

Oracle Database 11g: SQL Fundamentals I(한글판)

학생용 • 볼륨 I

D49996KR10

Edition 1.0

2007년 8월

D54000

ORACLE®

소개

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단원 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 과정 목표 이해
- **Oracle Database 11g**의 기능 나열
- 관계형 데이터베이스의 이론적 측면과 물리적 측면 설명
- **Oracle** 서버의 **RDBMS** 및 **ORDBMS**(객체 관계형 데이터베이스 관리 시스템) 구현 설명
- 이 과정에서 사용할 수 있는 개발 환경 식별
- **Oracle SQL Developer**의 주요 기능 설명 및 사용
- 이 과정에 사용된 데이터베이스 및 스키마 설명

ORACLE

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 RDBMS(관계형 데이터베이스 관리 시스템) 및 ORDBMS(객체 관계형 데이터베이스 관리 시스템)에 대해 살펴봅니다. 또한 SQL 문 실행과 형식 지정 및 Reporting을 위한 개발 환경으로서 Oracle SQL Developer 및 SQL*Plus를 소개합니다.

단원 내용

- 과정 목표, 내용 및 이 과정에 사용되는 부록
- Oracle Database 11g 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- SQL 및 개발 환경 소개
- Oracle SQL Developer 개요
- Oracle Database 11g 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

과정 목표

이 과정을 마치면 다음을 수행할 수 있습니다.

- **Oracle Database 11g**의 주요 구성 요소 파악
- **SELECT** 문을 사용하여 테이블에서 행 및 열 데이터 검색
- 정렬되고 제한된 데이터로 보고서 작성
- **SQL** 함수를 사용하여 커스터마이즈된 데이터 생성 및 검색
- 복합 **query**를 실행하여 여러 테이블에서 데이터 검색
- **Oracle Database 11g**에서 **DML**(데이터 조작어) 문을 실행하여 데이터 갱신
- **DDL**(데이터 정의어) 문을 실행하여 스키마 객체 생성 및 관리

ORACLE

Copyright © 2007, Oracle. All rights reserved.

과정 목표

이 과정에서는 Oracle Database 11g 데이터베이스 기술에 대해 소개합니다. 관계형 데이터베이스 및 강력한 SQL 프로그래밍 언어의 기본 개념을 살펴본 후 단일 또는 여러 테이블에 대한 query를 작성하고, 테이블의 데이터를 조작하고, 데이터베이스 객체를 생성하고, 메타 데이터를 query할 수 있는 필수 SQL 기술을 배웁니다.

과정 내용

- 첫째날:
 - 소개
 - **SQL SELECT** 문을 사용하여 데이터 검색
 - 데이터 제한 및 정렬
 - 단일 행 함수를 사용하여 출력 커스터마이즈
 - 변환 함수 및 조건부 표현식 사용
- 둘째날:
 - 그룹 함수를 사용하여 집계된 데이터 보고
 - 여러 테이블의 데이터 표시
 - **subquery**를 사용하여 **query** 해결
 - 집합 연산자 사용

ORACLE

Copyright © 2007, Oracle. All rights reserved.

과정 내용

- 셋째날:
 - 데이터 조작
 - **DDL** 문을 사용하여 테이블 생성 및 관리
 - 기타 스키마 객체 생성

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

이 과정에 사용되는 부록

- 부록 **A**: 연습 문제 해답
- 부록 **B**: 테이블 설명
- 부록 **C**: Oracle 조인 구문
- 부록 **D**: **SQL*Plus** 사용
- 부록 **E**: **Oracle SQL Developer GUI**를 사용하여 **DML** 및 **DDL** 작업 수행
- 추가 연습
- 추가 연습 해답

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단원 내용

- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g: 핵심 영역



Infrastructure
그리드

정보
관리

응용 프로그램
개발

ORACLE

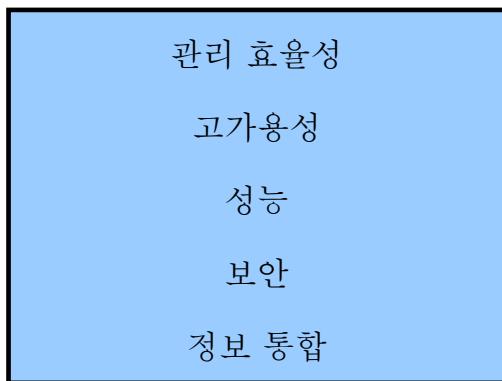
Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g: 핵심 영역

Oracle Database 11g에서는 다음 핵심 영역에 걸쳐 광범위한 기능을 제공합니다.

- **Infrastructure 그리드:** Oracle의 Infrastructure 그리드 기술을 사용하면 저비용 서버 및 저장 영역을 도입하여 관리 효율성, 고가용성 및 성능 측면에서 최고 품질의 서비스를 제공하는 시스템을 구축할 수 있습니다. Oracle Database 11g에서는 그리드 컴퓨팅의 이점을 통합하고 확장합니다. 그리드 컴퓨팅을 완전히 이용하는 것과 별도로 Oracle Database 11g에서는 고유한 변경 보증 기능을 통해 통제되고 비용 효율적인 방법으로 변경 내용을 관리합니다.
- **정보 관리:** Oracle Database 11g에서는 기존의 정보 관리 기능을 컨텐트 관리, 정보 통합 및 정보 주기 관리 영역으로 확장합니다. Oracle에서는 XML(Extensible Markup Language), 텍스트, 공간 정보, 멀티미디어, 의료 이미지 및 시맨틱 기술과 같은 고급 데이터 유형의 컨텐트를 관리합니다.
- **응용 프로그램 개발:** Oracle Database 11g에서는 PL/SQL, Java/JDBC, .NET과 Windows, PHP, SQL Developer 및 Application Express와 같은 모든 주요 응용 프로그램 개발 환경을 사용하고 관리할 수 있습니다.

Oracle Database 11g



ORACLE

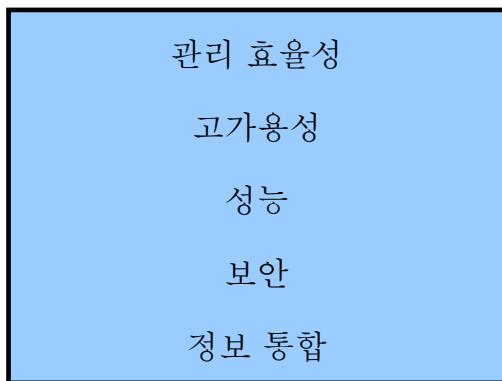
Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g

조직에서는 업무 응용 프로그램에 대한 신속하고 안전한 액세스를 끊임없이 요구하는 유저를 위해 테라바이트 단위의 정보를 지원해야 합니다. 데이터베이스 시스템은 신뢰할 수 있어야 하고 어떤 형태의 failure에서도 신속하게 복구할 수 있어야 합니다. Oracle Database 11g는 조직에서 Infrastructure 그리드를 쉽게 관리하고 고품질의 서비스를 제공할 수 있도록 다음 기능 영역에 중점을 두고 설계되었습니다.

- **관리 효율성:** 변경 보증, 관리 자동화 및 오류 진단 기능 중 몇 가지를 사용하여 DBA(데이터베이스 관리자)가 생산성을 향상하고, 비용을 줄이고, 오류를 최소화하며, 서비스 품질을 최대화할 수 있습니다. 관리 효율성을 높일 수 있는 몇 가지 유용한 기능으로는 Database Replay 기능, SQL Performance Analyzer 및 Automatic SQL Tuning 기능이 있습니다.
- **고가용성:** 고가용성 기능을 사용하여 다운타임 및 데이터 손실을 줄일 수 있습니다. 이러한 기능으로 온라인 작업을 개선하고 데이터베이스를 더 빠르게 업그레이드할 수 있습니다.

Oracle Database 11g



A solid red horizontal bar at the bottom of the slide.

Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g(계속)

- **성능:** SecureFiles, OLTP(Online Transaction Processing) 압축, RAC(Real Application Clusters) 최적화 및 Result Cache 등의 기능을 사용하여 데이터베이스의 성능을 크게 개선할 수 있습니다. Oracle Database 11g를 사용하는 조직에서는 빠른 데이터 액세스를 제공하며 확장 가능한 대형 트랜잭션 및 데이터 웨어하우징 시스템을 저비용 모듈식 저장 영역을 사용하여 관리할 수 있습니다.
- **보안:** Oracle Database 11g는 조직에서 고유 보안 구성, 데이터 암호화(encryption)와 마스킹 및 정교한 감사(auditing) 기능으로 정보를 보호할 수 있도록 합니다. 모든 유형의 정보에 대한 빠르고 신뢰할 수 있는 액세스가 가능하도록 산업 표준 인터페이스를 사용하여 안전하고 확장성 있는 플랫폼을 제공합니다.
- **정보 통합:** Oracle Database 11g는 기업 전체에서 데이터를 더 잘 통합할 수 있도록 하는 많은 기능을 제공하며 고급 정보 수명 주기 관리 기능도 지원합니다. 따라서 데이터베이스의 데이터 변경을 쉽게 관리할 수 있습니다.

Oracle Fusion Middleware

통합 서비스, **Business Intelligence**, 협업 및 컨텐트 관리를 비롯한 광범위한 **J2EE**의 툴과 서비스 및 개발 툴을 포괄하며 고객에 의해 입증된 첨단의 표준 기반 소프트웨어 제품 포트폴리오



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle Fusion Middleware

Oracle Fusion Middleware는 SOA(Service-Oriented Architecture)의 개발, 배포 및 관리를 완전하게 지원하는 포괄적이고 잘 통합된 제품군입니다. SOA에서는 쉽게 통합할 수 있고 재사용할 수 있는 모듈식 업무 서비스 개발을 지원하므로 개발 및 유지 관리 비용을 줄이고 더 높은 품질의 서비스를 제공할 수 있습니다. Oracle Fusion Middleware는 플러그 가능한 구조를 사용하므로 기준의 모든 응용 프로그램, 시스템 또는 기술에 대한 투자를 활용할 수 있습니다. 견고한 핵심 기술을 통해 계획되었거나 계획되지 않은 정전으로 인해 발생하는 중단을 최소화합니다.

Oracle Fusion Middleware 제품군에는 다음과 같은 제품이 포함되어 있습니다.

- 엔터프라이즈 **Application Server**: Application Server
- 통합 및 프로세스 관리: BPEL Process Manager, Oracle Business Process Analysis Suite
- 개발 툴: Oracle Application Development Framework, JDeveloper, SOA Suite
- **Business Intelligence**: Oracle Business Activity Monitoring, Oracle Data Integrator
- 시스템 관리: Enterprise Manager
- **ID Management**: Oracle Identity Management
- 컨텐트 관리: Oracle Content Database Suite
- 유저 상호 작용: Portal, WebCenter

Oracle Enterprise Manager Grid Control 10g

- 효율적인 **Oracle Fusion Middleware** 관리
- 응용 프로그램 및 **Infrastructure** 수명 주기 관리 단순화
- 개선된 데이터베이스 관리 및 응용 프로그램 관리 기능



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Oracle Enterprise Manager Grid Control 10g

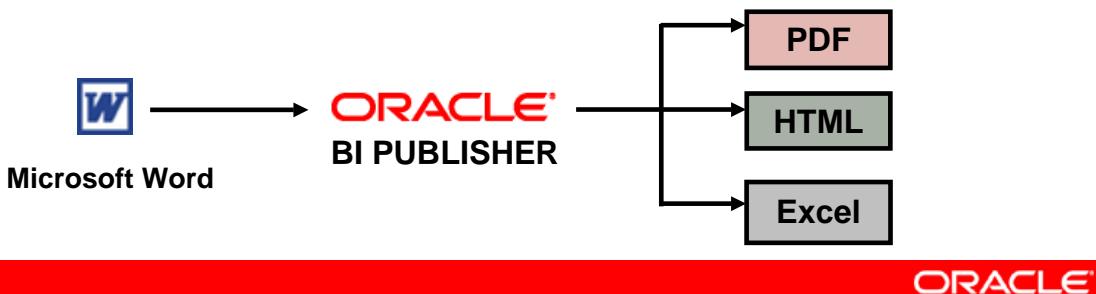
응용 프로그램, 미들웨어 및 데이터베이스 관리를 포괄하는 Oracle Enterprise Manager Grid Control 10g에서는 Oracle 시스템 및 타사 시스템에 대해 통합된 엔터프라이즈 관리를 제공합니다.

Oracle Enterprise Manager Grid Control 10g에는 업무 응용 프로그램에서 사용하는 SOA, Business Activity Monitoring, Identity Management 등의 서비스에 대한 고급 Oracle Fusion Middleware 관리 기능이 포함되어 있습니다.

- 응용 프로그램에 대한 광범위한 관리 기능 - 서비스 레벨 관리, 응용 프로그램 성능 관리, 구성 관리 및 변경 자동화 포함
- 그리드 자동화 기능 내장 - 정보 기술을 통해 변화하는 요구에 미리 반응하고 새로운 서비스를 더 신속하게 구현하므로 성공적인 비즈니스 지원
- 깊이 있는 진단 및 즉각적인 문제 해결 - 커스터마이즈된 응용 프로그램, Oracle E-Business Suite, PeopleSoft, Siebel, Oracle Fusion Middleware, Oracle Database 및 기본 Infrastructure를 비롯한 다양한 응용 프로그램에 걸쳐 사용할 수 있습니다.
- 광범위한 수명 주기 관리 기능 - 전체 응용 프로그램 및 **Infrastructure** 수명 주기에 대한 테스트, 준비 및 작업을 통한 생산 등의 솔루션을 제공하여 그리드 컴퓨팅을 확장합니다. 이 기능은 동기화된 패치, 추가적 운영 체제 지원 및 충돌 감지 기능을 통해 패치 관리를 단순화합니다.

Oracle BI Publisher

- 안전하고 다양한 형식으로 정보를 작성, 관리 및 전달하기 위한 중앙 구조 제공
- 모든 유형의 보고서 개발, 테스트 및 배포에 필요한 시간 및 복잡성 감소
 - 재무 보고서, 송장, 판매 또는 구매 주문, **XML** 및 **EDI/EFT(eText 문서)**
- 융통성 있는 커스터마이제이션 가능
 - 예를 들어, **PDF**, **HTML**, **Excel**, **RTF** 등의 다양한 형식으로 **Microsoft Word** 문서 보고서를 생성할 수 있습니다.



Copyright © 2007, Oracle. All rights reserved.

Oracle BI Publisher

Oracle Database 11g에는 Oracle의 엔터프라이즈 Reporting 솔루션인 Oracle BI Publisher도 포함되어 있습니다. Oracle BI Publisher(이전의 XML Publisher)는 복잡하고 분산된 환경에서 사용할 수 있는 가장 효율적이고 확장할 수 있는 Reporting 솔루션을 제공합니다.

Oracle BI Publisher는 업무 문서의 개발, 커스터마이제이션 및 유지 관리와 관련된 높은 비용을 줄이고 보고서 관리의 효율을 높입니다. 유저는 IT 직원 또는 개발자가 작성한 데이터 query에 기반한 자신의 보고서 형식을 친숙한 데스크톱 도구 집합을 사용하여 생성하고 유지 관리할 수 있습니다.

Oracle BI Publisher 보고서 형식은 대부분의 유저에게 이미 친숙한 도구인 Microsoft Word 또는 Adobe Acrobat을 사용하여 디자인할 수 있습니다. 또한 Oracle BI Publisher를 사용하여 다양한 데이터 소스에서 단일 출력 문서로 데이터를 가져올 수 있습니다. 보고서는 프린터, 전자 메일 또는 팩스로 전달할 수 있습니다. 보고서를 포털에 게시할 수 있습니다. 더 나아가 여러 유저가 WebDav(Web-based Distributed Authoring and Versioning) 웹 서버에서 공동으로 보고서를 편집하고 관리하도록 할 수도 있습니다.

단원 내용

- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

관계형 및 객체 관계형 데이터베이스 관리 시스템

- 관계형 모델 및 객체 관계형 모델
- 유저 정의 데이터 유형 및 객체
- 관계형 데이터베이스와의 완벽한 호환성
- 멀티미디어 및 대형 객체 지원
- 고품질 데이터베이스 서버 기능



ORACLE

Copyright © 2007, Oracle. All rights reserved.

관계형 및 객체 관계형 데이터베이스 관리 시스템

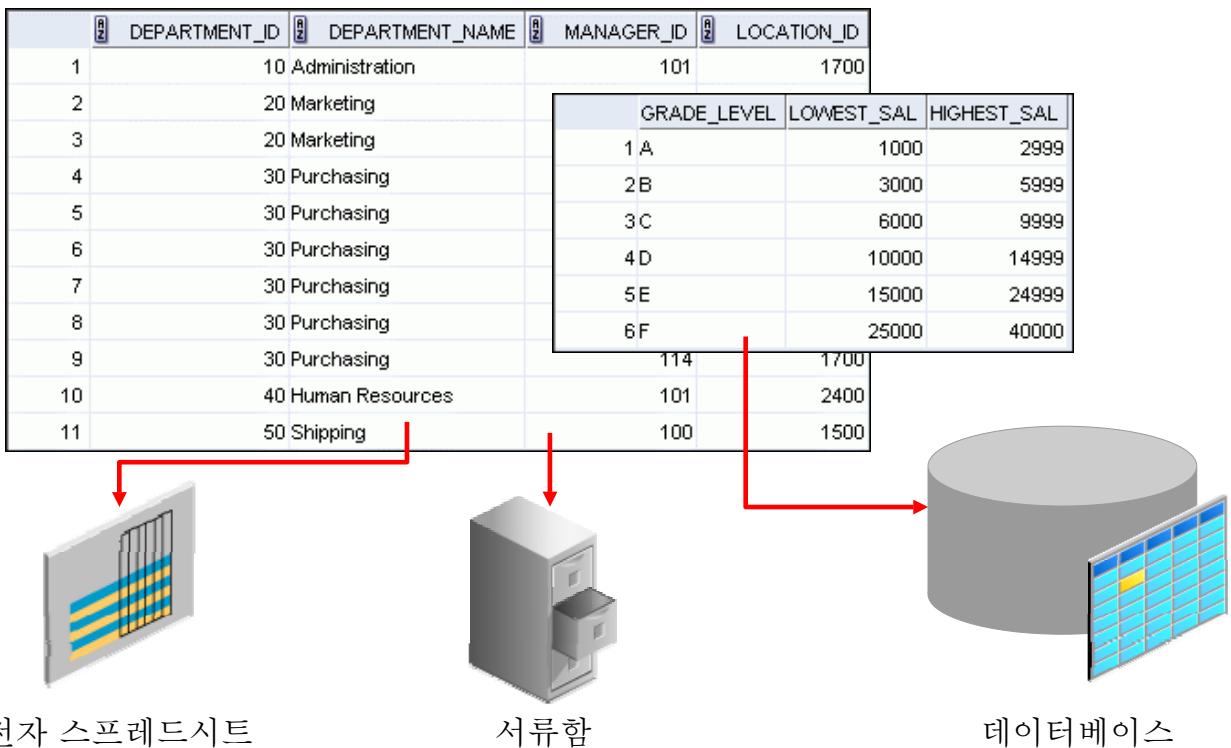
Oracle 서버는 관계형 및 객체 관계형 데이터베이스 모델을 모두 지원합니다.

Oracle 서버는 객체 지향 프로그래밍, 복잡한 데이터 유형, 복잡한 업무 객체, 관계형 세계와의 완벽한 호환성 등을 제공하는 객체 관계형 데이터베이스 모델을 지원하도록 데이터 모델링 기능이 확장되었습니다.

여기에는 런타임 데이터 구조의 공유 기능 향상, 대용량 버퍼 캐시, deferrable 제약 조건 등 OLTP 응용 프로그램의 향상된 성능과 기능을 위한 여러 기능이 포함됩니다. 데이터 웨어하우스 응용 프로그램은 삽입, 갱신, 삭제 작업의 병렬 실행, partition 기능, 병렬 인식 query 최적화와 같은 향상된 기능을 활용합니다. NCA(Network Computing Architecture) 프레임워크 내에서 작동되는 Oracle 모델은 분산된 다계층 클라이언트/서버 및 웹 기반 응용 프로그램을 지원합니다.

관계형 및 객체 관계형 모델에 대한 자세한 내용은 *Oracle Database Concepts 11g Release 1 (11.1)* 설명서를 참조하십시오.

다양한 미디어의 데이터 저장 영역



ORACLE

Copyright © 2007, Oracle. All rights reserved.

다양한 미디어의 데이터 저장 영역

모든 조직은 정보 요구 사항을 가지고 있습니다. 도서관은 회원, 도서, 반납 날짜, 연체료 등의 리스트를 보유합니다. 회사에서는 사원, 부서 및 급여와 관련된 정보를 저장해야 합니다. 이러한 정보 단위를 데이터라고 합니다.

조직은 데이터를 다양한 미디어에 다양한 형식으로 저장합니다. 예를 들어, 하드카피 문서는 서류함에 보관하고 데이터는 전자 스프레드시트나 데이터베이스에 저장합니다.

데이터베이스는 체계적으로 구성된 정보 모음입니다.

데이터베이스를 관리하려면 DBMS(데이터베이스 관리 시스템)가 필요합니다. DBMS는 요청에 따라 데이터베이스의 데이터를 저장, 검색 및 수정하는 프로그램입니다. 기본 데이터베이스 유형에는 계층형, 네트워크형, 관계형 및 가장 최신 유형인 객체 관계형의 네 가지가 있습니다.

관계형 데이터베이스 개념

- 데이터베이스 시스템의 관계형 모델은 **1970년 E. F. Codd** 박사가 처음 제안했습니다.
- 이 모델은 **RDBMS(관계형 데이터베이스 관리 시스템)**의 기초가 되었습니다.
- 관계형 모델은 다음과 같은 요소로 구성됩니다.
 - 객체 또는 관계 모음
 - 관계에서 실행되는 연산자 집합
 - 정확성 및 일관성을 보장하는 데이터 무결성

ORACLE

Copyright © 2007, Oracle. All rights reserved.

관계형 데이터베이스 개념

관계형 모델의 원리는 E. F. Codd 박사가 1970년 6월 발표한 "대용량 공유 데이터 뱅크용 데이터의 관계형 모델"이라는 논문을 통해 처음 소개되었습니다. 이 논문에서 Codd 박사는 데이터베이스 시스템의 관계형 모델을 제안했습니다.

그 당시 사용되던 일반적인 모델은 계층형 모델과 네트워크형 모델이었으며 간단한 플랫 파일 데이터 구조도 사용되었습니다. RDBMS(관계형 데이터베이스 관리 시스템)는 즉시 사용하기 쉽다는 점과 구조의 유연성으로 인해 큰 인기를 얻게 되었습니다. 또한 Oracle과 같은 수많은 혁신적인 업체에서 강력한 응용 프로그램 개발 및 유저 인터페이스 제품이 포함된 토탈 솔루션을 제공하여 RDBMS를 지원했습니다.

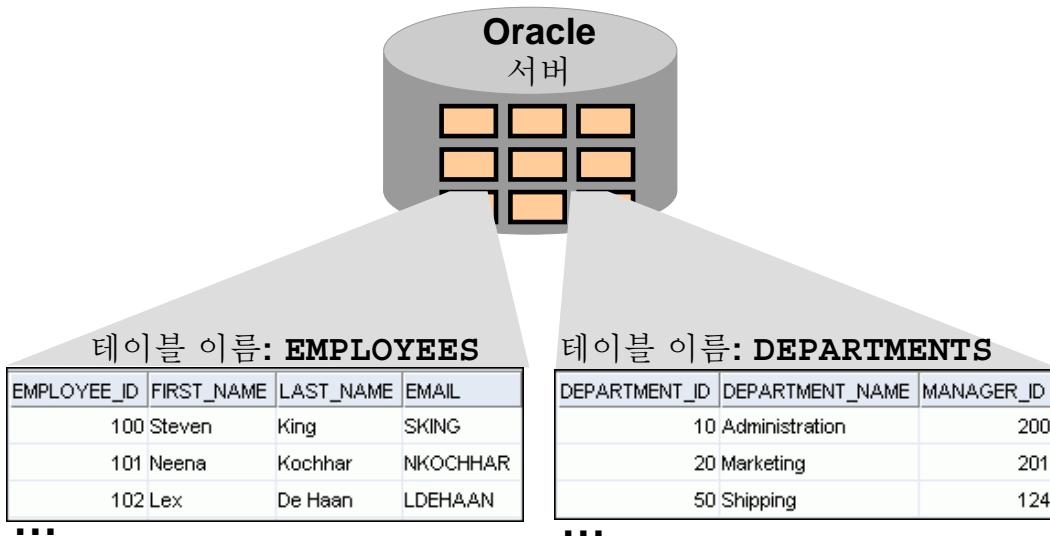
관계형 모델의 구성 요소

- 데이터를 저장하는 객체 또는 관계 모음
- 관계에서 다른 관계를 생성하는 데 사용할 수 있는 연산자 집합
- 정확성 및 일관성을 보장하는 데이터 무결성

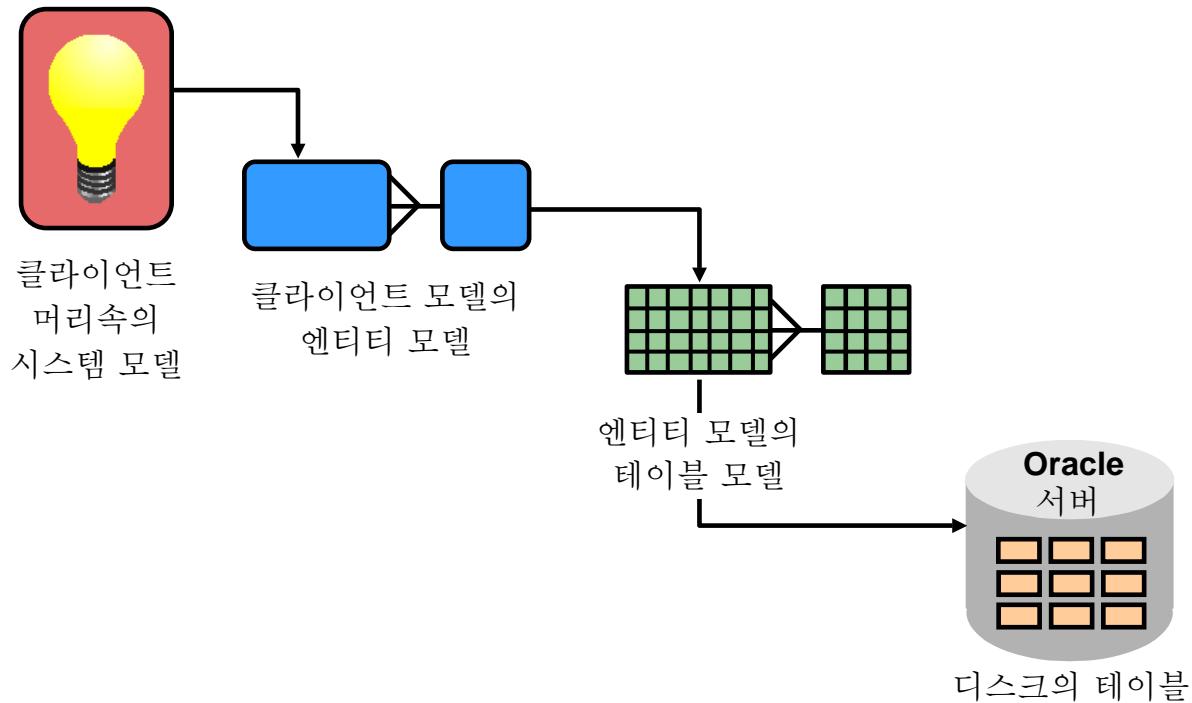
자세한 내용은 Chris Date가 저술한 *An Introduction to Database Systems, Eighth Edition*(Addison-Wesley: 2004)을 참조하십시오.

관계형 데이터베이스 정의

관계형 데이터베이스는 관계 또는 2차원 테이블 모음입니다.



데이터 모델



ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터 모델

모델은 설계의 기반입니다. 엔지니어는 자동차를 제품화하기 전에 모델을 작성하여 세부 사항들을 확인합니다. 같은 방식으로 시스템 설계자는 모델을 개발하여 아이디어를 점검하고 데이터베이스 설계에 대한 이해를 높입니다.

모델의 목적

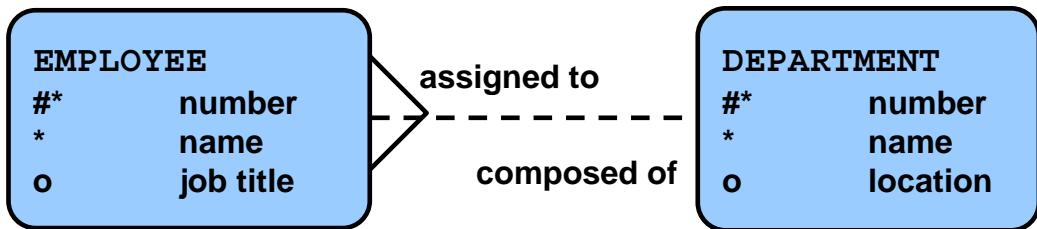
모델은 사람들의 머리속에 있는 개념을 표출하는 데 도움이 됩니다. 모델은 다음 작업을 수행하는 데 사용할 수 있습니다.

- 의사 소통
- 분류
- 설명
- 지정
- 조사
- 전개
- 분석
- 모방

이처럼 다양한 용도에 맞고, 일반 유저가 이해할 수 있고, 개발자가 데이터베이스 시스템을 구축하기에 충분한 세부 정보를 포함하는 모델을 만드는 것이 목표입니다.

엔티티 관계 모델

- 업무 사양 또는 전술을 토대로 엔티티 관계 다이어그램을 생성합니다.



- 시나리오:
 - "... 한 부서에 한 명 이상의 사원을 배정합니다..."
 - "... 일부 부서는 아직 배정된 사원이 없습니다..."

ORACLE

Copyright © 2007, Oracle. All rights reserved.

엔티티 관계 모델

효율적인 시스템에서는 데이터를 별개의 범주 또는 엔티티로 나눕니다. ER(엔티티 관계) 모델은 업무의 다양한 엔티티 및 이들 간의 관계를 도식화한 것입니다. ER 모델은 업무 사양 또는 전술로부터 파생되고 시스템 개발 초기의 분석 단계에서 구축됩니다. ER 모델은 업무에 필요한 정보와 업무 내에서 수행되는 작업을 구분합니다. 업무는 해당 작업을 변경할 수 있지만 정보 유형은 일관되게 유지됩니다. 따라서 데이터 구조도 일관성을 유지합니다.

ER 모델링의 이점:

- 조직의 정보를 명확한 형식으로 문서화
- 정보 요구 사항의 범위를 쉽게 파악할 수 있음
- 쉽게 이해할 수 있는 데이터베이스 설계용 그림 지도 제공
- 여러 응용 프로그램 통합을 위한 효과적인 프레임워크 제공

주요 구성 요소

- **엔티티:** 정보가 알려져야 하는 중요한 어떤 측면. 엔티티의 예로는 부서, 사원, 주문 등이 있습니다.
- **속성:** 엔티티를 설명하거나 분류하는 것. 예를 들어, 사원 엔티티의 경우 사원 번호, 이름, 직책, 채용 날짜, 부서 번호 등이 속성에 포함될 수 있습니다. 각 속성은 필수 항목이거나 선택 항목입니다. 이러한 특성을 선택 가능성(optionality)이라고 합니다.
- **관계:** 선택 가능성과 정도를 보여주는 엔티티 간의 명명된 연관. 관계의 예로는 사원과 부서, 주문과 품목 등이 있습니다.

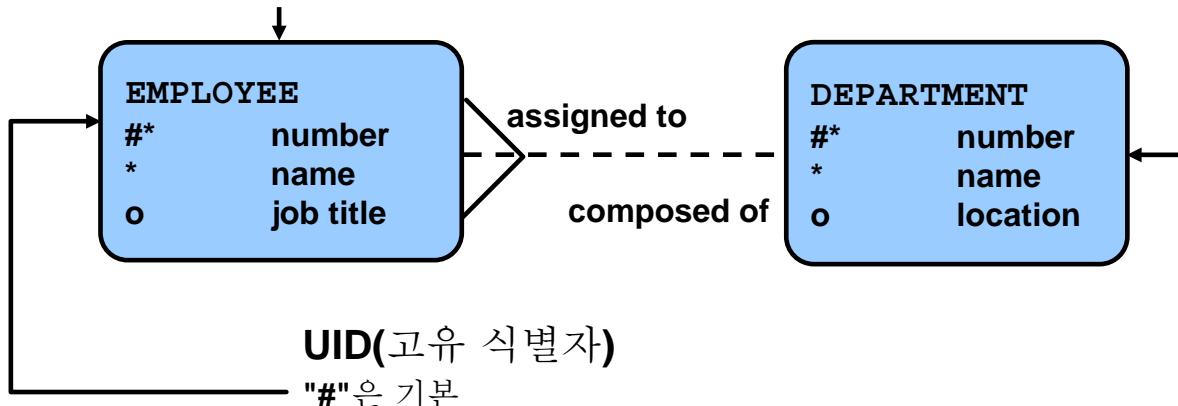
엔티티 관계 모델링 규칙

엔티티:

- 단수형 고유 이름
- 대문자
- 모서리가 둥근 상자
- 괄호 안은 동의어

속성:

- 단수형 이름
- 소문자
- "*"는 필수 사항
- "o"는 선택 사항



ORACLE

Copyright © 2007, Oracle. All rights reserved.

ER 모델링 규칙

엔티티

모델에서 엔티티를 나타내려면 다음 규칙을 사용합니다.

- 단수형 고유 엔티티 이름
- 엔티티 이름은 대문자
- 모서리가 둥근 상자
- 선택 사항인 동의어 이름은 괄호(()) 안에 대문자로 표기

속성

모델에서 속성을 나타내려면 다음 규칙을 사용합니다.

- 소문자의 단수형 이름
- 필수 속성(즉, 반드시 알려져야 하는 값)에는 별표(*) 태그
- 선택적 속성(즉, 알려질 수도 있는 값)에는 "o" 태그

관계

기호	설명
점선	"maybe"를 나타내는 선택적 요소
실선	"must be"를 나타내는 필수 요소
까치발	"one or more"를 나타내는 정도 요소
일방선	"one and only one"을 나타내는 정도 요소

ER 모델링 규칙(계속)

관계

관계의 각 방향에는 다음이 포함됩니다.

- 레이블: 예를 들어 *taught by* 또는 *assigned to*
- 선택 가능성: *must be* 또는 *may be*
- 정도: *one and only one* 또는 *one or more*

주: 기수(cardinality)는 정도(degree)의 동의어입니다.

각 소스 엔티티 {may be | must be} 관계 {one and only one | one or more} 대상 엔티티.

주: 규칙은 시계방향으로 읽습니다.

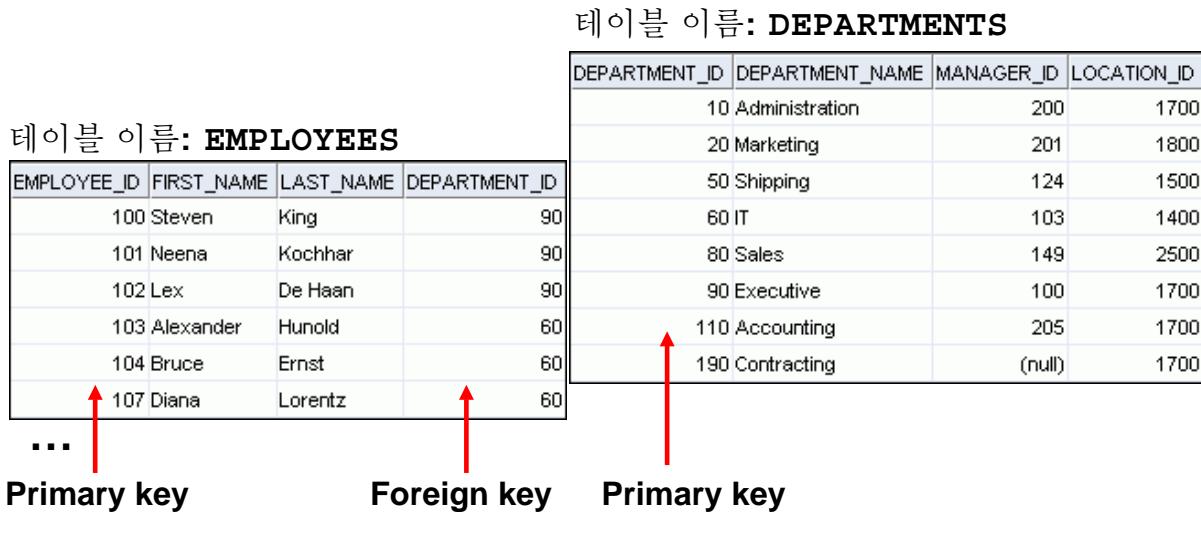
고유 식별자

UID(고유 식별자)는 속성이나 관계 또는 둘의 조합으로, 엔티티의 발생 값을 구분하는 역할을 합니다. 각 엔티티 발생 값을 고유하게 식별될 수 있어야 합니다.

- UID의 일부인 각 속성에는 해시 기호인 (#) 태그를 지정합니다.
- 보조 UID에는 팔호로 묶인 해시 기호인 (#) 태그를 지정합니다.

여러 테이블 연관짓기

- 테이블의 각 데이터 행은 **Primary key**를 통해 고유하게 식별됩니다.
- **Foreign key**를 사용하여 여러 테이블의 데이터를 논리적으로 연관시킬 수 있습니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

여러 테이블 연관짓기

각 테이블은 정확히 하나의 엔티티를 설명하는 데이터를 포함합니다. 예를 들어, EMPLOYEES 테이블은 사원에 대한 정보를 포함합니다. 각 테이블 상단에는 데이터의 범주가 나열되고 각각의 예가 아래에 나열됩니다. 테이블 형식을 사용하면 정보를 쉽게 읽고 이해하고 사용할 수 있도록 시각적으로 나타낼 수 있습니다.

다른 엔티티에 대한 데이터는 다른 테이블에 저장되기 때문에 특정 질문에 답하기 위해 두 개 이상의 테이블을 결합해야 할 수도 있습니다. 예를 들어, 사원이 근무하는 부서의 위치를 알아보려고 합니다. 이 시나리오에서는 사원에 대한 데이터가 있는 EMPLOYEES 테이블과 부서에 대한 정보가 있는 DEPARTMENTS 테이블의 정보가 필요합니다. RDBMS에서는 Foreign key를 사용하여 한 테이블의 데이터를 다른 테이블의 데이터와 연관시킬 수 있습니다. Foreign key는 동일한 테이블이나 다른 테이블의 Primary key를 참조하는 열(또는 열 집합)입니다.

한 테이블의 데이터를 다른 테이블의 데이터와 연관시키는 기능을 사용하여 별도의 관리 단위로 정보를 구성할 수 있습니다. 사원 데이터를 별도의 테이블에 저장하는 방식으로 부서 데이터와 논리적으로 구분할 수 있습니다.

여러 테이블 연관짓기(계속)

Primary Key 및 Foreign Key에 대한 지침

- Primary key에서 중복 값을 사용할 수 없습니다.
- Primary key는 일반적으로 변경할 수 없습니다.
- Foreign key는 데이터 값을 기반으로 하며 순전히 논리적(물리적 아님) 포인터입니다.
- Foreign key 같은 기존 Primary key 값이나 Unique key 값과 일치해야 하며 그렇지 않은 경우 null이어야 합니다.
- Foreign key는 Primary key나 Unique key 열을 참조해야 합니다.

관계형 데이터베이스 용어

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	90
101	Neena	Kochhar	17000	(null)	90
102	Lex	De Haan	17000	(null)	90
103	Alexander	Hunold	9000	(null)	60
104	Bruce	Ernst	6000	(null)	60
107	Diana	Lorentz	4200	(null)	60
124	Kevin	Mourgos	5800	(null)	50
141	Trenna	Rajs	3500	(null)	50
142	Curtis	Davies	3100	(null)	50
143	Randall	Matos	2600	(null)	50
144	Peter	Vargas	2500	(null)	50
149	Eleni	Zlotkey	10500	0.2	80
174	Ellen	Abel	11000	0.3	80
176	Jonathon	Taylor	8600	0.2	80
178	Kimberely	Grant	7000	0.15	(null)
200	Jennifer	Whalen	4400	(null)	10
201	Michael	Hartstein	13000	(null)	20
202	Pat	Fay	6000	(null)	20
205	Shelley	Higgins	12000	(null)	110
206	William	Gietz	8300	(null)	110

ORACLE

Copyright © 2007, Oracle. All rights reserved.

관계형 데이터베이스 용어

관계형 데이터베이스는 하나 이상의 테이블을 포함할 수 있습니다. 테이블은 RDBMS의 기본 저장 구조입니다. 테이블은 사원, 견적서, 고객 등과 같이 현실 세계의 대상에 대해 필요한 모든 데이터를 보유합니다.

슬라이드는 EMPLOYEES 테이블 또는 관계의 내용을 보여줍니다. 각 숫자는 다음 내용을 나타냅니다.

- 특정 사원에 필요한 모든 데이터를 나타내는 단일 행(또는 투플)입니다. 테이블의 각 행은 중복 행을 허용하지 않는 Primary key로 식별되어야 합니다. 행의 순서는 중요하지 않습니다. 데이터를 검색할 때 행 순서를 지정합니다.
- 사원 번호를 포함하는 열 또는 속성입니다. 사원 번호는 EMPLOYEES 테이블에서 고유한 사원을 식별합니다. 이 예제에서 사원 번호 열은 Primary key로 지정됩니다. Primary key는 값을 포함해야 하고 그 값은 고유해야 합니다.
- 키 값이 아닌 열입니다. 열은 테이블에 있는 일종의 데이터를 나타냅니다. 이 예제에서 데이터는 모든 사원의 급여입니다. 데이터를 저장할 때 열 순서는 중요하지 않습니다. 데이터를 검색할 때 열 순서를 지정합니다.

관계형 데이터베이스 용어(계속)

4. 부서 번호를 포함하는 열이며 *Foreign key*입니다. *Foreign key*는 테이블이 서로 연관되는 방식을 정의하는 열입니다. *Foreign key*는 동일한 테이블이나 다른 테이블에 있는 Primary key나 Unique key를 참조합니다. 예제에서 DEPARTMENT_ID는 DEPARTMENTS 테이블에서 부서를 고유하게 식별합니다.
5. 필드는 행과 열의 교차점에서 찾을 수 있습니다. 각 필드는 하나의 값만 가질 수 있습니다.
6. 필드에 값이 없을 수도 있습니다. 이를 null 값이라고 합니다. EMPLOYEES 테이블에서 판매 담당자 를을 가진 사원만 COMMISSION_PCT(commission) 필드에 값이 있습니다.

단원 내용

- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

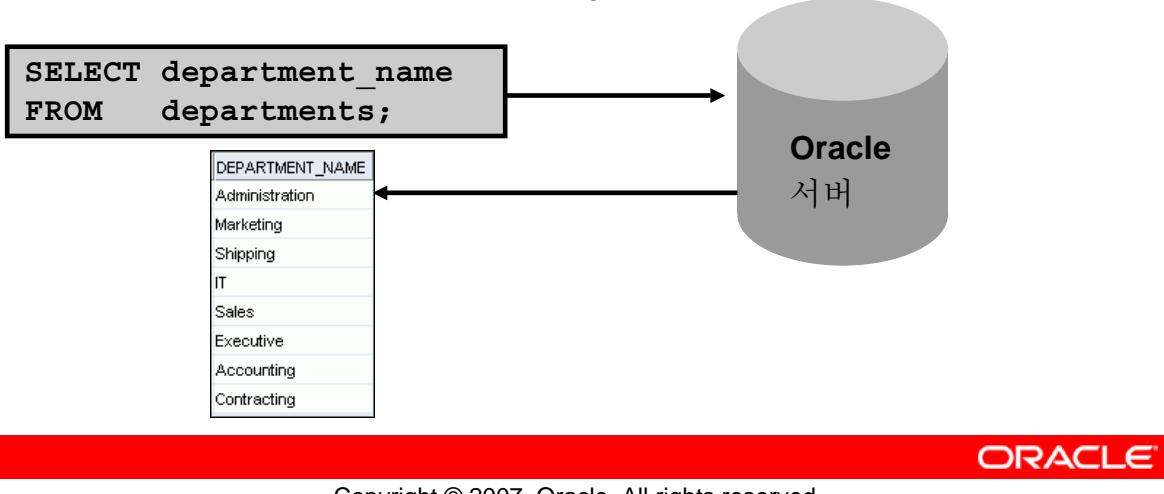
ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SQL을 사용하여 데이터베이스 query

SQL(구조적 질의어)은 다음과 같은 특징이 있습니다.

- 관계형 데이터베이스 작동을 위한 **ANSI** 표준 언어
- 효율적이며 쉽게 배워 사용할 수 있음
- 완벽한 기능(**SQL**을 사용하면 테이블의 데이터를 정의, 검색 및 조작할 수 있습니다.)



Copyright © 2007, Oracle. All rights reserved.

SQL을 사용하여 데이터베이스 query

관계형 데이터베이스에서는 테이블에 대한 액세스 경로를 지정하지 않으면 데이터가 물리적으로 정렬되는 방식을 알 필요가 없습니다.

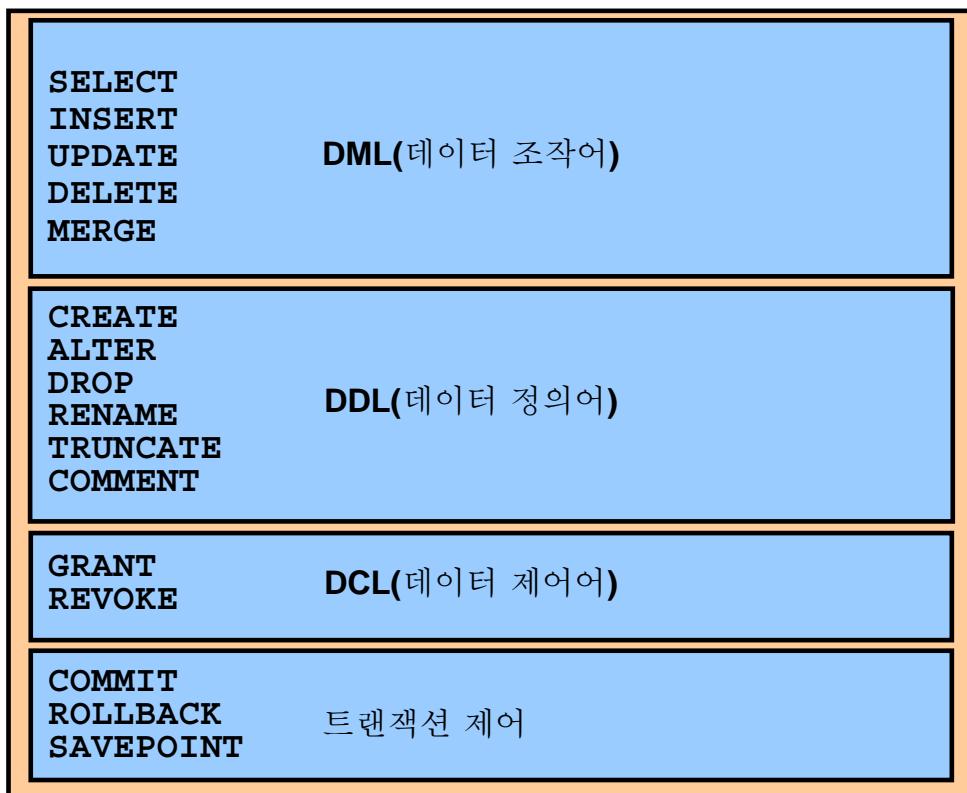
데이터베이스에 액세스하려면 관계형 데이터베이스 작동을 위한 ANSI(American National Standards Institute) 표준 언어인 SQL(구조적 질의어) 문을 실행합니다. SQL은 모든 프로그램 및 유저가 오라클 데이터베이스의 데이터에 액세스하기 위해 사용하는 일련의 명령문입니다. 응용 프로그램 및 Oracle 툴에서 유저가 SQL을 직접 사용하지 않고 데이터베이스에 액세스하도록 허용하는 경우가 있지만 이러한 응용 프로그램에서 유저의 요청을 실행할 때는 SQL을 사용해야 합니다.

SQL에서는 다음을 포함하여 다양한 작업에 대한 명령문을 제공합니다.

- 데이터 Query
- 테이블에서 행 삽입, 갱신 및 삭제
- 객체 생성, 대체, 변경 및 삭제
- 데이터베이스 및 해당 객체에 대한 액세스 제어
- 데이터베이스 일관성 및 무결성 보장

위에 설명된 모든 작업은 SQL이라는 하나의 일관된 언어로 표현할 수 있으며 유저는 SQL을 통해 논리적 레벨에서 데이터를 다룰 수 있습니다.

SQL 문



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL 문

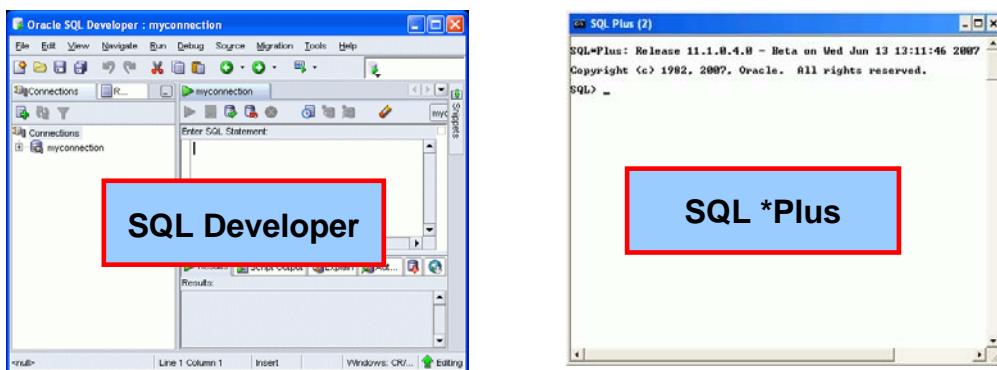
Oracle에서 지원하는 SQL 문은 산업 표준을 준수합니다. Oracle Corporation은 SQL 표준 위원회의 핵심 멤버로 적극 참여하여 앞으로 개발되는 표준도 준수할 것입니다. 업계 승인 위원회인 ANSI와 ISO(International Standards Organization)는 모두 SQL을 관계형 데이터베이스에 적합한 표준 언어로 승인했습니다.

명령문	설명
<code>SELECT</code> <code>INSERT</code> <code>UPDATE</code> <code>DELETE</code> <code>MERGE</code>	각각 데이터베이스에서 데이터를 검색하고, 새 행을 입력하고, 기존 행을 변경하고, 데이터베이스의 테이블에서 불필요한 행을 제거합니다. 이러한 명령문을 통틀어 DML(데이터 조작어)이라고 합니다.
<code>CREATE</code> <code>ALTER</code> <code>DROP</code> <code>RENAME</code> <code>TRUNCATE</code> <code>COMMENT</code>	테이블에서 데이터 구조를 설정, 변경, 제거합니다. 이러한 명령문을 통틀어 DDL(데이터 정의어)이라고 합니다.
<code>GRANT</code> <code>REVOKE</code>	오라클 데이터베이스와 그 안의 구조에 대한 액세스 권한을 부여하거나 제거합니다.
<code>COMMIT</code> <code>ROLLBACK</code> <code>SAVEPOINT</code>	DML 문으로 인한 변경 사항을 관리합니다. 데이터의 변경 사항은 논리적 트랜잭션으로 함께 그룹화 할 수 있습니다.

SQL 개발 환경

이 과정에서 사용하는 툴은 다음과 같습니다.

- **Oracle SQL Developer, Release 1.2**를 주로 사용
- 다음과 같은 경우에 **SQL*Plus** 사용
 - Oracle SQL Developer에 액세스할 수 없는 경우
 - 또는 Oracle SQL Developer에서 명령이 작동하지 않는 경우



Copyright © 2007, Oracle. All rights reserved.

SQL 개발 환경

이 과정은 슬라이드 및 연습의 예제에 설명된 SQL 문을 실행하는 툴로 Oracle SQL Developer를 사용하여 개발되었습니다. Oracle SQL Developer에서 지원하지 않는 명령의 경우 SQL*Plus 환경을 사용합니다.

단원 내용

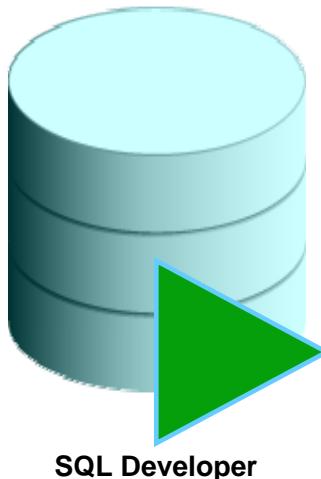
- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Oracle SQL Developer란?

- **Oracle SQL Developer**는 생산성을 향상하고 데이터베이스 개발 작업을 단순화하는 그래픽 툴입니다.
- 표준 오라클 데이터베이스 인증을 사용하여 대상 오라클 데이터베이스 스키마에 연결할 수 있습니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle SQL Developer란?

Oracle SQL Developer는 생산성을 향상하고 일상적인 데이터베이스 작업의 개발을 단순화하기 위해 설계된 무료 그래픽 툴입니다. 마우스를 몇 번 누르는 것만으로 간단하게 내장 프로시저를 생성 및 디버그하고, SQL 문을 테스트하고, 옵티마이저 계획을 볼 수 있습니다.

데이터베이스 개발을 위한 시각적 툴인 Oracle SQL Developer는 다음 작업을 단순화합니다.

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

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

주: Oracle SQL Developer, Release 1.2는 *Developer Migration Workbench*와 긴밀하게 통합되어 있으므로 *Migration release*로 불립니다. 따라서 타사 데이터베이스의 데이터베이스 객체 및 데이터를 탐색하고 해당 데이터베이스에서 Oracle로 이전할 수 있는 단일 지점을 유저에게 제공합니다. 또한 MySQL, Microsoft SQL Server 및 Microsoft Access 등의 타사 데이터베이스를 선택하고 스키마에 연결하여 이러한 데이터베이스의 메타 데이터 및 데이터를 볼 수 있습니다.

Oracle SQL Developer, Release 1.2에서는 Oracle APEX(Oracle Application Express), Release 3.0.1도 지원합니다.

Oracle SQL Developer 사양

- Java로 개발됨
- Windows, Linux 및 Mac OS X 플랫폼 지원
- JDBC Thin 드라이버를 사용하는 기본 연결
- 설치 프로그램이 필요 없음
 - 다운로드한 Oracle SQL Developer 키트의 압축을 푼 다음 `sqldeveloper.exe`를 두 번 눌러 Oracle SQL Developer를 시작합니다.
- Oracle Database, 9.2.0.1 및 이후 버전에 연결
- 다음 링크에서 무료 다운로드 가능
 - http://www.oracle.com/technology/products/database/sql_developer/index.html
- 다음 링크에서 다운로드 가능한 JDK 1.5가 컴퓨터에 설치되어 있어야 함
 - http://java.sun.com/javase/downloads/index_jdk5.jsp

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle SQL Developer 사양

Oracle SQL Developer는 Oracle JDeveloper IDE(통합 개발 환경)를 활용하여 Java로 개발되었으며 플랫폼 간에 호환되는 툴입니다. 이 툴은 Windows, Linux 및 Mac OS(운영 체제) X 플랫폼에서 실행됩니다. Oracle SQL Developer를 데이터베이스 서버에 설치하고 데스크톱에서 원격으로 연결할 수 있으므로 클라이언트/서버 네트워크 트래픽을 방지할 수 있습니다.

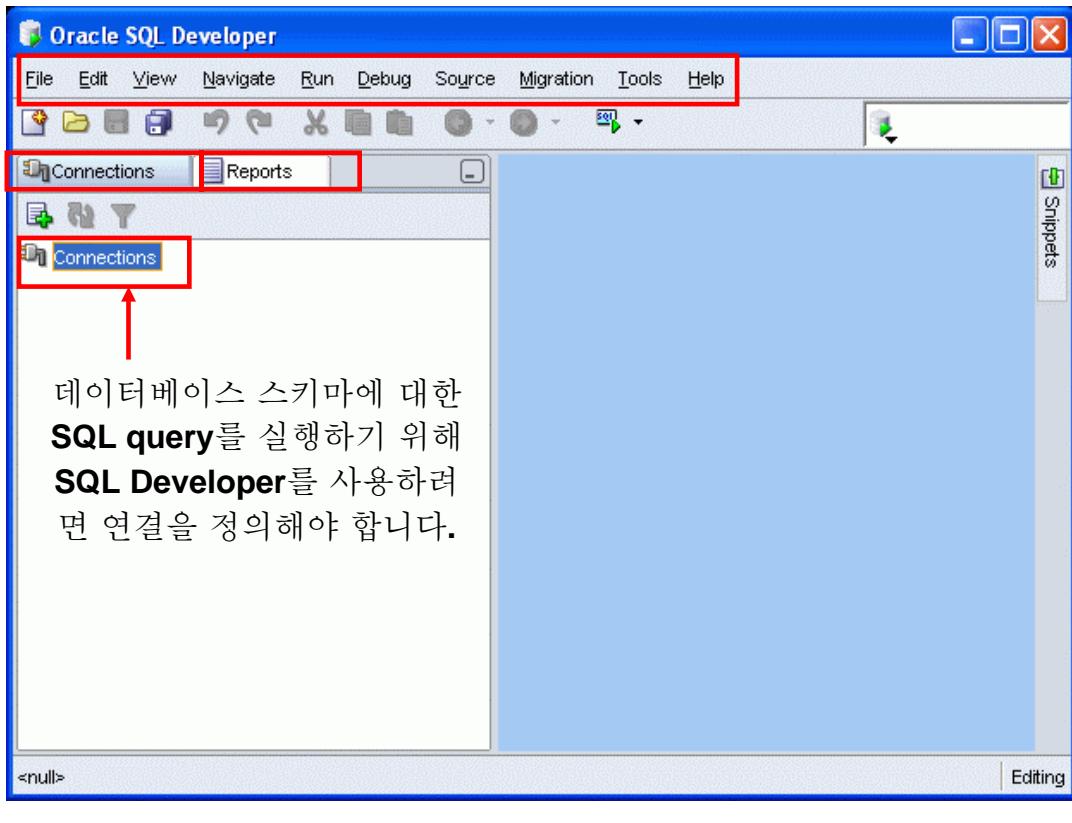
데이터베이스에 대한 기본 연결은 JDBC(Java Database Connectivity) Thin 드라이버를 통해 이루어 지므로 Oracle Home이 필요하지 않습니다. Oracle SQL Developer는 설치 프로그램이 필요 없으며 다운로드한 파일의 압축을 풀기만 하면 됩니다. 유저는 Oracle SQL Developer를 사용하여 Oracle Database 9.2.0.1 이상 버전의 Express Edition을 포함하는 모든 Oracle Database Edition에 연결할 수 있습니다.

Oracle SQL Developer는 다음 패키지 옵션으로 다운로드할 수 있습니다.

- Windows용 Oracle SQL Developer(JDK(Java Development Kit) 1.5 포함은 옵션)
- 다중 플랫폼용 Oracle SQL Developer(JDK 1.5가 설치되어 있어야 함)
- Mac OS X 플랫폼용 Oracle SQL Developer(JDK 1.5가 설치되어 있어야 함)
- Linux용 Oracle SQL Developer RPM(JDK 1.5가 설치되어 있어야 함)

Oracle SQL Developer, Release 1.2는 JDK 6.0이 설치된 컴퓨터에서도 실행됩니다.

Oracle SQL Developer 인터페이스



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle SQL Developer 인터페이스

Oracle SQL Developer에는 두 개의 기본 탐색 템이 있습니다.

- **Connections 템:** 이 템을 사용하면 액세스할 수 있는 데이터베이스 객체 및 유저를 찾아볼 수 있습니다.
- **Reports 템:** 이 템을 사용하면 미리 정의된 보고서를 실행하거나 자신의 보고서를 작성 및 추가할 수 있습니다.

Oracle SQL Developer에서는 왼쪽 창을 탐색에 사용하여 객체를 찾고 선택하며 오른쪽 창을 사용하여 선택한 객체에 대한 정보를 표시합니다. 환경설정을 통해 Oracle SQL Developer의 모양 및 동작에 대한 여러 측면을 커스터마이즈할 수 있습니다. 상단의 메뉴에는 표준 항목과 Oracle SQL Developer 전용 기능을 위한 추가 항목이 포함되어 있습니다.

1. **View:** Oracle SQL Developer 인터페이스에 표시되는 항목에 영향을 주는 옵션을 포함합니다.
2. **Navigate:** 창으로의 이동 및 서브 프로그램 실행에 대한 옵션을 포함합니다.
3. **Run:** 함수 또는 프로시저를 선택할 때 관련되는 Run File 및 Execution Profile 옵션을 포함합니다.
4. **Debug:** 함수 또는 프로시저를 디버깅하기 위해 선택할 때 관련되는 옵션을 포함합니다.
5. **Source:** 함수 및 프로시저를 편집할 때 사용할 옵션을 포함합니다.
6. **Migration:** 타사 데이터베이스를 Oracle로 마이그레이션할 때 관련되는 옵션을 포함합니다.
7. **Tools:** SQL*Plus, Preferences 및 SQL Worksheet와 같은 툴을 호출합니다.

주: 데이터베이스 스키마에 연결하고 SQL query를 실행하거나 프로시저/함수를 실행하려면 최소한 하나의 연결을 정의해야 합니다.

데이터베이스 연결 생성

- **Oracle SQL Developer**를 사용하려면 최소한 하나의 데이터베이스 연결이 있어야 합니다.
- 다음 대상에 대해 연결을 생성하고 테스트할 수 있습니다.
 - 하나 이상의 데이터베이스
 - 하나 이상의 스키마
- **Oracle SQL Developer**는 시스템의 **tnsnames.ora** 파일에 정의된 모든 연결을 자동으로 임포트합니다.
- **XML** 파일로 연결을 엑스포트할 수 있습니다.
- 추가로 생성된 각각의 데이터베이스 연결은 **Connections Navigator** 계층에 나열됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 연결 생성

연결은 특정 데이터베이스의 특정 유저로서 해당 데이터베이스에 연결하는 데 필요한 정보를 지정하는 Oracle SQL Developer 객체입니다. Oracle SQL Developer를 사용하려면 기존 연결, 생성된 연결, 임포트된 연결 등의 데이터베이스 연결이 최소한 하나 있어야 합니다.

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

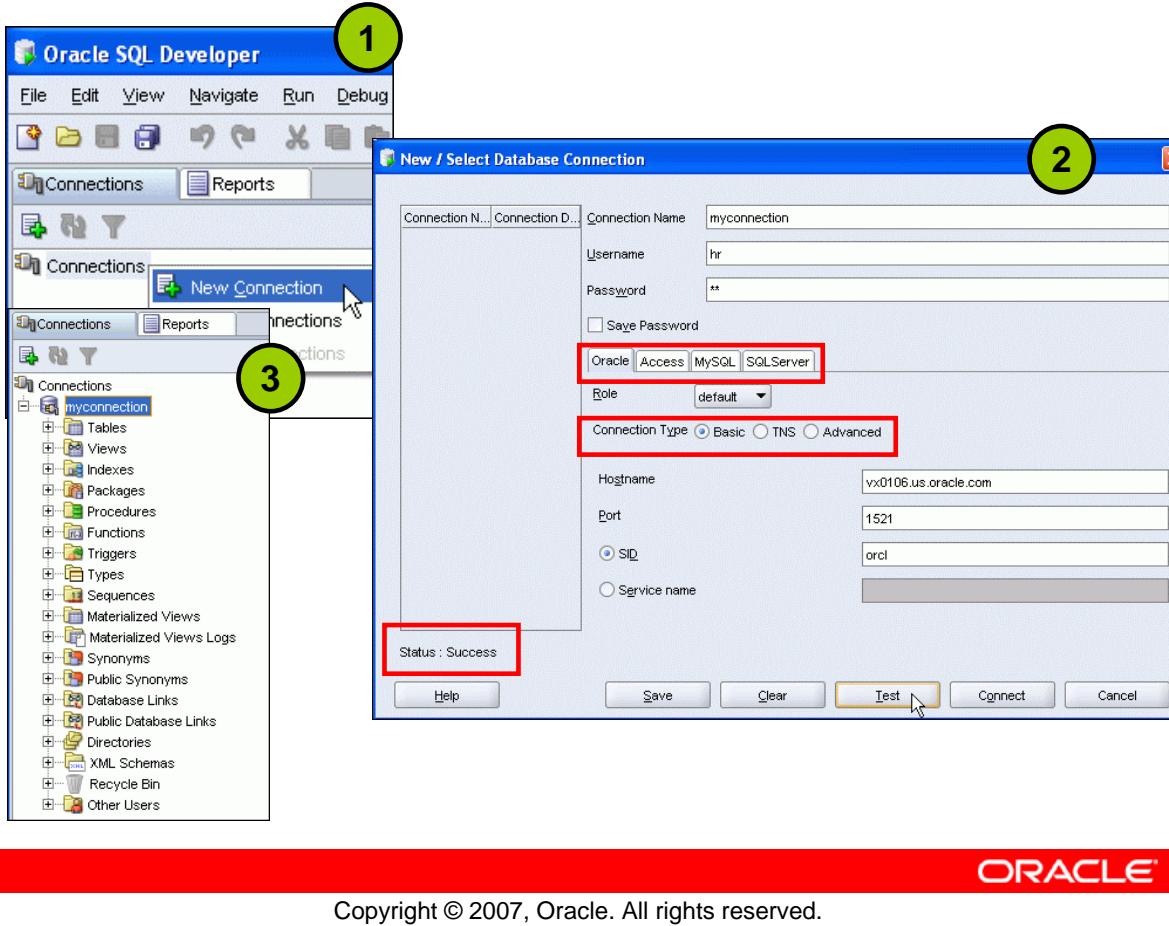
기본적으로 **tnsnames.ora** 파일은 \$ORACLE_HOME/network/admin 디렉토리에 위치합니다. 그러나 TNS_ADMIN 환경 변수 또는 레지스트리 값에 지정된 디렉토리에 있을 수도 있습니다. Oracle SQL Developer를 시작하고 Database Connections 대화상자를 표시하면 Oracle SQL Developer는 컴퓨터의 **tnsnames.ora** 파일에 정의된 모든 연결을 자동으로 임포트합니다.

주: Windows 시스템에 **tnsnames.ora** 파일이 존재하지만 Oracle SQL Developer에서 해당 연결을 사용하지 않는 경우 TNS_ADMIN을 시스템 환경 변수로 정의합니다.

연결을 XML 파일로 엑스포트하여 나중에 재사용할 수 있습니다.

동일한 데이터베이스에 다른 유저로 연결하거나 다른 데이터베이스에 연결하도록 추가 연결을 생성할 수 있습니다.

데이터베이스 연결 생성



ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 연결 생성(계속)

데이터베이스 연결을 생성하려면 다음 단계를 수행하십시오.

1. Connections 탭 페이지에서 Connections를 마우스 오른쪽 버튼으로 누른 다음 New Connection을 선택합니다.
2. New>Select Database Connection window에 연결 이름을 입력합니다. 연결하려는 스키마의 유저 이름 및 암호를 입력합니다.
 1. Role drop-down list에서 *default* 또는 SYSDBA를 선택할 수 있습니다. sys 유저 또는 DBA 권한이 있는 모든 유저에 대해서는 SYSDBA를 선택합니다.
 2. 연결 유형은 다음과 같이 선택할 수 있습니다.
 - Basic: 이 유형에서는 연결하려는 데이터베이스의 호스트 이름 및 SID(시스템 식별자)를 입력합니다. 포트는 이미 1521로 설정되어 있으며 원격 데이터베이스 연결을 사용하는 경우 서비스 이름을 직접 입력할 수도 있습니다.
 - TNS: Tnsnames.ora 파일에서 임포트한 데이터베이스 alias 중 하나를 선택합니다.
 - Advanced: 데이터베이스 연결을 위한 커스텀 JDBC URL을 정의합니다.
3. Test를 눌러 연결이 올바르게 설정되었는지 확인합니다.
4. Connect를 누릅니다.

Save Password 체크 박스를 선택하면 암호가 XML 파일에 저장됩니다. 따라서 Oracle SQL Developer 연결을 닫은 후 다시 열면 암호 입력 프롬프트가 나타나지 않습니다.

데이터베이스 연결 생성(계속)

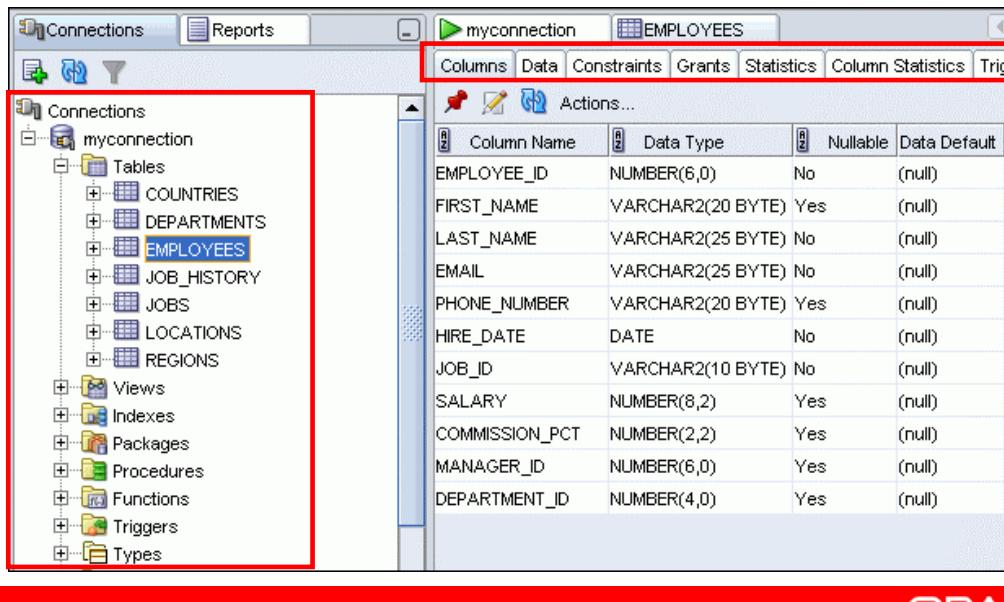
3. Connections Navigator에 연결이 추가됩니다. 연결을 확장하여 데이터베이스 객체를 볼 수 있으며 종속성, 상세 내역, 통계 등의 객체 정의를 볼 수 있습니다.

주: 동일한 New>Select Database Connection window에서 Access, MySQL 및 SQL Server 탭을 사용하여 타사 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 © 2007, Oracle. All rights reserved.

데이터베이스 객체 탐색

데이터베이스 연결을 생성한 후 **Connections Navigator**를 사용하여 데이터베이스 스키마에서 테이블, 뷰, 인덱스, 패키지, 프로시저, 트리거, 유형 등의 여러 객체를 탐색할 수 있습니다.

Oracle SQL Developer에서는 왼쪽 창을 탐색에 사용하여 객체를 찾고 선택하며 오른쪽 창을 사용하여 선택한 객체에 대한 정보를 표시합니다. 환경 설정을 통해 Oracle SQL Developer의 모양에 대한 여러 측면을 커스터마이즈할 수 있습니다.

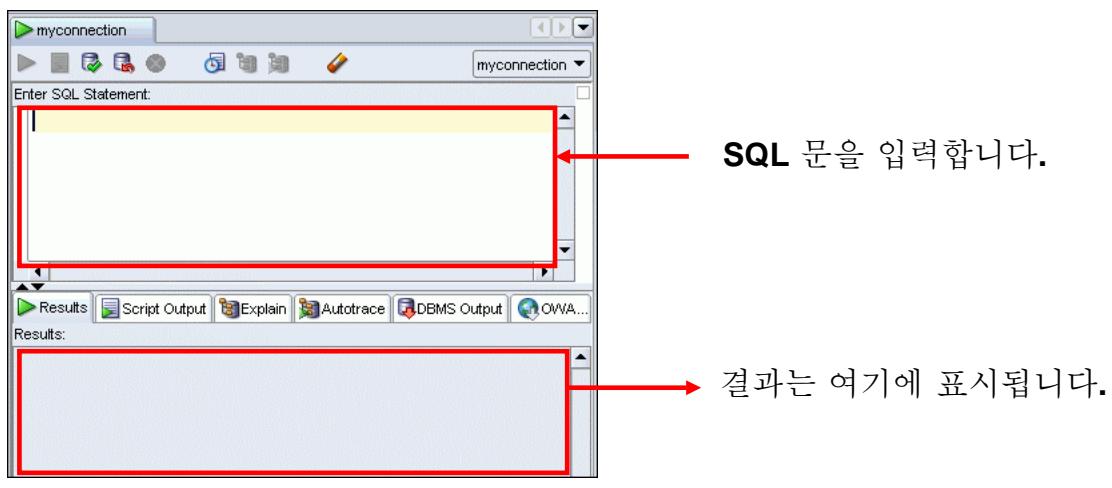
데이터 덕셔너리에서 추출된 객체 정의가 여러 정보 탭에 분할되어 있는 것을 볼 수 있습니다. 예를 들어, Navigator에서 테이블을 선택하면 열, 제약 조건, 권한 부여, 통계, 트리거 등에 대한 상세 내역이 읽기 쉬운 탭 페이지에 표시됩니다.

슬라이드에서처럼 EMPLOYEES 테이블의 정의를 보려면 다음 단계를 수행합니다.

- Connections Navigator에서 Connections 노드를 확장합니다.
- Tables를 확장합니다.
- EMPLOYEES를 누릅니다. 기본적으로 Columns 탭이 선택됩니다. 테이블의 열 설명이 표시됩니다. Data 탭을 사용하여 테이블 데이터를 볼 수 있으며 또한 새 행을 입력하고 데이터를 갱신하며 이러한 변경 사항을 데이터베이스에 커밋할 수 있습니다.

SQL Worksheet 사용

- **SQL Worksheet**를 사용하여 **SQL**, **PL/SQL** 및 **SQL*Plus** 문을 입력하고 실행합니다.
- **Worksheet**와 관련된 데이터베이스 연결에서 처리될 수 있는 모든 작업을 지정합니다.



Copyright © 2007, Oracle. All rights reserved.

ORACLE

SQL Worksheet 사용

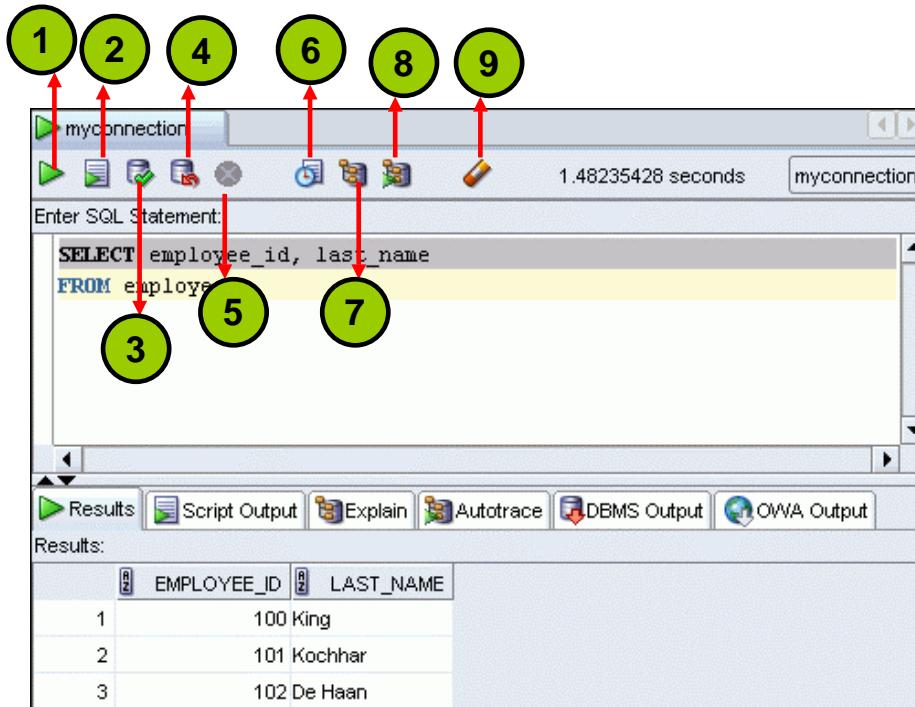
데이터베이스에 연결하면 해당 연결에 대한 SQL Worksheet window가 자동으로 열립니다. SQL Worksheet을 사용하여 SQL, PL/SQL 및 SQL*Plus 문을 입력하고 실행할 수 있습니다. 모든 SQL 및 PL/SQL 명령이 지원되며 이러한 명령은 SQL Worksheet에서 오라클 데이터베이스로 직접 전달됩니다. 그러나 Oracle SQL Developer에서 사용되는 SQL*Plus 명령은 데이터베이스로 전달되기 전에 SQL Worksheet에서 해석되어야 합니다.

현재 SQL Worksheet에서는 다수의 SQL*Plus 명령을 지원합니다. SQL Worksheet에서 지원하지 않는 명령은 무시되며 오라클 데이터베이스로 전달되지 않습니다. SQL Worksheet을 통해 SQL 문을 실행하거나 SQL*Plus 명령 중 일부를 실행할 수 있습니다.

다음 두 가지 옵션 중 하나를 사용하여 SQL Worksheet을 표시할 수 있습니다.

- Tools > SQL Worksheet를 선택합니다.
- 기본 도구 모음에 있는 Open SQL Worksheet 아이콘을 누릅니다.

SQL Worksheet 사용



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL Worksheet 사용(계속)

단축키 또는 단축 아이콘을 사용하여 SQL 문 실행, 스크립트 실행 또는 실행한 SQL 문 기록 보기와 같은 특정 작업을 수행할 수 있습니다.

여러 아이콘이 있는 SQL Worksheet 도구 모음을 사용하여 다음 작업을 수행할 수 있습니다.

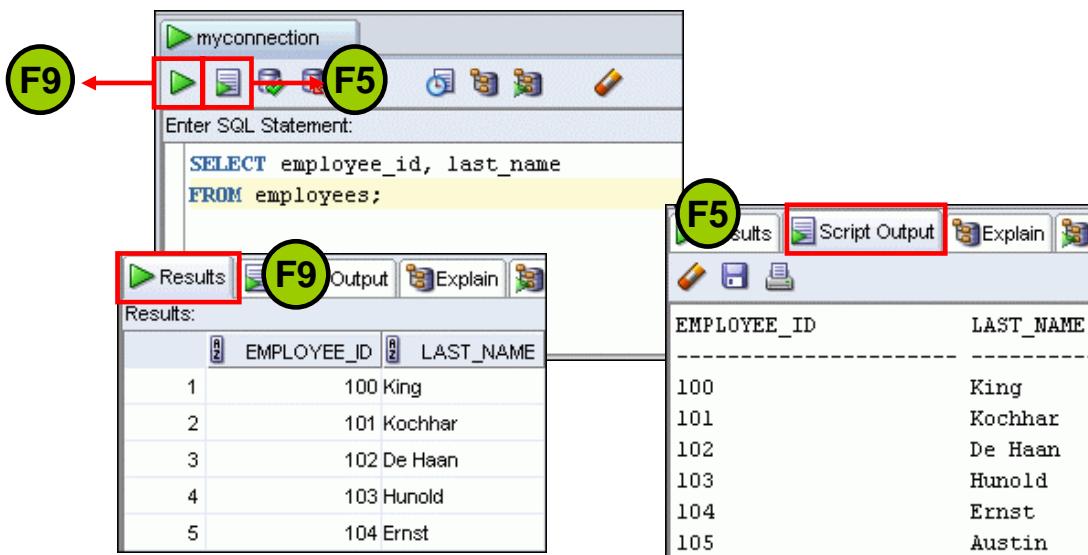
1. **Execute Statement:** Enter SQL Statement 상자에서 커서 위치에 있는 명령문을 실행합니다. 또는 [F9]를 누를 수도 있습니다. 일반적으로 출력은 지정된 형식에 따라 Results 탭 페이지에 표시됩니다.
2. **Run Script:** Script Runner를 사용하여 Enter SQL Statement 상자에 있는 모든 명령문을 실행합니다. 출력은 일반적으로 Scripts 탭 페이지에 기본 스크립트 형식으로 표시됩니다.
3. **Commit:** 모든 변경 사항을 데이터베이스에 기록하고 트랜잭션을 종료합니다.
4. **Rollback:** 데이터베이스의 모든 변경 사항을 무시하고 기록하지 않으며 트랜잭션을 종료합니다.
5. **Cancel:** 현재 실행 중인 모든 명령문의 실행을 정지합니다.

SQL Worksheet 사용(계속)

6. **SQL History:** 실행되었던 SQL 문에 대한 정보를 포함하는 대화상자를 표시합니다.
7. **Execute Explain Plan:** Explain 탭을 눌러 볼 수 있는 실행 계획을 생성합니다.
8. **Autotrace:** Autotrace 아이콘을 눌러 SQL 문을 실행하는 경우 추적 관련 정보가 표시됩니다. 이 정보를 사용하면 튜닝을 통해 개선할 수 있는 SQL 문을 식별할 수 있습니다.
9. **Clear:** Enter SQL Statement 상자의 명령문을 지웁니다. 또는 [Ctrl]+[D]를 누르고 있어도 명령문을 지울 수 있습니다.

SQL 문 실행

Enter SQL Statement 상자를 사용하여 하나 이상의 **SQL** 문을 입력합니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL 문 실행

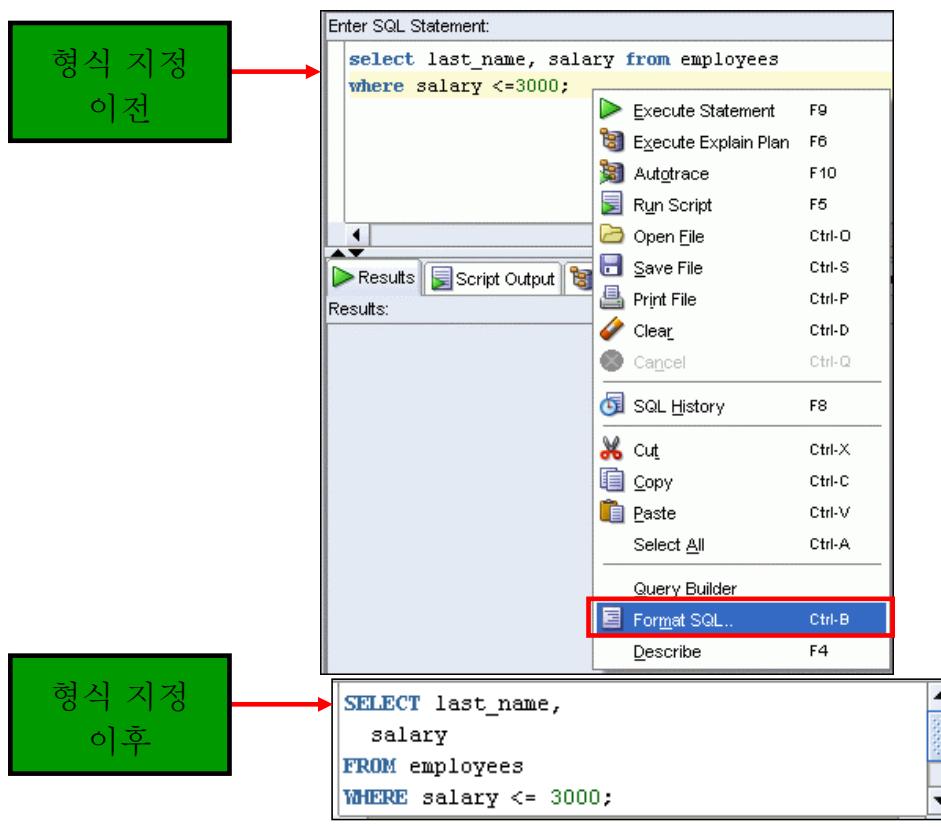
SQL Worksheet에서 Enter SQL Statement 상자를 사용하여 하나 이상의 SQL 문을 입력할 수 있습니다. 단일 명령문의 경우 끝에 세미콜론을 입력하지 않아도 됩니다.

명령문을 입력하면 SQL 키워드가 자동으로 강조 표시됩니다. SQL 문을 실행하려면 커서가 해당 명령문 내에 있는지 확인하고 Execute Statement 아이콘을 누릅니다. 또는 [F9]를 누를 수도 있습니다.

여러 SQL 문을 실행하고 결과를 보려면 Run Script 아이콘을 누릅니다. 또는 [F5]를 누를 수도 있습니다.

슬라이드의 예제에서는 동일한 query에 대해 F9 키 또는 Execute Statement를 사용한 출력과 F5 키 또는 Run Script를 사용한 출력의 차이를 보여줍니다.

SQL 코드 형식 지정



ORACLE

Copyright © 2007, Oracle. All rights reserved.

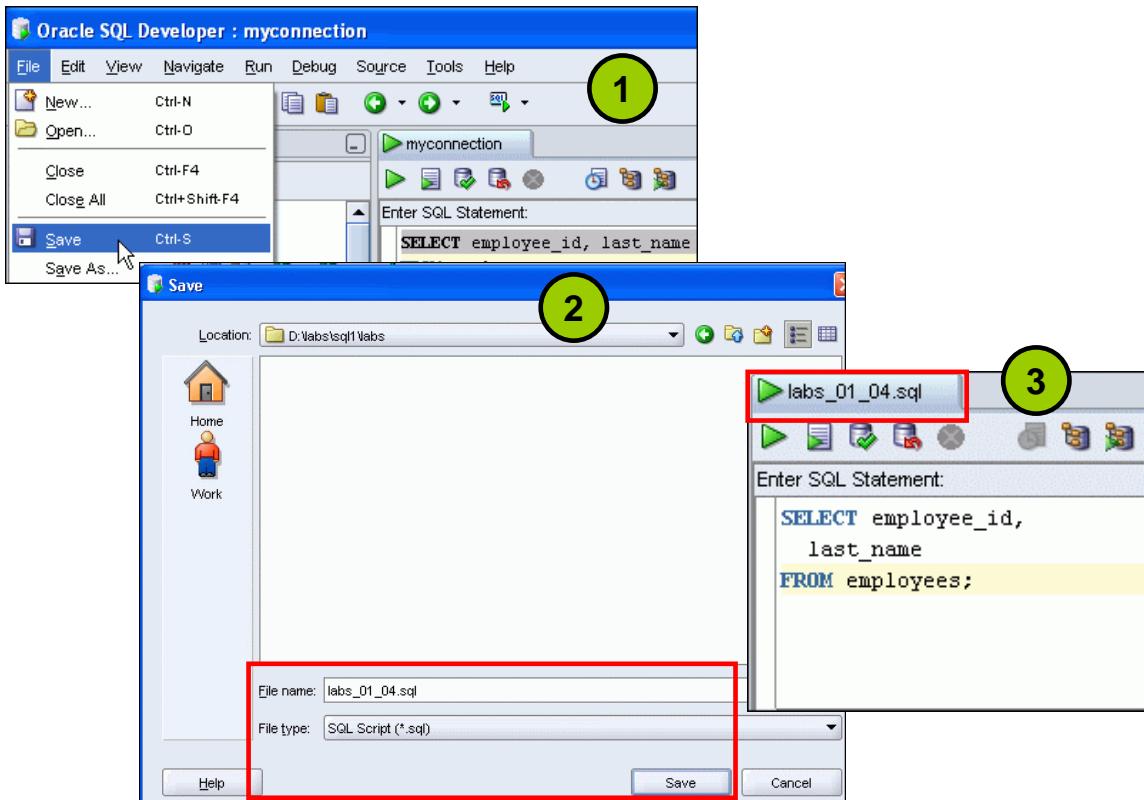
SQL 코드 형식 지정

SQL 코드의 들여쓰기, 간격, 대문자 표시 및 행 구분을 보기 좋게 다듬을 수 있습니다. Oracle SQL Developer에는 SQL 코드 형식 지정을 위한 기능이 있습니다.

SQL 코드에 형식을 지정하려면 명령문 영역을 마우스 오른쪽 버튼으로 누른 다음 Format SQL을 선택합니다.

슬라이드의 예제에서 형식 지정 이전의 SQL 코드에서는 키워드가 대문자로 표시되어 있지 않고 명령문이 적절하게 들여쓰기되어 있지 않습니다. 형식 지정 이후의 SQL 코드에서는 키워드가 대문자로 표시되고 명령문이 적절히 들여쓰기되어 보기 좋게 다듬어졌습니다.

SQL 문 저장



ORACLE

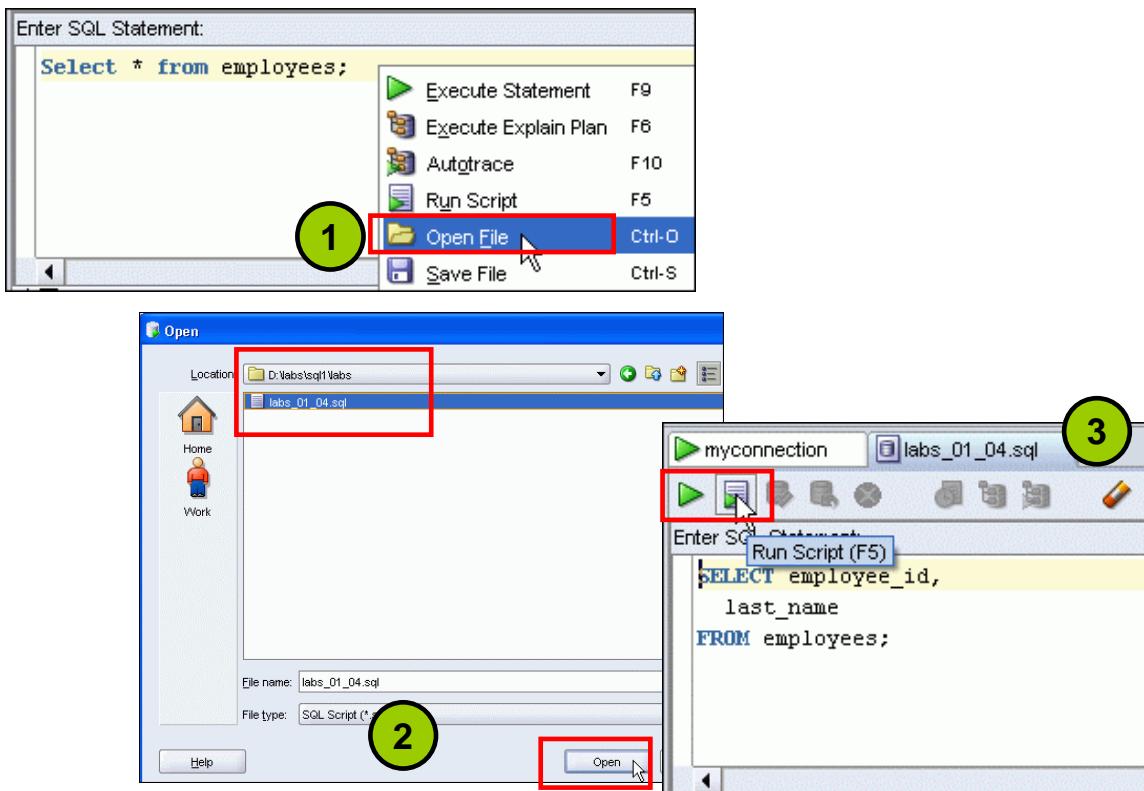
Copyright © 2007, Oracle. All rights reserved.

SQL 문 저장

앞으로 수행하는 대부분의 연습에서는 SQL Worksheet의 특정 query를 .sql 파일로 저장해야 합니다. 이를 위해 다음을 수행하십시오.

1. File 메뉴에서 Save 또는 Save As(현재 .sql 스크립트의 이름을 변경하는 경우)를 선택하거나 SQL Worksheet를 마우스 오른쪽 버튼으로 누르고 Save File을 선택합니다. 또는 [CTRL]+[S]를 누르고 있어도 됩니다.
2. Save 대화상자에 적절한 파일 이름을 입력합니다. 확장자가 .sql이거나 파일 유형이 SQL Script (*.sql)로 선택되어 있는지 확인합니다. Save를 누릅니다.
주: 이 과정에서는 sql 스크립트를 D:\labs\sql1\labs 폴더에 저장해야 합니다.
3. SQL Worksheet의 이름이 스크립트를 저장한 파일 이름으로 변경됩니다. 동일한 워크시트에 다른 SQL 문을 입력하지 않도록 합니다. 다른 SQL query를 계속하려면 새 워크시트를 엽니다.

스크립트 파일 실행



ORACLE

Copyright © 2007, Oracle. All rights reserved.

스크립트 파일 실행

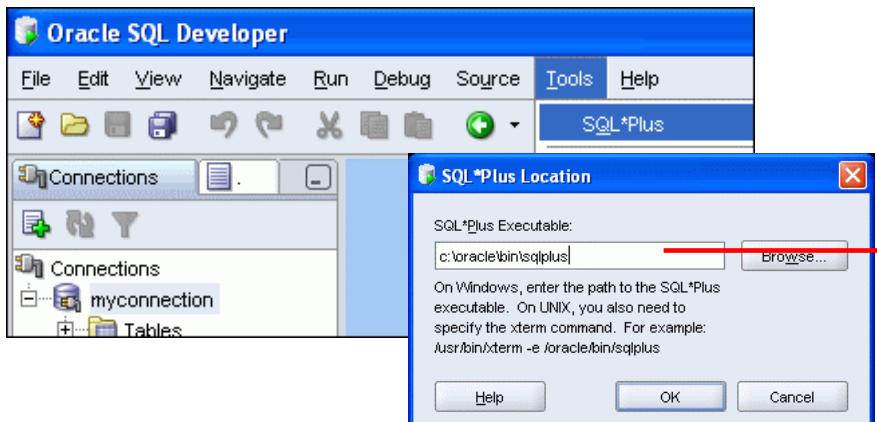
저장된 .sql 스크립트 파일을 실행하려면 다음을 수행합니다.

1. SQL Worksheet를 마우스 오른쪽 버튼으로 누른 다음 Open File을 선택하거나 File 메뉴에서 Open을 선택합니다. 또는 [CTRL]+[O]를 누르고 있어도 됩니다.
2. Open 대화상자에서 D:\labs\sql\labs 폴더 또는 스크립트 파일을 저장한 위치로 이동한 다음 파일을 선택하고 Open을 누릅니다.
3. 스크립트 파일이 새 워크시트에 열립니다. 이제 Execute Statement 아이콘 또는 Run Script 아이콘을 눌러 스크립트를 실행할 수 있습니다. 앞에서 언급했듯이 동일한 워크시트에 다른 SQL 문을 입력하지 않도록 합니다. 다른 SQL query를 계속하려면 새 워크시트를 엽니다.
주: 기본 디렉토리를 D:\labs\sql1 폴더로 설정하면 스크립트를 열거나 저장할 때마다 SQL Developer에서 동일한 경로를 선택하여 스크립트를 검색하도록 할 수 있습니다. Tools 메뉴에서 Preferences를 선택합니다. Preferences 대화상자에서 Database를 확장하고 Worksheet Parameters를 선택합니다. 오른쪽 창에서 Browse를 눌러 스크립트를 검색할 기본 경로를 설정한 다음 OK를 누릅니다.

주: 다른 데이터 객체 생성 및 데이터 검색 작업에 Oracle SQL Developer GUI 인터페이스를 사용하는 방법에 대한 자세한 내용은 부록 G "Oracle SQL Developer GUI를 사용하여 DML 및 DDL 작업 수행"을 참조하십시오.

Oracle SQL Developer에서 SQL*Plus 시작

Oracle SQL Developer에서 SQL*Plus 명령행 인터페이스를 호출할 수 있습니다.



SQL*Plus를 처음
호출하는 경우에만
sqlplus.exe
파일의 위치를
제공합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle SQL Developer에서 SQL*Plus 시작

SQL Worksheet에서는 대부분의 SQL*Plus 문을 지원합니다. SQL*Plus 문은 데이터베이스로 전달되기 전에 SQL Worksheet에서 해석되어야 하며 SQL Worksheet에서 지원하지 않는 모든 SQL*Plus 문은 무시되어 데이터베이스로 전달되지 않습니다. SQL*Plus 명령 window를 표시하려면 Tools 메뉴에서 SQL*Plus를 선택합니다. 이 기능을 사용하려면 Oracle SQL Developer를 사용하는 컴퓨터에 Oracle 홈 디렉토리 또는 폴더가 있어야 하며 SQL*Plus 실행 파일이 이 폴더에 있어야 합니다. SQL*Plus 실행 파일의 위치가 Oracle SQL Developer 환경 설정에 이미 저장되어 있지 않으면 위치를 지정해야 합니다.

주: Tools > SQL*Plus 메뉴 옵션이 비활성화되어 있으면 Connections Navigator에서 myconnection과 같은 데이터베이스 연결을 누릅니다. SQL Worksheet가 활성 상태이면 메뉴 옵션이 비활성화됩니다.

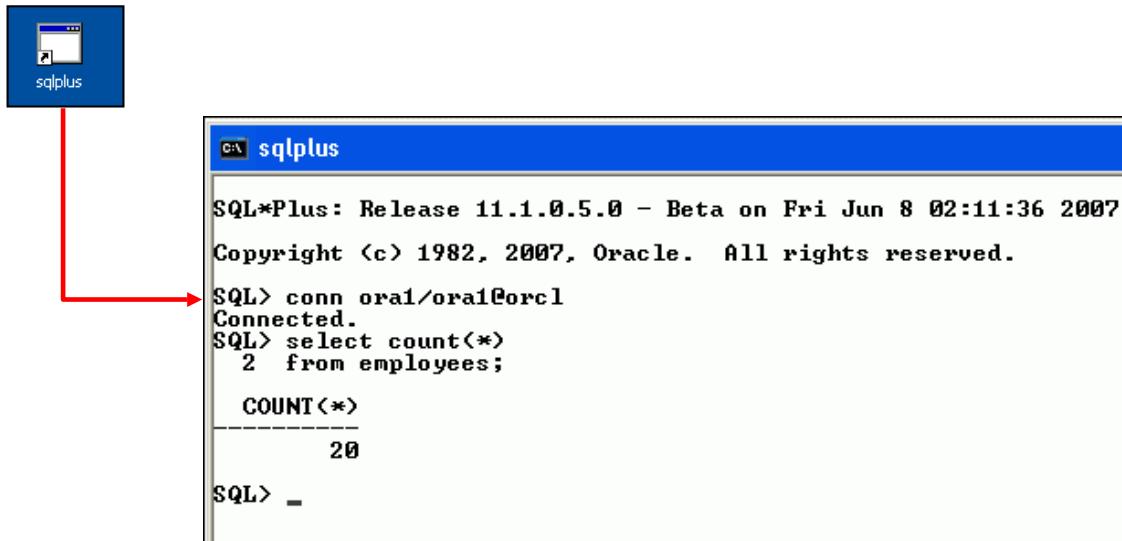
예를 들어, SQL Worksheet에서 지원하지 않는 SQL*Plus 문은 다음과 같습니다.

- append
- archive
- attribute
- break

SQL Worksheet에서 지원하는 SQL*Plus 문과 지원하지 않는 SQL*Plus 문의 전체 리스트는 Oracle SQL Developer 온라인 도움말의 *SQL Worksheet*에서 지원하는 SQL*Plus 문과 지원하지 않는 SQL*Plus 문 항목을 참조하십시오.

SQL*Plus의 SQL 문

Oracle Database 11g에서 SQL*Plus는 명령행 인터페이스입니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL*Plus의 SQL 문

Oracle SQL*Plus는 SQL 문 및 PL/SQL 블록을 제출하여 실행하고 결과를 응용 프로그램 또는 명령 window에서 전달받을 수 있는 명령행 인터페이스입니다.

SQL*Plus는 다음 특징을 갖습니다.

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

주: Oracle SQL Developer에 액세스할 수 없고 SQL*Plus를 사용하려는 경우 강의실 설정에서는 바탕 화면에 SQL*Plus 아이콘을 제공합니다. 이 아이콘은 Oracle SQL Developer에서 SQL*Plus 명령을 지원하지 않는 경우에도 유용합니다.

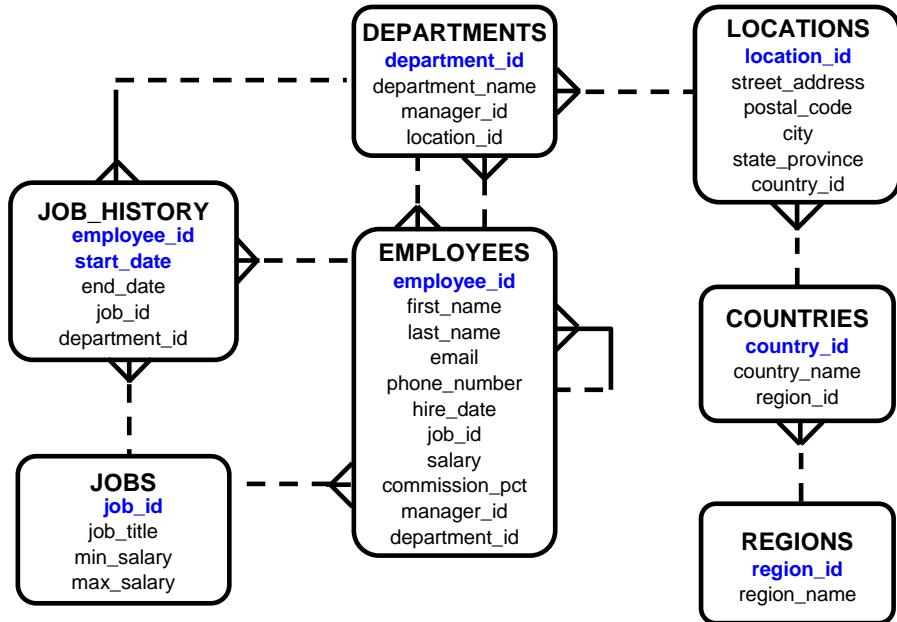
단원 내용

- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

HR(Human Resources) 스키마



ORACLE

Copyright © 2007, Oracle. All rights reserved.

HR (Human Resources) 스키마 설명

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

테이블 설명

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

이 과정에 사용되는 테이블

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
					50	RMATOS	650.121.2874	15-MAR-98
DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID					
10	Administration	200	1700					
20	Marketing	201	1800					
50	Shipping	124	1500					
60	IT	103	1400					
80	Sales	149	2500					
90	Executive	100	1700					
110	Accounting	205	1700					
190	Contracting	(null)	1700					

DEPARTMENTS

GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

JOB_GRADES

ORACLE

Copyright © 2007, Oracle. All rights reserved.

이 과정에 사용되는 테이블

다음은 이 과정에 사용되는 기본 테이블입니다.

- EMPLOYEES 테이블: 모든 사원에 대한 세부 정보를 제공합니다.
- DEPARTMENTS 테이블: 모든 부서에 대한 세부 정보를 제공합니다.
- JOB_GRADES 테이블: 다양한 등급의 급여에 대한 세부 정보를 제공합니다.

이 테이블과는 별도로 LOCATIONS 및 JOB_HISTORY 테이블과 같이 이전 슬라이드에 나열된 다른 테이블도 사용합니다.

주: 모든 테이블에 대한 구조 및 데이터는 부록 B를 참조하십시오.

단원 내용

- 과정 목표, 과정 내용 및 이 과정에 사용되는 부록
- **Oracle Database 11g** 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- **SQL** 및 개발 환경 소개
- **Oracle SQL Developer** 개요
- 이 과정에 사용되는 **HR** 스키마 및 테이블
- **Oracle Database 11g** 설명서 및 추가 자료

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g 설명서

- ***Oracle Database New Features Guide 11g, Release 1 (11.1)***
- ***Oracle Database Reference 11g, Release 1 (11.1)***
- ***Oracle Database SQL Language Reference 11g, Release 1 (11.1)***
- ***Oracle Database Concepts 11g, Release 1 (11.1)***
- ***Oracle Database SQL Developer User's Guide, Release 1.2***



Copyright © 2007, Oracle. All rights reserved.

Oracle Database 11g 설명서

Oracle Database 11g 설명서 라이브러리에 액세스하려면 <http://www.oracle.com/pls/db111/homepage>를 방문하십시오.

추가 자료

Oracle Database 11g에 대한 자세한 내용은 다음을 참조하십시오.

- **Oracle Database 11g: New Features eStudies**
- **Oracle by Example series (OBE): Oracle Database 11g**
 - http://www.oracle.com/technology/oobe/11gr1_db/index.htm



Copyright © 2007, Oracle. All rights reserved.

요약

- Oracle Database 11g는 다음과 같은 이점 및 기능을 확장합니다.
 - Infrastructure 그리드의 이점
 - 기준 정보 관리 기능
 - PL/SQL, Java/JDBC, .NET, XML 등의 주요 응용 프로그램 개발 환경 사용 기능
- 데이터베이스는 ORDBMS에 기반합니다.
- 관계형 데이터베이스는 관계로 구성되고 관계형 작업을 통해 관리되며 데이터 무결성 제약 조건으로 제어됩니다.
- Oracle 서버를 사용하면 SQL을 통해 정보를 저장하고 관리할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

관계형 데이터베이스 관리 시스템은 객체 또는 관계로 구성됩니다. 이를 시스템은 작업으로 관리하며 무결성 제약 조건으로 제어합니다.

Oracle Corporation은 유저의 RDBMS 요구 사항을 충족하는 제품과 서비스를 제공합니다. 주요 제품은 다음과 같습니다.

- SQL을 사용하여 정보를 저장하고 관리하는 Oracle Database 11g
- 통합 및 재사용 가능한 모듈식 업무 서비스를 개발, 배포 및 관리할 수 있는 Oracle Fusion Middleware
- 그리드 환경에서 전체 시스템의 작업을 관리하고 자동화하는 데 사용하는 Oracle Enterprise Manager Grid Control 10g

SQL

Oracle 서버는 ANSI 표준의 SQL을 지원하며 확장 기능을 포함합니다. SQL은 데이터를 액세스, 조작 및 제어하기 위해 서버와 통신하는 데 사용되는 언어입니다.

연습 I: 개요

이 연습에서는 다음 내용을 다룹니다.

- **Oracle SQL Developer** 데모 실행
- **Oracle SQL Developer** 시작, 새 데이터베이스 연결 생성 및 **HR** 테이블 탐색

ORACLE

Copyright © 2007, Oracle. All rights reserved.

연습 I: 개요

이 연습에서는 다음을 수행합니다.

- Oracle SQL Developer 데모를 실행합니다.
- Oracle SQL Developer를 사용하여 할당된 ORA 계정으로 데이터 객체를 검사합니다.
ORA 계정에는 HR 스키마 테이블이 포함됩니다.

다음 lab 파일 위치를 기록해 두십시오.

D:\labs\SQL1\labs

lab 파일을 저장하라는 요청을 받으면 이 위치에 저장하십시오.

각 연습에서 "시간 여유가 있을 경우" 또는 "심화 연습에 도전하려면"으로 시작하는 연습 과정이 제공될 수 있습니다. 이러한 연습은 할당된 시간 내에 다른 연습을 모두 완료한 다음 자신의 실력을 좀더 테스트해 보고자 하는 경우에만 수행하십시오.

연습은 천천히 정확하게 수행하십시오. 명령 파일을 저장하거나 실행해 볼 수 있습니다.
의문 사항이 있으면 언제든지 강사에게 문의하십시오.

주: 모든 연습 문제는 Oracle SQL Developer를 개발 환경으로 사용합니다. Oracle SQL Developer를 사용하는 것이 좋지만 이 과정에서 사용 가능한 SQL*Plus를 사용할 수도 있습니다.

연습 |

이 과정에 나오는 여러 연습 중 첫번째입니다. 필요한 경우 부록 A에서 해답을 찾아볼 수 있습니다. 각 연습은 해당 단원에 나오는 대부분의 주제를 다루도록 구성되었습니다.

Oracle SQL Developer 데모 실행: 데이터베이스 연결 생성

1. 다음 링크에서 "데이터베이스 연결 생성" 데모에 액세스합니다.

http://st-curriculum.oracle.com/tutorial/SQLDeveloper/html/module2/mod02_cp_newdbconn.htm

Oracle SQL Developer 시작

2. sqldeveloper 바탕 화면 아이콘을 사용하여 Oracle SQL Developer를 시작합니다.

주: SQL Developer를 처음 실행하는 경우 java.exe 파일의 경로를 제공해야 합니다.

이 작업은 강의실 설정의 일부로 이미 실행되었습니다. 프롬프트가 나타나는 경우 다음 경로를 입력합니다.

D:\app\Administrator\product\11.1.0\client_1\jdevstudio\jdk\bin

새 Oracle SQL Developer 데이터베이스 연결 생성

3. 새 데이터베이스 연결을 생성하려면 Connections Navigator에서 Connections를 마우스 오른쪽 버튼으로 누릅니다. 메뉴에서 New Connection을 선택합니다. New>Select Database Connection 대화상자가 나타납니다.
4. 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.
 - a. Connection Name: myconnection.
 - b. Username: oraxx. 여기에서 xx는 유저 PC 번호(강사에게 문의하여 계정 범위 ora1- ora20 내의 한 계정을 할당받도록 합니다.)
 - c. Password: oraxx
 - d. Hostname: 데이터베이스 서버가 실행되고 있는 컴퓨터의 호스트 이름을 입력합니다.
 - e. Port: 1521
 - f. SID: ORCL
 - g. Save Password 체크 박스를 선택합니다.

연습 1(계속)

Oracle SQL Developer 데이터베이스 연결을 사용하여 테스트 및 연결

5. 새 연결을 테스트합니다.
6. 상태가 Success이면 새로운 이 연결을 사용하여 데이터베이스에 연결합니다.

Connections Navigator에서 테이블 탐색

7. Connections Navigator의 Tables 노드에서 사용할 수 있는 객체를 확인합니다. 다음 테이블이 있는지 확인합니다.

COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS
LOCATIONS
REGIONS

8. EMPLOYEES 테이블의 구조를 테이블의 데이터를 확인합니다 살펴봅니다.
9. DEPARTMENTS.

SQL Worksheet 열기

10. 새 SQL Worksheet를 엽니다. 해당 SQL Worksheet에 대해 사용할 수 있는 단축 아이콘을 조사합니다.

1

SQL SELECT 문을 사용하여 데이터 검색

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **SQL SELECT** 문의 기능 나열
- 기본 **SELECT** 문 실행



Copyright © 2007, Oracle. All rights reserved.

목표

데이터베이스에서 데이터를 추출하려면 SQL SELECT 문을 사용해야 합니다. SELECT 문을 사용할 때는 너무 많은 열이 표시되지 않도록 제한해야 하는 경우도 있습니다. 이 단원에서는 이러한 작업을 수행하는 데 필요한 모든 SQL 문에 대해 설명합니다. 또한 두 번 이상 사용할 수 있는 SELECT 문을 작성할 수도 있습니다.

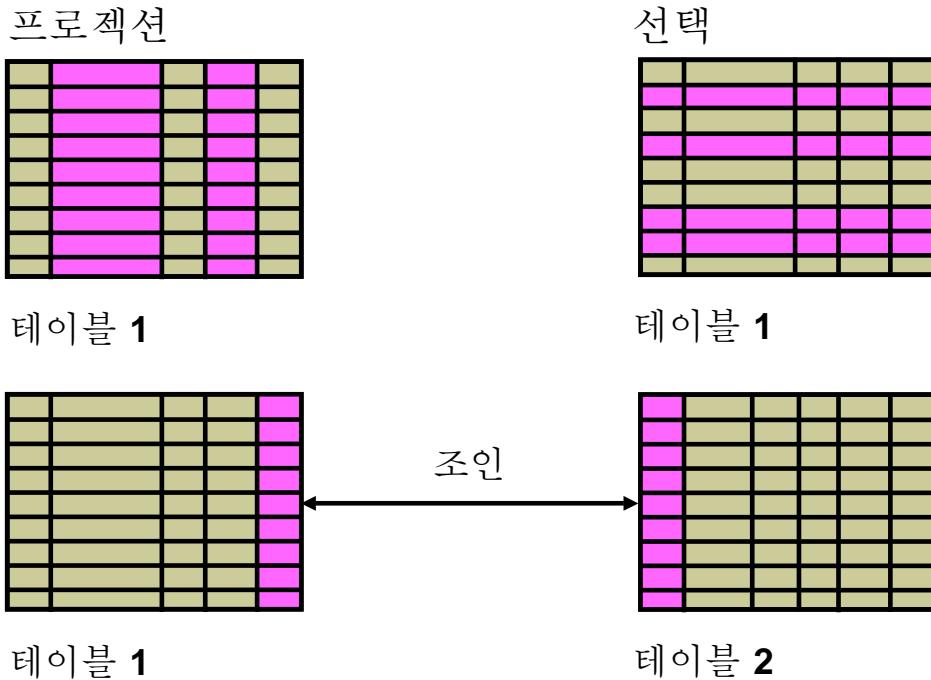
단원 내용

- 기본 **SELECT** 문
- **SELECT** 문의 산술식 및 **NULL** 값
- 열 **alias**
- 연결 연산자, 리터럴 문자열, 대체 인용 연산자 및 **DISTINCT** 키워드 사용
- **DESCRIBE** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SQL SELECT 문의 기능



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL SELECT 문의 기능

SELECT 문은 데이터베이스에서 정보를 검색합니다. SELECT 문으로 다음 기능을 사용할 수 있습니다.

- **프로젝션:** query에 의해 반환되는 테이블의 열을 선택합니다. 필요한 수만큼 열을 선택할 수 있습니다.
- **선택:** query에 의해 반환되는 테이블의 행을 선택합니다. 다양한 조건을 사용하여 검색되는 행을 제한할 수 있습니다.
- **조인:** 두 테이블 사이에 링크를 지정하여 서로 다른 테이블에 저장된 데이터를 함께 가져옵니다. SQL 조인은 "여러 테이블의 데이터 표시" 단원에서 자세히 다룹니다.

기본 SELECT 문

```
SELECT * | { [DISTINCT] column / expression [alias], ... }
FROM table;
```

- **SELECT**는 표시할 열을 식별합니다.
- **FROM**은 이러한 열을 포함한 테이블을 식별합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

기본 SELECT 문

가장 간단한 형식인 경우 SELECT 문은 다음을 포함해야 합니다.

- 표시할 열을 지정하는 SELECT 절
- SELECT 절에 나열된 열을 포함한 테이블을 식별하는 FROM 절

구문 설명:

SELECT	둘 이상의 열로 이루어진 리스트입니다.
*	모든 열을 선택합니다.
DISTINCT	중복을 방지합니다.
column / expression	명명된 열 또는 표현식을 선택합니다.
alias	선택된 열에 다른 머리글을 지정합니다.
FROM table	열을 포함하는 테이블을 지정합니다.

주: 이 과정 전체에 걸쳐 키워드, 절, 명령문이라는 단어는 다음과 같이 사용됩니다.

- 키워드는 개별 SQL 요소를 나타냅니다.
예를 들어, SELECT 및 FROM은 키워드입니다.
- 절은 SQL 문의 일부입니다.
예를 들어, SELECT employee_id, last_name 등은 절입니다.
- 명령문은 둘 이상의 절이 결합된 것입니다.
예를 들어, SELECT * FROM employees는 SQL 문입니다.

모든 열 선택

```
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

ORACLE

Copyright © 2007, Oracle. All rights reserved.

모든 열 선택

SELECT 키워드 다음에 별표(*)를 사용하여 테이블의 모든 데이터 열을 표시할 수 있습니다. 슬라이드의 예제에서 DEPARTMENTS 테이블은 DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID라는 네 개의 열을 포함하고 각 부서마다 여덟 개의 행을 포함합니다.

SELECT 키워드 뒤에 모든 열을 나열하는 방식으로 테이블의 모든 열을 표시할 수도 있습니다. 예를 들어, 다음 SQL 문(슬라이드의 예제와 유사)은 DEPARTMENTS 테이블의 모든 열과 모든 행을 표시합니다.

```
SELECT department_id, department_name, manager_id, location_id
FROM departments;
```

주: SQL Developer에서 SQL 문을 SQL Worksheet에 입력하고 "Execute Statement" 아이콘을 누르거나 [F9]를 눌러 해당 명령문을 실행할 수 있습니다. 슬라이드에 나오는 것과 같은 출력이 Results 탭 페이지에 표시됩니다.

특정 열 선택

```
SELECT department_id, location_id
FROM departments;
```

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

ORACLE

Copyright © 2007, Oracle. All rights reserved.

특정 열 선택

SELECT 문에서 열 이름을 쉼표로 구분하여 지정하는 방식으로 테이블의 특정 열을 표시할 수 있습니다. 슬라이드의 예제는 DEPARTMENTS 테이블의 모든 부서 번호와 위치 번호를 표시합니다.

SELECT 절에서 원하는 열을 출력에 나타내려는 순서대로 지정합니다. 예를 들어, 왼쪽에서 오른쪽 방향으로 위치 다음에 부서 번호를 표시하려면 다음 명령문을 사용합니다.

```
SELECT location_id, department_id
FROM departments;
```

	LOCATION_ID	DEPARTMENT_ID
1	1700	10
2	1800	20
3	1500	50
4	1400	60

...

SQL 문 작성

- **SQL** 문은 대소문자를 구분하지 않습니다.
- **SQL** 문은 한 줄 또는 여러 줄에 입력할 수 있습니다.
- 키워드는 약어로 표기하거나 여러 줄에 걸쳐 입력할 수 없습니다.
- 절은 대개 별도의 줄에 입력합니다.
- 가독성을 높이기 위해 들여쓰기를 사용합니다.
- **SQL Developer**에서 **SQL** 문은 선택적으로 세미콜론(;)으로 끝날 수 있습니다. 세미콜론은 여러 **SQL** 문을 실행하는 경우에 필요합니다.
- **SQL*plus**에서는 각 **SQL** 문이 반드시 세미콜론(;)으로 끝나야 합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL 문 작성

다음에 제시된 간단한 규칙과 지침에 따라 읽기 쉽고 편집하기 쉬운 유효한 명령문을 작성할 수 있습니다.

- SQL 문은 따로 지정하지 않는 한 대소문자를 구분하지 않습니다.
- SQL 문은 한 줄 또는 여러 줄에 입력할 수 있습니다.
- 키워드는 여러 줄에 걸쳐 입력하거나 약어로 표기할 수 없습니다.
- 절은 일반적으로 읽기 쉽고 편집하기 쉽도록 별도의 행에 입력합니다.
- 코드의 가독성을 높이기 위해 들여쓰기를 사용합니다.
- 키워드는 일반적으로 대문자로 입력하지만 테이블 이름 및 열 이름 등의 다른 단어는 모두 소문자로 입력합니다.

SQL 문 실행

SQL Developer의 SQL Worksheet에서 Run Script 아이콘을 누르거나 [F5]를 눌러 하나 이상의 명령을 실행할 수 있습니다. 또한 SQL Worksheet에서 Execute Statement 아이콘을 누르거나 [F9]를 눌러 SQL 문을 실행할 수도 있습니다. Execute Statement 아이콘을 누르면 Enter SQL Statement 상자에서 현재 마우스 포인터가 있는 위치의 명령문이 실행되지만 Run Script 아이콘을 누르면 Enter SQL Statement 상자의 모든 명령문이 실행됩니다. Execute Statement 아이콘을 누르면 query 출력 결과가 Results 탭 페이지에 표시되지만 Run Script 아이콘을 누르면 SQL*Plus에서처럼 출력 결과가 Script Output 탭 페이지에 표시됩니다.

SQL*Plus에서는 SQL 문을 세미콜론으로 끝낸 다음 [Enter]를 눌러 명령을 실행합니다.

열 머리글 기본값

- **SQL Developer:**
 - 기본 머리글 정렬: 왼쪽 정렬
 - 기본 머리글 표시: 대문자
- **SQL*Plus:**
 - 문자와 날짜 열 머리글은 왼쪽 정렬됩니다.
 - 숫자 열 머리글은 오른쪽 정렬됩니다.
 - 기본 머리글 표시: 대문자

ORACLE

Copyright © 2007, Oracle. All rights reserved.

열 머리글 기본값

SQL Developer에서 열 머리글은 대문자로 표시되고 왼쪽 정렬됩니다.

```
SELECT last_name, hire_date, salary
  FROM employees;
```

LAST_NAME	HIRE_DATE	SALARY
King	17-JUN-87	24000
Kochhar	21-SEP-89	17000
De Haan	13-JAN-93	17000
Hunold	03-JAN-90	9000
Ernst	21-MAY-91	6000
Lorentz	07-FEB-99	4200
Mourgos	16-NOV-99	5800
Rajs	17-OCT-95	3500

...

alias를 사용하여 열 머리글 표시를 재정의할 수 있습니다. 열 alias는 이 단원 뒷부분에서 다룹니다.

단원 내용

- 기본 **SELECT** 문
- **SELECT** 문의 산술식 및 **NULL** 값
- 열 **alias**
- 연결 연산자, 리터럴 문자열, 대체 인용 연산자 및 **DISTINCT** 키워드 사용
- **DESCRIBE** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

산술식

산술 연산자를 사용하여 숫자 및 날짜 데이터로 표현식을 작성합니다.

연산자	설명
+	더하기
-	빼기
*	곱하기
/	나누기

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

산술식

데이터 표시 방식을 수정하거나 계산을 수행하거나 가정(what-if) 시나리오를 조사해야 하는 경우가 있습니다. 산술식을 사용하여 이러한 모든 작업을 수행할 수 있습니다. 산술식은 열 이름, 상수 숫자 값 및 산술 연산자를 포함할 수 있습니다.

산술 연산자

슬라이드에 SQL에서 사용할 수 있는 산술 연산자가 나열되어 있습니다. FROM 절을 제외한 SQL 문의 모든 절에서 산술 연산자를 사용할 수 있습니다.

주: DATE 및 TIMESTAMP 데이터 유형은 더하기/빼기 연산자만 사용할 수 있습니다.

산술 연산자 사용

```
SELECT last_name, salary, salary + 300
FROM employees;
```

	LAST_NAME	SALARY	SALARY+300
1	King	24000	24300
2	Kochhar	17000	17300
3	De Haan	17000	17300
4	Hunold	9000	9300
5	Ernst	6000	6300
6	Lorentz	4200	4500
7	Mourgos	5800	6100
8	Rajs	3500	3800
9	Davies	3100	3400
10	Matos	2600	2900
...			

ORACLE

Copyright © 2007, Oracle. All rights reserved.

산술 연산자 사용

슬라이드의 예제는 더하기 연산자를 사용하여 모든 사원에 대해 \$300의 급여 인상액을 계산합니다. 또한 출력에 SALARY+300 열을 표시합니다.

계산 결과로 나타나는 SALARY+300 열은 EMPLOYEES 테이블의 새 열이 아니라 표시용일 뿐입니다. 기본적으로 새 열의 이름은 해당 열을 생성한 계산에서 제공되며 이 경우에는 salary+300입니다.

주: Oracle 서버는 산술 연산자 앞뒤의 공백을 무시합니다.

연산자 우선 순위

산술식에 둘 이상의 연산자가 포함된 경우 곱하기와 나누기가 먼저 평가됩니다. 표현식의 연산자 우선 순위가 동일한 경우 왼쪽에서 오른쪽 순서로 계산이 수행됩니다.

괄호를 사용하여 괄호로 묶인 표현식이 맨 먼저 계산되도록 할 수 있습니다.

우선 순위 규칙:

- 곱하기와 나누기는 더하기와 빼기보다 먼저 수행됩니다.
- 동일한 우선 순위를 갖는 연산자는 왼쪽에서 오른쪽으로 평가됩니다.
- 괄호는 기본 우선 순위를 재정의하거나 표현식을 명확히 하기 위해 사용됩니다.

연산자 우선 순위

```
SELECT last_name, salary, 12*salary+100  
FROM employees;
```

1

	LAST_NAME	SALARY	12*SALARY+100
1	King	24000	288100
2	Kochhar	17000	204100
3	De Haan	17000	204100

```
SELECT last_name, salary, 12*(salary+100)  
FROM employees;
```

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	King	24000	289200
2	Kochhar	17000	205200
3	De Haan	17000	205200

ORACLE

Copyright © 2007, Oracle. All rights reserved.

연산자 우선 순위(계속)

슬라이드의 첫번째 예제는 사원의 성, 급여 및 연간 총수입을 표시합니다. 연간 총수입은 월급에 12를 곱한 다음 1회 상여금 \$100를 더하여 계산합니다. 곱하기가 더하기보다 먼저 수행되는 것을 알 수 있습니다.

주: 표준 우선 순위를 적용하고 보다 명확하게 식을 작성하려면 괄호를 사용합니다. 예를 들어, 슬라이드의 표현식은 $(12*salary) + 100$ 으로 작성할 수 있으며 결과에는 아무런 변화가 없습니다.

괄호 사용

괄호를 사용하여 연산자 실행 순서를 원하는대로 지정함으로써 우선 순위 규칙을 재정의 할 수 있습니다.

슬라이드의 두번째 예제는 사원의 성, 급여 및 연간 총수입을 표시합니다. 연간 총 수입은 월급에 매달 상여금 \$100를 더한 다음 여기에 12를 곱하여 계산합니다. 괄호 때문에 더하기가 곱하기보다 우선 순위가 높습니다.

Null 값 정의

- Null은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값입니다.
- Null은 0이나 공백과는 다릅니다.

```
SELECT last_name, job_id, salary, commission_pct  
FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	King	AD_PRES	24000	(null)
2	Kochhar	AD_VP	17000	(null)
...				
12	Zlotkey	SA_MAN	10500	0.2
13	Abel	SA_REP	11000	0.3
14	Taylor	SA_REP	8600	0.2
...				
19	Higgins	AC_MGR	12000	(null)
20	Gietz	AC_ACCOUNT	8300	(null)

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Null 값 정의

행에 특정 열에 대한 데이터 값이 없는 경우 해당 값이 null이거나 null을 포함한다고 합니다.

Null은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값입니다. Null은 0이나 공백과는 다릅니다. 0은 숫자이며 공백은 문자입니다.

모든 데이터 유형의 열은 null을 포함할 수 있습니다. 그러나 일부 제약 조건(NOT NULL, PRIMARY KEY)이 지정된 열에서는 null을 사용할 수 없습니다.

EMPLOYEES 테이블의 COMMISSION_PCT 열에서 커미션은 판매 관리자나 판매 담당자만 받을 수 있습니다. 다른 사원은 커미션을 받을 수 있는 자격이 없습니다. null 값은 이러한 사실을 나타냅니다.

주: 기본적으로 SQL Developer에서는 (null) 리터럴을 사용하여 null 값을 식별합니다. 그러나 더 의미 있는 값으로 null 값을 설정할 수 있습니다. 이렇게 하려면 Tools 메뉴에서 Preferences를 선택합니다. Preferences 대화상자에서 Database 노드를 확장합니다. 오른쪽 창에서 Advanced Parameters를 누르고 "Display Null value As"에 대해 적절한 값을 입력합니다.

산술식의 Null 값

null 값을 포함하는 산술식은 **null**로 계산됩니다.

```
SELECT last_name, 12*salary*commission_pct
FROM employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
King	(null)
Kochhar	(null)
...	
Zlotkey	25200
Abel	39600
Taylor	20640
...	
Higgins	(null)
Gietz	(null)

Copyright © 2007, Oracle. All rights reserved.

산술식의 Null 값

산술식의 열 값이 null인 경우 결과는 null입니다. 예를 들어, 0으로 나누려고 하면 오류가 발생합니다. 하지만 숫자를 null로 나누면 결과는 null이 되거나 알 수 없는 상태가 됩니다.

슬라이드의 예제에서 King이라는 사원은 커미션을 받지 않습니다. 산술식의 COMMISSION_PCT 열이 null이기 때문에 결과는 null입니다.

자세한 사항은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "Basic Elements of Oracle SQL" 관련 섹션을 참조하십시오.

단원 내용

- 기본 **SELECT** 문
- **SELECT** 문의 산술식 및 **NULL** 값
- 열 **alias**
- 연결 연산자, 리터럴 문자열, 대체 인용 연산자 및 **DISTINCT** 키워드 사용
- **DESCRIBE** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

열 alias 정의

열 alias:

- 열 머리글의 이름을 바꿉니다.
- 계산에서 유용합니다.
- 열 이름 바로 뒤에 나옵니다. (열 이름과 **alias** 사이에 선택 사항인 **AS** 키워드가 올 수도 있습니다.)
- 공백이나 특수 문자를 포함하거나 대소문자를 구분하는 경우 큰따옴표가 필요합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

열 alias 정의

query 결과를 표시할 때 SQL Developer는 일반적으로 선택된 열의 이름을 열 머리글로 사용합니다. 이 머리글은 설명형이 아닐 수도 있으며 따라서 이해하기 어려울 수 있습니다. 열 alias를 사용하여 열 머리글을 변경할 수 있습니다.

공백을 구분자로 사용하여 SELECT 리스트의 열 뒤에 alias를 지정합니다. 기본적으로 alias 머리글은 대문자로 나타납니다. alias가 공백이나 특수 문자(예: # 또는 \$)를 포함하거나 대소문자를 구분하는 경우 alias를 큰 따옴표(" ")로 묶습니다.

열 alias 사용

```
SELECT last_name AS [name], commission_pct [comm]
FROM employees;
```

	NAME	COMM
1	King	(null)
2	Kochhar	(null)
3	De Haan	(null)

...

```
SELECT last_name ["Name"] , salary*12 ["Annual Salary"]
FROM employees;
```

	Name	Annual Salary
1	King	288000
2	Kochhar	204000
3	De Haan	204000

...

Copyright © 2007, Oracle. All rights reserved.

열 alias 사용

첫번째 예제는 모든 사원의 이름과 커미션 백분율을 표시합니다. 열 alias 이름 앞에 선택 사항인 AS 키워드가 사용되었습니다. query 결과는 AS 키워드 사용 여부에 관계없이 동일합니다. 또한 SQL 문에는 열 alias인 name 및 comm이 소문자이지만 query 결과에는 열 머리글이 대문자로 표시됩니다. 이전 슬라이드에서 언급했듯이 열 머리글은 기본적으로 대문자로 나타납니다.

두번째 예제는 모든 사원의 성과 연간 급여를 표시합니다. Annual Salary에 공백이 있으므로 큰 따옴표로 묶었습니다. 출력의 열 머리글은 열 alias와 정확하게 같습니다.

단원 내용

- 기본 **SELECT** 문
- **SELECT** 문의 산술식 및 **NULL** 값
- 열 **alias**
- 연결 연산자, 리터럴 문자열, 대체 인용 연산자 및 **DISTINCT** 키워드 사용
- **DESCRIBE** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

연결 연산자

연결 연산자:

- 열이나 문자열을 다른 열에 연결합니다.
- 두 개의 세로선(||)으로 나타냅니다.
- 결과 열로 문자 표현식을 작성합니다.

```
SELECT last_name || job_id AS "Employees"
FROM   employees;
```

	Employees
1	AbelSA_REP
2	DaviesST_CLERK
3	De HaanAD_VP
4	ErnstIT_PROG
5	FayMK_REP

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

연결 연산자

연결 연산자(||)를 사용하면 열을 다른 열, 산술식 또는 상수 값에 연결하여 문자 표현식을 작성할 수 있습니다. 연산자 양쪽의 열이 결합되어 단일 출력 열을 생성합니다.

예제에서는 LAST_NAME과 JOB_ID가 연결되고 Employees라는 alias가 주어집니다. 사원의 성과 직무 코드가 결합되어 단일 출력 열이 만들어집니다.

SELECT 절을 보다 쉽게 읽을 수 있도록 alias 이름 앞에 AS 키워드가 추가되었습니다.

연결 연산자와 null 값

문자열에 null 값을 결합할 경우 결과는 문자열입니다. LAST_NAME || NULL의 결과는 LAST_NAME입니다.

주: 날짜 표현식을 다른 표현식이나 열과 연결할 수도 있습니다.

리터럴 문자열

- 리터럴은 **SELECT** 문에 포함된 문자, 숫자 또는 날짜입니다.
- 날짜 및 문자 리터럴 값은 작은 따옴표로 묶어야 합니다.
- 각 문자열은 반환되는 각 행에 한 번 출력됩니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

리터럴 문자열

리터럴은 SELECT 리스트에 포함된 문자, 숫자 또는 날짜입니다. 열 이름 또는 열 alias는 리터럴이 아닙니다. 리터럴은 반환되는 각 행에 대해 출력됩니다. 자유 형식 텍스트의 리터럴 문자열은 query 결과에 포함될 수 있으며 SELECT 리스트의 열과 동일하게 취급됩니다. 날짜 및 문자 리터럴은 반드시 작은 따옴표(' ')로 묶어야 하지만 숫자 리터럴은 그럴 필요가 없습니다.

리터럴 문자열 사용

```
SELECT last_name || ' is a ' || job_id
AS "Employee Details"
FROM employees;
```

Employee Details	
1	Abel is a SA_REP
2	Davies is a ST_CLERK
3	De Haan is a AD_VP
4	Ernst is a IT_PROG
5	Fay is a MK_REP
...	
18	Vargas is a ST_CLERK
19	Whalen is a AD_ASST
20	Zlotkey is a SA_MAN

ORACLE

Copyright © 2007, Oracle. All rights reserved.

리터럴 문자열 사용

슬라이드의 예제는 모든 사원의 성과 직무 코드를 표시합니다. 열 머리글은 Employee Details입니다. SELECT 문에서 작은 따옴표 사이에 공백이 있는 점에 유의하십시오. 이 공백은 출력의 가독성을 높여줍니다.

다음 예제에서는 반환되는 행의 의미를 더욱 명확히 하기 위해 각 사원의 성과 급여가 리터럴로 연결됩니다.

```
SELECT last_name || ': ' || salary Monthly
FROM employees;
```

MONTHLY	
1	King: 1 Month salary = 24000
2	Kochhar: 1 Month salary = 17000
3	De Haan: 1 Month salary = 17000
4	Hunold: 1 Month salary = 9000
5	Ernst: 1 Month salary = 6000
6	Lorentz: 1 Month salary = 4200
7	Mourgos: 1 Month salary = 5800
8	Rajs: 1 Month salary = 3500
...	

대체 인용(q) 연산자

- 자신의 따옴표 구분자를 지정합니다.
- 구분자를 임의로 선택합니다.
- 가독성 및 사용성이 증가합니다.

```
SELECT department_name || ' Department' ||
       q'[ 's Manager Id: ]'
       || manager_id
      AS "Department and Manager"
FROM departments;
```

Department and Manager
1 Administration Department's Manager Id:200
2 Marketing Department's Manager Id:201
3 Shipping Department's Manager Id:124
4 IT Department's Manager Id:103
5 Sales Department's Manager Id:149
6 Executive Department's Manager Id:100
7 Accounting Department's Manager Id:205
8 Contracting Department's Manager Id:

ORACLE

Copyright © 2007, Oracle. All rights reserved.

대체 인용(q) 연산자

많은 SQL 문에서 표현식이나 조건에 문자 리터럴을 사용합니다. 리터럴 자체에 작은 따옴표가 포함된 경우 인용(q) 연산자를 사용하여 자신의 따옴표 구분자를 선택할 수 있습니다.

단일 바이트든 멀티 바이트든 [], {}, (), <> 문자 쌍 중에서 사용하기 편한 구분자를 선택할 수 있습니다.

위의 예제에서는 문자열에 작은 따옴표가 포함되어 있으며, 이것은 대개 문자열 구분자로 해석됩니다. 그러나 q 연산자를 사용할 경우 대괄호 []가 따옴표 구분자로 사용됩니다. 대괄호 구분자 사이의 문자열은 리터럴 문자열로 해석됩니다.

중복 행

기본적으로 **query** 결과에는 중복 행을 포함한 모든 행이 표시됩니다.

```
SELECT department_id  
FROM employees;
```

1

DEPARTMENT_ID
1
2
3
4
5

...

```
SELECT DISTINCT department_id  
FROM employees;
```

2

DEPARTMENT_ID
1
2
3
4

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

중복 행

별도로 지정하지 않는 한 SQL은 중복 행을 제거하지 않고 query 결과를 표시합니다. 슬라이드의 첫번째 예제는 EMPLOYEES 테이블의 모든 부서 번호를 표시합니다. 부서 번호가 반복되는 것을 알 수 있습니다.

결과에서 중복 행을 제거하려면 SELECT 절에서 SELECT 키워드 바로 뒤에 DISTINCT 키워드를 포함시킵니다. 슬라이드의 두번째 예제에서 EMPLOYEES 테이블에는 실제로 20개의 행이 있지만 테이블의 고유한 부서 번호는 일곱 개뿐입니다.

DISTINCT 수식자 뒤에 여러 열을 지정할 수 있습니다. DISTINCT 수식자는 선택된 모든 열에 영향을 주며 결과에는 모든 고유한 열 조합이 나타납니다.

```
SELECT DISTINCT department_id, job_id  
FROM employees;
```

DEPARTMENT_ID	JOB_ID
1	110 AC_ACCOUNT
2	90 AD_VP
3	50 ST_CLERK
4	80 SA_REP
5	50 ST_MAN

...

단원 내용

- 기본 **SELECT** 문
- **SELECT** 문의 산술식 및 **NULL** 값
- 열 **alias**
- 연결 연산자, 리터럴 문자열, 대체 인용 연산자 및 **DISTINCT** 키워드 사용
- **DESCRIBE** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

테이블 구조 표시

- **DESCRIBE** 명령을 사용하여 테이블의 구조를 표시합니다.
- 또는 **Connections** 트리에서 테이블을 선택하고 **Columns** 탭을 사용하여 테이블 구조를 확인합니다.

DESC [RIBE] tablename

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	Comments
DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	1	Primary key column of departments table
DEPARTMENT_N...	VARCHAR2(30 BYTE)	No	(null)	2	(null)	A not null column that shows name of a department.
MANAGER_ID	NUMBER(6,0)	Yes	(null)	3	(null)	Manager_id of a department. Foreign key
LOCATION_ID	NUMBER(4,0)	Yes	(null)	4	(null)	Location id where a department is located.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 구조 표시

SQL Developer에서 DESCRIBE 명령을 사용하여 테이블 구조를 표시할 수 있습니다. 이 명령은 열 이름 및 데이터 유형을 표시하고 열에 반드시 데이터가 포함되어야 하는지 여부(즉, 열에 NOT NULL 제약 조건이 있는지 여부)를 보여줍니다.

구문에서 *table name*은 유저가 액세스할 수 있는 기준의 테이블, 뷰 또는 동의어의 이름입니다.

SQL Developer GUI 인터페이스를 사용하면 Connections 트리에서 테이블을 선택하고 Columns 탭을 사용하여 테이블 구조를 확인할 수 있습니다.

주: DESCRIBE 명령은 SQL*Plus 및 SQL Developer에서 모두 지원됩니다.

DESCRIBE 명령 사용

```
DESCRIBE employees
```

DESCRIBE employees		
Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)
11 rows selected		

Copyright © 2007, Oracle. All rights reserved.

DESCRIBE 명령 사용

슬라이드의 예제는 DESCRIBE 명령을 사용하여 EMPLOYEES 테이블의 구조에 대한 정보를 표시합니다.

결과 표시에서 *Null*은 이 열의 값이 알 수 없는 값이 될 수 있음을 나타냅니다. NOT NULL은 열에 데이터가 포함되어야 함을 나타냅니다. *Type*은 열의 데이터 유형을 표시합니다.

데이터 유형은 다음 표에 설명되어 있습니다.

데이터 유형	설명
NUMBER (<i>p, s</i>)	최대 자릿수가 <i>p</i> 이고 소수점 이하 자릿수가 <i>s</i> 인 숫자 값
VARCHAR2 (<i>s</i>)	최대 크기가 <i>s</i> 인 가변 길이 문자 값
DATE	기원전 4712년 1월 1일에서 기원후 9999년 12월 31일 사이의 날짜 및 시간 값
CHAR (<i>s</i>)	크기가 <i>s</i> 인 고정 길이 문자 값

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- **SELECT** 문 작성:

- 테이블에서 모든 행과 열 반환
- 테이블에서 지정된 열 반환
- 열 **alias**를 사용하여 열 머리글을 알기 쉽게 표시

```
SELECT * | { [DISTINCT] column / expression [alias] , ... }  
FROM table;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SELECT 문

이 단원에서는 SELECT 문을 사용하여 데이터베이스 테이블에서 데이터를 검색하는 방법에 대해 설명했습니다.

```
SELECT * | { [DISTINCT] column [alias] , ... }  
FROM table;
```

구문 설명:

SELECT	둘 이상의 열로 이루어진 리스트입니다.
*	모든 열을 선택합니다.
DISTINCT	중복을 방지합니다.
column / expression	명명된 열 또는 표현식을 선택합니다.
alias	선택된 열에 다른 머리글을 지정합니다.
FROM table	열을 포함하는 테이블을 지정합니다.

연습 1: 개요

이 연습에서는 다음 내용을 다룹니다.

- 다른 테이블에서 모든 데이터 선택
- 테이블의 구조 설명
- 산술식을 수행하고 열 이름 지정



Copyright © 2007, Oracle. All rights reserved.

연습 1: 개요

이 연습에서는 간단한 SELECT query를 작성합니다. 이러한 query에는 이 단원에서 학습한 대부분의 SELECT 절과 연산이 포함됩니다.

연습 1

1부

지식을 테스트해 보십시오.

1. 다음 SELECT 문이 성공적으로 실행됩니다.

```
SELECT last_name, job_id, salary AS Sal
FROM employees;
```

참/거짓

2. 다음 SELECT 문이 성공적으로 실행됩니다.

```
SELECT *
FROM job_grades;
```

참/거짓

3. 다음 명령문에 네 개의 코딩 오류가 있습니다. 식별할 수 있습니까?

```
SELECT employee_id, last_name
sal * 12 ANNUAL SALARY
FROM employees;
```

2부

연습을 시작하기 전에 다음과 같은 점에 주의합니다.

- 모든 실습 파일을 D:\labs\SQL1\labs에 저장합니다.
- SQL Worksheet에 SQL 문을 입력합니다. SQL Developer에서 스크립트를 저장하려면 필요한 SQL Worksheet가 활성화되어 있는지 확인한 다음 File 메뉴에서 Save As를 선택하거나 SQL Worksheet에서 마우스 오른쪽 버튼을 누르고 Save File을 선택하여 SQL 문을 lab_<lessonno>_<stepno>.sql 스크립트로 저장합니다. 기존 스크립트를 수정하는 경우 Save As를 사용하여 다른 파일 이름으로 스크립트를 저장해야 합니다.
- query를 실행하려면 SQL Worksheet에서 Execute Statement 아이콘을 누릅니다. 또는 [F9]를 누를 수도 있습니다. DML 및 DDL 문의 경우에는 Run Script 아이콘을 사용하거나 [F5]를 누릅니다.
- query를 실행한 후 동일한 워크시트에 다음 query를 입력하지 않도록 합니다. 새 워크시트를 엽니다.

여러분은 Acme Corporation의 SQL 프로그래머로 채용되었습니다. 첫 작업은 Human Resources 테이블의 데이터를 기반으로 몇 가지 보고서를 작성하는 것입니다.

4. 첫 작업은 DEPARTMENTS 테이블의 구조와 해당 내용을 파악하는 것입니다.

DESCRIBE departments		
Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

4 rows selected

연습 1(계속)

	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

5. EMPLOYEES 테이블의 구조를 파악해야 합니다.

```
DESCRIBE employees
Name           Null    Type
-----          -----
EMPLOYEE_ID      NOT NULL NUMBER(6)
FIRST_NAME        VARCHAR2(20)
LAST_NAME         NOT NULL VARCHAR2(25)
EMAIL            NOT NULL VARCHAR2(25)
PHONE_NUMBER      VARCHAR2(20)
HIRE_DATE         NOT NULL DATE
JOB_ID            NOT NULL VARCHAR2(10)
SALARY             NUMBER(8,2)
COMMISSION_PCT     NUMBER(2,2)
MANAGER_ID         NUMBER(6)
DEPARTMENT_ID       NUMBER(4)

11 rows selected
```

HR 부서에서 사원 번호부터 시작해서 각 사원에 대한 성, 직무 코드, 채용 날짜 및 사원 번호를 표시하는 query를 요구합니다. HIRE_DATE 열에 대한 alias로 STARTDATE를 입력하십시오. 작성한 SQL 문을 HR 부서에 전달할 수 있도록 lab_01_05.sql이라는 파일에 저장합니다.

연습 1(계속)

6. lab_01_05.sql 파일의 query가 제대로 실행되는지 테스트합니다.

주: query를 실행한 후 동일한 워크시트에 다음 query를 입력하지 않도록 합니다.
새 워크시트를 엽니다.

EMPLOYEE_ID	LAST_NAME	JOB_ID	STARTDATE
1	100 King	AD_PRES	17-JUN-87
2	101 Kochhar	AD_VP	21-SEP-89
3	102 De Haan	AD_VP	13-JAN-93
4	103 Hunold	IT_PROG	03-JAN-90
5	104 Ernst	IT_PROG	21-MAY-91
6	107 Lorentz	IT_PROG	07-FEB-99
7	124 Mourgos	ST_MAN	16-NOV-99
8	141 Rajs	ST_CLERK	17-OCT-95
9	142 Davies	ST_CLERK	29-JAN-97
10	143 Matos	ST_CLERK	15-MAR-98
...			
19	205 Higgins	AC_MGR	07-JUN-94
20	206 Gietz	AC_ACCOUNT	07-JUN-94

7. HR 부서에서 EMPLOYEES 테이블의 모든 고유 직무 코드를 표시하는 query를 요구합니다.

JOB_ID
1 AC_ACCOUNT
2 AC_MGR
3 AD_ASST
4 AD_PRES
5 AD_VP
6 IT_PROG
7 MK_MAN
8 MK_REP
9 SA_MAN
10 SA_REP
11 ST_CLERK
12 ST_MAN

연습 1(계속)

3부

시간 여유가 있을 경우 다음 연습을 완료하십시오.

8. HR 부서에서 보고에 적합하도록 열 머리글의 사원 정보를 쉽게 표시해 달라고 합니다.

`lab_01_05.sql`의 명령문을 새 SQL Worksheet로 복사합니다. 열 머리글을 각각 Emp #, Employee, Job 및 Hire Date로 지정합니다. 그런 다음 query를 다시 실행합니다.

	Emp #	Employee	Job	Hire Date
1	100	King	AD_PRES	17-JUN-87
2	101	Kochhar	AD_VP	21-SEP-89
3	102	De Haan	AD_VP	13-JAN-93
4	103	Hunold	IT_PROG	03-JAN-90
5	104	Ernst	IT_PROG	21-MAY-91
6	107	Lorentz	IT_PROG	07-FEB-99
7	124	Mourgos	ST_MAN	16-NOV-99
8	141	Rajs	ST_CLERK	17-OCT-95
9	142	Davies	ST_CLERK	29-JAN-97
10	143	Matos	ST_CLERK	15-MAR-98
...				
19	205	Higgins	AC_MGR	07-JUN-94
20	206	Gietz	AC_ACCOUNT	07-JUN-94

9. HR 부서에서 모든 사원과 그들의 직무 ID에 대한 보고서를 요청했습니다. 성과 직무 ID를 이어서 표시하고(쉼표와 공백으로 구분) 열 이름을 Employee and Title로 지정합니다.

	Employee and Title
1	Abel, SA_REP
2	Davies, ST_CLERK
3	De Haan, AD_VP
4	Ernst, IT_PROG
5	Fay, MK_REP
6	Gietz, AC_ACCOUNT
7	Grant, SA_REP
8	Hartstein, MK_MAN
9	Higgins, AC_MGR
10	Hunold, IT_PROG
...	

19	Whalen, AD_ASST
20	Zlotkey, SA_MAN

연습 1(계속)

심화 연습에 도전하려면 다음 연습을 완료하십시오.

10. EMPLOYEES 테이블의 데이터에 익숙해지도록 해당 테이블의 모든 데이터를 표시하는 query를 작성합니다. 각 열 출력은 쉼표로 구분합니다. 열 제목의 이름을 THE_OUTPUT으로 지정합니다.

Results:								
	THE_OUTPUT							
1	100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90							
2	101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90							
3	102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90							
4	103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60							
5	104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60							
6	107,Diana,Lorentz,DLORENTZ,590.423.5567,IT_PROG,103,07-FEB-99,4200,,60							
7	124,Kevin,Mourgos,KMOURGOS,650.123.5234,ST_MAN,100,16-NOV-99,5800,,50							
8	141,Trenna,Rajs,TRAJS,650.121.8009,ST_CLERK,124,17-OCT-95,3500,,50							
9	142,Curtis,Davies,CDAVIES,650.121.2994,ST_CLERK,124,29-JAN-97,3100,,50							
10	143,Randall,Matos,RMATOS,650.121.2874,ST_CLERK,124,15-MAR-98,2600,,50							
...								
19	205,Shelley,Higgins,SHIGGINS,515.123.8080,AC_MGR,101,07-JUN-94,12000,,110							
20	206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110							

데이터 제한 및 정렬

Copyright © 2007, Oracle. All rights reserved.

ORACLE®

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **query**로 검색되는 행 제한
- **query**로 검색되는 행 정렬
- 앤퍼샌드 치환을 사용하여 런타임 시 출력 제한 및 정렬



Copyright © 2007, Oracle. All rights reserved.

목표

데이터베이스에서 데이터를 검색할 때 다음 작업을 수행해야 할 수도 있습니다.

- 표시되는 데이터의 행 제한
- 행이 표시되는 순서 지정

이 단원에서는 아래에 나열된 작업을 수행하는 데 사용되는 SQL 문에 대해 설명합니다.

단원 내용

- 다음과 같은 방법을 사용하여 행을 제한합니다.
 - **WHERE** 절
 - **=, <=, BETWEEN, IN, LIKE** 및 **NULL** 조건을 사용하는 비교 조건
 - **AND, OR** 및 **NOT** 연산자를 사용하는 논리 조건
- 표현식의 연산자 우선 순위 규칙
- **ORDER BY** 절을 사용하여 행 정렬
- 치환 변수
- **DEFINE** 및 **VERIFY** 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

선택을 사용하여 행 제한

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90
4	103	Hunold	IT_PROG	60
5	104	Ernst	IT_PROG	60
6	107	Lorentz	IT_PROG	60

...

"부서 90의
모든 사원 검색"

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

ORACLE

Copyright © 2007, Oracle. All rights reserved.

선택을 사용하여 행 제한

슬라이드의 예제에서 부서 90의 모든 사원을 표시한다고 가정해 보겠습니다. DEPARTMENT_ID 열의 값이 90인 행만 반환됩니다. 이러한 제한 방식이 SQL에서 WHERE 절의 기반이 됩니다.

선택되는 행 제한

- **WHERE** 절을 사용하여 반환되는 행을 제한합니다.

```
SELECT * | { [DISTINCT] column / expression [alias], ... }
  FROM table
[WHERE condition(s)];
```

- **WHERE** 절은 **FROM** 절 다음에 나옵니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

선택되는 행 제한

WHERE 절을 사용하여 query에서 반환되는 행을 제한할 수 있습니다. WHERE 절은 충족되어야 하는 조건을 포함하며 FROM 절 바로 뒤에 나옵니다. 조건이 참인 경우 해당 조건을 충족하는 행이 반환됩니다.

구문 설명:

WHERE 조건을 충족하는 행으로 query를 제한합니다.

condition 열 이름, 표현식,
상수 및 비교 연산자로 구성됩니다. 조건은
하나 이상의 표현식과 논리(부울) 연산자의 조합을 지정하고
TRUE, FALSE 또는 UNKNOWN 중 하나의 값을 반환합니다.

WHERE 절은 열의 값, 리터럴, 산술식 또는 함수를 비교할 수 있으며 다음 세 가지 요소로 구성됩니다.

- 열 이름
- 비교 조건
- 열 이름, 상수 또는 값 리스트

WHERE 절 사용

```
SELECT employee_id, last_name, job_id, department_id  
FROM   employees  
WHERE  department_id = 90;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100 King	AD_PRES	90	
2	101 Kochhar	AD_VP	90	
3	102 De Haan	AD_VP	90	

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

WHERE 절 사용

예제에서 SELECT 문은 부서 90에 소속된 모든 사원의 사원 ID, 성, 직무 ID 및 부서 번호를 검색합니다.

주: WHERE 절에서는 열 alias를 사용할 수 없습니다.

문자열 및 날짜

- 문자열 및 날짜 값은 작은 따옴표로 묶습니다.
- 문자 값은 대소문자를 구분하고 날짜 값은 형식을 구분합니다.
- 기본 날짜 표시 형식은 **DD-MON-RR**입니다.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Whalen' ;
```

```
SELECT last_name
FROM   employees
WHERE  hire_date = '17-FEB-96' ;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

문자열 및 날짜

WHERE 절에서 문자열과 날짜는 작은 따옴표('')로 묶어야 합니다. 그러나 숫자 상수는 작은 따옴표로 묶으면 안됩니다.

모든 문자 검색은 대소문자를 구분합니다. 다음 예제에서는 EMPLOYEES 테이블에 모든 성이 대소문자가 혼용된 형식으로 저장되기 때문에 행이 반환되지 않습니다.

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'WHALEN' ;
```

오라클 데이터베이스는 세기, 년, 월, 일, 시, 분, 초를 나타내는 내부 숫자 형식으로 날짜를 저장합니다. 기본 날짜 표시 형식은 DD-MON-RR입니다.

주: RR 형식에 대한 자세한 내용과 기본 날짜 형식을 변경하는 방법은 "단일 행 함수를 사용하여 출력 커스터마이즈" 단원을 참조하십시오. 또한 해당 단원에서는 UPPER 및 LOWER 등의 단일 행 함수를 사용하여 대소문자 구분을 재정의하는 방법에 대해서도 학습합니다.

비교 연산자

연산자	의미
=	같음
>	보다 큼
>=	보다 크거나 같음
<	보다 작음
<=	보다 작거나 같음
≠	같지 않음
BETWEEN ...AND...	두 값 사이(경계값 포함)
IN (set)	값 리스트 중 일치하는 값 검색
LIKE	일치하는 문자 패턴 검색
IS NULL	null 값인지 여부

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

비교 연산자

비교 연산자는 특정 표현식을 다른 값이나 표현식과 비교하는 조건에서 사용됩니다. WHERE 절에서 다음과 같은 형식으로 사용됩니다.

구문

... WHERE *expr operator value*

예제

```
... WHERE hire_date = '01-JAN-95'  
... WHERE salary >= 6000  
... WHERE last_name = 'Smith'
```

WHERE 절에서 alias는 사용할 수 없습니다.

주: != 및 ^= 기호도 같지 않음 조건을 나타냅니다.

비교 연산자 사용

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

	LAST_NAME	SALARY
1	Matos	2600
2	Vargas	2500

ORACLE

Copyright © 2007, Oracle. All rights reserved.

비교 연산자 사용

예제에서 SELECT 문은 EMPLOYEES 테이블에서 급여가 \$3,000보다 작거나 같은 사원의 성과 급여를 검색합니다. WHERE 절에 명시적 값이 제공되었습니다. 3000이라는 명시적 값이 EMPLOYEES 테이블의 SALARY 열에 있는 급여 값과 비교됩니다.

BETWEEN 연산자를 사용하는 범위 조건

BETWEEN 연산자를 사용하여 값의 범위를 기반으로 행을 표시합니다.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

↑ ↑
하한 상한

	LAST_NAME	SALARY
1	Rajs	3500
2	Davies	3100
3	Matos	2600
4	Vargas	2500

ORACLE

Copyright © 2007, Oracle. All rights reserved.

BETWEEN 연산자를 사용하는 범위 조건

BETWEEN 연산자를 사용하여 값의 범위를 기반으로 행을 표시할 수 있습니다. 지정한 범위에는 상한 및 하한이 포함됩니다.

슬라이드의 SELECT 문은 EMPLOYEES 테이블에서 급여가 \$2,500 ~ \$3,500 사이인 사원의 행을 반환합니다.

BETWEEN 연산자로 지정되는 값은 범위에 포함됩니다. 그러나 먼저 하한을 지정해야 합니다.

문자 값에도 BETWEEN 연산자를 사용할 수 있습니다.

```
SELECT last_name
FROM employees
WHERE last_name BETWEEN 'King' AND 'Smith';
```

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos
6	Rajs

IN 연산자를 사용하는 멤버 조건

IN 연산자를 사용하여 리스트의 값을 테스트합니다.

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201) ;
```

	EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
1	101 Kochhar	17000	100	
2	102 De Haan	17000	100	
3	124 Mourgos	5800	100	
4	149 Zlotkey	10500	100	
5	201 Hartstein	13000	100	
6	200 Whalen	4400	101	
7	205 Higgins	12000	101	
8	202 Fay	6000	201	

ORACLE

Copyright © 2007, Oracle. All rights reserved.

IN 연산자를 사용하는 멤버 조건

지정된 값 집합에서 값을 테스트하려면 IN 연산자를 사용합니다. IN 연산자를 사용하여 정의한 조건을 멤버 조건이라고도 합니다.

슬라이드 예제는 관리자의 사원 번호가 100, 101 또는 201인 모든 사원에 대해 사원 번호, 성, 급여 및 관리자의 사원 번호를 표시합니다.

IN 연산자는 어떠한 데이터 유형에도 사용할 수 있습니다. 다음 예제는 EMPLOYEES 테이블에서 WHERE 절의 이름 리스트에 자신의 성이 포함된 사원의 행을 반환합니다.

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE last_name IN ('Hartstein', 'Vargas');
```

리스트에 문자나 날짜가 사용되는 경우 작은 따옴표('')로 묶어야 합니다.

주: IN 연산자는 Oracle 서버 내부에서 a=value1 or a=value2 or a=value3과 같이 OR 조건의 집합으로 평가됩니다. 따라서 IN 연산자를 사용해도 성능상의 이점은 없으며 논리적으로 간결한 SELECT 문을 작성하려는 경우에만 사용됩니다.

LIKE 연산자를 사용하여 패턴 일치

- **LIKE** 연산자를 사용하여 유효한 검색 문자열 값의 대체 문자 검색을 수행합니다.
- 검색 조건에는 리터럴 문자나 숫자가 포함될 수 있습니다.
 - %는 0개 이상의 문자를 나타냅니다.
 - _은 한 문자를 나타냅니다.

```
SELECT first_name
  FROM employees
 WHERE first_name LIKE 'S%' ;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

LIKE 연산자를 사용하여 패턴 일치

검색할 값을 항상 정확히 알 수 있는 것은 아닙니다. LIKE 연산자를 사용하여 문자 패턴이 일치하는 행을 선택할 수 있습니다. 문자 패턴을 일치시키는 작업을 대체 문자 검색이라고 합니다. 두 가지 기호를 사용하여 검색 문자열을 구성할 수 있습니다.

기호	설명
%	0개 이상의 임의의 문자 시퀀스를 나타냅니다.
_	임의의 단일 문자를 나타냅니다.

슬라이드의 SELECT 문은 EMPLOYEES 테이블에서 이름이 "S"로 시작하는 사원의 이름을 반환합니다. "S"는 대문자임에 유의하십시오. 따라서 소문자 "s"로 시작하는 이름은 반환되지 않습니다.

LIKE 연산자는 일부 BETWEEN 비교의 단축형으로 사용될 수 있습니다. 다음 예제는 1995년 1월부터 1995년 12월 사이에 입사한 모든 사원의 성과 채용 날짜를 표시합니다.

```
SELECT last_name, hire_date
  FROM employees
 WHERE hire_date LIKE '%95' ;
```

대체 문자 결합

- 패턴 일치를 위해 두 대체 문자(%,_)를 리터럴 문자와 결합할 수 있습니다.

```
SELECT last_name
  FROM employees
 WHERE last_name LIKE '_o%' ;
```

LAST_NAME
Kochhar
Lorentz
Mourgos

- ESCAPE** 식별자를 사용하여 실제 % 및 _ 기호를 검색할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

대체 문자 결합

% 및 _ 기호는 리터럴 문자와 임의로 조합하여 사용할 수 있습니다. 슬라이드의 예제는 성의 두번째 문자가 "o"인 모든 사원의 이름을 표시합니다.

ESCAPE 식별자

실제 % 및 _ 문자와 정확히 일치해야 하는 경우 ESCAPE 식별자를 사용합니다. 이 옵션은 이스케이프 문자를 지정합니다. SA_ 를 포함한 문자열을 검색하려는 경우 다음 SQL 문을 사용할 수 있습니다.

```
SELECT employee_id, last_name, job_id
  FROM employees WHERE job_id LIKE '%SA\_%' ESCAPE '\';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
1	Zlotkey	SA_MAN
2	Abel	SA_REP
3	Taylor	SA_REP
4	Grant	SA_REP

ESCAPE 식별자가 백슬래시(\)를 이스케이프 문자로 식별합니다. 이 SQL 문에서 이스케이프 문자가 밑줄(_) 앞에 옵니다. 이렇게 하면 Oracle 서버가 밑줄을 문자 그대로 해석하게 됩니다.

NULL 조건 사용

IS NULL 연산자로 **null**을 테스트합니다.

```
SELECT last_name, manager_id
  FROM employees
 WHERE manager_id IS NULL ;
```

	LAST_NAME	MANAGER_ID
1	King	(null)

ORACLE

Copyright © 2007, Oracle. All rights reserved.

NULL 조건 사용

NULL 조건은 **IS NULL** 조건과 **IS NOT NULL** 조건을 포함합니다.

IS NULL 조건은 **null**을 테스트합니다. **null** 값은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값을 의미합니다. 따라서 **null**은 어떠한 값과도 같거나 같지 않을 수 없기 때문에 등호(=)를 사용하여 테스트할 수 없습니다. 슬라이드의 예제는 관리자가 없는 모든 사원의 성과 관리자를 검색합니다.

또 다른 예로, 커미션을 받을 자격이 없는 모든 사원의 성, 직무 ID 및 커미션을 표시하려면 다음 SQL 문을 사용하십시오.

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

	LAST_NAME	JOB_ID	COMMISSION_PCT
1	King	AD_PRES	(null)
2	Kochhar	AD_VP	(null)
...			
15	Higgins	AC_MGR	(null)
16	Gietz	AC_ACCOUNT	(null)

논리 연산자를 사용하여 조건 정의

연산자	의미
AND	구성 요소 조건이 모두 참인 경우 TRUE 를 반환합니다.
OR	구성 요소 조건 중 하나가 참인 경우 TRUE 를 반환합니다.
NOT	조건이 거짓인 경우 TRUE 를 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

논리 연산자를 사용하여 조건 정의

논리 조건은 두 구성 요소 조건의 결과를 결합하여 해당 조건을 기반으로 단일 결과를 산출하거나, 단일 조건의 결과를 뒤바꿉니다. 조건의 전체 결과가 참인 경우에만 행이 반환됩니다.

SQL에서는 다음 세 가지 논리 연산자가 제공됩니다.

- AND
- OR
- NOT

지금까지의 모든 예제는 WHERE 절에서 하나의 조건만 지정했습니다. AND 및 OR 연산자를 사용하여 하나의 WHERE 절에서 여러 조건을 사용할 수 있습니다.

AND 연산자 사용

AND 연산에서는 두 구성 요소 조건이 모두 참이어야 합니다.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
AND job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	149 Zlotkey	SA_MAN	10500	
2	201 Hartstein	MK_MAN	13000	

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

AND 연산자 사용

예제에서 레코드가 선택되려면 두 구성 요소 조건이 모두 참이어야 합니다. 따라서 문자열 'MAN'을 포함한 직책을 가지고 있고 \$10,000 이상의 급여를 받는 사원만 선택됩니다.

모든 문자 검색에서는 대소문자가 구분되므로 'MAN'이 대문자가 아닌 경우에는 어떠한 행도 반환되지 않습니다. 또한 문자열은 따옴표로 묶어야 합니다.

AND 진리표

다음 표는 두 표현식을 AND로 결합한 결과를 보여줍니다.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR 연산자 사용

OR 연산에서는 두 구성 요소 조건 중 하나가 참이어야 합니다.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%' ;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	100 King	AD_PRES	24000	
2	101 Kochhar	AD_VP	17000	
3	102 De Haan	AD_VP	17000	
4	124 Mourgos	ST_MAN	5800	
5	149 Zlotkey	SA_MAN	10500	
6	174 Abel	SA_REP	11000	
7	201 Hartstein	MK_MAN	13000	
8	205 Higgins	AC_MGR	12000	

ORACLE

Copyright © 2007, Oracle. All rights reserved.

OR 연산자 사용

예제에서 레코드가 선택되려면 두 구성 요소 조건 중 하나가 참이어야 합니다. 따라서 직무 ID에 문자열 'MAN'이 포함되거나 또는 \$10,000 이상의 급여를 받는 사원이 선택됩니다.

OR 진리표

다음 표는 두 표현식을 OR로 결합한 결과를 보여줍니다.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT 연산자 사용

```
SELECT last_name, job_id
FROM employees
WHERE job_id
    NOT IN ('IT_PROG', 'ST_CLERK', 'SA REP') ;
```

	LAST_NAME	JOB_ID
1	De Haan	AD_VP
2	Fay	MK_REP
3	Gietz	AC_ACCOUNT
4	Hartstein	MK_MAN
5	Higgins	AC_MGR
6	King	AD_PRES
7	Kochhar	AD_VP
8	Mourgos	ST_MAN
9	Whalen	AD_ASST
10	Zlotkey	SA_MAN

ORACLE

Copyright © 2007, Oracle. All rights reserved.

NOT 연산자 사용

슬라이드의 예제는 직무 ID가 IT_PROG, ST_CLERK 또는 SA_REP가 아닌 모든 사원의 성과 직무 ID를 표시합니다.

NOT 진리표

다음 표는 NOT 연산자를 조건에 적용한 결과를 보여줍니다.

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

주: NOT 연산자는 BETWEEN, LIKE, NULL 등의 다른 SQL 연산자와 함께 사용할 수도 있습니다.

```
... WHERE job_id      NOT IN ('AC_ACCOUNT', 'AD_VP')
... WHERE salary      NOT BETWEEN 10000 AND 15000
... WHERE last_name   NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL
```

단원 내용

- 다음과 같은 방법을 사용하여 행을 제한합니다.
 - WHERE 절
 - =, <=, BETWEEN, IN, LIKE 및 NULL 연산자를 사용하는 비교 조건
 - AND, OR 및 NOT 연산자를 사용하는 논리 조건
- 표현식의 연산자 우선 순위 규칙
- ORDER BY 절을 사용하여 행 정렬
- 치환 변수
- DEFINE 및 VERIFY 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

우선 순위 규칙

연산자	의미
1	산술 연산자
2	연결 연산자
3	비교 조건
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	같지 않음
7	NOT 논리 조건
8	AND 논리 조건
9	OR 논리 조건

괄호를 사용하여 우선 순위 규칙을 재정의할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

우선 순위 규칙

우선 순위 규칙은 표현식이 평가되고 계산되는 순서를 결정합니다. 슬라이드의 표에는 우선 순위의 기본 순서가 나열되어 있습니다. 그러나 먼저 계산하려는 표현식을 괄호로 묶어 기본 순서를 재정의할 수 있습니다.

우선 순위 규칙

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR    job_id = 'AD_PRES'
AND   salary > 15000;
```

1

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000
2	Abel	SA_REP	11000
3	Taylor	SA_REP	8600
4	Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR    job_id = 'AD_PRES')
AND   salary > 15000;
```

2

	LAST_NAME	JOB_ID	SALARY
1	King	AD_PRES	24000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

우선 순위 규칙(계속)

1. AND 연산자의 우선 순위: 예제

이 예제에는 다음 두 가지 조건이 있습니다.

- 첫번째 조건은 직무 ID가 AD_PRES 이고 급여가 \$15,000보다 많다는 것입니다.
- 두번째 조건은 직무 ID가 SA_REP라는 것입니다.

따라서 SELECT 문은 다음과 같이 인식됩니다.

"직원이 사장이면서 \$15,000가 넘는 급여를 받거나 직원이 판매 담당자인 경우의 행을 선택하십시오."

2. 괄호 사용: 예제

이 예제에는 다음 두 가지 조건이 있습니다.

- 첫번째 조건은 직무 ID가 AD_PRES 또는 SA_REP라는 것입니다.
- 두번째 조건은 급여가 \$15,000보다 많다는 것입니다.

따라서 SELECT 문은 다음과 같이 인식됩니다.

"직원이 사장이거나 판매 담당자이고 직원이 \$15,000가 넘는 급여를 받는 경우의 행을 선택하십시오."

단원 내용

- 다음과 같은 방법을 사용하여 행을 제한합니다.
 - WHERE 절
 - =, <=, BETWEEN, IN, LIKE 및 NULL 연산자를 사용하는 비교 조건
 - AND, OR 및 NOT 연산자를 사용하는 논리 조건
- 표현식의 연산자 우선 순위 규칙
- ORDER BY 절을 사용하여 행 정렬
- 치환 변수
- DEFINE 및 VERIFY 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

ORDER BY 절 사용

- ORDER BY 절을 사용하여 검색된 행을 정렬합니다.
 - **ASC:** 오름차순, 기본값
 - **DESC:** 내림차순
- ORDER BY 절은 SELECT 문의 맨 마지막에 옵니다.

```
SELECT last_name, job_id, department_id, hire_date
  FROM employees
 ORDER BY hire_date ;
```

	LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
1	King	AD_PRES		90 17-JUN-87
2	Whalen	AD_ASST		10 17-SEP-87
3	Kochhar	AD_VP		90 21-SEP-89
4	Hunold	IT_PROG		60 03-JAN-90
5	Ernst	IT_PROG		60 21-MAY-91
6	De Haan	AD_VP		90 13-JAN-93

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ORDER BY 절 사용

query 결과에 반환된 행의 순서는 정의되어 있지 않습니다. ORDER BY 절을 사용하여 행을 정렬할 수 있습니다. 그러나 SQL 문에서 ORDER BY 절을 사용하는 경우 그 뒤에 다른 절을 사용할 수 없습니다. 또한 표현식, alias 또는 열 위치를 정렬 조건으로 지정할 수 있습니다.

구문

```
SELECT      expr
  FROM       table
  [WHERE      condition(s)]
  [ORDER BY  {column, expr, numeric_position} [ASC|DESC] ] ;
```

구문 설명:

ORDER BY	검색된 행이 표시되는 순서를 지정합니다.
ASC	오름차순으로 행을 정렬합니다(기본 순서).
DESC	내림차순으로 행을 정렬합니다.

ORDER BY 절을 사용하지 않는 경우에는 정렬 순서가 정의되지 않으며 Oracle 서버는 동일한 query에 대해 동일한 순서로 행을 두 번 패치(fetch)하지 못할 수 있습니다. 특정 순서로 행을 표시하려면 ORDER BY 절을 사용합니다.

주: NULLS FIRST 또는 NULLS LAST 키워드를 사용하여 null 값을 포함하는 반환된 행이 정렬 순서상 맨 처음 나타나거나 마지막에 나타나도록 지정할 수 있습니다.

정렬

- 내림차순으로 정렬:

```
SELECT last_name, job_id, department_id, hire_date  
FROM employees  
ORDER BY hire_date DESC ;
```

1

- 열 alias를 기준으로 정렬:

```
SELECT employee_id, last_name, salary*12 annsal  
FROM employees  
ORDER BY annsal ;
```

2

ORACLE

Copyright © 2007, Oracle. All rights reserved.

정렬

기본 정렬 순서는 오름차순입니다.

- 숫자 값은 가장 낮은 값이 맨 앞에 표시됩니다. (예를 들어, 1부터 999 순서로 표시됩니다.)
- 날짜 값은 가장 이른 값이 맨 앞에 표시됩니다. (예를 들어, 01-JAN-92가 01-JAN-95 앞에 표시됩니다.)
- 문자 값은 사전순으로 표시됩니다. (예를 들어, "A"부터 "Z" 순서로 표시됩니다.)
- Null 값은 오름차순에서는 맨 뒤에 표시되고 내림차순에서는 맨 앞에 표시됩니다.
- SELECT 리스트에 없는 열을 기준으로 정렬할 수도 있습니다.

예제:

- 행이 표시되는 순서를 뒤집으려면 ORDER BY 절에서 열 이름 뒤에 DESC 키워드를 지정합니다. 슬라이드의 예제는 가장 최근에 채용된 사원을 기준으로 결과를 정렬합니다.
- ORDER BY 절에서 열 alias를 사용할 수도 있습니다. 슬라이드의 예제는 연봉 기준으로 데이터를 정렬합니다.

정렬

- 열의 숫자 위치를 사용하여 정렬

```
SELECT last_name, job_id, department_id, hire_date  
FROM employees  
ORDER BY 3;
```

3

- 여러 열을 기준으로 정렬

```
SELECT last_name, department_id, salary  
FROM employees  
ORDER BY department_id, salary DESC;
```

4

ORACLE

Copyright © 2007, Oracle. All rights reserved.

정렬(계속)

예제:

3. SELECT 절에 열의 숫자 위치를 지정하여 query 결과를 정렬할 수 있습니다.
department_id 열이 SELECT 절에서 세번째 자리에 있으므로 예제 슬라이드는 이
열을 기준으로 결과를 정렬합니다.
4. 둘 이상의 열을 기준으로 query 결과를 정렬할 수 있습니다. 제공된 테이블의 열 개수만큼
정렬 기준을 지정할 수 있습니다. ORDER BY 절에서 열을 지정하고 쉼표를 사용하여 열
이름을 구분합니다. 열 순서를 뒤바꾸려면 열 이름 뒤에 DESC를 지정합니다.

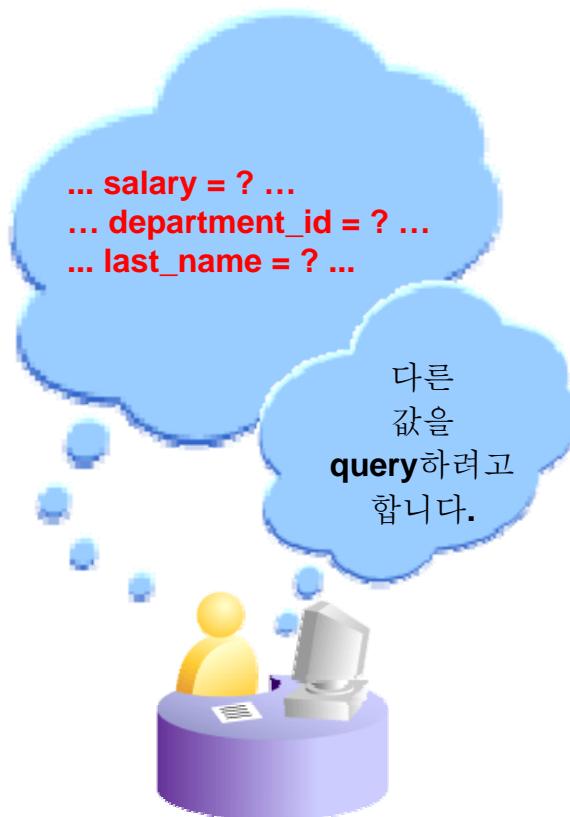
단원 내용

- 다음과 같은 방법을 사용하여 행을 제한합니다.
 - WHERE 절
 - =, <=, BETWEEN, IN, LIKE 및 NULL 연산자를 사용하는 비교 조건
 - AND, OR 및 NOT 연산자를 사용하는 논리 조건
- 표현식의 연산자 우선 순위 규칙
- ORDER BY 절을 사용하여 행 정렬
- 치환 변수
- DEFINE 및 VERIFY 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

치환 변수



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

치환 변수

지금까지 배운 모든 SQL 문은 열, 조건 및 값이 미리 결정된 채 실행되었습니다. job_ID가 SA REP인 사원뿐만 아니라 다양한 직무의 사원을 나열하는 query가 필요하다고 가정해 봅니다. WHERE 절을 편집하여 명령을 실행할 때마다 다른 값을 제공할 수도 있지만 그보다 편리한 방법이 있습니다.

WHERE 절에서 정확한 값 대신 치환 변수를 사용하여 여러 값에 대해 동일한 query를 실행할 수 있습니다.

치환 변수를 사용하면 반환되는 데이터의 범위를 제한하는 값을 유저가 직접 입력할 수 있는 보고서를 작성할 수 있습니다. 치환 변수를 명령 파일이나 단일 SQL 문에 포함시킬 수 있습니다. 변수는 값을 임시로 저장하는 컨테이너로 간주할 수 있습니다. 명령문이 실행되면 변수에 저장된 값이 치환됩니다.

치환 변수

- 치환 변수를 사용하여 다음을 수행할 수 있습니다.
 - 단일 앤페샌드(&) 및 이중 앤페샌드(&&) 치환을 사용하여 값을 임시로 저장합니다.
- 치환 변수를 사용하여 다음을 보완할 수 있습니다.
 - WHERE 조건
 - ORDER BY 절
 - 열 표현식
 - 테이블 이름
 - 전체 SELECT 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

치환 변수(계속)

단일 앤페샌드(&) 치환 변수를 사용하여 값을 임시로 저장할 수 있습니다.

또한 DEFINE 명령을 사용하여 변수를 미리 정의할 수도 있습니다. DEFINE은 변수를 생성하고 값을 할당합니다.

제한된 데이터 범위: 예제

- 현재 분기 또는 지정된 날짜 범위의 수치만 보고
- 보고서를 요청하는 유저에 관련된 데이터만 보고
- 제공된 부서 내의 사원만 표시

기타 대화식 효과

대화식 효과는 WHERE 절을 통한 직접적인 유저 상호 작용에만 국한되지는 않습니다. 다음과 같은 다른 목적을 위해 동일한 원칙을 사용할 수도 있습니다.

- 유저가 아니라 파일로부터 입력 값을 구하기
- SQL 문 사이에 값을 전달

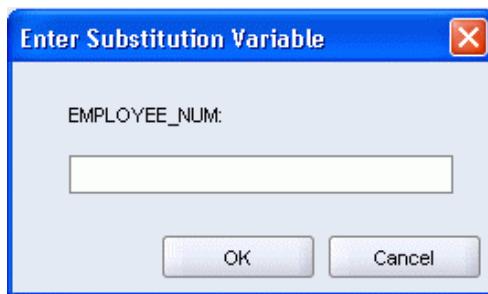
주: SQL Developer 및 SQL* Plus는 모두 치환 변수 및 DEFINE/UNDEFINE 명령을 지원합니다.

그러나 SQL Developer 또는 SQL* Plus는 유저 입력에 대한 유효성 검사를 지원하지 않습니다(데이터 유형 제외).

단일 앤퍼샌드 치환 변수 사용

변수 앞에 앤퍼샌드(**&**)를 붙이면 유저가 값을 입력하도록 할 수 있습니다.

```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```



ORACLE

Copyright © 2007, Oracle. All rights reserved.

단일 앤퍼샌드 치환 변수 사용

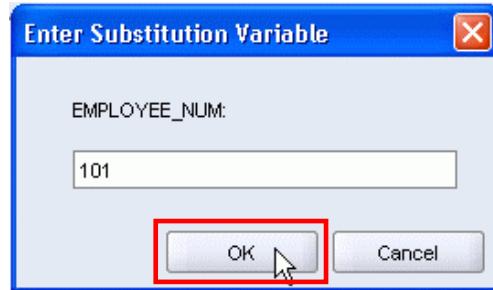
보고서 실행 시 유저는 종종 반환되는 데이터를 동적으로 제한해야 합니다. SQL*Plus 또는 SQL Developer는 유저 변수를 통해 이러한 유연성을 제공합니다. 앤퍼샌드(&)를 사용하여 SQL 문에서 각 변수를 식별합니다. 그러나 각 변수 값을 정의할 필요는 없습니다.

표기법	설명
<code>&user_variable</code>	SQL 문의 변수를 표시합니다. 변수가 존재하지 않을 경우 SQL*Plus 또는 SQL Developer는 유저에게 값을 입력하도록 요청합니다. (새 변수는 사용된 후 폐기됩니다.)

슬라이드의 예제는 사원 번호에 대해 SQL Developer 치환 변수를 생성합니다. 명령문이 실행되면 SQL Developer는 유저에게 사원 번호를 입력하도록 요청한 다음 해당 사원의 사원 번호, 성, 급여 및 부서 번호를 표시합니다.

단일 앤퍼샌드를 사용하는 경우 변수가 존재하지 않으면 명령이 실행될 때마다 유저에게 값을 입력하도록 요청합니다.

단일 앤퍼샌드 치환 변수 사용



	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단일 앤퍼샌드 치환 변수 사용(계속)

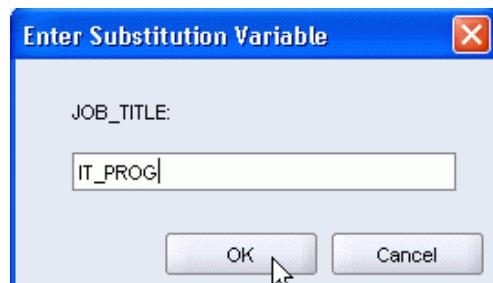
SQL Developer는 SQL 문에 앤퍼샌드가 포함된 것을 감지하면 SQL 문에 지정된 치환 변수의 값을 입력하도록 요청합니다.

값을 입력한 후에 OK 버튼을 누르면 SQL Developer 세션의 Results 탭에 결과가 표시됩니다.

문자 및 날짜 값을 치환 변수로 지정

날짜 값 및 문자 값에 대해 작은 따옴표를 사용합니다.

```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```



	LAST_NAME	DEPARTMENT_ID	SALARY*12
1	Hunold	60	108000
2	Ernst	60	72000
3	Lorentz	60	50400

ORACLE

Copyright © 2007, Oracle. All rights reserved.

문자 및 날짜 값을 치환 변수로 지정

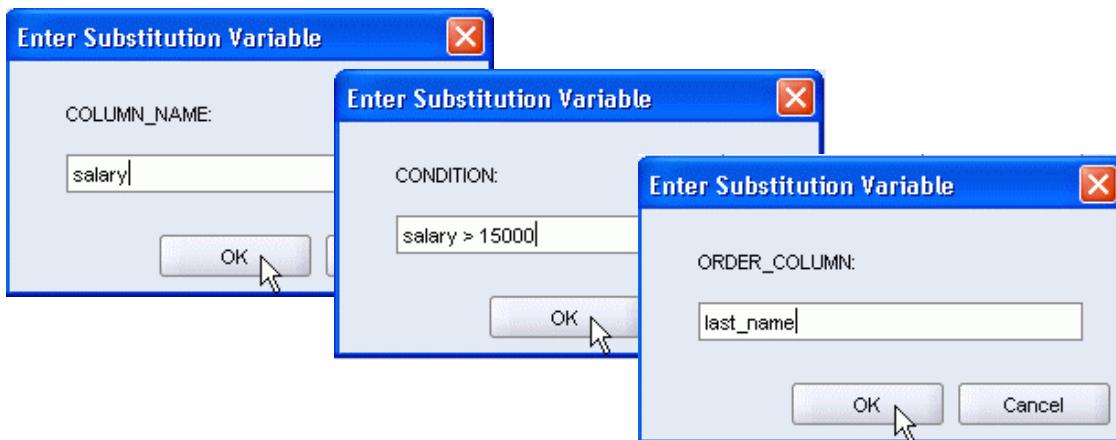
WHERE 절에서 날짜 및 문자 값은 작은 따옴표로 묶어야 합니다. 같은 규칙이 치환 변수에 적용됩니다.

SQL 문 자체 내에서 변수를 작은 따옴표로 묶습니다.

슬라이드는 SQL Developer 치환 변수의 직책 값을 기반으로 모든 사원의 사원 이름, 부서 번호 및 연봉을 검색하는 query를 보여줍니다.

열 이름, 표현식 및 텍스트 지정

```
SELECT employee_id, last_name, job_id, &column_name
FROM employees
WHERE &condition
ORDER BY &order_column;
```



ORACLE

Copyright © 2007, Oracle. All rights reserved.

열 이름, 표현식 및 텍스트 지정

치환 변수는 SQL 문의 WHERE 절에 사용할 수 있을 뿐만 아니라 열 이름, 표현식 또는 텍스트를 치환하는 데 사용할 수도 있습니다.

예제:

슬라이드의 예제는 EMPLOYEES 테이블에서 사원 번호, 성, 직책 및 런타임 시 유저가 지정한 기타 열을 표시합니다. SELECT 문의 각 치환 변수에 대해 값을 입력하도록 요청하면 OK를 눌러 계속 진행합니다.

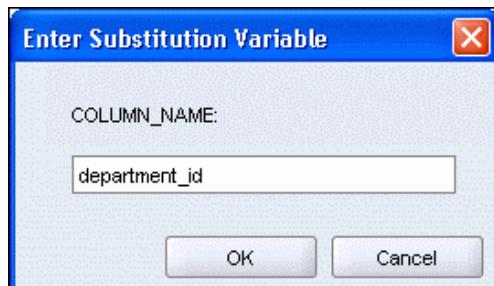
치환 변수의 값을 입력하지 않으면 앞의 명령문을 실행할 때 오류 메시지가 나타납니다.

주: 치환 변수는 명령 프롬프트에서 첫번째 단어로 입력하지만 않으면 SELECT 문의 어느 위치에서나 사용할 수 있습니다.

이중 앤퍼샌드 치환 변수 사용

유저가 매번 값을 입력할 필요 없이 변수 값을 재사용하려는 경우 이중 앤퍼샌드(&&)를 사용합니다.

```
SELECT employee_id, last_name, job_id, &&column_name
  FROM employees
 ORDER BY &column_name ;
```



	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	200	Whalen	AD_ASST	10
2	201	Hartstein	MK_MAN	20
3	202	Fay	MK_REP	20

Copyright © 2007, Oracle. All rights reserved.

ORACLE

이중 앤퍼샌드 치환 변수 사용

유저가 매번 값을 입력할 필요 없이 변수 값을 재사용하려는 경우 이중 앤퍼샌드(&&) 치환 변수를 사용할 수 있습니다. 유저는 값을 한 번만 입력하면 됩니다. 슬라이드의 예제에서는 유저에게 column_name 변수의 값을 한 번만 입력하도록 요구합니다. 유저가 제공하는 값(department_id)은 데이터를 표시하고 순서를 지정하는 데 사용됩니다. query를 다시 실행하면 유저에게 해당 변수 값을 입력하라고 요청하지 않습니다.

SQL Developer는 DEFINE 명령을 사용하여 제공한 값을 저장한 다음 변수 이름이 참조될 때마다 다시 사용합니다. 유저 변수가 배치된 후에 다음과 같이 UNDEFINE 명령을 사용하여 삭제해야 합니다.

```
UNDEFINE column_name
```

단원 내용

- 다음과 같은 방법을 사용하여 행을 제한합니다.
 - WHERE 절
 - =, <=, BETWEEN, IN, LIKE 및 NULL 연산자를 사용하는 비교 조건
 - AND, OR 및 NOT 연산자를 사용하는 논리 조건
- 표현식의 연산자 우선 순위 규칙
- ORDER BY 절을 사용하여 행 정렬
- 치환 변수
- DEFINE 및 VERIFY 명령

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

DEFINE 명령 사용

- **DEFINE** 명령을 사용하여 변수를 생성하고 값을 할당합니다.
- **UNDEFINE** 명령을 사용하여 변수를 제거합니다.

```
DEFINE employee_num = 200
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num;

UNDEFINE employee_num
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DEFINE 명령 사용

위의 예제는 DEFINE 명령을 사용하여 사원 번호에 대한 치환 변수를 생성합니다. 런타임 시 이 변수는 해당 사원의 사원 번호, 이름, 급여 및 부서 번호를 표시합니다.

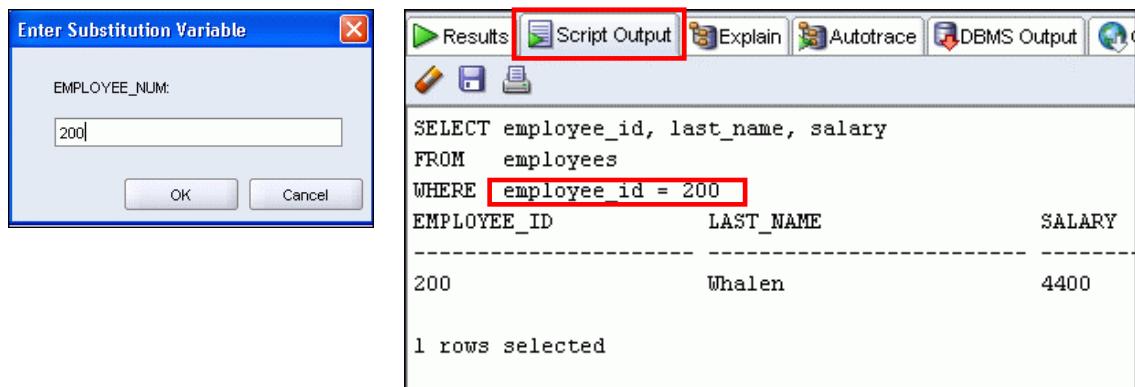
이 변수는 SQL Developer DEFINE 명령을 사용하여 생성되기 때문에 유저에게 사원 번호에 대한 값을 입력하도록 요청하지 않습니다. 대신 정의된 변수 값이 SELECT 문에서 자동으로 치환됩니다.

EMPLOYEE_NUM 치환 변수는 유저가 해당 변수의 정의를 해제하거나 SQL Developer 세션을 종료할 때까지 세션에 남아 있습니다.

VERIFY 명령 사용

VERIFY 명령을 사용하여 **SQL Developer**가 치환 변수를 값으로 바꾸기 전후에 치환 변수의 표시를 토글합니다.

```
SET VERIFY ON
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = &employee_num;
```



ORACLE

Copyright © 2007, Oracle. All rights reserved.

VERIFY 명령 사용

SQL 문의 변경 사항을 확인하려면 **VERIFY** 명령을 사용합니다. **SET VERIFY**를 **ON**으로 설정하면 **SQL Developer**가 치환 변수를 값으로 바꾼 후의 명령 텍스트가 표시됩니다. **VERIFY** 출력을 보려면 **SQL Worksheet**에서 **Run Script(F5)** 아이콘을 사용해야 합니다. **SQL Developer**는 치환 변수를 값으로 바꾼 후 슬라이드에 나오는 것처럼 **Script Output** 탭에 명령 텍스트를 표시합니다. 슬라이드의 예제는 **EMPLOYEE_ID** 열의 새 값을 SQL 문에 표시한 다음 결과를 출력합니다.

SQL*Plus 시스템 변수

SQL*Plus는 작업 환경을 제어하는 다양한 시스템 변수를 사용합니다. **VERIFY**는 이러한 시스템 변수 중 하나입니다. 시스템 변수의 전체 리스트를 보려면 **SQL*Plus** 명령 프롬프트에서 **SHOW ALL** 명령을 실행하십시오.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- **WHERE** 절을 사용하여 출력 행 제한
 - 비교 조건 사용
 - **BETWEEN, IN, LIKE** 및 **NULL** 연산자 사용
 - **AND, OR, NOT** 논리 연산자 적용
- **ORDER BY** 절을 사용하여 출력 행 정렬

```
SELECT * | { [DISTINCT] column / expression [alias], ... }  
FROM   table  
[WHERE condition(s)]  
[ORDER BY {column, expr, alias} [ASC|DESC]] ;
```

- 앤퍼샌드 치환을 사용하여 런타임 시 출력 제한 및 정렬

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

이 단원에서는 SELECT 문에 의해 반환되는 행을 제한하고 정렬하는 방법에 대해 배웠습니다.

또한 다양한 연산자와 조건을 구현하는 방법도 배웠습니다.

치환 변수를 사용하면 SQL 문에 유연성을 부여할 수 있습니다. 이를 활용하여 런타임 시 행에 대한 필터 조건을 입력하도록 요청하는 query를 작성할 수 있습니다.

연습 2: 개요

이 연습에서는 다음 내용을 다룹니다.

- 데이터 선택 및 표시되는 행의 순서 변경
- WHERE** 절을 사용하여 행 제한
- ORDER BY** 절을 사용하여 행 정렬
- 치환 변수를 사용하여 **SQL SELECT** 문에 유연성 부여

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

연습 2: 개요

이 연습에서는 WHERE 절과 ORDER BY 절을 사용하는 명령문을 비롯하여 추가 보고서를 작성하는 과정을 다룹니다. 앤퍼샌드 치환을 포함시키면 SQL 문의 재사용 및 범용성을 높일 수 있습니다.

연습 2

HR 부서에서 몇 가지 query 작성과 관련해 여러분의 도움을 요청합니다.

1. HR 부서에서 예산 문제 때문에 급여가 \$12,000가 넘는 사원의 성과 급여를 표시하는 보고서가 필요합니다. 작성한 SQL 문을 lab_02_01.sql이라는 텍스트 파일로 저장합니다. query를 실행합니다.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

2. 새 SQL Worksheet를 엽니다. 사원 번호 176의 성과 부서 번호를 표시하는 보고서를 작성합니다. query를 실행합니다.

	LAST_NAME	DEPARTMENT_ID
1	Taylor	80

3. HR 부서에서 급여가 높은 사원과 급여가 낮은 사원을 찾으려고 합니다. 급여가 \$5,000 ~ \$12,000 범위에 속하지 않는 사원의 성과 급여를 표시하도록 lab_02_01.sql을 수정합니다. 작성한 SQL 문을 lab_02_03.sql이라는 텍스트 파일로 저장합니다.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Lorentz	4200
5	Rajs	3500
6	Davies	3100
7	Matos	2600
8	Vargas	2500
9	Whalen	4400
10	Hartstein	13000

연습 2(계속)

4. Matos 및 Taylor라는 성을 가진 사원에 대해 성, 직무 ID, 시작 날짜를 표시하는 보고서를 작성합니다. 시작 날짜를 기준으로 오름차순으로 query를 정렬합니다.

	LAST_NAME	JOB_ID	HIRE_DATE
1	Matos	ST_CLERK	15-MAR-98
2	Taylor	SA_REP	24-MAR-98

5. 부서 20 또는 50에 속하는 모든 사원의 성과 부서 번호를 이름을 기준으로 오름차순으로 정렬하여 표시합니다.

	LAST_NAME	DEPARTMENT_ID
1	Davies	50
2	Fay	20
3	Hartstein	20
4	Matos	50
5	Mourgos	50
6	Rajs	50
7	Vargas	50

6. \$5,000 ~ \$12,000의 급여를 받고 부서 20 또는 50에 속하는 사원의 성과 급여를 표시하도록 lab_02_03.sql을 수정합니다. 열 레이블을 각각 Employee 및 Monthly Salary로 지정합니다. lab_02_03.sql을 lab_02_06.sql로 다시 저장합니다.
lab_02_06.sql의 명령문을 실행합니다.

	Employee	Monthly Salary
1	Fay	6000
2	Mourgos	5800

연습 2(계속)

7. HR 부서에서 1994년에 채용된 모든 사원의 성과 채용 날짜를 표시하는 보고서를 요구합니다.

	LAST_NAME	HIRE_DATE
1	Higgins	07-JUN-94
2	Gietz	07-JUN-94

8. 담당 관리자가 없는 모든 사원의 성과 직책을 표시하는 보고서를 작성합니다.

	LAST_NAME	JOB_ID
1	King	AD_PRES

9. 커미션을 받는 모든 사원의 성, 급여 및 커미션을 표시하는 보고서를 작성합니다. 급여 및 커미션의 내림차순으로 데이터를 정렬합니다.
ORDER BY 절에서 열의 숫자 위치를 사용합니다.

	LAST_NAME	SALARY	COMMISSION_PCT
1	Abel	11000	0.3
2	Zlotkey	10500	0.2
3	Taylor	8600	0.2
4	Grant	7000	0.15

10. HR 부서의 멤버는 여러분이 작성 중인 query에 유연성이 확대되기를 원합니다. 그들은 유저가 프롬프트에 지정하는 액수보다 많은 급여를 받은 사원이 있을 경우 이들의 성과 급여를 표시하는 보고서를 기대합니다. 이 query를 lab_02_10.sql이라는 파일에 저장합니다. 프롬프트가 표시되었을 때 12000을 입력하면 보고서에 다음 결과가 표시됩니다.

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000

연습 2(계속)

11. HR 부서에서 관리자를 기준으로 보고서를 실행하려고 합니다. 유저에게 관리자 ID 입력 프롬프트를 표시하고 해당 관리자에 속한 사원의 사원 ID, 성, 급여 및 부서를 생성하는 query를 작성합니다. HR 부서에서 선택한 열을 기준으로 보고서를 정렬하는 기능을 원합니다. 다음 값을 사용하여 데이터를 테스트 할 수 있습니다.

manager_id = 103, last_name을 기준으로 정렬:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	104	Ernst	6000	60
2	107	Lorentz	4200	60

manager_id = 201, salary를 기준으로 정렬:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	202	Fay	6000	20

manager_id = 124, employee_id를 기준으로 정렬:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	141	Rajs	3500	50
2	142	Davies	3100	50
3	143	Matos	2600	50
4	144	Vargas	2500	50

연습 2(계속)

시간 여유가 있을 경우 다음 연습을 완료하십시오.

12. 성의 세번째 문자가 "a"인 모든 사원의 성을 표시합니다.

	LAST_NAME
1	Grant
2	Whalen

13. 성에 "a"와 "e"가 모두 포함된 모든 사원의 성을 표시합니다.

	LAST_NAME
1	Davies
2	De Haan
3	Hartstein
4	Whalen

심화 연습에 도전하려면 다음 연습을 완료하십시오.

14. 직무가 판매 사원이나 자재 담당자이고 급여가 \$2,500, \$3,500 또는 \$7,000가 아닌 모든 사원의 성, 직무 및 급여를 표시합니다.

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000
2	Taylor	SA_REP	8600
3	Davies	ST_CLERK	3100
4	Matos	ST_CLERK	2600

15. 커미션이 20%인 모든 사원의 성, 급여 및 커미션을 표시하도록 lab_02_06.sql을 수정합니다. lab_02_06.sql을 lab_02_15.sql로 다시 저장합니다.
lab_02_15.sql의 명령문을 재실행합니다.

	Employee	Monthly Salary	COMMISSION_PCT
1	Zlotkey	10500	0.2
2	Taylor	8600	0.2

3

단일 행 함수를 사용하여 출력 커스터마이즈

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **SQL**에서 사용할 수 있는 다양한 유형의 함수 설명
- **SELECT** 문에서 문자, 숫자 및 날짜 함수 사용



Copyright © 2007, Oracle. All rights reserved.

목표

함수를 사용하면 기본 query 블록을 보다 강력하게 구현할 수 있으며 데이터 값을 조작할 수 있습니다. 이 단원부터 시작하여 두 단원에 걸쳐 함수에 대해 설명합니다. 이 단원에서는 단일 행 문자, 숫자 및 날짜 함수를 중점적으로 설명합니다.

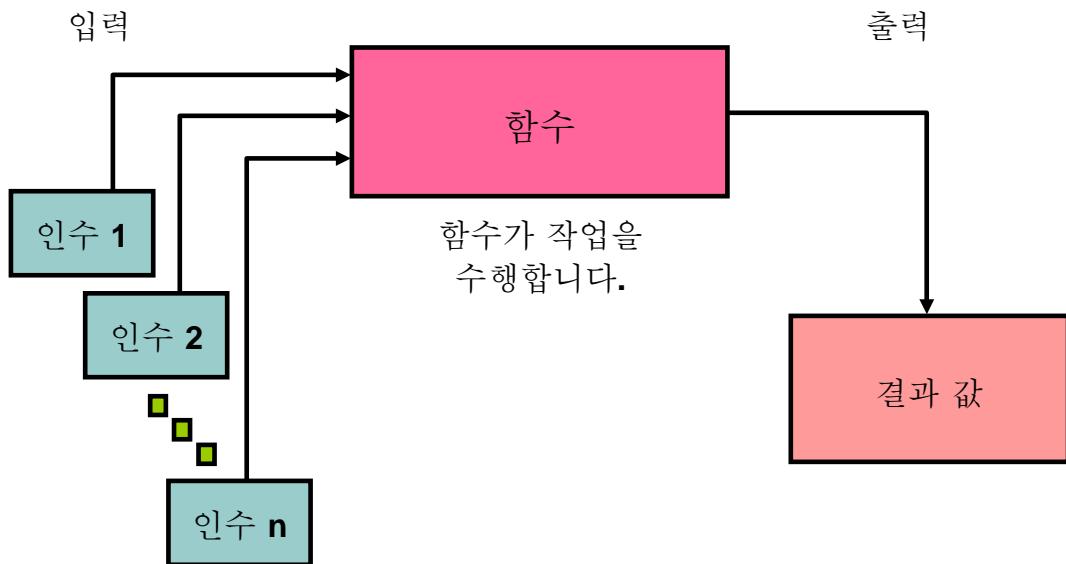
단원 내용

- 단일 행 **SQL** 함수
- 문자 함수
- 숫자 함수
- 날짜 작업
- 날짜 함수

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SQL 함수



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SQL 함수

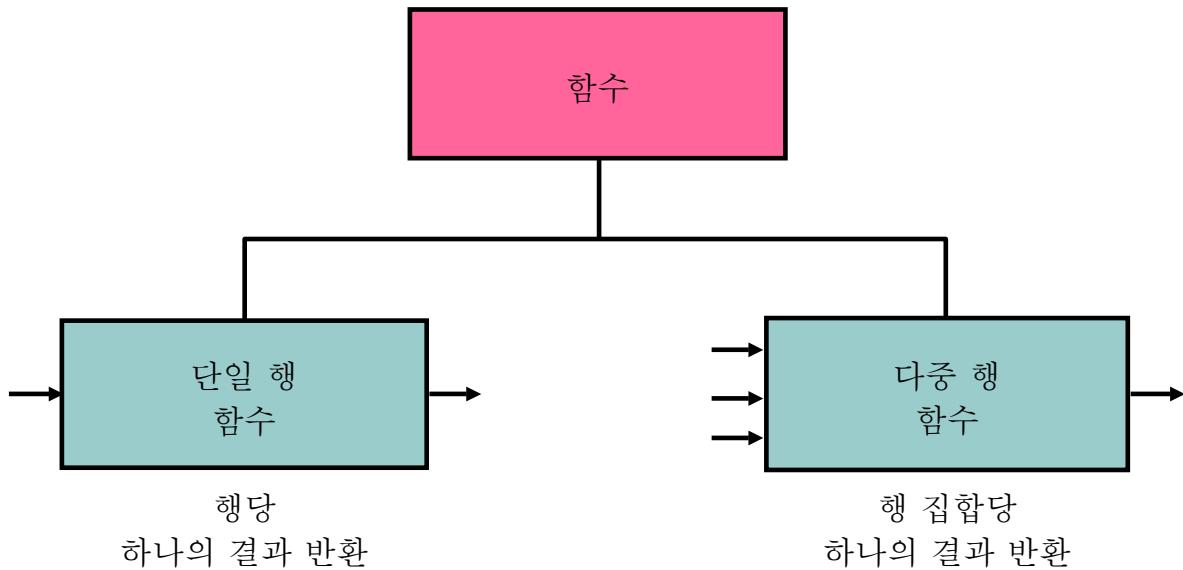
함수는 SQL의 매우 강력한 기능입니다. 다음 작업을 수행하는 데 함수를 사용할 수 있습니다.

- 데이터에 대해 계산 수행
- 개별 데이터 항목 수정
- 행 그룹에 대한 출력 조작
- 표시할 날짜 및 숫자의 형식 지정
- 열 데이터 유형 변환

SQL 함수는 때때로 인수를 받아들일 수 있으며 항상 값을 반환합니다.

주: 함수가 SQL:2003 호환 함수인지 알아보려면 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*의 *Oracle Compliance To Core SQL:2003* 섹션을 참조하십시오.

SQL 함수의 두 가지 유형



ORACLE

Copyright © 2007, Oracle. All rights reserved.

SQL 함수의 두 가지 유형

다음 두 가지 유형의 함수가 있습니다.

- 단일 행 함수
- 다중 행 함수

단일 행 함수

이 함수는 단일 행에서만 실행되며 행당 하나의 결과를 반환합니다. 단일 행 함수에는 여러 가지 유형이 있습니다. 이 단원에서는 다음 유형에 대해 다릅니다.

- 문자
- 숫자
- 날짜
- 변환
- 일반

다중 행 함수

이 함수는 행 그룹당 하나의 결과를 산출하도록 행 그룹을 조작할 수 있습니다. 이러한 함수는 그룹 함수라고도 하며 5단원 "그룹 함수를 사용한 집계 데이터 보고"에서 다릅니다.

주: 사용할 수 있는 함수 및 해당 구문에 대한 자세한 내용과 전체 리스트는 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*의 *Functions* 항목을 참조하십시오.

단일 행 함수

단일 행 함수:

- 데이터 항목을 조작합니다.
- 인수를 받아들이고 하나의 값을 반환합니다.
- 반환되는 각 행에서 실행됩니다.
- 해당 하나의 결과를 반환합니다.
- 데이터 유형을 수정할 수 있습니다.
- 중첩될 수 있습니다.
- 열이나 표현식을 인수로 받아들일 수 있습니다.

```
function_name [(arg1, arg2, ...)]
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

단일 행 함수

단일 행 함수는 데이터 항목을 조작하는 데 사용됩니다. 하나 이상의 인수를 받아들이고 query에 의해 반환되는 각 행에 대해 하나의 값을 반환합니다. 인수는 다음 중 하나가 될 수 있습니다.

- 유저가 제공하는 상수
- 변수 값
- 열 이름
- 표현식

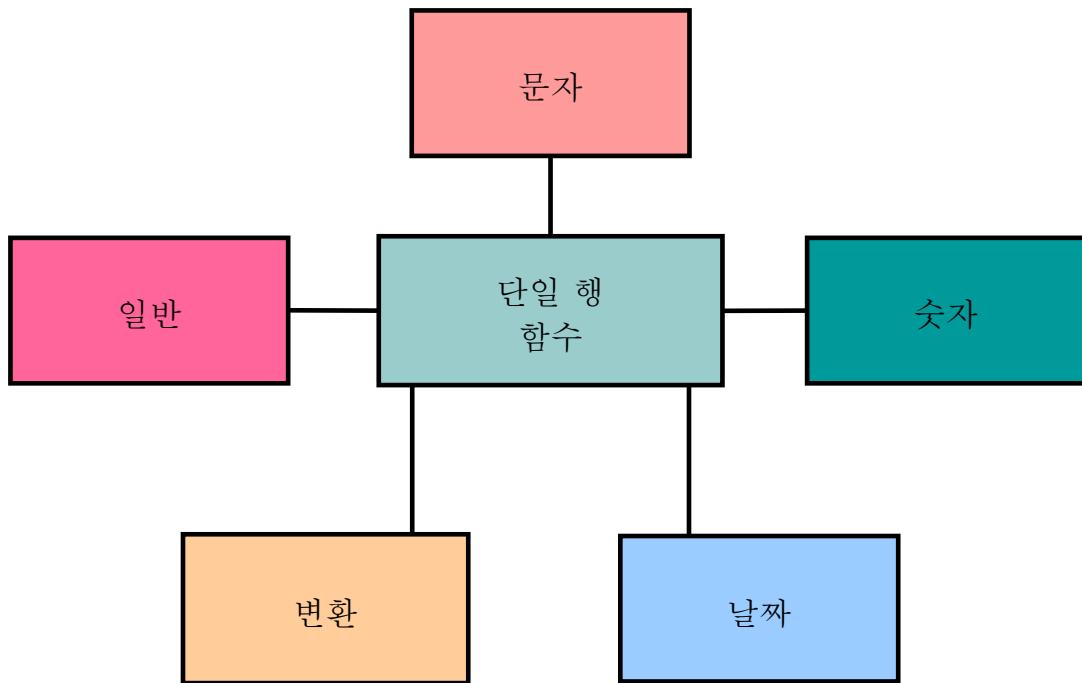
단일 행 함수의 기능에는 다음이 포함됩니다.

- query를 통해 반환되는 각 행에서 실행
- 해당 하나의 결과 반환
- 참조되는 유형이 아닌 다른 유형의 데이터 값을 반환할 수 있음
- 하나 이상의 인수를 받아들일 수 있음
- SELECT, WHERE, ORDER BY 절에서 사용할 수 있고 중첩될 수 있음

구문 설명:

<i>function_name</i>	함수의 이름입니다.
<i>arg1, arg2</i>	함수에 사용될 임의의 인수입니다. 열 이름이나 표현식으로 나타낼 수 있습니다.

단일 행 함수



ORACLE

Copyright © 2007, Oracle. All rights reserved.

단일 행 함수(계속)

이 단원에서는 다음의 단일 행 함수에 대해 다릅니다.

- 문자 함수: 문자 입력을 받아들이며 문자 및 숫자 값을 모두 반환할 수 있습니다.
- 숫자 함수: 숫자 입력을 받아들이고 숫자 값을 반환합니다.
- 날짜 함수: DATE 데이터 유형의 값에 대해 실행됩니다(숫자를 반환하는 MONTHS_BETWEEN 함수를 제외하고 모든 날짜 함수는 DATE 데이터 유형의 값을 반환합니다.)

다음의 단일 행 함수는 다음 단원 "변환 함수 및 조건부 표현식 사용"에서 설명합니다.

- 변환 함수: 값의 데이터 유형을 변환합니다.
- 일반 함수:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
 - CASE
 - DECODE

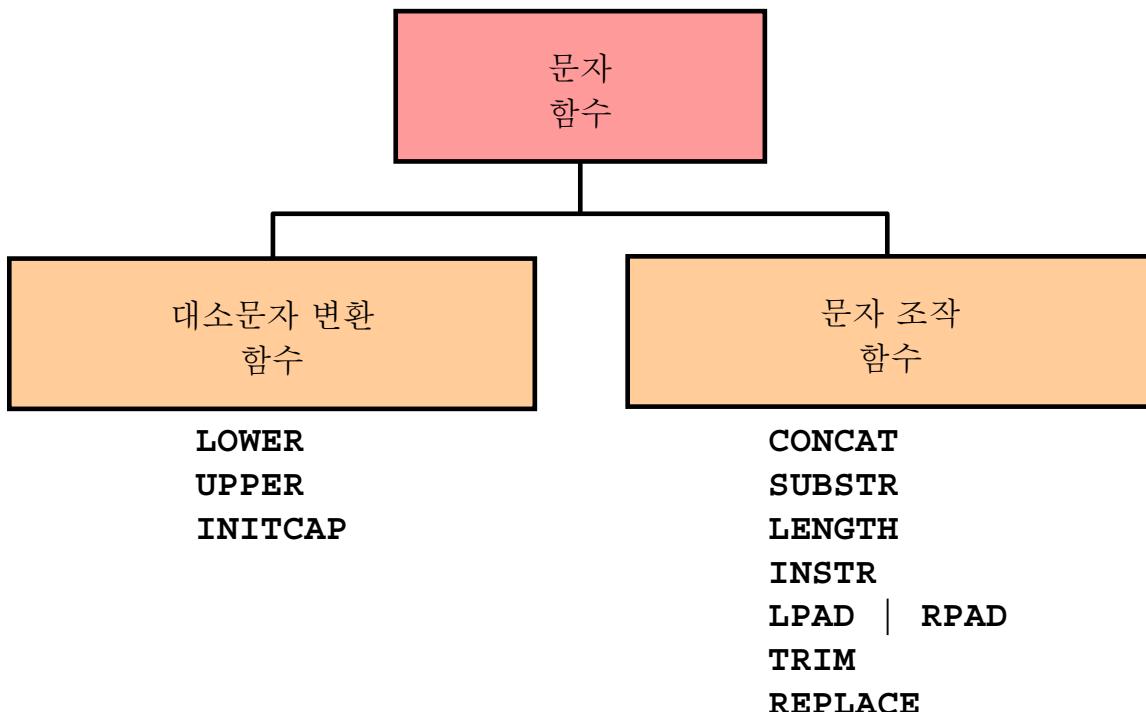
단원 내용

- 단일 행 **SQL** 함수
- 문자 함수
- 숫자 함수
- 날짜 작업
- 날짜 함수

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

문자 함수



ORACLE

Copyright © 2007, Oracle. All rights reserved.

문자 함수

단일 행 문자 함수는 입력으로 문자 데이터를 받아들이고 문자 및 숫자 값을 모두 반환할 수 있습니다. 문자 함수는 다음과 같이 분류될 수 있습니다.

- 대소문자 변환 함수
- 문자 조작 함수

함수	목적
LOWER (<i>column/expression</i>)	영문자 값을 소문자로 변환합니다.
UPPER (<i>column/expression</i>)	영문자 값을 대문자로 변환합니다.
INITCAP (<i>column/expression</i>)	영문자 값의 첫번째 문자를 대문자로 변환하고 나머지 문자는 소문자로 합니다.
CONCAT (<i>column1/expression1</i> , <i>column2/expression2</i>)	첫번째 문자 값을 두번째 문자 값과 연결합니다. 연결 연산자()와 같은 기능입니다.
SUBSTR (<i>column/expression</i> , <i>m [,n]</i>)	<i>m</i> 위치에서 시작하는 문자 값에서 <i>n</i> 개의 문자 길이만큼 지정된 문자들을 반환합니다(<i>m</i> 이 음수인 경우 문자 값 끝에서부터 카운트를 시작합니다. <i>n</i> 이 생략된 경우 문자열의 끝까지 모든 문자가 반환됩니다.)

주: 이 단원에서 설명하는 함수는 제공되는 함수 중 일부입니다.

문자 함수(계속)

함수	목적
LENGTH(<i>column/expression</i>)	표현식의 문자 수를 반환합니다.
INSTR(<i>column/expression</i> , ' <i>string</i> ', [<i>m</i>], [<i>n</i>])	지정된 문자열의 숫자 위치를 반환합니다. 선택적으로 검색 시작 위치 <i>m</i> 과 문자열의 발생 수 <i>n</i> 을 제공할 수 있습니다. <i>m</i> 과 <i>n</i> 의 기본값은 1이며, 이 경우 문자열의 처음부터 검색을 시작하고 첫번째로 찾은 결과를 보고합니다.
LPAD(<i>column/expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column/expression</i> , <i>n</i> , ' <i>string</i> ')	길이가 <i>n</i> 이 되도록 왼쪽부터 문자식으로 채운 표현식을 반환합니다. 길이가 <i>n</i> 이 되도록 오른쪽부터 문자식으로 채운 표현식을 반환합니다.
TRIM(<i>leading/trailing/both</i> , <i>trim_character FROM</i> <i>trim_source</i>)	문자열에서 선행 또는 후행 문자(또는 둘 다)를 자를 수 있습니다. <i>trim_character</i> 또는 <i>trim_source</i> 가 문자 리터럴인 경우 작은 따옴표로 묶어야 합니다. 이 기능은 Oracle8i 이상의 버전에서 제공됩니다.
REPLACE(<i>text</i> , <i>search_string</i> , <i>replacement_string</i>)	텍스트 표현식에서 문자열을 검색하여 해당 문자열을 찾으면 지정된 대체 문자열로 바꿉니다.

주: SQL:2003에 완전히 또는 부분적으로 호환되는 함수 중 일부는 다음과 같습니다.

UPPER

LOWER

TRIM

LENGTH

SUBSTR

INSTR

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*의 *Oracle Compliance To Core SQL:2003* 섹션을 참조하십시오.

대소문자 변환 함수

다음은 문자열의 대소문자를 변환하는 함수입니다.

함수	결과
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

Copyright © 2007, Oracle. All rights reserved.

대소문자 변환 함수

LOWER, UPPER 및 INITCAP의 세 가지 대소문자 변환 함수가 있습니다.

- LOWER: 대소문자가 혼합되어 있거나 대문자로 된 문자열을 소문자로 변환합니다.
- UPPER: 대소문자가 혼합되어 있거나 소문자로 된 문자열을 대문자로 변환합니다.
- INITCAP: 각 단어의 첫번째 문자를 대문자로 변환하고 나머지 문자를 소문자로 변환합니다.

```
SELECT 'The job id for '||UPPER(last_name)||' is '
      ||LOWER(job_id) AS "EMPLOYEE DETAILS"
   FROM employees;
```

EMPLOYEE DETAILS	
1	The job id for ABEL is sa_rep
2	The job id for DAVIES is st_clerk
3	The job id for DE HAAN is ad_vp
...	
19	The job id for WHALEN is ad_asst
20	The job id for ZLOTKEY is sa_man

대소문자 변환 함수 사용

사원 Higgins의 사원 번호, 이름 및 부서 번호를 표시합니다.

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
0 rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

ORACLE

Copyright © 2007, Oracle. All rights reserved.

대소문자 변환 함수 사용

슬라이드의 예제는 사원 Higgins의 사원 번호, 이름 및 부서 번호를 표시합니다.

첫번째 SQL 문의 WHERE 절은 사원 이름을 higgins로 지정합니다. EMPLOYEES 테이블의 모든 데이터는 대소문자를 구분하여 저장되기 때문에 테이블에서 higgins라는 이름과 일치하는 항목을 찾지 못하고 행이 선택되지 않습니다.

두번째 SQL 문의 WHERE 절은 EMPLOYEES 테이블의 사원 이름을 higgins와 비교하도록 지정하는데, 이때 비교를 위해 LAST_NAME 열을 소문자로 변환합니다. 이제 두 이름이 모두 소문자이기 때문에 일치하는 항목을 찾게 되고 한 행이 선택됩니다. WHERE 절을 다음과 같이 다시 작성해도 동일한 결과를 얻을 수 있습니다.

```
...WHERE last_name = 'Higgins'
```

출력되는 이름은 데이터베이스에 저장된 그대로 나타납니다. 이름을 대문자로 표시하려면 SELECT 문에 UPPER 함수를 사용합니다.

```
SELECT employee_id, UPPER(last_name), department_id
FROM   employees
WHERE  INITCAP(last_name) = 'Higgins';
```

문자 조작 함수

다음은 문자열을 조작하는 함수입니다.

함수	결과
<code>CONCAT('Hello', 'World')</code>	HelloWorld
<code>SUBSTR('HelloWorld', 1, 5)</code>	Hello
<code>LENGTH('HelloWorld')</code>	10
<code>INSTR('HelloWorld', 'W')</code>	6
<code>LPAD(salary, 10, '*')</code>	*****24000
<code>RPAD(salary, 10, '*')</code>	24000*****
<code>REPLACE('JACK and JUE', 'J', 'BL')</code>	BLACK and BLUE
<code>TRIM('H' FROM 'HelloWorld')</code>	elloWorld



Copyright © 2007, Oracle. All rights reserved.

문자 조작 함수

이 단원에서는 문자 조작 함수 CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, TRIM을 다룹니다.

- **CONCAT:** 값을 함께 연결합니다(CONCAT에서 사용할 수 있는 파라미터는 두 개로 제한됩니다.)
- **SUBSTR:** 지정된 길이의 문자열을 추출합니다.
- **LENGTH:** 문자열 길이를 숫자 값으로 표시합니다.
- **INSTR:** 명명된 문자의 숫자 위치를 찾습니다.
- **LPAD:** 길이가 n 이 되도록 왼쪽부터 문자식으로 채운 표현식을 반환합니다.
- **RPAD:** 길이가 n 이 되도록 오른쪽부터 문자식으로 채운 표현식을 반환합니다.
- **TRIM:** 문자열에서 선형 문자나 후행 문자(또는 둘 다)를 자릅니다(*trim_character* 또는 *trim_source*가 문자 리터럴인 경우 작은 따옴표로 묶어야 합니다.)

주: UPPER 및 LOWER와 같은 함수를 앤퍼샌드 치환과 함께 사용할 수 있습니다. 예를 들어, `UPPER('&job_title')`을 사용하면 유저가 직책을 입력할 때 대소문자를 구분할 필요가 없습니다.

문자 조작 함수 사용

```

SELECT employee_id, CONCAT(first_name, last_name) NAME,
       job_id, LENGTH(last_name),
       INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
    
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
1	EllenAbel	SA_REP	4	0
2	JonathonTaylor	SA_REP	6	2
3	KimberelyGrant	SA_REP	5	3
4	PatFay	MK_REP	3	2

ORACLE

Copyright © 2007, Oracle. All rights reserved.

문자 조작 함수 사용

슬라이드의 예제는 직무 ID의 네번째 위치에 REP 문자열이 포함된 모든 사원에 대해 사원 이름과 성이 함께 연결된 값, 사원 성의 길이, 사원 성에서 문자 "a"가 나타나는 숫자 위치를 표시합니다.

예제:

사원 성이 문자 "n"으로 끝나는 사원의 데이터를 표시하도록 슬라이드의 SQL 문을 수정합니다.

```

SELECT employee_id, CONCAT(first_name, last_name) NAME,
       LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
    
```

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	LexDe Haan	7	5
2	JenniferWhalen	6	3
3	MichaelHartstein	9	2

단원 내용

- 단일 행 **SQL** 함수
- 문자 함수
- 숫자 함수
- 날짜 작업
- 날짜 함수

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

숫자 함수

- **ROUND:** 지정된 소수점 자릿수로 값을 반올림합니다.
- **TRUNC:** 지정된 소수점 자릿수로 값을 **truncate**합니다.
- **MOD:** 나눈 나머지를 반환합니다.

함수	결과
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

숫자 함수

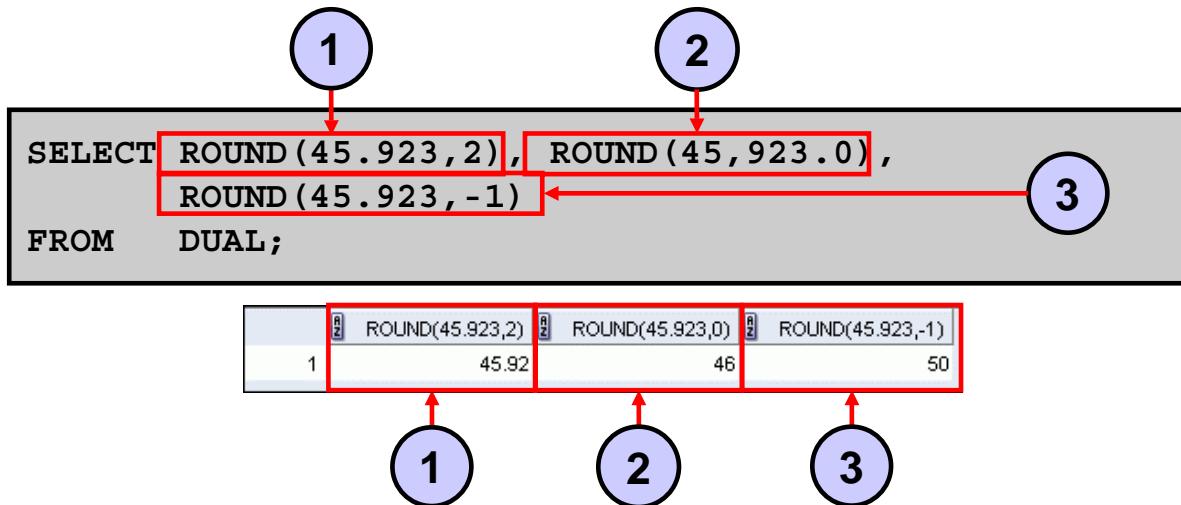
숫자 함수는 숫자 입력을 받아들이고 숫자 값을 반환합니다. 이 섹션에서는 숫자 함수 중 일부에 대해 설명합니다.

함수	목적
ROUND(<i>column expression, n</i>)	열, 표현식 또는 값을 소수점 <i>n</i> 자릿수로 반올림합니다. <i>n</i> 이 생략된 경우 소수점 자릿수가 없습니다. (<i>n</i> 이 음수인 경우 소수점 왼쪽의 숫자가 반올림됩니다.)
TRUNC(<i>column expression, n</i>)	열, 표현식 또는 값을 소수점 <i>n</i> 자릿수로 truncate합니다. <i>n</i> 이 생략된 경우 기본값은 0입니다.
MOD(<i>m, n</i>)	<i>m</i> 을 <i>n</i> 으로 나눈 나머지를 반환합니다.

주: 이 리스트에는 제공되는 숫자 함수 중 일부만 포함되어 있습니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 *Numeric Functions* 관련 섹션을 참조하십시오.

ROUND 함수 사용



DUAL은 함수 및 계산의 결과를 볼 때 사용할 수 있는 더미 테이블입니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ROUND 함수 사용

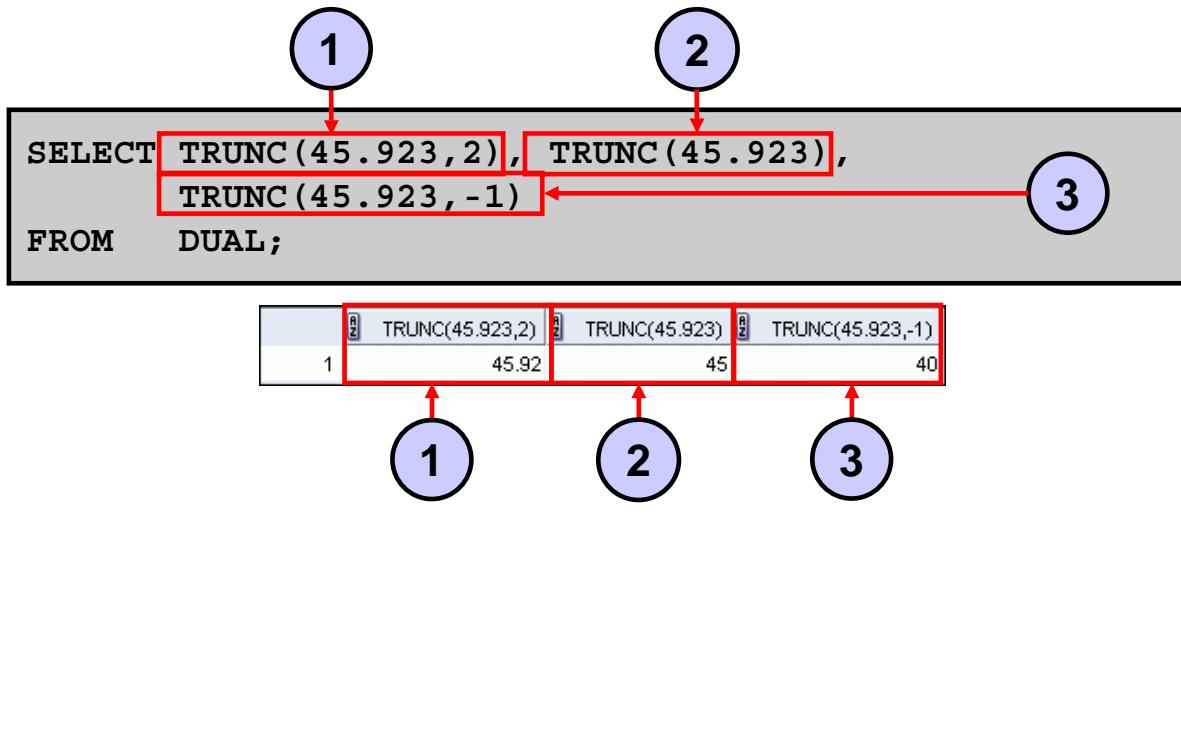
ROUND 함수는 열, 표현식 또는 값을 소수점 n 자릿수로 반올림합니다. 두번째 인수가 0이거나 누락되었으면 값은 소수점 0자릿수로 반올림됩니다. 두번째 인수가 2이면 값은 소수점 2자릿수로 반올림됩니다. 반대로, 두번째 인수가 -2이면 소수점 왼쪽의 둘째 자릿수로 반올림됩니다(가장 가까운 100 단위 값으로 반올림됨).

ROUND 함수는 날짜 함수와도 함께 사용할 수 있습니다. 이 단원의 뒷부분에서 이에 대한 예제를 볼 수 있습니다.

DUAL 테이블

DUAL 테이블은 SYS 유저가 소유하며 모든 유저가 액세스할 수 있습니다. 이 테이블은 DUMMY라는 하나의 열과 값이 X인 하나의 행을 포함합니다. DUAL 테이블은 값을 한 번만 반환하려는 경우에 유용합니다(예: 유저 데이터가 있는 테이블에서 파생되지 않은 상수, 의사 열 또는 표현식 값). SELECT 및 FROM 절이 모두 필수이고 일부 계산의 경우 실제 테이블에서 값을 선택할 필요가 없으므로 DUAL 테이블은 일반적으로 SELECT 절 구문을 완전하게 구현하는 데 사용됩니다.

TRUNC 함수 사용



ORACLE

Copyright © 2007, Oracle. All rights reserved.

TRUNC 함수 사용

TRUNC 함수는 열, 표현식 또는 값을 소수점 n자릿수로 truncate합니다.

TRUNC 함수는 ROUND 함수에 사용된 것과 유사한 인수를 사용합니다. 두번째 인수가 0이거나 누락되었으면 값은 소수점 0자릿수로 truncate됩니다. 두번째 인수가 2이면 값은 소수점 2자릿수로 truncate됩니다. 반대로, 두번째 인수가 -2이면 소수점 왼쪽의 둘째 자릿수로 truncate되고 두번째 인수가 -1이면 소수점 왼쪽의 첫째 자릿수로 truncate됩니다.

ROUND 함수와 마찬가지로 TRUNC 함수도 날짜 함수와 함께 사용할 수 있습니다.

MOD 함수 사용

직책이 **Sales Representative**인 모든 사원에 대해 급여를 **5,000**으로 나눈 나머지를 계산합니다.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM employees
WHERE job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

MOD 함수 사용

MOD 함수는 첫번째 인수를 두번째 인수로 나눈 나머지를 찾습니다. 슬라이드의 예제는 직무 ID 가 SA_REP인 모든 사원에 대해 급여를 5,000으로 나눈 나머지를 계산합니다.

주: MOD 함수는 종종 값이 홀수인지 짝수인지 판별하는 데 사용됩니다.

단원 내용

- 단일 행 **SQL** 함수
- 문자 함수
- 숫자 함수
- 날짜 작업
- 날짜 함수



Copyright © 2007, Oracle. All rights reserved.

날짜 작업

- 오라클 데이터베이스는 내부 숫자 형식(세기, 년, 월, 일, 시, 분, 초)으로 날짜를 저장합니다.
- 기본 날짜 표시 형식은 **DD-MON-RR**입니다.
 - 연도의 마지막 두 자릿수만 지정하면 21세기 날짜를 20세기 날짜로 저장할 수 있습니다.
 - 같은 방식으로 20세기 날짜를 21세기 날짜로 저장할 수 있습니다.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	King	17-JUN-87
2	Whalen	17-SEP-87

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜 작업

오라클 데이터베이스는 세기, 년, 월, 일, 시, 분, 초를 나타내는 내부 숫자 형식으로 날짜를 저장합니다.

기본 날짜 표시 및 입력 형식은 DD-MON-RR입니다. 유효한 Oracle 날짜는 기원전 4712년 1월 1일부터 서기 9999년 12월 31일까지입니다.

슬라이드의 예제에서 HIRE_DATE 열 출력은 기본 형식인 DD-MON-RR로 표시됩니다. 하지만 날짜가 이 형식으로 데이터베이스에 저장되지는 않습니다. 날짜 및 시간의 모든 구성 요소가 저장됩니다. 따라서 07-JUN-87과 같은 HIRE_DATE는 일, 월, 년으로 표시되지만 날짜와 관련되어 저장되는 정보에는 시간 및 세기도 포함됩니다. 완전한 데이터는 June 17, 1987, 5:10:43 PM과 같은 형태입니다.

RR 날짜 형식

현재 연도	지정된 날짜	RR 형식	YY 형식
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		지정된 두 자리 연도가 다음과 같을 경우:	
		0-49	50-99
현재 연도의 두 자리가 다음과 같을 경우:	0-49	반환 날짜는 현재 세기의 날짜입니다.	반환 날짜는 이전 세기의 날짜입니다.
	50-99	반환 날짜는 이후 세기의 날짜입니다.	반환 날짜는 현재 세기의 날짜입니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

RR 날짜 형식

RR 날짜 형식은 YY 요소와 비슷하지만 이 형식을 사용하여 다른 세기를 지정할 수 있습니다. YY 대신 RR 날짜 형식 요소를 사용하면 반환 값의 세기는 지정된 2자리 연도 및 현재 연도의 마지막 2자리에 따라 달라집니다. 슬라이드의 표는 RR 요소의 동작을 요약해서 보여줍니다.

현재 연도	지정된 날짜	해석(RR)	해석(YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

이 날짜는 내부적으로는 다음과 같이 저장됩니다.

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	87	06	17	17	10	43

세기 및 Y2K 문제

날짜 열이 있는 레코드를 테이블에 삽입하면 *century* 정보가 SYSDATE 함수에서 선택됩니다. 하지만 날짜 열이 화면에 표시될 때 century 구성 요소는 기본적으로 표시되지 않습니다. DATE 데이터 유형은 항상 내부적으로 년 정보를 4자리 숫자(세기에 2자리, 연도에 2자리)로 저장합니다. 예를 들어, 오라클 데이터베이스는 년을 87 또는 04가 아니라 1987 또는 2004로 저장합니다.

SYSDATE 함수 사용

SYSDATE 함수는 다음 값을 반환합니다.

- 날짜
- 시간

```
SELECT sysdate  
FROM   dual;
```

	SYSDATE
1	31-MAY-07

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SYSDATE 함수 사용

SYSDATE는 현재의 데이터베이스 서버 날짜 및 시간을 반환하는 날짜 함수입니다. SYSDATE는 다른 열 이름을 사용하듯이 사용할 수 있습니다. 예를 들어, 테이블에서 SYSDATE를 선택하여 현재 날짜를 표시할 수 있습니다. DUAL이라는 더미 테이블에서 SYSDATE를 선택하는 것이 일반적입니다.

주: SYSDATE는 데이터베이스가 상주하는 운영 체제에 설정된 현재 날짜 및 시간을 반환합니다. 따라서 호주에 있는 유저가 미국에 있는 원격 데이터베이스에 연결하면 sysdate 함수는 미국의 날짜 및 시간을 반환합니다. 이 경우 세션 시간대의 현재 날짜를 반환하는 CURRENT_DATE 함수를 사용할 수 있습니다.

CURRENT_DATE 함수 및 관련된 다른 시간대 함수는 *Oracle Database 11g: SQL Fundamentals II* 과정에서 자세히 설명합니다.

날짜 연산

- 날짜에 숫자를 더하거나 빼서 결과 날짜 값을 구합니다.
- 두 날짜 사이의 일 수를 알아내기 위해 빼기 연산을 합니다.
- 시간 수를 **24**로 나눠 날짜에 시간을 더합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜 연산

데이터베이스가 날짜를 숫자로 저장하므로 더하기 및 빼기와 같은 산술 연산자를 사용하여 계산을 수행할 수 있습니다. 날짜뿐 아니라 숫자 상수도 더하고 뺄 수 있습니다.

다음 연산을 수행할 수 있습니다.

연산	결과	설명
날짜 + 숫자	날짜	날짜에 일 수를 더합니다.
날짜 - 숫자	날짜	날짜에서 일 수를 뺍니다.
날짜 - 날짜	일 수	한 날짜를 다른 날짜에서 뺍니다.
날짜 + 숫자/24	날짜	날짜에 시간 수를 더합니다.

날짜에 산술 연산자 사용

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS  
FROM employees  
WHERE department_id = 90;
```

LAST_NAME	WEEKS
King	1041.168239087301587301587301587301587302
Kochhar	923.025381944444444444444444444444444444444444444
De Haan	750.168239087301587301587301587301587302

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜에 산술 연산자 사용

슬라이드의 예제는 부서 90의 모든 사원에 대해 성 및 근속 시간(주 단위)을 표시합니다. 현재 날짜(SYSDATE)에서 사원이 채용된 날짜를 빼고 결과를 7로 나누어 근속 시간을 주 단위로 계산합니다.

주: SYSDATE는 현재 날짜 및 시간을 반환하는 SQL 함수입니다. 결과는 SQL query를 실행할 때 로컬 데이터베이스의 운영 체제에 설정된 날짜 및 시간에 따라 다를 수 있습니다. 오래된 날짜에서 최신 날짜를 빼면 결과는 음수가 됩니다.

단원 내용

- 단일 행 **SQL** 함수
- 문자 함수
- 숫자 함수
- 날짜 작업
- 날짜 함수

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

날짜 조작 함수

함수	결과
MONTHS_BETWEEN	두 날짜 간의 월 수
ADD_MONTHS	날짜에 월 추가
NEXT_DAY	지정된 날짜의 다음 날
LAST_DAY	월의 마지막 날
ROUND	날짜 반올림
TRUNC	날짜 truncate

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜 조작 함수

날짜 함수는 Oracle 날짜에 대해 실행됩니다. 숫자 값을 반환하는 MONTHS_BETWEEN을 제외한 모든 날짜 함수는 DATE 데이터 유형의 값을 반환합니다.

- **MONTHS_BETWEEN(date1, date2)**: date1과 date2 사이의 월 수를 찾습니다. 결과는 양수나 음수가 될 수 있습니다. date1이 date2보다 늦은 날짜인 경우 결과는 양수이고 date1이 date2보다 앞선 날자인 경우 결과는 음수입니다. 결과에서 정수가 아닌 부분은 월의 일부분을 나타냅니다.
- **ADD_MONTHS(date, n)**: date에 월 수 n을 추가합니다. n 값은 정수여야 하며 음수가 될 수도 있습니다.
- **NEXT_DAY(date, 'char')**: date 다음에 오는 지정된 요일 ('char')의 날짜를 찾습니다. char 값은 요일을 나타내는 숫자나 문자열이 될 수 있습니다.
- **LAST_DAY(date)**: date에 해당하는 날짜가 있는 월의 말일 날짜를 찾습니다.

날짜 조작 함수(계속)

위 리스트에는 제공되는 날짜 함수의 일부만 나와 있습니다. ROUND 및 TRUNC 숫자 함수는 아래에 나오는 것처럼 날짜 조작에도 사용할 수 있습니다.

- ROUND(*date*[, '*fmt*']): *date*를 형식 모델 *fmt*에서 지정한 단위로 반올림하여 반환합니다. 형식 모델 *fmt*가 생략된 경우 *date*는 가장 가까운 일로 반올림됩니다.
- TRUNC(*date*[, '*fmt*']): *date*를 형식 모델 *fmt*에서 지정한 단위로 truncate하여 날짜의 시간 부분과 함께 반환합니다. 형식 모델 *fmt*가 생략된 경우 *date*는 가장 가까운 일로 truncate됩니다.

형식 모델은 다음 단원 "변환 함수 및 조건부 표현식 사용"에서 자세히 다룹니다.

날짜 함수 사용

함수	결과
<code>MONTHS_BETWEEN ('01-SEP-95', '11-JAN-94')</code>	<code>19.6774194</code>
<code>ADD_MONTHS ('31-JAN-96', 1)</code>	<code>'29-FEB-96'</code>
<code>NEXT_DAY ('01-SEP-95', 'FRIDAY')</code>	<code>'08-SEP-95'</code>
<code>LAST_DAY ('01-FEB-95')</code>	<code>'28-FEB-95'</code>

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜 함수 사용

슬라이드 예제에서 ADD_MONTHS 함수는 제공된 날짜 값 "31-JAN-96"에 한 달을 더하여 "29-FEB-96"을 반환합니다. 이 함수는 1996년을 윤년으로 인식하여 2월의 마지막 날짜를 반환합니다. 입력 날짜 값을 "31-JAN-95"로 변경하면 이 함수는 "28-FEB-95"를 반환합니다. 예를 들어, 근속 기간이 100개월 미만인 모든 사원에 대해 사원 번호, 채용 날짜, 근속 월 수, 6개월 평가 날짜, 채용 날짜 이후의 첫번째 금요일, 채용된 월의 말일을 표시합니다.

```
SELECT employee_id, hire_date,
       MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
       ADD_MONTHS (hire_date, 6) REVIEW,
       NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
  FROM   employees
 WHERE  MONTHS_BETWEEN (SYSDATE, hire_date) < 100;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(HIRE_DATE,'FRIDAY')	LAST_DAY(HIRE_DATE)
1	124	16-NOV-99	91.1099600...	16-MAY-00	19-NOV-99	30-NOV-99
2	149	29-JAN-00	88.6906052...	29-JUL-00	04-FEB-00	31-JAN-00
3	178	24-MAY-99	96.8518955...	24-NOV-99	28-MAY-99	31-MAY-99
4	99999	07-JUN-99	96.4002826...	07-DEC-99	11-JUN-99	30-JUN-99
5	113	11-JUN-07	0.25824335...	11-DEC-07	15-JUN-07	30-JUN-07

날짜에 ROUND 및 TRUNC 함수 사용

SYSDATE = '25-JUL-03'이라고 가정합니다.

함수	결과
ROUND(SYSDATE, 'MONTH')	01-AUG-03
ROUND(SYSDATE, 'YEAR')	01-JAN-04
TRUNC(SYSDATE, 'MONTH')	01-JUL-03
TRUNC(SYSDATE, 'YEAR')	01-JAN-03

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜에 ROUND 및 TRUNC 함수 사용

ROUND 및 TRUNC 함수는 숫자 및 날짜 값에 사용할 수 있습니다. 날짜와 함께 사용할 경우 이러한 함수는 지정된 형식 모델로 반올림하거나 truncate합니다. 따라서 가장 가까운 년 또는 월로 날짜를 반올림할 수 있습니다. 형식 모델이 월인 경우 날짜 1-15는 현재 월의 시작일이 됩니다. 날짜 16-31은 다음 월의 시작일이 됩니다. 형식 모델이 년인 경우 월 1-6은 현재 연도의 1월 1일이 됩니다. 월 7-12는 다음 연도의 1월 1일이 됩니다.

예제:

1997년에 입사한 모든 사원에 대해 채용 날짜를 비교합니다. 사원 번호를 표시하고 ROUND 및 TRUNC 함수를 사용하여 채용 날짜 및 시작 월을 표시합니다.

```
SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
  FROM   employees
 WHERE  hire_date LIKE '%97';
```

	EMPLOYEE_ID	HIRE_DATE	ROUND(HIRE_DATE,'MONTH')	TRUNC(HIRE_DATE,'MONTH')
1	142	29-JAN-97	01-FEB-97	01-JAN-97
2	202	17-AUG-97	01-SEP-97	01-AUG-97

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 함수를 사용하여 데이터에 대해 계산 수행
- 함수를 사용하여 개별 데이터 항목 설정



Copyright © 2007, Oracle. All rights reserved.

요약

단일 행 함수는 어떠한 레벨로도 중첩될 수 있습니다. 단일 행 함수는 다음을 조작할 수 있습니다.

- 문자 데이터: LOWER, UPPER, INITCAP, CONCAT, SUBSTR, INSTR, LENGTH
- 숫자 데이터: ROUND, TRUNC, MOD
- 날짜 값: SYSDATE, MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY

다음 사항을 기억하십시오.

- 날짜 값에도 산술 연산자를 사용할 수 있습니다.
- ROUND 및 TRUNC 함수는 날짜 값에도 사용할 수 있습니다.

SYSDATE 및 DUAL

SYSDATE는 현재 날짜 및 시간을 반환하는 날짜 함수입니다. DUAL이라는 더미 테이블에서 SYSDATE를 선택하는 것이 일반적입니다.

연습 3: 개요

이 연습에서는 다음 내용을 다룹니다.

- 현재 날짜를 표시하는 **query** 작성
- 숫자, 문자 및 날짜 함수를 사용해야 하는 **query** 작성
- 사원의 근무 연 수 및 월 수 계산



Copyright © 2007, Oracle. All rights reserved.

연습 3: 개요

이 연습에서는 문자, 숫자 및 날짜 데이터 유형에 사용할 수 있는 다양한 함수의 사용법을 익힙니다.

연습 3

1부

1. 시스템 날짜를 표시하기 위한 query를 작성합니다. 열 레이블을 Date로 지정합니다.

주: 시간대가 다른 지역에 데이터베이스가 있는 경우 해당 데이터베이스가 상주하는 운영 체제의 날짜가 출력됩니다.

	Date
1	31-MAY-07

2. HR 부서에서 각 사원에 대해 사원 번호, 성, 급여 및 15.5% 인상된 급여(정수로 표현)를 표시하는 보고서가 필요합니다. 열 레이블을 New Salary로 지정합니다. 작성한 SQL 문을 lab_03_02.sql이라는 파일에 저장합니다.
3. lab_03_02.sql 파일의 query를 실행합니다.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
1	100 King		24000	27720
2	101 Kochhar		17000	19635
3	102 De Haan		17000	19635
4	103 Hunold		9000	10395
5	104 Ernst		6000	6930
6	107 Lorentz		4200	4851
7	124 Mourgos		5800	6699
8	141 Rajs		3500	4043
9	142 Davies		3100	3581
10	143 Matos		2600	3003
...				
19	205 Higgins		12000	13860
20	206 Gietz		8300	9587

4. 새 급여에서 이전 급여를 뺀 값을 추가하도록 lab_03_02.sql의 query를 수정합니다. 열 레이블을 Increase로 지정합니다. 파일 내용을 lab_03_04.sql로 저장합니다. 수정한 query를 실행합니다.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
1	100 King		24000	27720	3720
2	101 Kochhar		17000	19635	2635
3	102 De Haan		17000	19635	2635
4	103 Hunold		9000	10395	1395
5	104 Ernst		6000	6930	930
...					
20	206 Gietz		8300	9587	1287

연습 3(계속)

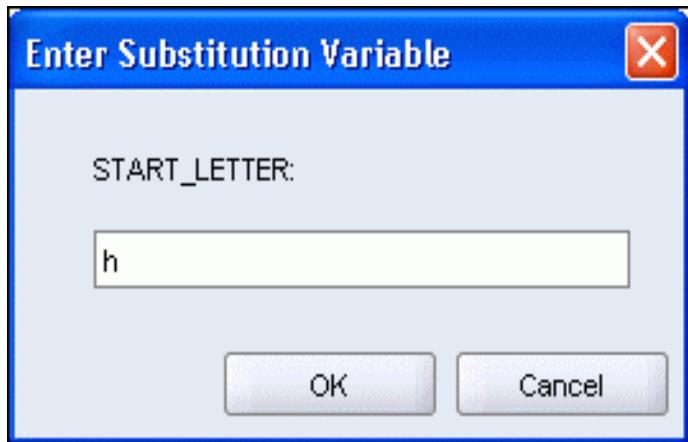
5. "J", "A" 또는 "M"으로 시작하는 이름을 가진 모든 사원의 성(첫번째 문자는 대문자, 나머지는 모두 소문자)과 성의 길이를 표시하는 query를 작성합니다. 각 열에 적절한 레이블을 지정합니다. 사원의 성을 기준으로 결과를 정렬합니다.

	Name	Length
1	Abel	4
2	Matos	5
3	Mourgos	7

유저에게 성의 첫 문자를 입력하는 프롬프트를 표시하도록 query를 재작성합니다. 예를 들어, 문자 입력 프롬프트가 표시되었을 때 유저가 "H"(대문자)를 입력하면 성이 "H"로 시작하는 모든 사원이 출력에 표시되어야 합니다.

	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

입력된 문자의 대소문자 여부에 따라 출력이 달라지지 않도록 쿼리를 수정합니다. 입력된 문자는 SELECT query에서 처리되기 전에 대문자로 변경해야 합니다.



	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

연습 3(계속)

6. HR 부서에서 각 사원의 근속 기간을 파악하려고 합니다. 각 사원에 대해 성을 표시하고 채용일부터 오늘까지 경과한 개월 수를 계산합니다. 열 레이블을 MONTHS_WORKED로 지정합니다. 재직 개월 수에 따라 결과를 정렬합니다. 개월 수를 가장 가까운 정수로 반올림합니다.

주: 이 query는 실행된 날짜에 의존하므로 MONTHS_WORKED 열의 값은 실행된 날짜에 따라 달라집니다.

	LAST_NAME	MONTHS_WORKED
1	Zlotkey	88
2	Mourgos	90
3	Grant	96
4	Lorentz	100
5	Vargas	107
6	Taylor	110
7	Matos	111
8	Fay	117
9	Davies	124
10	Abel	133
11	Hartstein	135
12	Rajs	139
13	Higgins	156
14	Gietz	156
15	De Haan	173
16	Ernst	192
17	Hunold	209
18	Kochhar	212
19	Whalen	236
20	King	239

연습 3(계속)

시간 여유가 있을 경우 다음 연습을 완료하십시오.

- 모든 사원의 성과 급여를 표시하기 위한 query를 작성합니다. 급여가 15자 길이로 표시되고 왼쪽에 \$ 기호가 채워지도록 형식을 지정합니다. 열 레이블을 SALARY로 지정합니다.

	LAST_NAME	SALARY
1	King	\$\$\$\$\$\$\$\$\$\$24000
2	Kochhar	\$\$\$\$\$\$\$\$\$\$17000

...

20	Gietz	\$\$\$\$\$\$\$\$\$\$8300
----	-------	--------------------------

- 사원의 성에서 처음 8자를 표시하고 급여 액수를 별표로 나타내는 query를 작성합니다. 각 별표는 \$1,000를 나타냅니다. 급여의 내림차순으로 데이터를 정렬합니다. 열 레이블을 EMPLOYEES_AND_THEIR_SALARIES로 지정합니다.

	EMPLOYEES_AND_THEIR_SALARIES
1	King *****
2	Kochhar *****
3	De Haan *****
4	Hartstei *****
5	Higgins *****

...

19	Matos	**
20	Vargas	**

- 부서 90의 모든 사원에 대해 성 및 재직 기간(주 단위)을 표시하도록 query를 작성합니다. 주를 나타내는 숫자 열의 레이블로 TENURE를 지정합니다. 주를 나타내는 숫자 값을 소수점 왼쪽에서 truncate합니다. 직원 재직 기간의 내림차순으로 레코드를 표시합니다. 주: TENURE 값은 query를 실행한 날짜에 의존하므로 실행한 날짜에 따라 달라집니다.

	LAST_NAME	TENURE
1	King	1041
2	Kochhar	923
3	De Haan	750

변환 함수 및 조건부 표현식 사용

4

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **SQL**에서 사용할 수 있는 다양한 유형의 변환 함수 설명
- **TO_CHAR**, **TO_NUMBER** 및 **TO_DATE** 변환 함수 사용
- **SELECT** 문에 조건부 표현식 적용



Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 한 유형에서 다른 유형으로 데이터를 변환하는(예: 문자 데이터에서 숫자 데이터로 변환) 함수를 중점적으로 다루고 SQL SELECT 문의 조건부 표현식에 대해 설명합니다.

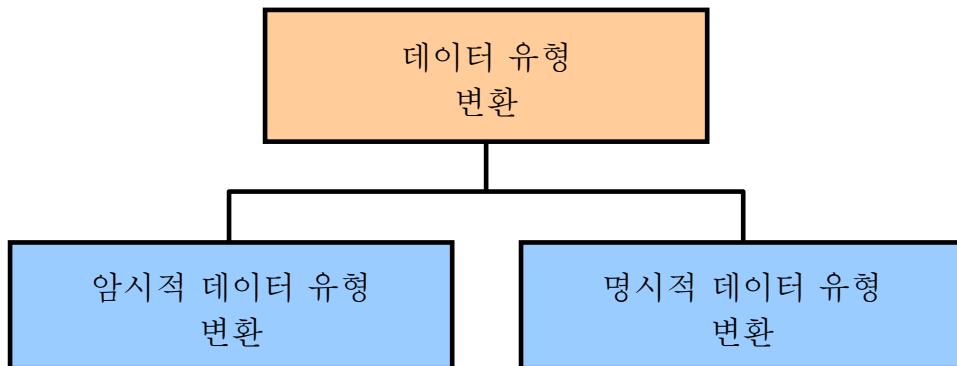
단원 내용

- 암시적 및 명시적 데이터 유형 변환
- **TO_CHAR, TO_DATE, TO_NUMBER** 함수
- 함수 중첩
- 일반 함수:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- 조건부 표현식:
 - CASE
 - DECODE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

변환 함수



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

변환 함수

Oracle 데이터베이스에서 테이블의 열은 Oracle 데이터 유형 이외에 ANSI (American National Standards Institute), DB2 및 SQL/DS 데이터 유형을 사용하여 정의할 수도 있습니다. 하지만 Oracle 서버는 내부적으로 이러한 데이터 유형을 Oracle 데이터 유형으로 변환합니다.

어떤 경우에는 Oracle 서버가 특정 데이터 유형의 데이터를 사용하는 위치에 다른 데이터 유형의 데이터가 전달되기도 합니다. 이 경우 Oracle 서버가 데이터를 예상 데이터 유형으로 자동으로 변환할 수 있습니다. 이러한 데이터 유형 변환은 Oracle 서버에 의해 암시적으로 수행되거나 유저에 의해 명시적으로 수행될 수 있습니다.

암시적 데이터 유형 변환은 다음 두 슬라이드에 설명된 규칙에 따라 수행됩니다.

명시적 데이터 유형 변환은 변환 함수를 사용하여 수행됩니다. 변환 함수는 값의 데이터 유형을 변환합니다. 일반적으로 함수 이름의 형식은 `data type TO data type` 규칙을 따릅니다. 첫번째 데이터 유형은 입력 데이터 유형이고 두번째 데이터 유형은 출력 데이터 유형입니다.

주: 암시적 데이터 유형 변환을 사용할 수 있어도 SQL 문의 신뢰성을 높이기 위해 명시적 데이터 유형 변환을 수행할 것을 권장합니다.

암시적 데이터 유형 변환

Oracle 서버는 표현식에서 다음을 자동으로 변환할 수 있습니다.

소스	대상
VARCHAR2 또는 CHAR	NUMBER
VARCHAR2 또는 CHAR	DATE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

암시적 데이터 유형 변환

Oracle 서버는 자동으로 표현식에서 데이터 유형 변환을 수행합니다. 예를 들어, 표현식 `hire_date > '01-JAN-90'`에서 문자열 '01-JAN-90'은 암시적으로 날짜로 변환됩니다. 따라서 표현식의 VARCHAR2 또는 CHAR 값은 암시적으로 숫자 또는 날짜 데이터 유형으로 변환될 수 있습니다.

암시적 데이터 유형 변환

표현식 평가 시 **Oracle** 서버는 다음을 자동으로 변환할 수 있습니다.

소스	대상
NUMBER	VARCHAR2 또는 CHAR
DATE	VARCHAR2 또는 CHAR

ORACLE

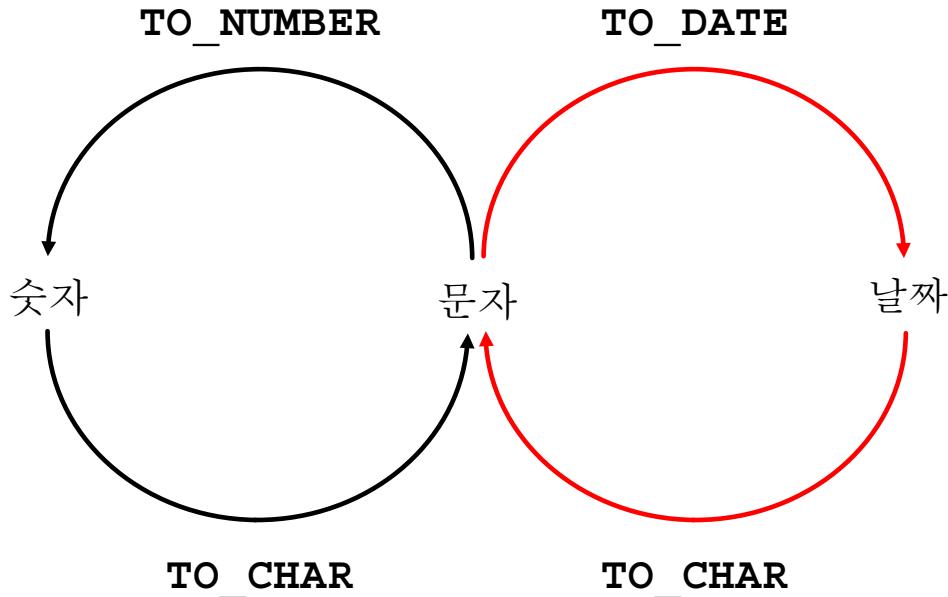
Copyright © 2007, Oracle. All rights reserved.

암시적 데이터 유형 변환(계속)

일반적으로 Oracle 서버에서는 데이터 유형 변환이 필요한 경우 표현식에 대한 규칙을 사용합니다. 예를 들어, grade가 CHAR (2) 열인 경우 표현식 grade = 2에서 숫자 20000이 문자열 "2"로 암시적으로 변환됩니다.

주: 문자열이 유효한 숫자를 나타내는 경우에만 CHAR를 NUMBER로 변환할 수 있습니다.

명시적 데이터 유형 변환



ORACLE

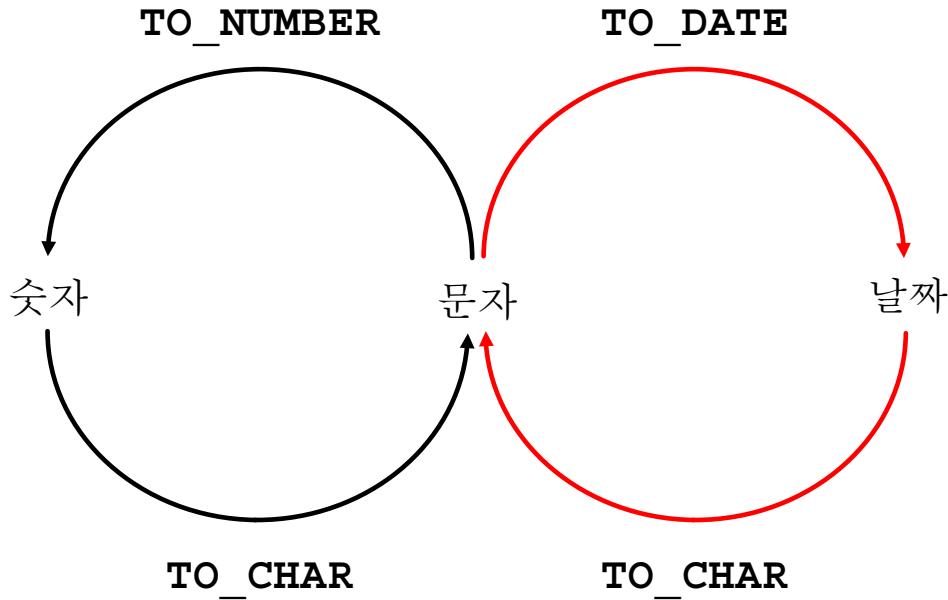
Copyright © 2007, Oracle. All rights reserved.

명시적 데이터 유형 변환

SQL은 세 가지 데이터 유형 변환 함수를 제공합니다.

함수	목적
<code>TO_CHAR(number date, [fmt], [nlsparams])</code>	<p>형식 모델 <i>fmt</i>를 사용하여 숫자 또는 날짜 값을 VARCHAR2 문자열로 변환합니다.</p> <p>숫자 변환: <i>nlsparams</i> 파라미터는 다음 문자를 지정하며, 이 문자는 숫자 형식 요소로 반환됩니다.</p> <ul style="list-style-type: none"> • 소수점 문자 • 그룹 구분자 • 로컬 통화 기호 • 국제 통화 기호 <p><i>nlsparams</i> 또는 기타 파라미터가 생략된 경우 이 함수는 세션에 대해 기본 파라미터 값을 사용합니다.</p>

명시적 데이터 유형 변환



ORACLE

Copyright © 2007, Oracle. All rights reserved.

명시적 데이터 유형 변환(계속)

함수	목적
<code>TO_CHAR(number date, [fmt], [nlsparams])</code>	날짜 변환: <code>nlsparams</code> 파라미터는 월 및 일 이름과 약어가 반환되는 언어를 지정합니다. 이 파라미터가 생략된 경우 이 함수는 세션에 대해 기본 날짜 언어를 사용합니다.
<code>TO_NUMBER(char, [fmt], [nlsparams])</code>	숫자를 포함한 문자열을 선택적 형식 모델 <code>fmt</code> 로 지정된 형식의 숫자로 변환합니다. 이 함수에서 <code>nlsparams</code> 파라미터는 숫자 변환용 <code>TO_CHAR</code> 함수에서와 동일한 용도로 사용됩니다.
<code>TO_DATE(char, [fmt], [nlsparams])</code>	날짜를 나타내는 문자열을 지정된 <code>fmt</code> 에 따라 날짜 값으로 변환합니다. <code>fmt</code> 가 생략된 경우 형식은 DD-MON-YY입니다. 이 함수에서 <code>nlsparams</code> 파라미터는 날짜 변환용 <code>TO_CHAR</code> 함수에서와 동일한 용도로 사용됩니다.

명시적 데이터 유형 변환(계속)

주: 이 단원에서 언급된 함수 리스트에는 제공되는 변환 함수 중 일부만 포함되어 있습니다.
자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 *Conversion Functions* 관련 섹션을 참조하십시오.

단원 내용

- 암시적 및 명시적 데이터 유형 변환
- **TO_CHAR, TO_DATE, TO_NUMBER** 함수
- 함수 중첩
- 일반 함수:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- 조건부 표현식:
 - CASE
 - DECODE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜에 TO_CHAR 함수 사용

```
TO_CHAR(date, 'format_model')
```

형식 모델:

- 작은 따옴표로 묶어야 합니다.
- 대소문자를 구분합니다.
- 임의의 유효한 날짜 형식 요소를 포함할 수 있습니다.
- 채워진 공백을 제거하거나 선행 0을 출력하지 않는 **fm** 요소를 갖습니다.
- 쉼표로 날짜 값과 구분됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜에 TO_CHAR 함수 사용

TO_CHAR는 *format_model*에 지정된 형식으로 datetime 데이터 유형의 값을 VARCHAR2 데이터 유형의 값으로 변환합니다. 형식 모델은 문자열에 저장된 datetime의 형식을 설명하는 문자 리터럴입니다. 예를 들어, 문자열 '11-Nov-1999'의 datetime 형식 모델은 'DD-Mon-YYYY'입니다. TO_CHAR 함수를 사용하여 기본 형식의 날짜를 유저가 지정하는 형식으로 변환할 수 있습니다.

지침

- 형식 모델은 작은 따옴표로 묶어야 하며 대소문자를 구분합니다.
- 형식 모델은 임의의 유효한 날짜 형식 요소를 포함할 수 있습니다. 그러나 쉼표를 사용하여 날짜 값을 형식 모델과 구분해야 합니다.
- 일 및 월 이름은 자동으로 공백이 채워져 출력됩니다.
- 채워진 공백을 제거하거나 선행 0을 출력하지 않으려면 채우기 모드 **fm** 요소를 사용합니다.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
  FROM employees
 WHERE last_name = 'Higgins';
```

EMPLOYEE_ID	MONTH_HIRED
1	205 06/94

날짜 형식 모델의 요소

요소	결과
YYYY	숫자로 된 전체 연도
YEAR	영어 철자로 표기된 연도
MM	월의 2자리 값
MONTH	전체 월 이름
MON	월의 3자 약어
DY	3 문자로 된 요일 약어
DAY	요일의 전체 이름
DD	숫자 형식의 월간 일

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

유효한 날짜 형식을 보여주는 예제 형식 요소

요소	설명
SCC 또는 CC	세기. 기원전 날짜 앞에는 자동으로 -가 붙습니다.
YYYY 또는 SYYYY 날짜 형식의 연도	연도. 기원전 날짜 앞에는 자동으로 -가 붙습니다.
YY 또는 YY 또는 Y	연도의 마지막 3자리, 2자리 또는 1자리
Y,YYY	이 위치에 쉼표가 있는 연도
IYYY, IYY, IY, I	ISO 표준을 따르는 4자리, 3자리, 2자리 또는 1자리 연도
SYEAR 또는 YEAR	영어 철자로 표기된 연도. 기원전 날짜 앞에는 자동으로 -가 붙습니다.
BC 또는 AD	기원전 또는 서기 연도를 나타냄
B.C. 또는 A.D.	마침표를 사용하여 기원전 또는 서기 연도를 나타냄
Q	분기
MM	월. 2자리 값
MONTH	월 이름. 9자까지 공백으로 채워짐
MON	월 이름. 3자 약어
RM	로마식 월 숫자
WW 또는 W	연 또는 월의 주
DDD 또는 DD 또는 D	연, 월 또는 주의 일
DAY	일 이름. 9자까지 공백으로 채워짐
DY	일 이름. 3자 약어
J	율리우스 일. 기원전 4713년 12월 31일 이후의 일 수
IW	ISO 표준에 따른 연의 주(1-53)

날짜 형식 모델의 요소

- 시간 요소는 날짜에서 시간 부분의 형식을 지정합니다.

HH24 : MI : SS AM	15 : 45 : 32 PM
-------------------	-----------------

- 문자열은 큰 따옴표로 묶어 추가합니다.

DD "of" MONTH	12 of OCTOBER
---------------	---------------

- 숫자 접미어는 숫자를 영어 철자로 표기합니다.

ddspth	fourteenth
--------	------------

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

날짜 형식 모델의 요소

다음 표에 나열된 형식을 사용하여 시간 정보 및 리터럴을 표시하고 숫자를 영어 철자로 된 숫자로 변경합니다.

요소	설명
AM 또는 PM	자오선 표시
A.M. 또는 P.M.	마침표가 있는 자오선 표시
HH 또는 HH12 또는 HH24	하루 시간 또는 반일 시간(1-12) 또는 전일 시간(0-23)
MI	분(0-59)
SS	초(0-59)
SSSSS	자정 이후의 초(0-86399)

기타 형식

요소	설명
/ . ,	결과에 구두점 생성
"of the"	결과에 따옴표로 묶은 문자열 생성

접미어를 지정하여 숫자 표시 조정

요소	설명
TH	서수(예: 4TH를 DDTH로 표기)
SP	영어 철자로 된 숫자(예: FOUR를 DDSP로 표기)
SPTH 또는 THSP	영어 철자로 된 서수(예: FOURTH를 DDSPTH로 표기)

날짜에 TO_CHAR 함수 사용

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
  AS HIREDATE
FROM employees;
```

LAST_NAME	HIREDATE
1 King	17 June 1987
2 Kochhar	21 September 1989
3 De Haan	13 January 1993
4 Hunold	3 January 1990
5 Ernst	21 May 1991
6 Lorentz	7 February 1999
7 Mourgos	16 November 1999
8 Rajs	17 October 1995
9 Davies	29 January 1997
10 Matos	15 March 1998
...	
19 Higgins	7 June 1994
20 Gietz	7 June 1994

ORACLE

Copyright © 2007, Oracle. All rights reserved.

날짜에 TO_CHAR 함수 사용

슬라이드의 SQL 문은 모든 사원의 성과 채용 날짜를 표시합니다. 채용 날짜는 17 June 1987 형식으로 나타냅니다.

예제:

슬라이드의 예제를 수정하여 날짜를 "Seventeenth of June 1987 12:00:00 AM" 형식으로 나타냅니다.

```
SELECT last_name,
       TO_CHAR(hire_date,
              'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
  AS HIREDATE
FROM employees;
```

LAST_NAME	HIREDATE
1 King	Seventeenth of June 1987 12:00:00 AM
2 Kochhar	Twenty-First of September 1989 12:00:00 AM

월은 지정된 형식 모델을 따릅니다. 즉, 첫번째 문자는 대문자이고 나머지는 소문자입니다.

숫자에 **TO_CHAR** 함수 사용

TO_CHAR (number, 'format_model')

다음은 **TO_CHAR** 함수와 함께 사용하여 숫자 값을 문자로 표시할 수 있는 몇 가지 형식 요소입니다.

요소	결과
9	숫자를 나타냄
0	0이 표시되도록 강제 적용
\$	부동 달러 기호 배치
L	부동 로컬 통화 기호 사용
.	소수점 출력
,	천단위 표시자로 첨표 출력

ORACLE

Copyright © 2007, Oracle. All rights reserved.

숫자에 **TO_CHAR** 함수 사용

숫자 값을 문자열로 취급하여 사용하는 경우 **TO_CHAR** 함수를 사용하여 이러한 숫자를 문자 데이터 유형으로 변환해야 합니다. 이 함수는 NUMBER 데이터 유형 값을 VARCHAR2 데이터 유형으로 변환합니다. 이 방법은 연결 연산에서 특히 유용합니다.

숫자에 **TO_CHAR** 함수 사용(계속)

숫자 형식 요소

숫자를 문자 데이터 유형으로 변환하는 경우 다음 형식 요소를 사용할 수 있습니다.

요소	설명	예제	결과
9	숫자 위치(9의 개수로 표시 너비 결정)	999999	1234
0	선행 0을 표시	099999	001234
\$	부동 달러 기호	\$999999	\$1234
L	부동 로컬 통화 기호	L999999	FF1234
D	지정된 위치에서 소수점 문자 반환. 기본값은 마침표(.)입니다.	99D99	99.99
.	지정된 위치의 소수점	999999.99	1234.00
G	지정된 위치에서 그룹 구분자 반환. 숫자 형식 모델에서 여러 그룹 구분자를 지정할 수 있습니다.	9,999	9G999
,	지정된 위치의 쉼표	999,999	1,234
MI	오른쪽 빼기 기호(음수 값)	999999MI	1234-
PR	음수 팔호로 묶기	999999PR	<1234>
EEEE	과학적 표기법(네 개의 E를 형식으로 지정)	99.999EEEE	1.234E+03
U	지정된 위치에서 "유로"(또는 다른) 이중 통화 반환	U9999	€1234
V	10을 n 배 곱하기($n = V$ 다음에 오는 9의 개수)	9999V99	123400
S	음수 또는 양수 값 반환	S9999	-1234 또는 +1234
B	0 값을 0이 아닌 공백으로 표시	B9999.99	1234.00

숫자에 **TO_CHAR** 함수 사용

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY  
FROM   employees  
WHERE  last_name = 'Ernst';
```

	SALARY
1	\$6,000.00

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

숫자에 **TO_CHAR** 함수 사용(계속)

- Oracle 서버는 정수의 자릿수가 형식 모델에 제공된 자릿수를 초과하는 경우 해당 위치에 숫자 대신 일련의 숫자 기호(#) 문자열을 표시합니다.
- Oracle 서버는 저장된 십진값을 형식 모델에서 제공하는 십진값의 자릿수로 반올림합니다.

TO_NUMBER 및 TO_DATE 함수 사용

- **TO_NUMBER** 함수를 사용하여 문자열을 숫자 형식으로 변환합니다.

```
TO_NUMBER(char[, 'format_model'])
```

- **TO_DATE** 함수를 사용하여 문자열을 날짜 형식으로 변환합니다.
- 이러한 함수는 **fx** 수정자를 가집니다. 이 수정자는 **TO_DATE** 함수의 문자 인수 및 날짜 형식 모델에 대한 정확한 일치를 지정합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

TO_NUMBER 및 TO_DATE 함수 사용

문자열을 숫자 또는 날짜로 변환할 수 있습니다. 이 작업을 수행하려면 **TO_NUMBER** 또는 **TO_DATE** 함수를 사용합니다. 형식 모델은 앞에서 예시한 형식 요소를 기반으로 선택해야 합니다.

fx 수정자는 **TO_DATE** 함수의 문자 인수 및 날짜 형식 모델에 대한 정확한 일치를 지정합니다.

- 문자 인수에서 구두점과 따옴표로 묶인 텍스트는 형식 모델의 해당 부분과 정확히 일치해야 합니다(대소문자 제외).
- 문자 인수는 추가 공백을 가질 수 없습니다. **fx**가 없는 경우 Oracle 서버는 추가 공백을 무시합니다.
- 문자 인수의 숫자 데이터는 형식 모델의 해당 요소와 동일한 자릿수를 가져야 합니다. **fx**가 없는 경우 문자 인수의 숫자에서 선형 0을 생략할 수 있습니다.

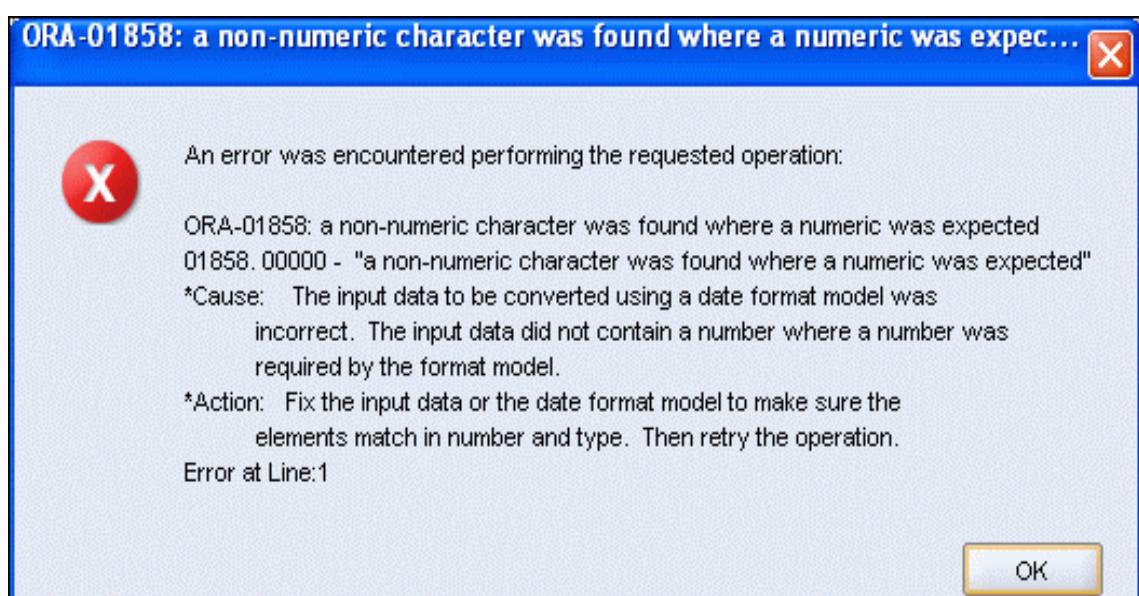
TO_NUMBER 및 TO_DATE 함수 사용(계속)

예제:

1999년 5월 24일부터 근무하기 시작한 모든 사원의 이름 및 채용 날짜를 표시합니다. 다음 예제에서는 월 *May*와 숫자 24 사이에 세 개의 공백이 있습니다. *fx* 수정자를 사용하므로 문자 인수와 날짜 형식 모델이 정확히 일치해야 하며 단어 *May* 뒤의 공백은 인식되지 않습니다.

```
SELECT last_name, hire_date  
FROM   employees  
WHERE  hire_date = TO_DATE('May      24, 1999', 'fxMonth DD, YYYY');
```

오류:



RR 날짜 형식으로 TO_CHAR 및 TO_DATE 함수 사용

1990년 이전에 채용된 사원을 찾으려면 RR 날짜 형식을 사용합니다.
그러면 명령이 1999년에 실행되는 현재 실행되는 동일한 결과를
나타냅니다.

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

LAST_NAME	TO_CHAR(HIRE_DATE,'DD-MON-YYYY')
King	17-Jun-1987
Kochhar	21-Sep-1989
Whalen	17-Sep-1987

Copyright © 2007, Oracle. All rights reserved.

RR 날짜 형식으로 TO_CHAR 및 TO_DATE 함수 사용

1990년 이전에 채용된 사원을 찾을 경우 RR 형식을 사용할 수 있습니다. 현재 연도는 1999보다
크기 때문에 RR 형식은 1950부터 1999 사이의 날짜에서 연도 부분을 해석합니다.

그러나 다음 명령은 YY 형식이 현재 세기(2090)의 날짜에서 연도 부분을 해석하기 때문에 행이
선택되지 않습니다.

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM employees
WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

0 rows selected

단원 내용

- 암시적 및 명시적 데이터 유형 변환
- **TO_CHAR, TO_DATE, TO_NUMBER** 함수
- 함수 중첩
- 일반 함수:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- 조건부 표현식:
 - CASE
 - DECODE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

함수 중첩

- 단일 행 함수는 어떠한 레벨로도 중첩될 수 있습니다.
- 중첩된 함수는 가장 깊은 레벨에서 가장 낮은 레벨로 평가됩니다.

F3 (F2 (F1 (col,arg1) , arg2) , arg3)

Step 1 = Result 1

Step 2 = Result 2

Step 3 = Result 3

ORACLE

Copyright © 2007, Oracle. All rights reserved.

함수 중첩

단일 행 함수는 어떠한 깊이로도 중첩될 수 있습니다. 중첩된 함수는 가장 안쪽 레벨에서 가장 바깥쪽 레벨로 평가됩니다. 이러한 함수의 융통성을 보여주기 위한 몇 가지 예제가 이어질 것입니다.

함수 중첩

```
SELECT last_name,  
       UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))  
  FROM employees  
 WHERE department_id = 60;
```

LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1 Hunold	HUNOLD_US
2 Ernst	ERNST_US
3 Lorentz	LORENTZ_US

ORACLE

Copyright © 2007, Oracle. All rights reserved.

함수 중첩(계속)

슬라이드의 예제는 부서 60에 있는 사원의 성을 표시합니다. SQL 문의 평가에는 다음 세 단계가 포함됩니다.

1. 안쪽 함수가 성의 앞부분 8자를 검색합니다.

```
Result1 = SUBSTR (LAST_NAME, 1, 8)
```

2. 바깥쪽 함수가 결과를 _US와 연결합니다.

```
Result2 = CONCAT(Result1, '_US')
```

3. 가장 바깥쪽 함수가 결과를 대문자로 변환합니다.

열 alias가 제공되지 않았으므로 전체 표현식이 열 머리글로 됩니다.

예제:

채용 날짜로부터 6개월이 경과한 날 다음에 오는 금요일의 날짜를 표시합니다. 결과 날짜는 1999년 8월 13일, 금요일로 나타납니다. 결과를 채용 날짜별로 정렬합니다.

```
SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS  
          (hire_date, 6), 'FRIDAY'),  
          'fmDay, Month ddth, YYYY')  
          "Next 6 Month Review"  
  FROM employees  
 ORDER BY hire_date;
```

단원 내용

- 암시적 및 명시적 데이터 유형 변환
- **TO_CHAR, TO_DATE, TO_NUMBER** 함수
- 함수 중첩
- 일반 함수:
 - **NVL**
 - **NVL2**
 - **NULLIF**
 - **COALESCE**
- 조건부 표현식:
 - **CASE**
 - **DECODE**

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

일반 함수

다음 함수는 임의의 데이터 유형을 사용하며 **null** 사용과 관련이 있습니다.

- **NVL (expr1, expr2)**
- **NVL2 (expr1, expr2, expr3)**
- **NULLIF (expr1, expr2)**
- **COALESCE (expr1, expr2, . . . , exprn)**

Copyright © 2007, Oracle. All rights reserved.

일반 함수

다음 함수는 임의의 데이터 유형을 사용하며 표현식 리스트에서 null 값의 사용과 관련이 있습니다.

함수	설명
NVL	null 값을 실제 값으로 변환합니다.
NVL2	expr1이 null이 아닌 경우 NVL2는 expr2를 반환합니다. expr1이 null인 경우 NVL2는 expr3을 반환합니다. 인수 expr1은 임의의 데이터 유형을 가질 수 있습니다.
NULLIF	두 표현식을 비교하여 같으면 null을 반환하고 같지 않으면 첫번째 표현식을 반환합니다.
COALESCE	표현식 리스트에서 null이 아닌 첫번째 표현식을 반환합니다.

주: 제공되는 수백 가지 함수에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 *Functions* 관련 섹션을 참조하십시오.

NVL 함수

null 값을 실제 값으로 변환합니다.

- 사용할 수 있는 데이터 유형은 날짜, 문자 및 숫자입니다.
- 데이터 유형이 일치해야 합니다.
 - `NVL(commission_pct, 0)`
 - `NVL(hire_date, '01-JAN-97')`
 - `NVL(job_id, 'No Job Yet')`

ORACLE

Copyright © 2007, Oracle. All rights reserved.

NVL 함수

null 값을 실제 값으로 변환하려면 NVL 함수를 사용합니다.

구문

`NVL(expr1, expr2)`

구문 설명:

- `expr1`은 **null**을 포함할 수 있는 소스 값 또는 표현식입니다.
- `expr2`는 **null**을 변환하기 위한 대상 값입니다.

NVL 함수를 사용하여 임의의 데이터 유형을 변환할 수 있지만 반환 값은 항상 `expr1`의 데이터 유형과 동일합니다.

다양한 데이터 유형에 대한 NVL 변환

데이터 유형	변환 예제
NUMBER	<code>NVL(number_column, 9)</code>
DATE	<code>NVL(date_column, '01-JAN-95')</code>
CHAR 또는 VARCHAR2	<code>NVL(character_column, 'Unavailable')</code>

NVL 함수 사용

```
SELECT last_name, salary, NVL(commission_pct, 0),
       (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	King	24000	0	288000
2	Kochhar	17000	0	204000
3	De Haan	17000	0	204000
4	Hunold	9000	0	108000
5	Ernst	6000	0	72000
6	Lorentz	4200	0	50400
7	Mourgos	5800	0	69600
8	Rajs	3500	0	42000
9	Davies	3100	0	37200
10	Matos	2600	0	31200
11	Vargas	2500	0	30000
12	Zlotkey	10500	0.2	151200
...				

1

2

ORACLE

Copyright © 2007, Oracle. All rights reserved.

NVL 함수 사용

모든 사원의 연봉을 계산하려면 월급에 12를 곱한 다음 결과에 커미션을 더합니다.

```
SELECT last_name, salary, commission_pct,
       (salary*12) + (salary*12*commission_pct) AN_SAL
FROM   employees;
```

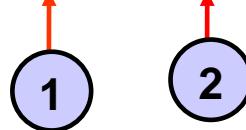
	LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
1	King	24000	(null)	(null)
...				
11	Vargas	2500	(null)	(null)
12	Zlotkey	10500	0.2	151200
13	Abel	11000	0.3	171600
...				

연봉은 커미션을 받는 사원에 대해서만 계산된다는 점에 유의하십시오. 표현식의 열 값이 null인 경우 결과는 null입니다. 모든 사원에 대해 값을 계산하려면 산술 연산자를 적용하기 전에 null 값을 숫자로 변환해야 합니다. 슬라이드의 예제에서 NVL 함수는 null 값을 0으로 변환하는 데 사용됩니다.

NVL2 함수 사용

```
SELECT last_name, salary, commission_pct ← 1
      NVL2(commission_pct,
            'SAL+COMM', 'SAL') income ← 2
  FROM employees WHERE department_id IN (50, 80);
```

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM



ORACLE

Copyright © 2007, Oracle. All rights reserved.

NVL2 함수 사용

NVL2 함수는 첫번째 표현식을 검사합니다. 첫번째 표현식이 null이 아니면 NVL2 함수는 두번째 표현식을 반환합니다. 첫번째 표현식이 null이면 세번째 표현식이 반환됩니다.

구문

`NVL2(expr1, expr2, expr3)`

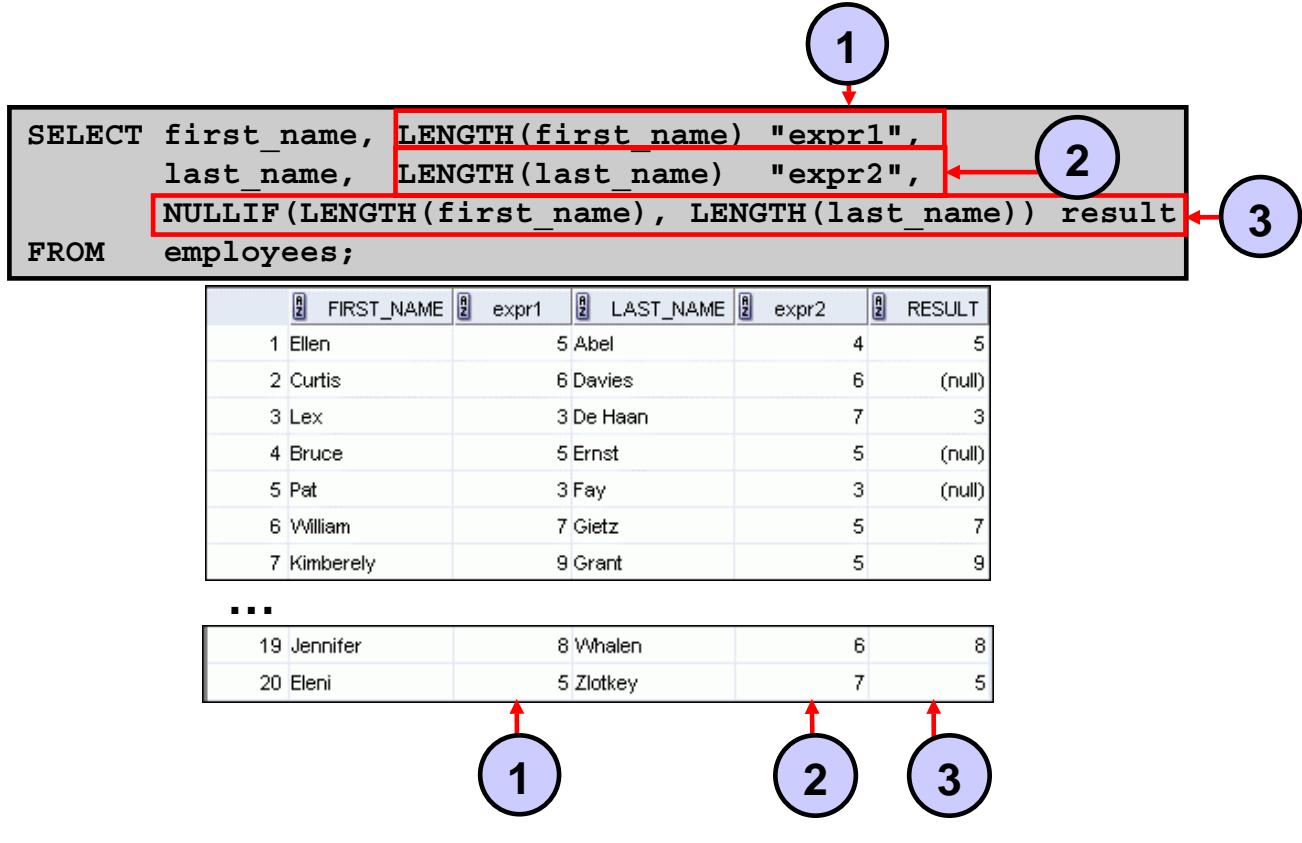
구문 설명:

- `expr1`은 null을 포함할 수 있는 소스 값 또는 표현식입니다.
- `expr2`는 `expr1`이 null이 아닌 경우에 반환되는 값입니다.
- `expr3`은 `expr1`이 null인 경우에 반환되는 값입니다.

슬라이드의 예제에서는 COMMISSION_PCT 열이 검사됩니다. 값이 발견되면 두번째 표현식 SAL+COMM이 반환됩니다. COMMISSION_PCT 열이 null 값을 보유하는 경우 세번째 표현식 SAL이 반환됩니다.

인수 `expr1`은 임의의 데이터 유형을 가질 수 있습니다. 인수 `expr2` 및 `expr3`은 LONG을 제외한 임의의 데이터 유형을 가질 수 있습니다. `expr2`와 `expr3`의 데이터 유형이 서로 다른 경우 Oracle 서버는 `expr3`이 null 상수가 아닌 한, 두 데이터 유형을 비교하기 전 `expr3`을 `expr2`의 데이터 유형으로 변환합니다. null 상수인 경우 데이터 유형 변환이 필요하지 않습니다. 반환 값의 데이터 유형은 `expr2`가 문자 데이터인 경우를 제외하고 항상 `expr2`의 데이터 유형과 동일합니다. `expr2`가 문자 데이터인 경우 반환 값의 데이터 유형은 VARCHAR2입니다.

NULLIF 함수 사용



ORACLE

Copyright © 2007, Oracle. All rights reserved.

NULLIF 함수 사용

`NULLIF` 함수는 두 표현식을 비교합니다. 두 표현식이 같으면 이 함수는 null을 반환합니다. 두 표현식이 같지 않으면 이 함수는 첫번째 표현식을 반환합니다. 그러나 첫번째 표현식에 대해 리터럴 `NULL`을 지정할 수 없습니다.

구문

```
NULLIF (expr1, expr2)
```

구문 설명:

- `NULLIF`는 `expr1`과 `expr2`를 비교합니다. 두 표현식이 같으면 이 함수는 `null`을 반환합니다. 두 표현식이 다르면 이 함수는 `expr1`을 반환합니다. 그러나 `expr1`에 대해 리터럴 `NULL`을 지정할 수 없습니다.

슬라이드의 예제에서 EMPLOYEES 테이블에 있는 이름의 길이를 EMPLOYEES 테이블에 있는 성의 길이와 비교합니다. 이름과 성의 길이가 같으면 `null` 값이 표시됩니다. 이름과 성의 길이가 다르면 이름의 길이가 표시됩니다.

주: `NULLIF` 함수는 논리적으로 다음 CASE 식과 동일합니다. CASE 식은 후속 페이지에서 설명합니다.

```
CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END
```

COALESCE 함수 사용

- NVL 함수 대신 COALESCE 함수를 사용했을 때의 이점은 COALESCE 함수가 여러 대체 값을 수용할 수 있다는 것입니다.
- 첫번째 표현식이 null이 아닌 경우 COALESCE 함수는 해당 표현식을 반환합니다. 그렇지 않은 경우 나머지 표현식에 COALESCE를 수행합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

COALESCE 함수 사용

COALESCE 함수는 리스트에서 null이 아닌 첫번째 표현식을 반환합니다.

구문

COALESCE (expr1, expr2, ... exprn)

구문 설명:

- expr1 - 이 표현식이 null이 아닌 경우 이 표현식을 반환합니다.
- expr2 - 첫번째 표현식이 null이고 이 표현식이 null이 아닌 경우 이 표현식을 반환합니다.
- exprn - 선행 표현식이 null인 경우 이 표현식을 반환합니다.

모든 표현식은 동일한 데이터 유형이어야 합니다.

COALESCE 함수 사용

```
SELECT last_name, employee_id,
       COALESCE(TO_CHAR(commission_pct), TO_CHAR(manager_id),
                 'No commission and no manager')
  FROM employees;
```

LAST_NAME	EMPLOYEE_ID	COALESCE(TO_CHAR(COMMISSION_PCT), TO_CHAR(MANAGER_ID), 'No commission and no manager')
King	100	No commission and no manager
Kochhar	101	100
De Haan	102	100
Hunold	103	102
Ernst	104	103
Lorentz	107	103
Mourgos	124	100
Rajs	141	124
...		
Zlotkey	149	.2
Abel	174	.3
Taylor	176	.2
Grant	178	.15
Whalen	200	101
...		

ORACLE

Copyright © 2007, Oracle. All rights reserved.

COALESCE 함수 사용(계속)

슬라이드의 예제에서 manager_id 값이 null이 아닌 경우 해당 값이 표시됩니다. manager_id 값이 null인 경우 commission_pct가 표시됩니다. manager_id 및 commission_pct 값이 null이면 "No commission and no manager"가 표시됩니다. TO_CHAR 함수가 적용되어 모든 표현식의 데이터 유형이 동일합니다.

COALESCE 함수 사용(계속)

예제:

조직에서 커미션을 받지 않는 사원에게 \$2,000의 급여 인상을 제공하려고 합니다. 또한 커미션을 받는 사원의 경우에는 기존 급여에 커미션 금액을 추가한 새 급여를 query에서 계산해야 합니다.

```
SELECT last_name, salary, commission_pct,
       COALESCE((salary+(commission_pct*salary)), salary+2000, salary)
              "New Salary"
        FROM employees;
```

주: 출력을 검사합니다. 커미션을 받지 않는 사원의 경우 \$2,000가 인상된 새 급여가 New Salary 열에 표시되고 커미션을 받는 사원의 경우 커미션 금액을 추가하여 계산된 급여가 New Salary 열에 표시됩니다.

	LAST_NAME	SALARY	COMMISSION_PCT	New Salary
1	King	24000	(null)	26000
2	Kochhar	17000	(null)	19000
3	De Haan	17000	(null)	19000
4	Hunold	9000	(null)	11000

...

9	Davies	3100	(null)	5100
10	Matos	2600	(null)	4600
11	Vargas	2500	(null)	4500
12	Zlotkey	10500	0.2	12600
13	Abel	11000	0.3	14300
14	Taylor	8600	0.2	10320
15	Grant	7000	0.15	8050
16	Whalen	4400	(null)	6400
17	Hartstein	13000	(null)	15000
18	Fay	6000	(null)	8000
19	Higgins	12000	(null)	14000
20	Gietz	8300	(null)	10300

...

단원 내용

- 암시적 및 명시적 데이터 유형 변환
- **TO_CHAR**, **TO_DATE**, **TO_NUMBER** 함수
- 함수 중첩
- 일반 함수:
 - **NVL**
 - **NVL2**
 - **NULLIF**
 - **COALESCE**
- 조건부 표현식:
 - **CASE**
 - **DECODE**

ORACLE

Copyright © 2007, Oracle. All rights reserved.

조건부 표현식

- SQL 문에서 **IF-THEN-ELSE** 논리를 사용할 수 있습니다.
- 다음 두 가지 방법을 사용합니다.
 - **CASE** 식
 - **DECODE** 함수

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

조건부 표현식

SQL 문에서 조건부 처리(IF-THEN-ELSE 논리)를 구현하는 데 사용되는 두 가지 방법은 CASE 식과 DECODE 함수입니다.

주: CASE 식은 ANSI SQL을 준수합니다. DECODE 함수는 Oracle 구문에만 해당됩니다.

CASE 식

IF-THEN-ELSE 문의 작업을 수행하여 조건부 조회를 편리하게 수행합니다.

```
CASE expr WHEN comparison_expr1 THEN return_expr1
           [WHEN comparison_expr2 THEN return_expr2
            WHEN comparison_exprn THEN return_exprn
            ELSE else_expr]
END
```



Copyright © 2007, Oracle. All rights reserved.

CASE 식

CASE 식을 사용하면 프로시저를 호출하지 않고도 SQL 문에서 IF-THEN-ELSE 논리를 사용할 수 있습니다.

간단한 CASE 식의 경우 Oracle 서버는 expr이 comparison_expr과 동일한 첫번째 WHEN ... THEN 쌍을 검색하여 return_expr을 반환합니다. 이 조건을 충족하는 WHEN ... THEN 쌍이 없고 ELSE 절이 존재하면 Oracle 서버는 else_expr을 반환합니다. 그렇지 않은 경우 Oracle 서버는 null을 반환합니다. 일부 return_expr 및 else_expr에 대해서는 리터럴 NULL을 지정할 수 없습니다.

모든 표현식(expr, comparison_expr 및 return_expr)은 동일한 데이터 유형이어야 하며 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2가 될 수 있습니다.

CASE 식 사용

IF-THEN-ELSE 문의 작업을 수행하여 조건부 조회를 편리하게 수행합니다.

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
                     WHEN 'ST_CLERK' THEN 1.15*salary
                     WHEN 'SA REP' THEN 1.20*salary
       ELSE          salary END      "REVISED_SALARY"
FROM employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
Ernst	IT_PROG	6000	6600
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
Davies	ST_CLERK	3100	3565
...			
Abel	SA REP	11000	13200
Taylor	SA REP	8600	10320
...			

ORACLE

Copyright © 2007, Oracle. All rights reserved.

CASE 식 사용

슬라이드의 SQL 문에서는 JOB_ID 값이 디코딩됩니다. JOB_ID가 IT_PROG이면 급여 인상률이 10%이고, JOB_ID가 ST_CLERK이면 급여 인상률이 15%이며, JOB_ID가 SA REP이면 급여 인상률이 20%입니다. 다른 모든 직무의 경우 급여 인상은 없습니다.

동일한 명령문을 DECODE 함수를 사용하여 작성할 수 있습니다.

다음은 검색된 CASE 식의 예입니다. 검색된 CASE 식의 경우 나열된 조건의 발생 값이 발견될 때까지 왼쪽에서 오른쪽으로 검색한 다음 반환식을 반환합니다. 참인 조건을 찾을 수 없고 ELSE 절이 존재하는 경우 ELSE 절의 반환식이 반환되고, 그렇지 않으면 NULL이 반환됩니다.

```
SELECT last_name, salary,
       (CASE WHEN salary<5000 THEN 'Low'
             WHEN salary<10000 THEN 'Medium'
             WHEN salary<20000 THEN 'Good'
             ELSE 'Excellent'
       END) qualified_salary
FROM employees;
```

DECODE 함수

CASE 식 또는 **IF-THEN-ELSE** 문의 작업을 수행하여 조건부 조회를 편리하게 수행합니다.

```
DECODE(col / expression, search1, result1
       [, search2, result2, ...]
       [, default])
```

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

DECODE 함수

DECODE 함수는 다양한 언어에서 사용되는 IF-THEN-ELSE 논리와 비슷한 방식으로 표현식을 디코딩합니다. DECODE 함수는 표현식을 각 검색 값과 비교한 후에 디코딩합니다. 표현식이 검색 값과 동일하면 결과 값이 반환됩니다.

기본값이 생략된 경우 검색 값과 일치하는 결과 값이 없으면 null 값이 반환됩니다.

DECODE 함수 사용

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG', 1.10*salary,
              'ST_CLERK', 1.15*salary,
              'SA REP', 1.20*salary,
              salary)
       REVISED_SALARY
  FROM employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
6 Lorentz	IT_PROG	4200	4620	
7 Mourgos	ST_MAN	5800	5800	
8 Rajs	ST_CLERK	3500	4025	
...				
13 Abel	SA REP	11000	13200	
14 Taylor	SA REP	8600	10320	
...				

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DECODE 함수 사용

슬라이드의 SQL 문에서는 JOB_ID 값이 테스트됩니다. JOB_ID가 IT_PROG이면 급여 인상률이 10%이고, JOB_ID가 ST_CLERK이면 급여 인상률이 15%이며, JOB_ID가 SA REP이면 급여 인상률이 20%입니다. 다른 모든 직무의 경우 급여 인상은 없습니다.

의사 코드를 사용하여 동일한 명령문을 IF-THEN-ELSE 문으로 표현할 수 있습니다.

```
IF job_id = 'IT_PROG'      THEN salary = salary*1.10
IF job_id = 'ST_CLERK'      THEN salary = salary*1.15
IF job_id = 'SA REP'        THEN salary = salary*1.20
ELSE salary = salary
```

DECODE 함수 사용

부서 80의 각 사원에 대해 적용 가능한 세율을 표시합니다.

```
SELECT last_name, salary,  
       DECODE (TRUNC(salary/2000, 0),  
                0, 0.00,  
                1, 0.09,  
                2, 0.20,  
                3, 0.30,  
                4, 0.40,  
                5, 0.42,  
                6, 0.44,  
                0.45) TAX_RATE  
FROM   employees  
WHERE  department_id = 80;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DECODE 함수 사용(계속)

이 슬라이드는 DECODE 함수를 사용하는 다른 예제를 보여줍니다. 이 예제에서는 월급을 기준으로 부서 80의 각 사원에 대해 세율을 판별합니다. 세율은 다음과 같습니다.

월급 범위	세율
\$0.00–1,999.99	00%
\$2,000.00–3,999.99	09%
\$4,000.00–5,999.99	20%
\$6,000.00–7,999.99	30%
\$8,000.00–9,999.99	40%
\$10,000.00–11,999.99	42%
\$12,200.00–13,999.99	44%
\$14,000.00 이상	45%

LAST_NAME	SALARY	TAX_RATE
1 Zlotkey	10500	0.42
2 Abel	11000	0.42
3 Taylor	8600	0.4

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 함수를 사용하여 날짜 표시 형식 변경
- 함수를 사용하여 열 데이터 유형 변환
- **NVL** 함수 사용
- **SELECT** 문에서 **IF-THEN-ELSE** 논리 및 다른 조건부 표현식 사용



Copyright © 2007, Oracle. All rights reserved.

요약

다음 사항을 기억하십시오.

- **TO_CHAR**, **TO_DATE**, **TO_NUMBER** 변환 함수는 문자, 날짜 및 숫자 값을 변환할 수 있습니다.
- **NVL**, **NVL2**, **NULLIF** 및 **COALESCE**를 포함하여 null과 관련된 여러 함수가 있습니다.
- **CASE** 식 또는 **DECODE** 함수를 사용하여 SQL 문에 **IF-THEN-ELSE** 논리를 적용할 수 있습니다.

연습 4: 개요

이 연습에서는 다음 내용을 다룹니다.

- **TO_CHAR, TO_DATE** 및 다른 **DATE** 함수를 사용하는 **query** 작성
- **DECODE** 및 **CASE**와 같은 조건부 표현식을 사용하는 **query** 작성



Copyright © 2007, Oracle. All rights reserved.

연습 4: 개요

이 연습에서는 **TO_CHAR** 및 **TO_DATE** 함수와 **DECODE** 및 **CASE**와 같은 조건부 표현식을 사용하는 다양한 실습을 제공합니다. 중첩 함수의 경우 결과는 가장 안쪽 함수에서 가장 바깥쪽 함수로 평가됩니다.

연습 4

1. 각 사원에 대해 다음과 같이 출력되는 보고서를 작성합니다.

<employee last name> earns <salary> monthly but wants
<3 times salary> 열 레이블을 Dream Salaries로 지정합니다.

Dream Salaries	
1	King earns \$24,000.00 monthly but wants \$72,000.00.
2	Kochhar earns \$17,000.00 monthly but wants \$51,000.00.
3	De Haan earns \$17,000.00 monthly but wants \$51,000.00.
4	Hunold earns \$9,000.00 monthly but wants \$27,000.00.
5	Ernst earns \$6,000.00 monthly but wants \$18,000.00.
...	
19	Higgins earns \$12,000.00 monthly but wants \$36,000.00.
20	Gietz earns \$8,300.00 monthly but wants \$24,900.00.

2. 각 사원의 성, 채용 날짜 및 근무 6개월 후 첫번째 월요일에 해당하는 급여 심의 날짜를 표시합니다. 열 레이블을 REVIEW로 지정합니다. 날짜 형식을 "Monday, the Thirty-First of July, 2000"과 유사한 형식으로 지정합니다.

LAST_NAME	HIRE_DATE	REVIEW
1 King	17-JUN-87	Monday, the Twenty-First of December, 1987
2 Kochhar	21-SEP-89	Monday, the Twenty-Sixth of March, 1990
3 De Haan	13-JAN-93	Monday, the Nineteenth of July, 1993
4 Hunold	03-JAN-90	Monday, the Ninth of July, 1990
5 Ernst	21-MAY-91	Monday, the Twenty-Fifth of November, 1991
...		
19 Higgins	07-JUN-94	Monday, the Twelfth of December, 1994
20 Gietz	07-JUN-94	Monday, the Twelfth of December, 1994

연습 4(계속)

3. 사원의 성, 채용 날짜, 근무 시작 요일을 표시합니다. 열 레이블을 DAY로 지정합니다.
월요일부터 시작하여 요일순으로 결과를 정렬합니다.

	LAST_NAME	HIRE_DATE	DAY
1	Grant	24-MAY-99	MONDAY
2	Gietz	07-JUN-94	TUESDAY
3	Taylor	24-MAR-98	TUESDAY
4	Higgins	07-JUN-94	TUESDAY
5	Rajs	17-OCT-95	TUESDAY

...

19	Lorentz	07-FEB-99	SUNDAY
20	Fay	17-AUG-97	SUNDAY

4. 사원의 성과 커미션 금액을 표시하는 query를 작성합니다. 사원이 커미션을 받지 않으면 "No Commission"을 표시합니다. 열 레이블을 COMM으로 지정합니다.

	LAST_NAME	COMM
1	King	No Commission
2	Kochhar	No Commission
3	De Haan	No Commission
4	Hunold	No Commission
5	Ernst	No Commission
6	Lorentz	No Commission

...

12	Zlotkey	.2
13	Abel	.3
14	Taylor	.2
15	Grant	.15
16	Whalen	No Commission
17	Hartstein	No Commission
18	Fay	No Commission
19	Higgins	No Commission
20	Gietz	No Commission

연습 4(계속)

시간 여유가 있을 경우 다음 연습을 완료하십시오.

5. 다음 데이터를 사용하여 DECODE 함수를 통해 JOB_ID 열의 값을 기반으로 모든 사원의 등급을 표시하는 query를 작성합니다.

직책	등급
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA REP	D
ST_CLERK	E
기타	O

JOB_ID	GRADE
1 AC_ACCOUNT	O
2 AC_MGR	O
3 AD_ASST	O
4 AD_PRES	A
5 AD_VP	O

...
18 ST_CLERK E
19 ST_CLERK E
20 ST_MAN B

6. CASE 구문을 사용하여 앞의 연습에 나오는 명령문을 재작성합니다.

JOB_ID	GRADE
1 AC_ACCOUNT	O
2 AC_MGR	O
3 AD_ASST	O
4 AD_PRES	A
5 AD_VP	O

...
18 ST_CLERK E
19 ST_CLERK E
20 ST_MAN B

그룹 함수를 사용하여 집계 데이터 보고

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 사용 가능한 그룹 함수 식별
- 그룹 함수 사용법 설명
- **GROUP BY** 절을 사용하여 데이터 그룹화
- **HAVING** 절을 사용하여 그룹화된 행 포함 또는 제외



Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 함수에 대해 추가로 설명합니다. 행 그룹에 대한 평균 등의 요약 정보를 구하는 방법에 대해 중점적으로 살펴봅니다. 테이블의 행을 더 작은 집합으로 그룹화하는 방법과 행 그룹에 대한 검색 조건을 지정하는 방법에 대해 설명합니다.

단원 내용

- 그룹 함수:
 - 유형 및 구문
 - **AVG, SUM, MIN, MAX, COUNT** 사용
 - 그룹 함수 내에 **DISTINCT** 키워드 사용
 - 그룹 함수의 **NULL** 값
- 다음과 같은 방법을 사용하여 행을 그룹화합니다.
 - **GROUP BY** 절
 - **HAVING** 절
- 그룹 함수 중첩

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

그룹 함수란?

그룹 함수는 행 집합에 대해 실행되어 그룹당 하나의 결과를 산출합니다.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

EMPLOYEES

테이블의 최대 급여



ORACLE

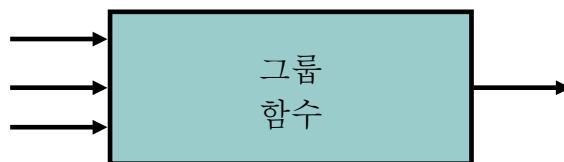
Copyright © 2007, Oracle. All rights reserved.

그룹 함수란?

단일 행 함수와 달리 그룹 함수는 행 집합에 대해 실행되어 그룹당 하나의 결과를 산출합니다.
이러한 행 집합은 전체 테이블이나 그룹으로 분할된 테이블로 구성될 수 있습니다.

그룹 함수 유형

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**



ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 함수 유형

각 함수는 인수를 받아들입니다. 다음 표는 구문에서 사용할 수 있는 옵션에 대한 설명입니다.

함수	설명
AVG ([DISTINCT <u>ALL</u>] n)	n의 평균값. null 값은 무시합니다.
COUNT ({ * [DISTINCT <u>ALL</u>] expr })	행 개수. 여기서 expr은 null이 아닌 값을 평가합니다. (*를 사용하여 중복된 행과 null 값으로 된 행을 비롯하여 선택된 모든 행의 수를 셩니다.)
MAX ([DISTINCT <u>ALL</u>] expr)	expr의 최대값. null 값은 무시합니다.
MIN ([DISTINCT <u>ALL</u>] expr)	expr의 최소값. null 값은 무시합니다.
STDDEV ([DISTINCT <u>ALL</u>] x)	n의 표준 편차. null 값은 무시합니다.
SUM ([DISTINCT <u>ALL</u>] n)	n의 합계 값. null 값은 무시합니다.
VARIANCE ([DISTINCT <u>ALL</u>] x)	n의 분산. null 값은 무시합니다.

그룹 함수: 구문

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

ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 함수: 구문

그룹 함수는 SELECT 키워드 뒤에 배치합니다. 여러 그룹 함수를 쉼표로 구분하여 함께 사용할 수 있습니다.

그룹 함수 사용 지침:

- DISTINCT는 함수가 중복되지 않는 값만 사용하도록 만듭니다. ALL은 중복된 값을 포함하여 모든 값을 사용하도록 만듭니다. 기본값은 ALL이므로 별도로 ALL을 지정할 필요는 없습니다.
- expr 인수를 사용하는 함수의 데이터 유형은 CHAR, VARCHAR2, NUMBER 또는 DATE가 될 수 있습니다.
- 모든 그룹 함수는 null 값을 무시합니다. null을 값으로 치환하려면 NVL, NVL2 또는 COALESCE 함수를 사용합니다.

AVG 및 SUM 함수 사용

숫자 데이터에 대해 **AVG** 및 **SUM** 함수를 사용할 수 있습니다.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
  FROM employees  
 WHERE job_id LIKE '%REP%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

ORACLE

Copyright © 2007, Oracle. All rights reserved.

AVG 및 SUM 함수 사용

숫자 데이터를 저장할 수 있는 열에 대해 AVG, SUM, MIN 및 MAX 함수를 사용할 수 있습니다.
슬라이드의 예제는 모든 판매 담당자의 월급에 대해 평균, 최고값, 최저값 및 합계를 표시합니다.

MIN 및 MAX 함수 사용

숫자, 문자 및 날짜 데이터 유형에 대해 **MIN** 및 **MAX** 함수를 사용할 수 있습니다.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

MIN(HIRE_DATE)	MAX(HIRE_DATE)
1 17-JUN-87	29-JAN-00

ORACLE

Copyright © 2007, Oracle. All rights reserved.

MIN 및 MAX 함수 사용

숫자, 문자 및 날짜 데이터 유형에 대해 MAX 및 MIN 함수를 사용할 수 있습니다. 슬라이드의 예제에서는 가장 최근에 입사한 사원과 가장 오래 근무한 사원을 표시합니다.

다음 예제에서는 사전순으로 정렬된 모든 사원 리스트에서 맨 먼저 나오는 사원의 성과 맨 끝에 나오는 사원의 성을 표시합니다.

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Abel	Zlotkey

주: AVG, SUM, VARIANCE 및 STDDEV 함수는 숫자 데이터 유형에만 사용할 수 있습니다. MAX 및 MIN 함수는 LOB 또는 LONG 데이터 유형에 사용할 수 없습니다.

COUNT 함수 사용

COUNT (*) 는 테이블의 행 수를 반환합니다.

1

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
```

	COUNT(*)
1	5

COUNT (expr) 은 expr에 대해 null이 아닌 값을 가진 행의 수를 반환합니다.

2

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

	COUNT(COMMISSION_PCT)
1	3

ORACLE

Copyright © 2007, Oracle. All rights reserved.

COUNT 함수 사용

COUNT 함수에는 다음 세 가지 형식이 있습니다.

- COUNT (*)
- COUNT (expr)
- COUNT (DISTINCT expr)

COUNT (*) 는 SELECT 문의 조건을 충족하는 테이블의 행 수를 반환하며 여기에는 중복 행과 열에 null 값을 포함한 행이 포함됩니다. SELECT 문에 WHERE 절이 포함된 경우 COUNT (*) 는 WHERE 절의 조건을 충족하는 행 수를 반환합니다.

이와 반대로 COUNT (expr) 은 expr에 의해 식별되는 열에 있는 null이 아닌 값의 수를 반환합니다.

COUNT (DISTINCT expr) 은 expr에 의해 식별되는 열에 있는 고유하고 null이 아닌 값의 수를 반환합니다.

예제:

1. 슬라이드의 예제에서는 부서 50의 사원 수를 표시합니다.
2. 슬라이드의 예제에서는 커미션을 받을 수 있는 부서 80의 사원 수를 표시합니다.

DISTINCT 키워드 사용

- COUNT(DISTINCT *expr*)은 *expr*의 null이 아닌 구분 값의 수를 반환합니다.
- 다음은 EMPLOYEES 테이블에서 부서의 구분 값 수를 표시합니다.

```
SELECT COUNT(DISTINCT department_id)  
FROM employees;
```

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

DISTINCT 키워드 사용

열에서 중복 값을 세지 않으려면 DISTINCT 키워드를 사용합니다.

슬라이드의 예제에서는 EMPLOYEES 테이블에 있는 부서의 구분 값 수를 표시합니다.

그룹 함수 및 null 값

그룹 함수는 열에 있는 null 값을 무시합니다.

1

```
SELECT AVG(commission_pct)
FROM employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

NVL 함수는 강제로 그룹 함수에 null 값이 포함되도록 합니다.

2

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 함수 및 null 값

모든 그룹 함수는 열에 있는 null 값을 무시합니다.

그러나 NVL 함수는 강제로 그룹 함수에 null 값이 포함되도록 합니다.

예제:

- 평균 계산에는 테이블에서 COMMISSION_PCT 열에 유효한 값이 저장된 행만 사용됩니다. 평균은 모든 사원에게 지급된 총 커미션을 커미션을 받는 사원 수(4)로 나누어 계산합니다.
- 평균 계산에는 COMMISSION_PCT 열에 null 값이 저장되었는지 여부에 관계없이 테이블의 모든 행이 사용됩니다. 평균은 모든 사원에게 지급된 총 커미션을 회사의 총 사원 수(20)로 나누어 계산합니다.

단원 내용

- 그룹 함수:
 - 유형 및 구문
 - **AVG, SUM, MIN, MAX, COUNT** 사용
 - 그룹 함수 내에 **DISTINCT** 키워드 사용
 - 그룹 함수의 **NULL** 값
- 다음과 같은 방법을 사용하여 행을 그룹화합니다.
 - **GROUP BY** 절
 - **HAVING** 절
- 그룹 함수 중첩

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터 그룹 생성

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600
...		
19	110	12000
20	(null)	7000

EMPLOYEES 테이블의
각 부서에 대한 평균 급여

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.33333333333...
6	90	19333.33333333333...
7	110	10150
8	(null)	7000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터 그룹 생성

지금까지의 설명에서는 모든 그룹 함수가 테이블을 하나의 커다란 정보 그룹으로 취급했습니다. 그러나 정보 테이블을 더 작은 그룹으로 나눠야 하는 경우도 있습니다. 이러한 작업은 GROUP BY 절을 사용하여 수행할 수 있습니다.

데이터 그룹 생성: GROUP BY 절 구문

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

GROUP BY 절을 사용하여 테이블의 행을 더 작은 그룹으로 나눌 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터 그룹 생성: GROUP BY 절 구문

GROUP BY 절을 사용하여 테이블의 행을 그룹으로 나눌 수 있습니다. 그런 다음 그룹 함수를 사용하여 각 그룹에 대한 요약 정보를 반환할 수 있습니다.

구문 설명:

group_by_expression 행 그룹화의 기초를 결정하는 값이 포함된 열을 지정합니다.

지침

- SELECT 절에 그룹 함수를 포함하고 GROUP BY 절에 개별 열을 지정하지 않는 경우 개별 결과도 선택할 수 없습니다. GROUP BY 절에 열 리스트를 포함하지 않으면 오류 메시지가 나타납니다.
- WHERE 절을 사용하면 행을 그룹으로 나누기 전에 행을 제외시킬 수 있습니다.
- GROUP BY 절에 열을 포함시켜야 합니다.
- GROUP BY 절에서 열 alias를 사용할 수 없습니다.

GROUP BY 절 사용

그룹 함수의 인수가 아닌 **SELECT** 리스트의 모든 열은 **GROUP BY** 절에 있어야 합니다.

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

	DEPARTMENT_ID	Avg(SALARY)
1	(null)	7000
2	90	19333.3333333333...
3	20	9500
4	110	10150
5	50	3500
6	80	10033.3333333333...
7	60	6400
8	10	4400

ORACLE

Copyright © 2007, Oracle. All rights reserved.

GROUP BY 절 사용

GROUP BY 절을 사용할 경우 그룹 함수가 아닌 SELECT 리스트의 모든 열은 GROUP BY 절에 포함되어야 합니다. 슬라이드의 예제에서는 각 부서의 부서 번호와 평균 급여를 표시합니다. GROUP BY 절을 포함한 이 SELECT 문이 평가되는 방식은 다음과 같습니다.

- SELECT 절은 다음과 같이 검색할 열을 지정합니다.
 - EMPLOYEES 테이블의 부서 번호 열
 - GROUP BY 절에 지정한 그룹의 모든 급여 평균
- FROM 절은 데이터베이스가 액세스해야 하는 테이블, 즉 EMPLOYEES 테이블을 지정합니다.
- WHERE 절은 검색할 행을 지정합니다. WHERE 절이 없기 때문에 기본적으로 모든 행이 검색됩니다.
- GROUP BY 절은 행을 그룹화하는 방법을 지정합니다. 행은 부서 번호별로 그룹화되므로 salary 열에 적용되는 AVG 함수는 각 부서의 평균 급여를 계산합니다.

GROUP BY 절 사용

GROUP BY 열은 **SELECT** 리스트에 없어도 됩니다.

```
SELECT      AVG(salary)
FROM        employees
GROUP BY    department_id;
```

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

GROUP BY 절 사용(계속)

GROUP BY 열은 SELECT 절에 없어도 됩니다. 예를 들어, 슬라이드의 SELECT 문은 개별 부서 번호를 표시하지 않고 각 부서의 평균 급여를 표시합니다. 그러나 부서 번호가 없으면 결과가 의미가 없습니다.

`ORDER BY` 절에서도 그룹 함수를 사용할 수 있습니다.

```
SELECT      department_id,  AVG(salary)
  FROM      employees
 GROUP BY  department_id
 ORDER BY  AVG(salary);
```

두 개 이상의 열로 그룹화

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1		10 AD_ASST	4400
2		20 MK_MAN	13000
3		20 MK_REP	6000
4		50 ST_MAN	5800
5		50 ST_CLERK	2500
6		50 ST_CLERK	2600
7		50 ST_CLERK	3100
8		50 ST_CLERK	3500
9		60 IT_PROG	4200
10		60 IT_PROG	6000
11		60 IT_PROG	9000
12		80 SA REP	11000
13		80 SA_MAN	10500
14		80 SA REP	8600
...			
19		110 AC_MGR	12000
20		(null) SA REP	7000

EMPLOYEES 테이블에서 부서별로 그룹화한 각 직무에 대해 급여를 더합니다.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1		10 AD_ASST	4400
2		20 MK_MAN	13000
3		20 MK_REP	6000
4		50 ST_CLERK	11700
5		50 ST_MAN	5800
6		60 IT_PROG	19200
7		80 SA_MAN	10500
8		80 SA REP	19600
9		90 AD_PRES	24000
10		90 AD_VP	34000
11		110 AC_ACCOUNT	8300
12		110 AC_MGR	12000
13		(null) SA REP	7000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

두 개 이상의 열로 그룹화

때때로 그룹 내 그룹에 대해 결과를 확인해야 할 경우가 있습니다. 슬라이드는 각 부서에서 각 직책에 지급된 총 급여를 표시하는 보고서를 보여줍니다.

EMPLOYEES 테이블은 먼저 부서 번호별로 그룹화된 다음 해당 그룹 내에서 직무별로 그룹화됩니다. 예를 들어, 부서 50에서 네 명의 stock clerk이 함께 그룹화되고, 이 그룹의 모든 stock clerk에 대해 단일 결과(총 급여)가 산출됩니다.

다중 열에서 GROUP BY 절 사용

```
SELECT      department_id dept_id, job_id, SUM(salary)
FROM        employEes
GROUP BY    department_id, job_id
ORDER BY    department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다중 열에서 Group By 절 사용

두 개 이상의 GROUP BY 열을 나열하여 그룹과 하위 그룹에 대한 요약 결과를 반환할 수 있습니다.

GROUP BY 절에 나열된 열의 순서에 따라 결과가 정렬되도록 기본 순서를 결정할 수 있습니다.

슬라이드의 예제에서 GROUP BY 절을 포함한 SELECT 문은 다음과 같이 평가됩니다.

- SELECT 절은 다음과 같이 검색할 열을 지정합니다.
 - EMPLOYEES 테이블의 부서 번호
 - EMPLOYEES 테이블의 직무 ID
 - GROUP BY 절에서 지정한 그룹의 모든 급여 합계
- FROM 절은 데이터베이스가 액세스해야 하는 테이블, 즉 EMPLOYEES 테이블을 지정합니다.
- GROUP BY 절은 행을 그룹화하는 방법을 지정합니다.
 - 먼저 행이 부서 번호별로 그룹화됩니다.
 - 이어서 행이 부서 번호 그룹에서 직무 ID별로 그룹화됩니다.

따라서 SUM 함수는 각 부서 번호 그룹의 모든 직무 ID에 대한 salary 열에 적용됩니다.

그룹 함수를 사용한 잘못된 query

집계 함수가 아닌 **SELECT** 리스트의 열이나 표현식은 **GROUP BY** 절에 있어야 합니다.

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"

각 **department_id**에 대해 성의 개수를 세려면 **GROUP BY** 절을 추가해야 합니다.

```
SELECT department_id, job_id, COUNT(last_name)
FROM   employees
GROUP BY department_id;
```

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"

GROUP BY에 **job_id**를 추가하거나 **SELECT** 리스트에서 **job_id** 열을 제거합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 함수를 사용한 잘못된 query

개별 항목(DEPARTMENT_ID)과 그룹 함수(COUNT)를 동일한 SELECT 문에서 함께 사용할 경우 개별 항목(이 경우 DEPARTMENT_ID)을 지정하는 GROUP BY 절을 포함시켜야 합니다. GROUP BY 절이 누락된 경우 "not a single-group group function"이라는 오류 메시지가 나타납니다. 별표(*)는 오류가 발생한 열을 가리킵니다. GROUP BY 절을 추가하여 슬라이드의 첫번째 예제에서 오류를 바로잡을 수 있습니다.

```
SELECT      department_id, count(last_name)
FROM        employees
GROUP BY    department_id;
```

집계 함수가 아닌 **SELECT** 리스트의 열이나 표현식은 **GROUP BY** 절에 있어야 합니다.

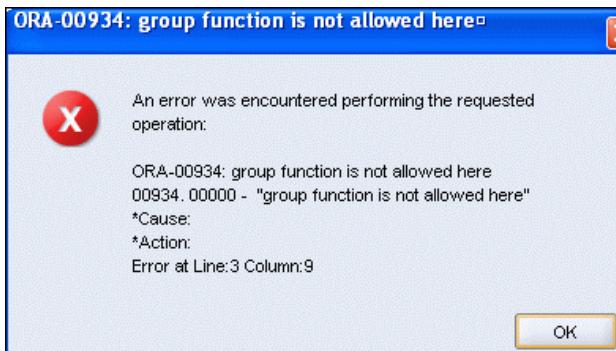
슬라이드의 두번째 예제에서 **job_id**는 GROUP BY 절에 있지도 않고 그룹 함수에 사용되지도 않으므로 "not a GROUP BY expression" 오류가 발생합니다. GROUP BY 절에 **job_id**를 추가하여 두번째 슬라이드 예제에서 오류를 바로잡을 수 있습니다.

```
SELECT      department_id, job_id, COUNT(last_name)
FROM        employees
GROUP BY    department_id, job_id;
```

그룹 함수를 사용한 잘못된 query

- **WHERE** 절은 그룹을 제한하는 데 사용할 수 없습니다.
- 그룹을 제한하려면 **HAVING** 절을 사용합니다.
- **WHERE** 절에서 그룹 함수를 사용할 수 없습니다.

```
SELECT      department_id, AVG(salary)
FROM        employees
WHERE       AVG(salary) > 8000
GROUP BY   department_id;
```



WHERE 절은 그룹을
제한하는 데 사용할
수 없음

ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 함수를 사용한 잘못된 query(계속)

WHERE 절은 그룹을 제한하는 데 사용할 수 없습니다. 슬라이드 예제의 SELECT 문은 평균 급여가 \$8,000가 넘는 부서에 대해서만 평균 급여를 표시하도록 제한하는 데 WHERE 절을 사용하기 때문에 오류가 발생합니다.

그러나 그룹을 제한하는 데 HAVING 절을 사용하면 예제의 오류를 바로 잡을 수 있습니다.

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY   department_id
HAVING     AVG(salary) > 8000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.3333333333333333...
2	20	9500
3	110	10150
4	80	10033.3333333333333333...

그룹 결과 제한

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

최고 급여가 \$10,000가 넘는
각 부서의 최고 급여

DEPARTMENT_ID	MAX(SALARY)
1	20
2	80
3	90
4	110

ORACLE

Copyright © 2007, Oracle. All rights reserved.

그룹 결과 제한

WHERE 절을 사용하여 선택할 행을 제한하는 것과 동일한 방식으로 HAVING 절을 사용하여 그룹을 제한합니다. 최고 급여가 \$10,000가 넘는 각 부서의 최고 급여를 알아내려면 다음 작업을 수행해야 합니다.

1. 부서 번호별로 그룹화하여 각 부서의 평균 급여를 알아냅니다.
2. 최고 급여가 \$10,000가 넘는 부서로 그룹을 제한합니다.

HAVING 절을 사용하여 그룹 결과 제한

HAVING 절을 사용할 경우 **Oracle** 서버는 다음과 같이 그룹을 제한합니다.

1. 행이 그룹화됩니다.
2. 그룹 함수가 적용됩니다.
3. **HAVING** 절과 일치하는 그룹이 표시됩니다.

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

Copyright © 2007, Oracle. All rights reserved.

HAVING 절을 사용하여 그룹 결과 제한

HAVING 절을 사용하여 표시할 그룹을 지정하면 집계 정보를 기초로 그룹을 추가로 제한할 수 있습니다.

구문에서 *group_condition*은 반환되는 행 그룹을 지정된 조건이 참인 그룹으로 제한합니다. 유저가 HAVING 절을 사용하는 경우 Oracle 서버는 다음 단계를 수행합니다.

1. 행이 그룹화됩니다.
2. 그룹 함수가 그룹에 적용됩니다.
3. HAVING 절의 조건과 일치하는 그룹이 표시됩니다.

HAVING 절이 GROUP BY 절 앞에 올 수도 있지만, GROUP BY 절을 앞에 배치하는 것이 더 논리적이므로 권장됩니다. HAVING 절이 SELECT 리스트의 그룹에 적용되기 전에 그룹이 형성되고 그룹 함수가 계산됩니다.

HAVING 절 사용

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING      MAX(salary) >10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12000
4	80	11000

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

HAVING 절 사용

슬라이드의 예제에서는 최고 급여가 \$10,000가 넘는 부서에 대해 부서 번호와 최고 급여를 표시합니다.

SELECT 리스트에 그룹 함수를 사용하지 않은 경우에도 GROUP BY 절을 사용할 수 있습니다. 그룹 함수의 결과를 기반으로 행을 제한하는 경우 GROUP BY 절은 물론 HAVING 절도 있어야 합니다.

다음 예제는 최고 급여가 \$10,000가 넘는 부서에 대해 부서 번호와 평균 급여를 표시합니다.

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY    department_id
HAVING      max(salary) >10000 ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333...
2	20	9500
3	110	10150
4	80	10033.333333333333...

HAVING 절 사용

```
SELECT      job_id, SUM(salary) PAYROLL  
FROM        employees  
WHERE       job_id NOT LIKE '%REP%'  
GROUP BY    job_id  
HAVING     SUM(salary) > 13000  
ORDER BY    SUM(salary);
```

JOB_ID	PAYROLL
1 IT_PROG	19200
2 AD_PRES	24000
3 AD_VP	34000

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

HAVING 절 사용(계속)

슬라이드의 예제에서는 총 급여가 \$13,000가 넘는 각 직무에 대해 직무 ID와 월급 총액을 표시합니다. 이 예제에서는 판매 담당자를 제외시키고 월급 총액별로 리스트를 정렬합니다.

단원 내용

- 그룹 함수:
 - 유형 및 구문
 - **AVG, SUM, MIN, MAX, COUNT** 사용
 - 그룹 함수 내에 **DISTINCT** 키워드 사용
 - 그룹 함수의 **NULL** 값
- 다음과 같은 방법을 사용하여 행을 그룹화합니다.
 - **GROUP BY** 절
 - **HAVING** 절
- 그룹 함수 중첩

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

그룹 함수 중첩

최고 평균 급여를 표시합니다.

```
SELECT MAX(AVG(salary))  
FROM employees  
GROUP BY department_id;
```

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

그룹 함수 중첩

그룹 함수는 두 함수 깊이까지 중첩될 수 있습니다. 슬라이드의 예제에서는 각 department_id에 대해 평균 급여를 계산하고 최고 평균 급여를 표시합니다.

그룹 함수를 중첩하는 경우 반드시 GROUP BY 절을 사용해야 합니다.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 그룹 함수 **COUNT, MAX, MIN, SUM** 및 **AVG** 사용
- GROUP BY** 절을 사용하는 **query** 작성
- HAVING** 절을 사용하는 **query** 작성

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

SQL에서 사용할 수 있는 그룹 함수는 다음과 같이 여러 가지가 있습니다.

AVG, COUNT, MAX, MIN, SUM, STDDEV, VARIANCE

GROUP BY 절을 사용하여 하위 그룹을 생성할 수 있습니다. 추가적으로 HAVING 절을 사용하여 그룹을 제한할 수 있습니다.

명령문에서 HAVING 및 GROUP BY 절은 WHERE 절 뒤에 배치합니다. WHERE 절 다음에 나오는 GROUP BY 및 HAVING 절의 순서는 중요하지 않습니다. ORDER BY 절을 끝에 배치합니다.

Oracle 서버는 다음과 같은 순서로 절을 평가합니다.

- 명령문에 WHERE 절이 포함된 경우 후보 행을 설정합니다.
- GROUP BY 절에 지정된 그룹을 식별합니다.
- HAVING 절은 HAVING 절의 그룹 조건을 충족하지 않는 결과 그룹을 추가로 제한합니다.

주: 그룹 함수의 전체 리스트는 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*을 참조하십시오.

연습 5: 개요

이 연습에서는 다음 내용을 다룹니다.

- 그룹 함수를 사용하는 **query** 작성
- 둘 이상의 결과를 나타내도록 행별로 그룹화
- **HAVING** 절을 사용하여 그룹 제한



Copyright © 2007, Oracle. All rights reserved.

연습 5: 개요

이 연습을 마친 후에는 그룹 함수 사용과 데이터 그룹 선택에 능숙해져야 합니다.

연습 5

다음 세 명령문의 유효성을 판별합니다. 참 또는 거짓에 O표 하십시오.

- 그룹 함수는 다수 행에 대해 실행되어 그룹당 하나의 결과를 산출합니다.
참/거짓

- 그룹 함수는 계산에 null을 포함시킵니다.

참/거짓

- WHERE 절은 그룹 계산에 포함시키기 전에 행을 제한합니다.

참/거짓

HR 부서에서 다음 보고서를 요구합니다.

- 모든 사원의 최고, 최저, 합계 및 평균 급여를 찾습니다. 열 레이블을 각각 Maximum, Minimum, Sum, 및 Average로 지정합니다. 결과를 가장 가까운 정수로 반올림합니다. SQL 문을 lab_05_04.sql로 저장합니다. query를 실행합니다.

	Maximum	Minimum	Sum	Average
1	24000	2500	175500	8775

- 각 직무 유형에 대해 최소, 최대, 합계 및 평균 급여를 표시하도록 lab_05_04.sql의 query를 수정합니다. lab_05_04.sql을 lab_05_05.sql로 다시 저장합니다. lab_05_05.sql의 명령문을 실행합니다.

	JOB_ID	Maximum	Minimum	Sum	Average
1	IT_PROG	9000	4200	19200	6400
2	AC_MGR	12000	12000	12000	12000
3	AC_ACCOUNT	8300	8300	8300	8300
4	ST_MAN	5800	5800	5800	5800
5	AD_ASST	4400	4400	4400	4400
6	AD_VP	17000	17000	34000	17000
7	SA_MAN	10500	10500	10500	10500
8	MK_MAN	13000	13000	13000	13000
9	AD_PRES	24000	24000	24000	24000
10	SA_REP	11000	7000	26600	8867
11	MK_REP	6000	6000	6000	6000
12	ST_CLERK	3500	2500	11700	2925

연습 5(계속)

6. 동일한 직무를 수행하는 사람 수를 표시하는 query를 작성합니다.

JOB_ID	COUNT(*)
1 AC_ACCOUNT	1
2 AC_MGR	1
3 AD_ASST	1
4 AD_PRES	1
5 AD_VP	2
6 IT_PROG	3
7 MK_MAN	1
8 MK_REP	1
9 SA_MAN	1
10 SA_REP	3
11 ST_CLERK	4
12 ST_MAN	1

HR 부서의 유저에게 직무를 입력하는 프롬프트를 표시하도록 query를 일반화합니다. 이 스크립트를 lab_05_06.sql이라는 파일에 저장합니다. query를 실행합니다. 프롬프트가 나타나면 IT_PROG를 입력합니다.

JOB_ID	COUNT(*)
1 IT_PROG	3

7. 관리자를 나열하지 않는 채로 관리자 수를 확인합니다. 열 레이블을 Number of Managers로 지정합니다. 힌트: MANAGER_ID 열을 사용하여 관리자 수를 확인합니다.

Number of Managers
1 8

8. 최고 급여와 최저 급여의 차이를 알아냅니다. 열 레이블을 DIFFERENCE로 지정합니다.

DIFFERENCE
1 21500

연습 5(계속)

시간 여유가 있을 경우 다음 연습을 완료하십시오.

9. 관리자 번호 및 해당 관리자의 부하 사원 중 최저 급여를 받는 사원의 급여를 표시하는 보고서를 작성합니다. 관리자가 알려져 있지 않은 모든 사원을 제외합니다. 최소 급여가 \$6,000 이하인 그룹을 제외시킵니다. 급여의 내림차순으로 출력을 정렬합니다.

	MANAGER_ID	MIN(SALARY)
1	102	9000
2	205	8300
3	149	7000

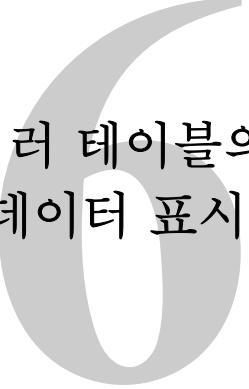
심화 연습에 도전하려면 다음 연습을 완료하십시오.

10. 사원의 총 수와 1995년, 1996년, 1997년, 1998년에 채용된 사원의 수를 표시하는 query를 작성합니다. 적절한 열 머리글을 지정하십시오.

	TOTAL	1995	1996	1997	1998
1	20	1	2	2	3

11. 부서 20, 50, 80 및 90에 대해 직무, 부서 번호별 해당 직무에 대한 급여 및 해당 직무에 대한 총 급여를 표시하고 각 열에 적절한 머리글을 지정하기 위한 행렬 query를 작성합니다.

Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
1 IT_PROG	(null)	(null)	(null)	(null)	19200
2 AC_MGR	(null)	(null)	(null)	(null)	12000
3 AC_ACCOUNT	(null)	(null)	(null)	(null)	8300
4 ST_MAN	(null)	5800	(null)	(null)	5800
5 AD_ASST	(null)	(null)	(null)	(null)	4400
6 AD_VP	(null)	(null)	(null)	34000	34000
7 SA_MAN	(null)	(null)	10500	(null)	10500
8 MK_MAN	13000	(null)	(null)	(null)	13000
9 AD_PRES	(null)	(null)	(null)	24000	24000
10 SA REP	(null)	(null)	19600	(null)	26600
11 MK REP	6000	(null)	(null)	(null)	6000
12 ST_CLERK	(null)	11700	(null)	(null)	11700



여러 테이블의 데이터 표시

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **Equijoin** 및 **nonequijoin**을 사용하여 두 개 이상의 테이블에서 데이터에 액세스하는 **SELECT** 문 작성
- **Self-join**을 사용하여 테이블을 자체 조인
- **Outer join**을 사용하여 일반적으로 조인 조건을 충족하지 않는 데이터 보기
- 두 개 이상의 테이블에서 모든 행의 **Cartesian Product** 생성



Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 두 개 이상의 테이블에서 데이터를 가져오는 방법에 대해 설명합니다. 조인은 여러 테이블의 정보를 보는 데 사용됩니다. 따라서 테이블을 조인하여 두 개 이상의 테이블에 있는 정보를 볼 수 있습니다.

주: 조인에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 *SQL Queries and Subqueries: Joins* 관련 섹션을 참조하십시오.

단원 내용

- 조인 유형 및 구문
- Natural join:
 - USING 절
 - ON 절
- Self-join
- Nonequijoin
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

여러 테이블에서 데이터 가져오기

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
...			
18	202	Fay	20
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
5	144	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

여러 테이블에서 데이터 가져오기

때때로 두 개 이상의 테이블에서 데이터를 사용해야 할 경우가 있습니다. 슬라이드 예제에서는 별도의 두 테이블에서 가져온 데이터가 보고서에 표시됩니다.

- 사원 ID는 EMPLOYEES 테이블에 있습니다.
- 부서 ID는 EMPLOYEES 테이블과 DEPARTMENTS 테이블에 모두 있습니다.
- 부서 이름은 DEPARTMENTS 테이블에 있습니다.

이 보고서를 작성하려면 EMPLOYEES 및 DEPARTMENTS 테이블을 연결하고 두 테이블에서 데이터에 액세스해야 합니다.

조인 유형

SQL:1999 표준과 호환되는 조인에는 다음이 포함됩니다.

- **Natural join:**
 - NATURAL JOIN 절
 - USING 절
 - ON 절
- **Outer join:**
 - LEFT OUTER JOIN
 - RIGHT OUTER JOIN
 - FULL OUTER JOIN
- **Cross join**

ORACLE

Copyright © 2007, Oracle. All rights reserved.

조인 유형

테이블을 조인하려면 SQL:1999 표준과 호환되는 조인 구문을 사용합니다.

주: Oracle9i 릴리스 이전에는 조인 구문이 ANSI (American National Standards Institute) 표준과 달랐습니다. SQL:1999 호환 조인 구문은 이전 릴리스에서 제공하던 Oracle 고유의 조인 구문에 성능상의 이점을 제공하지 않습니다. 고유 조인 구문에 대한 자세한 내용은 부록 C: Oracle 조인 구문을 참조하십시오.

주: 다음 슬라이드에서는 SQL:1999 조인 구문에 대해 설명합니다.

SQL:1999 구문을 사용한 테이블 조인

조인을 사용하여 둘 이상의 테이블에서 데이터를 **query**합니다.

```
SELECT    table1.column, table2.column
FROM      table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)] |
[CROSS JOIN table2];
```

Copyright © 2007, Oracle. All rights reserved.

SQL:1999 구문을 사용한 테이블 조인

구문 설명:

*Table1.column*은 데이터가 검색되는 테이블과 열을 나타냅니다.

*NATURAL JOIN*은 동일한 열 이름을 기반으로 두 테이블을 조인합니다.

*JOIN table2 USING column_name*은 열 이름을 기반으로 Equijoin을 수행합니다.

*JOIN table2 ON table1.column_name = table2.column_name*은 ON 절의 조건을 기반으로 Equijoin을 수행합니다.

*LEFT/RIGHT/FULL OUTER*는 Outer join을 수행하는 데 사용됩니다.

*CROSS JOIN*은 두 테이블에서 Cartesian product를 반환합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*의 *SELECT* 섹션을 참조하십시오.

모호한 열 이름 한정

- 테이블 접두어를 사용하여 여러 테이블에 있는 열 이름을 한정합니다.
- 테이블 접두어를 사용하여 성능을 향상시킵니다.
- 전체 테이블 이름 접두어 대신 테이블 **alias**를 사용합니다.
- 테이블 **alias**로 테이블에 짧은 이름을 지정합니다.
 - SQL 코드 크기를 줄여 메모리를 적게 사용합니다.
- 열 **alias**를 사용하여 이름은 같지만 서로 다른 테이블에 상주하는 열을 구분합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

모호한 열 이름 한정

두 개 이상의 테이블을 조인하는 경우 모호성을 피하기 위해 열 이름을 테이블 이름으로 한정해야 합니다. 테이블 접두어를 사용하지 않는 경우 SELECT 리스트의 DEPARTMENT_ID 열을 DEPARTMENTS 테이블 또는 EMPLOYEES 테이블에서 가져올 수 있습니다. query를 실행하려면 테이블 접두어를 추가해야 합니다. 두 테이블 간에 공통되는 열 이름이 없으면 열을 한정할 필요가 없습니다. 하지만 테이블 접두어를 사용하면 Oracle 서버에 열을 찾을 위치를 정확히 알려줄 수 있으므로 성능이 향상됩니다.

그러나 테이블 이름으로 열 이름을 한정하는 것은 시간이 많이 소요될 수 있으며 테이블 이름이 길 경우 더욱 그렇습니다. 이러한 경우 테이블 alias를 사용할 수 있습니다. 열 alias로 열에 다른 이름을 지정할 수 있는 것처럼 테이블에도 테이블 alias로 다른 이름을 지정합니다. 테이블 alias를 사용하면 SQL 코드를 더 쉽게 유지할 수 있으므로 메모리 사용량도 줄어듭니다.

테이블의 전체 이름을 지정하고 그 뒤에 공백과 테이블 alias를 차례로 배치합니다. 예를 들어 EMPLOYEES 테이블에는 alias e를 부여하고 DEPARTMENTS 테이블에는 alias d를 부여할 수 있습니다.

모호한 열 이름 한정(계속)

지침

- 테이블 alias는 30자까지 사용할 수 있지만 길이는 짧을수록 좋습니다.
- FROM 절의 특정 테이블 이름에 대해 테이블 alias가 사용될 경우 SELECT 문 전체에서 테이블 이름 대신 테이블 alias를 사용해야 합니다.
- 테이블 alias는 의미 있는 단어여야 합니다.
- 테이블 alias는 현재 SELECT 문에 대해서만 유효합니다.

단원 내용

- 조인 유형 및 구문
- **Natural join:**
 - USING 절
 - ON 절
- Self-join
- Nonequijoin
- OUTER join:
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- Cartesian product
 - Cross join

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Natural Join 생성

- **NATURAL JOIN** 절은 두 테이블에서 이름이 같은 모든 열을 기반으로 합니다.
- 이 절은 두 테이블에서 대응되는 모든 열의 값이 동일한 행을 선택합니다.
- 동일한 이름을 가진 열이 서로 다른 데이터 유형을 가지면 오류가 반환됩니다.



Copyright © 2007, Oracle. All rights reserved.

Natural Join 생성

두 테이블에서 데이터 유형과 이름이 일치하는 열을 기반으로 자동으로 테이블을 조인할 수 있습니다. 이 작업은 NATURAL JOIN 키워드를 사용하여 수행합니다.

주: 조인은 두 테이블의 이름과 데이터 유형이 동일한 열에서만 발생합니다. 열 이름은 같지만 데이터 유형이 다를 경우 NATURAL JOIN 구문에서 오류가 발생합니다.

Natural join으로 레코드 검색

```
SELECT department_id, department_name,
       location_id, city
  FROM departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Natural join으로 레코드 검색

슬라이드의 예제에서 LOCATIONS 테이블은 두 테이블에서 유일하게 이름이 같은 열인 LOCATION_ID 열에 의해 DEPARTMENT 테이블에 조인됩니다. 공통되는 다른 열이 있을 경우 모두 조인에 사용됩니다.

WHERE 절을 사용하는 Natural join

Natural join에 대한 추가적인 제한은 WHERE 절을 사용하여 구현됩니다. 다음 예제는 출력 행을 부서 ID가 20 또는 50인 행으로 제한합니다.

```
SELECT department_id, department_name,
       location_id, city
  FROM departments
NATURAL JOIN locations
 WHERE department_id IN (20, 50);
```

USING 절을 사용하여 조인 생성

- 여러 열이 이름은 같지만 데이터 유형이 일치하지 않을 경우 **USING** 절을 사용하여 **equijoin**에 사용될 열을 지정하도록 **natural join**을 적용할 수 있습니다.
- **USING** 절을 사용하면 두 개 이상의 열이 일치하는 경우 하나의 열만 일치하도록 할 수 있습니다.
- **NATURAL JOIN**과 **USING** 절은 상호 배타적입니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

USING 절을 사용하여 조인 생성

Natural join은 이름과 데이터 유형이 대응되는 모든 열을 사용하여 테이블을 조인합니다.
USING 절을 사용하면 equijoin에 사용될 열만 지정할 수 있습니다.

열 이름 조인

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
100	90
101	90
102	90
103	60
104	60
107	60
124	50
141	50
142	50
143	50
144	50
149	80
174	80
176	80

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
1	10 Administration
2	20 Marketing
3	50 Shipping
4	60 IT
5	80 Sales
6	90 Executive
7	110 Accounting
8	190 Contracting

Primary key

Foreign key

ORACLE

Copyright © 2007, Oracle. All rights reserved.

열 이름 조인

사원의 부서 이름을 파악하려면 EMPLOYEES 테이블의 DEPARTMENT_ID 열의 값을 DEPARTMENTS 테이블의 DEPARTMENT_ID 값과 비교합니다. EMPLOYEES 테이블과 DEPARTMENTS 테이블 사이의 관계는 *equijoin*입니다. 즉, 두 테이블의 DEPARTMENT_ID 열에 있는 값이 같아야 합니다. 대개 이러한 유형의 조인에는 Primary key 및 Foreign key가 보완 요소로 포함됩니다.

주: Equijoin은 *simple join* 또는 *inner join*이라고도 합니다.

USING 절을 사용하여 레코드 검색

```
SELECT employee_id, last_name,
       location_id, department_id
  FROM employees JOIN departments
 USING (department_id);
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	124	Mourgos	1500	50
5	144	Vargas	1500	50
6	143	Matos	1500	50
7	142	Davies	1500	50
8	141	Rajs	1500	50
9	107	Lorentz	1400	60
10	104	Ernst	1400	60
...				
19	205	Higgins	1700	110

ORACLE

Copyright © 2007, Oracle. All rights reserved.

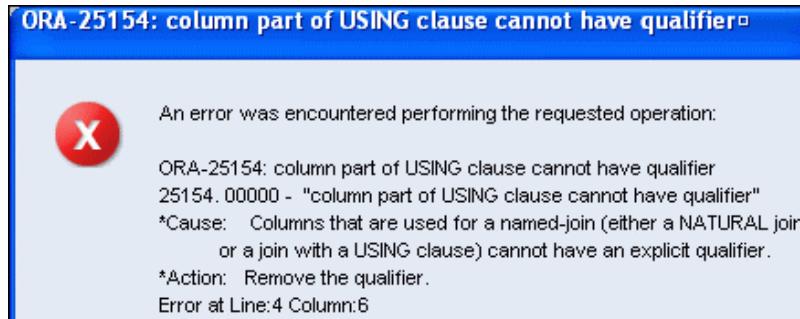
USING 절을 사용하여 레코드 검색

슬라이드의 예제에서는 EMPLOYEES 및 DEPARTMENTS 테이블의 DEPARTMENT_ID 열이 조인되어 사원이 근무하는 부서의 LOCATION_ID가 표시됩니다.

USING 절에 테이블 alias 사용

- USING 절에 사용되는 열을 한정하지 마십시오.
- 동일한 열이 SQL 문의 다른 곳에서 사용되는 경우 alias를 지정하지 마십시오.

```
SELECT l.city, d.department_name
  FROM locations l JOIN departments d
 USING (location_id)
 WHERE d.location_id = 1400;
```



ORACLE

Copyright © 2007, Oracle. All rights reserved.

USING 절에 테이블 Alias 사용

USING 절을 사용하여 조인을 수행할 때 USING 절 자체에서 사용되는 열을 한정할 수 없습니다. 또한 해당 열이 SQL 문의 임의 위치에서 사용되는 경우 alias를 지정할 수 없습니다. 예를 들어 슬라이드에 언급된 query에서 location_id 열이 USING 절에 사용되므로 WHERE 절의 location_id 열에 alias를 지정하면 안됩니다.

USING 절에서 참조되는 열은 SQL 문의 어느 위치에도 수식자(테이블 이름 또는 alias)를 포함해서는 안됩니다. 예를 들어, 다음 명령문은 유효합니다.

```
SELECT l.city, d.department_name
  FROM locations l JOIN departments d USING (location_id)
 WHERE location_id = 1400;
```

USING 절에 사용되지 않지만 두 테이블에 공통인 다른 열에는 테이블 alias를 접두어로 사용해야 하며 그렇지 않으면 "column ambiguously defined" 오류가 발생합니다.

manager_id가 employees 및 departments 테이블에 모두 존재하므로 다음 명령문에서 manager_id에 테이블 alias가 접두어로 사용되지 않으면 "column ambiguously defined" 오류가 발생합니다.

다음 명령문은 유효합니다.

```
SELECT first_name, d.department_name, d.manager_id
  FROM employees e JOIN departments d USING (department_id)
 WHERE department_id = 50;
```

ON 절을 사용하여 조인 생성

- **Natural join**의 조인 조건은 기본적으로 이름이 같은 모든 열의 **equijoin**입니다.
- **ON** 절을 사용하여 임의 조건을 지정하거나 조인할 열을 지정합니다.
- 조인 조건은 다른 검색 조건과는 별개입니다.
- **ON** 절을 사용하면 코드를 이해하기 쉽습니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

ON 절을 사용하여 조인 생성

ON 절을 사용하여 조인 조건을 지정합니다. 이렇게 하면 WHERE 절의 검색 또는 필터 조건과 별도로 조인 조건을 지정할 수 있습니다.

ON 절을 사용하여 레코드 검색

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200 Whalen	10	10	1700
2	201 Hartstein	20	20	1800
3	202 Fay	20	20	1800
4	124 Mourgos	50	50	1500
5	144 Vargas	50	50	1500
6	143 Matos	50	50	1500
7	142 Davies	50	50	1500
8	141 Raji	50	50	1500
9	107 Lorentz	60	60	1400
10	104 Ernst	60	60	1400
...				

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ON 절을 사용하여 레코드 검색

이 예제에서 EMPLOYEES 및 DEPARTMENTS 테이블의 DEPARTMENT_ID 열은 ON 절을 사용하여 조인됩니다. EMPLOYEES 테이블의 부서 ID가 DEPARTMENTS 테이블의 부서 ID와 같은 경우 항상 행이 반환됩니다. 일치하는 column_name을 한정하려면 테이블 alias가 필요합니다.

또한 ON 절을 사용하여 서로 다른 이름을 가진 열을 조인할 수 있습니다. 슬라이드 예제의 (e.department_id = d.department_id)처럼 조인된 열을 괄호로 둘러싸는 것은 선택 사항입니다. 따라서 ON e.department_id = d.department_id도 제대로 작동합니다.

주: SQL Developer에서는 접미어 '_1'을 사용하여 두 department_id를 구분합니다.

ON 절을 사용하여 3-Way 조인 생성

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100 Seattle	Executive
2	101 Seattle	Executive
3	102 Seattle	Executive
4	103 Southlake	IT
5	104 Southlake	IT
6	107 Southlake	IT
7	124 South San Francisco	Shipping
8	141 South San Francisco	Shipping

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ON 절을 사용하여 3-Way 조인 생성

3-way 조인이란 세 개의 테이블을 조인하는 것을 말합니다. SQL:1999 호환 구문에서 조인은 왼쪽에서 오른쪽으로 수행됩니다. 따라서 가장 먼저 수행되는 조인은 EMPLOYEES JOIN DEPARTMENTS입니다. 첫번째 조인 조건은 EMPLOYEES 및 DEPARTMENTS의 열을 참조할 수 있지만 LOCATIONS의 열은 참조할 수 없습니다. 두번째 조인 조건은 세 개의 테이블 모두에서 열을 참조할 수 있습니다.

주: 슬라이드의 코드 예제는 USING 절을 사용하여 작성할 수도 있습니다.

```
SELECT e.employee_id, l.city, d.department_name
FROM employees e
JOIN departments d
USING (department_id)
JOIN locations l
USING (location_id)
```

조인에 추가 조건 적용

AND 절 또는 **WHERE** 절을 사용하여 추가 조건을 적용합니다.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id)
   AND e.manager_id = 149 ;
```

또는

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
  FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id)
   WHERE e.manager_id = 149 ;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

조인에 추가 조건 적용

조인에 추가 조건을 적용할 수 있습니다.

위의 예제는 EMPLOYEES 및 DEPARTMENTS 테이블에서 조인을 수행하고 관리자 ID가 149인 사원만 표시합니다. ON 절에 다른 조건을 추가하려면 AND 절을 사용하면 됩니다. 또는 WHERE 절을 사용하여 추가 조건을 적용할 수 있습니다.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	Abel	80	80	2500
2	Taylor	80	80	2500

단원 내용

- 조인 유형 및 구문
- **Natural join:**
 - USING 절
 - ON 절
- **Self-join**
- Nonequijoin
- **OUTER join:**
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- **Cartesian product**
 - Cross join

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 자체 조인

EMPLOYEES (WORKER)

	EMPLOYEE_ID	LAST_NAME	MANAGER_ID
1	100	King	(null)
2	101	Kochhar	100
3	102	De Haan	100
4	103	Hunold	102
5	104	Ernst	103
6	107	Lorentz	103
7	124	Mourgos	100
8	141	Rajs	124
9	142	Davies	124
10	143	Matos	124

EMPLOYEES (MANAGER)

	EMPLOYEE_ID	LAST_NAME
	100	King
	101	Kochhar
	102	De Haan
	103	Hunold
	104	Ernst
	107	Lorentz
	124	Mourgos
	141	Rajs
	142	Davies
	143	Matos

...



...

WORKER 테이블의 MANAGER_ID는 MANAGER 테이블의 EMPLOYEE_ID와 같습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 자체 조인

때로 테이블을 자체 조인해야 할 경우가 있습니다. 각 사원의 관리자 이름을 찾으려면 EMPLOYEES 테이블을 자체 조인하거나 self-join을 수행해야 합니다. 예를 들어, Lorentz의 관리자 이름을 찾으려면 다음을 수행해야 합니다.

- EMPLOYEES 테이블에서 LAST_NAME 열을 조사하여 Lorentz를 찾습니다.
- MANAGER_ID 열을 조사하여 Lorentz의 관리자 번호를 찾습니다. Lorentz의 관리자 번호는 103입니다.
- LAST_NAME 열을 조사하여 EMPLOYEE_ID가 103인 관리자의 이름을 찾습니다. Hunold의 사원 번호가 103이므로 Hunold가 Lorentz의 관리자입니다.

이 과정에서 EMPLOYEES 테이블을 두 번 조사하게 됩니다. 먼저 테이블을 조사하여 LAST_NAME 열에서 Lorentz를 찾고 MANAGER_ID 값 103을 찾습니다. 이어서 EMPLOYEE_ID 열을 조사하여 103을 찾고 LAST_NAME 열에서 Hunold를 찾습니다.

ON 절을 사용하는 Self-join

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King
8	Kochhar	King
9	Hartstein	King
10	De Haan	King

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ON 절을 사용하는 Self-join

ON 절은 동일한 테이블이나 다른 테이블에서 서로 다른 이름을 가진 열을 조인하는 데도 사용할 수 있습니다.

위의 예제는 EMPLOYEE_ID 및 MANAGER_ID 열을 기반으로 하는 EMPLOYEES 테이블의 Self-join입니다.

주: 슬라이드 예제의 (e.manager_id = m.employee_id)처럼 조인된 열을 팔호로 둘러싸는 것은 선택 사항입니다. 따라서 ON e.manager_id = m.employee_id도 제대로 작동합니다.

단원 내용

- 조인 유형 및 구문
- **Natural join:**
 - USING 절
 - ON 절
- **Self-join**
- **Nonequijoin**
- **OUTER join:**
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- **Cartesian product**
 - Cross join

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Nonequijoin

EMPLOYEES

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hunold	9000
5	Ernst	6000
6	Lorentz	4200
7	Mourgos	5800
8	Rajs	3500
9	Davies	3100
10	Matos	2600
...		
19	Higgins	12000
20	Gietz	8300

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

JOB_GRADES 테이블에서는 각 **GRADE_LEVEL**에 대해 **LOWEST_SAL** 및 **HIGHEST_SAL** 값의 범위를 정의합니다. 따라서 **GRADE_LEVEL** 열은 각 직원에게 등급을 지정하는 데 사용할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Nonequijoin

Nonequijoin은 등호 연산자가 아닌 다른 연산자를 포함하는 조인 조건입니다.

Nonequijoin의 한 예로 EMPLOYEES 테이블과 JOB_GRADES 테이블 사이의 관계를 들 수 있습니다. EMPLOYEES 테이블의 SALARY 열은 JOB_GRADES 테이블의 LOWEST_SAL 열에 있는 값과 HIGHEST_SAL 열에 있는 값 사이의 범위에 해당하는 값을 가집니다. 따라서 급여를 기준으로 각 사원의 등급을 지정할 수 있습니다. 관계는 등호(=) 연산자가 아닌 다른 연산자를 사용하여 구합니다.

Nonequijoin으로 레코드 검색

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
      BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Nonequijoin으로 레코드 검색

슬라이드의 예제에서는 사원의 급여 등급을 평가하는 nonequijoin을 생성합니다. 급여는 낮은 급여 범위와 높은 급여 범위 쌍 사이에 있어야 합니다.

이 query가 실행될 때 각 사원은 한 번만 나타납니다. 리스트에서 반복되는 사원은 없습니다. 여기에는 다음과 같은 두 가지 이유가 있습니다.

- JOB_GRADES 테이블의 어떠한 행도 중복되는 등급을 포함하지 않습니다. 즉, 한 사원에 대한 급여 값은 직무 등급 테이블에서 한 행의 낮은 급여 값과 높은 급여 값 범위에만 속할 수 있습니다.
- 모든 사원의 급여는 직무 등급 테이블에서 제공하는 한계 내에 속합니다. 즉, LOWEST_SAL 열에 포함된 최저값보다 적은 급여를 받거나 HIGHEST_SAL 열에 포함된 최고값보다 많은 급여를 받는 사원은 없습니다.

주: <= 및 >=와 같은 다른 조건을 사용할 수 있지만 BETWEEN이 가장 간단합니다. BETWEEN 조건을 사용할 때는 맨 처음에 낮은 값을 지정하고 맨 마지막에 높은 값을 지정해야 합니다. Oracle 서버는 BETWEEN 조건을 AND 조건 쌍으로 변환합니다. 따라서 BETWEEN을 사용해도 성능상 이점이 전혀 없으므로 논리적으로 간결한 SQL 문을 작성하는 경우에만 BETWEEN을 사용해야 합니다.

슬라이드의 예제에서는 모호성을 피하기 위해서가 아니라 성능상의 이유로 테이블 alias를 지정했습니다.

단원 내용

- 조인 유형 및 구문
- **Natural join:**
 - USING 절
 - ON 절
- **Self-join**
- **Nonequijoin**
- **OUTER join:**
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- **Cartesian product**
 - Cross join

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Outer Join으로 직접 일치되지 않는 레코드 반환

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
1	90 King
2	90 Kochhar
3	90 De Haan
4	60 Hunold
5	60 Ernst
6	60 Lorentz
7	50 Mourgos
8	50 Rajs
9	50 Davies
10	50 Matos
...	
19	110 Higgins
20	110 Gietz

부서 190에는 사원이 없습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Outer join으로 직접 일치되지 않는 레코드 반환

조인 조건을 충족하지 못하는 행은 query 결과에 나타나지 않습니다. 예를 들어, EMPLOYEES 테이블과 DEPARTMENTS 테이블의 equijoin 조건에서 부서 ID 190은 나타나지 않습니다. EMPLOYEES 테이블에 해당 부서 ID가 기록된 사원이 없기 때문입니다. 그러므로 결과 집합에는 20명의 사원을 나타내는 20개의 레코드 대신 19개의 레코드가 표시됩니다. 사원이 없는 부서 레코드를 반환하려면 outer join을 사용하면 됩니다.

INNER join과 OUTER join 비교

- SQL:1999에서는 일치하는 행만 반환하는 두 테이블의 조인을 **Inner join**이라고 합니다.
- **Inner join**의 결과는 물론, 왼쪽(또는 오른쪽) 테이블의 일치하지 않는 행도 반환하는 두 테이블 간의 조인을 **Left(또는 Right) outer join**이라고 합니다.
- **Inner join**의 결과는 물론, **Left** 및 **Right outer join**의 결과를 반환하는 두 테이블 간의 조인을 **Full outer join**이라고 합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

INNER join과 OUTER join 비교

NATURAL JOIN, USING 또는 ON 절을 사용하여 테이블을 조인하면 결과는 inner join이 됩니다. 일치하지 않는 행은 출력에 표시되지 않습니다. 일치하지 않는 행을 반환하려면 outer join을 사용하면 됩니다. Outer Join은 조인 조건을 만족하는 모든 행을 반환하며, 한 테이블의 행이 다른 테이블의 어떤 행과도 조인 조건을 만족하지 않는 경우 테이블의 일부 또는 전체 행을 반환합니다.

Outer join에는 다음 세 가지 유형이 있습니다.

- LEFT OUTER
- RIGHT OUTER
- FULL OUTER

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e LEFT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Vargas	50	Shipping
Matos	50	Shipping
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant	(null)	(null)

ORACLE

Copyright © 2007, Oracle. All rights reserved.

LEFT OUTER JOIN

- query에서는 DEPARTMENTS 테이블에 일치하는 행이 없어도 왼쪽 테이블인 EMPLOYEES 테이블의 모든 행을 검색합니다.

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e RIGHT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
...		

19 Taylor	80 Sales
20 Grant	(null) (null)
21 (null)	190 Contracting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

RIGHT OUTER JOIN

- query에서는 EMPLOYEES 테이블에 일치하는 행이 없어도 오른쪽 테이블인 DEPARTMENTS 테이블의 모든 행을 검색합니다.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name  
FROM employees e FULL OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1 Whalen	10	Administration
2 Hartstein	20	Marketing
3 Fay	20	Marketing
4 Higgins	110	Accounting
...		

19 Taylor	80	Sales
20 Grant	(null)	(null)
21 (null)	190	Contracting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

FULL OUTER JOIN

- query에서는 DEPARTMENTS 테이블에 일치하는 행이 없어도 EMPLOYEES 테이블의 모든 행을 검색합니다. 또한 EMPLOYEES 테이블에 일치하는 행이 없어도 DEPARTMENTS 테이블의 모든 행을 검색합니다.

단원 내용

- 조인 유형 및 구문
- **Natural join:**
 - USING 절
 - ON 절
- **Self-join**
- **Nonequijoin**
- **OUTER join:**
 - LEFT OUTER join
 - RIGHT OUTER join
 - FULL OUTER join
- **Cartesian product**
 - Cross join

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Cartesian Product

- 다음과 같은 경우에 **Cartesian Product**가 형성됩니다.
 - 조인 조건이 생략된 경우
 - 조인 조건이 잘못된 경우
 - 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인되는 경우
- **Cartesian Product**를 피하려면 항상 유효한 조인 조건을 포함하십시오.



Copyright © 2007, Oracle. All rights reserved.

Cartesian Product

조인 조건이 잘못되거나 완전히 생략된 경우 결과는 모든 행 조합이 표시되는 *Cartesian Product*로 나타납니다. 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인됩니다.

Cartesian Product는 많은 행을 생성하므로 결과는 그다지 유용하지 않습니다. 따라서 특별히 모든 테이블에서 모든 행을 조합해야 하는 경우가 아니면 항상 유효한 조인 조건을 포함해야 합니다.

그러나 Cartesian Product는 일부 테스트에서 적당량의 데이터를 시뮬레이트하기 위해 많은 행을 생성해야 하는 경우에 유용합니다.

Cartesian Product 생성

EMPLOYEES (20 행)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	100	King	90
2	101	Kochhar	90
3	102	De Haan	90
4	103	Hunold	60
...			
19	205	Higgins	110
20	206	Gietz	110

DEPARTMENTS (8 행)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian Product:
 $20 \times 8 = 160$ 행

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	100	90	1700
2	101	90	1700
3	102	90	1700
4	103	60	1700
...			
159	205	110	1700
160	206	110	1700

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Cartesian Product 생성

조인 조건을 생략하면 Cartesian Product가 생성됩니다. 슬라이드의 예제는 EMPLOYEES 및 DEPARTMENTS 테이블에서 가져온 사원의 성과 부서 이름을 표시합니다. 조인 조건이 지정되지 않았기 때문에 EMPLOYEES 테이블의 모든 행(20행)이 DEPARTMENTS 테이블의 모든 행(8행)과 조인되며, 따라서 출력에 160행이 생성됩니다.

Cross Join 생성

- **CROSS JOIN** 절은 두 테이블의 교차 곱을 생성합니다.
- 이를 두 테이블 간의 **Cartesian Product**라고도 합니다.

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration
...		
159	Whalen	Contracting
160	Zlotkey	Contracting

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Cross Join 생성

슬라이드의 예제는 EMPLOYEES 및 DEPARTMENTS 테이블의 Cartesian Product를 생성합니다.

요약

이 단원에서는 다음과 같은 조인을 사용하여 여러 테이블의 데이터를 표시하는 방법을 배웠습니다.

- **Equijoin**
- **Nonequijoin**
- **Outer join**
- **Self-join**
- **Cross join**
- **Natural join**
- **Full(또는 양쪽) outer join**

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

다양한 방법으로 테이블을 조인할 수 있습니다.

조인 유형

- Equijoin
- Nonequijoin
- Outer join
- Self-join
- Cross join
- Natural join
- Full(또는 양쪽) outer join

Cartesian Product

Cartesian Product는 결과에 모든 행 조합을 표시합니다. WHERE 절을 생략하거나 CROSS JOIN 절을 지정하여 Cartesian Product를 생성할 수 있습니다.

테이블 alias

- 테이블 alias는 데이터베이스 액세스 속도를 높입니다.
- 테이블 alias를 사용하면 SQL 코드를 더 작게 유지할 수 있으므로 메모리 사용량도 줄어듭니다.
- 열 모호성을 피하기 위해 테이블 alias를 반드시 사용해야 하는 경우도 있습니다.

연습 6: 개요

이 연습에서는 다음 내용을 다룹니다.

- **Equijoin**을 사용하여 테이블 조인
- **Outer join** 및 **Self-join** 수행
- 조건 추가



Copyright © 2007, Oracle. All rights reserved.

연습 6: 개요

이 연습은 SQL:1999 호환 조인을 사용하여 두 개 이상의 테이블에서 데이터를 추출하는 과정을 실습할 수 있도록 구성되었습니다.

연습 6

- HR 부서를 위해 모든 부서의 주소를 생성하는 query를 작성합니다. LOCATIONS 및 COUNTRIES 테이블을 사용합니다. 출력에 번지, 동/리, 구/군, 시/도 및 국가를 표시합니다. NATURAL JOIN을 사용하여 결과를 생성합니다.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The ...	Oxford		United Kingdom

- HR 부서에서 모든 사원에 대한 보고서를 요구합니다. 모든 사원의 성, 부서 번호 및 부서 이름을 표시하는 query를 작성합니다.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping
9	Hunold	60	IT
10	Ernst	60	IT
...			
18	Higgins	110	Accounting
19	Gietz	110	Accounting

연습 6(계속)

3. HR 부서에서 Toronto에 근무하는 사원에 대한 보고서를 요구합니다. Toronto에서 근무하는 모든 사원의 성, 직무, 부서 번호 및 부서 이름을 표시합니다.

LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1 Hartstein	MK_MAN	20	Marketing
2 Fay	MK_REP	20	Marketing

4. 사원의 성 및 사원 번호를 해당 관리자의 성 및 관리자 번호와 함께 표시하는 보고서를 작성합니다. 열 레이블을 각각 Employee, Emp#, Manager 및 Mgr#으로 지정합니다. SQL 문을 lab_06_04.sql로 저장합니다. query를 실행합니다.

Employee	EMP#	Manager	Mgr#
1 Kochhar	101 King	100	
2 De Haan	102 King	100	
3 Hunold	103 De Haan	102	
4 Ernst	104 Hunold	103	
5 Lorentz	107 Hunold	103	
6 Mourgos	124 King	100	
7 Rajs	141 Mourgos	124	
8 Davies	142 Mourgos	124	
9 Matos	143 Mourgos	124	
10 Vargas	144 Mourgos	124	

...

15 Whalen	200 Kochhar	101
16 Hartstein	201 King	100
17 Fay	202 Hartstein	201
18 Higgins	205 Kochhar	101
19 Gietz	206 Higgins	205

연습 6(계속)

5. King을 비롯하여 해당 관리자가 지정되지 않은 모든 사원을 표시하도록 lab_06_04.sql 을 수정합니다. 사원 번호순으로 결과를 정렬합니다. SQL 문을 lab_06_05.sql로 저장합니다. lab_06_05.sql의 query를 실행합니다.

Employee	EMP#	Manager	Mgr#
1 King	100 (null)	(null)	
2 Kochhar	101 King	100	
3 De Haan	102 King	100	
4 Hunold	103 De Haan	102	
5 Ernst	104 Hunold	103	
6 Lorentz	107 Hunold	103	
7 Mourgos	124 King	100	
8 Rajs	141 Mourgos	124	
9 Davies	142 Mourgos	124	
10 Matos	143 Mourgos	124	
...			
18 Fay	202 Hartstein	201	
19 Higgins	205 Kochhar	101	
20 Gietz	206 Higgins	205	

6. HR 부서를 위해 사원의 성과 부서 번호 및 주어진 사원과 동일한 부서에 근무하는 모든 사원을 표시하는 보고서를 작성합니다. 각 열에 적절한 레이블을 지정합니다. 이 스크립트를 lab_06_06.sql이라는 파일에 저장합니다.

DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20 Fay	Hartstein
2	20 Hartstein	Fay
3	50 Davies	Matos
4	50 Davies	Mourgos
5	50 Davies	Rajs
6	50 Davies	Vargas
7	50 Matos	Davies
8	50 Matos	Mourgos
9	50 Matos	Rajs
10	50 Matos	Vargas
...		
42	110 Higgins	Gietz

연습 6(계속)

7. HR 부서에서 직급 및 급여에 대한 보고서를 요구합니다. JOB_GRADES 테이블에 익숙해지도록 먼저 JOB_GRADES 테이블의 구조를 표시합니다. 그런 다음 모든 사원의 이름, 직무, 부서 이름, 급여 및 등급을 표시하는 query를 작성합니다.

DESC JOB_GRADES		
Name	Null	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

3 rows selected

LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1 Vargas	ST_CLERK	Shipping	2500	A
2 Matos	ST_CLERK	Shipping	2600	A
3 Davies	ST_CLERK	Shipping	3100	B
4 Rajs	ST_CLERK	Shipping	3500	B
5 Lorentz	IT_PROG	IT	4200	B
6 Whalen	AD_ASST	Administration	4400	B
7 Mourgos	ST_MAN	Shipping	5800	B
8 Ernst	IT_PROG	IT	6000	C
9 Fay	MK_REP	Marketing	6000	C
10 Gietz	AC_ACCOUNT	Accounting	8300	C
...				
18 De Haan	AD_VP	Executive	17000	E
19 King	AD_PRES	Executive	24000	E

연습 6(계속)

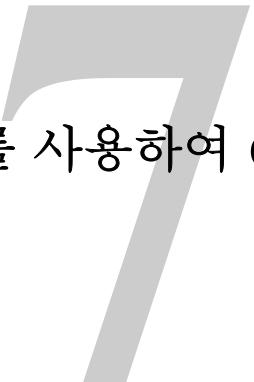
심화 연습에 도전하려면 다음 연습을 완료하십시오.

- HR 부서에서 Davies 이후에 채용된 모든 사원의 이름을 파악하려고 합니다. 사원 Davies 이후로 채용된 모든 사원의 이름과 채용 날짜를 표시하기 위한 query를 작성합니다.

	LAST_NAME	HIRE_DATE
1	Lorentz	07-FEB-99
2	Mourgos	16-NOV-99
3	Matos	15-MAR-98
4	Vargas	09-JUL-98
5	Zlotkey	29-JAN-00
6	Taylor	24-MAR-98
7	Grant	24-MAY-99
8	Fay	17-AUG-97

- HR 부서에서 관리자보다 먼저 채용된 모든 사원의 이름과 채용 날짜 및 해당 관리자의 이름과 채용 날짜를 찾으려고 합니다. 이 스크립트를 lab_06_09.sql이라는 파일에 저장합니다.

	LAST_NAME	HIRE_DATE		LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar		21-SEP-89
2	Hunold	03-JAN-90	De Haan		13-JAN-93
3	Vargas	09-JUL-98	Mourgos		16-NOV-99
4	Matos	15-MAR-98	Mourgos		16-NOV-99
5	Davies	29-JAN-97	Mourgos		16-NOV-99
6	Rajs	17-OCT-95	Mourgos		16-NOV-99
7	Grant	24-MAY-99	Zlotkey		29-JAN-00
8	Taylor	24-MAR-98	Zlotkey		29-JAN-00
9	Abel	11-MAY-96	Zlotkey		29-JAN-00



subquery를 사용하여 query 해결

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **subquery** 정의
- **subquery**가 해결할 수 있는 문제 유형 설명
- **subquery**의 유형 나열
- 단일 행 및 다중 행 **subquery** 작성

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 SELECT 문의 고급 기능에 대해 배웁니다. 다른 SQL 문의 WHERE 절에 subquery를 작성하여 알 수 없는 조건부 값을 기반으로 값을 구할 수 있습니다. 이 단원에서는 단일 행 subquery 및 다중 행 subquery에 대해서도 다룹니다.

단원 내용

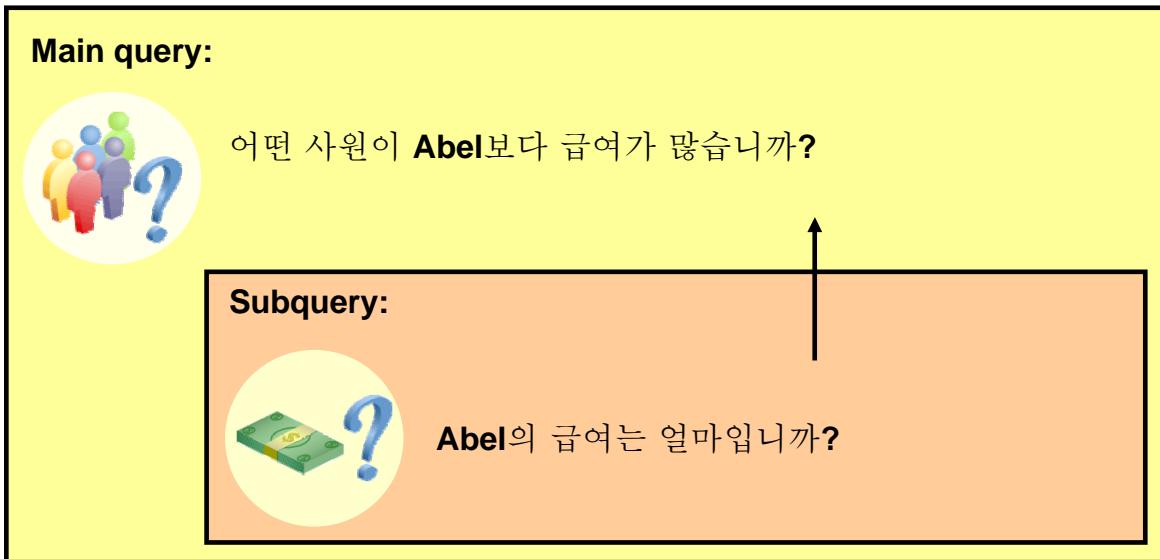
- **Subquery:** 유형, 구문 및 지침
- 단일 행 **subquery:**
 - **subquery**의 그룹 함수
 - **subquery**가 있는 **HAVING** 절
- 다중 행 **subquery**
 - **ALL** 또는 **ANY** 연산자 사용
- **subquery**의 Null 값

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Subquery를 사용하여 문제 해결

Abel보다 급여가 많은 사람은 누구입니까?



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery를 사용하여 문제 해결

Abel보다 급여가 많은 사람을 찾는 query를 작성한다고 가정해 보겠습니다.

이 문제를 해결하려면 두 개의 query가 필요합니다. 하나는 Abel이 받는 급여액을 찾는 query이고 또 하나는 이 액수보다 많은 급여를 받는 사람을 찾는 query입니다.

한 query를 다른 query 내부에 배치하는 방식으로 두 query를 결합하여 이 문제를 해결할 수 있습니다.

inner query(또는 *subquery*)는 outer query(또는 *main query*)에서 사용되는 값을 반환합니다. subquery를 사용하는 것은 두 query를 순차적으로 수행하여 첫번째 query 결과를 두번째 query의 검색 값으로 사용하는 것과 동일한 기능입니다.

Subquery 구문

```
SELECT      select_list  
FROM        table  
WHERE       expr operator  
           (SELECT      select_list  
            FROM       table);
```

- **subquery(inner query)**는 **main query(outer query)** 전에 실행됩니다.
- **subquery** 결과는 **main query**에서 사용됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery 구문

subquery는 다른 SELECT 문의 절에 포함되는 SELECT 문입니다. subquery를 사용하여 단순하면서도 강력한 명령문을 구축할 수 있습니다. subquery는 테이블 자체의 데이터에 종속되는 조건을 사용하여 테이블에서 행을 선택해야 하는 경우에 매우 유용합니다.

다음을 포함하여 다양한 SQL 절에 subquery를 배치할 수 있습니다.

- WHERE 절
- HAVING 절
- FROM 절

구문 설명:

연산자는 >, =, IN 등의 비교 조건을 포함합니다.

주: 비교 조건은 단일 행 연산자(>, =, >=, <, <>, <=)와 다중 행 연산자(IN, ANY, ALL)로 구분됩니다.

subquery는 종종 중첩된 SELECT 문, 하위 SELECT 문 또는 내부 SELECT 문이라고 합니다. 일반적으로 subquery가 먼저 실행되고 그 출력이 main query(또는 outer query)의 query 조건을 완료하는 데 사용됩니다.

Subquery 사용

```
SELECT last_name, salary  
FROM   employees  
WHERE  salary >  
       (SELECT salary  
        FROM   employees  
        WHERE  last_name = 'Abel');
```

	LAST_NAME	SALARY
1	King	24000
2	Kochhar	17000
3	De Haan	17000
4	Hartstein	13000
5	Higgins	12000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery 사용

슬라이드에서 inner query는 사원 Abel의 급여를 판별합니다. Outer query는 inner query의 결과를 가져와서 Abel 사원보다 많은 급여를 받는 모든 사원을 표시합니다.

subquery 사용 지침

- **subquery**는 괄호로 묶습니다.
- 가독성을 위해 비교 조건의 오른쪽에 **subquery**를 배치합니다. 그러나 **subquery**는 비교 연산자의 양쪽 어디에나 사용할 수 있습니다.
- 단일 행 **subquery**에는 단일 행 연산자를 사용하고 다중 행 **subquery**에는 다중 행 연산자를 사용합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

subquery 사용 지침

- subquery는 괄호로 묶어야 합니다.
- 가독성을 위해 비교 조건의 오른쪽에 subquery를 배치합니다. 그러나 subquery는 비교 연산자의 양쪽 어디에나 사용할 수 있습니다.
- subquery에서는 단일 행 연산자와 다중 행 연산자라는 두 가지 유형의 비교 조건을 사용합니다.

Subquery 유형

- 단일 행 subquery



- 다중 행 subquery



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery 유형

- 단일 행 subquery: 내부 SELECT 문에서 하나의 행만 반환하는 query
- 다중 행 subquery: 내부 SELECT 문에서 두 개 이상의 행을 반환하는 query

주: 내부 SELECT 문에서 두 개 이상의 열을 반환하는 다중 열 subquery도 있습니다. 이에 대해서는 *Oracle Database 11g: SQL Fundamentals II* 과정에서 다룹니다.

단원 내용

- **Subquery:** 유형, 구문 및 지침
- 단일 행 **subquery:**
 - **subquery**의 그룹 함수
 - **subquery**가 있는 **HAVING** 절
- 다중 행 **subquery**
 - **ALL** 또는 **ANY** 연산자 사용
- **subquery**의 Null 값

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단일 행 subquery

- 한 행만 반환합니다.
- 단일 행 비교 연산자를 사용합니다.

연산자	의미
=	같음
>	보다 큼
>=	보다 크거나 같음
<	보다 작음
<=	보다 작거나 같음
<>	같지 않음

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단일 행 subquery

단일 행 subquery는 내부 SELECT 문에서 한 행만 반환하는 query입니다. 이러한 유형의 subquery는 단일 행 연산자를 사용합니다. 슬라이드는 단일 행 연산자 리스트를 보여줍니다.

예제:

사원 141의 직무 ID와 동일한 직무 ID를 가진 사원을 표시합니다.

```
SELECT last_name, job_id  
  FROM employees  
 WHERE job_id =  
       (SELECT job_id  
        FROM   employees  
        WHERE employee_id = 141);
```

	LAST_NAME	JOB_ID
1	Rajs	ST_CLERK
2	Davies	ST_CLERK
3	Matos	ST_CLERK
4	Vargas	ST_CLERK

단일 행 subquery 실행

```

SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = SA_REP
       (SELECT job_id
        FROM   employees
        WHERE  last_name = 'Taylor')
AND    salary > 8600
       (SELECT salary
        FROM   employees
        WHERE  last_name = 'Taylor');
  
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

ORACLE

Copyright © 2007, Oracle. All rights reserved.

단일 행 subquery 실행

SELECT 문은 query 블록으로 간주할 수 있습니다. 슬라이드의 예제에서는 "Taylor"와 동일한 일을 하지만 그 보다 급여를 많이 받는 사원을 표시합니다.

이 예제는 세 개의 query 블록(outer query와 두 개의 inner query)으로 구성됩니다. inner query 블록이 먼저 실행되어 각각 SA_REP 및 8600이라는 query 결과를 생성합니다. 그런 다음 outer query 블록이 처리되고 inner query에서 반환된 값을 사용하여 검색 조건을 완성합니다.

두 inner query는 모두 단일 값(각각 SA_REP 및 8600)을 반환하므로 이러한 SQL 문을 단일 행 subquery라고 합니다.

주: outer query와 inner query는 다른 테이블에서 데이터를 가져올 수 있습니다.

Subquery에서 그룹 함수 사용

```

SELECT last_name, job_id, salary
FROM   employees
WHERE  salary = (SELECT MIN(salary)
                  FROM   employees);
    
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery에서 그룹 함수 사용

subquery에서 그룹 함수를 사용하여 단일 행을 반환하는 방식으로 main query에서 데이터를 표시할 수 있습니다. subquery는 괄호로 묶이고 비교 조건 뒤에 배치됩니다.

슬라이드의 예제는 급여가 최저 급여와 같은 모든 사원의 성, 직무 ID 및 급여를 표시합니다. MIN 그룹 함수는 outer query로 단일 값(2500)을 반환합니다.

Subquery가 있는 HAVING 절

- Oracle 서버는 subquery를 먼저 실행합니다.
- Oracle 서버는 main query의 HAVING 절로 결과를 반환합니다.

```

SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > 2500
      (SELECT MIN(salary)
       FROM employees
       WHERE department_id = 50);
  
```

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	90	17000
3	20	6000
...		
7	10	4400

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery가 있는 HAVING 절

WHERE 절뿐만 아니라 HAVING 절에서도 subquery를 사용할 수 있습니다. Oracle 서버가 subquery를 실행하고 그 결과는 main query의 HAVING 절로 반환됩니다.

슬라이드의 SQL 문은 부서 50보다 최저 급여가 많은 모든 부서를 표시합니다.

예제:

평균 급여가 가장 낮은 직무를 찾습니다.

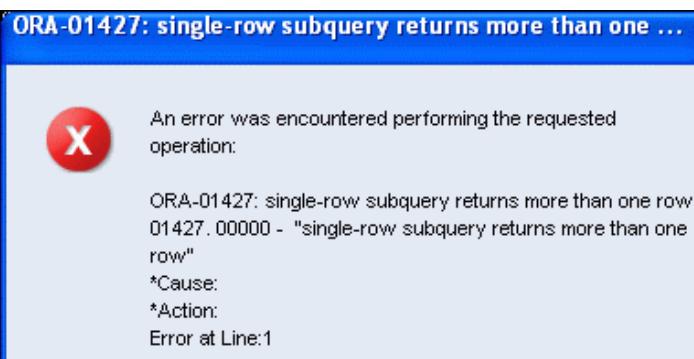
```

SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
                      FROM employees
                      GROUP BY job_id);
  
```

JOB_ID	AVG(SALARY)
ST_CLERK	2925

이 명령문에서 잘못된 점은?

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
      (SELECT MIN(salary)
       FROM employees
       GROUP BY department_id);
```



다중 행 subquery에 단일
행 연산자 사용

ORACLE

Copyright © 2007, Oracle. All rights reserved.

이 명령문에서 잘못된 점은?

subquery의 일반적인 오류는 단일 행 subquery에 대해 두 개 이상의 행이 반환되는 경우에 발생합니다.

슬라이드의 SQL 문에서 subquery는 GROUP BY 절을 포함합니다. 즉, subquery가 찾은 각 그룹에 대해 다중 행을 반환합니다. 이 경우에 subquery의 결과는 4400, 6000, 2500, 4200, 7000, 17000, 8300입니다.

outer query는 이러한 결과를 가져와서 WHERE 절에서 사용합니다. WHERE 절에는 하나의 값만 예상하는 단일 행 비교 연산자인 등호(=) 연산자가 있습니다. = 연산자는 subquery에서 두 개 이상의 값을 받을 수 없으므로 오류가 발생합니다.

이 오류를 해결하려면 = 연산자를 IN으로 바꿔야 합니다.

Inner query에서 반환된 행이 없음

```

SELECT last_name, job_id
FROM   employees
WHERE  job_id =
       (SELECT job_id
        FROM   employees
        WHERE  last_name = 'Haas');

0 rows selected
  
```

"Haas"라는 사원이 없으므로 subquery에서 결과로 반환되는 행이 없습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Inner query에서 반환된 행이 없음

subquery의 일반적인 문제는 inner query에서 행을 반환하지 않는 경우에 발생합니다.

슬라이드의 SQL 문에서 subquery는 WHERE 절을 포함합니다. 아마 이름이 Haas인 사원을 찾으려는 의도일 것입니다. 이 명령문은 구문상 올바르지만 실행될 때 아무 행도 선택하지 않습니다.

Haas라는 이름을 가진 사원이 없기 때문입니다. 따라서 subquery는 행을 반환하지 않습니다. outer query는 subquery의 결과(null)를 가져와서 WHERE 절에서 이 결과를 사용합니다. outer query는 직무 ID가 null인 사원을 찾지 못하고 따라서 행을 반환하지 않습니다. Null 값을 가진 직무가 존재했다면 두 null 값의 비교 결과는 null이기 때문에 행이 반환되지 않고 따라서 WHERE 조건은 참이 아닙니다.

단원 내용

- **Subquery:** 유형, 구문 및 지침
- 단일 행 **subquery:**
 - **subquery**의 그룹 함수
 - **subquery**가 있는 **HAVING** 절
- 다중 행 **subquery**
 - **ALL** 또는 **ANY** 연산자 사용
- **subquery**의 Null 값

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

다중 행 Subquery

- 두 개 이상의 행을 반환합니다.
- 다중 행 비교 연산자를 사용합니다.

연산자	의미
IN	리스트의 임의 멤버와 같음
ANY	$=, !=, >, <, <=, >=$ 연산자가 앞에 있어야 합니다. 값 하나를 리스트의 값 또는 query에서 반환된 값과 각각 비교합니다. query에서 반환된 행이 없으면 FALSE 로 평가됩니다.
ALL	$=, !=, >, <, <=, >=$ 연산자가 앞에 있어야 합니다. 값 하나를 리스트의 모든 값 또는 query에서 반환된 모든 값과 비교합니다. query에서 반환된 행이 없으면 TRUE 로 평가됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다중 행 Subquery

두 개 이상의 행을 반환하는 subquery를 다중 행 subquery라고 합니다. 다중 행 subquery에는 단일 행 연산자 대신 다중 행 연산자를 사용합니다. 다중 행 연산자는 하나 이상의 값을 예상합니다.

```
SELECT last_name, salary, department_id
  FROM employees
 WHERE salary IN (SELECT MIN(salary)
                   FROM employees
                   GROUP BY department_id);
```

예제:

각 부서에서 최저 급여와 동일한 급여를 받는 사원을 찾습니다.

Inner query가 먼저 실행되어 query 결과를 생성합니다. 그런 다음 main query 블록이 처리되고 inner query에서 반환된 값을 사용하여 검색 조건을 완성합니다. 실제로 main query는 Oracle 서버에 다음과 같이 나타납니다.

```
SELECT last_name, salary, department_id
  FROM employees
 WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300,
                  8600, 17000);
```

다중 행 Subquery에서 ANY 연산자 사용

```

SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ANY
       (SELECT salary
        FROM   employees
        WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
  
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400
...				
9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

다중 행 Subquery에서 ANY 연산자 사용

ANY 연산자 및 그 동의어인 SOME 연산자는 값을 subquery에서 반환되는 각 값과 비교합니다. 슬라이드의 예제는 IT 프로그래머가 아닌 사원 중 급여가 IT 프로그래머보다 적은 사원을 표시합니다. 프로그래머가 받는 최고 급여는 \$9,000입니다.

<ANY는 최대값보다 작음을 의미합니다. >ANY는 최소값보다 큼을 의미합니다. =ANY는 IN과 같습니다.

다중 행 Subquery에서 ALL 연산자 사용

```

SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ALL
          (SELECT salary
           FROM   employees
           WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
  
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141 Rajs	ST_CLERK	3500	
2	142 Davies	ST_CLERK	3100	
3	143 Matos	ST_CLERK	2600	
4	144 Vargas	ST_CLERK	2500	

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다중 행 Subquery에서 ALL 연산자 사용

ALL 연산자는 값을 subquery에서 반환되는 모든 값과 비교합니다. 슬라이드의 예제는 직무 ID가 IT_PROG인 모든 사원보다 급여가 적고 직무가 IT_PROG가 아닌 사원을 표시합니다.

>ALL은 최대값보다 큼을 의미하고 <ALL은 최소값보다 작음을 의미합니다.

NOT 연산자는 IN, ANY 및 ALL 연산자와 함께 사용할 수 있습니다.

단원 내용

- **Subquery:** 유형, 구문 및 지침
- 단일 행 **subquery:**
 - **subquery**의 그룹 함수
 - **subquery**가 있는 **HAVING** 절
- 다중 행 **subquery**
 - **ALL** 또는 **ANY** 연산자 사용
- **subquery**의 **Null** 값

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

Subquery의 null 값

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
       (SELECT mgr.manager_id
        FROM   employees mgr);
```

0 rows selected

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery의 null 값

슬라이드의 SQL 문은 부하 직원이 없는 모든 사원을 표시하려고 합니다. 논리적으로 이 SQL 문은 12개의 행을 반환해야 합니다. 그러나 이 SQL 문은 행을 반환하지 않습니다. inner query에서 반환되는 값 중 하나가 null 값이기 때문에 전체 query가 행을 반환하지 않습니다.

왜냐하면 null 값을 비교하는 모든 조건은 결과가 null이기 때문입니다. 따라서 subquery의 결과 집합의 일부가 null 값이 될 것으로 예상되는 경우 NOT IN 연산자를 사용하지 마십시오. NOT IN 연산자는 <> ALL과 같습니다.

IN 연산자를 사용할 경우 subquery의 결과 집합의 일부가 null 값인 것은 문제가 되지 않습니다. IN 연산자는 =ANY와 같습니다. 예를 들어, 부하 직원이 있는 사원을 표시하려면 다음 SQL 문을 사용합니다.

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id  IN
       (SELECT mgr.manager_id
        FROM   employees mgr);
```

Subquery의 null 값(계속)

또는 부하 직원이 없는 모든 사원을 표시하도록 subquery에 WHERE 절을 포함시킬 수 있습니다.

```
SELECT last_name FROM employees
WHERE employee_id NOT IN
      (SELECT manager_id
       FROM   employees
       WHERE  manager_id IS NOT NULL);
```

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- **subquery**가 문제 해결에 도움이 될 수 있는 경우 식별
- **query**가 알 수 없는 값을 기반으로 하는 경우 **subquery** 작성

```
SELECT      select_list
FROM        table
WHERE       expr operator
            (SELECT select_list
             FROM   table);
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

이 단원에서는 subquery 사용법에 대해 설명했습니다. subquery는 다른 SQL 문의 절에 포함된 SELECT 문입니다. subquery는 query가 알 수 없는 매개 값을 가진 검색 조건을 기반으로 하는 경우에 유용합니다.

subquery는 다음과 같은 특성을 가집니다.

- =, <>, >, >=, <, <= 등과 같은 단일 행 연산자를 포함하는 기본 명령문으로 하나의 데이터 행을 전달할 수 있습니다.
- IN과 같은 다중 행 연산자를 포함하는 기본 명령문으로 여러 데이터 행을 전달할 수 있습니다.
- Oracle 서버에서 먼저 처리된 다음 WHERE 또는 HAVING 절에서 결과를 사용합니다.
- 그룹 함수를 포함할 수 있습니다.

연습 7: 개요

이 연습에서는 다음 내용을 다룹니다.

- 알 수 없는 조건을 기반으로 값을 **query**하는 **subquery** 작성
- **subquery**를 사용하여 한 데이터 집합에 있고 다른 데이터 집합에는 없는 값 찾기

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

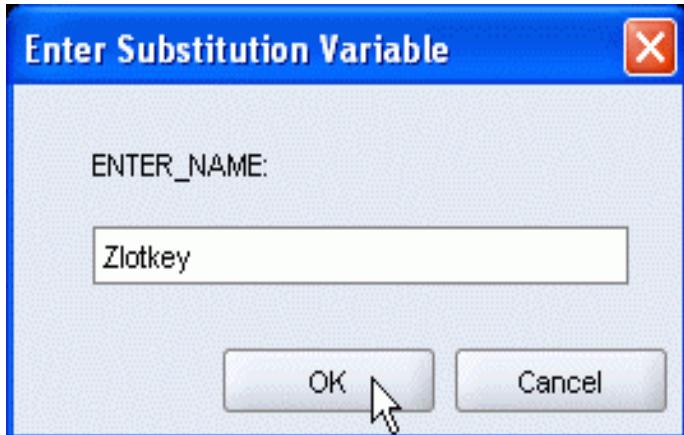
연습 7: 개요

이 연습에서는 중첩된 SELECT 문을 사용하여 복합 query를 작성합니다.

연습 문제의 경우 inner query를 먼저 작성할 수 있습니다. outer query를 코딩하기 전에 inner query를 실행하여 예상되는 데이터가 생성되는지 확인하십시오.

연습 7

1. HR 부서에서 유저에게 사원의 성을 입력하라는 프롬프트를 표시하는 query를 요구합니다. 그런 다음 이 query는 유저가 입력한 이름의 사원과 동일한 부서에서 근무하는 사원의 성과 채용 날짜를 표시합니다(입력한 사원은 제외). 예를 들어, 유저가 Zlotkey를 입력하면 Zlotkey와 함께 근무하는 모든 사원을 찾습니다(Zlotkey는 제외).



	LAST_NAME	HIRE_DATE
1	Abel	11-MAY-96
2	Taylor	24-MAR-98

2. 평균 급여 이상을 받는 모든 사원의 사원 번호, 성 및 급여를 표시하는 보고서를 작성합니다. 급여의 오름차순으로 결과를 정렬합니다.

	EMPLOYEE_ID	LAST_NAME	SALARY
1	103	Hunold	9000
2	149	Zlotkey	10500
3	174	Abel	11000
4	205	Higgins	12000
5	201	Hartstein	13000
6	101	Kochhar	17000
7	102	De Haan	17000
8	100	King	24000

연습 7(계속)

3. 성에 문자 "u"가 포함된 사원과 같은 부서에 근무하는 모든 사원의 사원 번호와 성을 표시하는 query를 작성합니다. 작성한 SQL 문을 lab_07_03.sql로 저장합니다. query를 실행합니다.

	EMPLOYEE_ID	LAST_NAME
1	124	Mourgos
2	141	Rajs
3	142	Davies
4	143	Matos
5	144	Vargas
6	103	Hunold
7	104	Ernst
8	107	Lorentz

4. HR 부서에서 부서 위치 ID가 1700인 모든 사원의 성, 부서 번호 및 직무 ID를 표시하는 보고서를 요구합니다.

	LAST_NAME	DEPARTMENT_ID	JOB_ID
1	Whalen		10 AD_ASST
2	King		90 AD_PRES
3	Kochhar		90 AD_VP
4	De Haan		90 AD_VP
5	Higgins		110 AC_MGR
6	Gietz		110 AC_ACCOUNT

위치 ID를 입력하는 프롬프트를 유저에게 표시하도록 query를 수정합니다. 입력한 내용을 lab_07_04.sql이라는 파일에 저장합니다.

5. HR을 위해 King에게 보고하는 모든 사원의 성과 급여를 표시하는 보고서를 작성합니다.

	LAST_NAME	SALARY
1	Kochhar	17000
2	De Haan	17000
3	Mourgos	5800
4	Zlotkey	10500
5	Hartstein	13000

연습 7(계속)

6. HR을 위해 Executive 부서의 모든 사원에 대해 부서 번호, 성 및 직무 ID를 표시하는 보고서를 작성합니다.

	DEPARTMENT_ID	LAST_NAME	JOB_ID
1		90 King	AD_PRES
2		90 Kochhar	AD_VP
3		90 De Haan	AD_VP

시간 여유가 있을 경우 다음 연습을 완료하십시오.

7. 평균보다 많은 급여를 받고 성에 "u"가 포함된 사원이 있는 부서에서 근무하는 모든 사원의 사원 번호, 성 및 급여를 표시하도록 lab_07_03.sql의 query를 수정합니다. lab_07_03.sql을 lab_07_07.sql로 다시 저장합니다. lab_07_07.sql의 명령문을 실행합니다.

	EMPLOYEE_ID	LAST_NAME	SALARY
1		103 Hunold	9000

8

집합 연산자 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 집합 연산자 설명
- 집합 연산자를 사용하여 여러 **query**를 단일 **query**로 조합
- 반환되는 행의 순서 제어



Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 집합 연산자를 사용하여 **query**를 작성하는 방법을 배웁니다.

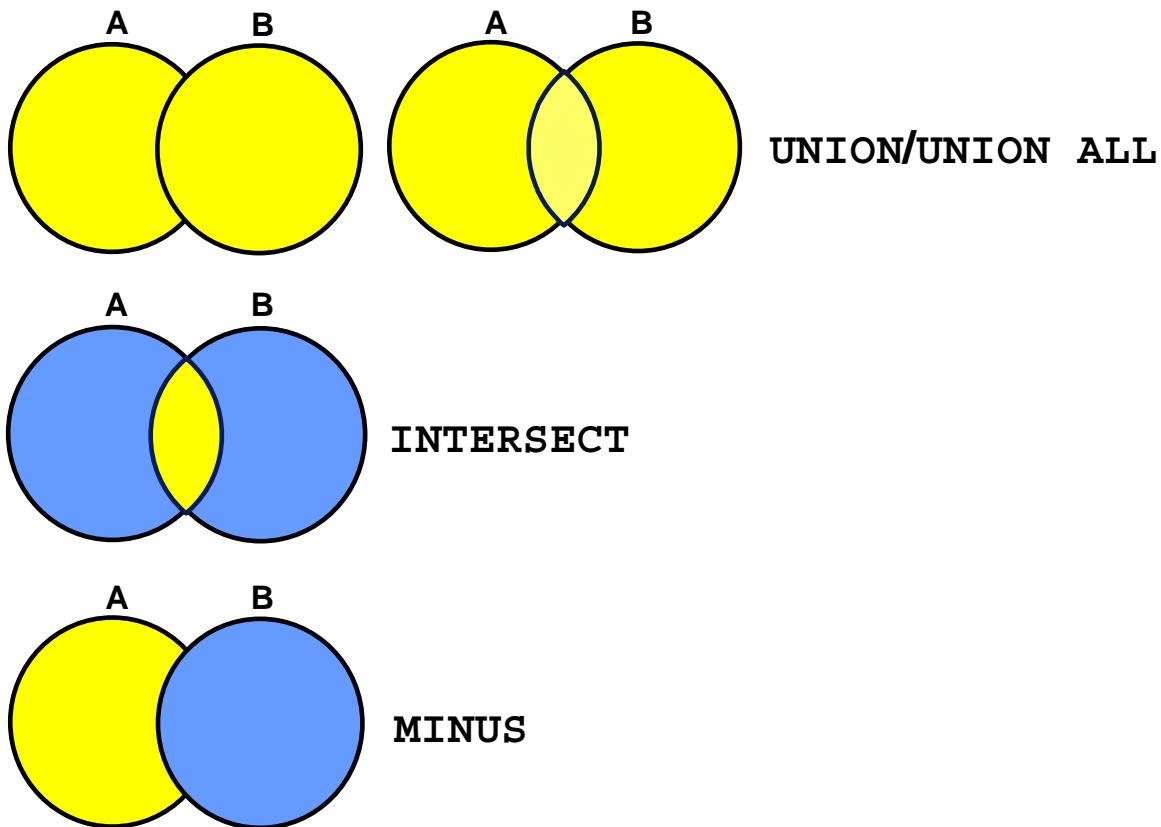
단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- UNION 및 UNION ALL 연산자
- INTERSECT 연산자
- MINUS 연산자
- SELECT 문 일치
- 집합 연산에서 ORDER BY 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

집합 연산자



ORACLE

Copyright © 2007, Oracle. All rights reserved.

집합 연산자

집합 연산자는 둘 이상의 구성 요소 query 결과를 하나의 결과로 조합합니다. 집합 연산자가 포함된 query는 **복합 질의**라고 합니다.

연산자	반환
UNION	중복 행이 제거된 두 query의 행
UNION ALL	중복 행이 포함된 두 query의 행
INTERSECT	두 query에 공통적인 행
MINUS	첫번째 query에 있는 행 중 두번째 query에 없는 행

집합 연산자는 모두 우선 순위가 같습니다. SQL 문에 여러 개의 집합 연산자가 포함되어 있으면 Oracle 서버는 팔호가 명시적으로 다른 순서를 지정하지 않는 한 왼쪽(위)에서 오른쪽(아래)으로 연산자를 평가합니다. INTERSECT 연산자가 다른 집합 연산자와 함께 사용된 query에서는 팔호를 사용하여 평가 순서를 명시적으로 지정해야 합니다.

집합 연산자 지침

- **SELECT** 리스트의 표현식은 개수가 일치해야 합니다.
- 두번째 **query**에 있는 각 열의 데이터 유형은 첫번째 **query**에 있는 상응하는 열의 데이터 유형과 일치해야 합니다.
- 괄호를 사용하여 실행 순서를 변경할 수 있습니다.
- **ORDER BY** 절은 명령문의 맨 끝에만 올 수 있습니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

집합 연산자 지침

- query의 SELECT 리스트에 있는 표현식은 개수 및 데이터 유형이 일치해야 합니다. WHERE 절에 UNION, UNION ALL, INTERSECT, MINUS 연산자를 사용하는 query는 SELECT 리스트의 열 개수와 데이터 유형이 동일해야 합니다. 복합 질의에서는 query의 SELECT 리스트에 있는 열의 데이터 유형이 정확히 일치하지 않을 수도 있습니다. 그러나 두번째 쿼리의 열은 첫번째 쿼리의 상응하는 열과 데이터 유형 그룹(숫자 또는 문자)이 동일해야 합니다.
- 집합 연산자는 subquery에서 사용할 수 있습니다.
- INTERSECT 연산자가 다른 집합 연산자와 함께 사용된 query에서는 괄호를 사용하여 평가 순서를 지정해야 합니다. 이렇게 하면 다른 집합 연산자보다 INTERSECT 연산자에 높은 우선 순위를 부여하는 최신 SQL 표준을 준수할 수 있습니다.

Oracle 서버 및 집합 연산자

- **UNION ALL**의 경우를 제외하면 중복 행은 자동으로 제거됩니다.
- 첫번째 **query**의 열 이름이 결과에 나타납니다.
- **UNION ALL**의 경우를 제외하고 출력은 기본적으로 오름차순으로 정렬됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Oracle 서버 및 집합 연산자

query에서 집합 연산자를 사용할 경우 Oracle 서버는 UNION ALL 연산자의 경우를 제외하고 자동으로 중복 행을 제거합니다. 출력의 열 이름은 첫번째 SELECT 문의 열 리스트에 따라 결정됩니다. 기본적으로 출력은 SELECT 절의 첫번째 열을 기준으로 오름차순으로 정렬됩니다. 복합 질의의 구성 요소 query에 있는 SELECT 리스트에서 해당 표현식은 개수 및 데이터 유형이 일치해야 합니다. 구성 요소 query에서 문자 데이터를 선택할 경우 반환 값의 데이터 유형은 다음과 같이 결정됩니다.

- 두 query에서 길이가 동일한 CHAR 데이터 유형의 값을 선택할 경우 해당 값과 길이 및 데이터 유형이 동일한 값이 반환됩니다. 두 query에서 길이가 서로 다른 CHAR 값을 선택할 경우 더 큰 CHAR 값과 동일한 길이의 VARCHAR2 값이 반환됩니다.
- 두 query 중 하나 또는 둘 모두에서 VARCHAR2 데이터 유형 값을 선택할 경우 VARCHAR2 데이터 유형 값이 반환됩니다.

구성 요소 query에서 숫자 데이터를 선택할 경우 반환 값의 데이터 유형은 숫자 우선 순위에 의해 결정됩니다. 모든 query에서 NUMBER 유형 값을 선택할 경우 NUMBER 데이터 유형 값이 반환됩니다. Oracle 서버는 집합 연산자를 사용하는 query에서 데이터 유형 그룹 간의 암시적 변환을 수행하지 않습니다. 따라서 구성 요소 query의 상응하는 두 표현식이 문자 데이터와 숫자 데이터로 분석되는 경우 Oracle 서버는 오류를 반환합니다.

단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- UNION 및 UNION ALL 연산자
- INTERSECT 연산자
- MINUS 연산자
- SELECT 문 일치
- 집합 연산에서 ORDER BY 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

이 단원에 사용되는 테이블

이 단원에 사용되는 테이블은 다음과 같습니다.

- **EMPLOYEES:** 모든 현재 사원에 대한 세부 정보 제공
- **JOB_HISTORY:** 사원의 직무가 바뀔 때 이전 직무의 시작 날짜와 종료 날짜, 그리고 직무 식별 번호 및 부서에 대한 세부 정보 기록



Copyright © 2007, Oracle. All rights reserved.

이 단원에 사용되는 테이블

이 단원에 사용되는 두 개의 테이블은 EMPLOYEES 테이블과 JOB_HISTORY 테이블입니다.

앞에서 살펴보았듯이 EMPLOYEES 테이블에는 고유 식별 번호, 전자 메일 주소, 직무 식별 정보 (ST_CLERK, SA REP 등), 급여, 관리자 등 사원에 대한 세부 정보가 저장됩니다.

일부 사원은 장기적으로 근무하면서 다양한 직무로 자리를 바꾸었습니다. 이러한 내용은 JOB_HISTORY 테이블을 사용하여 모니터합니다. 사원이 직무를 바꾸면 이전 직무를 시작한 날짜와 종료한 날짜, job_id(ST_CLERK, SA REP 등) 및 부서에 대한 세부 정보가 JOB_HISTORY 테이블에 기록됩니다.

EMPLOYEES 및 JOB_HISTORY 테이블의 구조와 데이터는 다음 페이지에 나옵니다.

이 단원에 사용되는 테이블(계속)

회사에는 재직 중에 두 번 이상 동일한 직책을 보유한 사원들이 있었습니다. 예를 들어, 1998년 3월 24일에 입사한 사원 Taylor의 경우를 살펴봅시다. Taylor는 1998년 3월 24일부터 1998년 12월 31일까지 SA_REP 직책을 보유했고, 1999년 1월 1일부터 1999년 12월 31일까지는 SA_MAN 직책을 보유하였습니다. Taylor는 다시 현재 직책인 SA_REP로 복귀하였습니다.

```
DESCRIBE employees
```

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)

이 단원에 사용되는 테이블(계속)

```
SELECT employee_id, last_name, job_id, hire_date, department_id
FROM employees;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	DEPARTMENT_ID
1	100 King	AD_PRES	17-JUN-87		90
2	101 Kochhar	AD_VP	21-SEP-89		90
3	102 De Haan	AD_VP	13-JAN-93		90
4	103 Hunold	IT_PROG	03-JAN-90		60
5	104 Ernst	IT_PROG	21-MAY-91		60
6	107 Lorentz	IT_PROG	07-FEB-99		60
7	124 Mourgos	ST_MAN	16-NOV-99		50
8	141 Rajs	ST_CLERK	17-OCT-95		50
9	142 Davies	ST_CLERK	29-JAN-97		50
10	143 Matos	ST_CLERK	15-MAR-98		50
11	144 Vargas	ST_CLERK	09-JUL-98		50
12	149 Zlotkey	SA_MAN	29-JAN-00		80
13	174 Abel	SA_REP	11-MAY-96		80
14	176 Taylor	SA_REP	24-MAR-98		80
15	178 Grant	SA_REP	24-MAY-99		(null)
16	200 Whalen	AD_ASST	17-SEP-87		10
17	201 Hartstein	MK_MAN	17-FEB-96		20

...

```
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)

이 단원에 사용되는 테이블(계속)

```
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

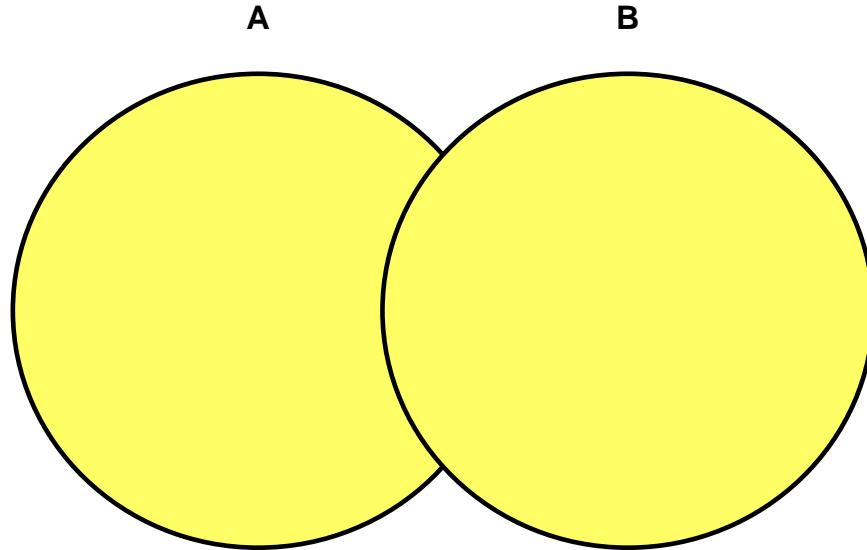
단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- **UNION** 및 **UNION ALL** 연산자
- **INTERSECT** 연산자
- **MINUS** 연산자
- **SELECT** 문 일치
- 집합 연산에서 **ORDER BY** 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

UNION 연산자



UNION 연산자는 중복 행을 제거한 후 양쪽 query에서 행을 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

UNION 연산자

UNION 연산자는 양쪽 query에 의해 선택된 모든 행을 반환합니다. UNION 연산자를 사용하여 여러 테이블에서 중복 행을 제거하고 모든 행을 반환합니다.

지침

- 선택한 열의 개수가 동일해야 합니다.
- 선택한 열의 데이터 유형이 동일한 데이터 유형 그룹(숫자 또는 문자)에 속해야 합니다.
- 열 이름은 동일하지 않아도 됩니다.
- UNION은 선택된 모든 열에 대해 실행됩니다.
- 중복 검사 시 NULL 값을 무시하지 않습니다.
- 기본적으로 출력은 SELECT 절의 열을 기준으로 오름차순으로 정렬됩니다.

UNION 연산자 사용

모든 사원의 현재 및 이전 직무 세부 정보를 표시합니다. 각 사원을 한 번만 표시합니다.

```
SELECT employee_id, job_id  
FROM employees  
UNION  
SELECT employee_id, job_id  
FROM job_history;
```

EMPLOYEE_ID	JOB_ID
1	100 AD_PRES
2	101 AC_ACCOUNT
...	
22	200 AC_ACCOUNT
23	200 AD_ASST
24	201 MK_MAN
...	

ORACLE

Copyright © 2007, Oracle. All rights reserved.

UNION 연산자 사용

UNION 연산자는 중복 레코드를 제거합니다. EMPLOYEES 테이블과 JOB_HISTORY 테이블에서 나오는 레코드가 동일하면 레코드가 한 번만 표시됩니다. 슬라이드에 표시된 출력에서, EMPLOYEE_ID가 200인 사원의 레코드는 JOB_ID가 각 행에서 다르기 때문에 두 번 나타나는 것을 알 수 있습니다.

다음 예제를 살펴보십시오.

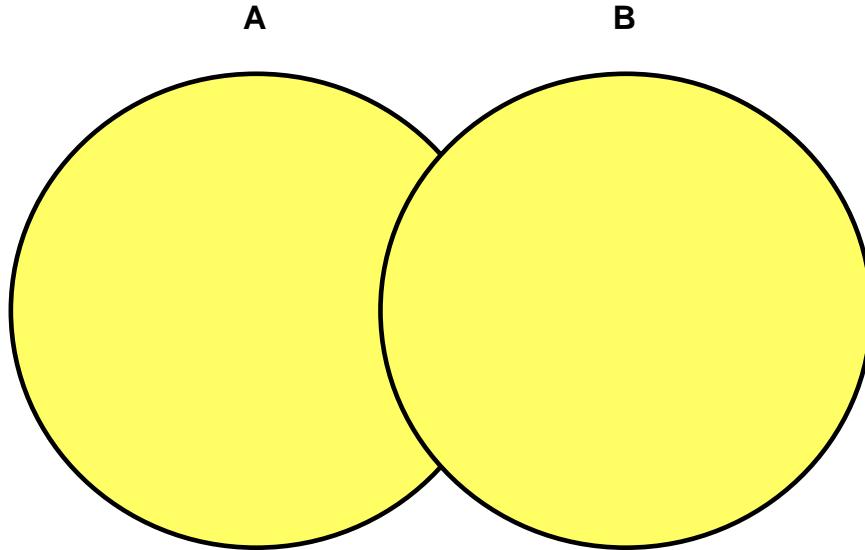
```
SELECT employee_id, job_id, department_id  
FROM employees  
UNION  
SELECT employee_id, job_id, department_id  
FROM job_history;
```

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100 AD_PRES	90
2	101 AC_ACCOUNT	110
...		
22	200 AC_ACCOUNT	90
23	200 AD_ASST	10
24	200 AD_ASST	90
...		

UNION 연산자 사용(계속)

앞의 출력에서 사원 200은 세 번 나타납니다. 그 이유는 무엇입니까? 사원 200의 DEPARTMENT_ID 값은 보십시오. DEPARTMENT_ID 값이 각 행에서 90, 10, 90으로 다르게 표시되어 있습니다. 이렇게 직무 ID와 부서 ID의 조합이 서로 다르기 때문에 사원 200의 각 행은 고유하며, 따라서 중복된 값으로 처리되지 않습니다. 출력이 SELECT 절의 첫번째 열, 이 경우 EMPLOYEE_ID의 오름차순으로 정렬된 것을 알 수 있습니다.

UNION ALL 연산자



UNION ALL 연산자는 모든 중복 행을 포함하여 양쪽 **query**의 결과를 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

UNION ALL 연산자

UNION ALL 연산자를 사용하여 여러 query에서 모든 행을 반환합니다.

지침

UNION에 대한 지침과 UNION ALL에 대한 지침은 두 가지 점 외에는 동일합니다. 즉, UNION ALL의 경우 UNION과 달리 기본적으로 중복 행이 제거되지 않고 출력이 정렬되지 않습니다.

UNION ALL 연산자 사용

모든 사원의 현재 및 이전 부서를 표시합니다.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
16	144	ST_CLERK	50
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
...			
30	206	AC_ACCOUNT	110

ORACLE

Copyright © 2007, Oracle. All rights reserved.

UNION ALL 연산자 사용

이 예에서는 30개 행이 선택되었습니다. 두 테이블의 조합은 총 30개 행입니다. UNION ALL 연산자는 중복 행을 제거하지 않습니다. UNION은 양쪽 query에 의해 선택된 모든 구분 행을 반환하고, UNION ALL은 모든 중복 행을 포함하여 양쪽 query에 의해 선택된 모든 행을 반환합니다. 슬라이드의 query를 살펴보십시오. 이제 UNION 절로 작성해 보겠습니다.

```
SELECT employee_id, job_id, department_id
FROM employees
UNION
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

앞의 query는 29개 행을 반환합니다. 이 query가 다음 중복 행을 제거하기 때문입니다.

176 SA_REP	80
------------	----

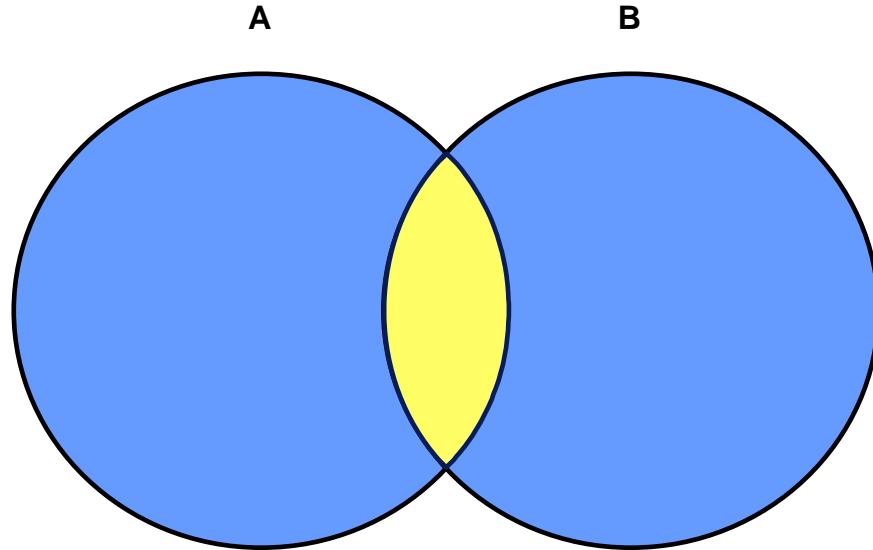
단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- **UNION** 및 **UNION ALL** 연산자
- **INTERSECT** 연산자
- **MINUS** 연산자
- **SELECT** 문 일치
- 집합 연산에서 **ORDER BY** 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

INTERSECT 연산자



INTERSECT 연산자는 양쪽 query에 공통되는 행을 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

INTERSECT 연산자

INTERSECT 연산자를 사용하여 여러 query에 공통되는 모든 행을 반환합니다.

지침

- query에서 SELECT 문에 의해 선택될 열의 개수와 데이터 유형은 query에 사용된 모든 SELECT 문에서 동일해야 합니다. 그러나 열 이름은 동일하지 않아도 됩니다.
- 교집합 테이블의 순서를 역으로 바꾸어도 결과는 바뀌지 않습니다.
- INTERSECT는 NULL 값을 무시하지 않습니다.

INTERSECT 연산자 사용

현재 직책이 이전 직책과 동일한 사원(즉, 직무가 변경된 적이 있지만 현재는 이전 직무로 복귀한 사원)의 사원 ID와 직무 ID를 표시합니다.

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

ORACLE

Copyright © 2007, Oracle. All rights reserved.

INTERSECT 연산자 사용

슬라이드의 예제에서 query는 양쪽 테이블에서 선택된 열에 동일한 값을 가지는 레코드만 반환합니다.

EMPLOYEES 테이블에 대한 SELECT 문에 DEPARTMENT_ID 열을 추가하고 JOB_HISTORY 테이블에 대한 SELECT 문에 DEPARTMENT_ID 열을 추가한 다음 이 query를 실행하면 어떤 결과가 발생합니까? 추가된 다른 열의 값이 중복될 수도 있고 중복되지 않을 수도 있으므로 결과는 달라질 수 있습니다.

예제:

```
SELECT employee_id, job_id, department_id
FROM employees
INTERSECT
SELECT employee_id, job_id, department_id
FROM job_history;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	176	SA_REP	80

EMPLOYEES.DEPARTMENT_ID 값이 JOB_HISTORY.DEPARTMENT_ID 값과 다르기 때문에 사원 200은 더 이상 결과에 포함되지 않습니다.

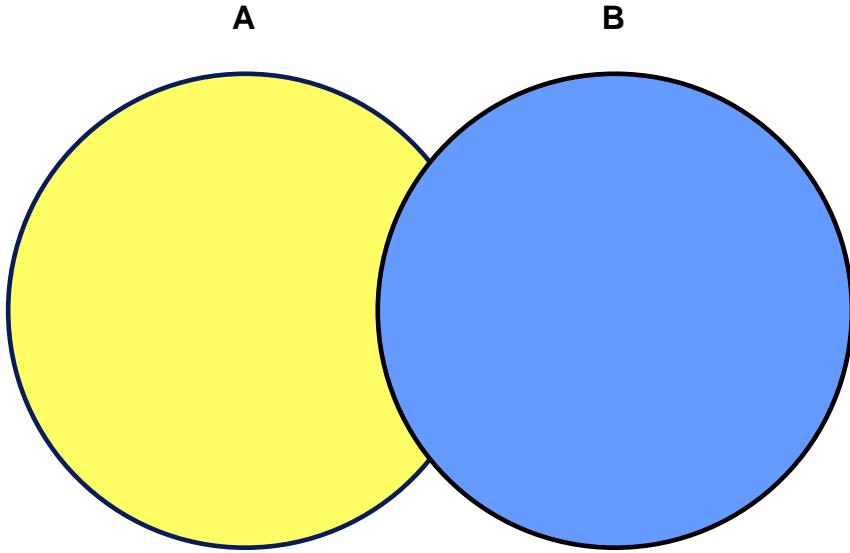
단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- UNION 및 UNION ALL 연산자
- INTERSECT 연산자
- MINUS 연산자
- SELECT 문 일치
- 집합 연산에서 ORDER BY 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

MINUS 연산자



MINUS 연산자는 첫번째 **query**에 의해 선택되지만 두번째 **query** 결과 집합에는 없는 모든 구분 행을 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

MINUS 연산자

MINUS 연산자를 사용하여 첫번째 query에 의해 선택되지만 두번째 query 결과 집합에는 없는 모든 구분 행을 반환합니다(첫번째 SELECT 문 MINUS 두번째 SELECT 문).

주: query에 사용되는 모든 SELECT 문에서 열의 개수가 동일해야 하며 해당 query의 SELECT 문에서 선택될 열의 데이터 유형은 동일한 데이터 유형 그룹에 속해야 합니다. 그러나 열 이름은 동일하지 않아도 됩니다.

MINUS 연산자 사용

한 번도 직무를 변경하지 않은 사원의 사원 ID를 표시합니다.

```
SELECT employee_id  
FROM   employees  
MINUS  
SELECT employee_id  
FROM   job_history;
```

	EMPLOYEE_ID
1	100
2	103
3	104
4	107
5	124
...	
14	205
15	206

ORACLE

Copyright © 2007, Oracle. All rights reserved.

MINUS 연산자 사용

슬라이드의 예제에서는 EMPLOYEES 테이블에 있는 사원 ID에서 JOB_HISTORY 테이블에 있는 사원 ID를 뺍니다. 결과 집합은 빼기 후에 남은 사원을 표시하며, 이들은 EMPLOYEES 테이블에 존재하지만 JOB_HISTORY 테이블에는 존재하지 않는 행으로 표시됩니다. 이것이 한 번도 직무를 변경하지 않은 사원의 레코드입니다.

단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- UNION 및 UNION ALL 연산자
- INTERSECT 연산자
- MINUS 연산자
- **SELECT** 문 일치
- 집합 연산에서 ORDER BY 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SELECT 문 일치

- **UNION** 연산자를 사용하여 위치 **ID**, 부서 이름 및 해당 부서가 소재하는 지역을 표시합니다.
- 선택될 열이 하나 이상의 테이블에 존재하지 않는 경우 **TO_CHAR** 함수 또는 다른 변환 함수를 사용하여 데이터 유형을 일치시켜야 합니다.

```
SELECT location_id, department_name "Department",
       TO_CHAR(NULL) "Warehouse location"
  FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
       state_province
  FROM locations;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SELECT 문 일치

query에서 SELECT 리스트의 표현식은 개수가 일치해야 하기 때문에 더미 열과 데이터 유형 변환 함수를 사용하여 이러한 규칙을 준수합니다. 슬라이드에서는 더미 열 머리글로 Warehouse location이라는 이름이 제공되었습니다. 두번째 query에 의해 검색되는 state_province 열의 VARCHAR2 데이터 유형과 일치시키기 위해 첫번째 query에서 TO_CHAR 함수를 사용합니다. 이와 마찬가지로 첫번째 query에 의해 검색되는 department_name 열의 VARCHAR2 데이터 유형과 일치시키기 위해 두번째 query에서 TO_CHAR 함수를 사용합니다.
해당 query의 출력은 다음과 같습니다.

	LOCATION_ID	Department	Warehouse location
1	1400	IT	(null)
2	1400	(null)	Texas
3	1500	Shipping	(null)
4	1500	(null)	California
5	1700	Accounting	(null)
6	1700	Administration	(null)
7	1700	Contracting	(null)
8	1700	Executive	(null)
9	1700	(null)	Washington
...			

SELECT 문 일치: 예제

UNION 연산자를 사용하여 모든 사원의 사원 **ID**, 직무 **ID** 및 급여를 표시합니다.

```
SELECT employee_id, job_id, salary
FROM employees
UNION
SELECT employee_id, job_id, 0
FROM job_history;
```

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000
...			
29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SELECT 문 일치: 예제

EMPLOYEES 테이블과 JOB_HISTORY 테이블은 공통되는 여러 열을 가지고 있습니다 (EMPLOYEE_ID, JOB_ID, DEPARTMENT_ID 등). 그러나 급여가 EMPLOYEES 테이블에만 있는 상황에서, UNION 연산자를 사용하여 사원 ID, 직무 ID 및 급여를 표시하도록 query하려면 어떻게 해야 할까요?

슬라이드의 코드 예제는 EMPLOYEES 테이블과 JOB_HISTORY 테이블의 EMPLOYEE_ID 열과 JOB_ID 열을 일치시킵니다. EMPLOYEES SELECT 문의 숫자 SALARY 열과 일치시키기 위해 JOB_HISTORY SELECT 문에 리터럴 값 0이 추가됩니다.

슬라이드에 표시된 결과에서 JOB_HISTORY 테이블의 레코드에 해당되는 출력의 각 행에는 SALARY 열에 0이 포함되어 있습니다.

단원 내용

- 집합 연산자: 유형 및 지침
- 이 단원에 사용되는 테이블
- **UNION** 및 **UNION ALL** 연산자
- **INTERSECT** 연산자
- **MINUS** 연산자
- **SELECT** 문 일치
- 집합 연산에서 **ORDER BY** 절 사용

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

집합 연산에서 **ORDER BY** 절 사용

- **ORDER BY** 절은 복합 질의의 맨 끝에 한 번만 올 수 있습니다.
- 구성 요소 **query**에 개별적으로 **ORDER BY** 절을 사용할 수 없습니다.
- **ORDER BY** 절은 첫번째 **SELECT query**의 열만 인식합니다.
- 기본적으로 첫번째 **SELECT query**의 첫번째 열을 기준으로 오름차순으로 출력이 정렬됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

집합 연산에서 **ORDER BY** 절 사용

ORDER BY 절은 복합 질의에서 한 번만 사용될 수 있습니다. **ORDER BY** 절을 사용할 경우에는 **query** 맨 끝에 두어야 합니다. **ORDER BY** 절에서 열 이름이나 alias를 사용할 수 있습니다.

기본적으로 출력은 첫번째 **SELECT query**의 첫번째 열을 기준으로 오름차순으로 정렬됩니다.

주: **ORDER BY** 절은 두번째 **SELECT query**의 열 이름을 인식하지 못합니다. 열 이름을 혼동하지 않도록 **ORDER BY** 열 위치를 사용하는 것이 일반적입니다.

예를 들어, 다음 명령문의 경우 **job_id**의 오름차순으로 출력이 표시됩니다.

```
SELECT employee_id, job_id, salary
  FROM employees
  UNION
SELECT employee_id, job_id, 0
  FROM job_history
 ORDER BY 2;
```

ORDER BY 절을 생략하는 경우 출력은 기본적으로 **employee_id**의 오름차순으로 정렬됩니다. 두번째 **query**의 열을 사용하여 출력을 정렬할 수는 없습니다.

요약

이 단원에서는 다음 항목의 사용법에 대해 설명했습니다.

- 모든 구분 행을 반환하는 **UNION**
- 중복 행을 포함한 모든 행을 반환하는 **UNION ALL**
- 두 **query**에서 공유하는 모든 행을 반환하는 **INTERSECT**
- 첫번째 **query**에서는 선택되지만 두번째 **query**에서는 선택되지 않는 모든 구분 행을 반환하는 **MINUS**
- 명령문의 맨 끝에만 올 수 있는 **ORDER BY**

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

- UNION 연산자는 복합 질의에서 각 query에 의해 선택된 모든 구분 행을 반환합니다. UNION 연산자를 사용하여 여러 테이블에서 중복 행을 제거하고 모든 행을 반환합니다.
- UNION ALL 연산자를 사용하여 여러 query에서 모든 행을 반환합니다. UNION 연산자와 달리 중복 행이 제거되지 않고 출력은 기본적으로 정렬되지 않습니다.
- INTERSECT 연산자를 사용하여 여러 query에 공통되는 모든 행을 반환합니다.
- MINUS 연산자를 사용하여 첫번째 query에서 반환된 행 중 두번째 query에 없는 행을 반환합니다.
- ORDER BY 절은 복합 명령문의 맨 끝에서만 사용합니다.
- SELECT 리스트에 있는 해당 표현식에서 개수와 데이터 유형이 일치하는지 확인합니다.

연습 8: 개요

이 연습에서는 다음을 사용하여 보고서를 작성합니다.

- **UNION** 연산자
- **INTERSECTION** 연산자
- **MINUS** 연산자



Copyright © 2007, Oracle. All rights reserved.

연습 8: 개요

이 연습에서는 집합 연산자를 사용하여 query를 작성합니다.

연습 8

1. HR 부서에서 직무 ID ST_CLERK을 포함하지 않는 부서에 대한 부서 ID 리스트를 요구합니다. 집합 연산자를 사용하여 이 보고서를 작성합니다.

	DEPARTMENT_ID
1	10
2	20
3	60
4	80
5	90
6	110
7	190

2. HR 부서에서 부서가 소재하지 않는 국가의 리스트를 요구합니다. 해당 국가의 국가 ID 및 이름을 표시합니다. 집합 연산자를 사용하여 이 보고서를 작성합니다.

COUNTRY_ID	COUNTRY_NAME
1 DE	Germany

3. 부서 10, 50 및 20에 대한 직무 리스트를 이 순서대로 생성합니다. 집합 연산자를 사용하여 직무 ID 및 부서 ID를 표시합니다.

JOB_ID	DEPARTMENT_ID
1 AD_ASST	10
2 ST_MAN	50
3 ST_CLERK	50
4 MK_MAN	20
5 MK_REP	20

4. 현재 직책이 회사에 처음 입사할 때의 직책과 동일한 사원(즉, 직무가 변경된 적은 있지만 현재 원래 직무로 복귀한 사원)의 사원 ID와 직무 ID를 나열하는 보고서를 작성합니다.

EMPLOYEE_ID	JOB_ID
1	176 SA_REP
2	200 AD_ASST

연습 8(계속)

5. HR 부서에서 다음과 같은 사양의 보고서를 요구합니다.

- 부서에 소속되었는지 여부에 관계없이 EMPLOYEES 테이블에 있는 모든 사원의 성 및 부서 ID
- 해당 부서에 근무 중인 사원이 있는지 여부에 관계없이 DEPARTMENTS 테이블에 있는 모든 부서의 부서 ID 및 부서 이름

이를 수행하는 복합 질의를 작성합니다.

	LAST_NAME	DEPARTMENT_ID	TO_CHAR(NULL)
1	Abel		80 (null)
2	Davies		50 (null)
3	De Haan		90 (null)
4	Ernst		60 (null)
5	Fay		20 (null)
6	Gietz		110 (null)
7	Grant		(null) (null)
8	Hartstein		20 (null)
9	Higgins		110 (null)
10	Hunold		60 (null)
11	King		90 (null)
12	Kochhar		90 (null)
13	Lorentz		60 (null)
14	Matos		50 (null)
15	Mourgos		50 (null)
16	Rajs		50 (null)
17	Taylor		80 (null)
18	Vargas		50 (null)
19	Whalen		10 (null)
20	Zlotkey		80 (null)
21	(null)		10 Administration
22	(null)		20 Marketing
23	(null)		50 Shipping
24	(null)		60 IT
25	(null)		80 Sales
26	(null)		90 Executive
27	(null)		110 Accounting
28	(null)		190 Contracting

9

데이터 조작

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 각 **DML**(데이터 조작어) 문 설명
- 테이블에 행 삽입
- 테이블의 행 갱신
- 테이블에서 행 삭제
- 트랜잭션 제어

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 DML(데이터 조작어) 문을 사용하여 테이블에 행을 삽입하고, 테이블의 기존 행을 갱신하고, 테이블에서 기존 행을 삭제하는 방법을 배웁니다. 또한 COMMIT, SAVEPOINT 및 ROLLBACK 문을 사용하여 트랜잭션을 제어하는 방법을 배웁니다.

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT**, **ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터 조작어

- **DML** 문은 다음과 같은 경우에 실행합니다.
 - 테이블에 새 행 추가
 - 테이블의 기존 행 수정
 - 테이블에서 기존 행 제거
- 트랜잭션은 논리적 작업 단위를 형성하는 **DML** 문의 모음으로 구성됩니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터 조작어

DML(데이터 조작어)은 SQL의 핵심 부분입니다. 데이터베이스에서 데이터를 추가, 갱신 또는 삭제하려면 DML 문을 실행합니다. 논리적 작업 단위를 형성하는 DML 문의 모음을 트랜잭션이라고 합니다.

은행 데이터베이스를 생각해 봅시다. 은행 고객이 예금 계좌에서 결제 계좌로 현금을 이체하는 경우 트랜잭션은 예금 계좌 감소, 결제 계좌 증가, 트랜잭션 저널에 트랜잭션 기록이라는 세 가지 별도 작업으로 구성됩니다. Oracle 서버에서 세 개의 SQL 문을 수행할 때 계좌 잔액이 정확히 유지되어야 합니다. 특정 문제로 인해 트랜잭션의 명령문 중 하나가 실행되지 못하면 트랜잭션의 다른 명령문도 언두되어야 합니다.

주: 이 단원에 나오는 대부분의 DML 문에서는 테이블에 대한 제약 조건이 위반되지 않은 것으로 가정합니다. 제약 조건은 이 과정의 뒷부분에서 다룹니다.

주: SQL Developer에서 DML 문을 실행하려면 Run Script 아이콘 또는 [F5]를 누릅니다. 피드백 메시지가 Script Output 탭 페이지에 표시됩니다.

테이블에 새 행 추가

DEPARTMENTS

70 Public Relations	100	1700	새 행
---------------------	-----	------	-----

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

DEPARTMENTS 테이블에
새 행 삽입

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
9	70 Public Relations	100	1700

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블에 새 행 추가

슬라이드의 그림은 DEPARTMENTS 테이블에 새 부서를 추가하는 과정을 보여줍니다.

INSERT 문 구문

- **INSERT** 문을 사용하여 테이블에 새 행을 추가합니다.

```
INSERT INTO table [(column [, column...])]  
VALUES      (value [, value...]);
```

- 이 구문을 사용할 경우 한 번에 한 행만 삽입됩니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

INSERT 문 구문

INSERT 문을 실행하여 테이블에 새 행을 추가할 수 있습니다.

구문 설명:

<i>table</i>	테이블의 이름입니다.
<i>column</i>	테이블에 채울 열의 이름입니다.
<i>value</i>	열의 해당 값입니다.

주: VALUES 절이 있는 이 명령문은 한 번에 한 행만 테이블에 추가합니다.

새 행 삽입

- 각 열에 대한 값을 포함하는 새 행을 삽입합니다.
- 테이블에 있는 열의 기본 순서로 값을 나열합니다.
- 선택적으로 **INSERT** 절에 열을 나열합니다.

```
INSERT INTO departments(department_id,
    department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

```
1 rows inserted
```

- 문자와 날짜 같은 작은 따옴표로 묶습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

새 행 삽입

각 열에 대한 값을 포함하는 새 행을 삽입할 수 있기 때문에 **INSERT** 절에서 열 리스트가 필요하지 않습니다. 그러나 열 리스트를 사용하지 않을 경우 테이블에 있는 열의 기본 순서에 따라 값이 나열되어야 하고 각 열에 대해 값이 제공되어야 합니다.

```
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)

명확성을 위해 **INSERT** 절에서 열 리스트를 사용하십시오.

문자와 날짜 같은 작은 따옴표로 묶습니다. 그러나 숫자 값을 작은 따옴표로 묶는 것은 권장되지 않습니다.

null 값을 가진 행 삽입

- 암시적 방법: 열 리스트에서 열을 생략합니다.

```
INSERT INTO departments (department_id,
                        department_name)
VALUES          (30, 'Purchasing');
```

- 명시적 방법: VALUES 절에서 NULL 키워드를 지정합니다.

```
INSERT INTO departments
VALUES          (100, 'Finance', NULL, NULL);
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

null 값을 가진 행 삽입

방법	설명
암시적	열 리스트에서 열을 생략합니다.
명시적	VALUES 리스트에 NULL 키워드를 지정합니다. 문자열과 날짜에 대해 VALUES 리스트에 빈 문자열('')을 지정합니다.

대상 열에서 null 값을 사용할 수 있는지 확인하려면 DESCRIBE 명령을 사용하여 Null 상태를 확인해야 합니다.

Oracle 서버는 모든 데이터 유형, 데이터 범위 및 데이터 무결성 제약 조건을 자동으로 강제 적용합니다. 명시적으로 나열되지 않은 열은 새 행에서 null 값을 가집니다.

다음 순서에 따라 유저가 입력하는 동안 발생할 수 있는 일반적인 오류를 검사합니다.

- NOT NULL 열에 누락된 필수 값
- unique key 또는 primary key 제약 조건을 위반하는 중복 값
- CHECK 제약 조건을 위반하는 값
- foreign key에 대해 유지되는 참조 무결성 제약 조건
- 데이터 유형 불일치 또는 열 크기를 초과하는 값

주: 열 리스트를 통해 보다 읽기 쉽고 신뢰할 수 있으며 틀릴 우려가 적은 INSERT 문을 작성할 수 있으므로 열 리스트를 사용하는 것이 좋습니다.

특수 값 삽입

SYSDATE 함수는 현재 날짜와 시간을 기록합니다.

```
INSERT INTO employees (employee_id,
                      first_name, last_name,
                      email, phone_number,
                      hire_date, job_id, salary,
                      commission_pct, manager_id,
                      department_id)
VALUES (113,
        'Louis', 'Popp',
        'LPOPP', '515.124.4567',
        SYSDATE, 'AC_ACCOUNT', 6900,
        NULL, 205, 110);

1 rows inserted
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

특수 값 삽입

함수를 사용하여 테이블에 특수 값을 입력할 수 있습니다.

슬라이드의 예제는 EMPLOYEES 테이블에 사원 Popp의 정보를 기록합니다. 이 예제는 HIRE_DATE 열에 현재 날짜와 시간을 제공합니다. 이 예제에서는 데이터베이스 서버의 현재 날짜와 시간을 반환하는 SYSDATE 함수를 사용합니다. CURRENT_DATE 함수를 사용하여 해당 세션 시간대의 현재 날짜를 구할 수도 있습니다. 또한 테이블에 행을 삽입할 때 USER 함수를 사용할 수 있습니다. USER 함수는 현재 유저 이름을 기록합니다.

테이블에 추가된 내용 확인

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	AC_ACCOUNT	11-JUN-07	(null)

특정 날짜 및 시간 값 삽입

- 새 사원을 추가합니다.

```
INSERT INTO employees
VALUES      (114,
              'Den', 'Raphealy',
              'DRAPHEAL', '515.127.4561',
              TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
              'SA REP', 11000, 0.2, 100, 60);
1 rows inserted
```

- 추가한 내용을 확인합니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
114 Den	Raphealy	DRAPHEAL	515.127.4561		03-FEB-99	SA REP	11000	0.2

ORACLE

Copyright © 2007, Oracle. All rights reserved.

특정 날짜 및 시간 값 삽입

일반적으로 DD-MON-RR 형식을 사용하여 날짜 값을 삽입합니다. RR 형식을 사용하는 경우 시스템은 정확한 세기를 자동으로 제공합니다.

날짜 값을 DD-MON-YYYY 형식으로 제공할 수도 있습니다. 이 형식은 정확한 세기를 지정하는 내부 RR 형식 논리에 의존하지 않으며 세기를 명확히 지정하므로 이 형식을 사용하는 것이 좋습니다.

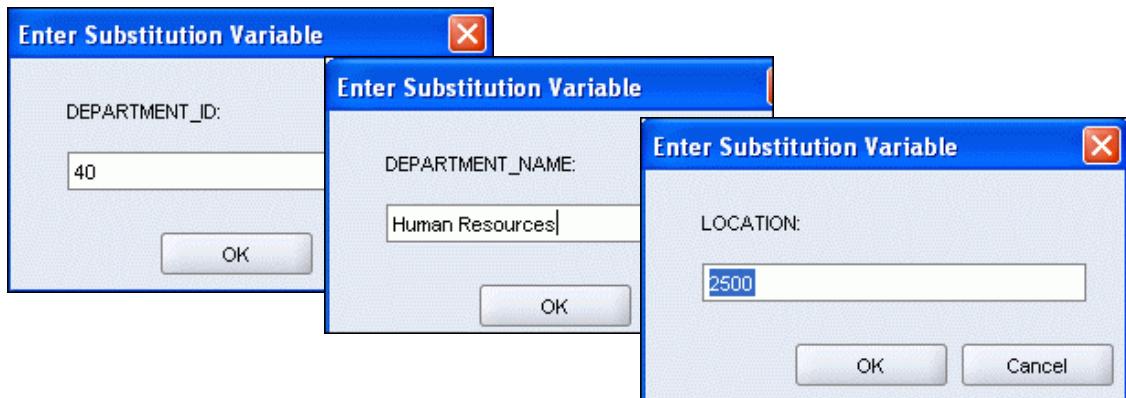
기본 형식이 아닌 다른 형식으로 날짜를 입력해야 하는 경우(예를 들어, 다른 세기나 특정 시간) TO_DATE 함수를 사용해야 합니다.

슬라이드의 예제는 EMPLOYEES 테이블에 Raphealy 사원의 정보를 기록합니다. 이 예제에서는 HIRE_DATE 열을 February 3, 1999로 설정합니다.

스크립트 작성

- **SQL** 문에서 & 치환을 사용하여 값을 입력하도록 요구합니다.
- & 는 변수 값에 대한 위치 표시자입니다.

```
INSERT INTO departments
    (department_id, department_name, location_id)
VALUES (&department_id, '&department_name', &location);
```



ORACLE

Copyright © 2007, Oracle. All rights reserved.

스크립트 작성

치환 변수를 사용하여 파일에 명령을 저장하고 파일의 명령을 실행할 수 있습니다. 슬라이드의 예제는 DEPARTMENTS 테이블에 부서에 대한 정보를 기록합니다.

스크립트 파일을 실행하면 각 앤퍼샌드(&) 치환 변수에 대해 값을 입력하도록 요구합니다. 치환 변수에 대한 값을 입력한 후에 OK 버튼을 누릅니다. 그러면 명령문 내에서 치환 변수가 입력한 값으로 치환됩니다. 이렇게 하면 동일한 스크립트 파일을 반복적으로 실행할 수 있고 파일을 실행할 때마다 다른 값 집합을 제공할 수 있습니다.

다른 테이블에서 행 복사

- **subquery**를 사용하여 **INSERT** 문을 작성합니다.

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM   employees
WHERE  job_id LIKE '%REP%';
```

4 rows inserted

- **VALUES** 절을 사용하지 마십시오.
- **INSERT** 절의 열 개수를 **subquery**의 열 개수와 일치시킵니다.
- **subquery**에서 반환되는 모든 행을 **sales_reps** 테이블에 삽입합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다른 테이블에서 행 복사

INSERT 문을 사용하여 기존 테이블에서 파생된 값으로 테이블에 행을 추가할 수 있습니다. 슬라이드 예제에서 INSERT INTO 문이 작동하도록 하려면 먼저 CREATE TABLE 문을 사용하여 sales_reps 테이블을 만들어 두어야 합니다. CREATE TABLE에 대해서는 다음 단원 "DDL 문을 사용하여 테이블 생성 및 관리"에서 설명합니다.

VALUES 절 자리에 subquery를 사용합니다.

구문

```
INSERT INTO table [ column (, column) ] subquery;
```

구문 설명:

table 테이블의 이름입니다.

column 테이블에 채울 열의 이름입니다.

subquery 테이블에 행을 반환하는 subquery입니다.

INSERT 절의 열 리스트에 나오는 열 개수 및 해당 데이터 유형은 subquery의 값 개수 및 해당 데이터 유형과 일치해야 합니다. subquery에 의해 반환되는 행의 개수에 따라 하나 이상의 행이 추가되거나 어떠한 행도 추가되지 않습니다. 테이블의 행 복사본을 생성하려면 subquery에서 SELECT *를 사용합니다.

```
INSERT INTO copy_emp
SELECT *
FROM   employees;
```

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT**, **ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

테이블의 데이터 변경

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

EMPLOYEES 테이블의 행을 갱신합니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블의 데이터 변경

슬라이드는 부서 60의 사원에 대해 부서 번호를 부서 80으로 변경하는 과정을 보여줍니다.

UPDATE 문 구문

- **UPDATE** 문을 사용하여 테이블의 기존 값을 수정합니다.

```
UPDATE      table  
SET         column = value [, column = value, ...]  
[WHERE      condition];
```

- 필요한 경우 한 번에 두 개 이상의 행을 갱신합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

UPDATE 문 구문

UPDATE 문을 사용하여 테이블의 기존 값을 수정할 수 있습니다.

구문 설명:

table 테이블의 이름입니다.

column 테이블에 채울 열의 이름입니다.

value 열의 해당 값 또는 subquery입니다.

condition 갱신할 행을 식별하며 열 이름, 표현식, 상수, subquery 및 비교 연산자로 구성됩니다.

테이블을 query하여 갱신된 행을 표시하고 갱신 작업을 확인합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "UPDATE" 관련 섹션을 참조하십시오.

주: 갱신할 단일 행을 식별하려면 일반적으로 WHERE 절에 primary key 열을 사용합니다. 다른 열을 사용하면 예상치 못한 여러 행이 갱신될 수도 있습니다. 예를 들어, EMPLOYEES 테이블의 단일 행을 이름으로 식별하는 것은 위험합니다. 두 명 이상의 사원이 동일한 이름을 가질 수도 있기 때문입니다.

테이블의 행 갱신

- **WHERE** 절을 지정하면 특정 행에서 값이 수정됩니다.

```
UPDATE employees
SET department_id = 50
WHERE employee_id = 113;
1 rows updated
```

- **WHERE** 절을 생략하면 테이블의 모든 행에서 값이 수정됩니다.

```
UPDATE copy_emp
SET department_id = 110;
22 rows updated
```

- 열 값을 **NULL**로 갱신하려면 **SET column_name = NULL**을 지정합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블의 행 갱신

WHERE 절이 지정된 경우 UPDATE 문은 특정 행의 값을 수정합니다. 슬라이드의 예제는 사원 113(Popp)을 부서 50으로 이동하는 것을 보여줍니다.

WHERE 절을 생략하면 테이블의 모든 행에서 값이 수정됩니다. COPY_EMP 테이블의 갱신된 행을 살펴봅니다.

```
SELECT last_name, department_id
FROM copy_emp;
```

	LAST_NAME	DEPARTMENT_ID
1	King	110
2	Kochhar	110

■ ■ ■

예를 들어, 직무가 SA REP이었던 사원이 IT PROG으로 직무를 변경한 경우를 가정해 보겠습니다. 이에 따라 해당 사원의 JOB_ID가 갱신되고 커미션 필드는 NULL로 설정되어야 합니다.

```
UPDATE employees
SET job_id = 'IT_PROG', commission_pct = NULL
WHERE employee_id = 114;
```

주: COPY_EMP 테이블은 EMPLOYEES 테이블과 동일한 데이터를 가지고 있습니다.

subquery를 사용하여 두 개의 열 갱신

사원 113의 직무와 급여를 사원 205와 일치하도록 갱신합니다.

```
UPDATE employees
SET job_id = (SELECT job_id
               FROM employees
               WHERE employee_id = 205),
    salary = (SELECT salary
               FROM employees
               WHERE employee_id = 205)
WHERE employee_id = 113;
1 rows updated
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

subquery를 사용하여 두 개의 열 갱신

여러 subquery를 작성하여 UPDATE 문의 SET 절에서 여러 열을 갱신할 수 있습니다. 구문은 다음과 같습니다.

```
UPDATE table
SET column =
    (SELECT column
     FROM table
     WHERE condition)
[ ,
column =
    (SELECT column
     FROM table
     WHERE condition)]
[WHERE condition] ;
```

슬라이드의 예제를 다음과 같이 작성할 수도 있습니다.

```
UPDATE employees
SET (job_id, salary) = (SELECT job_id, salary
                        FROM employees
                        WHERE employee_id = 205)
WHERE employee_id = 113;
```

다른 테이블을 기반으로 행 갱신

UPDATE 문에서 **subquery**를 사용하여 다른 테이블의 값을 기반으로 테이블의 행 값을 갱신합니다.

```
UPDATE copy_emp
SET department_id = (SELECT department_id
                      FROM employees
                      WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
                 FROM employees
                 WHERE employee_id = 200);

1 rows updated
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다른 테이블을 기반으로 행 갱신

UPDATE 문에서 subquery를 사용하여 테이블의 값을 갱신할 수 있습니다. 슬라이드의 예제는 EMPLOYEES 테이블의 값을 기반으로 COPY_EMP 테이블을 갱신합니다. 이 예제는 사원 200의 직무 ID를 가진 모든 사원의 부서 번호를 사원 100의 현재 부서 번호로 변경합니다.

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT**, **ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

테이블에서 행 제거

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

DEPARTMENTS 테이블에서 행을 삭제합니다.

	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

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블에서 행 제거

슬라이드의 그림에 나오는 것처럼 DEPARTMENTS 테이블에서 Contracting 부서가 제거되었습니다. 이 과정에서 DEPARTMENTS 테이블에 대한 제약 조건이 위반되지 않은 것으로 가정합니다.

DELETE 문

DELETE 문을 사용하여 테이블에서 기존 행을 제거할 수 있습니다.

```
DELETE [FROM]    table  
[WHERE           condition];
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DELETE 문 구문

DELETE 문을 사용하여 테이블에서 기존 행을 제거할 수 있습니다.

구문 설명:

table 테이블의 이름입니다.

condition 삭제할 행을 식별하며 열 이름, 표현식, 상수, subquery 및 비교 연산자로 구성됩니다.

주: 삭제된 행이 없는 경우 SQL Developer의 Script Output 탭에 "0 rows deleted"라는 메시지가 반환됩니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "DELETE" 관련 섹션을 참조하십시오.

테이블에서 행 삭제

- **WHERE** 절을 지정하면 특정 행이 삭제됩니다.

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 rows deleted
```

- **WHERE** 절을 생략하면 테이블의 모든 행이 삭제됩니다.

```
DELETE FROM copy_emp;
22 rows deleted
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블에서 행 삭제

DELETE 문에서 WHERE 절을 지정하여 특정 행을 삭제할 수 있습니다. 슬라이드의 첫번째 예제는 DEPARTMENTS 테이블에서 Accounting 부서를 삭제합니다. SELECT 문을 사용하여 삭제된 행을 표시하는 방식으로 삭제 작업을 확인할 수 있습니다.

```
SELECT *
FROM departments
WHERE department_name = 'Finance';
0 rows selected
```

그러나 WHERE 절을 생략하면 테이블의 모든 행이 삭제됩니다. 슬라이드의 두번째 예제는 WHERE 절이 지정되지 않았기 때문에 COPY_EMP 테이블에서 모든 행을 삭제합니다.

예제:

WHERE 절에서 식별된 행을 제거합니다.

```
DELETE FROM employees WHERE employee_id = 114;
1 rows deleted
DELETE FROM departments WHERE department_id IN (30, 40);
2 rows deleted
```

다른 테이블을 기반으로 행 삭제

DELETE 문에서 **subquery**를 사용하여 다른 테이블의 값을 기반으로 테이블에서 행을 제거합니다.

```
DELETE FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name
           LIKE '%Public%');

1 rows deleted
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다른 테이블을 기반으로 행 삭제

subquery를 사용하여 다른 테이블의 값을 기반으로 테이블에서 행을 삭제할 수 있습니다. 슬라이드의 예제는 부서 이름에 문자열 Public이 포함된 부서에서 근무하는 모든 사원을 삭제합니다. subquery는 문자열 Public을 포함한 부서 이름을 기반으로 DEPARTMENTS 테이블을 검색하여 부서 번호를 찾습니다. 그러면 subquery가 main query에 부서 번호를 전달하고 main query는 이 부서 번호를 기반으로 EMPLOYEES 테이블에서 데이터 행을 삭제합니다.

TRUNCATE 문

- 테이블은 빈 상태로, 테이블 구조는 그대로 남겨둔 채 테이블에서 모든 행을 제거합니다.
- DML 문이 아니라 DDL(데이터 정의어) 문이므로 쉽게 연두할 수 없습니다.
- 구문:

```
TRUNCATE TABLE table_name;
```

- 예제:

```
TRUNCATE TABLE copy_emp;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

TRUNCATE 문

테이블을 비우는 효율적인 방법은 TRUNCATE 문을 사용하는 것입니다.

TRUNCATE 문을 사용하면 테이블이나 클러스터에서 신속하게 모든 행을 제거할 수 있습니다. 다음과 같은 이유로 TRUNCATE 문으로 행을 제거하는 방법이 DELETE 문으로 행을 제거하는 방법보다 빠릅니다.

- TRUNCATE 문은 DDL(데이터 정의어) 문이며 롤백 정보를 생성하지 않습니다. 롤백 정보는 이 단원 뒷부분에서 다룹니다.
- 테이블 자르기는 테이블의 삭제 트리거를 유발하지 않습니다.

테이블이 참조 무결성 제약 조건의 상위 요소인 경우 해당 테이블을 truncate할 수 없습니다. TRUNCATE 문을 실행하기 전에 제약 조건을 비활성화해야 합니다. 제약 조건 비활성화는 후속 단원에서 다룹니다.

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT, ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 트랜잭션

데이터베이스 트랜잭션은 다음 중 하나로 구성됩니다.

- 데이터를 일관되게 변경하는 여러 **DML** 문
- 하나의 **DDL** 문
- 하나의 **DCL(데이터 제어어)** 문



Copyright © 2007, Oracle. All rights reserved.

데이터베이스 트랜잭션

Oracle 서버는 트랜잭션에 준하여 데이터 일관성이 유지되도록 합니다. 트랜잭션은 데이터를 변경할 때 많은 유연성과 제어 기능을 제공하며 유저 프로세스 failure 또는 시스템 failure 시 데이터 일관성을 보장합니다.

트랜잭션은 데이터를 일관되게 변경하는 여러 DML 문으로 구성됩니다. 예를 들어, 두 계좌 간에 자금을 이체할 때 한 계좌의 차변과 다른 계좌의 대변에서 변경되는 금액이 동일해야 합니다. 이 두 작업은 동시에 성공하거나 실패해야 하며 차변 없이 대변만 커밋될 수는 없습니다.

트랜잭션 유형

유형	설명
DML(데이터 조작어)	Oracle 서버가 단일 엔티티나 논리적 작업 단위로 취급하는 임의 개수의 DML 문으로 구성됩니다.
DDL(데이터 정의어)	하나의 DDL 문으로만 구성됩니다.
DCL(데이터 제어어)	하나의 DCL 문으로만 구성됩니다.

데이터베이스 트랜잭션: 시작과 종료

- 첫번째 **DML SQL** 문이 실행될 때 시작됩니다.
- 다음 상황 중 하나가 발생하면 종료됩니다.
 - **COMMIT** 또는 **ROLLBACK** 문 실행
 - **DDL** 또는 **DCL** 문 실행(자동 커밋)
 - 유저가 **SQL Developer** 또는 **SQL*Plus**를 종료
 - 시스템 중단

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 트랜잭션: 시작과 종료

데이터베이스 트랜잭션이 언제 시작되고 종료되는지 알아보겠습니다.

트랜잭션은 첫번째 DML 문을 만나면 시작되고 다음 상황 중 하나가 발생하면 종료됩니다.

- COMMIT 또는 ROLLBACK 문 실행
- CREATE와 같은 DDL 문 실행
- DCL 문 실행
- 유저가 SQL Developer 또는 SQL*Plus를 종료
- 시스템 failure 또는 시스템 중단

한 트랜잭션이 끝나면 다음 실행 가능한 SQL 문이 다음 트랜잭션을 자동으로 시작합니다.

DDL 문 또는 DCL 문은 자동으로 커밋되기 때문에 트랜잭션을 암시적으로 종료합니다.

COMMIT 및 ROLLBACK 문의 이점

COMMIT 및 ROLLBACK 문을 사용할 경우 다음과 같은 이점이 있습니다.

- 데이터 일관성 보장
- 변경 사항을 영구 적용하기 전에 데이터 변경 사항 검토
- 논리적으로 관련된 작업 그룹화

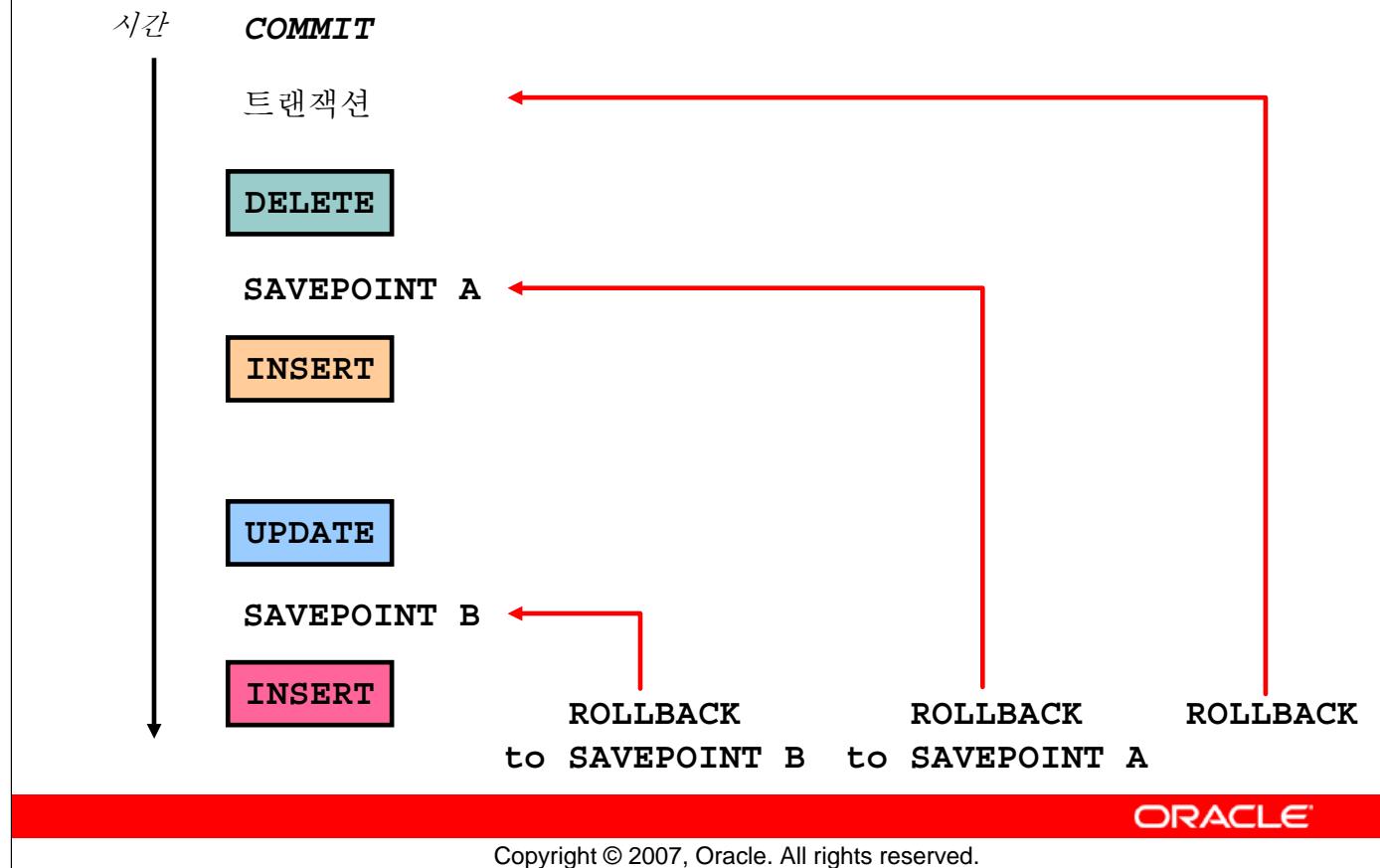


Copyright © 2007, Oracle. All rights reserved.

COMMIT 및 ROLLBACK 문의 이점

COMMIT 및 ROLLBACK 문을 사용하면 데이터 변경 사항의 영구 적용 여부를 제어할 수 있습니다.

명시적 트랜잭션 제어문



Copyright © 2007, Oracle. All rights reserved.

명시적 트랜잭션 제어문

COMMIT, SAVEPOINT 및 ROLLBACK 문을 사용하여 트랜잭션 논리를 제어할 수 있습니다.

명령문	설명
COMMIT	보류 중인 모든 데이터 변경 사항을 영구적으로 적용하여 현재 트랜잭션을 종료합니다.
SAVEPOINT name	현재 트랜잭션 내에 저장점을 표시합니다.
ROLLBACK	ROLLBACK은 보류 중인 모든 데이터 변경 사항을 폐기하여 현재 트랜잭션을 종료합니다.
ROLLBACK TO SAVEPOINT name	ROLLBACK TO SAVEPOINT는 현재 트랜잭션을 지정된 저장점으로 롤백합니다. 이에 따라 롤백 중인 저장점 이후에 생성된 변경 사항 및/또는 저장점은 폐기됩니다. TO SAVEPOINT 절을 생략하면 ROLLBACK 문은 전체 트랜잭션을 롤백합니다. 저장점은 논리적이기 때문에 생성한 저장점을 리스트로 표시할 수 없습니다.

주: SAVEPOINT로 커밋할 수는 없습니다. SAVEPOINT는 ANSI 표준 SQL이 아닙니다.

변경 사항을 표시자로 롤백

- **SAVEPOINT** 문을 사용하여 현재 트랜잭션에서 표시자를 생성합니다.
- **ROLLBACK TO SAVEPOINT** 문을 사용하여 해당 표시자로 롤백합니다.

```
UPDATE...
SAVEPOINT update_done;
SAVEPOINT update_done succeeded.

INSERT...
ROLLBACK TO update_done;
ROLLBACK TO succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

변경 사항을 표시자로 롤백

트랜잭션을 작은 세션으로 나누는 **SAVEPOINT** 문을 사용하여 현재 트랜잭션에서 표시자를 생성할 수 있습니다. 그런 다음 **ROLLBACK TO SAVEPOINT** 문을 사용하여 해당 표시자에 보류 중인 변경 사항을 폐기할 수 있습니다.

이전의 저장점과 동일한 이름으로 두번째 저장점을 만들면 이전의 저장점이 삭제됩니다.

암시적 트랜잭션 처리

- 다음 상황에서는 자동 커밋이 발생합니다.
 - DDL 문 실행
 - DCL 문 실행
 - COMMIT 또는 ROLLBACK 문을 명시적으로 실행하지 않은 채 SQL Developer 또는 SQL*Plus를 정상적으로 종료
- SQL Developer 또는 SQL*Plus가 비정상적으로 종료되거나 시스템 failure가 발생된 경우 자동 롤백이 발생합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

암시적 트랜잭션 처리

상태	상황
자동 커밋	DDL 문 또는 DCL 문 실행 COMMIT 또는 ROLLBACK 명령을 명시적으로 실행하지 않은 채 SQL Developer 또는 SQL*Plus를 정상적으로 종료
자동 롤백	SQL Developer 또는 SQL*Plus가 비정상적으로 종료되거나 시스템 failure가 발생

주: SQL*Plus에서는 AUTOCOMMIT 명령에 대한 설정을 ON 또는 OFF로 변경할 수 있습니다. ON으로 설정된 경우 각 DML 문은 실행되는 즉시 커밋됩니다. 변경 사항은 롤백할 수 없습니다. OFF로 설정된 경우 COMMIT 문은 계속 명시적으로 실행될 수 있습니다. 또한 COMMIT 문은 DDL 문이 실행되거나 SQL*Plus가 종료되는 경우에도 실행됩니다. SQL Developer에서는 SET AUTOCOMMIT ON/OFF 명령을 실행하지 않고 건너뜁니다. Autocommit 환경 설정을 활성화한 경우에만 SQL Developer 정상 종료 시 DML 문이 커밋됩니다. Autocommit를 활성화하려면 다음을 수행합니다.

- Tools 메뉴에서 Preferences를 선택합니다. Preferences 대화상자에서 Database를 확장하고 Worksheet Parameters를 선택합니다.
- 오른쪽 창의 SQL Worksheet 옵션에서 Autocommit를 선택합니다. OK를 누릅니다.

암시적 트랜잭션 처리(계속)

시스템 failure

시스템 failure로 트랜잭션이 중단될 경우 전체 트랜잭션은 자동으로 롤백됩니다. 이렇게 하면 오류로 인해 데이터가 원하지 않는 방식으로 변경되는 것을 방지할 수 있으며 마지막 커밋 작업 시의 상태로 테이블을 반환합니다. Oracle 서버는 이러한 방식으로 테이블의 무결성을 보호합니다.

SQL Developer에서 세션을 정상적으로 종료하려면 File 메뉴에서 Exit를 선택하고 SQL*Plus에서 세션을 정상적으로 종료하려면 프롬프트에서 EXIT 명령을 입력합니다. window를 닫는 것은 비정상적인 종료로 간주됩니다.

COMMIT 또는 ROLLBACK 전의 데이터 상태

- 이전의 데이터 상태를 복구할 수 있습니다.
- 현재 유저는 **SELECT** 문을 사용하여 **DML** 작업의 결과를 검토할 수 있습니다.
- 다른 유저는 현재 유저가 실행한 **DML** 문의 결과를 볼 수 없습니다.
- 영향을 받는 행이 잠기므로 다른 유저가 영향을 받는 행의 데이터를 변경할 수 없습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

COMMIT 또는 ROLLBACK 전의 데이터 상태

트랜잭션 동안의 모든 데이터 변경 사항은 트랜잭션이 커밋되기 전까지는 임시적인 상태입니다.

다음은 COMMIT 또는 ROLLBACK 명령이 실행되기 전의 데이터 상태에 대한 설명입니다.

- 데이터 조작 작업은 기본적으로 데이터베이스 버퍼에 영향을 주기 때문에 이전의 데이터 상태를 복구할 수 있습니다.
- 현재 유저는 테이블을 query하여 데이터 조작 작업의 결과를 확인할 수 있습니다.
- 다른 유저는 현재 유저의 데이터 조작 작업 결과를 확인할 수 없습니다. Oracle 서버는 읽기 일관성을 제공하여 각 유저가 마지막 커밋된 시점의 상태대로 데이터를 볼 수 있도록 보장합니다.
- 영향을 받는 행이 잠기므로 다른 유저가 영향을 받는 행의 데이터를 변경할 수 없습니다.

COMMIT 후의 데이터 상태

- 데이터 변경 사항이 데이터베이스에 저장됩니다.
- 이전의 데이터 상태를 겹쳐씁니다.
- 모든 유저가 결과를 확인할 수 있습니다.
- 영향을 받는 행의 **Lock**이 해제되어 이러한 행을 다른 유저가 조작할 수 있습니다.
- 모든 저장점이 지워집니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

COMMIT 후의 데이터 상태

COMMIT 명령을 사용하여 보류 중인 모든 변경 사항을 영구 적용합니다. 다음은 COMMIT 문이 실행된 이후에 발생하는 상황입니다.

- 데이터 변경 사항이 데이터베이스에 기록됩니다.
- 더 이상 정상적인 SQL query를 통해 이전의 데이터 상태를 사용할 수 없습니다.
- 모든 유저가 트랜잭션 결과를 확인할 수 있습니다.
- 영향 받는 행의 Lock이 해제되고 다른 유저가 해당 행에 대해 새로운 데이터 변경 작업을 수행할 수 있습니다.
- 모든 저장점이 지워집니다.

데이터 커밋

- 데이터를 변경합니다.

```
DELETE FROM employees
WHERE employee_id = 99999;
1 rows deleted

INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
1 rows inserted
```

- 변경 사항을 커밋합니다.

```
COMMIT;
COMMIT succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터 커밋

슬라이드의 예제에서는 EMPLOYEES 테이블에서 행이 삭제되고 DEPARTMENTS 테이블에 새 행이 삽입됩니다. 이러한 변경 사항은 COMMIT 문을 실행하여 저장됩니다.

예제:

DEPARTMENTS 테이블에서 부서 290 및 300을 제거하고 EMPLOYEES 테이블의 행을 갱신합니다. 데이터 변경 사항을 저장합니다.

```
DELETE FROM departments
WHERE department_id IN (290, 300);

UPDATE employees
SET department_id = 80
WHERE employee_id = 206;

COMMIT;
```

ROLLBACK 후의 데이터 상태

ROLLBACK 문을 사용하여 보류 중인 모든 변경 사항을 폐기합니다.

- 데이터 변경 사항이 언두됩니다.
- 이전의 데이터 상태가 복원됩니다.
- 영향 받는 행의 Lock이 해제됩니다.

```
DELETE FROM copy_emp;  
ROLLBACK ;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ROLLBACK 후의 데이터 상태

ROLLBACK 문을 사용하여 보류 중인 모든 변경 사항을 폐기합니다. 다음과 같은 결과가 발생합니다.

- 데이터 변경 사항이 언두됩니다.
- 이전의 데이터 상태가 복원됩니다.
- 영향 받는 행의 Lock이 해제됩니다.

ROLLBACK 후의 데이터 상태: 예제

```
DELETE FROM test;
25,000 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test WHERE id = 100;
1 row deleted.

SELECT * FROM test WHERE id = 100;
No rows selected.

COMMIT;
Commit complete.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ROLLBACK 후의 데이터 상태: 예제

TEST 테이블에서 레코드를 제거하려다 실수로 테이블을 모두 비웠습니다. 그러나 이러한 실수를 바로잡고 적절한 명령문을 다시 실행하여 데이터 변경 사항을 영구 적용할 수 있습니다.

명령문 레벨 롤백

- 단일 **DML** 문을 실행하는 중에 오류가 발생하면 해당 명령문만 롤백됩니다.
- **Oracle** 서버는 저장점을 암시적으로 구현합니다.
- 다른 모든 변경 사항은 보존됩니다.
- 유저는 **COMMIT** 또는 **ROLLBACK** 문을 실행하여 트랜잭션을 명시적으로 종료해야 합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

명령문 레벨 롤백

명령문 실행 오류가 감지되면 암시적 롤백으로 트랜잭션의 일부가 폐기될 수 있습니다. 트랜잭션 실행 중 단일 DML 문에 오류가 발생하면 명령문이 미치는 영향은 명령문 레벨 롤백에 의해 언두되지만, 트랜잭션에서 이전 DML 문에 의해 변경된 사항은 폐기되지 않습니다. 이러한 변경 사항은 유저가 명시적으로 커밋하거나 롤백할 수 있습니다.

Oracle 서버는 DDL 문을 실행하기 전과 실행한 후에 암시적 커밋을 실행합니다. 따라서 DDL 문이 성공적으로 실행되지 않더라도 서버가 이미 커밋을 실행했기 때문에 이전 명령문을 롤백할 수 없습니다.

COMMIT 또는 ROLLBACK 문을 실행하여 트랜잭션을 명시적으로 종료합니다.

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT**, **ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

읽기 일관성

- 읽기 일관성은 데이터에 대한 일관성 있는 뷰를 보장합니다.
- 한 유저가 변경한 사항이 다른 유저가 변경한 사항과 충돌하지 않습니다.
- 읽기 일관성은 동일한 데이터에 대해 다음 사항을 보장합니다.
 - 읽는 사람은 쓰는 사람의 작업이 완료되기를 기다릴 필요가 없습니다.
 - 쓰는 사람은 읽는 사람의 작업이 완료되기를 기다릴 필요가 없습니다.
 - 쓰는 사람은 다른 쓰는 사람의 작업이 완료되기를 기다려야 합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

읽기 일관성

데이터베이스 유저는 다음과 같은 두 가지 방식으로 데이터베이스를 액세스합니다.

- 읽기 작업(SELECT 문)
- 쓰기 작업(INSERT, UPDATE, DELETE 문)

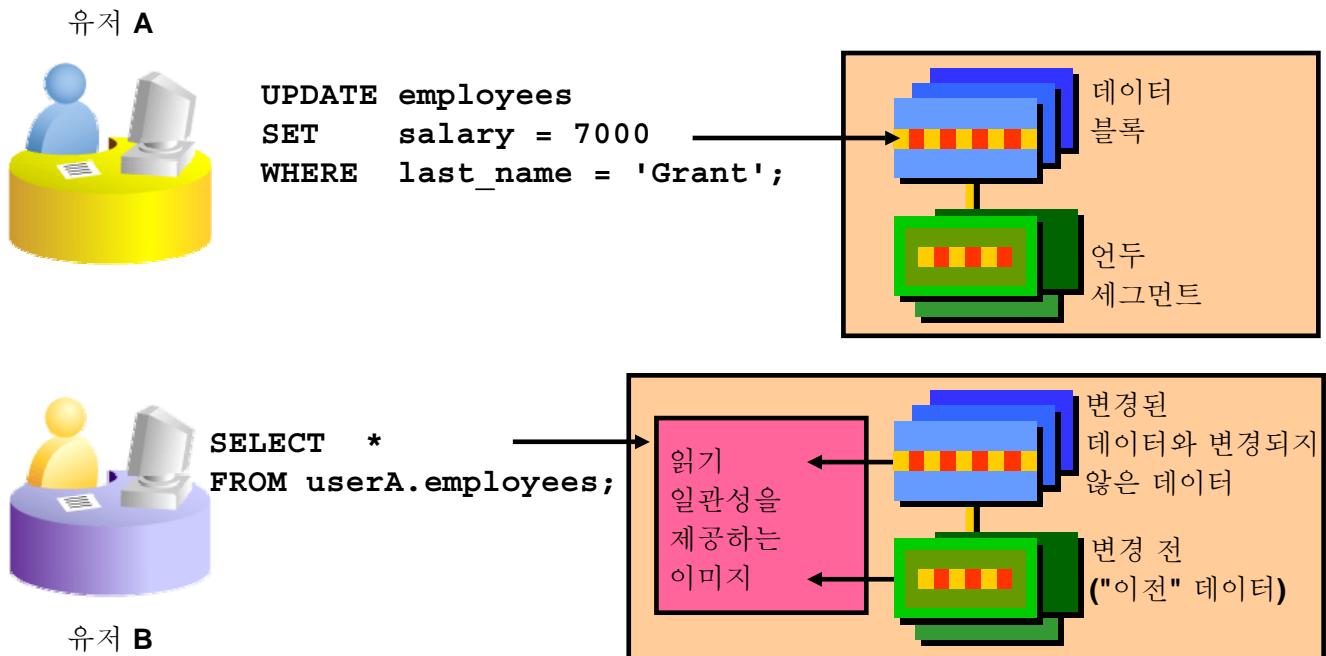
읽기 일관성을 유지할 경우 다음과 같은 효과를 볼 수 있습니다.

- 데이터베이스 기록자와 읽는 사람이 일관된 데이터 뷰를 볼 수 있습니다.
- 현재 변경 중인 데이터는 읽는 사람이 볼 수 없습니다.
- 쓰는 사람은 데이터베이스 변경 작업을 일관되게 수행할 수 있습니다.
- 쓰는 사람이 변경한 사항은 다른 쓰는 사람이 변경하는 사항과 충돌하지 않습니다.

읽기 일관성의 목적은 유저가 DML 작업을 시작하기 전 마지막 커밋 시점의 상태대로 데이터를 볼 수 있도록 보장하는 것입니다.

주: 동일한 유저가 서로 다른 세션으로 로그인할 수 있습니다. 여러 세션에서 유저가 동일한 경우에도 각 세션은 위에 설명된 방식으로 읽기 일관성을 유지합니다.

읽기 일관성 구현



ORACLE

Copyright © 2007, Oracle. All rights reserved.

읽기 일관성 구현

읽기 일관성은 자동으로 구현됩니다. 읽기 일관성은 데이터베이스의 일부 복사본을 언두 세그먼트에 보관합니다. 읽기 일관성을 제공하는 이미지는 테이블에서 커밋된 데이터와 언두 세그먼트에서 아직 커밋되지 않은 변경 중인 이전 데이터로 구성됩니다.

데이터베이스에 대한 삽입, 생성, 삭제 등의 작업을 수행하면 Oracle 서버에서는 데이터가 변경되기 전에 데이터 복사본을 만들어 이를 언두 세그먼트에 기록합니다.

변경 작업을 직접 수행한 유저 이외의 모든 읽는 사람에게는 변경 전 상태의 데이터베이스, 즉 언두 세그먼트의 데이터 "스냅샷"이 표시됩니다.

변경 사항이 데이터베이스에 커밋되기 전에는 해당 데이터를 수정하는 유저만 데이터베이스에서 변경 사항을 볼 수 있습니다. 그 밖의 모든 사람은 언두 세그먼트의 스냅샷을 볼 수 있습니다. 이렇게 하면 해당 데이터를 읽는 사람들은 현재 변경 중이 아닌 일관된 데이터를 보게 됩니다.

DML 문이 커밋되는 경우 커밋이 완료된 후에 SELECT 문을 실행하는 모든 사람이 데이터베이스에 대한 변경 사항을 볼 수 있습니다. 언두 세그먼트 파일의 이전 데이터가 점유한 공간은 다시 사용할 수 있도록 해제됩니다.

트랜잭션을 롤백하면 변경 내용이 언두됩니다.

- 언두 세그먼트에 들어 있던 이전의 원래 데이터가 테이블에 다시 기록됩니다.
- 모든 유저는 데이터베이스를 트랜잭션 시작 전의 상태로 보게 됩니다.

단원 내용

- 테이블에 새 행 추가
 - **INSERT** 문
- 테이블의 데이터 변경
 - **UPDATE** 문
- 테이블에서 행 제거
 - **DELETE** 문
 - **TRUNCATE** 문
- **COMMIT**, **ROLLBACK** 및 **SAVEPOINT**를 사용하여 데이터베이스 트랜잭션 제어
- 읽기 일관성
- **SELECT** 문의 **FOR UPDATE** 절

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

SELECT 문의 FOR UPDATE 절

- EMPLOYEES 테이블에서 job_id가 SA REP인 행을 잠금니다.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA REP'
FOR UPDATE
ORDER BY employee_id;
```

- ROLLBACK 또는 COMMIT를 실행하는 경우에만 Lock이 해제됩니다.
- 다른 유저가 접근 행을 SELECT 문에서 잠그려고 하면 데이터베이스는 해당 행을 사용할 수 있을 때까지 기다린 다음 SELECT 문의 결과를 반환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

SELECT 문의 FOR UPDATE 절

일부 레코드를 query하기 위해 데이터베이스에 대해 SELECT 문을 실행하는 경우 선택된 행에는 Lock이 걸리지 않습니다. 일반적으로 특정 시점에서 잠겨 있는 레코드 수는 기본적으로 절대 최소치로 유지되므로, 즉 변경되었지만 아직 커밋되지 않은 레코드만 잠기므로 SELECT 문에 의해 선택된 행이 잠기지 않아야 합니다. 레코드가 잠긴 경우에도 다른 유저는 변경되기 전의 모습, 즉 데이터의 "이전 이미지"로 해당 레코드를 읽을 수 있습니다. 그러나 때로는 사용 중인 프로그램에서 레코드 집합을 변경하지 않았어도 해당 레코드 집합을 잠그려는 경우가 있습니다. Oracle은 이러한 잠금을 수행할 수 있도록 SELECT 문의 FOR UPDATE 절을 제공합니다.

SELECT...FOR UPDATE 문을 실행하면 RDBMS(관계형 데이터베이스 관리 시스템)는 SELECT 문에 의해 식별된 모든 행에 대해 배타적 행 레벨 잠금(exclusive row-level lock)을 자동으로 획득하여 "해당 유저만 변경할 수 있는" 레코드를 보유하게 됩니다. 해당 유저가 ROLLBACK 또는 COMMIT를 수행하기 전까지 다른 유저는 이러한 레코드를 변경할 수 없습니다.

FOR UPDATE 절에 선택적 키워드인 NOWAIT를 추가하면 다른 유저가 해당 테이블을 접근 경우 Oracle 서버가 기다리지 않도록 할 수 있습니다. 이 경우 사용 중인 프로그램이나 SQL Developer 환경으로 제어가 즉시 반환되므로 유저는 다른 작업을 수행하거나 일정 시간 동안 기다린 후 다시 시도할 수 있습니다. NOWAIT 절을 사용하지 않으면 해당 테이블을 사용할 수 있을 때까지, 즉 다른 유저가 COMMIT 또는 ROLLBACK 명령을 실행하여 Lock이 해제될 때까지 프로세스가 차단됩니다.

FOR UPDATE 절: 예제

- **SELECT** 문에서 여러 테이블에 대해 **FOR UPDATE** 절을 사용할 수 있습니다.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

- **EMPLOYEES** 테이블 및 **DEPARTMENTS** 테이블의 행이 잠깁니다.
- **FOR UPDATE OF column_name**을 사용하여 변경할 열을 지정하면 특정 테이블의 해당 행만 잠깁니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

FOR UPDATE 절: 예제

슬라이드의 예제에서 명령문은 EMPLOYEES 테이블에서 JOB_ID가 ST_CLERK으로 설정되고 LOCATION_ID가 1500으로 설정된 행을 잠그고 DEPARTMENTS 테이블에서는 LOCATION_ID가 1500으로 설정된 행을 잠깁니다.

FOR UPDATE OF *column_name*을 사용하여 변경할 열을 지정할 수 있습니다. 선택된 행에서 변경할 수 있는 열은 FOR UPDATE 절의 OF 리스트에 나열된 열로 제한되지 않습니다. 모든 행에는 여전히 Lock이 걸려 있으므로 query에 FOR UPDATE만 지정하고 OF 키워드 뒤에 하나 이상의 열을 포함하지 않는 경우 데이터베이스는 FROM 절에 나열된 모든 테이블에서 식별된 모든 행을 잠깁니다.

다음 명령문은 EMPLOYEES 테이블에서 LOCATION_ID 1500에 거주하는 ST_CLERK을 값으로 가지는 행만 잠깁니다. DEPARTMENTS 테이블의 행은 잠기지 않습니다.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
ORDER BY e.employee_id;
```

FOR UPDATE 절: 예제(계속)

다음 예제에서 데이터베이스는 행을 사용할 수 있을 때까지 5초간 기다린 후 유저에게 제어를 반환합니다.

```
SELECT employee_id, salary, commission_pct, job_id  
FROM employees  
WHERE job_id = 'SA_REP'  
FOR UPDATE WAIT 5  
ORDER BY employee_id;
```

요약

이 단원에서는 다음 명령문의 사용 방법에 대해 설명했습니다.

함수	설명
INSERT	테이블에 새 행 추가
UPDATE	테이블의 기존 행 수정
DELETE	테이블에서 기존 행 제거
TRUNCATE	테이블에서 모든 행 제거
COMMIT	보류 중인 변경 사항을 영구적으로 적용
SAVEPOINT	저장점 표시자로 롤백하는 데 사용
ROLLBACK	보류 중인 모든 데이터 변경 사항을 폐기
SELECT 의 FOR UPDATE 절	SELECT query 에 의해 식별된 행 잠그기



Copyright © 2007, Oracle. All rights reserved.

요약

이 단원에서는 INSERT, UPDATE, DELETE, TRUNCATE 문을 사용하여 오라클 데이터베이스의 데이터를 조작하는 방법과 COMMIT, SAVEPOINT, ROLLBACK 문을 사용하여 데이터 변경 사항을 제어하는 방법을 배웠습니다. 또한 SELECT 문의 FOR UPDATE 절을 사용하여 다른 유저가 변경할 수 없도록 행을 잠그는 방법도 배웠습니다.

Oracle 서버는 항상 데이터를 일관되게 볼 수 있도록 보장한다는 사실을 기억하십시오.

연습 9: 개요

이 연습에서는 다음 내용을 다룹니다.

- 테이블에 행 삽입
- 테이블에서 행 생성 및 삭제
- 트랜잭션 제어



Copyright © 2007, Oracle. All rights reserved.

연습 9: 개요

이 연습에서는 MY_EMPLOYEE 테이블에 행을 추가하고, 테이블에서 데이터를 생성 및 삭제하고, 트랜잭션을 제어하는 과정을 다룹니다. 스크립트를 실행하여 MY_EMPLOYEE 테이블을 생성합니다.

연습 9

HR 부서에서 사원 데이터를 삽입, 갱신 및 삭제하는 SQL 문을 작성해 달라고 합니다. HR 부서에 명령문을 제공하기에 앞서 프로토타입으로 MY_EMPLOYEE 테이블을 사용합니다.

주: 모든 DML 문의 경우에는 Run Script 아이콘 또는 [F5]를 사용하여 query를 실행합니다. 이렇게 하면 Script Output 탭 페이지에서 피드백 메시지를 볼 수 있습니다. SELECT 쿼리의 경우 계속해서 Execute Statement 아이콘을 사용하거나 [F9]를 누르면 Results 탭 페이지에서 형식이 지정된 출력을 볼 수 있습니다.

MY_EMPLOYEE 테이블에 데이터를 삽입합니다.

1. 이 연습에 사용되는 MY_EMPLOYEE 테이블을 생성하려면 lab_09_01.sql 스크립트의 명령문을 실행합니다.
2. 열 이름을 식별하도록 MY_EMPLOYEE 테이블의 구조를 기술합니다.

DESCRIBE MY_EMPLOYEE		
Name	Null	Type
ID	NOT NULL	NUMBER(4)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID		VARCHAR2(8)
SALARY		NUMBER(9,2)

3. 다음 예제 데이터의 첫번째 데이터 행을 MY_EMPLOYEE 테이블에 추가하는 INSERT 문을 작성합니다. INSERT 절에 열을 나열하지 마십시오. 아직 모든 행을 삽입하지 마십시오.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

4. 앞의 리스트에서 가져온 예제 데이터의 두번째 행으로 MY_EMPLOYEE 테이블을 채웁니다. 이번에는 INSERT 절에 명시적으로 열을 나열합니다.

연습 9(계속)

5. 테이블에 추가한 내용을 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1 Patel	Ralph	rpatel	895
2	2 Dancs	Betty	bdancs	860

6. 나머지 행을 MY_EMPLOYEE 테이블에 로드하는 INSERT 문을 재사용 가능한 동적 스크립트 파일에 작성합니다. 스크립트는 모든 열(ID, LAST_NAME, FIRST_NAME, USERID, SALARY)에 대해 프롬프트를 표시해야 합니다. 이 스크립트를 lab_09_06.sql 파일에 저장합니다.
 7. 작성한 스크립트에서 INSERT 문을 실행하여 3단계에 나열된 예제 데이터의 다음 두 행으로 테이블을 채웁니다.
 8. 테이블에 추가한 내용을 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1 Patel	Ralph	rpatel	895
2	2 Dancs	Betty	bdancs	860
3	3 Biri	Ben	bbiri	1100
4	4 Newman	Chad	cnewman	750

9. 데이터 추가 내용이 영구적으로 적용되도록 합니다.

MY_EMPLOYEE 테이블에서 데이터를 생성하고 삭제합니다.

10. 사원 3의 성을 Drexler로 변경합니다.
 11. 급여가 \$900 미만인 모든 사원에 대해 급여를 \$1000로 변경합니다.
 12. 테이블에 대한 변경 사항을 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1 Patel	Ralph	rpatel	1000
2	2 Dancs	Betty	bdancs	1000
3	3 Drexler	Ben	bbiri	1100
4	4 Newman	Chad	cnewman	1000

13. MY_EMPLOYEE 테이블에서 Betty Dancs를 삭제합니다.
 14. 테이블에 대한 변경 사항을 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1 Patel	Ralph	rpatel	1000
2	3 Drexler	Ben	bbiri	1100
3	4 Newman	Chad	cnewman	1000

연습 9(계속)

15. 보류 중인 모든 변경 사항을 커밋합니다.

MY_EMPLOYEE 테이블에 대한 데이터 트랜잭션을 제어합니다.

16. 6단계에서 작성한 스크립트의 명령문을 사용하여 3단계에 나열된 예제 데이터의 마지막 행으로 테이블을 채웁니다. 스크립트의 명령문을 실행합니다.

17. 테이블에 추가한 내용을 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Drexler	Ben	bbiri	1100
3	Newman	Chad	cnewman	1000
4	Ropeburn	Audrey	aropebur	1550

18. 트랜잭션 처리의 중간 지점에 표시합니다.

19. **MY_EMPLOYEE** 테이블에서 모든 행을 삭제합니다.

20. 테이블이 비어 있는지 확인합니다.

21. 이전의 INSERT 작업을 삭제하지 않은 채로 가장 최근의 DELETE 작업을 삭제합니다.

22. 새 행이 여전히 원래 상태를 유지하는지 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	1000
2	Drexler	Ben	bbiri	1100
3	Newman	Chad	cnewman	1000
4	Ropeburn	Audrey	aropebur	1550

23. 데이터 추가 내용이 영구적으로 적용되도록 합니다.

시간 여유가 있을 경우 다음 연습을 완료하십시오.

24. 이름의 첫번째 문자와 성의 앞부분 일곱 문자를 연결하여 USERID를 자동으로 생성하도록 `lab_09_06.sql` 스크립트를 수정합니다. 생성된 USERID는 소문자여야 합니다. 따라서 이 스크립트는 유저에게 USERID를 입력하도록 요구하지 않아야 합니다. `lab_09_24.sql`이라는 파일에 이 스크립트를 저장합니다.

25. `lab_09_24.sql` 스크립트를 실행하여 다음 레코드를 삽입합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

26. 새 행에 올바른 USERID가 추가되었는지 확인합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

10

DDL 문을 사용하여
테이블 생성 및 관리

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 기본 데이터베이스 객체 분류
- 테이블 구조 검토
- 열에 사용할 수 있는 데이터 유형 나열
- 간단한 테이블 생성
- 테이블 생성 시 제약 조건이 생성되는 방식 설명
- 스키마 객체의 작동 방식 설명

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 DDL(데이터 정의어) 문을 소개합니다. 간단한 테이블을 생성, 변경 및 제거하는 방법을 배웁니다. DDL에서 사용할 수 있는 데이터 유형을 살펴보고 스키마 개념이 소개됩니다. 이 단원에서 제약 조건에 대해 설명합니다. DML 작업 동안 제약 조건 위반으로 생성되는 예외 메시지에 대해 살펴봅니다.

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 객체

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
인덱스	일부 query 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 객체

오라클 데이터베이스는 여러 데이터 구조를 포함할 수 있습니다. 각 구조는 데이터베이스 개발 과정의 구축 단계에서 생성될 수 있도록 데이터베이스 설계 시에 기본적인 틀을 만들어야 합니다.

- **테이블:** 데이터를 저장합니다.
- **뷰:** 하나 이상의 테이블에 있는 데이터의 부분 집합입니다.
- **시퀀스:** 숫자 값을 생성합니다.
- **인덱스:** 일부 query의 성능을 향상시킵니다.
- **동의어:** 객체에 다른 이름을 부여합니다.

Oracle 테이블 구조

- 유저가 데이터베이스를 사용하고 있는 경우를 포함하여 언제든지 테이블을 생성할 수 있습니다.
- 테이블의 크기를 지정할 필요는 없습니다. 크기는 최종적으로 데이터베이스에 전체적으로 할당된 공간의 크기로 정의됩니다. 하지만 시간이 지남에 따라 테이블이 사용할 공간을 산정하는 것이 중요합니다.
- 테이블 구조는 온라인으로 수정할 수 있습니다.

주: 이밖에도 많은 데이터베이스 객체가 있지만 이 과정에서는 다루지 않습니다.

이름 지정 규칙

테이블 이름 및 열 이름:

- 문자로 시작해야 합니다.
- 길이는 **1–30**자 사이여야 합니다.
- **A–Z, a–z, 0–9, _, \$, #**만 포함할 수 있습니다.
- 동일한 유저가 소유한 다른 객체의 이름과 중복되면 안됩니다.
- **Oracle** 서버 예약어는 사용할 수 없습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

이름 지정 규칙

오라클 데이터베이스 객체에 대한 이름 지정 표준 규칙에 따라 데이터베이스 테이블 및 열 이름을 다음과 같이 지정합니다.

- 테이블 이름과 열 이름은 문자로 시작하고 길이는 1–30자 사이여야 합니다.
- 이름에는 문자 A–Z, a–z, 0–9, _(밑줄), \$, #(규칙에 위배되지 않지만 권장되지 않음)만 사용할 수 있습니다.
- 이름은 동일한 Oracle 서버 유저가 소유한 다른 객체의 이름과 중복되면 안됩니다.
- Oracle 서버 예약어는 이름으로 사용할 수 없습니다.
 - 따옴표로 묶은 식별자를 사용하여 객체의 이름을 나타낼 수도 있습니다. 따옴표로 묶은 식별자는 앞과 뒤에 큰 따옴표(“)를 붙입니다. 따옴표로 묶은 식별자를 사용하여 스키마 객체에 이름을 지정하면 해당 객체를 참조할 때마다 큰 따옴표를 사용해야 합니다. 권장 사항은 아니지만 따옴표로 묶은 식별자를 예약어로 사용할 수 있습니다.

이름 지정 지침

테이블 및 기타 데이터베이스 객체에 대해 설명형 이름을 사용합니다.

주: 이름은 대소문자를 구분하지 않습니다. 예를 들어, EMPLOYEES는 eMPloyees 또는 eMpLOYEES와 동일한 이름으로 간주됩니다. 그러나 따옴표로 묶은 식별자는 대소문자를 구분합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 *Schema Object Names and Qualifiers* 관련 섹션을 참조하십시오.

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE®

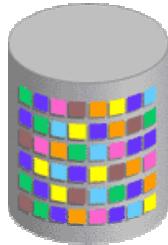
Copyright © 2007, Oracle. All rights reserved.

CREATE TABLE 문

- 다음 사항이 필요합니다.
 - CREATE TABLE 권한
 - 저장 영역

```
CREATE TABLE [schema.]table
(column datatype [DEFAULT expr] [, ...]);
```

- 다음을 지정합니다.
 - 테이블 이름
 - 열 이름, 열 데이터 유형 및 열 크기



ORACLE

Copyright © 2007, Oracle. All rights reserved.

CREATE TABLE 문

SQL CREATE TABLE 문을 실행하여 데이터를 저장할 테이블을 생성합니다. DDL 문 중 하나인 이 명령문은 오라클 데이터베이스 구조를 생성, 수정 또는 제거하는 데 사용되는 SQL 문의 일종입니다. 이러한 명령문은 데이터베이스에 즉시 적용되고 데이터 딕셔너리에 정보를 기록하기도 합니다.

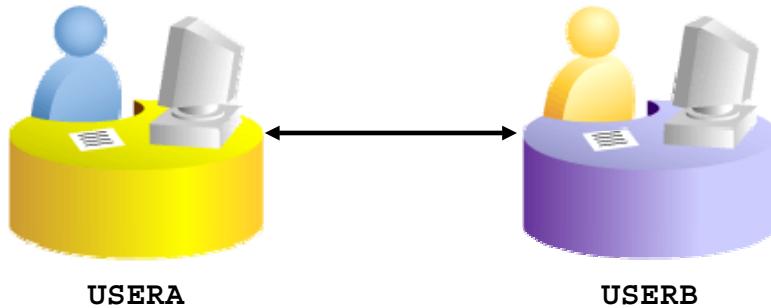
테이블을 생성하려면 유저는 CREATE TABLE 권한이 있어야 하며 객체를 생성할 저장 영역이 있어야 합니다. DBA(데이터베이스 관리자)는 DCL(데이터 제어어) 문을 사용하여 유저에게 권한을 부여합니다.

구문 설명:

<i>schema</i>	소유자 이름과 동일합니다.
<i>table</i>	테이블의 이름입니다.
<i>DEFAULT expr</i>	INSERT 문에서 값이 생략된 경우 기본값을 지정합니다.
<i>column</i>	열 이름입니다.
<i>datatype</i>	열의 데이터 유형 및 길이입니다.

다른 유저의 테이블 참조

- 다른 유저가 소유한 테이블은 유저의 스키마에 없습니다.
- 이러한 테이블에는 소유자의 이름을 접두어로 사용해야 합니다.



```
SELECT *
FROM userB.employees;
```

```
SELECT *
FROM userA.employees;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

다른 유저의 테이블 참조

스키마는 데이터의 논리적 구조 또는 스키마 객체의 모음입니다. 스키마는 데이터베이스 유저가 소유하며 해당 유저와 동일한 이름을 가지고 있습니다. 각 유저는 단일 스키마를 소유합니다.

스키마 객체에는 테이블, 뷰, 동의어, 시퀀스, 내장 프로시저, 인덱스, 클러스터 및 데이터베이스 링크가 있으며 SQL을 사용하여 스키마 객체를 생성 및 조작할 수 있습니다.

테이블이 유저 소유가 아닌 경우 테이블 이름 앞에 소유자의 이름을 추가해야 합니다. 예를 들어, USERA 및 USERB라는 스키마가 있고 양쪽에 모두 EMPLOYEES 테이블이 있는 경우 USERB가 소유한 EMPLOYEES 테이블에 USERA가 액세스하려면 USERA는 다음과 같이 테이블 이름 앞에 스키마 이름을 추가해야 합니다.

```
SELECT *
FROM userb.employees;
```

USERA가 소유한 EMPLOYEES 테이블에 USERB가 액세스하려면 USERB는 다음과 같이 테이블 이름 앞에 스키마 이름을 추가해야 합니다.

```
SELECT *
FROM usera.employees;
```

DEFAULT 옵션

- 삽입 시 열의 기본값을 지정합니다.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- 리터럴 값, 표현식 또는 **SQL** 함수는 올바른 값입니다.
- 다른 열의 이름이나 의사 열은 잘못된 값입니다.
- 기본 데이터 유형은 열 데이터 유형과 일치해야 합니다.

```
CREATE TABLE hire_dates
  (id          NUMBER(8),
   hire_date DATE DEFAULT SYSDATE);
```

```
CREATE TABLE succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DEFAULT 옵션

테이블을 정의할 때 열에 기본값이 제공되도록 DEFAULT 옵션을 사용하여 지정할 수 있습니다. 이 옵션은 열에 대한 값이 없는 행이 삽입되는 경우 해당 열에 null 값이 입력되는 것을 방지합니다. 기본값으로 리터럴, 표현식 또는 SYSDATE 및 USER와 같은 SQL 함수를 사용할 수 있지만 다른 열의 이름이나 NEXTVAL 또는 CURRVAL과 같은 의사 열은 사용할 수 없습니다. 기본 표현식은 열의 데이터 유형과 일치해야 합니다.

다음 예제를 살펴보십시오.

```
INSERT INTO hire_dates values(45, NULL);
```

위의 명령문에서는 기본값 대신 null 값을 삽입합니다.

```
INSERT INTO hire_dates(id) values(35);
```

위의 명령문에서는 HIRE_DATE 열에 SYSDATE를 삽입합니다.

주: SQL Developer에서 DDL 문을 실행하려면 Run Script 아이콘 또는 [F5]를 누릅니다. 피드백 메시지가 Script Output 탭 페이지에 표시됩니다.

테이블 생성

- 테이블 생성:

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc         VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
```

- 테이블 생성 확인:

```
DESCRIBE dept
```

Name	Null	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 생성

슬라이드의 예제에서는 네 개의 열(DEPTNO, DNAME, LOC, CREATE_DATE)을 가진 DEPT 테이블을 생성합니다. CREATE_DATE 열은 기본값을 가지고 있습니다. INSERT 문에 값이 제공되지 않으면 시스템 날짜가 자동으로 삽입됩니다.

테이블이 생성되었는지 확인하려면 DESCRIBE 명령을 실행합니다.

테이블 생성은 DDL 문 작업이기 때문에 이 명령문이 실행되면 자동 커밋이 발생합니다.

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**,
FOREIGN KEY, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

데이터 유형

데이터 유형	설명
VARCHAR2 (<i>size</i>)	가변 길이 문자 데이터
CHAR (<i>size</i>)	고정 길이 문자 데이터
NUMBER (<i>p, s</i>)	가변 길이 숫자 데이터
DATE	날짜 및 시간 값
LONG	가변 길이 문자 데이터(최대 2GB)
CLOB	문자 데이터(최대 4GB)
RAW 및 LONG RAW	원시 이진 데이터
BLOB	바이너리 데이터(최대 4GB)
BFILE	외부 파일에 저장된 바이너리 데이터(최대 4GB)
ROWID	테이블에 있는 행의 고유한 주소를 나타내는 base-64 숫자 체계

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터 유형

테이블에 대해 열을 식별할 때 열의 데이터 유형을 제공해야 합니다. 다음과 같은 여러 데이터 유형을 사용할 수 있습니다.

데이터 유형	설명
VARCHAR2 (<i>size</i>)	가변 길이 문자 데이터(최대 <i>size</i> 를 지정해야 합니다. 최소 <i>size</i> 는 1이고 최대 <i>size</i> 는 4,000입니다.)
CHAR [(<i>size</i>)]	길이가 <i>size</i> 바이트인 고정 길이 문자 데이터(기본 및 최소 <i>size</i> 는 1이며 최대 <i>size</i> 는 2,000입니다.)
NUMBER [(<i>p, s</i>)]	자릿수가 <i>p</i> 이고 소수점 이하 자릿수가 <i>s</i> 인 숫자(자릿수는 십진 숫자의 총 수이고 소수점 이하 자릿수는 소수점 오른쪽에 있는 숫자의 수입니다. 자릿수는 1~38까지 될 수 있고 소수점 이하 자릿수는 -84~127까지 될 수 있습니다.)
DATE	기원전 년 1월 1일부터 서기 년 12월 31일 사이의 가장 가까운 초 단위에 대한 날짜 및 시간 값
LONG	가변 길이 문자 데이터(최대 2GB)
CLOB	문자 데이터(최대 4GB)

데이터 유형(계속)

데이터 유형	설명
RAW(<i>size</i>)	길이가 <i>size</i> 인 원시 바이너리 데이터(최대 <i>size</i> 를 지정해야 합니다. 최대 <i>size</i> 는 2,000입니다.)
LONG RAW	가변 길이의 원시 바이너리 데이터(최대 2GB)
BLOB	바이너리 데이터(최대 4GB)
BFILE	외부 파일에 저장된 바이너리 데이터(최대 4GB)
ROWID	테이블에 있는 행의 고유한 주소를 나타내는 base-64 숫자 체계

지침

- LONG 열은 subquery를 사용하여 테이블을 생성할 때 복사되지 않습니다.
- LONG 열은 GROUP BY 또는 ORDER BY 절에 포함될 수 없습니다.
- 각 테이블당 하나의 LONG 열만 사용할 수 있습니다.
- LONG 열에 대해 제약 조건을 정의할 수 없습니다.
- LONG 열 대신 CLOB 열을 사용할 수 있습니다.

Datetime 데이터 유형

다음과 같은 여러 **Datetime** 데이터 유형을 사용할 수 있습니다.

데이터 유형	설명
TIMESTAMP	소수 표시 초 단위의 날짜
INTERVAL YEAR TO MONTH	년, 월 간격으로 저장됨
INTERVAL DAY TO SECOND	일, 시, 분, 초 간격으로 저장됨



ORACLE

Copyright © 2007, Oracle. All rights reserved.

Datetime 데이터 유형

데이터 유형	설명
TIMESTAMP	소수 표시 초 단위의 날짜로 시간을 저장할 수 있습니다. 소수 표시 초 값뿐만 아니라 DATE 데이터 유형의 년, 월, 일, 시, 분 및 초를 저장합니다. 이 데이터 유형에는 WITH TIMEZONE, WITH LOCALTIMEZONE과 같은 여러 변형이 있습니다.
INTERVAL YEAR TO MONTH	년, 월 간격으로 시간을 저장할 수 있습니다. 년과 월로만 유효한 부분을 나타내는 두 datetime 값 사이의 차이를 표시하는 데 사용됩니다.
INTERVAL DAY TO SECOND	일, 시, 분, 초 간격으로 시간을 저장할 수 있습니다. 두 datetime 값 사이의 정밀한 차이를 나타내는 데 사용됩니다.

주: 이러한 Datetime 데이터 유형은 Oracle9i 이상의 릴리스에서 사용할 수 있습니다. Datetime 데이터 유형은 *Oracle Database 11g: SQL Fundamentals II* 과정의 "Managing Data in Different Time Zones" 단원에서 자세히 설명합니다.

또한 datetime 데이터 유형에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*의 *TIMESTAMP Datatype*, *INTERVAL YEAR TO MONTH Datatype* 및 *INTERVAL DAY TO SECOND Datatype* 항목을 참조하십시오.

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL, PRIMARY KEY, FOREIGN KEY, CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

제약 조건 포함

- 제약 조건은 테이블 레벨에서 규칙을 강제 적용합니다.
- 제약 조건은 테이블에 종속 관계가 있는 경우 삭제를 방지합니다.
- 유효한 제약 조건 유형은 다음과 같습니다.
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건

Oracle 서버는 제약 조건을 사용하여 테이블에 잘못된 데이터를 입력하는 것을 방지합니다.

제약 조건을 사용하여 다음을 수행할 수 있습니다.

- 테이블에서 행을 삽입, 갱신 또는 삭제할 때마다 테이블의 데이터에 규칙을 강제 적용합니다.
작업이 성공하려면 제약 조건이 충족되어야 합니다.
- 다른 테이블과 종속 관계가 있는 경우 테이블 삭제를 방지합니다.
- Oracle Developer와 같은 Oracle 툴에 규칙을 제공합니다.

데이터 무결성 제약 조건

제약 조건	설명
NOT NULL	열에 null 값을 포함할 수 없음을 지정합니다.
UNIQUE	테이블의 모든 행에 대해 값이 고유해야 하는 열 또는 열 조합을 지정합니다.
PRIMARY KEY	테이블의 각 행을 고유하게 식별합니다.
FOREIGN KEY	특정 테이블의 열과 참조 테이블의 열 간에 참조 무결성을 설정하고 적용하여 한 테이블의 값이 다른 테이블의 값과 일치하도록 합니다.
CHECK	참이어야 하는 조건을 지정합니다.

제약 조건 지침

- 유저가 제약 조건의 이름을 지정하거나 **Oracle** 서버가 **SYS_Cn** 형식을 사용하여 이름을 생성할 수 있습니다.
- 다음 시점 중 하나에서 제약 조건을 생성합니다.
 - 테이블이 생성되는 시점
 - 테이블 생성 후
- 열 또는 테이블 레벨에서 제약 조건을 정의합니다.
- 데이터 딕셔너리에서 제약 조건을 확인합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건 지침

모든 제약 조건은 데이터 딕셔너리에 저장됩니다. 제약 조건에 의미 있는 이름을 제공하면 쉽게 참조할 수 있습니다. 제약 조건 이름은 표준 객체 이름 지정 규칙을 따라야 하며 예외적으로 동일한 유저가 소유한 다른 객체와 이름이 같지 않아야 한다는 규칙은 적용되지 않습니다. 제약 조건의 이름을 지정하지 않으면 Oracle 서버가 **SYS_Cn** 형식으로 이름을 생성합니다. 여기서 *n*은 제약 조건 이름에 고유하게 사용되는 정수입니다.

테이블 생성 시 또는 테이블이 생성된 후에 제약 조건을 정의할 수 있습니다. 제약 조건은 열 또는 테이블 레벨에서 정의할 수 있습니다. 가능한 면에서 테이블 레벨의 제약 조건은 열 레벨의 제약 조건과 동일합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "Constraints" 관련 섹션을 참조하십시오.

제약 조건 정의

- 구문:

```
CREATE TABLE [schema.]table
    (column datatype [DEFAULT expr]
     [column_constraint] ,
     ...
     [table_constraint][,...]);
```

- 열 레벨 제약 조건 구문:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- 테이블 레벨 제약 조건 구문:

```
column, ...
[CONSTRAINT constraint_name] constraint_type
(column, ...),
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건 정의

슬라이드는 테이블 생성 시 제약 조건을 정의하는 구문을 제공합니다. 열 레벨이나 테이블 레벨에서 제약 조건을 생성할 수 있습니다. 열 레벨에서 정의되는 제약 조건은 열이 정의될 때 포함됩니다. 테이블 레벨 제약 조건은 테이블 정의의 끝에서 정의되며 제약 조건이 괄호로 묶인 열을 참조해야 합니다. 두 제약 조건은 구문에서 주된 차이점을 보이지만 가능한 면에서 열 레벨 제약 조건은 테이블 레벨 제약 조건과 동일합니다.

NOT NULL 제약 조건은 열 레벨에서 정의되어야 합니다.

두 개 이상의 열에 적용되는 제약 조건은 테이블 레벨에서 정의되어야 합니다.

구문 설명:

<i>schema</i>	소유자 이름과 동일합니다.
<i>table</i>	테이블의 이름입니다.
DEFAULT <i>expr</i>	INSERT 문에서 값이 생략된 경우 사용할 기본값을 지정합니다.
<i>column</i>	열 이름입니다.
<i>datatype</i>	열의 데이터 유형 및 길이입니다.
<i>column_constraint</i>	열 정의의 일부인 무결성 제약 조건입니다.
<i>table_constraint</i>	테이블 정의의 일부인 무결성 제약 조건입니다.

제약 조건 정의

- 열 레벨 제약 조건 예제:

```
CREATE TABLE employees(
    employee_id NUMBER(6)
        CONSTRAINT emp_emp_id_pk PRIMARY KEY,
    first_name VARCHAR2(20),
    ...);
```

1

- 테이블 레벨 제약 조건 예제:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    first_name       VARCHAR2(20),
    ...
    job_id           VARCHAR2(10) NOT NULL,
    CONSTRAINT emp_emp_id_pk
        PRIMARY KEY (EMPLOYEE_ID));
```

2

ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건 정의(계속)

제약 조건은 대개 테이블과 동시에 생성됩니다. 제약 조건은 테이블 생성 후에 테이블에 추가할 수 있고 일시적으로 비활성화할 수도 있습니다.

슬라이드의 예제에서는 모두 EMPLOYEES 테이블의 EMPLOYEE_ID 열에 Primary key 제약 조건을 생성합니다.

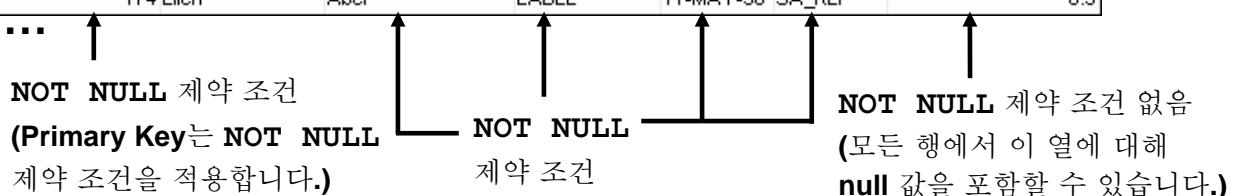
1. 첫번째 예제는 열 레벨 구문을 사용하여 제약 조건을 정의합니다.
2. 두번째 예제는 테이블 레벨 구문을 사용하여 제약 조건을 정의합니다.

Primary key 제약 조건에 대한 자세한 내용은 이 단원의 뒷부분에서 제공합니다.

NOT NULL 제약 조건

열에 null 값이 허용되지 않도록 보장합니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	(null)
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	(null)
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	(null)
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	(null)
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	(null)
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	(null)
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	(null)
141	Trenna	Rajs	TRAJS	17-OCT-95	ST_CLERK	(null)
142	Curtis	Davies	CDAVIES	29-JAN-97	ST_CLERK	(null)
143	Randall	Matos	RMATOS	15-MAR-98	ST_CLERK	(null)
144	Peter	Vargas	PVARGAS	09-JUL-98	ST_CLERK	(null)
149	Eleni	Zlotkey	EZLOTKEY	29-JAN-00	SA_MAN	0.2
174	Ellen	Abel	EABEL	11-MAY-96	SA_REP	0.3



ORACLE

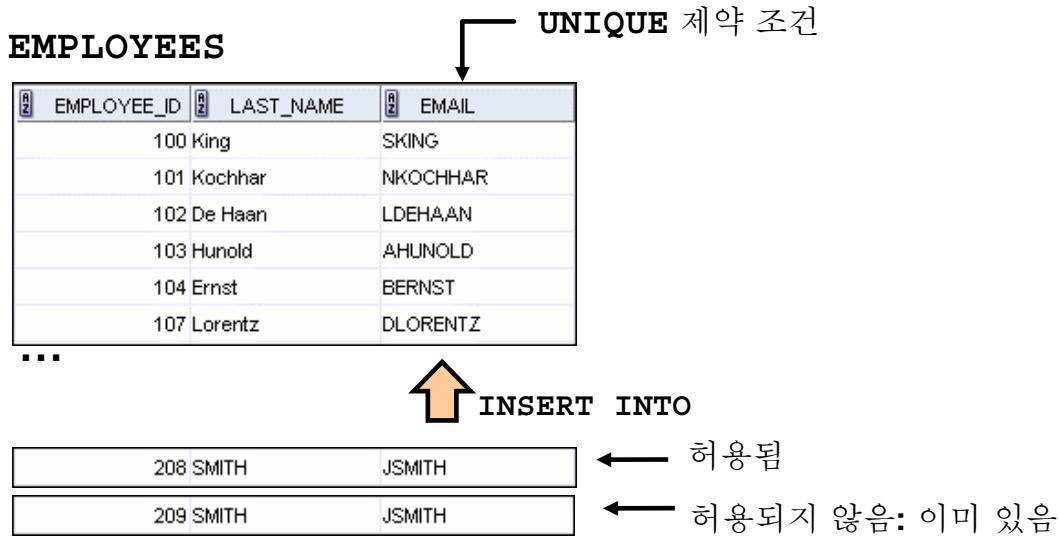
Copyright © 2007, Oracle. All rights reserved.

NOT NULL 제약 조건

NOT NULL 제약 조건은 열이 null 값을 포함하지 않도록 보장합니다. NOT NULL 제약 조건이 없는 열은 기본적으로 null 값을 포함할 수 있습니다. NOT NULL 제약 조건은 열 레벨에서 정의되어야 합니다. EMPLOYEES 테이블에서 EMPLOYEE_ID 열은 Primary key로 정의되었으므로 NOT NULL을 상속합니다. 그렇지 않으면 LAST_NAME, EMAIL, HIRE_DATE 및 JOB_ID 열에 NOT NULL 제약 조건이 적용됩니다.

주: Primary key 제약 조건은 이 단원의 뒷부분에서 자세히 설명합니다.

UNIQUE 제약 조건



ORACLE

Copyright © 2007, Oracle. All rights reserved.

UNIQUE 제약 조건

UNIQUE key 무결성 제약 조건에서는 하나의 열 또는 여러 열에 있는 모든 값(키)이 고유해야 합니다. 즉, 테이블의 두 행은 지정된 열 또는 열 집합에서 중복된 값을 가질 수 없습니다.

UNIQUE key 제약 조건 정의에 포함된 열 또는 열 집합을 *unique key*라고 합니다. UNIQUE 제약 조건은 두 개 이상의 열로 구성되며 이러한 열 그룹을 *조합 unique key*라고 합니다.

UNIQUE 제약 조건에서는 동일한 열에 대해 NOT NULL 제약 조건을 정의하지 않는 한, null을 입력할 수 있습니다. 실제로 null은 어떤 값과도 같지 않기 때문에 모든 행에서 NOT NULL 제약 조건이 없는 열에 대해 null을 포함할 수 있습니다. 열 또는 조합 UNIQUE key의 모든 열에서 null은 항상 UNIQUE 제약 조건을 충족합니다.

주: 두 개 이상의 열에 대한 UNIQUE 제약 조건의 겹친 메커니즘 때문에 부분적으로 null인 조합 UNIQUE key 제약 조건의 null이 아닌 열에서 동일한 값을 가질 수 없습니다.

UNIQUE 제약 조건

테이블 레벨 또는 열 레벨에서 정의됩니다.

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
    ...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

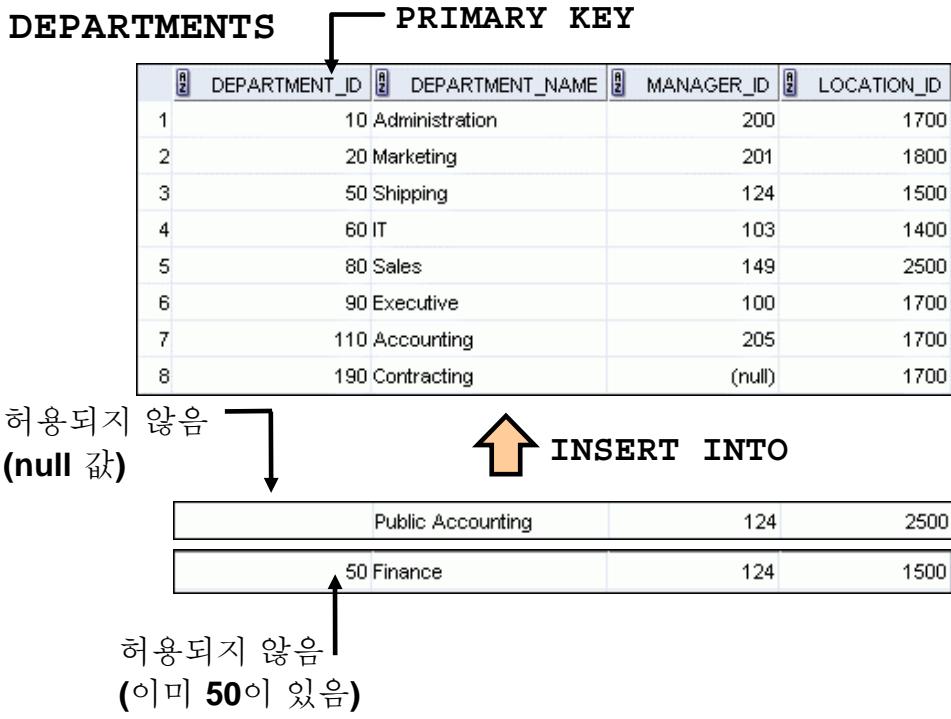
UNIQUE 제약 조건(계속)

UNIQUE 제약 조건은 열 레벨 또는 테이블 레벨에서 정의될 수 있습니다. 조합 Unique key를 생성하려는 경우 테이블 레벨에서 제약 조건을 정의합니다. 행을 고유하게 식별할 수 있는 속성이 하나도 없는 경우 조합 키가 정의됩니다. 이 경우 두 개 이상의 열로 구성된, 항상 고유하고 행을 식별할 수 있는 값의 결합인 Unique key를 사용할 수 있습니다.

슬라이드의 예제는 EMPLOYEES 테이블의 EMAIL 열에 UNIQUE 제약 조건을 적용합니다. 제약 조건의 이름은 EMP_EMAIL_UK입니다.

주: Oracle 서버는 unique key 열에서 고유 인덱스를 암시적으로 생성하는 방식으로 UNIQUE 제약 조건을 적용합니다.

PRIMARY KEY 제약 조건



ORACLE

Copyright © 2007, Oracle. All rights reserved.

PRIMARY KEY 제약 조건

PRIMARY KEY 제약 조건은 테이블에 대해 Primary key를 생성합니다. 각 테이블에는 하나의 Primary key만 생성할 수 있습니다. PRIMARY KEY 제약 조건은 테이블의 각 행을 고유하게 식별하는 열 또는 열 집합입니다. 이 제약 조건은 열 또는 열 조합에 고유성을 적용하고 Primary key에 속하는 열이 null 값을 포함할 수 없도록 합니다.

주: 고유성은 Primary key 제약 조건 정의의 일부이기 때문에 Oracle 서버는 암시적으로 Primary key 열에 고유 인덱스를 생성하는 방식으로 고유성을 적용합니다.

FOREIGN KEY 제약 조건

DEPARTMENTS

**PRIMARY
KEY**

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	Administration	200	1700
2	Marketing	201	1800
3	Shipping	124	1500
4	IT	103	1400
5	Sales	149	2500

...

EMPLOYEES

**FOREIGN
KEY**

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60

...

INSERT INTO

200 Ford	9
201 Ford	60

ORACLE

Copyright © 2007, Oracle. All rights reserved.

FOREIGN KEY 제약 조건

FOREIGN KEY(또는 참조 무결성) 제약 조건은 열 또는 열 조합을 Foreign key로 지정하고 동일한 테이블이나 다른 테이블의 Primary key 또는 Unique key와의 관계를 설정합니다.

슬라이드의 예제에서 DEPARTMENT_ID는 EMPLOYEES 테이블(종속 또는 하위 테이블)의 Foreign key로 정의되었습니다. 이 테이블은 DEPARTMENTS 테이블(참조 또는 상위 테이블)의 DEPARTMENT_ID 열을 참조합니다.

지침

- Foreign key 값은 상위 테이블의 기준 값과 일치하거나 NULL이어야 합니다.
- Foreign key는 데이터 값을 기반으로 하며 물리적 포인터가 아니라 순전히 논리적 포인터입니다.

FOREIGN KEY 제약 조건

테이블 레벨 또는 열 레벨에서 정의됩니다.

```
CREATE TABLE employees (
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
    ...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
        REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

FOREIGN KEY 제약 조건(계속)

FOREIGN KEY 제약 조건은 열 또는 테이블 제약 조건 레벨에서 정의될 수 있습니다. 조합 Foreign key는 테이블 레벨 정의를 사용하여 생성되어야 합니다.

슬라이드의 예제는 테이블 레벨 구문을 사용하여 EMPLOYEES 테이블의 DEPARTMENT_ID 열에 FOREIGN KEY 제약 조건을 정의합니다. 제약 조건의 이름은 EMP_DEPT_FK입니다.

제약 조건이 단일 열을 기반으로 하는 경우 Foreign key는 열 레벨에서도 정의될 수 있습니다. 이 구문은 FOREIGN KEY 키워드가 나타나지 않는다는 점에서 차이가 있습니다. 예를 들면 다음과 같습니다.

```
CREATE TABLE employees
(
    ...
    department_id NUMBER(4) CONSTRAINT emp_deptid_fk
        REFERENCES departments(department_id),
    ...
)
```

FOREIGN KEY 제약 조건:

키워드

- **FOREIGN KEY:** 테이블 제약 조건 레벨에서 하위 테이블의 열을 정의합니다.
- **REFERENCES:** 테이블 및 상위 테이블의 열을 식별합니다.
- **ON DELETE CASCADE:** 상위 테이블의 행이 삭제될 때 하위 테이블의 종속 행을 삭제합니다.
- **ON DELETE SET NULL:** 종속 Foreign key 값을 null로 변환합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

FOREIGN KEY 제약 조건: 키워드

Foreign key는 하위 테이블에서 정의되며 참조되는 열을 포함하는 테이블은 상위 테이블입니다.
Foreign key는 다음 키워드 조합을 사용하여 정의됩니다.

- FOREIGN KEY는 테이블 제약 조건 레벨에서 하위 테이블의 열을 정의하는 데 사용됩니다.
- REFERENCES는 테이블 및 상위 테이블의 열을 식별합니다.
- ON DELETE CASCADE는 상위 테이블의 행이 삭제될 때 하위 테이블의 종속 행도 삭제됨을 나타냅니다.
- ON DELETE SET NULL은 상위 테이블의 행이 삭제될 때 Foreign key 값이 null로 설정됨을 나타냅니다.

기본 동작을 *restrict rule*이라고 하며 참조되는 데이터의 생성이나 삭제를 허용하지 않습니다.

ON DELETE CASCADE 또는 ON DELETE SET NULL 옵션이 없으면 상위 테이블의 행을 하위 테이블에서 참조하는 경우 삭제할 수 없습니다.

CHECK 제약 조건

- 각 행이 충족해야 하는 조건을 정의합니다.
- 다음 표현식은 허용되지 않습니다.
 - CURRVAL, NEXTVAL, LEVEL, ROWNUM 의사 열에 대한 참조
 - SYSDATE, UID, USER, USERENV 함수에 대한 호출
 - 다른 행의 다른 값을 참조하는 query

```
..., salary NUMBER(2)
CONSTRAINT emp_salary_min
    CHECK (salary > 0),...
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

CHECK 제약 조건

CHECK 제약 조건은 각 행이 충족해야 하는 조건을 정의합니다. 이 조건은 다음의 경우를 제외하고 query 조건과 동일한 구성을 사용할 수 있습니다.

- CURRVAL, NEXTVAL, LEVEL, ROWNUM 의사 열에 대한 참조
- SYSDATE, UID, USER, USERENV 함수에 대한 호출
- 다른 행의 다른 값을 참조하는 query

단일 열에서 해당 정의의 열을 참조하는 여러 CHECK 제약 조건을 가질 수 있습니다. 열에 정의할 수 있는 CHECK 제약 조건의 수에는 제한이 없습니다.

CHECK 제약 조건은 열 레벨이나 테이블 레벨에서 정의될 수 있습니다.

```
CREATE TABLE employees
(
    ...
    salary NUMBER(8,2) CONSTRAINT emp_salary_min
        CHECK (salary > 0),
    ...
)
```

CREATE TABLE: 예제

```

CREATE TABLE employees
(
    employee_id      NUMBER(6)
        CONSTRAINT emp_employee_id PRIMARY KEY
    , first_name      VARCHAR2(20)
    , last_name       VARCHAR2(25)
        CONSTRAINT emp_last_name_nn NOT NULL
    , email           VARCHAR2(25)
        CONSTRAINT emp_email_nn     NOT NULL
        CONSTRAINT emp_email_uk     UNIQUE
    , phone_number    VARCHAR2(20)
    , hire_date       DATE
        CONSTRAINT emp_hire_date_nn NOT NULL
    , job_id          VARCHAR2(10)
        CONSTRAINT emp_job_nn      NOT NULL
    , salary          NUMBER(8,2)
        CONSTRAINT emp_salary_ck   CHECK (salary>0)
    , commission_pct  NUMBER(2,2)
    , manager_id      NUMBER(6)
        CONSTRAINT emp_manager_fk REFERENCES
            employees (employee_id)
    , department_id   NUMBER(4)
        CONSTRAINT emp_dept_fk      REFERENCES
            departments (department_id);
)

```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

CREATE TABLE: 예제

슬라이드의 예제에서는 HR 스키마에서 EMPLOYEES 테이블을 생성하는 데 사용되는 명령문을 보여줍니다.

제약 조건 위반

```
UPDATE employees
SET department_id = 55
WHERE department_id = 110;
```

```
Error starting at line 1 in command:
UPDATE employees
SET department_id = 55
WHERE department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA16.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

부서 55가 존재하지 않습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건 위반

열에 제약 조건이 있는 경우 제약 조건 규칙을 위반하려고 하면 오류가 반환됩니다. 예를 들어, 무결성 제약 조건이 적용된 값이 있는 레코드를 생성하려고 하면 오류가 반환됩니다.

슬라이드의 예제에서 상위 테이블 DEPARTMENTS에 부서 55가 없기 때문에 "parent key not found" 위반 오류 ORA-02291이 발생합니다.

제약 조건 위반

다른 테이블에서 **Foreign key**로 사용되는 **Primary key**를 포함한 행은 삭제할 수 없습니다.

```
DELETE FROM departments
WHERE department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE      department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA16.EMP_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

제약 조건 위반(계속)

무결성 제약 조건이 적용된 값이 있는 레코드를 삭제하려고 하면 오류가 반환됩니다.

슬라이드의 예제는 DEPARTMENTS 테이블에서 부서 60을 삭제하려고 하지만 해당 부서 번호가 EMPLOYEES 테이블에서 Foreign key로 사용되기 때문에 오류가 발생합니다. 삭제하려고 하는 상위 레코드가 하위 레코드를 가지고 있는 경우 "child record found" 위반 오류 ORA-02292가 발생합니다.

다음 명령문은 부서 70에 사원이 없기 때문에 제대로 실행됩니다.

```
DELETE FROM departments
WHERE department_id = 70;
1 rows deleted
```

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery를 사용하여 테이블 생성

- **CREATE TABLE** 문과 **AS subquery** 옵션을 결합하여 테이블을 생성하고 행을 삽입합니다.

```
CREATE TABLE table
  [(column, column...)]
AS subquery;
```

- 지정된 열 개수와 **subquery** 열 개수를 일치시킵니다.
- 열 이름과 기본값을 가진 열을 정의합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery를 사용하여 테이블 생성

테이블을 생성하는 두번째 방법은 **AS subquery** 절을 적용하는 것입니다. 이 절은 테이블을 생성하고 **subquery**에서 반환된 행을 삽입합니다.

구문 설명:

table 테이블의 이름입니다.

column 열 이름, 기본값, 무결성 제약 조건입니다.

subquery 새 테이블에 삽입될 행 집합을 정의하는 **SELECT** 문입니다.

지침

- 지정된 열 이름으로 테이블이 생성되고 **SELECT** 문으로 검색한 행이 테이블에 삽입됩니다.
- 열 정의에는 열 이름과 기본값만 포함할 수 있습니다.
- 열 사양이 주어진 경우 열 개수는 **subquery** **SELECT** 리스트의 열 개수와 같아야 합니다.
- 열 사양이 주어지지 않은 경우 테이블의 열 이름은 **subquery**의 열 이름과 동일합니다.
- 열 데이터 유형 정의 및 **NOT NULL** 제약 조건이 새 테이블로 전달됩니다. 명시적 **NOT NULL** 제약 조건만 상속됩니다. **PRIMARY KEY** 열은 **NOT NULL** 기능을 새 열에 전달하지 않습니다. 다른 제약 조건 규칙은 새 테이블로 전달되지 않습니다. 하지만 열 정의에 제약 조건을 추가할 수 있습니다.

Subquery를 사용하여 테이블 생성

```
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
       salary*12 ANNSAL,
       hire_date
  FROM employees
 WHERE department_id = 80;
```

CREATE TABLE succeeded.

DESCRIBE dept80

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

ORACLE

Copyright © 2007, Oracle. All rights reserved.

Subquery를 사용하여 테이블 생성(계속)

슬라이드의 예제에서는 부서 80에서 근무하는 모든 사원의 세부 정보가 담긴 DEPT80이라는 테이블을 생성합니다. DEPT80 테이블의 데이터는 EMPLOYEES 테이블에서 가져옵니다.

DESCRIBE 명령을 사용하여 데이터베이스 테이블이 있는지 확인하고 열 정의를 검사할 수 있습니다.

그러나 표현식을 선택할 때 열 alias를 제공해야 합니다. SALARY*12 표현식에는 alias ANNSAL이 제공됩니다. alias가 없으면 다음과 같은 오류가 발생합니다.

```
Error starting at line 1 in command:
CREATE TABLE      dept80
AS SELECT  employee_id, last_name,
salary*12,
hire_date FROM employees WHERE      department_id = 80
Error at Command Line:3 Column:6
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 -  "must name this expression with a column alias"
*Cause:
*Action:
```

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

ALTER TABLE 문

ALTER TABLE 문을 사용하여 다음을 수행합니다.

- 새 열 추가
- 기존 열 정의 수정
- 새 열에 기본값 정의
- 열 삭제
- 열 이름 바꾸기
- 읽기 전용 상태로 테이블 변경

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

ALTER TABLE 문

테이블을 생성한 후에 다음과 같은 이유로 인해 테이블 구조를 변경해야 할 수 있습니다.

- 열을 생략했습니다.
- 열 정의 또는 이름을 변경해야 합니다.
- 열을 제거하려고 합니다.
- 테이블을 읽기 전용 모드로 변경하려 합니다.

이 작업은 ALTER TABLE 문을 사용하여 수행할 수 있습니다.

읽기 전용 테이블

테이블을 읽기 전용 모드로 변경하려면 **ALTER TABLE** 구문을 사용합니다.

- 테이블을 유지 관리하는 동안 **DDL** 문 또는 **DML** 문에 의한 변경을 방지합니다.
- 읽기/쓰기 모드로 되돌립니다.

```
ALTER TABLE employees READ ONLY;
-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

Copyright © 2007, Oracle. All rights reserved.

읽기 전용 테이블

Oracle Database 11g에서는 READ ONLY를 지정하여 테이블을 읽기 전용 모드로 둘 수 있습니다.

테이블이 READ-ONLY 모드이면 테이블에 영향을 주는 DML 문 또는 SELECT ... FOR UPDATE 문을 실행할 수 없습니다. 테이블의 데이터를 수정하지 않는 한 DDL 문은 실행할 수 있습니다. 테이블과 연결된 인덱스에 대한 작업은 테이블이 READ ONLY 모드일 때 가능합니다.

읽기 전용 테이블을 읽기/쓰기 모드로 되돌리려면 READ/WRITE를 지정합니다.

주: READ ONLY 모드인 테이블을 삭제할 수 있습니다. DROP 명령은 데이터 덕셔너리에서만 실행되므로 테이블 내용에 액세스할 필요가 없습니다. 테이블에서 사용한 공간은 테이블스페이스를 읽기/쓰기로 다시 변경하기 전에는 회수되지 않으며 그 이후 블록 세그먼트 헤더 등에 필요한 변경 사항을 적용할 수 있습니다.

ALTER TABLE 문에 대한 자세한 내용은 *Oracle Database 10g SQL Fundamentals II* 과정을 참조하십시오.

단원 내용

- 데이터베이스 객체
 - 이름 지정 규칙
- **CREATE TABLE** 문:
 - 다른 유저의 테이블에 액세스
 - **DEFAULT** 옵션
- 데이터 유형
- 제약 조건 개요: **NOT NULL**, **PRIMARY KEY**, **FOREIGN KEY**, **CHECK** 제약 조건
- **subquery**를 사용하여 테이블 생성
- **ALTER TABLE**
 - 읽기 전용 테이블
- **DROP TABLE** 문

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 삭제

- 테이블을 **Recycle bin**으로 이동
- **PURGE** 절이 지정되면 테이블 및 해당 데이터를 완전히 제거
- 종속 객체 무효화 및 테이블의 객체 권한 제거

```
DROP TABLE dept80;
```

```
DROP TABLE dept80 succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

테이블 삭제

`DROP TABLE` 문은 테이블을 Recycle bin으로 이동하거나 테이블 및 해당 데이터를 데이터베이스에서 완전히 제거합니다. `PURGE` 절을 지정하지 않으면 `DROP TABLE` 문에 의해 공간이 테이블스페이스로 회수되지 않으므로 해당 공간을 다른 객체에서 사용하지 못하며 해당 공간은 유저의 공간 할당량 계산에 계속 포함됩니다. 테이블을 삭제하면 종속 객체가 무효화되며 테이블의 객체 권한이 제거됩니다.

테이블을 삭제하면 데이터베이스에서 테이블의 모든 데이터 및 해당 테이블과 연관된 모든 인덱스를 잊게 됩니다.

구문

```
DROP TABLE table [PURGE]
```

이 구문에서 `table`은 테이블 이름입니다.

지침

- 테이블에서 모든 데이터가 삭제됩니다.
- 뷰와 동의어는 그대로 남아있지만 더 이상 유효하지 않습니다.
- 보류 중인 모든 트랜잭션이 커밋됩니다.
- 테이블 생성자나 `DROP ANY TABLE` 권한을 가진 유저만 테이블을 제거할 수 있습니다.

주: 삭제된 테이블을 Recycle bin에서 복원하려면 `FLASHBACK TABLE` 문을 사용합니다. 이에 대해서는 *Oracle Database 11g: SQL Fundamentals II* 과정에서 자세히 설명합니다.

요약

이 단원에서는 **CREATE TABLE** 문을 사용하여 테이블을 생성하고 제약 조건을 포함시키는 방법을 배웠습니다.

- 기본 데이터베이스 객체 분류
- 테이블 구조 검토
- 열에 사용할 수 있는 데이터 유형 나열
- 간단한 테이블 생성
- 테이블 생성 시 제약 조건이 생성되는 방식 설명
- 스키마 객체의 작동 방식 설명

ORACLE

Copyright © 2007, Oracle. All rights reserved.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

CREATE TABLE

- **CREATE TABLE** 문을 사용하여 테이블을 생성하고 제약 조건을 포함시킵니다.
- **subquery**를 사용하여 다른 테이블을 기반으로 테이블을 생성합니다.

DROP TABLE

- 행과 테이블 구조를 제거합니다.
- 이 명령문이 실행되면 롤백할 수 없습니다.

연습 10: 개요

이 연습에서는 다음 내용을 다룹니다.

- 새 테이블 생성
- **CREATE TABLE AS** 구문을 사용하여 새 테이블 생성
- 테이블이 있는지 확인
- 읽기 전용 상태로 테이블 설정
- 테이블 삭제



Copyright © 2007, Oracle. All rights reserved.

연습 10: 개요

`CREATE TABLE` 문을 사용하여 새 테이블을 생성합니다. 데이터베이스에 새 테이블이 추가되었는지 확인합니다. 또한 테이블 상태를 `READ ONLY`로 설정한 다음 `READ/WRITE`로 복귀시키는 방법을 배웁니다.

주: 모든 DDL 및 DML 문의 경우 SQL Developer에서 query를 실행하려면 Run Script 아이콘 또는 [F5]를 누릅니다. 이렇게 하면 Script Output 탭 페이지에서 피드백 메시지를 볼 수 있습니다. SELECT 쿼리의 경우 계속해서 Execute Statement 아이콘을 누르거나 [F9]를 누르면 Results 탭 페이지에서 형식이 지정된 출력을 볼 수 있습니다.

연습 10

1. 다음 테이블 instance 차트를 기반으로 DEPT 테이블을 생성합니다. lab_10_01.sql이라는 스크립트에 명령문을 저장한 다음 해당 스크립트의 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	NAME
키 유형	Primary key	
Nulls/Unique		
FK 테이블		
FK 열		
데이터 유형	NUMBER	VARCHAR2
길이	7	25

Name	Null	Type
ID	NOT NULL	NUMBER(7)
NAME		VARCHAR2(25)

2. DEPARTMENTS 테이블의 데이터로 DEPT 테이블을 채웁니다. 필요한 열만 포함시키십시오.
 3. 다음 테이블 instance 차트를 기반으로 EMP 테이블을 생성합니다. lab_10_03.sql이라는 스크립트에 명령문을 저장한 다음 해당 스크립트의 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

열 이름	ID	LAST_NAME	FIRST_NAME	DEPT_ID
키 유형				
Nulls/Unique				
FK 테이블				DEPT
FK 열				ID
데이터 유형	NUMBER	VARCHAR2	VARCHAR2	NUMBER
길이	7	25	25	7

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

연습 10(계속)

4. EMPLOYEES 테이블의 구조에 준하여 EMPLOYEES2 테이블을 생성합니다. EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPARTMENT_ID 열만 포함시키십시오. 새 테이블의 열 이름을 각각 ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPT_ID로 지정합니다.
5. EMPLOYEES2 테이블 상태를 읽기 전용으로 변경합니다.
6. EMPLOYEES2 테이블에 다음 행을 삽입해 봅니다.

ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_ID
34	Grant	Marcie	5678	10

다음 오류 메시지가 나타납니다.

```
Error starting at line 1 in command:
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
Error at Command Line:1 Column:12
Error report:
SQL Error: ORA-12081: update operation not allowed on table "ORA16"."EMPLOYEES2"
12081. 00000 -  "update operation not allowed on table \"%s\".\"%s\""
*Cause:    An attempt was made to update a read-only materialized view.
*Action:   No action required. Only Oracle is allowed to update a
          read-only materialized view.
```

7. EMPLOYEES2 테이블을 읽기/쓰기 상태로 복귀시킵니다. 이제 동일한 행을 다시 삽입해 봅니다. 다음 메시지가 나타나야 합니다.

```
ALTER TABLE employees2 succeeded.
1 rows inserted
```

8. EMPLOYEES2 테이블을 삭제합니다.

11

기타 스키마 객체 생성

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 단순 뷰 및 복합 뷰 생성
- 뷰에서 데이터 검색
- 시퀀스 생성, 유지 관리 및 사용
- 인덱스 생성 및 유지 관리
- 전용(**private**) 및 공용(**public**) 동의어 생성



Copyright © 2007, Oracle. All rights reserved.

목표

이 단원에서는 뷰, 시퀀스, 동의어 및 인덱스 객체를 소개합니다. 뷰, 시퀀스 및 인덱스를 생성하고 사용하는 기본적인 방법을 배웁니다.

단원 내용

- 뷰 개요:
 - 뷰에서 데이터 생성, 수정 및 검색
 - 뷰에서 **DML**(데이터 조작어) 작업
 - 뷰 삭제
- 시퀀스 개요:
 - 시퀀스 생성, 사용 및 수정
 - 시퀀스 값을 캐시에 저장
 - **NEXTVAL** 및 **CURRVAL pseudocolumn**
- 인덱스 개요
 - 인덱스 생성, 삭제
- 동의어 개요
 - 동의어 생성, 삭제

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 객체

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
인덱스	데이터 검색 query 의 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

데이터베이스 객체

데이터베이스에는 테이블 외에도 기타 여러 객체가 있습니다. 이 단원에서는 뷰, 시퀀스, 인덱스 및 동의어에 대해 배웁니다.

뷰를 사용하면 테이블의 데이터를 표시하거나 숨길 수 있습니다.

많은 응용 프로그램에서는 primary key 값으로 고유 번호를 사용해야 합니다. 응용 프로그램에 이러한 요구 사항을 처리하는 코드를 작성하거나 시퀀스를 사용하여 고유 번호를 생성할 수 있습니다.

데이터 검색 query의 성능을 향상시키려면 인덱스를 생성하는 것을 고려해 보십시오. 인덱스를 사용하여 열 또는 열 모음에서 고유성을 강화할 수도 있습니다.

동의어를 사용하여 객체에 다른 이름을 지정할 수 있습니다.

뷰란?

EMPLOYEES 테이블

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
5								6000
6								4200
7								5800
8								3500
9								3100
10								2600
11								2500
12								10500
13								11000
14								8600
15								7000
16								4400
17								13000
18								6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

#	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
1	100	Steven	King	24000
2	101	Neena	Kochhar	17000
3	102	Lex	De Haan	17000
4	103	Alexander	Hunold	9000
5	104	Bruce	Ernst	6000
6	19	205 Shelley	Higgins	SHIGGINS
7	20	206 William	Gietz	WGIETZ

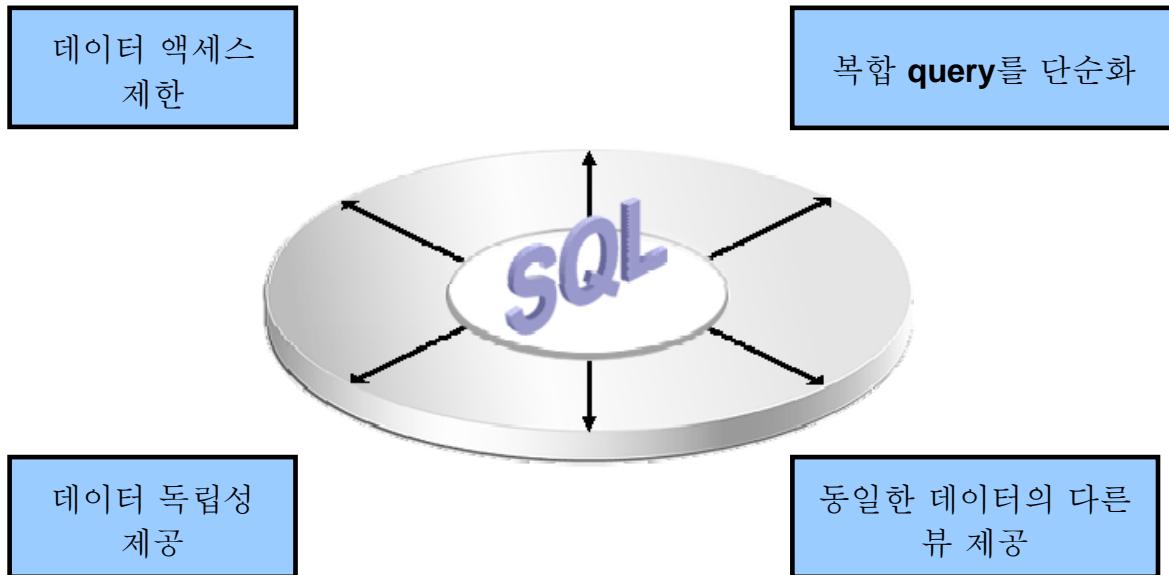
ORACLE®

Copyright © 2007, Oracle. All rights reserved.

뷰란?

테이블의 뷰를 생성하여 데이터의 논리적 하위 집합 또는 조합을 나타낼 수 있습니다. 뷰는 테이블 또는 다른 뷰를 기반으로 하는 논리적 테이블입니다. 뷰는 자체적으로 데이터를 갖고 있지 않지만 테이블의 데이터를 보거나 변경할 수 있는 window와 같습니다. 뷰의 기반이 되는 테이블을 기본 테이블이라고 합니다. 뷰는 데이터 딕셔너리에 SELECT 문으로 저장됩니다.

뷰의 이점



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

뷰의 이점

- 뷰는 테이블의 열을 선택적으로 표시하므로 데이터 액세스를 제한합니다.
- 뷰를 사용하여 복잡한 query 결과를 검색하는 간단한 query를 만들 수 있습니다. 예를 들어, 조인 문을 작성하는 방법을 몰라도 뷰를 사용하여 여러 테이블에서 정보를 query할 수 있습니다.
- 뷰는 특정 유저와 응용 프로그램에 대해 데이터 독립성을 제공합니다. 하나의 뷰를 사용하여 여러 테이블의 데이터를 검색할 수 있습니다.
- 뷰는 특정 기준에 따라 유저 그룹에게 데이터 액세스 권한을 부여합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "CREATE VIEW" 관련 섹션을 참조하십시오.

단순 뷰와 복합 뷰

기능	단순 뷰	복합 뷰
테이블 수	한 개	하나 이상
함수 포함	아니오	예
데이터 그룹 포함	아니오	예
뷰를 통해 DML 작업	예	항상은 아님

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

단순 뷰와 복합 뷰

뷰에는 단순 뷰와 복합 뷰의 두 가지 유형이 있습니다. 기본적인 차이점은 DML 작업(INSERT, UPDATE 및 DELETE)과 관련이 있습니다.

- 단순 뷰의 특징은 다음과 같습니다.
 - 하나의 테이블에서만 데이터를 가져옵니다.
 - 함수나 데이터 그룹을 포함하지 않습니다.
 - 뷰를 통해 DML 작업을 수행할 수 있습니다.
- 복합 뷰의 특징은 다음과 같습니다.
 - 여러 테이블에서 데이터를 가져옵니다.
 - 함수나 데이터 그룹을 포함합니다.
 - 뷰를 통한 DML 작업을 허용하지 않는 경우도 있습니다.

뷰 생성

- **CREATE VIEW** 문에 **subquery**를 포함시킵니다.

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view  
[(alias[, alias]...)]  
AS subquery  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY [CONSTRAINT constraint]];
```

- ㅇ) **subquery**에 복합 **SELECT** 구문을 포함할 수 있습니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

뷰 생성

CREATE VIEW 문에 **subquery**를 포함하여 뷰를 생성할 수 있습니다.

구문 설명:

OR REPLACE	뷰가 이미 있는 경우 다시 생성합니다.
FORCE	기본 테이블의 존재 여부에 관계없이 뷰를 생성합니다.
NOFORCE	기본 테이블이 있는 경우에만 뷰를 생성합니다(기본값).
view	뷰의 이름입니다.
alias	뷰의 query에서 선택한 표현식의 이름을 지정합니다(alias의 수와 뷰에서 선택한 표현식의 수가 일치해야 합니다.).
subquery	완전한 SELECT 문입니다(SELECT 리스트에 열의 alias를 사용할 수 있습니다.).
WITH CHECK OPTION	뷰에서 액세스할 수 있는 행만 삽입하거나 갱신할 수 있도록 지정합니다.
constraint	CHECK OPTION 제약 조건에 할당되는 이름입니다.
WITH READ ONLY	현재 뷰에서 DML 작업을 수행하지 못하도록 합니다.

주: SQL Developer에서 Run Script 아이콘을 누르거나 [F5]를 눌러 DDL(데이터 정의어) 문을 실행합니다. 피드백 메시지가 Script Output 탭 페이지에 표시됩니다.

뷰 생성

- 부서 80의 사원에 대한 세부 정보를 포함하는 **EMPVU80** 뷰를 생성합니다.

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
      FROM employees
     WHERE department_id = 80;
CREATE VIEW succeeded.
```

- iSQL*Plus DESCRIBE** 명령을 사용하여 뷰의 구조를 기술합니다.

```
DESCRIBE empvu80
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

뷰 생성(계속)

슬라이드의 예제는 부서 80의 각 사원에 대한 사원 번호, 성 및 급여를 포함하는 뷰를 생성합니다. DESCRIBE 명령을 사용하여 뷰의 구조를 표시할 수 있습니다.

Name	Null	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)

지침

- 뷰를 정의하는 subquery는 조인, 그룹 및 subquery를 포함한 복합 SELECT 구문을 포함할 수 있습니다.
- WITH CHECK OPTION을 사용하여 생성한 뷰에 대해 제약 조건 이름을 지정하지 않으면 시스템이 SYS_Cn 형식으로 기본 이름을 할당합니다.
- OR REPLACE 옵션을 사용하면 뷰를 삭제 및 재생성하거나 이전에 부여한 객체 권한을 다시 부여하지 않고서도 뷰의 정의를 변경할 수 있습니다.

뷰 생성

- **subquery**에서 열 **alias**를 사용하여 뷰를 생성합니다.

```
CREATE VIEW salvu50
AS SELECT employee_id ID_NUMBER, last_name NAME,
           salary*12 ANN_SALARY
      FROM employees
     WHERE department_id = 50;
CREATE VIEW succeeded.
```

- 제공된 **alias** 이름으로 이 뷰에서 열을 선택합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

뷰 생성(계속)

subquery에 열 alias를 포함하여 열 이름을 제어할 수 있습니다.

슬라이드의 예제는 부서 50의 모든 사원에 대해 alias가 ID_NUMBER인 사원 번호(EMPLOYEE_ID), alias가 NAME인 이름(LAST_NAME) 및 alias가 ANN_SALARY인 연봉(SALARY)을 포함하는 뷰를 생성합니다.

또는 CREATE 문과 SELECT subquery 사이에서 alias를 사용할 수 있습니다. 나열된 alias의 수와 subquery에서 선택한 표현식의 수가 일치해야 합니다.

```
CREATE OR REPLACE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
AS SELECT employee_id, last_name, salary*12
      FROM employees
     WHERE department_id = 50;
```

```
CREATE VIEW succeeded.
```

뷰에서 데이터 검색

```
SELECT *  
FROM salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

뷰에서 데이터 검색

테이블처럼 뷰에서도 데이터를 검색할 수 있습니다. 전체 뷰나 특정 행과 열의 내용을 표시할 수 있습니다.

뷰 수정

- **CREATE OR REPLACE VIEW** 절을 사용하여 **EMPVU80** 뷰를 수정합니다. 각 열 이름에 **alias**를 추가합니다.

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT employee_id, first_name || ' '
                                || last_name, salary, department_id
   FROM employees
 WHERE department_id = 80;
CREATE OR REPLACE VIEW succeeded.
```

- **CREATE OR REPLACE VIEW** 절의 열 **alias**는 **subquery**의 열과 동일한 순서로 나열됩니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

뷰 수정

OR REPLACE 옵션을 사용하면 이미 동일한 이름의 뷰가 있는 경우에도 뷰가 생성되므로 이전 버전의 뷰를 해당 소유자에 맞게 대체할 수 있습니다. 즉, 뷰를 삭제하고 다시 생성하고 객체 권한을 다시 부여하지 않고서도 뷰를 변경할 수 있습니다.

주: **CREATE OR REPLACE VIEW** 절에서 열 alias를 지정할 때 alias가 subquery의 열과 동일한 순서로 나열되어야 합니다.

복합 뷰 생성

그룹 함수를 포함하는 복합 뷰를 생성하여 두 테이블의 값을 표시합니다.

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT      d.department_name, MIN(e.salary),
                MAX(e.salary), AVG(e.salary)
  FROM        employees e JOIN departments d
  ON          (e.department_id = d.department_id)
 GROUP BY    d.department_name;
CREATE OR REPLACE VIEW succeeded.
```

Copyright © 2007, Oracle. All rights reserved.

복합 뷰 생성

슬라이드의 예제에서는 각 부서별로 부서 이름, 최소 급여, 최대 급여 및 평균 급여를 포함하는 복합 뷰를 생성합니다. 뷰에 다른 이름이 지정되었습니다. 뷰의 모든 열이 함수나 표현식으로부터 파생되는 경우에는 다른 이름을 지정해야 합니다.

DESCRIBE 명령을 사용하여 뷰의 구조를 볼 수 있습니다. SELECT 문을 실행하여 뷰의 내용을 표시합니다.

```
SELECT *
  FROM dept_sum_vu;
```

	NAME	MIN SAL	MAX SAL	Avg SAL
1	Administration	4400	4400	4400
2	Accounting	8300	12000	10150
3	IT	4200	9000	6400
4	Executive	17000	24000	19333.3333333333333333...
5	Shipping	2500	5800	3500
6	Sales	8600	11000	10033.3333333333333333...
7	Marketing	6000	13000	9500

뷰에 대한 **DML** 작업 수행 규칙

- 단순 뷰에서는 대개 **DML** 작업을 수행할 수 있습니다.
- 뷰에 다음 항목이 포함되어 있으면 행을 제거할 수 없습니다.
 - 그룹 함수
 - **GROUP BY** 절
 - **DISTINCT** 키워드
 - **pseudocolumn ROWNUM** 키워드



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

뷰에 대한 **DML** 작업 수행 규칙

해당 작업이 특정 규칙을 따르는 경우 뷰를 통해 데이터에 DML 작업을 수행할 수 있습니다.

뷰에 다음 항목들이 포함되어 있지 않으면 뷰에서 행을 제거할 수 있습니다.

- 그룹 함수
- **GROUP BY** 절
- **DISTINCT** 키워드
- **pseudocolumn ROWNUM** 키워드

뷰에 대한 **DML** 작업 수행 규칙

뷰에 다음 항목이 포함되어 있으면 뷰의 데이터를 수정할 수 없습니다.

- 그룹 함수
- **GROUP BY** 절
- **DISTINCT** 키워드
- **pseudocolumn ROWNUM** 키워드
- 표현식으로 정의된 열



Copyright © 2007, Oracle. All rights reserved.

뷰에 대한 **DML** 작업 수행 규칙(계속)

뷰에 이전 슬라이드에서 언급한 조건이나 표현식으로 정의된 열(예: `SALARY * 12`)이 포함되어 있지 않으면 뷰를 통해 데이터를 수정할 수 있습니다.

뷰에 대한 **DML** 작업 수행 규칙

뷰에 다음 항목이 포함되어 있으면 뷰를 통해 데이터를 추가할 수 없습니다.

- 그룹 함수
- **GROUP BY** 절
- **DISTINCT** 키워드
- **pseudocolumn ROWNUM** 키워드
- 표현식으로 정의된 열
- 뷰에서 선택되지 않은 기본 테이블의 **NOT NULL** 열



Copyright © 2007, Oracle. All rights reserved.

뷰에 대한 **DML** 작업 수행 규칙(계속)

슬라이드에 나열된 항목이 뷰에 포함되어 있지 않으면 뷰를 통해 데이터를 추가할 수 있습니다. 기본 테이블의 기본값이 없는 NOT NULL 열이 뷰에 포함되어 있으면 뷰에 데이터를 추가할 수 없습니다. 필요한 모든 값이 뷰에 있어야 합니다. 뷰를 통해 기본 테이블에 직접 값을 추가한다는 점을 기억하십시오.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "CREATE VIEW" 관련 섹션을 참조하십시오.

WITH CHECK OPTION 절 사용

- **WITH CHECK OPTION** 절을 사용하여 뷰에 수행된 **DML** 작업이 뷰 영역에만 적용되도록 할 수 있습니다.

```
CREATE OR REPLACE VIEW empvu20
AS SELECT *
  FROM employees
 WHERE department_id = 20
 WITH CHECK OPTION CONSTRAINT empvu20_ck ;
CREATE OR REPLACE VIEW succeeded.
```

- **department_id**가 20이 아닌 행을 삽입하거나 뷰에 있는 임의의 행에서 부서 번호를 갱신하려는 시도는 **WITH CHECK OPTION** 제약 조건에 위반되므로 실패합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

WITH CHECK OPTION 절 사용

뷰를 통해 참조 무결성 검사를 수행할 수 있습니다. 또한 데이터베이스 레벨에서 제약 조건을 적용할 수도 있습니다. 매우 제한적이기는 하지만 뷰를 사용하여 데이터 무결성을 보호할 수 있습니다.

WITH CHECK OPTION 절은 뷰를 통해 수행된 INSERT 및 UPDATE 문이 해당 뷰에서 선택할 수 없는 행을 생성하지 못하도록 지정합니다. 따라서 삽입 또는 갱신될 데이터에 대해 무결성 제약 조건과 데이터 유효성 검사를 강제로 적용할 수 있습니다. 뷰에서 선택하지 않은 행에서 DML 작업을 수행하려고 시도하면 제약 조건 이름이 지정된 경우 해당 이름과 함께 오류가 표시됩니다.

```
UPDATE empvu20
SET department_id = 10
WHERE employee_id = 201;
```

원인:

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
```

주: 부서 번호가 10으로 변경된 경우 뷰에 더 이상 해당 사원을 표시할 수 없으므로 행은 갱신되지 않습니다. 따라서 WITH CHECK OPTION 절을 사용할 경우 뷰에서 부서 20에 있는 사원만 볼 수 있으며 이러한 사원의 부서 번호를 뷰를 통해 변경할 수 없습니다.

DML 작업 거부

- **WITH READ ONLY** 옵션을 뷰 정의에 추가하여 **DML** 작업이 발생하지 않도록 할 수 있습니다.
- 뷰에 있는 임의의 행에서 **DML** 작업을 수행하려고 시도하면 **Oracle** 서버 오류가 발생합니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

DML 작업 거부

WITH READ ONLY 옵션을 사용하여 뷰를 생성하면 뷰에서 DML 작업이 발생하지 않도록 할 수 있습니다. 다음 슬라이드의 예제는 EMPVU10 뷰의 모든 DML 작업을 차단하도록 뷰를 수정합니다.

DML 작업 거부

```
CREATE OR REPLACE VIEW empvu10
  (employee_number, employee_name, job_title)
AS SELECT      employee_id, last_name, job_id
   FROM        employees
  WHERE      department_id = 10
    WITH READ ONLY ;
CREATE OR REPLACE VIEW succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

DML 작업 거부(계속)

읽기 전용 제약 조건이 있는 뷰에서 행을 제거하려고 시도하면 오류가 발생합니다.

```
DELETE FROM empvu10
WHERE employee_number = 200;
```

마찬가지로 읽기 전용 제약 조건이 있는 뷰를 사용하여 행을 삽입하거나 수정하려고 하면 같은 오류가 발생합니다.

Error report:

```
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

뷰 제거

뷰는 데이터베이스의 기본 테이블을 기반으로 하기 때문에 뷰를 제거해도 데이터는 손실되지 않습니다.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```

```
DROP VIEW empvu80 succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

뷰 제거

DROP VIEW 문을 사용하여 뷰를 제거할 수 있습니다. 이 명령문은 데이터베이스에서 뷰 정의를 제거합니다. 그러나 뷰를 삭제해도 뷰의 기반이 되는 테이블에는 영향을 미치지 않습니다. 이와 달리 삭제된 뷰를 기반으로 하는 뷰나 기타 응용 프로그램은 무효화됩니다. 생성자나 DROP ANY VIEW 권한을 가진 유저만 뷰를 제거할 수 있습니다.

구문 설명:

`view` 뷰의 이름입니다.

연습 11: 1부 개요

이 연습에서는 다음 내용을 다룹니다.

- 단순 뷰 생성
- 복합 뷰 생성
- **CHECK** 제약 조건을 가진 뷰 생성
- 뷰의 데이터 수정 시도
- 뷰 제거



Copyright © 2007, Oracle. All rights reserved.

연습 11: 1부 개요

이 단원의 1부 연습에서는 뷰를 생성, 사용 및 제거하는 다양한 연습을 제공합니다. 이 단원 끝부분에서 문제 1-6을 완성하십시오.

단원 내용

- 뷰 개요:
 - 뷰에서 데이터 생성, 수정 및 검색
 - 뷰에 대한 **DML** 작업
 - 뷰 삭제
- 시퀀스 개요:
 - 시퀀스 생성, 사용 및 수정
 - 시퀀스 값을 캐시에 저장
 - **NEXTVAL** 및 **CURRVAL pseudocolumn**
- 인덱스 개요
 - 인덱스 생성, 삭제
- 동의어 개요
 - 동의어 생성, 삭제

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

시퀀스

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
인덱스	일부 query 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

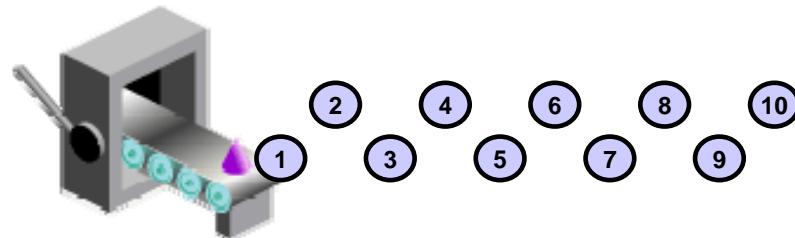
시퀀스

시퀀스는 정수 값을 생성하는 데이터베이스 객체입니다. 시퀀스를 생성한 다음 이 시퀀스를 사용하여 번호를 생성합니다.

시퀀스

시퀀스:

- 고유 번호를 자동으로 생성할 수 있습니다.
- 공유할 수 있는 객체입니다.
- **Primary key** 값을 생성하는 데 사용할 수 있습니다.
- 응용 프로그램 코드를 대체합니다.
- 시퀀스 값이 메모리에 캐시된 경우 액세스 속도가 향상됩니다.



ORACLE®

Copyright © 2007, Oracle. All rights reserved.

시퀀스(계속)

시퀀스는 유저가 생성한 데이터베이스 객체이며 정수를 생성하는 여러 유저가 공유할 수 있습니다.

시퀀스를 정의하여 고유 값을 생성하거나 동일한 번호를 재사용할 수 있습니다.

시퀀스의 일반적 용도는 각 행에 고유해야 하는 Primary key 값을 생성하는 경우입니다. 시퀀스는 내부 Oracle 루틴에 의해 생성되고 순차적으로 증가하거나 감소합니다. 이것은 시퀀스 생성 루틴을 작성하는 데 필요한 응용 프로그램 코드의 양을 줄일 수 있기 때문에 시간 절약형 객체라 할 수 있습니다.

시퀀스 번호는 테이블과 별도로 저장되고 생성됩니다. 따라서 여러 테이블에 동일한 시퀀스를 사용할 수 있습니다.

CREATE SEQUENCE 문: 구문

자동으로 일련 번호를 생성하도록 시퀀스를 정의합니다.

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [{MAXVALUE n | NOMAXVALUE}]
  [{MINVALUE n | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{CACHE n | NOCACHE}] ;
```

Copyright © 2007, Oracle. All rights reserved.

CREATE SEQUENCE 문: 구문

CREATE SEQUENCE 문을 사용하여 자동으로 일련 번호를 생성합니다.

구문 설명:

<i>sequence</i>	시퀀스 생성기의 이름입니다.
<i>INCREMENT BY n</i>	시퀀스 번호 사이의 간격을 지정하며, 여기서 <i>n</i> 은 정수입니다(이 절을 생략하면 시퀀스는 1씩 증가합니다.).
<i>START WITH n</i>	생성할 첫번째 시퀀스 번호를 지정합니다(이 절을 생략하면 시퀀스는 1부터 시작합니다.).
<i>MAXVALUE n</i>	시퀀스가 생성할 수 있는 최대값을 지정합니다.
<i>NOMAXVALUE</i>	오름차순 시퀀스의 경우 최대값 10^{27} 을, 내림차순 시퀀스의 경우 -1을 지정합니다(기본 옵션).
<i>MINVALUE n</i>	최소 시퀀스 값을 지정합니다.
<i>NOMINVALUE</i>	오름차순 시퀀스의 경우 최소값 1을, 내림차순 시퀀스의 경우 $-(10^{26})$ 을 지정합니다(기본 옵션).

시퀀스 생성

- **DEPARTMENTS** 테이블의 **Primary key**에 사용할 **DEPT_DEPTID_SEQ**라는 시퀀스를 생성합니다.
- **CYCLE** 옵션을 사용하지 마십시오.

```
CREATE SEQUENCE dept_deptid_seq
    INCREMENT BY 10
    START WITH 120
    MAXVALUE 9999
    NOCACHE
    NOCYCLE;

CREATE SEQUENCE succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

시퀀스 생성(계속)

CYCLE | NOCYCLE

최대값이나 최소값에 도달한 후에 시퀀스를 계속 생성할지 여부를 지정합니다(NOCYCLE이 기본 옵션입니다.).

CACHE n | NOCACHE

Oracle 서버가 메모리에 미리 할당하고 저장하는 값의 개수를 지정합니다(Oracle 서버는 기본적으로 20개의 값을 캐시합니다.).

슬라이드의 예제는 DEPARTMENTS 테이블의 DEPARTMENT_ID 열에 사용할 DEPT_DEPTID_SEQ라는 시퀀스를 생성합니다. 이 시퀀스는 120에서 시작하고 캐시에 저장할 수 있으며 순환하지 않습니다.

시퀀스를 사용하여 Primary key 값을 생성하는 경우 시퀀스 주기보다 더 빠르게 이전 행을 지우는 믿을 만한 방법이 없으면 CYCLE 옵션을 사용하지 마십시오.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "CREATE SEQUENCE" 관련 섹션을 참조하십시오.

주: 시퀀스는 테이블과 연관되지 않습니다. 일반적으로 시퀀스는 용도에 따라 이름을 지정해야 합니다. 하지만 이름과 상관없이 어디에나 사용할 수 있습니다.

NEXTVAL 및 CURRVAL Pseudocolumn

- **NEXTVAL**은 사용 가능한 다음 시퀀스 값을 반환합니다. 다른 유저인 경우도 포함하여 참조될 때마다 고유 값을 반환합니다.
- **CURRVAL**은 현재 시퀀스 값을 구합니다.
- **CURRVAL**이 값을 포함하기 전에 해당 시퀀스에 대해 **NEXTVAL**이 실행되어야 합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

NEXTVAL 및 CURRVAL Pseudocolumn

시퀀스를 생성한 후에 테이블에서 사용할 수 있도록 일련 번호가 생성됩니다. NEXTVAL 및 CURRVAL pseudocolumn을 사용하여 시퀀스 값을 참조합니다.

NEXTVAL pseudocolumn은 지정된 시퀀스에서 연속적인 시퀀스 번호를 추출하는 데 사용됩니다. NEXTVAL을 시퀀스 이름으로 한정해야 합니다. *sequence.NEXTVAL*을 참조할 경우 새 시퀀스 번호가 생성되고 현재 시퀀스 번호가 CURRVAL에 입력됩니다.

CURRVAL pseudocolumn은 현재 유저가 방금 생성한 시퀀스 번호를 참조하는 데 사용됩니다. 그러나 먼저 NEXTVAL을 사용하여 현재 유저의 세션에서 시퀀스 번호를 생성해야 CURRVAL을 참조할 수 있습니다. CURRVAL을 시퀀스 이름으로 한정해야 합니다. *sequence.CURRVAL*을 참조할 경우 유저의 프로세스로 반환된 마지막 값이 표시됩니다.

NEXTVAL 및 CURRVAL Pseudocolumn(계속)

NEXTVAL 및 CURRVAL 사용 규칙

다음과 같은 상황에서 NEXTVAL 및 CURRVAL을 사용할 수 있습니다.

- subquery의 일부가 아닌 SELECT 문의 SELECT 리스트
- INSERT 문에서 subquery의 SELECT 리스트
- INSERT 문의 VALUES 절
- UPDATE 문의 SET 절

다음과 같은 상황에서는 NEXTVAL 및 CURRVAL을 사용할 수 없습니다.

- 뷰의 SELECT 리스트
- DISTINCT 키워드가 있는 SELECT 문
- GROUP BY, HAVING 또는 ORDER BY 절이 있는 SELECT 문
- SELECT, DELETE 또는 UPDATE 문의 subquery
- CREATE TABLE 또는 ALTER TABLE 문의 DEFAULT 식

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "Pseudocolumns" 및 "CREATE SEQUENCE"에 대한 섹션을 참조하십시오.

시퀀스 사용

- 위치 ID 2500에 "Support"라는 새 부서를 삽입합니다.

```
INSERT INTO departments(department_id,
                       department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
              'Support', 2500);
```

1 rows inserted

- DEPT_DEPTID_SEQ 시퀀스의 현재 값을 확인합니다.

```
SELECT dept_deptid_seq.CURRVAL
FROM    dual;
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

시퀀스 사용

슬라이드의 예제는 DEPARTMENTS 테이블에 새 부서를 삽입합니다. 다음과 같이 DEPT_DEPTID_SEQ 시퀀스를 사용하여 새 부서 번호를 생성합니다.

두번째 슬라이드 예제에 표시된 대로 *sequence_name.CURRVAL*을 사용하여 시퀀스의 현재 값을 확인할 수 있습니다. 이 query의 출력은 다음과 같습니다.

AZ	CURRVAL
1	120

이제 사원을 채용하여 새 부서에 배치하려 한다고 가정합시다. 모든 신입 사원에 대해 실행되는 INSERT 문에 다음 코드를 포함할 수 있습니다.

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq.CURRVAL, ...);
```

주: 위의 예제에서는 새 사원 번호를 생성하기 위해 이미 EMPLOYEE_SEQ라는 시퀀스가 생성되었다고 가정합니다.

시퀀스 값 캐시

- 시퀀스 값을 메모리에 캐시하면 해당 값에 빠르게 액세스할 수 있습니다.
- 다음과 같은 경우 시퀀스 값에 간격이 발생할 수 있습니다.
 - 롤백이 발생하는 경우
 - 시스템에 장애가 발생하는 경우
 - 시퀀스가 다른 테이블에서 사용되는 경우

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

시퀀스 값 캐시

시퀀스를 메모리에 캐시하면 해당 시퀀스 값에 빠르게 액세스할 수 있습니다. 캐시는 처음 시퀀스를 참조할 때 채워집니다. 다음 시퀀스 값을 요청할 때마다 캐시된 시퀀스에서 검색합니다. 마지막 시퀀스 값이 사용된 후 다음 시퀀스 요청 시 다른 시퀀스 캐시를 메모리로 가져옵니다.

시퀀스 간격

시퀀스 생성기는 간격 없이 일련 번호를 생성하지만 이 작업은 커밋이나 롤백에 관계없을 때만 그렇습니다. 따라서 시퀀스가 포함된 명령문을 롤백하면 번호를 잃게 됩니다.

시퀀스에 간격을 발생시킬 수 있는 또 다른 이벤트는 시스템에 장애가 발생하는 경우입니다. 시퀀스가 메모리에 값을 캐시한 경우 시스템이 중단되면 해당 값을 잃게 됩니다.

시퀀스는 테이블과 직접 연관되지 않기 때문에 여러 테이블에 대해 동일한 시퀀스를 사용할 수 있습니다. 그러나 이렇게 하면 각 테이블의 일련 번호에 간격이 포함될 수 있습니다.

시퀀스 수정

증분값, 최대값, 최소값, 순환 옵션 또는 캐시 옵션을 변경합니다.

```
ALTER SEQUENCE dept_deptid_seq
    INCREMENT BY 20
    MAXVALUE 999999
    NOCACHE
    NOCYCLE;
```

```
ALTER SEQUENCE dept_deptid_seq succeeded.
```

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

시퀀스 수정

시퀀스의 MAXVALUE 한계에 도달하면 시퀀스에서 추가 값이 할당되지 않고 시퀀스가 MAXVALUE 한계를 초과했음을 나타내는 오류 메시지가 수신됩니다. ALTER SEQUENCE 문을 사용하여 시퀀스를 수정하면 계속 사용할 수 있습니다.

구문

```
ALTER SEQUENCE      sequence
    [INCREMENT BY n]
    [{MAXVALUE n | NOMAXVALUE}]
    [{MINVALUE n | NOMINVALUE}]
    [{CYCLE | NOCYCLE}]
    [{CACHE n | NOCACHE}];
```

이 구문에서 *sequence*는 시퀀스 생성기의 이름입니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "ALTER SEQUENCE" 관련 섹션을 참조하십시오.

시퀀스 수정 지침

- 시퀀스의 소유자이거나 시퀀스에 대해 **ALTER** 권한을 가져야 합니다.
- 다음 시퀀스 번호에만 적용됩니다.
- 다른 번호로 시퀀스를 다시 시작하려면 시퀀스를 삭제하고 다시 생성해야 합니다.
- 일부 유효성 검사가 수행됩니다.
- 시퀀스를 제거하려면 **DROP** 문을 사용합니다.

```
DROP SEQUENCE dept_deptid_seq;
DROP SEQUENCE dept_deptid_seq succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

시퀀스 수정 지침

- 시퀀스를 수정하려면 시퀀스의 소유자이거나 시퀀스에 대해 ALTER 권한을 가져야 합니다.
시퀀스를 제거하려면 시퀀스의 소유자이거나 DROP ANY SEQUENCE 권한을 가져야 합니다.
- ALTER SEQUENCE 문은 후속 시퀀스 번호에만 적용됩니다.
- START WITH 옵션은 ALTER SEQUENCE를 사용하여 변경할 수 없습니다. 다른 번호로 시퀀스를 다시 시작하려면 시퀀스를 삭제하고 다시 생성해야 합니다.
- 일부 유효성 검사가 수행됩니다. 예를 들어, 새 MAXVALUE를 현재 시퀀스 번호보다 작게 지정 할 수 없습니다.

```
ALTER SEQUENCE dept_deptid_seq
    INCREMENT BY 20
    MAXVALUE 90
    NOCACHE
    NOCYCLE;
```

- 오류:

```
Error report:
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value
04009. 00000 -  "MAXVALUE cannot be made to be less than the current value"
*Cause:    the current value exceeds the given MAXVALUE
*Action:   make sure that the new MAXVALUE is larger than the current value
```

단원 내용

- 뷰 개요:
 - 뷰에서 데이터 생성, 수정 및 검색
 - 뷰에 대한 **DML** 작업
 - 뷰 삭제
- 시퀀스 개요:
 - 시퀀스 생성, 사용 및 수정
 - 시퀀스 값을 캐시에 저장
 - **NEXTVAL** 및 **CURRVAL pseudocolumn**
- 인덱스 개요
 - 인덱스 생성, 삭제
- 동의어 개요
 - 동의어 생성, 삭제

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

인덱스

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
인덱스	일부 query 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

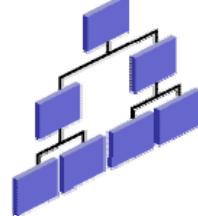
인덱스

데이터베이스 객체인 인덱스를 생성하면 일부 query의 성능을 향상시킬 수 있습니다. 인덱스는 유저가 Primary key 또는 Unique 제약 조건을 생성할 때 서버에 의해 자동으로 생성됩니다.

인덱스

인덱스:

- 스키마 객체입니다.
- **Oracle** 서버에서 포인터를 사용하여 행 검색 속도를 높이는 데 사용할 수 있습니다.
- 신속한 경로 액세스 방식을 사용하여 데이터를 빠르게 찾아 디스크 I/O(입/출력)를 줄일 수 있습니다.
- 인덱스의 대상인 테이블에 독립적입니다.
- **Oracle** 서버에서 자동으로 사용되고 유지 관리됩니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

인덱스(계속)

Oracle 서버 인덱스는 포인터를 사용하여 행 검색 속도를 높일 수 있는 스키마 객체입니다. 명시적으로 또는 자동으로 인덱스를 생성할 수 있습니다. 열에 인덱스가 없으면 전체 테이블 스캔이 수행됩니다.

인덱스를 사용하면 테이블의 행에 직접 빠르게 액세스할 수 있습니다. 인덱스는 인덱스화된 경로를 사용하여 데이터를 신속하게 찾음으로써 디스크 I/O를 줄이는 데 그 목적이 있습니다. 인덱스는 Oracle 서버에서 자동으로 사용되고 유지 관리됩니다. 인덱스가 생성된 후에는 유저가 직접 조작할 필요가 없습니다.

인덱스는 논리적, 물리적으로 인덱스의 대상인 테이블에 독립적입니다. 즉, 인덱스는 언제든지 생성하고 삭제할 수 있으며 기본 테이블이나 다른 인덱스에 영향을 미치지 않습니다.

주: 테이블을 삭제하면 해당 인덱스 또한 삭제됩니다.

자세한 내용은 *Oracle Database Concepts 11g, Release 1 (11.1)*에서 "Schema Objects: Indexes" 관련 섹션을 참조하십시오.

인덱스가 생성되는 방식

- 자동으로: 테이블 정의에서 **PRIMARY KEY** 또는 **UNIQUE** 제약 조건을 정의하면 고유 인덱스가 자동으로 생성됩니다.



- 수동으로: 행에 액세스하는 속도를 높이기 위해 유저가 열의 비고유 인덱스를 생성할 수 있습니다.



ORACLE

Copyright © 2007, Oracle. All rights reserved.

인덱스가 생성되는 방식

두 가지 유형의 인덱스를 생성할 수 있습니다.

고유 인덱스: 테이블의 열이 **PRIMARY KEY** 또는 **UNIQUE** 제약 조건을 갖도록 정의하면 Oracle 서버가 자동으로 고유 인덱스를 생성합니다. 인덱스의 이름은 제약 조건에 지정된 이름입니다.

비고유 인덱스: 유저가 생성할 수 있는 인덱스입니다. 예를 들어, query의 조인을 위해 **FOREIGN KEY** 열 인덱스를 생성하여 검색 속도를 높일 수 있습니다.

주: 고유 인덱스를 수동으로 생성할 수도 있지만, 고유 인덱스를 암시적으로 생성하는 Unique 제약 조건을 생성하는 것이 좋습니다.

인덱스 생성

- 하나 이상의 열에 인덱스를 생성합니다.

```
CREATE [UNIQUE] [BITMAP] INDEX index
ON table (column[, column]...);
```

- EMPLOYEES** 테이블의 **LAST_NAME** 열에 대한 **query** 액세스 속도를 향상시킵니다.

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
```

```
CREATE INDEX succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

인덱스 생성

CREATE INDEX 문을 실행하여 하나 이상의 열에 인덱스를 생성합니다.

구문 설명:

- index 인덱스의 이름입니다.
- table 테이블의 이름입니다.
- column 인덱스화될 테이블의 열 이름입니다.

인덱스가 기반으로 하는 열의 값이 고유해야 함을 나타내려면 UNIQUE를 지정합니다. 각 행을 별도로 인덱스화하지 않고 각 구분 키에 대한 비트맵을 사용하여 인덱스가 생성되도록 하려면 BITMAP을 지정합니다. 비트맵 인덱스에서는 키 값과 연관된 rowid를 비트맵으로 저장합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "CREATE INDEX" 관련 섹션을 참조하십시오.

인덱스 생성 지침

다음 경우에 인덱스를 생성하십시오.

	열에 광범위한 값이 포함된 경우
	열에 많은 널 값이 포함된 경우
	하나 이상의 열이 WHERE 절이나 조인 조건에서 함께 자주 사용되는 경우
	테이블이 크고 대부분의 query 가 테이블에서 2%~4% 미만의 행을 검색할 것으로 예상되는 경우

다음 경우에는 인덱스를 생성하지 마십시오.

	열이 query 에서 조건으로 자주 사용되지 않는 경우
	테이블이 작거나 대부분의 query 가 테이블에서 2%~4% 이상의 행을 검색할 것으로 예상되는 경우
	테이블이 자주 생성되는 경우
	인덱스화된 열이 표현식의 일부로 참조되는 경우

ORACLE

Copyright © 2007, Oracle. All rights reserved.

인덱스 생성 지침

많다고 좋은 것은 아님

테이블에 인덱스가 많다고 해서 query 속도가 빨라지는 것은 아닙니다. 각 DML 작업이 인덱스가 있는 테이블에 커밋되어 있다는 것은 인덱스가 생성되어야 함을 의미합니다. 테이블과 연관된 인덱스가 많을수록 DML 작업 후에 모든 인덱스를 생성해야 하므로 Oracle 서버의 부담이 늘어납니다.

인덱스를 생성해야 하는 경우

따라서 다음과 같은 경우에만 인덱스를 생성해야 합니다.

- 열에 광범위한 값이 포함된 경우
- 열에 많은 널 값이 포함된 경우
- 하나 이상의 열이 WHERE 절이나 조인 조건에서 함께 자주 사용되는 경우
- 테이블이 크고 대부분의 query가 2% ~ 4% 미만의 행을 검색할 것으로 예상되는 경우

고유성을 강화하려면 테이블 정의에서 Unique 제약 조건을 정의해야 합니다. 그러면 고유 인덱스가 자동으로 생성됩니다.

인덱스 제거

- **DROP INDEX** 명령을 사용하여 데이터 딕셔너리에서 인덱스를 제거합니다.

```
DROP INDEX index;
```

- 데이터 딕셔너리에서 **emp_last_name_idx** 인덱스를 제거합니다.

```
DROP INDEX emp_last_name_idx;
```

```
DROP INDEX emp_last_name_idx succeeded.
```

- 인덱스를 삭제하려면 인덱스의 소유자이거나 **DROP ANY INDEX** 권한이 있어야 합니다.

ORACLE

Copyright © 2007, Oracle. All rights reserved.

인덱스 제거

인덱스는 수정할 수 없습니다. 인덱스를 변경하려면 인덱스를 삭제한 다음 다시 생성해야 합니다.

DROP INDEX 문을 실행하여 데이터 딕셔너리에서 인덱스 정의를 제거합니다. 인덱스를 삭제하려면 인덱스의 소유자이거나 **DROP ANY INDEX** 권한이 있어야 합니다.

이 구문에서 *index*는 인덱스의 이름입니다.

주: 테이블을 삭제하면 인덱스와 제약 조건이 자동으로 삭제되지만 뷰와 시퀀스는 남아 있습니다.

단원 내용

- 뷰 개요:
 - 뷰에서 데이터 생성, 수정 및 검색
 - 뷰에 대한 **DML** 작업
 - 뷰 삭제
- 시퀀스 개요:
 - 시퀀스 생성, 사용 및 수정
 - 시퀀스 값을 캐시에 저장
 - **NEXTVAL** 및 **CURRVAL pseudocolumn**
- 인덱스 개요
 - 인덱스 생성, 삭제
- 동의어 개요
 - 동의어 생성, 삭제

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

동의어

객체	설명
테이블	기본 저장 단위이며 행으로 구성되어 있습니다.
뷰	하나 이상의 테이블에 있는 데이터의 부분 집합을 논리적으로 나타냅니다.
시퀀스	숫자 값을 생성합니다.
인덱스	일부 query 성능을 향상시킵니다.
동의어	객체에 다른 이름을 부여합니다.

ORACLE®

Copyright © 2007, Oracle. All rights reserved.

동의어

동의어는 다른 이름으로 테이블을 호출할 수 있는 데이터베이스 객체입니다. 동의어를 생성하여 테이블에 대체 이름을 부여할 수 있습니다.

객체의 동의어 생성

동의어(객체의 또 다른 이름)를 생성하면 객체에 쉽게 액세스할 수 있습니다. 동의어를 사용하여 다음과 같은 작업을 할 수 있습니다.

- 다른 유저가 소유한 테이블을 쉽게 참조할 수 있습니다.
- 긴 객체 이름을 짧게 만듭니다.

```
CREATE [PUBLIC] SYNONYM synonym
FOR      object;
```

Copyright © 2007, Oracle. All rights reserved.

객체의 동의어 생성

다른 유저가 소유한 테이블을 참조하려면 테이블 이름 앞에 해당 테이블을 만든 유저 이름을 추가하고 마침표를 입력해야 합니다. 동의어를 생성하면 객체 이름을 스키마로 한정할 필요가 없고 테이블, 뷰, 시퀀스, 프로시저 또는 다른 객체에 대해 다른 이름을 지정할 수 있습니다. 이 방법은 뷰와 같이 객체 이름이 긴 경우에 특히 유용합니다.

구문 설명:

PUBLIC	모든 유저가 액세스할 수 있는 동의어를 생성합니다.
<i>synonym</i>	생성할 동의어의 이름입니다.
<i>object</i>	동의어를 생성할 객체를 식별합니다.

지침

- 객체는 패키지에 포함될 수 없습니다.
- 전용(private) 동의어 이름은 동일한 유저가 소유한 모든 다른 객체와 구분되어야 합니다.

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "CREATE SYNONYM" 관련 섹션을 참조하십시오.

동의어 생성 및 제거

- **DEPT_SUM_VU** 뷰의 짧은 이름을 생성합니다.

```
CREATE SYNONYM d_sum
FOR dept_sum_vu;
CREATE SYNONYM succeeded.
```

- 동의어를 삭제합니다.

```
DROP SYNONYM d_sum;
DROP SYNONYM d_sum succeeded.
```

ORACLE

Copyright © 2007, Oracle. All rights reserved.

동의어 생성 및 제거

동의어 생성

슬라이드의 예제는 빠른 참조를 위해 DEPT_SUM_VU 뷰의 동의어를 생성합니다.

데이터베이스 관리자는 모든 유저가 액세스할 수 있는 공용(public) 동의어를 생성할 수 있습니다.
다음 예제는 Alice의 DEPARTMENTS 테이블에 대해 DEPT라는 공용 동의어를 생성합니다.

```
CREATE PUBLIC SYNONYM dept
CREATE SYNONYM succeeded.
```

동의어 제거

동의어를 제거하려면 DROP SYNONYM 문을 사용합니다. 데이터베이스 관리자만 공용(public)
동의어를 삭제할 수 있습니다.

```
DROP PUBLIC SYNONYM dept;
```

자세한 내용은 *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*에서 "DROP SYNONYM"
관련 섹션을 참조하십시오.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 뷰 생성, 사용 및 제거
- 시퀀스 생성기를 사용하여 자동으로 시퀀스 번호 생성
- 인덱스를 생성하여 **query** 검색 속도 향상
- 동의어를 사용하여 객체에 대체 이름 제공



Copyright © 2007, Oracle. All rights reserved.

요약

이 단원에서는 뷰, 시퀀스, 인덱스 및 동의어와 같은 데이터베이스 객체에 대해 배웠습니다.

연습 11: 2부 개요

이 연습에서는 다음 내용을 다룹니다.

- 시퀀스 생성
- 시퀀스 사용
- 비고유 인덱스 생성
- 동의어 생성



Copyright © 2007, Oracle. All rights reserved.

연습 11: 2부 개요

이 단원의 2부 연습에서는 시퀀스, 인덱스 및 동의어를 생성하고 사용하는 다양한 연습을 제공합니다.

이 단원 끝부분에서 문제 7-10을 완성하십시오.

연습 11

1부

1. HR 부서의 임원이 EMPLOYEES 테이블의 일부 데이터를 숨길 것을 요구합니다.
이 임원은 EMPLOYEES 테이블의 사원 번호, 사원 이름 및 부서 번호를 기반으로 하는 EMPLOYEES_VU라는 뷰를 원합니다. 또한 사원 이름의 머리글을 EMPLOYEE로 표시하도록 요구합니다.
2. 뷰가 작동하는지 확인합니다. EMPLOYEES_VU 뷰의 내용을 표시합니다.

EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	100 King	90
2	101 Kochhar	90
3	102 De Haan	90
4	103 Hunold	60
5	104 Ernst	60
...		
19	205 Higgins	110
20	206 Gietz	110

3. EMPLOYEES_VU 뷰를 사용하여 HR 부서를 위해 모든 사원 이름과 부서 번호를 표시하는 query를 작성합니다.

EMPLOYEE	DEPARTMENT_ID
1 King	90
2 Kochhar	90
3 De Haan	90
4 Hunold	60
5 Ernst	60
...	
19 Higgins	110
20 Gietz	110

연습 11(계속)

4. 부서 50의 사원 데이터에 액세스하려고 합니다. 부서 50의 모든 사원에 대해 사원 번호, 사원 성 및 부서 번호를 포함하는 DEPT50이라는 뷰를 생성합니다. 뷰 열의 레이블을 EMPNO, EMPLOYEE 및 DEPTNO로 지정하라는 요청을 받았습니다. 보안상의 이유로 사원이 뷰를 통해 다른 부서에 재할당되는 것을 허용하지 않습니다.
5. DEPT50 뷰의 구조 및 내용을 표시합니다.

Name	Null	Type
EMPNO	NOT NULL	NUMBER(6)
EMPLOYEE	NOT NULL	VARCHAR2(25)
DEPTNO		NUMBER(4)

	EMPNO	EMPLOYEE	DEPTNO
1	124	Mourgos	50
2	141	Rajs	50
3	142	Davies	50
4	143	Matos	50
5	144	Vargas	50

6. 뷰를 테스트합니다. Matos를 부서 80에 재할당해 보십시오.

연습 11(계속)

2부

7. DEPT 테이블의 PRIMARY KEY 열에 사용할 수 있는 시퀀스가 필요합니다. 시퀀스는 200부터 시작하고 최대값은 1,000이어야 합니다. 시퀀스 증분값을 10으로 설정하고 시퀀스 이름을 DEPT_ID_SEQ로 지정합니다.
8. 시퀀스를 테스트하기 위해 DEPT 테이블에 두 행을 삽입하는 스크립트를 작성합니다. 스크립트 이름을 lab_11_08.sql로 지정합니다. ID 열에 대해 생성한 시퀀스를 사용해야 합니다. Education 및 Administration이라는 두 개의 부서를 추가합니다. 추가한 내용을 확인합니다. 스크립트에서 해당 명령을 실행합니다.
9. DEPT 테이블의 NAME 열에 비고유 인덱스를 생성합니다.
10. EMPLOYEES 테이블의 동의어를 생성합니다. EMP라는 이름을 지정합니다.

A
연습 해답

연습 I 해답: 소개

아래에서 "소개" 단원 연습의 해답에 대해 설명합니다.

Oracle SQL Developer 데모 실행: 데이터베이스 연결 생성

1. 다음 링크에 있는 데모: "데이터베이스 연결 생성"에 액세스합니다.

http://st-curriculum.oracle.com/tutorial/SQLDeveloper/html/module2/mod02_cp_newdbconn.htm

Oracle SQL Developer 시작

2. "sqldeveloper" 바탕 화면 아이콘을 사용하여 Oracle SQL Developer를 시작합니다.

1. "sqldeveloper" 바탕 화면 아이콘을 두 번 누릅니다.

주: SQL Developer를 처음 실행하는 경우 java.exe 파일의 경로를 제공해야 합니다. 이 작업은 강의실 설정의 일부로 이미 실행되었습니다. 프롬프트가 나타나는 경우 다음 경로를 입력합니다.

D:\app\Administrator\product\11.1.0\client_1\jdevstudio\jdk\bin

새 SQL Developer 데이터베이스 연결 생성

3. 새 데이터베이스 연결을 생성하려면 Connections Navigator에서 Connections를 마우스 오른쪽 버튼으로 누릅니다. 메뉴에서 New Connection을 선택합니다. New>Select Database Connection 대화상자가 나타납니다.

4. 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

a. Connection Name: myconnection

b. Username: oraxx. 여기에서 xx는 유저의 PC 번호(강사에게 문의하여 계정 범위 ora1- ora20 내의 한 계정을 할당받도록 합니다.)

c. Password: oraxx

d. Hostname: 데이터베이스 서버가 실행되고 있는 컴퓨터의 호스트 이름을 입력합니다.

e. Port: 1521

f. SID: ORCL

g. Save Password 체크 박스를 선택합니다.

연습 1 해답: 소개(계속)

SQL Developer 데이터베이스 연결을 사용하여 테스트 및 연결

5. 새 연결을 테스트합니다.

1. New>Select Database Connection window의 Test 버튼을 누릅니다.

6. 상태가 Success이면 새로운 이 연결을 사용하여 데이터베이스에 연결합니다.

1. 상태가 Success이면 Connect 버튼을 누릅니다.

Connections Navigator에서 테이블 탐색

7. Connections Navigator의 Tables 노드 아래에서 사용할 수 있는 객체를 확인합니다.
다음 테이블이 있는지 확인합니다.

COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS

LOCATIONS
REGIONS

1. 옆에 있는 더하기(+) 기호를 눌러 myconnection 연결을 확장합니다.

2. 옆에 있는 더하기(+) 기호를 눌러 Tables 아이콘을 확장합니다.

8. EMPLOYEES 테이블의 구조를 살펴봅니다.

1. EMPLOYEES 테이블의 구조를 표시하기 위해 EMPLOYEES를 누르면
기본적으로 Columns 탭이 활성화되어 테이블의 구조가 표시됩니다.

9. DEPARTMENTS 테이블의 데이터를 확인합니다.

1. DEPARTMENTS 테이블을 누릅니다.

2. Data 탭을 누릅니다. 테이블 데이터가 표시됩니다.

연습 | 해답: 소개(계속)

SQL Worksheet 열기

10. 새 SQL Worksheet를 엽니다. 해당 SQL Worksheet에 대해 사용할 수 있는 단축 아이콘을 조사합니다.

1. 새 SQL Worksheet를 열려면 Tools 메뉴에서 SQL Worksheet를 선택합니다.
2. 또는 myconnection을 마우스 오른쪽 버튼으로 누르고 Open SQL Worksheet를 선택합니다.
3. SQL Worksheet의 단축 아이콘을 확인합니다. 특히 Execute Statement 및 Run Script 아이콘을 찾아봅니다.

연습 1 해답: SQL SELECT 문을 사용하여 데이터 검색

1부

지식을 테스트해 보십시오.

- 다음 SELECT 문이 성공적으로 실행됩니다.

```
SELECT last_name, job_id, salary AS Sal
FROM   employees;
```

참/거짓

- 다음 SELECT 문이 성공적으로 실행됩니다.

```
SELECT *
FROM   job_grades;
```

참/거짓

- 이 명령문에 네 개의 코딩 오류가 있습니다. 식별할 수 있습니까?

```
SELECT      employee_id, last_name
sal x 12  ANNUAL SALARY
FROM       employees;
```

- EMPLOYEES** 테이블에 **sal**이라는 열이 없습니다. 열 이름은 **SALARY**입니다.
- 두 번째 행의 곱하기 연산자는 **x**가 아니라 *****입니다.
- ANNUAL SALARY alias**는 공백을 포함할 수 없습니다. alias는 **ANNUAL_SALARY**로 표기하거나 큰 따옴표로 묶어야 합니다.
- LAST_NAME** 열 뒤에 쉼표가 누락되었습니다.

2부

연습을 시작하기 전에 다음과 같은 점에 주의합니다.

- 모든 실습 파일을 *D:\labs\SQL1\labs*에 저장합니다.
- SQL 문을 SQL Worksheet에 입력합니다. SQL Developer에서 스크립트를 저장하려면 File 메뉴에서 Save As를 선택하거나 SQL Worksheet를 마우스 오른쪽 버튼으로 누른 다음 Save File을 선택하여 SQL 문을 *lab_<lessonno>_<stepno>.sql* 스크립트로 저장합니다. 기존 스크립트를 수정하려면 File > Open을 사용하여 스크립트 파일을 열고 변경한 다음 Save As를 사용하여 다른 파일 이름으로 저장합니다.
- query를 실행하려면 SQL Worksheet에서 Execute Statement 아이콘을 누르거나 [F9]를 누릅니다. DML 및 DDL 문의 경우에는 Run Script 아이콘을 누르거나 [F5]를 누릅니다.
- 저장된 스크립트를 실행한 후 다음 query를 동일한 워크시트에 입력하지 않도록 합니다. 새 워크시트를 엽니다.

연습 1 해답: SQL SELECT 문을 사용하여 데이터 검색(계속)

여러분은 Acme Corporation의 SQL 프로그래머로 채용되었습니다. 첫 작업은 Human Resources 테이블의 데이터를 기반으로 몇 가지 보고서를 작성하는 것입니다.

4. 첫 작업은 DEPARTMENTS 테이블의 구조와 해당 내용을 파악하는 것입니다.

a. DEPARTMENTS 테이블의 구조를 파악하려면 다음 명령문을 사용합니다.

```
DESCRIBE departments
```

b. DEPARTMENTS 테이블에 포함된 데이터를 확인하려면 다음 명령문을 사용합니다.

```
SELECT *
FROM   departments;
```

5. EMPLOYEES 테이블의 구조를 파악해야 합니다.

```
DESCRIBE employees
```

HR 부서에서 사원 번호부터 시작해서 각 사원에 대한 성, 직무 코드, 채용 날짜 및 사원 번호를 표시하는 query를 요구합니다. HIRE_DATE 열에 alias STARTDATE를 부여합니다. 작성한 SQL 문을 HR 부서에 전달할 수 있도록 lab_01_05.sql이라는 파일에 저장합니다.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM   employees;
```

6. lab_01_05.sql 파일의 query가 제대로 실행되는지 테스트합니다.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM   employees;
```

7. HR 부서에서 EMPLOYEES 테이블의 모든 고유 직무 코드를 표시하는 query를 요구합니다.

```
SELECT DISTINCT job_id
FROM   employees;
```

연습 1 해답: SQL SELECT 문을 사용하여 데이터 검색(계속)

3부

시간 여유가 있을 경우 다음 연습을 완료하십시오.

8. HR 부서에서 보고에 적합하도록 열 머리글의 사원 정보를 쉽게 표시해 달라고 합니다. lab_01_05.sql의 명령문을 새 SQL Worksheet로 복사합니다. 열 머리글에 Emp #, Employee, Job 및 Hire Date라는 이름을 각각 지정합니다. 그런 다음 query를 다시 실행합니다.

```
SELECT employee_id "Emp #", last_name "Employee",
       job_id "Job", hire_date "Hire Date"
FROM   employees;
```

9. HR 부서에서 모든 사원과 그들의 직무 ID에 대한 보고서를 요청했습니다. 성과 직무 ID를 이어서 표시하고(쉼표와 공백으로 구분) 열 이름을 Employee and Title로 지정합니다.

```
SELECT last_name||', '||job_id "Employee and Title"
FROM   employees;
```

심화 연습에 도전하려면 다음 연습을 완료하십시오.

10. EMPLOYEES 테이블의 데이터에 익숙해지도록 해당 테이블의 모든 데이터를 표시하는 query를 작성합니다. 각 열 출력은 쉼표로 구분합니다. 열 이름을 THE_OUTPUT로 지정합니다.

```
SELECT employee_id || ',' || first_name || ',' || last_name
      || ',' || email || ',' || phone_number || ',' || job_id
      || ',' || manager_id || ',' || hire_date || ',' ||
      || salary || ',' || commission_pct || ',' || department_id
THE_OUTPUT
FROM   employees;
```

연습 2 해답: 데이터 제한 및 정렬

HR 부서에서 몇 가지 query 작성과 관련해 여러분의 도움을 요청합니다.

1. HR 부서에서 예산 문제 때문에 급여가 \$12,000가 넘는 사원의 성과 급여를 표시하는 보고서가 필요합니다. 작성한 SQL 문을 `lab_02_01.sql`이라는 파일로 저장합니다. query를 실행합니다.

```
SELECT last_name, salary
FROM employees
WHERE salary > 12000;
```

2. 새 SQL Worksheet를 엽니다. 사원 번호 176의 성과 부서 번호를 표시하는 보고서를 작성합니다.

```
SELECT last_name, department_id
FROM employees
WHERE employee_id = 176;
```

3. HR 부서에서 급여가 높은 사원과 급여가 낮은 사원을 찾으려고 합니다. 급여가 \$5,000에서 \$12,000의 범위에 속하지 않는 모든 사원의 성 및 급여를 표시하도록 `lab_02_01.sql`을 수정합니다. 작성한 SQL 문을 `lab_02_03.sql`로 저장합니다.

```
SELECT last_name, salary
FROM employees
WHERE salary NOT BETWEEN 5000 AND 12000;
```

4. Matos 및 Taylor라는 성을 가진 사원에 대해 성, 직무 ID, 시작 날짜를 표시하는 보고서를 작성합니다. 시작 날짜를 기준으로 오름차순으로 query를 정렬합니다.

```
SELECT last_name, job_id, hire_date
FROM employees
WHERE last_name IN ('Matos', 'Taylor')
ORDER BY hire_date;
```

5. 부서 20 또는 50에 속하는 모든 사원의 성과 부서 번호를 이름을 기준으로 오름차순으로 정렬하여 표시합니다.

```
SELECT last_name, department_id
FROM employees
WHERE department_id IN (20, 50)
ORDER BY last_name ASC;
```

연습 2 해답: 데이터 제한 및 정렬(계속)

6. \$5,000 ~ \$12,000의 급여를 받고 부서 20 또는 50에 속하는 사원의 성과 급여를 나열하도록 lab_02_03.sql을 수정합니다. 열 레이블을 각각 Employee 및 Monthly Salary로 지정합니다. lab_02_03.sql을 lab_02_06.sql로 다시 저장합니다. lab_02_06.sql의 명령문을 실행합니다.

```
SELECT last_name "Employee", salary "Monthly Salary"
FROM employees
WHERE salary BETWEEN 5000 AND 12000
AND department_id IN (20, 50);
```

7. HR 부서에서 1994년에 채용된 모든 사원의 성과 채용 날짜를 표시하는 보고서를 요구합니다.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%94';
```

8. 담당 관리자가 없는 모든 사원의 성과 직책을 표시하는 보고서를 작성합니다.

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

9. 커미션을 받는 모든 사원의 성, 급여 및 커미션을 표시하는 보고서를 작성합니다. 급여 및 커미션의 내림차순으로 데이터를 정렬합니다. ORDER BY 절에서 열의 숫자 위치를 사용합니다.

```
SELECT last_name, salary, commission_pct
FROM employees
WHERE commission_pct IS NOT NULL
ORDER BY 2 DESC, 3 DESC;
```

10. HR 부서의 멤버는 여러분이 작성 중인 query에 유연성이 확대되기를 원합니다. 그들은 유저가 프롬프트에 지정하는 액수보다 많은 급여를 받은 사원이 있을 경우 이들의 성과 급여를 표시하는 보고서를 기대합니다. (연습 1에서 생성한 query를 수정하여 사용할 수 있습니다.) 이 query를 lab_02_10.sql이라는 파일에 저장합니다.

- a. 프롬프트가 나타나면 대화상자에 값으로 12000을 입력합니다. OK를 누릅니다.

```
SELECT last_name, salary
FROM employees
WHERE salary > &sal_amt;
```

연습 2 해답: 데이터 제한 및 정렬(계속)

11. HR 부서에서 관리자를 기준으로 보고서를 실행하려고 합니다. 유저에게 관리자 ID 입력 프롬프트를 표시하고 해당 관리자에 속한 사원의 사원 ID, 성, 급여 및 부서를 생성하는 query를 작성합니다. HR 부서에서 선택한 열을 기준으로 보고서를 정렬하는 기능을 원합니다. 다음 값으로 데이터를 테스트할 수 있습니다.

manager_id = 103, last_name을 기준으로 정렬

manager_id = 201, salary를 기준으로 정렬

manager_id = 124, employee_id를 기준으로 정렬

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE manager_id = &mgr_num
ORDER BY &order_col;
```

시간 여유가 있을 경우 다음 연습을 완료하십시오.

12. 성의 세번째 문자가 "a"인 모든 사원의 성을 표시합니다.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '__a%';
```

13. 성에 "a"와 "e"가 모두 포함된 모든 사원의 성을 표시합니다.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '%a%' AND last_name LIKE '%e%';
```

심화 연습에 도전하려면 다음 연습을 완료하십시오.

14. 직무가 판매 사원이나 자재 담당자이고 급여가 \$2,500, \$3,500 또는 \$7,000가 아닌 모든 사원의 성, 직무 및 급여를 표시합니다.

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id IN ('SA_REP', 'ST_CLERK')
AND salary NOT IN (2500, 3500, 7000);
```

15. 커미션이 20%인 모든 사원의 성, 급여 및 커미션을 표시하도록

lab_02_06.sql을 수정합니다. lab_02_06.sql을 lab_02_15.sql로 다시 저장합니다. lab_02_15.sql의 명령문을 다시 실행합니다.

```
SELECT last_name "Employee", salary "Monthly Salary",
commission_pct
FROM employees
WHERE commission_pct = .20;
```

연습 3 해답: 단일 행 함수를 사용하여 출력 커스터마이즈

- 시스템 날짜를 표시하기 위한 query를 작성합니다. 열 레이블을 Date로 지정합니다.

주: 시간대가 다른 지역에 데이터베이스가 있는 경우 해당 데이터베이스가 상주하는 운영 체제의 날짜가 출력됩니다.

```
SELECT sysdate "Date"
FROM dual;
```

- HR 부서에서 각 사원에 대해 사원 번호, 성, 급여 및 15.5% 인상된 급여(정수로 표현)를 표시하는 보고서가 필요합니다. 열 레이블을 New Salary로 지정합니다. SQL 문을 lab_03_02.sql이라는 파일에 저장합니다.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary"
FROM employees;
```

- lab_03_02.sql 파일에 있는 query를 실행합니다.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary"
FROM employees;
```

- 새 급여에서 이전 급여를 뺀 열을 추가하도록 lab_03_02.sql의 query를 수정합니다. 열 레이블을 Increase로 지정합니다. 파일의 내용을 lab_03_04.sql로 저장합니다. 수정한 query를 실행합니다.

```
SELECT employee_id, last_name, salary,
       ROUND(salary * 1.155, 0) "New Salary",
       ROUND(salary * 1.155, 0) - salary "Increase"
FROM employees;
```

- "J", "A" 또는 "M"으로 시작하는 이름을 가진 모든 사원의 성(첫번째 문자는 대문자, 나머지는 모두 소문자)과 성의 길이를 표시하는 query를 작성합니다. 각 열에 적절한 레이블을 지정합니다. 사원의 성을 기준으로 결과를 정렬합니다.

```
SELECT INITCAP(last_name) "Name",
       LENGTH(last_name) "Length"
FROM employees
WHERE last_name LIKE 'J%'
OR last_name LIKE 'M%'
OR last_name LIKE 'A%'
ORDER BY last_name ;
```

연습 3 해답: 단일 행 함수를 사용하여 출력 커스터마이즈(계속)

유저에게 성의 첫 문자를 입력하는 프롬프트를 표시하도록 query를 재작성합니다. 예를 들어, 문자 입력 프롬프트가 표시되었을 때 유저가 H(대문자)를 입력하면 출력에 성이 "H"로 시작하는 모든 사원이 표시되어야 합니다.

```
SELECT INITCAP(last_name) "Name",
       LENGTH(last_name) "Length"
  FROM employees
 WHERE last_name LIKE '&start_letter%'
 ORDER BY last_name;
```

입력된 문자의 대소문자 여부에 따라 출력이 달라지지 않도록 query를 수정합니다.
입력된 문자는 SELECT query에서 처리되기 전에 대문자로 변경해야 합니다.

```
SELECT INITCAP(last_name) "Name",
       LENGTH(last_name) "Length"
  FROM employees
 WHERE last_name LIKE UPPER('&start_letter%')
 ORDER BY last_name;
```

6. HR 부서에서 각 사원의 근속 기간을 파악하려고 합니다. 각 사원에 대해 성을 표시하고 채용일부터 오늘까지 경과한 개월 수를 계산합니다. 열 레이블을 MONTHS_WORKED로 지정합니다. 재직 개월 수에 따라 결과를 정렬합니다. 개월 수를 가장 가까운 정수로 반올림합니다.

주: 이 query는 실행된 날짜에 종속되므로 MONTHS_WORKED 열의 값은 실행된 날짜에 따라 달라집니다.

```
SELECT last_name, ROUND(MONTHS_BETWEEN(
          SYSDATE, hire_date)) MONTHS_WORKED
  FROM employees
 ORDER BY months_worked;
```

시간 여유가 있을 경우 다음 연습을 완료하십시오.

7. 모든 사원의 성과 급여를 표시하기 위한 query를 작성합니다. 급여의 형식을 15자 길이로 지정하고 왼쪽을 \$ 기호로 채웁니다. 열 레이블을 SALARY로 지정합니다.

```
SELECT last_name,
       LPAD(salary, 15, '$') SALARY
  FROM employees;
```

연습 3 해답: 단일 행 함수를 사용하여 출력 커스터마이즈(계속)

8. 사원의 성에서 처음 8자를 표시하고 급여 액수를 별표로 나타내는 query를 작성합니다. 각 별표는 \$1,000를 나타냅니다. 급여의 내림차순으로 데이터를 정렬합니다. 열 레이블을 EMPLOYEES_AND THEIR_SALARIES로 지정합니다.

```
SELECT rpad(last_name, 8) || ' ' ||  
      rpad(' ', salary/1000+1, '*')  
          EMPLOYEES_AND_THEIR_SALARIES  
FROM   employees  
ORDER BY salary DESC;
```

9. 부서 90의 모든 사원에 대해 성 및 재직 기간(주 단위)을 표시하도록 query를 작성합니다. 주의 수를 나타내는 열에 TENURE라는 레이블을 지정합니다. 주를 나타내는 숫자 값을 소수점 왼쪽에서 truncate합니다. 직원 재직 기간의 내림차순으로 레코드를 표시합니다.
- a. 주: TENURE 값은 query를 실행한 날짜에 따라 달라집니다.

```
SELECT last_name, trunc((SYSDATE-hire_date)/7) AS TENURE  
FROM   employees  
WHERE  department_id = 90  
ORDER BY TENURE DESC
```

연습 4 해답: 변환 함수 및 조건부 표현식 사용

1. 각 사원에 대해 다음과 같이 출력되는 보고서를 작성합니다.

<employee last name> earns <salary> monthly but wants <3 times salary> 열 레이블을 Dream Salaries로 지정합니다.

```
SELECT  last_name || ' earns '
        || TO_CHAR(salary, 'fm$99,999.00')
        || ' monthly but wants '
        || TO_CHAR(salary * 3, 'fm$99,999.00')
        || '.' "Dream Salaries"
FROM    employees;
```

2. 각 사원의 성, 채용 날짜 및 근무 6개월 후 첫번째 월요일에 해당하는 급여 심의 날짜를 표시합니다. 열 레이블을 REVIEW로 지정합니다. 날짜 형식을 "Monday, the Thirty-First of July, 2000"과 유사한 형식으로 지정합니다.

```
SELECT last_name, hire_date,
       TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'MONDAY'),
               'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
FROM    employees;
```

3. 사원의 성, 채용 날짜, 근무 시작 요일을 표시합니다. 열 레이블을 DAY로 지정합니다. 월요일부터 시작하여 요일순으로 결과를 정렬합니다.

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') DAY
FROM    employees
ORDER BY TO_CHAR(hire_date - 1, 'd');
```

4. 사원의 성과 커미션 금액을 표시하는 query를 작성합니다. 사원이 커미션을 받지 않으면 "No Commission"을 표시합니다. 열 레이블을 COMM으로 지정합니다.

```
SELECT last_name,
       NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM    employees;
```

연습 4 해답: 변환 함수 및 조건부 표현식 사용(계속)

5. 다음 데이터를 사용하여 DECODE 함수를 통해 JOB_ID 열의 값을 기반으로 모든 사원의 등급을 표시하는 query를 작성합니다.

<i>Job</i>	<i>Grade</i>
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA REP	D
ST_CLERK	E
None of the above	0

```
SELECT job_id, decode (job_id,
                      'ST_CLERK',      'E',
                      'SA REP',        'D',
                      'IT PROG',       'C',
                      'ST MAN',        'B',
                      'AD PRES',       'A',
                      '0') GRADE
FROM employees;
```

6. CASE 구문을 사용하여 앞의 연습에 나오는 명령문을 재작성합니다.

```
SELECT job_id, CASE job_id
                  WHEN 'ST_CLERK' THEN 'E'
                  WHEN 'SA REP'   THEN 'D'
                  WHEN 'IT PROG'  THEN 'C'
                  WHEN 'ST MAN'   THEN 'B'
                  WHEN 'AD PRES'  THEN 'A'
                  ELSE '0' END   GRADE
FROM employees;
```

연습 5 해답: 그룹 함수를 사용하여 집계 데이터 보고

다음 세 명령문의 유효성을 판별합니다. 참 또는 거짓에 동그라미로 표시합니다.

- 그룹 함수는 다수 행에 실행되어 그룹당 하나의 결과를 산출합니다.

참/거짓

- 그룹 함수는 계산에 null을 포함시킵니다.

참/거짓

- WHERE 절은 그룹 계산에 포함될 행을 제한합니다.

참/거짓

HR 부서에서 다음 보고서를 요구합니다.

- 모든 사원의 최고, 최저, 합계 및 평균 급여를 찾습니다. 열 레이블을 Maximum, Minimum, Sum 및 Average로 각각 지정합니다. 결과를 가장 가까운 정수로 반올림합니다. SQL 문을 lab_05_04.sql로 저장합니다. query를 실행합니다.

```
SELECT ROUND(MAX(salary), 0) "Maximum",
       ROUND(MIN(salary), 0) "Minimum",
       ROUND(SUM(salary), 0) "Sum",
       ROUND(AVG(salary), 0) "Average"
  FROM employees;
```

- 각 직무 유형에 대해 최소, 최대, 합계 및 평균 급여를 표시하도록 lab_05_04.sql의 query를 수정합니다. lab_05_04.sql을 lab_05_05.sql로 다시 저장합니다. lab_05_05.sql의 명령문을 실행합니다.

```
SELECT job_id, ROUND(MAX(salary), 0) "Maximum",
       ROUND(MIN(salary), 0) "Minimum",
       ROUND(SUM(salary), 0) "Sum",
       ROUND(AVG(salary), 0) "Average"
  FROM employees
 GROUP BY job_id;
```

- 동일한 직무를 수행하는 사람 수를 표시하는 query를 작성합니다.

```
SELECT job_id, COUNT(*)
  FROM employees
 GROUP BY job_id;
```

HR 부서의 유저에게 직책을 입력하는 프롬프트를 표시하도록 query를 일반화합니다. lab_05_06.sql이라는 파일에 스크립트를 저장합니다. query를 실행합니다. 프롬프트가 나타나면 IT_PROG를 입력하고 OK를 누릅니다.

```
SELECT job_id, COUNT(*)
  FROM employees
 WHERE job_id = '&job_title'
 GROUP BY job_id;
```

연습 5 해답: 그룹 함수를 사용하여 집계 데이터 보고(계속)

7. 관리자를 나열하지 않는 채로 관리자 수를 확인합니다. 열 레이블을 Number of Managers로 지정합니다. 힌트: MANAGER_ID 열을 사용하여 관리자 수를 확인합니다.

```
SELECT COUNT(DISTINCT manager_id) "Number of Managers"
FROM employees;
```

8. 최고 급여와 최저 급여의 차이를 알아냅니다. 열 레이블을 DIFFERENCE로 지정합니다.

```
SELECT MAX(salary) - MIN(salary) DIFFERENCE
FROM employees;
```

시간 여유가 있을 경우 다음 연습을 완료하십시오.

9. 관리자 번호 및 해당 관리자의 부하 사원 중 최저 급여를 받는 사원의 급여를 표시하는 보고서를 작성합니다. 관리자가 알려져 있지 않은 모든 사원을 제외합니다. 최소 급여가 \$6,000 이하인 그룹을 제외시킵니다. 급여의 내림차순으로 출력을 정렬합니다.

```
SELECT manager_id, MIN(salary)
FROM employees
WHERE manager_id IS NOT NULL
GROUP BY manager_id
HAVING MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

심화 연습에 도전하려면 다음 연습을 완료하십시오.

10. 사원의 총 수와 1995년, 1996년, 1997년, 1998년에 채용된 사원의 수를 표시하는 query를 작성합니다. 적절한 열 머리글을 지정하십시오.

```
SELECT COUNT(*) total,
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1995, 1, 0)) "1995",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1996, 1, 0)) "1996",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1997, 1, 0)) "1997",
       SUM(DECODE(TO_CHAR(hire_date, 'YYYY'), 1998, 1, 0)) "1998"
  FROM employees;
```

연습 5 해답: 그룹 함수를 사용하여 집계 데이터 보고(계속)

11. 부서 20, 50, 80 및 90에 대해 직무, 부서 번호별 해당 직무에 대한 급여 및 해당 직무에 대한 총 급여를 표시하고 각 열에 적절한 머리글을 지정하기 위한 행렬 query를 작성합니다.

```
SELECT    job_id "Job",
          SUM(DECODE(department_id , 20, salary)) "Dept 20",
          SUM(DECODE(department_id , 50, salary)) "Dept 50",
          SUM(DECODE(department_id , 80, salary)) "Dept 80",
          SUM(DECODE(department_id , 90, salary)) "Dept 90",
          SUM(salary) "Total"
FROM      employees
GROUP BY job_id;
```

연습 6 해답: 다중 테이블 데이터 표시

1. HR 부서를 위해 모든 부서의 주소를 생성하는 query를 작성합니다. LOCATIONS 및 COUNTRIES 테이블을 사용하십시오. 출력에 번지, 동/리, 구/군, 시/도 및 국가를 표시합니다. NATURAL JOIN을 사용하여 결과를 생성합니다.

```
SELECT location_id, street_address, city, state_province,
       country_name
  FROM   locations
NATURAL JOIN countries;
```

2. HR 부서에서 모든 사원에 대한 보고서를 요구합니다. 모든 사원의 성, 부서 번호 및 부서 이름을 표시하는 query를 작성합니다.

```
SELECT last_name, department_id, department_name
  FROM   employees
JOIN   departments
  USING (department_id);
```

3. HR 부서에서 Toronto에 근무하는 사원에 대한 보고서를 요구합니다. Toronto에서 근무하는 모든 사원의 성, 직무, 부서 번호 및 부서 이름을 표시합니다.

```
SELECT e.last_name, e.job_id, e.department_id, d.department_name
  FROM   employees e JOIN departments d
  ON     (e.department_id = d.department_id)
JOIN   locations l
  ON     (d.location_id = l.location_id)
 WHERE LOWER(l.city) = 'toronto';
```

4. 사원의 성 및 사원 번호를 해당 관리자의 성 및 관리자 번호와 함께 표시하는 보고서를 작성합니다. 열 레이블을 Employee, Emp#, Manager 및 Mgr#으로 각각 지정합니다. SQL 문을 lab_06_04.sql로 저장합니다. query를 실행합니다.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
  FROM   employees w join employees m
  ON     (w.manager_id = m.employee_id);
```

5. King을 비롯하여 해당 관리자가 지정되지 않은 모든 사원을 표시하도록 lab_06_04.sql을 수정합니다. 사원 번호순으로 결과를 정렬합니다. SQL 문을 lab_06_05.sql로 저장합니다. lab_06_05.sql의 query를 실행합니다.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
  FROM   employees w
LEFT   OUTER JOIN employees m
  ON     (w.manager_id = m.employee_id)
 ORDER BY 2;
```

연습 6 해답: 다중 테이블 데이터 표시(계속)

6. HR 부서를 위해 사원의 성과 부서 번호 및 주어진 사원과 동일한 부서에 근무하는 모든 사원을 표시하는 보고서를 작성합니다. 각 열에 적절한 레이블을 지정합니다. lab_06_06.sql이라는 파일에 스크립트를 저장합니다. query를 실행합니다.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
  FROM employees e JOIN employees c
 WHERE (e.department_id = c.department_id)
 ORDER BY e.department_id, e.last_name, c.last_name;
```

7. HR 부서에서 직급 및 급여에 대한 보고서를 요구합니다. JOB_GRADES 테이블에 익숙해지도록 먼저 JOB_GRADES 테이블의 구조를 표시합니다. 그런 다음 모든 사원의 이름, 직무, 부서 이름, 급여 및 등급을 표시하는 query를 작성합니다.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
  FROM employees e JOIN departments d
 WHERE (e.department_id = d.department_id)
  JOIN job_grades j
 WHERE (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

심화 연습에 도전하려면 다음 연습을 완료하십시오.

8. HR 부서에서 Davies 이후에 채용된 모든 사원의 이름을 파악하려고 합니다. 사원 Davies 이후로 채용된 모든 사원의 이름과 채용 날짜를 표시하기 위한 query를 작성합니다.

```
SELECT e.last_name, e.hire_date
  FROM employees e JOIN employees davies
 WHERE (davies.last_name = 'Davies')
 ORDER BY davies.hire_date < e.hire_date;
```

9. HR 부서에서 관리자보다 먼저 채용된 모든 사원의 이름과 채용 날짜 및 해당 관리자의 이름과 채용 날짜를 찾으려고 합니다. lab_06_09.sql이라는 파일에 스크립트를 저장합니다.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
  FROM employees w JOIN employees m
 WHERE (w.manager_id = m.employee_id)
 ORDER BY w.hire_date < m.hire_date;
```

연습 7 해답: subquery를 사용하여 query 해결

- HR 부서에서 유저에게 사원의 성을 입력하라는 프롬프트를 표시하는 query를 요구합니다. 그런 다음 이 query는 유저가 입력한 이름의 사원과 동일한 부서에서 근무하는 사원의 성과 채용 날짜를 표시합니다(입력한 사원은 제외). 예를 들어, 유저가 Zlotkey를 입력하면 Zlotkey와 함께 근무하는 모든 사원을 찾습니다(Zlotkey는 제외).

```
UNDEFINE Enter_name

SELECT last_name, hire_date
FROM employees
WHERE department_id = (SELECT department_id
                        FROM employees
                        WHERE last_name = '&&Enter_name')
AND last_name <> '&Enter_name';
```

- 평균 급여 이상을 받는 모든 사원의 사원 번호, 성 및 급여를 표시하는 보고서를 작성합니다. 오름차순으로 급여의 결과를 정렬합니다.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary)
                  FROM employees)
ORDER BY salary;
```

- 성에 "u"가 포함된 사원과 같은 부서에 근무하는 모든 사원의 사원 번호와 성을 표시하는 query를 작성합니다. 작성한 SQL 문을 lab_07_03.sql로 저장합니다. query를 실행합니다.

```
SELECT employee_id, last_name
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM employees
                        WHERE last_name like '%u%');
```

- HR 부서에서 부서 위치 ID가 1700인 모든 사원의 성, 부서 번호 및 작업 ID를 표시하는 보고서를 요구합니다.

```
SELECT last_name, department_id, job_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE location_id = 1700);
```

유저에게 번지를 입력하는 프롬프트를 표시하도록 query를 수정합니다. 입력한 내용을 lab_07_04.sql이라는 파일에 저장합니다.

```
SELECT last_name, department_id, job_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE location_id = &Enter_location);
```

연습 7 해답: subquery를 사용하여 query 해결(계속)

5. HR을 위해 King에게 보고하는 모든 사원의 성과 급여를 표시하는 보고서를 작성합니다.

```
SELECT last_name, salary
FROM employees
WHERE manager_id = (SELECT employee_id
                      FROM employees
                      WHERE last_name = 'King');
```

6. HR을 위해 Executive 부서의 모든 사원에 대해 부서 번호, 성 및 직무 ID를 표시하는 보고서를 작성합니다.

```
SELECT department_id, last_name, job_id
FROM employees
WHERE department_id IN (SELECT department_id
                          FROM departments
                          WHERE department_name = 'Executive');
```

시간 여유가 있을 경우 다음 연습을 완료하십시오.

7. 평균보다 많은 급여를 받고 성에 "u"가 포함된 사원이 있는 부서에서 근무하는 모든 사원의 사원 번호, 성 및 급여를 표시하도록 lab_07_03.sql의 query를 수정합니다. lab_07_03.sql을 lab_07_07.sql로 다시 저장합니다.
lab_07_07.sql의 명령문을 실행합니다.

```
SELECT employee_id, last_name, salary
FROM employees
WHERE department_id IN (SELECT department_id
                          FROM employees
                          WHERE last_name like '%u%')
AND salary > (SELECT AVG(salary)
               FROM employees);
```

연습 8 해답: 집합 연산자 사용

1. HR 부서에서 직무 ID ST_CLERK을 포함하지 않는 부서에 대한 부서 ID 리스트를 요구합니다. 집합 연산자를 사용하여 이 보고서를 작성합니다.

```
SELECT department_id
FROM departments
MINUS
SELECT department_id
FROM employees
WHERE job_id = 'ST_CLERK';
```

2. HR 부서에서 부서가 소재하지 않는 국가의 리스트를 요구합니다. 해당 국가의 국가 ID 및 이름을 표시합니다. 집합 연산자를 사용하여 이 보고서를 작성합니다.

```
SELECT country_id, country_name
FROM countries
MINUS
SELECT l.country_id, c.country_name
FROM locations l JOIN countries c
ON (l.country_id = c.country_id)
JOIN departments d
ON d.location_id=l.location_id;
```

3. 부서 10, 50 및 20에 대한 직무 리스트를 이 순서대로 생성합니다. 집합 연산자를 사용하여 직무 ID 및 부서 ID를 표시합니다.

```
SELECT distinct job_id, department_id
FROM employees
WHERE department_id = 10
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 50
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 20
```

연습 8 해답: 집합 연산자 사용(계속)

4. 현재 직책이 회사에 처음 입사할 때의 직책과 동일한 사원(즉, 직무가 변경된 적은 있지만 현재 원래 직무로 복귀한 사원)의 사원 ID와 직무 ID를 나열하는 보고서를 작성합니다.

```
SELECT      employee_id, job_id
FROM        employees
INTERSECT
SELECT      employee_id, job_id
FROM        job_history;
```

5. HR 부서에서 다음과 같은 사양의 보고서를 요구합니다.

- 부서에 소속되었는지 여부에 관계없이 EMPLOYEES 테이블에 있는 모든 사원의 성 및 부서 ID
- 해당 부서에 근무 중인 사원이 있는지 여부에 관계없이 DEPARTMENTS 테이블에 있는 모든 부서의 부서 ID 및 부서 이름

이를 수행하는 복합 질의를 작성합니다.

```
SELECT last_name, department_id, TO_CHAR(null)
FROM   employees
UNION
SELECT TO_CHAR(null), department_id, department_name
FROM   departments;
```

연습 9 해답: 데이터 조작

HR 부서에서 사원 데이터를 삽입, 갱신 및 삭제하는 SQL 문을 작성해 달라고 합니다. HR 부서에 명령문을 제공하기에 앞서 프로토타입으로 MY_EMPLOYEE 테이블을 사용합니다.

주: 모든 DML 문의 경우에는 Run Script 아이콘 또는 [F5]를 눌러 query를 실행합니다. 이렇게 하면 Script Output 탭 페이지에서 피드백 메시지를 볼 수 있습니다. SELECT query의 경우 계속해서 Execute Statement 아이콘을 누르거나 [F9]를 누르면 Results 탭 페이지에서 형식이 지정된 출력을 볼 수 있습니다.

MY_EMPLOYEE 테이블에 데이터를 삽입합니다.

1. 이 연습에 사용되는 MY_EMPLOYEE 테이블을 생성하려면 lab_09_01.sql 스크립트의 명령문을 실행합니다.
 - a. File 메뉴에서 Open을 선택합니다. Open 대화상자에서 D:\labs\sql1\labs 폴더로 이동한 다음 lab_09_01.sql을 두 번 누릅니다.
 - b. SQL Worksheet에서 명령문이 열리면 Run Script 아이콘을 눌러 스크립트를 실행합니다. Script Output 탭 페이지에 테이블 생성 성공 메시지가 나타납니다.
2. 열 이름을 식별하도록 MY_EMPLOYEE 테이블의 구조를 기술합니다.

```
DESCRIBE my_employee
```

3. 다음 예제 데이터의 첫번째 데이터 행을 MY_EMPLOYEE 테이블에 추가하는 INSERT 문을 작성합니다. INSERT 절에 열을 나열하지 마십시오.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

연습 9 해답: 데이터 조작(계속)

4. 앞의 리스트에 있는 예제 데이터의 두번째 행으로 MY_EMPLOYEE 테이블을 채웁니다. 이번에는 INSERT 절에 명시적으로 열을 나열합니다.

```
INSERT INTO my_employee (id, last_name, first_name,
                         userid, salary)
VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. 테이블에 추가한 내용을 확인합니다.

```
SELECT *
FROM my_employee;
```

6. 나머지 행을 MY_EMPLOYEE 테이블에 로드하는 insert 문을 재사용 가능한 동적 스크립트 파일에 작성합니다. 이 스크립트는 유저에게 모든 열(ID, LAST_NAME, FIRST_NAME, USERID 및 SALARY)에 대해 입력하도록 요구해야 합니다. lab_09_06.sql이라는 파일에 이 스크립트를 저장합니다.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
       '&p_userid', &p_salary);
```

7. 작성한 스크립트에서 INSERT 문을 실행하여 3단계에 나열된 예제 데이터의 다음 두 행으로 테이블을 채웁니다.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
       '&p_userid', &p_salary);
```

8. 테이블에 추가한 내용을 확인합니다.

```
SELECT *
FROM my_employee;
```

9. 데이터 추가 내용이 영구적으로 적용되도록 합니다.

```
COMMIT;
```

연습 9 해답: 데이터 조작(계속)

MY_EMPLOYEE 테이블에서 데이터를 생성하고 삭제합니다.

10. 사원 3의 성을 Drexler로 변경합니다.

```
UPDATE my_employee
SET last_name = 'Drexler'
WHERE id = 3;
```

11. 급여가 \$900 미만인 모든 사원에 대해 급여를 \$1000로 변경합니다.

```
UPDATE my_employee
SET salary = 1000
WHERE salary < 900;
```

12. 테이블에 대한 변경 사항을 확인합니다.

```
SELECT *
FROM my_employee;
```

13. MY_EMPLOYEE 테이블에서 Betty Dancs를 삭제합니다.

```
DELETE
FROM my_employee
WHERE last_name = 'Dancs';
```

14. 테이블에 대한 변경 사항을 확인합니다.

```
SELECT *
FROM my_employee;
```

15. 보류 중인 모든 변경 사항을 커밋합니다.

```
COMMIT;
```

MY_EMPLOYEE 테이블에 대한 데이터 트랜잭션을 제어합니다.

16. 6단계에서 작성한 스크립트의 명령문을 사용하여 3단계에 나열된 예제 데이터의 마지막 행으로 테이블을 채웁니다. 스크립트의 명령문을 실행합니다.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
'&p_userid', &p_salary);
```

연습 9 해답: 데이터 조작(계속)

17. 테이블에 추가한 내용을 확인합니다.

```
SELECT *
FROM my_employee;
```

18. 트랜잭션 처리의 중간 지점에 표시합니다.

```
SAVEPOINT step_17;
```

19. MY_EMPLOYEE 테이블의 모든 행을 삭제합니다.

```
DELETE
FROM my_employee;
```

20. 테이블이 비어 있는지 확인합니다.

```
SELECT *
FROM my_employee;
```

21. 이전의 INSERT 작업을 삭제하지 않은 채로 가장 최근의 DELETE 작업을 삭제합니다.

```
ROLLBACK TO step_17;
```

22. 새 행이 여전히 원래 상태를 유지하는지 확인합니다.

```
SELECT *
FROM my_employee;
```

23. 데이터 추가 내용이 영구적으로 적용되도록 합니다.

```
COMMIT;
```

시간 여유가 있을 경우 다음 연습을 완료하십시오.

24. 이름의 첫번째 문자와 성의 앞부분 일곱 문자를 연결하여 USERID를 자동으로 생성하도록 lab_09_06.sql 스크립트를 수정합니다. 생성된 USERID는 소문자여야 합니다. 따라서 이 스크립트는 유저에게 USERID를 입력하도록 요구하지 않아야 합니다. lab_09_24.sql이라는 파일에 이 스크립트를 저장합니다.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES ('&p_id', '&&p_last_name', '&&p_first_name',
        lower(substr('&p_first_name', 1, 1)) ||
        substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

연습 9 해답: 데이터 조작(계속)

25. lab_09_24.sql 스크립트를 실행하여 다음 레코드를 삽입합니다.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

26. 새 행에 올바른 USERID가 추가되었는지 확인합니다.

```
SELECT *
FROM my_employee
WHERE ID='6';
```

연습 10 해답: DDL 문을 사용하여 테이블 생성 및 관리

주: 모든 DDL 및 DML 문의 경우 Run Script 아이콘 또는 [F5]를 눌러 query를 실행합니다. 이렇게 하면 Script Output 탭 페이지에서 피드백 메시지를 볼 수 있습니다. SELECT query의 경우 계속해서 Execute Statement 아이콘을 누르거나 [F9]를 누르면 Results 탭 페이지에서 형식이 지정된 출력을 볼 수 있습니다.

1. 다음 테이블 instance 차트에 준하여 DEPT 테이블을 생성합니다.

lab_10_01.sql이라는 스크립트에 명령문을 저장한 다음 해당 스크립트의 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

```
CREATE TABLE dept
  (id      NUMBER(7) CONSTRAINT department_id_pk PRIMARY KEY,
   name    VARCHAR2(25));
```

- a. 테이블이 생성되었는지 확인하고 테이블 구조를 보려면 다음 명령을 실행합니다.

```
DESCRIBE dept
```

2. DEPARTMENTS 테이블의 데이터로 DEPT 테이블을 채웁니다. 필요한 열만 포함시키십시오.

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

3. 다음 테이블 instance 차트에 준하여 EMP 테이블을 생성합니다.

lab_10_03.sql이라는 스크립트에 명령문을 저장한 다음 해당 스크립트의 명령문을 실행하여 테이블을 생성합니다. 테이블이 생성되었는지 확인합니다.

```
CREATE TABLE emp
  (id          NUMBER(7),
   last_name   VARCHAR2(25),
   first_name  VARCHAR2(25),
   dept_id     NUMBER(7)
   CONSTRAINT emp_dept_id_FK REFERENCES dept (id)
  );
```

- a. 테이블이 생성되었는지 확인하고 테이블 구조를 보려면 다음 명령을 실행합니다.

```
DESCRIBE emp
```

연습 10 해답: DDL 문을 사용하여 테이블 생성 및 관리(계속)

4. EMPLOYEES 테이블의 구조를 기반으로 EMPLOYEES2 테이블을 생성합니다. EMPLOYEE_ID, FIRST_NAME, LAST_NAME, SALARY 및 DEPARTMENT_ID 열만 포함합니다. 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;
```

5. EMPLOYEES2 테이블 상태를 읽기 전용으로 변경합니다.

```
ALTER TABLE employees2 READ ONLY
```

6. EMPLOYEES2 테이블에 다음 행을 삽입해 봅니다.

- a. "Update operation not allowed on table" 오류 메시지가 나타납니다.
테이블의 상태가 읽기 전용으로 지정되어 있으므로 행을 추가할 수 없습니다.

```
INSERT INTO employees2
VALUES (34, 'Grant', 'Marcie', 5678, 10)
```

7. EMPLOYEES2 테이블의 상태를 읽기/쓰기로 되돌립니다. 이제 동일한 행을 다시 삽입해 봅니다.

- a. 테이블의 상태가 READ WRITE로 지정되어 있으므로 행을 삽입할 수 있습니다.

```
ALTER TABLE employees2 READ WRITE

INSERT INTO employees2
VALUES (34, 'Grant', 'Marcie', 5678, 10)
```

8. EMPLOYEES2 테이블을 삭제하십시오.

- a. 주: READ ONLY 모드인 테이블도 삭제할 수 있습니다. 이를 테스트하려면 테이블을 다시 READ ONLY 상태로 변경한 다음 DROP TABLE 명령을 실행합니다. EMPLOYEES2 테이블이 삭제됩니다.

```
DROP TABLE employees2;
```

연습 11 해답: 다른 스키마 객체 생성

1부

- HR 부서의 임원이 EMPLOYEES 테이블의 일부 데이터를 숨길 것을 요구합니다. EMPLOYEES 테이블의 사원 번호, 사원 성 및 부서 번호를 기반으로 하는 EMPLOYEES_VU라는 뷰를 요구합니다. 사원 이름의 머리글을 EMPLOYEE로 표시할 것을 원합니다.

```
CREATE OR REPLACE VIEW employees_vu AS
    SELECT employee_id, last_name employee, department_id
    FROM employees;
```

- 뷰가 작동하는지 확인합니다. EMPLOYEES_VU 뷰의 내용을 표시합니다.

```
SELECT *
FROM employees_vu;
```

- EMPLOYEES_VU 뷰를 사용하여 HR 부서를 위해 모든 사원 이름과 부서 번호를 표시하는 query를 작성합니다.

```
SELECT employee, department_id
FROM employees_vu;
```

- 부서 50의 사원 데이터에 액세스하려고 합니다. 부서 50의 모든 사원에 대해 사원 번호, 사원 성 및 부서 번호를 포함하는 DEPT50이라는 뷰를 생성합니다. 뷰 열의 레이블을 각각 EMPNO, EMPLOYEE 및 DEPTNO로 지정하라는 요청을 받았습니다. 보안상의 이유로 사원이 뷰를 통해 다른 부서에 재할당되는 것을 허용하지 않습니다.

```
CREATE VIEW dept50 AS
    SELECT employee_id empno, last_name employee,
           department_id deptno
    FROM employees
    WHERE department_id = 50
    WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

- DEPT50 뷰의 구조 및 내용을 표시합니다.

```
DESCRIBE dept50

SELECT *
FROM dept50;
```

- 뷰를 테스트합니다. Matos를 부서 80에 재할당해 보십시오.

```
UPDATE dept50
SET deptno = 80
WHERE employee = 'Matos';
```

DEPT50 뷰가 WITH CHECK OPTION 제약 조건을 사용하여 생성되었으므로 오류가 발생합니다. 이 제약 조건은 뷰의 DEPTNO 열이 변경되지 않도록 합니다.

연습 11 해답: 다른 스키마 객체 생성(계속)

2부

7. DEPT 테이블의 primary key 열로 사용할 수 있는 시퀀스가 필요합니다. 시퀀스는 200부터 시작하고 최대값은 1,000이어야 합니다. 시퀀스 증분 값을 10으로 설정하고 시퀀스 이름을 DEPT_ID_SEQ로 지정합니다.

```
CREATE SEQUENCE dept_id_seq
START WITH 200
INCREMENT BY 10
MAXVALUE 1000;
```

8. 시퀀스를 테스트하기 위해 DEPT 테이블에 두 행을 삽입하는 스크립트를 작성합니다. 스크립트에 lab_11_08.sql이라는 이름을 지정합니다. ID 열에 대해 생성한 시퀀스를 사용해야 합니다. Education 및 Administration이라는 두 개의 부서를 추가합니다. 추가한 내용을 확인합니다. 스크립트에서 해당 명령을 실행합니다.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

9. DEPT 테이블의 NAME 열에 Nonunique 인덱스를 생성합니다.

```
CREATE INDEX dept_name_idx ON dept (name);
```

10. EMPLOYEES 테이블의 동의어를 생성합니다. EMP라는 이름을 지정합니다.

```
CREATE SYNONYM emp FOR EMPLOYEES;
```

연습 C 해답: Oracle 조인 구문

1. HR 부서를 위해 모든 부서의 주소를 생성하는 query를 작성합니다. LOCATIONS 및 COUNTRIES 테이블을 사용하십시오. 출력에 번지, 동/리, 구/군, 시/도 및 국가를 표시합니다. query를 실행합니다.

```
SELECT location_id, street_address, city, state_province,
       country_name
  FROM   locations, countries
 WHERE  locations.country_id = countries.country_id;
```

2. HR 부서에서 모든 사원에 대한 보고서를 요구합니다. 모든 사원의 성, 부서 번호 및 부서 이름을 표시하는 query를 작성합니다. query를 실행합니다.

```
SELECT e.last_name, e.department_id, d.department_name
  FROM   employees e, departments d
 WHERE  e.department_id = d.department_id;
```

3. HR 부서에서 Toronto에 근무하는 사원에 대한 보고서를 요구합니다. Toronto에서 근무하는 모든 사원의 성, 직무, 부서 번호 및 부서 이름을 표시합니다.

```
SELECT e.last_name, e.job_id, e.department_id, d.department_name
  FROM   employees e, departments d, locations l
 WHERE  e.department_id = d.department_id
   AND  d.location_id = l.location_id
   AND  LOWER(l.city) = 'toronto';
```

4. 사원의 성 및 사원 번호를 해당 관리자의 성 및 관리자 번호와 함께 표시하는 보고서를 작성합니다. 열 레이블을 Employee, Emp#, Manager 및 Mgr#으로 각각 지정합니다. SQL 문을 lab_c_04.sql로 저장합니다.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
  FROM   employees w, employees m
 WHERE  w.manager_id = m.employee_id;
```

5. King을 비롯하여 해당 관리자가 지정되지 않은 모든 사원을 표시하도록 lab_c_04.sql을 수정합니다. 사원 번호순으로 결과를 정렬합니다. SQL 문을 lab_c_05.sql로 저장합니다. lab_c_05.sql의 query를 실행합니다.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id "Mgr#"
  FROM   employees w, employees m
 WHERE  w.manager_id = m.employee_id (+);
```

연습 C 해답: Oracle 조인 구문(계속)

6. HR 부서를 위해 사원의 성과 부서 번호 및 주어진 사원과 동일한 부서에 근무하는 모든 사원을 표시하는 보고서를 작성합니다. 각 열에 적절한 레이블을 지정합니다. lab_c_06.sql이라는 파일에 스크립트를 저장합니다.

```
SELECT e1.department_id department, e1.last_name employee,
       e2.last_name colleague
  FROM employees e1, employees e2
 WHERE e1.department_id = e2.department_id
   AND e1.employee_id <> e2.employee_id
ORDER BY e1.department_id, e1.last_name, e2.last_name;
```

7. HR 부서에서 직급 및 급여에 대한 보고서를 요구합니다. JOB_GRADES 테이블에 익숙해지도록 먼저 JOB_GRADES 테이블의 구조를 표시합니다. 그런 다음 모든 사원의 이름, 직무, 부서 이름, 급여 및 등급을 표시하는 query를 작성합니다.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
  FROM employees e, departments d, job_grades j
 WHERE e.department_id = d.department_id
   AND e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

심화 연습에 도전하려면 다음 연습을 완료하십시오.

8. HR 부서에서 Davies 이후에 채용된 모든 사원의 이름을 파악하려고 합니다. 사원 Davies 이후로 채용된 모든 사원의 이름과 채용 날짜를 표시하기 위한 query를 작성합니다.

```
SELECT e.last_name, e.hire_date
  FROM employees e , employees davies
 WHERE davies.last_name = 'Davies'
   AND davies.hire_date < e.hire_date;
```

9. HR 부서에서 관리자보다 먼저 채용된 모든 사원의 이름과 채용 날짜 및 해당 관리자의 이름과 채용 날짜를 찾으려고 합니다. 열 레이블을 Employee, Emp Hired, Manager 및 Mgr Hired로 각각 지정합니다. lab_c_09.sql이라는 파일에 스크립트를 저장합니다.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
  FROM employees w , employees m
 WHERE w.manager_id = m.employee_id
   AND w.hire_date < m.hire_date;
```

추가 연습

추가 연습

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

1. HR 부서에서 1997년 이후 채용된 모든 clerk에 대한 데이터를 찾으려고 합니다.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	143 Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600
2	144 Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500

2. HR 부서에서 커미션을 받는 사원에 대한 보고서를 요구합니다. 해당 사원의 성, 직무, 급여 및 커미션을 표시합니다. 급여의 내림차순으로 데이터를 정렬합니다.

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
Abel	SA_REP	11000	0.3
Zlotkey	SA_MAN	10500	0.2
Taylor	SA_REP	8600	0.2
Grant	SA_REP	7000	0.15

3. HR 부서에서 예산 책정을 위해 예상되는 급여 인상에 대한 보고서를 요구합니다. 이 보고서는 커미션을 받지 않지만 급여가 10% 인상되는 사원을 표시해야 합니다(급여 반올림).

New salary
1 The salary of King after a 10% raise is 26400
2 The salary of Kochhar after a 10% raise is 18700
3 The salary of De Haan after a 10% raise is 18700
4 The salary of Hunold after a 10% raise is 9900
5 The salary of Ernst after a 10% raise is 6600
6 The salary of Lorentz after a 10% raise is 4620
7 The salary of Mourgos after a 10% raise is 6380
8 The salary of Rajs after a 10% raise is 3850
9 The salary of Davies after a 10% raise is 3410
10 The salary of Matos after a 10% raise is 2860
11 The salary of Vargas after a 10% raise is 2750
12 The salary of Whalen after a 10% raise is 4840
13 The salary of Hartstein after a 10% raise is 14300
14 The salary of Fay after a 10% raise is 6600
15 The salary of Higgins after a 10% raise is 13200
16 The salary of Gietz after a 10% raise is 9130

추가 연습(계속)

4. 사원 및 근속 기간에 대한 보고서를 작성합니다. 모든 사원들의 성 및 근무 기간(년, 개월)을 함께 표시합니다. 근속 기간별로 보고서를 정렬합니다. 근속 기간이 가장 긴 사원이 리스트의 맨 위에 나타나야 합니다.

	LAST_NAME	YEARS	MONTHS
1	King	20	0
2	Whalen	19	9
3	Kochhar	17	9
4	Hunold	17	5
5	Ernst	16	1

• • •

17	Lorentz	8	4
18	Grant	8	1
19	Mourgos	7	7
20	Zlotkey	7	5

5. 성이 "J", "K", "L" 또는 "M"으로 시작하는 사원을 표시합니다.

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos

6. 모든 사원을 표시하고 각 사원이 커미션을 받는지 여부를 Yes 또는 No로 나타내는 보고서를 작성합니다. query에서 DECODE 식을 사용합니다.

주: 결과는 다음 페이지로 이어집니다.

	LAST_NAME	SALARY	COMMISSION
1	King	24000	No
2	Kochhar	17000	No
3	De Haan	17000	No
4	Hunold	9000	No
5	Ernst	6000	No
6	Lorentz	4200	No
7	Mourgos	5800	No
8	Rajs	3500	No
9	Davies	3100	No
10	Matos	2600	No

추가 연습(계속)

6. (계속)

11	Vargas	2500	No
12	Zlotkey	10500	Yes
13	Abel	11000	Yes
14	Taylor	8600	Yes
15	Grant	7000	Yes
16	Whalen	4400	No
17	Hartstein	13000	No
18	Fay	6000	No
19	Higgins	12000	No
20	Gietz	8300	No

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수, 조인, 그룹 함수 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

7. 특정 위치에서 근무하는 사원의 부서 이름, 위치 ID, 성, 직책 및 급여를 표시하는 보고서를 작성합니다. 유저에게 위치를 입력하는 프롬프트를 표시합니다. 예를 들어, 유저가 1800을 입력하면 출력 결과는 다음과 같습니다.

	DEPARTMENT_NAME	LOCATION_ID	LAST_NAME	JOB_ID	SALARY
1	Marketing	1800	Hartstein	MK_MAN	13000
2	Marketing	1800	Fay	MK_REP	6000

8. 성이 "n"으로 끝나는 사원의 수를 알아냅니다. 가능한 두 가지 해결책을 작성합니다.

	COUNT(*)
1	3

각 부서에 대한 이름, 위치 및 사원 수를 보여주는 보고서를 작성합니다. 보고서에 사원이 없는 부서도 포함되는지 확인합니다.

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	COUNT(E.EMPLOYEE_ID)
1	80	Sales	2500	3
2	110	Accounting	1700	2
3	10	Administration	1700	1
4	60	IT	1400	3
5	20	Marketing	1800	2
6	90	Executive	1700	3
7	50	Shipping	1500	5
8	190	Contracting	1700	0

추가 연습(계속)

9. HR 부서에서 부서 번호 10 및 20에 있는 직책을 찾으려고 합니다. 해당 부서의 직무 ID를 표시하는 보고서를 작성합니다.

JOB_ID
1 AD_ASST
2 MK_MAN
3 MK_REP

10. Administration 및 Executive 부서에서 찾은 직무를 표시하는 보고서를 작성합니다. 또한 해당 직무에 대한 사원 수도 표시합니다. 사원 수가 가장 많은 직무를 가장 먼저 표시합니다.

JOB_ID	FREQUENCY
1 AD_VP	2
2 AD_PRES	1
3 AD_ASST	1

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수, 조인, 그룹 함수, subquery 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

11. 각 월의 16일 이전에 채용된 사원을 모두 표시합니다.

LAST_NAME	HIRE_DATE
1 De Haan	13-JAN-93
2 Hunold	03-JAN-90
3 Lorentz	07-FEB-99
4 Matos	15-MAR-98
5 Vargas	09-JUL-98
6 Abel	11-MAY-96
7 Higgins	07-JUN-94
8 Gietz	07-JUN-94

추가 연습(계속)

12. 모든 사원에 대해 성, 급여 및 \$1000 단위로 표현된 급여를 표시하는 보고서를 작성합니다.

주: 결과는 다음 페이지로 이어집니다.

	LAST_NAME	SALARY	THOUSANDS
1	King	24000	24
2	Kochhar	17000	17
3	De Haan	17000	17
4	Hunold	9000	9
5	Ernst	6000	6
6	Lorentz	4200	4
7	Mourgos	5800	5
8	Rajs	3500	3
9	Davies	3100	3
10	Matos	2600	2

• • •

11	Vargas	2500	2
12	Zlotkey	10500	10
13	Abel	11000	11
14	Taylor	8600	8
15	Grant	7000	7
16	Whalen	4400	4
17	Hartstein	13000	13
18	Fay	6000	6
19	Higgins	12000	12
20	Gietz	8300	8

13. 급여가 \$15,000 이상인 관리자 휘하의 모든 사원을 표시합니다. 사원 이름, 관리자 이름, 관리자 급여 및 관리자의 급여 등급을 표시합니다.

	LAST_NAME	MANAGER	SALARY	GRADE_LEVEL
1	Whalen	Kochhar	17000 E	
2	Higgins	Kochhar	17000 E	
3	Hunold	De Haan	17000 E	
4	Kochhar	King	24000 E	
5	De Haan	King	24000 E	
6	Mourgos	King	24000 E	
7	Zlotkey	King	24000 E	
8	Hartstein	King	24000 E	

추가 연습(계속)

14. 모든 부서의 부서 번호, 이름, 사원 수, 평균 급여와 각 부서에서 일하는 사원의 이름, 급여 및 직무를 표시합니다.

DEPARTMENT_ID	DEPARTMENT_NAME	EMPLOYEES	Avg_Sal	Last_Name	Salary	Job_ID
1	10 Administration	1	4400.00	Whalen	4400	AD_ASST
2	20 Marketing	2	9500.00	Hartstein	13000	MK_MAN
3	20 Marketing	2	9500.00	Fay	6000	MK_REP
4	50 Shipping	5	3500.00	Davies	3100	ST_CLERK
5	50 Shipping	5	3500.00	Matos	2600	ST_CLERK
6	50 Shipping	5	3500.00	Rajs	3500	ST_CLERK
• • •						
19	110 Accounting	2	10150.00	Higgins	12000	AC_MGR
20	(null) (null)	0	No average	Grant	7000	SA_REP

16. 평균 급여가 가장 높은 부서의 부서 번호와 최저 급여를 표시하는 보고서를 작성합니다.

DEPARTMENT_ID	MIN(SALARY)
1	90

17. 영업 사원이 근무하지 않는 부서를 표시하는 보고서를 작성합니다. 출력에 부서 번호, 부서 이름, 관리자 ID 및 위치를 포함합니다.

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	90 Executive	100	1700
6	110 Accounting	205	1700
7	190 Contracting	(null)	1700

추가 연습(계속)

18. HR 부서를 위해 통계 보고서를 작성합니다. 이 보고서에는 다음 조건의 부서에 대한 부서 번호, 부서 이름 및 근무하는 사원 수를 표시합니다.

- a. 사원 수가 3명 미만인 부서:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	10	Administration	1
2	110	Accounting	2
3	20	Marketing	2

- b. 사원 수가 가장 많은 부서:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	50	Shipping	5

- c. 사원 수가 가장 적은 부서:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	10	Administration	1

19. 모든 사원에 대해 사원 번호, 성, 급여, 부서 번호 및 해당 부서의 평균 급여를 표시하는 보고서를 작성합니다.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	SALARY	Avg(S.SALARY)
1	149	Zlotkey	80	10500	10033.333333333333...
2	174	Abel	80	11000	10033.333333333333...
3	144	Vergas	50	2500	3500
4	101	Kochhar	90	17000	19333.333333333333...
5	100	King	90	24000	19333.333333333333...
6	103	Hunold	60	9000	6400
7	142	Davies	50	3100	3500
8	205	Higgins	110	12000	10150
9	104	Ernst	60	6000	6400
10	143	Matos	50	2600	3500

• • •

18	206	Gietz	110	8300	10150
19	124	Mourgos	50	5800	3500

추가 연습(계속)

20. 사원 채용 수가 가장 많은 요일에 채용된 사원을 모두 표시합니다.

	LAST_NAME	DAY
1	Ernst	TUESDAY
2	Mourgos	TUESDAY
3	Rajs	TUESDAY
4	Taylor	TUESDAY
5	Higgins	TUESDAY
6	Gietz	TUESDAY

21. 사원의 채용일에 준하여 기념일 개요를 생성합니다. 기념일을 오름차순으로 정렬하십시오.

	LAST_NAME	BIRTHDAY
1	Hunold	January 03
2	De Haan	January 13
3	Davies	January 29
4	Zlotkey	January 29
5	Lorentz	February 07
6	Hartstein	February 17
7	Matos	March 15
8	Taylor	March 24
9	Abel	May 11
10	Ernst	May 21

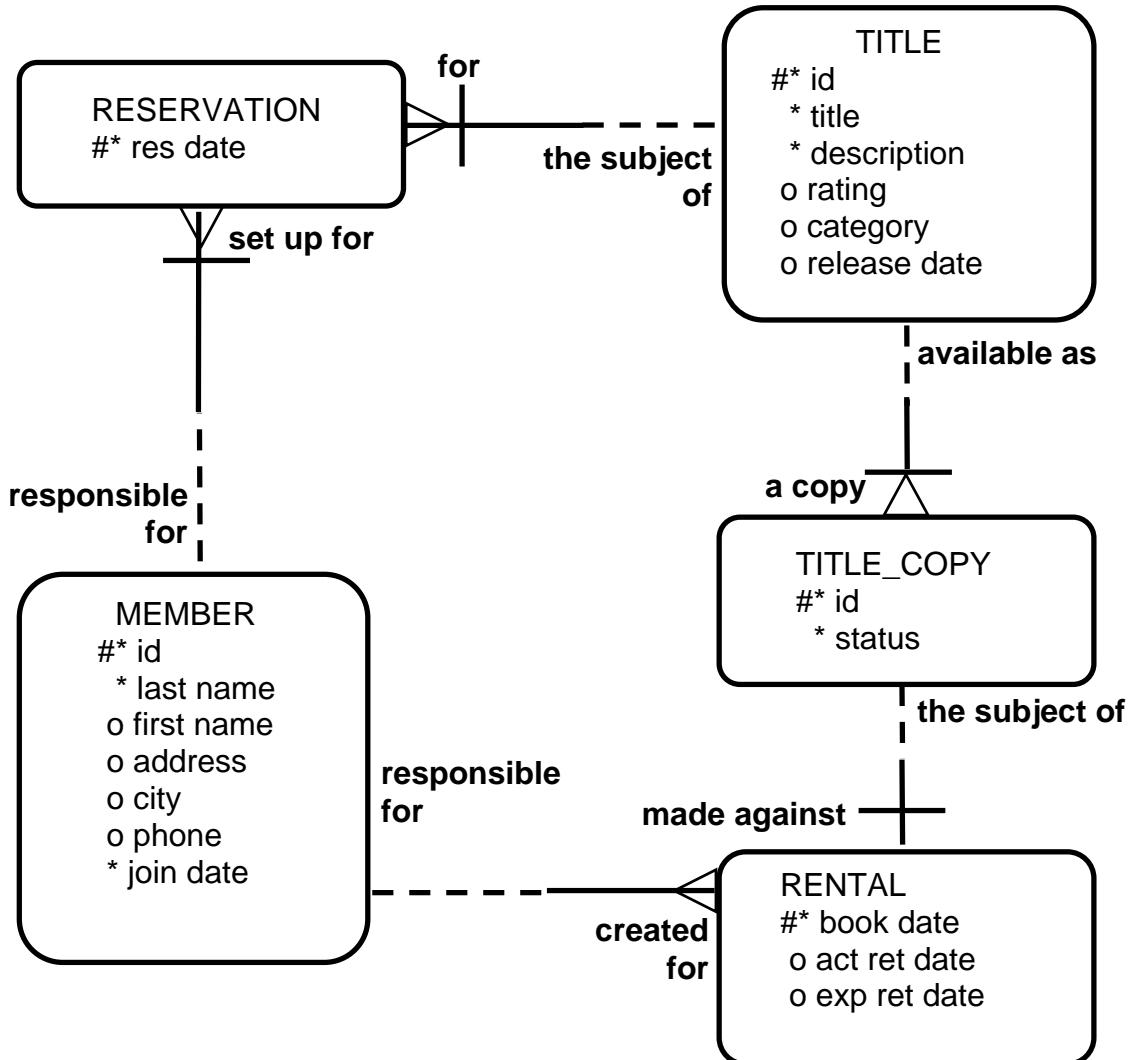
• • •

11	Grant	May	24
12	Higgins	June	07
13	Gietz	June	07
14	King	June	17
15	Vargas	July	09
16	Fay	August	17
17	Whalen	September	17
18	Kochhar	September	21
19	Rajs	October	17
20	Mourgos	November	16

추가 연습: 사례 연구

이 사례 연구에서는 비디오 응용 프로그램에 사용할 일련의 데이터베이스 테이블을 구축합니다. 테이블을 생성한 후 비디오 대여점 데이터베이스의 레코드를 삽입, 갱신 및 삭제하고 보고서를 생성합니다. 데이터베이스에는 필수 테이블만 포함됩니다.

다음은 비디오 응용 프로그램의 엔티티 및 속성을 나타낸 다이어그램입니다.



추가 연습: 사례 연구(계속)

주: 테이블을 생성하려면 SQL Developer에서 buildtab.sql 스크립트의 명령을 실행합니다. 테이블을 삭제하려면 SQL Developer에서 dropvid.sql 스크립트의 명령을 실행합니다. 그런 다음 SQL Developer에서 buildvid.sql 스크립트의 명령을 실행하여 테이블을 작성하고 채울 수 있습니다. 세 가지 sql 스크립트는 모두 D:\labs\sql1\labs 폴더에 있습니다.

- buildtab.sql 스크립트를 사용하여 테이블을 생성하는 경우 4단계부터 시작하십시오.
- dropvid.sql 스크립트를 사용하여 비디오 테이블을 제거하는 경우 1단계부터 시작하십시오.
- buildvid.sql 스크립트를 사용하여 테이블을 생성하고 채우는 경우 6(b)단계부터 시작하십시오.

1. 다음 테이블 instance 차트를 기반으로 테이블을 생성합니다. 적절한 데이터 유형을 선택하고 무결성 제약 조건을 추가합니다.

- a. 테이블 이름: MEMBER

열 이름	MEMBER_ID	LAST_NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN_DATE
키 유형	PK						
Null/ Unique	NN,U	NN					NN
기본 값							System Date
데이터 유형	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
길이	10	25	25	100	30	15	

추가 연습: 사례 연구(계속)

b. 테이블 이름: TITLE

열 이름	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_DATE
키 유형	PK					
Null/ Unique	NN,U	NN	NN			
확인				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMENTARY	
데이터 유형	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
길이	10	60	400	4	20	

c. 테이블 이름: TITLE_COPY

열 이름	COPY_ID	TITLE_ID	STATUS
키 유형	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
확인			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref 테이블		TITLE	
FK Ref 열		TITLE_ID	
데이터 유형	NUMBER	NUMBER	VARCHAR2
길이	10	10	15

추가 연습: 사례 연구(계속)

d. 테이블 이름: RENTAL

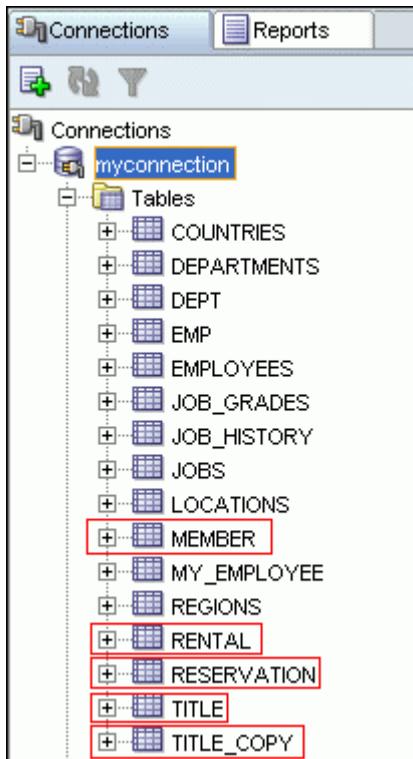
열 이름	BOOK_DATE	MEMBER_ID	COPY_ID	ACT_RET_DATE	EXP_RET_DATE	TITLE_ID
키 유형	PK	PK,FK1	PK,FK2			PK,FK2
기본 값	System Date				System Date + 2 days	
FK Ref 테이블		MEMBER	TITLE_COPY			TITLE_COPY
FK Ref 열		MEMBER_ID	COPY_ID			TITLE_ID
데이터 유형	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
길이		10	10			10

e. 테이블 이름: RESERVATION

열 이름	RES_DATE	MEMBER_ID	TITLE_ID
키 유형	PK	PK,FK1	PK,FK2
Null/Unique	NN,U	NN,U	NN
FK Ref 테이블		MEMBER	TITLE
FK Ref 열		MEMBER_ID	TITLE_ID
데이터 유형	DATE	NUMBER	NUMBER
길이		10	10

추가 연습: 사례 연구(계속)

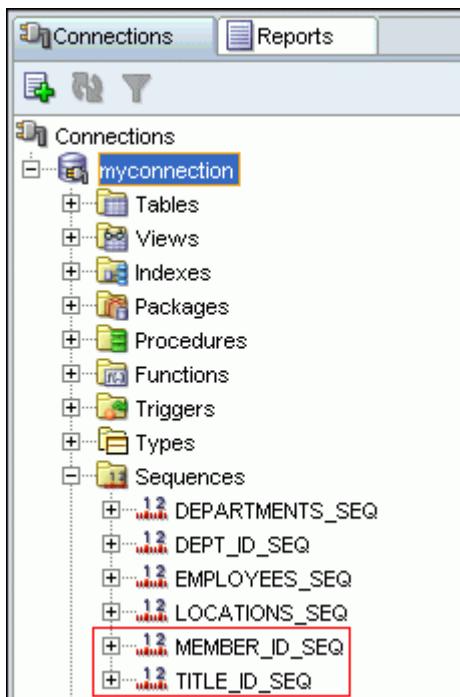
2. SQL Developer의 Connections Navigator에서 검사를 진행하여 해당 테이블이 제대로 생성되었는지 확인합니다.



3. MEMBER 테이블 및 TITLE 테이블의 각 행을 고유하게 식별하기 위한 시퀀스를 생성합니다.
- MEMBER 테이블의 멤버 번호: 101부터 시작하고 값을 캐시에 저장할 수 없습니다. 시퀀스 이름을 MEMBER_ID_SEQ로 지정하십시오.
 - TITLE 테이블의 제목 번호: 92부터 시작하고 값을 캐시에 저장할 수 없습니다. 시퀀스 이름을 TITLE_ID_SEQ로 지정하십시오.

추가 연습: 사례 연구(계속)

- c. SQL Developer의 Connections Navigator에 시퀀스가 존재하는지 확인합니다.



4. 테이블에 데이터를 추가합니다. 추가할 각 데이터 집합에 대해 스크립트를 작성합니다.
- a. TITLE 테이블에 영화 제목을 추가합니다. 영화 정보를 입력하기 위한 스크립트를 작성합니다. lab_apcs_4a.sql이라는 스크립트에 해당 명령문을 저장합니다. 시퀀스를 사용하여 각 영화 제목을 고유하게 식별합니다. 출시 날짜를 DD-MON-YYYY 형식으로 입력합니다. 문자 필드의 작은 따옴표는 특수하게 처리해야 한다는 점을 기억하십시오. 추가한 내용을 확인합니다.

	TITLE
1	Willie and Christmas Too
2	Alien Again
3	The Glob
4	My Day Off
5	Miracles on Ice
6	Soda Gang

추가 연습: 사례 연구(계속)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

- b. MEMBER 테이블에 데이터를 추가합니다. lab_apcs_4b.sql이라는 스크립트에 insert 문을 저장합니다. 스크립트의 명령을 실행합니다. 시퀀스를 사용하여 멤버 번호를 추가하십시오.

First_Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

추가 연습: 사례 연구(계속)

c. TITLE_COPY 테이블에 다음 영화 복사본을 추가합니다.

주: 이 연습에 사용할 수 있는 TITLE_ID 번호가 있어야 합니다.

Title	Copy_Id	Status	Title	Copy_Id
Willie and Christmas Too	1	AVAILABLE	Willie and Christmas Too	1
Alien Again	1	AVAILABLE	Alien Again	1
	2	RENTED		2
The Glob	1	AVAILABLE	The Glob	1
My Day Off	1	AVAILABLE	My Day Off	1
	2	AVAILABLE		2
	3	RENTED		3
Miracles on Ice	1	AVAILABLE	Miracles on Ice	1
Soda Gang	1	AVAILABLE	Soda Gang	1

d. RENTAL 테이블에 다음 대여 비디오를 추가합니다.

주: 제목 번호는 시퀀스 번호에 따라 다를 수도 있습니다.

Title_Id	Copy_Id	Member_Id	Book_date	Exp_Ret_Date
92	1	101	3 days ago	1 days ago
93	2	101	1 days ago	1 day from now
95	3	102	2 days ago	Today
97	1	106	4 days ago	2 days ago

추가 연습: 사례 연구(계속)

5. 영화 제목, 각 복사본의 대여 가능성 및 대여된 경우 예상 반납 날짜를 표시하는 TITLE_AVAIL이라는 뷰를 생성합니다. 뷰에서 모든 행을 query합니다. 제목에 따라 결과를 정렬합니다.

주: 결과는 다를 수 있습니다.

TITLE	COPY_ID	STATUS	EXP_RET_DATE
1 Alien Again	1	AVAILABLE	(null)
2 Alien Again	2	RENTED	29-JUN-07
3 My Day Off	1	AVAILABLE	(null)
4 My Day Off	2	AVAILABLE	(null)
5 My Day Off	3	RENTED	30-JUN-07
6 The Glob	1	AVAILABLE	(null)
7 Willie and Christmas Too	1	AVAILABLE	29-JUN-07

6. 테이블의 데이터를 변경합니다.

- 새 제목을 추가합니다. 영화는 공상 과학 영화로 분류된 PG 등급의 "Interstellar Wars"입니다. 출시 날짜는 07-JUL-77입니다. 설명은 "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?"입니다. 두 사본에 대해 영화 사본 레코드를 추가해야 합니다.
- 두 건의 예약을 입력합니다. 하나는 "Interstellar Wars"를 대여하려는 Carmen Velasquez 고객이고 다른 하나는 "Soda Gang"을 대여하려는 Mark Quick-to-See 고객입니다.

추가 연습: 사례 연구(계속)

7. 테이블 중 하나를 수정합니다.

- a. D:\labs\sql1\labs 폴더에 있는 lab_apcs_7a.sql 스크립트를 실행하여 TITLE 테이블에 비디오 구입 가격을 기록하는 PRICE 열을 추가합니다. 수정 사항을 확인합니다.

DESCRIBE title		
Name	Null	Type
TITLE_ID	NOT NULL	NUMBER(10)
TITLE	NOT NULL	VARCHAR2(60)
DESCRIPTION	NOT NULL	VARCHAR2(400)
RATING		VARCHAR2(4)
CATEGORY		VARCHAR2(20)
RELEASE_DATE		DATE
PRICE		NUMBER(8,2)

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

- b. 앞의 리스트에 따른 가격으로 각 비디오를 갱신하는 update 문이 포함된 lab_apcs_7b.sql이라는 스크립트를 작성합니다. 스크립트에서 명령을 실행합니다.

주: 이 연습에 사용할 수 있는 TITLE_ID 번호가 있어야 합니다.

추가 연습: 사례 연구(계속)

8. 각 고객의 비디오 대여 내역을 포함하는 보고서를 작성합니다. 고객 이름, 대여한 영화, 대여 날짜 및 대여 기간을 포함해야 합니다. Reporting 기간 동안 모든 고객이 대여한 비디오 수를 합산합니다. lab_apcs_8.sql이라는 스크립트 파일에 보고서를 생성하는 명령을 저장합니다.

주: 결과는 다를 수 있습니다.

MEMBER	TITLE	BOOK_DATE	DURATION
1 Carmen Velasquez	Willie and Christmas Too	20-JUL-07	1
2 Carmen Velasquez	Alien Again	22-JUL-07	(null)
3 LaDoris Ngao	My Day Off	21-JUL-07	(null)
4 Molly Urguhart	Soda Gang	19-JUL-07	2

추가 연습:

해답

추가 연습: 해답

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

1. HR 부서에서 1997년 이후 채용된 모든 clerk에 대한 데이터를 찾으려고 합니다.

```
SELECT *
FROM   employees
WHERE  job_id = 'ST_CLERK'
AND    hire_date > '31-DEC-1997';
```

2. HR 부서에서 커미션을 받는 사원에 대한 보고서를 요구합니다. 해당 사원의 성, 직무, 급여 및 커미션을 표시합니다. 급여의 내림차순으로 데이터를 정렬합니다.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees
WHERE  commission_pct IS NOT NULL
ORDER BY salary DESC;
```

3. HR 부서에서 예산 책정을 위해 예상되는 급여 인상에 대한 보고서를 요구합니다. 이 보고서는 커미션을 받지 않지만 급여가 10% 인상되는 사원을 표시해야 합니다(급여 반올림).

```
SELECT 'The salary of ' || last_name || ' after a 10% raise is '
       || ROUND(salary*1.10) "New salary"
FROM   employees
WHERE  commission_pct IS NULL;
```

4. 사원 및 근속 기간에 대한 보고서를 작성합니다. 모든 사원들의 성 및 근무 기간(년, 개월)을 함께 표시합니다. 근속 기간별로 보고서를 정렬합니다. 근속 기간이 가장 긴 사원이 리스트의 맨 위에 나타나야 합니다.

```
SELECT last_name,
       TRUNC(MONTHS_BETWEEN(SYSDATE, hire_date) / 12) YEARS,
       TRUNC(MOD(MONTHS_BETWEEN(SYSDATE, hire_date), 12)) MONTHS
FROM   employees
ORDER BY years DESC, months desc;
```

5. 성이 "J", "K", "L" 또는 "M"으로 시작하는 사원을 표시합니다.

```
SELECT last_name
FROM   employees
WHERE  SUBSTR(last_name, 1, 1) IN ('J', 'K', 'L', 'M');
```

추가 연습: 해답(계속)

6. 모든 사원을 표시하고 각 사원이 커미션을 받는지 여부를 Yes 또는 No로 나타내는 보고서를 작성합니다. query에서 DECODE 식을 사용합니다.

```
SELECT last_name, salary,
       decode(commission_pct, NULL, 'No', 'Yes') commission
  FROM employees;
```

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수, 조인, 그룹 함수 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

7. 특정 위치에서 근무하는 사원의 부서 이름, 위치 ID, 이름, 직책 및 급여를 표시하는 보고서를 작성합니다. 유저에게 위치를 입력하는 프롬프트를 표시합니다.

- a. 프롬프트가 나타나면 location_id에 1800을 입력합니다.

```
SELECT d.department_name, d.location_id, e.last_name, e.job_id,
e.salary
  FROM employees e, departments d
 WHERE e.department_id = d.department_id
   AND d.location_id = &location_id;
```

8. 성이 "n"으로 끝나는 사원의 수를 알아냅니다. 가능한 두 가지 해결책을 작성합니다.

```
SELECT COUNT(*)
  FROM employees
 WHERE last_name LIKE '%n';
--or
SELECT COUNT(*)
  FROM employees
 WHERE SUBSTR(last_name, -1) = 'n';
```

9. 각 부서에 대한 이름, 위치 및 사원 수를 보여주는 보고서를 작성합니다. 보고서에 사원이 없는 부서도 포함되는지 확인합니다.

```
SELECT d.department_id, d.department_name,
       d.location_id, COUNT(e.employee_id)
  FROM employees e RIGHT OUTER JOIN departments d
    ON e.department_id = d.department_id
 GROUP BY d.department_id, d.department_name, d.location_id;
```

10. HR 부서에서 부서 번호 10 및 20에 있는 직책을 찾으려고 합니다. 해당 부서의 직무 ID를 표시하는 보고서를 작성합니다.

```
SELECT DISTINCT job_id
  FROM employees
 WHERE department_id IN (10, 20);
```

추가 연습: 해답(계속)

11. Administration 및 Executive 부서에서 찾은 직무를 표시하는 보고서를 작성합니다.
또한 해당 직무에 대한 사원 수도 표시합니다. 사원 수가 가장 많은 직무를 가장 먼저 표시합니다.

```
SELECT e.job_id, count(e.job_id) FREQUENCY
FROM employees e JOIN departments d
ON e.department_id = d.department_id
WHERE d.department_name IN ('Administration', 'Executive')
GROUP BY e.job_id
ORDER BY FREQUENCY DESC;
```

다음 연습은 기본 SQL SELECT 문, 기본 SQL Developer 명령, SQL 함수, 조인, 그룹 함수, subquery 등의 항목을 살펴본 후에 추가로 수행할 수 있습니다.

12. 각 월의 16일 이전에 채용된 사원을 모두 표시합니다.

```
SELECT last_name, hire_date
FROM employees
WHERE TO_CHAR(hire_date, 'DD') < 16;
```

13. 모든 사원에 대해 성, 급여 및 \$1000 단위로 표현된 급여를 표시하는 보고서를 작성합니다.

```
SELECT last_name, salary, TRUNC(salary, -3)/1000 Thousands
FROM employees;
```

14. 급여가 \$15,000 이상인 관리자 휘하의 모든 사원을 표시합니다. 사원 이름, 관리자 이름, 관리자 급여 및 관리자의 급여 등급을 표시합니다.

```
SELECT e.last_name, m.last_name manager, m.salary, j.grade_level
FROM employees e JOIN employees m
ON e.manager_id = m.employee_id
JOIN job_grades j
ON m.salary BETWEEN j.lowest_sal AND j.highest_sal
AND m.salary > 15000;
```

추가 연습: 해답(계속)

15. 모든 부서의 부서 번호, 이름, 사원 수, 평균 급여와 각 부서에서 일하는 사원의 이름, 급여 및 직무를 표시합니다.

```
SELECT d.department_id, d.department_name,
       count(e1.employee_id) employees,
       NVL(TO_CHAR(AVG(e1.salary), '99999.99'), 'No average') avg_sal,
       e2.last_name, e2.salary, e2.job_id
  FROM departments d RIGHT OUTER JOIN employees e1
  ON d.department_id = e1.department_id
 RIGHT OUTER JOIN employees e2
  ON d.department_id = e2.department_id
 GROUP BY d.department_id, d.department_name, e2.last_name,
          e2.salary,
          e2.job_id
 ORDER BY d.department_id, employees;
```

16. 평균 급여가 가장 높은 부서의 부서 번호와 최저 급여를 표시하는 보고서를 작성합니다.

```
SELECT department_id, MIN(salary)
  FROM employees
 GROUP BY department_id
 HAVING AVG(salary) = (SELECT MAX(AVG(salary))
                           FROM employees
                           GROUP BY department_id);
```

17. 영업 사원이 근무하지 않는 부서를 표시하는 보고서를 작성합니다. 출력에 부서 번호, 부서 이름 및 위치를 포함합니다.

```
SELECT *
  FROM departments
 WHERE department_id NOT IN(SELECT department_id
                               FROM employees
                               WHERE job_id = 'SA_REP'
                                 AND department_id IS NOT NULL);
```

추가 연습: 해답(계속)

18. HR 부서를 위해 통계 보고서를 작성합니다. 이 보고서에는 다음 조건의 부서에 대한 부서 번호, 부서 이름 및 근무하는 사원 수를 표시합니다.

a. 사원 수가 3명 미만인 부서:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM departments d JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) < 3;
```

b. 사원 수가 가장 많은 부서:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM departments d JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                     FROM employees
                     GROUP BY department_id);
```

c. 사원 수가 가장 적은 부서:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM departments d JOIN employees e
ON d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) = (SELECT MIN(COUNT(*))
                     FROM employees
                     GROUP BY department_id);
```

19. 모든 사원에 대해 사원 번호, 성, 급여, 부서 번호 및 해당 부서의 평균 급여를 표시하는 보고서를 작성합니다.

```
SELECT e.employee_id, e.last_name, e.department_id, e.salary,
AVG(s.salary)
FROM employees e JOIN employees s
ON e.department_id = s.department_id
GROUP BY e.employee_id, e.last_name, e.department_id, e.salary;
```

추가 연습: 해답(계속)

20. 사원 채용 수가 가장 많은 요일에 채용된 사원을 모두 표시합니다.

```
SELECT last_name, TO_CHAR(hire_date, 'DAY') day
FROM   employees
WHERE  TO_CHAR(hire_date, 'Day') =
       (SELECT TO_CHAR(hire_date, 'Day')
        FROM   employees
        GROUP BY TO_CHAR(hire_date, 'Day')
        HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                            FROM   employees
                            GROUP BY TO_CHAR(hire_date, 'Day')));
```

21. 사원의 채용일에 준하여 기념일 개요를 생성합니다. 기념일을 오름차순으로 정렬하십시오.

```
SELECT last_name, TO_CHAR(hire_date, 'Month DD') BIRTHDAY
FROM   employees
ORDER BY TO_CHAR(hire_date, 'DDD');
```

추가 연습: 사례 연구 해답

1. 다음 테이블 instance 차트를 기반으로 테이블을 생성합니다. 적절한 데이터 유형을 선택하고 무결성 제약 조건을 추가합니다.

- a. 테이블 이름: MEMBER

```
CREATE TABLE member
  (member_id      NUMBER(10)
   CONSTRAINT member_member_id_pk PRIMARY KEY,
   last_name      VARCHAR2(25)
   CONSTRAINT member_last_name_nn NOT NULL,
   first_name     VARCHAR2(25),
   address        VARCHAR2(100),
   city           VARCHAR2(30),
   phone          VARCHAR2(15),
   join_date      DATE DEFAULT SYSDATE
   CONSTRAINT member_join_date_nn NOT NULL);
```

- b. 테이블 이름: TITLE

```
CREATE TABLE title
  (title_id      NUMBER(10)
   CONSTRAINT title_title_id_pk PRIMARY KEY,
   title          VARCHAR2(60)
   CONSTRAINT title_title_nn NOT NULL,
   description    VARCHAR2(400)
   CONSTRAINT title_description_nn NOT NULL,
   rating         VARCHAR2(4)
   CONSTRAINT title_rating_ck CHECK
     (rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
   category       VARCHAR2(20)
   CONSTRAINT title_category_ck CHECK
     (category IN ('DRAMA', 'COMEDY', 'ACTION',
                   'CHILD', 'SCIFI', 'DOCUMENTARY')),
   release_date   DATE);
```

- c. 테이블 이름: TITLE_COPY

```
CREATE TABLE title_copy
  (copy_id        NUMBER(10),
   title_id       NUMBER(10)
   CONSTRAINT title_copy_title_if_fk REFERENCES
   title(title_id),
   status          VARCHAR2(15)
   CONSTRAINT title_copy_status_nn NOT NULL
   CONSTRAINT title_copy_status_ck CHECK (status IN
     ('AVAILABLE', 'DESTROYED', 'RENTED', 'RESERVED')),
   CONSTRAINT title_copy_copy_id_title_id_pk
   PRIMARY KEY (copy_id, title_id));
```

추가 연습: 사례 연구 해답(계속)

d. 테이블 이름: RENTAL

```
CREATE TABLE rental
  (book_date      DATE DEFAULT SYSDATE,
   member_id      NUMBER(10)
    CONSTRAINT rental_member_id_fk REFERENCES
member(member_id),
   copy_id        NUMBER(10),
   act_ret_date   DATE,
   exp_ret_date  DATE DEFAULT SYSDATE + 2,
   title_id      NUMBER(10),
    CONSTRAINT rental_book_date_copy_title_pk
      PRIMARY KEY (book_date, member_id, copy_id, title_id),
    CONSTRAINT rental_copy_id_title_id_fk
      FOREIGN KEY (copy_id, title_id)
      REFERENCES title_copy(copy_id, title_id));
```

e. 테이블 이름: RESERVATION

```
CREATE TABLE reservation
  (res_date       DATE,
   member_id      NUMBER(10)
    CONSTRAINT reservation_member_id REFERENCES
member(member_id),
   title_id       NUMBER(10)
    CONSTRAINT reservation_title_id REFERENCES
title(title_id),
    CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
  (res_date, member_id, title_id));
```

2. SQL Developer의 Connections Navigator에서 검사를 진행하여 해당 테이블이 제대로 생성되었는지 확인합니다.
 - a. Connections Navigator에서 Connections > myconnection > Tables를 확장합니다.
3. MEMBER 테이블 및 TITLE 테이블의 각 행을 고유하게 식별하기 위한 시퀀스를 생성합니다.
 - a. MEMBER 테이블의 멤버 번호: 101부터 시작하고 값을 캐시에 저장할 수 없습니다. 시퀀스 이름을 MEMBER_ID_SEQ로 지정하십시오.

```
CREATE SEQUENCE member_id_seq
START WITH 101
NOCACHE;
```

- b. TITLE 테이블의 제목 번호: 92부터 시작하고 값을 캐시에 저장할 수 없습니다. 시퀀스 이름을 TITLE_ID_SEQ로 지정하십시오.

```
CREATE SEQUENCE title_id_seq
START WITH 92
NOCACHE;
```

추가 연습: 사례 연구 해답(계속)

- c. SQL Developer의 Connections Navigator에 시퀀스가 존재하는지 확인합니다.
- Connections Navigator에서 myconnection 노드가 확장되어 있으면 Sequences를 확장합니다.
4. 테이블에 데이터를 추가합니다. 추가할 각 데이터 집합에 대해 스크립트를 작성합니다.
- TITLE 테이블에 영화 제목을 추가합니다. 영화 정보를 입력하기 위한 스크립트를 작성합니다. lab_apcs_4a.sql이라는 스크립트에 해당 명령문을 저장합니다. 시퀀스를 사용하여 각 영화 제목을 고유하게 식별합니다. 출시 날짜를 DD-MON-YYYY 형식으로 입력합니다. 문자 필드의 작은 따옴표는 특수하게 처리해야 한다는 점을 기억하십시오. 추가한 내용을 확인합니다.

```

INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES  (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
          'All of Willie''s friends make a Christmas list for
          Santa, but Willie has yet to add his own wish list.',
          'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id , title, description, rating,
                  category, release_date)
VALUES  (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
          installment of science fiction history. Can the
          heroine save the planet from the alien life form?',
          'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES  (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
          near a small American town and unleashes carnivorous
          goo in this classic.', 'NR', 'SCIFI',
          TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES  (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
          luck and a lot ingenuity, a teenager skips school for
          a day in New York.', 'PG', 'COMEDY',
          TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
/

```

```

INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES      (title_id_seq.NEXTVAL, 'Miracles on Ice', 'A six-year-
old has      doubts about Santa Claus, but she discovers that
miracles really do exist.', 'PG', 'DRAMA',
                  TO_DATE('12-SEP-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES      (title_id_seq.NEXTVAL, 'Soda Gang', 'After discovering
a cache of drugs, a young couple find themselves pitted against a
vicious gang.', 'NR', 'ACTION', TO_DATE('01-JUN-1995','DD-MON-
YYYY'))
/
...
COMMIT
/
SELECT  title
FROM    title;

```

- b. MEMBER 테이블에 데이터를 추가합니다. lab_apcs_4b.sql이라는 스크립트에 insert 문을 삽입합니다. 스크립트의 명령을 실행합니다. 시퀀스를 사용하여 멤버 번호를 추가하십시오.

```

SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Carmen', 'Velasquez',
        '283 King Street', 'Seattle', '206-899-6666',
        TO_DATE('08-MAR-1990',
                'DD-MM-YYYY'))
/
INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'LaDoris', 'Ngao',
        '5 Modrany', 'Bratislava', '586-355-8882', TO_DATE('08-
MARCH-1990',
               'DD-MM-YYYY'))
/
INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Midori', 'Nagayama',
        '68 Via Centrale', 'Sao Paolo', '254-852-5764',
        TO_DATE('17-JUN-1991',
                'DD-MM-YYYY'))
/
INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Mark', 'Quick-to-See',
        '6921 King Way', 'Lagos', '63-559-7777', TO_DATE('07-APR-
1990',
               'DD-MM-YYYY'))
/

```

```

INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Audry', 'Ropeburn',
        '86 Chu Street', 'Hong Kong', '41-559-87', TO_DATE
('18-JAN-1991',
     'DD-MM-YYYY'))/
INSERT INTO member(member_id, first_name, last_name,
                   address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Molly', 'Urguhart',
        '3035 Laurier', 'Quebec', '418-542-9988', TO_DATE('18-
JAN-1991',
     'DD-MM-YYYY'))/
COMMIT
SET VERIFY ON

```

c. TITLE_COPY 테이블에 다음 영화 복사본을 추가합니다.

주: 이 연습에 사용할 수 있는 TITLE_ID 번호가 있어야 합니다.

```

INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 92, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 93, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 93, 'RENTED')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 94, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 95, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 95, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (3, 95, 'RENTED')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 96, 'AVAILABLE')/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 97, 'AVAILABLE')/

```

추가 연습: 사례 연구 해답(계속)

d. RENTAL 테이블에 다음 대여 비디오를 추가합니다.

주: 제목 번호는 시퀀스 번호에 따라 다를 수도 있습니다.

```
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (95, 3, 102, sysdate-2, sysdate, NULL)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2)
/
COMMIT
/
```

5. 영화 제목, 각 복사본의 대여 가능성 및 대여된 경우 예상 반납 날짜를 표시하는 TITLE_AVAIL이라는 뷰를 생성합니다. 뷰에서 모든 행을 query합니다. 제목에 따라 결과를 정렬합니다.

주: 결과는 다를 수 있습니다.

```
CREATE VIEW title_avail AS
  SELECT t.title, c.copy_id, c.status, r.exp_ret_date
  FROM title t JOIN title_copy c
  ON t.title_id = c.title_id
  FULL OUTER JOIN rental r
  ON c.copy_id = r.copy_id
  AND c.title_id = r.title_id;

SELECT *
FROM title_avail
ORDER BY title, copy_id;
```

추가 연습: 사례 연구 해답(계속)

6. 테이블의 데이터를 변경합니다.

- a. 새 제목을 추가합니다. 영화는 공상 과학 영화로 분류된 PG 등급의 "Interstellar Wars"입니다. 출시 날짜는 07-JUL-77입니다. 설명은 "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?"입니다. 두 사본에 대해 영화 사본 레코드를 추가해야 합니다.

```
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
        'Futuristic interstellar action movie. Can the
        rebels save the humans from the evil empire?',
        'PG', 'SCIFI', '07-JUL-77')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE')
```

- b. 두 건의 예약을 입력합니다. 하나는 "Interstellar Wars"를 대여하려는 Carmen Velasquez 고객이고 다른 하나는 "Soda Gang"을 대여하려는 Mark Quick-to-See 고객입니다.

```
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 101, 98)
/
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 104, 97)
```

7. 테이블 중 하나를 수정합니다.

- a. D:\labs\sql1\labs 폴더에 있는 lab_apcs_7a.sql 스크립트를 실행하여 TITLE 테이블에 비디오 구입 가격을 기록하는 PRICE 열을 추가합니다. 수정 사항을 확인합니다.

```
ALTER TABLE title
ADD (price NUMBER(8,2));

DESCRIBE title
```

추가 연습: 사례 연구 해답(계속)

- b. 제공된 리스트에 따른 가격으로 각 비디오를 개신하는 update 문이 포함된 lab_apcs_7b.sql이라는 스크립트를 작성합니다. 스크립트에서 명령을 실행합니다.

주: 이 연습에 사용할 수 있는 TITLE_ID 번호가 있어야 합니다.

```
SET ECHO OFF
SET VERIFY OFF
UPDATE title
SET price = &price
WHERE title_id = &title_id;
SET VERIFY OFF
SET ECHO OFF
```

8. 각 고객의 비디오 대여 내역을 포함하는 보고서를 작성합니다. 고객 이름, 대여한 영화, 대여 날짜 및 대여 기간을 포함해야 합니다. Reporting 기간 동안 모든 고객이 대여한 비디오 수를 합산합니다. lab_apcs_8.sql이라는 스크립트 파일에 보고서를 생성하는 명령을 저장합니다.

주: 결과는 다를 수 있습니다.

```
SELECT m.first_name || ' ' || m.last_name MEMBER, t.title,
       r.book_date, r.act_ret_date - r.book_date DURATION
FROM   member m
JOIN   rental r
ON     r.member_id = m.member_id
JOIN   title t
ON     r.title_id = t.title_id
ORDER BY member;
```

1 유저 액세스 제어

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 시스템 권한과 객체 권한의 구분
- 테이블에 권한 부여
- 룰 부여
- 권한과 룰 구별

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 특정 객체에 대한 데이터베이스 액세스를 제어하고 서로 다른 레벨의 액세스 권한을 가진 새로운 유저를 추가하는 방법에 대해 배웁니다.

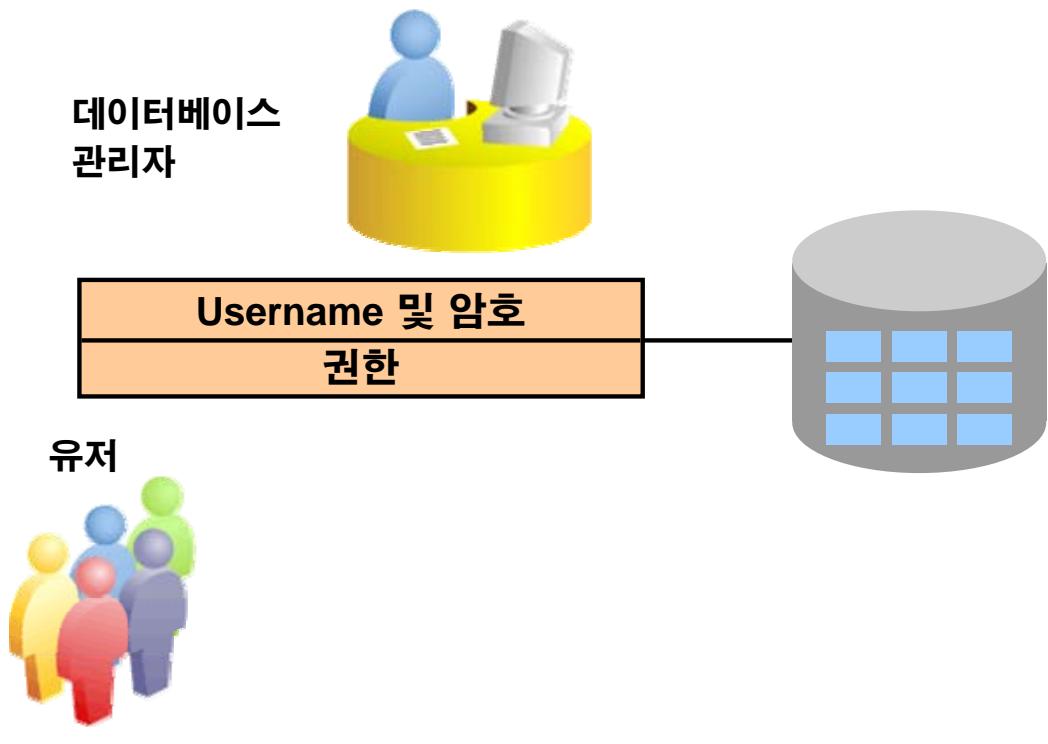
단원 내용

- 시스템 권한
- 룰 생성
- 객체 권한
- 객체 권한 취소

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

유저 액세스 제어



Copyright © 2009, Oracle. All rights reserved.

유저 액세스 제어

다중 유저 환경에서는 데이터베이스 액세스 및 사용과 관련한 보안을 유지해야 합니다.

Oracle 서버에서는 다음과 같은 데이터베이스 보안 기능을 사용할 수 있습니다.

- 데이터베이스 액세스 제어
- 데이터베이스의 특정 객체에 대한 액세스 권한 부여
- Oracle 데이터 딕셔너리를 통한 권한 부여 및 수신 확인

데이터베이스 보안은 시스템 보안과 데이터 보안이라는 두 가지 범주로 분류할 수 있습니다.

시스템 보안에는 username과 암호, 유저에게 할당된 디스크 공간, 유저가 수행할 수 있는 시스템 작업 등과 같은 시스템 레벨의 데이터베이스 액세스 및 사용과 관련된 사항이 포함됩니다.

데이터베이스 보안에는 데이터베이스 객체의 액세스 및 사용, 객체에 대해 해당 유저가 수행할 수 있는 작업과 관련된 사항이 포함됩니다.

권한

- **데이터베이스 보안:**
 - 시스템 보안
 - 데이터 보안
- **시스템 권한:** 데이터베이스 내에서 특정 작업 수행
- **객체 권한:** 데이터베이스 객체의 내용 조작
- **스키마:** 테이블, 뷰 및 시퀀스와 같은 객체들의 모음

ORACLE

Copyright © 2009, Oracle. All rights reserved.

권한

권한이란 특정 SQL 문을 실행할 수 있는 권리입니다. DBA(데이터베이스 관리자)는 유저를 생성하고 유저에게 데이터베이스 및 해당 객체에 대한 액세스 권한을 부여할 수 있는 고급 레벨의 유저입니다. 데이터베이스에 액세스하려면 시스템 권한이 필요하며, 데이터베이스의 객체 내용을 조작하려면 객체 권한이 필요합니다. 또한 다른 유저나 롤(관련 권한들로 이루어진 명명된 그룹)에 추가 권한을 부여할 수 있는 권한을 부여받을 수도 있습니다.

스키마

스키마는 테이블, 뷰 및 시퀀스와 같은 객체들의 모음입니다. 스키마의 소유자는 데이터베이스 유저이며 해당 유저와 이름이 동일합니다.

시스템 권한은 특정 작업을 수행하거나 특정 유형의 스키마 객체에 대한 작업을 수행할 수 있는 권리입니다. 객체 권한은 특정 스키마 객체에 대해 특정 작업을 수행할 수 있는 권한을 유저에게 제공합니다.

자세한 내용은 *Oracle Database 2 Day DBA 11g Release 2 (11.2)* 참조 설명서를 참조하십시오.

시스템 권한

- 100개 이상의 권한을 사용할 수 있습니다.
- 데이터베이스 관리자는 다음과 같은 작업을 위해 높은 레벨의 시스템 권한을 가집니다.
 - 새 유저 생성
 - 유저 제거
 - 테이블 제거
 - 테이블 백업

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시스템 권한

100개 이상의 특정 시스템 권한을 유저와 룰에 대해 지정할 수 있습니다. 시스템 권한은 일반적으로 DBA (데이터베이스 관리자)에 의해 제공됩니다.

일반적인 DBA 권한

시스템 권한	인증된 작업
CREATE USER	피부여자가 다른 Oracle 유저를 생성할 수 있습니다.
DROP USER	피부여자가 다른 유저를 삭제할 수 있습니다.
DROP ANY TABLE	피부여자가 임의 스키마에서 테이블을 삭제할 수 있습니다.
BACKUP ANY TABLE	피부여자가 Export 유틸리티를 사용하여 임의 스키마에서 테이블을 백업할 수 있습니다.
SELECT ANY TABLE	피부여자가 임의 스키마에서 테이블, 뷰 또는 Materialized View를 query 할 수 있습니다.
CREATE ANY TABLE	피부여자가 임의 스키마에 테이블을 생성할 수 있습니다.

유저 생성

DBA는 CREATE USER 문을 사용하여 유저를 생성합니다.

```
CREATE USER user
IDENTIFIED BY password;
```

```
CREATE USER demo
IDENTIFIED BY demo;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

유저 생성

DBA는 CREATE USER 문을 실행하여 유저를 생성합니다. 이때 유저는 아무 권한도 없는 상태입니다. 그런 다음 DBA는 이러한 유저에게 권한을 부여할 수 있습니다. 권한에 따라 유저가 데이터베이스 레벨에서 수행할 수 있는 작업이 결정됩니다.

슬라이드는 유저를 생성하는 요약 구문을 보여줍니다.

이 구문에서 다음이 적용됩니다.

user 생성할 유저의 이름입니다.

password 유저가 로그인할 때 사용할 암호를 지정합니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

참고: Oracle Database 11g부터는 암호의 대소문자를 구분합니다.

유저 시스템 권한

- 유저를 생성한 후 DBA는 해당 유저에게 특정 시스템 권한을 부여할 수 있습니다.

```
GRANT privilege [, privilege...]  
TO user [, user/ role, PUBLIC...];
```

- 예를 들어 응용 프로그램 개발자는 다음과 같은 시스템 권한을 가질 수 있습니다.
 - CREATE SESSION
 - CREATE TABLE
 - CREATE SEQUENCE
 - CREATE VIEW
 - CREATE PROCEDURE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

일반적인 유저 권한

유저를 생성한 후 DBA는 해당 유저에게 권한을 할당할 수 있습니다.

시스템 권한	인증된 작업
CREATE SESSION	데이터베이스에 연결합니다.
CREATE TABLE	유저의 스키마에 테이블을 생성합니다.
CREATE SEQUENCE	유저의 스키마에 시퀀스를 생성합니다.
CREATE VIEW	유저의 스키마에 뷰를 생성합니다.
CREATE PROCEDURE	유저의 스키마에 내장 프로시저, 함수 또는 패키지를 생성합니다.

이 구문에서 다음이 적용됩니다.

privilege 부여할 시스템 권한입니다.

user /role/PUBLIC 유저 이름, 롤 이름 또는 PUBLIC(모든 유저가 권한을 부여받도록 지정함)입니다.

참고: 현재의 시스템 권한은 SESSION_PRIVS 딕셔너리 뷰에서 찾을 수 있습니다. 데이터 딕셔너리는 Oracle 서버에서 생성하여 유지 관리하는 테이블과 뷰의 모음으로, 데이터베이스에 대한 정보를 포함하고 있습니다.

시스템 권한 부여

DBA는 유저에게 특정 시스템 권한을 부여할 수 있습니다.

```
GRANT create session, create table,  
       create sequence, create view  
TO      demo;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시스템 권한 부여

DBA는 GRANT 문을 사용하여 유저에게 시스템 권한을 할당합니다. 유저는 권한을 부여받은 후 해당 권한을 즉시 사용할 수 있습니다.

위 슬라이드 예제에서는 유저 `demo`에게 세션, 테이블, 시퀀스 및 뷰를 생성할 수 있는 권한이 할당되었습니다.

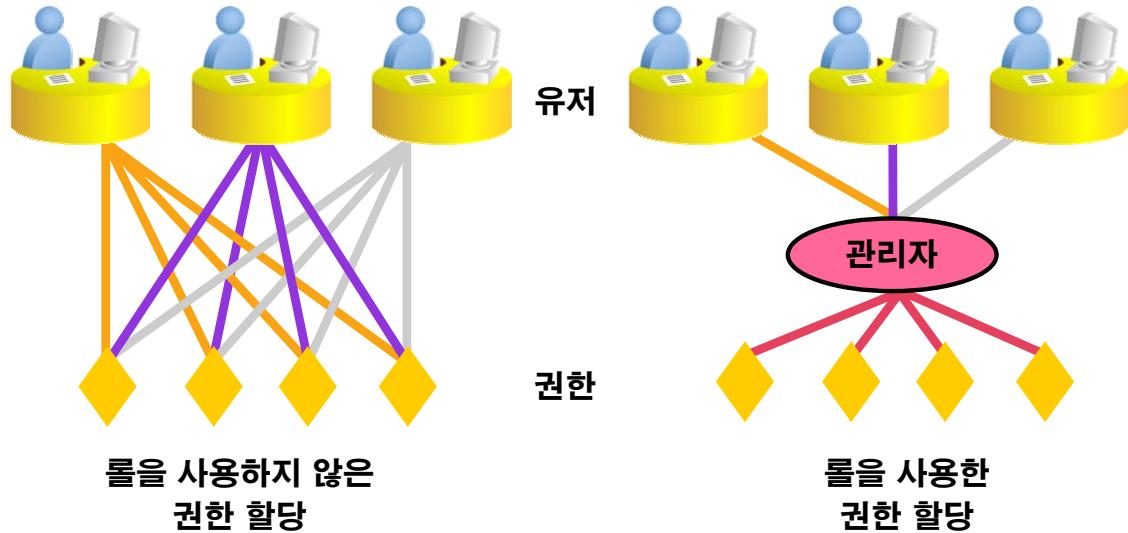
단원 내용

- 시스템 권한
- 룰 생성
- 객체 권한
- 객체 권한 취소

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

롤이란?



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

롤이란?

롤은 유저에게 부여할 수 있는 관련 권한들로 이루어진 명명된 그룹입니다. 룰을 사용하면 권한을 쉽게 취소하거나 유지 관리할 수 있습니다.

한 유저에게 여러 룰에 대한 액세스 권한을 부여하거나 동일 룰에 여러 유저를 할당할 수도 있습니다. 일반적으로 룰은 데이터베이스 응용 프로그램에 대해 생성됩니다.

룰 생성 및 할당

먼저 DBA가 룰을 생성해야 합니다. 그런 다음 DBA는 이 룰에 권한을 할당하고 유저에게 룰을 할당할 수 있습니다.

구문

```
CREATE ROLE role;
```

이 구문에서 다음이 적용됩니다.

role 생성할 룰의 이름입니다.

룰을 생성한 후 DBA는 GRANT 문을 사용하여 해당 룰에 권한을 할당하는 것은 물론 룰에 유저를 할당할 수도 있습니다. 룰은 스키마 객체가 아니므로 아무 유저나 룰에 권한을 추가할 수 있습니다.

롤 생성 및 룰에 권한 부여

- **룰 생성:**

```
CREATE ROLE manager;
```

- **룰에 권한 부여:**

```
GRANT create table, create view  
TO manager;
```

- **유저에게 룰 부여:**

```
GRANT manager TO alice;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

룰 생성

위 슬라이드 예제에서는 `manager` 룰을 생성한 다음 관리자가 테이블과 뷰를 생성할 수 있도록 설정되었습니다. 그런 다음 유저 `alice`에게 관리자 룰을 부여합니다. 그 결과 `alice`는 테이블과 뷰를 생성할 수 있습니다.

유저에게 여러 룰이 부여된 경우 해당 룰과 관련된 모든 권한을 갖게 됩니다.

암호 변경

- DBA는 유저 계정을 생성하고 암호를 초기화합니다.
 - ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.

```
ALTER USER demo  
IDENTIFIED BY employ;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

암호 변경

DBA는 모든 유저의 계정을 생성하고 암호를 초기화합니다. ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.

위 슬라이드 예제에서는 유저 demo가 ALTER USER 문을 사용하여 암호를 변경하는 것을 보여줍니다.

구문

```
ALTER USER user IDENTIFIED BY password;
```

이 구문에서 다음이 적용됩니다.

user 유저의 이름입니다.

password 새 암호를 지정합니다.

이 명령문으로 암호를 변경할 수 있지만 그 외에도 여러 옵션이 있습니다. 다른 옵션을 변경하려면 `ALTER USER` 권한이 있어야 합니다.

자세한 내용은 *Oracle Database 11g SQL Reference* 설명서를 참조하십시오.

참고: SQL*Plus에는 유저가 로그인할 때 유저의 암호를 변경하는 데 사용할 수 있는 PASSWORD 명령(PASSW)이 있습니다. 이 명령은 SQL Developer에서는 사용할 수 없습니다.

단원 내용

- 시스템 권한
- 룰 생성
- 객체 권한
- 객체 권한 취소

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

객체 권한

객체 권한	테이블	뷰	시퀀스
ALTER	✓		✓
DELETE	✓	✓	
INDEX	✓		
INSERT	✓	✓	
REFERENCES	✓		
SELECT	✓	✓	✓
UPDATE	✓	✓	

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

객체 권한

객체 권한은 특정 테이블, 뷰, 시퀀스 및 프로시저에 대해 특정 작업을 수행할 수 있는 권한입니다. 각 객체에는 부여할 수 있는 일련의 권한이 있습니다. 위 슬라이드의 표에는 다양한 객체에 대한 권한이 나열되어 있습니다. 시퀀스에는 SELECT 및 ALTER 권한만 적용됩니다. UPDATE, REFERENCES 및 INSERT는 간접 가능한 열의 부분 집합을 지정하여 제한할 수 있습니다. SELECT 권한은 열의 부분 집합으로 뷰를 생성하고 해당 뷰에만 SELECT 권한을 부여하여 제한할 수 있습니다. 동의어에 부여된 권한은 해당 동의어가 참조하는 기본 테이블에 대한 권한으로 변환됩니다.

참고: REFERENCES 권한을 사용하면 다른 유저가 자신의 테이블을 참조하는 FOREIGN KEY 제약 조건을 생성할 수 있습니다.

객체 권한

- 객체 권한은 객체마다 다양합니다.
- 객체 소유자는 해당 객체에 대한 모든 권한을 소유합니다.
- 소유자는 자신의 객체에 대한 특정 권한을 부여할 수 있습니다.

```
GRANT      object_priv [(columns)]
ON        object
TO        {user|role|PUBLIC}
[WITH GRANT OPTION];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

객체 권한 부여

스키마 객체의 유형에 따라 다양한 객체 권한을 사용할 수 있습니다. 유저는 자신의 스키마에 포함된 스키마 객체에 대해 자동으로 모든 객체 권한을 가집니다. 또한 유저는 자신이 소유하는 스키마 객체에 대한 객체 권한을 다른 유저 또는 룰에 부여할 수 있습니다. 권한을 부여할 때 WITH GRANT OPTION을 사용하면 권한 피부여자가 다른 유저에게 객체 권한을 부여할 수 있습니다. 이 옵션을 사용하지 않으면 권한 피부여자가 해당 권한을 사용할 수는 있지만 다른 유저에게 부여할 수 없습니다.

이 구문에서 다음이 적용됩니다.

<i>object_priv</i>	부여할 객체 권한입니다.
ALL	모든 객체 권한을 지정합니다.
<i>columns</i>	권한이 부여되는 테이블 또는 뷰의 열을 지정합니다.
ON <i>object</i>	권한이 부여되는 객체입니다.
TO	권한이 부여되는 대상을 식별합니다.
PUBLIC	모든 유저에게 객체 권한을 부여합니다.
WITH GRANT OPTION	권한을 부여받은 사람이 다른 유저 및 룰에 객체 권한을 부여할 수 있습니다.

참고: 이 구문에서 *schema*는 소유자의 이름과 동일합니다.

객체 권한 부여

- EMPLOYEES 테이블에 대한 query 권한을 부여합니다.

```
GRANT select
ON employees
TO demo;
```

- 특정 열을 갱신할 수 있는 권한을 유저와 룰에 부여합니다.

```
GRANT update (department_name, location_id)
ON departments
TO demo, manager;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

지침

- 객체에 권한을 부여하려면 해당 객체가 자신의 스키마에 있거나 WITH GRANT OPTION 객체 권한을 부여받아야 합니다.
- 객체 소유자는 객체에 대한 객체 권한을 데이터베이스의 다른 유저나 룰에 부여할 수 있습니다.
- 해당 객체에 대한 모든 객체 권한을 자동으로 획득합니다.

슬라이드의 첫번째 예제에서는 EMPLOYEES 테이블 query 권한을 유저 demo에게 부여합니다. 두번째 예제에서는 DEPARTMENTS 테이블의 특정 열에 대한 UPDATE 권한을 demo와 manager 룰에 부여합니다.

예를 들어, 스키마가 oraxx이고 유저 demo가 SELECT 문을 사용하여 EMPLOYEES 테이블에서 데이터를 가져오려는 경우에 사용해야 하는 구문은 다음과 같습니다.

```
SELECT * FROM oraxx.employees;
```

또는 유저 demo가 테이블에 대해 동의어를 생성하고 동의어에서 SELECT 문을 실행할 수 있습니다.

```
CREATE SYNONYM emp FOR oraxx.employees;
SELECT * FROM emp;
```

참고: 일반적으로 DBA는 시스템 권한을 할당하며 객체 소유자는 객체 권한을 부여할 수 있습니다.

권한 전달

- 유저에게 권한을 전달할 수 있는 자격을 부여합니다.

```
GRANT select, insert
ON   departments
TO   demo
WITH GRANT OPTION;
```

- 시스템의 모든 유저가 Alice의 DEPARTMENTS 테이블에서 데이터를 query할 수 있도록 허용합니다.

```
GRANT select
ON   alice.departments
TO   PUBLIC;
```



Copyright © 2009, Oracle. All rights reserved.

권한 전달

WITH GRANT OPTION 키워드

WITH GRANT OPTION 절을 사용하여 부여된 권한은 권한을 부여받은 사람에 의해 다른 유저와 롤에 전달될 수 있습니다. 권한 부여자의 권한이 취소되면 WITH GRANT OPTION 절을 사용하여 부여된 객체 권한도 취소됩니다.

위 슬라이드 예제에서 유저 demo는 테이블을 query하고 테이블에 행을 추가할 수 있는 권한으로 DEPARTMENTS 테이블에 액세스할 수 있습니다. 또한 user1이 이러한 권한을 다른 유저에게 부여할 수 있음을 보여줍니다.

PUBLIC 키워드

테이블 소유자는 PUBLIC 키워드를 사용하여 모든 유저에게 액세스 권한을 부여할 수 있습니다. 두번째 예제에서는 시스템의 모든 유저가 Alice의 DEPARTMENTS 테이블에서 데이터를 query할 수 있도록 허용합니다.

부여된 권한 확인

데이터 딕셔너리 뷰	설명
ROLE_SYS_PRIVS	롤에 부여되는 시스템 권한
ROLE_TAB_PRIVS	롤에 부여되는 테이블 권한
USER_ROLE_PRIVS	유저가 액세스할 수 있는 롤
USER_SYS_PRIVS	유저에게 부여되는 시스템 권한
USER_TAB_PRIVS_MADE	유저의 객체에 대해 부여되는 객체 권한
USER_TAB_PRIVS_REC'D	유저에게 부여되는 객체 권한
USER_COL_PRIVS_MADE	유저 객체의 열에 대해 부여되는 객체 권한
USER_COL_PRIVS_REC'D	특정 열에 대해 유저에게 부여되는 객체 권한

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

부여된 권한 확인

Oracle 서버에서는 DELETE 권한이 없는 테이블에서 행을 삭제하는 것처럼 승인되지 않은 작업은 수행할 수 없습니다.

다음 중 하나를 수행한 경우 "Table or view does not exist"라는 Oracle 서버 오류 메시지가 나타납니다.

- 존재하지 않는 테이블 또는 뷰의 이름을 지정했습니다.
- 적절한 권한이 없는 테이블 또는 뷰에 대해 작업을 수행하려 했습니다.

데이터 딕셔너리는 테이블과 뷰로 구성되며 데이터베이스에 대한 정보를 포함합니다. 자신이 소유한 권한은 데이터 딕셔너리에서 확인할 수 있습니다. 위 슬라이드의 표에는 다양한 데이터 딕셔너리 테이블에 대한 설명이 있습니다.

데이터 딕셔너리 뷰에 대한 자세한 내용은 "데이터 딕셔너리 뷰를 사용하여 객체 관리" 단원을 참조하십시오.

참고: ALL_TAB_PRIVS_MADE 딕셔너리 뷰는 유저가 부여한 모든 객체 권한 또는 유저가 소유한 객체에 대한 모든 객체 권한에 대해 설명합니다.

단원 내용

- 시스템 권한
- 룰 생성
- 객체 권한
- 객체 권한 취소

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

객체 권한 취소

- REVOKE 문을 사용하여 다른 유저에게 부여된 권한을 취소할 수 있습니다.
- WITH GRANT OPTION 절을 통해 다른 유저에게 부여된 권한을 취소할 수도 있습니다.

```
REVOKE {privilege [, privilege...]|ALL}
ON     object
FROM   {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

객체 권한 취소

REVOKE 문을 사용하여 다른 유저에게 부여된 권한을 제거할 수 있습니다. REVOKE 문을 사용하면 자신이 직접 이름을 지정한 유저와 권한이 취소된 유저에게 권한을 부여받은 다른 유저로부터 지정한 권한을 취소할 수 있습니다.

이 구문에서 다음이 적용됩니다.

CASCADE

REFERENCES 권한을 통해 CONSTRAINTS 객체에 대해 지정된 참조 무결성 제약 조건을 제거하는 데 필요합니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

참고: 퇴사한 유저의 권한을 취소한 경우 이 유저가 다른 유저에게 부여한 모든 권한을 다시 부여해야 합니다. 유저 계정에서 권한을 취소하지 않고 유저 계정을 삭제하면 해당 유저가 다른 유저에게 부여한 시스템 권한이 이 작업의 영향을 받지 않습니다.

객체 권한 취소

DEPARTMENTS 테이블에 대해 유저 demo에게 부여된 SELECT 및 INSERT 권한을 취소합니다.

```
REVOKE select, insert  
ON departments  
FROM demo;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

객체 권한 취소(계속)

위 슬라이드 예제에서는 DEPARTMENTS 테이블에서 유저 demo에게 부여된 SELECT 및 INSERT 권한을 취소합니다.

참고: WITH GRANT OPTION 절을 사용하여 권한을 부여받은 유저는 다시 WITH GRANT OPTION 절을 사용하여 해당 권한을 다른 유저에게 부여할 수 있습니다. 따라서 권한을 부여받은 사람의 연결 관계가 길게 이어질 수도 있지만, 순환적인 권한 부여 (조상 권한 부여자에게 권한 부여) 관계는 허용되지 않습니다. 소유자가 다른 유저에게 권한을 부여한 유저로부터 권한을 취소하면 부여된 모든 권한이 연쇄적으로 취소됩니다.

예를 들어, 유저 A가 특정 테이블에 대해 WITH GRANT OPTION 절을 포함한 SELECT 권한을 유저 B에게 부여하면 유저 B는 WITH GRANT OPTION 절을 사용하여 SELECT 권한을 유저 C에게 부여할 수 있고 유저 C는 유저 D에게 SELECT 권한을 부여할 수 있습니다. 유저 A가 유저 B에게서 권한을 취소하면 유저 C 및 D에게 부여된 권한도 취소됩니다.

퀴즈

다음 설명 중 옳은 것은 무엇입니까?

- 1.** 유저는 객체를 생성한 후에 GRANT 문을 사용하여 사용 가능한 객체 권한을 다른 유저에게 전달할 수 있습니다.
- 2.** 유저는 CREATE ROLE 문으로 룰을 생성하여 다른 유저에게 시스템 또는 객체 권한 모음을 전달할 수 있습니다.
- 3.** 유저는 자신의 암호를 변경할 수 있습니다.
- 4.** 유저는 자신과 자신의 객체에 부여된 권한을 볼 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3, 4

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 시스템 권한과 객체 권한의 구분
- 테이블에 권한 부여
- 룰 부여
- 권한과 룰 구별

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

DBA는 유저에게 권한을 할당하여 유저에 대한 초기 데이터베이스 보안을 설정합니다.

- DBA는 유저를 생성하고 암호를 지정합니다. DBA는 또한 유저에 대한 초기 시스템 권한을 설정해야 합니다.
- 유저는 객체를 생성한 후에 GRANT 문을 사용하여 사용 가능한 객체 권한을 다른 유저 또는 모든 유저에게 전달할 수 있습니다.
- DBA는 CREATE ROLE 문으로 룰을 생성하여 여러 유저에게 시스템 또는 객체 권한 모음을 전달할 수 있습니다. 룰을 사용하면 권한 부여 및 취소를 보다 쉽게 관리할 수 있습니다.
- 유저는 ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.
- REVOKE 문을 사용하면 유저로부터 권한을 취소할 수 있습니다.
- 유저는 데이터 딕셔너리 뷰에서 자신에게 부여된 권한과 자신의 객체에 부여된 권한을 확인할 수 있습니다.

연습 1: 개요

이 연습에서는 다음 내용을 다룹니다.

- 자신의 테이블에 대한 권한을 다른 유저에게 부여
- 자신에게 부여된 권한을 통해 다른 유저의 테이블 수정

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 1: 개요

이 연습에서는 다른 수강생과 팀을 구성하여 데이터베이스 객체에 대한 액세스 제어 과정을 연습합니다.

스키마 객체 관리

2

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 제약 조건 추가
- 인덱스 생성
- CREATE TABLE 문을 사용하여 인덱스 생성
- 함수 기반(Function-based) 인덱스 생성
- 열 삭제 및 UNUSED 열로 설정
- FLASHBACK 작업 수행
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원은 인덱스 및 제약 조건 생성과 기존 객체 변경에 대한 정보를 제공합니다. 또한 External Table 및 PRIMARY KEY 제약 조건을 생성할 때 인덱스 이름을 지정하는 규칙에 대해 설명합니다.

단원 내용

- **ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제**
- **제약 조건 관리:**
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- **인덱스 생성:**
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- **Flashback 작업 수행**
- **임시 테이블(Temporary Table) 생성 및 사용**
- **External Table 생성 및 사용**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

ALTER TABLE 문

ALTER TABLE 문을 사용하여 다음을 수행합니다.

- 새 열 추가
- 기존 열 수정
- 새 열의 기본값 정의
- 열 삭제

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

ALTER TABLE 문

테이블을 생성한 후에 열을 생략했거나 열 정의를 변경하거나 열을 제거하기 위해 테이블 구조를 변경해야 할 수 있습니다. 이 작업은 ALTER TABLE 문을 사용하여 수행할 수 있습니다.

ALTER TABLE 문

ALTER TABLE 문을 사용하여 열을 추가, 수정 또는 삭제합니다.

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
[ , column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
[ , column datatype]...);
```

```
ALTER TABLE table
DROP ( column [, column] ...);
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

ALTER TABLE 문(계속)

ALTER TABLE 문을 사용하여 테이블에서 열을 추가, 수정 및 삭제할 수 있습니다.

이 구문에서 다음이 적용됩니다.

<i>table</i>	테이블의 이름입니다.
ADD MODIFY DROP	수정의 유형입니다.
<i>column</i>	열 이름입니다.
<i>datatype</i>	열의 데이터 유형과 길이입니다.
DEFAULT <i>expr</i>	열 기본값을 지정합니다.

열 추가

- ADD 절을 사용하여 열을 추가합니다.

```
ALTER TABLE dept80  
ADD (job_id VARCHAR2(9));
```

```
ALTER TABLE dept80 succeeded.
```

- 새 열은 마지막 열이 됩니다.

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	145	Russell	14000	01-OCT-96	(null)
2	146	Partners	13500	05-JAN-97	(null)
3	147	Errazuriz	12000	10-MAR-97	(null)
4	148	Cambrault	11000	15-OCT-99	(null)
5	149	Zlotkey	10500	29-JAN-00	(null)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

열 추가를 위한 지침

- 열을 추가하거나 수정할 수 있습니다.
- 열을 표시할 위치를 지정할 수 없습니다. 새 열은 마지막 열이 됩니다.

위 슬라이드 예제에서는 DEPT80 테이블에 JOB_ID라는 열을 추가합니다. JOB_ID 열은 테이블의 마지막 열이 됩니다.

참고: 열을 추가할 때 테이블에 이미 행이 포함되어 있으면 새 열의 초기값은 모든 행에 대해 널이거나 기본값을 사용합니다. 다른 열에 데이터를 포함하고 있는 테이블은 기본값을 지정한 경우에만 필수 NOT NULL 열을 추가할 수 있습니다. 빈 테이블에는 기본값을 지정하지 않고도 NOT NULL 열을 추가할 수 있습니다.

열 수정

- 열의 데이터 유형, 크기 및 기본값을 변경할 수 있습니다.

```
ALTER TABLE dept80  
MODIFY      (last_name VARCHAR2(30));
```

```
ALTER TABLE dept80 succeeded.
```

- 기본값 변경은 이후에 테이블에 삽입하는 항목에만 적용됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

열 수정

ALTER TABLE 문을 MODIFY 절과 함께 사용하여 열 정의를 수정할 수 있습니다.
열 수정 작업에는 열의 데이터 유형, 크기 및 기본값에 대한 변경이 포함될 수 있습니다.

지침

- 숫자 열의 너비나 전체 자릿수를 늘릴 수 있습니다.
- 문자열의 너비를 늘릴 수 있습니다.
- 다음과 같은 경우에는 열의 너비를 줄일 수 있습니다.
 - 열이 널 값만 포함하는 경우
 - 테이블에 행이 없는 경우
 - 열 너비의 감소가 해당 열의 기존 값보다 작지 않은 경우
- 열에 널 값만 있는 경우 데이터 유형을 변경할 수 있습니다. 단, 예외적으로 CHAR에서 VARCHAR2로의 변환은 열에 데이터가 있어도 수행할 수 있습니다.
- 열에 널 값이 포함된 경우나 크기를 변경하지 않은 경우에만 CHAR 열을 VARCHAR2 데이터 유형으로 변환하거나 VARCHAR2 열을 CHAR 데이터 유형으로 변환할 수 있습니다.
- 열의 기본값 변경은 이후에 테이블에 삽입하는 항목에만 적용됩니다.

열 삭제

DROP COLUMN 절을 사용하여 테이블에서 더 이상 필요 없는
열을 삭제할 수 있습니다.

```
ALTER TABLE dept80
DROP COLUMN job_id;
```

```
ALTER TABLE dept80 succeeded.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	145	Russell	14000	01-OCT-96
2	146	Partners	13500	05-JAN-97
3	147	Errazuriz	12000	10-MAR-97
4	148	Cambrault	11000	15-OCT-99
5	149	Zlotkey	10500	29-JAN-00

Copyright © 2009, Oracle. All rights reserved.

열 삭제

ALTER TABLE 문을 DROP COLUMN 절과 함께 사용하여 테이블에서 열을 삭제할 수 있습니다.

지침

- 열에는 데이터가 포함되어 있거나 포함되어 있지 않습니다.
- ALTER TABLE DROP COLUMN 문을 사용하면 한 번에 한 개의 열만 삭제할 수 있습니다.
- 테이블이 변경된 후에는 최소 하나 이상의 열이 테이블에 남아 있어야 합니다.
- 열이 삭제된 후에는 recovery할 수 없습니다.
- CASCADE 옵션이 추가되지 않은 경우 열이 제약 조건의 일부이거나 인덱스 키의 일부라면 삭제할 수 없습니다.
- 많은 수의 값을 가진 열을 삭제하는 경우 약간의 시간이 걸릴 수 있습니다. 이 경우 확장된 잠금을 피하려면 시스템에 유저의 수가 적을 때 열을 unused로 설정하고 열을 삭제하는 것이 좋습니다.

참고: Partition 테이블의 Partitioning key의 일부를 형성하는 열이나 인덱스 구성 테이블(index-organized table)의 PRIMARY KEY의 일부를 형성하는 열과 같은 특정 열은 삭제할 수 없습니다. 인덱스 구성 테이블(Index-organized Table)과 Partition 테이블에 대한 자세한 내용은 *Oracle Database Concepts* 및 *Oracle Database Administrator's Guide*를 참조하십시오.

SET UNUSED 옵션

- SET UNUSED 옵션을 사용하여 하나 이상의 열을 unused로 표시합니다.
- DROP UNUSED COLUMNS 옵션을 사용하여 unused로 표시된 열을 제거할 수 있습니다.

```
ALTER TABLE <table_name>
SET UNUSED(<column_name> [ , <column_name>]);
OR
ALTER TABLE <table_name>
SET UNUSED COLUMN <column_name> [ , <column_name>];
```



```
ALTER TABLE <table_name>
DROP UNUSED COLUMNS;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SET UNUSED 옵션

SET UNUSED 옵션은 시스템 자원 사용률이 낮은 경우 열을 삭제할 수 있도록 하나 이상의 열을 unused로 표시합니다. 이 절을 지정해도 대상 열이 테이블의 각 행에서 실제로 제거되는 것은 아닙니다.(즉, 이 열이 사용하는 디스크 공간이 복원되는 것은 아닙니다.) 따라서 DROP 절을 실행한 경우보다 응답 시간이 빨라집니다. Unused 열은 테이블의 행에 열 데이터가 남아 있는 경우에도 삭제된 것으로 간주됩니다. Unused로 표시된 후에는 해당 열에 액세스할 수 없습니다. SELECT * query는 unused 열에서 데이터를 검색하지 않습니다. 또한 unused로 표시된 열의 이름 및 유형은 DESCRIBE 문이 실행될 때 표시되지 않고 unused 열과 동일한 이름을 가진 새 열을 테이블에 추가할 수 있습니다. SET UNUSED 정보는 USER_UNUSED_COL_TABS 딕셔너리 뷰에 저장됩니다.

참고: UNUSED 열 설정 지침은 열 삭제 지침과 유사합니다.

SET UNUSED 옵션(계속)

DROP UNUSED COLUMNS 옵션

DROP UNUSED COLUMNS는 현재 unused로 표시된 모든 열을 테이블에서 제거합니다.
테이블의 사용되지 않은 열에서 추가 디스크 공간을 회수하려면 이 명령문을 사용할 수 있습니다.
사용되지 않은 열이 테이블에 없는 경우 이 명령문은 아무 오류도 반환하지 않습니다.

```
ALTER TABLE dept80  
SET UNUSED (last_name);
```

```
ALTER TABLE succeeded
```

```
ALTER TABLE dept80  
DROP UNUSED COLUMNS;
```

```
ALTER TABLE succeeded
```

단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 구문 추가

ALTER TABLE 문을 사용하여 다음을 수행합니다.

- 제약 조건 추가 또는 삭제. 제약 조건의 구조는 수정하지 않음
- 제약 조건 활성화 또는 비활성화
- MODIFY 절을 사용하여 NOT NULL 제약 조건 추가

```
ALTER TABLE <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>);
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 추가

ALTER TABLE 문을 ADD 절과 함께 사용하여 기존 테이블에 제약 조건을 추가할 수 있습니다.

이 구문에서 다음이 적용됩니다.

table 테이블의 이름입니다.

constraint 제약 조건의 이름입니다.

type 제약 조건 유형입니다.

column 제약 조건의 영향을 받는 열의 이름입니다.

제약 조건 이름 구문은 지정하면 좋지만 선택 사항입니다. 제약 조건의 이름을 지정하지 않으면 시스템이 제약 조건 이름을 생성합니다.

지침

- 제약 조건을 추가, 삭제, 활성화 또는 비활성화할 수 있지만 제약 조건의 구조를 수정할 수는 없습니다.
- ALTER TABLE 문의 MODIFY 절을 사용하여 기존 열에 NOT NULL 제약 조건을 추가할 수 있습니다.

참고: 테이블이 비어 있는 경우 또는 열의 모든 행에 값이 있는 경우에만 NOT NULL 열을 정의할 수 있습니다.

제약 조건 추가

EMP2 테이블에 FOREIGN KEY 제약 조건을 추가하면
관리자가 이미 EMP2 테이블에 유효한 사원으로 존재해야
함을 나타냅니다.

```
ALTER TABLE emp2
MODIFY employee_id PRIMARY KEY;
```

```
ALTER TABLE emp2 succeeded.
```

```
ALTER TABLE emp2
ADD CONSTRAINT emp_mgr_fk
FOREIGN KEY(manager_id)
REFERENCES emp2(employee_id);
```

```
ALTER TABLE succeeded.
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 추가(계속)

슬라이드의 첫번째 예제에서는 EMP2 테이블을 수정하여 EMPLOYEE_ID 열에 PRIMARY KEY 제약 조건을 추가합니다. 제약 조건의 이름을 제공하지 않았으므로 Oracle 서버가 자동으로 제약 조건의 이름을 지정합니다. 슬라이드의 두번째 예제에서는 EMP2 테이블에 FOREIGN KEY 제약 조건을 생성합니다. 이 제약 조건은 관리자가 EMP2 테이블에서 유효한 사원으로 존재하도록 보장합니다.

ON DELETE 절

- 상위 키가 삭제될 때 하위 행을 삭제하려면 ON DELETE CASCADE 절을 사용합니다.

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments(department_id) ON DELETE CASCADE;
```

ALTER TABLE Emp2 succeeded.

- 상위 키가 삭제될 때 하위 행을 널로 설정하려면 ON DELETE SET NULL 절을 사용합니다.

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments(department_id) ON DELETE SET NULL;
```

ALTER TABLE Emp2 succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ON DELETE

ON DELETE 절을 사용하면 참조된 Primary Key 또는 Unique Key 값을 제거할 경우 오라클 데이터베이스에서 참조 무결성을 처리하는 방법을 결정할 수 있습니다.

ON DELETE CASCADE

ON DELETE CASCADE 작업은 하위 테이블에서 참조되는 상위 키 데이터가 삭제되도록 합니다(갱신되지는 않음). 상위 키의 데이터가 삭제되면 해당 상위 키 값에 종속된 하위 테이블의 모든 행도 삭제됩니다. 이 참조 작업을 지정하려면 FOREIGN KEY 제약 조건의 정의에 ON DELETE CASCADE 옵션을 포함시키십시오.

ON DELETE SET NULL

상위 키의 데이터가 삭제되면 ON DELETE SET NULL 작업으로 인해 해당 상위 키 값에 종속된 하위 테이블의 모든 행이 널로 변환됩니다.

이 절을 생략할 경우 하위 테이블에 종속 행이 있는 상위 테이블의 참조된 키 값을 삭제할 수 없습니다.

제약 조건 지연

제약 조건은 다음 속성을 가질 수 있습니다.

- DEFERRABLE 또는 NOT DEFERRABLE
- INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE

```
ALTER TABLE dept2
ADD CONSTRAINT dept2_id_pk
PRIMARY KEY (department_id)
DEFERRABLE INITIALLY DEFERRED
```

생성 시 제약 조건 지연

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE
```

특정 제약 조건 속성 변경

```
ALTER SESSION
SET CONSTRAINTS= IMMEDIATE
```

세션의 모든 제약 조건 변경

ORACLE

Copyright © 2009, Oracle. All rights reserved.

제약 조건 지연

트랜잭션이 끝날 때까지 제약 조건의 유효성 검사를 지연할 수 있습니다. 시스템에서 제약 조건이 충족되는지 검사하지 않을 경우 COMMIT 문이 실행될 때까지 제약 조건이 지연됩니다. 지연된 제약 조건을 위반할 경우 데이터베이스에서 오류를 반환하며 트랜잭션이 커밋되지 않고 롤백됩니다. 제약 조건이 deferred가 아니라 immediate인 경우 각 명령문의 끝에서 제약 조건이 검사됩니다. 제약 조건을 위반할 경우 명령문이 즉시 롤백됩니다. 제약 조건으로 인해 DELETE CASCADE와 같은 작업이 발생하면, 제약 조건이 deferred인지 또는 immediate인지에 관계없이 항상 해당 작업을 발생시킨 명령문의 일부로 작업이 실행됩니다. SET CONSTRAINTS 문을 사용하여 특정 트랜잭션에 대해 deferrable 제약 조건을 각 DML(데이터 조작어) 문 이후에 검사할지 또는 트랜잭션이 커밋될 때 검사할지 지정합니다. deferrable 제약 조건을 생성하려면 해당 제약 조건에 대한 비고유 인덱스를 생성해야 합니다.

제약 조건을 DEFERRABLE 또는 NOT DEFERRABLE로 정의하거나 INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE로 정의할 수 있습니다. 이러한 속성은 각 제약 조건에 따라 다를 수 있습니다.

사용 시나리오: 회사 방침에 따라 부서 번호 40을 45로 변경해야 합니다. DEPARTMENT_ID 열을 변경하면 이 부서에 할당된 사원에게 영향이 있습니다. 따라서 PRIMARY KEY 및 FOREIGN KEY를 DEFERRABLE 및 INITIALLY DEFERRED로 지정합니다. 부서 정보 및 사원 정보를 모두 갱신하면 커밋 시 모든 행이 검사됩니다.

INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이

INITIALLY DEFERRED	트랜잭션이 종료될 때까지 제약 조건 검사를 기다립니다.
INITIALLY IMMEDIATE	명령문 실행이 완료되면 제약 조건을 검사합니다.

```
CREATE TABLE emp_new_sal (salary NUMBER
                           CONSTRAINT sal_ck
                           CHECK (salary > 100)
                           DEFERRABLE INITIALLY IMMEDIATE,
                           bonus NUMBER
                           CONSTRAINT bonus_ck
                           CHECK (bonus > 0 )
                           DEFERRABLE INITIALLY DEFERRED );
```

create table succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이

Deferrable로 정의된 제약 조건은 INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE로 지정할 수 있습니다. 기본값은 INITIALLY IMMEDIATE 절입니다.

슬라이드 예제 설명:

- sal_ck 제약 조건이 DEFERRABLE INITIALLY IMMEDIATE로 생성됩니다.
- bonus_ck 제약 조건이 DEFERRABLE INITIALLY DEFERRED로 생성됩니다.

슬라이드에 표시된 것과 같이 emp_new_sal 테이블을 생성한 후에는 테이블에 값을 삽입하고 그 결과를 확인하십시오. sal_ck 제약 조건과 bonus_ck 제약 조건을 모두 만족하면 오류 없이 행이 삽입됩니다.

예제 1: sal_ck를 위반하는 행을 삽입합니다. CREATE TABLE 문에 sal_ck가 initially immediate 제약 조건으로 지정됩니다. 따라서 INSERT 문 직후에 제약 조건이 검사되고 오류가 반환됩니다.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

SQL Error: ORA-02290: check constraint (ORA21.SAL_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"

예제 2: bonus_ck를 위반하는 행을 삽입합니다. CREATE TABLE 문에 bonus_ck가 deferrable 및 initially deferred로 지정됩니다. 따라서 COMMIT을 수행하거나 제약 조건 상태를 다시 immediate로 설정할 때까지 제약 조건이 검사되지 않습니다.

INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이(계속)

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

1 rows inserted

행이 성공적으로 삽입됩니다. 그러나 트랜잭션을 커밋하면 오류가 반환됩니다.

```
COMMIT;
```

SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.BONUS_CK) violated
02091. 00000 - "transaction rolled back"

제약 조건을 위반했기 때문에 커밋을 실패했습니다. 그러므로 이때 트랜잭션이 데이터베이스에 의해 롤백됩니다.

예제 3: 자연 가능한 모든 제약 조건에 DEFERRED 상태를 설정합니다. 필요한 경우 DEFERRED 상태를 한 개의 제약 조건으로 설정할 수도 있습니다.

```
SET CONSTRAINTS ALL DEFERRED;
```

SET CONSTRAINTS succeeded.

이제 sal_ck 제약 조건을 위반하는 행을 삽입하려고 하면 명령문이 성공적으로 실행됩니다.

```
INSERT INTO emp_new_sal VALUES(90, 5);
```

1 rows inserted

그러나 트랜잭션을 커밋하면 오류가 반환됩니다. 트랜잭션이 실패하여 롤백됩니다.

이는 COMMIT 시 두 제약 조건이 검사되기 때문입니다.

```
COMMIT;
```

SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.SAL_CK) violated
02091. 00000 - "transaction rolled back"

예제 4: IMMEDIATE 상태를 이전 예에서 DEFERRED로 설정한 두 제약 조건으로 설정합니다.

```
SET CONSTRAINTS ALL IMMEDIATE;
```

SET CONSTRAINTS succeeded.

sal_ck 또는 bonus_ck를 위반하는 행을 삽입하려고 하면 오류가 반환됩니다.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

SQL Error: ORA-02290: check constraint (ORA21.BONUS_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"

참고: 제약 조건 자연 가능성을 지정하지 않고 테이블을 생성하면 각 명령문의 끝에서 제약 조건이 즉시 검사됩니다. 예를 들어, newemp_details 테이블의 CREATE TABLE 문을 사용할 경우 newemp_det_pk 제약 조건 자연 가능성을 지정하지 않으면 제약 조건이 즉시 검사됩니다.

```
CREATE TABLE newemp_details(emp_id NUMBER, emp_name
VARCHAR2(20),
CONSTRAINT newemp_det_pk PRIMARY KEY(emp_id));
```

자연 불가능한 newemp_det_pk 제약 조건을 자연하려고 하면 다음 오류가 반환됩니다.

```
SET CONSTRAINT newemp_det_pk DEFERRED;
```

SQL Error: ORA-02447: cannot defer a constraint that is not deferrable

제약 조건 삭제

- EMP2 테이블에서 관리자 제약 조건을 제거합니다.

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```

ALTER TABLE Emp2 succeeded.

- DEPT2 테이블에서 PRIMARY KEY 제약 조건을 제거하고 EMP2 .DEPARTMENT_ID 열에서 연관된 FOREIGN KEY 제약 조건을 삭제합니다.

```
ALTER TABLE dept2
DROP PRIMARY KEY CASCADE;
```

ALTER TABLE dept2 succeeded.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 삭제

제약 조건을 삭제하려면 USER_CONSTRAINTS 및 USER_CONS_COLUMNS 데이터 딕셔너리에서 제약 조건 이름을 식별합니다. 그런 다음 ALTER TABLE 문을 DROP 절과 함께 사용합니다. DROP 절의 CASCADE 옵션으로 인해 종속 제약 조건도 삭제됩니다.

구문

```
ALTER TABLE    table
  DROP PRIMARY KEY | UNIQUE (column) |
    CONSTRAINT constraint [CASCADE];
```

이 구문에서 다음이 적용됩니다.

table 테이블의 이름입니다.

column 제약 조건의 영향을 받는 열의 이름입니다.

constraint 제약 조건의 이름입니다.

무결성 제약 조건을 삭제하면 해당 제약 조건은 더 이상 Oracle 서버에 의해 수행되지 않으며 데이터 딕셔너리에서도 사용할 수 없습니다.

제약 조건 비활성화

- ALTER TABLE 문의 DISABLE 절을 실행하여 무결성 제약 조건을 비활성화합니다.
- CASCADE 옵션을 적용하여 종속 무결성 제약 조건을 비활성화할 수 있습니다.

```
ALTER TABLE emp2
DISABLE CONSTRAINT emp_dt_fk;
```

ALTER TABLE Emp2 succeeded.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 비활성화

ALTER TABLE 문을 DISABLE 절과 함께 사용하여 제약 조건을 삭제하거나 다시 생성하지 않고 비활성화할 수 있습니다.

구문

```
ALTER TABLE table
DISABLE CONSTRAINT constraint [CASCADE];
```

이 구문에서 다음이 적용됩니다.

table 테이블의 이름입니다.

constraint 제약 조건의 이름입니다.

자침

- CREATE TABLE 문과 ALTER TABLE 문에서 모두 DISABLE 절을 사용할 수 있습니다.
- CASCADE 절은 종속 무결성 제약 조건을 비활성화합니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 비활성화하면 고유 인덱스가 제거됩니다.

제약 조건 활성화

- **ENABLE 절을 사용하여 테이블 정의에서 현재 비활성화된 무결성 제약 조건을 활성화할 수 있습니다.**

```
ALTER TABLE          emp2
ENABLE CONSTRAINT emp_dt_fk;
```

[ALTER TABLE Emp2 succeeded.]

- **UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하면 UNIQUE 인덱스가 자동으로 생성됩니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 활성화

ALTER TABLE 문을 ENABLE 절과 함께 사용하여 제약 조건을 삭제하거나 다시 생성하지 않고 활성화할 수 있습니다.

구문

```
ALTER TABLE      table
ENABLE CONSTRAINT constraint;
```

이 구문에서 다음이 적용됩니다.

table 테이블의 이름입니다.

constraint 제약 조건의 이름입니다.

자침

- 제약 조건을 활성화하면 해당 제약 조건이 테이블의 모든 데이터에 적용됩니다. 테이블의 모든 데이터가 제약 조건을 따라야 합니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하면 UNIQUE 또는 PRIMARY KEY 인덱스가 자동으로 생성됩니다. 인덱스가 이미 존재하는 경우 해당 인덱스를 이러한 키로 사용할 수 있습니다.
- CREATE TABLE 문 및 ALTER TABLE 문에서 모두 ENABLE 절을 사용할 수 있습니다.

제약 조건 활성화(계속)

- CASCADE 옵션으로 비활성화된 PRIMARY KEY 제약 조건을 활성화하면 PRIMARY KEY에 종속된 FOREIGN KEY가 활성화되지 않습니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하려면 테이블에 인덱스를 생성하는 데 필요한 권한을 가지고 있어야 합니다.

계단식 제약 조건

- CASCADE CONSTRAINTS 절은 DROP COLUMN 절과 함께 사용됩니다.
- CASCADE CONSTRAINTS 절은 삭제된 열에 정의된 PRIMARY KEY 및 UNIQUE KEY를 참조하는 모든 참조 무결성 제약 조건을 삭제합니다.
- CASCADE CONSTRAINTS 절은 또한 삭제된 열에 정의된 모든 다중 열 제약 조건을 삭제합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

계단식 제약 조건

다음 명령문은 CASCADE CONSTRAINTS 절의 사용을 보여줍니다. TEST1 테이블이 다음과 같이 생성된다고 가정합니다.

```
CREATE TABLE test1 (
    col1_pk NUMBER PRIMARY KEY,
    col2_fk NUMBER,
    col1 NUMBER,
    col2 NUMBER,
    CONSTRAINT fk_constraint FOREIGN KEY (col2_fk) REFERENCES
        test1,
    CONSTRAINT ck1 CHECK (col1_pk > 0 and col1 > 0),
    CONSTRAINT ck2 CHECK (col2_fk > 0));
```

다음 명령문에 대해 오류가 반환됩니다.

ALTER TABLE test1 DROP (col1_pk); -col1_pk는 상위 키입니다.

ALTER TABLE test1 DROP (col1); -col1은 다중 열 제약 조건
ck1에 의해 참조됩니다.

계단식 제약 조건

예제:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

```
ALTER TABLE Emp2 succeeded.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1) CASCADE CONSTRAINTS;
```

```
ALTER TABLE test1 succeeded.
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

제약 조건 연쇄화(계속)

다음 명령문을 실행하면 EMPLOYEE_ID 열, PRIMARY KEY 제약 조건 및 EMP2 테이블의 PRIMARY KEY 제약 조건을 참조하는 모든 FOREIGN KEY 제약 조건이 삭제됩니다.

```
ALTER TABLE emp2 DROP COLUMN employee_id CASCADE CONSTRAINTS;
삭제된 열에 정의된 제약 조건에서 참조하는 모든 열도 삭제된 경우 CASCADE
CONSTRAINTS는 필요하지 않습니다. 예를 들어, 다른 테이블의 참조 제약 조건이
COL1_PK 열을 참조하지 않는다고 가정하면 이전 페이지에서 생성한 TEST1 테이블에
대해 CASCADE CONSTRAINTS 절 없이 다음 명령문을 실행하는 것은 유효합니다.
```

```
ALTER TABLE test1 DROP (col1_pk, col2_fk, col1);
```

테이블 열 및 제약 조건 이름 바꾸기

ALTER TABLE 문의 **RENAME COLUMN** 절을 사용하여 테이블 열의 이름을 바꿀 수 있습니다.

```
ALTER TABLE marketing RENAME COLUMN team_id
TO id;
```

```
ALTER TABLE marketing succeeded.
```

a

ALTER TABLE 문의 **RENAME CONSTRAINT** 절을 사용하여 테이블의 기존 제약 조건에 대한 이름을 바꿀 수 있습니다.

```
ALTER TABLE marketing RENAME CONSTRAINT mktg_pk
TO new_mktg_pk;
```

```
ALTER TABLE marketing succeeded.
```

b

Copyright © 2009, Oracle. All rights reserved.

테이블 열 및 제약 조건 이름 바꾸기

테이블 열의 이름을 바꾸는 경우 새 이름이 기존 테이블 열의 이름과 충돌해서는 안되며, **RENAME COLUMN** 절과 함께 다른 절을 사용할 수 없습니다.

위 슬라이드 예제에서는 **id** 열에 PRIMARY KEY **mktg_pk**가 정의된 **marketing** 테이블을 사용합니다.

```
CREATE TABLE marketing (team_id NUMBER(10),
                      target VARCHAR2(50),
                      CONSTRAINT mktg_pk PRIMARY KEY(team_id));
```

```
CREATE TABLE succeeded.
```

예제 **a**는 **marketing** 테이블의 **id** 열 이름이 **mktg_id**로 바뀌었음을 보여주고, 예제 **b**는 **mktg_pk**의 이름이 **new_mktg_pk**로 바뀌었음을 보여줍니다.

테이블의 기존 제약 조건 이름을 바꾸는 경우 새 이름이 기존 제약 조건 이름과 충돌해서는 안됩니다. **RENAME CONSTRAINT** 절을 사용하면 시스템에서 생성한 제약 조건의 이름을 바꿀 수 있습니다.

단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

인덱스 개요

인덱스는 다음과 같이 생성됩니다.

- **자동**
 - PRIMARY KEY 생성
 - UNIQUE KEY 생성
- **수동**
 - CREATE INDEX 문
 - CREATE TABLE 문

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

인덱스 개요

두 가지 유형의 인덱스를 생성할 수 있습니다. 첫번째 유형은 고유 인덱스입니다. PRIMARY KEY 또는 UNIQUE KEY 제약 조건을 갖도록 테이블의 열 또는 열 그룹을 정의할 때 Oracle 서버가 자동으로 고유 인덱스를 생성합니다. 인덱스의 이름은 제약 조건에 지정된 이름입니다. 두번째 인덱스 유형은 비고유 인덱스이며 유저가 생성할 수 있습니다. 예를 들어 검색 속도를 향상시키기 위해 조인에서 사용될 FOREIGN KEY 열에 대한 인덱스를 생성할 수 있습니다. CREATE INDEX 문을 실행하여 하나 이상의 열에 인덱스를 생성할 수 있습니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

참고: 고유 인덱스를 수동으로 생성할 수도 있지만, 고유 인덱스를 암시적으로 생성하는 UNIQUE 제약 조건을 생성하는 것이 좋습니다.

CREATE TABLE 문을 사용한 CREATE INDEX

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
 PRIMARY KEY USING INDEX
 (CREATE INDEX emp_id_idx ON
 NEW_EMP(employee_id)),
first_name VARCHAR2(20),
last_name VARCHAR2(25));
```

CREATE TABLE succeeded.

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'NEW_EMP';
```

INDEX_NAME	TABLE_NAME
EMP_ID_IDX	NEW_EMP

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CREATE TABLE 문을 사용한 CREATE INDEX

슬라이드의 예제에서 CREATE INDEX 절은 CREATE TABLE 문과 함께 사용되어 PRIMARY KEY 인덱스를 명시적으로 생성합니다. PRIMARY KEY 생성 시 PRIMARY KEY 제약 조건의 이름과 다르게 인덱스의 이름을 지정할 수 있습니다.

USER_INDEXES 데이터 딕셔너리 뷰를 query하여 인덱스에 대한 정보를 확인할 수 있습니다.

참고: USER_INDEXES에 대한 자세한 내용은 "데이터 딕셔너리 뷰를 사용하여 객체 관리" 단원을 참조하십시오.

다음 예제는 인덱스의 이름이 명시적으로 지정되지 않은 경우의 데이터베이스 동작을 보여줍니다.

```
CREATE TABLE EMP_UNNAMED_INDEX
(employee_id NUMBER(6) PRIMARY KEY ,
first_name VARCHAR2(20),
last_name VARCHAR2(25));
```

CREATE TABLE succeeded.

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

INDEX_NAME	TABLE_NAME
SYS_C0017294	EMP_UNNAMED_INDEX

CREATE TABLE 문을 사용한 CREATE INDEX(계속)

Oracle 서버는 PRIMARY KEY 열에 대해 생성되는 인덱스에 일반적인 이름을 지정합니다.

또한 예를 들어 데이터 로드가 많을 것으로 예상되어 작업 속도를 높이려는 경우 PRIMARY KEY 열에 기존 인덱스를 사용할 수 있습니다. 로드를 수행하는 동안 제약 조건을 비활성화한 다음 제약 조건을 다시 활성화하기를 원하는 경우, PRIMARY KEY에 대한 고유 인덱스가 있으면 로드 중에 데이터가 계속 검사됩니다. 따라서 우선 PRIMARY KEY로 지정된 열에 비고유 인덱스를 생성한 다음 PRIMARY KEY 열을 생성하고 이 열이 기존 인덱스를 사용하도록 지정할 수 있습니다. 다음 예제는 이 프로세스를 보여줍니다.

1단계: 테이블 생성:

```
CREATE TABLE NEW_EMP2
  (employee_id NUMBER(6),
   first_name VARCHAR2(20),
   last_name VARCHAR2(25)
  );
```

2단계: 인덱스 생성:

```
CREATE INDEX emp_id_idx2 ON
  new_emp2(employee_id);
```

3단계: PRIMARY KEY 생성:

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
emp_id_idx2;
```

함수 기반 인덱스

- 함수 기반 인덱스는 표현식을 기반으로 합니다.
- 인덱스 표현식은 테이블 열, 제약 조건, SQL 함수 및 유저 정의 함수에서 작성됩니다.

```
CREATE INDEX upper_dept_name_idx
ON dept2(UPPER(department_name));
```

CREATE INDEX succeeded.

```
SELECT *
FROM    dept2
WHERE   UPPER(department_name) = 'SALES';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 기반 인덱스

UPPER(column_name) 또는 LOWER(column_name) 키워드로 정의된 함수 기반 인덱스는 검색 시 대소문자를 구분하지 않습니다. 예를 들어, 다음 인덱스를 살펴 보십시오.

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

이 인덱스는 다음과 같은 query를 효과적으로 처리합니다.

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

Oracle 서버는 query에서 특정 함수가 사용되는 경우에만 인덱스를 사용합니다. 예를 들어, 다음 명령문은 인덱스를 사용할 수도 있지만 WHERE 절이 없는 경우 Oracle 서버가 전체 테이블 스캔을 수행할 수도 있습니다.

```
SELECT *
FROM    employees
WHERE   UPPER (last_name) IS NOT NULL
ORDER BY UPPER (last_name);
```

참고: 사용할 함수 기반 인덱스에 대해 QUERY_REWRITE_ENABLED 초기화 파라미터가 TRUE로 설정되어야 합니다.

Oracle 서버는 DESC로 표시된 열이 있는 인덱스를 함수 기반 인덱스로 처리합니다. DESC로 표시된 열은 내림차순으로 정렬됩니다.

인덱스 제거

- **DROP INDEX 명령을 사용하여 데이터 딕셔너리에서 인덱스를 제거할 수 있습니다.**

```
DROP INDEX index;
```

- **데이터 딕셔너리에서 UPPER_DEPT_NAME_IDX 인덱스를 제거합니다.**

```
DROP INDEX upper_dept_name_idx;
```

```
DROP INDEX upper_dept_name_idx succeeded.
```

- **인덱스를 삭제하려면 인덱스의 소유자이거나 DROP ANY INDEX 권한이 있어야 합니다.**

Copyright © 2009, Oracle. All rights reserved.

인덱스 제거

인덱스는 수정할 수 없습니다. 인덱스를 변경하려면 인덱스를 삭제한 다음 다시 생성해야 합니다. **DROP INDEX** 문을 실행하여 데이터 딕셔너리에서 인덱스 정의를 제거합니다.

인덱스를 삭제하려면 인덱스의 소유자이거나 **DROP ANY INDEX** 권한이 있어야 합니다.

이 구문에서 다음이 적용됩니다.

index 인덱스의 이름입니다.

참고: 테이블을 삭제하면 인덱스, 제약 조건 및 트리거는 자동으로 삭제되지만 뷰와 시퀀스는 남아 있습니다.

DROP TABLE ... PURGE

```
DROP TABLE dept80 PURGE;
```

```
DROP TABLE dept80 succeeded.
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DROP TABLE ... PURGE

오라클 데이터베이스는 테이블 삭제 기능을 제공합니다. 테이블을 삭제할 때 데이터베이스는 테이블과 연관된 공간을 즉시 해제하지 않습니다. 대신 테이블의 이름을 바꾸고 테이블을 Recycle bin에 넣습니다. 따라서 테이블을 실수로 삭제했을 경우 나중에 FLASHBACK TABLE 문으로 테이블을 recovery할 수 있습니다. DROP TABLE 문을 실행할 때 테이블과 연관된 공간을 즉시 해제하려면 슬라이드의 명령문에서와 같이 PURGE 절을 포함시키십시오.

한 단계에서 테이블을 삭제하고 테이블과 연관된 공간을 해제하려는 경우에만 PURGE를 지정하십시오. PURGE를 지정한 경우 데이터베이스는 테이블과 종속 객체를 Recycle bin에 넣지 않습니다.

이 절을 사용하는 것은 먼저 테이블을 삭제한 다음 Recycle bin에서 테이블을 지우는 것과 같습니다. 이 절을 사용하면 프로세스를 한 단계 줄일 수 있습니다. 또한 중요한 자료를 Recycle bin에서 숨길 수 있는 고급 보안 기능을 제공합니다.

단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

FLASHBACK TABLE 문

- 한 개의 명령문으로 테이블을 지정된 시점으로 recovery할 수 있습니다.
- 테이블 데이터를 연관된 인덱스 및 제약 조건과 함께 복원합니다.
- 테이블과 테이블의 내용을 특정 시점이나 시스템 변경 번호(SCN)로 되돌릴 수 있습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

FLASHBACK TABLE 문

Oracle Flashback Table을 사용하면 한 개의 명령문으로 테이블을 지정된 시점으로 recovery할 수 있습니다. 데이터베이스가 온라인 상태인 동안에 연관된 인덱스 및 제약 조건과 함께 테이블 데이터를 복원하여 지정된 테이블에 대한 변경 사항만 염두할 수 있습니다.

Flashback Table 기능은 셀프 서비스 복구 도구와 비슷합니다. 예를 들어, 테이블에서 실수로 삭제한 중요한 행을 recovery하려는 경우 FLASHBACK TABLE 문을 사용하여 테이블을 삭제 이전 시기로 복원하면 없어졌던 행을 테이블에서 볼 수 있습니다.

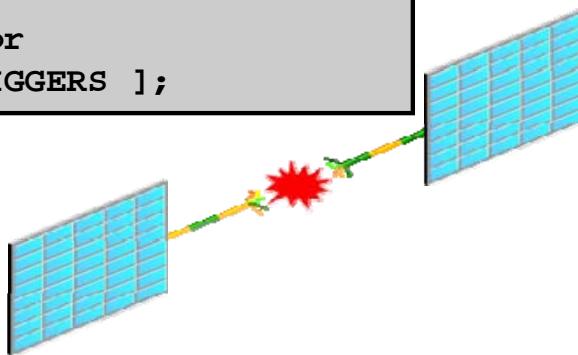
FLASHBACK TABLE 문을 사용할 경우 테이블과 테이블의 내용을 특정 시점이나 SCN으로 되돌릴 수 있습니다.

참고: SCN은 각 데이터베이스 변경 사항과 연관된 정수 값으로, 데이터베이스에서 고유한 incremental 숫자입니다. 트랜잭션을 커밋할 때마다 새로운 SCN이 기록됩니다.

FLASHBACK TABLE 문

- 실수로 테이블이 수정된 경우 복구 도구로 사용됩니다.
 - 테이블을 과거 시점으로 복원
 - 이점: 사용 편의성, 가용성, 빠른 실행
 - 현재 위치에서 수행됨
- 구문:

```
FLASHBACK TABLE[schema.]table[,  
schema.]table ]...  
TO { TIMESTAMP | SCN } expr  
[ { ENABLE | DISABLE } TRIGGERS ];
```



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

FLASHBACK TABLE 문(계속)

셀프 서비스 복구 도구

오라클 데이터베이스는 테이블이 삭제되거나 수정된 경우 테이블의 상태를 과거 시점으로 복원할 수 있는 새로운 SQL DDL(데이터 정의어) 명령 FLASHBACK TABLE을 제공합니다. FLASHBACK TABLE 명령은 테이블의 데이터를 인덱스 또는 뷰와 같은 관련 속성과 함께 복원하는 셀프 서비스 복구 도구입니다. 이 명령은 데이터베이스가 온라인일 때 지정된 테이블의 이후 변경 사항만 롤백합니다. 기존 recovery 방식과 비교하여 이 기능은 사용 편의성, 가용성 및 빠른 복원과 같은 중요한 이점을 제공합니다. 또한 응용 프로그램 특정 속성을 찾아 복원하므로 DBA의 작업 부담을 줄여 줍니다. FLASHBACK TABLE 기능은 잘못된 디스크로 인해 발생하는 물리적 손상을 해결하지 못합니다.

구문

하나 이상의 테이블에서(서로 다른 스키마의 테이블인 경우도 포함) FLASHBACK TABLE 작업을 호출할 수 있습니다. 유효한 시간 기록을 제공하여 되돌아 가려는 시점을 지정합니다. 기본적으로 flashback 작업 동안에는 관련된 모든 테이블에 대해 데이터베이스 트리거가 비활성화됩니다. ENABLE TRIGGERS 절을 지정하여 이 기본 동작을 재정의할 수 있습니다.

참고: Recycle bin 및 Flashback 구문에 대한 자세한 내용은

*Oracle Database Administrator's Guide 11g Release 2 (11.2)*를 참조하십시오.

FLASHBACK TABLE 문 사용

```
DROP TABLE emp2;
```

DROP TABLE emp2 succeeded.

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

ORIGINAL_NAME	OPERATION	DROPTIME
EMP2	DROP	2009-05-20 18:00:39

...

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

FLASHBACK TABLE succeeded.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

FLASHBACK TABLE 문 사용

구문 및 예제

이 예제에서는 EMP2 테이블을 DROP 문 이전 상태로 복원합니다.

Recycle bin은 실제로는 삭제된 객체에 대한 정보가 담긴 데이터 딕셔너리 테이블입니다. 삭제된 테이블 및 인덱스, 제약 조건, 중첩 테이블 등과 같은 모든 연관된 객체는 제거되지 않고 여전히 공간을 차지합니다. 이러한 객체는 Recycle bin에서 완전히 지워지거나 테이블스페이스 공간 제약 조건으로 인해 데이터베이스에서 삭제될 때까지 계속해서 유저 공간 할당량에 포함됩니다.

유저가 SYSDBA 권한을 갖고 있지 않은 경우 자신이 소유한 객체만 Recycle bin에서 액세스할 수 있으므로 각 유저를 Recycle bin의 소유자로 간주할 수 있습니다. 유저는 다음 명령문을 사용하여 Recycle bin에서 자신의 객체를 볼 수 있습니다.

```
SELECT * FROM RECYCLEBIN;
```

유저를 삭제하면 해당 유저에 속한 모든 객체는 Recycle bin에 저장되지 않고 Recycle bin의 모든 객체가 지워집니다.

다음 명령문으로 Recycle bin을 비울 수 있습니다.

```
PURGE RECYCLEBIN;
```

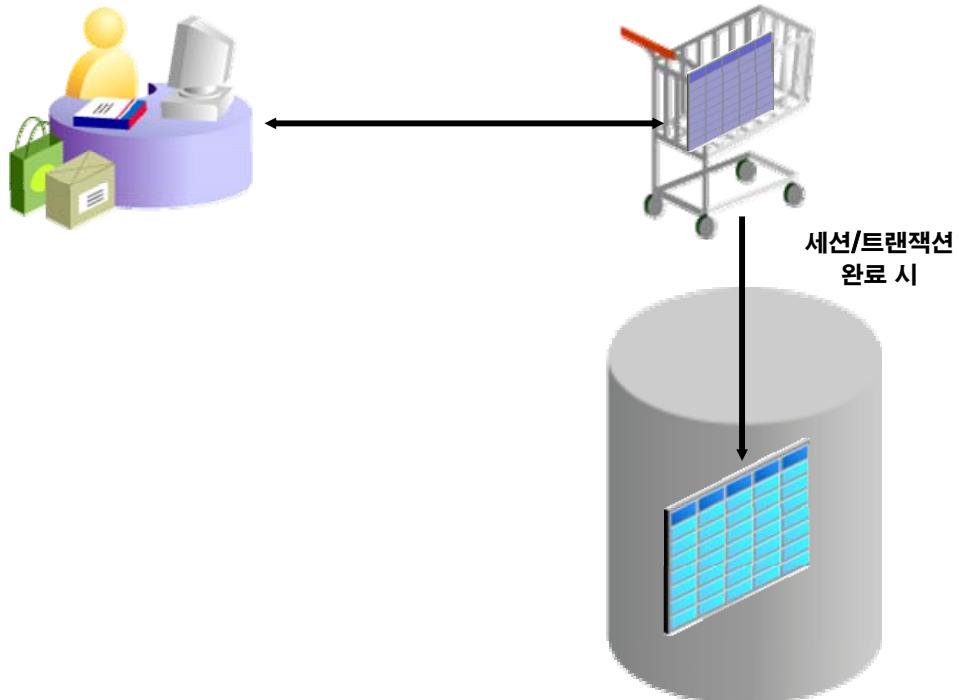
단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

임시 테이블



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

임시 테이블

임시 테이블은 트랜잭션이나 세션 기간 동안에만 존재하는 데이터를 보유하는 테이블입니다. 임시 테이블의 데이터는 세션 전용(private)으로 각 세션은 자신의 데이터만 보고 수정할 수 있습니다.

임시 테이블은 결과 집합을 버퍼링해야 하는 응용 프로그램에서 유용합니다. 예를 들어, 온라인 응용 프로그램의 쇼핑 카트가 임시 테이블일 수 있습니다. 각 항목은 임시 테이블의 한 행으로 표현됩니다. 온라인 상점에서 쇼핑하는 동안 카트에서 계속 항목을 추가하거나 제거할 수 있습니다. 세션 동안 이 카트 데이터는 전용(private)입니다. 쇼핑을 마치고 지불한 후에는 선택한 카트의 행이 영구 테이블로 이동됩니다. 세션이 끝날 때 임시 테이블의 데이터가 자동으로 삭제됩니다.

임시 테이블은 정적으로 정의되기 때문에 그에 대한 인덱스를 생성할 수 있습니다. 임시 테이블에 생성된 인덱스도 임시입니다. 인덱스 데이터의 세션 또는 트랜잭션 범위는 임시 테이블의 데이터와 같습니다. 임시 테이블에 뷰나 트리거를 생성할 수도 있습니다.

임시 테이블 생성

```
CREATE GLOBAL TEMPORARY TABLE cart
ON COMMIT DELETE ROWS;
```

1

```
CREATE GLOBAL TEMPORARY TABLE today_sales
ON COMMIT PRESERVE ROWS AS
SELECT * FROM orders
WHERE order_date = SYSDATE;
```

2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

임시 테이블 생성

임시 테이블을 생성하기 위해 다음 명령을 사용할 수 있습니다.

```
CREATE GLOBAL TEMPORARY TABLE tablename
ON COMMIT [PRESERVE | DELETE] ROWS
```

다음 설정 중 하나를 ON COMMIT 절과 연관시켜 임시 테이블의 데이터가 트랜잭션 관련 데이터(기본값)인지 아니면 세션 관련 데이터인지 결정할 수 있습니다.

1. **DELETE ROWS:** 슬라이드의 예제 1과 같이 DELETE ROWS 설정은 트랜잭션 관련 임시 테이블을 생성합니다. 테이블에 트랜잭션이 처음 삽입될 때 세션이 임시 테이블에 바인드되며, 트랜잭션이 끝날 때 바인드가 해제됩니다. 커밋할 때마다 데이터베이스가 테이블을 truncate하여 모든 행을 삭제합니다.
2. **PRESERVE ROWS:** 슬라이드의 예제 2와 같이 PRESERVE ROWS 설정은 세션 관련 임시 테이블을 생성합니다. 영업 사원 세션마다 테이블에 해당 날짜의 영업 데이터를 저장할 수 있습니다. 영업 사원이 today_sales 테이블에 데이터를 처음 삽입할 때 해당 세션이 today_sales 테이블에 바인드되며, 세션이 끝나거나 세션에서 테이블의 TRUNCATE를 실행할 때 바인드가 해제됩니다. 세션을 종료할 때 데이터베이스가 테이블을 truncate합니다.

오라클 데이터베이스에 임시 테이블을 생성하면 정적 테이블 정의가 생성됩니다. 영구 테이블과 마찬가지로 임시 테이블은 데이터 딕셔너리에 정의됩니다. 그러나 임시 테이블과 해당 인덱스는 생성 시 자동으로 세그먼트를 할당하지 않습니다. 대신 데이터를 처음 삽입할 때 임시 세그먼트가 할당됩니다. 데이터가 세션에 로드될 때까지 테이블이 빈 상태로 나타납니다.

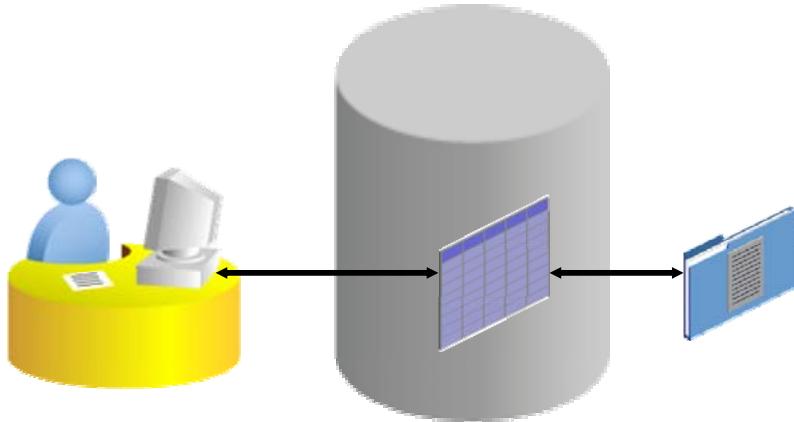
단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
 - 제약 조건 추가 및 삭제
 - 제약 조건 지연
 - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
 - CREATE TABLE 문 사용
 - 함수 기반 인덱스 생성
 - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

External Table



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

External Table

External Table은 메타 데이터가 데이터베이스에 저장되지만 데이터는 데이터베이스 외부에 저장되는 읽기 전용 테이블입니다. 이 External Table은 external 데이터를 먼저 데이터베이스에 로드할 필요 없이 external 데이터에 SQL query를 실행할 수 있는 뷰로 간주할 수 있습니다. External Table 데이터는 external 데이터를 먼저 데이터베이스에 로드할 필요 없이 직접 또는 명령어로 query하고 조인할 수 있습니다. SQL, PL/SQL 및 Java를 사용하여 External Table의 데이터를 query할 수 있습니다.

External Table과 일반 테이블의 주요 차이점은 external에서 구성된 테이블은 읽기 전용이라는 것입니다. 따라서 DML(데이터 조작어) 작업을 수행할 수 없으며 이러한 테이블에는 인덱스를 생성할 수 없습니다. 그러나 External Table을 생성할 수 있으므로 CREATE TABLE AS SELECT 명령을 사용하여 데이터를 언로드할 수 있습니다.

External Table(계속)

Oracle 서버는 External Table에 두 개의 기본 액세스 드라이버를 제공합니다. 하나는 로더 액세스 드라이버(또는 ORACLE_LOADER)로, SQL*Loader 유ти리티가 형식을 해석할 수 있는 external 파일의 데이터를 읽는 데 사용됩니다. External Table에서 모든 SQL*Loader 기능이 지원되는 것은 아닙니다. ORACLE_DATAPUMP 액세스 드라이버는 플랫폼 독립적인 형식을 사용하여 데이터를 임포트 및 эксп포트하는 데 사용할 수 있습니다. ORACLE_DATAPUMP 액세스 드라이버는 CREATE TABLE ... ORGANIZATION EXTERNAL...AS SELECT 문의 일부로 External Table로 로드할 SELECT 문의 행을 기록합니다. 그러면 유저가 SELECT를 사용하여 해당 데이터 파일에서 데이터를 읽을 수 있습니다. 또한 다른 시스템에서 External Table 정의를 생성하고 해당 데이터 파일을 사용할 수 있습니다. 이렇게 하면 오라클 데이터베이스 간에 데이터를 이동할 수 있습니다.

External Table의 디렉토리 생성

External 데이터 소스가 있는 파일 시스템의 디렉토리에
해당하는 DIRECTORY 객체를 생성합니다.

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/.../emp_dir';

GRANT READ ON DIRECTORY emp_dir TO ora_21;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

External Table 생성의 예

CREATE DIRECTORY 문을 사용하여 디렉토리 객체를 생성할 수 있습니다. 디렉토리 객체는 external 데이터 소스가 상주하는 서버의 파일 시스템에 있는 디렉토리에 대한 alias를 지정합니다. 파일 관리의 유연성을 높이기 위해 external 데이터 소스를 참조할 때 운영 체제 경로 이름을 하드 코딩하는 대신 디렉토리 이름을 사용할 수 있습니다.

디렉토리를 생성하려면 CREATE ANY DIRECTORY 시스템 권한이 있어야 합니다. 디렉토리를 생성하면 READ 및 WRITE 객체 권한이 자동으로 부여되고 READ 및 WRITE 권한을 다른 유저와 롤에 부여할 수 있습니다. DBA가 이러한 권한을 다른 유저와 롤에 부여할 수도 있습니다.

유저는 액세스할 External Table에서 사용되는 모든 디렉토리에 대해 READ 권한이 필요하고, 사용될 로그 파일, 손상된 파일 및 discard file 위치에 대해 WRITE 권한이 필요합니다.

또한 External Table 프레임워크가 데이터를 언로드하는 데 사용되고 있는 경우에도 WRITE 권한이 필요합니다.

오라클은 또한 데이터를 언로드(즉, 데이터베이스의 테이블에서 데이터를 읽고 External Table로 데이터를 삽입)한 다음 해당 데이터를 오라클 데이터베이스에 다시 로드할 수 있는 ORACLE_DATAPUMP 유형을 제공합니다. 이 작업은 테이블이 생성될 때 한 번만 수행할 수 있는 작업입니다. 생성 및 최초의 채우기가 수행된 후에는 행을 갱신, 삽입 또는 삭제할 수 없습니다.

External Table 생성의 예(계속)

구문

```
CREATE [OR REPLACE] DIRECTORY AS 'path_name' ;
```

이 구문에서 다음이 적용됩니다.

OR REPLACE	이미 디렉토리 데이터베이스 객체가 있는 경우 다시 생성하려면 OR REPLACE를 지정합니다. 이 절을 사용하여 디렉토리에 이전에 부여된 데이터베이스 객체 권한을 삭제, 재생성 및 재부여하지 않고 기존 디렉토리 정의를 변경할 수 있습니다. 재정의된 디렉토리에 대해 이전에 권한을 부여받은 유저는 권한을 다시 부여받지 않아도 이 디렉토리에 계속 액세스할 수 있습니다.
directory	생성할 디렉토리 객체의 이름을 지정합니다. 디렉토리 이름의 최대 길이는 30바이트입니다. 디렉토리 객체에 스키마 이름을 사용할 수 없습니다.
'path_name'	액세스할 운영 체제 디렉토리의 전체 경로 이름을 지정합니다. 경로 이름은 대소문자를 구분합니다.

External Table 생성

```

CREATE TABLE <table_name>
  ( <col_name> <datatype>, ... )
ORGANIZATION EXTERNAL
  (TYPE <access_driver_type>
   DEFAULT DIRECTORY <directory_name>
   ACCESS PARAMETERS
   (... ) )
   LOCATION ('<locationSpecifier>')
REJECT LIMIT [0 | <number> | UNLIMITED];

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

External Table 생성

CREATE TABLE 문의 ORGANIZATION EXTERNAL 절을 사용하여 External Table을 생성합니다. 실제로는 테이블을 생성하는 것이 아니라 external 데이터에 액세스할 때 사용할 수 있도록 데이터 딕셔너리에 메타 데이터를 생성하는 것입니다. ORGANIZATION 절을 사용하여 테이블의 데이터 행이 저장되는 순서를 지정합니다. ORGANIZATION 절에 EXTERNAL을 지정하여 테이블이 데이터베이스 외부에 있는 읽기 전용 테이블임을 나타냅니다. External 파일이 이미 데이터베이스 외부에 존재해야 합니다.

TYPE <access_driver_type>은 External Table의 액세스 드라이버를 나타냅니다. 액세스 드라이버는 데이터베이스의 external 데이터를 해석하는 API(응용 프로그램 프로그래밍 인터페이스)입니다. TYPE을 지정하지 않으면 오라클은 기본 액세스 드라이버인 ORACLE_LOADER를 사용합니다. 다른 옵션은 ORACLE_DATAPUMP입니다.

DEFAULT DIRECTORY 절을 사용하여 external 데이터 소스가 상주할 수 있는 파일 시스템의 디렉토리에 해당하는 여러 오라클 데이터베이스 디렉토리 객체를 지정할 수 있습니다.

선택 사항인 ACCESS PARAMETERS 절을 사용하여 이 External Table에 대한 특정 액세스 드라이버의 파라미터에 값을 할당할 수 있습니다.

External Table 생성(계속)

LOCATION 절을 사용하여 각 external 데이터 소스에 대해 하나의 external 위치 지정자를 지정할 수 있습니다. 일반적으로 <*location_specifier*>는 파일이지만 반드시 그럴 필요는 없습니다.

REJECT LIMIT 절을 사용하면 Oracle 오류가 반환되고 query가 중단되기 전, external 데이터 query 중에 발생할 수 있는 변환 오류 횟수를 지정할 수 있습니다. 기본값은 0입니다.

ORACLE_DATAPUMP 액세스 드라이버를 사용하는 구문은 다음과 같습니다.

```
CREATE TABLE extract_emps
ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP
                         DEFAULT DIRECTORY ...
                         ACCESS PARAMETERS ( ... )
                         LOCATION ( ... )
                         PARALLEL 4
                         REJECT LIMIT UNLIMITED
AS
SELECT * FROM ...;
```

ORACLE_LOADER를 사용하여 External Table 생성

```

CREATE TABLE oldemp (
    fname char(25), lname CHAR(25))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
DEFAULT DIRECTORY emp_dir
ACCESS PARAMETERS
(RECORDS DELIMITED BY NEWLINE
NOBADFILE
NOLOGFILE
FIELDS TERMINATED BY ','
(fname POSITION ( 1:20) CHAR,
 lname POSITION (22:41) CHAR))
LOCATION ('emp.dat'))
PARALLEL 5
REJECT LIMIT 200;

```

CREATE TABLE succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ORACLE_LOADER 액세스 드라이버를 사용한 External Table 생성의 예

다음 형식의 레코드가 있는 플랫 파일이 있다고 가정합니다.

```

10,jones,11-Dec-1934
20,smith,12-Jun-1972

```

레코드는 개행 문자로 구분되며 필드는 모두 쉼표(,)로 종료됩니다. 파일 이름은 /emp_dir/emp.dat입니다.

이 파일을 메타 데이터가 데이터베이스에 상주하는 External Table의 데이터 소스로 변환하려면 다음 단계를 수행해야 합니다.

1. 다음과 같이 디렉토리 객체 emp_dir을 생성합니다.

```
CREATE DIRECTORY emp_dir AS '/emp_dir' ;
```

2. 슬라이드에 표시된 CREATE TABLE 명령을 실행합니다.

위 슬라이드 예제는 파일에 대한 External table을 생성하는 테이블 사양을 보여줍니다.

```
/emp_dir/emp.dat
```

ORACLE_LOADER 액세스 드라이버를 사용한 External Table 생성의 예(계속)

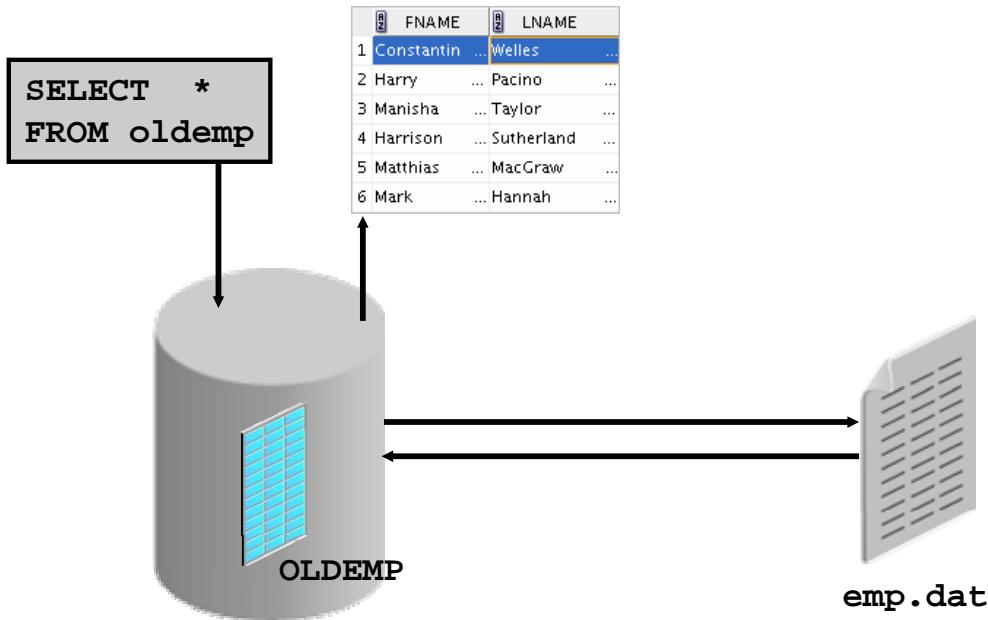
예제에서 TYPE 사양은 단지 그 용도를 보여주기 위해 제시된 것입니다. 액세스 드라이버를 지정하지 않은 경우 ORACLE_LOADER가 기본값으로 사용됩니다. ACCESS PARAMETERS 옵션은 특정 액세스 드라이버의 파라미터에 값을 제공하며, 이 값은 Oracle 서버가 아닌 액세스 드라이버에 의해 해석됩니다.

PARALLEL 절을 사용하면 INSERT INTO TABLE 문을 실행할 때 다섯 개의 병렬 실행 서버에서 동시에 external 데이터 소스(파일)를 스캔할 수 있습니다. 예를 들어, PARALLEL=5로 지정된 경우 한 대 이상의 병렬 실행 서버가 데이터 소스에 작업을 수행할 수 있습니다. External Table이 아주 클 수 있으므로 성능 문제를 고려하여 PARALLEL 절을 지정하거나 query에 대해 병렬 힌트를 지정하는 것이 좋습니다.

REJECT LIMIT 절은 external 데이터를 query하는 동안 200개가 넘는 변환 오류가 발생하면 query를 중지하고 오류를 반환하도록 지정합니다. 이러한 변환 오류는 액세스 드라이버가 데이터 파일의 데이터를 External Table 정의에 일치하도록 변형시키려고 할 때 발생할 수 있습니다.

CREATE TABLE 명령이 성공적으로 실행된 후에는 OLDEMP External Table을 관계형 테이블과 같은 방식으로 나타내고 query할 수 있습니다.

External Table 조회



ORACLE

Copyright © 2009, Oracle. All rights reserved.

External Table 조회

External Table은 데이터베이스에 저장되어 있는 데이터를 설명하지 않습니다. external 소스에 데이터를 저장하는 방법도 설명하지 않습니다. 대신 External Table 층에서 서버에 데이터를 어떻게 표시해야 하는지를 나타냅니다. External Table 정의에 맞게 데이터 파일의 데이터에 필요한 변형을 수행하는 작업은 액세스 드라이버와 External Table 층에서 담당합니다.

데이터베이스 서버에서 external 소스의 데이터에 액세스하는 경우 해당 액세스 드라이버를 호출하여 데이터베이스 서버에서 읽을 수 있는 형태로 external 소스의 데이터를 가져옵니다.

데이터 소스의 데이터 설명은 External Table 정의와 별개라는 점에 유의하십시오. 소스 파일에는 테이블의 열 수보다 많거나 적은 필드 수가 포함될 수 있습니다. 또한 데이터 소스의 필드에 대한 데이터 유형이 테이블의 열과 다를 수도 있습니다. 액세스 드라이버는 데이터 소스의 데이터가 External Table 정의와 일치하도록 처리되는지 확인합니다.

ORACLE_DATAPUMP를 사용하여 External Table 생성: 예제

```

CREATE TABLE emp_ext
(employee_id, first_name, last_name)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY emp_dir
  LOCATION
  ( 'emp1.exp', 'emp2.exp' )
)
PARALLEL
AS
SELECT employee_id, first_name, last_name
FROM employees;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ORACLE_DATAPUMP를 사용하여 External Table 생성: 예제

ORACLE_DATAPUMP 액세스 드라이버를 사용하여 External Table 관련 작업을 언로드 및 재로드할 수 있습니다.

참고: External Table 컨텍스트에서 데이터 로딩은 External Table에서 데이터가 읽혀져서 데이터베이스의 테이블로 로드되는 작업을 말합니다. 데이터 언로딩은 데이터베이스의 테이블에서 데이터를 읽어 External Table로 데이터를 삽입하는 작업을 말합니다.

위 슬라이드 예제는 ORACLE_DATAPUMP 액세스 드라이버를 사용하여 External Table을 생성하기 위한 테이블 사양을 보여줍니다. 데이터가 emp1.exp 파일과 emp2.exp 파일에 채워집니다.

EMPLOYEES 테이블에서 External Table로 읽어 들인 데이터를 채우려면 다음 단계를 수행해야 합니다.

1. 다음과 같이 디렉토리 객체 emp_dir을 생성합니다.

```
CREATE DIRECTORY emp_dir AS '/emp_dir' ;
```

2. 슬라이드에 표시된 CREATE TABLE 명령을 실행합니다.

참고: emp_dir 디렉토리는 ORACLE_LOADER를 사용하는 이전 예에서 생성된 것과 동일합니다.

다음 코드를 실행하여 External Table을 query 할 수 있습니다.

```
SELECT * FROM emp_ext;
```

퀴즈

FOREIGN KEY 제약 조건을 사용할 경우 상위 키의 데이터가
삭제되면 해당 상위 키 값에 종속된 하위 테이블의 모든 행도
삭제됩니다.

- 1. 맞습니다.**
- 2. 틀립니다.**



Copyright © 2009, Oracle. All rights reserved.

정답: 2

퀴즈

DROP TABLE 명령을 실행하면 항상 데이터베이스가 테이블의 이름을 바꾸고 테이블을 Recycle bin에 넣습니다. 나중에 FLASHBACK TABLE 문을 사용하여 Recycle bin에서 테이블을 recovery할 수 있습니다.

1. 맞습니다.
2. 틀립니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 2

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 제약 조건 추가
- 인덱스 생성
- CREATE TABLE 문을 사용하여 인덱스 생성
- 함수 기반(Function-based) 인덱스 생성
- 열 삭제 및 UNUSED 열로 설정
- FLASHBACK 작업 수행
- External Table 생성 및 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 스키마 객체 관리를 위해 다음 작업을 수행하는 방법을 배웠습니다.

- 테이블을 변경하여 열 또는 제약 조건을 추가하거나 수정합니다.
- CREATE INDEX 문을 사용하여 인덱스 및 함수 기반 인덱스를 생성합니다.
- unused 열을 삭제합니다.
- FLASHBACK 방식을 사용하여 테이블을 복원합니다.
- CREATE TABLE 문의 ORGANIZATION EXTERNAL 절을 사용하여 External Table을 생성합니다. External Table은 메타 데이터가 데이터베이스에 저장되지만 데이터는 데이터베이스 외부에 저장되는 읽기 전용 테이블입니다.
- 데이터를 데이터베이스에 먼저 로드하지 않고 External Table을 사용하여 데이터를 query합니다.
- CREATE TABLE 문으로 테이블을 생성할 때 PRIMARY KEY 열 인덱스의 이름을 지정합니다.

연습 2: 개요

이 연습에서는 다음 내용을 다룹니다.

- 테이블 변경
- 열 추가
- 열 삭제
- 인덱스 생성
- External Table 생성

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 2: 개요

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.

데이터 딕셔너리 뷰를 사용하여 객체 관리

3

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 데이터 딕셔너리 뷰를 사용하여 객체의 데이터 검색
- 다양한 데이터 딕셔너리 뷰 조회

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 데이터 딕셔너리 뷰를 소개합니다. 데이터 딕셔너리 뷰를 사용하여 메타 데이터를 검색하고 스키마 객체에 대한 보고서를 생성하는 방법을 배웁니다.

단원 내용

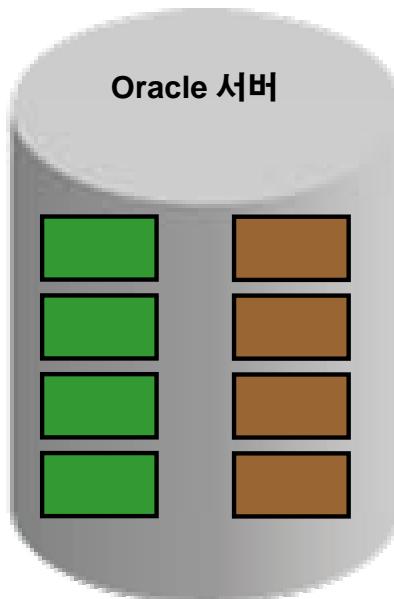
- 데이터 딕셔너리 소개
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 테이블 정보
 - 열 정보
 - 제약 조건 정보
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 뷰 정보
 - 시퀀스 정보
 - 동의어 정보
 - 인덱스 정보
- 테이블에 주석을 추가하고 딕셔너리 뷰를 query하여 주석 정보 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리

업무 데이터를 포함하는 테이블:
EMPLOYEES
DEPARTMENTS
LOCATIONS
JOB_HISTORY
...



데이터 딕셔너리 뷰:
DICTIONARY
USER_OBJECTS
USER_TABLES
USER_TAB_COLUMNS
...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리

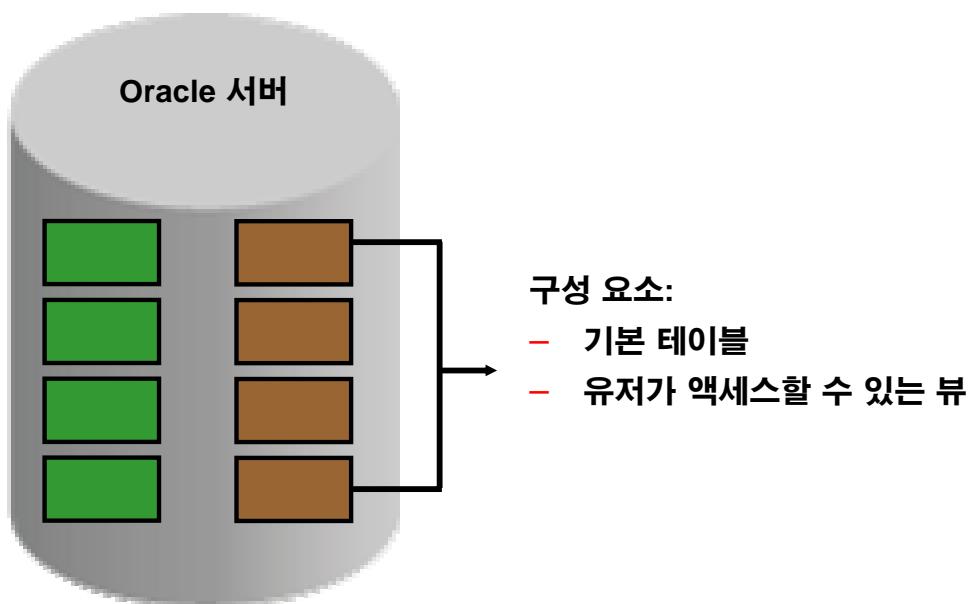
EMPLOYEES와 같은 유저 테이블은 유저가 생성하는 테이블로 업무 데이터를 포함합니다. 오라클 데이터베이스에는 데이터 딕셔너리로 알려진 또 다른 테이블 및 뷰 모음이 있습니다. 이 모음은 Oracle 서버에 의해 생성되고 유지 관리되며 데이터베이스에 대한 정보를 포함합니다. 데이터 딕셔너리는 다른 데이터베이스 데이터처럼 테이블 및 뷰로 구조화됩니다. 데이터 딕셔너리는 모든 오라클 데이터베이스의 핵심일 뿐만 아니라 일반 유저에서 응용 프로그램 설계자나 데이터베이스 관리자에 이르는 모든 유저에게 중요한 도구입니다.

SQL 문을 사용하여 데이터 딕셔너리에 액세스할 수 있습니다. 데이터 딕셔너리는 읽기 전용이기 때문에 해당 테이블과 뷰에 대해 query만 실행할 수 있습니다.

딕셔너리 테이블을 기반으로 하는 딕셔너리 뷰를 query하여 다음과 같은 정보를 찾을 수 있습니다.

- 데이터베이스의 모든 스키마 객체(테이블, 뷰, 인덱스, 동의어, 시퀀스, 프로시저, 함수, 패키지, 트리거 등)의 정의
- 열의 기본값
- 무결성 제약 조건 정보
- Oracle 유저 이름
- 각 유저에게 부여된 권한 및 롤
- 기타 일반 데이터베이스 정보

데이터 딕셔너리 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리 구조

기본 테이블은 연관된 데이터베이스에 대한 정보를 저장합니다. Oracle 서버만 이러한 테이블에 대해 읽기/쓰기를 수행할 수 있습니다. 유저가 직접 테이블에 액세스하는 경우는 드뭅니다.

데이터 딕셔너리의 기본 테이블에 저장된 정보를 요약하고 표시하는 여러 뷰가 있습니다. 이러한 뷰는 정보를 단순화하는 조인 및 WHERE 절을 사용하여 기본 테이블 데이터를 유저 또는 테이블 이름과 같은 유용한 정보로 해석합니다. 대부분의 유저는 기본 테이블이 아니라 뷰에 액세스합니다.

Oracle 유저 SYS는 데이터 딕셔너리의 모든 기본 테이블과 유저가 액세스할 수 있는 뷰를 소유합니다. 데이터 무결성이 손상될 수도 있기 때문에 Oracle 유저는 SYS 스키마에 포함된 행이나 스키마 객체를 변경(UPDATE, DELETE 또는 INSERT) 해서는 안됩니다.

데이터 딕셔너리 구조

뷰 이름 지정 규칙:

뷰 접두어	목적
USER	유저의 뷰(유저의 스키마에 있는 내용, 유저가 소유한 내용)
ALL	확장된 유저의 뷰(유저가 액세스할 수 있는 내용)
DBA	데이터베이스 관리자의 뷰(모든 사람의 스키마에 있는 내용)
V\$	성능 관련 데이터

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 딕셔너리 구조(계속)

데이터 딕셔너리는 뷰 집합으로 구성됩니다. 대부분의 경우 집합은 유사한 정보를 포함하고 접두어를 통해 서로 구분되는 세 개의 뷰로 구성됩니다. 예를 들어, USER_OBJECTS, ALL_OBJECTS 및 DBA_OBJECTS라는 뷰가 있을 수 있습니다.

이러한 세 개의 뷰는 범위가 다르다는 점을 제외하고 데이터베이스의 객체에 대한 유사한 정보를 포함합니다. USER_OBJECTS는 유저가 소유하거나 생성한 객체에 대한 정보를 포함합니다. ALL_OBJECTS는 유저가 액세스 권한이 있는 모든 객체에 대한 정보를 포함합니다. DBA_OBJECTS는 모든 유저가 소유한 모든 객체에 대한 정보를 포함합니다. ALL 또는 DBA 접두어가 붙은 뷰의 경우 대개 객체를 소유한 사람을 식별하기 위해 뷰에 OWNER라는 추가 열이 있습니다.

v\$ 접두어가 붙은 뷰 집합도 있습니다. 이러한 뷰는 특성상 동적이며 성능에 대한 정보를 보유합니다. 동적 성능 테이블은 실제 테이블이 아니며 대부분의 유저가 액세스할 수 없어야 합니다. 그러나 데이터베이스 관리자는 테이블에서 뷰를 query하거나 생성하고 다른 유저에게 이러한 뷰에 대한 액세스 권한을 부여할 수 있습니다. 본 과정에서는 이러한 뷰에 대해 자세히 다루지 않습니다.

딕셔너리 뷰 사용 방법

DICTIONARY로 시작합니다. 딕셔너리 테이블 및 뷰의 이름과 설명을 포함합니다.

DESCRIBE DICTIONARY

Name	Null	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)
2 rows selected		

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

TABLE_NAME	COMMENTS
1 USER_OBJECTS	Objects owned by the user

ORACLE

Copyright © 2009, Oracle. All rights reserved.

딕셔너리 뷰 사용 방법

딕셔너리 뷰에 익숙해지려면 DICTIONARY라는 딕셔너리 뷰를 사용하면 됩니다. 여기에는 액세스 권한이 있는 각 딕셔너리 뷰의 이름과 간단한 설명이 포함되어 있습니다.

특정 뷰 이름에 대한 정보를 검색하는 query를 작성하거나 COMMENTS 열에서 특정 단어나 구문을 검색할 수 있습니다. 위의 예제에는 DICTIONARY 뷰가 설명되어 있습니다. 이 뷰에는 두 개의 열이 있습니다. SELECT 문은 USER_OBJECTS라는 딕셔너리 뷰에 대한 정보를 검색합니다. USER_OBJECTS 뷰는 유저가 소유한 모든 객체에 대한 정보를 포함합니다.

COMMENTS 열에서 특정 단어나 구문을 검색하는 query를 작성할 수 있습니다. 예를 들어, 다음 query는 COMMENTS 열에 *columns*라는 단어가 포함된 액세스 권한이 있는 모든 뷰의 이름을 반환합니다.

```
SELECT table_name
FROM   dictionary
WHERE  LOWER(comments) LIKE '%columns%';
```

참고: 데이터 딕셔너리의 이름은 대문자입니다.

USER_OBJECTS 및 ALL_OBJECTS 뷰

USER_OBJECTS:

- USER_OBJECTS를 query하여 자신이 소유한 모든 객체를 볼 수 있습니다.
- USER_OBJECTS를 사용하면 유저의 스키마에 있는 모든 객체 이름 및 유형 리스트와 함께 다음 정보를 얻을 수 있습니다.
 - 생성된 날짜
 - 마지막 수정 날짜
 - 상태 (valid 또는 invalid)

ALL_OBJECTS:

- ALL_OBJECTS를 query하여 액세스 권한이 있는 모든 객체를 봅니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

USER_OBJECTS 및 ALL_OBJECTS 뷰

USER_OBJECTS 뷰를 query하여 유저의 스키마에 있는 모든 객체의 이름과 유형을 볼 수 있습니다. 이 뷰에는 다음과 같은 여러 열이 있습니다.

- **OBJECT_NAME:** 객체 이름입니다.
- **OBJECT_ID:** 객체의 딕셔너리 객체 번호
- **OBJECT_TYPE:** 객체의 유형(TABLE, VIEW, INDEX, SEQUENCE 등)
- **CREATED:** 객체가 생성된 시간 기록
- **LAST_DDL_TIME:** DDL(데이터 정의어) 명령으로 인해 객체가 마지막으로 수정된 시간 기록
- **STATUS:** 객체의 상태(VALID, INVALID 또는 N/A)
- **GENERATED:** 이 객체의 이름을 시스템이 생성했는지 여부 (Y | N)

참고: 여기에는 일부 열만 나열되어 있습니다. 전체 리스트는 *Oracle Database Reference*의 "USER_OBJECTS"를 참조하십시오.

ALL_OBJECTS 뷰를 query하여 액세스 권한이 있는 모든 객체 리스트를 볼 수도 있습니다.

USER_OBJECTS 뷰

```
SELECT object_name, object_type, created, status  
FROM user_objects  
ORDER BY object_type;
```

OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
1 LOC_COUNTRY_IX	INDEX	19-MAY-09	VALID

...

53 EMPLOYEES2	TABLE	22-MAY-09	VALID
54 SECURE_EMPLOYEES	TRIGGER	19-MAY-09	VALID
55 UPDATE_JOB_HISTORY	TRIGGER	19-MAY-09	VALID
56 EMP_DETAILS_VIEW	VIEW	19-MAY-09	VALID

...

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

USER_OBJECTS 뷰

예제는 이 유저가 소유한 모든 객체의 이름, 유형, 생성 날짜 및 상태를 보여줍니다.

OBJECT_TYPE 열은 TABLE, VIEW, SEQUENCE, INDEX, PROCEDURE, FUNCTION, PACKAGE 또는 TRIGGER 값 중 하나를 보유합니다.

STATUS 열은 VALID, INVALID 또는 N/A 값을 가집니다. 테이블은 항상 유효하지만 뷰, 프로시저, 함수, 패키지 및 트리거는 유효하지 않을 수도 있습니다.

CAT 뷰

간단한 query 및 출력의 경우 CAT 뷰를 query할 수 있습니다. 이 뷰는 TABLE_NAME 및 TABLE_TYPE의 두 열만 포함합니다. 유저의 모든 INDEX, TABLE, CLUSTER, VIEW, SYNONYM, SEQUENCE 또는 UNDEFINED 객체의 이름을 제공합니다.

참고: CAT는 유저가 소유한 테이블, 뷰, 동의어 및 시퀀스를 나열하는 뷰인 USER_CATALOG의 동의어입니다.

단원 내용

- 데이터 딕셔너리 소개
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 테이블 정보
 - 열 정보
 - 제약 조건 정보
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 뷰 정보
 - 시퀀스 정보
 - 동의어 정보
 - 인덱스 정보
- 테이블에 주석을 추가하고 딕셔너리 뷰를 query하여 주석 정보 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블 정보

USER_TABLES:

```
DESCRIBE user_tables
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

```
SELECT table_name
FROM user_tables;
```

TABLE_NAME
1 REGIONS
2 LOCATIONS
3 DEPARTMENTS
4 JOBS
5 EMPLOYEES
6 JOB_HISTORY

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 정보

USER_TABLES 뷰를 사용하여 모든 테이블의 이름을 확인할 수 있습니다. USER_TABLES 뷰는 유저의 테이블에 대한 정보를 포함합니다. 테이블 이름을 제공하는 것 외에도 저장 영역에 대한 자세한 정보를 포함합니다.

TABS 뷰는 USER_TABLES 뷰의 동의어입니다. 이 뷰를 query하여 유저가 소유한 테이블 리스트를 볼 수 있습니다.

```
SELECT table_name
FROM tabs;
```

참고: USER_TABLES 뷰의 전체 열 리스트는 *Oracle Database Reference*의 "USER_TABLES"를 참조하십시오.

ALL_TABLES 뷰를 query하여 액세스 권한이 있는 모든 테이블 리스트를 볼 수도 있습니다.

열 정보

USER_TAB_COLUMNS:

```
DESCRIBE user_tab_columns
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

열 정보

USER_TAB_COLUMNS 뷰를 query하면 테이블의 열에 대한 상세 정보를 찾을 수 있습니다.
 USER_TABLES 뷰는 테이블 이름 및 저장 영역에 대한 정보를 제공하지만 자세한 열 정보는
 USER_TAB_COLUMNS 뷰에서 찾을 수 있습니다.

이 뷰는 다음과 같은 정보를 포함합니다.

- 열 이름
- 열 데이터 유형
- 데이터 유형의 길이
- NUMBER 열의 전체 자릿수 및 소수점 이하 자릿수
- 널 허용 여부(열에 NOT NULL 제약 조건이 있는지 여부)
- 기본값

참고: USER_TAB_COLUMNS 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference*의 "USER_TAB_COLUMNS"를 참조하십시오.

열 정보

```
SELECT column_name, data_type, data_length,
       data_precision, data_scale, nullable
  FROM user_tab_columns
 WHERE table_name = 'EMPLOYEES';
```

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION
1 EMPLOYEE_ID	NUMBER	22	6
2 FIRST_NAME	VARCHAR2	20	(null)
3 LAST_NAME	VARCHAR2	25	(null)
4 EMAIL	VARCHAR2	25	(null)
5 PHONE_NUMBER	VARCHAR2	20	(null)
6 HIRE_DATE	DATE	7	(null)
7 JOB_ID	VARCHAR2	10	(null)
8 SALARY	NUMBER	22	8
9 COMMISSION_PCT	NUMBER	22	2
10 MANAGER_ID	NUMBER	22	6
11 DEPARTMENT_ID	NUMBER	22	4

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

열 정보(계속)

USER_TAB_COLUMNS 테이블을 query하면 열의 이름, 데이터 유형, 데이터 유형 길이, null 제약 조건 및 기본값 등 열에 대한 상세 정보를 찾을 수 있습니다.

위의 예제는 EMPLOYEES 테이블의 열, 데이터 유형, 데이터 길이 및 널 제약 조건을 표시합니다. 이 정보는 DESCRIBE 명령의 출력과 유사합니다.

unused로 설정된 열에 대한 정보를 보려면 USER_UNUSED_COL_TABS 딕셔너리 뷰를 사용하십시오.

참고: 데이터 딕셔너리의 객체 이름은 대문자입니다.

제약 조건 정보

- **USER_CONSTRAINTS**는 유저의 테이블에 있는 제약 조건 정의를 설명합니다.
- **USER_CONS_COLUMNS**는 유저가 소유하고 제약 조건에 지정된 열에 대해 설명합니다.

DESCRIBE user_constraints

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG()
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

제약 조건 정보

제약 조건의 이름, 제약 조건의 유형, 제약 조건이 적용되는 테이블 이름, check 제약 조건에 대한 조건, Foreign key 제약 조건 정보, Foreign key 제약 조건에 대한 삭제 규칙, 상태 및 기타 다양한 유형의 제약 조건 정보를 찾을 수 있습니다.

참고: USER_CONSTRAINTS 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference*의 "USER_CONSTRAINTS"를 참조하십시오.

USER_CONSTRAINTS: 예제

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
  FROM user_constraints
 WHERE table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	C...	SEARCH_CONDITION	R_CONSTR...	DELET...	STATUS
1	EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL	(null)	(null)	ENABLED
2	EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL	(null)	(null)	ENABLED
3	EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL	(null)	(null)	ENABLED
4	EMP_JOB_NN	C	"JOB_ID" IS NOT NULL	(null)	(null)	ENABLED
5	EMP_SALARY_MIN	C	salary > 0	(null)	(null)	ENABLED
6	EMP_EMAIL_UK	U	(null)	(null)	(null)	ENABLED
7	EMP_EMP_ID_PK	P	(null)	(null)	(null)	ENABLED
8	EMP_DEPT_FK	R	(null)	DEPT_ID_PK	NO ACTION	ENABLED
9	EMP_JOB_FK	R	(null)	JOB_ID_PK	NO ACTION	ENABLED
10	EMP_MANAGER_FK	R	(null)	EMP_EMP_ID_PK	NO ACTION	ENABLED

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

USER_CONSTRAINTS: 예제

위의 예제에서는 USER_CONSTRAINTS 뷰를 query하여 EMPLOYEES 테이블의 제약 조건에 대한 이름, 유형, check 조건, Foreign key가 참조하는 unique 제약 조건의 이름, Foreign key에 대한 삭제 규칙 및 상태를 찾습니다.

CONSTRAINT_TYPE은 다음 값이 될 수 있습니다.

- C (테이블의 check 제약 조건 또는 NOT NULL)
- P (Primary key)
- U (Unique key)
- R (참조 무결성)
- V (뷰에서 check 옵션으로 사용)
- O (뷰에서 읽기 전용으로 사용)

DELETE_RULE은 다음 값이 될 수 있습니다.

- **CASCADE:** 상위 레코드가 삭제되면 하위 레코드도 삭제됩니다.
- **SET NULL:** 상위 레코드가 삭제되면 해당 하위 레코드가 널로 변경됩니다.
- **NO ACTION:** 하위 레코드가 없는 경우에만 상위 레코드를 삭제할 수 있습니다.

STATUS는 다음 값이 될 수 있습니다.

- **ENABLED:** 제약 조건이 활성화됩니다.
- **DISABLED:** 제약 조건이 비활성화됩니다.

USER_CONS_COLUMNS 조회

```
DESCRIBE user_cons_columns
```

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

#	CONSTRAINT_NAME	COLUMN_NAME
1	EMP_LAST_NAME_NN	LAST_NAME
2	EMP_EMAIL_NN	EMAIL
3	EMP_HIRE_DATE_NN	HIRE_DATE
4	EMP_JOB_NN	JOB_ID
5	EMP_SALARY_MIN	SALARY
6	EMP_EMAIL_UK	EMAIL

Copyright © 2009, Oracle. All rights reserved.

ORACLE

USER_CONS_COLUMNS 조회

제약 조건이 적용된 열 이름을 찾으려면 USER_CONS_COLUMNS 덕셔너리 뷰를 query하십시오. 이 뷰는 객체 정의에서 제약 조건 소유자의 이름, 제약 조건의 이름, 제약 조건이 있는 테이블, 제약 조건을 가진 열의 이름 및 열이나 속성의 원래 위치를 알려줍니다.

참고: 제약 조건은 두 개 이상의 열에도 적용할 수 있습니다.

또한 USER_CONSTRAINTS와 USER_CONS_COLUMNS 사이에 조인을 작성하여 두 테이블에서 커스터마이즈된 출력을 생성할 수 있습니다.

단원 내용

- 데이터 딕셔너리 소개
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 테이블 정보
 - 열 정보
 - 제약 조건 정보
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 뷰 정보
 - 시퀀스 정보
 - 동의어 정보
 - 인덱스 정보
- 테이블에 주석을 추가하고 딕셔너리 뷰를 query하여 주석 정보 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

뷰 정보

1

DESCRIBE user_views

Name	Null	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

2

SELECT view_name FROM user_views;

VIEW_NAME
1 EMP_DETAILS_VIEW

3

**SELECT text FROM user_views
WHERE view_name = 'EMP_DETAILS_VIEW';**

TEXT
1 SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co
...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

뷰 정보

뷰를 생성한 후에 USER_VIEWS라는 데이터 딕셔너리 뷰를 query하여 뷰의 이름과 뷰 정의를 볼 수 있습니다. 뷰를 구성하는 SELECT 문의 텍스트는 LONG 열에 저장됩니다. LENGTH 열은 SELECT 문의 문자 수입니다. 기본적으로, LONG 열에서 선택할 때 열 값의 처음 80자만 표시됩니다. SQL*Plus에서 80자 이상을 표시하려면 SET LONG 명령을 사용하십시오.

SET LONG 1000

슬라이드 예제 설명:

1. USER_VIEWS 열이 표시됩니다. 이것은 부분 리스트입니다.
2. 뷰의 이름을 검색합니다.
3. 딕셔너리에서 EMP_DETAILS_VIEW의 SELECT 문이 표시됩니다.

뷰를 사용하여 데이터 액세스

뷰를 사용하여 데이터에 액세스할 때 Oracle 서버는 다음 작업을 수행합니다.

- USER_VIEWS 데이터 딕셔너리 테이블에서 뷰 정의를 검색합니다.
- 뷰 기본 테이블의 액세스 권한을 검사합니다.
- 이렇게 하면 뷔 query는 기본 테이블의 해당 작업으로 변환됩니다. 즉, 기본 테이블에서 데이터를 검색하거나 기본 테이블을 갱신합니다.

시퀀스 정보

DESCRIBE user_sequences

Name	Null	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

ORACLE

Copyright © 2009, Oracle. All rights reserved.

시퀀스 정보

USER_SEQUENCES 뷰는 유저가 소유한 모든 시퀀스를 설명합니다. 시퀀스를 생성할 때 USER_SEQUENCES 뷰에 저장되는 조건을 지정합니다. 이 뷰의 열은 다음과 같습니다.

- **SEQUENCE_NAME:** 시퀀스의 이름
- **MIN_VALUE:** 시퀀스의 최소값
- **MAX_VALUE:** 시퀀스의 최대값
- **INCREMENT_BY:** 시퀀스의 증분값
- **CYCLE_FLAG:** 시퀀스가 제한에 도달하면 래핑되는지 여부
- **ORDER_FLAG:** 시퀀스 번호가 순서대로 생성되는지 여부
- **CACHE_SIZE:** 캐시할 시퀀스 번호 수
- **LAST_NUMBER:** 디스크에 기록된 마지막 시퀀스 번호. 시퀀스가 캐시를 사용하는 경우 디스크에 기록된 번호는 시퀀스 캐시에 배치된 마지막 번호입니다. 이 번호는 대개 사용된 마지막 시퀀스 번호보다 큽니다.

시퀀스 확인

- USER_SEQUENCES 데이터 딕셔너리 테이블에서 시퀀스 값을 확인합니다.

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

	SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
1	DEPARTMENTS_SEQ	1	9990	10	280
2	EMPLOYEES_SEQ	1	9999999999999999...	1	207
3	LOCATIONS_SEQ	1	9900	100	3300

- NOCACHE가 지정된 경우 LAST_NUMBER 열은 사용 가능한 다음 시퀀스를 표시합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시퀀스 확인

시퀀스를 생성하면 데이터 딕셔너리에 해당 시퀀스가 기록됩니다. 시퀀스는 데이터베이스 객체이므로 USER_OBJECTS 데이터 딕셔너리 테이블에서 식별할 수 있습니다.

USER_SEQUENCES 데이터 딕셔너리 뷰에서 시퀀스를 선택하여 시퀀스 설정을 확인할 수도 있습니다.

시퀀스를 증분하지 않고 사용 가능한 다음 시퀀스 값 확인

NOCACHE로 시퀀스를 생성한 경우 USER_SEQUENCES 테이블을 query하여 시퀀스를 증가시키지 않고도 사용 가능한 다음 시퀀스 값을 확인할 수 있습니다.

인덱스 정보

- **USER_INDEXES**는 인덱스에 대한 정보를 제공합니다.
- **USER_IND_COLUMNS**는 인덱스로 구성된 열과 테이블의 인덱스 열에 대해 설명합니다.

```
DESCRIBE user_indexes
```

Name	Null	Type
INDEX_NAME	NOT NULL	VARCHAR2(30)
INDEX_TYPE		VARCHAR2(27)
TABLE_OWNER	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLE_TYPE		VARCHAR2(11)
UNIQUENESS		VARCHAR2(9)

...

Copyright © 2009, Oracle. All rights reserved.

인덱스 정보

USER_INDEXES 뷰를 query하면 인덱스 이름, 인덱스가 생성된 테이블 이름 및 고유 인덱스인지 여부를 확인할 수 있습니다.

참고: USER_INDEXES 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference 11g Release 2(11.1)*의 "USER_INDEXES"를 참조하십시오.

USER_INDEXES: 예제

a

```
SELECT index_name, table_name, uniqueness
FROM   user_indexes
WHERE  table_name = 'EMPLOYEES';
```

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_EMAIL_UK	EMPLOYEES	UNIQUE
2	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
3	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
4	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
5	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
6	EMP_NAME_IX	EMPLOYEES	NONUNIQUE

b

```
SELECT index_name, table_name
FROM   user_indexes
WHERE  table_name = 'emp_lib';
```

	INDEX_NAME	TABLE_NAME
1	SYS_C0011777	EMP_LIB

ORACLE

Copyright © 2009, Oracle. All rights reserved.

USER_INDEXES: 예제

위 슬라이드 예제 a에서 USER_INDEXES 뷰를 query하면 인덱스 이름, 인덱스가 생성된 테이블 이름 및 고유 인덱스인지 여부를 확인할 수 있습니다.

위 슬라이드 예제 b에서 Oracle 서버는 PRIMARY KEY 열에 대해 생성되는 인덱스에 일반적인 이름을 지정합니다. EMP_LIB 테이블은 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE EMP_LIB
  (book_id NUMBER(6)PRIMARY KEY ,
   title VARCHAR2(25),
   category VARCHAR2(20));
```

CREATE TABLE succeeded.

USER_IND_COLUMNS 조회

```
DESCRIBE user_ind_columns
```

Name	Null	Type
INDEX_NAME		VARCHAR2(30)
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
COLUMN_POSITION		NUMBER
COLUMN_LENGTH		NUMBER
CHAR_LENGTH		NUMBER
DESCEND		VARCHAR2(4)

```
SELECT index_name, column_name, table_name
FROM   user_ind_columns
WHERE  index_name = 'LNAME_IDX';
```

INDEX_NAME	COLUMN_NAME	TABLE_NAME
1 LNAME_IDX	LAST_NAME	EMP_TEST

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

USER_IND_COLUMNS 조회

USER_IND_COLUMNS 덕셔너리 뷰는 인덱스 이름, 인덱스화된 테이블의 이름, 인덱스 내에 있는 열의 이름 및 인덱스 내에서 열의 위치 등의 정보를 제공합니다.

위 슬라이드 예제에서 emp_test 테이블과 LNAME_IDX 인덱스는 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE emp_test AS SELECT * FROM employees;
CREATE INDEX LNAME_IDX ON emp_test(Last_Name);
```

동의어 정보

```
DESCRIBE user_synonyms
```

Name	Null	Type
SYNONYM_NAME	NOT NULL	VARCHAR2(30)
TABLE_OWNER		VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
DB_LINK		VARCHAR2(128)

```
SELECT *
FROM    user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	TEAM2	ORA22	DEPARTMENTS	(null)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

동의어 정보

USER_SYNONYMS 딕셔너리 뷰는 전용 (private) 동의어 (유저가 소유한 동의어)에 대해 설명합니다.

이 뷰를 query하면 동의어를 찾을 수 있습니다. ALL_SYNONYMS를 query하면 유저에게 제공되는 모든 동의어 이름 및 이러한 동의어가 적용되는 객체를 찾을 수 있습니다.

이 뷰의 열은 다음과 같습니다.

- **SYNONYM_NAME:** 동의어의 이름
- **TABLE_OWNER:** 동의어를 통해 참조되는 객체의 소유자
- **TABLE_NAME:** 동의어를 통해 참조되는 테이블이나 뷰의 이름
- **DB_LINK:** 데이터베이스 링크 참조의 이름(있는 경우)

단원 내용

- 데이터 딕셔너리 소개
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 테이블 정보
 - 열 정보
 - 제약 조건 정보
- 딕셔너리 뷰를 query하여 다음 정보 검색
 - 뷰 정보
 - 시퀀스 정보
 - 동의어 정보
 - 인덱스 정보
- 테이블에 주석을 추가하고 딕셔너리 뷰를 query하여 주석 정보 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

테이블에 주석 추가

- COMMENT 문을 사용하여 테이블이나 열에 주석을 추가할 수 있습니다.

```
COMMENT ON TABLE employees
IS 'Employee Information';
```

```
COMMENT ON COLUMN employees.first_name
IS 'First name of the employee';
```

- 주석은 데이터 딕셔너리 뷰를 통해 볼 수 있습니다.
 - ALL_COL_COMMENTS
 - USER_COL_COMMENTS
 - ALL_TAB_COMMENTS
 - USER_TAB_COMMENTS

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블에 주석 추가

COMMENT 문을 사용하여 열, 테이블, 뷰 또는 스냅샷에 대해 최대 4,000바이트의 주석을 추가할 수 있습니다. 주석은 데이터 딕셔너리에 저장되며 COMMENTS 열의 다음 데이터 딕셔너리 뷰 중 하나를 통해 볼 수 있습니다.

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

구문

```
COMMENT ON {TABLE table | COLUMN table.column}
IS 'text';
```

이 구문에서 다음이 적용됩니다.

table 테이블의 이름입니다.

column 테이블에 있는 열의 이름입니다.

text 주석의 텍스트입니다.

주석을 빈 문자열(' ')로 설정하는 방식으로 데이터베이스에서 주석을 삭제할 수 있습니다.

```
COMMENT ON TABLE employees IS '';
```

퀴즈

딕셔너리 테이블을 기반으로 하는 딕셔너리 뷰에는 다음과 같은 정보가 들어 있습니다.

1. 데이터베이스의 모든 스키마 객체에 대한 정의
2. 열의 기본값
3. 무결성 제약 조건 정보
4. 각 유저에게 부여된 권한 및 룰
5. 위 항목 모두

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 5

요약

이 단원에서는 다음 딕셔너리 뷰를 통해 객체에 대한 정보를 찾는 방법을 배웠습니다.

- **DICTIONARY**
- **USER_OBJECTS**
- **USER_TABLES**
- **USER_TAB_COLUMNS**
- **USER_CONSTRAINTS**
- **USER_CONS_COLUMNS**
- **USER_VIEWS**
- **USER_SEQUENCES**
- **USER_INDEXES**
- **USER_SYNONYMS**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 유저에게 제공되는 일부 딕셔너리 뷰에 대해 배웠습니다. 이러한 딕셔너리 뷰를 사용하여 테이블, 제약 조건, 뷰, 시퀀스 및 동의어에 대한 정보를 찾을 수 있습니다.

연습 3: 개요

이 연습에서는 다음 내용을 다룹니다.

- 딕셔너리 뷰를 query하여 테이블 및 열 정보 검색
- 딕셔너리 뷰를 query하여 제약 조건 정보 검색
- 딕셔너리 뷰를 query하여 뷰 정보 검색
- 딕셔너리 뷰를 query하여 시퀀스 정보 검색
- 딕셔너리 뷰를 query하여 동의어 정보 검색
- 딕셔너리 뷰를 query하여 인덱스 정보 검색
- 테이블에 주석을 추가하고 딕셔너리 뷰를 query하여 주석 정보 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 3: 개요

이 연습에서는 딕셔너리 뷰를 query하여 스키마의 객체에 대한 정보를 찾습니다.

대형 데이터 집합 조작

4

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다중 테이블 INSERT의 기능 설명
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 subquery를 사용하여 오라클 데이터베이스의 데이터를 조작하는 방법에 대해 설명합니다. INSERT 문과 UPDATE 문에서 DEFAULT 키워드를 사용하여 기본 열 값을 식별하는 방법과 다중 테이블 INSERT 문, MERGE 문 및 데이터베이스의 변경 사항 추적에 대해 설명합니다.

단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Subquery를 사용하여 데이터 조작

DML(데이터 조작어) 문의 subquery를 사용하여 다음 작업을 수행할 수 있습니다.

- 인라인 뷰에서 데이터 검색
- 테이블 간에 데이터 복사
- 다른 테이블의 값을 기반으로 테이블의 데이터 생성
- 다른 테이블의 행을 기반으로 테이블에서 행 삭제



Copyright © 2009, Oracle. All rights reserved.

Subquery를 사용하여 데이터 조작

Subquery를 사용하면 테이블에서 데이터를 검색하여 다른 테이블에서 INSERT에 대한 입력으로 사용할 수 있습니다. 이러한 방식으로 단일 SELECT 문을 사용하여 테이블 간에 대량의 데이터를 쉽게 복사할 수 있습니다. 마찬가지로 UPDATE 문과 DELETE 문의 WHERE 절에서 subquery를 사용하여 대량 생성 및 삭제를 수행할 수 있습니다. 또한 SELECT 문의 FROM 절에서도 subquery를 사용할 수 있습니다. 이를 인라인 뷰라고 합니다.

참고: *Oracle Database 11g: SQL Fundamentals I* 과정에서 다른 테이블을 기반으로 행을 생성하고 삭제하는 방법을 배웠습니다.

Subquery를 스스로 사용하여 데이터 검색

```

SELECT department_name, city
FROM departments
NATURAL JOIN (SELECT l.location_id, l.city, l.country_id
               FROM loc l
               JOIN countries c
                 ON(l.country_id = c.country_id)
               JOIN regions USING(region_id)
              WHERE region_name = 'Europe');

```

	DEPARTMENT_NAME	CITY
1	Human Resources	London
2	Sales	Oxford
3	Public Relations	Munich

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Subquery를 스스로 사용하여 데이터 검색

SELECT 문의 FROM 절에 subquery를 사용할 수 있으며 그 방법은 뷰 사용 방법과 비슷합니다. SELECT 문의 FROM 절에 있는 subquery를 인라인 뷰라고도 합니다. SELECT 문의 FROM 절에 있는 subquery는 해당하는 특정 SELECT 문과 해당 SELECT 문에 대한 데이터 소스를 정의합니다. 데이터베이스 뷰와 마찬가지로 subquery의 SELECT 문은 필요에 따라 간단하거나 복잡할 수 있습니다.

데이터베이스 뷰가 생성될 때 연관된 SELECT 문이 데이터 딕셔너리에 저장됩니다. 데이터베이스 뷰를 생성하는 데 필요한 권한이 없거나 SELECT 문의 뷰 적합성을 테스트하는 경우에 인라인 뷰를 사용할 수 있습니다.

인라인 뷰에서는 query를 지원하는 데 필요한 모든 코드를 제공합니다. 즉, 복잡하게 데이터베이스 뷰를 별도로 생성하지 않아도 됩니다. 슬라이드의 예제는 인라인 뷰를 사용하여 유럽의 부서 이름과 도시를 표시하는 방법을 보여줍니다. FROM 절의 subquery는 3개의 서로 다른 테이블을 조인하여 위치 ID, 도시 이름 및 국가를 패치(fetch)합니다. Inner query의 출력은 outer query에 대한 테이블로 간주됩니다. Inner query는 데이터베이스 뷰의 query와 유사하지만 물리적 이름이 없습니다.

슬라이드의 예제에서는 다음 명령문을 실행하여 loc 테이블을 생성합니다.

```
CREATE TABLE loc AS SELECT * FROM locations;
```

subquery를 스스로 사용하여 데이터 검색(계속)

다음 두 단계를 수행하면 슬라이드의 예제와 동일한 결과를 얻을 수 있습니다.

- 데이터베이스 뷰를 생성합니다.

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN   regions USING(region_id)
WHERE  region_name = 'Europe';
```

- EUROPEAN_CITIES 뷰를 DEPARTMENTS 테이블과 조인합니다.

```
SELECT department_name, city
FROM   departments
NATURAL JOIN european_cities;
```

참고: *Oracle Database 11g: SQL Fundamentals I* 과정에서 데이터베이스 뷰를 생성하는 방법을 배웠습니다.

Subquery를 대상으로 사용하여 삽입

```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
             FROM   locations l
             JOIN   countries c
            ON(l.country_id = c.country_id)
             JOIN regions USING(region_id)
            WHERE region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

1 rows inserted

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Subquery를 대상으로 사용하여 삽입

INSERT 문의 INTO 절에서 테이블 이름 자리에 subquery를 사용할 수 있습니다. 이 subquery의 SELECT 리스트와 VALUES 절 열 리스트의 열 개수는 동일해야 합니다. INSERT 문이 성공적으로 실행되려면 기본 테이블 열의 모든 규칙을 따라야 합니다. 예를 들어, 중복된 위치 ID를 입력하거나 필수 NOT NULL 열의 값을 생략할 수 없습니다.

이러한 subquery를 사용하면 단지 INSERT를 수행하기 위해 뷔를 생성하지 않아도 됩니다. 슬라이드의 예제는 LOC 대신 subquery를 사용하여 새 유럽 도시에 대한 레코드를 생성합니다.

참고: 다음 코드를 사용하여 EUROPEAN_CITIES 뷔에서 INSERT 작업을 수행할 수도 있습니다.

```
INSERT INTO european_cities
VALUES (3300, 'Cardiff', 'UK');
```

Subquery를 대상으로 사용하여 삽입

결과를 확인합니다.

```
SELECT location_id, city, country_id
FROM loc
```

	LOCATION_ID	CITY	COUNTRY_ID
20	2900	Geneva	CH
21	3000	Bern	CH
22	3100	Utrecht	NL
23	3200	Mexico City	MX
24	3300	Cardiff	UK

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Subquery를 대상으로 사용하여 삽입(계속)

슬라이드의 예제는 인라인 뷰를 통한 삽입으로 인해 기본 테이블 LOC에 새 레코드가 생성되었음을 보여줍니다.

다음 예제는 INSERT 문에서 테이블을 식별하는 데 사용된 subquery의 결과를 보여줍니다.

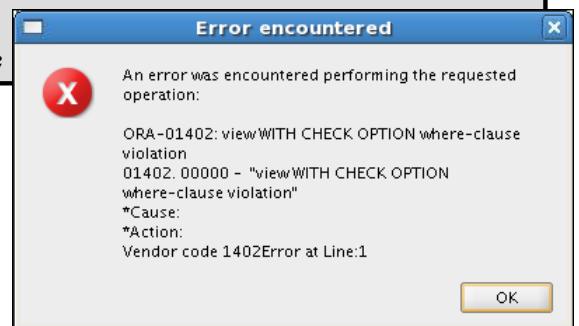
```
SELECT l.location_id, l.city, l.country_id
FROM loc l
JOIN countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe'
```

	LOCATION_ID	CITY	COUNTRY_ID
6	2700	Munich	DE
7	2900	Geneva	CH
8	3000	Bern	CH
9	3100	Utrecht	NL
10	3300	Cardiff	UK

DML 문에 WITH CHECK OPTION 키워드 사용

WITH CHECK OPTION 키워드를 사용하면 subquery에 없는 행은 변경할 수 없습니다.

```
INSERT INTO ( SELECT location_id, city, country_id
    FROM loc
    WHERE country_id IN
        (SELECT country_id
            FROM countries
            NATURAL JOIN regions
            WHERE region_name = 'Europe' )
        WITH CHECK OPTION )
VALUES ( 3600, 'Washington', 'US');
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

DML 문에 WITH CHECK OPTION 키워드 사용

INSERT, UPDATE, DELETE 문의 테이블 자리에 subquery가 사용되는 경우 해당 테이블에 대해 subquery에 포함되지 않은 행을 생성하도록 변경하지 못하게 하려면 WITH CHECK OPTION 키워드를 지정합니다.

슬라이드의 예제는 인라인 뷰를 WITH CHECK OPTION과 함께 사용하는 방법을 보여줍니다. INSERT 문은 유럽에 없는 도시에 대해 LOC 테이블에 레코드를 생성하지 못하게 합니다.

다음 예제는 VALUES 리스트의 변경 사항 때문에 성공적으로 실행됩니다.

```
INSERT INTO (SELECT location_id, city, country_id
    FROM loc
    WHERE country_id IN
        (SELECT country_id
            FROM countries
            NATURAL JOIN regions
            WHERE region_name = 'Europe' )
        WITH CHECK OPTION)
VALUES ( 3500, 'Berlin', 'DE');
```

DML 문에 WITH CHECK OPTION 키워드 사용(계속)

인라인 뷰에 WITH CHECK OPTION을 사용하여 간단하게 테이블을 변경하지 못하게 할 수 있습니다.

비유럽 도시의 생성을 방지하기 위해 다음 단계를 수행하여 데이터베이스 뷰를 사용할 수도 있습니다.

1. 데이터베이스 뷰를 생성합니다.

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT location_id, city, country_id
FROM   locations
WHERE  country_id in
       (SELECT country_id
        FROM   countries
        NATURAL JOIN regions
        WHERE  region_name = 'Europe')
WITH CHECK OPTION;
```

2. 데이터를 삽입하여 결과를 확인합니다.

```
INSERT INTO european_cities
VALUES (3400, 'New York', 'US');
```

두번째 단계를 수행하면 슬라이드에 표시된 것과 동일한 오류가 발생합니다.

단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

명시적 기본값 기능의 개요

- 기본 열 값이 필요한 경우 **DEFAULT** 키워드를 열 값으로 사용합니다.
- 이 기능을 사용하면 유저가 기본값을 데이터에 적용해야 하는 위치 및 시기를 제어할 수 있습니다.
- 명시적 기본값은 **INSERT** 문과 **UPDATE** 문에서 사용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

명시적 기본값

DEFAULT 키워드는 **INSERT** 문과 **UPDATE** 문에서 기본 열 값을 식별하는 데 사용할 수 있습니다. 기본값이 존재하지 않는 경우 널 값이 사용됩니다.

DEFAULT 옵션이 도입됨에 따라 이전처럼 프로그램에서 기본값을 하드 코딩하거나 기본값을 찾기 위해 딕셔너리를 query 할 필요가 없어졌습니다. 기본값을 하드 코딩하면 기본값이 변경될 경우 그에 따라 코드도 변경해야 하기 때문에 문제가 됩니다. 딕셔너리 액세스 기능은 대개 응용 프로그램에서 수행되지 않으므로 이 기능은 매우 중요한 기능입니다.

명시적 기본값 사용

- **INSERT 문에서 DEFAULT 사용:**

```
INSERT INTO deptm3
(department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- **UPDATE 문에서 DEFAULT 사용:**

```
UPDATE deptm3
SET manager_id = DEFAULT
WHERE department_id = 10;
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

명시적 기본값 사용

이전에 열에 대한 기본값으로 지정된 값으로 열을 설정하려면 DEFAULT를 지정합니다.

해당 열에 대해 기본값이 지정되지 않으면 Oracle 서버에서 열을 널로 설정합니다.

슬라이드의 첫번째 예제에서 INSERT 문은 MANAGER_ID 열에 대해 기본값을 사용합니다. 열에 정의된 기본값이 없는 경우 대신 널 값이 삽입됩니다.

두번째 예제는 UPDATE 문을 사용하여 MANAGER_ID 열을 부서 10에 대한 기본값으로 설정합니다. 열에 대해 기본값이 정의되지 않은 경우 널 값으로 변경됩니다.

참고: 테이블을 생성할 때 열에 대한 기본값을 지정할 수 있습니다. 이에 대한 자세한 내용은 *SQL Fundamentals I*을 참조하십시오.

다른 테이블에서 행 복사

- **INSERT 문을 subquery로 작성합니다.**

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

33 rows inserted

- **VALUES 절을 사용하지 마십시오.**
- **INSERT 절의 열 개수와 subquery의 열 개수를 일치시킵니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다른 테이블에서 행 복사

INSERT 문을 사용하여 기존 테이블에서 파생된 값으로 테이블에 행을 추가할 수 있습니다. VALUES 절 자리에 subquery를 사용합니다.

구문

```
INSERT INTO table [ column ( , column ) ] subquery;
```

이 구문에서 다음이 적용됩니다.

table 테이블 이름입니다.

column 테이블에 채울 열의 이름입니다.

subquery 테이블에 행을 반환하는 subquery입니다.

INSERT 절의 열 리스트에 나오는 열 개수 및 해당 데이터 유형은 subquery의 값 개수 및 해당 데이터 유형과 일치해야 합니다. 테이블 행의 복사본을 생성하려면 subquery에서 SELECT *를 사용합니다.

```
INSERT INTO EMPL3
SELECT *
FROM employees;
```

참고: DML 문에서 LOG ERRORS 절을 사용하여 오류에 관계없이 DML 작업을 완료할 수 있습니다. 오라클은 유저가 생성한 오류 로깅 테이블에 오류 메시지의 세부 정보를 기록합니다. 자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

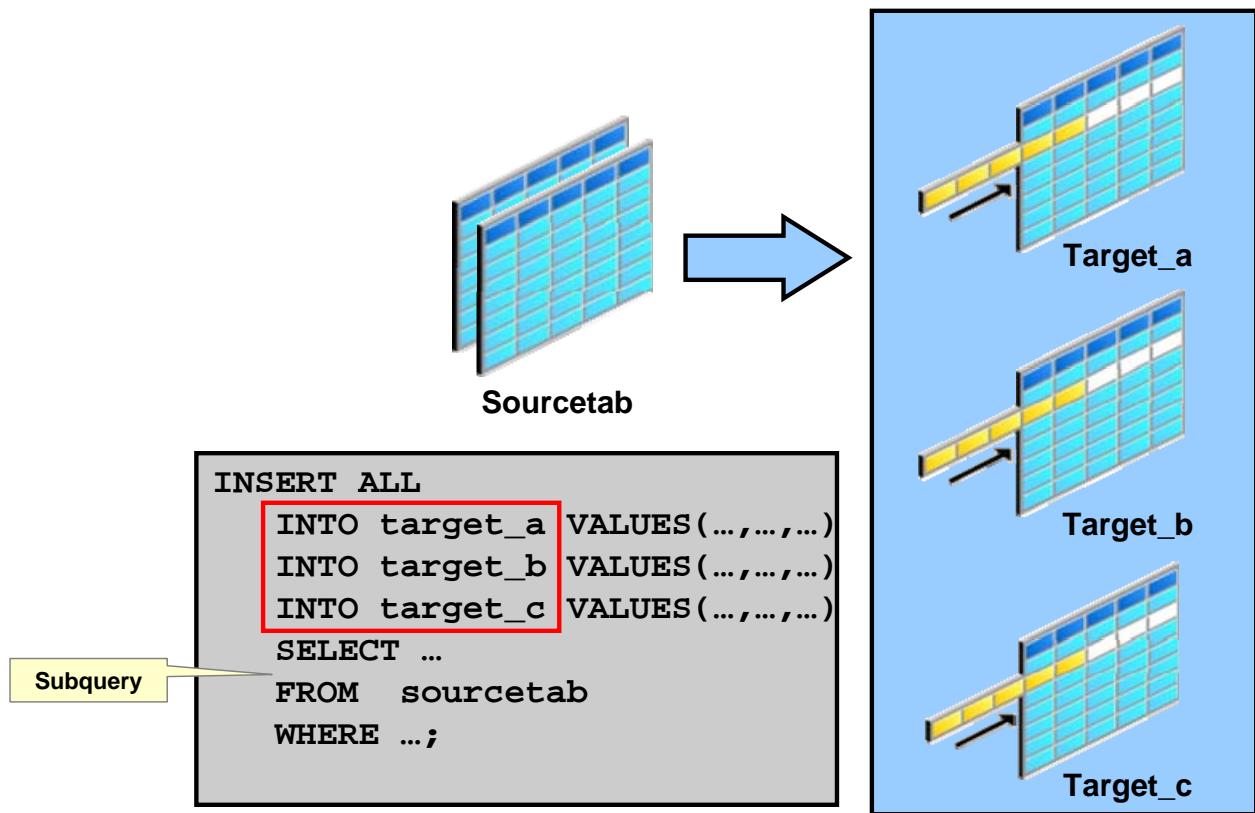
단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다중 테이블 INSERT 문의 개요



ORACLE

Copyright © 2009, Oracle. All rights reserved.

다중 테이블 INSERT 문의 개요

Subquery 평가 결과 반환된 행에서 발생된 계산된 행을 다중 테이블 INSERT 문의 테이블에 삽입합니다.

다중 테이블 INSERT 문은 데이터 웨어하우스 시나리오에서 유용합니다. 업무 분석을 더욱 효율적으로 수행하려면 데이터 웨어하우스를 정기적으로 로드해야 합니다. 이렇게 하려면 하나 이상의 운영 시스템에서 데이터를 추출하여 웨어하우스에 복사해야 합니다. 소스 시스템에서 데이터를 추출하여 데이터 웨어하우스로 가져오는 과정을 일반적으로 ETL이라고 하며, 이는 추출(Extraction), 변형(Transformation) 및 로드>Loading)를 의미합니다.

추출 과정에서는 데이터베이스 시스템이나 응용 프로그램과 같은 여러 소스에서 원하는 데이터를 식별하여 추출해야 합니다. 추출이 끝나면 데이터를 추가로 처리할 수 있도록 대상 시스템이나 중개 시스템으로 데이터를 물리적으로 전송해야 합니다. 선택한 전송 방식에 따라 프로세스 도중에 일부 변형이 수행될 수 있습니다. 예를 들어 게이트웨이를 통해 원격 대상에 직접 액세스하는 SQL 문은 두 개 열을 SELECT 문의 일부로 연결할 수 있습니다.

데이터가 오라클 데이터베이스에 로드되면 SQL 작업을 통해 데이터 변형을 실행할 수 있습니다. 다중 테이블 INSERT 문은 SQL 데이터 변형을 구현하는 기술 중 하나입니다.

다중 테이블 INSERT 문의 개요

- **INSERT...ELECT 문을 사용하여 행을 단일 DML 문의 일부로 다중 테이블에 삽입합니다.**
- **데이터 웨어하우징 시스템에서 다중 테이블 INSERT 문을 사용하여 하나 이상의 운영 소스에서 대상 테이블 집합으로 데이터를 전송합니다.**
- **다음과 같은 비교를 통해 성능이 크게 향상됨을 확인할 수 있습니다.**
 - 단일 DML 문과 다중 INSERT...ELECT 문 비교
 - 단일 DML과 IF...THEN 구문을 사용하여 다중 삽입을 수행하는 프로시저 비교

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다중 테이블 INSERT 문의 개요(계속)

다중 테이블 INSERT 문은 다중 테이블이 대상으로 포함된 경우 INSERT ... SELECT 문의 이점을 제공합니다. 다중 테이블 INSERT를 사용하지 않을 경우 n 개의 독립적인 INSERT ... SELECT 문을 처리해야 하므로 동일한 소스 데이터를 n 번 처리하고 변환 작업 로드도 n 배 늘어납니다.

기존의 INSERT ... SELECT 문을 사용하는 경우와 마찬가지로, 새 명령문을 병렬화하고 직접 로드 방식과 함께 사용하여 성능을 향상시킬 수 있습니다.

이제 비관계형 데이터베이스 테이블과 같은 입력 스트림의 각 레코드를 관계형 데이터베이스 테이블 환경을 위한 다중 레코드로 변환할 수 있습니다. 다른 방식으로 이 기능을 구현하려면 다중 INSERT 문을 작성해야 했습니다.

다중 테이블 INSERT 문의 유형

다중 테이블 INSERT 문에는 다음과 같은 다양한 유형이 있습니다.

- 무조건 INSERT
- 조건부 INSERT ALL
- 피벗팅 INSERT
- 조건부 INSERT FIRST

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다중 테이블 INSERT 문의 유형

다른 절을 사용하여 실행할 INSERT 유형을 나타낼 수 있습니다. 다중 테이블 INSERT 문에는 다음과 같은 유형이 있습니다.

- 무조건 INSERT: subquery에 의해 반환된 각 행에 대해 각 대상 테이블에 행이 삽입됩니다.
- 조건부 INSERT ALL: subquery에 의해 반환된 각 행에 대해 지정된 조건이 충족되는 경우 각 대상 테이블에 행이 삽입됩니다.
- 피벗팅 INSERT: 조건부 INSERT ALL의 특별한 경우입니다.
- 조건부 INSERT FIRST: subquery에 의해 반환된 각 행에 대해 조건이 충족되는 첫번째 대상 테이블에 행이 삽입됩니다.

다중 테이블 INSERT 문

- **다중 테이블 INSERT의 구문:**

```
INSERT [conditional_insert_clause]
[insert_into_clause values_clause] (subquery)
```

- **conditional_insert_clause:**

```
[ALL|FIRST]
[WHEN condition THEN] [insert_into_clause values_clause]
[ELSE] [insert_into_clause values_clause]
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

다중 테이블 INSERT 문

슬라이드는 다중 테이블 INSERT 문의 일반 형식을 보여줍니다.

무조건 INSERT: ALL into_clause

무조건 다중 테이블 INSERT를 수행하려면 ALL 다음에 여러 개의 insert_into_clause를 지정합니다. Oracle 서버는 subquery에 의해 반환된 각 행에 대해 각 insert_into_clause를 한 번 실행합니다.

조건부 INSERT: conditional_insert_clause

조건부 다중 테이블 INSERT를 수행하려면 conditional_insert_clause를 지정합니다. Oracle 서버는 해당 WHEN 조건을 통해 각 insert_into_clause를 필터링하여 해당 insert_into_clause가 실행되는지 여부를 판별합니다. 하나의 다중 테이블 INSERT 문은 최대 127개의 WHEN 절을 포함할 수 있습니다.

조건부 INSERT: ALL

ALL을 지정하면 Oracle 서버는 다른 WHEN 절의 평가 결과에 관계없이 각 WHEN 절을 평가합니다. 조건이 true로 평가된 각 WHEN 절에 대해 Oracle 서버는 해당 INTO 절 리스트를 실행합니다.

다중 테이블 INSERT 문(계속)

조건부 INSERT: FIRST

FIRST를 지정하면 Oracle 서버는 각 WHEN 절을 명령문에 나타난 순서대로 평가합니다. 첫번째 WHEN 절이 True로 평가되면 Oracle 서버는 해당 INTO 절을 실행하고 주어진 행에 대해 다음 WHEN 절은 건너뜁니다.

조건부 INSERT: ELSE 절

주어진 행에 대해 True로 평가된 WHEN 절이 없는 경우:

- ELSE 절을 지정한 경우 Oracle 서버는 ELSE 절과 연관된 INTO 절 리스트를 실행합니다.
- ELSE 절을 지정하지 않은 경우 Oracle 서버는 해당 행에 대해 아무 작업도 수행하지 않습니다.

다중 테이블 INSERT 문에 대한 제한 사항

- 뷰 또는 Materialized View가 아닌 테이블에 대해서만 다중 테이블 INSERT 문을 수행할 수 있습니다.
- 원격 테이블에서는 다중 테이블 INSERT를 수행할 수 없습니다.
- 다중 테이블 INSERT를 수행할 때 테이블 컬렉션 표현식을 지정할 수 없습니다.
- 다중 테이블 INSERT에서 모든 insert_into_clause를 결합하여 999개를 초과하는 대상 열을 지정할 수 없습니다.

무조건 INSERT ALL

- EMPLOYEES 테이블에서 EMPLOYEE_ID가 200보다 큰 사원의 EMPLOYEE_ID, HIRE_DATE, SALARY 및 MANAGER_ID 값을 선택합니다.
- 다중 테이블 INSERT를 사용하여 SAL_HISTORY 및 MGR_HISTORY 테이블에 이 값을 삽입합니다.

```
INSERT ALL
  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;
```

12 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

무조건 INSERT ALL

슬라이드의 예제는 행을 SAL_HISTORY 테이블과 MGR_HISTORY 테이블에 모두 삽입합니다.

SELECT 문은 EMPLOYEES 테이블에서 사원 ID가 200보다 큰 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID의 세부 사항을 검색합니다. 사원 ID, 채용 날짜 및 급여의 세부 정보가 SAL_HISTORY 테이블에 삽입됩니다. 사원 ID, 관리자 ID 및 급여 세부 사항을 MGR_HISTORY 테이블에 삽입합니다.

이 INSERT 문은 SELECT 문으로 검색된 행에 더 이상의 제한이 적용되지 않으므로 무조건 INSERT라고 합니다. SELECT 문으로 검색된 모든 행은 SAL_HISTORY와 MGR_HISTORY라는 두 테이블에 삽입됩니다. INSERT 문의 VALUES 절은 SELECT 문에서 각 테이블에 삽입해야 하는 열을 지정합니다. SELECT 문에 의해 반환된 각 행은 SAL_HISTORY 테이블과 MGR_HISTORY 테이블에 각각 하나씩 삽입됩니다.

무조건 INSERT ALL(계속)

총 12개의 행이 선택되었습니다.

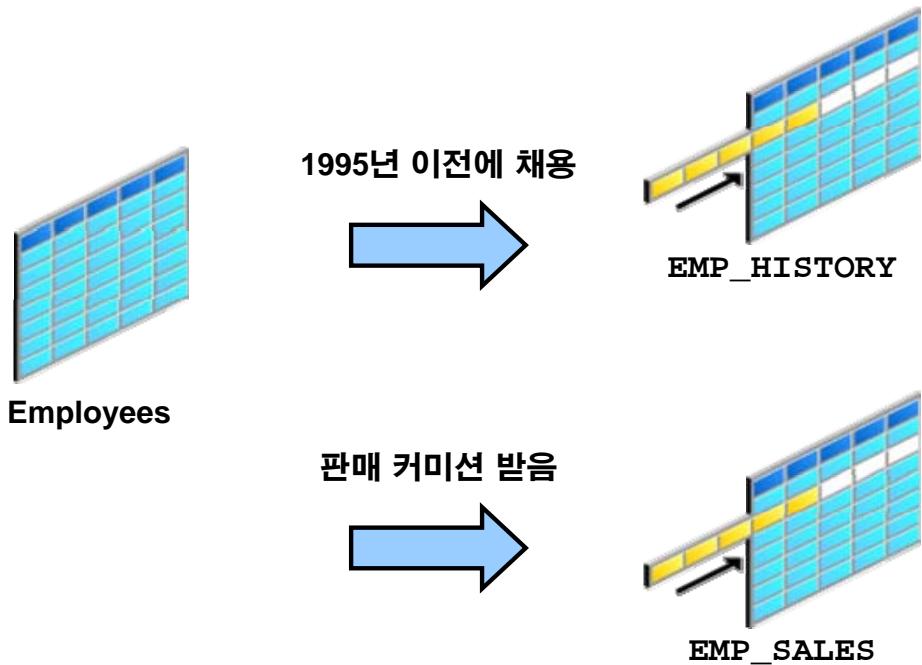
```
SELECT COUNT(*) total_in_sal FROM sal_history;
```

TOTAL_IN_SAL	
1	6

```
SELECT COUNT(*) total_in_mgr FROM mgr_history;
```

TOTAL_IN_MGR	
1	6

조건부 INSERT ALL: 예제



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

조건부 INSERT ALL: 예제

`Employees` 테이블의 모든 사원 중 1995년 이전에 채용된 사원의 레코드를 사원 기록에 삽입합니다. 사원이 판매 커미션을 받는 경우 `EMP_SALES` 테이블에 레코드 정보를 삽입합니다. SQL 문은 다음 페이지에 표시됩니다.

조건부 INSERT ALL

```
INSERT ALL
```

```
WHEN HIREDATE < '01-JAN-95' THEN
    INTO emp_history VALUES(EMPID,HIREDATE,SAL)
WHEN COMM IS NOT NULL THEN
    INTO emp_sales VALUES(EMPID,COMM,SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
       salary SAL, commission_pct COMM
FROM employees
```

48 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조건부 INSERT ALL

슬라이드의 예제는 이전 슬라이드의 예제와 유사하게 EMP_HISTORY 테이블과 EMP_SALES 테이블에 모두 행을 삽입합니다. SELECT 문은 EMPLOYEES 테이블에서 모든 사원에 대해 사원 ID, 채용 날짜, 급여 및 커미션 비율과 같은 세부 정보를 검색합니다. 사원 ID, 채용 날짜, 급여 등의 세부 정보는 EMP_HISTORY 테이블에 삽입됩니다. 사원 ID, 수수료율, 급여 등의 세부 정보는 EMP_SALES 테이블에 삽입됩니다.

이 INSERT 문은 SELECT 문으로 검색된 행에 제한 사항이 추가로 적용되므로 조건부 INSERT ALL이라고 합니다. SELECT 문에 의해 검색된 행에서 채용 날짜가 1995년 이전인 행만 EMP_HISTORY 테이블에 삽입됩니다. 마찬가지로, 수수료율의 값이 널이 아닌 행만 EMP_SALES 테이블에 삽입됩니다.

```
SELECT count(*) FROM emp_history;
```

	COUNT(*)
1	13

```
SELECT count(*) FROM emp_sales;
```

	COUNT(*)
1	35

조건부 INSERT ALL(계속)

INSERT ALL 문과 함께 ELSE 절을 사용할 수도 있습니다.

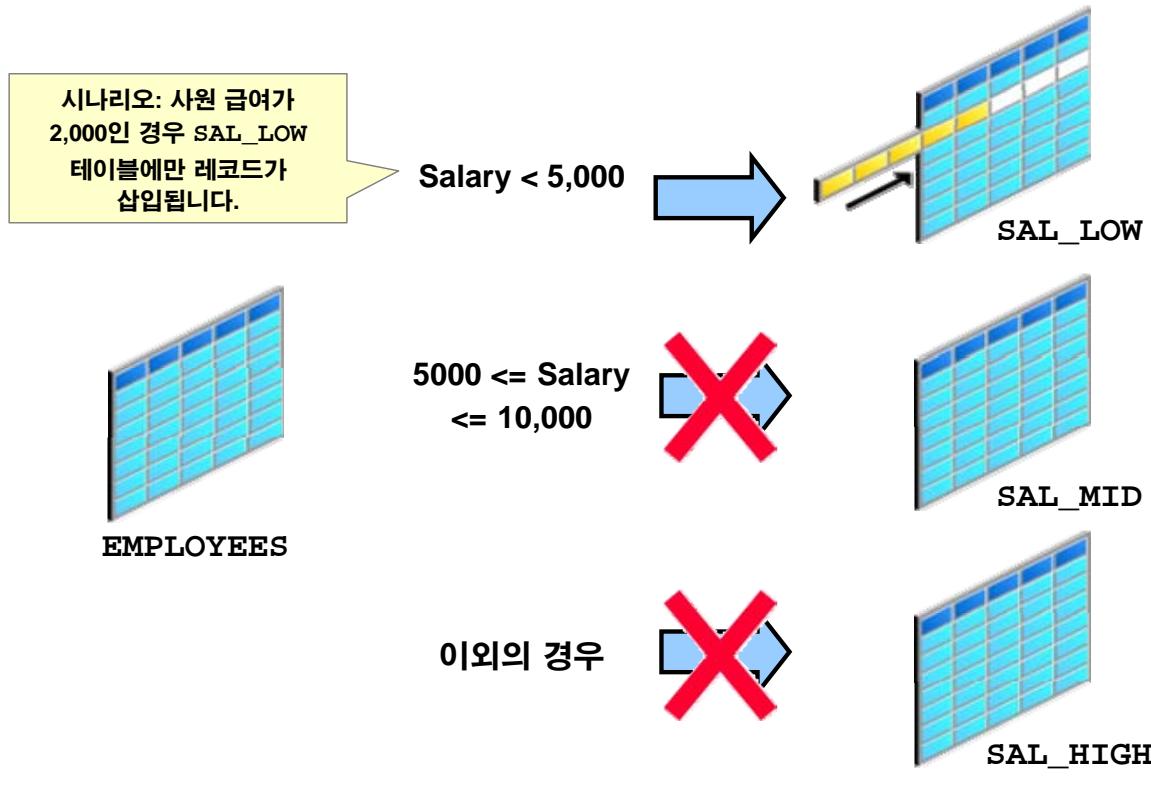
예제:

```
INSERT ALL
WHEN job_id IN
  (select job_id FROM jobs WHERE job_title LIKE '%Manager%') THEN
    INTO managers2(last_name,job_id,SALARY)
      VALUES (last_name,job_id,SALARY)
WHEN SALARY>10000 THEN
    INTO richpeople(last_name,job_id,SALARY)
      VALUES (last_name,job_id,SALARY)
ELSE
    INTO poorpeople VALUES (last_name,job_id,SALARY)
SELECT * FROM employees;
```

결과:

116개의 행이 삽입되었습니다.

조건부 INSERT FIRST: 예제



Copyright © 2009, Oracle. All rights reserved.

조건부 INSERT FIRST: 예제

EMPLOYEES 테이블의 모든 사원에 대해 조건을 충족하는 첫번째 대상 테이블에 사원 정보를 삽입합니다. 예제에서 사원의 급여가 2,000인 경우 SAL_LOW 테이블에만 레코드가 삽입됩니다. SQL 문은 다음 페이지에 표시됩니다.

조건부 INSERT FIRST

```

INSERT FIRST

WHEN salary < 5000 THEN

    INTO sal_low VALUES (employee_id, last_name, salary)

WHEN salary between 5000 and 10000 THEN

    INTO sal_mid VALUES (employee_id, last_name, salary)

ELSE

    INTO sal_high VALUES (employee_id, last_name, salary)

SELECT employee_id, last_name, salary
FROM employees

```

107 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

조건부 INSERT FIRST

SELECT 문은 EMPLOYEES 테이블에서 모든 사원의 사원 ID, 성, 급여와 같은 세부 정보를 검색합니다. 각 사원 레코드는 조건을 충족하는 첫번째 대상 테이블에 삽입됩니다.

이 INSERT 문을 조건부 INSERT FIRST라고 합니다. WHEN salary < 5000 조건이 가장 먼저 평가됩니다. 이 첫번째 WHEN 절이 true로 평가되면 Oracle 서버가 해당 INTO 절을 실행하고 SAL_LOW 테이블에 레코드를 삽입합니다. 이 행에 대해 후속 WHEN 절은 건너뜁니다.

행이 첫번째 WHEN 조건 (WHEN salary < 5000)을 충족하지 않는 경우 다음 조건(WHEN salary between 5000 and 10000)이 평가됩니다. 이 조건이 true로 평가되면 SAL_MID 테이블에 레코드가 삽입되고 마지막 조건을 건너뜁니다.

첫번째 조건 (WHEN salary < 5000)과 두번째 조건 (WHEN salary between 5000 and 10000)이 모두 true로 평가되지 않으면 Oracle 서버가 ELSE 절에 대해 해당 INTO 절을 실행합니다.

조건부 INSERT FIRST(계속)

총 20개의 행이 삽입되었습니다.

```
SELECT count(*) low FROM sal_low;
```

	LOW
1	49

```
SELECT count(*) mid FROM sal_mid;
```

	MID
1	43

```
SELECT count(*) high FROM sal_high;
```

	HIGH
1	15

피벗팅 INSERT

비관계형 데이터베이스 테이블에서 판매 레코드 집합을 관계형 형식으로 변환합니다.

The diagram illustrates the concept of pivoting (pivoting insert) by showing the transformation of a single row of data from a wide-row format into multiple rows of data in a relational format.

Wide-Row Data:

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000

Relational Data (Result of Pivoting):

Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

피벗팅 INSERT

피벗팅은 비관계형 데이터베이스 테이블과 같은 입력 스트림의 각 레코드를 관계형 데이터베이스 테이블 환경의 다중 레코드로 변환하는 등의 변형 작업을 수행하는 것을 말합니다.

다음 비관계형 데이터베이스 테이블에서 판매 레코드 집합을 검색한다고 가정합시다.

SALES_SOURCE_DATA에서 다음 형식의 판매 레코드 집합 검색:

EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED,
SALES_THUR, SALES_FRI

이러한 레코드를 다음과 같은 보다 일반적인 관계형 형식으로 SALES_INFO 테이블에 저장하려고 합니다.

EMPLOYEE_ID, WEEK, SALES

이 문제를 해결하려면 원래의 비관계형 데이터베이스 테이블 SALES_SOURCE_DATA의 각 레코드가 데이터 웨어하우스의 SALES_INFO 테이블의 다섯 개의 레코드로 변환되도록 변형해야 합니다. 이러한 작업을 일반적으로 피벗팅이라고 합니다.

이 문제의 해결 방법은 다음 페이지에 나옵니다.

피벗팅 INSERT

```

INSERT ALL
  INTO sales_info VALUES (employee_id,week_id,sales_MON)
  INTO sales_info VALUES (employee_id,week_id,sales_TUE)
  INTO sales_info VALUES (employee_id,week_id,sales_WED)
  INTO sales_info VALUES (employee_id,week_id,sales_THUR)
  INTO sales_info VALUES (employee_id,week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR,sales_FRI
  FROM sales_source_data;

```

5 rows inserted

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

피버팅 INSERT(계속)

슬라이드의 예제에서 판매 데이터는 비관계형 데이터베이스 테이블 SALES_SOURCE_DATA에서 가져옵니다. 이 테이블은 판매 사원이 특정 주 ID의 주에 각 요일마다 판매한 판매 세부 정보입니다.

DESC SALES_SOURCE_DATA

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)

피벗팅 INSERT(계속)

```
SELECT * FROM SALES_SOURCE_DATA;
```

	EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
1	178	6	1750	2200	1500	1500	3000

```
DESC SALES_INFO
```

Name	Null	Type
EMPLOYEE_ID		NUMBER(6)
WEEK		NUMBER(2)
SALES		NUMBER(8,2)

```
SELECT * FROM sales_info;
```

	EMPLOYEE_ID	WEEK	SALES
1	178	6	1750
2	178	6	2200
3	178	6	1500
4	178	6	1500
5	178	6	3000

피벗팅 INSERT를 사용하는 앞의 예제에서 SALES_SOURCE_DATA 테이블의 한 행은 관계형 테이블 SALES_INFO의 레코드 다섯 개로 변환됩니다.

단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

MERGE 문

- 데이터베이스 테이블을 조건부로 갱신하거나 데이터를 삽입 또는 삭제하는 기능을 제공합니다.
- 행이 존재하는 경우 UPDATE를 수행하고, 새 행인 경우 INSERT를 수행합니다.
 - 별도의 갱신을 방지합니다.
 - 성능 및 사용 편의성이 향상됩니다.
 - 데이터 웨어하우징 응용 프로그램에서 유용합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

MERGE 문

Oracle 서버는 INSERT, UPDATE 및 DELETE 작업을 위한 MERGE 문을 지원합니다.

이 명령문을 사용하여 행을 조건부로 갱신하거나 테이블에 삽입하거나 테이블에서 삭제할 수 있으므로 다중 DML 문을 피할 수 있습니다. 대상 테이블에 대해 갱신, 삽입 또는 삭제 작업을 수행할지 여부는 ON 절의 조건에 따라 결정됩니다.

대상 테이블에 대한 INSERT 및 UPDATE 객체 권한과 소스 테이블에 대한 SELECT 객체 권한을 가져야 합니다. merge_update_clause의 DELETE 절을 지정하려면 대상 테이블에 대한 DELETE 객체 권한도 가져야 합니다.

MERGE 문은 결정적(deterministic) 명령문입니다. 동일한 MERGE 문에서 대상 테이블의 동일한 행을 여러 번 갱신할 수 없습니다.

다른 방법은 PL/SQL 루프와 다중 DML 문을 사용하는 것입니다. 그러나 MERGE 문은 사용하기 쉽고 간단하게 단일 SQL 문으로 표현됩니다.

MERGE 문은 다양한 데이터 웨어하우징 응용 프로그램에 적합합니다. 예를 들어, 데이터 웨어하우징 응용 프로그램에서 여러 소스의 데이터(이 데이터 중 일부는 중복될 수 있음)로 작업해야 하는 경우가 있습니다. MERGE 문을 사용하면 조건부로 행을 추가하거나 수정할 수 있습니다.

MERGE 문 구문

MERGE 문을 사용하여 테이블에 행을 조건부로 삽입, 갱신 또는 삭제할 수 있습니다.

```

MERGE INTO table_name table_alias
  USING (table/view/sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col1_val,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
  
```

Copyright © 2009, Oracle. All rights reserved.

행 병합

MERGE 문을 사용하여 조건부로 기존 행을 갱신하고 새 행을 삽입할 수 있습니다. MERGE 문을 사용하여 테이블에서 행을 갱신하면서 동시에 불필요한 행을 삭제할 수 있습니다. 이렇게 하려면 MERGE 문의 구문에 자체 WHERE 절이 있는 DELETE 절을 포함시킵니다.

이 구문에서 다음이 적용됩니다.

INTO 절	갱신하거나 삽입할 대상 테이블을 지정합니다.
USING 절	갱신하거나 삽입할 데이터의 소스를 식별합니다. 테이블, 뷰 또는 subquery가 될 수 있습니다.
ON 절	MERGE 작업의 갱신 또는 삽입을 결정하는 조건입니다.
WHEN MATCHED WHEN NOT MATCHED	조인 조건의 결과에 응답하는 방법을 서버에 지시합니다.

참고: 자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

행 병합: 예제

EMPLOYEES 테이블과 일치하도록 COPY_EMP3 테이블에 행을 삽입하거나 갱신합니다.

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...
DELETE WHERE (e.commission_pct IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

행 병합: 예제

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
c.phone_number = e.phone_number,
c.hire_date = e.hire_date,
c.job_id = e.job_id,
c.salary = e.salary*2,
c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,
c.department_id = e.department_id
DELETE WHERE (e.commission_pct IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

행 병합: 예제(계속)

COPY_EMP3 테이블은 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE COPY_EMP3 AS SELECT * FROM EMPLOYEES
WHERE SALARY<10000;
```

그린 다음 COPY_EMP3 테이블을 query합니다.

```
SELECT employee_id, salary, commission_pct FROM COPY_EMP3;
```

EMPLOYEE_ID	SALARY	COMMISSION_PCT
1	198	5200
2	199	5200
3	200	8800
4	202	12000
5	203	13000

...

64	197	6000	(null)
65	162	10500	0.25
66	146	13500	0.3
67	150	10000	0.3

...

SALARY < 10000인 여러 명의 사원과 COMMISSION_PCT를 가진 두 명의 사원이 있습니다.

슬라이드의 예제는 EMPLOYEES 테이블의 EMPLOYEE_ID를 COPY_EMP3 테이블의 EMPLOYEE_ID와 일치시킵니다. 일치하는 행이 발견되면 COPY_EMP3 테이블의 행이 EMPLOYEES 테이블의 행과 일치하도록 갱신되고 사원의 급여가 두 배가 됩니다.

COMMISSION_PCT 열에 값이 있는 두 명의 사원 레코드가 삭제됩니다. 일치하는 행이 없으면 COPY_EMP3 테이블에 행이 삽입됩니다.

행 병합: 예제

```
TRUNCATE TABLE copy_emp3;
SELECT * FROM copy_emp3;
0 rows selected
```

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...
DELETE WHERE (e.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, ...)
```

```
SELECT * FROM copy_emp3;
107 rows selected.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

행 병합: 예제(계속)

슬라이드의 예제는 COPY_EMP3 테이블이 비어있음을 보여줍니다. `c.employee_id = e.employee_id` 조건이 평가됩니다. 이 조건은 false를 반환합니다. 즉, 일치하는 항목이 없습니다. 논리는 WHEN NOT MATCHED 절로 분류되고 MERGE 명령은 EMPLOYEES 테이블의 행을 COPY_EMP3 테이블에 삽입합니다. 이제 COPY_EMP3 테이블은 EMPLOYEES 테이블과 정확히 동일한 데이터를 갖습니다.

```
SELECT employee_id, salary, commission_pct from copy_emp3;
```

	EMPLOYEE_ID	SALARY	COMMISSION_PCT
1	144	2500	(null)
2	143	2600	(null)
3	202	6000	(null)
4	141	3500	(null)
5	174	11000	0.3
...			
15	149	10500	0.2
16	206	8300	(null)
17	176	8600	0.2
18	124	5800	(null)
19	205	12000	(null)
20	178	7000	0.15

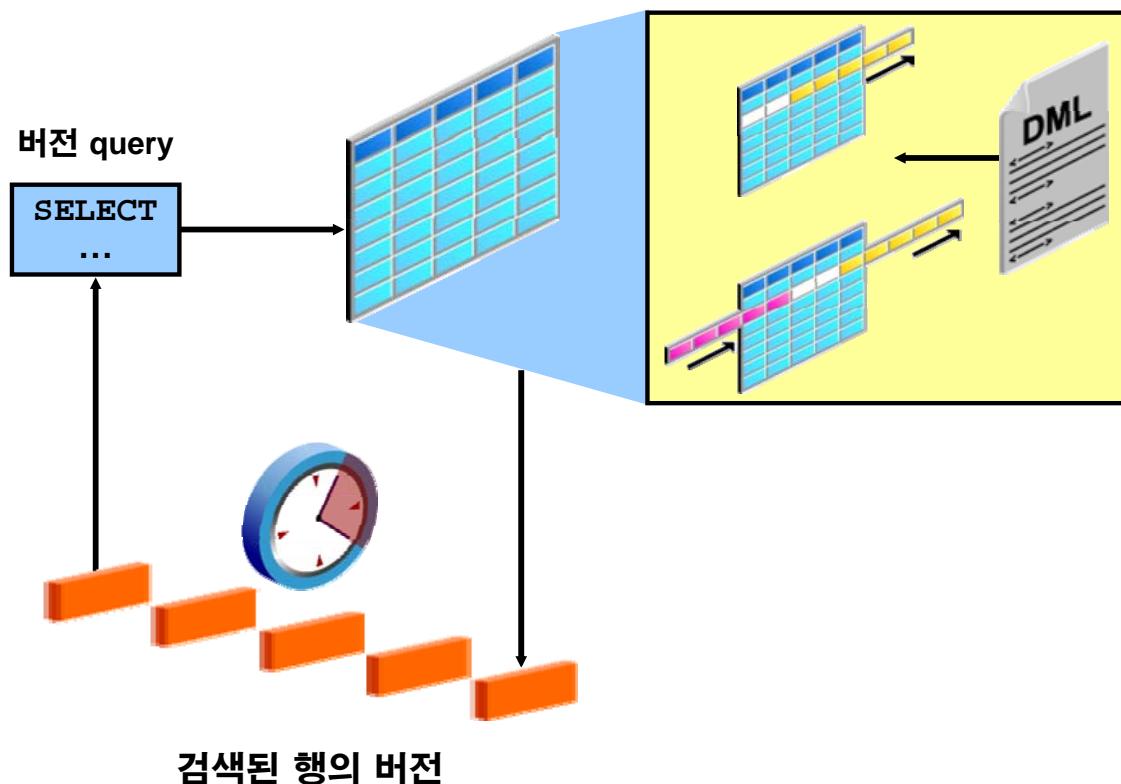
단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

데이터 변경 사항 추적



데이터 변경 사항 추적

테이블의 데이터가 부적절하게 변경된 사실을 발견하게 되는 경우가 있습니다. 이를 조사하기 위해 다중 flashback query를 사용하여 특정 시점의 행 데이터를 볼 수 있습니다. 보다 효과적인 방법으로는 flashback version query 기능을 사용하여 특정 기간의 행에 대해 모든 변경 사항을 볼 수 있습니다. 이 기능을 사용하면 SELECT 문에 SCN(시스템 변경 번호) 또는 행 값 변경 사항을 보려는 시간 기록 범위를 지정하는 VERSIONS 절을 추가할 수 있습니다. 또한 query에서 변경을 담당하는 트랜잭션과 같은 관련 메타 데이터를 반환할 수 있습니다.

뿐만 아니라 잘못된 트랜잭션을 식별한 후에 flashback transaction query 기능을 사용하여 해당 트랜잭션에서 수행한 다른 변경 사항을 식별할 수 있습니다. 그런 다음 Flashback Table 기능을 사용하여 변경되기 전의 상태로 테이블을 복원할 수 있습니다.

VERSIONS 절이 포함된 테이블 query를 사용하여 query가 실행된 시간과 현재 시간 전의 undo_retention 초 사이에 존재하거나 존재했던 모든 행의 모든 버전을 생성할 수 있습니다. undo_retention은 자동 튜닝 파라미터인 초기화 파라미터입니다. VERSIONS 절을 포함한 query를 version query라고 합니다. Version query의 결과는 WHERE 절이 행의 버전에 적용된 것처럼 동작합니다. Version query는 트랜잭션에서만 행 버전을 반환합니다.

시스템 변경 번호(SCN): Oracle 서버는 SCN을 할당하여 각각의 커밋된 트랜잭션에 대한 리두 레코드를 식별합니다.

Flashback Version Query의 예

```
SELECT salary FROM employees3
WHERE employee_id = 107;
```

1

```
UPDATE employees3 SET salary = salary * 1.30
WHERE employee_id = 107;
```

2

```
COMMIT;
```

```
SELECT salary FROM employees3
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 107;
```

3

1

	SALARY
1	4200

3

	SALARY
1	5460
2	4200

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Flashback Version Query의 예

슬라이드의 예제에서 사원 107의 급여가 검색됩니다(1). 사원 107의 급여가 30% 인상되고 이 변경 사항이 커밋됩니다(2). 서로 다른 버전의 급여가 표시됩니다(3).

VERSIONS 절은 query 계획을 변경하지 않습니다. 예를 들어, 인덱스 액세스 방식을 사용하는 테이블 query를 실행하는 경우 VERSIONS 절을 사용하여 동일한 테이블에 동일한 query를 수행하면 계속 인덱스 액세스 방식을 사용합니다. Version query에서 반환되는 행 버전은 전체 트랜잭션에서의 행의 버전입니다. VERSIONS 절은 query의 트랜잭션 동작에 영향을 주지 않습니다. 이것은 VERSIONS 절이 포함된 테이블 query가 진행 중인 트랜잭션의 query 환경을 계속 상속한다는 것을 의미합니다.

기본 VERSIONS 절은 VERSIONS BETWEEN { SCN | TIMESTAMP } MINVALUE AND MAXVALUE로 지정될 수 있습니다.

VERSIONS 절은 query에만 사용할 수 있는 SQL 확장입니다. Subquery 내에서 VERSIONS 절을 사용하는 DML 및 DDL 작업을 가질 수 있습니다. 행 version query는 선택된 행의 모든 커밋된 버전을 검색합니다. 현재 활성 트랜잭션에 의해 수행된 변경 사항은 반환되지 않습니다. Version query는 행의 모든 incarnation을 검색합니다. 이것은 본질적으로 반환되는 버전에 삭제된 이후 재삽입된 행 버전이 포함됨을 의미합니다.

Flashback Version Query의 예(계속)

Version query를 위한 행 액세스는 다음 두 가지 범주 중 하나로 정의될 수 있습니다.

- **ROWID 기반 행 액세스:** ROWID 기반 액세스의 경우 행 내용에 관계없이 지정된 ROWID의 모든 버전이 반환됩니다. 이는 본질적으로 ROWID로 나타난 블록에 있는 슬롯의 모든 버전이 반환됨을 의미합니다.
- **다른 모든 행 액세스:** 다른 모든 행 액세스의 경우 행의 모든 버전이 반환됩니다.

VERSIONS BETWEEN 절

```

SELECT versions_starttime "START_DATE",
       versions_endtime    "END_DATE",
       salary
  FROM employees
  VERSIONS BETWEEN SCN MINVALUE
              AND MAXVALUE
 WHERE last_name = 'Lorentz';

```

	START_DATE	END_DATE	SALARY
1	18-JUN-09 05.07.10.000000000 PM (null)		5460
2	(null)	18-JUN-09 05.07.10.000000000 PM	4200

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

VERSIONS BETWEEN 절

VERSIONS BETWEEN 절을 사용하여 query가 실행된 시간과 과거의 특정 시점 사이에 존재하거나 존재했던 행의 모든 버전을 검색할 수 있습니다.

Undo retention 시간이 BETWEEN 절의 하한값 시간 또는 SCN보다 작을 경우 query는 undo retention 시간까지만 버전을 검색합니다. BETWEEN 절의 시간 간격은 SCN 간격 또는 실제 시간 간격으로 지정될 수 있습니다. 이 시간 간격은 하한값 및 상한값에서 모두 닫힙니다.

예제에서 Lorentz의 급여 변경 사항이 검색됩니다. 첫번째 버전의 END_DATE에 대한 NULL 값은 이 버전이 query 시점에 존재했던 버전임을 나타냅니다. 마지막 버전의 START_DATE에 대한 NULL 값은 이 버전이 undo retention 시간 전에 생성되었음을 나타냅니다.

퀴즈

DEFAULT 키워드가 도입됨에 따라 INSERT 또는 UPDATE 명령을 사용할 때 프로그램에서 기본값을 하드 코딩하거나 기본값을 찾기 위해 딕셔너리를 query할 필요가 없습니다.

1. 맞습니다.
2. 틀립니다.



Copyright © 2009, Oracle. All rights reserved.

정답: 1

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- DML 문 사용 및 트랜잭션 제어
- 다중 테이블 INSERT의 기능 설명
- 다음과 같은 다중 테이블 INSERT 유형 사용
 - 무조건 INSERT
 - 피벗팅 INSERT
 - 조건부 INSERT ALL
 - 조건부 INSERT FIRST
- 테이블의 행 병합
- Subquery를 사용하여 데이터 조작
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 subquery를 사용하여 오라클 데이터베이스의 데이터를 조작하는 방법에 대해 설명했습니다. 또한 다중 테이블 INSERT 문, MERGE 문 및 데이터베이스의 변경 사항 추적에 대해 설명했습니다.

연습 4: 개요

이 연습에서는 다음 내용을 다룹니다.

- **다중 테이블 INSERT 수행**
- **MERGE 작업 수행**
- **행 버전 추적**



Copyright © 2009, Oracle. All rights reserved.

다른 시간대에서 데이터 관리

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 소수 표시 초를 저장하고 시간대를 추적하는 DATE와 유사한 데이터 유형 사용
- 두 datetime 값의 차이를 저장하는 데이터 유형 사용
- 다음 datetime 함수 사용:
 - CURRENT_DATE
 - CURRENT_TIMESTAMP
 - LOCALTIMESTAMP
 - DBTIMEZONE
 - SESSIONTIMEZONE
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 소수 표시 초를 저장하고 시간대를 추적하는 DATE와 유사한 데이터 유형을 사용하는 방법을 배웁니다. 이 단원에서는 오라클 데이터베이스에서 사용할 수 있는 일부 datetime 함수에 대해 다룹니다.

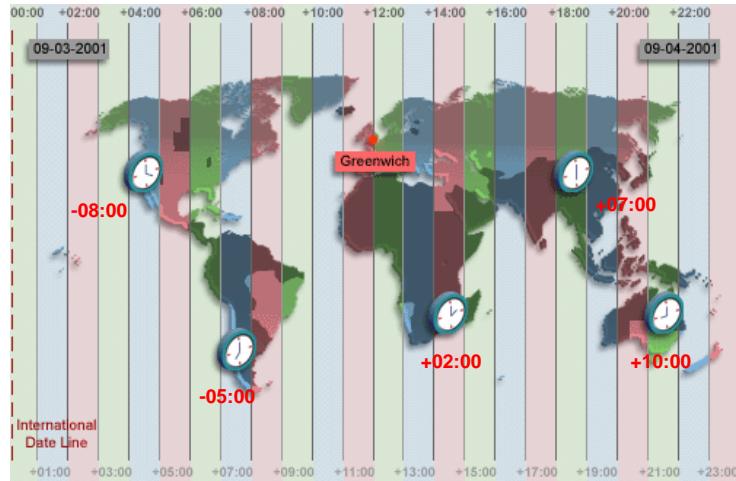
단원 내용

- CURRENT_DATE, CURRENT_TIMESTAMP
및 LOCALTIMESTAMP
- INTERVAL 데이터 유형
- 다음 함수 사용:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시간대



이 그림은 그리니치 표준시가 12:00일 때 각 시간대의 시간을 나타냅니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

시간대

하루 중 시는 지구의 자전으로 측정됩니다. 특정한 순간의 하루 중 시간은 장소에 따라 달라집니다. 영국 그리니치시로 정오이면 날짜 변경선의 시간은 자정입니다. 지구는 24개의 시간대로 나뉘어져 있으며 각 시간대는 한 시간을 나타냅니다. 영국 그리니치의 본초 자오선 시간을 GMT(그리니치 표준시)라고 합니다. GMT는 UTC(협정 세계 표준시)로 알려져 있습니다. UTC는 전세계의 다른 모든 시간대에서 참조의 기준으로 삼는 시간 표준입니다. GMT는 연중 동일하며 써머 타임 또는 일광 절약 시간의 영향을 받지 않습니다. 자오선은 북극에서 남극까지 그어진 가상의 선입니다. 이 선을 경도 0도라고 하며 다른 모든 경선은 이 선에서부터 측정됩니다. 모든 시간은 UTC를 기준으로 측정되며 모든 위치는 위도(적도에서 북쪽 또는 남쪽으로의 거리)와 경도(그리니치 자오선에서 동쪽 또는 서쪽으로의 거리)를 가집니다.

TIME_ZONE 세션 파라미터

다음과 같이 TIME_ZONE을 설정할 수 있습니다.

- 절대 오프셋
- 데이터베이스 시간대
- OS 로컬 시간대
- 명명된 지역

```
ALTER SESSION SET TIME_ZONE = '-05:00';
ALTER SESSION SET TIME_ZONE = dbtimezone;
ALTER SESSION SET TIME_ZONE = local;
ALTER SESSION SET TIME_ZONE = 'America/New_York';
```

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TIME_ZONE 세션 파라미터

오라클 데이터베이스는 날짜 및 시간 데이터는 물론 소수 표시 초 단위로 시간대를 저장할 수 있도록 지원합니다. ALTER SESSION 명령을 사용하여 유저 세션의 시간대 값을 변경할 수 있습니다. 시간대 값은 절대 오프셋, 명명된 시간대, 데이터베이스 시간대 또는 로컬 시간대로 설정할 수 있습니다.

CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP

- **CURRENT_DATE:**
 - 유저 세션의 현재 날짜를 반환합니다.
 - DATE 데이터 유형입니다.
- **CURRENT_TIMESTAMP:**
 - 유저 세션의 현재 날짜와 시간을 반환합니다.
 - TIMESTAMP WITH TIME ZONE 데이터 유형입니다.
- **LOCALTIMESTAMP:**
 - 유저 세션의 현재 날짜와 시간을 반환합니다.
 - TIMESTAMP 데이터 유형입니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP

CURRENT_DATE 및 CURRENT_TIMESTAMP 함수는 각각 현재 날짜 및 현재 시간 기록을 반환합니다. CURRENT_DATE의 데이터 유형은 DATE입니다. CURRENT_TIMESTAMP의 데이터 유형은 TIMESTAMP WITH TIME ZONE입니다. 반환되는 값은 함수를 실행하는 SQL 세션의 시간대 범위를 표시합니다. 시간대 범위는 현지 시간과 UTC 시간의 차이(시 및 분으로 표시)입니다. TIMESTAMP WITH TIME ZONE 데이터 유형의 형식은 다음과 같습니다.

TIMESTAMP[(fractional_seconds_precision)] WITH TIME ZONE

여기서 fractional_seconds_precision은 선택적으로 SECOND datetime 필드의 소수 부분에 들어갈 자릿수를 지정하며 0~9 범위의 숫자가 될 수 있습니다. 기본값은 6입니다.

LOCALTIMESTAMP 함수는 세션 시간대의 현재 날짜 및 시간을 반환합니다.

LOCALTIMESTAMP와 CURRENT_TIMESTAMP의 차이는 LOCALTIMESTAMP가 TIMESTAMP 값을 반환하는 반면 CURRENT_TIMESTAMP는 TIMESTAMP WITH TIME ZONE 값을 반환한다는 것입니다.

이 두 함수는 NLS(국가별 언어 지원)를 구분합니다. 즉, 결과는 현재 NLS 달력 및 datetime 형식이 됩니다.

참고: SYSDATE 함수는 현재 날짜와 시간을 DATE 데이터 유형으로 반환합니다.

Oracle Database 11g: SQL Fundamentals I 과정에서 SYSDATE 함수의 사용법을 배웠습니다.

세션의 시간대에서 날짜와 시간 비교

TIME_ZONE 파라미터가 -5:00으로 설정된 다음 각 날짜 및 시간에 대해 SELECT 문이 실행되어 차이를 비교합니다.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
ALTER SESSION SET TIME_ZONE = '-5:00';

SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL; 1

SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL; 2

SELECT SESSIONTIMEZONE, LOCALTIMESTAMP FROM DUAL; 3
```

Copyright © 2009, Oracle. All rights reserved.

세션의 시간대에서 날짜와 시간 비교

ALTER SESSION 명령은 세션의 날짜 형식을 'DD-MON-YYYY HH24:MI:SS'로 설정합니다. 이는 일(1-31)-월의 약어-4자리 연도 시(0-23):분(0-59):초(0-59)입니다.

슬라이드의 예제는 TIME_ZONE 파라미터를 -5:00으로 설정하도록 세션이 변경됨을 나타냅니다. 그런 다음 형식의 차이점을 관찰하기 위해 CURRENT_DATE, CURRENT_TIMESTAMP 및 LOCALTIMESTAMP에 대해 SELECT 문이 실행됩니다.

참고: TIME_ZONE 파라미터는 현재 SQL 세션에 대한 기본 로컬 시간대 범위를 지정합니다. TIME_ZONE은 세션 파라미터일 뿐이며 초기화 파라미터가 아닙니다. TIME_ZONE 파라미터는 다음과 같이 설정됩니다.

TIME_ZONE = '[+ | -] hh:mm'

형식 마스크 ([+ | -] hh:mm) 는 UTC 이전 또는 이후의 시와 분을 나타냅니다.

세션의 시간대에서 날짜와 시간 비교

Query 결과:

```
ALTER SESSION succeeded.
```

SESSIONTIMEZONE	CURRENT_DATE
1 -05:00	23-JUN-2009 01:34:52

1

SESSIONTIMEZONE	CURRENT_TIMESTAMP
1 -05:00	23-JUN-09 01.35.26.239882000 AM -05:00

2

SESSIONTIMEZONE	LOCALTIMESTAMP
1 -05:00	23-JUN-09 01.36.21.811798000 AM

3

Copyright © 2009, Oracle. All rights reserved.

세션의 시간대에서 날짜와 시간 비교(계속)

이 경우에 CURRENT_DATE 함수는 세션의 시간대에서 현재 날짜를 반환하고 CURRENT_TIMESTAMP 함수는 세션의 시간대에서 현재 날짜 및 시간을 TIMESTAMP WITH TIME ZONE 데이터 유형의 값으로 반환하고 LOCALTIMESTAMP 함수는 세션의 시간대에서 현재 날짜와 시간을 반환합니다.

DBTIMEZONE 및 SESSIONTIMEZONE

- 데이터베이스 시간대 값을 표시합니다.

```
SELECT DBTIMEZONE FROM DUAL;
```

	DBTIMEZONE
1	+00:00

- 세션의 시간대 값을 표시합니다.

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

	SESSIONTIMEZONE
1	-05:00

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

DBTIMEZONE 및 SESSIONTIMEZONE

DBA는 CREATE DATABASE 문의 SET TIME_ZONE 절을 지정하여 데이터베이스의 기본 시간대를 설정합니다. 이 절을 생략할 경우 기본 데이터베이스 시간대는 운영 체제의 시간대입니다. ALTER SESSION 문을 사용하여 세션에 대한 데이터베이스 시간대를 변경할 수 없습니다.

DBTIMEZONE 함수는 데이터베이스 시간대 값을 반환합니다. 반환 유형은 유저가 가장 최근의 CREATE DATABASE 또는 ALTER DATABASE 문에서 데이터베이스 시간대 값을 지정한 방식에 따라 시간대 오프셋 ('[+|-]TZH:TZM' 형식의 문자 유형) 또는 시간대 지역 이름입니다. 슬라이드의 예제에서는 TIME_ZONE 파라미터가 다음과 같은 형식으로 되어 있기 때문에 데이터베이스 시간대가 "-05:00"으로 설정됩니다.

TIME_ZONE = '[+|-] hh:mm'

SESSIONTIMEZONE 함수는 현재 세션의 시간대 값을 반환합니다. 반환 유형은 유저가 가장 최근의 ALTER SESSION 문에서 세션 시간대 값을 지정한 방식에 따라 시간대 오프셋 ('[+|-]TZH:TZM' 형식의 문자 유형) 또는 시간대 지역 이름입니다. 슬라이드의 예제는 세션 시간대가 UTC에 대해 -8시간 오프셋되었음을 보여줍니다. 데이터베이스 시간대는 현재 세션의 시간대와 다르다는 것을 확인합니다.

TIMESTAMP 데이터 유형

데이터 유형	필드
TIMESTAMP	년, 월, 일, 시, 분, 초(소수 표시 초)
TIMESTAMP WITH TIME ZONE	<p>TIMESTAMP 데이터 유형과 동일하며 또한 다음을 포함합니다.</p> <p>TIMEZONE_HOUR 및 TIMEZONE_MINUTE 또는 TIMEZONE_REGION</p>
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP 데이터 유형과 동일하며 또한 값에 시간대 오프셋을 포함합니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TIMESTAMP 데이터 유형

TIMESTAMP 데이터 유형은 DATE 데이터 유형의 확장입니다.

TIMESTAMP (fractional_seconds_precision)

이 데이터 유형은 년, 월, 일 날짜 값과 시, 분, 초 시간 값을 포함하며 여기서 fractional_seconds_precision은 SECOND datetime 페르드의 소수 부분 자릿수입니다. fractional_seconds_precision에서 허용되는 값의 범위는 0~9이고 기본값은 6입니다.

TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE

이 데이터 유형은 TIMESTAMP의 모든 값에 대해 시간대 범위 값을 포함합니다.

TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE

이 데이터 유형은 다음 예외 사항을 제외하고 TIMESTAMP의 모든 값을 포함합니다.

- 데이터는 데이터베이스에 저장될 때 데이터베이스 시간대로 정규화됩니다.
- 데이터가 검색될 때 유저는 세션 시간대의 데이터를 보게 됩니다.

TIMESTAMP 필드

Datetime 필드	유효한 값
YEAR	-4712 ~ 9999(연도 0 제외)
MONTH	01 ~ 12
DAY	01 ~ 31
HOUR	00 ~ 23
MINUTE	00 ~ 59
SECOND	00 ~ 59.9(N). 여기서 9(N)은 자릿수
TIMEZONE_HOUR	-12 ~ 14
TIMEZONE_MINUTE	00 ~ 59

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TIMESTAMP 필드

각 datetime 데이터 유형은 이러한 여러 개의 필드로 구성됩니다. Datetime은 동일한 datetime 필드를 가진 경우에만 상호 비교 및 할당이 가능합니다.

DATE와 TIMESTAMP의 차이

A

```
-- when hire_date is
of type DATE

SELECT hire_date
FROM employees;
```

HIRE_DATE
1 21-JUN-99
2 13-JAN-00
3 17-SEP-87
4 17-FEB-96
5 17-AUG-97
6 07-JUN-94
7 07-JUN-94
8 07-JUN-94

B

```
ALTER TABLE employees
MODIFY hire_date TIMESTAMP;

SELECT hire_date
FROM employees;
```

HIRE_DATE
1 21-JUN-99 12.00.00.000000000 AM
2 13-JAN-00 12.00.00.000000000 AM
3 17-SEP-87 12.00.00.000000000 AM
4 17-FEB-96 12.00.00.000000000 AM
5 17-AUG-97 12.00.00.000000000 AM
6 07-JUN-94 12.00.00.000000000 AM
7 07-JUN-94 12.00.00.000000000 AM
8 07-JUN-94 12.00.00.000000000 AM

...

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TIMESTAMP 데이터 유형: 예제

슬라이드에서 예제 A는 hire_date 열의 데이터 유형이 DATE인 경우 EMPLOYEES 테이블의 hire_date 열에 있는 데이터를 보여줍니다. 예제 B에서 테이블이 변경되어 hire_date 열의 데이터 유형이 TIMESTAMP로 되었습니다. 출력 결과가 다르게 표시되는 것을 확인할 수 있습니다. 열에 데이터가 있는 경우 DATE에서 TIMESTAMP로 변환할 수 있지만, 열이 비어 있지 않는 한 DATE 또는 TIMESTAMP에서 TIMESTAMP WITH TIME ZONE으로 변환할 수 없습니다.

TIMESTAMP에 대해 소수 표시 초의 자릿수를 지정할 수 있습니다. 이 예제와 같이 아무것도 지정되지 않으면 기본값 6으로 설정됩니다.

예를 들어 다음 명령문은 소수 표시 초의 자릿수를 7로 설정합니다.

```
ALTER TABLE employees
MODIFY hire_date TIMESTAMP(7);
```

참고: Oracle 날짜 데이터 유형은 기본적으로 이 예제에 표시된 것과 같이 나타납니다. 그러나 날짜 데이터 유형은 시간, 분, 초, AM, PM과 같은 추가 정보도 포함합니다. 이 형식으로 날짜를 구하려면 날짜 값에 형식 마스크나 함수를 적용하면 됩니다.

TIMESTAMP 데이터 유형 비교

```
CREATE TABLE web_orders
(order_date TIMESTAMP WITH TIME ZONE,
 delivery_time TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO web_orders values
(current_date, current_timestamp + 2);
```

```
SELECT * FROM web_orders;
```

ORDER_DATE	DELIVERY_TIME
23-JUN-09 01.56.39.000000000 AM -05:00	25-JUN-09 01.56.39.000000000 AM

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TIMESTAMP 데이터 유형 비교

슬라이드의 예제에서 TIMESTAMP WITH TIME ZONE 데이터 형식의 열과 TIMESTAMP WITH LOCAL TIME ZONE 데이터 형식의 열을 포함하는 새 테이블 `web_orders`가 생성됩니다. 이 테이블은 `web_order`가 발생할 때마다 채워집니다. 주문하는 유저의 시간 기록 및 시간대는 `CURRENT_DATE` 값을 기준으로 삽입됩니다. 이 테이블은 주문이 발생할 때마다 `CURRENT_TIMESTAMP`에 2일을 더한 값을 삽입하는 방식으로 채워집니다. 웹 기반 회사에서 배송을 보증하는 경우 이러한 방법으로 주문하는 사람의 시간대에 따라 배송 시간을 계산할 수 있습니다.

단원 내용

- CURRENT_DATE, CURRENT_TIMESTAMP
및 LOCALTIMESTAMP
- INTERVAL 데이터 유형
- 다음 함수 사용:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

INTERVAL 데이터 유형

- INTERVAL 데이터 유형은 두 datetime 값의 차이를 저장하는 데 사용됩니다.
- 두 가지 간격 유형:
 - year-month
 - day-time
- 간격의 자릿수:
 - 간격을 구성하는 필드의 실제 부분 집합입니다.
 - 간격 수식자에 지정됩니다.

데이터 유형	필드
INTERVAL YEAR TO MONTH	년, 월
INTERVAL DAY TO SECOND	일, 시, 분, 초(소수 표시 초)

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

INTERVAL 데이터 유형

INTERVAL 데이터 유형은 두 datetime 값의 차이를 저장하는 데 사용됩니다. 간격에는 year-month 간격과 day-time 간격의 두 종류가 있습니다. year-month 간격이 YEAR 및 MONTH 필드의 연속된 부분 집합으로 구성되는 반면, day-time 간격은 DAY, HOUR, MINUTE, SECOND로 구성되는 필드의 연속된 부분 집합으로 구성됩니다. 간격을 구성하는 필드의 실제 부분 집합은 간격의 자릿수라고 하며 간격 수식자에 지정됩니다. 연도의 일수는 달력에 따라 결정되므로 year-month 간격은 NLS 종속적인 반면 day-time 간격은 NLS 독립적입니다.

간격 수식자는 선행 필드 또는 유일한 필드의 자릿수를 지정할 수도 있으며, 후행 필드가 SECOND인 경우 소수 표시 초의 자릿수(SECOND 값의 소수 부분 자릿수)를 지정할 수도 있습니다. 자릿수가 지정되지 않은 경우 선행 필드 자릿수의 기본값은 2이고 소수 표시 초 자릿수의 기본값은 6입니다.

INTERVAL 데이터 유형(계속)

INTERVAL YEAR (year_precision) TO MONTH

이 데이터 유형은 년과 월로 기간을 저장하며, 여기서 year_precision은 YEAR datetime 필드의 자릿수입니다. 허용되는 값의 범위는 0~9이고 기본값은 6입니다.

INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)

이 데이터 유형은 일, 시, 분, 초로 기간을 저장합니다. 여기서 day_precision은 DAY datetime 필드의 최대 자릿수이고 (허용되는 값의 범위는 0~9, 기본값은 2), fractional_seconds_precision은 SECOND 필드의 소수 표시 부분의 자릿수입니다. 허용되는 값의 범위는 0~9이고 기본값은 6입니다.

INTERVAL 필드

INTERVAL 필드	간격에 대해 유효한 값
YEAR	임의의 양의 정수 또는 음의 정수
MONTH	00 ~ 11
DAY	임의의 양의 정수 또는 음의 정수
HOUR	00 ~ 23
MINUTE	00 ~ 59
SECOND	00 ~ 59.9(N) 여기서 9(N)은 자릿수

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

INTERVAL 필드

INTERVAL YEAR TO MONTH는 YEAR 및 MONTH 필드를 가질 수 있습니다.

INTERVAL DAY TO SECOND는 DAY, HOUR, MINUTE, SECOND 필드를 가질 수 있습니다.

간격 유형의 항목을 구성하는 필드의 실제 부분 집합은 간격 수식자에 의해 정의되며 이 부분 집합을 항목의 자릿수라고 합니다.

year-month 간격은 다른 year-month 간격과만 상호 비교 및 할당이 가능하고 day-time 간격은 다른 day-time 간격과만 상호 비교 및 할당이 가능합니다.

INTERVAL YEAR TO MONTH: 예제

```

CREATE TABLE warranty
(prod_id number, warranty_time INTERVAL YEAR(3) TO
MONTH);

INSERT INTO warranty VALUES (123, INTERVAL '8' MONTH);
INSERT INTO warranty VALUES (155, INTERVAL '200'
YEAR(3));
INSERT INTO warranty VALUES (678, '200-11');
SELECT * FROM warranty;

```

	PROD_ID	WARRANTY_TIME
1	123	0-8
2	155	200-0
3	678	200-11

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

INTERVAL YEAR TO MONTH 데이터 유형

INTERVAL YEAR TO MONTH는 YEAR 및 MONTH datetime 필드를 사용하여 기간을 저장합니다. INTERVAL YEAR TO MONTH를 다음과 같이 지정하십시오.

INTERVAL YEAR [(year_precision)] TO MONTH

여기에서 year_precision은 YEAR datetime 필드의 자릿수입니다. year_precision의 기본값은 2입니다.

제한 사항: 선행 필드는 후행 필드보다 값이 커야 합니다. 예를 들어, INTERVAL '0-1' MONTH TO YEAR는 유효하지 않습니다.

예제

- INTERVAL '123-2' YEAR(3) TO MONTH
123년, 2개월의 간격을 나타냅니다.
- INTERVAL '123' YEAR(3)
123년, 0개월의 간격을 나타냅니다.
- INTERVAL '300' MONTH(3)
300개월의 간격을 나타냅니다.
- INTERVAL '123' YEAR
기본 자릿수는 2인데 123은 세 자리이므로 오류를 반환합니다.

INTERVAL YEAR TO MONTH 데이터 유형(계속)

오라클 데이터베이스는 INTERVAL YEAR TO MONTH 및 INTERVAL DAY TO SECOND의 두 가지 간격 데이터 유형과 열 유형, PL/SQL 인수, 변수를 지원하며 반환 유형은 두 가지 중 하나가 됩니다. 그러나 간격 리터럴의 경우 시스템이 INTERVAL '2' YEAR 또는 INTERVAL '10' HOUR와 같은 다른 ANSI 간격 유형을 인식합니다. 이 경우 각 간격은 지원되는 두 유형 중 하나로 변환됩니다.

위의 예제에서는 WARRANTY 테이블이 생성되는데, 이 테이블은 INTERVAL YEAR(3) TO MONTH 데이터 유형을 취하는 warranty_time 열을 포함합니다. 다양한 제품에 대한 년과 월을 나타내기 위해 여러 다른 값이 테이블로 삽입됩니다. 이러한 행을 테이블에서 검색하면 년 값과 월 값이 (-)로 구분되어 표시됩니다.

INTERVAL DAY TO SECOND

데이터 유형: 예제

```
CREATE TABLE lab
( exp_id number, test_time INTERVAL DAY(2) TO SECOND);

INSERT INTO lab VALUES (100012, '90 00:00:00');
INSERT INTO lab VALUES (56098,
                      INTERVAL '6 03:30:16' DAY TO SECOND);
```

```
SELECT * FROM lab;
```

	EXP_ID	TEST_TIME
1	100012	90 0:0:0.0
2	56098	6 3:30:16.0

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

INTERVAL DAY TO SECOND 데이터 유형: 예제

슬라이드의 예제에서는 데이터 유형이 INTERVAL DAY TO SECOND인 test_time 열을 가진 lab 테이블을 생성합니다. 그런 다음 90일, 0시간, 0분, 0초를 나타내는 '90 00:00:00' 값과 6일, 3시간, 30분, 16초를 나타내는 INTERVAL '6 03:30:16' DAY TO SECOND를 삽입합니다. SELECT 문은 이 데이터가 데이터베이스에서 어떻게 표시되는지 보여줍니다.

단원 내용

- CURRENT_DATE, CURRENT_TIMESTAMP
및 LOCALTIMESTAMP
- INTERVAL 데이터 유형
- 다음 함수 사용:
 - EXTRACT
 - TZ_OFFSET
 - FROM_TZ
 - TO_TIMESTAMP
 - TO_YMINTERVAL
 - TO_DSINTERVAL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

EXTRACT

- SYSDATE에서 YEAR 구성 요소를 표시합니다.

```
SELECT EXTRACT (YEAR FROM SYSDATE) FROM DUAL;
```

	EXTRACT(YEARFROMSYSDATE)
1	2009

- MANAGER_ID가 100인 사원에 대해 HIRE_DATE에서 MONTH 구성 요소를 표시합니다.

```
SELECT last_name, hire_date,  
       EXTRACT (MONTH FROM HIRE_DATE)  
  FROM employees  
 WHERE manager_id = 100;
```

	LAST_NAME	HIRE_DATE	EXTRACT(MONTHFROMHIRE_DATE)
1	Hartstein	17-FEB-1996 00:00:00	2
2	Kochhar	21-SEP-1989 00:00:00	9
3	De Haan	13-JAN-1993 00:00:00	1
4	Raphaely	07-DEC-1994 00:00:00	12
5	Weiss	18-JUL-1996 00:00:00	7

ORACLE

Copyright © 2009, Oracle. All rights reserved.

EXTRACT

EXTRACT 식은 datetime 또는 interval 값 표현식에서 지정된 datetime 필드의 값을 추출하여 반환합니다. 사용자는 EXTRACT 함수를 사용하여 다음 구문에서 언급된 임의의 구성 요소를 추출할 수 있습니다. EXTRACT 함수의 구문은 다음과 같습니다.

```
SELECT EXTRACT ([YEAR] [MONTH] [DAY] [HOUR] [MINUTE][SECOND]  
                [TIMEZONE_HOUR] [TIMEZONE_MINUTE]  
                [TIMEZONE_REGION] [TIMEZONE_ABBR])  
  FROM [datetime_value_expression] [interval_value_expression]);
```

TIMEZONE_REGION 또는 TIMEZONE_ABBR(약어)을 추출할 때 반환된 값은 해당 시간대 이름이나 약어를 포함하는 문자열입니다. 다른 값을 추출할 때 반환되는 값은 그레고리력 형식의 날짜입니다. 시간대 값을 가진 datetime에서 추출할 경우 반환되는 값은 UTC 형식입니다.

슬라이드의 첫번째 예제에서 EXTRACT 함수는 SYSDATE에서 YEAR를 추출하는 데 사용됩니다. 슬라이드의 두번째 예제에서 EXTRACT 함수는 EMPLOYEE_ID가 100인 관리자에게 보고하는 사원에 대해 EMPLOYEES 테이블의 HIRE_DATE 열에서 MONTH를 추출하는 데 사용됩니다.

TZ_OFFSET

'US/Eastern', 'Canada/Yukon' 및 'Europe/London'
시간대에 대한 시간대 오프셋을 표시합니다.

```
SELECT TZ_OFFSET('US/Eastern'),
       TZ_OFFSET('Canada/Yukon'),
       TZ_OFFSET('Europe/London')
  FROM DUAL;
```

	TZ_OFFSET('US/EASTERN')	TZ_OFFSET('CANADA/YUKON')	TZ_OFFSET('EUROPE/LONDON')
1	-04:00	-07:00	+01:00

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TZ_OFFSET

TZ_OFFSET 함수는 입력된 값에 해당하는 시간대 오프셋을 반환합니다. 반환 값은 명령문이 실행되는 날짜에 따라 달라집니다. 예를 들어, TZ_OFFSET 함수가 값 -08:00을 반환하는 경우 이 값은 명령이 실행된 시간대가 UTC로부터 8시간 이후임을 나타냅니다. 사용자는 적합한 시간대 이름, UTC로부터의 시간대 오프셋(자체를 반환함) 또는 키워드 SESSIONTIMEZONE 또는 DBTIMEZONE을 입력할 수 있습니다. TZ_OFFSET 함수의 구문은 다음과 같습니다.

```
TZ_OFFSET ( [ 'time_zone_name' ] '[+ | -] hh:mm' ]
            [ SESSIONTIMEZONE ] [ DBTIMEZONE ]
```

Fold Motor Company는 US/Eastern 시간대에 속하는 USA Michigan에 본사가 있습니다. 이 회사의 대표 이사인 Fold 씨는 캐나다 지사의 부사장과 유럽 지사의 부사장과 함께 전화 회의를 하려고 합니다. 이 두 명의 부사장은 각각 Canada/Yukon 및 Europe/London 시간대에 있습니다. Fold 씨는 회사의 고위 경영진이 회의에 참석할 수 있는지 확인하기 위해 이 두 장소의 시간을 알아보려고 합니다. 그의 비서인 Scott은 예제에 표시된 query를 실행하여 다음과 같은 결과를 얻습니다.

- 'US/Eastern' 시간대는 UTC보다 4시간 늦습니다.
- 'Canada/Yukon' 시간대는 UTC보다 7시간 늦습니다.
- 'Europe/London' 시간대는 UTC보다 1시간 빠릅니다.

TZ_OFFSET(계속)

유효한 시간대 이름 값 리스트를 보려면 V\$TIMEZONE_NAMES Dynamic Performance 뷔를 query하면 됩니다.

```
SELECT * FROM V$TIMEZONE_NAMES;
```

TZNAME	TZABBREV
1 Africa/Abidjan	LMT
2 Africa/Abidjan	GMT
3 Africa/Accra	LMT
4 Africa/Accra	GMT
5 Africa/Accra	GHST

FROM_TZ

'Australia/North' 시간대 지역에 대해 TIMESTAMP
값 '2000-03-28 08:00:00'을 TIMESTAMP WITH
TIME ZONE 값으로 표시합니다.

```
SELECT FROM_TZ(TIMESTAMP
    '2000-07-12 08:00:00', 'Australia/North')
FROM DUAL;
```

	FROM_TZ(TIMESTAMP'2000-07-12 08:00:00','AUSTRALIA/NORTH')
1	12-JUL-00 08.00.00.000000000 AM AUSTRALIA/NORTH

Copyright © 2009, Oracle. All rights reserved.

FROM_TZ

FROM_TZ 함수는 TIMESTAMP 값을 TIMESTAMP WITH TIME ZONE 값으로 변환합니다.

FROM_TZ 함수의 구문은 다음과 같습니다.

```
FROM_TZ(TIMESTAMP timestamp_value, time_zone_value)
```

여기서 time_zone_value는 'TZH:TZM' 형식의 문자열이거나 선택 사항인 TZD 형식의 TZR (시간대 지역)로 문자열을 반환하는 문자 표현식입니다. TZD는 일광 절약 시간 정보를 포함하는 시간대 문자열의 약어입니다. TZR은 datetime 입력 문자열에서 시간대 지역을 나타냅니다. 예제에서는 'Australia/North'이고 미국/태평양 표준시의 경우 'PST', 미국/태평양 일광 절약 시간의 경우 'PDT' 등입니다.

슬라이드의 예제는 TIMESTAMP 값을 TIMESTAMP WITH TIME ZONE으로 변환합니다.

참고: TZR 및 TZD 형식 요소에 대해 유효한 값 리스트를 보려면 V\$TIMEZONE_NAMES Dynamic Performance 뷰를 query합니다.

TO_TIMESTAMP

문자열 '2007-03-06 11:00:00'을 TIMESTAMP 값으로 표시합니다.

```
SELECT TO_TIMESTAMP ('2007-03-06 11:00:00',
                      'YYYY-MM-DD HH:MI:SS')
FROM DUAL;
```

```
TO_TIMESTAMP('2007-03-0611:00:00','YYYY-MM-DDHH:MI:SS')
06-MAR-07 11.00.00.000000000
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

TO_TIMESTAMP

TO_TIMESTAMP 함수는 CHAR, VARCHAR2, NCHAR, NVARCHAR2 데이터 유형의 문자열을 TIMESTAMP 데이터 유형의 값으로 변환합니다. TO_TIMESTAMP 함수의 구문은 다음과 같습니다.

```
TO_TIMESTAMP (char,[fmt],[ 'nlsparam' ])
```

선택적 fmt는 char 형식을 지정합니다. fmt를 생략할 경우 문자열은 TIMESTAMP 데이터 유형의 기본 형식이어야 합니다. 선택 사항인 nlsparam은 월 및 일 이름과 약어가 반환되는 언어를 지정합니다. 이 인수는 다음과 같은 형식일 수 있습니다.

```
'NLS_DATE_LANGUAGE = language'
```

nlsparam을 생략할 경우 이 함수는 사용자 세션에 대해 기본 날짜 언어를 사용합니다.

슬라이드의 예제는 문자열을 TIMESTAMP의 값으로 변환합니다.

참고: TO_TIMESTAMP_TZ 함수는 CHAR, VARCHAR2, NCHAR, NVARCHAR2 데이터 유형의 문자열을 TIMESTAMP WITH TIME ZONE 데이터 유형의 값으로 변환합니다. 이 함수에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g Release 1 (11.1)*을 참조하십시오.

TO_YMINTERVAL

DEPARTMENT_ID가 20인 부서에서 근무하는 사원에 대해 채용 날짜로부터 1년 2개월이 지난 날짜를 표시합니다.

```
SELECT hire_date,
       hire_date + TO_YMINTERVAL('01-02') AS
       HIRE_DATE_YMININTERVAL
FROM employees
WHERE department_id = 20;
```

	HIRE_DATE	HIRE_DATE_YMININTERVAL
1	17-FEB-1996 00:00:00	17-APR-1997 00:00:00
2	17-AUG-1997 00:00:00	17-OCT-1998 00:00:00

Copyright © 2009, Oracle. All rights reserved.

TO_YMINTERVAL

TO_YMINTERVAL 함수는 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2 데이터 유형의 문자열을 INTERVAL YEAR TO MONTH 데이터 유형으로 변환합니다. INTERVAL YEAR TO MONTH 데이터 유형은 YEAR 및 MONTH datetime 필드를 사용하여 기간을 저장합니다. INTERVAL YEAR TO MONTH의 형식은 다음과 같습니다.

INTERVAL YEAR [(year_precision)] TO MONTH

여기에서 year_precision은 YEAR datetime 필드의 자릿수입니다. year_precision의 기본값은 2입니다. TO_YMINTERVAL 함수의 구문은 다음과 같습니다.

TO_YMINTERVAL (char)

여기에서 char은 변환될 문자열입니다.

슬라이드의 예제는 EMPLOYEES 테이블의 부서 20에서 근무하는 사원에 대해 채용 날짜로부터 1년 2개월이 지난 날짜를 계산합니다.

TO_DSINTERVAL

모든 사원에 대해 채용 날짜 이후 100일 10시간이 지난 날짜를 표시합니다.

```
SELECT last_name,
       TO_CHAR(hire_date, 'mm-dd-yy:hh:mi:ss') hire_date,
       TO_CHAR(hire_date +
               TO_DSINTERVAL('100 10:00:00'),
               'mm-dd-yy:hh:mi:ss') hiredate2
  FROM employees;
```

	LAST_NAME	HIRE_DATE	HIREDATE2
1	OConnell	06-21-99:12:00:00	09-29-99:10:00:00
2	Grant	01-13-00:12:00:00	04-22-00:10:00:00
3	Whalen	09-17-87:12:00:00	12-26-87:10:00:00
4	Hartstein	02-17-96:12:00:00	05-27-96:10:00:00
5	Fay	08-17-97:12:00:00	11-25-97:10:00:00
6	Mavris	06-07-94:12:00:00	09-15-94:10:00:00
7	Baer	06-07-94:12:00:00	09-15-94:10:00:00
8	Higgins	06-07-94:12:00:00	09-15-94:10:00:00
...			

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

TO_DSINTERVAL

TO_DSINTERVAL은 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2 데이터 유형을 INTERVAL DAY TO SECOND 데이터 유형으로 변환합니다.

슬라이드의 예제에서는 채용 날짜로부터 100일 10시간이 지난 날짜를 구합니다.

일광 절약 시간

- **4월의 첫번째 일요일**
 - 시간이 01:59:59 AM에서 03:00:00 AM으로 건너뜁니다.
 - 02:00:00 AM에서 02:59:59 AM 사이의 값은 유효하지 않습니다.
- **10월의 마지막 일요일**
 - 시간이 02:00:00 AM에서 01:00:01 AM으로 건너뜁니다.
 - 01:00:01 AM에서 02:00:00 AM 사이의 값은 두 번 나타나기 때문에 모호해집니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

일광 절약 시간 (DST)

대부분의 서방 국가에서는 여름 몇 개월 동안 시계를 1시간 앞당깁니다. 이 기간을 일광 절약 시간이라고 합니다. 미국, 멕시코 및 캐나다의 경우 일광 절약 시간은 4월의 첫번째 일요일부터 10월의 마지막 일요일까지 지속됩니다. 유럽 연합 국가에서도 일광 절약 시간을 지키지만 써머 타임 기간이라고 부릅니다. 유럽의 써머 타임 기간은 북미 지역보다 한 주 먼저 시작하며 끝나는 시점은 동일합니다.

오라클 데이터베이스는 지정된 시간대 지역에서 일광 절약 시간이 시행되는지 여부를 자동으로 확인하고 그에 따라 로컬 시간을 반환합니다. Datetime 값은 경계 지역인 경우를 제외한 모든 경우에 오라클 데이터베이스가 지정된 지역에 일광 절약 시간이 적용되고 있는지 판단하기에 충분한 정보를 제공합니다. 경계 지역인 경우 일광 절약 시간이 적용되거나 해제되는 시간 동안 발생합니다. 예를 들어, 미국/동부 지역의 경우 일광 절약 시간이 적용되는 시점에서 시간이 01:59:59 AM에서 03:00:00 AM으로 변경됩니다. 02:00:00 AM과 02:59:59 AM 사이의 1시간 간격은 존재하지 않습니다. 일광 절약 시간이 해제될 때는 시간이 02:00:00 AM에서 다시 01:00:01 AM로 변경되어 01:00:01 AM과 02:00:00 AM 사이의 1시간 간격이 반복됩니다.

일광 절약 시간 (DST)(계속)

ERROR_ON_OVERLAP_TIME

ERROR_ON_OVERLAP_TIME은 겹치는 기간에 datetime이 발생했는데 기간을 구별하는 시간대 약어가 지정되지 않았을 경우 오류 발생을 통보하는 세션 파라미터입니다.

예를 들어, 일광 절약 시간이 10월 31일 02:00:01 AM에 끝나는 경우 겹치는 기간은 다음과 같습니다.

- 10/31/2004 01:00:01 AM to 10/31/2004 02:00:00 AM (EDT)
- 10/31/2004 01:00:01 AM to 10/31/2004 02:00:00 AM (EST)

이 두 기간 중 하나에서 발생하는 datetime 문자열을 입력하는 경우 시스템이 기간을 결정할 수 있도록 입력 문자열에 시간대 약어 (예: EDT 또는 EST)를 지정해야 합니다. 이 시간대 약어가 없으면 시스템은 다음을 수행합니다.

ERROR_ON_OVERLAP_TIME 파라미터가 FALSE인 경우 입력 시간이 표준 시간 (예: EST)이라고 가정합니다. 그렇지 않은 경우 오류가 발생합니다.

퀴즈

다음 중 TIME_ZONE 세션 파라미터를 설정할 수 있는 시간대
값을 고르십시오.

1. 상대 오프셋
2. 데이터베이스 시간대
3. OS 로컬 시간대
4. 명명된 지역



Copyright © 2009, Oracle. All rights reserved.

정답: 2, 3, 4

요약

이 단원에서는 다음 함수의 사용 방법에 대해 설명했습니다.

- CURRENT_DATE
- CURRENT_TIMESTAMP
- LOCALTIMESTAMP
- DBTIMEZONE
- SESSIONTIMEZONE
- EXTRACT
- TZ_OFFSET
- FROM_TZ
- TO_TIMESTAMP
- TO_YMINTERVAL
- TO_DSINTERVAL

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 오라클 데이터베이스에서 사용할 수 있는 일부 datetime 함수에 대해 설명했습니다.

연습 5: 개요

이 연습에서는 `datetime` 함수의 사용에 대해 다룹니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 5: 개요

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

Subquery를 사용하여 데이터 검색

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- Correlated Subquery를 사용하여 행 갱신 및 삭제
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 SELECT 문의 여러 열 subquery 및 FROM 절의 subquery를 작성하는 방법에 대해 배웁니다. 또한 스칼라 subquery, correlated subquery 및 WITH 절을 사용하여 문제를 해결하는 방법을 설명합니다.

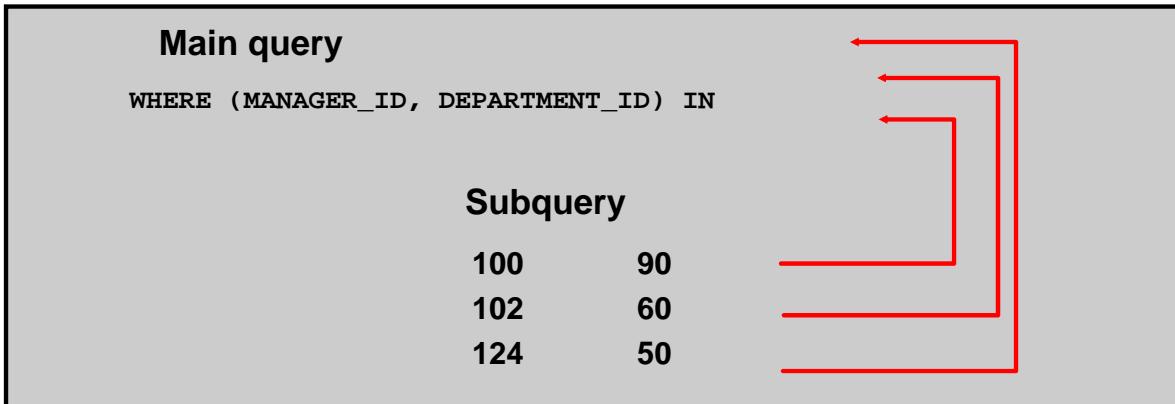
단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

여러 열 Subquery



Main query의 각 행은 여러 행 및 여러 열 subquery의 값과 비교됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

여러 열 Subquery

지금까지는 inner SELECT 문에서 한 열만 반환되고 이 열을 사용하여 상위 SELECT 문의 표현식을 평가하는 단일 행 subquery 및 여러 행 subquery를 작성했습니다. 둘 이상의 열을 비교하려면 논리 연산자를 사용하여 WHERE 절을 작성해야 합니다. 여러 열 subquery를 사용하여 중복되는 WHERE 조건을 단일 WHERE 절로 조합할 수 있습니다.

구문

```
SELECT      column, column, ...
FROM        table
WHERE       (column, column, ...) IN
            (SELECT column, column, ...
             FROM  table
             WHERE condition);
```

슬라이드의 그림은 main query의 MANAGER_ID 및 DEPARTMENT_ID 값을 subquery에서 검색된 MANAGER_ID 및 DEPARTMENT_ID 값과 비교하여 보여줍니다. 비교되는 열 수가 두 개 이상이므로 이 예제는 여러 열 subquery로 적합합니다.

참고: 다음 몇 개의 슬라이드 예제를 실행하기 전에 lab_06_insert_empdata.sql 파일을 사용하여 emp1_demo 테이블을 생성하고 데이터를 채워야 합니다.

열 비교

Subquery를 포함한 여러 열 비교 유형:

- **비쌍 방식 비교**
- **쌍 방식 비교**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

쌍 방식 비교와 비쌍 방식 비교

Subquery를 포함한 여러 열 비교는 비쌍방식 비교 또는 쌍방식 비교가 될 수 있습니다. "Daniel과 동일한 부서에서 근무하고 동일한 관리자의 감독을 받는 사원의 세부 정보를 표시합니다." 예제를 고려할 경우 다음 명령문을 사용하여 정확한 결과를 얻을 수 있습니다.

```
SELECT first_name, last_name, manager_id, department_id
FROM emp1_demo
WHERE manager_id IN (SELECT manager_id
                      FROM emp1_demo
                      WHERE first_name = 'Daniel')
      AND department_id IN (SELECT department_id
                            FROM emp1_demo
                            WHERE first_name = 'Daniel');
```

EMPL_DEMO 테이블에는 한 명의 "Daniel"만 있습니다(사원 108의 감독을 받고 부서 100에서 근무하는 Daniel Faviet). 그러나 subquery가 두 개 이상의 행을 반환하는 경우 결과가 정확하지 않을 수도 있습니다. 예를 들어, 동일한 query를 실행하지만 "Daniel"을 "John"으로 대체하면 잘못된 결과를 얻을 수 있습니다. 그 이유는 department_id와 manager_id의 조합이 중요하기 때문입니다. 이 query에 대해 정확한 결과를 얻으려면 쌍방식 비교를 사용해야 합니다.

쌍 방식 비교 Subquery

이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원에 대한 세부 정보를 표시합니다.

```
SELECT employee_id, manager_id, department_id
  FROM emp1_demo
 WHERE (manager_id, department_id) IN
       (SELECT manager_id, department_id
        FROM emp1_demo
        WHERE first_name = 'John')
  AND first_name <> 'John';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

쌍 방식 비교 Subquery

슬라이드의 예제는 EMPL_DEMO 테이블에 있는 각 행의 MANAGER_ID 열과 DEPARTMENT_ID 열의 값 조합을 FIRST_NAME이 "John"인 사원의 MANAGER_ID 열 및 DEPARTMENT_ID 열의 값과 비교합니다. 먼저 FIRST_NAME이 "John"인 사원의 MANAGER_ID 및 DEPARTMENT_ID 값을 검색하는 subquery가 실행됩니다. 이 subquery는 다음을 반환합니다.

	MANAGER_ID	DEPARTMENT_ID
1	108	100
2	123	50
3	100	80

쌍방식 비교 Subquery(계속)

이러한 값들은 EMPL_DEMO 테이블에 있는 각 행의 MANAGER_ID 열 및 DEPARTMENT_ID 열과 비교됩니다. 비교가 일치하면 행이 표시됩니다. 출력에서 FIRST_NAME이 "John"인 사원의 레코드는 표시되지 않습니다. 다음은 슬라이드의 query를 출력한 결과입니다.

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	113	108	100
2	112	108	100
3	111	108	100
4	109	108	100
5	195	123	50
6	194	123	50
7	193	123	50
8	192	123	50
9	140	123	50
10	138	123	50
11	137	123	50
12	149	100	80
13	148	100	80
14	147	100	80
15	146	100	80

비쌍 방식 비교 Subquery

이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원의 세부 정보를 표시합니다.

```

SELECT employee_id, manager_id, department_id
FROM empl_demo
WHERE manager_id IN
    (SELECT manager_id
     FROM empl_demo
     WHERE first_name = 'John')
AND department_id IN
    (SELECT department_id
     FROM empl_demo
     WHERE first_name = 'John')
AND first_name <> 'John';

```

Copyright © 2009, Oracle. All rights reserved.

비쌍 방식 비교 Subquery

이 예제는 열의 비쌍방식 비교를 보여줍니다. 먼저 FIRST_NAME이 "John"인 사원의 MANAGER_ID 값을 검색하는 subquery가 실행됩니다. 마찬가지로, FIRST_NAME이 "John"인 사원의 DEPARTMENT_ID 값을 검색하는 두번째 subquery가 실행됩니다. MANAGER_ID 및 DEPARTMENT_ID 열의 검색된 값은 EMPL_DEMO 테이블에 있는 각 행의 MANAGER_ID 및 DEPARTMENT_ID 열과 비교됩니다. EMPL_DEMO 테이블에 있는 행의 MANAGER_ID 열이 inner subquery에 의해 검색된 MANAGER_ID 값과 일치하고, EMPL_DEMO 테이블에 있는 행의 DEPARTMENT_ID 열이 두번째 subquery에 의해 검색된 DEPARTMENT_ID 값과 일치하면 레코드가 표시됩니다.

비쌍방식 비교 Subquery(계속)

다음은 이전 슬라이드의 query를 출력한 결과입니다.

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	109	108	100
2	111	108	100
3	112	108	100
4	113	108	100
5	120	100	50
6	121	100	50
7	122	100	50
8	123	100	50
9	124	100	50
10	137	123	50
11	138	123	50
12	140	123	50
13	192	123	50
14	193	123	50
15	194	123	50
16	195	123	50
17	146	100	80
18	147	100	80
19	148	100	80
20	149	100	80

이 query는 쌍방식 비교보다 많은 추가 행을 검색합니다. (manager_id=100이고 department_id=50 또는 80인 조합을 가진 이름이 "John"인 사원이 없는 경우에도 해당 조합을 가진 행을 검색합니다.)

단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

스칼라 Subquery 표현식

- 스칼라 subquery 표현식은 한 행에서 정확히 하나의 열 값을 반환하는 subquery입니다.
- 스칼라 subquery는 다음에서 사용할 수 있습니다.
 - DECODE 및 CASE의 조건 및 표현식 부분
 - GROUP BY를 제외한 SELECT의 모든 절
 - UPDATE 문의 SET 절 및 WHERE 절

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL의 스칼라 Subquery

한 행에서 정확히 하나의 열 값만 반환하는 subquery를 스칼라 subquery라고 합니다. 복합 WHERE 절과 논리 연산자를 사용하여 두 개 이상의 열을 비교하도록 작성된 여러 열 subquery는 스칼라 subquery가 아닙니다.

스칼라 subquery의 표현식 값은 subquery의 선택 리스트 항목의 값입니다. Subquery에서 반환되는 행이 없는 경우 스칼라 subquery 표현식의 값은 NULL입니다. Subquery가 두 개 이상의 행을 반환하면 Oracle 서버에서 오류를 반환합니다. Oracle 서버는 항상 SELECT 문에 스칼라 subquery를 사용할 수 있도록 지원합니다. 다음에서 스칼라 subquery를 사용할 수 있습니다.

- DECODE 및 CASE의 조건 및 표현식 부분
- GROUP BY를 제외한 SELECT의 모든 절
- UPDATE 문의 SET 절 및 WHERE 절

그러나 다음에서 스칼라 subquery는 유효한 표현식이 아닙니다.

- 열의 기본값과 클러스터의 해시 표현식
- DML(데이터 조작어) 문의 RETURNING 절
- 함수 기반 인덱스의 기준
- GROUP BY 절, CHECK 제약 조건 및 WHEN 조건
- CONNECT BY 절
- CREATE PROFILE과 같이 query와 관계없는 문

스칼라 Subquery: 예제

- CASE 표현식의 스칼라 Subquery:

```
SELECT employee_id, last_name,
       (CASE
        WHEN department_id = 20
            (SELECT department_id
             FROM departments
              WHERE location_id = 1800)
        THEN 'Canada' ELSE 'USA' END) location
  FROM employees;
```

- ORDER BY 절의 스칼라 Subquery:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

스칼라 Subquery: 예제

슬라이드의 첫번째 예제는 스칼라 subquery를 CASE 표현식에 사용하는 방법을 보여줍니다. Inner query는 값 20을 반환합니다. 이 값은 위치 ID가 1800인 부서의 부서 ID입니다. Outer query의 CASE 표현식은 inner query의 결과를 사용하여 사원 ID, 성 및 Canada 또는 USA 값을 표시합니다. 이 위치 값은 outer query에 의해 검색된 레코드의 부서 ID가 20인지 아닌지 여부에 따라 결정됩니다.

다음은 슬라이드의 첫번째 예제의 결과입니다.

...

	EMPLOYEE_ID	LAST_NAME	LOCATION
1	198	O'Connell	USA
2	199	Grant	USA
3	200	Whalen	USA
4	201	Hartstein	Canada
5	202	Fay	Canada
6	203	Mavris	USA

스칼라 Subquery: 예제(계속)

슬라이드의 두번째 예제는 스칼라 subquery를 ORDER BY 절에 사용하는 방법을 보여줍니다. 이 예제에서는 EMPLOYEES 테이블의 DEPARTMENT_ID를 DEPARTMENTS 테이블의 DEPARTMENT_ID와 일치시키고 DEPARTMENT_NAME을 기준으로 출력을 정렬합니다. 이 비교는 ORDER BY 절의 스칼라 subquery에서 수행됩니다. 다음은 두번째 예제의 결과입니다.

	EMPLOYEE_ID	LAST_NAME
1	205	Higgins
2	206	Gietz
3	200	Whalen
4	100	King
5	101	Kochhar
6	102	De Haan
7	112	Urman
8	108	Greenberg
9	109	Faviet
...		

두번째 예제에서는 correlated subquery가 사용됩니다. Correlated subquery에서 subquery는 상위 명령문에서 참조된 테이블의 열을 참조합니다. Correlated subquery는 이 단원의 뒷부분에서 설명합니다.

단원 내용

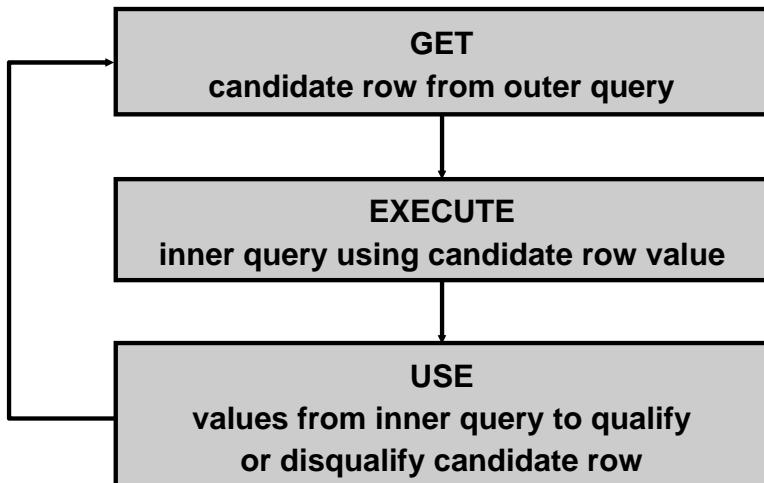
- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- **Correlated Subquery로 문제 해결**
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

Correlated Subquery

Correlated subquery는 행 단위 처리에 사용됩니다. 각 subquery는 outer query의 모든 행에 대해 한번씩 실행됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Correlated Subquery

Oracle 서버는 subquery가 상위 명령문에서 참조된 테이블의 열을 참조할 때 correlated subquery를 수행합니다. Correlated subquery는 상위 명령문에 의해 처리된 각 행에 대해 한 번 평가됩니다. 상위 명령문은 SELECT, UPDATE 또는 DELETE 문일 수 있습니다.

Nested Subquery와 Correlated Subquery 비교

일반 nested subquery를 사용하면 inner SELECT query가 먼저 한 번 실행되어 main query에서 사용할 값을 반환합니다. 그러나 correlated subquery는 outer query에서 고려한 각 후보 행에 대해 한 번 실행됩니다. 즉, inner query를 outer query가 제어합니다.

Nested Subquery 실행

- Inner query가 먼저 실행되어 값을 찾습니다.
- Outer query는 inner query의 값을 사용하여 한 번 실행됩니다.

Correlated Subquery 실행

- Outer query에 의해 패치(fetch)된 후보 행을 가져옵니다.
- 후보 행의 값을 사용하여 inner query를 실행합니다.
- Inner query 결과 값을 사용하여 후보 행을 지정하거나 지정을 취소합니다.
- 남아 있는 후보 행이 없을 때까지 반복합니다.

Correlated Subquery

Subquery는 상위 query에 있는 테이블의 열을 참조합니다.

```
SELECT column1, column2, ...
  FROM table1 Outer_table
 WHERE column1 operator
       (SELECT column1, column2
        FROM table2
        WHERE expr1 =
              Outer_table.expr2);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Correlated Subquery(계속)

Correlated subquery는 테이블에 있는 모든 행을 읽고 관련 데이터에 대해 각 행의 값을 비교하는 방법입니다. 이 query는 subquery가 main query에서 고려된 각 후보 행에 대해 다른 결과 또는 결과 집합을 반환해야 할 때마다 사용됩니다. 즉, 상위 명령문에서 처리한 각 행의 값에 따라 응답이 달라지는 multipart 질문에 응답할 때 correlated subquery를 사용합니다.

Oracle 서버는 subquery가 상위 query의 테이블의 열을 참조할 때 correlated subquery를 수행합니다.

참고: correlated subquery에서는 ANY 및 ALL 연산자를 사용할 수 있습니다.

Correlated Subquery 사용

자신의 부서의 평균 급여보다 많은 급여를 받는 사원을 모두 찾습니다.

```
SELECT last_name, salary, department_id
  FROM employees outer_table
 WHERE salary >
       (SELECT AVG(salary)
        FROM   employees inner_table
        WHERE inner_table.department_id =
              outer_table.department_id);
```

Outer query의 행이 처리될 때마다 inner query가 평가됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Correlated Subquery 사용

슬라이드의 예제는 자신의 부서의 평균 급여보다 많은 급여를 받는 사원을 확인합니다. 여기에서 correlated subquery는 특히 각 부서의 평균 급여를 계산합니다.

Outer query와 inner query 모두 FROM 절에 EMPLOYEES 테이블을 사용하므로 명확한 구분을 위해 outer SELECT 문의 EMPLOYEES에 alias가 제공됩니다. Alias는 전체 SELECT 문을 보다 읽기 쉽게 만듭니다. 내부 명령문은 inner 테이블 열과 outer 테이블 열을 구분하지 못하기 때문에 alias가 없을 경우 query가 제대로 작동하지 않습니다.

Correlated Subquery 사용

두 번 이상 직무가 바뀐 사원의 세부 사항을 표시합니다.

```
SELECT e.employee_id, last_name, e.job_id
FROM employees e
WHERE 2 <= (SELECT COUNT(*)
              FROM job_history
             WHERE employee_id = e.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	200	Whalen	AD_ASST
2	101	Kochhar	AD_VP
3	176	Taylor	SA_REP

Copyright © 2009, Oracle. All rights reserved.

Correlated Subquery 사용(계속)

슬라이드의 예제는 직무가 적어도 두 번 이상 변경된 사원의 세부 정보를 표시합니다.

Oracle 서버는 다음과 같이 correlated subquery를 평가합니다.

- Outer query에 지정된 테이블에서 행을 선택합니다. 이 행이 현재 후보 행이 됩니다.
- 이 후보 행의 subquery에서 참조된 열 값을 저장합니다.(슬라이드의 예제에서 subquery에서 참조된 열은 E.EMPLOYEE_ID입니다.)
- Outer query의 후보 행의 값을 참조하는 조건으로 subquery를 수행합니다.(슬라이드 예제에서 COUNT(*) 그룹 함수는 2단계에서 얻은 E.EMPLOYEE_ID 열의 값에 준하여 평가됩니다.)
- 3단계에서 수행된 subquery 결과를 기반으로 outer query의 WHERE 절을 평가하고, 이에 따라 해당 후보 행의 출력 여부가 결정됩니다.(예제에서 subquery에 의해 평가된 사원의 직무 변경 횟수는 outer query의 WHERE 절에 있는 2와 비교됩니다. 조건이 충족되면 해당 사원 레코드가 표시됩니다.)
- 테이블의 모든 행이 처리될 때까지 테이블의 다음 후보 행에 대해 이 과정을 반복합니다.

상관(correlation)은 subquery에서 outer query의 요소를 사용하여 설정됩니다. 이 예제에서 subquery에 있는 테이블의 EMPLOYEE_ID를 outer query에 있는 테이블의 EMPLOYEE_ID와 비교합니다.

단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- **EXISTS 및 NOT EXISTS 연산자 사용**
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

EXISTS 연산자 사용

- EXISTS 연산자는 subquery의 결과 집합에 행이 존재하는지 테스트합니다.
- Subquery 행 값이 있을 경우
 - 검색이 inner query에서 계속 수행되지 않습니다.
 - 조건에는 TRUE로 플래그 지정됩니다.
- Subquery 행 값이 없을 경우
 - 조건은 FALSE로 플래그가 지정됩니다.
 - Inner query에서도 검색이 수행됩니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

EXISTS 연산자

중첩되는 SELECT 문에서는 모든 논리 연산자를 사용할 수 있으며 EXISTS 연산자를 사용할 수 있습니다. 이 연산자는 outer query로 검색된 값이 inner query로 검색된 값의 결과 집합에 존재하는지 여부를 테스트하기 위해 correlated subquery에서 자주 사용됩니다. Subquery가 적어도 한 행을 반환하면 이 연산자는 TRUE를 반환합니다. 값이 없으면 FALSE를 반환합니다. 따라서 NOT EXISTS는 outer query에 의해 검색된 값이 inner query에 의해 검색된 값의 결과 집합의 일부가 아닌지 여부를 테스트합니다.

EXISTS 연산자 사용

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE EXISTS ( SELECT 'X'
                FROM employees
                WHERE manager_id =
                      outer.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	201	Hartstein	MK_MAN	20
2	205	Higgins	AC_MGR	110
3	100	King	AD_PRES	90
4	101	Kochhar	AD_VP	90
5	102	De Haan	AD_VP	90
6	103	Hunold	IT_PROG	60
7	108	Greenberg	FI_MGR	100
8	114	Raphaely	PU_MAN	30

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

EXISTS 연산자 사용

EXISTS 연산자를 사용하면 다음 조건을 가진 관리자와 사원 번호에 대해 일치하는 사원이 적어도 한 명 이상 있을 경우 inner query에서 검색이 중단됩니다.

WHERE manager_id = outer.employee_id.

Inner SELECT query는 특정 값을 반환할 필요가 없기 때문에 상수를 선택할 수 있습니다.

사원이 없는 부서 모두 찾기

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                   FROM employees
                   WHERE department_id = d.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME
1	120	Treasury
2	130	Corporate Tax
3	140	Control And Credit
4	150	Shareholder Services
5	160	Benefits
6	170	Manufacturing
7	180	Construction

...

All Rows Fetched: 16

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

NOT EXISTS 연산자 사용

대체 방법

아래 예제에 나오는 것처럼 NOT EXISTS 연산자 대신 NOT IN 생성자를 사용할 수 있습니다.

```
SELECT department_id, department_name
FROM   departments
WHERE  department_id NOT IN(SELECT department_id
                           FROM   employees);
```

All Rows Fetched: 0

그러나 집합의 한 멤버라도 NULL 값이면 NOT IN은 FALSE로 평가됩니다. 따라서 departments 테이블에 WHERE 조건을 만족하는 열이 있더라도 query에서는 아무 행도 반환되지 않습니다.

Correlated UPDATE

한 테이블의 행에 준하여 다른 테이블의 행을 갱신할 때
correlated subquery를 사용합니다.

```
UPDATE table1 alias1
SET    column = (SELECT expression
                  FROM   table2 alias2
                  WHERE  alias1.column =
                         alias2.column);
```



Copyright © 2009, Oracle. All rights reserved.

Correlated UPDATE

UPDATE 문의 경우 한 테이블의 행에 준하여 다른 테이블의 행을 갱신할 때 correlated subquery를 사용할 수 있습니다.

Correlated UPDATE 사용

- 부서 이름을 저장할 열을 추가하여 EMPL6 테이블을 비정규화합니다.
- Correlated update를 사용하여 테이블을 채웁니다.

```
ALTER TABLE empl6
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e
SET department_name =
  (SELECT department_name
   FROM departments d
   WHERE e.department_id = d.department_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Correlated UPDATE(계속)

슬라이드 예제에서는 부서 이름을 저장하기 위한 열을 추가하여 EMPL6 테이블을 비정규화한 다음 correlated update를 사용하여 테이블을 채웁니다.

다음은 correlated update에 대한 또 다른 예제입니다.

문제

REWARDS 테이블에는 성과 기대치를 초과하는 사원 리스트가 있습니다. Correlated subquery를 사용하여 REWARDS 테이블의 행을 기반으로 EMPL6 테이블의 행을 갱신합니다.

```
UPDATE empl6
SET salary = (SELECT empl6.salary + rewards.pay_raise
               FROM rewards
              WHERE employee_id =
                    empl6.employee_id
                AND payraise_date =
                  (SELECT MAX(payraise_date)
                   FROM rewards
                  WHERE employee_id = empl6.employee_id))
 WHERE empl6.employee_id
       IN (SELECT employee_id FROM rewards);
```

Correlated UPDATE(계속)

이 예제에서는 REWARDS 테이블을 사용합니다. REWARDS 테이블에는 EMPLOYEE_ID, PAY_RAISE 및 PAYRAISE_DATE 열이 있습니다. 사원의 급여가 인상될 때마다 사원 ID, 급여 인상폭 및 인상된 급여의 수령 날짜와 같은 세부 정보가 있는 레코드가 REWARDS 테이블에 삽입됩니다. REWARDS 테이블은 한 사원에 대해 두 개 이상의 레코드를 포함할 수 있습니다. PAYRAISE_DATE 열은 사원의 가장 최근 급여 인상을 식별하는데 사용됩니다.

예제에서 EMPL6 테이블의 SALARY 열은 사원이 받은 최근의 급여 인상을 반영하도록 갱신됩니다. 이렇게 하려면 REWARDS 테이블의 인상 급여에 해당 사원의 현재 급여를 추가합니다.

Correlated DELETE

Correlated subquery를 사용하면 한 테이블의 행에 준하여 다른 테이블의 행을 삭제할 수 있습니다.

```
DELETE FROM table1 alias1
WHERE column operator
      (SELECT expression
       FROM   table2 alias2
       WHERE  alias1.column = alias2.column);
```

Copyright © 2009, Oracle. All rights reserved.

Correlated DELETE

DELETE 문의 경우 correlated subquery를 사용하면 다른 테이블에도 존재하는 행만 삭제할 수 있습니다. JOB_HISTORY 테이블에서 마지막 네 개의 직무 기록 레코드만 관리하려면 사원이 직무를 다섯번째로 바꿨을 때 JOB_HISTORY 테이블에서 해당 사원의 MIN(START_DATE)를 찾아서 가장 오래된 JOB_HISTORY 행을 삭제합니다.
다음 코드는 correlated DELETE를 사용하여 이 작업을 수행하는 방법을 보여줍니다.

```
DELETE FROM emp_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 5 > (SELECT COUNT(*)
                  FROM job_history JH
                  WHERE JH.employee_id = E.employee_id
                  GROUP BY EMPLOYEE_ID
                  HAVING COUNT(*) >= 4));
```

Correlated DELETE 사용

Correlated subquery를 사용하여 EMPL6 테이블에서
EMP_HISTORY 테이블에도 있는 행만 삭제합니다.

```
DELETE FROM empl6 E
WHERE employee_id =
  (SELECT employee_id
   FROM emp_history
   WHERE employee_id = E.employee_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Correlated DELETE(계속)

예제

이 예제에서는 두 개의 테이블이 사용됩니다. 각 테이블은 다음과 같습니다.

- EMPL6 테이블: 현재 모든 사원에 대한 세부 정보를 제공합니다.
- EMP_HISTORY 테이블: 이전 사원의 세부 정보를 제공합니다.

EMP_HISTORY는 이전 사원에 대한 데이터를 포함하므로 같은 사원의 레코드가 EMPL6 테이블과 EMP_HISTORY 테이블에 모두 있을 경우 오류가 발생할 수 있습니다. 슬라이드에 표시된 correlated subquery를 사용하여 이러한 오류 레코드를 삭제할 수 있습니다.

단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

WITH 절

- WITH 절을 사용하면 복합 query 내에서 동일한 query 블록이 두 번 이상 발생하는 경우 그 블록을 SELECT 문에 사용할 수 있습니다.
- WITH 절은 query 블록 결과를 검색하여 유저의 임시 테이블스페이스에 저장합니다.
- WITH 절은 성능을 개선할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

WITH 절

WITH 절을 사용하여 query 블록을 정의한 다음 query에 사용할 수 있습니다. WITH 절은 일반적으로 `subquery_factoring_clause`라고도 하며, 이 절을 사용하면 복합 query 내에서 이 절이 두 번 이상 나타날 때 동일한 query 블록을 SELECT 문에 재사용할 수 있습니다. 이러한 방식은 특히 query에 동일 query 블록에 대한 참조가 여러 개 있고 조인 및 집계가 있을 경우에 유용합니다.

WITH 절을 사용하면 query 블록의 평가 비용이 높고 복합 query 내에서 query 블록이 두 번 이상 나타나는 경우 동일한 query를 재사용할 수 있습니다. WITH 절을 사용할 경우 Oracle 서버는 query 블록의 결과를 검색하여 유저의 임시 테이블스페이스에 저장합니다. 이러한 방식으로 성능을 향상시킬 수 있습니다.

WITH 절의 장점

- Query를 읽기 쉽게 만듭니다.
- Query에 절이 여러 번 나타나도 한 번만 평가합니다.
- 대부분의 경우 대용량 query에 대한 성능을 향상시킬 수 있습니다.

WITH 절: 예제

**WITH 절을 사용하여 총 급여가 전체 부서의 평균 급여보다
큰 부서의 부서 이름 및 총 급여를 표시하는 query를 작성합니다.**

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

WITH 절: 예제

슬라이드의 문제를 해결하려면 아래의 중간 계산이 필요합니다.

1. 모든 부서의 총 급여를 계산하고 WITH 절을 사용하여 결과를 저장합니다.
 2. 전체 부서의 평균 급여를 계산하고 WITH 절을 사용하여 결과를 저장합니다.
 3. 첫번째 단계에서 계산된 총 급여와 두번째 단계에서 계산된 평균 급여를 비교합니다.
- 특정 부서의 총 급여가 전체 부서의 평균 급여보다 많으면 부서 이름과 해당 부서의 총 급여를 표시합니다.

이 문제의 정답은 다음 페이지에 나옵니다.

WITH 절: 예제

```

WITH
dept_costs AS (
    SELECT d.department_name, SUM(e.salary) AS dept_total
    FROM employees e JOIN departments d
    ON e.department_id = d.department_id
    GROUP BY d.department_name),
avg_cost AS (
    SELECT SUM(dept_total)/COUNT(*) AS dept_avg
    FROM dept_costs)
SELECT *
FROM dept_costs
WHERE dept_total >
    (SELECT dept_avg
    FROM avg_cost)
ORDER BY department_name;

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

WITH 절: 예제(계속)

슬라이드의 SQL 코드는 WITH 절을 사용하여 훨씬 간단하게 SQL을 작성하고 성능을 향상시킬 수 있는 상황에 대한 예제입니다. Query는 query 이름 DEPT_COSTS 및 AVG_COST를 생성하여 main query 본문에 사용합니다. 내부적으로 WITH 절은 인라인 뷰나 임시 테이블(Temporal Table)로 분석됩니다. 옵티마이저는 WITH 절의 결과를 임시로 저장하는 비용이나 이점에 따라 적절한 분석 방법을 선택합니다. 슬라이드에서 SQL 코드로 생성된 출력 결과는 다음과 같습니다.

	DEPARTMENT_NAME	DEPT_TOTAL
1	Sales	304500
2	Shipping	156400

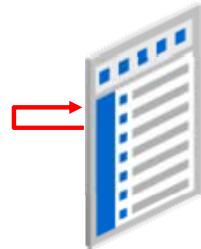
WITH 절 사용 정보

- 이 절은 항상 SELECT 문과 함께 사용됩니다.
- Query 이름은 query 이름 다음에 정의된 모든 WITH 요소 query 블록(subquery 블록 포함)과 main query 블록 자체(subquery 블록 포함)에서 볼 수 있습니다.
- Query 이름이 기존 테이블 이름과 동일한 경우, 구문 분석기는 내부에서 검색을 시작하며 query 블록 이름이 테이블 이름보다 우선합니다.
- WITH 절은 두 개 이상의 query를 보유할 수 있습니다. 각 query는 쉼표로 구분됩니다.

Recursive WITH 절

Recursive WITH 절

- Recursive query의 공식화를 가능하게 합니다.
- Recursive WITH 요소 이름이라는 이름으로 query를 작성합니다.
- 앵커와 recursive라는 두 가지 유형의 query 블록 멤버를 포함합니다.
- ANSI와 호환됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Recursive WITH 절

Oracle Database 11g Release 2에서 recursive query를 공식화할 수 있도록 WITH 절이 확장되었습니다.

Recursive WITH는 이름, 즉 *Recursive WITH 요소 이름*으로 recursive query를 정의합니다. Recursive WITH 요소 정의에는 앵커 멤버와 recursive 멤버라는 두 개 이상의 query 블록이 포함되어야 합니다. 앵커 멤버는 여러 개 있을 수 있지만 recursive 멤버는 하나만 있을 수 있습니다.

Oracle Database 11g Release 2의 Recursive WITH 절은 부분적으로 ANSI를 따릅니다. Recursive WITH를 사용하여 조직도 등의 계층 데이터를 query할 수 있습니다.

Recursive WITH 절: 예제

FLIGHTS 테이블

	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
    SELECT Source, Destin, Flight_time
    FROM Flights
    UNION ALL
        SELECT incoming.Source, outgoing.Destin,
               incoming.TotalFlightTime+outgoing.Flight_time
        FROM Reachable_From incoming, Flights outgoing
        WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	San Jose	New York	7.1
5	Los Angeles	Boston	6.9
6	San Jose	Boston	8.2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Recursive WITH 절: 예제

슬라이드의 예제 1은 두 도시 간의 항공편을 설명하는 FLIGHTS 테이블의 레코드를 보여줍니다.

예제 2의 query로 FLIGHTS 테이블을 query하여 출발지와 목적지 간의 총 비행 시간을 표시할 수 있습니다. 쿼리의 WITH Reachable_From 절에는 두 개의 분기를 포함하는 UNION ALL query가 있습니다. 첫번째 분기는 Flights 테이블의 모든 행을 선택하는 앵커 분기이고, 두번째 분기는 recursive 분기입니다. Recursive 분기는 Reachable_From의 내용을 Flights 테이블에 조인하여 이동 가능한 다른 도시를 찾아 Reachable_From의 내용에 추가합니다. Recursive 분기에서 더 이상 행을 찾을 수 없을 때 작업이 끝납니다.

예제 3은 WITH 절 요소 Reachable_From의 모든 내용을 선택하는 query의 결과입니다.

자세한 내용은 다음을 참조하십시오.

- Oracle Database SQL Language Reference 11g Release 2.0
- Oracle Database Data Warehousing Guide 11g Release 2.0

퀴즈

Correlated subquery를 사용하면 inner SELECT 문이 outer SELECT 문을 제어합니다.

1. 맞습니다.
2. 틀립니다.



Copyright © 2009, Oracle. All rights reserved.

정답: 2

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 여러 열 subquery는 두 개 이상의 열을 반환합니다.
- 여러 열 비교는 쌍방식이거나 비쌍방식일 수 있습니다.
- 여러 열 subquery는 SELECT 문의 FROM 절에 사용될 수도 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

여러 열 subquery를 사용하여 여러 WHERE 조건을 단일 WHERE 절에 결합할 수 있습니다.

여러 열 subquery의 열 비교는 쌍방식 비교이거나 비쌍방식 비교일 수 있습니다.

Subquery를 사용하여 포함 query로 처리할 테이블을 정의할 수 있습니다.

스칼라 subquery는 다음에서 사용할 수 있습니다.

- DECODE 및 CASE의 조건 및 표현식 부분
- GROUP BY를 제외한 SELECT의 모든 절
- UPDATE 문의 SET 절 및 WHERE 절

요약

- Correlated subquery는 subquery가 각 후보 행에 대해 항상 다른 결과를 반환해야 하는 경우에 유용합니다.
- EXISTS 연산자는 값이 존재하는지 여부를 테스트하는 부울 연산자입니다.
- Correlated subquery는 SELECT, UPDATE 및 DELETE 문과 함께 사용할 수 있습니다.
- 동일한 query 블록이 두 번 이상 나타날 경우 WITH 절을 사용하여 해당 query 블록을 SELECT 문에 사용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약(계속)

Oracle 서버는 subquery가 상위 명령문에서 참조된 테이블의 열을 참조할 때 correlated subquery를 수행합니다. Correlated subquery는 상위 명령문에 의해 처리된 각 행에 대해 한 번 평가됩니다. 상위 명령문은 SELECT, UPDATE 또는 DELETE 문일 수 있습니다. WITH 절을 사용하면 query 블록을 재평가하는 데 비용이 많이 들 때와 복합 query 내에서 두 번 이상 이 절이 나타날 때 동일한 query를 재사용할 수 있습니다.

연습 6: 개요

이 연습에서는 다음 내용을 다룹니다.

- 여러 열 Subquery 작성
- Correlated Subquery 작성
- EXISTS 연산자 사용
- 스칼라 Subquery 사용
- WITH 절 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 6: 개요

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



정규식 지원

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 정규식 사용 시 이점 나열
- 정규식을 사용하여 문자열 검색, 일치 및 대체

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 정규식 지원 기능의 사용법에 대해 알아봅니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
 - REGEXP_LIKE
 - REGEXP_REPLACE
 - REGEXP_INSTR
 - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용
- 정규식 및 Check 제약 조건

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정규식이란?

- 정규식에서 표준 구문 규칙을 사용하여 문자열 데이터의 간단한 패턴 및 복잡한 패턴을 검색하고 조작할 수 있습니다.
- SQL 함수 및 조건 집합을 사용하여 SQL 및 PL/SQL에서 문자열을 검색하고 조작할 수 있습니다.
- 다음을 사용하여 정규식을 지정합니다.
 - 메타 문자: 검색 알고리즘을 지정하는 연산자
 - 리터럴: 검색 중인 문자

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정규식이란?

오라클 데이터베이스는 정규식을 지원합니다. 정규식의 구현은 ASCII 데이터와 일치하는 의미 및 구문을 위해 IEEE(Institute of Electrical and Electronics Engineers)가 제정한 POSIX(Portable Operating System for UNIX) 표준을 따릅니다. 오라클의 다중 언어 기능은 POSIX 표준을 넘어 연산자의 일치 기능을 확장합니다. 정규식은 검색 및 조작을 위해 간단한 패턴 및 복잡한 패턴을 설명하는 방식입니다.

문자열 조작 및 검색은 웹 기반 응용 프로그램 논리의 대부분을 차지합니다. 사용 범위는 지정된 텍스트에서 "San Francisco"란 단어를 찾는 간단한 검색부터, 텍스트에서 모든 URL을 추출하는 복잡한 검색과 두번째 문자가 모음인 모든 단어를 찾는 매우 복잡한 검색에 이르기까지 다양합니다.

고유의 SQL과 정규식을 함께 사용하면 오라클 데이터베이스에 저장된 임의의 데이터에서 매우 강력한 검색 및 조작 연산을 수행할 수 있습니다. 이 기능을 사용하면 복잡한 프로그래밍을 간단하게 해결할 수도 있습니다.

정규식 사용 시 이점

정규식을 사용하여 데이터베이스에서 복잡한 일치 논리를 구현하면 다음과 같이 이점이 있습니다.

- **오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.**
- **서버측 정규식을 사용하여 제약 조건을 적용함으로써 클라이언트에서 데이터 검증 논리 코드를 작성할 필요가 없습니다.**
- **내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 11g의 이전 버전보다 더욱 쉽고 강력하게 문자열을 조작할 수 있습니다.**



Copyright © 2009, Oracle. All rights reserved.

정규식 사용 시 이점

정규식은 PERL 및 Java와 같은 프로그래밍 언어의 강력한 텍스트 처리 구성 요소입니다. 예를 들어, PERL 스크립트는 디렉토리의 각 HTML 파일을 처리하고 해당 내용을 단일 문자열로 스칼라 변수로 읽어들인 다음 정규식을 사용하여 문자열에서 URL을 검색합니다. 많은 개발자가 PERL을 사용하는 한 가지 이유는 강력한 패턴 일치 기능이 있기 때문입니다. 개발자는 Oracle의 정규식 지원을 통해 데이터베이스에서 복잡한 일치 논리를 구현할 수 있습니다. 이 기법은 다음과 같은 이유로 유용합니다.

- 오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다. SQL 정규식 함수는 처리 논리를 데이터에 더욱 근접하게 이동함으로써 더욱 효율적인 솔루션을 제공합니다.
- Oracle Database 10g 이전에는 개발자가 일반적으로 클라이언트에서 데이터 검증 논리 코드를 작성했으므로 여러 클라이언트에 대해 중복되는 동일한 검증 논리가 필요했습니다. 서버측 정규식을 사용하여 제약 조건을 적용하면 이러한 문제가 해결됩니다.
- 내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 10g의 이전 버전보다 더욱 강력하면서도 간단하게 문자열을 조작할 수 있습니다.

SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

함수 또는 조건 이름	설명
REGEXP_LIKE	LIKE 연산자와 유사하지만 간단한 패턴 일치(조건) 대신 정규식 일치를 수행합니다.
REGEXP_REPLACE	정규식 패턴을 검색하여 대체 문자열로 바꿉니다.
REGEXP_INSTR	정규식 패턴에 대해 문자열을 검색하고 일치가 발견된 위치를 반환합니다.
REGEXP_SUBSTR	지정된 문자열 내에서 정규식 패턴을 검색하고 일치하는 부분 문자열을 추출합니다.
REGEXP_COUNT	입력 문자열에서 패턴 일치가 발견되는 횟수를 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

오라클 데이터베이스는 정규식을 사용하여 문자열을 검색 및 조작할 수 있는 일련의 SQL 함수를 제공합니다. 텍스트 리터럴, 바인드 변수 또는 CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, VARCHAR2 (LONG 제외) 와 같은 문자 데이터를 포함하는 열에서 이러한 함수를 사용합니다. 정규식은 작은 따옴표로 끓어야 합니다. 이렇게 하면 SQL 함수가 전체 표현식을 해석하고 코드의 가독성을 높일 수 있습니다.

- REGEXP_LIKE: 이 조건은 패턴에 대한 문자열을 검색합니다. Query의 WHERE 절에서 이 조건을 사용하여 지정한 정규식과 일치하는 행을 반환합니다.
- REGEXP_REPLACE: 이 함수는 문자열에서 패턴을 검색하고 해당 패턴의 각 발생 값을 지정한 패턴으로 대체합니다.
- REGEXP_INSTR: 이 함수는 문자열에서 지정된 정규식 패턴의 발생 값을 검색합니다. 찾고자 하는 발생 값과 검색 시작 위치를 지정합니다. 이 함수는 일치가 발견된 문자열의 위치를 나타내는 정수를 반환합니다.
- REGEXP_SUBSTR: 이 함수는 지정한 정규식 패턴과 일치하는 실제 부분 문자열을 반환합니다.
- REGEXP_COUNT: 이 함수는 입력 문자열에서 패턴 일치가 발견되는 횟수를 반환합니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
 - REGEXP_LIKE
 - REGEXP_REPLACE
 - REGEXP_INSTR
 - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

메타 문자란?

- 메타 문자는 대체 문자, 반복 문자, 일치하지 않는 문자 또는 일련의 문자와 같이 특별한 의미를 지닌 특수 문자입니다.
- 패턴 일치에서 여러 개의 미리 정의된 메타 문자 기호를 사용할 수 있습니다.
- 예를 들어, `^(f | ht)tps? :$` 정규식은 문자열 시작 부분에서 다음을 검색합니다.
 - 리터럴 f 또는 ht
 - t 리터럴
 - p 리터럴(선택적으로 다음에 s 리터럴이 나옴)
 - 문자열 끝부분의 ":" 리터럴

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

메타 문자란?

슬라이드의 정규식은 http:, https:, ftp: 및 ftps: 문자열을 일치시킵니다.

참고: 정규식의 메타 문자 전체 리스트는

*Oracle Database Advanced Application Developer's Guide 11g Release 2*를 참조하십시오.

정규식에서 메타 문자 사용

구문	설명
.	지원되는 character set에서 NULL을 제외한 임의의 문자와 일치
+	한 번 이상 발생 수 일치
?	0 또는 1번 발생 수 일치
*	선행 하위식의 0번 이상 발생 수 일치
{m}	선행 표현식의 정확히 m번 발생 수 일치
{m, }	선행 하위식과 최소 m번 이상 발생 수 일치
{m,n}	선행 하위식의 최소 m번 이상, 최대 n번 이하 발생 수 일치
[...]	괄호 안의 리스트에 있는 임의의 단일 문자와 일치
	여러 대안 중 하나와 일치
(...)	괄호로 묶인 표현식을 한 단위로 취급합니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정규식 함수에서 메타 문자 사용

임의의 문자, ".": a.b는 문자열 **abb**, **acb**, **adb**와 일치하지만 **acc**와는 일치하지 않습니다.

하나 이상, "+": a+는 문자열 **a**, **aa**, **aaa**와 일치하지만 **bbb**와는 일치하지 않습니다.

0 또는 1, "?": ab?c는 문자열 **abc**, **ac**와 일치하지만 **abbc**와는 일치하지 않습니다.

0 이상, "*": ab*c는 문자열 **ac**, **abc**, **abbc**와 일치하지만 **abb**와는 일치하지 않습니다.

정확한 숫자, "{m}": a{3}는 문자열 **aaa**와 일치하지만 **aa**와는 일치하지 않습니다.

최소 숫자, "{m,}": a{3,}은 문자열 **aaa**, **aaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

사이 숫자, "{m,n}": a{3,5}는 문자열 **aaa**, **aaaa**, **aaaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

문자 리스트 일치, "[...)": [abc]는 문자열 **all**, **bill**, **cold**의 첫번째 문자와 일치하지만 **doll**의 어떠한 문자와도 일치하지 않습니다.

또는 "|": a|b는 문자 **a** 또는 문자 **b**와 일치합니다.

하위식, "(...)" : (abc)?def는 선택적 문자열 **abc** 다음에 **def**가 나옵니다. 이 표현식은 **abcdefghi** 및 **def**와 일치하지만 **ghi**와는 일치하지 않습니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

정규식에서 메타 문자 사용

구문	설명
^	문자열 시작 부분과 일치
\$	문자열 끝부분과 일치
\	표현식에서 후속 메타 문자를 리터럴로 처리합니다.
\n	괄호 안에 그룹화된 n번째 (1-9) 선행 하위식과 일치합니다. 괄호는 표현식이 기억되도록 만들고 backreference에서 표현식을 참조합니다.
\d	숫자 문자
[:class:]	지정된 POSIX 문자 클래스에 속한 임의의 문자와 일치
[^:class:]	괄호 안의 리스트에 없는 임의의 단일 문자와 일치

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정규식 함수에서 메타 문자 사용(계속)

행 앵커의 시작/끝, "^" 및 "\$": ^def는 문자열 defghi의 def와 일치하지만 abcdef의 def와는 일치하지 않습니다. def\$는 문자열 abcdef의 def와 일치하지만 문자열 defghi의 def와는 일치하지 않습니다.

이스케이프 문자 "\+": \+는 +를 검색합니다. 문자열 abc+def의 더하기(+) 문자와 일치하지만 Abcdef와는 일치하지 않습니다.

Backreference, "\n": (abc|def)xy\1은 문자열 abcxyabc 및 defxydef와 일치하지만 abcxydef 또는 abcxy와는 일치하지 않습니다. backreference를 사용하면 실제 문자열을 미리 알고 있지 않아도 반복되는 문자열을 검색할 수 있습니다. 예를 들어, ^(.*)\1\$ 표현식은 동일한 문자열의 2개의 인접한 instance로 구성된 행과 일치합니다.

숫자 문자, "\d": ^[\d{3}\] \d{3}-\d{4}\$ 표현식은 [650] 555-1212와 일치하지만 650-555-1212와는 일치하지 않습니다.

문자 클래스, "[:class:]": [[:upper:]]+는 하나 이상의 연속되는 대문자를 검색합니다. 이 경우 문자열 abcDEFghi의 DEF와 일치하지만 문자열 abcdefghi와는 일치하지 않습니다.

비일치 문자 리스트 (또는 클래스), "[^...]": [^abc]는 문자열 abcdef의 문자 d와 일치하지만 문자 a, b 또는 c와는 일치하지 않습니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
 - REGEXP_LIKE
 - REGEXP_REPLACE
 - REGEXP_INSTR
 - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정규식 함수 및 조건: 구문

```
REGEXP_LIKE (source_char, pattern [,match_option])
```

```
REGEXP_INSTR (source_char, pattern [, position  
[, occurrence [, return_option  
[, match_option [, subexpr]]]]])
```

```
REGEXP_SUBSTR (source_char, pattern [, position  
[, occurrence [, match_option  
[, subexpr]]]])
```

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence  
[, match_option]]]])
```

```
REGEXP_COUNT (source_char, pattern [, position  
[, occurrence [, match_option]]])
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정규식 함수 및 조건: 구문

정규식 함수 및 조건의 구문은 다음과 같습니다.

- **source_char:** 검색 값 역할을 하는 문자 표현식
- **pattern:** 정규식, 텍스트 리터럴
- **occurrence:** Oracle 서버가 검색해야 하는 source_char에서 패턴의 발생 값을 나타내는 양의 정수. 기본값은 1입니다.
- **position:** Oracle 서버가 검색을 시작해야 하는 source_char의 문자를 나타내는 양의 정수. 기본값은 1입니다.
- **return_option:**
 - 0: 발생 값의 첫번째 문자의 위치를 반환합니다(기본값).
 - 1: 발생 값 다음의 문자 위치를 반환합니다.
- **Replacestr:** 패턴 대체 문자열
- **match_parameter:**
 - "c": 대소문자를 구분하는 일치 사용(기본값)
 - "i": 대소문자를 구분하지 않는 일치 사용
 - "n": 임의의 문자 일치 연산자 허용
 - "m": 소스 문자열을 다중 행으로 처리
- **subexpr:** 괄호로 묶인 패턴 부분. 하위식은 이 단원 뒷부분에서 다룹니다.

REGEXP_LIKE 조건을 사용하여 기본 검색 수행

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REGEXP_LIKE 조건을 사용하여 기본 검색 수행

REGEXP_LIKE는 LIKE에서 수행하는 단순한 패턴 일치 대신 정규식 일치를 수행한다는 점을 제외하고 LIKE 조건과 일치합니다. 이 조건은 입력 character set에 의해 정의된 문자를 사용하여 문자열을 평가합니다.

REGEXP_LIKE의 예제

EMPLOYEES 테이블에 대한 이 query에서는 first name에 Steven 또는 Stephen이 포함된 모든 사원이 표시됩니다. 다음은 query에 사용된 표현식 '^Ste(v|ph)en\$'에 대한 설명입니다.

- ^는 표현식의 시작을 나타냅니다.
- \$는 표현식의 끝을 나타냅니다.
- |는 둘 중 하나(또는)를 나타냅니다.

REGEXP_REPLACE 함수를 사용하여 패턴 대체

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence [, match_option]]]])
```

```
SELECT REGEXP_REPLACE(phone_number, '\.', '-') AS phone  
FROM employees;
```

원본

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

일부 결과

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REGEXP_REPLACE 함수를 사용하여 패턴 대체

REGEXP_REPLACE 함수를 사용하여 마침표(.) 구분자를 대시(-) 구분자로 대체하도록 전화 번호 형식을 다시 지정할 수 있습니다. 다음은 정규식 예제에서 사용되는 각 요소에 대한 설명입니다.

- phone_number는 소스 열입니다.
- '\.'는 검색 패턴입니다.
 - 작은 따옴표('')를 사용하여 리터럴 문자 마침표(.)를 검색합니다.
 - 백슬래시(\)를 사용하여 일반적으로 메타 문자로 처리되는 문자를 검색합니다.
- '-'는 대체 문자열입니다.

REGEXP_INSTR 함수를 사용하여 패턴 찾기

```
REGEXP_INSTR (source_char, pattern [, position [, occurrence [, return_option [, match_option]]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[:alpha:]') AS
First_Alpha_Position
FROM locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REGEXP_INSTR 함수를 사용하여 패턴 찾기

이 예제에서는 REGEXP_INSTR 함수로 주소를 검색하여 대소문자 여부와 상관없이 첫번째 알파벳 문자의 위치를 찾습니다. [:<class>:]는 문자 클래스를 나타내며 해당 클래스 내의 임의의 문자와 일치합니다. [:alpha:]는 임의의 알파벳 문자와 일치합니다. 부분적인 결과가 표시됩니다.

다음은 query에 사용된 표현식 '[:alpha:]'에 대한 설명입니다.

- [는 표현식을 시작합니다.
- [:alpha:]는 알파벳 문자 클래스를 나타냅니다.
-]는 표현식을 종료합니다.

참고: POSIX 문자 클래스 연산자를 사용하면 특정 POSIX 문자 클래스의 멤버인 문자 리스트 내에서 표현식을 검색할 수 있습니다. 이 연산자를 사용하여 대문자와 같은 특정 형식을 검색하거나 자릿수 또는 구두점 문자 등의 특수 문자를 검색할 수 있습니다. 전체 POSIX 문자 클래스를 지원합니다. [:class:] 구문을 사용합니다. 여기서 class는 검색할 POSIX 문자 클래스의 이름을 나타냅니다. 정규식 [:upper:]+는 하나 이상의 연속하는 대문자를 검색합니다.

REGEXP_SUBSTR 함수를 사용하여 부분 문자열 추출

```
REGEXP_SUBSTR (source_char, pattern [, position  
[, occurrence [, match_option]]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ') AS Road  
FROM locations;
```

ROAD
1 Via
2 Calle
3 (null)
4 (null)
5 Jabberwocky

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

REGEXP_SUBSTR 함수를 사용하여 부분 문자열 추출

이 예제에서는 LOCATIONS 테이블에서 road name을 추출합니다. 이를 위해 REGEXP_SUBSTR 함수를 사용하여 STREET_ADDRESS 열에서 첫번째 공백 뒤에 있는 내용을 반환합니다. 다음은 query에 사용된 표현식 '[^]+'에 대한 설명입니다.

- [는 표현식을 시작합니다.
- ^는 NOT을 나타냅니다.
- 공백을 나타냅니다.
-]는 표현식을 종료합니다.
- +는 1 이상을 나타냅니다.
- 공백을 나타냅니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
 - REGEXP_LIKE
 - REGEXP_REPLACE
 - REGEXP_INSTR
 - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE®

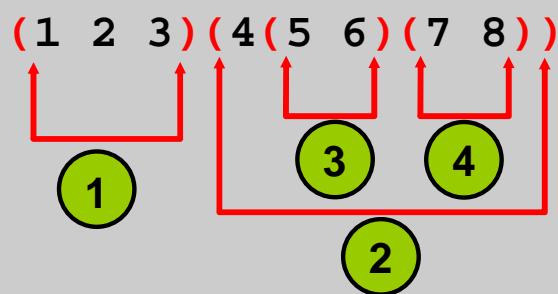
Copyright © 2009, Oracle. All rights reserved.

하위식

다음 표현식을 살펴보십시오.

(1 2 3)(4(5 6)(7 8))

하위식은 다음과 같습니다.



ORACLE®

Copyright © 2009, Oracle. All rights reserved.

하위식

Oracle Database 11g는 하위식에 액세스할 수 있도록 정규식 지원 파라미터를 제공합니다. 슬라이드 예제에 숫자 문자열이 표시되어 있습니다. 괄호는 숫자 문자열 내의 하위식을 식별합니다. 왼쪽에서 오른쪽, 외부 괄호에서 내부 괄호 순서로 읽을 경우 숫자 문자열의 하위식은 다음과 같습니다.

1. 123
2. 45678
3. 56
4. 78

REGEXP_INSTR 및 REGEXP_SUBSTR 함수를 사용하여 이러한 하위식을 검색할 수 있습니다.

정규식을 지원하는 하위식 사용

```

SELECT
  REGEXP_INSTR
  (
    '0123456789',
    '(123)(4(56)(78))',
    1,
    1,
    0,
    'i',
    1
  ) "Position"
FROM dual;
  
```

	Position
1	
2	

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정규식을 지원하는 하위식 사용

REGEXP_INSTR 및 REGEXP_SUBSTR에는 계산되는 정규식의 특정 부분 문자열을 대상으로 할 수 있는 선택적 SUBEXPR 파라미터가 있습니다.

슬라이드의 예제에서 하위식 리스트의 첫번째 하위식 패턴을 검색하려고 합니다. 제공된 예제는 REGEXP_INSTR 함수에 대한 여러 파라미터를 식별합니다.

1. 검색 중인 문자열이 식별됩니다.
2. 하위식이 식별됩니다. 첫번째 하위식은 123이고 두번째는 45678, 세번째는 56, 네번째는 78입니다.
3. 세번째 파라미터는 검색을 시작할 위치를 식별합니다.
4. 네번째 파라미터는 찾으려는 패턴의 발생 값을 식별합니다. 1은 첫번째 발생 값을 찾는다는 의미입니다.
5. 다섯번째 파라미터는 반환 옵션입니다. 발생 값의 첫번째 문자의 위치입니다. 1을 지정하면 발생 값 다음의 문자 위치가 반환됩니다.
6. 여섯번째 파라미터는 검색에서 대소문자를 구분해야 하는지 여부를 식별합니다.
7. 마지막 파라미터는 Oracle Database 11g에서 추가된 파라미터입니다. 이 파라미터는 찾으려는 하위식을 지정합니다. 제공된 예제에서는 첫번째 하위식을 검색하며, 결과는 123입니다.

*n*번째 하위식에 액세스하는 이유

- 더욱 실제적인 사용: DNA 시퀀싱
- 주의 DNA에서 면역에 필요한 단백질을 식별하는 특정 하위 패턴을 찾으려고 합니다.

```
SELECT
  REGEXP_INSTR('ccaccttccctccactcctcacgttctcacgtaaagcgccctc
cctcatccccatcccccttaccctgcaggtagagtaggctagaaaccagagagctccaagc
tccatctgtggagaggtgccatccctgggtgcagagagaggagaatttgcggcaagctgcc
tgcagagcttcaccacccttagtctcacaaagccttgagttcatagcattcttgagtttca
ccctgcccagcaggacactgcagcacccaaaggcctccaggagttagggttgcctcaagag
gctttgggtctgatggccacatcctggaattttcaagttgatggtcacagccctgaggc
atgttagggcgtgggatgcgcctgctctcctctcctgaaccctgaaccctctggc
tacccagagcacttagagccag',
  '(gtc(tcac)(aaag))',
  1, 1, 0, 'i',
  1) "Position"
FROM dual;
```

	Position
1	195

ORACLE

Copyright © 2009, Oracle. All rights reserved.

*n*번째 하위식에 액세스하는 이유

생명 과학 분야에서 추가 처리를 위해 DNA 시퀀스로부터 하위식 일치의 오프셋을 추출해야 하는 경우가 있습니다. 예를 들어, gtc가 앞에 나오고 이어서 tcac, aaag가 나오는 DNA 시퀀스의 시작 오프셋과 같은 특정 단백질 시퀀스를 찾으려고 합니다. 이러한 결과를 얻으려면 일치가 발견된 위치를 반환하는 REGEXP_INSTR 함수를 사용하면 됩니다.

슬라이드 예제에서는 첫번째 하위식 (gtc)의 위치가 반환됩니다. gtc는 DNA 문자열의 195 위치에서 시작되는 것으로 나타납니다.

슬라이드 예제를 수정하여 두번째 하위식 (tcac)를 검색하면 query 결과는 다음과 같이 출력됩니다. tcac는 DNA 문자열의 198 위치에서 시작되는 것으로 나타납니다.

	Position
1	198

슬라이드 예제를 수정하여 세번째 하위식 (aaag)를 검색하면 query 결과는 다음과 같이 출력됩니다. aaag는 DNA 문자열의 202 위치에서 시작하는 것으로 나타납니다.

	Position
1	202

REGEXP_SUBSTR: 예제

```

SELECT
  REGEXP_SUBSTR
  ①('acgctgcactgca', -- source char or search value
  ② 'acg(.*)gca',    -- regular expression pattern
  ③ 1,                -- position to start searching
  ④ 1,                -- occurrence
  ⑤ 'i',              -- match option (case insensitive)
  ⑥ 1)                -- sub-expression
  "Value"
FROM dual;

```

	Value
1	ctgact

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REGEXP_SUBSTR: 예제

슬라이드의 예제는 다음과 같이 설명됩니다.

1. acgctgcactgca는 검색할 소스입니다.
2. acg(.*)gca는 검색할 패턴입니다. acg 다음에 gca가 오는 문자열을 찾습니다.
acg와 gca 사이에 다른 문자가 포함될 수 있습니다.
3. 소스의 첫번째 문자에서 검색을 시작합니다.
4. 패턴의 첫번째 발생 값을 검색합니다.
5. 소스에서 대소문자를 구분하지 않는 일치를 사용합니다.
6. 대상으로 지정할 n번째 하위식을 식별하는 음이 아닌 정수 값을 사용합니다. 이는 하위식
파라미터입니다. 이 예제에서 1은 첫번째 하위식을 나타냅니다. 0-9 사이의 값을 사용할
수 있습니다. 0은 하위식이 대상으로 지정되지 않음을 의미합니다. 이 파라미터에 대한
기본값은 0입니다.

단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
 - REGEXP_LIKE
 - REGEXP_REPLACE
 - REGEXP_INSTR
 - REGEXP_SUBSTR
- 하위식 액세스
- REGEXP_COUNT 함수 사용

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

REGEXP_COUNT 함수 사용

```
REGEXP_COUNT (source_char, pattern [, position
               [, occurrence [, match_option]]])
```

```
SELECT REGEXP_COUNT(
  'ccaccccccactcactcacgttctcacctgtaaagcgccccatgcccccttaccctgcag
  ggttagagttaggctagaaaaccagagagactccaagctccatctgtggagaggtgccatc
  ttggctgcagagacttcaccacccttagtctcacaaggccttagttcatgcatttcttgagtt
  ttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgc
  ctcaagaggctctggcgttgatggggatgcgctctgtatggccacatc
  ctggaaattgtttcaagttgtatggtcacagccctgaggcatgttagggcgtggggatgcg
  ctctgtctctctctcgaaccctctggctaccctcagagcacttagagccag',
  'gtc') AS Count
FROM dual;
```

	COUNT
1	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

REGEXP_COUNT 함수 사용

REGEXP_COUNT 함수는 입력 character set에서 정의된 문자를 사용하여 문자열을 평가합니다. 이 함수는 패턴의 발생 수를 나타내는 정수를 반환합니다. 일치가 발견되지 않으면 이 함수는 0을 반환합니다.

슬라이드 예제에서는 REGEXP_COUNT 함수를 사용하여 DNA 부분 문자열의 발생 수를 판별합니다.

다음 예제는 문자열 123123123123에서 패턴 123이 발생하는 횟수가 세 번임을 보여줍니다. 검색은 문자열의 두번째 위치에서 시작됩니다.

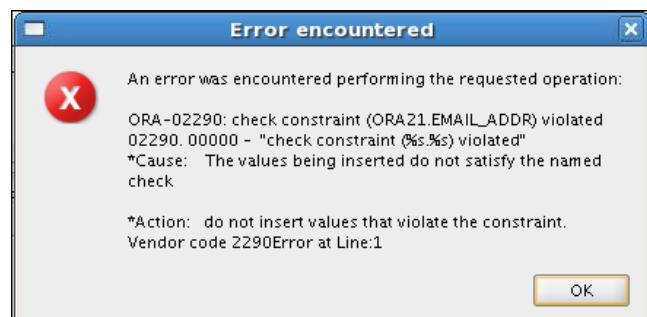
```
SELECT REGEXP_COUNT
  ('123123123123', -- source char or search value
   '123',           -- regular expression pattern
   2,                -- position where the search should start
   'i')              -- match option (case insensitive)
  As Count
FROM dual;
```

	COUNT
1	3

정규식 및 Check 제약 조건: 예제

```
ALTER TABLE emp8
ADD CONSTRAINT email_addr
CHECK(REGEXP_LIKE(email,'@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES
(500,'Christian','Patel','ChrisP2creme.com',
1234567890,'12-Jan-2004','HR_REP',2000,null,102,40);
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

정규식 및 Check 제약 조건: 예제

CHECK 제약 조건에서도 정규식을 사용할 수 있습니다. 이 예제에서는 CHECK 제약 조건이 EMPLOYEES 테이블의 EMAIL 열에 추가되었습니다. 이렇게 하면 "@" 기호를 포함한 문자열만을 받아들입니다. 제약 조건을 테스트합니다. 전자 메일 주소에 필요한 필수 기호가 포함되어 있지 않으므로 CHECK 제약 조건을 위반했습니다. NOVALIDATE 절은 기존 데이터를 검사했는지 여부를 확인합니다.

슬라이드 예제에서는 다음 코드를 사용하여 emp8 테이블을 생성합니다.

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

참고: 슬라이드의 예제는 SQL Developer의 "Execute Statement" 옵션을 사용하여 실행합니다. "Run Script" 옵션을 사용하는 경우 출력 형식이 달라집니다.

퀴즈

SQL 및 PL/SQL에서 정규식을 사용할 경우 옳은 설명을 고르십시오.

1. middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.
2. 클라이언트에서 데이터 검증 논리를 작성할 필요가 없습니다.
3. 서버의 제약 조건을 적용할 수 있습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3

요약

이 단원에서는 정규식을 사용하여 문자열을 검색, 일치 및 대체하는 방법을 배웠습니다.

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 정규식 지원 기능의 사용법을 배웠습니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

연습 7: 개요

이 연습에서는 정규식 함수를 사용하여 다음 작업을 수행하는 방법을 다룹니다.

- 데이터 검색, 대체 및 조작
- 새 CONTACTS 테이블을 생성하고 해당 전화 번호가 특정 표준 형식으로 데이터베이스에 입력되도록 p_number 열에 CHECK 제약 조건을 추가
- 다양한 형식을 사용하여 p_number 열에 일부 전화 번호 추가 테스트

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

연습 7: 개요

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.

부록 A

연습 및 해답

목차

단원 I 의 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 1 의 연습 및 해답.....	17
연습 1-1: 유저 액세스 제어	17
연습 문제 해답 1-1: 유저 액세스 제어	20
단원 2 의 연습 및 해답.....	24
연습 2-1: 스키마 객체 관리	24
연습 문제 해답 2-1: 스키마 객체 관리	30
단원 3 의 연습 및 해답.....	38
연습 3-1: 데이터 딕셔너리 뷰로 객체 관리	38
연습 문제 해답 3-1: 데이터 딕셔너리 뷰로 객체 관리	42
단원 4 의 연습 및 해답.....	45
연습 4-1: 대형 객체 집합 조작.....	45
연습 문제 해답 4-1: 대형 객체 집합 조작	50
단원 5 의 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 6 의 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 7 의 연습 및 해답.....	66
연습 7-1: 정규식 지원	66
연습 문제 해답 7-1: 정규식 지원	68

단원 I의 연습 및 해답

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 템 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 폴드에 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.
- File > Save As를 선택합니다.
 - 파일 이름을 intro_test.sql로 지정합니다.
 - /home/oracle/labs/sql2/labs 폴더에 파일을 저장합니다.
- 9) /home/oracle/labs/sql2/labs 폴더에서 confidence.sql을 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.

The screenshot shows the Oracle Technology Network homepage. The URL in the address bar is http://www.oracle.com/technology/products/database/sql_developer/index.html. The main content area is titled "Oracle SQL Developer" and describes it as a free and fully supported graphical tool for database development. It lists various features like browsing database objects, running SQL statements, and editing PL/SQL. Below this, there are several links: "What is Oracle SQL Developer?", "SQL Developer 1.5", "SQL Developer OTN Forum", "White papers and Supporting Documents", "SQL Developer Exchange", and "Online Demonstrations". On the left sidebar, there are sections for "PRODUCTS" (Database, Middleware, Developer Tools, etc.), "TECHNOLOGIES" (BI & Data Warehousing, Embedded, Java, Linux, .NET, PHP, Security, etc.), and "ARCHITECTURE" (Enterprise Architecture, Enterprise 2.0, Grid, Service Oriented Architecture). The top navigation bar includes File, Edit, View, History, Bookmarks, Tools, Help, and a search bar. The right side of the page shows a sidebar with news items and a calendar.

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>

그린 다음 아래 섹션 및 관련 데모를 검토합니다.

- a) What to Do First
- b) Working with Database Objects
- c) Accessing Data

연습 문제 해답 I-2: SQL Developer 사용

- 1) 바탕 화면 아이콘을 사용하여 SQL Developer를 시작합니다.

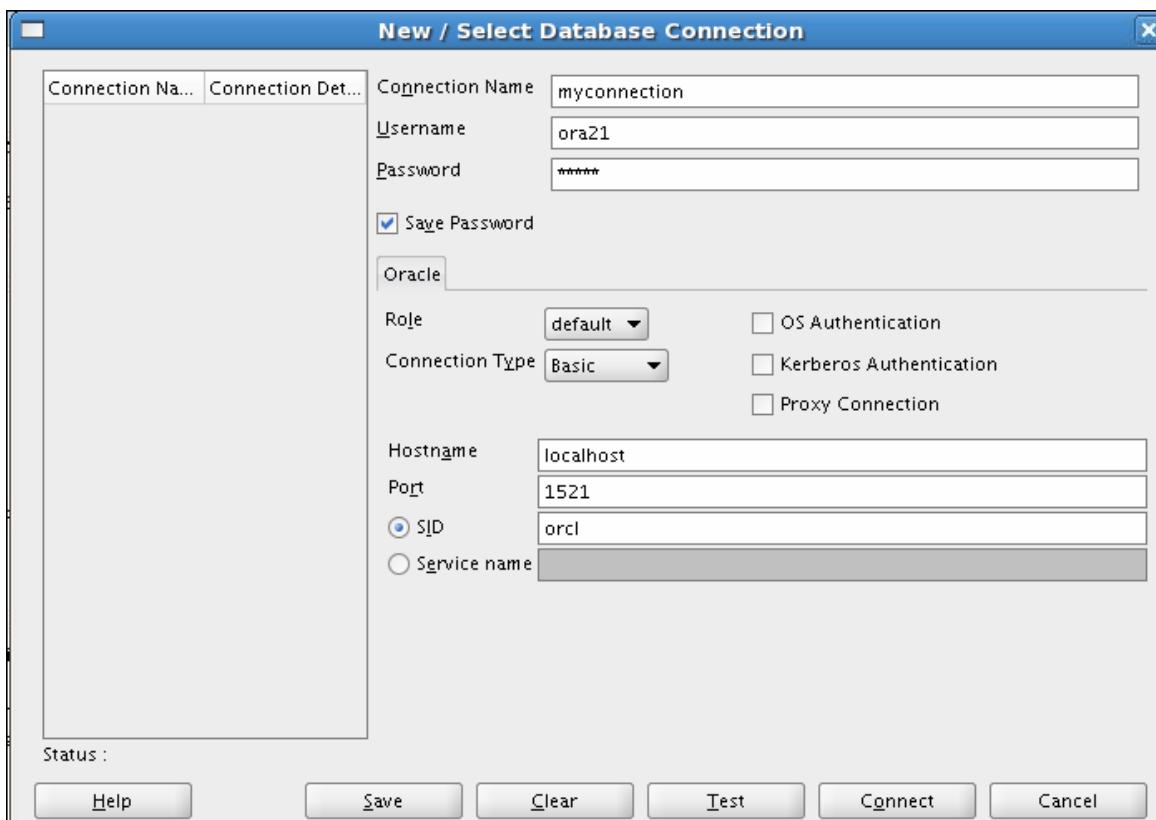


- 2) 다음 정보를 사용하여 데이터베이스 연결을 생성합니다.

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



연습 문제 해답 I-2: SQL Developer 사용(계속)

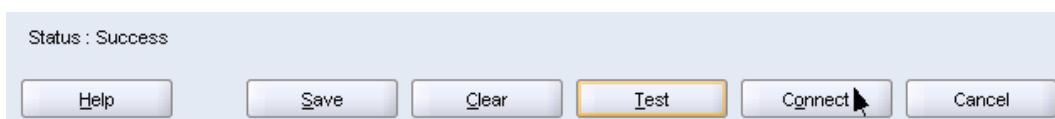


- 3) 새 연결을 테스트합니다. 상태가 Success이면 이 새 연결을 사용하여 데이터베이스에 연결합니다.

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



b) Status가 Success이면 Connect 버튼을 누릅니다.

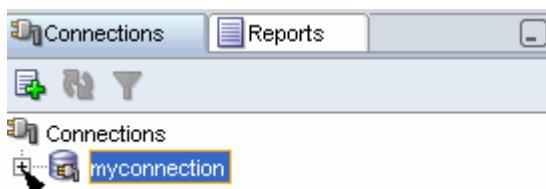


연습 문제 해답 I-2: SQL Developer 사용(계속)

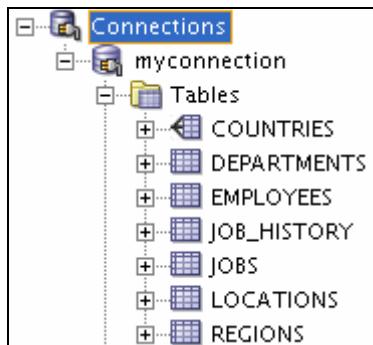
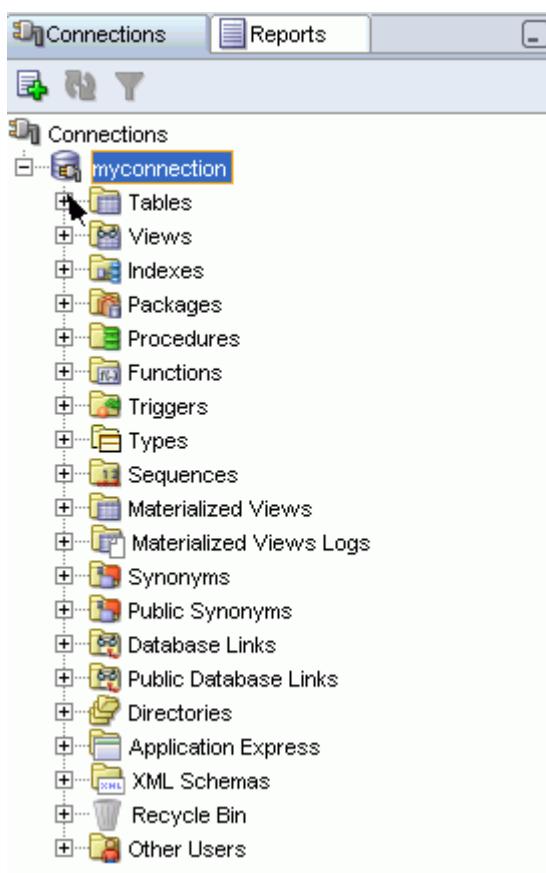
테이블 탐색

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 I-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
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 the 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	Administration	200	1700
2	Marketing	201	1800
3	Purchasing	114	1700
4	Human Resources	203	2400
5	Shipping	121	1500
6	IT	103	1400
7	Public Relations	204	2700

연습 문제 해답 I-2: SQL Developer 사용(계속)

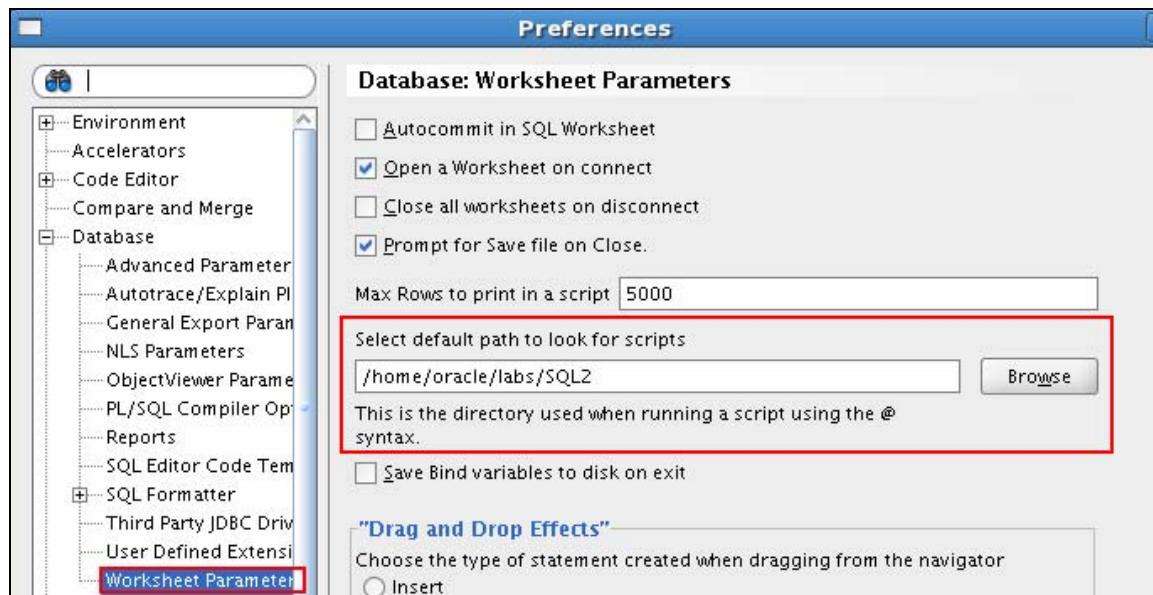
- 5) 몇 가지 기본 SELECT 문을 실행하여 SQL Worksheet 영역에 있는 EMPLOYEES 테이블의 데이터를 query합니다. Execute Statement(또는 F9)와 Run Script 아이콘(또는 F5)을 둘 다 사용하여 SELECT 문을 실행합니다. 해당 탭 페이지에서 SELECT 문을 실행하는 두 방법의 결과를 검토합니다.
- a) 급여가 \$3,000 이하인 사원의 성 및 급여를 선택하는 query를 작성합니다.

```
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.



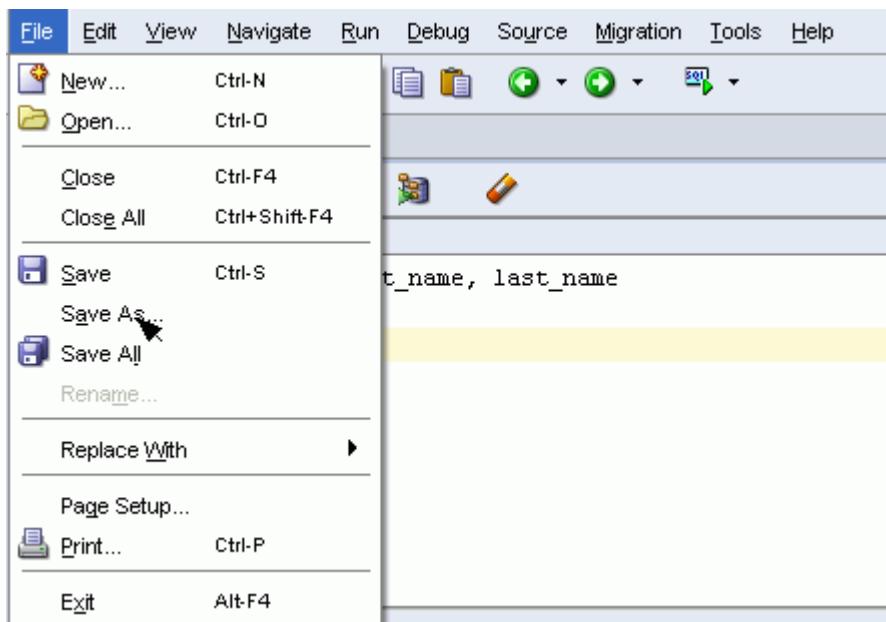
연습 문제 해답 I-2: SQL Developer 사용(계속)

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.

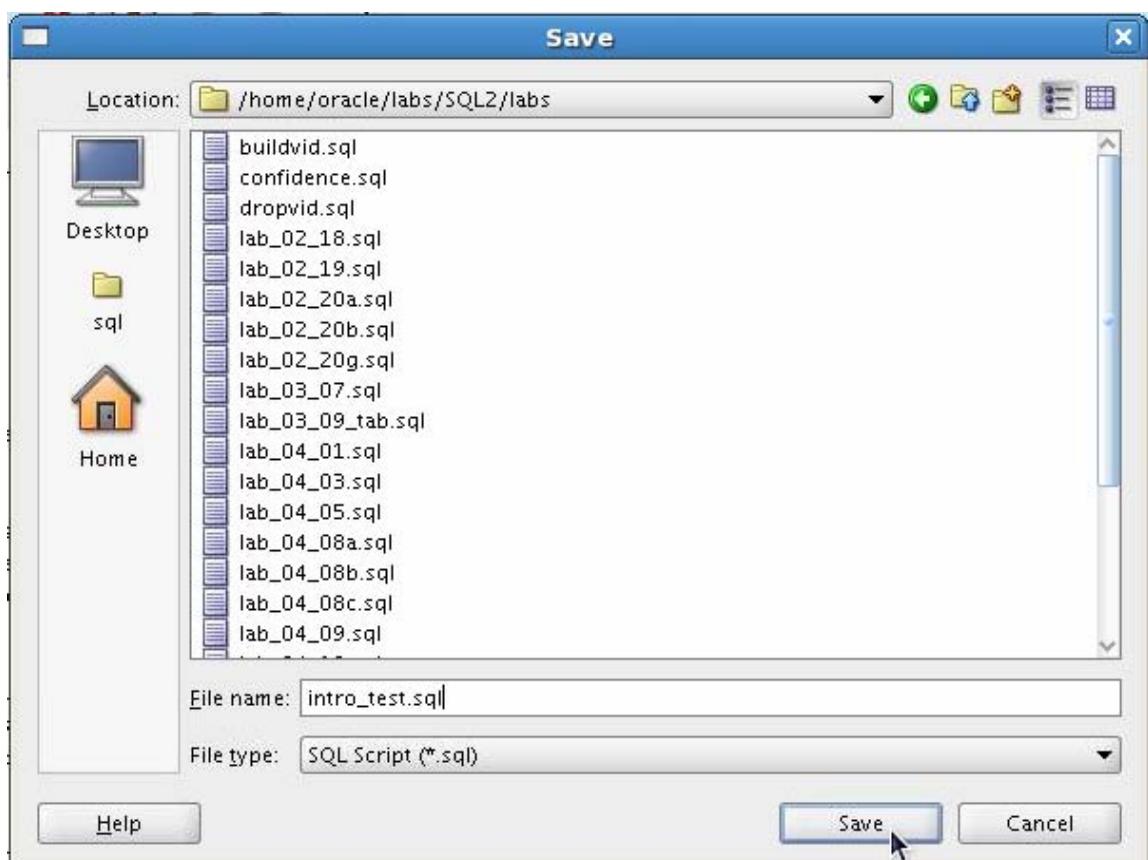


b) 파일 이름을 **intro_test.sql**로 지정합니다.

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.

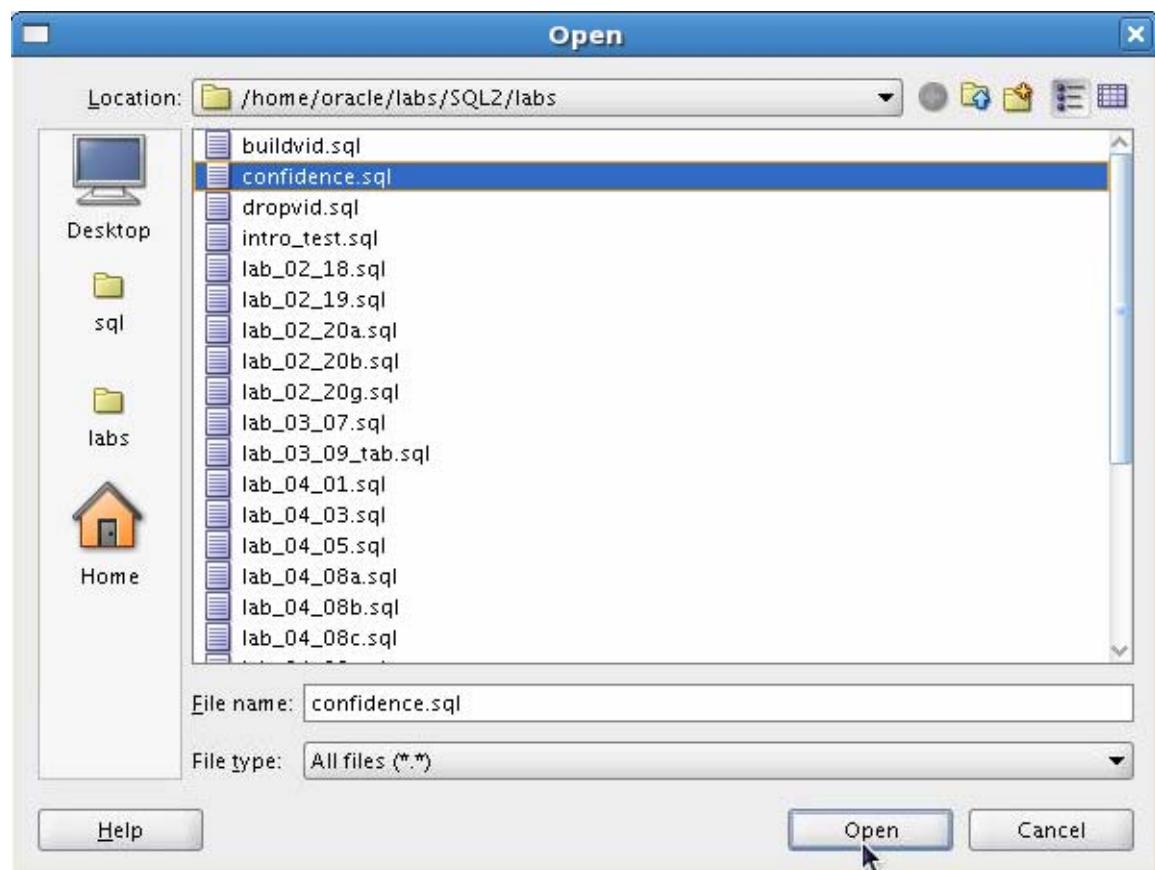


그린 다음 Save를 누릅니다.

- 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.



그린 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
```

단원 1의 연습 및 해답

연습 1-1: 유저 액세스 제어

- 1) Oracle 서버에 로그온하려면 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 로그온합니다.

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.

- REGIONS 테이블의 레코드를 볼 수 있는 권한을 다른 유저에게 부여합니다. 이 권한을 다른 유저에게 추가로 부여할 수 있는 옵션을 이 유저에 대해 포함시킵니다.
- 유저가 REGIONS 테이블을 query하도록 합니다.
- 유저가 세번째 유저(예: ora23)에게 query 권한을 부여하도록 합니다.
- 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	Administration	200	1700
2	Marketing	201	1800
3	Purchasing	114	1700
4	Human Resources	203	2400
5	Shipping	121	1500
6	IT	103	1400
7	Public Relations	204	2700
8	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
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
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.

- Oracle 서버에 로그온하려면 유저에게 어떤 권한이 부여되어야 합니까? 이것은 시스템 권한입니까, 객체 권한입니까?

CREATE SESSION 시스템 권한

- 테이블 생성용으로 유저에게 부여되는 권한은 무엇입니까?

CREATE TABLE 권한

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

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

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

작업을 보다 쉽게 수행하기 위해 무엇을 사용해야 합니까?

시스템 권한을 포함하는 룰을 생성하고 이 룰을 유저에게 부여해야 합니다.

- 암호를 변경하기 위해 어떤 명령을 사용합니까?

ALTER USER 문

- 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

- 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 username^o이고 <team2_oraxx>는 Team 2의 username^o입니다.

- 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_oraxxx>.DEPARTMENTS;
```

Team 2는 team1로 명명된 동의어를 생성합니다.

```
CREATE SYNONYM team1
    FOR <team1_oraxxx>. 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_oraxxx>;
```

Team 2는 권한을 취소합니다.

```
REVOKE select
    ON departments
    FROM <team1_oraxxx>;
```

- 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;
```

단원 2의 연습 및 해답

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.

ORIGINAL_NAME	OPERATION	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_ext를 생성합니다. ORACLE_LOADER 액세스 드라이버를 사용합니다.

참고: 이 연습에서는 emp_dir 디렉토리와 library_items.dat 파일이 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,
```

- 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.
- library_items_ext 테이블을 query합니다.

	CATEGOR...	BOOK_ID	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_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.

- 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.
- 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.

단원 3의 연습 및 해답

연습 3-1: 데이터 딕셔너리 뷰로 객체 관리

이 연습에서는 딕셔너리 뷰를 query하여 스키마의 객체에 대한 정보를 찾습니다.

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

TABLE_NAME
REGIONS
LOCATIONS
DEPARTMENTS
Jobs
EMPLOYEES
JOB_HISTORY
EMP_NEW_SAL
EMPLOYEES2
DEPT_NAMED_INDEX

...

- 2) ALL_TABLES 데이터 딕셔너리 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.

TABLE_NAME	OWNER
DUAL	SYS
SYSTEM_PRIVILEGE_MAP	SYS
TABLE_PRIVILEGE_MAP	SYS

...

98 PLAN_TABLE\$	SYS
99 WRI\$_ADV_ASA_RECO_DATA	SYS
100 PSTUBTBL	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.

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

연습 3-1: 데이터 딕셔너리 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 딕셔너리 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 *n*은 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 딕셔너리 뷰로 객체 관리(계속)

- 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.

SEQUENCE_NAME	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
1 DEPARTMENTS_SEQ	9990	10	280
2 EMPLOYEES_SEQ	99999999999999999999999999999999	1	207
3 LOCATIONS_SEQ	9900	100	3300

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.

- 기존 테이블 DEPT2 및 EMP2가 있는 경우 삭제합니다.
- DEPT2 및 EMP2 테이블을 생성합니다.

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

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

TABLE_NAME
1 DEPT2
2 EMP2

- 10) USER_CONSTRAINTS 뷰를 query하여 제약 조건이 추가되었는지 확인합니다.
제약 조건의 유형과 이름을 기록해둡니다.

CONSTRAINT_NAME	CONSTRAINT_TYPE
1 MY_DEPT_ID_PK	P
2 MY_EMP_ID_PK	P
3 MY_EMP_DEPT_ID_FK	R

- 11) EMP2 및 DEPT2 테이블에 대한 USER_OBJECTS 데이터 딕셔너리 뷰에서 가져온 객체 이름 및 유형을 표시합니다.

연습 3-1: 데이터 딕셔너리 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 딕셔너리 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
SALES_PK_IDX	SALES_DEPT	NONUNIQUE

연습 문제 해답 3-1: 데이터 딕셔너리 뷰로 객체 관리

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

```
SELECT table_name
  FROM user_tables;
```

- 2) 딕셔너리 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 딕셔너리 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 딕셔너리 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 딕셔너리 뷰로 객체 관리(계속)

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

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

10) 데이터 딕셔너리를 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 딕셔너리를 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 딕셔너리 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';
```

단원 4의 연습 및 해답

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

- 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:** 피벗팅 INSERT 문을 사용합니다.

연습 4-1: 대형 객체 집합 조작(계속)

- g) EMP_SALES_INFO 테이블의 레코드를 표시합니다.

	ID	WEEK	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.

	FIRST_NAME	LAST_NAME	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.

START_DATE	END_DATE	DEPARTMENT_ID
1 18-JUN-09 06.04.26.000000000 PM (null)		50
2 18-JUN-09 06.04.26.000000000 PM	18-JUN-09 06.04.26.000000000 PM	60
3 (null)	18-JUN-09 06.04.26.000000000 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 표시합니다.

```
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: 피벗팅 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
 WHERE VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
   AND LAST_NAME = 'Kochhar';
```

단원 5의 연습 및 해답

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

[SQL]	TZ_OFFSET('US/PACIFIC-NEW')
1	-07:00

- Singapore

[SQL]	TZ_OFFSET('SINGAPORE')
1	+08:00

- Egypt

[SQL]	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.

LAST_NAME	Review
King	not this year!
Whalen	not this year!
Kochhar	not this year!
Hunold	not this year!
Ernst	not this year!
De Haan	not this year!
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.

LAST_NAME	HIRE_DATE	SYSDATE	Awards
O'Connell	21-JUN-1999	23-JUN-2009	10 years of service
Grant	13-JAN-2000	23-JUN-2009	5 years of service
Whalen	17-SEP-1987	23-JUN-2009	15 years of service
Hartstein	17-FEB-1996	23-JUN-2009	10 years of service
Fay	17-AUG-1997	23-JUN-2009	10 years of service
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;
```

단원 6의 연습 및 해답

연습 6-1: Subquery를 사용하여 데이터 검색

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

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

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

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

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

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

참고: 결과 집합에 Kochhar를 표시하지 마십시오.

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

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

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

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

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1		202 Fay	20
2		201 Hartstein	20

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

- 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 정렬하고 소수점 2자리수로 반올림합니다. 예제 출력에 표시된 대로 query에 의해 검색되는 행에 alias를 사용합니다.

- 7) 관리자가 없는 모든 사원을 찾습니다.

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

	LAST_NAME
1	Abel
2	Ande
3	Atkinson
4	Austin
5	Baer
6	Baida

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

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

	LAST_NAME
1	Chen
2	Sciarrra
3	Urman
4	Popp
5	Khoo
6	Brida

- 9) 부서에서 자신보다 채용 날짜가 늦지만 더 많은 급여를 받는 동료 사원이 한 명 이상인 사원의 성을 풀시하도록 query를 작성합니다.

	LAST_NAME
1	Vargas
2	Patel
3	Olson
4	Marlow
5	Landry
6	Perkins

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.

	EMPLOYEE_ID	LAST_NAME	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.

	DEPARTMENT_NAME	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;
```

단원 7의 연습 및 해답

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.

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

	FIRST_NAME	LAST_NAME
1	Janette	King
2	Steven	King
3	Neena	Kochhar

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

	Street Address
1	1297 Via Cola di Rie
2	93091 Calle della Testa
3	2017 Shinjuku-ku
4	9450 Kamiya-cho
5	2014 Jabberwocky Rd
6	2011 Interiors Blvd
7	2007 Zagora St

- 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.

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

- 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) 555-15555'
NULL	'(215) 555-13427'
NULL	'650 555-15555'
NULL	'650 555 15555'
NULL	'650-555-15555'
NULL	'(650)555-15555'
NULL	'(650) 555-15555'

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

COUNT_DNA
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 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;
```

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

추가 연습

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
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_FORMAT을 DD-MON-YYYY HH24:MI:SS로 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 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

SYSDATE	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
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 LOCALTIMESTAMP를 표시합니다.

참고: 출력은 명령이 실행되는 날짜에 따라 다를 수 있습니다.

SYSDATE	CURRENT_DATE	CURRENT_TIMESTAMP	LOCALTIMESTAMP
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	1	13-JAN-2000
2	De Haan	1	13-JAN-1993
3	Hunold	1	03-JAN-1990
4	Landry	1	14-JAN-1999
5	Davies	1	29-JAN-1997
6	Partners	1	05-JAN-1997
7	Zlotkey	1	29-JAN-2000
8	Tucker	1	30-JAN-1997
9	King	1	30-JAN-1996
10	Marvins	1	24-JAN-2000
11	Fox	1	24-JAN-1998
12	Johnson	1	04-JAN-2000
13	Taylor	1	24-JAN-1998
14	Sarchand	1	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)

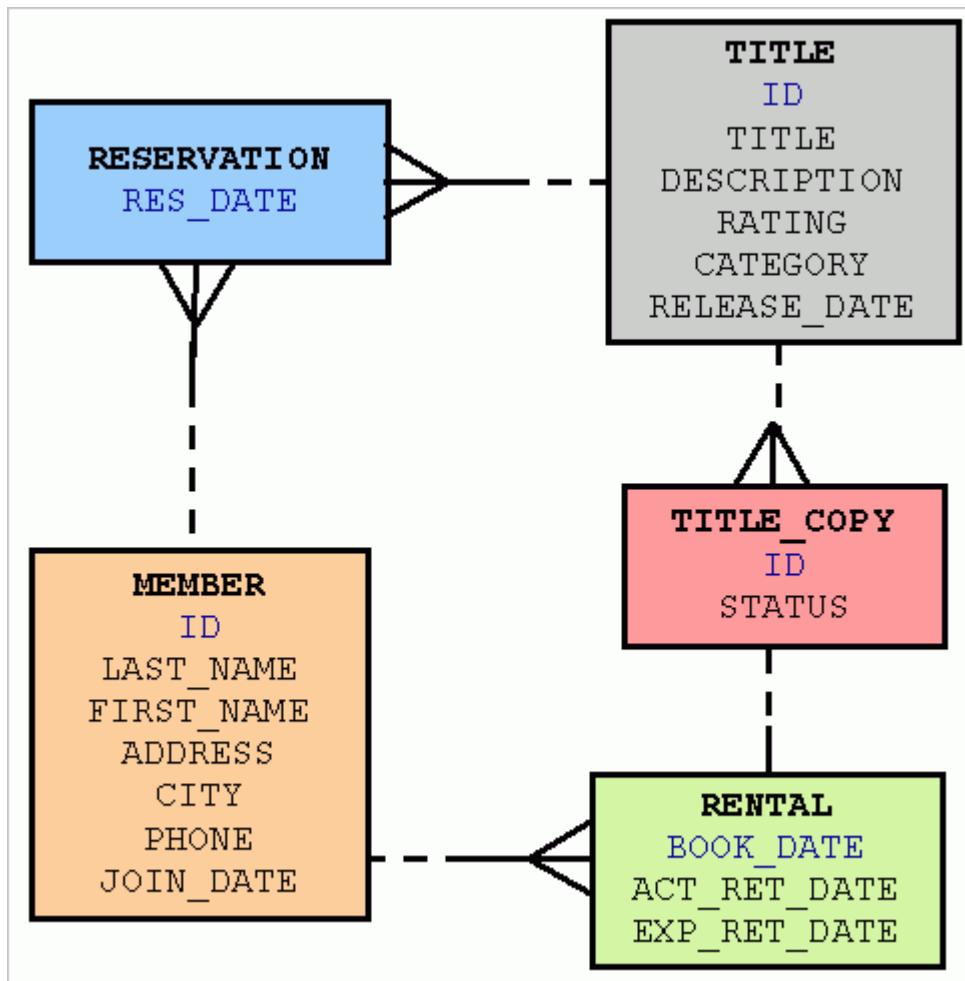
- 11) Human Resources 부사장은 사원과의 연례 간담회를 위해 전체 채용 기록을 필요로 합니다. 그는 급하게 전화해서 DBA의 지시를 중단하라고 말합니다.
트랜잭션을 롤백합니다.
- 12) 침체된 경제로 인해 비용 절감 노력을 강화하고 있습니다. CEO가 회사에서 최고 급여를 받는 직무를 검토하려고 합니다. 따라서 다음 조건에 해당하는 리스트를 CEO에게 제출해야 합니다.
- 최대 급여가 회사 전체 최대 급여의 절반 이상이 되는 직무의 직무 ID를 표시하는 query를 작성합니다. WITH 절을 사용하여 이 query를 작성하고 query 이름을 MAX_SAL_CALC로 지정합니다.

JOB_TITLE	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.

	TABLE_NAME	COLUMN_NAME	DATA_TYPE	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 딕셔너리에 MEMBER_ID_SEQ 및 TITLE_ID_SEQ 시퀀스가 있는지 확인합니다.

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 딕셔너리에서 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.

	CATEGORY	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.

	TITLE	MEMBER_ID	PRICE	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.

	MEMBER_ID	LAST_NAME	FIRST_NAME
1	101	Velasquez	Carmen

- 12) 대여 상태가 포함된 타이틀 목록을 표시합니다.

	TITLE
1	Alien Again
2	My Day Off
3	Interstellar Wars

- 13) 전화 번호에 "99"가 포함된 회원 리스트를 표시합니다.

	POSITION	MEMBER_ID	LAST_NAME	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 딕셔너리에 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 딕셔너리에서 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;
```