM <team2_oraxx>;
```

- 9) COUNTRIES 테이블에 대한 query 및 데이터 조작 권한을 다른 유저에게 부여합니다. 유저가 다른 유저에게 이러한 권한을 부여하지 못하도록 합니다.

Team 1은 다음 명령문을 실행합니다.

```
GRANT select, update, insert
ON COUNTRIES
TO <team2_oraxx>;
```

연습 문제 해답 1-1: 유저 액세스 제어(계속)

10) 다른 유저에게 부여된 COUNTRIES 테이블에 대한 권한을 취소합니다.

Team 1은 다음 명령문을 실행합니다.

```
REVOKE select, update, insert ON COUNTRIES FROM
<team2_oraxx>;
```

참고: 연습 11 - 17에서는 다른 그룹과 팀을 이루십시오.

11) 다른 유저에게 DEPARTMENTS 테이블에 대한 액세스 권한을 부여합니다. 해당 유저가 본인의 DEPARTMENTS 테이블에 대한 **query** 액세스 권한을 여러분에게 부여하도록 합니다.

Team 2는 GRANT 문을 실행합니다.

```
GRANT select
ON departments
TO <team1_oraxx>;
```

Team 1은 GRANT 문을 실행합니다.

```
GRANT select
ON departments
TO <team2_oraxx>;
```

여기서 <team1_oraxx>는 Team 1의 username이고 <team2_oraxx>는 Team 2의 username입니다.

12) DEPARTMENTS 테이블의 모든 행을 query합니다.

```
SELECT *
FROM departments;
```

13) DEPARTMENTS 테이블에 새 행을 추가합니다. Team 1은 부서 번호가 500인 Education 부서를 추가해야 합니다. Team 2는 부서 번호가 510인 Human Resources 부서를 추가해야 합니다. 다른 팀의 테이블을 query합니다.

Team 1은 다음 INSERT 문을 실행합니다.

```
INSERT INTO departments(department_id, department_name)
VALUES (500, 'Education');
COMMIT;
```

Team 2는 다음 INSERT 문을 실행합니다.

```
INSERT INTO departments(department_id, department_name)
VALUES (510, 'Human Resources');
COMMIT;
```

연습 문제 해답 1-1: 유저 액세스 제어(계속)

14) 다른 팀의 DEPARTMENTS 테이블에 대한 동의어를 생성합니다.

Team 1은 team2로 명명된 동의어를 생성합니다.

```
CREATE SYNONYM team2
FOR <team2_oraxx>.DEPARTMENTS;
```

Team 2는 team1로 명명된 동의어를 생성합니다.

```
CREATE SYNONYM team1
FOR <team1_oraxx>. DEPARTMENTS;
```

15) 동의어를 사용하여 다른 팀의 DEPARTMENTS 테이블에 있는 모든 행을 query합니다.

Team 1은 다음 SELECT 문을 실행합니다.

```
SELECT *
FROM team2;
```

Team 2는 다음 SELECT 문을 실행합니다.

```
SELECT *
FROM team1;
```

16) 다른 팀으로부터 SELECT 권한을 취소합니다.

Team 1은 권한을 취소합니다.

```
REVOKE select
ON departments
FROM <team2_oraxx>;
```

Team 2는 권한을 취소합니다.

```
REVOKE select
ON departments
FROM <team1_oraxx>;
```

17) 8단계에서 DEPARTMENTS 테이블에 삽입했던 행을 제거하고 변경 사항을 저장합니다.

Team 1은 다음 DELETE 문을 실행합니다.

```
DELETE FROM departments
WHERE department_id = 500;
COMMIT;
```

Team 2는 다음 DELETE 문을 실행합니다.

```
DELETE FROM departments
WHERE department_id = 510;
COMMIT;
```

연습 2-1: 스키마 객체 관리

이 연습에서는 ALTER TABLE 명령을 사용하여 열을 수정하고 제약 조건을 추가합니다. 테이블을 생성할 때 CREATE TABLE 명령과 함께 CREATE INDEX 명령을 사용하여 인덱스를 생성합니다. External Table을 생성합니다.

- 1) 다음 테이블 instance 차트에 준하여 DEPT2 테이블을 생성합니다. SQL Worksheet에 구문을 입력합니다. 그런 다음 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	NAME
키 유형		
Null/고유		
FK 테이블		
FK 열		
데이터 유형	NUMBER	VARCHAR2
길이	7	25

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
NAME		VARCHAR2(25)
2 rows selected		

- 2) DEPARTMENTS 테이블의 데이터로 DEPT2 테이블을 채웁니다. 필요한 열만 포함시킵니다.
- 3) 다음 테이블 instance 차트에 준하여 EMP2 테이블을 생성합니다. SQL Worksheet에 구문을 입력합니다. 그런 다음 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	LAST_NAME	FIRST_NAME	DEPT_ID
키 유형				
Null/고유				
FK 테이블				
FK 열				
데이터 유형	NUMBER	VARCHAR2	VARCHAR2	NUMBER
길이	7	25	25	7

연습 2-1: 스키마 객체 관리(계속)

Name	Null	Type
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

- 4) 긴 사원 성을 허용하도록 EMP2 테이블을 수정합니다. 수정한 내용을 확인합니다.

Name	Null	Type
ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

- 5) EMPLOYEES 테이블의 구조에 준하여 EMPLOYEES2 테이블을 생성합니다. EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPARTMENT_ID 열만 포함시킵니다. 새 테이블의 열 이름을 각각 ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPT_ID로 지정합니다.
- 6) EMP2 테이블을 삭제합니다.
- 7) Recycle bin을 query하여 테이블이 있는지 확인합니다.

R	ORIGINAL_NAME	R	OPERATION	R	DROPTIME
17	EMP_NEW_SAL		DROP		2009-05-22:14:44:15
18	EMP2		DROP		2009-05-22:14:57:57

- 8) EMP2 테이블을 DROP 문 이전의 상태로 복원합니다.

Name	Null	Type
ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)
4 rows selected		

- 9) EMPLOYEES2 테이블에서 FIRST_NAME 열을 삭제합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.

Name	Null	Type
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
DEPT_ID		NUMBER(4)
4 rows selected		

연습 2-1: 스키마 객체 관리(계속)

- 10) EMPLOYEES2 테이블에서 DEPT_ID 열을 UNUSED로 표시합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.

Name	Null	Type
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
3 rows selected		

- 11) EMPLOYEES2 테이블에서 모든 UNUSED 열을 삭제합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.
- 12) ID 열에서 EMP2 테이블에 테이블 레벨의 PRIMARY KEY 제약 조건을 추가합니다. 이 제약 조건은 생성 시 이름이 지정됩니다. 제약 조건 이름을 my_emp_id_pk로 지정합니다.
- 13) ID 열을 사용하여 DEPT2 테이블에 PRIMARY KEY 제약 조건을 생성합니다. 이 제약 조건은 생성 시 이름이 지정됩니다. 제약 조건 이름을 my_dept_id_pk로 지정합니다.
- 14) 존재하지 않는 부서에 사원이 할당되지 않도록 하는 Foreign Key 참조를 EMP2 테이블에 추가합니다. 제약 조건 이름을 my_emp_dept_id_fk로 지정합니다.
- 15) EMP2 테이블을 수정합니다. 데이터 유형이 NUMBER이고 전체 자릿수 2, 소수점 이하 자릿수가 2인 COMMISSION 열을 추가합니다. COMMISSION 열에 커미션 값을 0보다 큰 값으로 한정하는 제약 조건을 추가합니다.
- 16) EMP2 및 DEPT2 테이블을 복원할 수 없도록 삭제합니다. Recycle bin을 확인합니다.
- 17) 다음 테이블 instance 차트에 준하여 DEPT_NAMED_INDEX 테이블을 생성합니다. PRIMARY KEY 열의 인덱스 이름을 DEPT_PK_IDX로 지정합니다.

열 이름	Deptno	Dname
Primary Key	있음	
데이터 유형	Number	VARCHAR2
길이	4	30

연습 2-1: 스키마 객체 관리(계속)

18) External table `library_items_ext`를 생성합니다. `ORACLE_LOADER` 액세스 드라이버를 사용합니다.

참고: 이 연습에서는 `emp_dir` 디렉토리와 `library_items.dat` 파일이 이미 생성되었습니다. `library_items.dat`에는 다음 형식의 레코드가 포함되어 있습니다.

```
2354, 2264, 13.21, 150,
2355, 2289, 46.23, 200,
2355, 2264, 50.00, 100,
```

a) `lab_02_18.sql` 파일을 엽니다. `library_items_ext` external table을 생성하는 코드 snippet을 확인합니다. 그런 다음 `<TODO1>`, `<TODO2>`, `<TODO3>` 및 `<TODO4>`를 적절하게 바꾸고 파일을 `lab_02_18_soln.sql`로 저장합니다. 스크립트를 실행하여 external table을 생성합니다.

b) `library_items_ext` 테이블을 query합니다.

	CATEGOR...	BOO...	BOOK_P...	QUAN...
1	2354	2264	13.21	150
2	2355	2289	46.23	200
3	2355	2264	50	100

19) HR 부서에서 모든 부서의 주소에 대한 보고서를 요구합니다.

`ORACLE_DATAPUMP` 액세스 드라이버를 사용하여 external table을 `dept_add_ext`로 생성합니다. 보고서 출력에는 번지, 동/리, 구/군, 시/도 및 국가가 표시되어야 합니다. `NATURAL JOIN`을 사용하여 결과를 생성합니다.

참고: 이 연습을 위해 `emp_dir` 디렉토리가 이미 생성되어 있습니다.

a) `lab_02_19.sql` 파일을 엽니다. `dept_add_ext` external table을 생성하는 코드 snippet을 확인합니다. 그런 다음 `<TODO1>`, `<TODO2>` 및 `<TODO3>`을 적절한 코드로 바꾸고, `<oraxx_emp4.exp>` 및 `<oraxx_emp5.exp>`를 적절한 파일 이름으로 바꿉니다. 예를 들어, 유저 `ora21`일 경우 파일 이름은 `ora21_emp4.exp`와 `ora21_emp5.exp`입니다. 스크립트를 `lab_02_19_soln.sql`로 저장합니다.

b) `lab_02_19_soln.sql` 스크립트를 실행하여 external table을 생성합니다.

연습 2-1: 스키마 객체 관리(계속)

c) dept_add_ext 테이블을 query합니다.

	LOCAT...	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1000	1297 Via Cola di Rie	Roma	(null)	Italy
2	1100	93091 Calle della Testa	Venice	(null)	Italy
3	1200	2017 Shinjuku-ku	Tokyo	Tokyo Prefecture	Japan
4	1300	9450 Kamiya-cho	Hiroshima	(null)	Japan
5	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of Amer
6	1500	2011 Interiors Blvd	South San Francisco	California	United States of Amer
7	1600	2007 Zagora St	South Brunswick	New Jersey	United States of Amer
8	1700	2004 Charade Rd	Seattle	Washington	United States of Amer

참고: 이전 단계를 수행할 때 oraxx_emp4.exp 및 oraxx_emp5.exp 파일이 기본 디렉토리 emp_dir에 생성됩니다.

20) emp_books 테이블을 생성하고 데이터로 채웁니다. Primary key를 deferred로 설정하고 트랜잭션이 완료될 때 어떤 결과가 나타나는지 확인합니다.

a) lab_02_20_a.sql 파일을 실행하여 emp_books 테이블을 생성합니다. emp_books_pk Primary key가 deferrable로 생성되지 않았는지 확인합니다.

```
create table succeeded.
```

b) lab_02_20_b.sql 파일을 실행하여 emp_books 테이블에 데이터를 채웁니다. 어떤 결과가 나타납니까?

```
1 rows inserted

Error starting at line 2 in command:
insert into emp_books values(300,'Change Management')
Error report:
SQL Error: ORA-00001: unique constraint (ORA21.EMP_BOOKS_PK) violated
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.
```

c) emp_books_pk 제약 조건을 deferred로 설정합니다. 어떤 결과가 나타납니까?

```
Error starting at line 1 in command:
set constraint emp_books_pk deferred
Error report:
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
02447. 00000 - "cannot defer a constraint that is not deferrable"
*Cause:      An attempt was made to defer a nondeferrable constraint
*Action:     Drop the constraint and create a new one that is deferrable
```

d) emp_books_pk 제약 조건을 삭제합니다.

```
alter table emp_books succeeded.
```


연습 2-1: 스키마 객체 관리(계속)

- e) emp_books 테이블 정의를 수정하여 이번에는 emp_books_pk 제약 조건을 deferrable로 추가합니다.

```
alter table emp_books succeeded.
```

- f) emp_books_pk 제약 조건을 deferred로 설정합니다.

```
set constraint succeeded.
```

- g) lab_02_20_g.sql 파일을 실행하여 emp_books 테이블에 데이터를 채웁니다. 어떤 결과가 나타납니까?

```
1 rows inserted  
1 rows inserted  
1 rows inserted
```

- h) 트랜잭션을 커밋합니다. 어떤 결과가 나타납니까?

```
Error report:  
SQL Error: ORA-02091: transaction rolled back  
ORA-00001: unique constraint (ORA21.EMP_BOOKS_PK) violated  
02091. 00000 - "transaction rolled back"  
*Cause:      Also see error 2092. If the transaction is aborted at a remote  
              site then you will only see 2091; if aborted at host then you will  
              see 2092 and 2091.  
*Action:     Add rollback segment and retry the transaction.
```

연습 문제 해답 2-1: 스키마 객체 관리

- 1) 다음 테이블 instance 차트에 준하여 DEPT2 테이블을 생성합니다. SQL Worksheet에 구문을 입력합니다. 그런 다음 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	NAME
키 유형		
Null/고유		
FK 테이블		
FK 열		
데이터 유형	NUMBER	VARCHAR2
길이	7	25

```
CREATE TABLE dept2
  (id NUMBER(7),
   name VARCHAR2(25));

DESCRIBE dept2
```

- 2) DEPARTMENTS 테이블의 데이터로 DEPT2 테이블을 채웁니다. 필요한 열만 포함시킵니다.

```
INSERT INTO dept2
SELECT  department_id, department_name
FROM    departments;
```

- 3) 다음 테이블 instance 차트에 준하여 EMP2 테이블을 생성합니다. SQL Worksheet에 구문을 입력합니다. 그런 다음 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	LAST_NAME	FIRST_NAME	DEPT_ID
키 유형				
Null/고유				
FK 테이블				
FK 열				
데이터 유형	NUMBER	VARCHAR2	VARCHAR2	NUMBER
길이	7	25	25	7

연습 문제 해답 2-1: 스키마 객체 관리(계속)

```
CREATE TABLE emp2
(id          NUMBER(7),
 last_name   VARCHAR2(25),
 first_name  VARCHAR2(25),
 dept_id     NUMBER(7));

DESCRIBE emp2
```

- 4) 긴 사원 성을 허용하도록 EMP2 테이블을 수정합니다. 수정한 내용을 확인합니다.

```
ALTER TABLE emp2
MODIFY (last_name   VARCHAR2(50));

DESCRIBE emp2
```

- 5) EMPLOYEES 테이블의 구조에 준하여 EMPLOYEES2 테이블을 생성합니다. EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPARTMENT_ID 열만 포함시킵니다. 새 테이블의 열 이름을 각각 ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPT_ID로 지정합니다.

```
CREATE TABLE employees2 AS
SELECT  employee_id id, first_name, last_name, salary,
        department_id dept_id
FROM    employees;
```

- 6) EMP2 테이블을 삭제합니다.

```
DROP TABLE emp2;
```

- 7) Recycle bin을 query하여 테이블이 있는지 확인합니다.

```
SELECT original_name, operation, droptime
FROM recyclebin;
```

- 8) EMP2 테이블을 DROP 문 이전의 상태로 복원합니다.

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
DESC emp2;
```

- 9) EMPLOYEES2 테이블에서 FIRST_NAME 열을 삭제합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.

```
ALTER TABLE employees2
DROP COLUMN first_name;

DESCRIBE employees2
```

연습 문제 해답 2-1: 스키마 객체 관리(계속)

- 10) EMPLOYEES2 테이블에서 DEPT_ID 열을 UNUSED로 표시합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.

```
ALTER TABLE      employees2
SET      UNUSED (dept_id);

DESCRIBE employees2
```

- 11) EMPLOYEES2 테이블에서 모든 UNUSED 열을 삭제합니다. 테이블 설명을 검사하여 수정 사항을 확인합니다.

```
ALTER TABLE employees2
DROP UNUSED COLUMNS;

DESCRIBE employees2
```

- 12) ID 열에서 EMP2 테이블에 테이블 레벨의 PRIMARY KEY 제약 조건을 추가합니다. 이 제약 조건은 생성 시 이름이 지정됩니다. 제약 조건 이름을 my_emp_id_pk로 지정합니다.

```
ALTER TABLE      emp2
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (id);
```

- 13) ID 열을 사용하여 DEPT2 테이블에 PRIMARY KEY 제약 조건을 생성합니다. 이 제약 조건은 생성 시 이름이 지정됩니다. 제약 조건 이름을 my_dept_id_pk로 지정합니다

```
ALTER TABLE      dept2
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY(id);
```

- 14) 존재하지 않는 부서에 사원이 할당되지 않도록 하는 Foreign Key 참조를 EMP2 테이블에 추가합니다. 제약 조건 이름을 my_emp_dept_id_fk로 지정합니다.

```
ALTER TABLE emp2
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept2(id);
```

- 15) EMP2 테이블을 수정합니다. 데이터 유형이 NUMBER이고 전체 자릿수 2, 소수점 이하 자릿수가 2인 COMMISSION 열을 추가합니다. COMMISSION 열에 커미션 값을 0보다 큰 값으로 한정하는 제약 조건을 추가합니다.

```
ALTER TABLE emp2
ADD commission NUMBER(2,2)
CONSTRAINT my_emp_comm_ck CHECK (commission > 0);
```

연습 문제 해답 2-1: 스키마 객체 관리(계속)

- 16) EMP2 및 DEPT2 테이블을 복원할 수 없도록 삭제합니다. Recycle bin을 확인합니다.

```
DROP TABLE emp2 PURGE;  
DROP TABLE dept2 PURGE;  
  
SELECT original_name, operation, droptime  
FROM recyclebin;
```

- 17) 다음 테이블 instance 차트에 준하여 DEPT_NAMED_INDEX 테이블을 생성합니다. PRIMARY KEY 열의 인덱스 이름을 DEPT_PK_IDX로 지정합니다.

열 이름	Deptno	Dname
Primary Key	있음	
데이터 유형	Number	VARCHAR2
길이	4	30

```
CREATE TABLE DEPT_NAMED_INDEX  
(deptno NUMBER(4)  
PRIMARY KEY USING INDEX  
(CREATE INDEX dept_pk_idx ON  
DEPT_NAMED_INDEX(deptno)),  
dname VARCHAR2(30));
```

연습 문제 해답 2-1: 스키마 객체 관리(계속)

18) External table library_items_ext를 생성합니다. ORACLE_LOADER 액세스 드라이버를 사용합니다.

참고: 이 연습에서는 emp_dir 디렉토리와 library_items.dat가 이미 생성되었습니다.

library_items.dat에는 다음 형식의 레코드가 포함되어 있습니다.

```
2354, 2264, 13.21, 150,  
2355, 2289, 46.23, 200,  
2355, 2264, 50.00, 100,
```

- a) lab_02_18.sql 파일을 엽니다. library_items_ext external table을 생성하는 코드 snippet을 확인합니다. 그런 다음 <TODO1>, <TODO2>, <TODO3> 및 <TODO4>를 적절하게 바꾸고 파일을 lab_02_18_soln.sql로 저장합니다. 스크립트를 실행하여 external table을 생성합니다.

```
CREATE TABLE library_items_ext ( category_id  number(12)  
                                , book_id number(6)  
                                , book_price number(8,2)  
                                , quantity  number(8)  
                                )  
  
ORGANIZATION EXTERNAL  
(TYPE ORACLE_LOADER  
  DEFAULT DIRECTORY emp_dir  
  ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE  
                     FIELDS TERMINATED BY ',')  
  LOCATION ('library_items.dat')  
  )  
REJECT LIMIT UNLIMITED;
```

- b) library_items_ext 테이블을 query합니다.

```
SELECT * FROM library_items_ext;
```

연습 문제 해답 2-1: 스키마 객체 관리(계속)

19) HR 부서에서 모든 부서의 주소에 대한 보고서를 요구합니다.

ORACLE_DATAPUMP 액세스 드라이버를 사용하여 external table을 dept_add_ext로 생성합니다. 보고서 출력에는 번지, 동/리, 구/군, 시/도 및 국가가 표시되어야 합니다. NATURAL JOIN을 사용하여 결과를 생성합니다.

참고: 이 연습을 위해 emp_dir 디렉토리가 이미 생성되어 있습니다.

a) lab_02_19.sql 파일을 엽니다. dept_add_ext external table을 생성하는 코드 snippet을 확인합니다. 그런 다음 <TODO1>, <TODO2> 및 <TODO3>을 적절한 코드로 바꾸고, <oraxx_emp4.exp> 및 <oraxx_emp5.exp>를 적절한 파일 이름으로 바꿉니다. 예를 들어, 유저 ora21일 경우 파일 이름은 ora21_emp4.exp와 ora21_emp5.exp입니다. 스크립트를 lab_02_19_soln.sql로 저장합니다.

```
CREATE TABLE dept_add_ext (location_id,
                           street_address, city,
                           state_province,
                           country_name)

ORGANIZATION EXTERNAL(
TYPE ORACLE_DATAPUMP
DEFAULT DIRECTORY emp_dir
LOCATION ('oraxx_emp4.exp', 'oraxx_emp5.exp'))
PARALLEL
AS
SELECT location_id, street_address, city, state_province,
country_name
FROM locations
NATURAL JOIN countries;
```

참고: 이전 단계를 수행할 때 oraxx_emp4.exp 및 oraxx_emp5.exp 파일이 기본 디렉토리 emp_dir에 생성됩니다.

lab_02_19_soln.sql 스크립트를 실행하여 external table을 생성합니다.

b) dept_add_ext 테이블을 query합니다.

```
SELECT * FROM dept_add_ext;
```

연습 문제 해답 2-1: 스키마 객체 관리(계속)

20) emp_books 테이블을 생성하고 데이터로 채웁니다. Primary key를 deferred로 설정하고 트랜잭션이 완료될 때 어떤 결과가 나타나는지 확인합니다.

- a) lab_02_20a.sql 스크립트를 실행하여 emp_books 테이블을 생성합니다. emp_books_pk Primary key가 deferrable로 생성되지 않았는지 확인합니다.

```
CREATE TABLE emp_books (book_id number,  
                          title varchar2(20), CONSTRAINT  
emp_books_pk PRIMARY KEY (book_id));
```

- b) lab_02_20b.sql 스크립트를 실행하여 emp_books 테이블에 데이터를 채웁니다.

어떤 결과가 나타납니까?

```
INSERT INTO emp_books VALUES(300,'Organizations');  
INSERT INTO emp_books VALUES(300,'Change Management');
```

첫번째 행이 삽입되었습니다. 그러나 두번째 행 삽입의 경우 ora-00001 오류가 발생합니다.

- c) emp_books_pk 제약 조건을 deferred로 설정합니다. 어떤 결과가 나타납니까?

```
SET CONSTRAINT emp_books_pk DEFERRED;
```

"ORA-02447: Cannot defer a constraint that is not deferrable" 오류가 발생합니다.

- d) emp_books_pk 제약 조건을 삭제합니다.

```
ALTER TABLE emp_books DROP CONSTRAINT emp_books_pk;
```

- e) emp_books 테이블 정의를 수정하여 이번에는 emp_books_pk 제약 조건을 deferrable로 추가합니다.

```
ALTER TABLE emp_books ADD (CONSTRAINT emp_books_pk PRIMARY  
KEY (book_id) DEFERRABLE);
```

- f) emp_books_pk 제약 조건을 deferred로 설정합니다.

```
SET CONSTRAINT emp_books_pk DEFERRED;
```


연습 문제 해답 2-1: 스키마 객체 관리(계속)

- g) lab_02_20g.sql 스크립트를 실행하여 emp_books 테이블에 데이터를 채웁니다.
어떤 결과가 나타납니까?

```
INSERT INTO emp_books VALUES (300,'Change Management');  
INSERT INTO emp_books VALUES (300,'Personality');  
INSERT INTO emp_books VALUES (350,'Creativity');
```

모든 행이 삽입되었습니다.

- h) 트랜잭션을 커밋합니다. 어떤 결과가 나타납니까?

```
COMMIT;
```

트랜잭션이 롤백되었습니다.

연습 3-1: 데이터 디렉터리 뷰로 객체 관리

이 연습에서는 디렉터리 뷰를 query하여 스키마의 객체에 대한 정보를 찾습니다.

- 1) USER_TABLES 데이터 디렉터리 뷰를 query하여 본인 소유의 테이블에 대한 정보를 확인합니다.

R2	TABLE_NAME
1	REGIONS
2	LOCATIONS
3	DEPARTMENTS
4	JOBS
5	EMPLOYEES
6	JOB_HISTORY
7	EMP_NEW_SAL
8	EMPLOYEES2
9	DEPT_NAMED_INDEX

...

- 2) ALL_TABLES 데이터 디렉터리 뷰를 query하여 본인이 액세스할 수 있는 모든 테이블에 대한 정보를 확인합니다. 본인 소유의 테이블은 제외시킵니다.

참고: 여러분의 리스트가 아래 리스트와 정확히 일치하지 않을 수도 있습니다.

R2	TABLE_NAME	R2	OWNER
1	DUAL		SYS
2	SYSTEM_PRIVILEGE_MAP		SYS
3	TABLE_PRIVILEGE_MAP		SYS

...

98	PLAN_TABLE\$	SYS
99	WRI\$_ADV_ASA_RECO_DATA	SYS
100	PSTU8TBL	SYS

- 3) 지정된 테이블에 대해 열 이름, 데이터 유형, 데이터 유형의 길이, 널 허용 여부를 보고하는 스크립트를 생성합니다. 유저에게 테이블 이름을 입력하는 프롬프트를 표시합니다. DATA_PRECISION 및 DATA_SCALE 열에 적당한 alias를 지정합니다. 이 스크립트를 lab_03_01.sql이라는 파일에 저장합니다. 예를 들어, 유저가 DEPARTMENTS를 입력하면 출력 결과는 다음과 같습니다.

R2	COLUMN_NAME	R2	DATA_TYPE	R2	DATA_LENGTH	R2	PRECISION	R2	SCALE	R2	NULLABLE
1	DEPARTMENT_ID		NUMBER		22		4		0		N
2	DEPARTMENT_NAME		VARCHAR2		30		(null)		(null)		N
3	MANAGER_ID		NUMBER		22		6		0		Y
4	LOCATION_ID		NUMBER		22		4		0		Y

연습 3-1: 데이터 디렉터리 뷰로 객체 관리(계속)

- 4) 열 이름, 제약 조건 이름, 제약 조건 유형, 검색 조건, 지정된 테이블의 상태를 보고하는 스크립트를 생성합니다. 이러한 정보를 모두 얻으려면 USER_CONSTRAINTS 및 USER_CONS_COLUMNS 테이블을 조인해야 합니다. 유저에게 테이블 이름을 입력하는 프롬프트를 표시합니다. 이 스크립트를 lab_03_04.sql이라는 파일에 저장합니다. 예를 들어, 유저가 DEPARTMENTS를 입력하면 출력 결과는 다음과 같습니다.

	COLUMN_NAME	CONSTRAINT_NAME	CONSTRA...	SEARCH_CONDITION	STATUS
1	DEPARTMENT_NAME	DEPT_NAME_NN	C	"DEPARTMENT_NAME" IS NOT ...	ENABLED
2	DEPARTMENT_ID	DEPT_ID_PK	P	(null)	ENABLED
3	LOCATION_ID	DEPT_LOC_FK	R	(null)	ENABLED
4	MANAGER_ID	DEPT_MGR_FK	R	(null)	ENABLED

- 5) DEPARTMENTS 테이블에 주석을 추가합니다. 그런 다음 USER_TAB_COMMENTS 뷰를 query하여 주석이 있는지 확인합니다.

	COMMENTS
1	Company department information including name, code, and location.

- 6) EMPLOYEES 테이블의 동의어를 생성하고 이름을 EMP로 지정합니다. 스키마에 있는 모든 동의어의 이름을 찾습니다.

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	TEAM2	ORA22	DEPARTMENTS	(null)
2	EMP	ORA21	EMPLOYEES	(null)

- 7) lab_03_07.sql을 실행하여 이 연습에서 사용할 dept50 뷰를 생성합니다. 스키마에 있는 모든 뷰의 이름과 정의를 파악해야 합니다. USER_VIEWS 데이터 디렉터리 뷰에서 뷰 이름과 텍스트 등의 뷰 정보를 검색하는 보고서를 생성합니다.

참고: EMP_DETAILS_VIEW는 스키마의 일부로 생성되었습니다.

참고: SQL Developer에서 Run Script를 사용하거나 F5 키를 누르면 뷰의 전체 정의를 볼 수 있습니다. SQL Developer에서 Execute Statement를 사용하거나 F9 키를 누르는 경우 결과 창에서 가로로 스크롤하십시오. 참고: SQL*Plus를 사용하는 경우 LONG 열의 추가 내용을 보려면 SET LONG n 명령을 사용하십시오. 여기서 n은 보고자 하는 LONG 열의 문자 수입니다.

	VIEW_NAME	TEXT
1	DEPT50	SELECT employee_id empno, last_name employee, department_id deptno
2	EMP_DETAILS_VIEW	SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id,

연습 3-1: 데이터 디셔너리 뷰로 객체 관리(계속)

12) 다음 테이블 instance 차트에 준하여 SALES_DEPT 테이블을 생성합니다.

PRIMARY KEY 열의 인덱스 이름을 SALES_PK_IDX로 지정합니다. 그런 다음 데이터 디셔너리 뷰를 query하여 인덱스 이름, 테이블 이름 및 고유 인덱스인지 여부를 찾습니다

열 이름	Team_Id	Location
Primary Key	있음	
데이터 유형	Number	VARCHAR2
길이	3	30

INDEX_NAME	TABLE_NAME	UNIQUENESS
1 SALES_PK_IDX	SALES_DEPT	NONUNIQUE

연습 문제 해답 3-1: 데이터 디렉터리 뷰로 객체 관리

- 1) 데이터 디렉터리를 query하여 본인이 본인 소유의 테이블에 대한 정보를 확인합니다.

```
SELECT table_name
FROM   user_tables;
```

- 2) 디렉터리 뷰를 query하여 본인이 액세스할 수 있는 모든 테이블에 대한 정보를 확인합니다. 본인 소유의 테이블은 제외시킵니다.

```
SELECT table_name, owner
FROM   all_tables
WHERE  owner <> 'ORAxX';
```

- 3) 지정된 테이블에 대해 열 이름, 데이터 유형, 데이터 유형의 길이, 널 허용 여부를 보고하는 스크립트를 생성합니다. 유저에게 테이블 이름을 입력하는 프롬프트를 표시합니다. DATA_PRECISION 및 DATA_SCALE 열에 적당한 alias를 지정합니다. 이 스크립트를 lab_03_01.sql이라는 파일에 저장합니다.

```
SELECT column_name, data_type, data_length,
       data_precision PRECISION, data_scale SCALE, nullable
FROM   user_tab_columns
WHERE  table_name = UPPER('&tab_name');
```

테스트하려면 스크립트를 실행하고 DEPARTMENTS를 테이블 이름으로 입력합니다.

- 4) 열 이름, 제약 조건 이름, 제약 조건 유형, 검색 조건, 지정된 테이블의 상태를 보고하는 스크립트를 생성합니다. 이러한 정보를 모두 얻으려면 USER_CONSTRAINTS 및 USER_CONS_COLUMNS 테이블을 조인해야 합니다. 유저에게 테이블 이름을 입력하는 프롬프트를 표시합니다. 이 스크립트를 lab_03_04.sql이라는 파일에 저장합니다.

```
SELECT ucc.column_name, uc.constraint_name,
       uc.constraint_type,
       uc.search_condition, uc.status
FROM   user_constraints uc JOIN user_cons_columns ucc
ON     uc.table_name = ucc.table_name
AND    uc.constraint_name = ucc.constraint_name
AND    uc.table_name = UPPER('&tab_name');
```

테스트하려면 스크립트를 실행하고 DEPARTMENTS를 테이블 이름으로 입력합니다.

연습 문제 해답 3-1: 데이터 디렉터리 뷰로 객체 관리(계속)

- 5) DEPARTMENTS 테이블에 주석을 추가합니다. 그런 다음
USER_TAB_COMMENTS 뷰를 query하여 주석이 있는지 확인합니다.

```
COMMENT ON TABLE departments IS
    'Company department information including name, code, and
    location.';

SELECT COMMENTS
FROM   user_tab_comments
WHERE  table_name = 'DEPARTMENTS';
```

- 6) EMPLOYEES 테이블의 동의어를 생성하고 이름을 EMP로 지정합니다. 그런 다음
스키마에 있는 모든 동의어의 이름을 찾습니다.

```
CREATE SYNONYM emp FOR EMPLOYEES;
SELECT *
FROM   user_synonyms;
```

- 7) lab_03_07.sql을 실행하여 이 연습에서 사용할 dept50 뷰를 생성합니다.
스키마에 있는 모든 뷰의 이름과 정의를 파악해야 합니다. USER_VIEWS 데이터
디렉터리 뷰에서 뷰 이름과 텍스트 등의 뷰 정보를 검색하는 보고서를
생성합니다.

참고: EMP_DETAILS_VIEW는 스키마의 일부로 생성되었습니다.

참고: SQL Developer에서 Run Script를 사용하거나 F5 키를 누르면 뷰의 전체
정의를 볼 수 있습니다. SQL Developer에서 Execute Statement를 사용하거나 F9
키를 누르는 경우 결과 창에서 가로로 스크롤하십시오. SQL*Plus를 사용하는
경우 LONG 열의 추가 내용을 보려면 LONG n 명령을 사용하십시오. 여기서 n은
보고자 하는 LONG 열의 문자 수입니다.

```
SELECT   view_name, text
FROM     user_views;
```

- 8) 시퀀스의 이름을 찾습니다. 시퀀스의 이름, 최대값, 증분 크기 및 마지막
숫자를 표시하기 위한 query를 스크립트에 작성합니다. 스크립트 이름을
lab_03_08.sql로 지정합니다. 스크립트에서 해당 명령문을 실행합니다.

```
SELECT   sequence_name, max_value, increment_by, last_number
FROM     user_sequences;
```

연습 9 - 11을 위해 lab_03_09_tab.sql 스크립트를 미리 실행합니다. 또는
스크립트 파일을 열고 코드를 복사하여 SQL Worksheet에 붙여 넣습니다. 그런
다음 스크립트를 실행합니다. 이 스크립트는 다음 작업을 수행합니다.

- DEPT2 및 EMP2 테이블을 삭제합니다.
- DEPT2 및 EMP2 테이블을 생성합니다.

참고: DEPT2 및 EMP2 테이블을 복원할 수 없도록 연습 2에서 이미
삭제했어야 합니다.

연습 문제 해답 3-1: 데이터 디렉터리 뷰로 객체 관리(계속)

- 9) DEPT2 및 EMP2 테이블이 모두 데이터 디렉터리에 저장되어 있는지 확인합니다.

```
SELECT table_name
FROM user_tables
WHERE table_name IN ('DEPT2', 'EMP2');
```

- 10) 데이터 디렉터리를 query하여 두 테이블에 대한 제약 조건 이름과 유형을 찾습니다.

```
SELECT constraint_name, constraint_type
FROM user_constraints
WHERE table_name IN ('EMP2', 'DEPT2');
```

- 11) 데이터 디렉터리를 query하여 두 테이블에 대한 객체 이름과 유형을 표시합니다.

```
SELECT object_name, object_type
FROM user_objects
WHERE object_name LIKE 'EMP%'
OR object_name LIKE 'DEPT%';
```

- 12) 다음 테이블 instance 차트에 준하여 SALES_DEPT 테이블을 생성합니다.
PRIMARY KEY 열의 인덱스 이름을 SALES_PK_IDX로 지정합니다. 그런 다음
데이터 디렉터리 뷰를 query하여 인덱스 이름, 테이블 이름 및 고유 인덱스인지
여부를 찾습니다.

열 이름	Team_Id	Location
Primary Key	있음	길이
데이터 유형	Number	VARCHAR2
길이	3	30

```
CREATE TABLE SALES_DEPT
(team_id NUMBER(3)
PRIMARY KEY USING INDEX
(CREATE INDEX sales_pk_idx ON
SALES_DEPT(team_id),
location VARCHAR2(30));

SELECT INDEX_NAME, TABLE_NAME, UNIQUENESS
FROM USER_INDEXES
WHERE TABLE_NAME = 'SALES_DEPT';
```


연습 4-1: 대형 객체 집합 조작

이 연습에서는 다중 테이블 INSERT 및 MERGE 작업을 수행하고 행 버전을 추적합니다.

- 1) lab 폴더의 lab_04_01.sql 스크립트를 실행하여 SAL_HISTORY 테이블을 생성합니다.
- 2) SAL_HISTORY 테이블의 구조를 표시합니다.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
HIRE_DATE		DATE
SALARY		NUMBER(8,2)
3 rows selected		

- 3) lab 폴더의 lab_04_03.sql 스크립트를 실행하여 MGR_HISTORY 테이블을 생성합니다.
- 4) MGR_HISTORY 테이블의 구조를 표시합니다.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
MANAGER_ID		NUMBER(6)
SALARY		NUMBER(8,2)
3 rows selected		

- 5) lab 폴더의 lab_04_05.sql 스크립트를 실행하여 SPECIAL_SAL 테이블을 생성합니다.
- 6) SPECIAL_SAL 테이블의 구조를 표시합니다.

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
SALARY		NUMBER(8,2)
2 rows selected		

연습 4-1: 대형 객체 집합 조작(계속)

7) a) 다음을 수행하기 위한 query를 작성합니다.

- EMPLOYEES 테이블에서 사원 ID가 125보다 작은 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID와 같은 세부 정보를 검색합니다.
- 급여가 \$20,000보다 크면 사원 ID, 급여 등의 세부 정보를 SPECIAL_SAL 테이블에 삽입합니다.
- 사원 ID, 채용 날짜, 급여 등의 세부 정보를 SAL_HISTORY 테이블에 삽입합니다.
- 사원 ID, 관리자 ID, 급여 등의 세부 정보를 MGR_HISTORY 테이블에 삽입합니다.

b) SPECIAL_SAL 테이블의 레코드를 표시합니다.

	EMPLOYEE_ID	SALARY
1	100	24000

c) SAL_HISTORY 테이블의 레코드를 표시합니다.

	EMPLOYEE_ID	HIRE_DATE	SALARY
1	101	21-SEP-89	17000
2	102	13-JAN-93	17000
3	103	03-JAN-90	9000
4	104	21-MAY-91	6000
5	105	25-JUN-97	4800
6	106	05-FEB-98	4800
7	107	07-FEB-99	4200

d) MGR_HISTORY 테이블의 레코드를 표시합니다.

	EMPLOYEE_ID	MANAGER_ID	SALARY
1	101	100	17000
2	102	100	17000
3	103	102	9000
4	104	103	6000
5	105	103	4800
6	106	103	4800
7	107	103	4200

연습 4-1: 대형 객체 집합 조작(계속)

- 8) a) lab 폴더의 lab_04_08a.sql 스크립트를 실행하여 SALES_WEEK_DATA 테이블을 생성합니다.
- a) lab 폴더의 lab_04_08b.sql 스크립트를 실행하여 SALES_WEEK_DATA 테이블에 레코드를 삽입합니다.
- b) SALES_WEEK_DATA 테이블의 구조를 표시합니다.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
WEEK_ID		NUMBER(2)
QTY_MON		NUMBER(8,2)
QTY_TUE		NUMBER(8,2)
QTY_WED		NUMBER(8,2)
QTY_THUR		NUMBER(8,2)
QTY_FRI		NUMBER(8,2)
7 rows selected		

- c) SALES_WEEK_DATA 테이블의 레코드를 표시합니다.

	ID	WEEK_ID	QTY_MON	QTY_TUE	QTY_WED	QTY_THUR	QTY_FRI
1	200	6	2050	2200	1700	1200	3000

- d) lab 폴더의 lab_04_08_e.sql 스크립트를 실행하여 EMP_SALES_INFO 테이블을 생성합니다.
- e) EMP_SALES_INFO 테이블의 구조를 표시합니다.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
WEEK		NUMBER(2)
QTY_SALES		NUMBER(8,2)
3 rows selected		

- f) 다음을 수행하기 위한 query를 작성합니다.
- SALES_WEEK_DATA 테이블에서 ID, 주 ID, 월요일 매출, 화요일 매출, 수요일 매출, 목요일 매출, 금요일 매출과 같은 세부 정보를 검색합니다.
 - SALES_WEEK_DATA 테이블에서 검색된 각 레코드가 EMP_SALES_INFO 테이블에 대한 복수 레코드로 변환되도록 변형을 생성합니다.
- 힌트:** 피버팅 INSERT 문을 사용합니다.

연습 4-1: 대형 객체 집합 조작(계속)

g) EMP_SALES_INFO 테이블의 레코드를 표시합니다.

R2	ID	R2	WEEK	R2	QTY_SALES
1	200		6		2050
2	200		6		2200
3	200		6		1700
4	200		6		1200
5	200		6		3000

9) 퇴직 사원의 데이터는 emp.data 파일에 저장되어 있습니다. 퇴직 사원과 현재 사원 모두의 이름과 전자 메일 ID를 한 테이블에 저장하려고 합니다. 이렇게 하려면 먼저 emp_dir 디렉토리에 있는 emp.dat 소스 파일을 사용하여 EMP_DATA라는 external table을 생성합니다. lab_04_09.sql 스크립트를 사용하여 이를 실행합니다.

10) 그런 다음 lab_04_10.sql 스크립트를 실행하여 EMP_HIST 테이블을 생성합니다.

a) 전자 메일 열의 크기를 45로 늘립니다.

b) 마지막 lab에서 생성된 EMP_DATA 테이블의 데이터를 EMP_HIST 테이블의 데이터에 병합합니다. External EMP_DATA 테이블의 데이터는 최신 데이터라고 가정합니다. EMP_DATA 테이블의 행이 EMP_HIST 테이블과 일치하면 EMP_HIST 테이블의 전자 메일 열을 EMP_DATA 테이블 행과 일치하도록 갱신합니다. EMP_DATA 테이블의 행이 일치하지 않으면 EMP_HIST 테이블에 해당 행을 삽입합니다. 사원의 이름과 성이 동일하면 행이 일치하는 것으로 간주합니다.

c) 병합 후 EMP_HIST에서 행을 검색합니다.

R2	FIRST_NAME	R2	LAST_NAME	R2	EMAIL
1	Ellen		Abel		EABEL
2	Sundar		Ande		SANDE
3	Mozhe		Atkinson		MATKINSO
4	David		Austin		DAUSTIN
5	Hermann		Baer		HBAER
6	Shelli		Baida		SBAIDA
7	Amit		Banda		ABANDA
8	Elizabeth		Bates		EBATES
9	Sarah		Bell		SBELL
10	David		Bernstein		DBERNSTE
11	Laura		Bissot		LBISSOT

연습 4-1: 대형 객체 집합 조작(계속)

- 11) lab_04_11.sql 스크립트를 사용하여 EMP3 테이블을 생성합니다. EMP3 테이블에서 Kochhar의 부서를 60으로 변경하고 변경 사항을 커밋합니다. 그런 다음 Kochhar의 부서를 50으로 변경하고 변경 사항을 커밋합니다. Row Versions 기능을 사용하여 Kochhar에 대한 변경 사항을 추적합니다.

R	START_DATE	R	END_DATE	R	DEPARTMENT_ID
1	18-JUN-09 06.04.26.0000000000 PM	(null)			50
2	18-JUN-09 06.04.26.0000000000 PM	18-JUN-09 06.04.26.0000000000 PM			60
3	(null)	18-JUN-09 06.04.26.0000000000 PM			90

연습 문제 해답 4-1: 대형 객체 집합 조작

1) lab 폴더의 lab_04_01.sql 스크립트를 실행하여 SAL_HISTORY 테이블을 생성합니다.

2) SAL_HISTORY 테이블의 구조를 표시합니다.

```
DESC sal_history
```

3) lab 폴더의 lab_04_03.sql 스크립트를 실행하여 MGR_HISTORY 테이블을 생성합니다.

4) MGR_HISTORY 테이블의 구조를 표시합니다.

```
DESC mgr_history
```

5) lab 폴더의 lab_04_05.sql 스크립트를 실행하여 SPECIAL_SAL 테이블을 생성합니다.

6) SPECIAL_SAL 테이블의 구조를 표시합니

```
DESC special_sal
```

7) a) 다음 과정을 수행하는 query를 작성합니다.

- EMPLOYEES 테이블에서 사원 ID가 125보다 작은 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID와 같은 세부 정보를 검색합니다.
- 급여가 \$20,000보다 크면 사원 ID, 급여 등의 세부 정보를 SPECIAL_SAL 테이블에 삽입합니다.
- 사원 ID, 채용 날짜, 급여 등의 세부 정보를 SAL_HISTORY 테이블에 삽입합니다.
- 사원 ID, 관리자 ID, 급여 등의 세부 정보를 MGR_HISTORY 테이블에 삽입합니다.

```
INSERT ALL
WHEN SAL > 20000 THEN
  INTO special_sal VALUES (EMPID, SAL)
ELSE
  INTO sal_history VALUES(EMPID,HIREDATE,SAL)
  INTO mgr_history VALUES(EMPID,MGR,SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
salary SAL, manager_id MGR
FROM employees
WHERE employee_id < 125;
```

b) SPECIAL_SAL 테이블의 레코드를 표시합니다.

```
SELECT * FROM special_sal;
```

c) SAL_HISTORY 테이블의 레코드를 표시합니다.

```
SELECT * FROM sal_history;
```

연습 4-1: 대형 객체 집합 조작(계속)

- d) MGR_HISTORY 테이블의 레코드를 표시합니다.

```
SELECT * FROM mgr_history;
```

- 8) a) lab 폴더의 lab_04_08a.sql 스크립트를 실행하여 SALES_WEEK_DATA 테이블을 생성합니다.
- b) lab 폴더의 lab_04_08b.sql 스크립트를 실행하여 SALES_WEEK_DATA 테이블에 레코드를 삽입합니다.
- c) SALES_WEEK_DATA 테이블의 구조를 표시합니다.

```
DESC sales_week_data
```

- d) SALES_WEEK_DATA 테이블의 레코드를 표시합니다.

```
SELECT * FROM SALES_WEEK_DATA;
```

- e) lab 폴더의 lab_04_08_e.sql 스크립트를 실행하여 EMP_SALES_INFO 테이블을 생성합니다.
- f) EMP_SALES_INFO 테이블의 구조를 표시합니다.

```
DESC emp_sales_info
```

- g) 다음 과정을 수행하는 query를 작성합니다.
- SALES_WEEK_DATA 테이블에서 사원 ID, 주 ID, 월요일 판매량, 화요일 판매량, 수요일 판매량, 목요일 판매량 및 금요일 판매량 등의 세부 정보를 검색합니다.
 - SALES_WEEK_DATA 테이블에서 검색된 각 레코드가 EMP_SALES_INFO 테이블에 대한 복수 레코드로 변환되도록 변형을 생성합니다.
- 힌트:** 피버팅 INSERT 문을 사용합니다.

```
INSERT ALL
  INTO emp_sales_info VALUES (id, week_id, QTY_MON)
  INTO emp_sales_info VALUES (id, week_id, QTY_TUE)
  INTO emp_sales_info VALUES (id, week_id, QTY_WED)
  INTO emp_sales_info VALUES (id, week_id, QTY_THUR)
  INTO emp_sales_info VALUES (id, week_id, QTY_FRI)
SELECT ID, week_id, QTY_MON, QTY_TUE, QTY_WED,
       QTY_THUR, QTY_FRI FROM sales_week_data;
```

- h) SALES_INFO 테이블의 레코드를 표시합니다.

```
SELECT * FROM emp_sales_info;
```

연습 4-1: 대형 객체 집합 조작(계속)

- 9) 퇴직 사원의 데이터는 emp.data 파일에 저장되어 있습니다. 퇴직 사원과 현재 사원 모두의 이름과 전자 메일 ID를 한 테이블에 저장하려고 합니다. 이렇게 하려면 먼저 emp_dir 디렉토리에 있는 emp.dat 소스 파일을 사용하여 EMP_DATA라는 external table을 생성합니다. lab_04_09.sql 스크립트를 사용하면 됩니다.

```
CREATE TABLE emp_data
  (first_name  VARCHAR2(20)
   ,last_name   VARCHAR2(20)
   , email      VARCHAR2(30)
   )
ORGANIZATION EXTERNAL
(
  TYPE oracle_loader
  DEFAULT DIRECTORY emp_dir
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    NOBADFILE
    NOLOGFILE
    FIELDS
    ( first_name POSITION ( 1:20) CHAR
    , last_name POSITION (22:41) CHAR
    , email POSITION (43:72) CHAR )
  )
  LOCATION ('emp.dat') ) ;
```

- 10) 그런 다음 lab_04_10.sql 스크립트를 실행하여 EMP_HIST 테이블을 생성합니다.

- a) 전자 메일 열의 크기를 45로 늘립니다.

```
ALTER TABLE emp_hist MODIFY email varchar(45);
```

- b) 마지막 연습에서 생성된 EMP_DATA 테이블의 데이터를 EMP_HIST 테이블의 데이터에 병합합니다. External EMP_DATA 테이블의 데이터는 최신 데이터라고 가정합니다. EMP_DATA 테이블의 행이 EMP_HIST 테이블과 일치하면 EMP_HIST 테이블의 전자 메일 열을 EMP_DATA 테이블 행과 일치하도록 갱신합니다. EMP_DATA 테이블의 행이 일치하지 않으면 EMP_HIST 테이블에 해당 행을 삽입합니다. 사원의 이름과 성이 동일하면 행이 일치하는 것으로 간주합니다.

```
MERGE INTO EMP_HIST f USING EMP_DATA h
  ON (f.first_name = h.first_name
  AND f.last_name = h.last_name)
WHEN MATCHED THEN
  UPDATE SET f.email = h.email
WHEN NOT MATCHED THEN
  INSERT (f.first_name
    , f.last_name
    , f.email)
  VALUES (h.first_name
    , h.last_name
    , h.email);
```


연습 4-1: 대형 객체 집합 조작(계속)

c) 병합 후 EMP_HIST에서 행을 검색합니다.

```
SELECT * FROM emp_hist;
```

- 11) lab_04_11.sql 스크립트를 사용하여 EMP3 테이블을 생성합니다. EMP3 테이블에서 Kochhar의 부서를 60으로 변경하고 변경 사항을 커밋합니다. 그런 다음 Kochhar의 부서를 50으로 변경하고 변경 사항을 커밋합니다. Row Versions 기능을 사용하여 Kochhar에 대한 변경 사항을 추적합니다.

```
UPDATE emp3 SET department_id = 60
WHERE last_name = 'Kochhar';
COMMIT;
UPDATE emp3 SET department_id = 50
WHERE last_name = 'Kochhar';
COMMIT;
```

```
SELECT VERSIONS_STARTTIME "START_DATE",
       VERSIONS_ENDTIME "END_DATE",  DEPARTMENT_ID
FROM EMP3
      VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE LAST_NAME = 'Kochhar';
```

연습 5-1: 서로 다른 시간대에서의 데이터 관리

이 연습에서는 시간대 오프셋인 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다. 또한 시간대를 설정하고 EXTRACT 함수를 사용합니다.

- 1) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY HH24:MI:SS로 설정합니다.
- 2) a) 다음 시간대에 대해 시간대 오프셋(TZ_OFFSET)을 표시하는 query를 작성합니다.

- US/Pacific-New

	TZ_OFFSET('US/PACIFIC-NEW')
1	-07:00□

- Singapore

	TZ_OFFSET('SINGAPORE')
1	+08:00□

- Egypt

	TZ_OFFSET('EGYPT')
1	+03:00□

- b) 세션을 변경하여 TIME_ZONE 파라미터 값을 US/Pacific-New의 시간대 오프셋으로 설정합니다.
- c) 이 세션에 대해 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.
- d) 세션을 변경하여 TIME_ZONE 파라미터 값을 Singapore의 시간대 오프셋으로 설정합니다.
- e) 이 세션에 대해 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
1	23-JUN-2009 15:12:08	23-JUN-09 03.12.08.000000000 PM +08:00	23-JUN-09 03.12.08.000000000 PM

참고: 앞의 연습에서 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP는 모두 세션 시간대의 영향을 받습니다.

연습 5-1: 서로 다른 시간대에서의 데이터 관리(계속)

- 3) DBTIMEZONE 및 SESSIONTIMEZONE을 표시하는 query를 작성합니다.

	DBTIMEZONE	SESSIONTIMEZONE
1	+00:00	+08:00

- 4) EMPLOYEES 테이블의 HIRE_DATE 열에서 부서 80에서 근무하는 사원에 대한 YEAR를 추출하는 query를 작성합니다.

	LAST_NAME	EXTRACT(YEARFROMHIRE_DATE)
1	Russell	1996
2	Partners	1997
3	Errazuriz	1997
4	Cambrault	1999
5	Zlotkey	2000
6	Tucker	1997
7	Bernstein	1997

- 5) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY로 설정합니다.
- 6) lab_05_06.sql 스크립트를 조사하고 실행하여 SAMPLE_DATES 테이블을 생성하고 채웁니다.
- a) 테이블에서 선택하여 데이터를 확인합니다.

	DATE_COL
1	23-JUN-2009

- b) DATE_COL 열의 데이터 유형을 수정하고 TIMESTAMP로 변경합니다. 테이블에서 선택하여 데이터를 확인합니다.

	DATE_COL
1	23-JUN-09 02.14.52.000000000 PM

- c) DATE_COL 열의 데이터 유형을 수정하고 TIMESTAMP WITH TIME ZONE으로 변경합니다. 어떤 결과가 나타납니까?

연습 5-1: 서로 다른 시간대에서의 데이터 관리(계속)

- 7) EMPLOYEES 테이블에서 성을 검색하고 검토 상태를 계산하는 query를 작성합니다. 채용 연도가 1998년인 경우 검토 상태에 대해 Needs Review를 표시하고 그렇지 않은 경우 not this year!를 표시합니다. 검토 상태 열의 이름을 Review로 지정합니다. HIRE_DATE 열을 기준으로 결과를 정렬합니다.

힌트: 검토 상태를 계산하려면 CASE 표현식을 EXTRACT 함수와 함께 사용하십시오.

RANK	LAST_NAME	RANK	Review
1	King		not this year!
2	Whalen		not this year!
3	Kochhar		not this year!
4	Hunold		not this year!
5	Ernst		not this year!
6	De Haan		not this year!
7	Mavris		not this year!

...

- 8) 각 사원의 성과 근속 연수를 출력하는 query를 작성합니다. 사원의 근속 연수가 5년 이상인 경우 5 years of service를 출력합니다. 사원의 근속 연수가 10년 이상인 경우 10 years of service를 출력합니다. 사원의 근속 연수가 15년 이상인 경우 15 years of service를 출력합니다. 어떠한 조건과도 일치하지 않을 경우 maybe next year!를 출력합니다. HIRE_DATE 열을 기준으로 결과를 정렬합니다. EMPLOYEES 테이블을 사용합니다.

힌트: CASE 식과 TO_YMINTERVAL을 사용합니다.

RANK	LAST_NAME	RANK	HIRE_DATE	RANK	SYSDATE	RANK	Awards
1	OConnell		21-JUN-1999		23-JUN-2009		10 years of service
2	Grant		13-JAN-2000		23-JUN-2009		5 years of service
3	Whalen		17-SEP-1987		23-JUN-2009		15 years of service
4	Hartstein		17-FEB-1996		23-JUN-2009		10 years of service
5	Fay		17-AUG-1997		23-JUN-2009		10 years of service
6	Mavris		07-JUN-1994		23-JUN-2009		15 years of service

...

연습 문제 해답 5-1: 서로 다른 시간대에서의 데이터 관리

- 1) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY HH24:MI:SS로 설정합니다.

```
ALTER SESSION SET NLS_DATE_FORMAT =
'DD-MON-YYYY HH24:MI:SS';
```

- 2) a) *US/Pacific-New*, *Singapore* 및 *Egypt* 시간대에 대해 시간대 오프셋(TZ_OFFSET)을 표시하는 query를 작성합니다.

US/Pacific-New

```
SELECT TZ_OFFSET ('US/Pacific-New') from dual;
```

Singapore

```
SELECT TZ_OFFSET ('Singapore') from dual;
```

Egypt

```
SELECT TZ_OFFSET ('Egypt') from dual;
```

- b) 세션을 변경하여 TIME_ZONE 파라미터 값을 US/Pacific-New의 시간대 오프셋으로 설정합니다.

```
ALTER SESSION SET TIME_ZONE = '-7:00';
```

- c) 이 세션에 대해 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

- d) 세션을 변경하여 TIME_ZONE 파라미터 값을 Singapore의 시간대 오프셋으로 설정합니다.

```
ALTER SESSION SET TIME_ZONE = '+8:00';
```

- e) 이 세션에 대해 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

참고: 앞의 연습에서 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP는 모두 세션 시간대의 영향을 받습니다.

- 3) DBTIMEZONE 및 SESSIONTIMEZONE을 표시하는 query를 작성합니다.

```
SELECT DBTIMEZONE, SESSIONTIMEZONE
FROM DUAL;
```

연습 문제 해답 5-1: 서로 다른 시간대에서의 데이터 관리(계속)

- 4) EMPLOYEES 테이블의 HIRE_DATE 열에서 부서 80에서 근무하는 사원에 대한 YEAR를 추출하는 query를 작성합니다.

```
SELECT last_name, EXTRACT (YEAR FROM HIRE_DATE)
FROM employees
WHERE department_id = 80;
```

- 5) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY로 설정합니다.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

- 6) lab_05_06.sql 스크립트를 조사하고 실행하여 SAMPLE_DATES 테이블을 생성하고 채웁니다.

- a) 테이블에서 선택하여 데이터를 확인합니다.

```
SELECT * FROM sample_dates;
```

- b) DATE_COL 열의 데이터 유형을 수정하고 TIMESTAMP로 변경합니다.
테이블에서 선택하여 데이터를 확인합니다.

```
ALTER TABLE sample_dates MODIFY date_col TIMESTAMP;
SELECT * FROM sample_dates;
```

- c) DATE_COL 열의 데이터 유형을 수정하고 TIMESTAMP WITH TIME
ZONE으로 변경합니다. 어떤 결과가 나타납니까?

```
ALTER TABLE sample_dates MODIFY date_col
TIMESTAMP WITH TIME ZONE;
```

Oracle 서버는 ALTER 문을 사용하여 TIMESTAMP에서 TIMESTAMP WITH TIMEZONE으로 변환하는 것을 허용하지 않으므로 DATE_COL 열의 데이터 유형을 변경할 수 없습니다.

- 7) EMPLOYEES 테이블에서 성을 검색하고 검토 상태를 계산하는 query를 작성합니다. 채용 연도가 1998년인 경우 검토 상태에 대해 Needs Review를 표시하고 그렇지 않은 경우 not this year!를 표시합니다. 검토 상태 열의 이름을 Review로 지정합니다. HIRE_DATE 열을 기준으로 결과를 정렬합니다.
힌트: 검토 상태를 계산하려면 CASE 표현식을 EXTRACT 함수와 함께 사용하십시오.

```
SELECT e.last_name
,      (CASE extract(year from e.hire_date)
        WHEN 1998 THEN 'Needs Review'
        ELSE 'not this year!'
        END ) AS "Review "
FROM   employees e
ORDER BY e.hire_date;
```

연습 문제 해답 5-1: 서로 다른 시간대에서의 데이터 관리(계속)

- 8) 각 사원의 성과 근속 연수를 출력하는 query를 작성합니다. 사원의 근속 연수가 5년 이상인 경우 5 years of service를 출력합니다. 사원의 근속 연수가 10년 이상인 경우 10 years of service를 출력합니다. 사원의 근속 연수가 15년 이상인 경우 15 years of service를 출력합니다. 어떠한 조건과도 일치하지 않을 경우 maybe next year!를 출력합니다. HIRE_DATE 열을 기준으로 결과를 정렬합니다. EMPLOYEES 테이블을 사용합니다.

힌트: CASE 식과 TO_YMINTERVAL을 사용합니다.

```
SELECT e.last_name, hire_date, sysdate,
       (CASE
        WHEN (sysdate -TO_YMINTERVAL('15-0'))>=
             hire_date THEN      '15 years of service'
        WHEN (sysdate -TO_YMINTERVAL('10-0'))>= hire_date
             THEN      '10 years of service'
        WHEN (sysdate - TO_YMINTERVAL('5-0'))>= hire_date
             THEN      '5 years of service'
        ELSE 'maybe next year!'
        END) AS "Awards"
FROM   employees e;
```

연습 6-1: Subquery를 사용하여 데이터 검색

이 연습에서는 여러 열 subquery, correlated subquery 및 스칼라 subquery를 작성합니다. 또한 WITH 절을 작성하여 문제를 해결합니다.

- 1) 부서 번호 및 급여 모두가 커미션을 받는 사원의 부서 번호 및 급여와 일치하는 사원의 성, 부서 번호 및 급여를 표시하는 query를 작성합니다.

R2	LAST_NAME	R2	DEPARTMENT_ID	R2	SALARY
1	Russell		80		14000
2	Partners		80		13500
3	Errazuriz		80		12000

- 2) 급여와 커미션이 위치 ID의 사원과 일치하는 사원의 성, 부서 이름 및 급여를 표시합니다.

R2	LAST_NAME	R2	DEPARTMENT_NAME	R2	SALARY
1	Whalen		Administration		4400
2	Higgins		Accounting		12000
3	Greenberg		Finance		12000
4	Gietz		Accounting		8300

- 3) Kochhar와 동일한 급여 및 커미션을 받는 모든 사원의 성, 채용 날짜 및 급여를 표시하는 query를 작성합니다.

참고: 결과 집합에 Kochhar를 표시하지 마십시오.

R2	LAST_NAME	R2	HIRE_DATE	R2	SALARY
1	De Haan		13-JAN-1993		17000

- 4) 모든 영업 관리자(JOB_ID = 'SA_MAN')보다 많은 급여를 받는 사원을 표시하는 query를 작성합니다. 결과를 하향식으로 정렬합니다.

R2	LAST_NAME	R2	JOB_ID	R2	SALARY
1	King		AD_PRES		24000
2	De Haan		AD_VP		17000
3	Kochhar		AD_VP		17000

- 5) 이름이 T로 시작하는 도시에 거주하는 사원의 사원 ID, 성 및 부서 ID와 같은 세부 정보를 표시합니다.

R2	EMPLOYEE_ID	R2	LAST_NAME	R2	DEPARTMENT_ID
1	202	Fay		20	
2	201	Hartstein		20	

연습 문제 해답 6-1: Subquery 를 사용하여 데이터 검색(계속)

10) 모든 사원의 사원 ID, 성 및 부서 이름을 표시하는 query를 작성합니다.

참고: 스칼라 subquery를 사용하여 SELECT 문에서 부서 이름을 검색하십시오.

R	EMPLOYEE_ID	R	LAST_NAME	R	DEPARTMENT
1	205	Higgins	Accounting		
2	206	Gietz	Accounting		
3	200	Whalen	Administration		
4	100	King	Executive		
5	101	Kochhar	Executive		

105	196	Walsh	Shipping
106	197	Feeney	Shipping
107	178	Grant	(null)

11) 총 급여 비용이 전체 회사의 총 급여 비용의 8분의 1(1/8)을 초과하는 부서의 부서 이름을 표시하는 query를 작성합니다. WITH 절을 사용하여 이 query를 작성하고 query 이름을 SUMMARY로 지정합니다.

R	DEPARTMENT_NAME	R	DEPT_TOTAL
1	Sales	304500	
2	Shipping	156400	

연습 문제 해답 6-1: Subquery를 사용하여 데이터 검색

- 1) 임의 사원의 부서 번호와 급여를 커미션을 받는 사원의 부서 번호 및 급여와 비교하여 값이 일치하는 사원의 성, 부서 번호 및 급여를 표시하는 query를 작성합니다.

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (salary, department_id) IN
      (SELECT salary, department_id
       FROM   employees
       WHERE  commission_pct IS NOT NULL);
```

- 2) 급여와 커미션이 위치 ID1700에 위치한 사원의 급여 및 커미션과 일치하는 사원의 성, 부서 이름 및 급여를 표시합니다.

```
SELECT e.last_name, d.department_name, e.salary
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    (salary, NVL(commission_pct,0)) IN
      (SELECT salary, NVL(commission_pct,0)
       FROM   employees e, departments d
       WHERE  e.department_id = d.department_id
       AND d.location_id = 1700);
```

- 3) Kochhar와 동일한 급여 및 커미션을 받는 모든 사원의 성, 채용 날짜 및 급여를 표시하는 query를 작성합니다.

참고: 결과 집합에 Kochhar를 표시하지 마십시오.

```
SELECT last_name, hire_date, salary
FROM   employees
WHERE  (salary, NVL(commission_pct,0)) IN
      (SELECT salary, NVL(commission_pct,0)
       FROM   employees
       WHERE  last_name = 'Kochhar')
AND last_name != 'Kochhar';
```

- 4) 모든 영업 관리자(JOB_ID = 'SA_MAN')보다 많은 급여를 받는 사원을 표시하는 query를 작성합니다. 급여 결과를 하향식으로 정렬합니다.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary > ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'SA_MAN')
ORDER BY salary DESC;
```

연습 문제 해답 6-1: Subquery 를 사용하여 데이터 검색(계속)

- 5) 이름이 T로 시작하는 도시에 거주하는 사원의 사원 ID, 성 및 부서 ID와 같은 세부 정보를 표시합니다.

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE location_id IN
                            (SELECT location_id
                             FROM locations
                             WHERE city LIKE 'T%')));
```

- 6) 해당 부서의 평균 급여보다 급여 수준이 높은 모든 사원을 찾는 query를 작성합니다. 해당 부서에 대해 사원의 성, 급여, 부서 ID 및 평균 급여를 표시합니다. 평균 급여를 기준으로 정렬합니다. 예제 출력에 표시된 대로 query에 의해 검색되는 행에 alias를 사용합니다.

```
SELECT e.last_name ename, e.salary salary,
       e.department_id deptno, AVG(a.salary) dept_avg
FROM   employees e, employees a
WHERE  e.department_id = a.department_id
AND    e.salary > (SELECT AVG(salary)
                  FROM   employees
                  WHERE  department_id = e.department_id )
GROUP BY e.last_name, e.salary, e.department_id
ORDER BY AVG(a.salary);
```

- 7) 관리자가 없는 모든 사원을 찾습니다.

- a) NOT EXISTS 연산자를 사용하여 먼저 다음 작업을 수행합니다.

```
SELECT outer.last_name
FROM   employees outer
WHERE  NOT EXISTS (SELECT 'X'
                  FROM   employees inner
                  WHERE  inner.manager_id =
                        outer.employee_id);
```

- b) NOT IN 연산자를 사용하여 이 작업을 수행할 수 있습니까? 가능하다면 그 방법은 무엇이며 가능하지 않다면 그 이유는 무엇입니까?

```
SELECT outer.last_name
FROM   employees outer
WHERE  outer.employee_id
NOT IN (SELECT inner.manager_id
        FROM   employees inner);
```

두번째 방법은 바람직하지 않습니다. Subquery가 NULL 값을 취하므로 전체 query는 아무 행도 반환하지 않습니다. NULL 값을 비교하는 조건은 모두 NULL이 되기 때문에 행이 반환되지 않는 것입니다. 값 집합에 NULL 값이 포함될 경우에는 NOT EXISTS 대신 NOT IN을 사용하지 마십시오.

연습 문제 해답 6-1: Subquery 를 사용하여 데이터 검색(계속)

- 8) 해당 부서의 평균 급여보다 급여 수준이 낮은 사원의 성을 표시하는 query를 작성합니다.

```
SELECT last_name
FROM   employees outer
WHERE  outer.salary < (SELECT AVG(inner.salary)
                      FROM employees inner
                      WHERE inner.department_id
                        = outer.department_id);
```

- 9) 같은 부서에서 채용 날짜는 더 늦지만 더 높은 급여를 받는 1인 이상의 동료가 있는 사원의 성을 표시하는 query를 작성합니다.

```
SELECT last_name
FROM   employees outer
WHERE  EXISTS (SELECT 'X'
               FROM employees inner
               WHERE inner.department_id =
                     outer.department_id
               AND inner.hire_date > outer.hire_date
               AND inner.salary > outer.salary);
```

- 10) 모든 사원의 사원 ID, 성 및 부서 이름을 표시하는 query를 작성합니다.

참고: 스칼라 subquery를 사용하여 SELECT 문에서 부서 이름을 검색하십시오.

```
SELECT employee_id, last_name,
       (SELECT department_name
        FROM departments d
        WHERE e.department_id =
              d.department_id ) department
FROM employees e
ORDER BY department;
```

- 11) 총 급여 비용이 전체 회사의 총 급여 비용의 8분의 1(1/8)을 초과하는 부서의 부서 이름을 표시하는 query를 작성합니다. WITH 절을 사용하여 이 query를 작성하고 query 이름을 SUMMARY로 지정합니다.

```
WITH
summary AS (
  SELECT d.department_name, SUM(e.salary) AS dept_total
  FROM employees e, departments d
  WHERE e.department_id = d.department_id
  GROUP BY d.department_name)
SELECT department_name, dept_total
FROM summary
WHERE dept_total > ( SELECT SUM(dept_total) * 1/8
                   FROM summary )
ORDER BY dept_total DESC;
```

연습 7-1: 정규식 지원

이 연습에서는 정규식 함수를 사용하여 데이터를 검색, 대체 및 조작합니다. 또한 새 CONTACTS 테이블을 생성하고 해당 전화 번호가 특정 표준 형식으로 데이터베이스에 입력되도록 p_number 열에 CHECK 제약 조건을 추가합니다.

- 1) EMPLOYEES 테이블에서 이름이 "Ki" 또는 "Ko"로 시작하는 모든 사원을 검색하는 query를 작성합니다.

R	FIRST_NAME	R	LAST_NAME
1	Janette		King
2	Steven		King
3	Neena		Kochhar

- 2) LOCATIONS 테이블의 STREET_ADDRESS 열에서 공백을 제거하여 표시하는 query를 작성합니다. "Street Address"를 열 머리글로 사용합니다.

R	Street Address
1	1297ViaColadiRie
2	93091CalledellaTesta
3	2017Shinjuku-ku
4	9450Kamiya-cho
5	2014JabberwockyRd
6	2011InteriorsBlvd
7	2007ZagoraSt

- 3) LOCATIONS 테이블의 STREET_ADDRESS 열에서 "St"를 "Street"로 대체하여 표시하는 query를 작성합니다. 이미 "Street"가 있는 행에 영향을 주지 않도록 주의하십시오. 영향을 받는 행만 표시하십시오.

R	REGEXP_REPLACE(STREET_ADDRESS,'ST\$','STREET')
1	2007 Zagora Street
2	6092 Boxwood Street
3	12-98 Victoria Street
4	8204 Arthur Street

- 4) contacts 테이블을 생성하고 해당 전화 번호가 표준 형식 (XXX) XXX-XXXX로 데이터베이스에 입력되도록 다음 형식 마스크를 적용하여 p_number 열에 check 제약 조건을 추가합니다. 테이블에 다음 열이 있어야 합니다.

- l_name varchar2(30)
- p_number varchar2 (30)

연습 7-1: 정규식 지원(계속)

- 5) SQL 스크립트 lab_07_05.sql을 실행하여 다음 7개의 전화 번호를 contacts 테이블에 추가합니다. 어떤 번호가 추가되었습니까?

l_name 열 값	p_number 열 값
NULL	'(650) 555-;5555'
NULL	'(215) 555-;3427'
NULL	'650 555-;5555'
NULL	'650 555 ;5555'
NULL	'650-555-;5555'
NULL	'(650)555-;5555'
NULL	' (650) 555-;5555'

- 6) 문자열 gtctcgtctcgtttctgtctgtcgtttctg에서 DNA 패턴 ctc의 발생 수를 찾는 query를 작성합니다. 대소문자는 구분하지 않습니다.

COUNT_DNA	
1	2

연습 문제 해답 7-1: 정규식 지원

- 1) EMPLOYEES 테이블에서 이름이 "Ki" 또는 "Ko"로 시작하는 모든 사원을 검색하는 query를 작성합니다.

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (last_name, '^K(i|o).');
```

- 2) LOCATIONS 테이블의 STREET_ADDRESS 열에서 공백을 제거하여 표시하는 query를 작성합니다. "Street Address"를 열 머리글로 사용합니다.

```
SELECT regexp_replace (street_address, ' ', '') AS "Street
Address"
FROM locations;
```

- 3) LOCATIONS 테이블의 STREET_ADDRESS 열에서 "St"를 "Street"로 대체하여 표시하는 query를 작성합니다. 이미 "Street"가 있는 행에 영향을 주지 않도록 주의하십시오. 영향을 받는 행만 표시합니다.

```
SELECT regexp_replace (street_address, 'St$', 'Street')
FROM locations
WHERE regexp_like (street_address, 'St');
```

- 4) contacts 테이블을 생성하고 해당 전화 번호가 표준 형식 (XXX) XXX-XXXX로 데이터베이스에 입력되도록 다음 형식 마스크를 적용하여 p_number 열에 check 제약 조건을 추가합니다. 테이블에 다음 열이 있어야 합니다.

- l_name varchar2(30)
- p_number varchar2 (30)

```
CREATE TABLE contacts
(
  l_name      VARCHAR2(30),
  p_number    VARCHAR2(30)
  CONSTRAINT p_number_format
  CHECK ( REGEXP_LIKE ( p_number, '^(\d{3}) \d{3}-
\d{4}$' ) )
);
```

- 5) lab_07_05.sql SQL 스크립트를 실행하여 다음 7개의 전화 번호를 contacts 테이블에 삽입합니다. 어떤 번호가 추가되었습니까?

처음 두 개의 INSERT 문은 c_contacts_pnf 제약 조건을 따르는 형식을 사용하지만, 나머지 문에서는 CHECK 제약 조건 오류가 발생합니다.

- 6) 문자열 gtctcgtctcgttctgtctgtcgttctg에서 DNA 패턴 ctc의 발생 수를 찾는 query를 작성합니다. Alias Count_DNA를 사용합니다. 대소문자는 구분하지 않습니다.

```
SELECT REGEXP_COUNT('gtctcgtctcgttctgtctgtcgttctg', 'ctc')
AS Count_DNA
FROM dual;
```


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)

본 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 식별 코드, 직위, 최소 및 최대 급여 범위가 있습니다. 일부 사원은 오랫동안 회사에 근무하며 사내에서 다양한 업무를 맡았습니다. 사원이 사임하면 사원이 근무한 기간, 직무 식별 번호 및 부서가 기록됩니다.

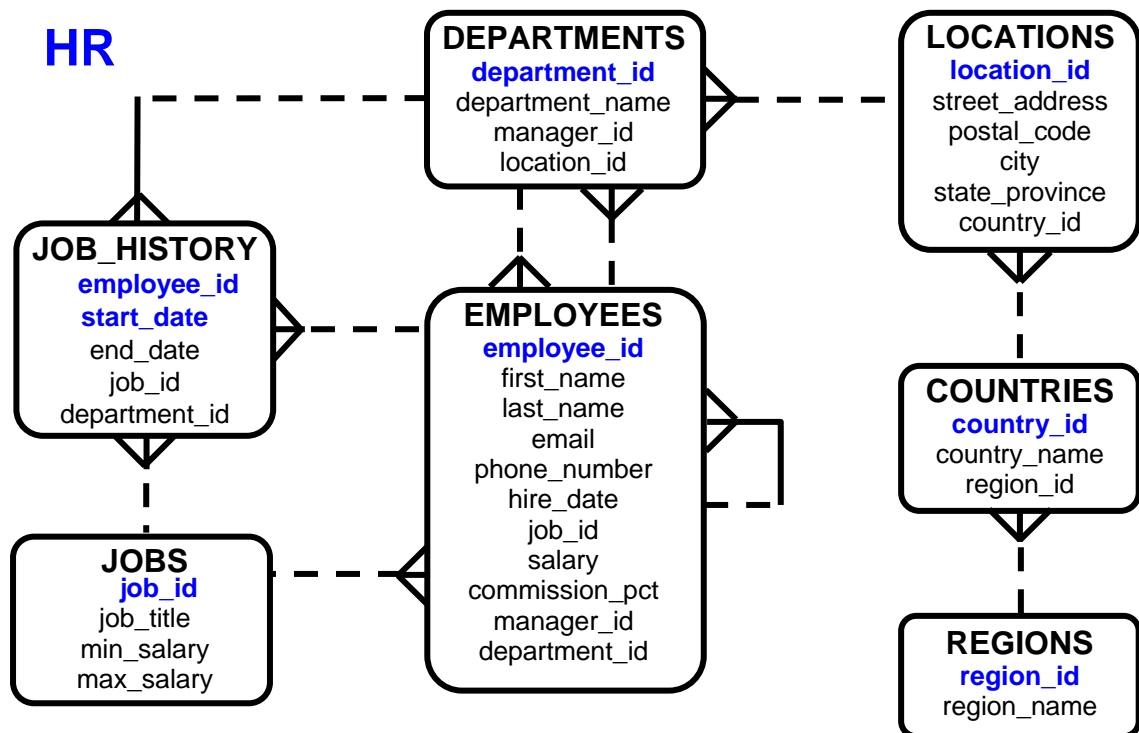
예제 회사는 여러 지역에 분포되어 있으므로 회사의 물류 창고 및 부서 위치를 추적합니다.

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	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

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	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

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_N...	LAST_N...	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMI...	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	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	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	(null)	103	60
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800	(null)	100	50
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500	(null)	124	50
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100	(null)	124	50
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600	(null)	124	50
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500	(null)	124	50
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500	0.2	100	80
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000	0.3	149	80
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600	0.2	149	80
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	0.15	149	(null)
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000	(null)	201	20
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110
20	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACC...	8300	(null)	205	110

HR(Human Resources) 테이블 설명(계속)

```
DESCRIBE job_history
```

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	AC_MGR	Accounting Manager	8200	16000
5	AC_ACCOUNT	Public Accountant	4200	9000
6	SA_MAN	Sales Manager	10000	20000
7	SA_REP	Sales Representative	6000	12000
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2000	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

HR(Human Resources) 테이블 설명(계속)

DESCRIBE locations

Name	Null	Type
-----	-----	-----
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)
STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

SELECT * FROM locations



	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700	2004 Charade Rd	98199	Seattle	Washington	US
4	1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

HR(Human Resources) 테이블 설명(계속)

```
DESCRIBE regions
```

Name	Null	Type
-----	-----	-----
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

```
SELECT * FROM regions
```

	 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 작성 및 저장

ORACLE

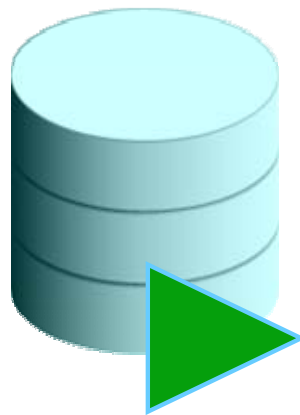
Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 SQL Developer라는 그래픽 도구에 대해 알아봅니다. 먼저, 데이터베이스 개발 작업에 SQL Developer를 사용하는 방법에 대해 알아보고, SQL Worksheet를 사용하여 SQL 문과 SQL 스크립트를 실행하는 방법을 학습합니다.

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 Developer를 사용하면 다음 작업을 간단하게 수행할 수 있습니다.

- 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 사용자가 단일 위치에서 third-party 데이터베이스의 데이터 및 데이터베이스 객체를 탐색하고 이러한 데이터베이스에서 오라클로 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 홈이 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

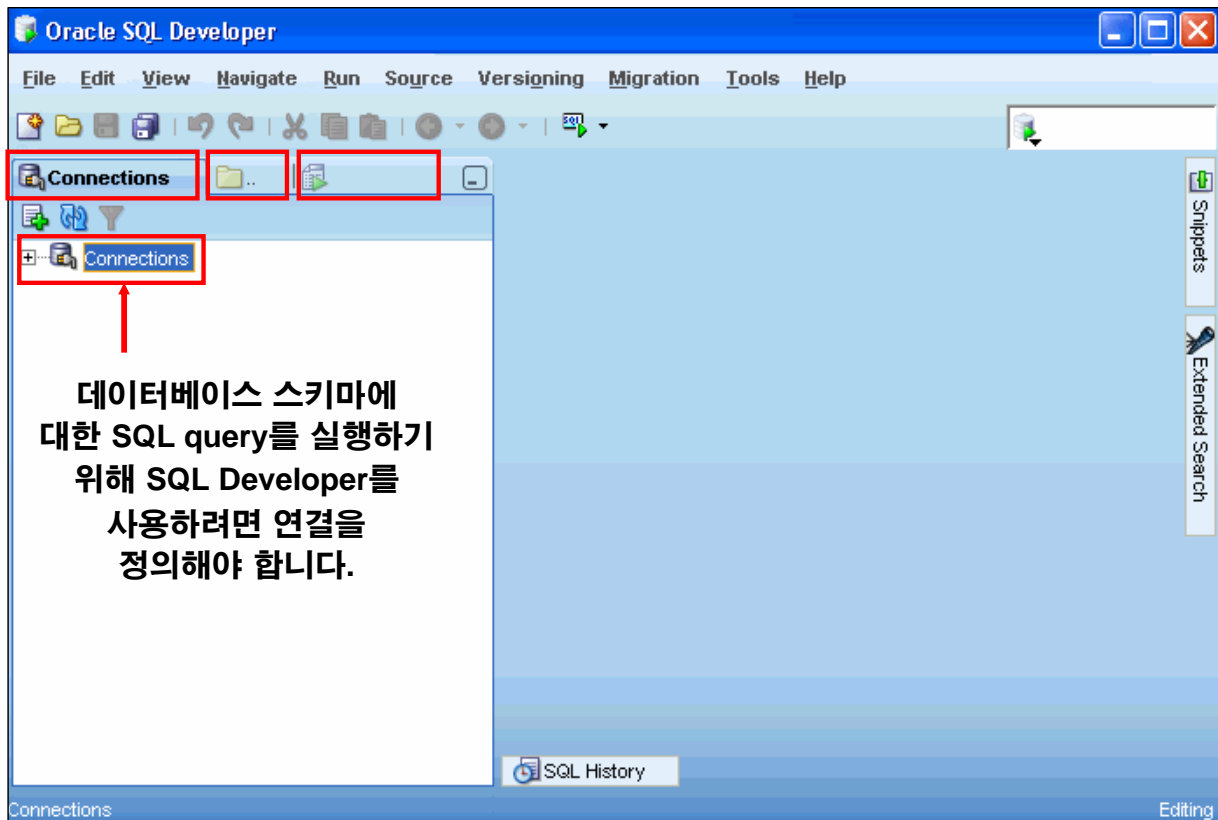
Oracle Database 11g Release 2 이전의 Oracle Database 버전을 사용하는 경우 SQL Developer를 다운로드하여 설치해야 합니다. SQL Developer 1.5는 다음 링크에서 무료로 다운로드할 수 있습니다.

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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 1.5 인터페이스

SQL Developer 1.5에는 다음과 같은 세 개의 기본 탐색 탭(왼쪽부터)이 있습니다.

- **Connections 탭:** 이 탭을 사용하면 액세스할 수 있는 데이터베이스 객체 및 유저를 찾아볼 수 있습니다.
- **Files 탭:** Files 폴더 아이콘으로 식별됩니다. 이 탭을 사용하면 File > Open 메뉴를 사용하지 않고도 로컬 시스템에서 파일에 액세스할 수 있습니다.
- **Reports 탭:** Reports 아이콘으로 식별됩니다. 이 탭을 사용하면 미리 정의된 보고서를 실행하거나 사용자 보고서를 직접 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 커스터마이징할 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 익스포트할 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 하는데, 이러한 연결은 기존 연결일 수도 있고 새로 생성하거나 임포트할 수도 있습니다.

여러 데이터베이스 및 여러 스키마에 대해 연결을 생성하고 테스트할 수 있습니다.

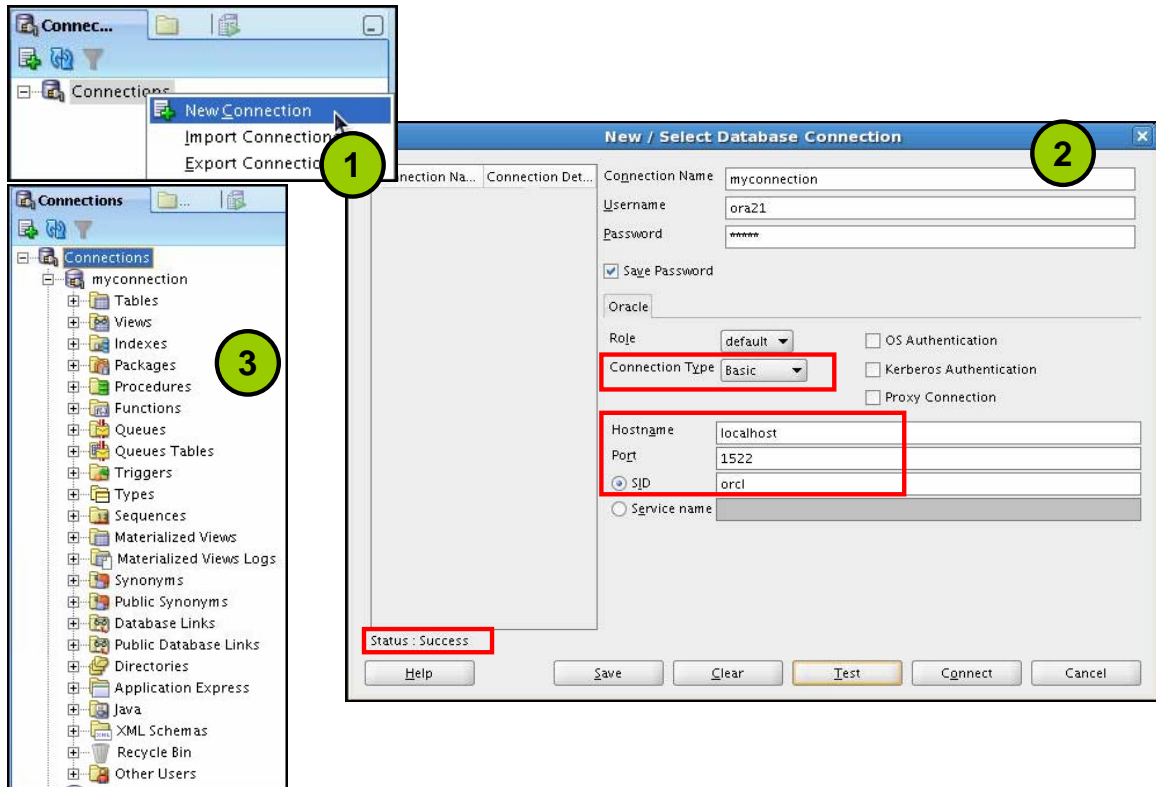
기본적으로 `tnsnames.ora` 파일은 `$ORACLE_HOME/network/admin` 디렉토리에 있지만, `TNS_ADMIN` 환경 변수 또는 레지스트리 값에 지정된 디렉토리에 있을 수도 있습니다. SQL Developer를 시작하고 Database Connections 대화상자를 표시하면 SQL Developer는 시스템의 `tnsnames.ora` 파일에 정의된 모든 연결을 자동으로 임포트합니다.

참고: Windows에서 `tnsnames.ora` 파일이 존재하지만 SQL Developer가 해당 연결을 사용하고 있지 않으면 `TNS_ADMIN`을 시스템 환경 변수로 정의하십시오.

연결을 XML 파일로 익스포트하여 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 사용자 정의 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. 따라서 SQL Developer 연결을 닫았다가 다시 열 때 암호를 입력하라는 메시지가 표시되지 않습니다.

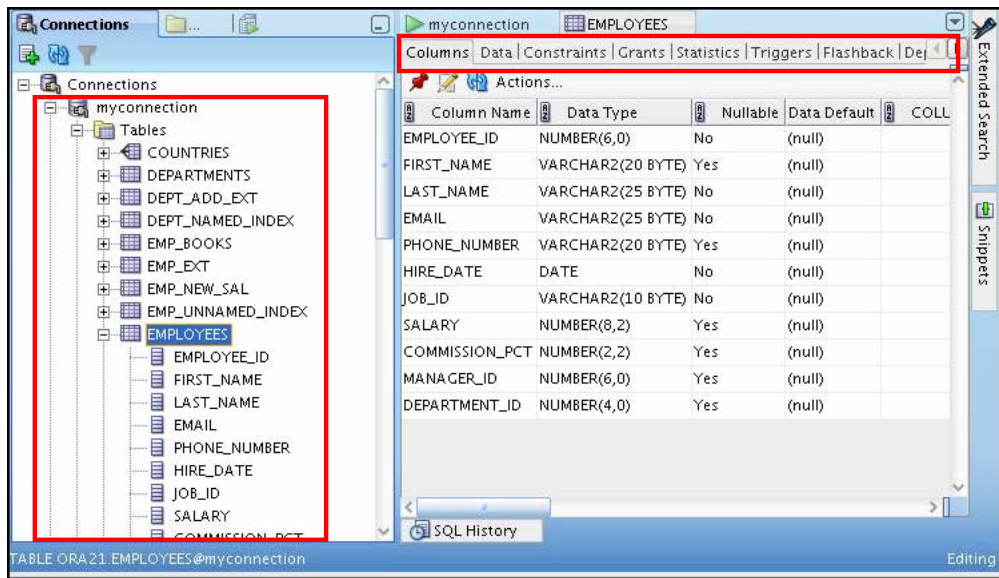
3. Connections Navigator에 연결이 추가됩니다. 연결을 확장하여 데이터베이스 객체를 볼 수 있으며 종속성, 상세 내역, 통계 등의 객체 정의를 볼 수 있습니다.

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.

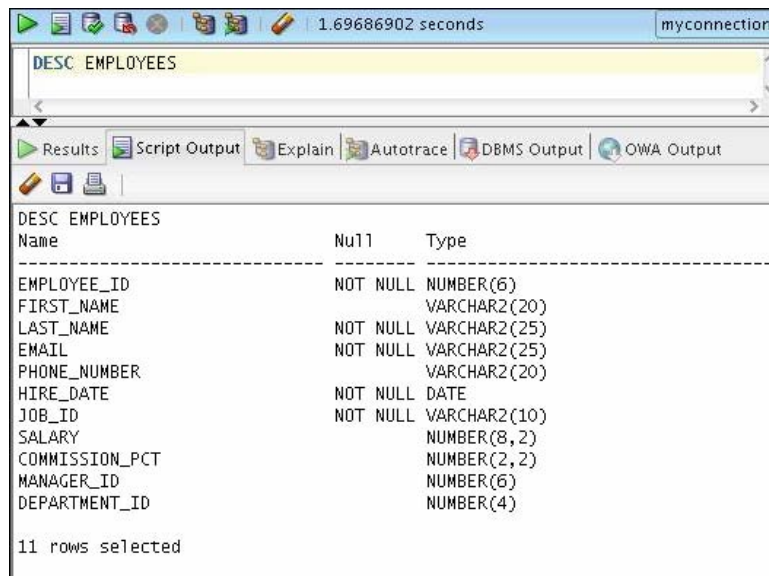
데이터 디렉터리에서 추출된 객체 정의가 여러 정보 탭에 분할되어 있는 것을 볼 수 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 읽기 쉬운 탭 페이지에 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 세부 정보가 표시됩니다.

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 SQL Developer interface with the command 'DESC EMPLOYEES' entered in the SQL window. The output is displayed in the Results tab, showing the structure of the EMPLOYEES table. The output is as follows:

DESC EMPLOYEES	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

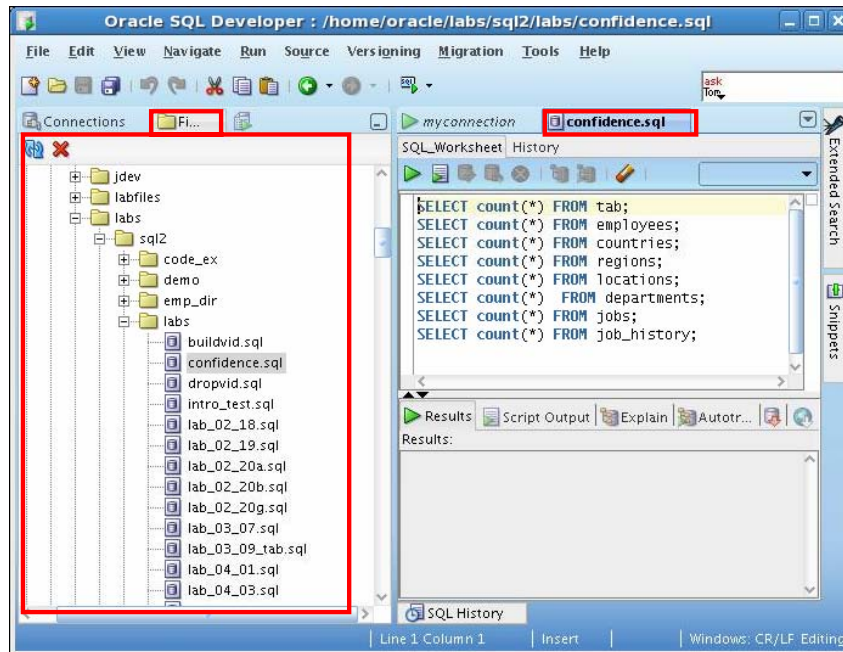
Copyright © 2009, Oracle. All rights reserved.

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.

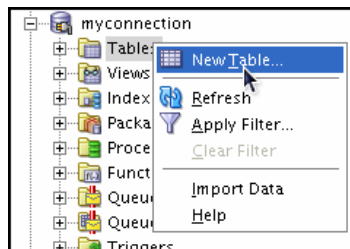
데이터베이스 객체 탐색

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.

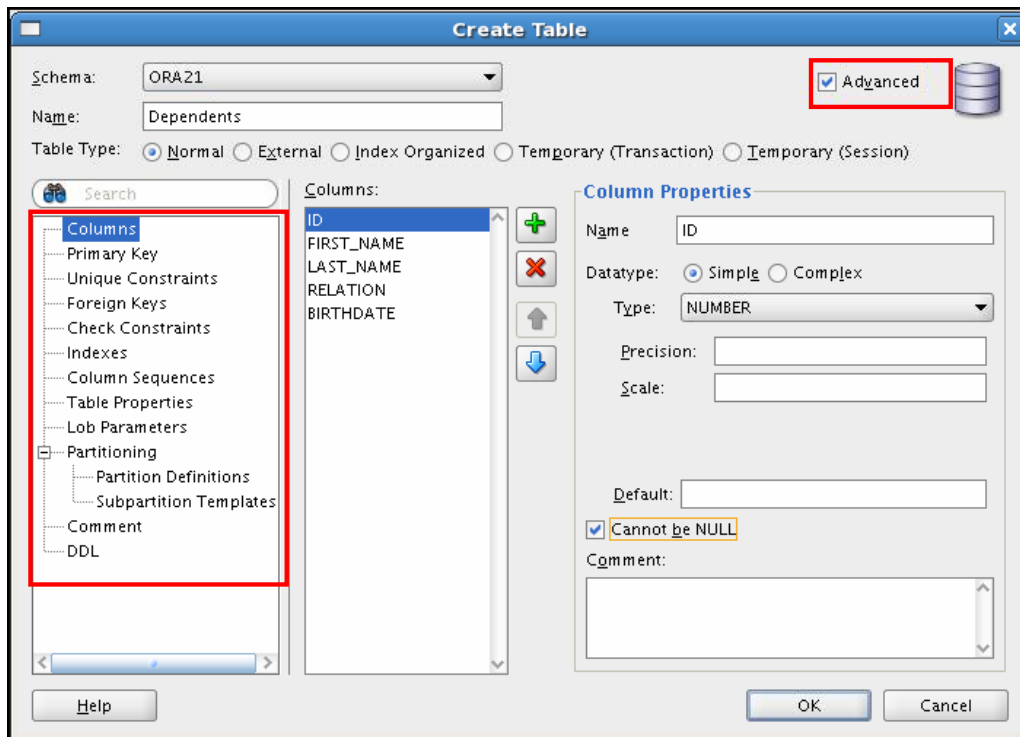
스키마 객체 생성

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.

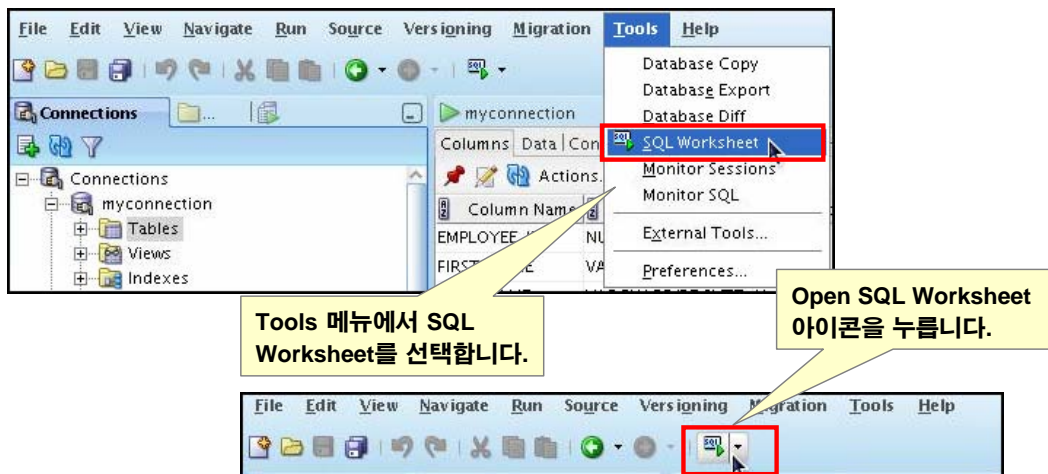
새 테이블을 생성하려면 다음 단계를 수행하십시오.

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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Worksheet 사용

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. SQL Worksheet는 특정 범위까지 SQL*Plus 문을 지원합니다. SQL Worksheet에서 지원되지 않는 SQL*Plus 문은 무시되고 데이터베이스에 전달되지 않습니다.

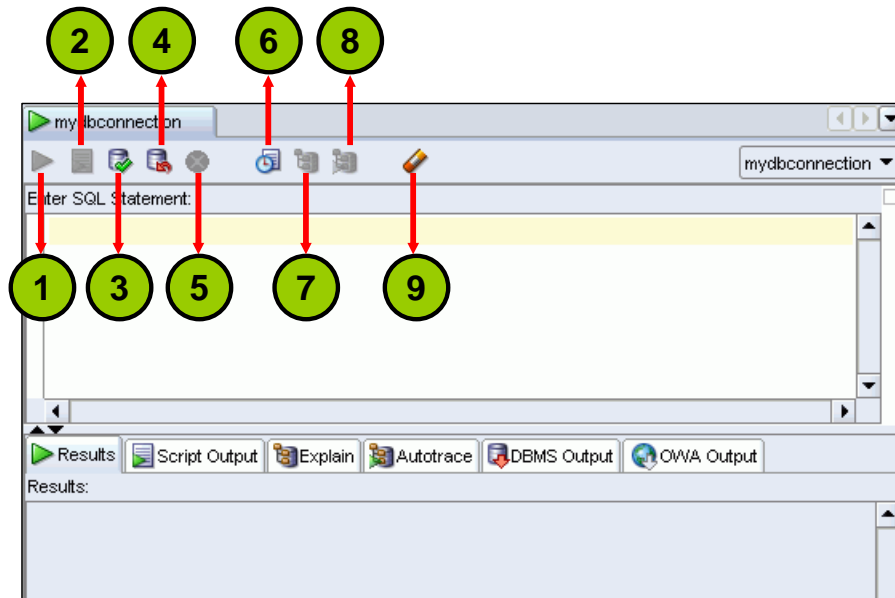
다음과 같이 워크시트와 연관된 데이터베이스 연결에 의해 처리될 수 있는 작업을 지정합니다.

- 테이블 생성
- 데이터 삽입
- 트리거 생성 및 편집
- 테이블에서 데이터 선택
- 파일에 선택한 데이터 저장

다음 중 하나를 사용하여 SQL Worksheet를 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- Open SQL Worksheet 아이콘을 누릅니다.

SQL Worksheet 사용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

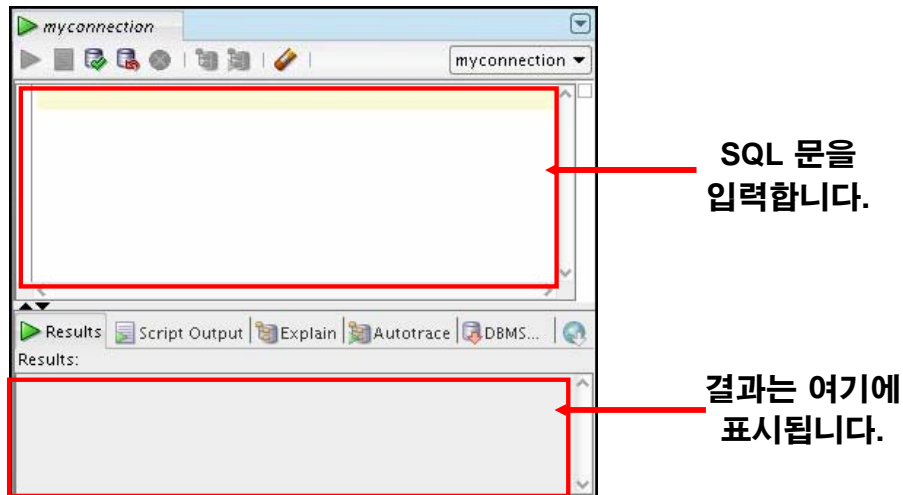
SQL Worksheet 사용(계속)

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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Worksheet 사용(계속)

데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet를 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. 모든 SQL 및 PL/SQL 명령이 지원되며 이러한 명령은 SQL Worksheet에서 오라클 데이터베이스로 직접 전달됩니다. 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.

The screenshot shows the Oracle SQL Developer interface. The 'Enter SQL Statement' window contains the following SQL query:

```
SELECT employee_id, last_name
FROM employees;
```

The 'Results' window displays the following data:

EMPLOYEE_ID	LAST_NAME
1	100 King
2	101 Kochhar
3	102 De Haan
4	103 Hunold
5	104 Ernst

The 'Script Output' window displays the same data in a text format:

```
EMPLOYEE_ID      LAST_NAME
-----
100              King
101              Kochhar
102              De Haan
103              Hunold
104              Ernst
105              Austin
```

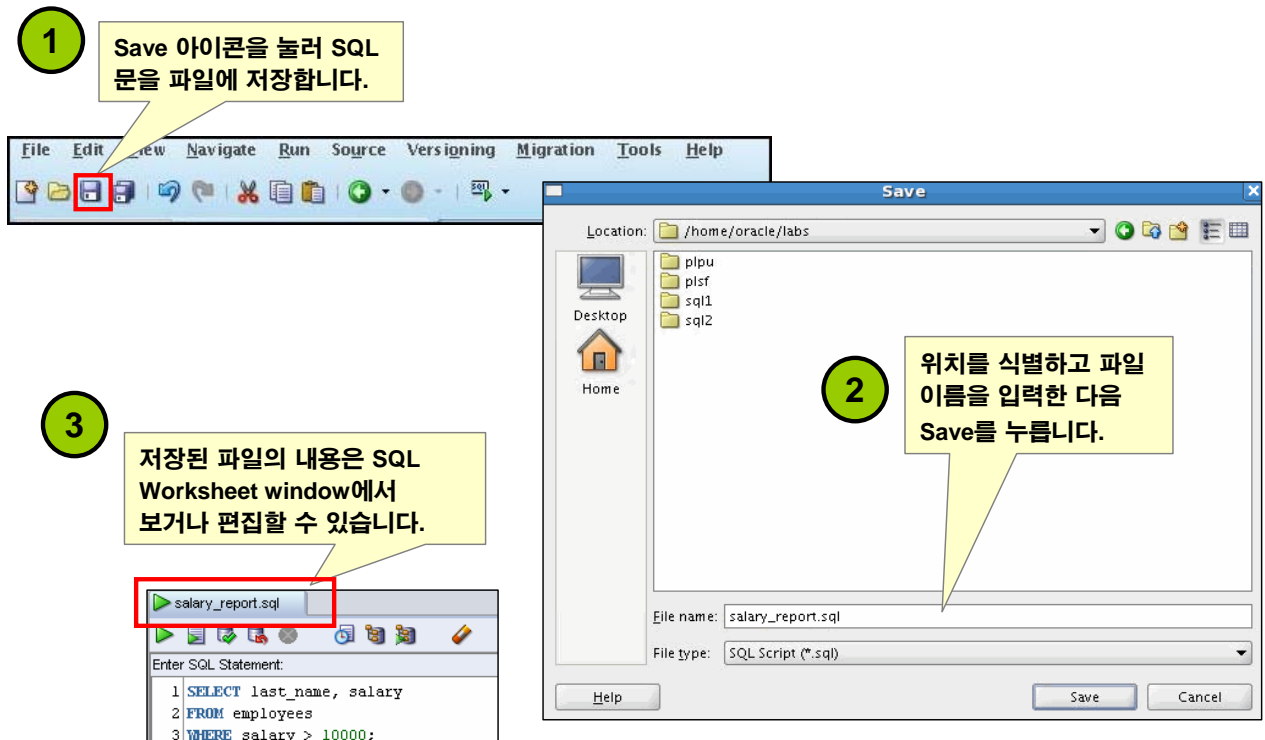
ORACLE

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.

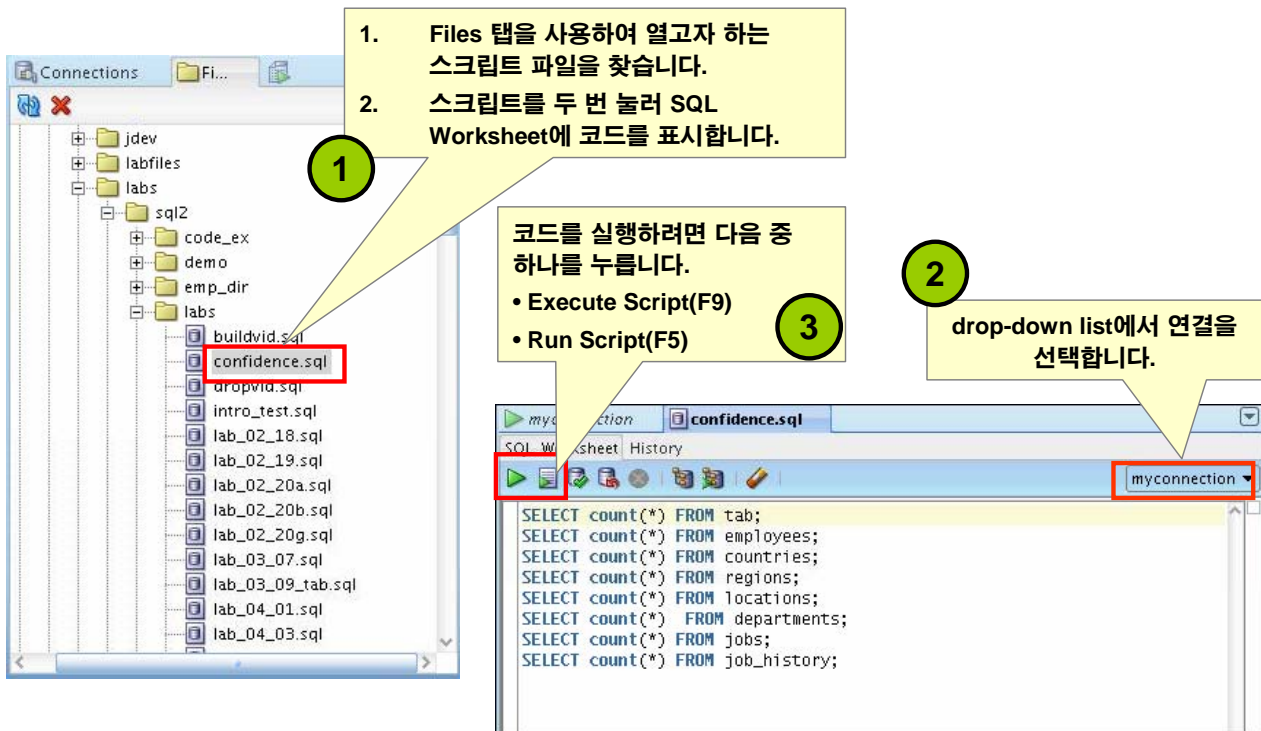
1. Save 아이콘을 누르거나 File > Save 메뉴 항목을 사용합니다.
2. Windows Save 대화상자에서 파일 이름과 파일을 저장할 위치를 입력합니다.
3. Save를 누릅니다.

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

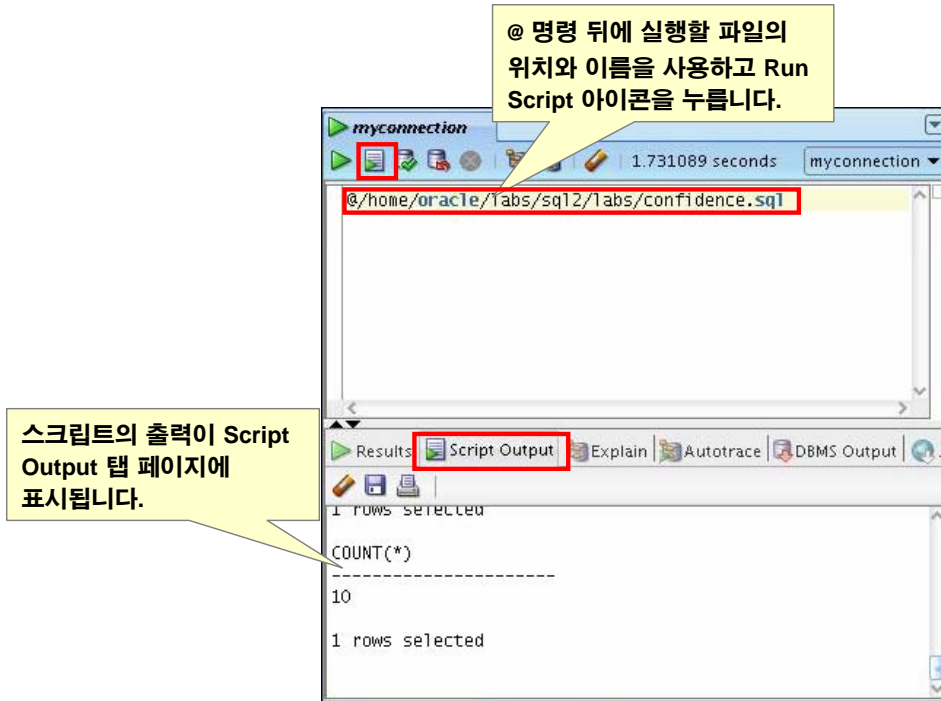
스크립트 파일을 열고 SQL Worksheet 영역에 코드를 표시하려면 다음을 수행합니다.

1. Files Navigator에서 열고자 하는 스크립트 파일을 선택합니다. 또는 해당 파일이 있는 위치로 이동합니다.
2. 두 번 눌러 엽니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
3. connection drop-down list에서 연결을 선택합니다.
4. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다. connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다. 스크립트 실행에 사용할 연결을 선택합니다.

또는 다음과 같이 할 수도 있습니다.

1. File > Open을 선택합니다. Open 대화상자가 표시됩니다.
2. Open 대화상자에서 열고자 하는 스크립트 파일을 선택합니다. (또는 해당 파일이 있는 위치로 이동합니다).
3. Open을 누릅니다. 스크립트 파일의 코드가 SQL Worksheet 영역에 표시됩니다.
4. connection drop-down list에서 연결을 선택합니다.
5. 코드를 실행하려면 SQL Worksheet 도구 모음에서 Run Script (F5) 아이콘을 누릅니다. connection drop-down list에서 연결을 선택하지 않은 경우 connection 대화상자가 나타납니다. 스크립트 실행에 사용할 연결을 선택합니다.

저장된 SQL 스크립트 실행: 방법 2



ORACLE

Copyright © 2009, Oracle. All rights reserved.

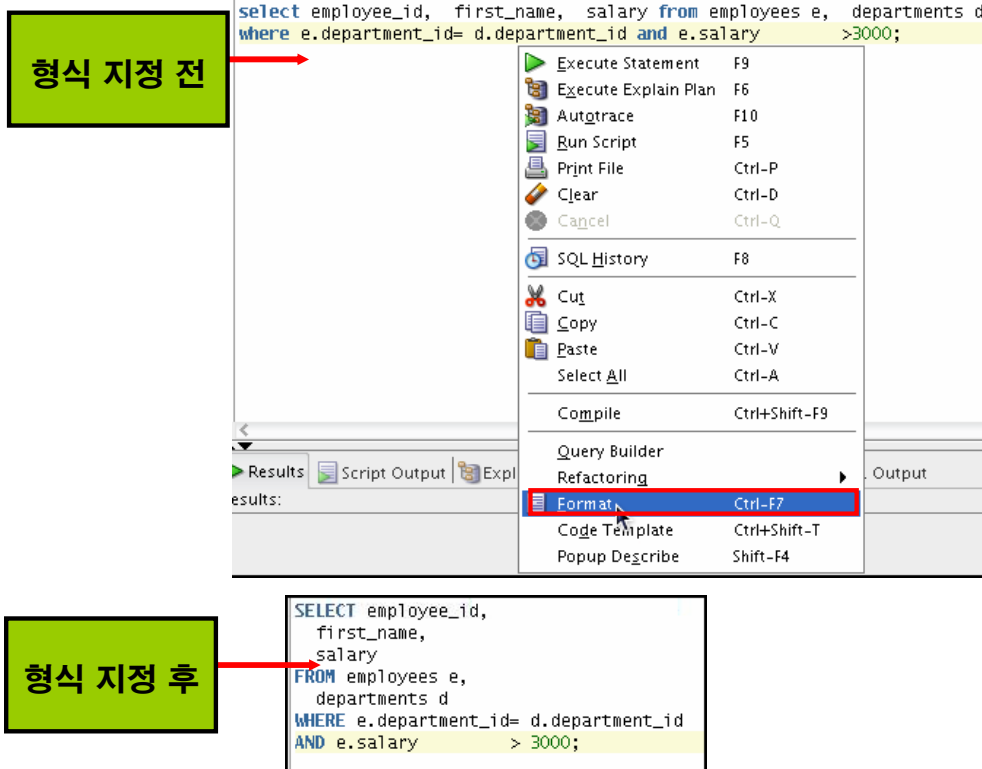
저장된 SQL 스크립트 실행: 방법 2

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 식별할 수 있습니다.

SQL 코드 형식 지정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 코드 형식 지정

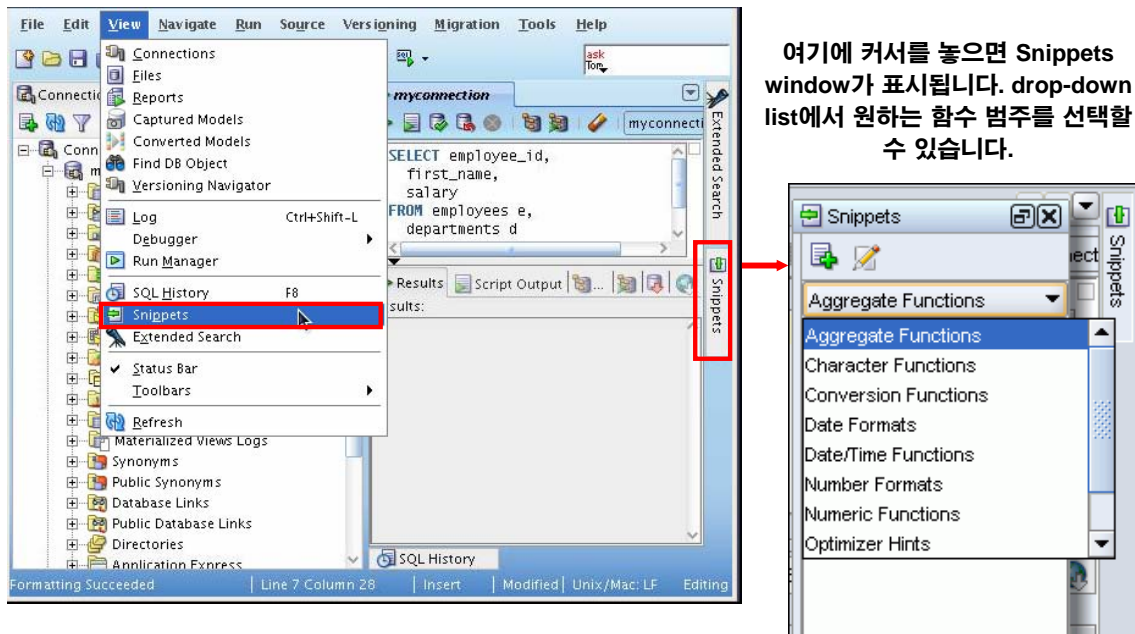
SQL 코드의 들여쓰기, 간격, 대소문자 및 줄 구분을 보기 좋게 지정해야 할 경우가 있습니다. SQL Developer에는 SQL 코드의 형식을 지정하는 기능이 있습니다.

SQL 코드에 형식을 지정하려면 명령문 영역을 마우스 오른쪽 버튼으로 누른 다음 **Format SQL**을 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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Snippet 사용

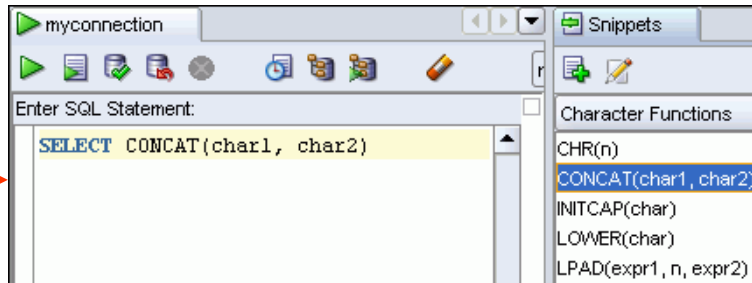
SQL Worksheet를 사용하거나 PL/SQL 함수 또는 프로시저를 생성하거나 편집할 때 특정 코드 부분을 사용해야 하는 경우가 있습니다. SQL Developer에는 Snippet이라는 기능이 있습니다. Snippet은 SQL 함수, 옵티마이저 힌트 및 기타 PL/SQL 프로그래밍 기법과 같은 코드 부분입니다. Snippet을 Editor window로 끌어올 수 있습니다.

Snippet을 표시하려면 View > Snippets를 선택합니다.

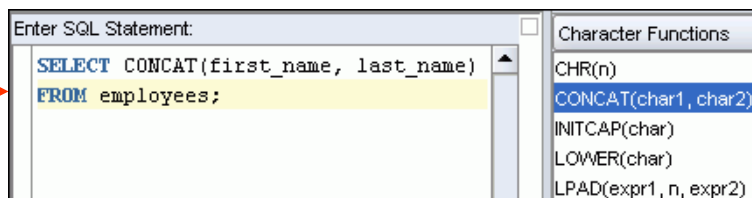
그러면 Snippets window가 오른쪽에 표시됩니다. drop-down list를 사용하여 그룹을 선택할 수 있습니다. Snippets window가 숨겨진 경우 이를 표시할 수 있도록 오른쪽 window 여백에 Snippets 버튼이 표시됩니다.

Snippet 사용: 예제

Snippet 삽입



Snippet 편집



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Snippet 사용: 예제

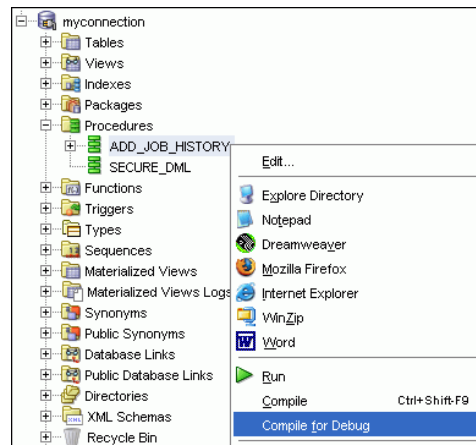
SQL Worksheet 또는 PL/SQL 함수나 프로시저의 코드에 Snippet을 삽입하려면 Snippets window에서 코드의 원하는 위치로 Snippet을 끌어옵니다. 그런 다음 SQL 함수가 현재 컨텍스트에서 유효하도록 구문을 편집할 수 있습니다. 도구 설명에서 SQL 함수에 대한 간단한 설명을 보려면 커서를 함수 이름 위에 둡니다.

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 디버깅합니다.
- 프로시저를 디버깅할 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

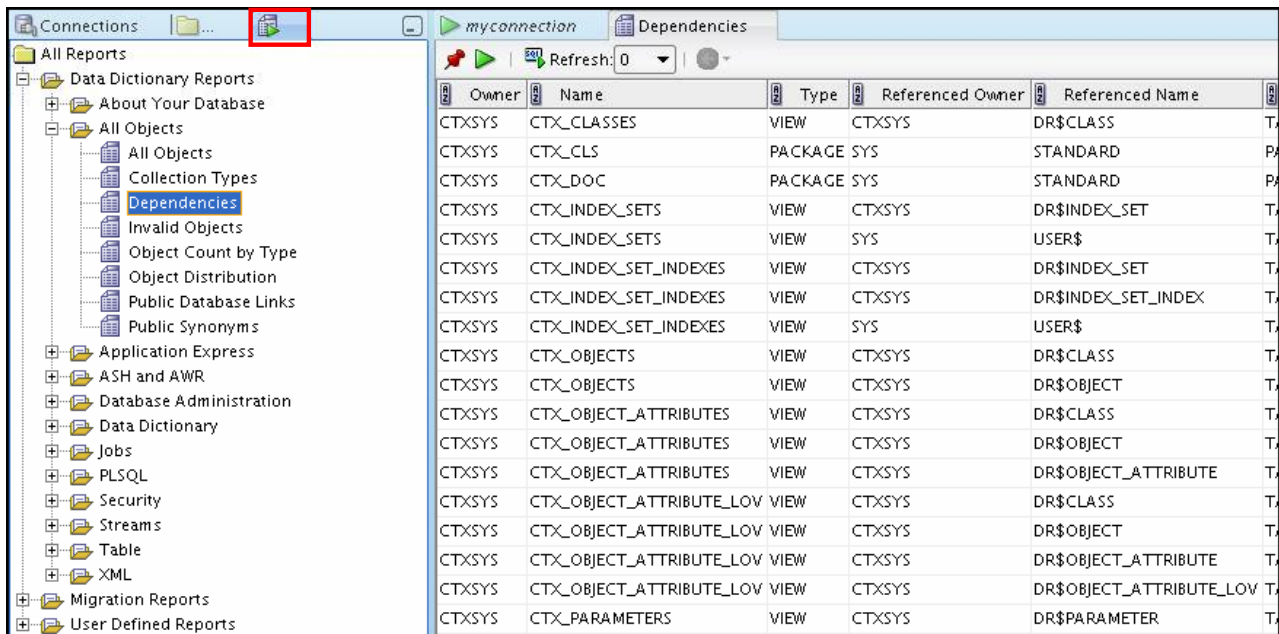
SQL Developer에서는 PL/SQL 프로시저 및 함수를 디버깅할 수 있습니다. Debug 메뉴 옵션을 사용하여 다음 디버깅 작업을 수행할 수 있습니다.

- **Find Execution Point**를 사용하면 다음 실행 지점으로 이동합니다.
- **Resume**를 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**를 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.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 보고

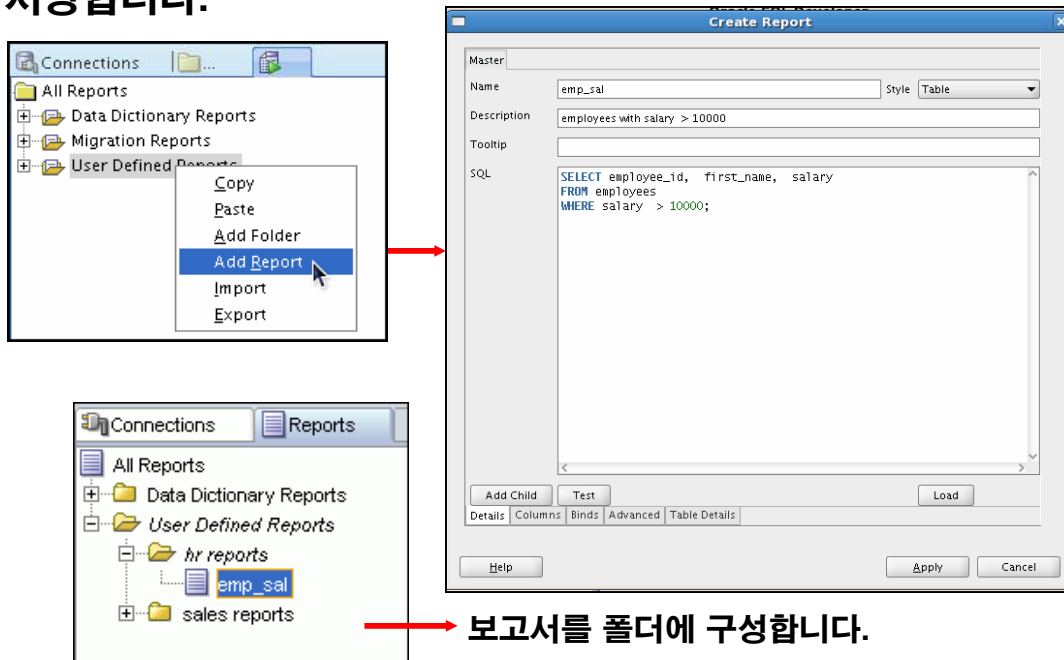
SQL Developer는 데이터베이스 및 객체에 대한 다양한 보고서를 제공합니다. 이러한 보고서는 다음 범주로 그룹화할 수 있습니다.

- 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 사용자가 볼 수 있는 객체만 표시되고 행은 일반적으로 Owner별로 정렬됩니다. 고유한 사용자 정의 보고서를 작성할 수도 있습니다.

유저 정의 보고서 작성

반복적으로 사용할 수 있도록 유저 정의 보고서를 생성하고 저장합니다.



보고서를 폴더에 구성합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

유저 정의 보고서 작성

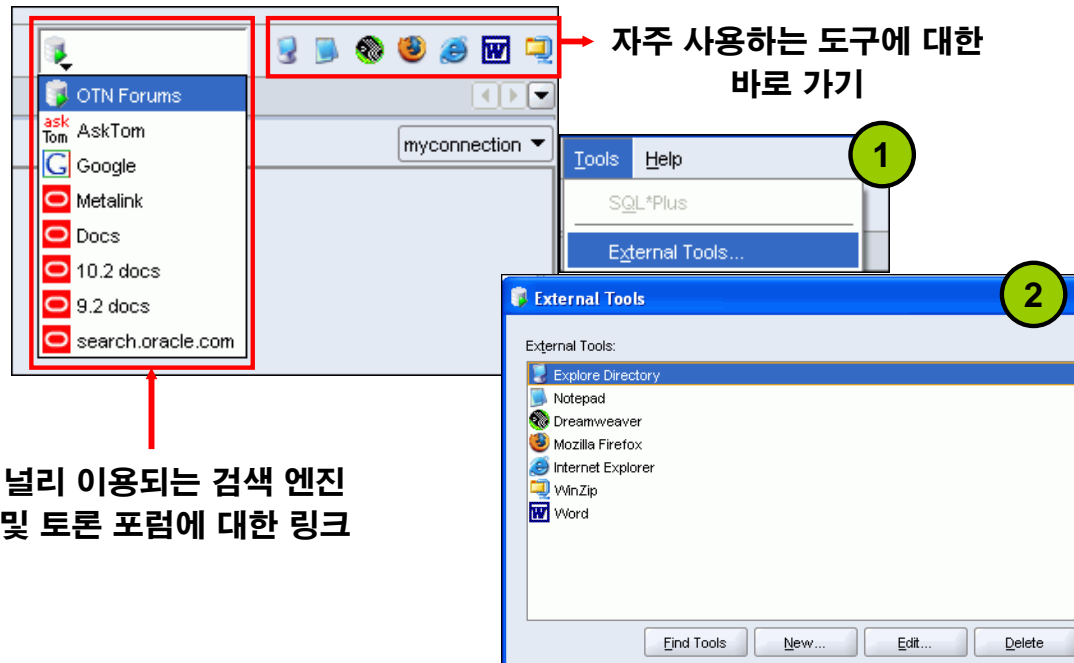
유저 정의 보고서는 SQL Developer 유저가 생성하는 보고서입니다. 유저 정의 보고서를 작성하려면 다음 단계를 수행하십시오.

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 salary >= 10000인 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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

검색 엔진 및 External 도구

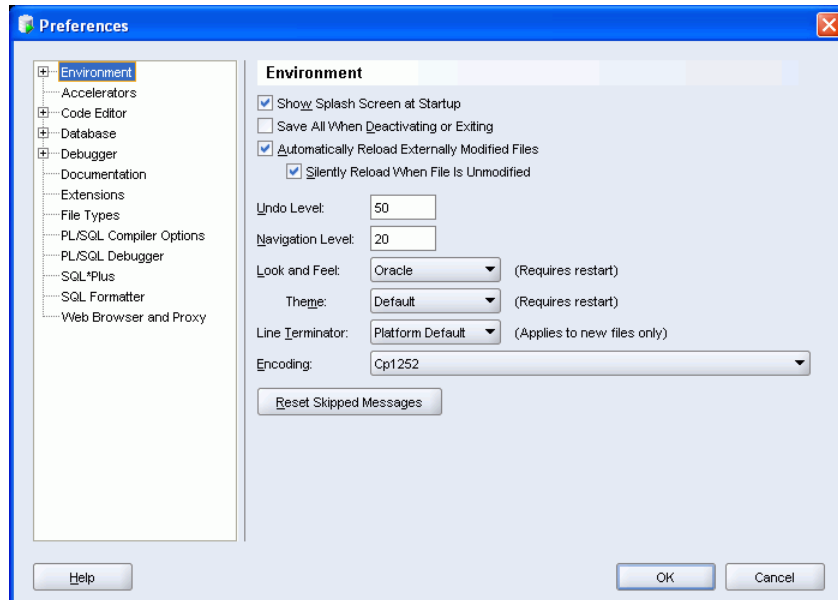
SQL 개발자의 생산성을 높이기 위해 SQL Developer에는 AskTom, Google 등의 유명 검색 엔진과 토의 포럼에 대한 빠른 링크가 포함되어었습니다. 또한 메모장, Microsoft Word, Dreamweaver 등의 자주 사용하는 일부 도구에 대한 단축키 아이콘도 사용할 수 있습니다.

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 커스터마이징합니다.
- Tools 메뉴에서 Preferences를 선택합니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

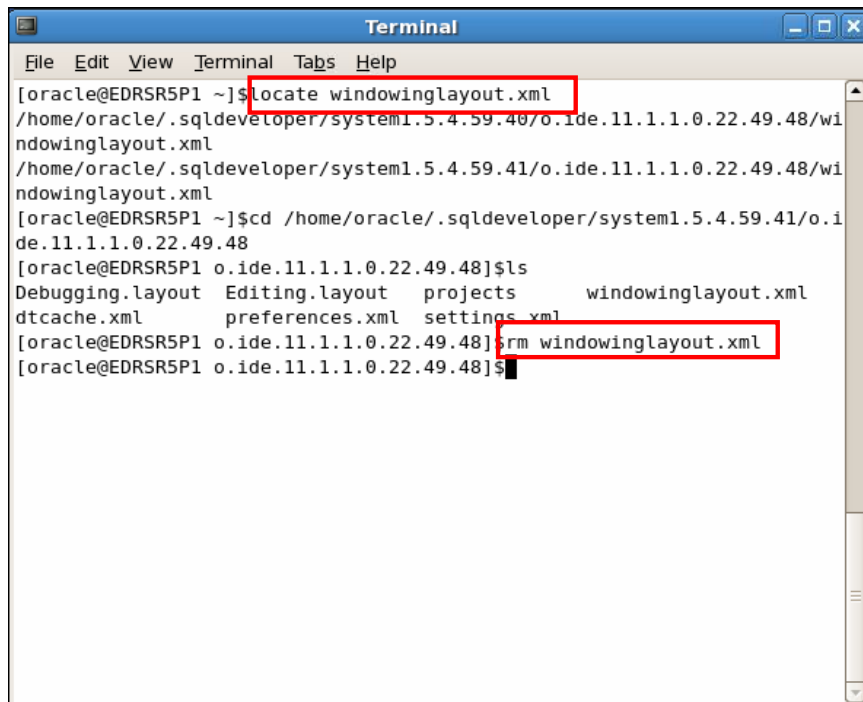
환경 설정

SQL Developer 환경 설정을 자신의 취향과 요구 사항에 맞게 수정함으로써 SQL Developer 인터페이스와 환경의 다양한 요소를 커스터마이징할 수 있습니다. SQL Developer 환경 설정을 수정하려면 Tools와 Preferences를 차례로 선택합니다.

환경 설정은 다음 범주로 분류되어 있습니다.

- 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
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ ls
Debugging.layout  Editing.layout  projects        windowinglayout.xml
dtcache.xml       preferences.xml  settings.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$ rm windowinglayout.xml
[oracle@EDRSR5P1 o.ide.11.1.1.0.22.49.48]$
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL Developer 레이아웃 재설정

SQL Developer로 작업하는 동안 Connections Navigator가 사라지거나 Log window를 원래의 위치에 도킹할 수 없는 경우 다음 단계를 수행하여 문제를 해결합니다.

1. SQL Developer를 종료합니다.
2. 터미널 window를 열고 locate 명령을 사용하여 windowinglayout.xml의 위치를 찾습니다.
3. windowinglayout.xml이 있는 디렉토리로 이동하여 해당 파일을 삭제합니다.
4. SQL Developer를 재시작합니다.

요약

이 부록에서는 SQL Developer를 사용하여 다음 작업을 수행하는 방법을 배웠습니다.

- 데이터베이스 객체 탐색, 생성 및 편집
- SQL Worksheet에서 SQL 문 및 스크립트 실행
- 사용자 정의 보고서 작성 및 저장

ORACLE

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.



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**

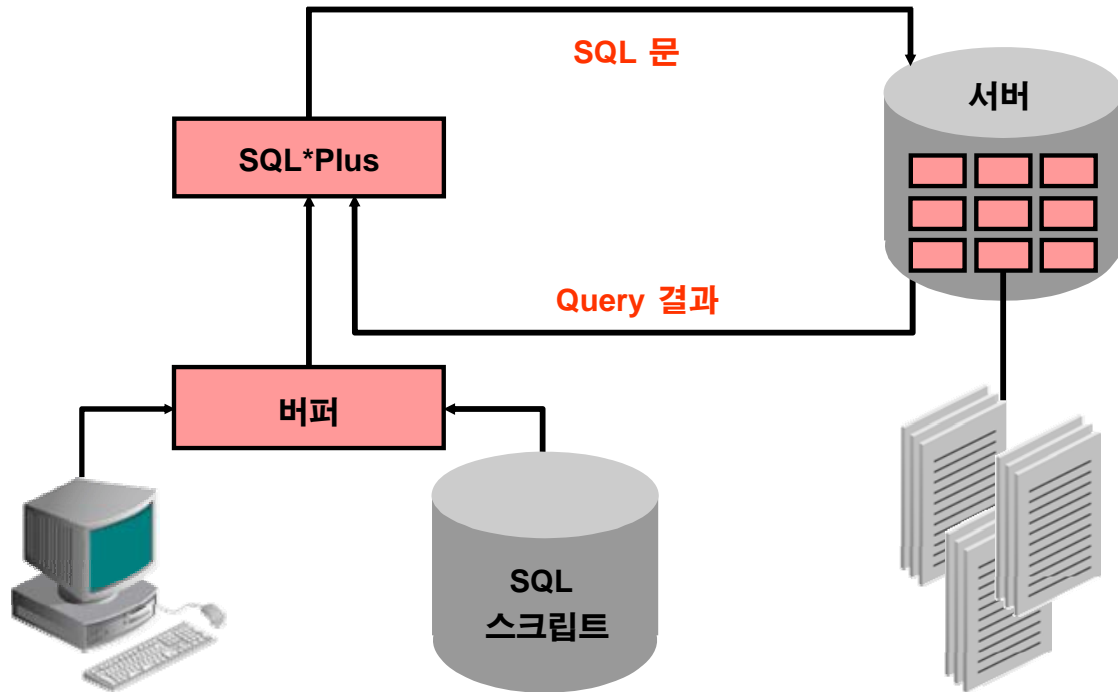
ORACLE

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 다룹니다. `SQL*Plus` 명령을 사용하여 출력 형식을 지정하고, `SQL` 명령을 편집하고, `SQL*Plus`로 스크립트를 저장하는 방법을 배웁니다.

SQL과 SQL*Plus의 상호 작용



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus

SQL은 임의의 도구 또는 응용 프로그램에서 Oracle 서버와 통신하는 데 사용하는 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 오라클 도구로, 자체 명령어를 포함합니다.

SQL의 특성

- 프로그래밍의 경험이 별로 없거나 아예 없는 유저를 포함하여 다양한 유저가 사용할 수 있습니다.
- 비절차적 언어입니다.
- 시스템 생성 및 유지 관리에 필요한 시간을 줄여 줍니다.
- 영어와 유사한 언어입니다.

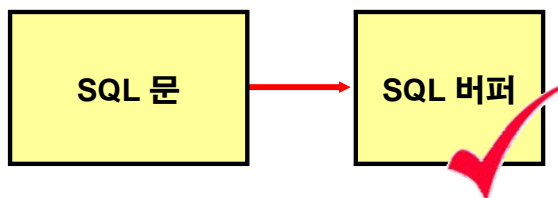
SQL*Plus의 특성

- 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

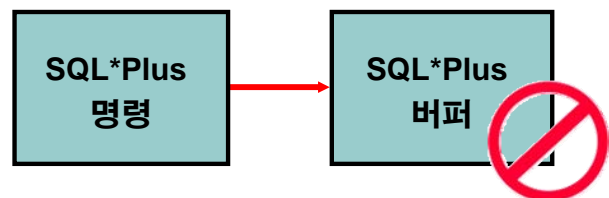
SQL

- 언어
- ANSI 표준
- 키워드를 약어로 표기할 수 없음
- 명령문으로 데이터베이스의 데이터 및 테이블 정의를 조작함



SQL*Plus

- 환경
- Oracle 고유
- 키워드를 약어로 표시할 수 있음
- 명령으로 데이터베이스의 값을 조작할 수 없음



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL과 SQL*Plus(계속)

다음 표에서는 SQL과 SQL*Plus를 비교합니다.

SQL	SQL*Plus
Oracle 서버와 통신하여 데이터에 액세스하기 위한 언어	SQL 문을 인식하고 서버로 보냄
ANSI(American National Standards Institute) 표준 SQL을 기반으로 함	SQL 문을 실행하기 위한 오라클 고유의 인터페이스
데이터베이스의 데이터 및 테이블을 조작함	데이터베이스의 값을 조작할 수 없음
SQL 버퍼에 하나 이상의 행이 입력됨	한 번에 한 행씩 입력되고 SQL 버퍼에 저장되지 않음
연속 문자가 없음	명령이 한 줄보다 긴 경우 연속 문자로 대시(-)를 사용
약어를 사용할 수 없음	약어를 사용할 수 있음
종료 문자를 사용하여 명령을 즉시 실행	종료 문자를 사용할 필요 없이 명령을 즉시 실행
함수를 사용하여 일부 형식 지정 수행	명령을 사용하여 데이터의 형식 지정

SQL*Plus 개요

- SQL*Plus에 로그인합니다.
- 테이블 구조를 설명합니다.
- SQL 문을 편집합니다.
- SQL*Plus에서 SQL을 실행합니다.
- SQL 문을 파일에 저장하고 첨부합니다.
- 저장된 파일을 실행합니다.
- 파일에서 버퍼로 명령을 로드하여 편집합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus

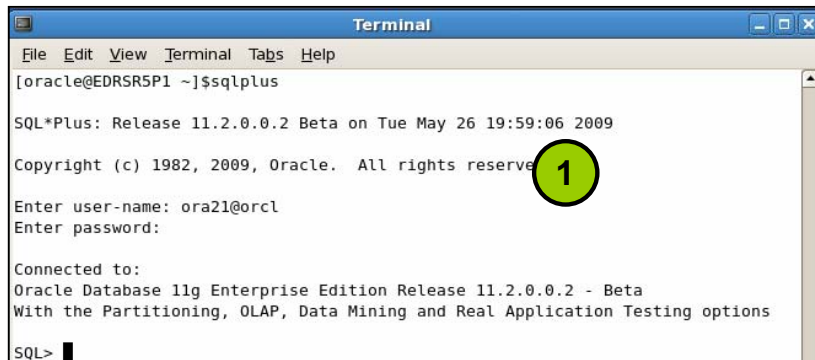
SQL*Plus는 다음 작업을 수행할 수 있는 환경입니다.

- 데이터베이스에서 데이터를 검색, 수정, 추가 및 제거하는 SQL 문을 실행합니다.
- query 결과의 형식을 지정하고 계산을 수행하고, 저장하고, 보고서 형식으로 인쇄합니다.
- 앞으로 반복 사용할 수 있도록 SQL 문을 저장하는 스크립트 파일을 생성합니다.

SQL*Plus 명령은 다음의 주요 범주로 나눌 수 있습니다.

범주	목적
환경	세션에 대한 SQL 문의 일반 동작에 영향을 줍니다.
형식	Query 결과의 형식을 지정합니다.
파일 조작	스크립트 파일을 저장, 로드, 실행합니다.
실행	SQL 문을 SQL 버퍼에서 Oracle 서버로 보냅니다.
편집	버퍼의 SQL 문을 수정합니다.
상호 작용	변수를 생성하여 SQL 문에 전달하고, 변수 값을 인쇄하고, 화면에 메시지를 출력합니다.
기타	데이터베이스에 연결하고, SQL*Plus 환경을 조작하며, 열 정의를 표시합니다.

SQL*Plus에 로그인



```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus

SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:59:06 2009

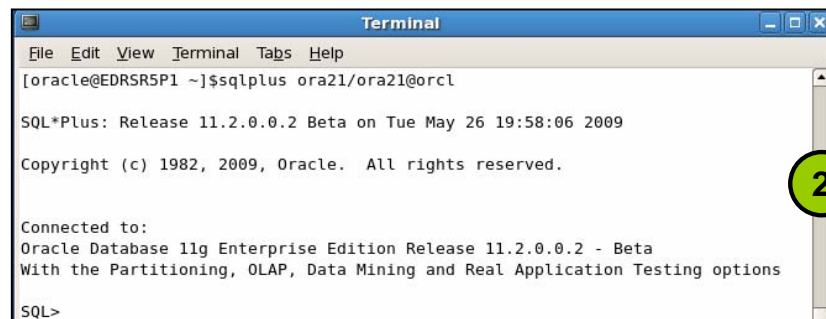
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Enter user-name: ora21@orcl
Enter password: 1

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>
```

sqlplus [username[/password[@database]]]



```
Terminal
File Edit View Terminal Tabs Help
[oracle@EDRSR5P1 ~]$sqlplus ora21/ora21@orcl

SQL*Plus: Release 11.2.0.0.2 Beta on Tue May 26 19:58:06 2009

Copyright (c) 1982, 2009, Oracle. All rights reserved.

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

SQL>
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL*Plus에 로그인

SQL*Plus를 호출하는 방법은 오라클 데이터베이스를 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[RIBE] tablename
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시

SQL*Plus에서 DESCRIBE 명령을 사용하여 테이블의 구조를 표시할 수 있습니다. 이 명령의 결과로 열 이름, 데이터 유형 및 열에 반드시 데이터가 포함되어야 하는지 여부가 표시됩니다. 이 구문에서 다음이 적용됩니다.

tablename 사용자가 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 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 스푼 파일을 닫습니다. OUT은 스푼 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 | UNL[IMITED]}]  
[FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | 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

- SIZE 는 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[ATE] | REP[LACE] |  
APP[END]] | OFF | OUT]
```

옵션	설명
file_name[.ext]	출력을 지정된 파일 이름으로 스푼합니다.
CRE[ATE]	지정한 이름으로 새 파일을 생성합니다.
REP[LACE]	기존 파일의 내용을 바꿉니다. 파일이 존재하지 않을 경우 REPLACE를 사용하면 파일이 생성됩니다.
APP[END]	지정한 파일의 끝에 버퍼의 내용을 추가합니다.
OFF	스푼을 중지합니다.
OUT	스푼 작업을 중지하고 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 스푼하려면 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]]  
[STAT[ISTICS]]
```

```
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

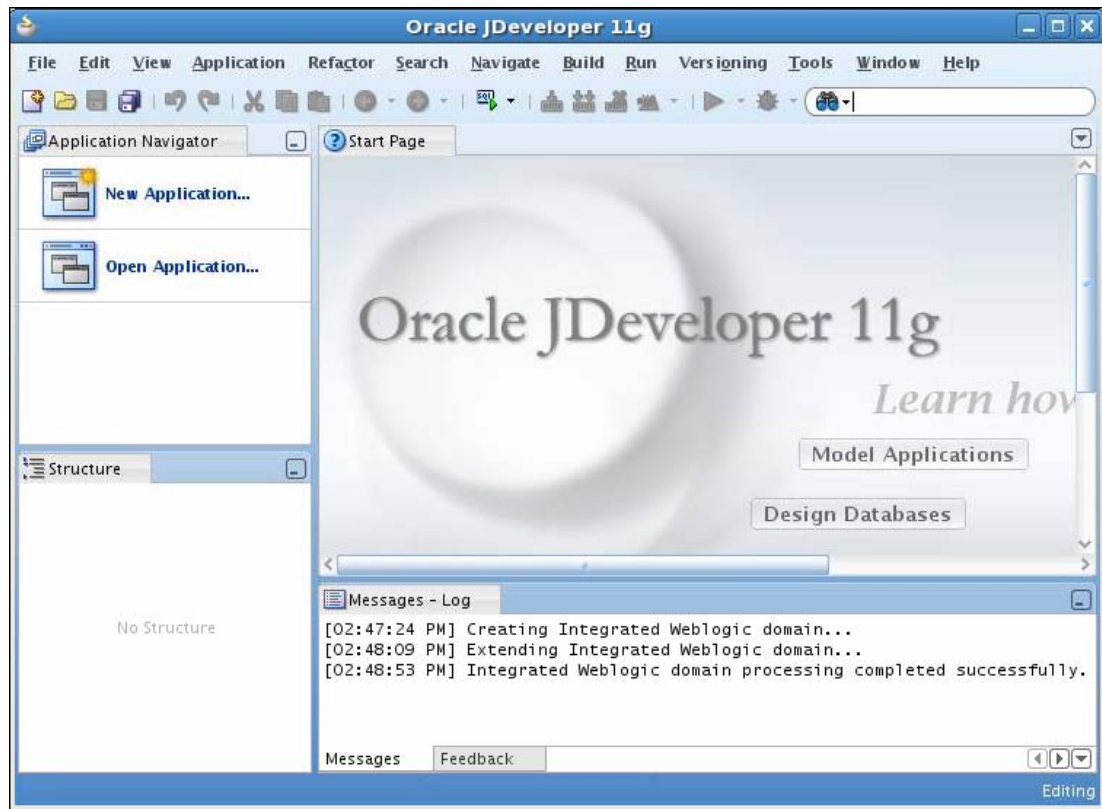
ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 JDeveloper 도구를 소개합니다. 먼저, 데이터베이스 개발 작업에 JDeveloper를 사용하는 방법에 대해 알아봅니다.

Oracle JDeveloper



ORACLE

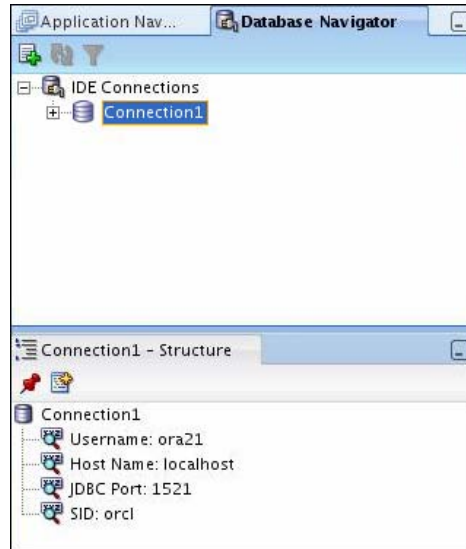
Copyright © 2009, Oracle. All rights reserved.

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 익스포트하고 임포트할 수 있습니다. 또한 데이터베이스를 찾고 응용 프로그램을 구축하는 것에서부터 배치에 이르기까지 다양한 용도로 사용됩니다.

연결 생성

1 Database Navigator에서 New Connection 아이콘을 누릅니다.

2 Create Database Connection window에서 username, 암호 및 SID를 입력합니다.

3 연결을 테스트합니다.

4 OK를 누릅니다.

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.

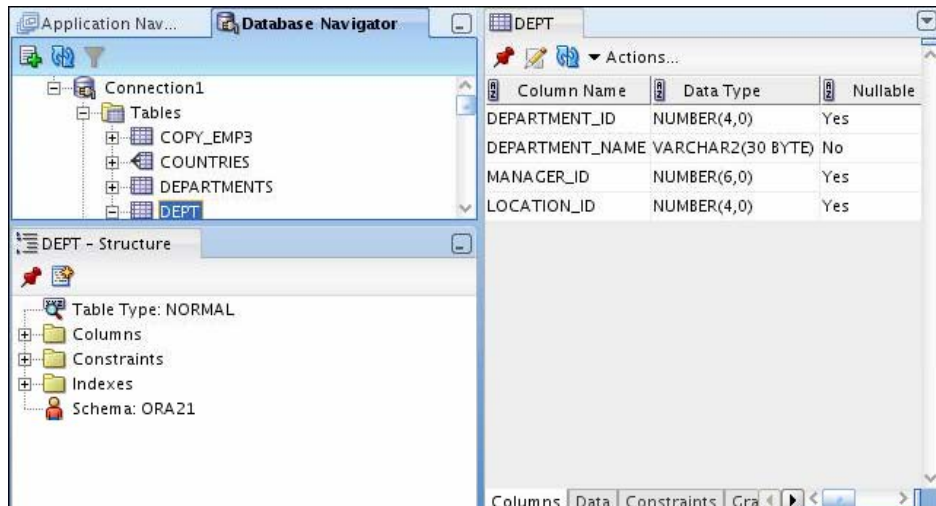
데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

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

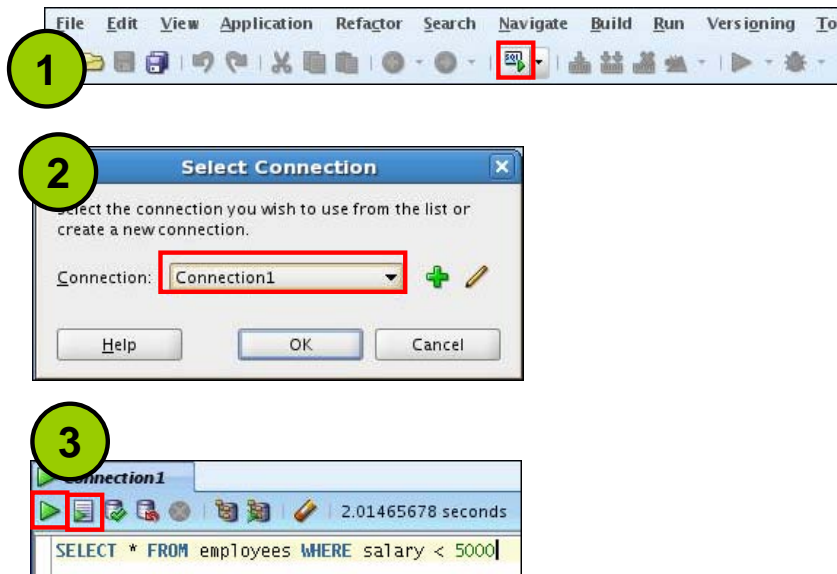
Copyright © 2009, Oracle. All rights reserved.

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 디렉터리에서 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.

The 'Results' tab displays the output of the SQL query. The data is as follows:

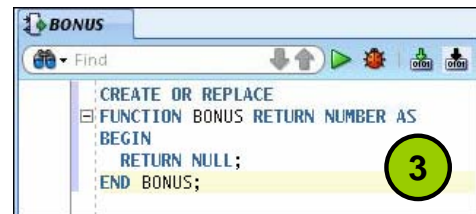
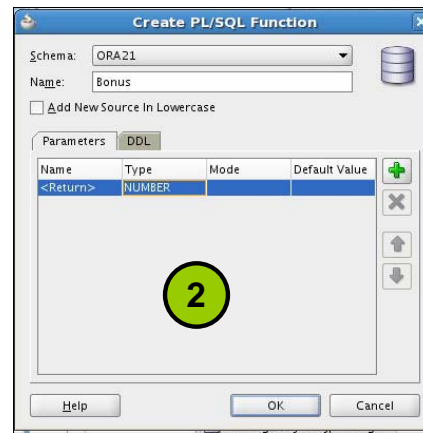
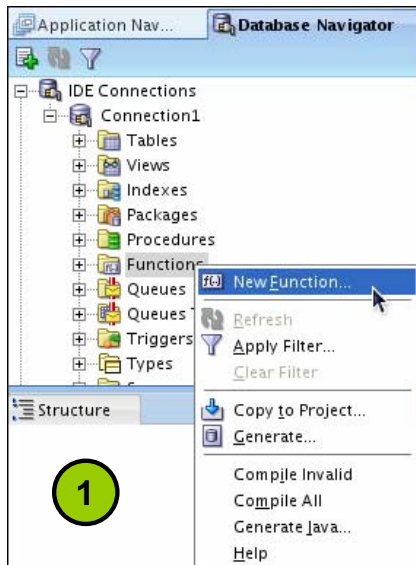
	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	
1	100	Steven	King	SK
2	101	Neena	Kochhar	NK

- **Run Script** 버튼 또는 F5 키를 누릅니다. 출력 결과는 다음과 같습니다.

The 'Script Output' tab displays the output of the SQL query. The data is as follows:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME
100	Steven	King

프로그램 단위 생성



함수의 기본 구조

프로그램 단위 생성

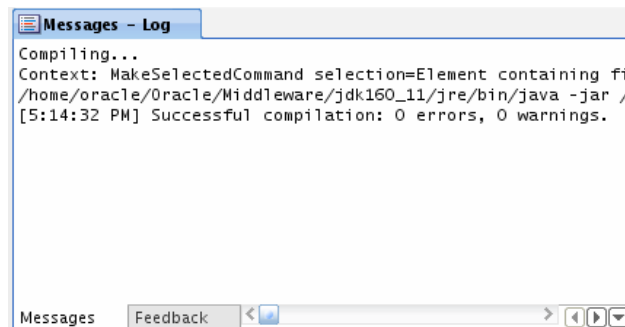
PL/SQL 프로그램 단위를 생성하려면 다음을 수행하십시오.

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 서버 프로그램을 편집할 수 있습니다.

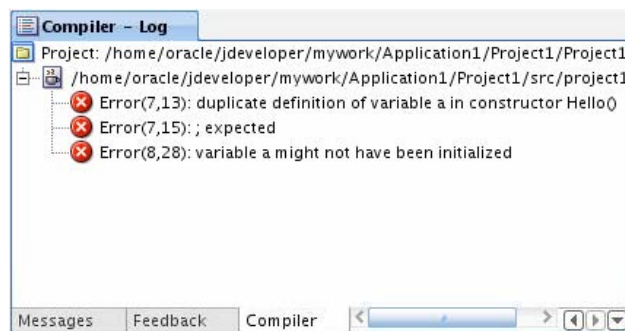
ORACLE

Copyright © 2009, Oracle. All rights reserved.

컴파일



오류가 있는 컴파일



오류가 없는 컴파일

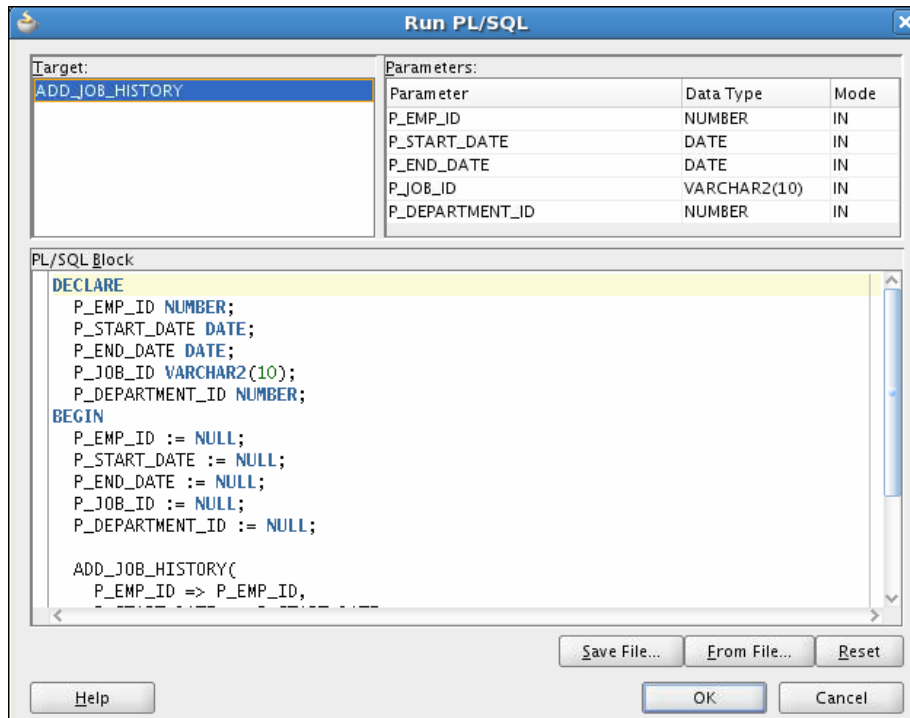
ORACLE

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



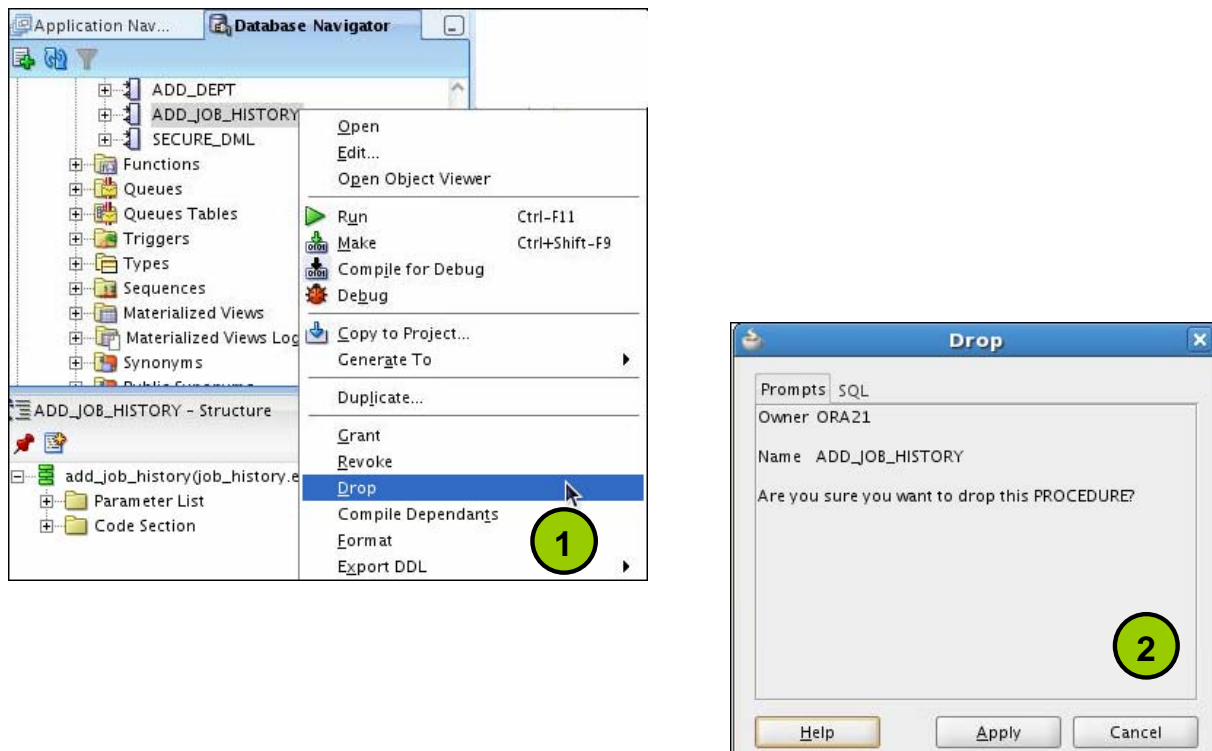
ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램 단위 실행

프로그램 단위를 실행하려면 객체를 마우스 오른쪽 버튼으로 누르고 **Run**을 선택합니다. Run PL/SQL 대화상자가 나타납니다. NULL 값을 프로그램 단위로 전달하기에 적절한 값으로 변경해야 할 수도 있습니다. 해당 값을 변경한 후 **OK**를 누릅니다. Message-Log window에 출력 결과가 표시됩니다.

프로그램 단위 삭제



프로그램 단위 삭제

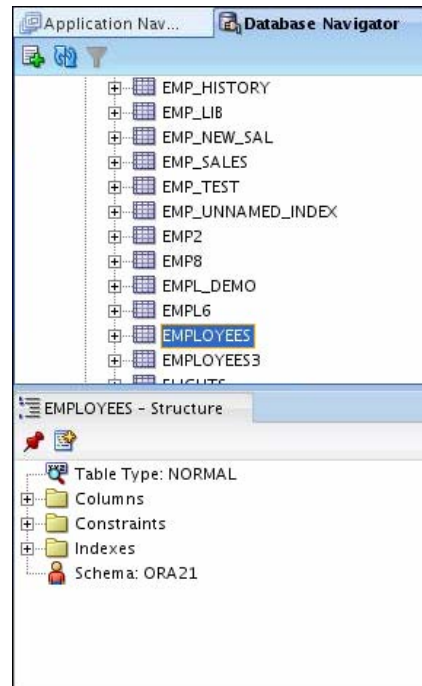
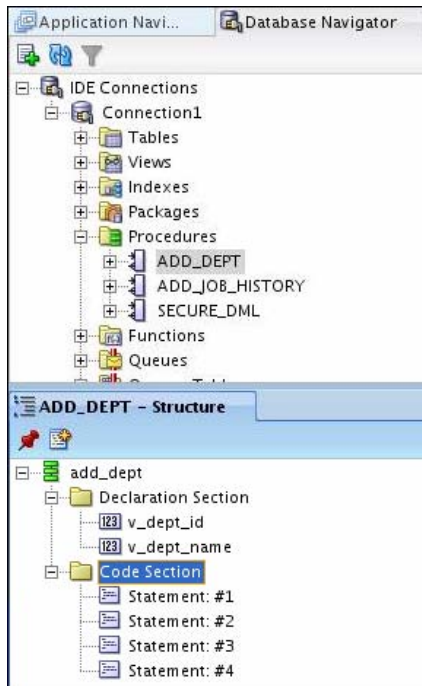
프로그램 단위를 삭제하려면 다음을 수행하십시오.

1. 객체를 마우스 오른쪽 버튼으로 누르고 Drop을 선택합니다.
Drop Confirmation 대화상자가 나타납니다.
2. Apply를 누릅니다.
객체가 데이터베이스에서 삭제됩니다.

Copyright © 2009, Oracle. All rights reserved.

ORACLE

Structure Window



ORACLE

Copyright © 2009, Oracle. All rights reserved.

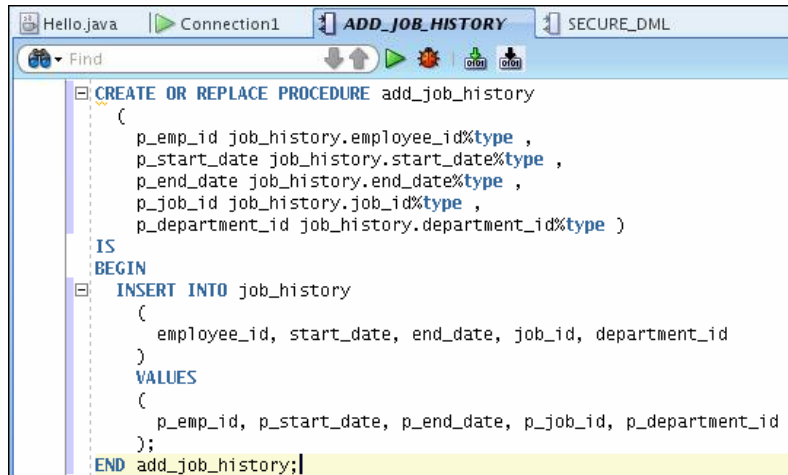
Structure Window

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



```
CREATE OR REPLACE PROCEDURE add_job_history
(
  p_emp_id job_history.employee_id%type ,
  p_start_date job_history.start_date%type ,
  p_end_date job_history.end_date%type ,
  p_job_id job_history.job_id%type ,
  p_department_id job_history.department_id%type )
IS
BEGIN
  INSERT INTO job_history
  (
    employee_id, start_date, end_date, job_id, department_id
  )
  VALUES
  (
    p_emp_id, p_start_date, p_end_date, p_job_id, p_department_id
  );
END add_job_history;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

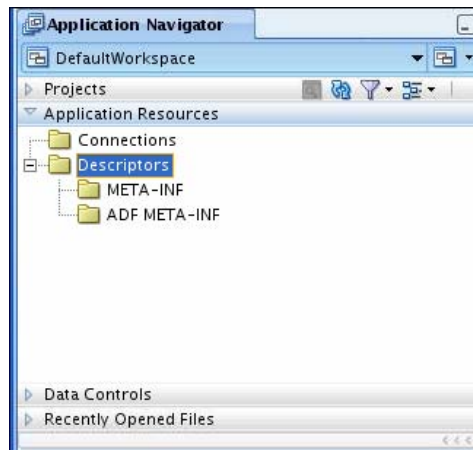
Editor Window

단일 Editor window에서 프로젝트 파일을 모두 보거나, 한 파일을 여러 뷰로 열거나, 다양한 파일을 여러 뷰로 열 수 있습니다.

편집기 window의 상단에 있는 탭은 문서 탭입니다. 문서 탭을 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.

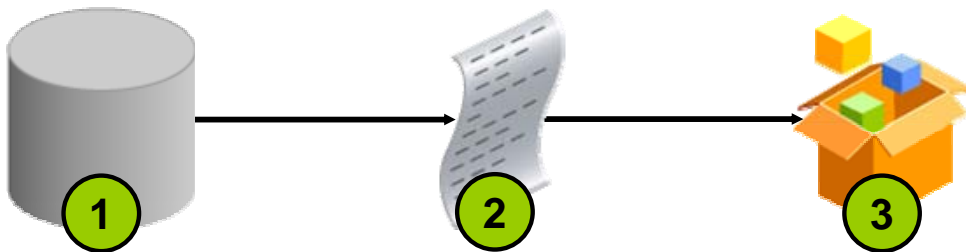
Application Navigator

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 유틸리티를 사용합니다.

- `loadjava`는 내장 프로시저를 포함하는 Java 클래스를 오라클 데이터베이스에 로드합니다.
- `publish`는 로드된 공용(public) 정적 메소드에 대해 PL/SQL 호출 특정 래퍼를 생성합니다. 게시(publishing) 후에는 Java 메소드를 PL/SQL 함수 또는 프로시저로 호출할 수 있습니다.

PL/SQL에 Java 게시(Publishing)



```
public class TrimLob
{
    public static void main (String args []) throws SQLException {
        Connection conn=null;
        if (System.getProperty("oracle.jserver.version") != null)
        {
            conn = DriverManager.getConnection("jdbc:default:connection:");
        }
        else
        {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:thin:scott/tiger");
        }
    }
}
```



```
CREATE OR REPLACE PROCEDURE TRIMLOBPROC
as language java
name 'TrimLob.main(java.lang.String[])';
/
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

PL/SQL에 Java 게시(Publishing)

위 슬라이드는 Java 코드 및 이 Java 코드를 PL/SQL 프로시저에 게시(publish)하는 방법을 보여줍니다.

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 자습서	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.

Oracle Internal & Oracle Academy Use Only

요약

이 부록에서는 JDeveloper를 사용하여 다음 작업을 수행하는 방법을 배웁니다.

- 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.

관련 데이터를 그룹화하여 보고서 생성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- **ROLLUP 연산을 사용하여 소계 값 생성**
- **CUBE 연산을 사용하여 교차 분석 값 생성**
- **GROUPING 함수를 사용하여 ROLLUP 또는 CUBE에 의해 생성된 행 값 식별**
- **GROUPING SETS를 사용하여 단일 결과 집합 생성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 다음 작업을 수행하는 방법을 설명합니다.

- ROLLUP 연산자로 데이터를 그룹화하여 소계 값 구하기
- CUBE 연산자로 데이터를 그룹화하여 교차 분석 값 구하기
- GROUPING 함수를 사용하여 ROLLUP 또는 CUBE 연산자에 의해 생성된 결과 집합의 집계 레벨 식별
- GROUPING SETS를 사용하여 UNION ALL 방식과 같은 단일 결과 집합 생성

그룹 함수 검토

- 그룹 함수는 행 집합 연산을 수행하여 그룹별로 하나의 결과를 산출합니다.

```
SELECT      [column,] group_function(column). . .
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

- 예제:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct), MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

그룹 함수

GROUP BY 절을 사용하여 테이블의 행을 그룹으로 나눌 수 있습니다. 그런 다음 그룹 함수를 사용하여 각 그룹에 대한 요약 정보를 반환할 수 있습니다. 그룹 함수는 SELECT 리스트와 ORDER BY 및 HAVING 절에 나타날 수 있습니다. Oracle 서버는 각 행 그룹에 그룹 함수를 적용하고 각 그룹에 대해 단일 결과 행을 반환합니다.

그룹 함수 유형: 그룹 함수 AVG, SUM, MAX, MIN, COUNT, STDDEV 및 VARIANCE는 각각 하나의 인수만 허용합니다. AVG, SUM, STDDEV 및 VARIANCE 함수는 숫자 값에 대해서만 실행됩니다. MAX 및 MIN은 숫자, 문자 또는 날짜 데이터 값에 대해 실행될 수 있습니다. COUNT는 제공된 표현식에 대해 널이 아닌 행 수를 반환합니다. 슬라이드의 예제는 JOB_ID가 SA로 시작하는 사원의 평균 급여, 급여의 표준 편차, 커미션을 받는 사원 수 및 최대 채용 날짜를 계산합니다.

그룹 함수 사용 지침

- 인수에 대한 데이터 유형은 CHAR, VARCHAR2, NUMBER 또는 DATE일 수 있습니다.
- COUNT(*)를 제외한 모든 그룹 함수는 널 값을 무시합니다. 널 값을 다른 값으로 치환하려면 NVL 함수를 사용합니다. COUNT는 숫자 또는 0을 반환합니다.
- Oracle 서버는 GROUP BY 절이 사용될 때 암시적으로 지정된 그룹화 열의 결과 집합을 오름차순으로 정렬합니다. 이 기본 정렬 방식 대신 내림차순으로 정렬하려면 ORDER BY 절에 DESC를 사용하면 됩니다.

GROUP BY 절 검토

- 구문:

```
SELECT      [column,]group_function(column). . .  
FROM        table  
[WHERE      condition]  
[GROUP BY   group_by_expression]  
[ORDER BY   column];
```

- 예제:

```
SELECT  department_id, job_id, SUM(salary),  
        COUNT(employee_id)  
FROM    employees  
GROUP BY department_id, job_id ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUP BY 절 검토

슬라이드에 표시된 예제는 Oracle 서버에 의해 다음과 같이 평가됩니다.

- SELECT 절은 다음 열을 검색하도록 지정합니다.
 - EMPLOYEES 테이블의 부서 ID 열과 직무 ID 열
 - GROUP BY 절에 지정된 각 그룹의 모든 급여 합계 및 사원 수
- GROUP BY 절은 테이블에서 행이 그룹화되는 방법을 지정합니다. 사원 수 및 급여 합계는 각 부서에서 직무 ID별로 계산됩니다. 행은 먼저 부서 ID로 그룹화된 다음 각 부서 내의 직무별로 그룹화됩니다.

HAVING 절 검토

- HAVING 절을 사용하여 표시할 그룹을 지정합니다.
- 제한 조건을 기준으로 추가로 그룹을 제한합니다.

```
SELECT      [column,] group_function(column)...  
FROM        table  
[WHERE      condition]  
[GROUP BY   group_by_expression]  
[HAVING     having_expression]  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

HAVING 절

그룹이 형성되고 그룹 함수가 계산된 다음 HAVING 절이 그룹에 적용됩니다. HAVING 절이 GROUP BY 절 앞에 올 수 있지만 GROUP BY 절을 먼저 두는 것이 더 논리적이므로 이렇게 하는 것이 좋습니다.

Oracle 서버는 사용자가 HAVING 절을 사용할 때 다음 단계를 수행합니다.

1. 행을 그룹화합니다.
2. 그룹에 그룹 함수를 적용하고 HAVING 절의 조건과 일치하는 그룹을 표시합니다.

GROUP BY에 ROLLUP 및 CUBE 연산자 사용

- GROUP BY에 ROLLUP 또는 CUBE를 사용하여 상호 참조 열별로 대집계 행을 생성합니다.
- ROLLUP 그룹화는 일반 그룹화 행과 소계 값을 포함한 결과 집합을 생성합니다.
- CUBE 그룹화는 ROLLUP에 따른 행과 교차 분석 행이 포함된 결과 집합을 생성합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUP BY에 ROLLUP 및 CUBE 연산자 사용

Query의 GROUP BY 절에 ROLLUP 및 CUBE 연산자를 지정하십시오. ROLLUP 그룹화는 일반 그룹화 행과 소계 값을 포함한 결과 집합을 생성합니다. ROLLUP 연산자는 총계도 계산합니다. GROUP BY 절의 CUBE 연산은 사양에서 가능한 모든 표현식의 조합 값에 준하여 선택한 행을 그룹화하고 각 그룹에 대한 요약 정보를 단일 행으로 반환합니다. CUBE 연산자를 사용하여 교차 분석 행을 생성할 수 있습니다.

참고: ROLLUP 및 CUBE를 사용할 경우에는 GROUP BY 절 다음의 열이 서로 의미 있는 실제 관계에 있어야 합니다. 그렇지 않은 경우 이 연산자는 부적절한 정보를 반환합니다.

ROLLUP 연산자

- ROLLUP은 GROUP BY 절의 확장입니다.
- ROLLUP 연산을 사용하여 소계와 같은 누적 집계를 생성합니다.

```
SELECT      [column,] group_function(column). . .  
FROM        table  
[WHERE      condition]  
[GROUP BY   [ROLLUP] group_by_expression]  
[HAVING     having_expression];  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ROLLUP 연산자

ROLLUP 연산자는 GROUP BY 문 내의 표현식에 대해 집계 및 대집계를 전달합니다. 보고서 작성자는 ROLLUP 연산자를 사용하여 결과 집합에서 통계 및 요약 정보를 추출할 수 있습니다. 누적 집계는 보고서, 차트 및 그래프에 사용할 수 있습니다.

ROLLUP 연산자는 GROUP BY 절에 지정된 열 리스트를 한 방향으로(오른쪽에서 왼쪽으로) 이동하여 그룹을 생성합니다. 그런 다음 이 그룹화에 집계 함수를 적용합니다.

참고

- ROLLUP 연산자 없이 n 차원(즉, GROUP BY 절에 있는 n 개의 열)에서 소계를 생성하려면 $n+1$ 개의 SELECT 문을 UNION ALL과 연결해야 합니다. 이렇게 하면 SELECT 문마다 테이블에 액세스하게 되므로 query가 비효율적으로 실행됩니다. ROLLUP 연산자는 테이블에 한 번만 액세스하여 해당 결과를 수집합니다. 소계 생성에 관련된 열이 많은 경우에는 ROLLUP 연산자가 유용합니다.
- 소계와 총계는 ROLLUP을 통해 생성됩니다. CUBE 역시 총계를 생성하며 가능한 각 방향으로 효과적으로 롤업하여 교차 분석 데이터를 생성합니다.

ROLLUP 연산자: 예제

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	10	(null)	4400
3	20	MK_MAN	13000
4	20	MK_REP	6000
5	20	(null)	19000
6	30	PU_MAN	11000
7	30	PU_CLERK	13900
8	30	(null)	24900
9	40	HR_REP	6500
10	40	(null)	6500
11	50	ST_MAN	36400
12	50	SH_CLERK	64300
13	50	ST_CLERK	55700
14	50	(null)	156400
15	(null)	(null)	211200

1

2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ROLLUP 연산자 예제

슬라이드의 예제는 다음과 같습니다.

- GROUP BY 절은 부서 ID가 60보다 작은 부서에 대해 부서 내의 모든 직무 ID에 대한 총 급여를 표시합니다.
- ROLLUP 연산자는 다음을 표시합니다.
 - 부서 ID가 60보다 작은 각 부서의 총 급여
 - 직무 ID와 관계없이 부서 ID가 60보다 작은 모든 부서의 총 급여

이 예제에서 1은 DEPARTMENT_ID와 JOB_ID로 집계된 그룹을 나타내고, 2는 DEPARTMENT_ID로만 집계된 그룹을 나타내며, 3은 총계를 나타냅니다.

ROLLUP 연산자는 GROUP BY 절에 지정된 그룹화 리스트를 따라 세부 레벨에서 전체 총계까지 롤업하는 소계를 생성합니다. 먼저 GROUP BY 절에 지정된 그룹에 대한 표준 집계 값을 계산합니다(예제에서는 부서 내의 각 직무로 그룹화된 급여 합계). 그런 다음 그룹 열 리스트에 따라 오른쪽에서 왼쪽으로 이동하여 점차 높은 레벨의 소계를 생성합니다. 예제에서는 각 부서의 급여 합계가 계산되고 그 다음에 모든 부서의 급여 합계가 계산됩니다.

- GROUP BY 절의 ROLLUP 연산자에서 n 표현식을 사용할 경우 연산 결과는 $n+1$ (이 경우 $2+1=3$)개의 그룹이 됩니다.
- 첫번째 n 표현식의 값을 기반으로 하는 행을 행 또는 일반 행이라고 하며 나머지 행을 대집계 행이라고 합니다.

CUBE 연산자

- CUBE는 GROUP BY 절의 확장입니다.
- CUBE 연산자를 사용하여 단일 SELECT 문으로 교차 분석 값을 생성할 수 있습니다.

```
SELECT      [column,] group_function(column)...  
FROM        table  
[WHERE      condition]  
[GROUP BY   [CUBE] group_by_expression]  
[HAVING     having_expression]  
[ORDER BY   column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CUBE 연산자

CUBE 연산자는 SELECT 문의 GROUP BY 절에 있는 추가 스위치입니다. CUBE 연산자는 AVG, SUM, MAX, MIN 및 COUNT를 포함하여 모든 집계 함수에 적용할 수 있습니다. CUBE 연산자는 일반적으로 교차 분석 보고서에 사용되는 결과 집합을 생성하는 데 사용됩니다. ROLLUP이 가능한 소계 조합의 일부만 생성하는 반면, CUBE는 GROUP BY 절에 지정된 가능한 모든 그룹화 조합의 소계와 총계를 생성합니다.

CUBE 연산자는 집계 함수와 함께 사용되어 결과 집합에 추가 행을 생성합니다. GROUP BY 절에 포함된 열은 상호 참조되어 대집합 그룹을 생성합니다. 선택 리스트에 지정된 집계 함수는 이러한 그룹에 적용되어 추가 대집계 행의 요약 값을 생성합니다. 결과 집합의 추가 그룹 수는 GROUP BY 절에 포함된 열 수에 의해 결정됩니다.

실제로 GROUP BY 절에서 가능한 열이나 표현식의 모든 조합을 사용하여 대집계를 생성할 수 있습니다. GROUP BY 절에 n 열이나 표현식이 있을 경우 2^n 개의 가능한 대집계 조합이 있습니다. 수학적으로 이러한 조합은 n 차원 큐브를 형성하며 연산자는 이러한 방법으로 해당 이름을 얻습니다.

응용 프로그램이나 프로그래밍 도구를 사용하여 이러한 대집계 값을 차트와 그래프에 제공하여 결과와 관계를 시각적이고 효율적으로 전달할 수 있습니다.

CUBE 연산자: 예제

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id < 60
GROUP BY CUBE (department_id, job_id);
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
1	(null)	(null)	211200	1
2	(null)	HR_REP	6500	
3	(null)	MK_MAN	13000	
4	(null)	MK_REP	6000	
5	(null)	PU_MAN	11000	
6	(null)	ST_MAN	36400	2
7	(null)	AD_ASST	4400	
8	(null)	PU_CLERK	13900	
9	(null)	SH_CLERK	64300	
10	(null)	ST_CLERK	55700	
11	10	(null)	4400	3
12	10	AD_ASST	4400	
13	20	(null)	19000	
14	20	MK_MAN	13000	
15	20	MK_REP	6000	
16	30	(null)	24900	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CUBE 연산자 예제

이 예제에서 SELECT 문의 출력은 다음과 같이 해석될 수 있습니다.

- 부서 내의 모든 직무에 대한 총 급여(부서 ID가 60보다 작은 부서의 경우)
- 부서 ID가 60보다 작은 각 부서의 총 급여
- 부서와 관계없이 각 직무에 대한 총 급여
- 직책과 관계없이 부서 ID가 60보다 작은 부서의 총 급여

이 예제에서 1은 총계를 나타내고, 2는 JOB_ID만으로 집계된 행을 나타내며, 3은 DEPARTMENT_ID와 JOB_ID로 집계된 일부 행을 나타내고, 4는 DEPARTMENT_ID만으로 집계된 일부 행을 나타냅니다.

CUBE 연산자도 ROLLUP 연산을 수행하여 직위와 관계없이 부서 ID가 60보다 작은 부서의 소계와 부서 ID가 60보다 작은 부서의 총 급여를 표시합니다. 또한 CUBE 연산자는 부서와 관계없이 모든 직무에 대한 총 급여를 표시합니다.

참고: ROLLUP 연산자와 마찬가지로 CUBE 연산자 없이 n 차원(즉, GROUP BY 절에 있는 n 개의 열)에서 소계를 생성하려면 2^n 개의 SELECT 문을 UNION ALL과 연결해야 합니다. 따라서 3차원의 보고서를 생성하기 위해서는 $2^3 = 8$, 즉 8개의 SELECT 문을 UNION ALL로 연결해야 합니다.

GROUPING 함수

GROUPING 함수:

- CUBE 또는 ROLLUP 연산자와 함께 사용됩니다.
- 행에서 소계를 형성하는 그룹을 찾는 데 사용됩니다.
- ROLLUP 또는 CUBE로 생성된 NULL 값과 저장된 NULL 값을 구분하는 데 사용됩니다.
- 0 또는 1을 반환합니다.

```
SELECT      [column,] group_function(column) .. ,  
            GROUPING(expr)  
FROM        table  
[WHERE      condition]  
[GROUP BY [ROLLUP][CUBE] group_by_expression]  
[HAVING     having_expression]  
[ORDER BY  column];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING 함수

GROUPING 함수를 CUBE 또는 ROLLUP 연산자와 함께 사용하면 요약 값을 구하는 방식을 이해하는 데 도움이 됩니다.

GROUPING 함수는 단일 열을 인수로 사용합니다. GROUPING 함수의 *expr*은 GROUP BY 절의 표현식 중 하나와 일치해야 합니다. 이 함수는 0 또는 값 1을 반환합니다.

GROUPING 함수에서 반환된 값은 다음 용도로 사용됩니다.

- 제공된 소계의 집계 레벨, 즉 소계의 기반이 되는 그룹을 결정합니다.
- 결과 집합의 행에 대한 표현식 열에 있는 NULL 값이 다음을 나타내는지 확인합니다.
 - 기본 테이블의 NULL 값(저장된 NULL 값)
 - ROLLUP 또는 CUBE로 생성한 NULL 값(해당 표현식의 그룹 함수의 결과)

표현식에 준하여 GROUPING 함수에 의해 반환된 값 0은 다음 중 하나를 나타냅니다.

- 집계 값을 계산하는 데 표현식이 사용되었습니다.
- 표현식 열의 NULL 값은 저장된 NULL 값입니다.

표현식에 준하여 GROUPING 함수에 의해 반환된 값 1은 다음 중 하나를 나타냅니다.

- 집계 값을 계산하는 데 표현식이 사용되지 않았습니다.
- 표현식 열의 NULL 값은 ROLLUP 또는 CUBE의 그룹화 결과 생성되었습니다.

GROUPING 함수: 예제

```
SELECT    department_id DEPTID, job_id JOB,
          SUM(salary),
          GROUPING(department_id) GRP_DEPT,
          GROUPING(job_id) GRP_JOB
FROM      employees
WHERE     department_id < 50
GROUP BY  ROLLUP(department_id, job_id);
```

	DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
1	10	AD_ASST	4400	0	0
2	10	(null)	4400	0	1
3	20	MK_MAN	13000	0	0
4	20	MK_REP	6000	0	0
5	20	(null)	19000	0	1
6	30	PU_MAN	11000	0	0
7	30	PU_CLERK	13900	0	0
8	30	(null)	24900	0	1
9	40	HR_REP	6500	0	0
10	40	(null)	6500	0	1
11	(null)	(null)	54800	1	1

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING 함수 예제

슬라이드 예제에서 첫번째 행의 요약 값 4400(레이블 1)을 살펴 보십시오. 이 요약 값은 부서 10 내의 직무 ID AD_ASST에 대한 총 급여입니다. 이 요약 값을 계산하려면 DEPARTMENT_ID와 JOB_ID 열을 모두 고려해야 합니다. 따라서 GROUPING(department_id) 및 GROUPING(job_id) 표현식 모두에 대해 0 값이 반환됩니다.

두번째 행의 요약 값 4400을 살펴 보십시오(레이블 2). 이 값은 부서 10의 총 급여이며 DEPARTMENT_ID 열을 고려하여 계산되었습니다. 따라서 GROUPING(department_id)에 의해 0 값이 반환되었습니다. 이 값을 계산할 때 JOB_ID 열은 고려되지 않았기 때문에 GROUPING(job_id)에 대해 값 1이 반환되었습니다. 다섯번째 행에서 비슷한 결과를 볼 수 있습니다.

마지막 행에서는 요약 값 54800(레이블 3)을 살펴 보십시오. 이 값은 부서 ID가 50 보다 작은 부서 및 모든 직무에 대한 총 급여입니다. 이 요약 값을 계산하는 데 DEPARTMENT_ID와 JOB_ID 열을 모두 고려하지 않았습니다. 따라서 GROUPING(department_id) 및 GROUPING(job_id) 표현식 모두에 대해 값 1이 반환됩니다.

GROUPING SETS

- GROUPING SETS 구문은 동일한 query에서 여러 개의 그룹화를 정의하는 데 사용됩니다.
- GROUPING SETS 절에 지정된 모든 그룹화를 계산하며 개별 그룹화 결과를 UNION ALL 연산과 결합합니다.
- 그룹화 집합의 효율성:
 - 기본 테이블에 대해 하나의 패스만 필요합니다.
 - 복잡한 UNION 문을 작성하지 않아도 됩니다.
 - GROUPING SETS 요소가 많을수록 성능이 향상됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING SETS

GROUPING SETS는 GROUP BY 절이 추가로 확장된 것으로 데이터에 대해 여러 그룹화를 지정하는 데 사용할 수 있습니다. 이렇게 하면 효율적인 집계가 이루어지고 그에 따라 쉽게 여러 차원(Dimension)의 데이터를 분석할 수 있습니다.

이제 UNION ALL 연산자로 여러 SELECT 문을 결합하지 않고 GROUPING SETS를 사용하여 여러 그룹화(ROLLUP 또는 CUBE 연산자 포함 가능)를 지정하도록 단일 SELECT 문을 작성할 수 있습니다. 예를 들면 다음과 같습니다.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY
GROUPING SETS
((department_id, job_id, manager_id),
(department_id, manager_id),(job_id, manager_id));
```

이 명령문은 다음 세 그룹에 대한 집계를 계산합니다.

```
(department_id, job_id, manager_id), (department_id,
manager_id)and (job_id, manager_id)
```

이러한 기능이 없는 경우 위와 같은 SELECT 문의 출력 결과를 얻기 위해서는 여러 query를 UNION ALL과 결합해야 합니다. 다중 query 접근 방법은 동일한 데이터를 여러 번 스캔해야 하므로 비효율적입니다.

GROUPING SETS(계속)

앞의 예제와 다음 방법을 비교해 보십시오.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

이 명령문은 (department_id, job_id, manager_id), (department_id, manager_id) 및 (job_id, manager_id) 그룹만 필요한 경우에도 8개(2*2*2)의 그룹화를 모두 계산합니다.

다음은 또 다른 대체 명령문입니다.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, job_id, manager_id
UNION ALL
SELECT department_id, NULL, manager_id, AVG(salary)
FROM employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY job_id, manager_id;
```

이 명령문은 기본 테이블을 세 번 스캔해야 하므로 비효율적입니다.

CUBE 및 ROLLUP은 매우 특수한 의미와 결과를 지닌 그룹화 집합으로 간주할 수 있습니다. 다음은 이러한 사실을 보여줍니다.

CUBE(a, b, c) 는 다음과 동일합니다.	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())
ROLLUP(a, b, c) 는 다음과 동일합니다.	GROUPING SETS ((a, b, c), (a, b), (a), ())

GROUPING SETS: 예제

```
SELECT    department_id, job_id,
          manager_id, AVG(salary)
FROM      employees
GROUP BY  GROUPING SETS
          ((department_id, job_id), (job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
1	(null)	SH_CLERK	122	3200
2	(null)	AC_MGR	101	12000
3	(null)	ST_MAN	100	7280
4	...	(null)	ST_CLERK	2675

1

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
39	110	AC_MGR	(null)	12000
40	90	AD_PRES	(null)	24000
41	60	IT_PROG	(null)	5760
42	100	FL_MGR	(null)	12000

2

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

GROUPING SETS: 예제

슬라이드의 query는 두 그룹에 대한 집계를 계산합니다. 테이블은 다음의 그룹으로 나뉩니다.

- 부서 ID, 직무 ID
- 직무 ID, 관리자 ID

이러한 각 그룹의 평균 급여가 계산됩니다. 결과 집합에는 두 그룹 각각에 대한 평균 급여가 표시됩니다.

출력에서 1로 표시된 그룹은 다음과 같이 해석될 수 있습니다.

- 관리자 122 휘하의 직무 ID가 SH_CLERK인 모든 사원의 평균 급여는 3,200입니다.
 - 관리자 101 휘하의 직무 ID가 AC_MGR인 모든 사원의 평균 급여는 12,000입니다.
- 나머지 행도 이러한 방식으로 해석됩니다.

출력에서 2로 표시된 그룹은 다음과 같이 해석됩니다.

- 부서 110에서 직무 ID가 AC_MGR인 모든 사원의 평균 급여는 12,000입니다.
 - 부서 90에서 직무 ID가 AD_PRES인 모든 사원의 평균 급여는 24,000입니다.
- 나머지 행도 이러한 방식으로 해석됩니다.

GROUPING SETS: 예제(계속)

슬라이드의 예제를 다음과 같이 작성할 수도 있습니다.

```
SELECT department_id, job_id, NULL as manager_id,  
       AVG(salary) as AVGSAL  
FROM employees  
GROUP BY department_id, job_id  
UNION ALL  
SELECT NULL, job_id, manager_id, avg(salary) as AVGSAL  
FROM employees  
GROUP BY job_id, manager_id;
```

query 블록을 찾아서 실행 계획을 생성하는 옵티마이저가 없을 경우 이전 query는 기본 테이블 EMPLOYEES를 두 번 스캔해야 합니다. 이는 매우 비효율적일 수 있습니다. 따라서 GROUPING SETS 문을 사용하는 것이 좋습니다.

조합 열

- 조합 열은 한 단위로 처리되는 열의 모음입니다.

`ROLLUP (a, (b, c), d)`

- GROUP BY 절에서 괄호를 사용하여 열을 그룹화합니다. 이렇게 하면 ROLLUP 또는 CUBE 연산을 계산할 때 괄호로 묶인 열이 하나의 단위로 처리됩니다.
- 조합 열을 ROLLUP 또는 CUBE와 함께 사용할 경우 특정 레벨에서 집계를 생략해야 합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조합 열

조합 열은 그룹화 계산 중에 하나의 단위로 처리되는 열 모음입니다. `ROLLUP (a, (b, c), d)` 와 같이 열을 괄호 안에 지정하십시오.

여기서 (b, c)는 조합 열을 형성하고 하나의 단위로 처리됩니다. 일반적으로 조합 열은 ROLLUP, CUBE 및 GROUPING SETS에서 유용합니다. 예를 들어, CUBE 또는 ROLLUP에서 조합 열을 사용할 경우 특정 레벨에서 집계를 생략해야 합니다.

즉, `GROUP BY ROLLUP(a, (b, c))`는 다음과 같습니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ( )
```

여기서 (b, c)는 하나의 단위로 처리되며 ROLLUP은 (b, c)에 적용되지 않습니다. 이것은 마치 (b, c)에 alias(예: z)가 있는 것과 같으며 GROUP BY 표현식은 `GROUP BY ROLLUP(a, z)`로 줄어듭니다.

참고: `GROUP BY ()`는 일반적으로 a 및 b열에 대한 NULL 값과 집계 함수로만 구성된 SELECT 문입니다. 일반적으로 총계를 생성하는 데 사용됩니다.

```
SELECT NULL, NULL, aggregate_col
FROM <table_name>
GROUP BY ( );
```

조합 열(계속)

이 명령문을 다음과 같은 일반 ROLLUP과 비교해 보십시오.

```
GROUP BY ROLLUP(a, b, c)
```

이것은 다음을 의미합니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY a UNION ALL
GROUP BY ()
```

마찬가지로

```
GROUP BY CUBE((a, b), c)
```

이것은 다음과 동일합니다.

```
GROUP BY a, b, c UNION ALL
GROUP BY a, b UNION ALL
GROUP BY c UNION ALL
GROUP BY ()
```

다음 표는 GROUPING SETS 사양 및 해당 GROUP BY 사양을 보여줍니다.

GROUPING SETS 문	동일한 GROUP BY 문
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c)) (GROUPING SETS 식은 조합 열을 갖습니다.)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()
GROUP BY GROUPING SETS (a, ROLLUP(b, c)) (GROUPING SETS 식은 조합 열을 갖습니다.)	GROUP BY a UNION ALL GROUP BY ROLLUP(b, c)

조합 열: 예제

```
SELECT    department_id, job_id, manager_id,
          SUM(salary)
FROM      employees
GROUP BY ROLLUP( department_id, (job_id, manager_id));
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	(null)	(null)	(null)	7000
3	10	AD_ASST	101	4400
4	10	(null)	(null)	4400
5	20	MK_MAN	100	13000
6	20	MK_REP	201	6000
7	20	(null)	(null)	19000
...				
40	100	FI_MGR	101	12000
41	100	FI_ACCOUNT	108	39600
42	100	(null)	(null)	51600
43	110	AC_MGR	101	12000
44	110	AC_ACCOUNT	205	8300
45	110	(null)	(null)	20300
46	(null)	(null)	(null)	691400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조합 열: 예제

다음 예제를 살펴 보십시오.

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
GROUP BY ROLLUP( department_id, job_id, manager_id);
```

이 query에서 Oracle 서버는 다음 그룹화를 계산합니다.

- (job_id, manager_id)
- (department_id, job_id, manager_id)
- (department_id)
- 총계

특정 그룹에만 관심이 있는 경우 조합 열을 사용하지 않으면 이러한 그룹화로 계산을 제한할 수 없습니다. 조합 열의 경우에는 롤업 시 JOB_ID 및 MANAGER_ID 열을 하나의 단위로 처리함으로써 그룹화 제한이 가능합니다. 괄호로 묶인 열은 ROLLUP 및 CUBE 계산 시 하나의 단위로 처리됩니다. 이러한 내용이 슬라이드 예제에 설명되어 있습니다. JOB_ID 열과 MANAGER_ID 열을 괄호로 묶어 Oracle 서버에서 JOB_ID 및 MANAGER_ID를 단일 단위, 즉 조합 열로 처리하도록 지시하십시오.

조합 열: 예제(계속)

슬라이드의 예제는 다음 그룹화를 계산합니다.

- (department_id, job_id, manager_id)
- (department_id)
- ()

슬라이드의 예제는 다음을 표시합니다.

- 모든 직무 및 관리자에 대한 총 급여(레이블 1)
- 모든 부서, 직무 및 관리자에 대한 총 급여(레이블 2)
- 모든 부서에 대한 총 급여(레이블 3)
- 총계(레이블 4)

슬라이드의 예제를 다음과 같이 작성할 수도 있습니다.

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM   employees
GROUP  BY department_id, job_id, manager_id
UNION  ALL
SELECT  department_id, TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY department_id
UNION ALL
SELECT  TO_NUMBER(NULL), TO_CHAR(NULL), TO_NUMBER(NULL),
        SUM(salary)
FROM    employees
GROUP BY ( );
```

Query 블록을 찾아서 실행 계획을 생성하는 옵티마이저가 없을 경우 이전 query는 기본 테이블 EMPLOYEES를 세 번 스캔해야 합니다. 이는 매우 비효율적일 수 있습니다. 따라서 조합 열을 사용하는 것이 좋습니다.

연결된 그룹화

- 연결된 그룹화는 유용한 그룹화 조합을 생성하는 간단한 방법을 제공합니다.
- 연결된 그룹화 집합을 지정하려면 Oracle 서버에서 여러 그룹화 집합, ROLLUP 및 CUBE 연산을 단일 GROUP BY 절에 결합하도록 이들을 쉼표로 구분하십시오.
- 결과는 각 GROUPING SET에서 가져온 그룹화의 cross-product입니다.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연결된 그룹화

연결된 그룹화는 유용한 그룹화 조합을 생성하는 간단한 방법을 제공합니다. 여러 그룹화 집합, CUBE 및 ROLLUP을 나열하고 이들을 콤마로 구분하여 지정됩니다. 다음은 연결된 그룹화 집합의 예입니다.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

이 SQL 예제는 다음 그룹화를 정의합니다.

```
(a, c), (a, d), (b, c), (b, d)
```

그룹화 집합을 연결하면 다음과 같은 이점이 있습니다.

- **Query 개발의 용이성:** 모든 그룹화를 수동으로 열거하지 않아도 됩니다.
- **응용 프로그램에서 사용:** OLAP(Online Analytic Processing) 응용 프로그램에서 생성한 SQL에는 종종 그룹화 집합에 대한 연결이 포함됩니다. 각 GROUPING SET는 한 차원에 필요한 그룹화를 정의합니다.

연결된 그룹화: 예제

```
SELECT  department_id, job_id, manager_id,
        SUM(salary)
FROM    employees
GROUP BY department_id,
        ROLLUP(job_id),
        CUBE(manager_id);
```

	DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
1	(null)	SA_REP	149	7000
2	10	AD_ASST	101	4400
3	20	MK_MAN	100	13000
4	20	MK_REP	201	6000
...				
1	90	AD_VP	100	34000
	90	AD_PRES	(null)	24000
...				
	(null)	SA_REP	(null)	7000
	10	AD_ASST	(null)	4400
...				
2	110	(null)	101	12000
	110	(null)	205	8300
	110	(null)	(null)	20300

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연결된 그룹화: 예제

슬라이드 예제를 실행하면 다음과 같은 그룹화가 생성됩니다.

- (department_id, job_id,) (1)
- (department_id, manager_id) (2)
- (department_id) (3)

각 그룹에 대한 총 급여가 계산됩니다.

다음은 연결된 그룹화의 또 다른 예입니다.

```
SELECT department_id, job_id, manager_id, SUM(salary) totalsal
FROM employees
WHERE department_id < 60
GROUP BY GROUPING SETS (department_id),
        GROUPING SETS (job_id, manager_id);
```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- ROLLUP 연산을 사용하여 소계 값 생성
- CUBE 연산을 사용하여 교차 분석 값 생성
- GROUPING 함수를 사용하여 ROLLUP 또는 CUBE에 의해 생성된 행 값 식별
- GROUPING SETS 구문을 사용하여 동일한 query에서 여러 그룹화 정의
- GROUP BY 절을 사용하여 다음과 같은 방법으로 표현식 결합
 - 조합 열
 - 연결된 그룹화 집합

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

- ROLLUP과 CUBE는 GROUP BY 절의 확장입니다.
- ROLLUP은 소계와 총계 값을 표시하는 데 사용됩니다.
- CUBE는 교차 분석 값을 표시하는 데 사용됩니다.
- GROUPING 함수를 사용하면 행이 CUBE 또는 ROLLUP 연산자로 생성된 집계인지 여부를 판별할 수 있습니다.
- GROUPING SETS 구문을 사용하여 동일한 query에서 여러 그룹화를 정의할 수 있습니다. GROUP BY는 지정된 그룹화를 모두 계산하고 이들을 UNION ALL과 결합합니다.
- 여러 가지 방법으로 GROUP BY 절 내에 표현식을 결합할 수 있습니다.
 - 조합 열을 지정하려면 열을 괄호로 묶어서 그룹화합니다. 이렇게 하면 Oracle 서버에서 ROLLUP 또는 CUBE 연산을 계산할 때 괄호로 묶인 열을 하나의 단위로 처리합니다.
 - 연결된 그룹화 집합을 지정하려면 여러 그룹화 집합, ROLLUP 및 CUBE 연산을 콤마로 구분하여 Oracle 서버가 이들을 단일 GROUP BY 절로 결합할 수 있도록 합니다. 결과는 각 그룹화 집합에서 가져온 그룹화의 cross-product입니다.



계층적 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- Hierarchical query의 개념 이해
- 트리 구조의 보고서 생성
- 계층 구조의 데이터 서식 지정
- 트리 구조에서 분기 제거

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 hierarchical query를 사용하여 트리 구조의 보고서를 생성하는 방법을 설명합니다.

EMPLOYEES 테이블의 예제 데이터

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	100	King	AD_PRES	(null)
2	101	Kochhar	AD_VP	100
3	102	De Haan	AD_VP	100
4	103	Hunold	IT_PROG	102
5	104	Ernst	IT_PROG	103
6	107	Lorentz	IT_PROG	103

...

16	200	Whalen	AD_ASST	101
17	201	Hartstein	MK_MAN	100
18	202	Fay	MK_REP	201
19	205	Higgins	AC_MGR	101
20	206	Gietz	AC_ACCOUNT	205

ORACLE

Copyright © 2009, Oracle. All rights reserved.

EMPLOYEES 테이블의 예제 데이터

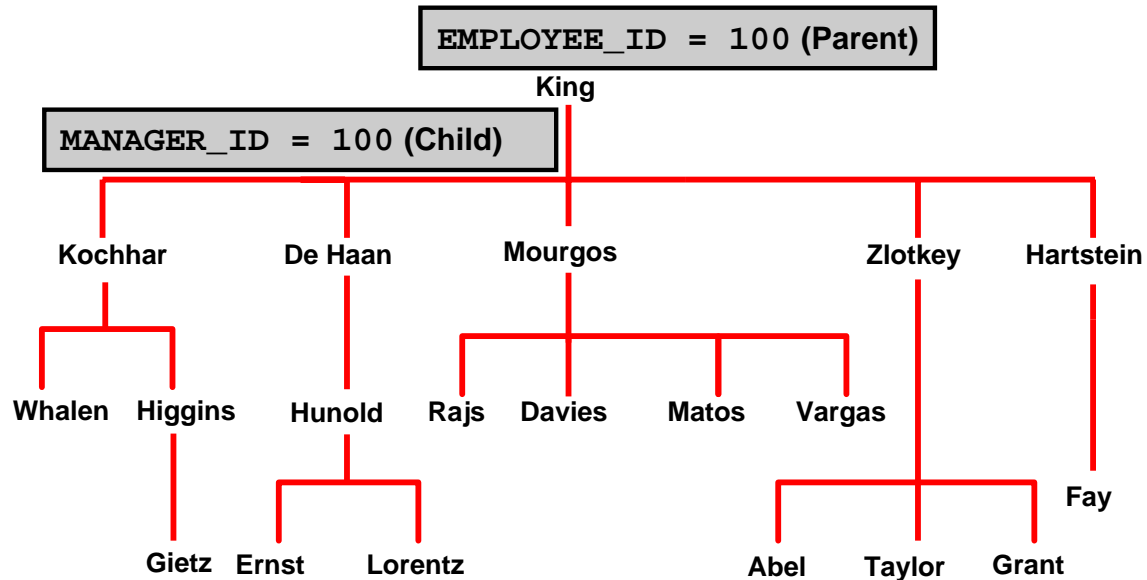
Hierarchical query를 사용하면 테이블에 있는 행 간의 자연적 계층 관계에 준하여 데이터를 검색할 수 있습니다. 관계형 데이터베이스는 레코드를 계층 방식으로 저장하지 않습니다. 그러나 한 테이블의 행 사이에 계층 관계가 존재하면 트리 탐색이라는 프로세스를 사용하여 계층을 구성할 수 있습니다. Hierarchical query는 보고의 한 방법으로, 특정 순서로 된 트리 분기가 있습니다.

가장 오래된 계열 멤버는 트리의 기부 또는 몸체 가까이에 있고 가장 최신 멤버는 트리의 분기를 나타내는 계열 트리를 가정해 봅시다. 분기에는 자체 분기가 있을 수 있습니다.

테이블의 행 사이에 관계가 있으면 hierarchical query가 가능합니다. 예를 들어, 위 슬라이드에서 Kochhar, De Haan, Hartstein은 MANAGER_ID 100에게 보고하게 되어 있는데, 이 ID는 King의 EMPLOYEE_ID입니다.

참고: 계층 트리는 계보(가계도), 가축류(품종 개량 목적), 기업 관리(관리 계층), 제조(제품 어셈블리), 진화론 연구(종의 진화), 과학 연구 등의 여러 분야에 사용됩니다.

일반 트리 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

일반 트리 구조

EMPLOYEES 테이블에는 관리 보고 라인을 나타내는 트리 구조가 있습니다. EMPLOYEE_ID 열과 MANAGER_ID 열의 해당 값 간의 관계를 보고 계층을 생성할 수 있습니다. 테이블을 자체에 조인하여 이 관계를 이용할 수 있습니다. MANAGER_ID 열은 해당 사원의 관리자의 사원 번호를 포함합니다.

트리 구조의 상/하위 관계를 사용하여 다음을 제어할 수 있습니다.

- 계층 탐색 방향
- 계층 내의 시작점

참고: 슬라이드는 EMPLOYEES 테이블에 있는 사원 관리 계층의 역 트리 구조를 표시한 것입니다.

Hierarchical Query

```
SELECT [LEVEL], column, expr...  
FROM table  
[WHERE condition(s)]  
[START WITH condition(s)]  
[CONNECT BY PRIOR condition(s)] ;
```

조건:

```
expr comparison_operator expr
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

키워드와 절

Hierarchical query는 CONNECT BY 절과 START WITH 절의 존재 여부로 식별할 수 있습니다. 이 구문에서 다음이 적용됩니다.

SELECT	표준 SELECT 절입니다.
LEVEL	hierarchical query에 의해 반환되는 각 행의 경우 LEVEL pseudocolumn은 루트 행에 대해 1을 반환하고 루트의 하위 행에 대해 2를 반환합니다(이하 같은 방식).
FROM table	열을 포함하는 테이블, 뷰 또는 스냅샷을 지정합니다. 한 테이블에서만 선택할 수 있습니다.
WHERE	계층 구조의 다른 행에 영향을 주지 않으면서 query에 의해 반환되는 행을 제한합니다.
condition	표현식을 사용한 비교입니다.
START WITH	계층의 루트 행(시작 위치)을 지정합니다. 이 절은 실제 hierarchical query에 필요합니다.
CONNECT BY	상위 및 하위 PRIOR 행 사이에 관계가 있는 열을 지정합니다. 이 절은 hierarchical query에 필요합니다.

트리 탐색

시작점

- 충족해야 하는 조건을 지정합니다.
- 유효한 조건을 받아들입니다.

```
START WITH column1 = value
```

EMPLOYEES 테이블을 사용하여 성이 Kochhar인 사원부터 시작합니다.

```
...START WITH last_name = 'Kochhar'
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리 탐색

트리의 루트로 사용할 행은 START WITH 절에 의해 결정됩니다. START WITH 절은 유효한 조건을 포함할 수 있습니다.

예제

EMPLOYEES 테이블을 사용하여 회사 대표인 King부터 시작합니다.

```
... START WITH manager_id IS NULL
```

EMPLOYEES 테이블을 사용하여 사원 Kochhar부터 시작합니다. START WITH 조건은 subquery를 포함할 수 있습니다.

```
... START WITH employee_id = (SELECT employee_id  
                                FROM employees  
                                WHERE last_name = 'Kochhar')
```

START WITH 절을 생략하면 테이블의 모든 행을 루트 행으로 간주하고 트리 탐색이 시작됩니다.

참고: CONNECT BY 및 START WITH 절은 ANSI(American National Standards Institute) SQL 표준이 아닙니다.

트리 탐색

```
CONNECT BY PRIOR column1 = column2
```

EMPLOYEES 테이블을 사용하여 하향식으로 탐색합니다.

```
... CONNECT BY PRIOR employee_id = manager_id
```

방향

하향식 → Column1 = 상위 키
 Column2 = 하위 키

상향식 → Column1 = 하위 키
 Column2 = 상위 키

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리 탐색

Query 방향은 CONNECT BY PRIOR 절의 위치에 의해 결정됩니다. 하향식의 경우 PRIOR 연산자를 상위 행을 참조하고, 하향식의 경우 PRIOR 연산자를 하위 행을 참조합니다. 상위 행의 하위 행을 찾기 위해 Oracle 서버는 상위 행에 대해 PRIOR 표현식을 평가하고 테이블의 각 행에 대해 다른 표현식을 평가합니다. 조건이 참인 행은 상위 행의 하위 행입니다. Oracle 서버는 항상 현재 상위 행에 대해 CONNECT BY 조건을 평가하여 하위 행을 선택합니다.

예제

EMPLOYEES 테이블을 사용하여 하향식으로 탐색합니다. 상위 행의 EMPLOYEE_ID 값이 하위 행의 MANAGER_ID 값과 같은 계층 관계를 정의합니다.

```
... CONNECT BY PRIOR employee_id = manager_id
```

EMPLOYEES 테이블을 사용하여 상향식으로 탐색합니다.

```
... CONNECT BY PRIOR manager_id = employee_id
```

PRIOR 연산자를 반드시 CONNECT BY 바로 뒤에 코딩할 필요는 없습니다. 따라서 다음 CONNECT BY PRIOR 절은 이전 예제의 절과 동일한 결과를 나타냅니다.

```
... CONNECT BY employee_id = PRIOR manager_id
```

참고: CONNECT BY 절은 subquery를 포함할 수 없습니다.

트리 탐색: 상향식

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
1	101	Kochhar	AD_VP	100
2	100	King	AD_PRES	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리 탐색: 상향식

슬라이드의 예제는 사원 ID가 101인 사원부터 시작되는 관리자 리스트를 표시합니다.

트리 탐색: 하향식

```
SELECT last_name || ' reports to ' ||  
PRIOR last_name "Walk Top Down"  
FROM employees  
START WITH last_name = 'King'  
CONNECT BY PRIOR employee_id = manager_id ;
```

Walk Top Down
1 King reports to
2 King reports to
3 Kochhar reports to King
4 Greenberg reports to Kochhar
5 Fawiet reports to Greenberg

...

105 Grant reports to Zlotkey
106 Johnson reports to Zlotkey
107 Hartstein reports to King
108 Fay reports to Hartstein

ORACLE

Copyright © 2009, Oracle. All rights reserved.

트리 탐색: 하향식

하향식으로 탐색하면서 사원과 관리자의 이름을 표시합니다. King 사원을 시작점으로 사용하고 한 열만 인쇄합니다.

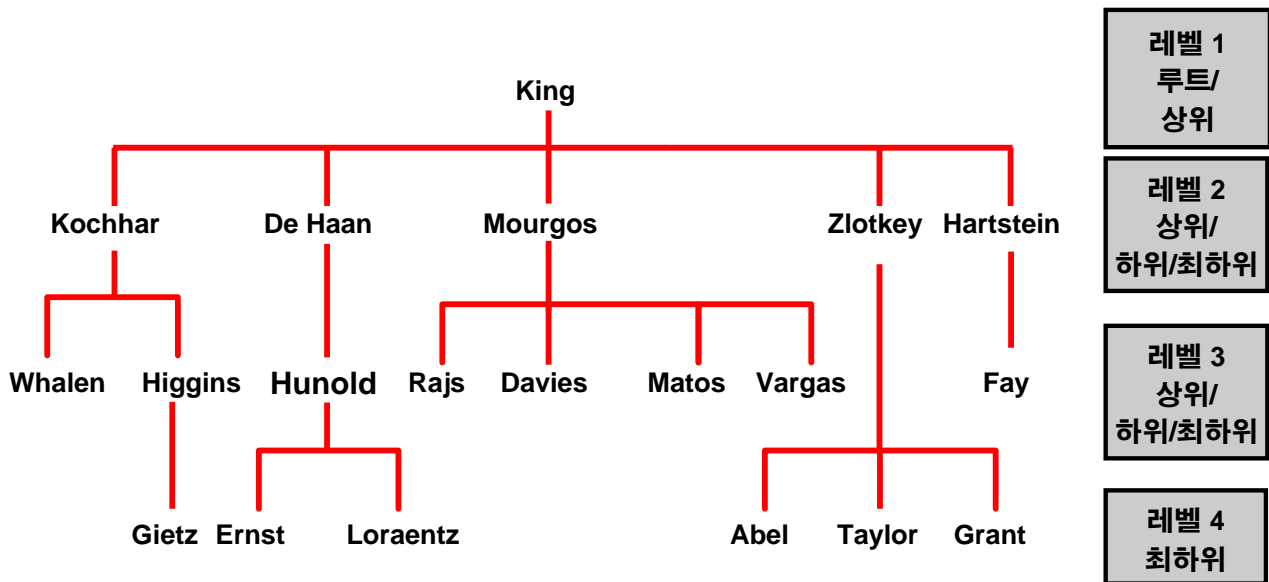
예제

아래 예에서 EMPLOYEE_ID 값은 상위 행과 MANAGER_ID에 대해 평가되고 SALARY 값은 하위 행에 대해 평가됩니다. PRIOR 연산자는 EMPLOYEE_ID 값에만 적용됩니다.

```
... CONNECT BY PRIOR employee_id = manager_id  
AND salary > 15000;
```

하위 행으로 지정하려면 상위 행의 EMPLOYEE_ID 값과 동일한 MANAGER_ID 값이 있어야 하고 \$15,000보다 큰 SALARY 값이 있어야 합니다.

LEVEL 의사 열로 행 순위 지정



ORACLE

Copyright © 2009, Oracle. All rights reserved.

LEVEL 의사 열로 행 순위 지정

LEVEL pseudocolumn을 사용하여 계층 구조의 행 순위나 레벨을 명시적으로 표시할 수 있습니다. 이렇게 하면 보고서를 보다 쉽게 읽을 수 있습니다. 큰 분기에서 하나 이상의 분기가 나뉘는 분기를 노드라고 하며 분기 끝을 최하위 행 또는 최하위 노드라고 합니다. 슬라이드의 그래픽은 역 트리의 노드를 LEVEL 값과 함께 보여줍니다. 예를 들어, 사원 Higgins는 상위 노드인 동시에 하위 노드이고 사원 Davies는 하위 노드인 동시에 최하위 노드입니다.

LEVEL Pseudocolumn

값	하향식 레벨	상향식 레벨
1	루트 노드	루트 노드
2	루트 노드의 하위 노드	루트 노드의 상위 노드
3	하위 노드의 하위 노드 등	상위 노드의 상위 노드 등

슬라이드에서 King은 루트 또는 상위 노드입니다(LEVEL = 1). Kochhar, De Haan, Mourgos, Zlotkey, Hartstein, Higgins 및 Hunold는 하위 노드인 동시에 상위 노드입니다(LEVEL = 2). Whalen, Rajs, Davies, Matos, Vargas, Gietz, Ernst, Lorentz, Abel, Taylor, Grant 및 Fay는 하위 노드인 동시에 최하위 노드입니다(LEVEL = 3 및 LEVEL = 4).

참고: 루트 노드는 역 트리 내의 최상위 노드입니다. 하위 노드는 루트가 아닌 노드입니다. 상위 노드는 하위 노드가 있는 노드입니다. 최하위 노드는 하위 노드가 없는 노드입니다. Hierarchical query에 의해 반환된 레벨 수는 사용 가능한 유저 메모리의 제한을 받습니다.

LEVEL 및 LPAD를 사용하여 계층 보고서 형식 지정

최상위 레벨에서 시작하여 다음 레벨을 각각 들여쓰기한 형태의 회사 관리 레벨이 표시된 보고서를 생성합니다.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LEVEL 및 LPAD를 사용하여 계층 보고서 형식 지정

트리의 노드에는 루트 기준의 레벨 수가 지정됩니다. LEVEL pseudocolumn과 함께 LPAD 함수를 사용하여 계층 보고서를 들여쓰기한 트리 형식으로 표시합니다.

슬라이드의 예제는 다음과 같습니다.

- `LPAD(char1, n [, char2])`는 길이 `n`의 왼쪽을 `char2`의 문자 시퀀스로 채운 `char1`을 반환합니다. 인수 `n`은 터미널 화면에 표시되는 반환 값의 총 길이입니다.
- `LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')`는 표시 형식을 정의합니다.
- `char1`은 `LAST_NAME`이고, `n`은 반환 값의 총 길이로 `LAST_NAME + (LEVEL*2) - 2`이고, `char2`는 `'_'`입니다.

즉, 이 명령문은 SQL에게 `LAST_NAME`을 가져와서 결과 문자열의 길이가 `LENGTH(last_name) + (LEVEL*2) - 2`에 의해 결정된 값과 같아질 때까지 왼쪽을 `'_'` 문자로 채우도록 지시합니다.

King의 경우 `LEVEL = 1`이므로, $(2 * 1) - 2 = 2 - 2 = 0$ 입니다. 따라서 King은 `'_'` 문자로 채워지지 않고 1열에 표시됩니다.

Kochhar의 경우 `LEVEL = 2`이므로 $(2 * 2) - 2 = 4 - 2 = 2$ 입니다. 따라서 Kochhar는 두 개의 `'_'` 문자로 채워지고 들여쓰기한 상태로 표시됩니다.

EMPLOYEES 테이블에서 레코드의 나머지 부분은 비슷하게 표시됩니다.

LEVEL 및 LPAD를 사용하여 계층 보고서 서식 지정(계속)

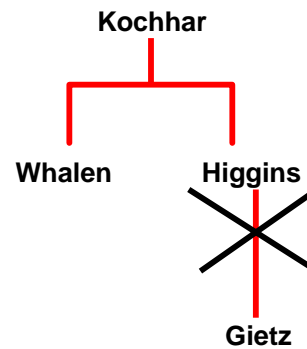
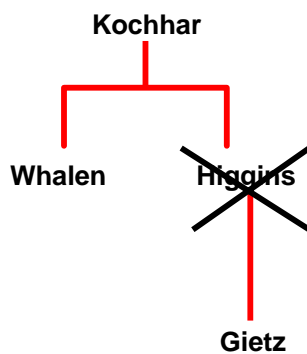
R2 ORG_CHART	
1	King
2	__Kochhar
3	___Greenberg
4	____Faviet
5	_____Chen
6	_____Sciarra
7	_____Urman
8	_____Popp
9	_____Whalen
10	_____Mavris
11	_____Baer
12	_____Higgins
13	_____Gietz
14	__De Haan
15	___Hunold
16	____Ernst
17	_____Austin

분기 제거

WHERE 절을 사용하여
노드를 제거합니다.

CONNECT BY 절을 사용하여
분기를 제거합니다.

```
WHERE last_name != 'Higgins' CONNECT BY PRIOR
employee_id = manager_id
AND last_name != 'Higgins'
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

분기 제거

WHERE 절과 CONNECT BY 절을 사용하여 트리를 제거합니다. 즉, 표시할 노드 또는 행을 제어합니다. 사용하는 술어는 부울 조건으로 동작합니다.

예제

루트에서 시작하여 하향식으로 탐색하고 결과에서 Higgins 사원은 제거하고 하위 행은 처리합니다.

```
SELECT department_id, employee_id, last_name, job_id, salary
FROM employees
WHERE last_name != 'Higgins'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

루트에서 시작하여 하향식으로 탐색하고 Higgins 사원과 하위 행을 모두 제거합니다.

```
SELECT department_id, employee_id, last_name, job_id, salary
FROM employees
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND last_name != 'Higgins';
```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- Hierarchical query를 사용하여 테이블의 행 사이의 계층 관계 확인
- Query 방향 및 시작점 지정
- 노드 또는 분기 제거

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

Hierarchical query를 사용하여 테이블에 있는 행 사이의 계층 관계에 준하여 데이터를 검색할 수 있습니다. LEVEL 의사 열은 탐색한 계층 트리의 레벨을 계산합니다. CONNECT BY PRIOR 절을 사용하여 query 방향을 지정할 수 있습니다. START WITH 절을 사용하여 시작점을 지정할 수 있습니다. WHERE 절과 CONNECT BY 절을 사용하여 트리 분기를 제거할 수 있습니다.

III

고급 스크립트 작성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- **SQL을 사용하여 SQL을 생성함으로써 해결되는 문제 유형 설명**
- **DROP TABLE 문의 스크립트를 생성하는 스크립트 작성**
- **INSERT INTO 문의 스크립트를 생성하는 스크립트 작성**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록에서는 SQL 스크립트를 생성하는 SQL 스크립트 작성 방법을 설명합니다.

SQL을 사용하여 SQL 생성

- SQL은 SQL 언어로 된 스크립트를 생성하는 데 사용됩니다.
- 데이터 디렉터리의 특성은 다음과 같습니다.
 - 데이터베이스 정보를 포함하는 테이블 및 뷰의 모음입니다.
 - Oracle 서버에 의해 생성 및 유지 관리됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL을 사용하여 SQL 생성

SQL은 다른 SQL 문을 생성하기 위한 강력한 도구가 될 수 있습니다. 대부분의 경우 SQL은 스크립트 파일 작성과 관련이 있습니다. SQL에서 SQL을 사용하여 다음을 수행할 수 있습니다.

- 반복 코딩을 피함
- 데이터 디렉터리의 정보 액세스
- 데이터베이스 객체 삭제 또는 재생성
- 런타임 파라미터를 포함하는 동적 SQL 생성

이 부록에 사용된 예제에는 데이터 디렉터리의 정보를 선택하는 과정이 포함되어 있습니다.

데이터 디렉터리란 데이터베이스에 대한 정보를 포함하는 테이블 및 뷰의 모음입니다. 이 모음은 Oracle 서버에 의해 생성 및 유지 관리됩니다. 모든 데이터 디렉터리 테이블은 SYS 사용자가 소유합니다. 데이터 디렉터리에 저장되는 정보에는 Oracle 서버 유저의 이름, 유저에게 부여된 권한, 데이터베이스 객체 이름, 테이블 제약 조건 및 감사(audit) 정보가 포함됩니다. 데이터 디렉터리 뷰에는 네 가지 범주가 있습니다. 범주마다 의도된 용도를 명확하게 반영하는 접두어가 있습니다.

SQL을 사용하여 SQL 생성(계속)

접두어	설명
USER_	유저가 소유한 객체의 세부 정보를 포함합니다.
ALL_	유저가 소유한 객체 외에도 유저에게 액세스 권한이 부여된 객체의 세부 정보를 포함합니다.
DBA_	데이터베이스에 있는 객체에 액세스할 수 있는 DBA 권한을 가진 유저의 세부 정보를 포함합니다.
V\$_	데이터베이스 서버 성능 및 잠금에 대한 정보를 저장하며 DBA만 사용할 수 있습니다.

기본 스크립트 작성

```
SELECT 'CREATE TABLE ' || table_name ||  
      '_test ' || 'AS SELECT * FROM '  
      || table_name || ' WHERE 1=2;'  
      AS "Create Table Script"  
FROM   user_tables;
```

1	Create Table Script
1	CREATE TABLE REGIONS_test AS SELECT * FROM REGIONS WHERE 1=2;
2	CREATE TABLE LOCATIONS_test AS SELECT * FROM LOCATIONS WHERE 1=2;
3	CREATE TABLE DEPARTMENTS_test AS SELECT * FROM DEPARTMENTS WHERE 1=2;
4	CREATE TABLE JOBS_test AS SELECT * FROM JOBS WHERE 1=2;
5	CREATE TABLE EMPLOYEES_test AS SELECT * FROM EMPLOYEES WHERE 1=2;
6	CREATE TABLE JOB_HISTORY_test AS SELECT * FROM JOB_HISTORY WHERE 1=2;

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 스크립트

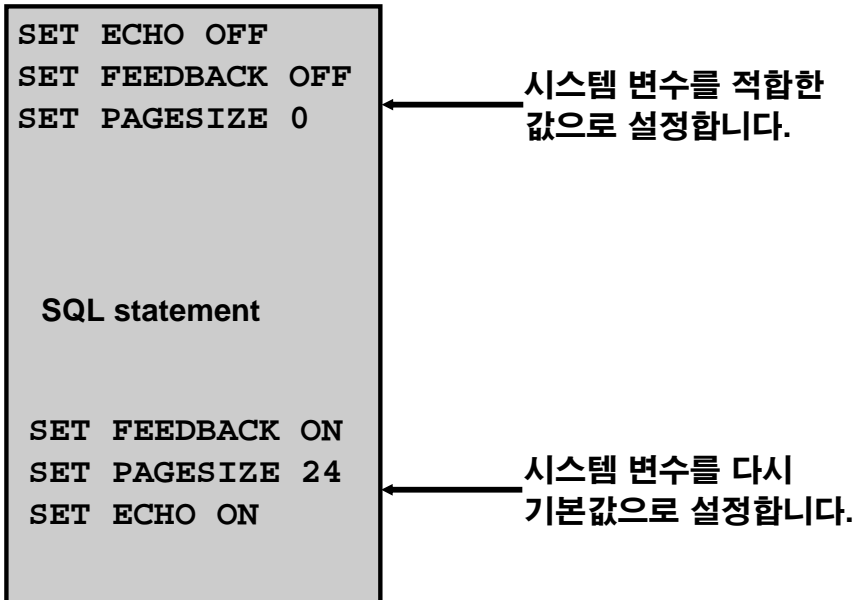
슬라이드의 예제는 사용자가 소유한 모든 테이블에서 CREATE TABLE 문을 사용하는 보고서를 생성합니다. 보고서에 생성된 각 CREATE TABLE 문의 구문은 접미어가 _test인 테이블 이름을 사용하고 해당되는 기존 테이블에서 구조만 취하여 테이블을 생성합니다. 이전 테이블 이름은 데이터 디렉터리 뷰 USER_TABLES의 TABLE_NAME 열에서 가져옵니다.

다음 단계는 프로세스를 자동화할 수 있도록 보고서를 향상시키는 것입니다.

참고: 데이터 디렉터리 테이블을 query하여 자신이 소유한 다양한 데이터베이스 객체를 조회할 수 있습니다. 자주 사용되는 데이터 디렉터리 뷰는 다음과 같습니다.

- USER_TABLES: 사용자가 소유한 테이블에 대한 설명을 표시합니다.
- USER_OBJECTS: 사용자가 소유한 모든 객체를 표시합니다.
- USER_TAB_PRIVS_MADE: 사용자가 소유한 객체에 대해 부여된 모든 권한을 표시합니다.
- USER_COL_PRIVS_MADE: 사용자가 소유한 객체의 열에 대해 부여된 모든 권한을 표시합니다.

환경 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

환경 제어

생성된 SQL 문을 실행하려면 SQL 문을 파일로 캡처한 다음 해당 파일을 실행해야 합니다. 또한 생성되는 출력을 정리할 계획을 세워서 머리글, 피드백 메시지, 상단 제목 등의 요소가 출력되지 않도록 해야 합니다. SQL Developer에서는 이러한 명령문을 스크립트로 저장할 수 있습니다.

Enter SQL Statement 상자의 내용을 저장하려면 Save 아이콘을 누르거나 **File > Save** 메뉴 항목을 사용하십시오. 또는 Enter SQL Statement 상자를 마우스 오른쪽 버튼으로 누른 다음 드롭다운 메뉴에서 Save File 옵션을 선택하십시오.

참고: 일부 SQL*Plus 문은 SQL Worksheet에서 지원되지 않습니다. 지원되는 SQL*Plus 문과 SQL Worksheet에서 지원하지 않는 SQL*Plus 문의 전체 리스트는 SQL Developer 온라인 도움말에서 *SQL*Plus Statements Supported and Not Supported in SQL Worksheet* 항목을 참조하십시오.

전체 그림

```
SET ECHO OFF
SET FEEDBACK OFF
SET PAGESIZE 0

SELECT 'DROP TABLE ' || object_name || ';'
FROM   user_objects
WHERE  object_type = 'TABLE'
/

SET FEEDBACK ON
SET PAGESIZE 24
SET ECHO ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

전체 그림

위 슬라이드의 명령 출력은 SQL Developer에서 dropem.sql 파일에 저장됩니다. SQL Developer에서 출력을 파일로 저장하려면 Script Output 창 아래에 있는 Save File 옵션을 사용하십시오. dropem.sql 파일은 다음 데이터를 포함합니다. 이 파일은 이제 스크립트 파일을 찾아 로드하고 실행하여 SQL Developer에서 시작될 수 있습니다.

	'DROPTABLE' OBJECT_NAME ';'
1	DROP TABLE REGIONS;
2	DROP TABLE COUNTRIES;
3	DROP TABLE LOCATIONS;
4	DROP TABLE DEPARTMENTS;
5	DROP TABLE JOBS;
6	DROP TABLE EMPLOYEES;
7	DROP TABLE JOB_HISTORY;
8	DROP TABLE JOB_GRADES;

테이블 내용을 파일로 덤프

```
SET HEADING OFF ECHO OFF FEEDBACK OFF
SET PAGESIZE 0

SELECT
  'INSERT INTO departments_test VALUES
  (' || department_id || ', ' || department_name ||
  ', ' || location_id || ');'
  AS "Insert Statements Script"
FROM departments
/

SET PAGESIZE 24
SET HEADING ON ECHO ON FEEDBACK ON
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 내용을 파일로 덤프

테이블 행에 대한 값을 텍스트 파일에 INSERT INTO VALUES 문 형식으로 저장해 놓으면 유용하게 사용할 수 있습니다. 테이블이 실수로 삭제되었을 경우 이 스크립트를 실행하여 테이블을 채울 수 있습니다.

위 슬라이드의 예제는 SQL Developer에서 Save File 옵션을 사용하여 data.sql 파일에 캡처된, DEPARTMENTS_TEST 테이블에 대한 INSERT 문을 생성합니다.

data.sql 스크립트 파일의 내용은 다음과 같습니다.

```
INSERT INTO departments_test VALUES
  (10, 'Administration', 1700);
INSERT INTO departments_test VALUES
  (20, 'Marketing', 1800);
INSERT INTO departments_test VALUES
  (50, 'Shipping', 1500);
INSERT INTO departments_test VALUES
  (60, 'IT', 1400);
...
```

테이블 내용을 파일로 덤프

소스	결과
<code>'X'</code>	<code>'X'</code>
<code>''</code>	<code>'</code>
<code>'' department_name ''</code>	<code>'Administration'</code>
<code>','</code>	<code>','</code>
<code>');'</code>	<code>');'</code>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 내용을 파일로 덤프(계속)

이전 슬라이드에서 작은 따옴표를 많이 볼 수 있었습니다. 네 개의 작은 따옴표로 구성된 한 집합은 최종 명령문에서 하나의 작은 따옴표로 나타납니다. 또한 문자 및 날짜 값은 따옴표로 묶어야 합니다.

한 문자열 내에서 하나의 작은 따옴표를 표시하려면, 그 앞에 또 다른 작은 따옴표를 붙여야 합니다. 예를 들어 슬라이드의 다섯번째 예제에서, 바깥을 둘러싼 따옴표는 전체 문자열에 대한 따옴표입니다. 두번째 따옴표는 세번째 따옴표를 표시하기 위한 접두어 역할을 합니다. 따라서 결과적으로 하나의 작은 따옴표 다음에 괄호가 오고 그 다음에 세미콜론이 옵니다.

동적 슬어 생성

```
COLUMN my_col NEW_VALUE dyn_where_clause

SELECT DECODE('&deptno', null,
DECODE ('&hiredate', null, ' ',
'WHERE hire_date=TO_DATE(''||'&hiredate'', 'DD-MON-YYYY'')),
DECODE ('&hiredate', null,
'WHERE department_id = ' || '&deptno',
'WHERE department_id = ' || '&deptno' ||
' AND hire_date = TO_DATE(''||'&hiredate'', 'DD-MON-YYYY''))
AS my_col FROM dual;
```

```
SELECT last_name FROM employees &dyn_where_clause;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

동적 슬어 생성

위 슬라이드의 예제는 부서에서 특정일에 채용된 모든 사원에 대한 데이터를 검색하는 SELECT 문을 생성합니다. 스크립트는 WHERE 절을 동적으로 생성합니다.

참고: 일단 유저 변수가 나타나면 UNDEFINE 명령을 사용하여 변수를 삭제해야 합니다.

첫번째 SELECT 문은 유저에게 부서 번호 입력을 요청합니다. 부서 번호를 입력하지 않을 경우 DECODE 함수에 의해 부서 번호는 널로 취급되며, 유저는 다음으로 채용 날짜 입력을 요청 받습니다. 채용 날짜를 입력하지 않을 경우 채용 날짜는 DECODE 함수에 의해 널로 처리되어 생성되는 동적 WHERE 절도 널이 되며, 그 결과 두번째 SELECT 문은 EMPLOYEES 테이블에서 모든 행을 검색하게 됩니다.

참고: NEW_V[ALUE] 변수는 열 값을 보유하기 위한 변수를 지정합니다. TTITLE 명령의 변수를 참조할 수 있습니다. NEW_VALUE를 사용하여 최상위 제목에 열 값 또는 날짜를 표시합니다. SKIP PAGE 작업을 사용하여 BREAK 명령에 열을 포함시켜야 합니다. 변수 이름에는 파운드 기호(#)가 포함될 수 없습니다. NEW_VALUE는 각 페이지에 대한 새 마스터 레코드가 있는 마스터/상세 보고서에 유용합니다.

동적 술어 생성(계속)

참고: 여기에서 채용 날짜는 DD-MON-YYYY 형식으로 입력해야 합니다.

슬라이드의 SELECT 문을 다음과 같이 해석할 수 있습니다.

```
IF (<<deptno>> is not entered) THEN
    IF (<<hiredate>> is not entered) THEN
        return empty string
    ELSE
        return the string 'WHERE hire_date =
TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
    ELSE
        IF (<<hiredate>> is not entered) THEN
            return the string 'WHERE department_id =
<<deptno>> entered'
        ELSE
            return the string 'WHERE department_id =
<<deptno>> entered
                                AND hire_date =
TO_DATE(' <<hiredate>>', 'DD-MON-YYYY')'
        END IF
    END IF
```

반환된 문자열은 DYN_WHERE_CLAUSE 변수의 값이 되고, 이 값은 두번째 SELECT 문에 사용됩니다.

참고: 이 예제에는 SQL*Plus를 사용하십시오.

슬라이드의 첫번째 예제가 실행되면 유저는 DEPTNO 및 HIREDATE에 대한 값을 입력해야 합니다.

Enter value for deptno: 10

Enter value for hiredate: 17-SEP-1987

다음과 같은 MY_COL 값이 생성됩니다.

```
MY_COL
-----
WHERE department_id = 10 AND hire_date = TO_DATE('27-SEP-1987','DD-MON-YYYY')
```

슬라이드의 두번째 예제가 실행되면 다음과 같은 출력이 생성됩니다.

```
LAST_NAME
-----
Whalen
```

요약

이 부록에서는 다음 항목에 대해 설명했습니다.

- SQL 스크립트를 작성하여 다른 SQL 스크립트 생성
- 스크립트 파일을 데이터 디렉터리로 사용
- 출력을 파일로 캡처

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

SQL은 SQL 스크립트를 생성하는 데 사용될 수 있습니다. 이러한 스크립트는 반복 코딩을 피하고, 객체를 삭제하거나 재생성하며, 데이터 디렉터리에서 도움말을 얻거나 런타임 파라미터를 포함하는 동적 SQL을 생성하는 데 사용될 수 있습니다.

오라클 데이터베이스 구조 구성 요소



ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 부록을 마치면 다음을 수행할 수 있습니다.

- **주요 데이터베이스 구조 구성 요소 나열**
- **백그라운드 프로세스 설명**
- **메모리 구조 설명**
- **논리적/물리적 저장 영역 구조 상호 연관**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 오라클 데이터베이스 구조에 대해 간략히 설명합니다. 또한 오라클 데이터베이스의 물리적/논리적 구조를 살펴보고 다양한 구성 요소와 해당 기능에 대해 배웁니다.

오라클 데이터베이스 구조: 개요

Oracle RDBMS(관계형 데이터베이스 관리 시스템)는 정보를 관리하는 데 있어 개방적이고 종합적이며 통합적인 접근 방식을 제공하는 데이터베이스 관리 시스템입니다.



ORACLE

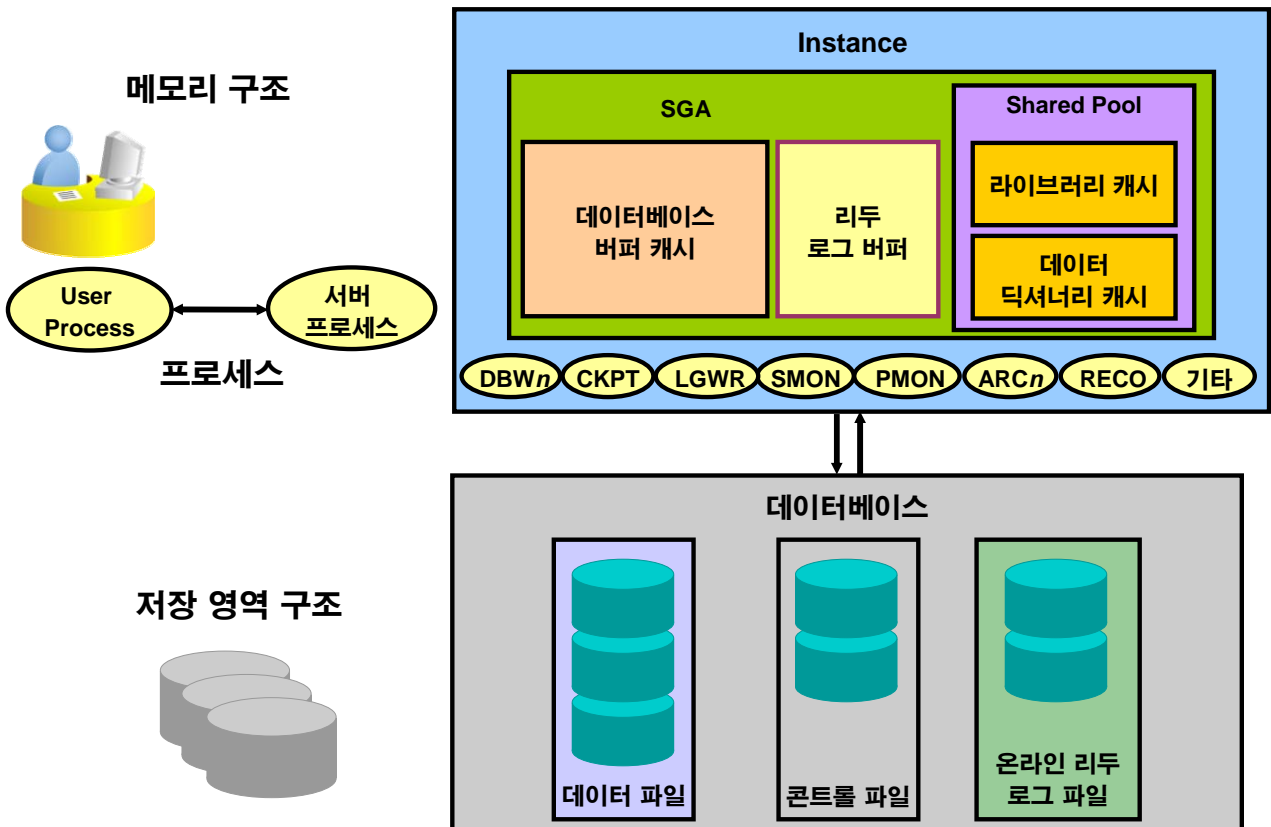
Copyright © 2009, Oracle. All rights reserved.

오라클 데이터베이스 구조: 개요

데이터베이스는 하나의 단위로 취급되는 데이터 모음입니다. 데이터베이스의 목적은 관련 정보를 저장하고 검색하는 것입니다.

오라클 데이터베이스는 다중 유저 환경에서 대량의 데이터를 안정적으로 관리하여 다수의 유저가 동일한 데이터에 동시에 액세스할 수 있도록 합니다. 단, 이 기능은 높은 성능을 제공해야 가능합니다. 동시에 오라클 데이터베이스는 무단 액세스를 차단하고 failure recovery에 대한 효과적인 해결책을 제공합니다.

오라클 데이터베이스 서버 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

오라클 데이터베이스 서버 구조

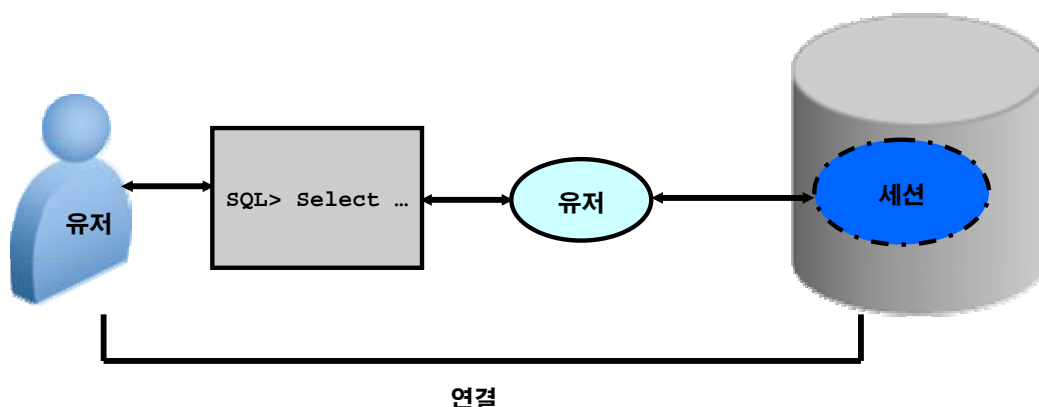
오라클 데이터베이스는 두 가지 주요 구성 요소인 instance와 데이터베이스로 구성됩니다.

- Instance는 메모리 구조인 시스템 글로벌 영역(SGA)과 데이터베이스 내에서 작업을 수행하는 백그라운드 프로세스로 구성됩니다. Instance가 시작될 때마다 SGA가 할당되고 백그라운드 프로세스가 시작됩니다.
- 데이터베이스는 논리적 구조와 물리적 구조로 구성됩니다. 논리적 구조와 물리적 구조는 개별적이므로 논리적 저장 영역 구조에 대한 액세스에 영향을 주지 않고 데이터의 물리적 저장 영역을 관리할 수 있습니다. 물리적 저장 영역 구조는 다음과 같습니다.
 - 데이터베이스 구성이 저장되는 컨트롤 파일
 - 데이터베이스 recovery에 필요한 정보를 포함하는 리두 로그 파일
 - 모든 데이터가 저장되는 데이터 파일

Oracle instance는 메모리 구조와 프로세스를 사용하여 데이터베이스 저장 영역 구조를 관리하고 액세스합니다. 모든 메모리 구조는 데이터베이스 서버를 구성하는 컴퓨터의 기본 메모리에 존재합니다. 프로세스는 이러한 컴퓨터의 메모리에서 작동하는 작업입니다. 프로세스는 일련의 단계를 실행할 수 있는 운영 체제의 "제어 스레드" 또는 메커니즘으로 정의할 수 있습니다.

데이터베이스에 연결

- **연결:** User process와 데이터베이스 instance 사이의 통신 경로
- **세션:** User process를 통해 이루어지는 데이터베이스 instance에 대한 특정 유저 연결



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스에 연결

데이터베이스의 정보에 액세스하기 위해서는 유저가 도구(예: SQL*Plus)를 사용하여 데이터베이스에 연결해야 합니다. 유저가 연결을 설정하면 해당 유저에 대한 세션이 생성됩니다. 연결과 세션은 user process와 밀접하게 관련되어 있지만 의미는 매우 다릅니다.

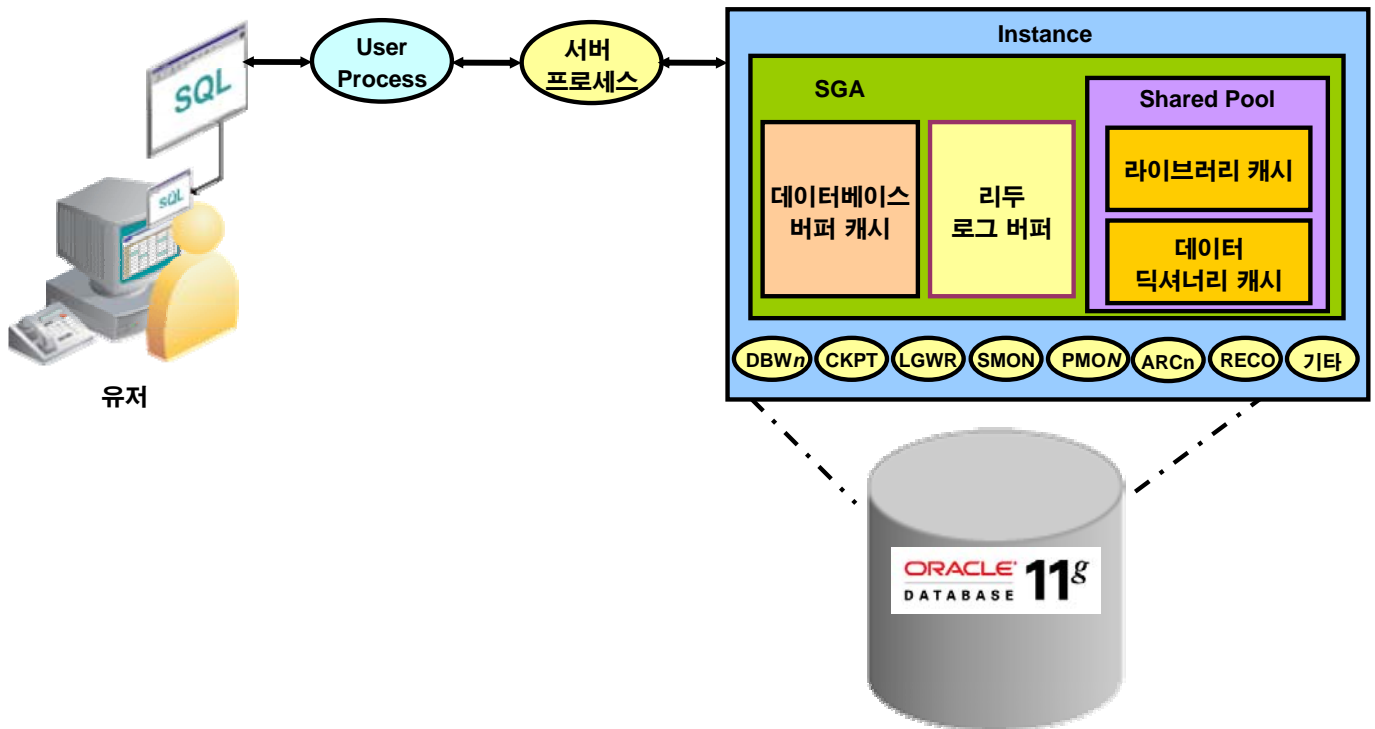
연결은 user process와 오라클 데이터베이스 instance 사이의 통신 경로입니다. 통신 경로는 사용 가능한 프로세스 간 통신 메커니즘 또는 네트워크 소프트웨어(서로 다른 컴퓨터에서 응용 프로그램과 오라클 데이터를 실행하며 네트워크를 통해 통신하는 경우)를 사용하여 설정됩니다.

세션은 데이터베이스 instance에 대한 현재 유저 로그인 상태를 나타냅니다. 예를 들어, 유저가 SQL*Plus를 시작하는 경우 유효한 username과 암호를 제공해야 합니다. 그러면 해당 유저에 대한 세션이 설정됩니다. 세션은 유저가 연결된 때부터 연결을 끊거나 데이터베이스 응용 프로그램을 종료할 때까지 지속됩니다.

전용 연결의 경우 영구 전용 프로세스에서 세션을 처리하고, 공유 연결의 경우 풀에서 선택된 사용 가능한 서버 프로세스, 즉 middle-tier나 Oracle Shared Server 구조에서 세션을 처리합니다.

동일한 username을 사용하여 한 명의 오라클 데이터베이스 유저에 대해 여러 세션을 생성할 수 있으며 이러한 여러 세션이 동시에 존재할 수 있습니다. 단, 다른 응용 프로그램이나 동일한 응용 프로그램의 다중 호출을 통해서만 가능합니다.

오라클 데이터베이스와 상호 작용



Copyright © 2009, Oracle. All rights reserved.

오라클 데이터베이스와 상호 작용

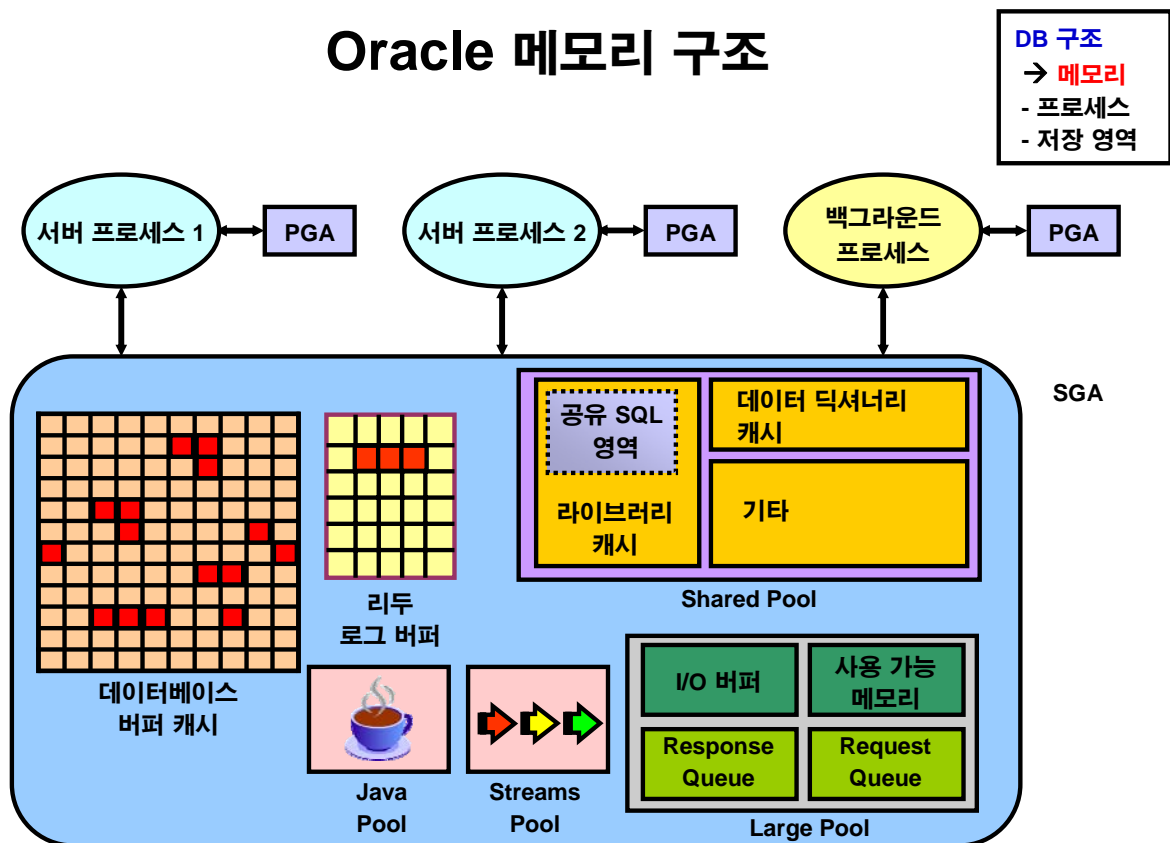
다음 예는 가장 기본적인 레벨의 오라클 데이터베이스 작업에 대해 설명하며, User process와 연관된 서버 프로세스가 네트워크로 연결된 별도의 컴퓨터에 있는 오라클 데이터베이스 구성을 보여줍니다.

1. 오라클 데이터베이스가 설치된 노드(호스트 또는 데이터베이스 서버라고도 함)에서 instance가 시작되었습니다.
2. 유저가 user process를 생성하는 응용 프로그램을 시작합니다. 응용 프로그램에서 서버에 대한 연결을 설정하려고 시도합니다. 로컬 또는 클라이언트 서버 연결이거나 middle-tier로부터의 3계층 연결일 수 있습니다.
3. 서버에서 적절한 Oracle Net Services 처리기가 있는 리스너를 실행합니다. 서버에서 응용 프로그램의 연결 요청을 감지하고 user process 대신 dedicated server 프로세스를 생성합니다.
4. 유저가 DML 유형 SQL 문을 실행하고 트랜잭션을 커밋합니다. 예를 들어, 유저가 테이블에서 고객의 주소를 변경하고 변경 내용을 커밋합니다.
5. 서버 프로세스에서 해당 명령문을 받아 shared pool(SGA 구성 요소)에 유사한 SQL 문이 포함된 공유 SQL 영역이 있는지 검사합니다. 공유 SQL 영역이 있으면 서버 프로세스는 요청된 데이터에 대한 유저의 액세스 권한을 확인하고 기존 공유 SQL 영역이 명령문 처리에 사용됩니다. 공유 SQL 영역이 없으면 명령문이 구문 분석되고 처리될 수 있도록 해당 명령문에 대해 새 공유 SQL 영역이 할당됩니다.

오라클 데이터베이스와 상호 작용(계속)

6. 서버 프로세스가 실제 데이터 파일(테이블이 저장됨)이나 SGA에 캐시된 데이터 파일에서 필요한 데이터 값을 검색합니다.
7. 서버 프로세스가 SGA의 데이터를 수정합니다. 트랜잭션이 커밋되었으므로 LGWR(로그 기록자 프로세스)이 즉시 트랜잭션을 리두 로그 파일에 기록합니다. DBWn(데이터베이스 기록자 프로세스)이 수정된 블록을 디스크에 영구적으로 기록합니다(이렇게 하는 것이 효율적인 경우).
8. 트랜잭션이 성공하면 서버 프로세스가 네트워크를 통해 메시지를 응용 프로그램으로 보냅니다. 그리고 트랜잭션이 실패하면 오류 메시지가 전송됩니다.
9. 이러한 전체 과정에서 개입이 필요한 상황이 되면 다른 백그라운드 프로세스가 실행됩니다. 또한 데이터베이스 서버는 다른 유저의 트랜잭션을 관리하며 동일한 데이터를 요청하는 트랜잭션 간의 경합을 방지합니다.

Oracle 메모리 구조



DB 구조
→ 메모리
- 프로세스
- 저장 영역

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle 메모리 구조

오라클 데이터베이스는 여러 가지 목적을 위해 메모리 구조를 생성하여 사용합니다. 예를 들어, 메모리에는 실행 중인 프로그램 코드, 유저 공유 데이터 및 연결된 개별 유저에 대한 전용 데이터 영역이 저장됩니다. Instance와 연관된 두 가지 기본 메모리 구조는 다음과 같습니다.

- 시스템 글로벌 영역(SGA) - SGA 구성 요소라고 하는 공유 메모리 구조의 그룹으로 한 개의 오라클 데이터베이스 instance에 대한 데이터 및 제어 정보를 포함합니다. SGA는 모든 서버 프로세스와 백그라운드 프로세스에서 공유합니다. SGA에 저장된 데이터의 예로는 캐시에 저장된 데이터 블록 및 공유 SQL 영역이 있습니다.
- 프로그램 글로벌 영역(PGA) - 서버 프로세스 또는 백그라운드 프로세스에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. PGA는 서버 프로세스 또는 백그라운드 프로세스가 시작될 때 오라클 데이터베이스에서 생성하는 비공유 메모리입니다. 서버 프로세스만 PGA에 액세스할 수 있습니다. 서버 프로세스와 백그라운드 프로세스는 각각 자체의 PGA를 갖습니다.

Oracle 메모리 구조(계속)

SGA는 instance에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. SGA는 다음 데이터 구조를 포함합니다.

- **데이터베이스 버퍼 캐시:** 데이터베이스에서 검색된 데이터 블록을 캐시합니다.
- **리두 로그 버퍼:** instance recovery에 사용되는 리두 정보가 디스크에 저장된 물리적 리두 로그 파일에 기록될 때까지 해당 정보를 캐시합니다.
- **Shared Pool:** 유저 간에 공유할 수 있는 다양한 생성자를 캐시합니다.
- **Large Pool:** Oracle 백업 및 recovery 작업이나 입/출력(I/O) 서버 프로세스와 같은 특정 대규모 프로세스에 대한 대용량 메모리 할당을 제공하는 선택적 영역입니다.
- **Java Pool:** JVM(Java Virtual Machine) 내의 모든 세션별 Java 코드 및 데이터에 사용됩니다.
- **Streams Pool:** Oracle Streams에서 캡처 및 적용에 필요한 정보를 저장하는 데 사용됩니다.

Enterprise Manager 또는 SQL*Plus를 사용하여 instance를 시작하면 SGA에 대해 할당된 메모리 양이 표시됩니다.

동적 SGA infrastructure를 사용하면 instance를 종료하지 않고 데이터베이스 버퍼 캐시, shared pool, large pool, Java pool 및 streams pool의 크기를 변경할 수 있습니다.

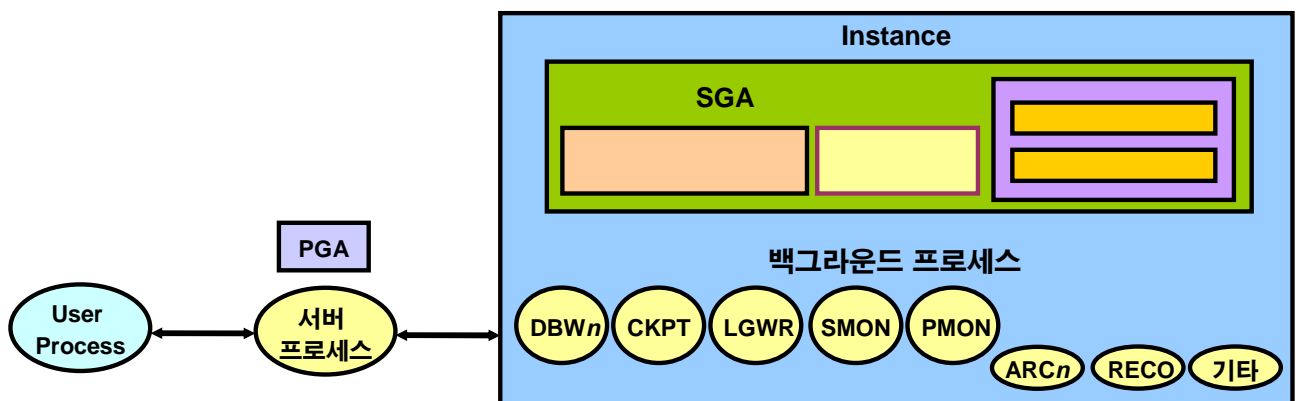
오라클 데이터베이스는 초기화 파라미터를 사용하여 메모리 구조를 생성하고 구성합니다. 예를 들어, SGA_TARGET 파라미터는 SGA 구성 요소의 총 크기를 지정합니다. SGA_TARGET을 0으로 설정하면 자동 공유 메모리 관리(Automatic Shared Memory Management)가 비활성화됩니다.

프로세스 구조

DB 구조

- 메모리
- 프로세스
- 저장 영역

- User process:
 - 데이터베이스 유저 또는 일괄 처리 프로세스가 오라클 데이터베이스에 연결될 때 시작됩니다.
- 데이터베이스 프로세스:
 - 서버 프로세스: Oracle instance에 연결하며, 유저가 세션을 설정할 때 시작됩니다.
 - 백그라운드 프로세스: Oracle instance가 시작될 때 시작됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로세스 구조

오라클 데이터베이스 서버의 프로세스는 다음 두 개의 주요 그룹으로 분류할 수 있습니다.

- 응용 프로그램 또는 Oracle 도구 코드를 실행하는 user process
- 오라클 데이터베이스 서버 코드를 실행하는 오라클 데이터베이스 프로세스. 여기에는 서버 프로세스와 백그라운드 프로세스가 해당됩니다.

유저가 응용 프로그램이나 Oracle 도구(예: SQL*Plus)를 실행하면 오라클 데이터베이스는 유저의 응용 프로그램을 실행하는 *user process*를 생성합니다. 오라클 데이터베이스는 *user process*에서 실행한 명령을 처리하는 *서버 프로세스*도 생성합니다. 또한 Oracle 서버는 *instance*에 대해 서로 상호 작용하거나 운영 체제와 상호 작용하는 일련의 *백그라운드 프로세스*를 포함하고 있습니다. 이러한 백그라운드 프로세스는 메모리 구조를 관리하고, 디스크에 데이터를 기록하기 위해 I/O를 비동기적으로 수행하고, 기타 필요한 작업을 수행하는 데 사용됩니다.

운영 체제와 선택한 오라클 데이터베이스 옵션에 따라 오라클 데이터베이스 구성마다 프로세스 구조가 다릅니다. 연결된 유저에 대한 코드는 *dedicated server* 또는 *shared server*로 구성할 수 있습니다.

- *Dedicated server*를 사용할 경우 각 유저에 대해 *user process*에서 데이터베이스 응용 프로그램을 실행하며 *user process*는 오라클 데이터베이스 서버 코드를 실행하는 *dedicated server 프로세스*에서 처리됩니다.
- *Shared server*를 사용하면 각 연결에 대해 *dedicated server 프로세스*를 실행할 필요가 없습니다. 디스패처가 여러 개의 수신 네트워크 세션 요청을 *shared server 프로세스* 풀로 전달합니다. *Shared server 프로세스*는 모든 클라이언트 요청을 처리합니다.

프로세스 구조(계속)

서버 프로세스

오라클 데이터베이스는 서버 프로세스를 생성하여 instance에 연결된 user process의 요청을 처리합니다. 응용 프로그램과 오라클 데이터베이스가 동일한 컴퓨터에서 작동하는 경우 user process와 해당 서버 프로세스를 단일 프로세스로 결합하여 시스템 오버헤드를 줄일 수 있습니다. 그러나 응용 프로그램과 오라클 데이터베이스가 다른 컴퓨터에서 작동할 경우 user process는 항상 별도의 서버 프로세스를 통해 오라클 데이터베이스와 통신합니다.

각 유저의 응용 프로그램 대신 생성된 서버 프로세스는 다음과 같은 작업을 수행할 수 있습니다.

- 응용 프로그램을 통해 실행된 SQL 문을 구문 분석하고 실행합니다.
- SGA에 데이터 블록이 없는 경우 디스크의 데이터 파일에서 SGA의 공유 데이터베이스 버퍼로 필요한 데이터 블록을 읽어 들입니다.
- 응용 프로그램이 정보를 처리할 수 있는 방식으로 결과를 반환합니다.

백그라운드 프로세스

성능을 최대화하고 여러 명의 유저를 수용하기 위해 다중 프로세스 오라클 데이터베이스 시스템에서는 백그라운드 프로세스라고 하는 몇 개의 추가 오라클 데이터베이스 프로세스를 사용합니다. 오라클 데이터베이스 instance는 여러 개의 백그라운드 프로세스를 사용할 수 있습니다.

데이터베이스 instance를 성공적으로 시작하는 데 필요한 백그라운드 프로세스는 다음과 같습니다.

- DBW_n(데이터베이스 기록자)
- LGWR(로그 기록자)
- 체크포인트(CKPT)
- SMON(시스템 모니터)
- PMON(프로세스 모니터)

다음 백그라운드 프로세스는 필요할 때 시작할 수 있는 선택적 백그라운드 프로세스에 대한 몇 가지 예입니다.

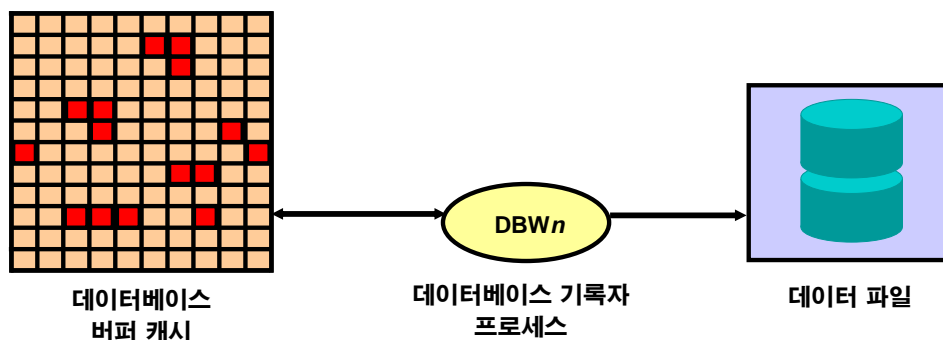
- 복구자(RECO)
- 작업 큐
- 아카이버(ARC_n)
- 큐 모니터(QMN_n)

다른 백그라운드 프로세스는 RAC(Real Application Cluster)와 같은 고급 구성에서 찾을 수 있습니다. 백그라운드 프로세스에 대한 자세한 내용은 V\$BGPROCESS 뷰를 참조하십시오. 대부분의 운영 체제에서 백그라운드 프로세스는 instance가 시작되면 자동으로 생성됩니다.

데이터베이스 기록자 프로세스

데이터베이스 버퍼 캐시의 수정된(더티) 버퍼를 디스크에 기록합니다.

- 다른 프로세싱을 수행하는 동안 비동기적으로
- 체크포인트를 전진시키기 위해 주기적으로



ORACLE

Copyright © 2009, Oracle. All rights reserved.

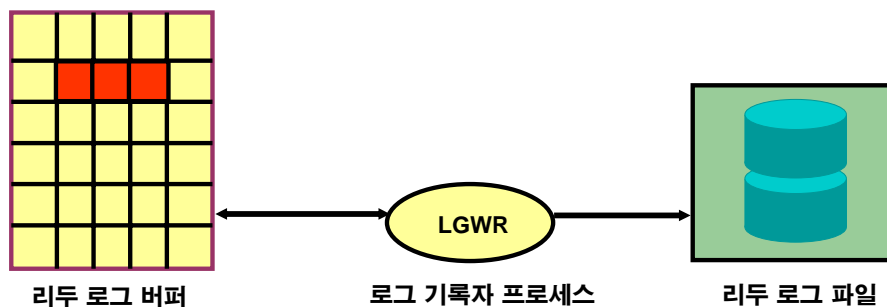
데이터베이스 기록자 프로세스

DBW_n(데이터베이스 기록자) 프로세스는 버퍼의 내용을 데이터 파일에 기록하며, 데이터베이스 버퍼 캐시의 수정된(더티) 버퍼를 디스크에 기록합니다. 대부분의 시스템에서는 하나의 데이터베이스 기록자 프로세스(DBW0)만 있으면 충분하지만 시스템에서 많은 데이터를 수정하는 경우 추가 프로세스(DBW1 - DBW9)를 구성하여 쓰기 성능을 향상시킬 수 있습니다. 단일 프로세서 시스템에서는 이러한 추가 DBW_n 프로세스가 유용하지 않습니다.

데이터베이스 버퍼 캐시의 버퍼가 수정된 경우 해당 버퍼는 "더티" 버퍼로 표시되고 시스템 변경 번호(SCN) 순서로 보관된 더티 버퍼의 LRUW 리스트에 추가됩니다. 따라서 이와 같이 변경된 버퍼에 해당하는, 리두 로그에 기록된 리두의 순서와 일치하게 됩니다. 버퍼 캐시에서 사용 가능한 버퍼의 수가 내부 임계값보다 적어서 서버 프로세스가 사용 가능한 버퍼를 확보하기 힘든 경우 DBW_n은 LRUW 리스트의 순서에 따라 수정된 순서대로 더티 버퍼를 데이터 파일에 기록합니다.

로그 기록자 프로세스

- 리두 로그 버퍼를 디스크의 리두 로그 파일에 기록합니다.
- LGWR은 다음 경우 기록합니다.
 - 프로세스가 트랜잭션을 커밋할 때
 - 리두 로그 버퍼의 1/3이 찼을 때
 - DBWn 프로세스가 수정된 버퍼를 디스크에 기록하기 전에



ORACLE

Copyright © 2009, Oracle. All rights reserved.

로그 기록자 프로세스

LGWR(로그 기록자) 프로세스는 리두 로그 버퍼 항목을 디스크의 리두 로그 파일에 기록하여 리두 로그 버퍼를 관리하며, 마지막으로 기록된 이후 버퍼로 복사된 모든 리두 항목을 기록합니다.

리두 로그 버퍼는 일종의 순환 버퍼이므로 LGWR이 리두 로그 버퍼의 리두 항목을 리두 로그 파일에 기록하면 서버 프로세스가 새 항목을 복사하여 디스크에 기록된 리두 로그 버퍼의 항목을 덮어 쓸 수 있습니다. LGWR은 일반적으로 빠르게 기록하여 리두 로그에 대한 액세스가 많을 때도 항상 새 항목을 위한 버퍼 공간을 확보해 둡니다.

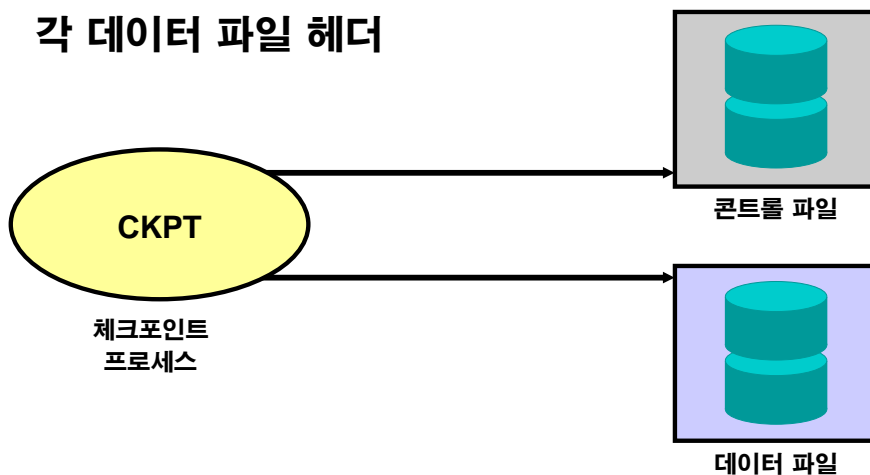
LGWR은 인접한 한 개의 버퍼 부분을 디스크에 기록합니다. LGWR은 다음 경우 기록합니다.

- User process가 트랜잭션을 커밋할 때
- 리두 로그 버퍼의 1/3이 찼을 때
- DBWn 프로세스가 수정된 버퍼를 디스크에 기록하기 전에(필요한 경우)

체크포인트 프로세스

체크포인트 정보를 읽는 위치:

- 컨트롤 파일
- 각 데이터 파일 헤더



ORACLE

Copyright © 2009, Oracle. All rights reserved.

체크포인트 프로세스

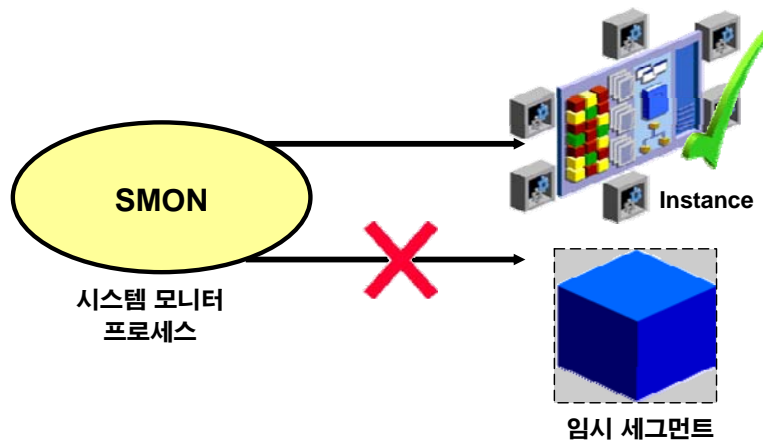
체크포인트는 데이터베이스의 리두 스레드에 SCN을 정의하는 데이터 구조로, 컨트롤 파일과 각 데이터 파일 헤더에 기록되며 매우 중요한 recovery 요소입니다.

체크포인트가 발생할 경우 오라클 데이터베이스는 모든 데이터 파일의 헤더를 갱신하여 체크포인트의 세부 사항을 기록해야 하는데, 이 작업은 CKPT 프로세스에 의해 수행됩니다. CKPT 프로세스는 블록을 디스크에 기록하지 않지만 DBWR은 항상 블록을 디스크에 기록합니다. 파일에 기록된 SCN은 해당 SCN이 디스크에 기록되기 전에 데이터베이스 블록에 대한 모든 변경이 이루어졌음을 나타냅니다.

Oracle Enterprise Manager에서 SYSTEM_STATISTICS 모니터에 의해 표시되는 정적 DBWR 체크포인트는 완료된 체크포인트 요청 수를 나타냅니다.

시스템 모니터 프로세스

- Instance 시작 시 recovery 수행
- 사용하지 않는 임시 세그먼트 정리



ORACLE

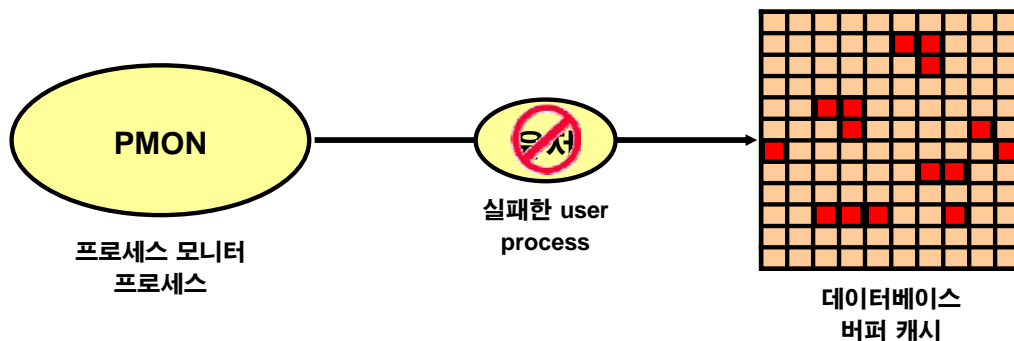
Copyright © 2009, Oracle. All rights reserved.

시스템 모니터 프로세스

SMON(시스템 모니터) 프로세스는 instance가 시작될 때 필요에 따라 recovery를 수행하며, 더 이상 사용하지 않는 임시 세그먼트도 정리합니다. 파일 읽기 또는 오프라인 오류로 인해 instance recovery 중 종료된 트랜잭션을 건너뛰는 경우 테이블스페이스나 파일이 다시 온라인으로 전환되면 SMON이 이를 recovery합니다. SMON은 정기적으로 필요 여부를 확인합니다. SMON이 필요할 경우 다른 프로세스에서 SMON을 호출할 수 있습니다.

프로세스 모니터 프로세스

- User process가 실패할 경우 프로세스 recovery 수행
 - 데이터베이스 버퍼 캐시 정리
 - User process에서 사용하는 리소스 해제
- Idle 세션 타임아웃에 대한 세션 모니터
- 리스너에 동적으로 데이터베이스 서비스 등록



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로세스 모니터 프로세스

PMON(프로세스 모니터)은 user process가 실패할 경우 프로세스 recovery를 수행하고, 데이터베이스 버퍼 캐시를 정리하며, user process에서 사용하고 있는 리소스를 해제합니다. 예를 들어, PMON은 활성 트랜잭션 테이블의 상태를 재설정하고, 잠금을 해제하며, 활성 프로세스 리스트에서 프로세스 ID를 제거합니다.

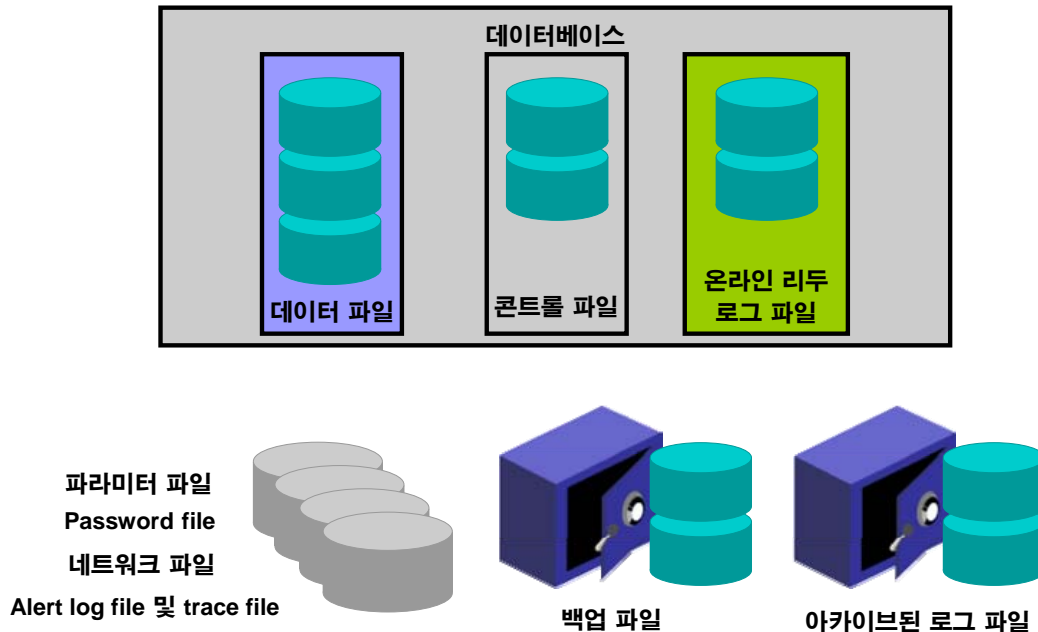
PMON은 정기적으로 디스패처와 서버 프로세스의 상태를 확인하여 실행이 정지된 경우(오라클 데이터베이스에서 의도적으로 종료한 경우 제외) 해당 디스패처나 서버 프로세스를 재시작합니다. 또한 instance 및 디스패처 프로세스에 대한 정보도 네트워크 리스너에 등록합니다.

SMON과 마찬가지로 PMON은 정기적으로 필요 여부를 확인하여 다른 프로세스에서 PMON을 필요로 할 경우 호출할 수 있습니다.

오라클 데이터베이스 저장 영역 구조

DB 구조

- 메모리
- 프로세스
- 저장 영역



ORACLE

Copyright © 2009, Oracle. All rights reserved.

오라클 데이터베이스 저장 영역 구조

오라클 데이터베이스는 다음과 같은 파일로 구성되어 있습니다.

- **컨트롤 파일:** 데이터베이스 자체에 대한 데이터(즉, 물리적 데이터베이스 구조 정보)를 포함합니다. 이 파일은 데이터베이스에 중요합니다. 이 파일이 없으면 데이터베이스 내의 데이터에 액세스할 때 데이터 파일을 열 수 없습니다.
- **데이터 파일:** 데이터베이스의 유저 또는 응용 프로그램 데이터, 메타 데이터 및 데이터 디렉토리를 포함합니다.
- **온라인 리두 로그 파일:** 데이터베이스의 instance recovery를 허용합니다. 데이터베이스 서버가 손상되었지만 해당 데이터 파일은 손실되지 않은 경우 instance는 이러한 파일 안에 있는 정보를 사용하여 데이터베이스를 recovery할 수 있습니다.

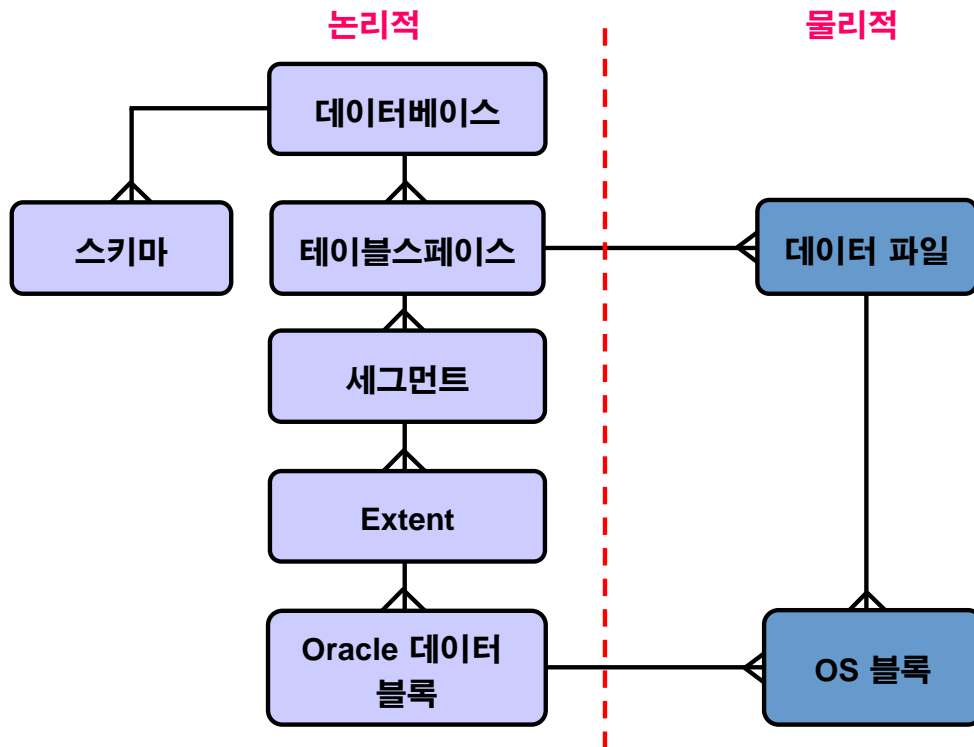
다음은 성공적인 데이터베이스 실행에 필요한 추가 파일입니다.

- **백업 파일:** 데이터베이스 recovery에 사용됩니다. 백업 파일은 일반적으로 Media Failure 또는 User Error로 원본 파일이 손상되었거나 삭제되었을 경우에 복원됩니다.
- **아카이브된 로그 파일:** instance에 의해 생성되는 데이터 변경(리두)에 대한 기록을 지속적으로 포함합니다. 이 파일과 데이터베이스 백업을 사용하면 손실된 데이터 파일을 recovery할 수 있습니다. 즉, 아카이브 로그는 복원된 데이터 파일의 recovery를 활성화합니다.
- **파라미터 파일:** instance 시작 시 어떻게 instance를 구성할지 정의하는 데 사용됩니다.
- **Password file:** sysdba/sysoper/sysasm이 데이터베이스에 원격으로 연결하여 관리 작업을 수행할 수 있도록 합니다.

오라클 데이터베이스 저장 영역 구조(계속)

- **네트워크 파일:** 데이터베이스 리스너를 시작하는 데 사용되며 유저 연결에 필요한 정보를 저장합니다.
- **Trace file:** 각 서버와 백그라운드 프로세스는 연관된 trace file에 정보를 기록할 수 있습니다. 내부 오류가 프로세스에서 감지되면 프로세스는 오류에 대한 정보를 해당 trace file에 덤프합니다. Trace file에 기록된 정보 중 일부는 데이터베이스 관리자가 사용하고 일부는 오라클 고객 지원 센터에서 사용합니다.
- **Alert log file:** 특수 trace 항목으로, 데이터베이스의 alert log에는 메시지와 오류가 시간순으로 기록되어 있습니다. Instance마다 한 개의 alert log file이 있습니다. 오라클은 이 alert log를 정기적으로 검토할 것을 권장합니다.

논리적 및 물리적 데이터베이스 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

논리적 및 물리적 데이터베이스 구조

오라클 데이터베이스는 논리적 저장 영역 구조와 물리적 저장 영역 구조를 갖습니다.

테이블스페이스

데이터베이스는 테이블스페이스라는 논리적 저장 영역 단위로 나뉘어 관련된 논리적 구조를 함께 그룹화합니다. 예를 들어, 테이블스페이스는 일반적으로 응용 프로그램의 모든 객체를 간단히 몇 개의 관리 작업으로 그룹화합니다. 응용 프로그램 데이터에 대한 테이블스페이스와 응용 프로그램 인덱스에 대한 추가 테이블스페이스를 가질 수 있습니다.

데이터베이스, 테이블스페이스 및 데이터 파일

위의 슬라이드에는 데이터베이스, 테이블스페이스 및 데이터 파일 간의 관계가 나와 있습니다. 각 데이터베이스는 논리적으로 하나 이상의 테이블스페이스로 나뉩니다. 각 테이블스페이스에는 테이블스페이스에 있는 모든 논리적 구조 데이터를 물리적으로 저장할 수 있는 하나 이상의 데이터 파일이 명시적으로 생성됩니다. TEMPORARY 테이블스페이스의 경우 데이터 파일 대신 테이블스페이스에 임시 파일이 있습니다.

논리적 및 물리적 데이터베이스 구조(계속)

스키마

스키마는 데이터베이스 사용자가 소유하는 데이터베이스 객체의 모음입니다. 스키마 객체는 데이터베이스의 데이터를 직접 참조하는 논리적 구조입니다. 스키마 객체에는 테이블, 뷰, 시퀀스, 내장 프로시저, 동의어, 인덱스, 클러스터 및 데이터베이스 링크가 있습니다. 일반적으로 스키마 객체에는 응용 프로그램이 데이터베이스에 생성하는 모든 것이 포함됩니다.

데이터 블록

가장 작은 세분성 레벨에서 오라클 데이터베이스의 데이터는 데이터 블록에 저장됩니다. 하나의 데이터 블록은 디스크에서 특정 바이트 수의 물리적 데이터베이스 공간에 해당합니다. 데이터 블록 크기는 생성 시 각 테이블스페이스에 대해 지정됩니다. 데이터베이스는 사용 가능한 데이터베이스 공간을 Oracle 데이터 블록으로 사용하고 할당합니다.

Extent

그 다음 레벨의 논리적 데이터베이스 공간은 extent입니다. Extent는 단일 할당으로 얻은 일정 수의 연속적인 데이터 블록으로, 특정 유형의 정보를 저장하는 데 사용됩니다.

세그먼트

Extent 다음 레벨의 논리적 데이터베이스 저장 영역은 세그먼트입니다. 세그먼트는 특정 논리적 구조에 할당된 일련의 extent입니다. 예를 들어, 다음과 같은 다양한 유형의 세그먼트가 있습니다.

- **데이터 세그먼트:** 클러스터화되지 않은 각각의 비인덱스 구성(non-index-organized) 테이블에는 데이터 세그먼트가 있습니다. 단, external table, 전역 임시 테이블(global temporary table) 및 partition 테이블은 예외입니다. 이 경우 테이블마다 한 개 이상의 세그먼트가 있습니다. 테이블 데이터는 모두 해당 데이터 세그먼트의 extent에 저장됩니다. Partition 테이블의 경우 각 partition은 데이터 세그먼트를 가집니다. 각 클러스터는 데이터 세그먼트를 가집니다. 클러스터에 있는 모든 테이블의 데이터는 클러스터의 데이터 세그먼트에 저장됩니다.
- **인덱스 세그먼트:** 각 인덱스는 해당 데이터를 모두 저장하는 인덱스 세그먼트를 가집니다. Partition 인덱스의 경우 각 partition은 인덱스 세그먼트를 가집니다.
- **언두 세그먼트:** 일시적으로 언두 정보를 저장하기 위해 수많은 언두 세그먼트를 포함하는 UNDO 테이블스페이스가 데이터베이스 instance당 한 개씩 생성됩니다. 언두 세그먼트의 정보는 데이터베이스 recovery 중 유저에게 커밋되지 않은 트랜잭션을 롤백하기 위해 읽기 일관성 데이터베이스 정보를 생성하는 데 사용됩니다.
- **임시 세그먼트:** 임시 세그먼트는 SQL 문에서 실행을 완료할 임시 작업 영역이 필요할 때 오라클 데이터베이스에 의해 생성됩니다. 명령문의 실행이 완료되면 나중에 사용할 수 있도록 임시 세그먼트의 extent가 instance로 반환됩니다. 모든 유저에 대해 기본 임시 테이블스페이스를 지정하거나 데이터베이스 차원에서 사용할 기본 임시 테이블스페이스를 지정하십시오.

오라클 데이터베이스는 동적으로 공간을 할당합니다. 세그먼트의 기존 extent가 가득 차면 다른 extent가 추가됩니다. Extent는 필요에 따라 할당되므로 세그먼트의 extent는 디스크에서 인접해 있을 수도 있고 그렇지 않을 수도 있습니다.

SQL 문 처리

- 다음을 사용하여 instance에 연결합니다.
 - User process
 - 서버 프로세스
- 사용되는 Oracle 서버 구성 요소는 SQL 문 유형에 따라 다릅니다.
 - Query는 행을 반환합니다.
 - DML(데이터 조작어) 문은 변경 사항을 기록합니다.
 - 커밋은 트랜잭션 recovery를 보장합니다.
- 일부 Oracle 서버 구성 요소는 SQL 문 처리에 관여하지 않습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문 처리

Oracle instance의 모든 구성 요소가 SQL 문 처리에 사용되는 것은 아닙니다. User processes와 서버 프로세스는 유저를 Oracle instance에 연결하는 데 사용됩니다. 이러한 프로세스는 Oracle instance에 속하지는 않지만 SQL 문을 처리하는 데 반드시 필요합니다.

일부 백그라운드 프로세스, SGA 구조 및 데이터베이스 파일도 SQL 문을 처리하는 데 사용됩니다. SQL 문의 유형에 따라 다른 구성 요소가 사용됩니다.

- Query에는 유저에게 행을 반환하기 위한 추가 처리가 필요합니다.
- DML 문에는 데이터의 변경 사항을 기록하기 위한 추가 처리가 합니다.
- 커밋 프로세스는 트랜잭션의 수정된 데이터가 recovery될 수 있도록 보장합니다.

일부 필수 백그라운드 프로세스는 SQL 문 처리에 직접 관여하지는 않지만 성능 개선 및 데이터베이스 recovery에 사용됩니다. 예를 들어, 선택적 아카이버 백그라운드 프로세스인 ARCn을 사용하면 운용 중인 데이터베이스를 recovery할 수 있습니다.

Query 처리

- **구문 분석:**
 - 동일한 명령문을 검색합니다.
 - 구문, 객체 이름 및 권한을 검사합니다.
 - 구문 분석 중 사용된 객체를 잠급니다.
 - 실행 계획을 생성하고 저장합니다.
- **실행: 선택된 행을 식별합니다.**
- **패치(fetch): User process로 행을 반환합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Query 처리

Query는 성공할 경우 데이터를 결과로 반환한다는 점에서 다른 SQL 문 유형과 다릅니다. 다른 명령문이 단순히 성공 또는 failure를 반환하는 반면 query는 한 행이나 수천 개의 행을 반환할 수 있습니다.

Query 처리의 주요 세 단계는 다음과 같습니다.

- 구문 분석
- 실행
- 패치(fetch)

구문 분석 단계에서는 SQL 문이 user process에서 서버 프로세스로 전달되고 구문 분석된 SQL 문이 공유 SQL 영역으로 로드됩니다.

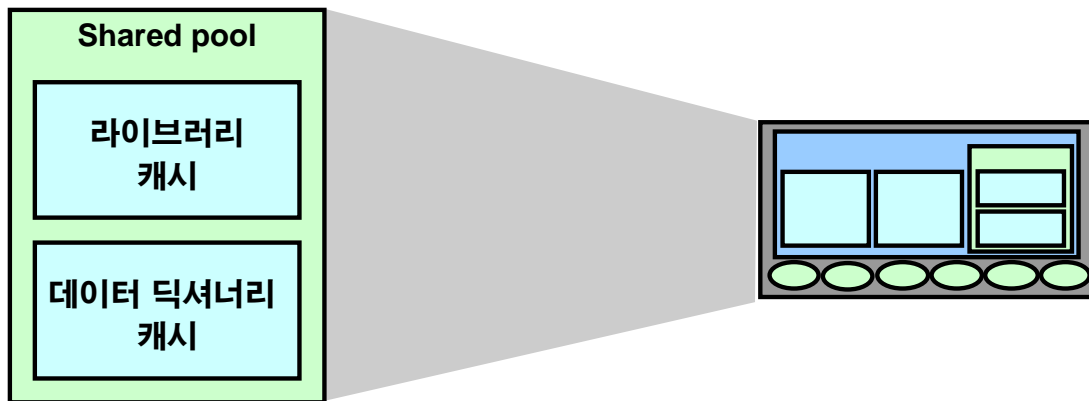
구문 분석 중 서버 프로세스는 다음 기능을 수행합니다.

- Shared pool에서 기존의 SQL 문 복사본을 검색합니다.
- 구문을 확인하여 SQL 문을 검증합니다.
- 데이터 디렉터리 조회를 실행하여 테이블 및 열 정의를 검증합니다.

실행 단계는 최적의 옵티마이저 접근 방식을 사용하여 명령문을 실행하고 패치(fetch)는 유저에게 다시 행을 가져옵니다.

Shared Pool

- 라이브러리 캐시는 SQL 문 텍스트, 구문 분석된 코드 및 실행 계획을 포함합니다.
- 데이터 디렉터리 캐시는 테이블, 열 및 기타 객체 정의 및 권한을 포함합니다.
- Shared pool은 SHARED_POOL_SIZE에 의해 크기가 조정됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Shared Pool

구문 분석 단계에서 서버 프로세스는 shared pool이라는 SGA의 영역을 사용하여 SQL 문을 컴파일합니다. Shared pool은 다음 두 가지 기본 구성 요소를 가집니다.

- 라이브러리 캐시
- 데이터 디렉터리 캐시

라이브러리 캐시

라이브러리 캐시는 공유 SQL 영역이라는 메모리 구조에 가장 최근에 사용된 SQL 문에 대한 정보를 저장합니다. 공유 SQL 영역은 다음을 포함합니다.

- SQL 문 텍스트
- 구문 분석 트리: 명령문의 컴파일된 버전
- 실행 계획: 명령문을 실행할 때 수행할 단계

옵티마이저는 최적의 실행 계획을 결정하는 Oracle 서버의 기능입니다.

SQL 문이 재실행되고 공유 SQL 영역이 이미 SQL 문에 대한 실행 계획을 포함하고 있을 경우 서버 프로세스는 SQL 문의 구문을 분석할 필요가 없습니다. 라이브러리 캐시는 구문 분석 시간과 메모리 요구 사항을 줄임으로써 SQL 문을 재사용하는 응용 프로그램의 성능을 향상시킵니다. SQL 문이 재사용되지 않을 경우 SQL 문은 결국 라이브러리 캐시에서 삭제됩니다.

Shared Pool(계속)

데이터 디렉터리 캐시

데이터 디렉터리 캐시(디렉터리 캐시 또는 행 캐시라고도 함)는 데이터베이스에서 가장 최근에 사용된 정의의 모음입니다. 데이터베이스 파일, 테이블, 인덱스, 열, 유저, 권한 및 기타 데이터베이스 객체에 대한 정보를 포함합니다.

구문 분석 단계에서 서버 프로세스는 디렉터리 캐시에서 정보를 찾아 SQL 문에 지정된 객체 이름을 분석하고 액세스 권한을 검증합니다. 필요한 경우 서버 프로세스는 데이터 파일에서 이 정보의 로드를 시작합니다.

Shared Pool 크기 조정

Shared pool의 크기는 초기화 파라미터 SHARED_POOL_SIZE에 의해 지정됩니다.

데이터베이스 버퍼 캐시

- 데이터베이스 버퍼 캐시는 가장 최근에 사용된 블록을 저장합니다.
- 버퍼 크기는 DB_BLOCK_SIZE를 기반으로 합니다.
- 버퍼 수는 DB_BLOCK_BUFFERS에 의해 정의됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터베이스 버퍼 캐시

Query를 처리할 때 서버 프로세스는 데이터베이스 버퍼 캐시에서 필요한 블록을 찾습니다. 데이터베이스 버퍼 캐시에서 블록이 발견되지 않을 경우 서버 프로세스는 데이터 파일에서 블록을 읽고 버퍼 캐시에 복사본을 배치합니다. 동일한 블록에 대해 발생하는 이후의 요청은 이 블록을 메모리에서 찾을 수 있기 때문에 물리적 읽기를 수행할 필요가 없습니다. Oracle 서버는 LRU(Least Recently Used) 알고리즘을 사용하여 최근에 액세스하지 않은 버퍼를 삭제함으로써 버퍼 캐시에 새 블록을 위한 공간을 만듭니다.

데이터베이스 버퍼 캐시 크기 조정

버퍼 캐시 내 각 버퍼의 크기는 Oracle 블록의 크기와 같고 DB_BLOCK_SIZE 파라미터에 의해 지정됩니다. 버퍼 수는 DB_BLOCK_BUFFERS 파라미터의 값과 같습니다.

프로그램 글로벌 영역(PGA)

- 공유되지 않습니다.
- 서버 프로세스에 의해서만 쓰기가 가능합니다.
- 다음을 포함합니다.
 - 정렬 영역
 - 세션 정보
 - Cursor State
 - Stack Space



ORACLE

Copyright © 2009, Oracle. All rights reserved.

프로그램 글로벌 영역(PGA)

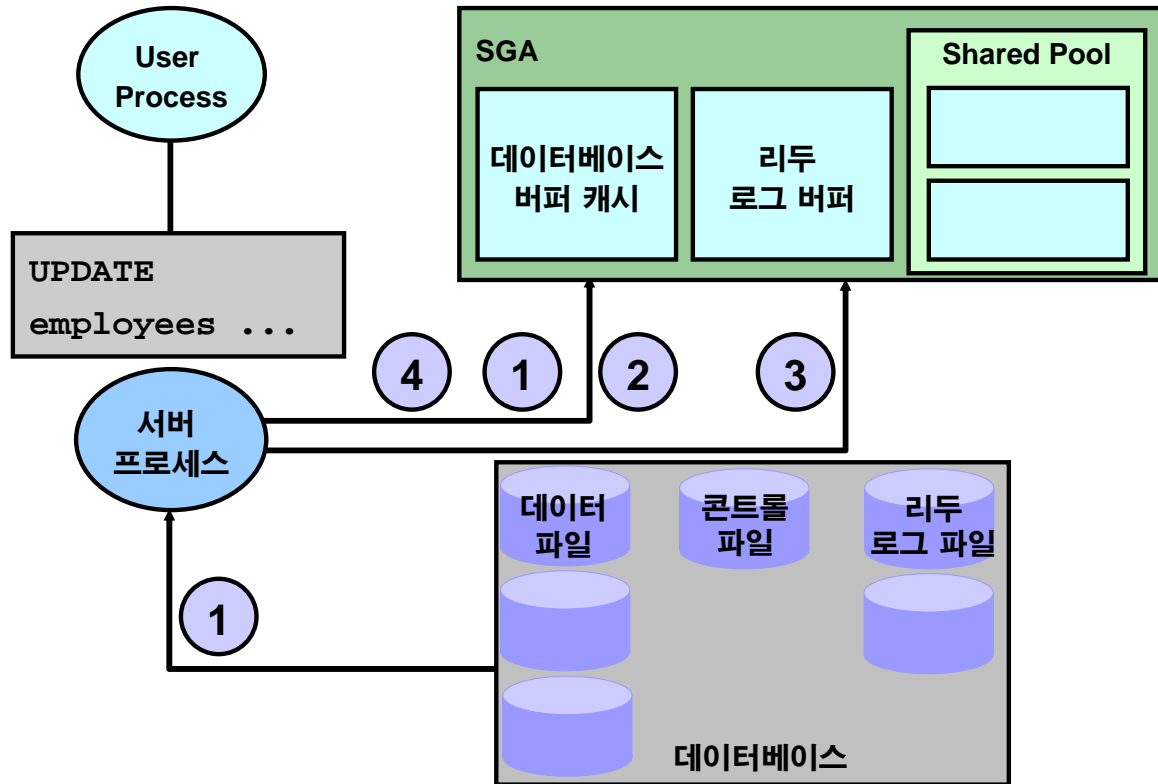
프로그램 글로벌 영역(PGA)은 서버 프로세스에 대한 데이터 및 제어 정보를 포함하는 메모리 영역입니다. 이것은 서버 프로세스가 시작될 때 오라클에서 생성하는 비공유 메모리입니다. PGA에는 해당 서버 프로세스만 액세스할 수 있으며, PGA를 대신하여 작동하는 Oracle 서버 코드로만 읽고 쓸 수 있습니다. Oracle instance에 연결된 각 서버 프로세스에 의해 할당된 PGA 메모리는 instance에 의해 할당된 집계 PGA 메모리라고도 합니다.

Dedicated server 구성에서 서버의 PGA는 다음 구성 요소를 포함합니다.

- **정렬 영역:** SQL 문을 처리하는 데 필요한 정렬에 사용됩니다.
- **세션 정보:** 세션에 대한 유저 권한 및 성능 통계를 포함합니다.
- **Cursor state:** 현재 세션에 의해 사용되는 SQL 문을 처리하는 단계를 나타냅니다.
- **Stack space:** 다른 세션 변수를 포함합니다.

PGA는 프로세스가 생성될 때 할당되고 프로세스가 종료될 때 할당이 해제됩니다.

DML 문 처리



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 문 처리

DML(데이터 조작어) 문을 처리하는 데는 구문 분석과 실행이라는 두 단계만 필요합니다.

- 구문 분석은 query를 처리하는 데 사용되는 구문 분석 단계와 동일합니다.
- 실행에는 데이터 변경을 위한 추가 처리가 필요합니다.

DML 실행 단계

DML 문을 실행하려면 다음을 수행합니다.

- 데이터 및 롤백 블록이 버퍼 캐시에 없는 경우 서버 프로세스가 이러한 블록을 데이터 파일에서 읽어 버퍼 캐시로 복사합니다.
- 서버 프로세스는 수정할 행을 잠급니다.
- 리두 로그 버퍼에서 서버 프로세스가 롤백 및 데이터 블록에 대한 변경 사항을 기록합니다.
- 롤백 블록 변경은 데이터가 수정되기 전에 데이터 값을 기록합니다. 롤백 블록은 필요한 경우 DML 문이 롤백될 수 있도록 데이터의 "이전 이미지"를 저장하는 데 사용됩니다.
- 데이터 블록 변이 새로운 데이터 값을 기록합니다.

SQL 문 처리(계속)

서버 프로세스는 "이전 이미지"를 롤백 블록에 기록하고 데이터 블록을 갱신합니다. 이 두 가지 변경은 모두 데이터베이스 버퍼 캐시에서 이루어집니다. 버퍼 캐시에서 변경된 블록은 더티 버퍼, 즉 디스크에 있는 해당 블록과 동일하지 않은 버퍼로 표시됩니다.

DELETE 또는 INSERT 명령의 처리는 유사한 단계를 사용합니다. DELETE에 대한 "이전 이미지"는 삭제된 행에 있는 열 값을 포함하며, INSERT의 이전 이미지는 행 위치 정보를 포함합니다.

블록에서 변경된 사항은 메모리 구조에만 기록되고 즉시 디스크에 쓰여지지 않기 때문에, SGA의 손실을 가져오는 컴퓨터 failure가 발생할 경우에는 이러한 변경 사항도 손실될 수 있습니다.

리두 로그 버퍼

- LOG_BUFFER에 의해 정의된 크기를 가집니다.
- Instance를 통해 변경된 사항을 기록합니다.
- 연속적으로 사용됩니다.
- 순환 버퍼입니다.



ORACLE

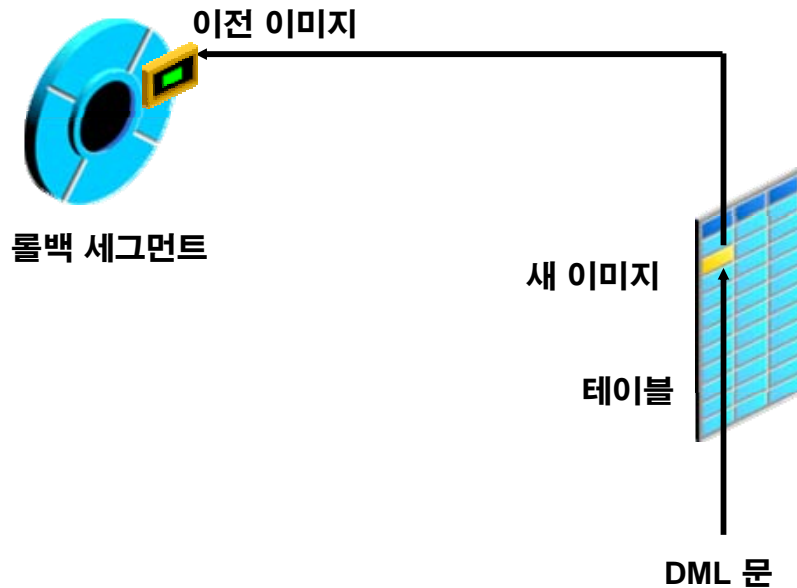
Copyright © 2009, Oracle. All rights reserved.

리두 로그 버퍼

서버 프로세스는 SGA의 일부인 리두 로그 버퍼에 데이터 파일 블록에서 변경된 사항을 대부분 기록합니다. 리두 로그 버퍼는 다음과 같은 특성을 가집니다.

- 바이트 단위의 크기는 LOG_BUFFER 파라미터에 의해 정의됩니다.
- 리두 항목에 변경된 블록, 변경 위치 및 새로운 값을 기록합니다. 리두 항목은 변경되는 블록에서 유형에 차이가 없으며, 단지 블록에서 변경된 바이트를 기록합니다.
- 리두 로그 버퍼는 연속적으로 사용되어 한 트랜잭션의 변경 사항을 다른 트랜잭션의 변경 사항에 끼울 수 있습니다.
- 버퍼가 다 차면 재사용되는 순환 버퍼입니다. 단, 모든 이전 리두 항목이 리두 로그 파일에 기록된 후에야 재사용할 수 있습니다.

롤백 세그먼트



ORACLE

Copyright © 2009, Oracle. All rights reserved.

롤백 세그먼트

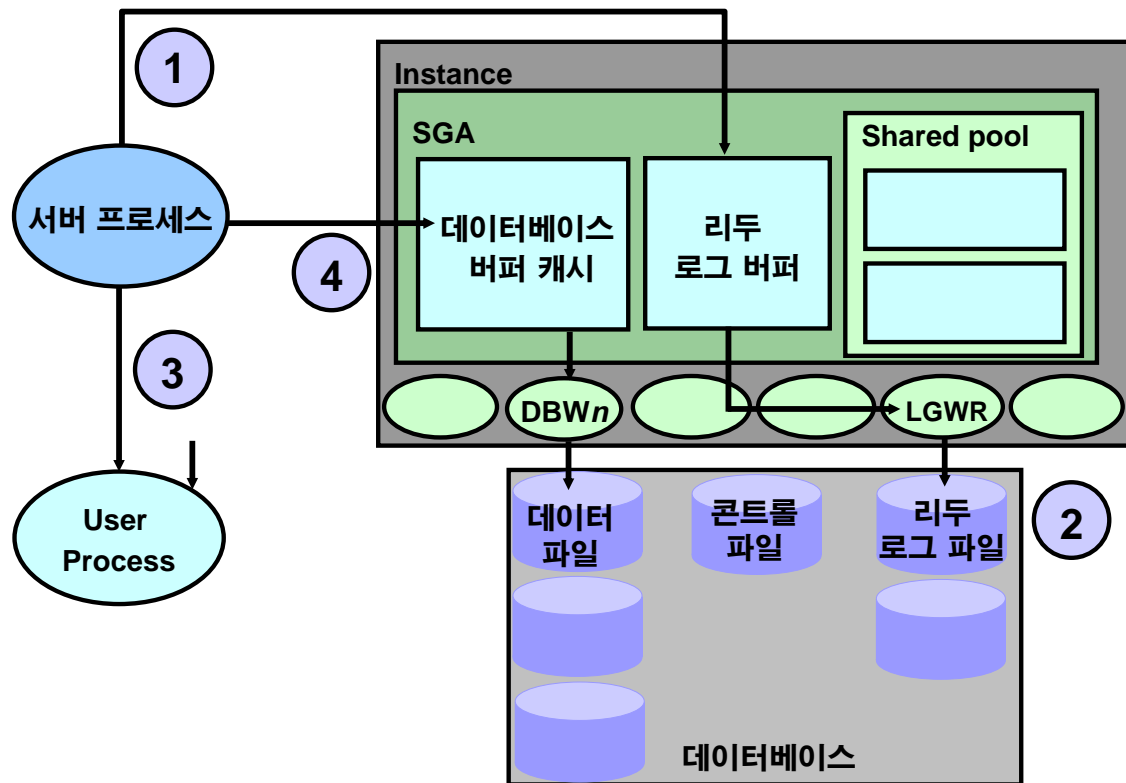
내용을 변경하기 전 서버 프로세스는 이전 데이터 값을 롤백 세그먼트에 저장합니다. 이 "이전 이미지"는 다음 작업에 사용됩니다.

- 트랜잭션이 롤백될 경우 변경 언두
- DML 문으로 수행한 커밋되지 않은 변경 사항을 다른 트랜잭션이 보지 못하도록 하여 읽기 일관성 제공
- Failure 발생 시 데이터베이스를 일관된 상태로 recovery

테이블 및 인덱스와 같은 롤백 세그먼트는 데이터 파일에 존재하며 롤백 블록은 필요에 따라 데이터베이스 버퍼 캐시로 유입됩니다. 롤백 세그먼트는 DBA에 의해 생성됩니다.

롤백 세그먼트에 대한 변경 사항은 리두 로그 버퍼에 기록됩니다.

COMMIT 처리



ORACLE

Copyright © 2009, Oracle. All rights reserved.

COMMIT 처리

Oracle 서버는 instance failure 발생 시 커밋된 변경 사항이 recovery될 수 있도록 하는 빠른 COMMIT 방식을 사용합니다.

시스템 변경 번호

트랜잭션이 커밋할 때마다 Oracle 서버는 트랜잭션에 커밋 SCN을 할당합니다. SCN은 일정하게 증가하며 데이터베이스 내에서 고유합니다. Oracle 서버는 SCN을 내부 시간 기록으로 사용하여 데이터를 동기화하고 데이터 파일에서 데이터를 검색할 때 읽기 일관성을 제공합니다. SCN을 사용하면 Oracle 서버는 운영 체제의 날짜와 시간에 관계없이 일관성 검사를 수행할 수 있습니다.

COMMIT 처리 단계

COMMIT이 실행될 때 다음 단계가 수행됩니다.

1. 서버 프로세스는 SCN과 함께 커밋 기록을 리두 로그 버퍼에 배치합니다.
2. LGWR은 커밋 레코드를 포함하여 모든 리두 로그 버퍼 항목을 리두 로그 파일에 연속적으로 기록합니다. 이 시점부터 Oracle 서버는 instance failure가 발생할 경우에도 변경 사항이 손실되지 않도록 보장할 수 있습니다.

커밋 처리(계속)

3. 유저는 커밋이 완료되었다는 통지를 받습니다.
4. 서버 프로세스는 트랜잭션이 완료되었고 자원 잠금이 해제될 수 있음을 나타내는 정보를 기록합니다.

더티 버퍼를 데이터 파일에 비우는 작업은 DBW0에 의해 독립적으로 수행되며 커밋 이전 또는 이후에 이루어질 수 있습니다.

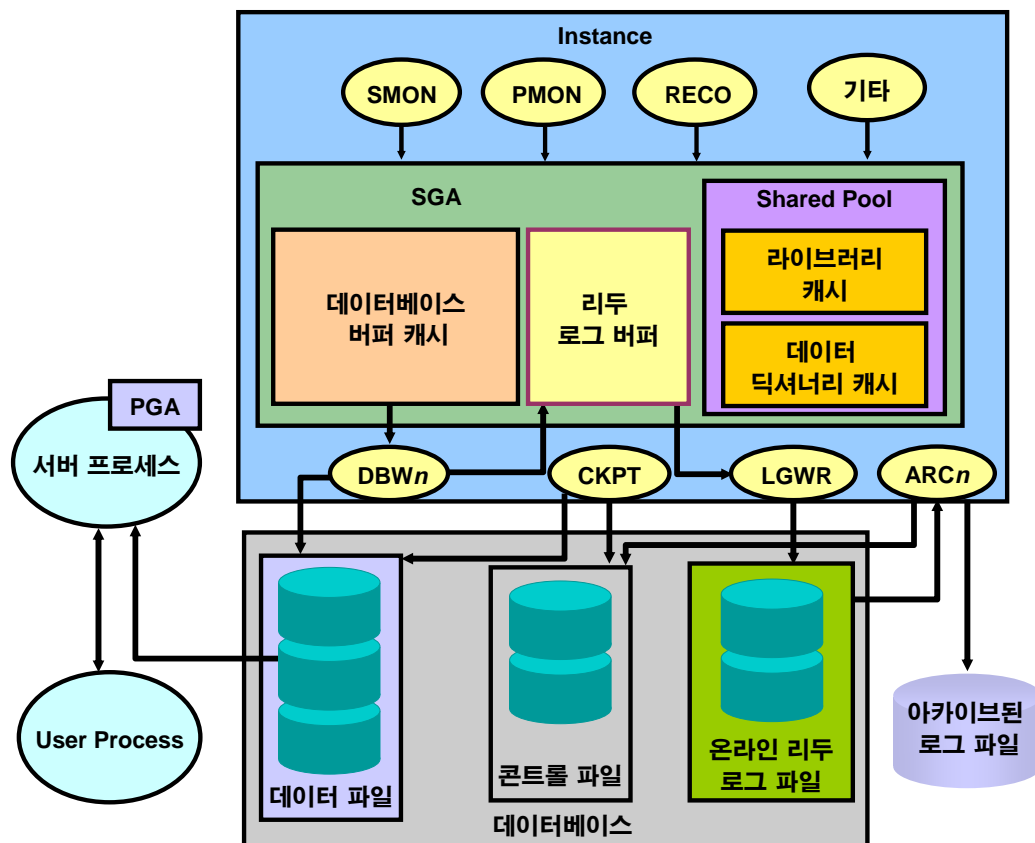
빠른 커밋의 이점

빠른 커밋 방식은 데이터 파일 대신 리두 로그 버퍼에 변경 사항을 기록함으로써 데이터 recovery를 보장합니다. 이 방식의 이점은 다음과 같습니다.

- 로그 파일에 순차적으로 쓰는 것이 데이터 파일의 다른 블록에 쓰는 것보다 속도가 빠릅니다.
- 로그 파일에는 변경 사항을 기록하는 데 필요한 최소 정보만 기록되는 반면, 데이터 파일에는 전체 데이터 블록이 기록되어야 합니다.
- 여러 트랜잭션이 동시에 커밋을 요청할 경우 instance는 리두 로그 레코드를 한 번의 쓰기로 피기백합니다.
- 리두 로그 버퍼가 특별히 가득 차지 않는 한, 트랜잭션당 한 번만 동기적으로 쓰면 됩니다. 피기백이 이루어질 경우 트랜잭션당 동기 쓰기 횟수는 한 번 미만일 수 있습니다.
- 리두 로그 버퍼는 커밋 이전에 비워질 수 있기 때문에 트랜잭션의 크기는 실제 커밋 연산에 필요한 시간에 영향을 미치지 않습니다.

참고: 트랜잭션을 롤백해도 LGWR은 디스크 쓰기를 시작하지 않습니다. Oracle 서버는 항상 failure recovery 시 커밋되지 않은 변경 사항을 롤백합니다. 롤백 후 롤백 항목이 디스크에 기록되기 전에 failure가 발생할 경우 커밋 레코드가 없으면 트랜잭션에 의해 변경된 사항은 롤백됩니다.

오라클 데이터베이스 구조 요약



ORACLE

Copyright © 2009, Oracle. All rights reserved.

오라클 데이터베이스 구조 요약

오라클 데이터베이스는 instance와 관련 데이터베이스로 구성됩니다.

- Instance는 SGA와 백그라운드 프로세스로 구성됩니다.
 - **SGA:** 데이터베이스 버퍼 캐시, 리두 로그 버퍼, shared pool 등
 - **백그라운드 프로세스:** SMON, PMON, DBWn, CKPT, LGWR 등
- 데이터베이스는 저장 영역 구조로 구성됩니다.
 - **논리적:** 테이블스페이스, 스키마, 세그먼트, extent 및 Oracle 블록
 - **물리적:** 데이터 파일, 컨트롤 파일, 리두 로그 파일

유저가 응용 프로그램을 통해 오라클 데이터베이스에 액세스하면 user process 대신 서버 프로세스가 instance와 통신합니다.

추가 연습 및 해답

목차

추가 연습	3
추가 연습	4
추가 연습: 사례 연구	10
추가 연습 문제 해답	13
추가 연습 문제 해답	14
추가 연습: 사례 연구 해답	20

다음 연습은 "스키마 객체 관리" 및 "대형 데이터 집합 조작" 단원의 DML(데이터 조작어) 문과 DDL(데이터 정의어) 문에 대해 살펴본 후 추가로 수행할 수 있습니다.

참고: SPECIAL_SAL, SAL_HISTORY 및 MGR_HISTORY 테이블을 생성하려면 labs 폴더에서 lab_ap_cre_special_sal.sql, lab_ap_cre_sal_history.sql 및 lab_ap_cre_mgr_history.sql 스크립트를 실행하십시오.

추가 연습

- 1) Human Resources 부서에서 산업별 임금 실태 조사를 기반으로 저임금 사원, 사원의 급여 내역 및 관리자의 급여 내역 리스트를 만들려고 합니다. 이에 따라 다음과 같은 작업을 요청했습니다.

다음 작업을 수행하는 명령문을 작성합니다.

- EMPLOYEES 테이블에서 사원 ID가 200보다 크거나 같은 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID 등의 세부 정보를 검색합니다.
- 급여가 \$5,000 미만일 경우 사원 ID, 급여 등의 세부 정보를 SPECIAL_SAL 테이블에 삽입합니다.
- 사원 ID, 채용 날짜, 급여 등의 세부 정보를 SAL_HISTORY 테이블에 삽입합니다.
- 사원 ID, 관리자 ID, 급여 등의 세부 정보를 MGR_HISTORY 테이블에 삽입합니다.

- 2) SPECIAL_SAL, SAL_HISTORY 및 MGR_HISTORY 테이블을 query하여 삽입된 레코드를 확인합니다.

SPECIAL_SAL

	EMPLOYEE_ID	SALARY
1	200	4400

SAL_HISTORY

	EMPLOYEE_ID	HIRE_DATE	SALARY
1	201	17-FEB-1996	13000
2	202	17-AUG-1997	6000
3	203	07-JUN-1994	6500
4	204	07-JUN-1994	10000
5	205	07-JUN-1994	12000
6	206	07-JUN-1994	8300

MGR_HISTORY

	EMPLOYEE_ID	MANAGER_ID	SALARY
	201	100	13000
	202	201	6000
	203	101	6500
	204	101	10000
	205	101	12000
	206	205	8300

추가 연습(계속)

- 3) DBA인 Nita가 테이블 생성을 요청합니다. 이 테이블에는 **Primary Key** 제약 조건이 있지만 인덱스 이름을 제약 조건과 다르게 지정하려고 합니다. 다음 테이블 instance 차트에 준하여 LOCATIONS_NAMED_INDEX 테이블을 생성합니다. PRIMARY KEY 열의 인덱스 이름을 LOCATIONS_PK_IDX로 지정합니다.

열 이름	Deptno	Dname
Primary Key	있음	
데이터 유형	Number	VARCHAR2
길이	4	30

- 4) USER_INDEXES 테이블을 query하여 LOCATIONS_NAMED_INDEX 테이블에 대한 INDEX_NAME을 표시합니다.

INDEX_NAME	TABLE_NAME
1 LOCATIONS_PK_IDX	LOCATIONS_NAMED_INDEX

추가 연습(계속)

다음 연습은 `datetime` 함수를 살펴본 후 추가로 수행할 수 있습니다.

글로벌 회사에서 새로 취임한 부사장이 모든 지사의 각 시간대를 파악하려고 합니다. 새 부사장이 다음과 같은 정보를 요청했습니다.

5) 세션을 변경하여 `NLS_DATE_FORMAT`을 `DD-MON-YYYY HH24:MI:SS`로 설정합니다.

6) a) 다음 시간대에 대해 시간대 오프셋(`TZ_OFFSET`)을 표시하는 query를 작성합니다.

– Australia/Sydney

	<code>TZ_OFFSET('AUSTRALIA/SYDNEY')</code>
1	+10:00

– Chile/Easter Island

	<code>TZ_OFFSET('CHILE/EASTERISLAND')</code>
1	-06:00

b) 세션을 변경하여 `TIME_ZONE` 파라미터 값을 Australia/Sydney의 시간대 오프셋으로 설정합니다.

c) 이 세션에 대해 `SYSDATE`, `CURRENT_DATE`, `CURRENT_TIMESTAMP` 및 `LOCALTIMESTAMP`를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

	<code>SYSDATE</code>	<code>CURRENT_DATE</code>	<code>CURRENT_TIMESTAMP</code>	<code>LOCALTIMESTAMP</code>
1	02-JUL-2009 17:11:46	02-JUL-2009 20:11:46	02-JUL-09 08.11.46.000000000 PM +10:00	02-JUL-09 08.11.46.000000000 PM

d) 세션을 변경하여 `TIME_ZONE` 파라미터 값을 Chile/Easter Island의 시간대 오프셋으로 설정합니다.

참고: 앞 질문의 결과는 다른 날짜에 준한 것이며 일부 수강생의 실제 결과와 다를 수 있습니다. 또한 시간대 오프셋은 일광 절약 시간에 따라 국가마다 다를 수 있습니다.

e) 이 세션에 대해 `SYSDATE`, `CURRENT_DATE`, `CURRENT_TIMESTAMP` 및 `LOCALTIMESTAMP`를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

	<code>SYSDATE</code>	<code>CURRENT_DATE</code>	<code>CURRENT_TIMESTAMP</code>	<code>LOCALTIMESTAMP</code>
1	02-JUL-2009 17:12:33	02-JUL-2009 04:12:33	02-JUL-09 04.12.33.000000000 AM -06:00	02-JUL-09 04.12.33.000000000 AM

추가 연습(계속)

f) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY로 설정합니다.

참고

- 앞의 질문에서 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP는 모두 세션 시간대의 영향을 받습니다. SYSDATE는 세션 시간대의 영향을 받지 않습니다.
- 앞 질문의 결과는 다른 날짜에 준한 것이며 일부 수강생의 실제 결과와 다를 수 있습니다. 또한 시간대 오프셋은 일광 절약 시간에 따라 국가마다 다를 수 있습니다.

7) Human Resources 부서에서 1월에 입사한 사원 리스트를 검토하려고 합니다. 이에 따라 다음과 같은 작업을 요청했습니다.

입사 연도에 관계없이 1월에 입사한 사원들의 성, 채용 월, 채용 날짜를 표시하는 query를 작성합니다.

	LAST_NAME	EXTRACT(MONTHFROMHIRE_DATE)	HIRE_DATE
1	Grant		13-JAN-2000
2	De Haan		13-JAN-1993
3	Hunold		03-JAN-1990
4	Landry		14-JAN-1999
5	Davies		29-JAN-1997
6	Partners		05-JAN-1997
7	Zlotkey		29-JAN-2000
8	Tucker		30-JAN-1997
9	King		30-JAN-1996
10	Marvins		24-JAN-2000
11	Fox		24-JAN-1998
12	Johnson		04-JAN-2000
13	Taylor		24-JAN-1998
14	Sarchand		27-JAN-1996

추가 연습(계속)

다음 연습은 고급 subquery를 살펴본 후 추가로 수행할 수 있습니다.

- 8) CEO가 이윤 분배를 위해 회사 내 상위 세 명의 고액 연봉자에 대한 보고서를 요구합니다. 따라서 CEO에게 고액 연봉자 리스트를 제출해야 합니다. EMPLOYEES 테이블에 급여 수준이 상위 세번째인 사원까지 표시하는 query를 작성합니다. 성 및 급여를 표시합니다.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000

- 9) California 주의 연금은 지방 조례에 따라 변경되어 왔습니다. 이에 따라 연금 담당자가 조례에 영향을 받는 사람들의 리스트를 수집해 달라고 요청했습니다. California 주에서 근무하는 사원의 사원 ID 및 성을 표시하는 query를 작성합니다. 힌트: 스칼라 subquery를 사용하십시오.

	EMPLOYEE_ID	LAST_NAME
1	198	OConnell
2	199	Grant
3	120	Weiss
4	121	Fripp
5	122	Kaufling
6	123	Vollman
7	124	Mourgos
8	125	Nayer
9	126	Mikkilineni
10	127	Landry
11	128	Markle

- 10) DBA인 Nita가 데이터베이스에서 오래된 정보를 제거하려고 합니다. 그녀는 오래된 채용 기록이 불필요하다고 생각하여 다음과 같은 작업을 요청했습니다.

JOB_HISTORY 테이블에서 사원에 대한 MIN(START_DATE)를 조회하여 사원의 가장 오래된 JOB_HISTORY 행을 삭제하는 query를 작성합니다. 적어도 두 개 이상의 직무를 변경한 사원의 레코드만 삭제합니다.

힌트: Correlated DELETE 명령을 사용하십시오.



추가 연습(계속)

- 11) Human Resources 부사장은 직원과의 연례 간담회를 위해 전체 채용 기록을 필요로 합니다. 그는 급하게 전화해서 DBA의 지시를 중단하라고 말합니다.

트랜잭션을 롤백합니다.

- 12) 침체된 경제로 인해 비용 절감 노력을 강화하고 있습니다. CEO가 회사에서 최고 급여를 받는 직무를 검토하려고 합니다. 따라서 다음 조건에 해당하는 리스트를 CEO에게 제출해야 합니다.

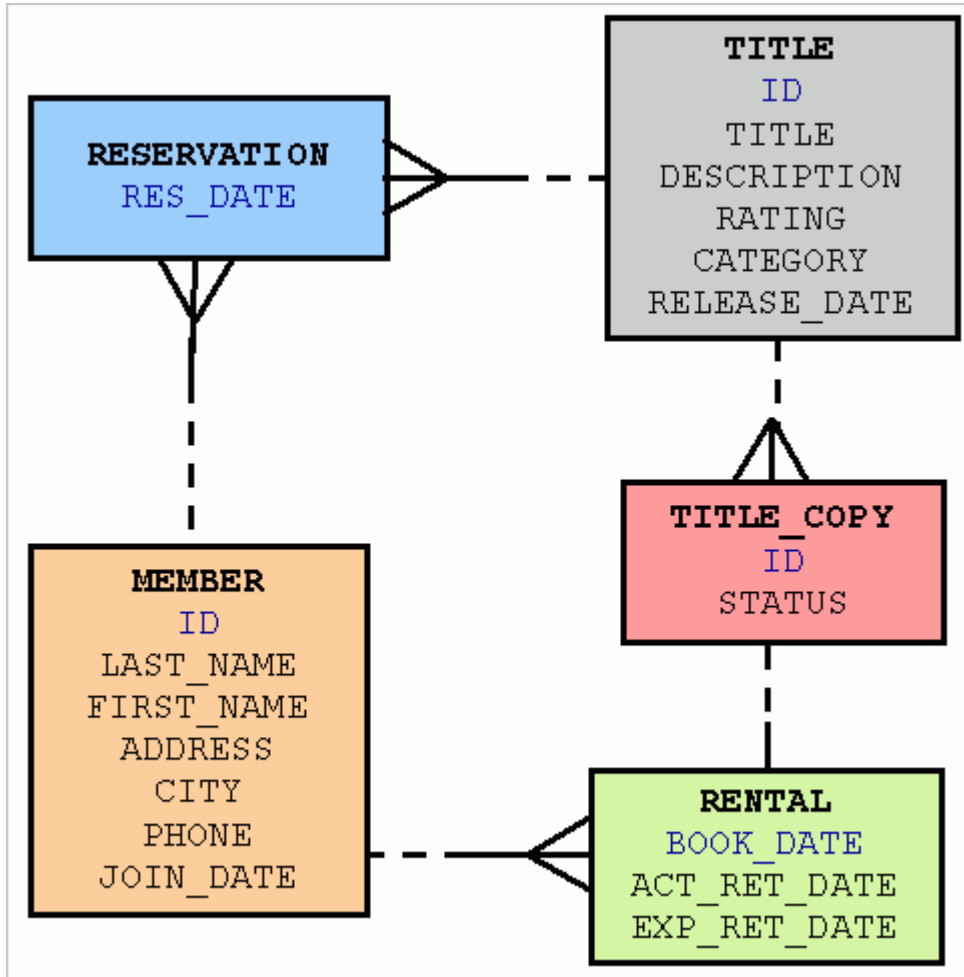
최대 급여가 회사 전체 최대 급여의 절반 이상이 되는 직무의 직무 ID를 표시하는 query를 작성합니다. WITH 절을 사용하여 이 query를 작성하고 query 이름을 MAX_SAL_CALC로 지정합니다.

 RANK	JOB_TITLE	 RANK	JOB_TOTAL
1	President		24000
2	Administration Vice President		17000
3	Sales Manager		14000
4	Marketing Manager		13000

추가 연습: 사례 연구

SQL Fundamentals I 과정의 사례 연구에서는 비디오 응용 프로그램에 사용할 일련의 데이터베이스 테이블을 구축했습니다. 또한 비디오 대여점 데이터베이스에서 레코드를 삽입, 갱신 및 삭제하고 보고서를 생성했습니다.

다음은 비디오 응용 프로그램용으로 생성된 테이블과 열을 나타낸 다이어그램입니다.



참고: 이러한 테이블이 이미 있는 경우에는 labs 폴더에서 dropvid.sql 스크립트를 실행하여 해당 테이블을 삭제하십시오. 그런 다음 labs 폴더에서 buildvid.sql 스크립트를 실행하여 테이블을 생성하고 채우십시오.

추가 연습: 사례 연구(계속)

- 1) 테이블 및 테이블의 열 정의 리스트를 표시하는 보고서를 실행하여 테이블이 제대로 생성되었는지 확인합니다.

R2	TABLE_NAME	R2	COLUMN_NAME	R2	DATA_TYPE	R2	NULLABLE
1	MEMBER		MEMBER_ID		NUMBER		N
2	MEMBER		LAST_NAME		VARCHAR2		N
3	MEMBER		FIRST_NAME		VARCHAR2		Y
4	MEMBER		ADDRESS		VARCHAR2		Y
5	MEMBER		CITY		VARCHAR2		Y
6	MEMBER		PHONE		VARCHAR2		Y
7	MEMBER		JOIN_DATE		DATE		N
8	RENTAL		BOOK_DATE		DATE		N
9	RENTAL		COPY_ID		NUMBER		N
10	RENTAL		MEMBER_ID		NUMBER		N
11	RENTAL		TITLE_ID		NUMBER		N
12	RENTAL		ACT_RET_DATE		DATE		Y
13	RENTAL		EXP_RET_DATE		DATE		Y
14	RESERVATION		RES_DATE		DATE		N
15	RESERVATION		MEMBER_ID		NUMBER		N
16	RESERVATION		TITLE_ID		NUMBER		N

- 2) 데이터 디렉터리에서 MEMBER_ID_SEQ 및 TITLE_ID_SEQ 시퀀스가 있는지 확인합니다.

R2	SEQUENCE_NAME
1	DEPARTMENTS_SEQ
2	EMPLOYEES_SEQ
3	LOCATIONS_SEQ
4	MEMBER_ID_SEQ
5	TITLE_ID_SEQ

- 3) 자신이 대여한 비디오에만 액세스할 수 있는 유저를 생성할 수도 있습니다. Carmen이라는 유저를 생성하고 RENTAL 테이블에서 선택할 수 있는 권한을 부여합니다.
참고: 데이터베이스 계정을 username의 접두어로 사용하십시오. 예를 들어, 유저 oraxx의 경우 oraxx_Carmen이라는 유저를 생성합니다.
- 4) price 열(number 4,2)을 TITLE 테이블에 추가하여 해당 타이틀의 대여 비용을 저장합니다.
- 5) CATEGORY 테이블을 추가하여 CATEGORY_ID와 CATEGORY_DESCRIPTION을 저장합니다. 이 테이블에는 TITLE 테이블의 CATEGORY 열과 함께 Foreign Key가 있습니다.
- 6) 데이터 디렉터리에서 모든 테이블을 선택합니다.
- 7) 더 이상 예약 사항을 저장할 필요 없이 테이블을 삭제할 수 있습니다.

추가 연습: 사례 연구(계속)

- 8) RENTAL_HISTORY 테이블을 생성하여 최근 6개월 동안의 회원별 대여 세부 사항을 저장합니다. **힌트:** RENTAL 테이블을 복사할 수 있습니다.
- 9) 지난 달에 대여된 타이틀 중 상위 10개 타이틀 리스트를 범주로 구분하여 표시합니다.

	A-Z	CATEGORY	A-Z	TITLE
1		ACTION		Soda Gang
2		CHILD		Willie and Christmas Too
3		COMEDY		My Day Off
4		SCIFI		Alien Again
5		SCIFI		Interstellar Wars

- 10) 회원이 비디오를 6일 연체하여 반납할 경우 연체료(타이틀 가격/일수)를 계산할 수도 있습니다.

	A-Z	TITLE	A-Z	MEMBER_ID	A-Z	PRICE	A-Z	LATEFEE
1		Alien Again		101		(null)		(null)
2		My Day Off		102		(null)		(null)
3		Interstellar Wars		101		(null)		(null)

- 11) 두 번 이상 대여한 회원 리스트를 표시합니다.

	A-Z	MEMBER_ID	A-Z	LAST_NAME	A-Z	FIRST_NAME
1		101		Velasquez		Carmen

- 12) 대여 상태가 포함된 타이틀 목록을 표시합니다.

	A-Z	TITLE
1		Alien Again
2		My Day Off
3		Interstellar Wars

- 13) 전화 번호에 "99"가 포함된 회원 리스트를 표시합니다.

	A-Z	POSITION	A-Z	MEMBER_ID	A-Z	LAST_NAME	A-Z	FIRST_NAME
1		1		101		Velasquez		Carmen
2		1		106		Urguhart		Molly
3		1		109		Catchpole		Antoinette

추가 연습 문제 해답

다음 연습은 "스키마 객체 관리" 및 "대형 데이터 집합 조작" 단원의 DML(데이터 조작어) 문과 DDL(데이터 정의어) 문에 대해 살펴본 후 추가로 수행할 수 있습니다.

참고: SPECIAL_SAL, SAL_HISTORY 및 MGR_HISTORY 테이블을 생성하려면 labs 폴더에서 lab_ap_cre_special_sal.sql, lab_ap_cre_sal_history.sql 및 lab_ap_cre_mgr_history.sql 스크립트를 실행하십시오.

.

추가 연습 문제 해답

- 1) Human Resources 부서에서 산업별 임금 실태 조사를 기반으로 저임금 사원, 사원의 급여 내역 및 관리자의 급여 내역 리스트를 만들려고 합니다. 이에 따라 다음과 같은 작업을 요청했습니다.

다음 작업을 수행하는 명령문을 작성합니다.

- EMPLOYEES 테이블에서 사원 ID가 200보다 크거나 같은 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID 등의 세부 정보를 검색합니다.
- 급여가 \$5,000 미만일 경우 사원 ID, 급여 등의 세부 정보를 SPECIAL_SAL 테이블에 삽입합니다.
- 사원 ID, 채용 날짜, 급여 등의 세부 정보를 SAL_HISTORY 테이블에 삽입합니다.
- 사원 ID, 관리자 ID, 급여 등의 세부 정보를 MGR_HISTORY 테이블에 삽입합니다.

```
INSERT ALL
WHEN SAL < 5000 THEN
INTO special_sal VALUES (EMPID, SAL)
ELSE
INTO sal_history VALUES(EMPID, HIREDATE, SAL)
INTO mgr_history VALUES(EMPID, MGR, SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
       salary SAL, manager_id MGR
FROM employees
WHERE employee_id >=200;
```

- 2) SPECIAL_SAL, SAL_HISTORY 및 MGR_HISTORY 테이블을 query하여 삽입된 레코드를 확인합니다.

```
SELECT * FROM special_sal;
SELECT * FROM sal_history;
SELECT * FROM mgr_history;
```


추가 연습 문제 해답(계속)

- 3) DBA인 Nita가 테이블 생성을 요청합니다. 이 테이블에는 Primary Key 제약 조건이 있지만 인덱스 이름을 제약 조건과 다르게 지정하려고 합니다. 다음 테이블 instance 차트에 준하여 LOCATIONS_NAMED_INDEX 테이블을 생성합니다. PRIMARY KEY 열의 인덱스 이름을 LOCATIONS_PK_IDX로 지정합니다.

열 이름	Deptno	Dname
Primary Key	있음	
데이터 유형	Number	VARCHAR2
길이	4	30

```
CREATE TABLE LOCATIONS_NAMED_INDEX  
(location_id NUMBER(4) PRIMARY KEY USING INDEX  
(CREATE INDEX locations_pk_idx ON  
LOCATIONS_NAMED_INDEX(location_id)),  
location_name VARCHAR2(20));
```

- 4) USER_INDEXES 테이블을 query하여 LOCATIONS_NAMED_INDEX 테이블에 대한 INDEX_NAME을 표시합니다.

```
SELECT INDEX_NAME, TABLE_NAME  
FROM USER_INDEXES  
WHERE TABLE_NAME = 'LOCATIONS_NAMED_INDEX';
```

추가 연습 문제 해답(계속)

다음 연습은 datetime 함수를 살펴본 후 추가로 수행할 수 있습니다.

글로벌 회사에서 새로 취임한 부사장이 모든 지사의 각 시간대를 파악하려고 합니다. 새 부사장이 다음과 같은 정보를 요청했습니다.

- 5) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY HH24:MI:SS로 설정합니다.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
```

- 6) a) 다음 시간대에 대해 시간대 오프셋(TZ_OFFSET)을 표시하는 query를 작성합니다.

– Australia/Sydney

```
SELECT TZ_OFFSET ('Australia/Sydney') from dual;
```

– Chile/Easter Island

```
SELECT TZ_OFFSET ('Chile/EasterIsland') from dual;
```

- b) 세션을 변경하여 TIME_ZONE 파라미터 값을 Australia/Sydney의 시간대 오프셋으로 설정합니다.

```
ALTER SESSION SET TIME_ZONE = '+10:00';
```

- c) 이 세션에 대해 SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

```
SELECT SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

- d) 세션을 변경하여 TIME_ZONE 파라미터 값을 Chile/Easter Island의 시간대 오프셋으로 설정합니다.

참고: 앞 질문의 결과는 다른 날짜에 준한 것이며 일부 수강생의 실제 결과와 다를 수 있습니다. 또한 시간대 오프셋은 일광 절약 시간에 따라 국가마다 다를 수 있습니다.

```
ALTER SESSION SET TIME_ZONE = '-06:00';
```

추가 연습 문제 해답(계속)

- e) 이 세션에 대해 SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

```
SELECT SYSDATE, CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

- f) 세션을 변경하여 NLS_DATE_FORMAT을 DD-MON-YYYY로 설정합니다.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

참고

- 앞의 질문에서 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP는 모두 세션 시간대의 영향을 받습니다. SYSDATE는 세션 시간대의 영향을 받지 않습니다.
 - 앞 질문의 결과는 다른 날짜에 준한 것이며 일부 수강생의 실제 결과와 다를 수 있습니다. 또한 시간대 오프셋은 일광 절약 시간에 따라 국가마다 다를 수 있습니다.
- 7) Human Resources 부서에서 1월에 입사한 사원 리스트를 검토하려고 합니다. 이에 따라 다음과 같은 작업을 요청했습니다.

입사 연도에 관계없이 1월에 입사한 사원들의 성, 채용 월, 채용 날짜를 표시하는 query를 작성합니다.

```
SELECT last_name, EXTRACT (MONTH FROM HIRE_DATE),
HIRE_DATE FROM employees
WHERE EXTRACT (MONTH FROM HIRE_DATE) = 1;
```

추가 연습 문제 해답(계속)

다음 연습은 고급 subquery를 살펴본 후 추가로 수행할 수 있습니다.

참고: code_05_12_sb.sql을 사용하여 HIRE_DATE 열을 TIMESTAMP로 변환한 경우 HIRE_DATE 열이 -JAN-00 12.00.00.000000000 AM과 같이 표시될 수 있습니다.

- 8) CEO가 이윤 분배를 위해 회사 내 상위 세 명의 고액 연봉자에 대한 보고서를 요구합니다. 따라서 CEO에게 고액 연봉자 리스트를 제출해야 합니다.

EMPLOYEES 테이블에 급여 수준이 상위 세번째인 사원까지 표시하는 query를 작성합니다. 성 및 급여를 표시합니다.

```
SELECT last_name, salary
FROM   employees e
WHERE  3 > (SELECT COUNT (*)
           FROM   employees
           WHERE  e.salary < salary);
```

- 9) California 주의 연금은 지방 조례에 따라 변경되어 왔습니다. 이에 따라 연금 담당자가 조례에 영향을 받는 사람들의 리스트를 수집해 달라고 요청했습니다. California 주에서 근무하는 사원의 사원 ID 및 성을 표시하는 query를 작성합니다. **힌트:** 스칼라 subquery를 사용하십시오.

```
SELECT employee_id, last_name
FROM   employees e
WHERE  ((SELECT location_id
         FROM   departments d
         WHERE  e.department_id = department_id )
        IN  (SELECT location_id
             FROM   locations l
             WHERE  state_province =
'California'));
```

추가 연습 문제 해답(계속)

- 10) DBA인 Nita가 데이터베이스에서 오래된 정보를 제거하려고 합니다. 그녀는 오래된 채용 기록이 불필요하다고 생각하여 다음과 같은 작업을 요청했습니다.

JOB_HISTORY 테이블에서 사원에 대한 MIN(START_DATE)를 조회하여 사원의 가장 오래된 JOB_HISTORY 행을 삭제하는 query를 작성합니다. 적어도 두 개 이상의 직무를 변경한 사원의 레코드만 삭제합니다.

힌트: Correlated DELETE 명령을 사용하십시오.

```
DELETE FROM job_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE = (SELECT MIN(start_date)
                        FROM job_history JH
                        WHERE JH.employee_id =
                              E.employee_id)
       AND 3 > (SELECT COUNT(*)
                FROM job_history JH
                WHERE JH.employee_id =
                      E.employee_id
                GROUP BY EMPLOYEE_ID
                HAVING COUNT(*) >= 2));
```

- 11) Human Resources 부사장은 사원과의 연례 간담회를 위해 전체 채용 기록을 필요로 합니다. 그는 급하게 전화해서 DBA의 지시를 중단하라고 말합니다. 트랜잭션을 롤백합니다.

```
ROLLBACK;
```

- 12) 침체된 경제로 인해 비용 절감 노력을 강화하고 있습니다. CEO가 회사에서 최고 급여를 받는 직무를 검토하려고 합니다. 따라서 다음 조건에 해당하는 리스트를 CEO에게 제출해야 합니다.

최대 급여가 회사 전체 최대 급여의 절반 이상이 되는 직무의 직무 ID를 표시하는 query를 작성합니다. WITH 절을 사용하여 이 query를 작성하고 query 이름을 MAX_SAL_CALC로 지정합니다.

```
WITH
MAX_SAL_CALC AS (SELECT job_title, MAX(salary) AS
job_total
FROM employees, jobs
WHERE employees.job_id = jobs.job_id
GROUP BY job_title)
SELECT job_title, job_total
FROM MAX_SAL_CALC
WHERE job_total > (SELECT MAX(job_total) * 1/2
FROM MAX_SAL_CALC)
ORDER BY job_total DESC;
```

추가 연습: 사례 연구 해답

이러한 테이블이 이미 있는 경우에는 labs 폴더에서 dropvid.sql 스크립트를 실행하여 해당 테이블을 삭제하십시오. 그런 다음 labs 폴더에서 buildvid.sql 스크립트를 실행하여 테이블을 생성하고 채우십시오.

- 1) 테이블 및 테이블의 열 정의 리스트를 표시하는 보고서를 실행하여 테이블이 제대로 생성되었는지 확인합니다.

```
SELECT table_name,column_name,data_type,nullable
FROM user_tab_columns
WHERE table_name
IN('MEMBER','TITLE','TITLE_COPY','RENTAL','RESERVATION');
```

- 2) 데이터 디렉터리에서 MEMBER_ID_SEQ 및 TITLE_ID_SEQ 시퀀스가 있는지 확인합니다.

```
SELECT sequence_name FROM user_sequences;
```

- 3) 자신이 대여한 비디오에만 액세스할 수 있는 유저를 생성할 수도 있습니다. Carmen이라는 유저를 생성하고 RENTAL 테이블에서 선택할 수 있는 권한을 부여합니다.

참고: 데이터베이스 계정을 username의 접두어로 사용하십시오. 예를 들어, 유저 oraxx의 경우 oraxx_Carmen이라는 유저를 생성합니다.

```
CREATE USER oraxx_carmen IDENTIFIED BY oracle ;
GRANT select ON rental TO oraxx_carmen;
```

- 4) price 열(number 4,2)을 TITLE 테이블에 추가하여 해당 타이틀의 대여 비용을 저장합니다.

```
ALTER TABLE title ADD(price NUMBER(6))
```

- 5) CATEGORY 테이블을 추가하여 CATEGORY_ID와 CATEGORY_DESCRIPTION을 저장합니다. 이 테이블에는 TITLE 테이블의 CATEGORY 열과 함께 Foreign Key가 있습니다.

```
CREATE TABLE CATEGORY
(
    "CATEGORY_ID" NUMBER(6,0) NOT NULL ENABLE,
    "CATEGORY_DESCRIPTION" VARCHAR2(4000 BYTE),
    CONSTRAINT "CATEGORY_PK" PRIMARY KEY ("CATEGORY_ID")
)
```

- 6) 데이터 디렉터리에서 모든 테이블을 선택합니다.

```
SELECT table_name FROM user_tables order by table_name;
```

- 7) 더 이상 예약 사항을 저장할 필요 없이 테이블을 삭제할 수 있습니다.

```
DROP TABLE reservation cascade constraints;
```

추가 연습: 사례 연구 해답(계속)

- 8) RENTAL_HISTORY 테이블을 생성하여 최근 6개월 동안의 회원별 대여 세부 사항을 저장합니다. **힌트:** RENTAL 테이블을 복사할 수 있습니다.

```
CREATE TABLE rental_history as select * from rental where
'1' = '1'
```

- 9) 지난 달에 대여된 타이틀 중 상위 10개 타이틀 리스트를 범주로 구분하여 표시합니다.

```
SELECT t.CATEGORY, t.TITLE
FROM TITLE t, RENTAL r
WHERE t.TITLE_ID = r.TITLE_ID AND
      r. BOOK_DATE > (SYSDATE - 30) AND
      rownum < 10
order by category;
```

- 10) 회원이 비디오를 6일 연체하여 반납할 경우 연체료(타이틀 가격/일수)를 계산할 수도 있습니다.

```
SELECT t.title, m.member_id, t.price, (t.price*6) latefee
FROM title t, member m, rental r
WHERE t.title_id = r.title_id AND
      m.member_id = r.member_id AND
      r.act_ret_date is null;
```

- 11) 두 번 이상 대여한 회원 리스트를 표시합니다.

```
SELECT member_id, last_name, first_name FROM member m
where 2 <= (select count(*) from rental_history where
member_id = m.member_id);
```

- 12) 대여 상태가 포함된 타이틀 목록을 표시합니다.

```
SELECT t.title
FROM title t
JOIN (select title_id, status from title_copy) b
ON t.title_id = b.title_id AND b.status = 'RENTED';
```

- 13) 전화 번호에 "99"가 포함된 회원 리스트를 표시합니다.

```
SELECT REGEXP_COUNT(phone, '99', 1, 'i') position, member_id,
last_name, first_name
FROM member
WHERE REGEXP_COUNT(phone, '99', 1, 'i') > 0;
```

