

Oracle Database 11g: SQL Fundamentals I(한글판)

블록 I • 학생용

D49996KR20

Edition 2.0

2009년 12월

D64452

ORACLE®

저자

Salome Clement

Brian Pottle

Puja Singh

기술 제공자 및 검토자

Anjulaponni Azhagulekshmi

Clair Bennett

Zarko Cesljas

Yanti Chang

Gerlinde Frenzen

Steve Friedberg

Joel Goodman

Nancy Greenberg

Pedro Neves

Surya Rekha

Helen Robertson

Lauran Serhal

Tulika Srivastava

편집자

Aju Kumar

Arijit Ghosh

그래픽 디자이너

Rajiv Chandrabhanu

발행인

Pavithran Adka

Veena Narasimhan

Copyright © 2009, Oracle. All rights reserved.

Disclaimer

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

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

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

목차

I 소개

단원 목표	I-2
단원 내용	I-3
과정 목표	I-4
과정 일정	I-5
본 과정에 사용되는 부록	I-7
단원 내용	I-8
Oracle Database 11g: 핵심 영역	I-9
Oracle Database 11g	I-10
Oracle Fusion Middleware	I-12
Oracle Enterprise Manager Grid Control	I-13
Oracle BI Publisher	I-14
단원 내용	I-15
관계형 및 객체 관계형 데이터베이스 관리 시스템	I-16
다양한 미디어의 데이터 저장 영역	I-17
관계형 데이터베이스 개념	I-18
관계형 데이터베이스 정의	I-19
데이터 모델	I-20
엔티티 관계 모델	I-21
엔티티 관계 모델링 규칙	I-23
여러 테이블 연관짓기	I-25
관계형 데이터베이스 용어	I-27
단원 내용	I-29
SQL을 사용하여 데이터베이스 조회	I-30
SQL 문	I-31
SQL 개발 환경	I-32
단원 내용	I-33
Human Resources(HR) 스키마	I-34
연습 과정에 사용되는 테이블	I-35
단원 내용	I-36
Oracle Database 11g 설명서	I-37
추가 자료	I-38
요약	I-39
연습 I: 개요	I-40

1 SQL SELECT 문을 사용하여 데이터 검색

- 목표 1-2
- 단원 내용 1-3
- SQL SELECT 문의 기능 1-4
- 기본 SELECT 문 1-5
- 모든 열 선택 1-6
- 특정 열 선택 1-7
- SQL 문 작성 1-8
- 열 머리글 기본값 1-9
- 단원 내용 1-10
- 산술식 1-11
- 산술 연산자 사용 1-12
- 연산자 우선 순위 1-13
- Null 값 정의 1-14
- 산술식의 Null 값 1-15
- 단원 내용 1-16
- 열 alias 정의 1-17
- 열 alias 사용 1-18
- 단원 내용 1-19
- 연결 연산자 1-20
- 리터럴 문자열 1-21
- 리터럴 문자열 사용 1-22
- 대체 인용(꺾) 연산자 1-23
- 중복 행 1-24
- 단원 내용 1-25
- 테이블 구조 표시 1-26
- DESCRIBE 명령 사용 1-27
- 퀴즈 1-28
- 요약 1-29
- 연습 1: 개요 1-30

2 데이터 제한 및 정렬

- 목표 2-2
- 단원 내용 2-3
- 선택을 사용하여 행 제한 2-4
- 선택되는 행 제한 2-5
- WHERE 절 사용 2-6
- 문자열 및 날짜 2-7
- 비교 연산자 2-8
- 비교 연산자 사용 2-9
- BETWEEN 연산자를 사용하는 범위 조건 2-10
- IN 연산자를 사용하는 멤버 조건 2-11

LIKE 연산자를 사용하여 패턴 일치	2-12
대체 문자 결합	2-13
NULL 조건 사용	2-14
논리 연산자를 사용하여 조건 정의	2-15
AND 연산자 사용	2-16
OR 연산자 사용	2-17
NOT 연산자 사용	2-18
단원 내용	2-19
우선 순위 규칙	2-20
단원 내용	2-22
ORDER BY 절 사용	2-23
정렬	2-24
단원 내용	2-26
치환 변수	2-27
단일 앰퍼샌드 치환 변수 사용	2-29
문자 및 날짜 값을 치환 변수로 지정	2-31
열 이름, 표현식 및 텍스트 지정	2-32
이중 앰퍼샌드 치환 변수 사용	2-33
단원 내용	2-34
DEFINE 명령 사용	2-35
VERIFY 명령 사용	2-36
퀴즈	2-37
요약	2-38
연습 2: 개요	2-39

3 단일 행 함수를 사용하여 출력 커스터마이징

목표	3-2
단원 내용	3-3
SQL 함수	3-4
SQL 함수의 두 가지 유형	3-5
단일 행 함수	3-6
단원 내용	3-8
문자 함수	3-9
대소문자 변환 함수	3-11
대소문자 변환 함수 사용	3-12
문자 조작 함수	3-13
문자 조작 함수 사용	3-14
단원 내용	3-15
숫자 함수	3-16
ROUND 함수 사용	3-17
TRUNC 함수 사용	3-18
MOD 함수 사용	3-19

단원 내용 3-20
 날짜 작업 3-21
 RR 날짜 형식 3-22
 SYSDATE 함수 사용 3-24
 날짜 연산 3-25
 날짜에 산술 연산자 사용 3-26
 단원 내용 3-27
 날짜 조작 함수 3-28
 날짜 함수 사용 3-30
 날짜에 ROUND 및 TRUNC 함수 사용 3-31
 퀴즈 3-32
 요약 3-33
 연습 3: 개요 3-34

4 변환 함수 및 조건부 표현식 사용

목표 4-2
 단원 내용 4-3
 변환 함수 4-4
 암시적 데이터 유형 변환 4-5
 명시적 데이터 유형 변환 4-7
 단원 내용 4-10
 날짜에 TO_CHAR 함수 사용 4-11
 날짜 형식 모델의 요소 4-12
 날짜에 TO_CHAR 함수 사용 4-16
 숫자에 TO_CHAR 함수 사용 4-17
 TO_NUMBER 및 TO_DATE 함수 사용 4-20
 RR 날짜 형식으로 TO_CHAR 및 TO_DATE 함수 사용 4-22
 단원 내용 4-23
 함수 중첩 4-24
 함수 중첩: 예제 1 4-25
 함수 중첩: 예제 2 4-26
 단원 내용 4-27
 일반 함수 4-28
 NVL 함수 4-29
 NVL 함수 사용 4-30
 NVL2 함수 사용 4-31
 NULLIF 함수 사용 4-32
 COALESCE 함수 사용 4-33
 단원 내용 4-36
 조건부 표현식 4-37
 CASE 식 4-38
 CASE 식 사용 4-39

DECODE 함수 4-40
DECODE 함수 사용 4-41
퀴즈 4-43
요약 4-44
연습 4: 개요 4-45

5 그룹 함수를 사용하여 집계된 데이터 보고

목표 5-2
단원 내용 5-3
그룹 함수란? 5-4
그룹 함수 유형 5-5
그룹 함수: 구문 5-6
AVG 및 SUM 함수 사용 5-7
MIN 및 MAX 함수 사용 5-8
COUNT 함수 사용 5-9
DISTINCT 키워드 사용 5-10
그룹 함수 및 null 값 5-11
단원 내용 5-12
데이터 그룹 생성 5-13
데이터 그룹 생성: GROUP BY 절 구문 5-14
GROUP BY 절 사용 5-15
두 개 이상의 열로 그룹화 5-17
여러 열에서 GROUP BY 절 사용 5-18
그룹 함수를 사용한 잘못된 Query 5-19
그룹 결과 제한 5-21
HAVING 절을 사용하여 그룹 결과 제한 5-22
HAVING 절 사용 5-23
단원 내용 5-25
그룹 함수 중첩 5-26
퀴즈 5-27
요약 5-28
연습 5: 개요 5-29

6 조인을 사용하여 여러 테이블의 데이터 표시

목표 6-2
단원 내용 6-3
여러 테이블에서 데이터 가져오기 6-4
조인 유형 6-5
SQL:1999 구문을 사용한 테이블 조인 6-6
모호한 열 이름 한정 6-7
단원 내용 6-9
Natural Join 생성 6-10

Natural join으로 레코드 검색 6-11
 USING 절로 조인 생성 6-12
 열 이름 조인 6-13
 USING 절을 사용하여 레코드 검색 6-14
 USING 절에 테이블 Alias 사용 6-15
 ON 절로 조인 생성 6-17
 ON 절을 사용하여 레코드 검색 6-18
 ON 절을 사용하여 3-Way 조인 생성 6-19
 조인에 추가 조건 적용 6-20
 단원 내용 6-21
 테이블 자체 조인 6-22
 ON 절을 사용하는 Self Join 6-23
 단원 내용 6-24
 Nonequijoin 6-25
 Nonequijoin을 사용하여 레코드 검색 6-26
 단원 내용 6-27
 OUTER Join과 직접 일치하지 않는 레코드 반환 6-28
 INNER Join과 OUTER Join 6-29
 LEFT OUTER JOIN 6-30
 RIGHT OUTER JOIN 6-31
 FULL OUTER JOIN 6-32
 단원 내용 6-33
 Cartesian Product 6-34
 Cartesian Product 생성 6-35
 Cross Join 생성 6-36
 퀴즈 6-37
 요약 6-38
 연습 6: 개요 6-39

7 Subquery를 사용하여 Query 해결

목표 7-2
 단원 내용 7-3
 Subquery를 사용하여 문제 해결 7-4
 Subquery 구문 7-5
 Subquery 사용 7-6
 Subquery 사용 지침 7-7
 Subquery 유형 7-8
 단원 내용 7-9
 단일 행 Subquery 7-10
 단일 행 Subquery 실행 7-11
 Subquery에서 그룹 함수 사용 7-12
 Subquery가 있는 HAVING 절 7-13

이 명령문에서 잘못된 점은? 7-14
 Inner query에서 반환된 행이 없음 7-15
 단원 내용 7-16
 여러 행 Subquery 7-17
 여러 행 Subquery에서 ANY 연산자 사용 7-18
 여러 행 Subquery에서 ALL 연산자 사용 7-19
 EXISTS 연산자 사용 7-20
 단원 내용 7-21
 Subquery의 null 값 7-22
 퀴즈 7-24
 요약 7-25
 연습 7: 개요 7-26

8 집합 연산자 사용

목표 8-2
 단원 내용 8-3
 집합 연산자 8-4
 집합 연산자 지침 8-5
 Oracle 서버 및 집합 연산자 8-6
 단원 내용 8-7
 이 단원에 사용되는 테이블 8-8
 단원 내용 8-12
 UNION 연산자 8-13
 UNION 연산자 사용 8-14
 UNION ALL 연산자 8-16
 UNION ALL 연산자 사용 8-17
 단원 내용 8-18
 INTERSECT 연산자 8-19
 INTERSECT 연산자 사용 8-20
 단원 내용 8-21
 MINUS 연산자 8-22
 MINUS 연산자 사용 8-23
 단원 내용 8-24
 SELECT 문 일치 8-25
 SELECT 문 일치: 예제 8-26
 단원 내용 8-27
 집합 연산에서 ORDER BY 절 사용 8-28
 퀴즈 8-29
 요약 8-30
 연습 8: 개요 8-31

9 데이터 조작

- 목표 9-2
- 단원 내용 9-3
- DML(데이터 조작어) 9-4
- 테이블에 새 행 추가 9-5
- INSERT 문 구문 9-6
- 새 행 삽입 9-7
- Null 값을 가진 행 삽입 9-8
- 특수 값 삽입 9-9
- 특정 날짜 및 시간 값 삽입 9-10
- 스크립트 작성 9-11
- 다른 테이블에서 행 복사 9-12
- 단원 내용 9-13
- 테이블의 데이터 변경 9-14
- UPDATE 문 구문 9-15
- 테이블의 행 갱신 9-16
- Subquery를 사용하여 두 개의 열 갱신 9-17
- 다른 테이블을 기반으로 행 갱신 9-19
- 단원 내용 9-20
- 테이블에서 행 제거 9-21
- DELETE 문 9-22
- 테이블에서 행 삭제 9-23
- 다른 테이블을 기반으로 행 삭제 9-24
- TRUNCATE 문 9-25
- 단원 내용 9-26
- 데이터베이스 트랜잭션 9-27
- 데이터베이스 트랜잭션: 시작과 종료 9-28
- COMMIT 및 ROLLBACK 문의 이점 9-29
- 명시적 트랜잭션 제어문 9-30
- 변경 사항을 표시자로 롤백 9-31
- 암시적 트랜잭션 처리 9-32
- COMMIT 또는 ROLLBACK 전의 데이터 상태 9-34
- COMMIT 후의 데이터 상태 9-35
- 데이터 커밋 9-36
- ROLLBACK 후의 데이터 상태 9-37
- ROLLBACK 후의 데이터 상태: 예제 9-38
- 명령문 레벨 롤백 9-39
- 단원 내용 9-40
- 읽기 일관성 9-41
- 읽기 일관성 구현 9-42
- 단원 내용 9-43
- SELECT 문의 FOR UPDATE 절 9-44

FOR UPDATE 절: 예제 9-45

퀴즈 9-47

요약 9-48

연습 9: 개요 9-49

10 DDL 문을 사용하여 테이블 생성 및 관리

목표 10-2

단원 내용 10-3

데이터베이스 객체 10-4

이름 지정 규칙 10-5

단원 내용 10-6

CREATE TABLE 문 10-7

다른 유저의 테이블 참조 10-8

DEFAULT 옵션 10-9

테이블 생성 10-10

단원 내용 10-11

데이터 유형 10-12

Datetime 데이터 유형 10-14

단원 내용 10-15

제약 조건 포함 10-16

제약 조건 지침 10-17

제약 조건 정의 10-18

NOT NULL 제약 조건 10-20

UNIQUE 제약 조건 10-21

PRIMARY KEY 제약 조건 10-23

FOREIGN KEY 제약 조건 10-24

FOREIGN KEY 제약 조건: 키워드 10-26

CHECK 제약 조건 10-27

CREATE TABLE: 예제 10-28

제약 조건 위반 10-29

단원 내용 10-31

Subquery를 사용하여 테이블 생성 10-32

단원 내용 10-34

ALTER TABLE 문 10-35

읽기 전용 테이블 10-36

단원 내용 10-37

테이블 삭제 10-38

퀴즈 10-39

요약 10-40

연습 10: 개요 10-41

11 기타 스키마 객체 생성

- 목표 11-2
- 단원 내용 11-3
- 데이터베이스 객체 11-4
- 뷰란? 11-5
- 뷰의 이점 11-6
- 단순 뷰와 복합 뷰 11-7
- 뷰 생성 11-8
- 뷰에서 데이터 검색 11-11
- 뷰 수정 11-12
- 복합 뷰 생성 11-13
- 뷰에 대한 DML 작업 수행 규칙 11-14
- WITH CHECK OPTION 절 사용 11-17
- DML 작업 거부 11-18
- 뷰 제거 11-20
- 연습 11: 1부 개요 11-21
- 단원 내용 11-22
- 시퀀스 11-23
- CREATE SEQUENCE 문: 구문 11-25
- 시퀀스 생성 11-26
- NEXTVAL 및 CURRVAL Pseudocolumn 11-27
- 시퀀스 사용 11-29
- 시퀀스 값 캐시 11-30
- 시퀀스 수정 11-31
- 시퀀스 수정 지침 11-32
- 단원 내용 11-33
- 인덱스 11-34
- 인덱스가 생성되는 방식 11-36
- 인덱스 생성 11-37
- 인덱스 생성 지침 11-38
- 인덱스 제거 11-39
- 단원 내용 11-40
- 동의어 11-41
- 객체의 동의어 생성 11-42
- 동의어 생성 및 제거 11-43
- 퀴즈 11-44
- 요약 11-45
- 연습 11: 2부 개요 11-46

부록 A: 연습 해답**부록 B: 테이블 설명****부록 C: SQL Developer 사용**

- 목표 C-2
- Oracle SQL Developer란? C-3
- SQL Developer 사양 C-4
- SQL Developer 1.5 인터페이스 C-5
- 데이터베이스 연결 생성 C-7
- 데이터베이스 객체 탐색 C-10
- 테이블 구조 표시 C-11
- 파일 탐색 C-12
- 스키마 객체 생성 C-13
- 새 테이블 생성: 예제 C-14
- SQL Worksheet 사용 C-15
- SQL 문 실행 C-18
- SQL 스크립트 저장 C-19
- 저장된 SQL 스크립트 실행: 방법 1 C-20
- 저장된 SQL 스크립트 실행: 방법 2 C-21
- SQL 코드 형식 지정 C-22
- Snippet 사용 C-23
- Snippet 사용: 예제 C-24
- 프로시저 및 함수 디버깅 C-25
- 데이터베이스 보고 C-26
- 유저 정의 보고서 작성 C-27
- 검색 엔진 및 External 도구 C-28
- 환경 설정 C-29
- SQL Developer 레이아웃 재설정 C-30
- 요약 C-31

부록 D: SQL*Plus 사용

- 목표 D-2
- SQL과 SQL*Plus의 상호 작용 D-3
- SQL 문과 SQL*Plus 명령 비교 D-4
- SQL*Plus 개요 D-5
- SQL*Plus에 로그인 D-6
- 테이블 구조 표시 D-7
- SQL*Plus 편집 명령 D-9
- LIST, n 및 APPEND 사용 D-11
- CHANGE 명령 사용 D-12
- SQL*Plus 파일 명령 D-13

SAVE 및 START 명령 사용 D-15
SERVEROUTPUT 명령 D-16
SQL*Plus SPOOL 명령 사용 D-17
AUTOTRACE 명령 사용 D-18
요약 D-19

부록 E: JDeveloper 사용

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

부록 F: Oracle 조인 구문

목표 F-2
여러 테이블에서 데이터 가져오기 F-3
Cartesian Product F-4
Cartesian Product 생성 F-5
Oracle 고유의 조인 유형 F-6
Oracle 구문을 사용하여 테이블 조인 F-7
모호한 열 이름 한정 F-8
Equijoin F-10
Equijoin을 사용하여 레코드 검색 F-11
Equijoin을 사용하여 레코드 검색: 예제 F-12
AND 연산자를 사용한 추가 검색 조건 F-13
세 개 이상의 테이블 조인 F-14

Nonequijoin F-15
Nonequijoin을 사용하여 레코드 검색 F-16
Outer Join을 사용하여 직접 일치되지 않는 레코드 반환 F-17
Outer Join: 구문 F-18
Outer Join 사용 F-19
Outer Join: 다른 예제 F-20
테이블 자체 조인 F-21
Self Join: 예제 F-22
요약 F-23
연습 F: 개요 F-24

추가 연습 및 해답

색인

I 소개

ORACLE®

Copyright © 2009, Oracle. All rights reserved.

단원 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 과정 목표 정의
- Oracle Database 11g의 기능 나열
- 관계형 데이터베이스의 이론적 측면과 물리적 측면 설명
- Oracle 서버의 RDBMS 및 ORDBMS (객체 관계형 데이터베이스 관리 시스템) 구현 설명
- 본 과정에서 사용할 수 있는 개발 환경 식별
- 본 과정에 사용된 데이터베이스 및 스키마 설명

ORACLE

Copyright © 2009, 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 환경으로서의 Oracle SQL Developer 및 SQL*Plus를 소개합니다.

단원 내용

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

과정 목표

본 과정을 마치면 다음을 수행할 수 있습니다.

- Oracle Database 11g의 주요 구성 요소 파악
- SELECT 문을 사용하여 테이블에서 행 및 열 데이터 검색
- 정렬되고 제한된 데이터로 보고서 작성
- SQL 함수를 사용하여 커스터마이징된 데이터 생성 및 검색
- 복합 query를 실행하여 여러 테이블에서 데이터 검색
- Oracle Database 11g에서 DML (데이터 조작어) 문을 실행하여 데이터 갱신
- DDL (데이터 정의어) 문을 실행하여 스키마 객체 생성 및 관리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

과정 목표

본 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 조작하며, 데이터베이스 객체를 생성하고, 메타 데이터를 query할 수 있는 필수 SQL 기술을 배웁니다.

과정 일정

- **첫째 날:**
 - 소개
 - SQL SELECT 문을 사용하여 데이터 검색
 - 데이터 제한 및 정렬
 - 단일 행 함수를 사용하여 출력 커스터마이징
 - 변환 함수 및 조건부 표현식 사용
- **둘째 날:**
 - 그룹 함수를 사용하여 집계된 데이터 보고
 - 조인을 사용하여 여러 테이블의 데이터 표시
 - subquery를 사용하여 query 해결
 - 집합 연산자 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

과정 일정

- 셋째 날:
 - 데이터 조작
 - DDL 문을 사용하여 테이블 생성 및 관리
 - 기타 스키마 객체 생성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

본 과정에 사용되는 부록

- 부록 A: 연습 및 해답
- 부록 B: 테이블 설명
- 부록 C: SQL Developer 사용
- 부록 D: SQL*Plus 사용
- 부록 E: JDeveloper 사용
- 부록 F: Oracle 조인 구문
- 부록 AP: 추가 연습 및 해답

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단원 내용

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Database 11g: 핵심 영역



Infrastructure
그리드

정보
관리

응용 프로그램
개발

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g: 핵심 영역

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

- **Infrastructure 그리드(Infrastructure Grids):** Oracle의 Infrastructure 그리드 기술을 사용하면 저비용 서버 및 저장 영역을 도입하여 관리 효율성, 고가용성(High Availability) 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(Information Management):** Oracle Database 11g에서는 기존의 정보 관리 기능을 콘텐츠 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 콘텐츠를 관리합니다.
- **응용 프로그램 개발(Application Development):** 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



관리 효율성

고가용성

성능

보안

정보 통합

ORACLE

Copyright © 2009, 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(Manageability):** 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(High availability):** 고가용성 기능을 사용하여 다운타임 및 데이터 손실을 줄일 수 있습니다. 이러한 기능으로 온라인 작업을 개선하고 데이터베이스를 더 빠르게 업그레이드할 수 있습니다.

Oracle Database 11g(계속)

- **성능(Performance):** 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(Security):** 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(Information integration):** Oracle Database 11g는 전사적으로 데이터를 보다 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 콘텐츠 관리를 비롯한 광범위한 Java EE의 도구와 서비스 및 개발 도구를 포괄하며 고객에 의해 입증된 첨단 표준 기반 소프트웨어 제품 포트폴리오



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Fusion Middleware

Oracle Fusion Middleware는 SOA (Service-Oriented Architecture)의 개발, 배포 및 관리를 완전하게 지원하는 포괄적인 통합 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 기존의 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 도구:** 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
- **컨텐츠 관리:** Oracle Content Database Suite
- **유저 상호 작용:** Portal, WebCenter

Oracle Enterprise Manager Grid Control

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Enterprise Manager Grid Control

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

Oracle Enterprise Manager Grid Control에는 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** - 커스터마이징된 응용 프로그램, Oracle E-Business Suite, PeopleSoft, Siebel, Oracle Fusion Middleware, 오라클 데이터베이스 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)
- 유연한 커스터마이제이션 가능
 - 예를 들어, PDF, HTML, Excel, RTF 등의 다양한 형식으로 Microsoft Word 문서 보고서를 생성할 수 있습니다.



ORACLE

Copyright © 2009, 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 친숙한 데스크톱 도구 집합을 사용하여, IT 직원이나 개발자가 작성한 데이터 query를 기반으로 보고서 형식을 직접 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 본 과정에 사용되는 부록
- Oracle Database 11g 및 관련 제품 개요
- **관계형 데이터베이스 관리 개념 및 용어 개요**
- SQL 및 개발 환경 소개
- 본 과정에 사용되는 HR 스키마 및 테이블
- Oracle Database 11g 설명서 및 추가 자료

ORACLE

Copyright © 2009, 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 © 2009, Oracle. All rights reserved.

관계형 및 객체 관계형 데이터베이스 관리 시스템

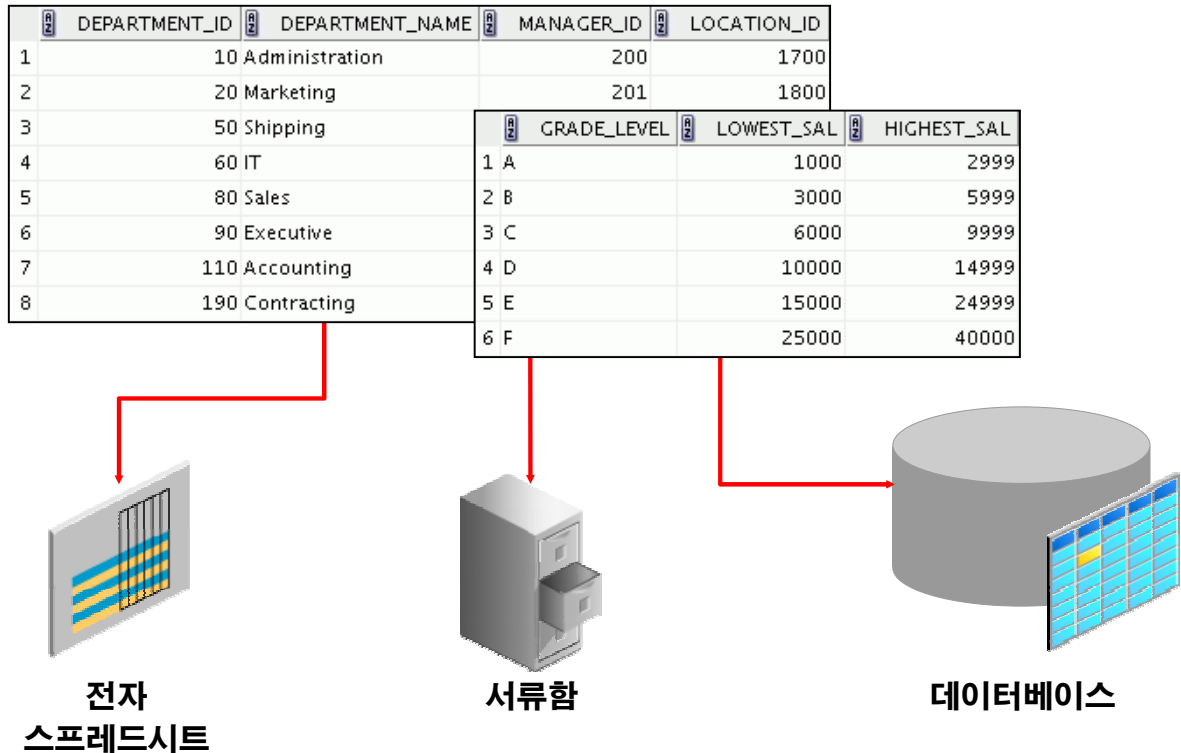
Oracle 서버는 관계형 및 객체 관계형 데이터베이스 모델을 모두 지원합니다.

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. 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)*을 참조하십시오.

다양한 미디어의 데이터 저장 영역



ORACLE

Copyright © 2009, 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 *데이터*라고 합니다.

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

*데이터베이스*는 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 © 2009, Oracle. All rights reserved.

관계형 데이터베이스 개념

관계형 모델의 원리는 E. F. Codd 박사가 1970년 6월 발표한 *A Relational Model of Data for Large Shared Data Banks*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 토털 솔루션을 제공하여 RDBMS를 지원했습니다.

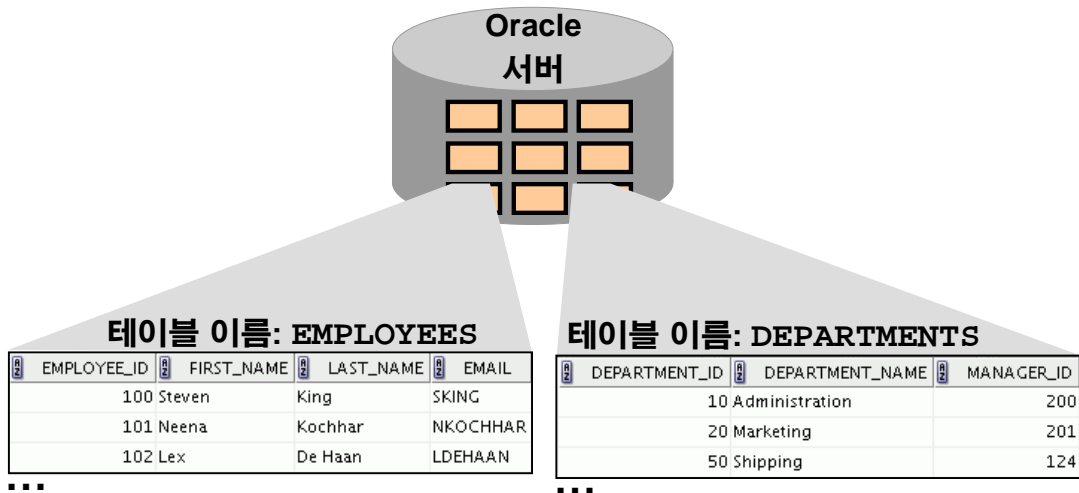
관계형 모델의 구성 요소

- 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.



ORACLE

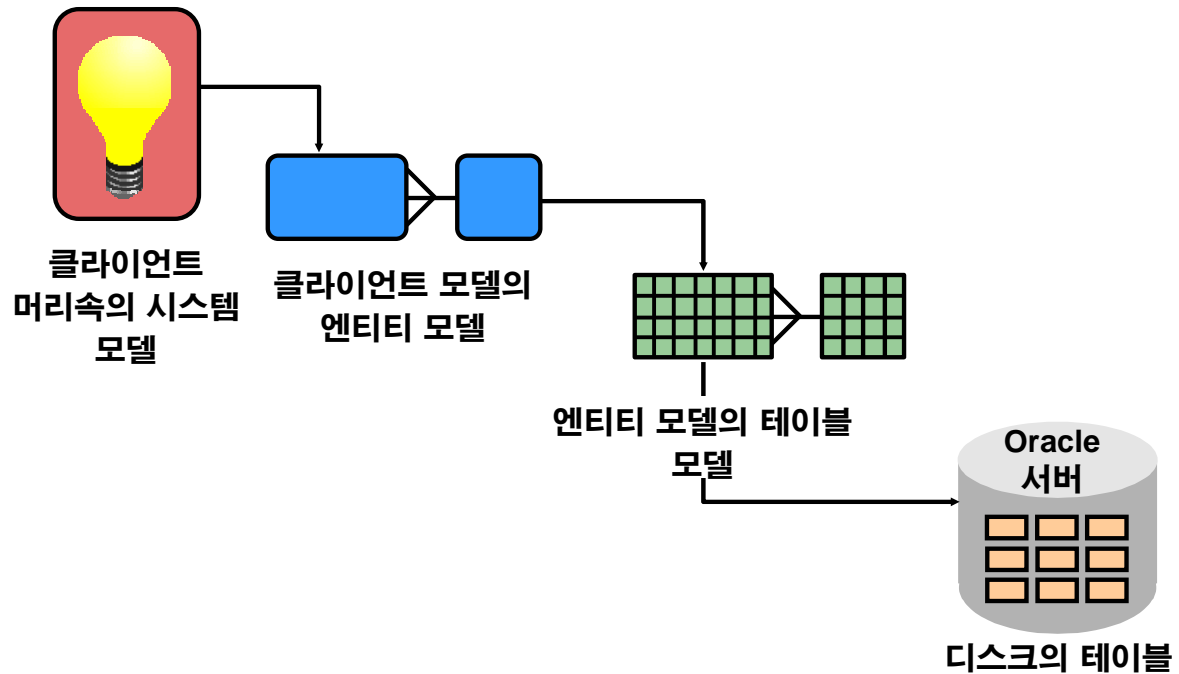
Copyright © 2009, Oracle. All rights reserved.

관계형 데이터베이스 정의

관계형 데이터베이스는 관계 또는 2차원 테이블을 사용하여 정보를 저장합니다.

예를 들어, 회사의 모든 사원에 대한 정보를 저장하려는 경우가 있습니다. 관계형 데이터베이스에서는 사원 테이블, 부서 테이블, 급여 테이블 등 여러 테이블을 생성하여 사원에 대한 서로 다른 정보 단위를 저장합니다.

데이터 모델



ORACLE

Copyright © 2009, 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

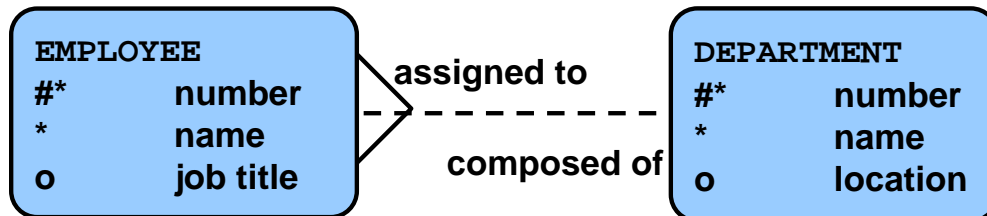
모델은 사람들의 머리속에 있는 개념을 표출하는 데 도움이 됩니다. 다음 작업을 수행하는 데 함수를 사용할 수 있습니다.

- 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 진술을 토대로 엔티티 관계 다이어그램을 생성합니다.



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

ORACLE

Copyright © 2009, 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 진술로부터 파생되고 시스템 개발 주기의 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.

엔티티 관계 모델(계속)

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 이름 지정된 연관. 관계의 예로는 사원과 부서, 주문과 품목 등이 있습니다.

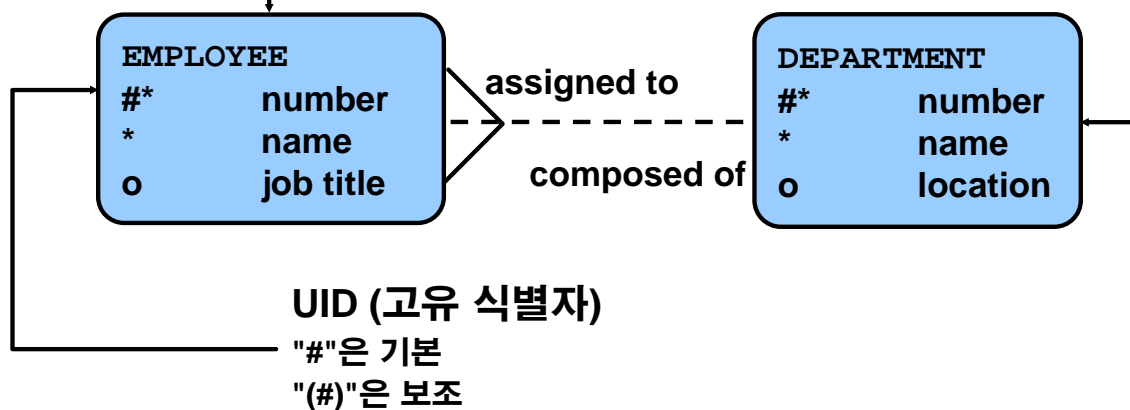
엔티티 관계 모델링 규칙

엔티티:

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

속성:

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



ORACLE

Copyright © 2009, 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*

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

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

참고: 규칙은 시계방향으로 읽습니다.

고유 식별자

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

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

여러 테이블 연관짓기

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

테이블 이름: EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
100	Steven	King	90
101	Neena	Kochhar	90
102	Lex	De Haan	90
103	Alexander	Hunold	60
104	Bruce	Ernst	60
107	Diana	Lorentz	60
124	Kevin	Mourgos	50
141	Trenna	Rajs	50
142	Curtis	Davies	50

Primary Key

Foreign Key

테이블 이름: DEPARTMENTS

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

Primary Key

ORACLE

Copyright © 2009, 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 필요가 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 값은 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	Jonathan	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 © 2009, Oracle. All rights reserved.

관계형 데이터베이스 용어

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

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

1. 특정 사원에 필요한 모든 데이터를 나타내는 단일 행(또는 튜플)입니다. 테이블의 각 행은 중복 행을 허용하지 않는 **Primary key**로 식별되어야 합니다. 행의 순서는 중요하지 않습니다. 데이터를 검색할 때 행 순서를 지정합니다.
2. 사원 번호를 포함하는 열 또는 속성입니다. 사원 번호는 EMPLOYEES 테이블에서 **고유한** 사원을 식별합니다. 이 예제에서 사원 번호 열은 **Primary key**로 지정됩니다. **Primary key**는 값을 포함해야 하고 그 값은 고유해야 합니다.
3. 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 key*나 *Unique key*를 참조합니다. 예제에서 `DEPARTMENT_ID`는 `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 롤을 가진 사원만 `COMMISSION_PCT` (*commission*) 필드에 값이 있습니다.

단원 내용

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

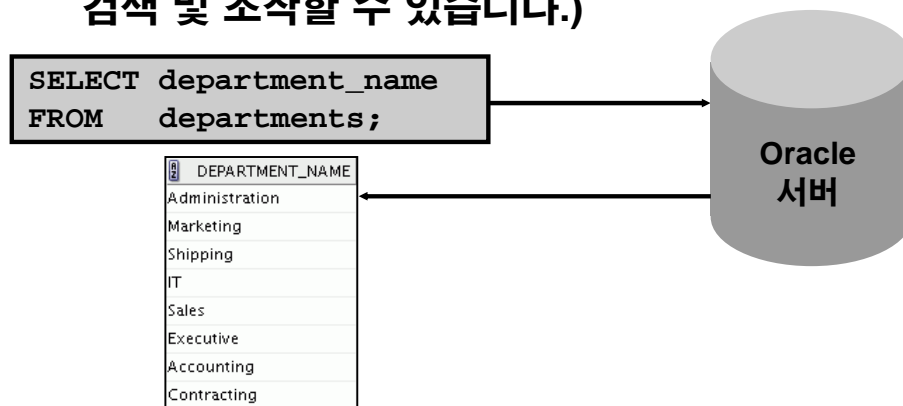
ORACLE

Copyright © 2009, 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(SQL을 사용하면 테이블의 데이터를 정의, 검색 및 조작할 수 있습니다.)



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL을 사용하여 데이터베이스 query

관계형 데이터베이스에서는 테이블에 대한 액세스 경로를 지정하지 않으며 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 사용자가 오라클 데이터베이스의 데이터에 액세스하기 위해 사용하는 일련의 명령문입니다. 응용 프로그램 및 Oracle 도구에서 사용자가 SQL을 직접 사용하지 않고 데이터베이스에 액세스하도록 허용하는 경우가 있지만 이러한 응용 프로그램에서 사용자의 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 사용자는 SQL을 통해 논리적 레벨에서 데이터를 다룰 수 있습니다.

SQL 문

SELECT INSERT UPDATE DELETE MERGE	DML (데이터 조작어)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	DDL (데이터 정의어)
GRANT REVOKE	DCL (데이터 제어어)
COMMIT ROLLBACK SAVEPOINT	트랜잭션 제어

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 문

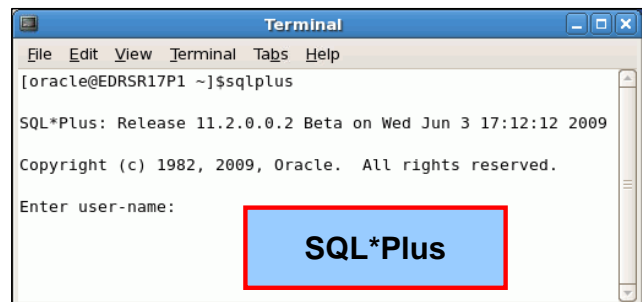
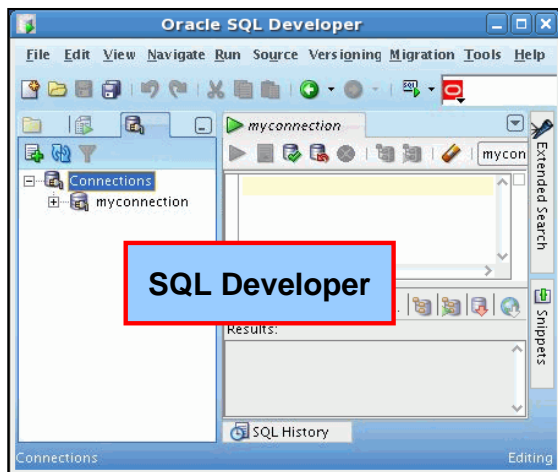
Oracle에서 지원하는 SQL 문은 산업 표준을 준수합니다. Oracle Corporation은 SQL 표준 위원회의 핵심 멤버로 적극 참여하여 앞으로 개발되는 표준도 준수할 것입니다. 업계 승인 위원회인 ANSI와 ISO (International Standards Organization)는 모두 SQL을 관계형 데이터베이스에 적합한 표준 언어로 승인했습니다.

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

SQL 개발 환경

본 과정에서는 다음 두 가지의 개발 환경을 사용합니다.

- 기본 도구는 Oracle SQL Developer입니다.
- SQL*Plus 명령행 인터페이스도 사용할 수 있습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 개발 환경

SQL Developer

본 과정은 슬라이드 및 연습의 예제에 설명된 SQL 문을 실행하는 도구로 Oracle SQL Developer를 사용하여 개발되었습니다. SQL Developer 버전 1.5.4는 Oracle Database 11g와 함께 제공되며 이 클래스의 기본 도구입니다.

SQL*Plus

SQL*Plus 환경은 본 과정에서 다루는 모든 SQL 명령을 실행하는 데 사용할 수도 있습니다.

참고

- 자세한 SQL Developer 사용법과 버전 1.5.4를 설치하는 간단한 지침은 부록 C를 참조하십시오.
- SQL*Plus 사용에 대한 자세한 내용은 부록 D를 참조하십시오.

단원 내용

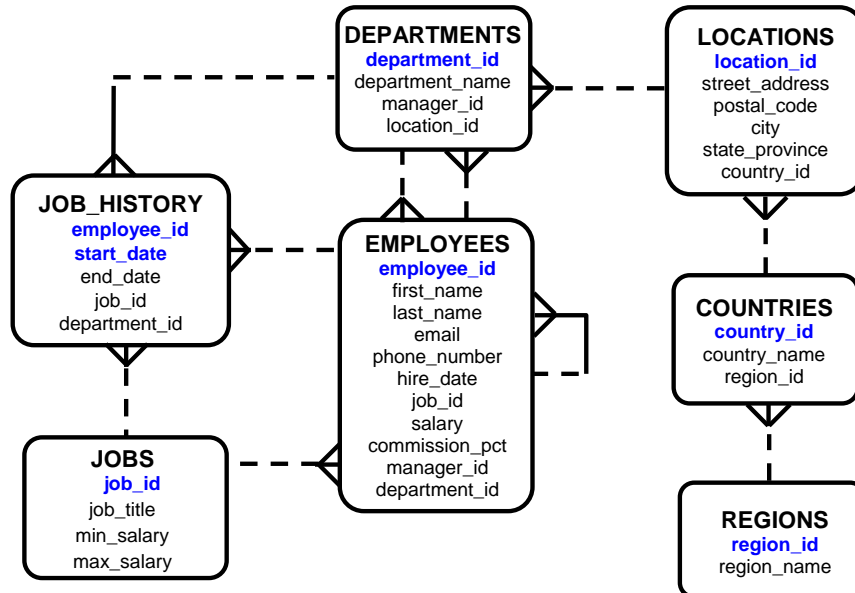
- 과정 목표, 과정 내용 및 본 과정에 사용되는 부록
- Oracle Database 11g 및 관련 제품 개요
- 관계형 데이터베이스 관리 개념 및 용어 개요
- SQL 및 개발 환경 소개
- **본 과정에 사용되는 HR 스키마 및 테이블**
- Oracle Database 11g 설명서 및 추가 자료

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Human Resources(HR) 스키마



ORACLE

Copyright © 2009, Oracle. All rights reserved.

HR (Human Resources) 스키마 설명

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

테이블 설명

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

연습 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
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	80	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIEZT	515.123.8181	07-JUN-94

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 과정에 사용되는 테이블

다음은 본 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.

참고: 모든 테이블에 대한 구조 및 데이터는 부록 B를 참조하십시오.

단원 내용

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g 설명서

- *Oracle Database New Features Guide 11g, Release 1 (11.2)*
- *Oracle Database Reference 11g, Release 1 (11.2)*
- *Oracle Database SQL Language Reference 11g, Release 1 (11.2)*
- *Oracle Database Concepts 11g, Release 1 (11.2)*
- *Oracle Database SQL Developer User's Guide, Release 1.5*

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Database 11g 설명서

Oracle Database 11g 설명서 라이브러리에 액세스하려면 <http://www.oracle.com/pls/db112/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/obe/11gr1_db/index.htm

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- Oracle Database 11g는 다음과 같은 이점 및 기능을 확장합니다.
 - Infrastructure 그리드의 이점
 - 기존 정보 관리 기능
 - PL/SQL, Java/JDBC, .NET, XML 등의 주요 응용 프로그램 개발 환경 사용 기능
- 데이터베이스는 ORDBMS를 기반으로 합니다.
- 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 © 2009, Oracle. All rights reserved.

요약

관계형 데이터베이스 관리 시스템은 객체 또는 관계로 구성됩니다. 이들 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

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 시작
- 새 데이터베이스 생성
- HR 테이블 탐색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 I: 개요

이 연습에서는 다음을 수행합니다.

- Oracle SQL Developer를 시작하고 ora1 계정에 대한 새 연결을 생성합니다.
- Oracle SQL Developer를 사용하여 ora1 계정에서 데이터 객체를 검사합니다. ora1 계정에는 HR 스키마 테이블이 포함됩니다.

다음 lab 파일 위치를 기록해 두십시오.

```
\home\oracle\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.

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

1

SQL SELECT 문을 사용하여 데이터 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

데이터베이스에서 데이터를 추출하려면 SQL SELECT 문을 사용해야 합니다. 표시되는 열을 제한해야 하는 경우도 있습니다. 이 단원에서는 이러한 작업을 수행하는 데 필요한 SELECT 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 © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

SQL SELECT 문의 기능

프로젝션

테이블 1

선택

테이블 1

테이블 1

조인

테이블 2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL SELECT 문의 기능

SELECT 문은 데이터베이스에서 정보를 검색합니다. SELECT 문으로 다음을 수행할 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 "조인을 사용하여 여러 테이블의 데이터 표시" 단원에서 자세히 다룹니다.

기본 SELECT 문

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

- **SELECT는 표시할 열을 식별합니다.**
- **FROM은 이러한 열을 포함한 테이블을 식별합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

기본 SELECT 문

가장 간단한 형태의 SELECT 문은 다음을 포함해야 합니다.

- 표시할 열을 지정하는 SELECT 절
- SELECT 절에 나열된 열을 포함한 테이블을 식별하는 FROM 절 이 구문에서 다음이 적용됩니다.

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

참고: 본 과정 전체에 걸쳐 *키워드*, *절*, *명령문*이라는 단어는 다음과 같이 사용됩니다.

- *키워드*는 개별 SQL 요소를 나타냅니다. 예를 들어, SELECT 및 FROM 등이 키워드입니다.
- *절*은 SQL 문의 일부입니다. 예를 들어, SELECT employee_id, last_name 등이 절입니다.
- *명령문*은 둘 이상의 절이 결합된 것입니다. SELECT * FROM employees를 예로 들 수 있습니다.

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 © 2009, 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;
```

참고: 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 © 2009, 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 Developer에서 SQL 문은 선택적으로 세미콜론(;)으로 끝낼 수 있습니다. 세미콜론은 여러 SQL 문을 실행하는 경우에 필요합니다.
- SQL*plus에서는 각 SQL 문이 반드시 세미콜론(;)으로 끝나야 합니다.

ORACLE

Copyright © 2009, 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 둡니다.
- 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 © 2009, Oracle. All rights reserved.

열 머리글 기본값

SQL Developer에서 열 머리글은 대문자로 표시되고 왼쪽 정렬됩니다.

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

	A Z LAST_NAME	A Z HIRE_DATE	A Z SALARY
1	Whalen	17-SEP-87	4400
2	Hartstein	17-FEB-96	13000
3	Fay	17-AUG-97	6000
4	Higgins	07-JUN-94	12000
5	Gietz	07-JUN-94	8300
6	King	17-JUN-87	24000
7	Kochhar	21-SEP-89	17000
8	De Haan	13-JAN-93	17000
9	Hunold	03-JAN-90	9000

■ ■ ■

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 © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

산술식

산술 연산자를 사용하여 숫자 및 날짜 데이터로 표현식을 작성합니다.

연산자	설명
+	더하기
-	빼기
*	곱하기
/	나누기

ORACLE

Copyright © 2009, Oracle. All rights reserved.

산술식

데이터 표시 방식을 수정하거나 계산을 수행하거나 가정(what-if) 시나리오를 조사해야 할 필요가 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.

참고: 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	Whalen	4400	4700
2	Hartstein	13000	13300
3	Fay	6000	6300
4	Higgins	12000	12300
5	Gietz	8300	8600
6	King	24000	24300
7	Kochhar	17000	17300
8	De Haan	17000	17300
9	Hunold	9000	9300
10	Ernst	6000	6300

...

ORACLE

Copyright © 2009, 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.

참고: 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 명령문을 명확히 하기 위해 사용됩니다.

연산자 우선 순위

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

1

	LAST_NAME	SALARY	12*SALARY+100
1	Whalen	4400	52900
2	Hartstein	13000	156100
3	Fay	6000	72100

...

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

2

	LAST_NAME	SALARY	12*(SALARY+100)
1	Whalen	4400	54000
2	Hartstein	13000	157200
3	Fay	6000	73200

...

ORACLE

Copyright © 2009, 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.

참고: 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 원하는 대로 지정함으로써 우선 순위 규칙을 재정의할 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;
```

R	LAST_NAME	R	JOB_ID	R	SALARY	R	COMMISSION_PCT
1	Whalen		AD_ASST		4400		(null)
2	Hartstein		MK_MAN		13000		(null)
...							
17	Zlotkey		SA_MAN		10500		0.2
18	Abel		SA_REP		11000		0.3
19	Taylor		SA_REP		8600		0.2
20	Grant		SA_REP		7000		0.15

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Null 값 정의

행에 특정 열에 대한 데이터 값이 없는 경우 해당 값이 *null*이거나 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.

참고: 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 노드를 확장합니다. 오른쪽 창에서 Advanced Parameters를 누르고 "Display Null value As"에 대해 적절한 값을 입력합니다.

산술식의 Null 값

null 값을 포함하는 산술식은 null로 계산됩니다.

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

	LAST_NAME	12*SALARY*COMMISSION_PCT
1	Whalen	(null)
2	Hartstein	(null)
3	Fay	(null)
...		
17	Zlotkey	25200
18	Abel	39600
19	Taylor	20640
20	Grant	12600

ORACLE

Copyright © 2009, 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 Whalen이라는 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 © 2009, 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 큰 따옴표가 필요합니다.

ORACLE

Copyright © 2009, 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	Whalen	(null)
2	Hartstein	(null)
3	Fay	(null)

...

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

	Name	Annual Salary
1	Whalen	52800
2	Hartstein	156000
3	Fay	72000

...

ORACLE

Copyright © 2009, 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. 앞의 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 © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

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
6 GietzAC_ACCOUNT

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연결 연산자

연결 연산자(||)를 사용하여 열을 다른 열, 산술식 또는 상수 값에 연결하여 문자 표현식을 생성할 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.

참고: 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 © 2009, Oracle. All rights reserved.

리터럴 문자열

리터럴은 `SELECT` 리스트에 포함된 문자, 숫자 또는 날짜입니다. 열 이름 또는 열 `alias`는 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
6	Gietz is a AC_ACCOUNT
7	Grant is a SA_REP
8	Hartstein is a MK_MAN
9	Higgins is a AC_MGR
10	Hunold is a IT_PROG

...

ORACLE

Copyright © 2009, 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 || ': 1 Month salary = ' || salary Monthly  
FROM   employees;
```

MONTHLY	
1	Whalen: 1 Month salary = 4400
2	Hartstein: 1 Month salary = 13000
3	Fay: 1 Month salary = 6000
4	Higgins: 1 Month salary = 12000
5	Gietz: 1 Month salary = 8300
6	King: 1 Month salary = 24000
7	Kochhar: 1 Month salary = 17000
8	De Haan: 1 Month salary = 17000

...

대체 인용(q) 연산자

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

```
SELECT department_name || q'[ Department'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 © 2009, 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.

1

```
SELECT department_id  
FROM employees;
```

	DEPARTMENT_ID
1	10
2	20
3	20
4	110
5	110
...	

2

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

	DEPARTMENT_ID
1	(null)
2	20
3	90
4	110
5	50
6	80
7	10
8	60

ORACLE

Copyright © 2009, 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
...		

참고: DISTINCT 키워드와 동의어인 UNIQUE 키워드를 지정해도 됩니다.

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**

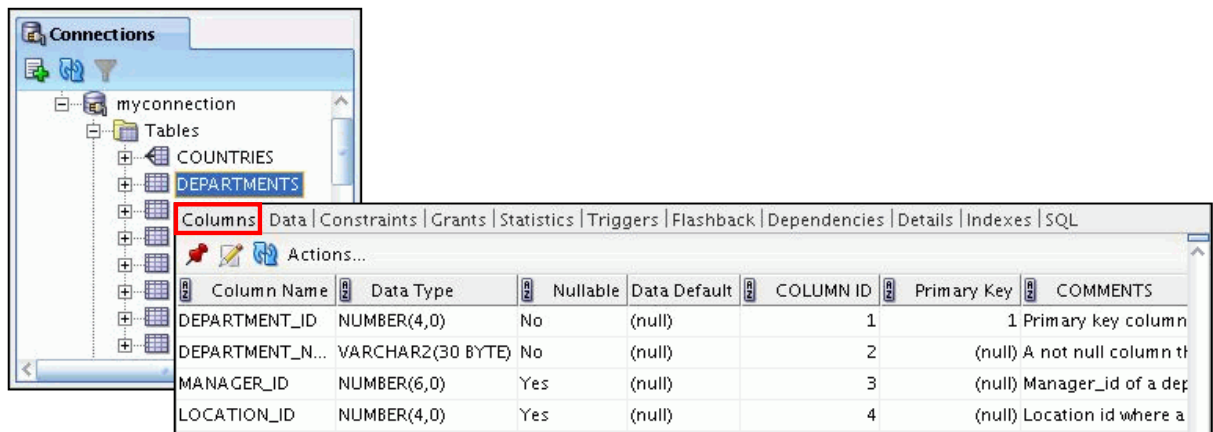
ORACLE

Copyright © 2009, 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
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 구조 표시

DESCRIBE 명령을 사용하여 테이블의 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 사용자가 액세스할 수 있는 기존의 테이블, 뷰 또는 동의어의 이름입니다.

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

참고: 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		

ORACLE

Copyright © 2009, 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 NULL*은 열에 데이터가 포함되어야 함을 나타냅니다. *Type*은 열의 데이터 유형을 표시합니다.

데이터 유형은 다음 표에 설명되어 있습니다.

데이터 유형	설명
NUMBER(<i>p</i> , <i>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

퀴즈

성공적으로 실행되는 SELECT 문을 식별하십시오.

1.

```
SELECT first_name, last_name, job_id, salary*12
AS Yearly Sal
FROM employees;
```
2.

```
SELECT first_name, last_name, job_id, salary*12
"yearly sal"
FROM employees;
```
3.

```
SELECT first_name, last_name, job_id, salary AS
"yearly sal"
FROM employees;
```
4.

```
SELECT first_name+last_name AS name, job_Id,
salary*12 yearly sal
FROM employees;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2, 3

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 © 2009, Oracle. All rights reserved.

요약

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

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

이 구문에서 다음이 적용됩니다.

SELECT	둘 이상의 열로 이루어진 리스트입니다.
*	모든 열을 선택합니다.
DISTINCT	중복을 방지합니다.
column/expression	이름 지정된 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 1: 개요

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

2

데이터 제한 및 정렬

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- query로 검색되는 행 제한
- query로 검색되는 행 정렬
- 앰퍼샌드 치환을 사용하여 런타임 시 출력 제한 및 정렬

ORACLE

Copyright © 2009, 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 © 2009, Oracle. All rights reserved.


선택을 사용하여 행 제한

EMPLOYEES

	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
4	205	Higgins	AC_MGR	110
5	206	Gietz	AC_ACCOUNT	110

...

"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 © 2009, 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 © 2009, 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.

이 구문에서 다음이 적용됩니다.

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 UNKNOWN의 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 © 2009, Oracle. All rights reserved.

WHERE 절 사용

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

참고: 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-RR입니다.

```
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 © 2009, Oracle. All rights reserved.

문자열 및 날짜

WHERE 절에서 문자열과 날짜는 작은 따옴표(' ')로 묶어야 합니다. 그러나 숫자 상수는 작은 따옴표로 묶을 필요가 없습니다.

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.

참고: RR 형식에 대한 자세한 내용과 기본 날짜 형식을 변경하는 방법은 "단일 행 함수를 사용하여 출력 커스터마이징" 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 © 2009, 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 절에서는 alias를 사용할 수 없습니다.

참고: != 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 © 2009, 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 © 2009, 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	201	Hartstein	13000	100
2	101	Kochhar	17000	100
3	102	De Haan	17000	100
4	124	Mourgos	5800	100
5	149	Zlotkey	10500	100
6	200	Whalen	4400	101
7	205	Higgins	12000	101
8	202	Fay	6000	201

ORACLE

Copyright © 2009, Oracle. All rights reserved.

IN 연산자를 사용하는 멤버 조건

지정된 값 집합에서 값을 테스트하려면 IN 연산자를 사용합니다. IN 연산자를 사용하여 정의한 조건을 *멤버 조건*이라고도 합니다.

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

참고: 값 집합은 (201,100,101) 등과 같이 임의의 순서로 지정할 수 있습니다.

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.

참고: IN 연산자는 Oracle 서버 내부에서 a=value1 또는 a=value2 또는 a=value3과 같이 OR 조건의 집합으로 평가됩니다. 따라서 IN 연산자를 사용해도 성능상의 이점은 없으며 논리적으로 간결하게 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 © 2009, 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. S는 대문자임에 유의하십시오. 소문자 s로 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
1	Kochhar
2	Lorentz
3	Mourgos

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

대체 문자 결합

% 및 _ 기호는 임의의 리터럴 문자 결합에서 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	149	Zlotkey	SA_MAN
2	174	Abel	SA_REP
3	176	Taylor	SA_REP
4	178	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 null을 테스트합니다.

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

	LAST_NAME	MANAGER_ID
1	King	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

NULL 조건 사용

NULL 조건은 IS NULL 조건과 IS NOT NULL 조건을 포함합니다.

IS NULL 조건은 null을 테스트합니다. null 값은 사용할 수 없거나, 할당되지 않았거나, 알 수 없거나, 적용할 수 없는 값을 의미합니다. 따라서 null은 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	Whalen	AD_ASST	(null)
2	Hartstein	MK_MAN	(null)
3	Fay	MK_REP	(null)
4	Higgins	AC_MGR	(null)
5	Gietz	AC_ACCOUNT	(null)

...

논리 연산자를 사용하여 조건 정의

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

ORACLE

Copyright © 2009, 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	201	Hartstein	MK_MAN	13000
2	149	Zlotkey	SA_MAN	10500

ORACLE

Copyright © 2009, 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	201	Hartstein	MK_MAN	13000
2	205	Higgins	AC_MGR	12000
3	100	King	AD_PRES	24000
4	101	Kochhar	AD_VP	17000
5	102	De Haan	AD_VP	17000
6	124	Mourgos	ST_MAN	5800
7	149	Zlotkey	SA_MAN	10500
8	174	Abel	SA_REP	11000

ORACLE

Copyright © 2009, 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 © 2009, 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

참고: 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 © 2009, 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 © 2009, 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 © 2009, 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 사장 이면서 \$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 사장 이거나 판매 담당자 이고 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 © 2009, 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 © 2009, Oracle. All rights reserved.

ORDER BY 절 사용

Query 결과에 반환되는 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]];
```

이 구문에서 다음이 적용됩니다.

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 사용하십시오.

참고: 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 ;
```



- 열 alias를 기준으로 정렬:

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



ORACLE

Copyright © 2009, 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, A부터 Z 순서로 표시됩니다.)
- Null 값은 오름차순에서는 맨 뒤에 표시되고 내림차순에서는 맨 앞에 표시됩니다.
- SELECT 리스트에 없는 열을 기준으로 정렬할 수도 있습니다.

예제:


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

참고: 여기서 내림차순으로 정렬하기 위해 사용된 DESC 키워드를, 테이블 구조를 설명하는 데 사용되는 DESC 키워드와 혼동하지 마십시오.

정렬


- 열의 숫자 위치를 사용하여 정렬

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



- 여러 열을 기준으로 정렬

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



ORACLE

Copyright © 2009, 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 슬라이드의 예제는 이 열을 기준으로 결과를 정렬합니다.
4. 둘 이상의 열을 기준으로 query 결과를 정렬할 수 있습니다. 정렬 한계는 제공된 테이블의 열 수입니다. ORDER BY 절에서 열을 지정하고 쉼표를 사용하여 열 이름을 구분합니다. 열 순서를 뒤바꾸려면 열 이름 뒤에 DESC를 지정합니다. 슬라이드에 표시된 query 예제의 결과는 department_id를 기준으로 오름차순으로 정렬되며 또한 salary를 기준으로 내림차순으로 정렬됩니다.

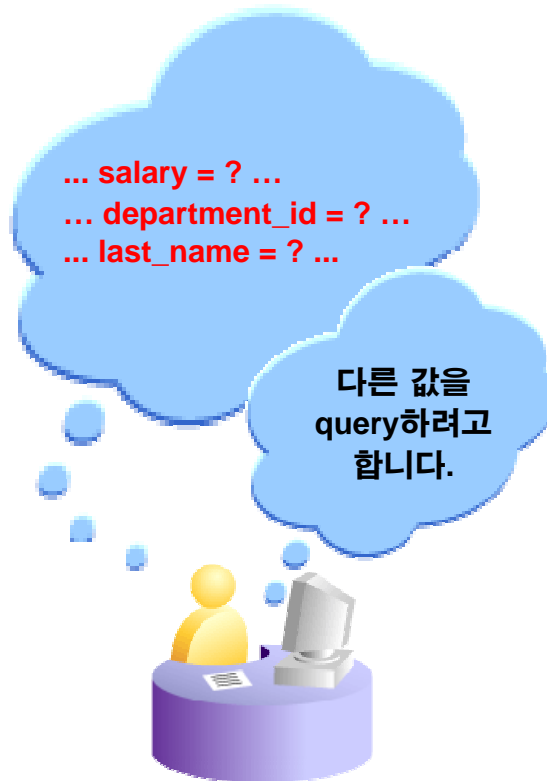
단원 내용

- 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 © 2009, Oracle. All rights reserved.

치환 변수



ORACLE

Copyright © 2009, 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 봅시다. 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 사용자가 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 앰퍼샌드(&) 및 이중 앰퍼샌드(&&) 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 © 2009, Oracle. All rights reserved.

치환 변수(계속)

단일 앰퍼샌드(&) 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 값 구하기
- SQL 문 사이에 값 전달

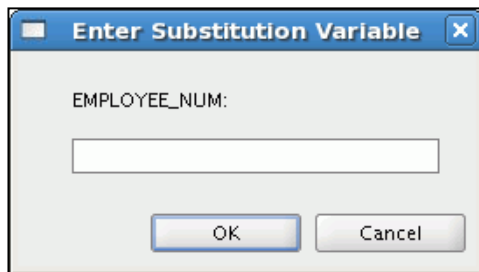
참고: SQL Developer 및 SQL* Plus는 모두 치환 변수 및 DEFINE/UNDEFINE 명령을 지원합니다.

그러나 SQL Developer 또는 SQL* Plus는 유저 입력에 대한 유효성 검사를 지원하지 않습니다(데이터 유형 제외). 유저에게 배포되는 스크립트에 사용할 경우 SQL 주입 공격으로 인해 치환 변수가 훼손될 수 있습니다.

단일 앰퍼샌드 치환 변수 사용

변수 앞에 앰퍼샌드(&)를 붙이면 사용자가 값을 입력하도록 할 수 있습니다.

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

단일 앰퍼샌드 치환 변수 사용

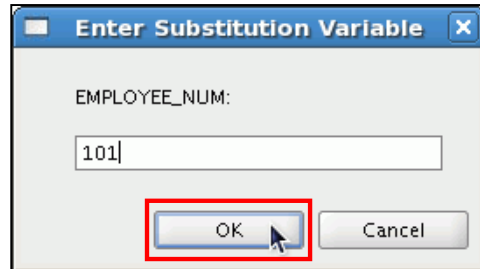
보고서 실행 시 유저는 종종 반환되는 데이터를 동적으로 제한해야 합니다. SQL*Plus 또는 SQL Developer는 유저 변수를 통해 이러한 유연성을 제공합니다. 앰퍼샌드(&)를 사용하여 SQL 문에서 각 변수를 식별합니다. 그러나 각 변수 값을 정의할 필요는 없습니다.

표기법	설명
<i>&user_variable</i>	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 앰퍼샌드를 사용하는 경우 변수가 존재하지 않으면 명령이 실행될 때마다 유저에게 값을 입력하도록 요청합니다.

단일 앰퍼샌드 치환 변수 사용



The dialog box titled "Enter Substitution Variable" has a close button (X) in the top right corner. It contains a label "EMPLOYEE_NUM:" followed by a text input field containing the value "101". Below the input field are two buttons: "OK" and "Cancel". The "OK" button is highlighted with a red rectangular border.

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	101	Kochhar	17000	90

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단일 앰퍼샌드 치환 변수 사용(계속)

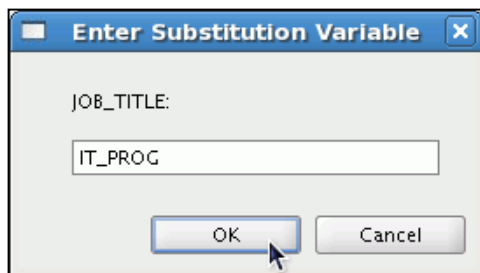
SQL Developer는 SQL 문에 앰퍼샌드가 포함된 것을 감지하면 SQL 문에 지정된 치환 변수의 값을 입력하도록 요청합니다.

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' ;
```

A dialog box titled "Enter Substitution Variable" with a close button (X). It contains a label "JOB_TITLE:" and a text input field containing "IT_PROG". Below the input field are "OK" and "Cancel" buttons. A mouse cursor is pointing at the "OK" button.

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

문자 및 날짜 값을 치환 변수로 지정

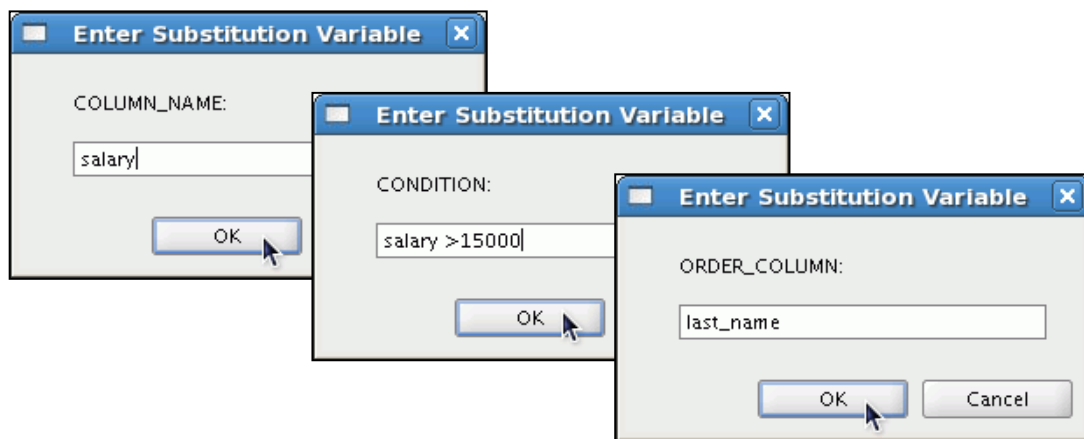
WHERE 절에서 날짜 및 문자 값은 작은 따옴표로 묶어야 합니다. 같은 규칙이 치환 변수에 적용됩니다.

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 © 2009, 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 사용자가 지정한 기타 열을 표시합니다. SELECT 문의 각 치환 변수에 대해 값을 입력하도록 요청하면 OK를 눌러 계속 진행합니다.

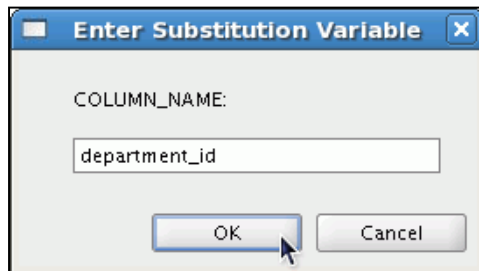
치환 변수의 값을 입력하지 않으면 앞의 명령문을 실행할 때 오류 메시지가 나타납니다.

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

이중 앰퍼샌드 치환 변수 사용

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

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



A dialog box titled "Enter Substitution Variable" with a close button (X). It contains a label "COLUMN_NAME:" and a text input field with the value "department_id". Below the input field are "OK" and "Cancel" buttons. A mouse cursor is pointing at the "OK" button.

	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

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

이중 앰퍼샌드 치환 변수 사용

유저가 매번 값을 입력할 필요 없이 변수 값을 재사용하려는 경우 이중 앰퍼샌드(&&) 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**

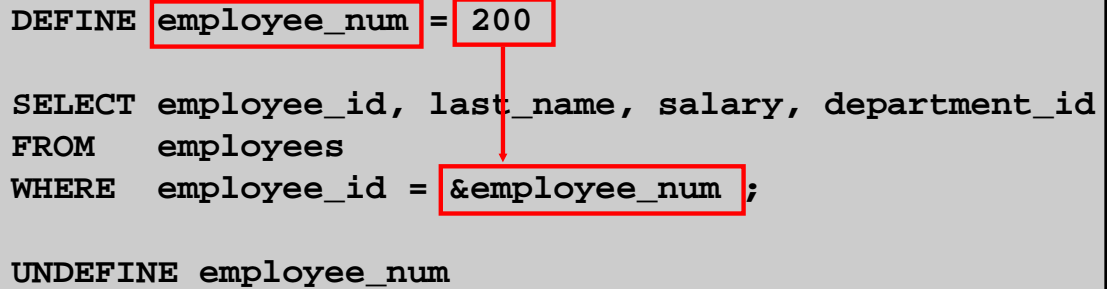
ORACLE

Copyright © 2009, 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
```

A red box highlights the variable 'employee_num' in the DEFINE statement and the '&employee_num' in the WHERE clause of the SQL query. A red arrow points from the variable in the DEFINE statement to the variable in the SQL query, illustrating the substitution process.

ORACLE

Copyright © 2009, 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 Developer가 치환 변수를 값으로 바꾸기 전후에 치환 변수의 표시를 토글합니다.

The screenshot shows the SQL Developer interface. On the left, the 'Enter Substitution Variable' dialog box is open, showing 'EMPLOYEE_NUM:' with the value '200' entered. On the right, the 'Script Output' tab is selected, displaying the SQL query and its results. The query is: `SELECT employee_id, last_name, salary FROM employees WHERE employee_id = &employee_num;`. The results show one row: `200 Whalen 4400`.

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

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400

1 rows selected

VERIFY 명령 사용

SQL 문의 변경 사항을 확인하려면 VERIFY 명령을 사용합니다. SET VERIFY를 ON으로 설정하면 SQL Developer가 치환 변수를 값으로 바꾼 후의 명령 텍스트가 표시됩니다. VERIFY 출력을 보려면 SQL Worksheet에서 Run Script(F5) 아이콘을 사용해야 합니다. SQL Developer는 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. VERIFY는 이러한 시스템 변수 중 하나입니다. 시스템 변수의 전체 리스트를 보려면 SQL*Plus 명령 프롬프트에서 SHOW ALL 명령을 실행하십시오.

퀴즈

다음 중 WHERE 절의 연산자로 사용할 수 있는 것은 무엇입니까?

1. >=
2. IS NULL
3. !=
4. IS LIKE
5. IN BETWEEN
6. <>

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 6

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]] ;
```

- 앰퍼샌드 치환을 사용하여 런타임 시 출력 제한 및 정렬

ORACLE

Copyright © 2009, 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 © 2009, Oracle. All rights reserved.

연습 2: 개요

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

단일 행 함수를 사용하여 출력 커스터마이징

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- **SQL에서 사용할 수 있는 다양한 유형의 함수 설명**
- **SELECT 문에 문자, 숫자 및 날짜 함수 사용**

ORACLE

Copyright © 2009, 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

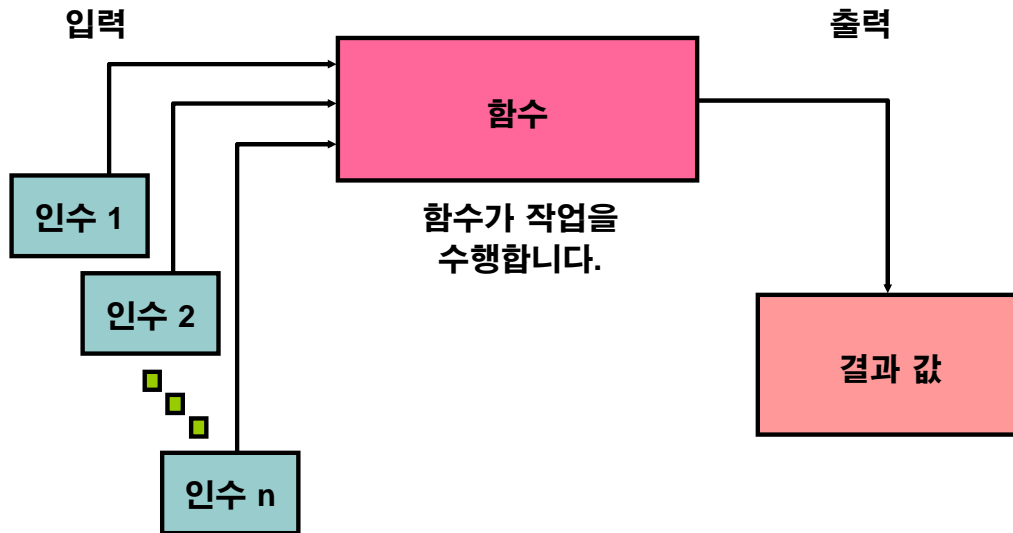
- **단일 행 SQL 함수**
 - 문자 함수
 - 숫자 함수
 - 날짜 작업
 - 날짜 함수

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

SQL 함수



ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL 함수

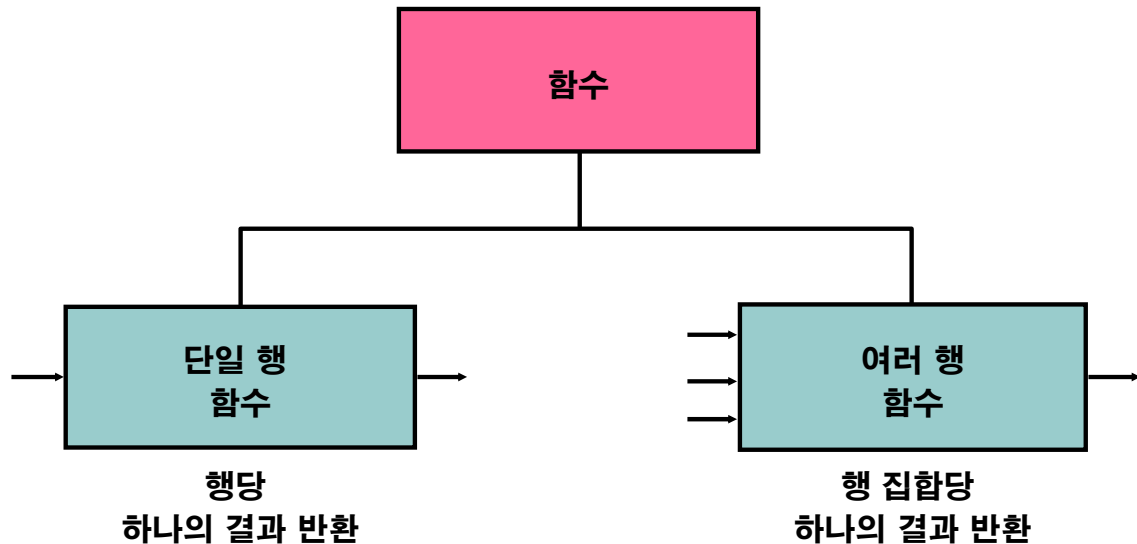
함수는 SQL의 매우 강력한 기능입니다. 다음 작업을 수행하는 데 함수를 사용할 수 있습니다.

- 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.

참고: 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 © 2009, Oracle. All rights reserved.

SQL 함수의 두 가지 유형

함수의 유형은 다음 두 가지입니다.

- 단일 행 함수
- 여러 행 함수

단일 행 함수

이 함수는 단일 행에 대해서만 실행되며 행당 하나의 결과를 반환합니다. 단일 행 함수에는 여러 가지 유형이 있습니다. 이 단원에서는 다음 함수에 대해 다룹니다.

- 문자
- 숫자
- 날짜
- 변환
- 일반

여러 행 함수

이 함수에서는 행 그룹당 하나의 결과를 산출하도록 행 그룹을 조작할 수 있습니다. 이러한 함수는 *그룹 함수*라고도 하며 "그룹 함수를 사용한 집계 데이터 보고" 단원에서 다룹니다.

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

단일 행 함수

단일 행 함수:

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

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

ORACLE

Copyright © 2009, 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
- 행당 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

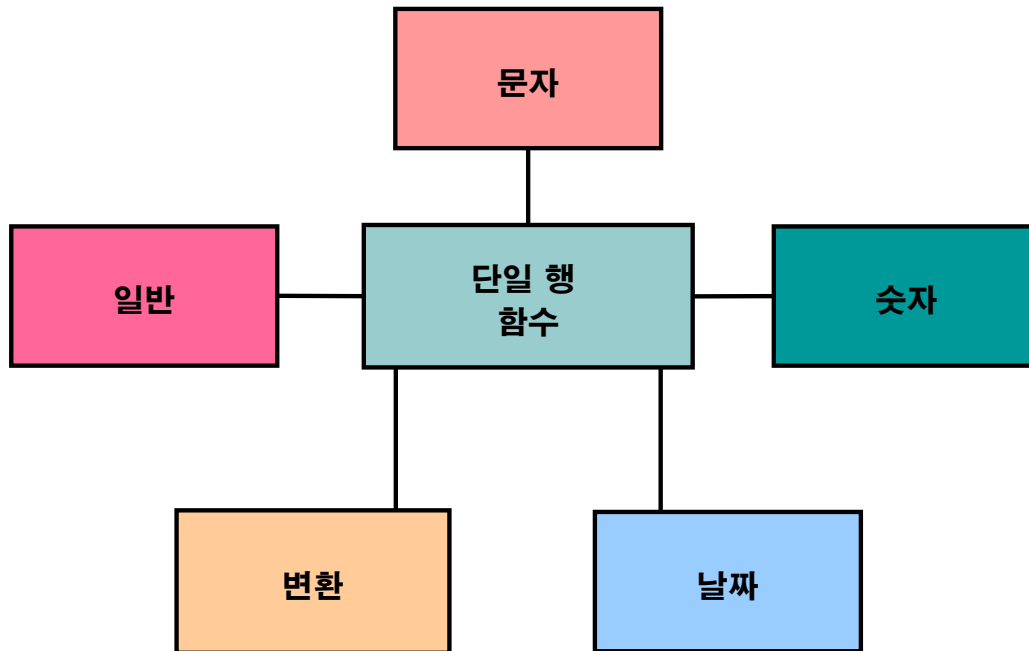
이 구문에서 다음이 적용됩니다.

`function_name` 함수의 이름입니다.

`arg1, arg2` 함수에 사용될 임의의 인수입니다. 열 이름이나 표현식으로 나타낼 수 있습니다.

.

단일 행 함수



ORACLE

Copyright © 2009, Oracle. All rights reserved.

단일 행 함수(계속)

이 단원에서는 다음의 단일 행 함수에 대해 다룹니다.

- **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

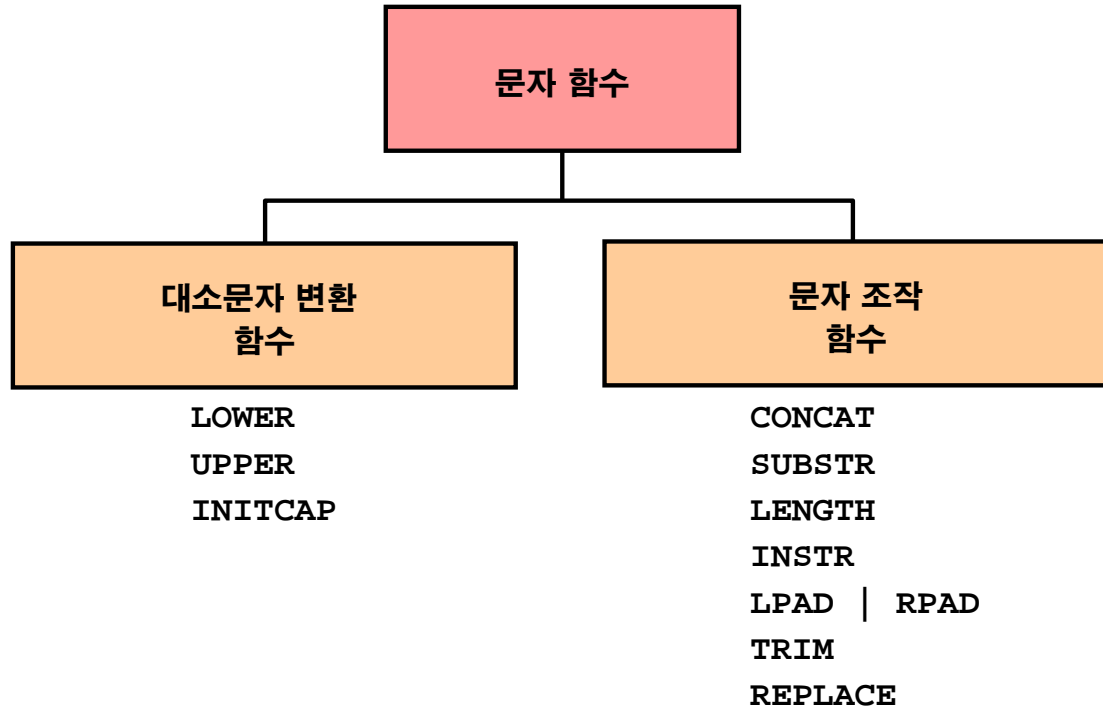
단원 내용

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

문자 함수



ORACLE

Copyright © 2009, 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</i> / <i>expression</i>)	영문자 값을 소문자로 변환합니다.
UPPER(<i>column</i> / <i>expression</i>)	영문자 값을 대문자로 변환합니다.
INITCAP(<i>column</i> / <i>expression</i>)	영문자 값의 첫번째 문자를 대문자로 변환하고 나머지 문자는 소문자로 둡니다.
CONCAT(<i>column1</i> / <i>expression1</i> , <i>column2</i> / <i>expression2</i>)	첫번째 문자 값을 두번째 문자 값과 연결합니다. 연결 연산자(∥)와 같은 기능입니다.
SUBSTR(<i>column</i> / <i>expression</i> , <i>m</i> [, <i>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.)

참고: 이 단원에서 설명하는 함수는 제공되는 함수 중 일부입니다.

문자 함수(계속)

함수	목적
LENGTH(<i>column</i> / <i>expression</i>)	표현식의 문자 수를 반환합니다.
INSTR(<i>column</i> / <i>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</i> / <i>expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column</i> / <i>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</i> FROM <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.

참고: 다음은 SQL:2003과 완전히 또는 부분적으로 호환되는 일부 함수입니다.

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 © 2009, 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;
```

AZ	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
4	The job id for ERNST is it_prog
5	The job id for FAY is mk_rep
6	The job id for GIETZ is ac_account

...

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 © 2009, 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
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

ORACLE

Copyright © 2009, 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: 이름 지정된 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 선행 문자나 후행 문자(또는 둘 다)를 자릅니다.(*trim_character* 또는 *trim_source*가 문자 리터럴인 경우 작은 따옴표로 묶어야 합니다.)

참고: UPPER 및 LOWER와 같은 함수를 앰퍼샌드 치환과 함께 사용할 수 있습니다. 예를 들어, UPPER('&job_title')을 사용하면 사용자가 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	202	PatFay	MK_REP	3	2
2	174	EllenAbel	SA_REP	4	0
3	176	JonathonTaylor	SA_REP	6	2
4	178	KimberelyGrant	SA_REP	5	3

1

2

3

ORACLE

Copyright © 2009, 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	102	LexDe Haan	7	5
2	200	JenniferWhalen	6	3
3	201	MichaelHartstein	9	2

단원 내용

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

숫자 함수

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

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

ORACLE

Copyright © 2009, 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</i> <i>expression</i> , <i>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</i> <i>expression</i> , <i>n</i>)	열, 표현식 또는 값을 소수점 <i>n</i> 자릿수로 자릅니다. <i>n</i> 이 생략된 경우 기본값은 0입니다
MOD(<i>m</i> , <i>n</i>)	<i>m</i> 을 <i>n</i> 으로 나눈 나머지를 반환합니다.

참고: 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

The diagram illustrates the syntax of the SQL ROUND function. It shows the following SQL statement:

```
SELECT ROUND(45.923,2), ROUND(45.923,0),  
       ROUND(45.923,-1)  
FROM DUAL;
```

Callouts identify the components:

- 1**: Points to the number `45.923`, which is the numeric value to be rounded.
- 2**: Points to the comma and the number `2` in `ROUND(45.923,2)`, representing the rounding precision.
- 3**: Points to the number `-1` in `ROUND(45.923,-1)`, representing the rounding scale.

	ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
1	45.92	46	50

DUAL은 함수 및 계산의 결과를 볼 때 사용할 수 있는 공용(public) 테이블입니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ROUND 함수 사용

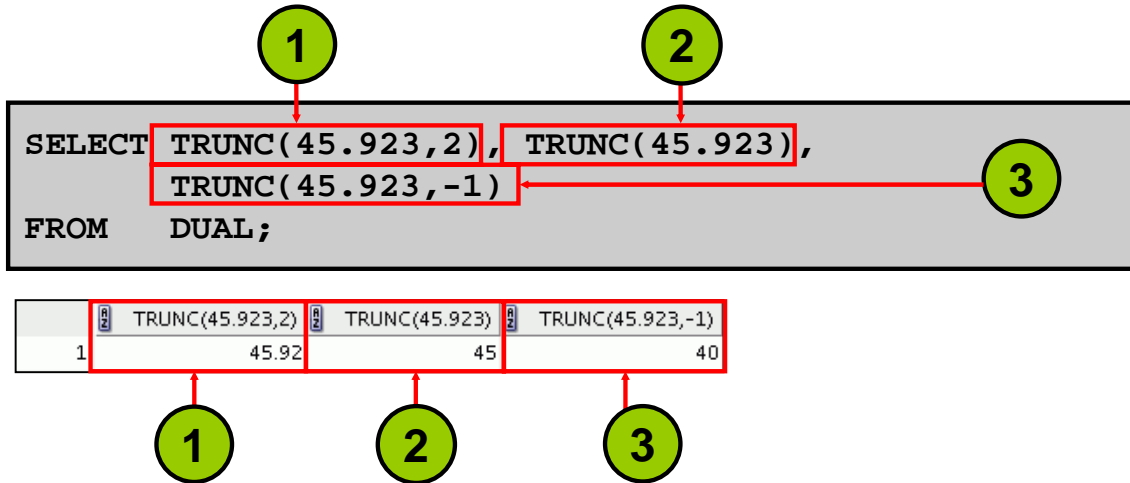
ROUND 함수는 열, 표현식 또는 값을 소수점 n 자릿수로 반올림합니다. 두번째 인수가 0이거나 누락되었으면 값을 소수점 0 자릿수로 반올림됩니다. 두번째 인수가 2이면 값을 소수점 2 자릿수로 반올림됩니다. 반대로, 두번째 인수가 -2이면 소수점 왼쪽의 둘째 자릿수로 반올림됩니다(가장 가까운 100 단위 값으로 반올림됨).

ROUND 함수는 날짜 함수와도 함께 사용할 수 있습니다. 이 단원의 후반부에서 예제를 볼 수 있습니다.

DUAL 테이블

DUAL 테이블은 유저 SYS가 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 © 2009, Oracle. All rights reserved.

TRUNC 함수 사용

TRUNC 함수는 열, 표현식 또는 값을 소수점 n 자릿수로 자릅니다.

TRUNC 함수는 ROUND 함수에 사용된 것과 유사한 인수를 사용합니다. 두번째 인수가 0이거나 누락되었으면 값은 소수점 0 자릿수로 잘립니다. 두번째 인수가 2이면 값은 소수점 2 자릿수로 잘립니다. 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 Representative인 모든 사원에 대해 급여를 5,000으로 나눈 나머지를 계산합니다.

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

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

MOD 함수 사용

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

참고: MOD 함수는 종종 값이 홀수인지 짝수인지 판별하는 데 사용됩니다. MOD 함수는 Oracle 해시 함수이기도 합니다.

단원 내용

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

ORACLE

Copyright © 2009, 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-RR입니다.
 - 연도의 마지막 두 자릿수만 지정하면 21세기 날짜를 20세기 날짜로 저장할 수 있습니다.
 - 같은 방식으로 20세기 날짜를 21세기에 저장할 수 있습니다.

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

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

ORACLE

Copyright © 2009, 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 © 2009, 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
2048	27-OCT-52	1952	2052
2051	27-OCT-47	2147	2047

위 테이블의 마지막 두 행에 표시된 값을 확인하십시오. 세기 중반에 이르면 RR 동작이 예상과 다를 것입니다.

RR 날짜 형식(계속)

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 날짜 형식은 연도 정보에 2바이트를 사용하며 그중 1바이트는 세기용, 1바이트는 연도용입니다. 세기 값은 지정되거나 표시되는 여부와 상관없이 항상 포함됩니다. 이 경우 RR은 INSERT의 세기에 대한 기본값을 결정합니다.

SYSDATE 함수 사용

SYSDATE 함수는 다음 값을 반환합니다.

- 날짜
- 시간

```
SELECT sysdate
FROM dual;
```

	SYSDATE
1	10-JUN-09

ORACLE

Copyright © 2009, 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 공용(public) 테이블에서 SYSDATE를 선택하는 것이 일반적입니다.

참고: 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 일수를 알아내기 위해 빼기 연산을 합니다.
- 시간 수를 24로 나눠 날짜에 시간을 더합니다.

ORACLE

Copyright © 2009, 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
1	King	1147.102432208994708994708994708995
2	Kochhar	1028.959575066137566137566137566138
3	De Haan	856.102432208994708994708994708995

ORACLE

Copyright © 2009, Oracle. All rights reserved.

날짜에 산술 연산자 사용

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

참고: 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 © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

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 © 2009, Oracle. All rights reserved.

날짜 조작 함수

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

- MONTHS_BETWEEN(*date1*, *date2*): *date1*과 *date2* 사이의 월수를 찾습니다. 결과는 양수나 음수가 될 수 있습니다. *date1*이 *date2*보다 늦은 날짜인 경우 결과는 양수이고 *date1*이 *date2*보다 앞선 날짜인 경우 결과는 음수입니다. 결과에서 정수가 아닌 부분은 월의 일부분을 나타냅니다.
- ADD_MONTHS(*date*, *n*): *date*에 월수 *n*을 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*): *date*에 해당하는 날짜가 있는 월의 말일 날짜를 찾습니다.

날짜 조작 함수

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 "변환 함수 및 조건부 표현식 사용" 단원에서 자세히 다룹니다.

날짜 함수 사용

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

ORACLE

Copyright © 2009, 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 150개월 미만인 모든 사원에 대해 사원 번호, 채용 날짜, 근속 월수, 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) < 150;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DA...	LAST_DAY...
1	202	17-AUG-97	141.79757989...	17-FEB-98	22-AUG-97	31-AUG-97
2	107	07-FEB-99	124.12016054...	07-AUG-99	12-FEB-99	28-FEB-99
3	124	16-NOV-99	114.82983796...	16-MAY-00	19-NOV-99	30-NOV-99
4	142	29-JAN-97	148.41048312...	29-JUL-97	31-JAN-97	31-JAN-97
5	143	15-MAR-98	134.86209602...	15-SEP-98	20-MAR-98	31-MAR-98
6	144	09-JUL-98	131.05564441...	09-JAN-99	10-JUL-98	31-JUL-98
7	149	29-JAN-00	112.41048312...	29-JUL-00	04-FEB-00	31-JAN-00
8	176	24-MAR-98	134.57177344...	24-SEP-98	27-MAR-98	31-MAR-98
9	178	24-MAY-99	120.57177344...	24-NOV-99	28-MAY-99	31-MAY-99

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 © 2009, 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	202	17-AUG-97	01-SEP-97	01-AUG-97
2	142	29-JAN-97	01-FEB-97	01-JAN-97

퀴즈

단일 행 함수에 대한 다음 설명 중 옳은 것은?

1. 데이터 항목을 조작합니다.
2. 인수를 받아들이고 인수당 하나의 값을 반환합니다.
3. 반환되는 각 행에서 실행됩니다.
4. 행 집합당 하나의 결과를 반환합니다.
5. 데이터 유형을 수정할 수 없습니다.
6. 중첩될 수 있습니다.
7. 열이나 표현식을 인수로 받아들일 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3, 6, 7

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

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

ORACLE

Copyright © 2009, 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 단일 행 공용(public) 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 연수 및 월수 계산

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 3: 개요

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

4

변환 함수 및 조건부 표현식 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

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

ORACLE

Copyright © 2009, 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.

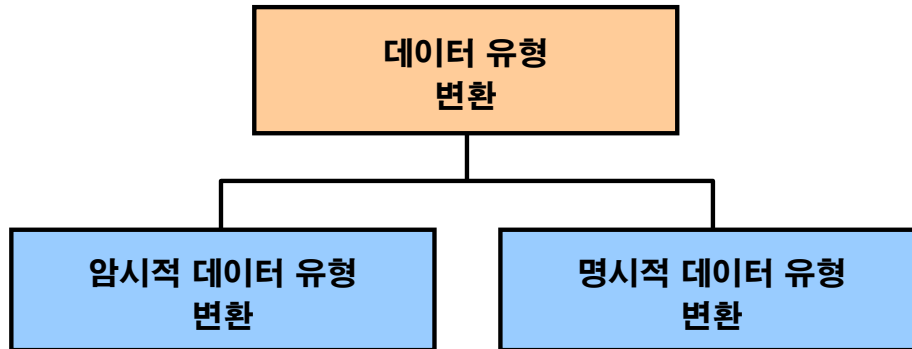
단원 내용

- **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 © 2009, Oracle. All rights reserved.

변환 함수



ORACLE

Copyright © 2009, Oracle. All rights reserved.

변환 함수

오라클 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 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.

참고: 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 © 2009, 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

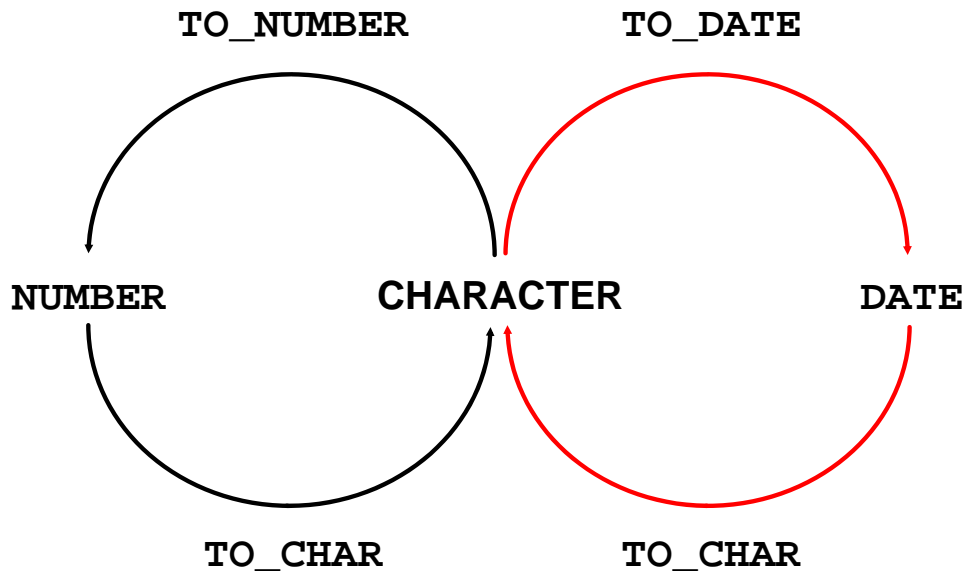
Copyright © 2009, Oracle. All rights reserved.

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 = 2`에서 숫자 2이 문자열 "2"로 암시적으로 변환됩니다.

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

명시적 데이터 유형 변환



ORACLE

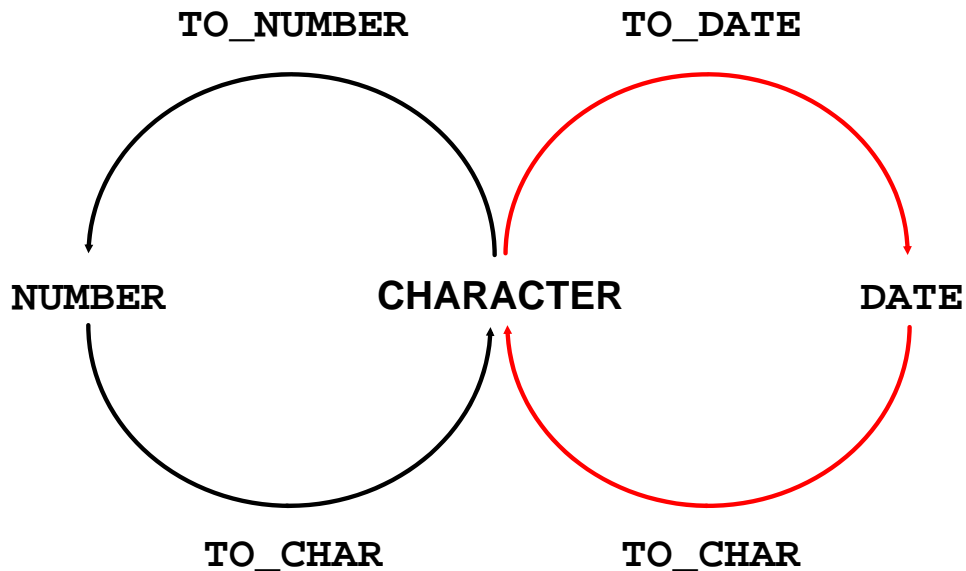
Copyright © 2009, Oracle. All rights reserved.

명시적 데이터 유형 변환

SQL은 세 가지 데이터 유형 변환 함수를 제공합니다.

함수	목적
<code>TO_CHAR(<i>number</i> <i>date</i>, [<i>fmt</i>], [<i>nlsparms</i>])</code>	<p>형식 모델 <i>fmt</i>를 사용하여 숫자 또는 날짜 값을 VARCHAR2 문자열로 변환합니다.</p> <p>숫자 변환: <i>nlsparms</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>nlsparms</i> 또는 기타 파라미터가 생략된 경우 이 함수는 세션에 대해 기본 파라미터 값을 사용합니다.</p>

명시적 데이터 유형 변환



ORACLE

Copyright © 2009, Oracle. All rights reserved.

명시적 데이터 유형 변환(계속)

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

명시적 데이터 유형 변환(계속)

참고: 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 © 2009, 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.
- 쉼표를 사용하여 날짜 값과 구분됩니다.

ORACLE

Copyright © 2009, 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 *datetime*의 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 사용자가 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.
- 자동으로 출력의 일 및 월 이름은 공백으로 채워집니다.
- 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	3문자로 된 요일 약어
DAY	요일의 전체 이름
DD	숫자 형식의 월간 일

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

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.
YYY 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.

ddsptth	fourteenth
---------	------------

ORACLE

Copyright © 2009, 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)

날짜 형식 모델의 요소 (계속)

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	Whalen	17 September 1987
2	Hartstein	17 February 1996
3	Fay	17 August 1997
4	Higgins	7 June 1994
5	Gietz	7 June 1994
6	King	17 June 1987
7	Kochhar	21 September 1989
8	De Haan	13 January 1993
9	Hunold	3 January 1990
10	Ernst	21 May 1991

...

ORACLE

Copyright © 2009, 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	Whalen	Seventeenth of September 1987 12:00:00 AM
2	Hartstein	Seventeenth of February 1996 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 쉼표 출력

ORACLE

Copyright © 2009, 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 괄호로 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 © 2009, 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.

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

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

ORACLE

Copyright © 2009, 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 선행 0 을 생략할 수 있습니다.

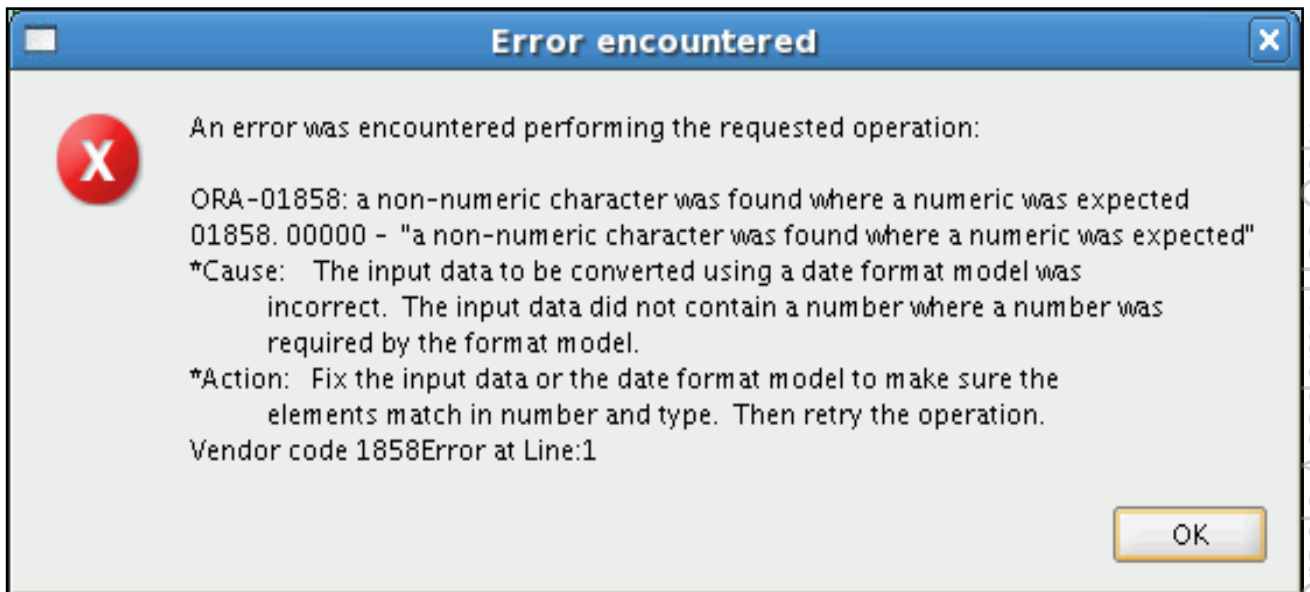
TO_NUMBER 및 TO_DATE 함수 사용(계속)

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');
```

다음과 유사한 결과 오류 출력이 표시됩니다.



출력을 보려면 'May'와 '24' 사이의 공백을 삭제하여 query를 수정합니다.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May24, 1999', 'fxMonth DD, YYYY');
```

	LAST_NAME	HIRE_DATE
1	Grant	24-MAY-99

RR 날짜 형식으로 TO_CHAR 및 TO_DATE 함수 사용

1990년 이전에 채용된 사원을 찾으려면 RR 날짜 형식을 사용합니다. 그러면 명령이 1999년에 실행되든 현재 실행되든 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')
1	Whalen	17-Sep-1987
2	King	17-Jun-1987
3	Kochhar	21-Sep-1989

ORACLE

Copyright © 2009, 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. 또한 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

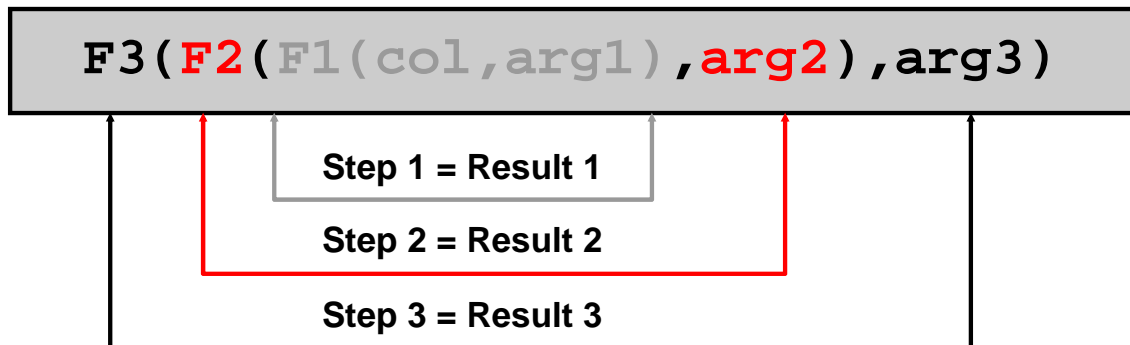
- 암시적 및 명시적 데이터 유형 변환
- TO_CHAR, TO_DATE, TO_NUMBER 함수
- 함수 중첩
- 일반 함수:
 - NVL
 - NVL2
 - NULLIF
 - COALESCE
- 조건부 표현식:
 - CASE
 - DECODE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 중첩

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



ORACLE

Copyright © 2009, 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: 예제 1

```
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 © 2009, 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 나타냅니다. 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;
```

함수 중첩: 예제 2

```
SELECT      TO_CHAR(ROUND((salary/7), 2), '99G999D99',  
              'NLS_NUMERIC_CHARACTERS = ','.'')  
              "Formatted Salary"  
FROM employees;
```

	Formatted Salary
1	628,57
2	1.857,14
3	857,14
4	1.714,29
5	1.185,71
6	3.428,57

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

함수 중첩(계속)

슬라이드의 예제는 사원의 급여를 7로 나눈 후 소수점 이하 둘째 자리로 반올림합니다. 그런 다음 결과는 형식이 지정되어 덴마크 표기법으로 급여를 표시합니다. 즉, 소수점에 쉼표가 사용되고 천 단위 구분 기호로 점이 사용됩니다.

먼저 안쪽 ROUND 함수가 실행되어 급여를 7로 나눈 값을 소수점 이하 둘째 자리로 반올림합니다. 그런 다음 TO_CHAR 함수를 사용하여 ROUND 함수 결과의 형식이 지정됩니다.

참고: TO_CHAR 함수 파라미터에 지정된 D 및 G는 숫자 형식 요소입니다. D는 지정된 위치에 소수점 문자를 반환합니다. G는 그룹 구분자로 사용됩니다.

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 © 2009, 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)

ORACLE

Copyright © 2009, 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.

참고: 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 © 2009, Oracle. All rights reserved.

NVL 함수

null 값을 실제 값으로 변환하려면 NVL 함수를 사용합니다.

구문

`NVL (expr1, expr2)`

이 구문에서 다음이 적용됩니다.

- `expr1`은 null을 포함할 수 있는 소스 값 또는 표현식입니다.
- `expr2`는 null을 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 or 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	Whalen	4400	0	52800
2	Hartstein	13000	0	156000
3	Fay	6000	0	72000
4	Higgins	12000	0	144000
5	Gietz	8300	0	99600
6	King	24000	0	288000
7	Kochhar	17000	0	204000
8	De Haan	17000	0	204000
9	Hunold	9000	0	108000
10	Ernst	6000	0	72000

...

1

2

ORACLE

Copyright © 2009, 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	Whalen	4400	(null)	(null)

...

16	Vargas	2500	(null)	(null)
17	Zlotkey	10500	0.2	151200
18	Abel	11000	0.3	171600
19	Taylor	8600	0.2	123840
20	Grant	7000	0.15	96600

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
      NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
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 © 2009, 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)

이 구문에서 다음이 적용됩니다.

- expr1은 null을 포함할 수 있는 소스 값 또는 표현식입니다.
- expr2는 expr1이 null이 아닌 경우에 반환되는 값입니다.
- expr3은 expr1이 null인 경우에 반환되는 값입니다.

슬라이드의 예제에서는 COMMISSION_PCT 열이 검사됩니다. 값이 발견되면 텍스트 리터럴 값 SAL+COMM이 반환됩니다. COMMISSION_PCT 열에 null 값이 있는 경우 텍스트 리터럴 값 SAL이 반환됩니다.

인수: expr1은 임의의 데이터 유형을 가질 수 있습니다. 인수 expr2 및 expr3은 LONG을 제외한 어떠한 데이터 유형도 될 수 있습니다.

NULLIF 함수 사용

1

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name, LENGTH(last_name) "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM employees;
```

2

3

	FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
1	Ellen	5	Abel	4	5
2	Curtis	6	Davies	6	(null)
3	Lex	3	De Haan	7	3
4	Bruce	5	Ernst	5	(null)
5	Pat	3	Fay	3	(null)
6	William	7	Gietz	5	7
7	Kimberely	9	Grant	5	9
8	Michael	7	Hartstein	9	7
9	Shelley	7	Higgins	7	(null)
...					

1

2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

NULLIF 함수 사용

NULLIF 함수는 두 표현식을 비교합니다.

구문

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

이 구문에서 다음이 적용됩니다.

- NULLIF는 *expr1*과 *expr2*를 비교합니다. 두 표현식이 같으면 이 함수는 null을 반환합니다. 두 표현식이 다르면 이 함수는 *expr1*을 반환합니다. 그러나 *expr1*에 대해 리터럴 NULL을 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.

참고: 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 © 2009, Oracle. All rights reserved.

COALESCE 함수 사용

COALESCE 함수는 리스트에서 null이 아닌 첫번째 표현식을 반환합니다.

구문

COALESCE (*expr1*, *expr2*, ... *exprn*)

이 구문에서 다음이 적용됩니다.

- *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;
```

	A 2	LAST_NAME	A 2	EMPLOYEE_ID	A 2	COALESCE(TO_CHAR(COMMISSION_PCT), TO_CHAR(MANAGER_ID), 'No commission and no manager')
1		Whalen		200	101	
2		Hartstein		201	100	
3		Fay		202	201	
4		Higgins		205	101	
5		Gietz		206	205	
6		King		100		No commission and no manager

...

17		Zlotkey		149	.2	
18		Abel		174	.3	
19		Taylor		176	.2	
20		Grant		178	.15	

ORACLE

Copyright © 2009, 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 적용되므로 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;
```

참고: 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.

	A Z	LAST_NAME	A Z	SALARY	A Z	COMMISSION_PCT	A Z	New Salary
1		Whalen		4400		(null)		6400
2		Hartstein		13000		(null)		15000
3		Fay		6000		(null)		8000
4		Higgins		12000		(null)		14000
5		Gietz		8300		(null)		10300
6		King		24000		(null)		26000

■ ■ ■

17		Zlotkey		10500		0.2		12600
18		Abel		11000		0.3		14300
19		Taylor		8600		0.2		10320
20		Grant		7000		0.15		8050

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 © 2009, 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 © 2009, 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.

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

CASE 식

IF-THEN-ELSE 문 작업을 수행하여 조건부 조회를 편리하게 수행하도록 합니다.

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

ORACLE

Copyright © 2009, 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 *expr*이 *comparison_expr*과 동일한 첫번째 WHEN ... THEN 쌍을 검색하여 *return_expr*을 반환합니다. 이 조건을 충족하는 WHEN ... THEN 쌍이 없고 ELSE 절이 존재하면 Oracle 서버는 *else_expr*을 반환합니다. 그렇지 않은 경우 Oracle 서버는 null을 반환합니다. 일부 *return_expr* 및 *else_expr*에 대해서는 리터럴 NULL을 지정할 수 없습니다.

expr 및 *comparison_expr* 식은 데이터 유형이 동일해야 합니다. 이러한 데이터 유형으로는 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2가 가능합니다. 모든 반환 값(*return_expr*)은 데이터 유형이 동일해야 합니다.

CASE 식 사용

IF-THEN-ELSE 문 작업을 수행하여 조건부 조회를 편리하게 수행하도록 합니다.

```
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
1	Whalen	AD_ASST	4400	4400
...				
9	Hunold	IT_PROG	9000	9900
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
14	Davies	ST_CLERK	3100	3565
...				
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

CASE 식 사용

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

동일한 명령문을 DECODE 함수를 사용하여 작성할 수 있습니다.

다음 코드는 검색된 CASE 식의 예제입니다. 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 © 2009, Oracle. All rights reserved.

DECODE 함수

DECODE 함수는 다양한 언어에서 사용되는 IF-THEN-ELSE 논리와 비슷한 방식으로 표현식을 디코딩합니다. DECODE 함수는 *expression*을 각 *search* 값에 비교한 후에 디코딩합니다. 표현식이 *search*와 동일하면 *result*가 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
...				
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
...				
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

ORACLE

Copyright © 2009, Oracle. All rights reserved.

DECODE 함수 사용

슬라이드의 SQL 문에서는 JOB_ID 값이 테스트됩니다. JOB_ID가 IT_PROG이면 급여 증가율은 10%이고, JOB_ID가 ST_CLERK이면 급여 증가율은 15%이고, JOB_ID가 SA_REP이면 급여 증가율은 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 © 2009, 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.

월 급여 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

퀴즈

TO_NUMBER 함수는 문자열 또는 데이터 값을, 선택적 형식 모델을 통해 지정된 형식의 숫자로 변환합니다.

1. 맞습니다.
2. 틀립니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2

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

ORACLE

Copyright © 2009, 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 COALESCE를 포함하여 null과 관련된 여러 함수가 있습니다.
- CASE 식이나 DECODE 함수를 사용하여 SQL 문 내에서 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 CASE와 같은 조건부 표현식을 사용하는 query 작성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 4: 개요

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

5

그룹 함수를 사용하여 집계된 데이터 보고

ORACLE

Copyright © 2009, 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

ORACLE

Copyright © 2009, 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 © 2009, Oracle. All rights reserved.

그룹 함수란?

그룹 함수는 행 집합 연산을 수행하여 그룹별로 하나의 결과를 산출합니다.

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
...		
18	80	11000
19	80	8600
20	(null)	7000

EMPLOYEES
테이블의 최대 급여

	MAX(SALARY)
1	24000

ORACLE

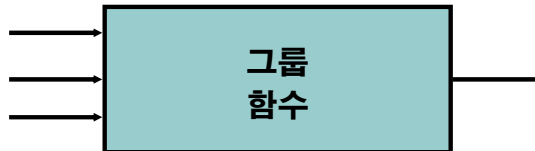
Copyright © 2009, Oracle. All rights reserved.

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 © 2009, Oracle. All rights reserved.

그룹 함수 유형

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

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

그룹 함수: 구문

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

ORACLE

Copyright © 2009, 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, COALESCE, CASE 또는 DECODE 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 © 2009, 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 © 2009, 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)
1	Abel	Zlotkey

참고: 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 © 2009, 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 © 2009, 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 © 2009, 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:

1. 평균 계산에는 테이블에서 COMMISSION_PCT 열에 유효한 값이 저장된 행 만 사용됩니다. 평균은 모든 사원에게 지급된 총 커미션을, 커미션을 받는 사원 수(4)로 나누어 계산합니다.
2. 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 © 2009, Oracle. All rights reserved.

데이터 그룹 생성

EMPLOYEES

R2	DEPARTMENT_ID	R2	SALARY
1	10		4400
2	20		13000
3	20		6000
4	50		2500
5	50		2600
6	50		3100
7	50		3500
8	50		5800
9	60		9000
10	60		6000
11	60		4200
12	80		11000
13	80		8600
...			
18	110		8300
19	110		12000
20	(null)		7000

4400

9500

3500

6400

10033

EMPLOYEES 테이블의 각 부서에 대한 평균 급여

R2	DEPARTMENT_ID	R2	AVG(SALARY)
1	(null)		7000
2	20		9500
3	90		19333.333333333333...
4	110		10150
5	50		3500
6	80		10033.333333333333...
7	10		4400
8	60		6400

ORACLE

Copyright © 2009, 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

GROUP BY 절을 사용하여 테이블의 행을 더 작은 그룹으로 나눌 수 있습니다.

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

ORACLE

Copyright © 2009, 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.

이 구문에서 다음이 적용됩니다.

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 *alias*를 사용할 수 없습니다.

GROUP BY 절 사용

그룹 함수에 속하지 않는 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	20	9500
3	90	19333.333333333333...
4	110	10150
5	50	3500
6	80	10033.333333333333...
7	10	4400
8	60	6400

ORACLE

Copyright © 2009, 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.

참고: query 결과를 오름차순 또는 내림차순으로 정렬하려면 query에 ORDER BY 절을 포함합니다.

GROUP BY 절 사용

GROUP BY 열은 SELECT 리스트에 없어도 됩니다.

```
SELECT  AVG(salary)
FROM    employees
GROUP BY department_id ;
```

	AVG(SALARY)
1	7000
2	9500
3	19333.33333333333333333333...
4	10150
5	3500
6	10033.33333333333333333333...
7	4400
8	6400

ORACLE

Copyright © 2009, 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) ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	50	3500
2	10	4400
3	60	6400

...

7	110	10150
8	90	19333.33333333333333333333...

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_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
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	110	AC_ACCOUNT	8300
2	110	AC_MGR	12000
3	10	AD_ASST	4400
4	90	AD PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

ORACLE

Copyright © 2009, 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.

다음 SELECT 문은 슬라이드에 표시된 결과를 반환합니다.

```
SELECT  department_id, job_id, sum(salary)
FROM    employees
GROUP BY department_id, job_id
ORDER BY job_id;
```

여러 열에서 GROUP BY 절 사용

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

여러 열에서 Group By 절 사용

여러 개의 GROUP BY 열을 나열하여 그룹과 하위 그룹에 대한 요약 결과를 반환할 수 있습니다. GROUP BY 절은 행을 그룹화하지만 결과 집합의 순서를 지정하지는 않습니다. 그룹화 순서를 지정하려면 ORDER BY 절을 사용합니다.

슬라이드의 예제에서 GROUP BY 절을 포함한 SELECT 문은 다음과 같이 평가됩니다.

- SELECT 절은 다음과 같이 검색할 열을 지정합니다.
 - EMPLOYEES 테이블의 부서 ID
 - EMPLOYEES 테이블의 직무 ID
 - GROUP BY 절에서 지정한 그룹의 모든 급여 합계
- FROM 절은 데이터베이스가 액세스해야 하는 테이블, 즉 EMPLOYEES 테이블을 지정합니다.
- WHERE 절은 결과 집합을 부서 ID가 40보다 큰 행으로만 한정합니다.
- GROUP BY 절은 결과 행을 그룹화하는 방법을 지정합니다.
 - 먼저 행이 부서 ID별로 그룹화됩니다.
 - 이어서 행이 부서 ID 그룹에서 직무 ID별로 그룹화됩니다.
- ORDER BY 절은 결과를 부서 ID별로 정렬합니다.

참고: SUM 함수는 결과 집합에서 각 부서 ID 그룹의 모든 직무 ID에 대한 salary 열에 적용됩니다. 또한 SA_REP 행은 반환되지 않습니다. 이 행의 부서 ID는 NULL이며 따라서 WHERE 조건을 만족하지 않습니다.

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_id에 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_id를 추가하거나
SELECT 리스트에서 job_id 열을
제거합니다.

ORACLE

Copyright © 2009, 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.

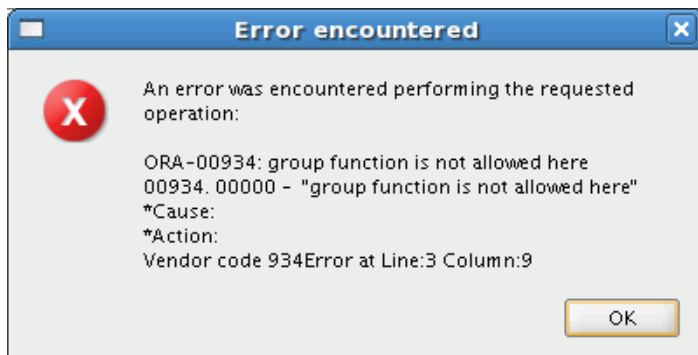
슬라이드의 두번째 예제에서 job_id는 GROUP BY 절에 있지도 않고 그룹 함수에 사용되지도 않으므로 "not a GROUP BY expression" 오류가 발생합니다. GROUP BY 절에 job_id를 추가하여 두번째 슬라이드 예제에서 오류를 바로잡을 수 있습니다.

```
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 © 2009, 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 바로잡을 수 있습니다.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 8000;
```

	DEPARTMENT_ID	AVG(SALARY)
1	20	9500
2	90	19333.3333333333...
3	110	10150
4	80	10033.3333333333...

그룹 결과 제한

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	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	13000
2	90	24000
3	110	12000
4	80	11000

ORACLE

Copyright © 2009, 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];
```

ORACLE

Copyright © 2009, 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.

Oracle 서버는 사용자가 HAVING 절을 사용할 때 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 수 있지만 GROUP BY 절을 먼저 두는 것이 더 논리적이므로 이렇게 하는 것이 좋습니다. HAVING 절이 SELECT 리스트의 그룹에 적용되기 전에 그룹이 형성되고 그룹 함수가 계산됩니다.

참고: WHERE 절은 행을 제한하는 반면 HAVING 절은 그룹을 제한합니다.

HAVING 절 사용

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY  department_id
HAVING    MAX(salary)>10000 ;
```

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	90	24000
3	110	12000
4	80	11000

ORACLE

Copyright © 2009, 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	20	9500
2	90	19333.333333333...
3	110	10150
4	80	10033.333333333...

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 © 2009, 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 © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

최고 평균 급여를 표시합니다.

```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

[illegible]

Copyright © 2009, 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.

퀴즈

그룹 함수 및 GROUP BY 절에 대한 설명을 고르십시오.

1. GROUP BY 절에서 열 alias를 사용할 수 없습니다.
2. GROUP BY 절은 반드시 SELECT 절에 있어야 합니다.
3. WHERE 절을 사용하면 행을 그룹으로 나누기 전에 행을 제외시킬 수 있습니다.
4. GROUP BY 절은 행을 그룹화하고 결과 집합의 순서를 지정합니다.
5. SELECT 절에 그룹 함수를 포함할 경우 개별 결과를 선택할 수 없습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3

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 © 2009, Oracle. All rights reserved.

요약

SQL에서는 AVG, COUNT, MAX, MIN, SUM, STDDEV 및 VARIANCE 등과 같은 여러 그룹 함수를 사용할 수 있습니다.

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.

1. 명령문에 WHERE 절이 포함된 경우 후보 행을 설정합니다.
2. GROUP BY 절에 지정된 그룹을 식별합니다.
3. HAVING 절은 HAVING 절의 그룹 조건을 충족하지 않는 결과 그룹을 추가로 제한합니다.

참고: 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 5: 개요

이 연습에서는 그룹 함수를 사용하고 데이터 그룹을 선택하는 것에 대해 배웁니다.

조인을 사용하여 여러 테이블의 데이터 표시



ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- Equijoin 및 Nonequijoin을 사용하여 두 개 이상의 테이블에서 데이터에 액세스하는 `SELECT` 문 작성
- Self Join을 사용하여 테이블을 자체 조인
- Outer Join을 사용하여 일반적으로 조인 조건을 충족하지 않는 데이터 보기
- 두 개 이상의 테이블에서 모든 행의 Cartesian Product 생성

ORACLE

Copyright © 2009, 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.

참고: 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

- **JOINS 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 © 2009, Oracle. All rights reserved.

여러 테이블에서 데이터 가져오기

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

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
...			
18	205	110	Accounting
19	206	110	Accounting

ORACLE

Copyright © 2009, 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 © 2009, Oracle. All rights reserved.

조인 유형

테이블을 조인하려면 SQL:1999 표준과 호환되는 조인 구문을 사용합니다.

참고

- Oracle9i 버전 이전에는 조인 구문이 ANSI(American National Standards Institute) 표준과 달랐습니다. SQL:1999 호환 조인 구문은 이전 버전에서 제공하던 Oracle 고유의 조인 구문에 성능상의 이점을 제공하지 않습니다. 고유 조인 구문에 대한 자세한 내용은 부록 F: Oracle 조인 구문을 참조하십시오.
- 다음 슬라이드에서는 SQL:1999 조인 구문에 대해 설명합니다.

SQL:1999 구문을 사용한 테이블 조인

조인을 사용하여 둘 이상의 테이블에서 데이터를 query합니다.

```
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];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

SQL:1999 구문을 사용한 테이블 조인

이 구문에서 다음이 적용됩니다.

- table1.column은 데이터가 검색되는 테이블과 열을 나타냅니다.
- NATURAL JOIN은 동일한 열 이름을 기반으로 두 테이블을 조인합니다.
- JOIN table2 USING column_name은 열 이름을 기반으로 Equijoin을 수행합니다.
- JOIN table2 ON table1.column_name = table2.column_name은 ON 절의 조건을 기반으로 Equijoin을 수행합니다.
- LEFT/RIGHT/FULL OUTER는 OUTER Join을 수행하는 데 사용됩니다.
- CROSS JOIN은 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 향상합니다.
- 전체 테이블 이름 접두어 대신 테이블 alias를 사용합니다.
- 테이블 alias로 테이블에 짧은 이름을 지정합니다.
 - SQL 코드 크기를 줄어 메모리를 적게 사용합니다.
- 열 alias를 사용하여 이름은 같지만 서로 다른 테이블에 상주하는 열을 구분합니다.

ORACLE

Copyright © 2009, 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 않으면 SELECT 리스트의 DEPARTMENT_ID 열을 DEPARTMENTS 테이블이나 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 작게 유지할 수 있으므로 메모리 사용량도 줄어듭니다.

테이블 이름은 완전하게 지정되며 뒤에 공백이 온 후에 테이블 alias가 옵니다. 예를 들어 EMPLOYEES 테이블에는 e라는 alias를 지정하고 DEPARTMENTS 테이블에는 d라는 alias를 지정할 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 해당 테이블 alias를 사용해야 합니다.
- 테이블 alias는 의미 있는 단어여야 합니다.
- 테이블 alias는 현재 SELECT 문에 대해서만 유효합니다.

단원 내용

- JOINS 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 © 2009, Oracle. All rights reserved.

Natural Join 생성

- **NATURAL JOIN 절은 이름이 같은 두 테이블의 모든 행을 기반으로 합니다.**
- **이 절은 두 테이블에서 대응되는 모든 열의 값이 동일한 행을 선택합니다.**
- **동일한 이름을 가진 열이 서로 다른 데이터 유형을 가지면 오류가 반환됩니다.**

ORACLE

Copyright © 2009, 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.

참고: 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 © 2009, 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 절로 조인 생성

- 여러 열이 이름은 같지만 데이터 유형은 다를 경우 USING 절을 사용하여 Equijoin에 대한 열을 지정할 수 있습니다.
- USING 절을 사용하면 두 개 이상의 열이 일치하는 경우 하나의 열만 일치하도록 할 수 있습니다.
- NATURAL JOIN과 USING 절은 상호 배타적입니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

USING 절로 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
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

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

Foreign Key

Primary Key

ORACLE

Copyright © 2009, Oracle. All rights reserved.

열 이름 조인

사원의 부서 이름을 확인하려면 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.

참고: 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	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50
...				
18	206	Gietz	1700	110
19	205	Higgins	1700	110

ORACLE

Copyright © 2009, Oracle. All rights reserved.

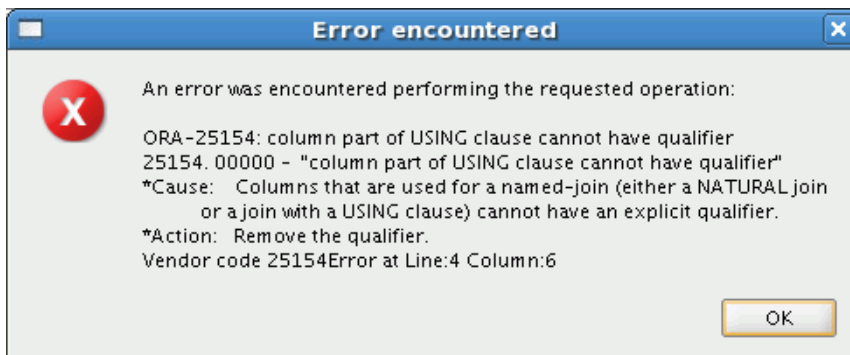
USING 절을 사용하여 레코드 검색

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 © 2009, 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 임의의 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"라는 오류가 발생합니다.

USING 절에 테이블 Alias 사용(계속)

manager_id가 employees 및 departments 테이블에 모두 존재하므로 다음 명령문에서 manager_id에 테이블 alias가 접두어로 사용되지 않으면 "column ambiguously defined"라는 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 절로 조인 생성

- Natural Join의 조인 조건은 기본적으로 이름이 같은 모든 열의 equijoin입니다.
- ON 절을 사용하여 임의 조건을 지정하거나 조인할 열을 지정합니다.
- 조인 조건은 다른 검색 조건과는 별개입니다.
- ON 절을 사용하면 코드를 이해하기 쉽습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

ON 절로 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  
ON     (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		144 Vargas	50	50	1500
5		143 Matos	50	50	1500
6		142 Davies	50	50	1500
7		141 Rajs	50	50	1500
8		124 Mourgos	50	50	1500
9		103 Hunold	60	60	1400
10		104 Ernst	60	60	1400
11		107 Lorentz	60	60	1400

...

ORACLE

Copyright © 2009, 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.

참고: SQL Developer에서 Execute Statement 아이콘을 사용하여 query를 실행할 경우 두 개의 department_id를 구분하기 위해 뒤에 '_1'이 붙습니다.

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
9	142	South San Francisco	Shipping

...

ORACLE

Copyright © 2009, 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 테이블에서 모두 열을 참조할 수 있습니다.

참고: 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  
ON     (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  
ON     (e.department_id = d.department_id)  
WHERE  e.manager_id = 149 ;
```

ORACLE

Copyright © 2009, 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	174	Abel	80	80	2500
2	176	Taylor	80	80	2500

단원 내용

- JOINS 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 © 2009, Oracle. All rights reserved.

테이블 자체 조인

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

**WORKER 테이블의 MANAGER_ID는 MANAGER
테이블의 EMPLOYEE_ID와 같습니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

테이블 자체 조인



때때로 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.

본 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

...

ORACLE

Copyright © 2009, 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.

참고: 슬라이드 예제의 (e.manager_id = m.employee_id)처럼 조인된 열을 괄호로 둘러싸는 것은 **선택 사항**입니다. 따라서 ON e.manager_id = m.employee_id도 제대로 작동합니다.

단원 내용

- JOINS 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 © 2009, Oracle. All rights reserved.

Nonequijoin

EMPLOYEES

	A	LAST_NAME	A	SALARY
1		Whalen		4400
2		Hartstein		13000
3		Fay		6000
4		Higgins		12000
5		Gietz		8300
6		King		24000
7		Kochhar		17000
8		De Haan		17000
9		Hunold		9000
10		Ernst		6000
...				
19		Taylor		8600
20		Grant		7000

JOB_GRADES

	A	GRADE_LEVEL	A	LOWEST_SAL	A	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 테이블은 각 GRADE_LEVEL에 대해 LOWEST_SAL 및 HIGHEST_SAL 값의 범위를 정의합니다. 따라서 GRADE_LEVEL 열을 사용하여 각 사원에 등급을 지정할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Nonequijoin

Nonequijoin은 등호 연산자 외의 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.

Nonequijoin을 사용하여 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 © 2009, Oracle. All rights reserved.

Nonequijoin을 사용하여 레코드 검색

슬라이드의 예제에서는 사원의 급여 등급을 평가하는 Nonequijoin을 생성합니다. 급여는 낮은 급여 범위와 높은 급여 범위 쌍(pair) 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 급여 등급 테이블의 한 행에서 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.

참고: <= 및 >=와 같은 다른 조건을 사용할 수 있지만 BETWEEN이 가장 간단합니다. BETWEEN 조건을 사용할 때는 맨 처음에 낮은 값을 지정하고 마지막에 높은 값을 지정해야 합니다. Oracle 서버는 BETWEEN 조건을 AND 조건 쌍(pair)으로 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

- JOINS 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 © 2009, Oracle. All rights reserved.

OUTER Join과 직접 일치하지 않는 레코드 반환

DEPARTMENTS

	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

부서 190에는 사원이
없습니다.

"Grant" 사원에게는
부서 ID가 할당되지
않았습니다.

EMPLOYEES와 Equijoin

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst

...

18	80	Abel
19	80	Taylor

ORACLE

Copyright © 2009, Oracle. All rights reserved.

OUTER Join과 직접 일치하지 않는 레코드 반환

조인 조건을 충족하지 못하는 행은 query 결과에 나타나지 않습니다.

슬라이드의 예제에서는 EMPLOYEES 및 DEPARTMENTS 테이블에 단일 Equijoin 조건이 사용되어 오른쪽에 결과를 반환합니다. 결과 집합에는 다음 항목이 포함되지 않습니다.

- 부서 ID 190(EMPLOYEES 테이블에 기록된 해당 부서 ID를 가진 사원이 없음)
- 성이 Grant인 사원(부서 ID가 할당되지 않음)

사원이 없는 부서 레코드 또는 부서가 할당되지 않은 사원을 반환하려면 OUTER Join을 사용하면 됩니다.

INNER Join과 OUTER Join

- SQL:1999에서 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 © 2009, Oracle. All rights reserved.

INNER Join과 OUTER Join

NATURAL JOIN, **USING** 또는 **ON** 절을 사용하여 테이블을 조인하면 결과는 **INNER Join**이 됩니다. 일치하지 않는 행은 출력에 표시되지 않습니다. 일치하지 않는 행을 반환하려면 **OUTER Join**을 사용하면 됩니다. **OUTER Join**은 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
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

LEFT OUTER JOIN

이 질의는 DEPARTMENTS 테이블에 대응되는 행이 없어도 왼쪽 테이블인 EMPLOYEES 테이블의 모든 행을 검색합니다.

RIGHT OUTER JOIN

```
SELECT e.last_name, d.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
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

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

RIGHT OUTER JOIN

이 query는 EMPLOYEES 테이블에 대응되는 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) ;
```

R2	LAST_NAME	R2	DEPARTMENT_ID	R2	DEPARTMENT_NAME
1	Whalen		10		Administration
2	Hartstein		20		Marketing
3	Fay		20		Marketing
4	Higgins		110		Accounting

...

17	Zlotkey		80		Sales
18	Abel		80		Sales
19	Taylor		80		Sales
20	Grant		(null)	(null)	
21	(null)		190		Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

FULL OUTER JOIN

이 query는 DEPARTMENTS 테이블에 대응되는 행이 없어도 EMPLOYEES 테이블의 모든 행을 검색합니다. 또한 EMPLOYEES 테이블에 일치하는 행이 없어도 DEPARTMENTS 테이블의 모든 행을 검색합니다.

단원 내용

- JOINS 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 © 2009, Oracle. All rights reserved.

Cartesian Product

- 다음과 같은 경우에 Cartesian Product가 생성됩니다.
 - 조인 조건이 생략된 경우
 - 조인 조건이 잘못된 경우
 - 첫번째 테이블의 모든 행이 두번째 테이블의 모든 행에 조인되는 경우
- Cartesian Product가 생성되지 않게 하려면 반드시 유효한 조인 조건을 포함해야 합니다.

ORACLE

Copyright © 2009, 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.

Cartesian Product는 일부 테스트에서 적당량의 데이터를 시뮬레이트하기 위해 많은 행을 생성해야 하는 경우에 유용합니다.

Cartesian Product 생성

EMPLOYEES (20행)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
...			
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS (8행)

	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 x 8 = 160행

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700
...			
21	200	10	1800
22	201	20	1800
...			
159	176	80	1700
160	178	(null)	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Cartesian Product 생성

조인 조건을 생략하면 Cartesian Product가 생성됩니다. 슬라이드의 예제에서는 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 Product라고도 합니다.

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

	1	LAST_NAME	2	DEPARTMENT_NAME
	1	Abel		Administration
	2	Davies		Administration
	3	De Haan		Administration
	4	Ernst		Administration
	5	Fay		Administration

...

	158	Vargas		Contracting
	159	Whalen		Contracting
	160	Zlotkey		Contracting

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Cross Join 생성

슬라이드의 예제는 EMPLOYEES 및 DEPARTMENTS 테이블의 Cartesian Product를 생성합니다. CROSS JOIN 기법은 여러 상황에 유용하게 적용할 수 있습니다. 예를 들어 사무소별로 월별 총 인건비를 반환하려면 X월에 인건비가 없다고 하더라도 Office를 모든 Month 테이블과 CROSS Join하면 됩니다.

Cartesian Product를 구하려는 경우 SELECT에 CROSS JOIN을 명시적으로 사용하는 것이 좋습니다. 이렇게 하면 Cartesian Product를 구하도록 명확하게 지정할 수 있으며 결과에 조인이 누락되지 않습니다.

퀴즈

SQL:1999 표준 조인 구문은 다음 유형의 조인을 지원합니다.

다음 조인 유형 중 오라클 조인 구문에서 지원하는 것은 무엇입니까?

1. Equijoin
2. Nonequijoin
3. Left OUTER Join
4. Right OUTER Join
5. Full OUTER Join
6. Self Join
7. Natural Join
8. Cartesian Product

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3, 4, 6, 8

요약

이 단원에서는 다음과 같은 조인을 사용하여 여러 테이블의 데이터를 표시하는 방법을 배웁니다.

- **Equijoin**
- **Nonequijoin**
- **OUTER Join**
- **Self Join**
- **Cross Join**
- **Natural Join**
- **Full(또는 양쪽) OUTER Join**

ORACLE

Copyright © 2009, 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.

- Equijoin을 사용하여 테이블 조인
- Outer Join 및 Self Join 수행
- 조건 추가

ORACLE

Copyright © 2009, 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.

Subquery를 사용하여 Query 해결

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- subquery 정의
- subquery가 해결할 수 있는 문제 유형 설명
- subquery의 유형 나열
- 단일 행 및 여러 행 subquery 작성

ORACLE

Copyright © 2009, 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 여러 행 subquery에 대해서도 다룹니다.

단원 내용

- **Subquery: 유형, 구문 및 지침**
- **단일 행 Subquery:**
 - Subquery의 그룹 함수
 - Subquery가 있는 HAVING 절
- **여러 행 Subquery**
 - ALL 또는 ANY 연산자 사용
- **EXISTS 연산자 사용**
- **Subquery의 Null 값**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Subquery를 사용하여 문제 해결

Abel보다 급여가 많은 사람은 누구입니까?

Main query:



어떤 사원이 Abel보다 급여가 많습니까?

Subquery:



Abel의 급여는 얼마입니까?



ORACLE

Copyright © 2009, 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 query에서 사용됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Subquery 구문

subquery는 다른 SELECT 문의 절에 포함되는 SELECT 문입니다. Subquery를 사용하면 단순하면서도 강력한 명령문을 구축할 수 있습니다. subquery는 테이블 자체의 데이터에 종속되는 조건을 사용하여 테이블에서 행을 선택해야 하는 경우에 매우 유용합니다.

다음은 포함하여 다양한 SQL 절에 subquery를 배치할 수 있습니다.

- WHERE 절
- HAVING 절
- FROM 절

이 구문에서 다음이 적용됩니다.

*operator*는 >, =, IN 등의 비교 조건을 포함합니다.

참고: 비교 조건은 단일 행 연산자(>, =, >=, <, <>, <=)와 여러 행 연산자(IN, ANY, ALL, EXISTS)로 구분됩니다.

Subquery는 중첩된 SELECT 문, 하위 SELECT 문 또는 내부 SELECT 문이라고도 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 > 11000
      (SELECT salary
       FROM employees
       WHERE last_name = 'Abel');
```

	1	LAST_NAME	2	SALARY
	1	Hartstein		13000
	2	Higgins		12000
	3	King		24000
	4	Kochhar		17000
	5	De Haan		17000

ORACLE

Copyright © 2009, 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

- subquery는 괄호로 묶습니다.
- 가독성을 위해 비교 조건의 오른쪽에 subquery를 배치합니다. 그러나 subquery는 비교 연산자의 양쪽 어디에나 사용할 수 있습니다.
- 단일 행 subquery에는 단일 행 연산자를 사용하고 여러 행 subquery에는 여러 행 연산자를 사용합니다.

ORACLE

Copyright © 2009, 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 여러 행 연산자라는 두 가지 유형의 비교 조건을 사용합니다.

Subquery 유형

- 단일 행 Subquery



- 여러 행 Subquery



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Subquery 유형

- 단일 행 subquery: 내부 SELECT 문에서 하나의 행만 반환하는 query
- 여러 행 subquery: 내부 SELECT 문에서 두 개 이상의 행을 반환하는 query

참고: 내부 SELECT 문에서 두 개 이상의 열을 반환하는 여러 열 subquery도 있습니다.
이에 대해서는 *Oracle Database 11g: SQL Fundamentals II* 과정에서 다룹니다.

단원 내용

- Subquery: 유형, 구문 및 지침
- 단일 행 Subquery:
 - Subquery의 그룹 함수
 - Subquery가 있는 HAVING 절
- 여러 행 Subquery
 - ALL 또는 ANY 연산자 사용
- EXISTS 연산자 사용
- Subquery의 Null 값

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

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 © 2009, 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 = (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary > (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

단일 행 Subquery 실행

SELECT 문은 query 블록으로 간주할 수 있습니다. 슬라이드의 예제는 "Taylor"와 직무가 같고 급여는 더 많은 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.

참고: Outer query와 Inner query는 다른 테이블에서 데이터를 가져올 수 있습니다.

Subquery에서 그룹 함수 사용

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary = 2500
          (SELECT MIN(salary)
           FROM   employees);
```

	LAST_NAME	JOB_ID	SALARY
1	Vargas	ST_CLERK	2500

ORACLE

Copyright © 2009, 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) > (SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

2500

	DEPARTMENT_ID	MIN(SALARY)
1	(null)	7000
2	20	6000
3	90	17000
4	110	8300
5	80	8600
6	10	4400
7	60	4200

ORACLE

Copyright © 2009, 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:

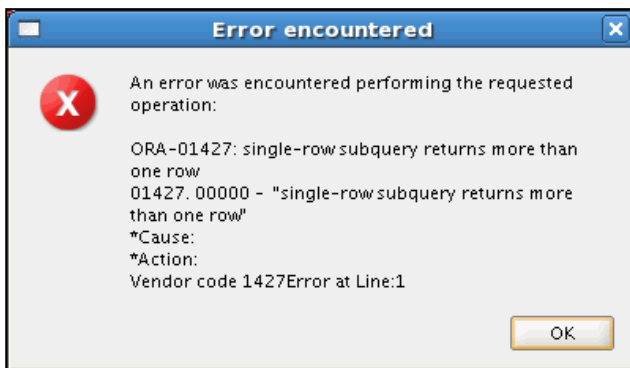
평균 급여가 가장 낮은 직무를 찾습니다.

```
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)
1	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);
```



여러 행 subquery에
단일 행 연산자 사용

ORACLE

Copyright © 2009, 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 여러 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 © 2009, Oracle. All rights reserved.

Inner query에서 반환된 행이 없음

subquery의 일반적인 또 다른 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 Haas라는 사원이 없기 때문에 실행될 때 아무 행도 선택하지 않습니다. 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:
 - Subquery의 그룹 함수
 - Subquery가 있는 HAVING 절
- 여러 행 Subquery
 - IN, ALL 또는 ANY 사용
- EXISTS 연산자 사용
- Subquery의 Null 값

ORACLE

Copyright © 2009, Oracle. All rights reserved.

여러 행 Subquery

- 두 개 이상의 행을 반환합니다.
- 여러 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 FALSE로 평가됩니다.
ALL	=, !=, >, <, <=, >= 연산자가 앞에 있어야 합니다. 값 하나를 리스트의 모든 값 또는 query에서 반환된 모든 값과 비교합니다. query에서 반환된 행이 없으면 TRUE로 평가됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

여러 행 Subquery

두 개 이상의 행을 반환하는 subquery를 여러 행 subquery라고 합니다. 여러 행 subquery에는 단일 행 연산자 대신 여러 행 연산자를 사용합니다. 여러 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);
```

여러 행 Subquery에서 ANY 연산자 사용

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
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 © 2009, Oracle. All rights reserved.

여러 행 Subquery에서 ANY 연산자 사용

ANY 연산자 및 그 동의어인 SOME 연산자는 값을 subquery에서 반환되는 각 값과 비교합니다. 슬라이드의 예제는 IT 프로그래머가 아닌 직원 중 급여가 IT 프로그래머보다 적은 직원을 표시합니다. 프로그래머가 받는 최고 급여는 \$9,000입니다.

- <ANY는 최대값보다 작음을 의미합니다.
- >ANY는 최소값보다 큼을 의미합니다.
- =ANY는 IN과 같습니다.

여러 행 Subquery에서 ALL 연산자 사용

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	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 © 2009, Oracle. All rights reserved.

여러 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.

EXISTS 연산자 사용

```
SELECT * FROM departments
WHERE NOT EXISTS
(SELECT * FROM employees
 WHERE employees.department_id=departments.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	190	Contracting	(null)	1700

ORACLE

Copyright © 2009, Oracle. All rights reserved.

EXISTS 연산자 사용

EXISTS 연산자는 테이블에 특정 행이 있는지 여부에 따라 query 결과가 달라지는 query에 사용됩니다. subquery에서 최소한 한 개의 행을 반환하면 TRUE로 평가됩니다.

슬라이드의 예제는 사원이 없는 부서를 표시합니다. DEPARTMENTS 테이블의 각 행에 대해 EMPLOYEES 테이블에 부서 ID가 동일한 행이 있는지를 확인하는 조건 검사가 수행됩니다. 그러한 행이 없으면 해당 행은 조건을 충족하는 것이므로 선택됩니다. EMPLOYEES 테이블에 그러한 행이 있을 경우 해당 행은 선택되지 않습니다.

단원 내용

- Subquery: 유형, 구문 및 지침
- 단일 행 Subquery:
 - Subquery의 그룹 함수
 - Subquery가 있는 HAVING 절
- 여러 행 Subquery
 - ALL 또는 ANY 연산자 사용
- EXISTS 연산자 사용
- Subquery의 Null 값

ORACLE

Copyright © 2009, 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 © 2009, 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);
```

퀴즈

subquery를 사용하는 것은 두 query를 순차적으로 수행하여 첫번째 query 결과를 두번째 query의 검색 값으로 사용하는 것과 동일한 기능입니다.

1. 맞습니다.
2. 틀립니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- subquery가 문제 해결에 도움이 될 수 있는 경우 식별
- query가 알 수 없는 값을 기반으로 하는 경우 subquery 작성

```
SELECT    select_list
FROM      table
WHERE     expr operator
          (SELECT select_list
           FROM    table);
```

ORACLE

Copyright © 2009, 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 여러 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 query하는 subquery 작성
- subquery를 사용하여 한 데이터 집합에 있고 다른 데이터 집합에는 없는 값 찾기

ORACLE

Copyright © 2009, 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.

8

집합 연산자 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 집합 연산자 설명
- 집합 연산자를 사용하여 여러 query를 단일 query로 조합
- 반환되는 행 순서 제어

ORACLE

Copyright © 2009, Oracle. All rights reserved.

목표

이 단원에서는 집합 연산자를 사용하여 query를 작성하는 방법을 배웁니다.

단원 내용

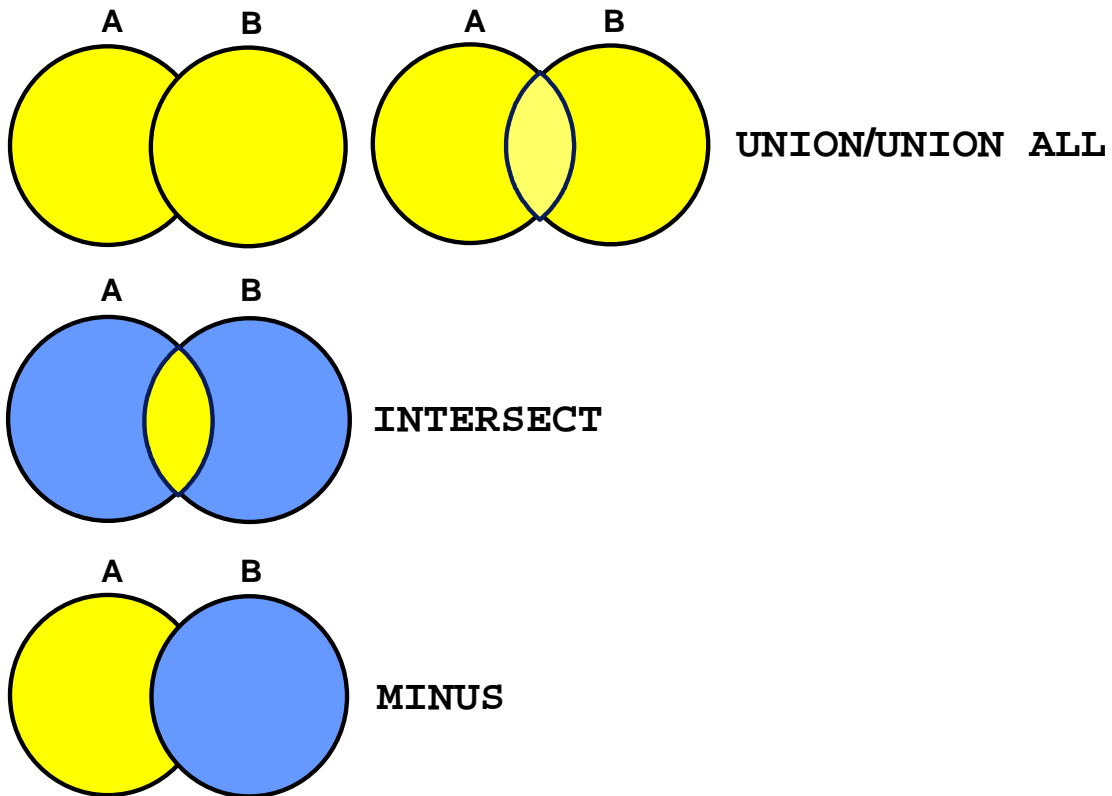
- **집합 연산자: 유형 및 지침**
- 이 단원에 사용되는 테이블
- UNION 및 UNION ALL 연산자
- INTERSECT 연산자
- MINUS 연산자
- SELECT 문 일치
- 집합 연산에서 ORDER BY 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

집합 연산자



ORACLE

Copyright © 2009, Oracle. All rights reserved.

집합 연산자

집합 연산자는 둘 이상의 구성 요소 query 결과를 하나의 결과로 조합합니다. 집합 연산자가 포함된 query를 복합 query라고 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 괄호가 명시적으로 다른 순서를 지정하지 않는 한 왼쪽(위)에서 오른쪽(아래)으로 연산자를 평가합니다. 다른 집합 연산자와 함께 INTERSECT 연산자가 사용된 query에서는 괄호를 사용하여 평가 순서를 명시적으로 지정해야 합니다.

집합 연산자 지침

- **SELECT 리스트의 표현식은 개수가 일치해야 합니다.**
- **두번째 query에 있는 각 열의 데이터 유형은 첫번째 query에 있는 상응하는 열의 데이터 유형과 일치해야 합니다.**
- **실행 순서를 변경하려면 괄호를 사용합니다.**
- **ORDER BY 절은 명령문의 맨 끝에만 올 수 있습니다.**

ORACLE

Copyright © 2009, 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 query에서는 query의 SELECT 리스트에 있는 열의 데이터 유형이 정확히 일치하지 않을 수도 있습니다. 그러나 두번째 query의 열은 첫번째 query의 상응하는 열과 데이터 유형 그룹(예: 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 ALL의 경우를 제외하면 중복 행은 자동으로 제거됩니다.
- 첫번째 query의 열 이름이 결과에 표시됩니다.
- UNION ALL의 경우를 제외하면 결과는 기본적으로 오름차순으로 정렬됩니다.

ORACLE

Copyright © 2009, 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. 출력에서 열 이름은 첫번째 SELECT 문의 열 목록에 따라 결정됩니다. 기본적으로 출력은 SELECT 절의 첫번째 열의 오름차순으로 정렬됩니다.

복합 query의 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 길이가 동일한 CHAR 데이터 유형의 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 © 2009, Oracle. All rights reserved.

이 단원에 사용되는 테이블

이 단원에서는 다음과 같은 테이블이 사용됩니다.

- **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 부서의 세부 사항 기록

ORACLE

Copyright © 2009, Oracle. All rights reserved.

이 단원에 사용되는 테이블

이 단원에서는 두 개의 테이블, 즉 EMPLOYEES 테이블과 JOB_HISTORY 테이블을 사용합니다.

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.

회사를 오랫동안 다닌 일부 사원의 경우 다양한 직무를 거쳤을 것입니다. 이러한 사항은 JOB_HISTORY 테이블로 모니터링됩니다. 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)

회사 재직 중에 동일한 직책을 두 번 이상 담당하게 될 수도 있습니다. 예를 들어, 1998년 3월 24일에 입사한 사원 Taylor의 경우를 살펴봅시다. Taylor는 1998년 3월 24일부터 1998년 12월 31일까지의 기간 동안 SA_REP 직위를 보유하고, 1999년 1월 1일부터 1999년 12월 31일까지의 기간 동안에는 SA_MAN 직위를 보유하고 있습니다. Taylor는 다시 현재 직위인 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)
11 rows selected		

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	200	Whalen	AD_ASST	17-SEP-87	10
2	201	Hartstein	MK_MAN	17-FEB-96	20
3	202	Fay	MK_REP	17-AUG-97	20
4	205	Higgins	AC_MGR	07-JUN-94	110
5	206	Gietz	AC_ACCOUNT	07-JUN-94	110
6	100	King	AD_PRES	17-JUN-87	90
7	101	Kochhar	AD_VP	21-SEP-89	90
8	102	De Haan	AD_VP	13-JAN-93	90
9	103	Hunold	IT_PROG	03-JAN-90	60
10	104	Ernst	IT_PROG	21-MAY-91	60
11	107	Lorentz	IT_PROG	07-FEB-99	60
12	124	Mourgos	ST_MAN	16-NOV-99	50
13	141	Rajs	ST_CLERK	17-OCT-95	50
14	142	Davies	ST_CLERK	29-JAN-97	50
15	143	Matos	ST_CLERK	15-MAR-98	50
16	144	Vargas	ST_CLERK	09-JUL-98	50
17	149	Zlotkey	SA_MAN	29-JAN-00	80
18	174	Abel	SA_REP	11-MAY-96	80
19	176	Taylor	SA_REP	24-MAR-98	80
20	178	Grant	SA_REP	24-MAY-99	(null)






```
DESCRIBE job_history
```

DESCRIBE job_history		
Name	Null	Type

EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)
5 rows selected		

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

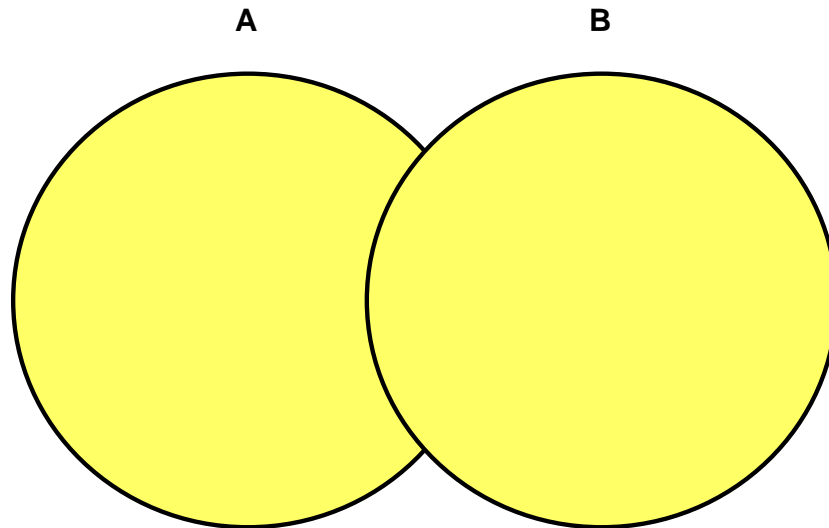
단원 내용

- 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 © 2009, Oracle. All rights reserved.

UNION 연산자



UNION 연산자는 중복 행을 제거한 후 양쪽 query에서 행을 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UNION 연산자

UNION 연산자는 양쪽 query에 의해 선택된 모든 행을 반환합니다. UNION 연산자를 사용하여 여러 테이블의 모든 행을 반환하고 중복된 모든 행을 제거합니다.

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 적용됩니다.
- NULL 값은 중복 검사 시 무시되지 않습니다.
- 기본적으로 출력은 SELECT 절의 열을 기준으로 오름차순으로 정렬됩니다.

UNION 연산자 사용

모든 사원의 현재 및 이전 직무 세부 사항을 표시합니다.
각 사원을 한번만 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
...		
27	205	AC_MGR
28	206	AC_ACCOUNT

ORACLE

Copyright © 2009, 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.

UNION 연산자 사용(계속)

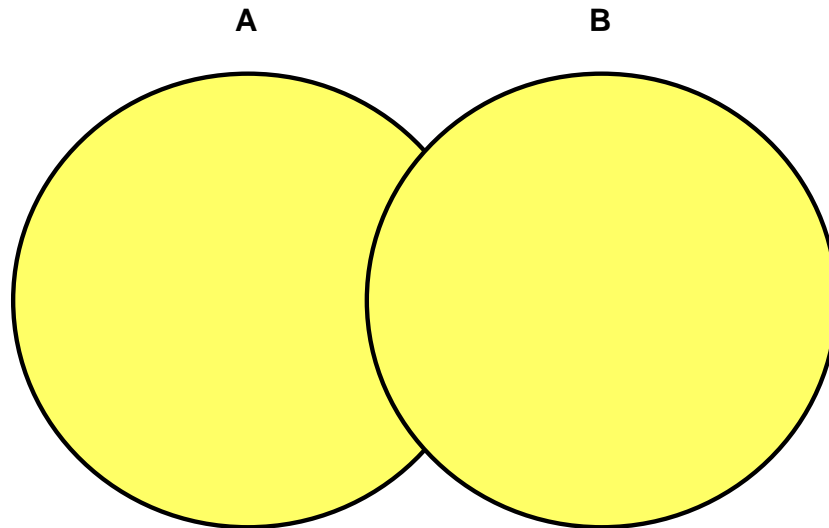
다음 예제를 살펴보십시오.

```
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
...			
22	200	AC_ACCOUNT	90
23	200	AD_ASST	10
24	200	AD_ASST	90
...			
29	206	AC_ACCOUNT	110

앞의 출력에서 사원 200은 세 번 표시되었습니다. 그 이유는 무엇입니까? 사원 200의 DEPARTMENT_ID 값을 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 query의 결과를 반환합니다.

ORACLE

Copyright © 2009, 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
...		
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)
23	200 AD_ASST	10
...		
30	206 AC_ACCOUNT	110

ORACLE

Copyright © 2009, Oracle. All rights reserved.

UNION ALL 연산자 사용

예제에서는 30개 행이 선택됩니다. 두 테이블의 총합은 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
------------	----

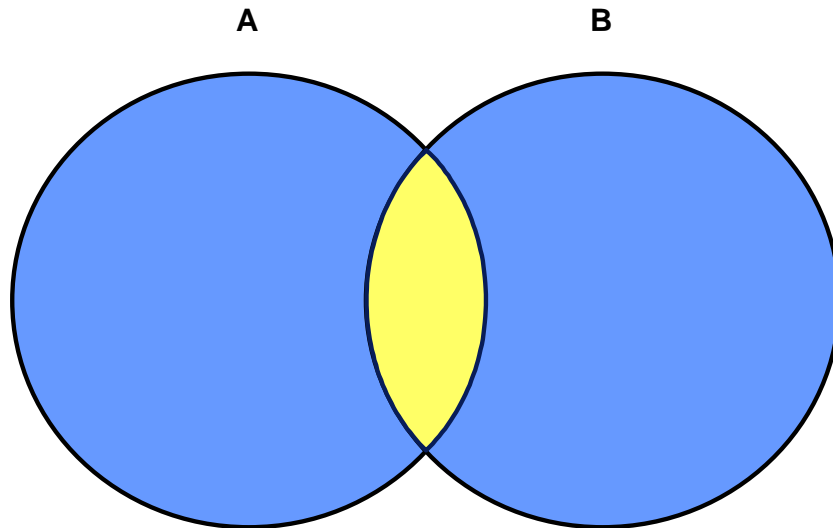
단원 내용

- 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 © 2009, Oracle. All rights reserved.

INTERSECT 연산자



INTERSECT 연산자는 양쪽 query에 공통되는 행을 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

INTERSECT 연산자

INTERSECT 연산자를 사용하여 여러 query에 공통되는 모든 행을 반환합니다.

지침

- query에서 SELECT 문으로 선택된 열의 개수와 데이터 유형은 query에 사용된 모든 SELECT 문에서 동일해야 합니다. 그러나 열 이름은 동일하지 않아도 됩니다.
- 교차 테이블 순서를 반대로 해도 결과에는 영향을 주지 않습니다.
- INTERSECT에서는 NULL 값이 무시되지 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 © 2009, 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? 이 결과는 사용된 다른 열 값의 중복 여부에 따라 다를 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 나타나지 않습니다.

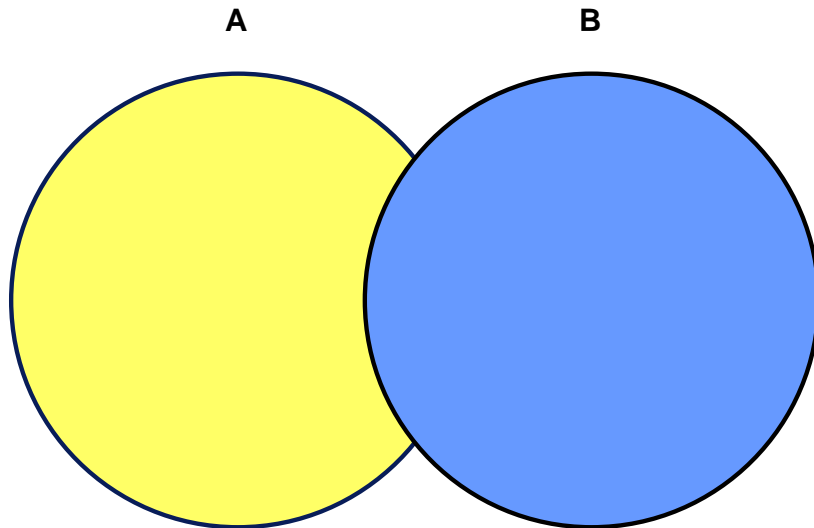
단원 내용

- 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 © 2009, Oracle. All rights reserved.

MINUS 연산자



MINUS 연산자는 첫번째 query에 의해 선택되지만 두번째 query 결과 집합에는 없는 모든 구분 행을 반환합니다.

ORACLE

Copyright © 2009, 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).

참고: 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

한번도 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
...	
13	202
14	205
15	206

ORACLE

Copyright © 2009, 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. 이것은 직무가 한 번도 변경되지 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 © 2009, 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 © 2009, 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 머릿글로 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)

...

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 © 2009, 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 © 2009, Oracle. All rights reserved.

집합 연산에서 ORDER BY 절 사용

- ORDER BY 절은 복합 query의 맨 끝에 한 번만 올 수 있습니다.
- 구성 요소 query에 개별적으로 ORDER BY 절을 사용할 수 없습니다.
- ORDER BY 절은 첫번째 SELECT query의 열만 인식합니다.
- 기본적으로 첫번째 SELECT query의 첫번째 열을 기준으로 오름차순으로 출력이 정렬됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

집합 연산에서 ORDER BY 절 사용

ORDER BY 절은 복합 query에서 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 query의 첫번째 열을 기준으로 오름차순으로 정렬됩니다.

참고: ORDER BY 절은 두번째 SELECT query의 열 이름을 인식하지 못합니다. 열 이름을 혼동하지 않도록 ORDER BY 열 위치를 사용하는 것이 일반적입니다.

예를 들어, 다음 명령문의 경우 job_id의 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_id의 오름차순으로 정렬됩니다. 두번째 query의 열을 사용하여 출력을 정렬할 수는 없습니다.

퀴즈

집합 연산자에 대한 설명을 고르십시오.

1. SELECT 리스트의 표현식은 개수가 일치해야 합니다.
2. 괄호를 사용하여 실행 순서를 변경할 수 없습니다.
3. 두번째 query에 있는 각 열의 데이터 유형은 첫번째 query에 있는 상응하는 열의 데이터 유형과 일치해야 합니다.
4. ORDER BY 절은 UNION ALL 연산자를 사용하지 않는 한 복합 query에서 한 번만 사용할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3

요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 모든 구분 행을 반환하는 UNION
- 중복 행을 포함한 모든 행을 반환하는 UNION ALL
- 두 query에서 공유하는 모든 행을 반환하는 INTERSECT
- 첫번째 query에서는 선택되지만 두번째 query에서는 선택되지 않는 모든 구분 행을 반환하는 MINUS
- 명령문의 맨 끝에만 올 수 있는 ORDER BY

ORACLE

Copyright © 2009, Oracle. All rights reserved.

요약

- UNION 연산자는 복합 query에서 각 query에 의해 선택된 모든 구분 행을 반환합니다. UNION 연산자를 사용하여 여러 테이블의 모든 행을 반환하고 중복된 모든 행을 제거합니다.
- 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 연산자는 두번째 query에 없는 첫번째 query에서 반환된 행을 반환합니다.
- ORDER BY 절은 복합 문의 맨 끝에서만 사용할 수 있습니다.
- SELECT 리스트에 있는 해당 표현식에서 개수와 데이터 유형이 일치하는지 확인합니다.

연습 8: 개요

이 연습에서는 다음을 사용하여 보고서를 작성합니다.

- UNION 연산자
- INTERSECT 연산자
- MINUS 연산자

ORACLE

Copyright © 2009, Oracle. All rights reserved.

연습 8: 개요

이 연습에서는 집합 연산자를 사용하여 query를 작성합니다.

