

# Oracle Database 11g: SQL Fundamentals II(한글판)

블록 I • 학생용

D49994KR20

Edition 2.0

2009년 12월

D64454

**ORACLE®**

## 저자

Chaitanya Koratamaddi

Brian Pottle

Tulika Srivastava

## 기술 제공자 및 검토자

Claire Bennett

Ken Cooper

Yanti Chang

Laszlo Czinkoczeki

Burt Demchick

Gerlinde Frenzen

Joel Goodman

Laura Garza

Richard Green

Nancy Greenberg

Akira Kinutani

Wendy Lo

Isabelle Marchand

Timothy Mcglue

Alan Paulson

Srinivas Putrevu

Bryan Roberts

Clinton Shaffer

Abhishek Singh

Jenny Tsai Smith

James Spiller

Lori Tritz

Lex van der Werff

Marcie Young

## 편집자

Amitha Narayan

Daniel Milne

## 그래픽 디자이너

Satish Bettegowda

## 발행인

Veena Narasimhan

Copyright © 2009, Oracle. All rights reserved.

### Disclaimer

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

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

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

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

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

### Restricted Rights Notice

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

#### U.S. GOVERNMENT RIGHTS

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

### Trademark Notice

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

## 목차

### I 소개

단원 목표	I-2
단원 내용	I-3
과정 목표	I-4
선수 과정	I-5
과정 일정	I-6
단원 내용	I-7
본 과정에 사용되는 테이블	I-8
본 과정에 사용되는 부록	I-9
개발 환경	I-10
단원 내용	I-11
데이터 제한 검토	I-12
데이터 정렬 검토	I-13
SQL 함수 검토	I-14
단일 행 함수 검토	I-15
그룹 함수의 유형 검토	I-16
Subquery 사용 검토	I-17
데이터 조작 검토	I-18
단원 내용	I-19
Oracle Database 11g SQL 설명서	I-20
추가 자료	I-21
요약	I-22
연습 I: 개요	I-23

### 1 유저 액세스 제어

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

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

## 2 스키마 객체 관리

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

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

### 3 데이터 디렉터리 뷰를 사용하여 객체 관리

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

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

## 4 대형 데이터 집합 조작

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

## 5 다른 시간대에서 데이터 관리

목표 5-2

단원 내용 5-3

시간대 5-4

TIME\_ZONE 세션 파라미터 5-5

CURRENT\_DATE, CURRENT\_TIMESTAMP 및 LOCALTIMESTAMP 5-6

세션의 시간대에서 날짜와 시간 비교 5-7

DBTIMEZONE 및 SESSIONTIMEZONE 5-9

TIMESTAMP 데이터 유형 5-10

TIMESTAMP 필드 5-11

DATE 와 TIMESTAMP 의 차이 5-12

TIMESTAMP 데이터 유형 비교 5-13

단원 내용 5-14

INTERVAL 데이터 유형 5-15

INTERVAL 필드 5-17

INTERVAL YEAR TO MONTH: 예제 5-18

INTERVAL DAY TO SECOND 데이터 유형: 예제 5-20

단원 내용 5-21

EXTRACT 5-22

TZ\_OFFSET 5-23

FROM\_TZ 5-25

TO\_TIMESTAMP 5-26

TO\_YMINTERVAL 5-27

TO\_DSINTERVAL 5-28

일광 절약 시간 5-29

퀴즈 5-31

요약 5-32

연습 5: 개요 5-33

## 6 Subquery 를 사용하여 데이터 검색

목표 6-2

단원 내용 6-3

여러 열 Subquery 6-4

열 비교 6-5

쌍 방식 비교 Subquery 6-6

비쌍 방식 비교 Subquery 6-8

단원 내용 6-10

스칼라 Subquery 표현식 6-11

스칼라 Subquery: 예제 6-12

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

## 7 정규식 지원

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



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

## **부록 A: 연습 해답**

## **부록 B: 테이블 설명**

## **부록 C: SQL Developer 사용**

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

**부록 D: SQL\*Plus 사용**

목표 D-2

SQL 과 SQL\*Plus 의 상호 작용 D-3

SQL 문과 SQL\*Plus 명령 비교 D-4

SQL\*Plus 개요 D-5

SQL\*Plus 에 로그인 D-6

테이블 구조 표시 D-7

SQL\*Plus 편집 명령 D-9

LIST, n 및 APPEND 사용 D-11

CHANGE 명령 사용 D-12

SQL\*Plus 파일 명령 D-13

SAVE 및 START 명령 사용 D-14

SERVEROUTPUT 명령 D-15

SQL\*Plus SPOOL 명령 사용 D-16

AUTOTRACE 명령 사용 D-17

요약 D-18

**부록 E: JDeveloper 사용**

목표 E-2

Oracle JDeveloper E-3

Database Navigator E-4

연결 생성 E-5

데이터베이스 객체 탐색 E-6

SQL 문 실행 E-7

프로그램 단위 생성 E-8

컴파일 E-9

프로그램 단위 실행 E-10

프로그램 단위 삭제 E-11

Structure Window E-12

Editor Window E-13

Application Navigator E-14

Java 내장 프로시저 배치 E-15

PL/SQL 에 Java 게시(Publishing) E-16

JDeveloper 11g 에 대해 자세히 배울 수 있는 방법 E-17

요약 E-18

**부록 F: 관련 데이터를 그룹화하여 보고서 생성**

목표 F-2

그룹 함수 검토 F-3

GROUP BY 절 검토 F-4

HAVING 절 검토 F-5

GROUP BY 에 ROLLUP 및 CUBE 연산자 사용 F-6

ROLLUP 연산자 F-7

ROLLUP 연산자: 예제 F-8

CUBE 연산자 F-9

CUBE 연산자: 예제 F-10

GROUPING 함수 F-11

GROUPING 함수: 예제 F-12

GROUPING SETS F-13

GROUPING SETS: 예제 F-15

조합 열 F-17

조합 열: 예제 F-19

연결된 그룹화 F-21

연결된 그룹화: 예제 F-22

요약 F-23

**부록 G: 계층적 검색**

목표 G-2

EMPLOYEES 테이블의 예제 데이터 G-3

일반 트리 구조 G-4

Hierarchical Query G-5

트리 탐색 G-6

트리 탐색: 상향식 G-8

트리 탐색: 하향식 G-9

LEVEL 의사 열로 행 순위 지정 G-10

LEVEL 및 LPAD 를 사용하여 계층 보고서 형식 지정 G-11

분기 제거 G-13

요약 G-14

**부록 H: 고급 스크립트 작성**

목표 H-2

SQL 을 사용하여 SQL 생성 H-3

기본 스크립트 작성 H-5

환경 제어 H-6

전체 그림 H-7

테이블 내용을 파일로 덤프 H-8

동적 술어 생성 H-10

요약 H-12

**부록 I: 오라클 데이터베이스 구조 구성 요소**

목표 I-2

오라클 데이터베이스 구조: 개요 I-3

오라클 데이터베이스 서버 구조 I-4

데이터베이스에 연결 I-5

오라클 데이터베이스와 상호 작용 I-6

Oracle 메모리 구조 I-8

프로세스 구조 I-10

데이터베이스 기록자 프로세스 I-12

로그 기록자 프로세스 I-13

체크포인트 프로세스 I-14

시스템 모니터 프로세스 I-15

프로세스 모니터 프로세스 I-16

오라클 데이터베이스 저장 영역 구조 I-17

논리적 및 물리적 데이터베이스 구조 I-19

SQL 문 처리 I-21

Query 처리 I-22

Shared Pool I-23

데이터베이스 버퍼 캐시 I-25

프로그램 글로벌 영역(PGA) I-26

DML 문 처리 I-27

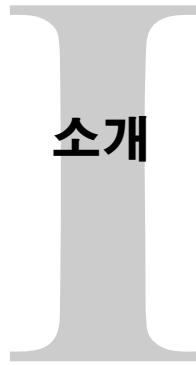
리두 로그 버퍼 I-29

롤백 세그먼트 I-30

COMMIT 처리 I-31

오라클 데이터베이스 구조 요약 I-33

**추가 연습****추가 연습 해답**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 단원 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 과정 목표 설명
- 본 과정에 사용되는 데이터베이스 스키마 및 테이블 설명
- 본 과정에서 사용할 수 있는 환경 식별
- 몇 가지 SQL 기본 개념 검토

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 단원 내용

- **과정 목표 및 과정 내용**
- 본 과정에 사용된 데이터베이스 스키마와 부록 및 본 과정에서 사용 가능한 개발 환경
- 몇 가지 SQL 기본 개념 검토
- 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.

- 특정 객체에 대한 데이터베이스 액세스 제어
- 다양한 액세스 권한 레벨을 가진 새 유저 추가
- 스키마 객체 관리
- 데이터 덱서너리 뷰를 사용하여 객체 관리
- Subquery를 사용하여 오라클 데이터베이스에서 대형 데이터 집합 조작
- 서로 다른 시간대에서의 데이터 관리
- 여러 열 Subquery 작성
- 스칼라 및 Correlated Subquery 사용
- SQL의 정규식 지원 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.



# 선수 과정

**Oracle Database 11g: SQL Fundamentals I 과정이 본 과정의  
선수 과정입니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 선수 과정

본 과정의 선수 과정은 *Oracle Database 11g: SQL Fundamentals I*입니다.

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
  - 유저 액세스 제어
  - 스키마 객체 관리
  - 데이터 디셔너리 뷰를 사용하여 객체 관리
- 둘째 날:
  - 대형 데이터 집합 조작
  - 다른 시간대에서 데이터 관리
  - Subquery를 사용하여 데이터 검색
  - 정규식 지원

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 단원 내용

- 과정 목표 및 과정 내용
- **본 과정에 사용된 데이터베이스 스키마와 부록  
및 본 과정에서 사용 가능한 개발 환경**
- 몇 가지 SQL 기본 개념 검토
- Oracle Database 11g 설명서 및 추가 자료

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

Oracle Internal & Oracle Academy Use Only



### 테이블 설명

## 테이블 설명

- Oracle Database 11g: SQL Fundamentals II I - 8**

## 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: 관련 데이터를 그룹화하여 보고서 생성
- 부록 G: 계층적 검색
- 부록 H: 고급 스크립트 작성
- 부록 I: 오라클 데이터베이스 구조 구성 요소

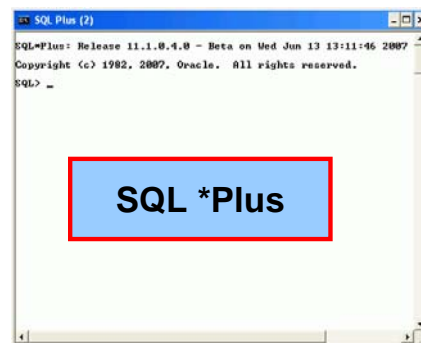
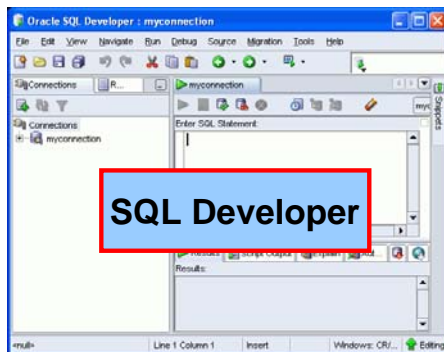
ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 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.

### 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 11g Release 2와 함께 제공되며 본 과정의 기본 도구입니다.
- 강의실 시스템에서도 SQL Developer 버전 1.5.4를 사용할 수 있습니다. 시스템에 SQL Developer 버전 1.5.4가 없는 경우 별도로 설치하여 사용할 수 있습니다. 본 과정의 등록 당시에는 버전 1.5.3이 SQL Developer의 최신 버전이었습니다.

#### SQL\*Plus

SQL\*Plus 환경을 사용하여 본 과정에서 다루는 모든 SQL 명령을 실행할 수도 있습니다.

#### 참고

- 버전 1.5.4 설치에 대한 간단한 지침을 비롯하여 SQL Developer 사용에 대한 자세한 내용은 부록 C "SQL Developer 사용"을 참조하십시오.
- SQL\*Plus 사용에 대한 자세한 내용은 부록 D "SQL\*Plus 사용"을 참조하십시오.

## 단원 내용

- 과정 목표 및 과정 내용
- 본 과정에 사용된 데이터베이스 스키마와 부록 및 본 과정에서 사용 가능한 개발 환경
- 몇 가지 SQL 기본 개념 검토
- Oracle Database 11g 설명서 및 추가 자료

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 단원 내용

다음 몇 개의 슬라이드는 *Oracle Database 11g: SQL Fundamentals I* 과정에서 설명한 몇 가지 기본 개념에 대한 간략한 개요를 제공합니다.

## 데이터 제한 검토

- WHERE 절을 사용하여 반환되는 행을 제한합니다.
- 비교 조건을 사용하여 특정 표현식을 다른 값이나 표현식과 비교할 수 있습니다.

연산자	의미
BETWEEN ...AND...	두 값 사이(경계값 포함)
IN( set )	값 리스트 중 일치하는 값 검색
LIKE	일치하는 문자 패턴 검색

- 논리 조건을 사용하여 두 구성 요소 조건의 결과를 결합하고 이러한 조건에 준하는 단일 결과를 생성할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 데이터 제한 검토

WHERE 절을 사용하여 query에서 반환되는 행을 제한할 수 있습니다. WHERE 절은 충족되어야 하는 조건을 포함하며 FROM 절 바로 뒤에 나옵니다.

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

- 열 이름
- 비교 조건
- 열 이름, 상수 또는 값 리스트

다음과 같은 형식으로 WHERE 절에 비교 조건을 사용할 수 있습니다.

... WHERE expr operator value

슬라이드에 표시된 것 이외에도 =, <, >, <>, <=, >= 등의 다른 비교 조건을 사용할 수 있습니다.

SQL에서는 다음 세 가지 논리 연산자가 제공됩니다.

- AND
- OR
- NOT



## 데이터 정렬 검토

- 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.

### 데이터 정렬 검토

Query 결과에 반환되는 행의 순서는 정의되어 있지 않습니다. ORDER BY 절을 사용하여 행을 정렬할 수 있습니다. ORDER BY 절을 사용할 경우 SQL 문의 맨 마지막에 와야 합니다. 표현식, alias 또는 열 위치를 정렬 조건으로 지정할 수 있습니다.

#### 구문

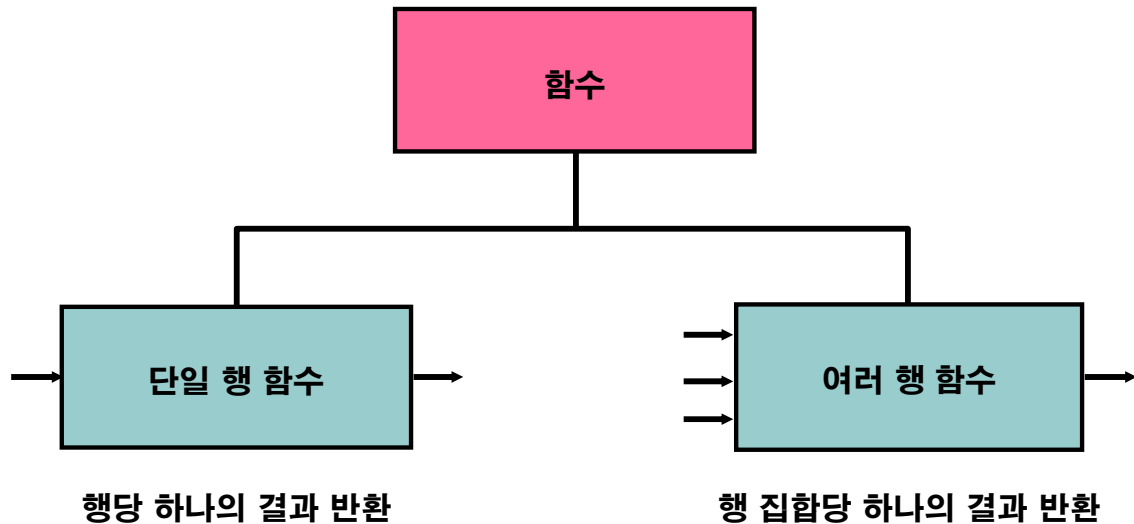
```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY {column, expr, numeric_position} [ASC|DESC]];
```

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.

# SQL 함수 검토



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## SQL 함수 검토

함수의 유형은 다음 두 가지입니다.

- 단일 행 함수
- 여러 행 함수

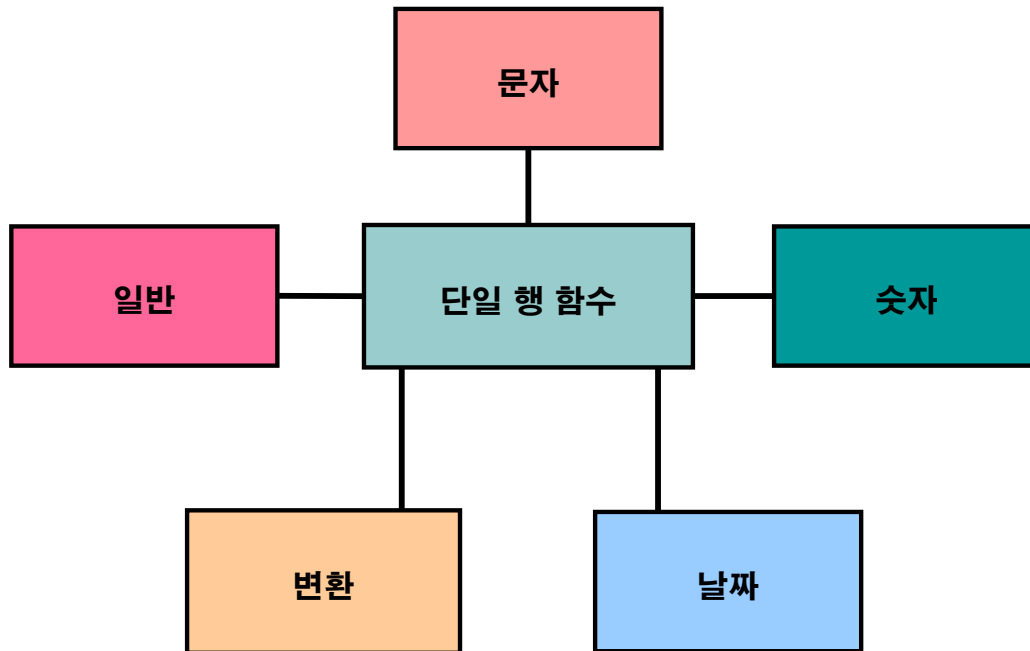
### 단일 행 함수

이 함수는 단일 행에 대해서만 실행되며 행당 하나의 결과를 반환합니다. 단일 행 함수의 유형으로는 문자, 숫자, 날짜, 변환 및 일반 함수 등이 있습니다.

### 여러 행 함수

이 함수에서는 행 그룹당 하나의 결과를 산출하도록 행 그룹을 조작할 수 있습니다. 이 함수를 *그룹 함수*라고도 합니다.

## 단일 행 함수 검토



ORACLE

Copyright © 2009, Oracle. All rights reserved.

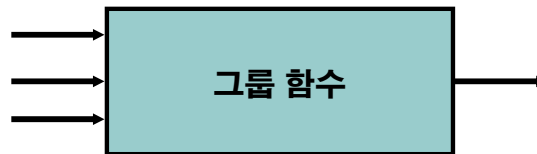
### 단일 행 함수 검토

단일 행 함수에는 여러 가지 유형이 있습니다.

- **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 제외한 모든 날짜 함수는 DATE 데이터 유형의 값을 반환합니다.
- **변환 함수:** 값의 데이터 유형을 변환합니다.
- **일반 함수:**
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
  - CASE
  - DECODE

## 그룹 함수의 유형 검토

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



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 그룹 함수의 유형 검토

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

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

## Subquery 사용 검토

- Subquery는 다른 SELECT 문의 절에 중첩된 SELECT 문입니다.
- 구문:

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

- Subquery 유형

단일 행 subquery	여러 행 subquery
한 행만 반환	둘 이상의 행 반환
단일 행 비교 연산자 사용	다중 행 비교 연산자 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Subquery 사용 검토

Subquery를 사용하면 단순하면서도 강력한 명령문을 구축할 수 있습니다. Subquery는 query가 알 수 없는 매개 값을 가진 검색 조건을 기반으로 하는 경우에 유용합니다.

다음은 포함하여 다양한 SQL 절에 subquery를 배치할 수 있습니다.

- WHERE 절
- HAVING 절
- FROM 절

Subquery (inner query) 는 main query (outer query) 전에 한 번 실행됩니다. Subquery 결과는 main query에서 사용됩니다.

단일 행 subquery는 =, >, <, >=, <=, <> 등의 단일 행 연산자를 사용하고, 여러 행 subquery는 IN, ANY, ALL 등의 여러 행 연산자를 사용합니다.

**예:** 급여가 최소 급여와 동일한 사원의 세부 정보를 표시합니다.

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

이 예에서 MIN 그룹 함수는 outer query에 단일 값을 반환합니다.

**참고:** 본 과정에서는 여러 열 subquery 사용법에 대해 설명합니다. 여러 열 subquery는 내부 SELECT 문에서 둘 이상의 열을 반환합니다.

## 데이터 조작 검토

DML(데이터 조작어) 문은 다음과 같은 경우에 실행합니다.

- 테이블에 새 행 추가
- 테이블의 기존 행 수정
- 테이블에서 기존 행 제거

함수	설명
INSERT	테이블에 새 행 추가
UPDATE	테이블의 기존 행 수정
DELETE	테이블에서 기존 행 제거
MERGE	조건부로 테이블에서 행 갱신, 삽입 또는 삭제

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 데이터 조작 검토

데이터베이스에서 데이터를 추가, 갱신 또는 삭제하려는 경우 DML 문을 실행하십시오. 논리적 작업 단위를 형성하는 DML 문의 모음을 트랜잭션이라고 합니다. INSERT 문을 사용하면 테이블에 새 행을 추가할 수 있습니다. 다음 구문을 사용할 경우 한 번에 한 행만 삽입됩니다.

```
INSERT INTO table [(column [, column...])]  
VALUES (value[, value...]);
```

INSERT 문을 사용하여 기존 테이블에서 파생된 값으로 테이블에 행을 추가할 수 있습니다.

VALUES 절 자리에 subquery를 사용합니다. INSERT 절의 열 리스트에 나오는 열 개수 및 해당 데이터 유형은 subquery의 값 개수 및 해당 데이터 유형과 일치해야 합니다.

WHERE 절을 지정하면 UPDATE 문은 특정 행을 수정합니다.

```
UPDATE table  
SET column = value [, column = value, ...]  
[WHERE condition];
```

DELETE 문을 사용하면 기존 행을 제거할 수 있습니다. DELETE 문에 WHERE 절을 지정하여 특정 행을 삭제할 수 있습니다.

```
DELETE [FROM] table  
[WHERE condition];
```

MERGE 문에 대해서는 "대형 데이터 집합 조작" 단원에서 설명합니다.

## 단원 내용

- 과정 목표 및 과정 내용
- 본 과정에 사용된 데이터베이스 스키마와 부록 및 본 과정에서 사용 가능한 개발 환경
- 몇 가지 SQL 기본 개념 검토
- **Oracle Database 11g 설명서 및 추가 자료**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# Oracle Database 11g SQL 설명서

- *Oracle Database New Features Guide 11g Release 2 (11.2)*
- *Oracle Database Reference 11g Release 2 (11.2)*
- *Oracle Database SQL Language Reference 11g Release 2 (11.2)*
- *Oracle Database Concepts 11g Release 2 (11.2)*
- *Oracle Database SQL Developer User's Guide Release 1.2*

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Oracle Database 11g SQL 설명서

Oracle Database 11g Release 2 설명서 라이브러리에 액세스하려면  
<http://www.oracle.com/pls/db112/homepage>로 이동하십시오.



## 추가 자료

새로운 Oracle 11g SQL에 대한 자세한 내용은 다음 자료를 참조하십시오.

- *Oracle Database 11g: New Features eStudies*
- *Oracle by Example series (OBE): Oracle Database 11g*

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 과정 목표
- 본 과정에 사용되는 예제 테이블

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 연습 I: 개요

이 연습에서는 다음 내용을 다룹니다.

- SQL Developer 온라인 자습서 실행
- SQL Developer를 시작하여 새 데이터베이스 연결 생성 및 테이블 탐색
- SQL Worksheet를 사용하여 SQL 문 실행
- SQL 기본 개념 검토

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 I: 개요

이 연습에서는 SQL Developer를 사용하여 SQL 문을 실행합니다.

**참고:** 모든 연습 문제는 SQL Developer를 개발 환경으로 사용합니다. SQL Developer 사용을 권장하지만 본 과정에서 사용할 수 있는 SQL\*Plus 환경을 사용해도 됩니다.



# 1

## 유저 액세스 제어

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 단원을 마치면 다음을 수행할 수 있습니다.**

- 시스템 권한과 객체 권한의 구분
- 테이블에 권한 부여
- 롤 부여
- 권한과 롤 구별

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 특정 객체에 대한 데이터베이스 액세스를 제어하고 서로 다른 레벨의 액세스 권한을 가진 새로운 유저를 추가하는 방법에 대해 배웁니다.

## 단원 내용

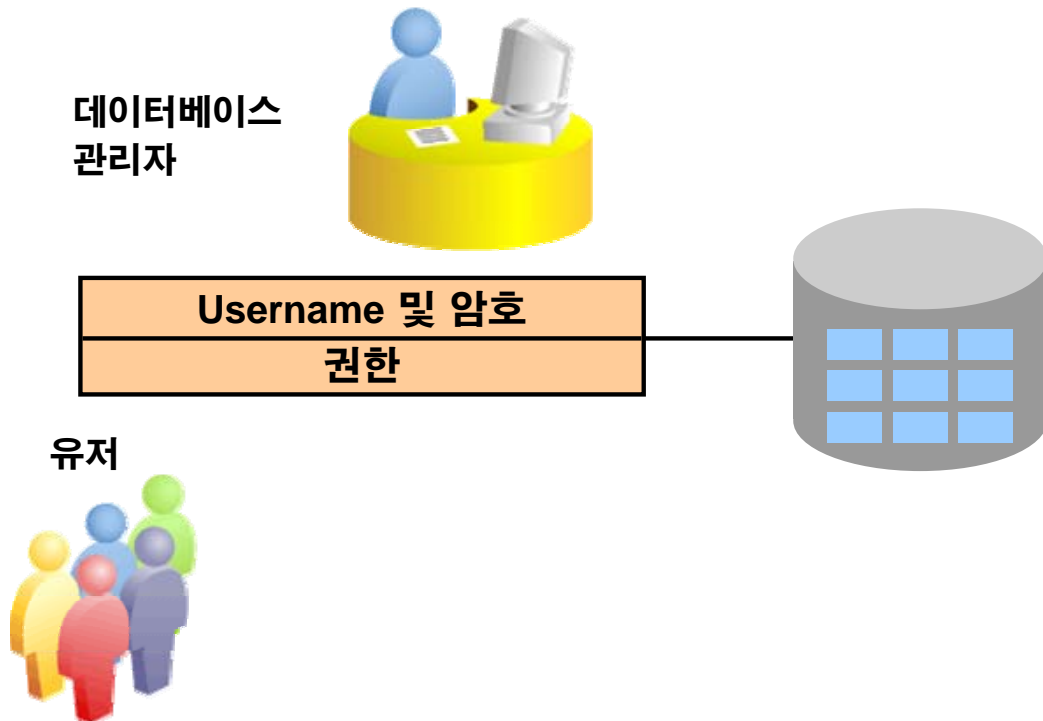
- 시스템 권한
- 롤 생성
- 객체 권한
- 객체 권한 취소

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# 유저 액세스 제어



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 유저 액세스 제어

다중 유저 환경에서는 데이터베이스 액세스 및 사용과 관련한 보안을 유지해야 합니다. Oracle 서버에서는 다음과 같은 데이터베이스 보안 기능을 사용할 수 있습니다.

- 데이터베이스 액세스 제어
- 데이터베이스의 특정 객체에 대한 액세스 권한 부여
- Oracle 데이터 디렉토리를 통한 권한 부여 및 수신 확인

데이터베이스 보안은 시스템 보안과 데이터 보안이라는 두 가지 범주로 분류할 수 있습니다. 시스템 보안에는 username과 암호, 유저에게 할당된 디스크 공간, 유저가 수행할 수 있는 시스템 작업 등과 같은 시스템 레벨의 데이터베이스 액세스 및 사용과 관련된 사항이 포함됩니다. 데이터베이스 보안에는 데이터베이스 객체의 액세스 및 사용, 객체에 대해 해당 유저가 수행할 수 있는 작업과 관련된 사항이 포함됩니다.



# 권한

- **데이터베이스 보안:**
  - 시스템 보안
  - 데이터 보안
- **시스템 권한:** 데이터베이스 내에서 특정 작업 수행
- **객체 권한:** 데이터베이스 객체의 내용 조작
- **스키마:** 테이블, 뷰 및 시퀀스와 같은 객체들의 모음

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 권한

권한이란 특정 SQL 문을 실행할 수 있는 권리입니다. DBA(데이터베이스 관리자)는 유저를 생성하고 유저에게 데이터베이스 및 해당 객체에 대한 액세스 권한을 부여할 수 있는 고급 레벨의 유저입니다. 데이터베이스에 액세스하려면 *시스템 권한*이 필요하며, 데이터베이스의 객체 내용을 조작하려면 *객체 권한*이 필요합니다. 또한 다른 유저나 롤(관련 권한들로 이루어진 명명된 그룹)에 추가 권한을 부여할 수 있는 권한을 부여받을 수도 있습니다.

## 스키마

*스키마*는 테이블, 뷰 및 시퀀스와 같은 객체들의 모음입니다. 스키마의 소유자는 데이터베이스 유저이며 해당 유저와 이름이 동일합니다.

시스템 권한은 특정 작업을 수행하거나 특정 유형의 스키마 객체에 대한 작업을 수행할 수 있는 권리입니다. 객체 권한은 특정 스키마 객체에 대해 특정 작업을 수행할 수 있는 권한을 유저에게 제공합니다.

자세한 내용은 *Oracle Database 2 Day DBA 11g Release 2 (11.2)* 참조 설명서를 참조하십시오.

## 시스템 권한

- 100개 이상의 권한을 사용할 수 있습니다.
- 데이터베이스 관리자는 다음과 같은 작업을 위해 높은 레벨의 시스템 권한을 가집니다.
  - 새 유저 생성
  - 유저 제거
  - 테이블 제거
  - 테이블 백업

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 시스템 권한

100개 이상의 특정 시스템 권한을 유저와 롤에 대해 지정할 수 있습니다. 시스템 권한은 일반적으로 DBA (데이터베이스 관리자)에 의해 제공됩니다.

#### 일반적인 DBA 권한

시스템 권한	인증된 작업
CREATE USER	피부여자가 다른 Oracle 유저를 생성할 수 있습니다.
DROP USER	피부여자가 다른 유저를 삭제할 수 있습니다.
DROP ANY TABLE	피부여자가 임의 스키마에서 테이블을 삭제할 수 있습니다.
BACKUP ANY TABLE	피부여자가 Export 유틸리티를 사용하여 임의 스키마에서 테이블을 백업할 수 있습니다.
SELECT ANY TABLE	피부여자가 임의 스키마에서 테이블, 뷰 또는 Materialized View를 query할 수 있습니다.
CREATE ANY TABLE	피부여자가 임의 스키마에 테이블을 생성할 수 있습니다.

# 유저 생성

DBA는 CREATE USER 문을 사용하여 유저를 생성합니다.

```
CREATE USER user  
IDENTIFIED BY password;
```

```
CREATE USER demo  
IDENTIFIED BY demo;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 유저 생성

DBA는 CREATE USER 문을 실행하여 유저를 생성합니다. 이때 유저는 아무 권한도 없는 상태입니다. 그런 다음 DBA는 이러한 유저에게 권한을 부여할 수 있습니다. 권한에 따라 유저가 데이터베이스 레벨에서 수행할 수 있는 작업이 결정됩니다.

슬라이드는 유저를 생성하는 요약 구문을 보여줍니다.

이 구문에서 다음이 적용됩니다.

*user*                    생성할 유저의 이름입니다.

*password*            유저가 로그인할 때 사용할 암호를 지정합니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

**참고:** Oracle Database 11g부터는 암호의 대소문자를 구분합니다.

## 유저 시스템 권한

- 유저를 생성한 후 DBA는 해당 유저에게 특정 시스템 권한을 부여할 수 있습니다.

```
GRANT privilege [, privilege...]  
TO user [, user/ role, PUBLIC...];
```

- 예를 들어 응용 프로그램 개발자는 다음과 같은 시스템 권한을 가질 수 있습니다.
  - CREATE SESSION
  - CREATE TABLE
  - CREATE SEQUENCE
  - CREATE VIEW
  - CREATE PROCEDURE

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 일반적인 유저 권한

유저를 생성한 후 DBA는 해당 유저에게 권한을 할당할 수 있습니다.

시스템 권한	인증된 작업
CREATE SESSION	데이터베이스에 연결합니다.
CREATE TABLE	유저의 스키마에 테이블을 생성합니다.
CREATE SEQUENCE	유저의 스키마에 시퀀스를 생성합니다.
CREATE VIEW	유저의 스키마에 뷰를 생성합니다.
CREATE PROCEDURE	유저의 스키마에 내장 프로시저, 함수 또는 패키지를 생성합니다.

이 구문에서 다음이 적용됩니다.

*privilege*

부여할 시스템 권한입니다.

*user* /*role*/PUBLIC

유저 이름, 롤 이름 또는 PUBLIC (모든 유저가 권한을 부여받도록 지정함)입니다.

**참고:** 현재의 시스템 권한은 SESSION\_PRIVS 디서너리 뷰에서 찾을 수 있습니다. 데이터 디서너리는 Oracle 서버에서 생성하여 유지 관리하는 테이블과 뷰의 모음으로, 데이터베이스에 대한 정보를 포함하고 있습니다.

# 시스템 권한 부여

DBA는 유저에게 특정 시스템 권한을 부여할 수 있습니다.

```
GRANT  create session, create table,  
        create sequence, create view  
TO      demo;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 시스템 권한 부여

DBA는 GRANT 문을 사용하여 유저에게 시스템 권한을 할당합니다. 유저는 권한을 부여받은 후 해당 권한을 즉시 사용할 수 있습니다.

위 슬라이드 예제에서는 유저 demo에게 세션, 테이블, 시퀀스 및 뷰를 생성할 수 있는 권한이 할당되었습니다.

## 단원 내용

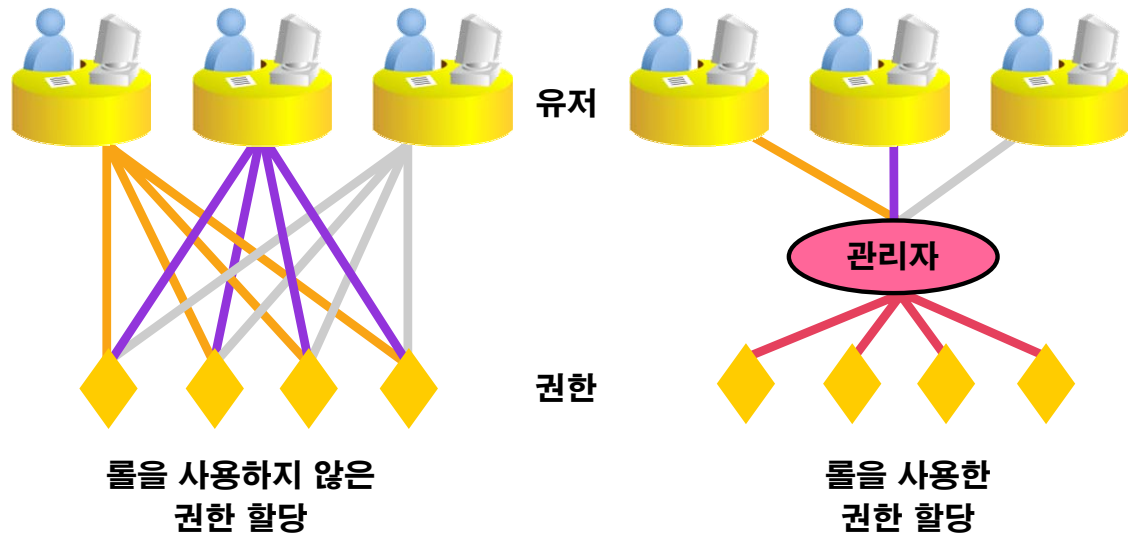
- 시스템 권한
- **롤 생성**
- 객체 권한
- 객체 권한 취소

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# 롤이란?



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 롤이란?

롤은 유저에게 부여할 수 있는 관련 권한들로 이루어진 명명된 그룹입니다. 롤을 사용하면 권한을 쉽게 취소하거나 유지 관리할 수 있습니다.

한 유저에게 여러 롤에 대한 액세스 권한을 부여하거나 동일 롤에 여러 유저를 할당할 수도 있습니다. 일반적으로 롤은 데이터베이스 응용 프로그램에 대해 생성됩니다.

### 롤 생성 및 할당

먼저 DBA가 롤을 생성해야 합니다. 그런 다음 DBA는 이 롤에 권한을 할당하고 유저에게 롤을 할당할 수 있습니다.

### 구문

```
CREATE ROLE role;
```

이 구문에서 다음이 적용됩니다.

*role* 생성할 롤의 이름입니다.

롤을 생성한 후 DBA는 GRANT 문을 사용하여 해당 롤에 권한을 할당하는 것은 물론 롤에 유저를 할당할 수도 있습니다. 롤은 스키마 객체가 아니므로 아무 유저나 롤에 권한을 추가할 수 있습니다.

## 롤 생성 및 롤에 권한 부여

- **롤 생성:**

```
CREATE ROLE manager;
```

- **롤에 권한 부여:**

```
GRANT create table, create view  
TO manager;
```

- **유저에게 롤 부여:**

```
GRANT manager TO alice;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 롤 생성

위 슬라이드 예제에서는 `manager` 롤을 생성한 다음 관리자가 테이블과 뷰를 생성할 수 있도록 설정되었습니다. 그런 다음 유저 `alice`에게 관리자 롤을 부여합니다. 그 결과 `alice`는 테이블과 뷰를 생성할 수 있습니다.

유저에게 여러 롤이 부여된 경우 해당 롤과 관련된 모든 권한을 갖게 됩니다.



## 암호 변경

- DBA는 유저 계정을 생성하고 암호를 초기화합니다.
- ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.

```
ALTER USER demo  
IDENTIFIED BY employ;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 암호 변경

DBA는 모든 유저의 계정을 생성하고 암호를 초기화합니다. ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.

위 슬라이드 예제에서는 유저 demo가 ALTER USER 문을 사용하여 암호를 변경하는 것을 보여줍니다.

#### 구문

```
ALTER USER user IDENTIFIED BY password;
```

이 구문에서 다음이 적용됩니다.

*user*                                      유저의 이름입니다.

*password*                                새 암호를 지정합니다.

이 명령문으로 암호를 변경할 수 있지만 그 외에도 여러 옵션이 있습니다. 다른 옵션을 변경하려면 ALTER USER 권한이 있어야 합니다.

자세한 내용은 *Oracle Database 11g SQL Reference* 설명서를 참조하십시오.

**참고:** SQL\*Plus에는 유저가 로그인할 때 유저의 암호를 변경하는 데 사용할 수 있는 PASSWORD 명령(PASSW)이 있습니다. 이 명령은 SQL Developer에서는 사용할 수 없습니다.

## 단원 내용

- 시스템 권한
- 롤 생성
- **객체 권한**
- 객체 권한 취소

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

## 객체 권한

객체 권한	테이블	뷰	시퀀스
ALTER	✓		✓
DELETE	✓	✓	
INDEX	✓		
INSERT	✓	✓	
REFERENCES	✓		
SELECT	✓	✓	✓
UPDATE	✓	✓	

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 객체 권한

객체 권한은 특정 테이블, 뷰, 시퀀스 및 프로시저에 대해 특정 작업을 수행할 수 있는 권한입니다. 각 객체에는 부여할 수 있는 일련의 권한이 있습니다. 위 슬라이드의 표에는 다양한 객체에 대한 권한이 나열되어 있습니다. 시퀀스에는 SELECT 및 ALTER 권한만 적용됩니다. UPDATE, REFERENCES 및 INSERT는 갱신 가능한 열의 부분 집합을 지정하여 제한할 수 있습니다.

SELECT 권한은 열의 부분 집합으로 뷰를 생성하고 해당 뷰에만 SELECT 권한을 부여하여 제한할 수 있습니다. 동의어에 부여된 권한은 해당 동의어가 참조하는 기본 테이블에 대한 권한으로 변환됩니다.

**참고:** REFERENCES 권한을 사용하면 다른 사용자가 자신의 테이블을 참조하는 FOREIGN KEY 제약 조건을 생성할 수 있습니다.

## 객체 권한

- 객체 권한은 객체마다 다양합니다.
- 객체 소유자는 해당 객체에 대한 모든 권한을 소유합니다.
- 소유자는 자신의 객체에 대한 특정 권한을 부여할 수 있습니다.

```
GRANT      object_priv [(columns)]  
ON         object  
TO         {user|role|PUBLIC}  
[WITH GRANT OPTION];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 객체 권한 부여

스키마 객체의 유형에 따라 다양한 객체 권한을 사용할 수 있습니다. 유저는 자신의 스키마에 포함된 스키마 객체에 대해 자동으로 모든 객체 권한을 가집니다. 또한 유저는 자신이 소유하는 스키마 객체에 대한 객체 권한을 다른 유저 또는 롤에 부여할 수 있습니다. 권한을 부여할 때 WITH GRANT OPTION을 사용하면 권한 피부여자가 다른 유저에게 객체 권한을 부여할 수 있습니다. 이 옵션을 사용하지 않으면 권한 피부여자가 해당 권한을 사용할 수는 있지만 다른 유저에게 부여할 수 없습니다.

이 구문에서 다음이 적용됩니다.

<i>object_priv</i>	부여할 객체 권한입니다.
ALL	모든 객체 권한을 지정합니다.
<i>columns</i>	권한이 부여되는 테이블 또는 뷰의 열을 지정합니다.
ON <i>object</i>	권한이 부여되는 객체입니다.
TO	권한이 부여되는 대상을 식별합니다.
PUBLIC	모든 유저에게 객체 권한을 부여합니다.
WITH GRANT OPTION	권한을 부여받은 사람이 다른 유저 및 롤에 객체 권한을 부여할 수 있습니다.

참고: 이 구문에서 *schema*는 소유자의 이름과 동일합니다.

## 객체 권한 부여

- **EMPLOYEES 테이블에 대한 query 권한을 부여합니다.**

```
GRANT  select
ON      employees
TO      demo;
```

- **특정 열을 갱신할 수 있는 권한을 유저와 롤에 부여합니다.**

```
GRANT  update (department_name, location_id)
ON      departments
TO      demo, manager;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 지침

- 객체에 권한을 부여하려면 해당 객체가 자신의 스키마에 있거나 WITH GRANT OPTION 객체 권한을 부여받아야 합니다.
- 객체 소유자는 객체에 대한 객체 권한을 데이터베이스의 다른 유저나 롤에 부여할 수 있습니다.
- 해당 객체에 대한 모든 객체 권한을 자동으로 획득합니다.

슬라이드의 첫번째 예제에서는 EMPLOYEES 테이블 query 권한을 유저 demo에게 부여합니다. 두번째 예제에서는 DEPARTMENTS 테이블의 특정 열에 대한 UPDATE 권한을 demo와 manager 롤에 부여합니다.

예를 들어, 스키마가 oraxx이고 유저 demo가 SELECT 문을 사용하여 EMPLOYEES 테이블에서 데이터를 가져오려는 경우에 사용해야 하는 구문은 다음과 같습니다.

```
SELECT * FROM oraxx.employees;
```

또는 유저 demo가 테이블에 대해 동의어를 생성하고 동의어에서 SELECT 문을 실행할 수 있습니다.

```
CREATE SYNONYM emp FOR oraxx.employees;
SELECT * FROM emp;
```

**참고:** 일반적으로 DBA는 시스템 권한을 할당하며 객체 소유자는 객체 권한을 부여할 수 있습니다.

## 권한 전달

- **유저에게 권한을 전달할 수 있는 자격을 부여합니다.**

```
GRANT  select, insert
ON      departments
TO      demo
WITH    GRANT OPTION;
```

- **시스템의 모든 유저가 Alice의 DEPARTMENTS 테이블에서 데이터를 query할 수 있도록 허용합니다.**

```
GRANT  select
ON      alice.departments
TO      PUBLIC;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 권한 전달

#### WITH GRANT OPTION 키워드

WITH GRANT OPTION 절을 사용하여 부여된 권한은 권한을 부여받은 사람에 의해 다른 유저와 롤에 전달될 수 있습니다. 권한 부여자의 권한이 취소되면 WITH GRANT OPTION 절을 사용하여 부여된 객체 권한도 취소됩니다.

위 슬라이드 예제에서 유저 demo는 테이블을 query하고 테이블에 행을 추가할 수 있는 권한으로 DEPARTMENTS 테이블에 액세스할 수 있습니다. 또한 user1이 이러한 권한을 다른 유저에게 부여할 수 있음을 보여줍니다.

#### PUBLIC 키워드

테이블 소유자는 PUBLIC 키워드를 사용하여 모든 유저에게 액세스 권한을 부여할 수 있습니다. 두번째 예제에서는 시스템의 모든 유저가 Alice의 DEPARTMENTS 테이블에서 데이터를 query할 수 있도록 허용합니다.

## 부여된 권한 확인

데이터 디렉터리 뷰	설명
ROLE_SYS_PRIVS	롤에 부여되는 시스템 권한
ROLE_TAB_PRIVS	롤에 부여되는 테이블 권한
USER_ROLE_PRIVS	유저가 액세스할 수 있는 롤
USER_SYS_PRIVS	유저에게 부여되는 시스템 권한
USER_TAB_PRIVS_MADE	유저의 객체에 대해 부여되는 객체 권한
USER_TAB_PRIVS_RECD	유저에게 부여되는 객체 권한
USER_COL_PRIVS_MADE	유저 객체의 열에 대해 부여되는 객체 권한
USER_COL_PRIVS_RECD	특정 열에 대해 유저에게 부여되는 객체 권한

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 부여된 권한 확인

Oracle 서버에서는 DELETE 권한이 없는 테이블에서 행을 삭제하는 것처럼 승인되지 않은 작업은 수행할 수 없습니다.

다음 중 하나를 수행한 경우 "Table or view does not exist"라는 Oracle 서버 오류 메시지가 나타납니다.

- 존재하지 않는 테이블 또는 뷰의 이름을 지정했습니다.
- 적절한 권한이 없는 테이블 또는 뷰에 대해 작업을 수행하려 했습니다.

데이터 디렉터리는 테이블과 뷰로 구성되며 데이터베이스에 대한 정보를 포함합니다. 자신이 소유한 권한은 데이터 디렉터리에서 확인할 수 있습니다. 위 슬라이드의 표에는 다양한 데이터 디렉터리 테이블에 대한 설명이 있습니다.

데이터 디렉터리 뷰에 대한 자세한 내용은 "데이터 디렉터리 뷰를 사용하여 객체 관리" 단원을 참조하십시오.

**참고:** ALL\_TAB\_PRIVS\_MADE 디렉터리 뷰는 유저가 부여한 모든 객체 권한 또는 유저가 소유한 객체에 대한 모든 객체 권한에 대해 설명합니다.

## 단원 내용

- 시스템 권한
- 롤 생성
- 객체 권한
- 객체 권한 취소

ORACLE

Copyright © 2009, Oracle. All rights reserved.



## 객체 권한 취소

- REVOKE 문을 사용하여 다른 유저에게 부여된 권한을 취소할 수 있습니다.
- WITH GRANT OPTION 절을 통해 다른 유저에게 부여된 권한을 취소할 수도 있습니다.

```
REVOKE {privilege [, privilege...]|ALL}  
ON      object  
FROM    {user[, user...]|role|PUBLIC}  
[CASCADE CONSTRAINTS];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 객체 권한 취소

REVOKE 문을 사용하여 다른 유저에게 부여된 권한을 제거할 수 있습니다. REVOKE 문을 사용하면 자신이 직접 이름을 지정한 유저와 권한이 취소된 유저에게 권한을 부여받은 다른 유저로부터 지정한 권한을 취소할 수 있습니다.

이 구문에서 다음이 적용됩니다.

CASCADE                      REFERENCES 권한을 통해 CONSTRAINTS 객체에 대해 지정된 참조 무결성 제약 조건을 제거하는 데 필요합니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

**참고:** 퇴사한 유저의 권한을 취소한 경우 이 유저가 다른 유저에게 부여한 모든 권한을 다시 부여해야 합니다. 유저 계정에서 권한을 취소하지 않고 유저 계정을 삭제하면 해당 유저가 다른 유저에게 부여한 시스템 권한이 이 작업의 영향을 받지 않습니다.

## 객체 권한 취소

DEPARTMENTS 테이블에 대해 유저 demo에게 부여된 SELECT 및 INSERT 권한을 취소합니다.

```
REVOKE select, insert
ON      departments
FROM    demo;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 객체 권한 취소(계속)

위 슬라이드 예제에서는 DEPARTMENTS 테이블에서 유저 demo에게 부여된 SELECT 및 INSERT 권한을 취소합니다.

**참고:** WITH GRANT OPTION 절을 사용하여 권한을 부여받은 유저는 다시 WITH GRANT OPTION 절을 사용하여 해당 권한을 다른 유저에게 부여할 수 있습니다. 따라서 권한을 부여받은 사람의 연결 관계가 길게 이어질 수도 있지만, 순환적인 권한 부여 (조상 권한 부여자에게 권한 부여) 관계는 허용되지 않습니다. 소유자가 다른 유저에게 권한을 부여한 유저로부터 권한을 취소하면 부여된 모든 권한이 연쇄적으로 취소됩니다.

예를 들어, 유저 A가 특정 테이블에 대해 WITH GRANT OPTION 절을 포함한 SELECT 권한을 유저 B에게 부여하면 유저 B는 WITH GRANT OPTION 절을 사용하여 SELECT 권한을 유저 C에게 부여할 수 있고 유저 C는 유저 D에게 SELECT 권한을 부여할 수 있습니다. 유저 A가 유저 B에게서 권한을 취소하면 유저 C 및 D에게 부여된 권한도 취소됩니다.

## 퀴즈

다음 설명 중 옳은 것은 무엇입니까?

1. 유저는 객체를 생성한 후에 GRANT 문을 사용하여 사용 가능한 객체 권한을 다른 유저에게 전달할 수 있습니다.
2. 유저는 CREATE ROLE 문으로 롤을 생성하여 다른 유저에게 시스템 또는 객체 권한 모음을 전달할 수 있습니다.
3. 유저는 자신의 암호를 변경할 수 있습니다.
4. 유저는 자신과 자신의 객체에 부여된 권한을 볼 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 3, 4

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 시스템 권한과 객체 권한의 구분
- 테이블에 권한 부여
- 롤 부여
- 권한과 롤 구별

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

DBA는 유저에게 권한을 할당하여 유저에 대한 초기 데이터베이스 보안을 설정합니다.

- DBA는 유저를 생성하고 암호를 지정합니다. DBA는 또한 유저에 대한 초기 시스템 권한을 설정해야 합니다.
- 유저는 객체를 생성한 후에 GRANT 문을 사용하여 사용 가능한 객체 권한을 다른 유저 또는 모든 유저에게 전달할 수 있습니다.
- DBA는 CREATE ROLE 문으로 롤을 생성하여 여러 유저에게 시스템 또는 객체 권한 모음을 전달할 수 있습니다. 롤을 사용하면 권한 부여 및 취소를 보다 쉽게 관리할 수 있습니다.
- 유저는 ALTER USER 문을 사용하여 암호를 변경할 수 있습니다.
- REVOKE 문을 사용하면 유저로부터 권한을 취소할 수 있습니다.
- 유저는 데이터 덱서너리 뷰에서 자신에게 부여된 권한과 자신의 객체에 부여된 권한을 확인할 수 있습니다.

## 연습 1: 개요

이 연습에서는 다음 내용을 다룹니다.

- 자신의 테이블에 대한 권한을 다른 유저에게 부여
- 자신에게 부여된 권한을 통해 다른 유저의 테이블 수정

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 1: 개요

이 연습에서는 다른 수강생과 팀을 구성하여 데이터베이스 객체에 대한 액세스 제어 과정을 연습합니다.



# 2

## 스키마 객체 관리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 제약 조건 추가
- 인덱스 생성
- CREATE TABLE 문을 사용하여 인덱스 생성
- 함수 기반(Function-based) 인덱스 생성
- 열 삭제 및 UNUSED 열로 설정
- FLASHBACK 작업 수행
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원은 인덱스 및 제약 조건 생성과 기존 객체 변경에 대한 정보를 제공합니다. 또한 External Table 및 PRIMARY KEY 제약 조건을 생성할 때 인덱스 이름을 지정하는 규칙에 대해 설명합니다.



## 단원 내용

- **ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제**
- **제약 조건 관리:**
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- **인덱스 생성:**
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- **Flashback 작업 수행**
- **임시 테이블(Temporary Table) 생성 및 사용**
- **External Table 생성 및 사용**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## ALTER TABLE 문

ALTER TABLE 문을 사용하여 다음을 수행합니다.

- 새 열 추가
- 기존 열 수정
- 새 열의 기본값 정의
- 열 삭제

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE 문

테이블을 생성한 후에 열을 생략했거나 열 정의를 변경하거나 열을 제거하기 위해 테이블 구조를 변경해야 할 수 있습니다. 이 작업은 ALTER TABLE 문을 사용하여 수행할 수 있습니다.

## ALTER TABLE 문

ALTER TABLE 문을 사용하여 열을 추가, 수정 또는 삭제합니다.

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
              [, column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
              [, column datatype]...);
```

```
ALTER TABLE table
DROP (column [, column] ...);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE 문(계속)

ALTER TABLE 문을 사용하여 테이블에서 열을 추가, 수정 및 삭제할 수 있습니다.

이 구문에서 다음이 적용됩니다.

<i>table</i>	테이블의 이름입니다.
ADD MODIFY DROP	수정의 유형입니다.
<i>column</i>	열 이름입니다.
<i>datatype</i>	열의 데이터 유형과 길이입니다.
DEFAULT <i>expr</i>	열 기본값을 지정합니다.

## 열 추가

- ADD 절을 사용하여 열을 추가합니다.

```
ALTER TABLE dept80
ADD      (job_id VARCHAR2(9));
```

```
ALTER TABLE dept80 succeeded.
```

- 새 열은 마지막 열이 됩니다.

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	145	Russell	14000	01-OCT-96	(null)
2	146	Partners	13500	05-JAN-97	(null)
3	147	Errazuriz	12000	10-MAR-97	(null)
4	148	Cambraut	11000	15-OCT-99	(null)
5	149	Zlotkey	10500	29-JAN-00	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 열 추가를 위한 지침

- 열을 추가하거나 수정할 수 있습니다.
- 열을 표시할 위치를 지정할 수 없습니다. 새 열은 마지막 열이 됩니다.

위 슬라이드 예제에서는 DEPT80 테이블에 JOB\_ID라는 열을 추가합니다. JOB\_ID 열은 테이블의 마지막 열이 됩니다.

**참고:** 열을 추가할 때 테이블에 이미 행이 포함되어 있으면 새 열의 초기값은 모든 행에 대해 널이거나 기본값을 사용합니다. 다른 열에 데이터를 포함하고 있는 테이블은 기본값을 지정한 경우에만 필수 NOT NULL 열을 추가할 수 있습니다. 빈 테이블에는 기본값을 지정하지 않고도 NOT NULL 열을 추가할 수 있습니다.

## 열 수정

- 열의 데이터 유형, 크기 및 기본값을 변경할 수 있습니다.

```
ALTER TABLE dept80
MODIFY      (last_name VARCHAR2(30));
```

```
ALTER TABLE dept80 succeeded.
```

- 기본값 변경은 이후에 테이블에 삽입하는 항목에만 적용됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 열 수정

ALTER TABLE 문을 MODIFY 절과 함께 사용하여 열 정의를 수정할 수 있습니다. 열 수정 작업에는 열의 데이터 유형, 크기 및 기본값에 대한 변경이 포함될 수 있습니다.

#### 지침

- 숫자 열의 너비나 전체 자릿수를 늘릴 수 있습니다.
- 문자 열의 너비를 늘릴 수 있습니다.
- 다음과 같은 경우에는 열의 너비를 줄일 수 있습니다.
  - 열이 널 값만 포함하는 경우
  - 테이블에 행이 없는 경우
  - 열 너비의 감소가 해당 열의 기존 값보다 작지 않은 경우
- 열에 널 값만 있는 경우 데이터 유형을 변경할 수 있습니다. 단, 예외적으로 CHAR에서 VARCHAR2로의 변환은 열에 데이터가 있어도 수행할 수 있습니다.
- 열에 널 값이 포함된 경우나 크기를 변경하지 않은 경우에만 CHAR 열을 VARCHAR2 데이터 유형으로 변환하거나 VARCHAR2 열을 CHAR 데이터 유형으로 변환할 수 있습니다.
- 열의 기본값 변경은 이후에 테이블에 삽입하는 항목에만 적용됩니다.

## 열 삭제

**DROP COLUMN 절을 사용하여 테이블에서 더 이상 필요 없는 열을 삭제할 수 있습니다.**

```
ALTER TABLE dept80
DROP COLUMN job_id;
```

```
ALTER TABLE dept80 succeeded.
```

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	145 Russell	14000	01-OCT-96
2	146 Partners	13500	05-JAN-97
3	147 Errazuriz	12000	10-MAR-97
4	148 Cambrault	11000	15-OCT-99
5	149 Zlotkey	10500	29-JAN-00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 열 삭제

ALTER TABLE 문을 DROP COLUMN 절과 함께 사용하여 테이블에서 열을 삭제할 수 있습니다.

#### 지침

- 열에는 데이터가 포함되어 있거나 포함되어 있지 않습니다.
- ALTER TABLE DROP COLUMN 문을 사용하면 한 번에 한 개의 열만 삭제할 수 있습니다.
- 테이블이 변경된 후에는 최소 하나 이상의 열이 테이블에 남아 있어야 합니다.
- 열이 삭제된 후에는 recovery할 수 없습니다.
- CASCADE 옵션이 추가되지 않은 경우 열이 제약 조건의 일부이거나 인덱스 키의 일부라면 삭제할 수 없습니다.
- 많은 수의 값을 가진 열을 삭제하는 경우 약간의 시간이 걸릴 수 있습니다. 이 경우 확장된 잠금을 피하려면 시스템에 유저의 수가 적을 때 열을 unused로 설정하고 열을 삭제하는 것이 좋습니다.

**참고:** Partition 테이블의 Partitioning key의 일부를 형성하는 열이나 인덱스 구성 테이블(index-organized table)의 PRIMARY KEY의 일부를 형성하는 열과 같은 특정 열은 삭제할 수 없습니다. 인덱스 구성 테이블(Index-organized Table)과 Partition 테이블에 대한 자세한 내용은 *Oracle Database Concepts* 및 *Oracle Database Administrator's Guide*를 참조하십시오.

## SET UNUSED 옵션

- SET UNUSED 옵션을 사용하여 하나 이상의 열을 unused로 표시합니다.
- DROP UNUSED COLUMNS 옵션을 사용하여 unused로 표시된 열을 제거할 수 있습니다.

```
ALTER TABLE <table_name>  
SET UNUSED(<column_name> [ , <column_name>]);  
OR  
ALTER TABLE <table_name>  
SET UNUSED COLUMN <column_name> [ , <column_name>];
```

```
ALTER TABLE <table_name>  
DROP UNUSED COLUMNS;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SET UNUSED 옵션

SET UNUSED 옵션은 시스템 자원 사용률이 낮은 경우 열을 삭제할 수 있도록 하나 이상의 열을 unused로 표시합니다. 이 절을 지정해도 대상 열이 테이블의 각 행에서 실제로 제거되는 것은 아닙니다. (즉, 이 열이 사용하는 디스크 공간이 복원되는 것은 아닙니다.) 따라서 DROP 절을 실행한 경우보다 응답 시간이 빨라집니다. Unused 열은 테이블의 행에 열 데이터가 남아 있는 경우에도 삭제된 것으로 간주됩니다. Unused로 표시된 후에는 해당 열에 액세스할 수 없습니다. SELECT \* query는 unused 열에서 데이터를 검색하지 않습니다. 또한 unused로 표시된 열의 이름 및 유형은 DESCRIBE 문이 실행될 때 표시되지 않고 unused 열과 동일한 이름을 가진 새 열을 테이블에 추가할 수 있습니다. SET UNUSED 정보는 USER\_UNUSED\_COL\_TABS 디렉터리 뷰에 저장됩니다.

**참고:** UNUSED 열 설정 지침은 열 삭제 지침과 유사합니다.

## SET UNUSED 옵션(계속)

### DROP UNUSED COLUMNS 옵션

DROP UNUSED COLUMNS는 현재 unused로 표시된 모든 열을 테이블에서 제거합니다.

테이블의 사용되지 않은 열에서 추가 디스크 공간을 회수하려면 이 명령문을 사용할 수 있습니다.

사용되지 않은 열이 테이블에 없는 경우 이 명령문은 아무 오류도 반환하지 않습니다.

```
ALTER TABLE dept80  
SET UNUSED (last_name);
```

```
ALTER TABLE succeeded
```

```
ALTER TABLE dept80  
DROP UNUSED COLUMNS;
```

```
ALTER TABLE succeeded
```



## 단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 제약 조건 구문 추가

**ALTER TABLE 문을 사용하여 다음을 수행합니다.**

- **제약 조건 추가 또는 삭제. 제약 조건의 구조는 수정하지 않음**
- **제약 조건 활성화 또는 비활성화**
- **MODIFY 절을 사용하여 NOT NULL 제약 조건 추가**

```
ALTER TABLE <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 추가

ALTER TABLE 문을 ADD 절과 함께 사용하여 기존 테이블에 제약 조건을 추가할 수 있습니다. 이 구문에서 다음이 적용됩니다.

*table*           테이블의 이름입니다.  
*constraint*    제약 조건의 이름입니다.  
*type*           제약 조건 유형입니다.  
*column*        제약 조건의 영향을 받는 열의 이름입니다.

제약 조건 이름 구문은 지정하면 좋지만 선택 사항입니다. 제약 조건의 이름을 지정하지 않으면 시스템이 제약 조건 이름을 생성합니다.

### 지침

- 제약 조건을 추가, 삭제, 활성화 또는 비활성화할 수 있지만 제약 조건의 구조를 수정할 수는 없습니다.
- ALTER TABLE 문의 MODIFY 절을 사용하여 기존 열에 NOT NULL 제약 조건을 추가할 수 있습니다.

**참고:** 테이블이 비어 있는 경우 또는 열의 모든 행에 값이 있는 경우에만 NOT NULL 열을 정의할 수 있습니다.

## 제약 조건 추가

EMP2 테이블에 FOREIGN KEY 제약 조건을 추가하면 관리자가 이미 EMP2 테이블에 유효한 사원으로 존재해야 함을 나타냅니다.

```
ALTER TABLE emp2  
MODIFY employee_id PRIMARY KEY;
```

```
ALTER TABLE emp2 succeeded.
```

```
ALTER TABLE emp2  
ADD CONSTRAINT emp_mgr_fk  
FOREIGN KEY(manager_id)  
REFERENCES emp2(employee_id);
```

```
ALTER TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 추가(계속)

슬라이드의 첫번째 예제에서는 EMP2 테이블을 수정하여 EMPLOYEE\_ID 열에 PRIMARY KEY 제약 조건을 추가합니다. 제약 조건의 이름을 제공하지 않았으므로 Oracle 서버가 자동으로 제약 조건의 이름을 지정합니다. 슬라이드의 두번째 예제에서는 EMP2 테이블에 FOREIGN KEY 제약 조건을 생성합니다. 이 제약 조건은 관리자가 EMP2 테이블에서 유효한 사원으로 존재하도록 보장합니다.

## ON DELETE 절

- 상위 키가 삭제될 때 하위 행을 삭제하려면 ON DELETE CASCADE 절을 사용합니다.

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments(department_id) ON DELETE CASCADE;
```

```
ALTER TABLE Emp2 succeeded.
```

- 상위 키가 삭제될 때 하위 행을 널로 설정하려면 ON DELETE SET NULL 절을 사용합니다.

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments(department_id) ON DELETE SET NULL;
```

```
ALTER TABLE Emp2 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ON DELETE

ON DELETE 절을 사용하면 참조된 Primary Key 또는 Unique Key 값을 제거할 경우 오라클 데이터베이스에서 참조 무결성을 처리하는 방법을 결정할 수 있습니다.

#### ON DELETE CASCADE

ON DELETE CASCADE 작업은 하위 테이블에서 참조되는 상위 키 데이터가 삭제되도록 합니다(갱신되지는 않음). 상위 키의 데이터가 삭제되면 해당 상위 키 값에 종속된 하위 테이블의 모든 행도 삭제됩니다. 이 참조 작업을 지정하려면 FOREIGN KEY 제약 조건의 정의에 ON DELETE CASCADE 옵션을 포함시키십시오.

#### ON DELETE SET NULL

상위 키의 데이터가 삭제되면 ON DELETE SET NULL 작업으로 인해 해당 상위 키 값에 종속된 하위 테이블의 모든 행이 널로 변환됩니다.

이 절을 생략할 경우 하위 테이블에 종속 행이 있는 상위 테이블의 참조된 키 값을 삭제할 수 없습니다.

## 제약 조건 지연

제약 조건은 다음 속성을 가질 수 있습니다.

- DEFERRABLE 또는 NOT DEFERRABLE
- INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE

```
ALTER TABLE dept2
ADD CONSTRAINT dept2_id_pk
PRIMARY KEY (department_id)
DEFERRABLE INITIALLY DEFERRED
```

생성 시 제약 조건 지연

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE
```

특정 제약 조건 속성 변경

```
ALTER SESSION
SET CONSTRAINTS= IMMEDIATE
```

세션의 모든 제약 조건 변경

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 지연

트랜잭션이 끝날 때까지 제약 조건의 유효성 검사를 지연할 수 있습니다. 시스템에서 제약 조건이 충족되는지 검사하지 않을 경우 COMMIT 문이 실행될 때까지 제약 조건이 지연됩니다. 지연된 제약 조건을 위반할 경우 데이터베이스에서 오류를 반환하며 트랜잭션이 커밋되지 않고 롤백됩니다. 제약 조건이 deferred가 아니라 immediate인 경우 각 명령문의 끝에서 제약 조건이 검사됩니다. 제약 조건을 위반할 경우 명령문이 즉시 롤백됩니다. 제약 조건으로 인해 DELETE CASCADE와 같은 작업이 발생하면, 제약 조건이 deferred인지 또는 immediate인지에 관계없이 항상 해당 작업을 발생시킨 명령문의 일부로 작업이 실행됩니다. SET CONSTRAINTS 문을 사용하여 특정 트랜잭션에 대해 deferrable 제약 조건을 각 DML(데이터 조작어) 문 이후에 검사할지 또는 트랜잭션이 커밋될 때 검사할지 지정합니다. deferrable 제약 조건을 생성하려면 해당 제약 조건에 대한 비고유 인덱스를 생성해야 합니다.

제약 조건을 DEFERRABLE 또는 NOT DEFERRABLE로 정의하거나 INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE로 정의할 수 있습니다. 이러한 속성은 각 제약 조건에 따라 다를 수 있습니다.

**사용 시나리오:** 회사 방침에 따라 부서 번호 40을 45로 변경해야 합니다. DEPARTMENT\_ID 열을 변경하면 이 부서에 할당된 사원에게 영향이 있습니다. 따라서 PRIMARY KEY 및 FOREIGN KEY를 DEFERRABLE 및 INITIALLY DEFERRED로 지정합니다. 부서 정보 및 사원 정보를 모두 갱신하면 커밋 시 모든 행이 검사됩니다.

## INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이

INITIALLY DEFERRED	트랜잭션이 종료될 때까지 제약 조건 검사를 기다립니다.
INITIALLY IMMEDIATE	명령문 실행이 완료되면 제약 조건을 검사합니다.

```
CREATE TABLE emp_new_sal (salary NUMBER
                           CONSTRAINT sal_ck
                           CHECK (salary > 100)
                           DEFERRABLE INITIALLY IMMEDIATE,
                           bonus NUMBER
                           CONSTRAINT bonus_ck
                           CHECK (bonus > 0 )
                           DEFERRABLE INITIALLY DEFERRED );
```

```
create table succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이

Deferrable로 정의된 제약 조건은 INITIALLY DEFERRED 또는 INITIALLY IMMEDIATE로 지정할 수 있습니다. 기본값은 INITIALLY IMMEDIATE 절입니다.

슬라이드 예제 설명:

- sal\_ck 제약 조건이 DEFERRABLE INITIALLY IMMEDIATE로 생성됩니다.
- bonus\_ck 제약 조건이 DEFERRABLE INITIALLY DEFERRED로 생성됩니다.

슬라이드에 표시된 것과 같이 emp\_new\_sal 테이블을 생성한 후에는 테이블에 값을 삽입하고 그 결과를 확인하십시오. sal\_ck 제약 조건과 bonus\_ck 제약 조건을 모두 만족하면 오류 없이 행이 삽입됩니다.

**예제 1:** sal\_ck를 위반하는 행을 삽입합니다. CREATE TABLE 문에 sal\_ck가 initially immediate 제약 조건으로 지정됩니다. 따라서 INSERT 문 직후에 제약 조건이 검사되고 오류가 반환됩니다.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
SQL Error: ORA-02290: check constraint (ORA21.SAL_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"
```

**예제 2:** bonus\_ck를 위반하는 행을 삽입합니다. CREATE TABLE 문에 bonus\_ck가 deferrable 및 initially deferred로 지정됩니다. 따라서 COMMIT을 수행하거나 제약 조건 상태를 다시 immediate로 설정할 때까지 제약 조건이 검사되지 않습니다.

**INITIALLY DEFERRED와 INITIALLY IMMEDIATE의 차이(계속)**

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
1 rows inserted
```

행이 성공적으로 삽입됩니다. 그러나 트랜잭션을 커밋하면 오류가 반환됩니다.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.BONUS_CK) violated
02091. 00000 - "transaction rolled back"
```

계약 조건을 위반했기 때문에 커밋을 실패했습니다. 그러므로 이때 트랜잭션이 데이터베이스에 의해 롤백됩니다.

**예제 3:** 지연 가능한 모든 제약 조건에 DEFERRED 상태를 설정합니다. 필요한 경우 DEFERRED 상태를 한 개의 제약 조건으로 설정할 수도 있습니다.

```
SET CONSTRAINTS ALL DEFERRED;
```

```
SET CONSTRAINTS succeeded.
```

이제 sal\_ck 제약 조건을 위반하는 행을 삽입하려고 하면 명령문이 성공적으로 실행됩니다.

```
INSERT INTO emp_new_sal VALUES(90,5);
```

```
1 rows inserted
```

그러나 트랜잭션을 커밋하면 오류가 반환됩니다. 트랜잭션이 실패하여 롤백됩니다.

이는 COMMIT 시 두 제약 조건이 검사되기 때문입니다.

```
COMMIT;
```

```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.SAL_CK) violated
02091. 00000 - "transaction rolled back"
```

**예제 4:** IMMEDIATE 상태를 이전 예에서 DEFERRED로 설정한 두 제약 조건으로 설정합니다.

```
SET CONSTRAINTS ALL IMMEDIATE;
```

```
SET CONSTRAINTS succeeded.
```

sal\_ck 또는 bonus\_ck를 위반하는 행을 삽입하려고 하면 오류가 반환됩니다.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
SQL Error: ORA-02290: check constraint (ORA21.BONUS_CK) violated
02290. 00000 - "check constraint (%s.%s) violated"
```

**참고:** 제약 조건 지연 가능성을 지정하지 않고 테이블을 생성하면 각 명령문의 끝에서 제약 조건이 즉시 검사됩니다. 예를 들어, newemp\_details 테이블의 CREATE TABLE 문을 사용할 경우 newemp\_det\_pk 제약 조건 지연 가능성을 지정하지 않으면 제약 조건이 즉시 검사됩니다.

```
CREATE TABLE newemp_details(emp_id NUMBER, emp_name
VARCHAR2(20),
CONSTRAINT newemp_det_pk PRIMARY KEY(emp_id));
```

지연 불가능한 newemp\_det\_pk 제약 조건을 지연하려고 하면 다음 오류가 반환됩니다.

```
SET CONSTRAINT newemp_det_pk DEFERRED;
```

```
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
```

## 제약 조건 삭제

- EMP2 테이블에서 관리자 제약 조건을 제거합니다.

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```

```
ALTER TABLE Emp2 succeeded.
```

- DEPT2 테이블에서 PRIMARY KEY 제약 조건을 제거하고 EMP2.DEPARTMENT\_ID 열에서 연관된 FOREIGN KEY 제약 조건을 삭제합니다.

```
ALTER TABLE dept2
DROP PRIMARY KEY CASCADE;
```

```
ALTER TABLE dept2 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 삭제

제약 조건을 삭제하려면 USER\_CONSTRAINTS 및 USER\_CONS\_COLUMNS 데이터 디렉터리 뷰에서 제약 조건 이름을 식별합니다. 그런 다음 ALTER TABLE 문을 DROP 절과 함께 사용합니다. DROP 절의 CASCADE 옵션으로 인해 종속 제약 조건도 삭제됩니다.

#### 구문

```
ALTER TABLE    table
DROP PRIMARY KEY | UNIQUE (column) |
    CONSTRAINT constraint [CASCADE];
```

이 구문에서 다음이 적용됩니다.

<i>table</i>	테이블의 이름입니다.
<i>column</i>	제약 조건의 영향을 받는 열의 이름입니다.
<i>constraint</i>	제약 조건의 이름입니다.

무결성 제약 조건을 삭제하면 해당 제약 조건은 더 이상 Oracle 서버에 의해 수행되지 않으며 데이터 디렉터리에서도 사용할 수 없습니다.



## 제약 조건 비활성화

- ALTER TABLE 문의 DISABLE 절을 실행하여 무결성 제약 조건을 비활성화합니다.
- CASCADE 옵션을 적용하여 종속 무결성 제약 조건을 비활성화할 수 있습니다.

```
ALTER TABLE emp2  
DISABLE CONSTRAINT emp_dt_fk;
```

```
ALTER TABLE Emp2 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 비활성화

ALTER TABLE 문을 DISABLE 절과 함께 사용하여 제약 조건을 삭제하거나 다시 생성하지 않고 비활성화할 수 있습니다.

#### 구문

```
ALTER TABLE      table  
DISABLE CONSTRAINT constraint [CASCADE];
```

이 구문에서 다음이 적용됩니다.

*table*            테이블의 이름입니다.

*constraint*      제약 조건의 이름입니다.

#### 지침

- CREATE TABLE 문과 ALTER TABLE 문에서 모두 DISABLE 절을 사용할 수 있습니다.
- CASCADE 절은 종속 무결성 제약 조건을 비활성화합니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 비활성화하면 고유 인덱스가 제거됩니다.

## 제약 조건 활성화

- **ENABLE 절을 사용하여 테이블 정의에서 현재 비활성화된 무결성 제약 조건을 활성화할 수 있습니다.**

```
ALTER TABLE      emp2
ENABLE CONSTRAINT emp_dt_fk;
```

```
ALTER TABLE Emp2 succeeded.
```

- **UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하면 UNIQUE 인덱스가 자동으로 생성됩니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 활성화

ALTER TABLE 문을 ENABLE 절과 함께 사용하여 제약 조건을 삭제하거나 다시 생성하지 않고 활성화할 수 있습니다.

#### 구문

```
ALTER      TABLE      table
ENABLE     CONSTRAINT constraint;
```

이 구문에서 다음이 적용됩니다.

*table*           테이블의 이름입니다.  
*constraint*    제약 조건의 이름입니다.

#### 지침

- 제약 조건을 활성화하면 해당 제약 조건이 테이블의 모든 데이터에 적용됩니다. 테이블의 모든 데이터가 제약 조건을 따라야 합니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하면 UNIQUE 또는 PRIMARY KEY 인덱스가 자동으로 생성됩니다. 인덱스가 이미 존재하는 경우 해당 인덱스를 이러한 키로 사용할 수 있습니다.
- CREATE TABLE 문 및 ALTER TABLE 문에서 모두 ENABLE 절을 사용할 수 있습니다.

### 제약 조건 활성화(계속)

- CASCADE 옵션으로 비활성화된 PRIMARY KEY 제약 조건을 활성화하면 PRIMARY KEY에 종속된 FOREIGN KEY가 활성화되지 않습니다.
- UNIQUE KEY 또는 PRIMARY KEY 제약 조건을 활성화하려면 테이블에 인덱스를 생성하는 데 필요한 권한을 가지고 있어야 합니다.

## 계단식 제약 조건

- CASCADE CONSTRAINTS 절은 DROP COLUMN 절과 함께 사용됩니다.
- CASCADE CONSTRAINTS 절은 삭제된 열에 정의된 PRIMARY KEY 및 UNIQUE KEY를 참조하는 모든 참조 무결성 제약 조건을 삭제합니다.
- CASCADE CONSTRAINTS 절은 또한 삭제된 열에 정의된 모든 다중 열 제약 조건을 삭제합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 계단식 제약 조건

다음 명령문은 CASCADE CONSTRAINTS 절의 사용을 보여줍니다. TEST1 테이블이 다음과 같이 생성된다고 가정합니다.

```
CREATE TABLE test1 (  
    col1_pk NUMBER PRIMARY KEY,  
    col2_fk NUMBER,  
    col1 NUMBER,  
    col2 NUMBER,  
    CONSTRAINT fk_constraint FOREIGN KEY (col2_fk) REFERENCES  
        test1,  
    CONSTRAINT ck1 CHECK (col1_pk > 0 and col1 > 0),  
    CONSTRAINT ck2 CHECK (col2_fk > 0));
```

다음 명령문에 대해 오류가 반환됩니다.

```
ALTER TABLE test1 DROP (col1_pk); -col1_pk는 상위 키입니다.  
ALTER TABLE test1 DROP (col1); -col1은 다중 열 제약 조건  
ck1에 의해 참조됩니다.
```

## 계단식 제약 조건

예제:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

```
ALTER TABLE Emp2 succeeded.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1) CASCADE CONSTRAINTS;
```

```
ALTER TABLE test1 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 연쇄화(계속)

다음 명령문을 실행하면 EMPLOYEE\_ID 열, PRIMARY KEY 제약 조건 및 EMP2 테이블의 PRIMARY KEY 제약 조건을 참조하는 모든 FOREIGN KEY 제약 조건이 삭제됩니다.

```
ALTER TABLE emp2 DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

삭제된 열에 정의된 제약 조건에서 참조하는 모든 열도 삭제된 경우 CASCADE CONSTRAINTS는 필요하지 않습니다. 예를 들어, 다른 테이블의 참조 제약 조건이 COL1\_PK 열을 참조하지 않는다고 가정하면 이전 페이지에서 생성한 TEST1 테이블에 대해 CASCADE CONSTRAINTS 절 없이 다음 명령문을 실행하는 것은 유효합니다.

```
ALTER TABLE test1 DROP (col1_pk, col2_fk, col1);
```

## 테이블 열 및 제약 조건 이름 바꾸기

ALTER TABLE 문의 RENAME COLUMN 절을 사용하여 테이블 열의 이름을 바꿀 수 있습니다.

```
ALTER TABLE marketing RENAME COLUMN team_id  
TO id;
```

```
ALTER TABLE marketing succeeded.
```

ALTER TABLE 문의 RENAME CONSTRAINT 절을 사용하여 테이블의 기존 제약 조건에 대한 이름을 바꿀 수 있습니다.

```
ALTER TABLE marketing RENAME CONSTRAINT mktg_pk  
TO new_mktg_pk;
```

```
ALTER TABLE marketing succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 테이블 열 및 제약 조건 이름 바꾸기

테이블 열의 이름을 바꾸는 경우 새 이름이 기존 테이블 열의 이름과 충돌해서는 안되며, RENAME COLUMN 절과 함께 다른 절을 사용할 수 없습니다.

위 슬라이드 예제에서는 id 열에 PRIMARY KEY mktg\_pk가 정의된 marketing 테이블을 사용합니다.

```
CREATE TABLE marketing (team_id NUMBER(10),  
                        target VARCHAR2(50),  
CONSTRAINT mktg_pk PRIMARY KEY(team_id));
```

```
CREATE TABLE succeeded.
```

예제 a는 marketing 테이블의 id 열 이름이 mktg\_id로 바뀌었음을 보여주고, 예제 b는 mktg\_pk의 이름이 new\_mktg\_pk로 바뀌었음을 보여줍니다.

테이블의 기존 제약 조건 이름을 바꾸는 경우 새 이름이 기존 제약 조건 이름과 충돌해서는 안됩니다. RENAME CONSTRAINT 절을 사용하면 시스템에서 생성한 제약 조건의 이름을 바꿀 수 있습니다.

## 단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- **인덱스 생성:**
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# 인덱스 개요

인덱스는 다음과 같이 생성됩니다.

- 자동
  - PRIMARY KEY 생성
  - UNIQUE KEY 생성
- 수동
  - CREATE INDEX 문
  - CREATE TABLE 문

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 인덱스 개요

두 가지 유형의 인덱스를 생성할 수 있습니다. 첫번째 유형은 고유 인덱스입니다. PRIMARY KEY 또는 UNIQUE KEY 제약 조건을 갖도록 테이블의 열 또는 열 그룹을 정의할 때 Oracle 서버가 자동으로 고유 인덱스를 생성합니다. 인덱스의 이름은 제약 조건에 지정된 이름입니다. 두번째 인덱스 유형은 비고유 인덱스이며 사용자가 생성할 수 있습니다. 예를 들어 검색 속도를 향상시키기 위해 조인에서 사용될 FOREIGN KEY 열에 대한 인덱스를 생성할 수 있습니다.

CREATE INDEX 문을 실행하여 하나 이상의 열에 인덱스를 생성할 수 있습니다.

자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

**참고:** 고유 인덱스를 수동으로 생성할 수도 있지만, 고유 인덱스를 암시적으로 생성하는 UNIQUE 제약 조건을 생성하는 것이 좋습니다.



## CREATE TABLE 문을 사용한 CREATE INDEX

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
PRIMARY KEY USING INDEX
(CREATE INDEX emp_id_idx ON
NEW_EMP(employee_id)),
first_name VARCHAR2(20),
last_name VARCHAR2(25));
```

CREATE TABLE succeeded.

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'NEW_EMP';
```

INDEX_NAME	TABLE_NAME
1 EMP_ID_IDX	NEW_EMP

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE TABLE 문을 사용한 CREATE INDEX

슬라이드의 예제에서 CREATE INDEX 절은 CREATE TABLE 문과 함께 사용되어 PRIMARY KEY 인덱스를 명시적으로 생성합니다. PRIMARY KEY 생성 시 PRIMARY KEY 제약 조건의 이름과 다르게 인덱스의 이름을 지정할 수 있습니다.

USER\_INDEXES 데이터 디렉터리 뷰를 query하여 인덱스에 대한 정보를 확인할 수 있습니다.

**참고:** USER\_INDEXES에 대한 자세한 내용은 "데이터 디렉터리 뷰를 사용하여 객체 관리" 단원을 참조하십시오.

다음 예제는 인덱스의 이름이 명시적으로 지정되지 않은 경우의 데이터베이스 동작을 보여줍니다.

```
CREATE TABLE EMP_UNNAMED_INDEX
(employee_id NUMBER(6) PRIMARY KEY ,
first_name VARCHAR2(20),
last_name VARCHAR2(25));
```

CREATE TABLE succeeded.

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

INDEX_NAME	TABLE_NAME
1 SYS_C0017294	EMP_UNNAMED_INDEX

## CREATE TABLE 문을 사용한 CREATE INDEX(계속)

Oracle 서버는 PRIMARY KEY 열에 대해 생성되는 인덱스에 일반적인 이름을 지정합니다.

또한 예를 들어 데이터 로드가 많을 것으로 예상되어 작업 속도를 높이려는 경우 PRIMARY KEY 열에 기존 인덱스를 사용할 수 있습니다. 로드를 수행하는 동안 제약 조건을 비활성화한 다음 제약 조건을 다시 활성화하기를 원하는 경우, PRIMARY KEY에 대한 고유 인덱스가 있으면 로드 중에 데이터가 계속 검사됩니다. 따라서 우선 PRIMARY KEY로 지정된 열에 비고유 인덱스를 생성한 다음 PRIMARY KEY 열을 생성하고 이 열이 기존 인덱스를 사용하도록 지정할 수 있습니다. 다음 예제는 이 프로세스를 보여줍니다.

### 1단계: 테이블 생성:

```
CREATE TABLE NEW_EMP2
(employee_id NUMBER(6),
first_name VARCHAR2(20),
last_name VARCHAR2(25)
);
```

### 2단계: 인덱스 생성:

```
CREATE INDEX emp_id_idx2 ON
new_emp2(employee_id);
```

### 3단계: PRIMARY KEY 생성:

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
emp_id_idx2;
```

## 함수 기반 인덱스

- 함수 기반 인덱스는 표현식을 기반으로 합니다.
- 인덱스 표현식은 테이블 열, 제약 조건, SQL 함수 및 유저 정의 함수에서 작성됩니다.

```
CREATE INDEX upper_dept_name_idx  
ON dept2(UPPER(department_name));
```

```
CREATE INDEX succeeded.
```

```
SELECT *  
FROM    dept2  
WHERE   UPPER(department_name) = 'SALES';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 함수 기반 인덱스

UPPER(*column\_name*) 또는 LOWER(*column\_name*) 키워드로 정의된 함수 기반 인덱스는 검색 시 대소문자를 구분하지 않습니다. 예를 들어, 다음 인덱스를 살펴 보십시오.

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

이 인덱스는 다음과 같은 query를 효과적으로 처리합니다.

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

Oracle 서버는 query에서 특정 함수가 사용되는 경우에만 인덱스를 사용합니다. 예를 들어, 다음 명령문은 인덱스를 사용할 수도 있지만 WHERE 절이 없는 경우 Oracle 서버가 전체 테이블 스캔을 수행할 수도 있습니다.

```
SELECT *  
FROM    employees  
WHERE   UPPER (last_name) IS NOT NULL  
ORDER BY UPPER (last_name);
```

**참고:** 사용할 함수 기반 인덱스에 대해 QUERY\_REWRITE\_ENABLED 초기화 파라미터가 TRUE로 설정되어야 합니다.

Oracle 서버는 DESC로 표시된 열이 있는 인덱스를 함수 기반 인덱스로 처리합니다. DESC로 표시된 열은 내림차순으로 정렬됩니다.

## 인덱스 제거

- DROP INDEX 명령을 사용하여 데이터 디렉터리에서 인덱스를 제거할 수 있습니다.

```
DROP INDEX index;
```

- 데이터 디렉터리에서 UPPER\_DEPT\_NAME\_IDX 인덱스를 제거합니다.

```
DROP INDEX upper_dept_name_idx;
```

```
DROP INDEX upper_dept_name_idx succeeded.
```

- 인덱스를 삭제하려면 인덱스의 소유자이거나 DROP ANY INDEX 권한이 있어야 합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 인덱스 제거

인덱스는 수정할 수 없습니다. 인덱스를 변경하려면 인덱스를 삭제한 다음 다시 생성해야 합니다. DROP INDEX 문을 실행하여 데이터 디렉터리에서 인덱스 정의를 제거합니다.

인덱스를 삭제하려면 인덱스의 소유자이거나 DROP ANY INDEX 권한이 있어야 합니다.

이 구문에서 다음이 적용됩니다.

*index*                    인덱스의 이름입니다.

**참고:** 테이블을 삭제하면 인덱스, 제약 조건 및 트리거는 자동으로 삭제되지만 뷰와 시퀀스는 남아 있습니다.

## DROP TABLE ... PURGE

```
DROP TABLE dept80 PURGE;
```

```
DROP TABLE dept80 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DROP TABLE ... PURGE

오라클 데이터베이스는 테이블 삭제 기능을 제공합니다. 테이블을 삭제할 때 데이터베이스는 테이블과 연관된 공간을 즉시 해제하지 않습니다. 대신 테이블의 이름을 바꾸고 테이블을 Recycle bin에 넣습니다. 따라서 테이블을 실수로 삭제했을 경우 나중에 FLASHBACK TABLE 문으로 테이블을 recovery할 수 있습니다. DROP TABLE 문을 실행할 때 테이블과 연관된 공간을 즉시 해제하려면 슬라이드의 명령문에서와 같이 PURGE 절을 포함시키십시오.

한 단계에서 테이블을 삭제하고 테이블과 연관된 공간을 해제하려는 경우에만 PURGE를 지정하십시오. PURGE를 지정한 경우 데이터베이스는 테이블과 종속 객체를 Recycle bin에 넣지 않습니다.

이 절을 사용하는 것은 먼저 테이블을 삭제한 다음 Recycle bin에서 테이블을 지우는 것과 같습니다. 이 절을 사용하면 프로세스를 한 단계 줄일 수 있습니다. 또한 중요한 자료를 Recycle bin에서 숨길 수 있는 고급 보안 기능을 제공합니다.

## 단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- **Flashback 작업 수행**
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## FLASHBACK TABLE 문

- 한 개의 명령문으로 테이블을 지정된 시점으로 recovery할 수 있습니다.
- 테이블 데이터를 연관된 인덱스 및 제약 조건과 함께 복원합니다.
- 테이블과 테이블의 내용을 특정 시점이나 시스템 변경 번호(SCN)로 되돌릴 수 있습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FLASHBACK TABLE 문

Oracle Flashback Table을 사용하면 한 개의 명령문으로 테이블을 지정된 시점으로 recovery할 수 있습니다. 데이터베이스가 온라인 상태인 동안에 연관된 인덱스 및 제약 조건과 함께 테이블 데이터를 복원하여 지정된 테이블에 대한 변경 사항만 언두할 수 있습니다.

Flashback Table 기능은 셀프 서비스 복구 도구와 비슷합니다. 예를 들어, 테이블에서 실수로 삭제한 중요한 행을 recovery하려는 경우 FLASHBACK TABLE 문을 사용하여 테이블을 삭제 이전 시기로 복원하면 없어졌던 행을 테이블에서 볼 수 있습니다.

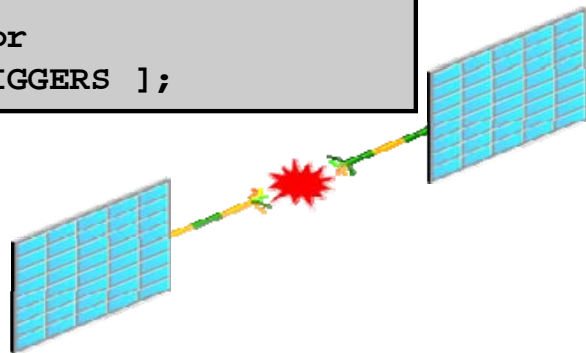
FLASHBACK TABLE 문을 사용할 경우 테이블과 테이블의 내용을 특정 시점이나 SCN으로 되돌릴 수 있습니다.

**참고:** SCN은 각 데이터베이스 변경 사항과 연관된 정수 값으로, 데이터베이스에서 고유한 incremental 숫자입니다. 트랜잭션을 커밋할 때마다 새로운 SCN이 기록됩니다.

## FLASHBACK TABLE 문

- 실수로 테이블이 수정된 경우 복구 도구로 사용됩니다.
  - 테이블을 과거 시점으로 복원
  - 이점: 사용 편의성, 가용성, 빠른 실행
  - 현재 위치에서 수행됨
- 구문:

```
FLASHBACK TABLE[schema.]table [, [schema.]table ]...  
TO { TIMESTAMP | SCN } expr  
[ { ENABLE | DISABLE } TRIGGERS ];
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FLASHBACK TABLE 문(계속)

#### 셀프 서비스 복구 도구

오라클 데이터베이스는 테이블이 실수로 삭제되거나 수정된 경우 테이블의 상태를 과거 시점으로 복원할 수 있는 새로운 SQL DDL(데이터 정의어) 명령 **FLASHBACK TABLE**을 제공합니다. **FLASHBACK TABLE** 명령은 테이블의 데이터를 인덱스 또는 뷰와 같은 관련 속성과 함께 복원하는 셀프 서비스 복구 도구입니다. 이 명령은 데이터베이스가 온라인일 때 지정된 테이블의 이후 변경 사항만 롤백합니다. 기존 **recovery** 방식과 비교하여 이 기능은 사용 편의성, 가용성 및 빠른 복원과 같은 중요한 이점을 제공합니다. 또한 응용 프로그램 특정 속성을 찾아 복원하므로 DBA의 작업 부담을 줄여 줍니다. **FLASHBACK TABLE** 기능은 잘못된 디스크로 인해 발생하는 물리적 손상은 해결하지 못합니다.

#### 구문

하나 이상의 테이블에서(서로 다른 스키마의 테이블인 경우도 포함) **FLASHBACK TABLE** 작업을 호출할 수 있습니다. 유효한 시간 기록을 제공하여 되돌아 가려는 시점을 지정합니다. 기본적으로 **flashback** 작업 동안에는 관련된 모든 테이블에 대해 데이터베이스 트리거가 비활성화됩니다. **ENABLE TRIGGERS** 절을 지정하여 이 기본 동작을 재정의할 수 있습니다.

**참고:** Recycle bin 및 Flashback 구문에 대한 자세한 내용은

*Oracle Database Administrator's Guide 11g Release 2 (11.2)*를 참조하십시오.



## FLASHBACK TABLE 문 사용

```
DROP TABLE emp2;
```

```
DROP TABLE emp2 succeeded.
```

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

ORIGINAL_NAME	OPERATION	DROPTIME
EMP2	DROP	2009-05-20:18:00:39

...

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

```
FLASHBACK TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FLASHBACK TABLE 문 사용

#### 구문 및 예제

이 예제에서는 EMP2 테이블을 DROP 문 이전 상태로 복원합니다.

Recycle bin은 실제로는 삭제된 객체에 대한 정보가 담긴 데이터 디크너리 테이블입니다. 삭제된 테이블 및 인덱스, 제약 조건, 중첩 테이블 등과 같은 모든 연관된 객체는 제거되지 않고 여전히 공간을 차지합니다. 이러한 객체는 Recycle bin에서 완전히 지워지거나 테이블스페이스 공간 제약 조건으로 인해 데이터베이스에서 삭제될 때까지 계속해서 유저 공간 할당량에 포함됩니다.

유저가 SYSDBA 권한을 갖고 있지 않은 경우 자신이 소유한 객체만 Recycle bin에서 액세스할 수 있으므로 각 유저를 Recycle bin의 소유자로 간주할 수 있습니다. 유저는 다음 명령문을 사용하여 Recycle bin에서 자신의 객체를 볼 수 있습니다.

```
SELECT * FROM RECYCLEBIN;
```

유저를 삭제하면 해당 유저에 속한 모든 객체는 Recycle bin에 저장되지 않고 Recycle bin의 모든 객체가 지워집니다.

다음 명령문으로 Recycle bin을 비울 수 있습니다.

```
PURGE RECYCLEBIN;
```

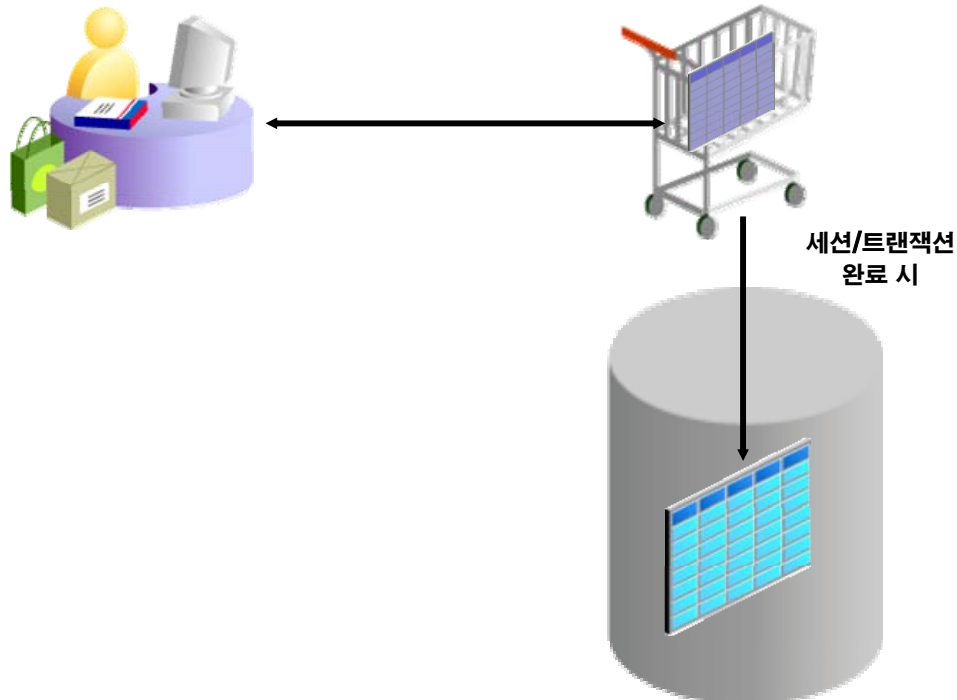
## 단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- Flashback 작업 수행
- **임시 테이블 생성 및 사용**
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 임시 테이블



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 임시 테이블

임시 테이블은 트랜잭션이나 세션 기간 동안에만 존재하는 데이터를 보유하는 테이블입니다. 임시 테이블의 데이터는 세션 전용(private)이므로 각 세션은 자신의 데이터만 보고 수정할 수 있습니다.

임시 테이블은 결과 집합을 버퍼링해야 하는 응용 프로그램에서 유용합니다. 예를 들어, 온라인 응용 프로그램의 쇼핑 카트가 임시 테이블일 수 있습니다. 각 항목은 임시 테이블의 한 행으로 표현됩니다. 온라인 상점에서 쇼핑하는 동안 카트에서 계속 항목을 추가하거나 제거할 수 있습니다. 세션 동안 이 카트 데이터는 전용(private)입니다. 쇼핑을 마치고 지불한 후에는 선택한 카트의 행이 영구 테이블로 이동됩니다. 세션이 끝날 때 임시 테이블의 데이터가 자동으로 삭제됩니다.

임시 테이블은 정적으로 정의되기 때문에 그에 대한 인덱스를 생성할 수 있습니다. 임시 테이블에 생성된 인덱스도 임시입니다. 인덱스 데이터의 세션 또는 트랜잭션 범위는 임시 테이블의 데이터와 같습니다. 임시 테이블에 뷰나 트리거를 생성할 수도 있습니다.

## 임시 테이블 생성

```
CREATE GLOBAL TEMPORARY TABLE cart
ON COMMIT DELETE ROWS;
```

1

```
CREATE GLOBAL TEMPORARY TABLE today_sales
ON COMMIT PRESERVE ROWS AS
  SELECT * FROM orders
  WHERE order_date = SYSDATE;
```

2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 임시 테이블 생성

임시 테이블을 생성하기 위해 다음 명령을 사용할 수 있습니다.

```
CREATE GLOBAL TEMPORARY TABLE tablename
ON COMMIT [PRESERVE | DELETE] ROWS
```

다음 설정 중 하나를 ON COMMIT 절과 연관시켜 임시 테이블의 데이터가 트랜잭션 관련 데이터(기본값)인지 아니면 세션 관련 데이터인지 결정할 수 있습니다.

1. **DELETE ROWS**: 슬라이드의 예제 1과 같이 DELETE ROWS 설정은 트랜잭션 관련 임시 테이블을 생성합니다. 테이블에 트랜잭션이 처음 삽입될 때 세션이 임시 테이블에 바인드되며, 트랜잭션이 끝날 때 바인드가 해제됩니다. 커밋할 때마다 데이터베이스가 테이블을 truncate하여 모든 행을 삭제합니다.
2. **PRESERVE ROWS**: 슬라이드의 예제 2와 같이 PRESERVE ROWS 설정은 세션 관련 임시 테이블을 생성합니다. 영업 사원 세션마다 테이블에 해당 날짜의 영업 데이터를 저장할 수 있습니다. 영업 사원이 today\_sales 테이블에 데이터를 처음 삽입할 때 해당 세션이 today\_sales 테이블에 바인드되며, 세션이 끝나거나 세션에서 테이블의 TRUNCATE를 실행할 때 바인드가 해제됩니다. 세션을 종료할 때 데이터베이스가 테이블을 truncate합니다.

오라클 데이터베이스에 임시 테이블을 생성하면 정적 테이블 정의가 생성됩니다. 영구 테이블과 마찬가지로 임시 테이블은 데이터 디렉토리에 정의됩니다. 그러나 임시 테이블과 해당 인덱스는 생성 시 자동으로 세그먼트를 할당하지 않습니다. 대신 데이터를 처음 삽입할 때 임시 세그먼트가 할당됩니다. 데이터가 세션에 로드될 때까지 테이블이 빈 상태로 나타납니다.

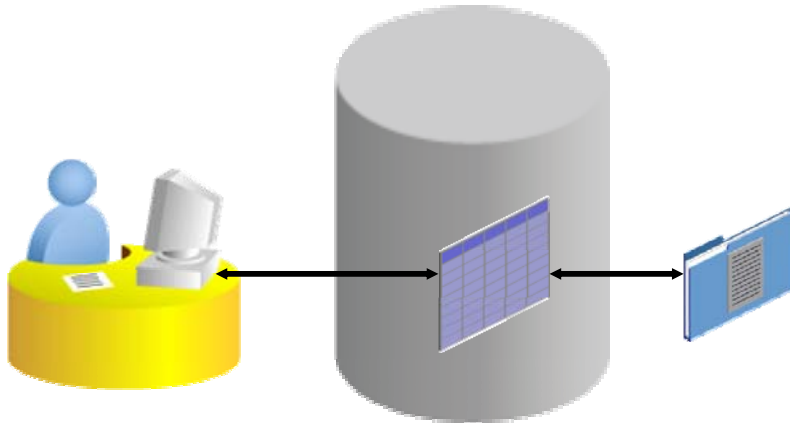
## 단원 내용

- ALTER TABLE 문을 사용하여 열 추가, 수정 및 삭제
- 제약 조건 관리:
  - 제약 조건 추가 및 삭제
  - 제약 조건 지연
  - 제약 조건 활성화 및 비활성화
- 인덱스 생성:
  - CREATE TABLE 문 사용
  - 함수 기반 인덱스 생성
  - 인덱스 제거
- Flashback 작업 수행
- 임시 테이블 생성 및 사용
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# External Table



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## External Table

External Table은 메타 데이터가 데이터베이스에 저장되지만 데이터는 데이터베이스 외부에 저장되는 읽기 전용 테이블입니다. 이 External Table은 external 데이터를 먼저 데이터베이스에 로드할 필요 없이 external 데이터에 SQL query를 실행할 수 있는 뷰로 간주할 수 있습니다. External Table 데이터는 external 데이터를 먼저 데이터베이스에 로드할 필요 없이 직접 또는 병렬로 query하고 조인할 수 있습니다. SQL, PL/SQL 및 Java를 사용하여 External Table의 데이터를 query할 수 있습니다.

External Table과 일반 테이블의 주요 차이점은 external에서 구성된 테이블은 읽기 전용이라는 것입니다. 따라서 DML(데이터 조작어) 작업을 수행할 수 없으며 이러한 테이블에는 인덱스를 생성할 수 없습니다. 그러나 External Table을 생성할 수 있으므로 CREATE TABLE AS SELECT 명령을 사용하여 데이터를 언로드할 수 있습니다.

## External Table(계속)

Oracle 서버는 External Table에 두 개의 기본 액세스 드라이버를 제공합니다. 하나는 로더 액세스 드라이버(또는 ORACLE\_LOADER)로, SQL\*Loader 유틸리티가 형식을 해석할 수 있는 external 파일의 데이터를 읽는 데 사용됩니다. External Table에서 모든 SQL\*Loader 기능이 지원되는 것은 아닙니다. ORACLE\_DATAPUMP 액세스 드라이버는 플랫폼 독립적인 형식을 사용하여 데이터를 импорт 및 익스포트하는 데 사용할 수 있습니다. ORACLE\_DATAPUMP 액세스 드라이버는

CREATE TABLE ...ORGANIZATION EXTERNAL...AS SELECT 문의 일부로 External Table로 로드할 SELECT 문의 행을 기록합니다. 그러면 사용자가 SELECT를 사용하여 해당 데이터 파일에서 데이터를 읽을 수 있습니다. 또한 다른 시스템에서 External Table 정의를 생성하고 해당 데이터 파일을 사용할 수 있습니다. 이렇게 하면 오라클 데이터베이스 간에 데이터를 이동할 수 있습니다.

## External Table의 디렉토리 생성

External 데이터 소스가 있는 파일 시스템의 디렉토리에 해당하는 DIRECTORY 객체를 생성합니다.

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/.../emp_dir';

GRANT READ ON DIRECTORY emp_dir TO ora_21;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### External Table 생성의 예

CREATE DIRECTORY 문을 사용하여 디렉토리 객체를 생성할 수 있습니다. 디렉토리 객체는 external 데이터 소스가 상주하는 서버의 파일 시스템에 있는 디렉토리에 대한 alias를 지정합니다. 파일 관리의 유연성을 높이기 위해 external 데이터 소스를 참조할 때 운영 체제 경로 이름을 하드 코딩하는 대신 디렉토리 이름을 사용할 수 있습니다.

디렉토리를 생성하려면 CREATE ANY DIRECTORY 시스템 권한이 있어야 합니다. 디렉토리를 생성하면 READ 및 WRITE 객체 권한이 자동으로 부여되고 READ 및 WRITE 권한을 다른 유저와 롤에 부여할 수 있습니다. DBA가 이러한 권한을 다른 유저와 롤에 부여할 수도 있습니다.

유저는 액세스할 External Table에서 사용되는 모든 디렉토리에 대해 READ 권한이 필요하고, 사용될 로그 파일, 손상된 파일 및 discard file 위치에 대해 WRITE 권한이 필요합니다.

또한 External Table 프레임워크가 데이터를 언로드하는 데 사용되고 있는 경우에도 WRITE 권한이 필요합니다.

오라클은 또한 데이터를 언로드(즉, 데이터베이스의 테이블에서 데이터를 읽고 External Table로 데이터를 삽입)한 다음 해당 데이터를 오라클 데이터베이스에 다시 로드할 수 있는 ORACLE\_DATAPUMP 유형을 제공합니다. 이 작업은 테이블이 생성될 때 한 번만 수행할 수 있는 작업입니다. 생성 및 최초의 채우기가 수행된 후에는 행을 갱신, 삽입 또는 삭제할 수 없습니다.



## External Table 생성의 예(계속)

### 구문

```
CREATE [OR REPLACE] DIRECTORY AS 'path_name';
```

이 구문에서 다음이 적용됩니다.

OR REPLACE	이미 디렉토리 데이터베이스 객체가 있는 경우 다시 생성하려면 OR REPLACE를 지정합니다. 이 절을 사용하여 디렉토리에 이전에 부여된 데이터베이스 객체 권한을 삭제, 재생성 및 재부여하지 않고 기존 디렉토리 정의를 변경할 수 있습니다. 재정의된 디렉토리에 대해 이전에 권한을 부여받은 유저는 권한을 다시 부여받지 않아도 이 디렉토리에 계속 액세스할 수 있습니다.
directory	생성할 디렉토리 객체의 이름을 지정합니다. 디렉토리 이름의 최대 길이는 30바이트입니다. 디렉토리 객체에 스키마 이름을 사용할 수 없습니다.
'path_name'	액세스할 운영 체제 디렉토리의 전체 경로 이름을 지정합니다. 경로 이름은 대소문자를 구분합니다.

# External Table 생성

```
CREATE TABLE <table_name>
  ( <col_name> <datatype>, ... )
ORGANIZATION EXTERNAL
  (TYPE <access_driver_type>
    DEFAULT DIRECTORY <directory_name>
    ACCESS PARAMETERS
      (... ) )
    LOCATION ('<location_specifier>')
REJECT LIMIT [0 | <number> | UNLIMITED];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## External Table 생성

CREATE TABLE 문의 ORGANIZATION EXTERNAL 절을 사용하여 External Table을 생성합니다. 실제로는 테이블을 생성하는 것이 아니라 external 데이터에 액세스할 때 사용할 수 있도록 데이터 디렉터리에서 메타 데이터를 생성하는 것입니다. ORGANIZATION 절을 사용하여 테이블의 데이터 행이 저장되는 순서를 지정합니다. ORGANIZATION 절에 EXTERNAL을 지정하여 테이블이 데이터베이스 외부에 있는 읽기 전용 테이블임을 나타냅니다. External 파일이 이미 데이터베이스 외부에 존재해야 합니다.

TYPE <access\_driver\_type>은 External Table의 액세스 드라이버를 나타냅니다. 액세스 드라이버는 데이터베이스의 external 데이터를 해석하는 API(응용 프로그램 프로그래밍 인터페이스)입니다. TYPE을 지정하지 않으면 오라클은 기본 액세스 드라이버인 ORACLE\_LOADER를 사용합니다. 다른 옵션은 ORACLE\_DATAPUMP입니다.

DEFAULT DIRECTORY 절을 사용하여 external 데이터 소스가 상주할 수 있는 파일 시스템의 디렉토리에 해당하는 여러 오라클 데이터베이스 디렉토리 객체를 지정할 수 있습니다.

선택 사항인 ACCESS PARAMETERS 절을 사용하여 이 External Table에 대한 특정 액세스 드라이버의 파라미터에 값을 할당할 수 있습니다.

## External Table 생성(계속)

LOCATION 절을 사용하여 각 external 데이터 소스에 대해 하나의 external 위치 지정자를 지정할 수 있습니다. 일반적으로 *<location\_specifier>*는 파일이지만 반드시 그럴 필요는 없습니다.

REJECT LIMIT 절을 사용하면 Oracle 오류가 반환되고 query가 중단되기 전, external 데이터 query 중에 발생할 수 있는 변환 오류 횟수를 지정할 수 있습니다. 기본값은 0입니다.

ORACLE\_DATAPUMP 액세스 드라이버를 사용하는 구문은 다음과 같습니다.

```
CREATE TABLE extract_emps
ORGANIZATION EXTERNAL (TYPE ORACLE_DATAPUMP
                        DEFAULT DIRECTORY ...
                        ACCESS PARAMETERS (... )
                        LOCATION (... )
                        PARALLEL 4
                        REJECT LIMIT UNLIMITED
AS
SELECT * FROM ...;
```

## ORACLE\_LOADER를 사용하여 External Table 생성

```
CREATE TABLE oldemp (  
  fname char(25), lname CHAR(25))  
  ORGANIZATION EXTERNAL  
  (TYPE ORACLE_LOADER  
  DEFAULT DIRECTORY emp_dir  
  ACCESS PARAMETERS  
  (RECORDS DELIMITED BY NEWLINE  
  NOBADFILE  
  NOLOGFILE  
  FIELDS TERMINATED BY ','  
  (fname POSITION ( 1:20) CHAR,  
  lname POSITION (22:41) CHAR))  
  LOCATION ('emp.dat'))  
  PARALLEL 5  
  REJECT LIMIT 200;
```

CREATE TABLE succeeded.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ORACLE\_LOADER 액세스 드라이버를 사용한 External Table 생성의 예

다음 형식의 레코드가 있는 플랫폼 파일이 있다고 가정합니다.

```
10,jones,11-Dec-1934  
20,smith,12-Jun-1972
```

레코드는 개행 문자로 구분되며 필드는 모두 쉼표(,)로 종료됩니다. 파일 이름은 /emp\_dir/emp.dat입니다.

이 파일을 메타 데이터가 데이터베이스에 상주하는 External Table의 데이터 소스로 변환하려면 다음 단계를 수행해야 합니다.

1. 다음과 같이 디렉토리 객체 emp\_dir을 생성합니다.

```
CREATE DIRECTORY emp_dir AS '/emp_dir' ;
```

2. 슬라이드에 표시된 CREATE TABLE 명령을 실행합니다.

위 슬라이드 예제는 파일에 대한 External table을 생성하는 테이블 사양을 보여줍니다.  
/emp\_dir/emp.dat

## ORACLE\_LOADER 액세스 드라이버를 사용한 External Table 생성의 예(계속)

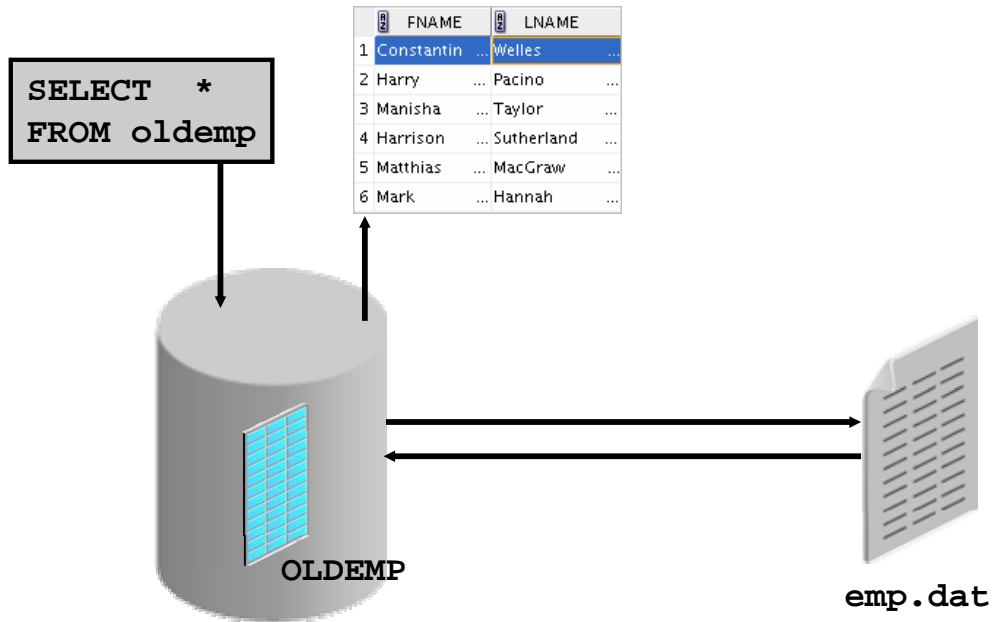
예제에서 TYPE 사양은 단지 그 용도를 보여주기 위해 제시된 것입니다. 액세스 드라이버를 지정하지 않은 경우 ORACLE\_LOADER가 기본값으로 사용됩니다. ACCESS PARAMETERS 옵션은 특정 액세스 드라이버의 파라미터에 값을 제공하며, 이 값은 Oracle 서버가 아닌 액세스 드라이버에 의해 해석됩니다.

PARALLEL 절을 사용하면 INSERT INTO TABLE 문을 실행할 때 다섯 개의 병렬 실행 서버에서 동시에 external 데이터 소스(파일)를 스캔할 수 있습니다. 예를 들어, PARALLEL=5로 지정된 경우 한 대 이상의 병렬 실행 서버가 데이터 소스에 작업을 수행할 수 있습니다. External Table이 아주 클 수 있으므로 성능 문제를 고려하여 PARALLEL 절을 지정하거나 query에 대해 병렬 힌트를 지정하는 것이 좋습니다.

REJECT LIMIT 절은 external 데이터를 query하는 동안 200개가 넘는 변환 오류가 발생하면 query를 중지하고 오류를 반환하도록 지정합니다. 이러한 변환 오류는 액세스 드라이버가 데이터 파일의 데이터를 External Table 정의에 일치하도록 변형시키려고 할 때 발생할 수 있습니다.

CREATE TABLE 명령이 성공적으로 실행된 후에는 OLDEMP External Table을 관계형 테이블과 같은 방식으로 나타내고 query할 수 있습니다.

## External Table 조회



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### External Table 조회

External Table은 데이터베이스에 저장되어 있는 데이터를 설명하지 않습니다. external 소스에 데이터를 저장하는 방법도 설명하지 않습니다. 대신 External Table 층에서 서버에 데이터를 어떻게 표시해야 하는지를 나타냅니다. External Table 정의에 맞게 데이터 파일의 데이터에 필요한 변형을 수행하는 작업은 액세스 드라이버와 External Table 층에서 담당합니다.

데이터베이스 서버에서 external 소스의 데이터에 액세스하는 경우 해당 액세스 드라이버를 호출하여 데이터베이스 서버에서 읽을 수 있는 형태로 external 소스의 데이터를 가져옵니다.

데이터 소스의 데이터 설명은 External Table 정의와 별개라는 점에 유의하십시오. 소스 파일에는 테이블의 열 수보다 많거나 적은 필드 수가 포함될 수 있습니다. 또한 데이터 소스의 필드에 대한 데이터 유형이 테이블의 열과 다를 수도 있습니다. 액세스 드라이버는 데이터 소스의 데이터가 External Table 정의와 일치하도록 처리되는지 확인합니다.

## ORACLE\_DATAPUMP를 사용하여 External Table 생성: 예제

```
CREATE TABLE emp_ext
(employee_id, first_name, last_name)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_DATAPUMP
  DEFAULT DIRECTORY emp_dir
  LOCATION
    ('emp1.exp', 'emp2.exp')
)
PARALLEL
AS
SELECT employee_id, first_name, last_name
FROM employees;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ORACLE\_DATAPUMP를 사용하여 External Table 생성: 예제

ORACLE\_DATAPUMP 액세스 드라이버를 사용하여 External Table 관련 작업을 언로드 및 재로드할 수 있습니다.

**참고:** External Table 컨텍스트에서 데이터 로딩은 External Table에서 데이터가 읽혀져서 데이터베이스의 테이블로 로드되는 작업을 말합니다. 데이터 언로딩은 데이터베이스의 테이블에서 데이터를 읽어 External Table로 데이터를 삽입하는 작업을 말합니다.

위 슬라이드 예제는 ORACLE\_DATAPUMP 액세스 드라이버를 사용하여 External Table을 생성하기 위한 테이블 사양을 보여줍니다. 데이터가 emp1.exp 파일과 emp2.exp 파일에 채워집니다.

EMPLOYEES 테이블에서 External Table로 읽어 들인 데이터를 채우려면 다음 단계를 수행해야 합니다.

1. 다음과 같이 디렉토리 객체 emp\_dir을 생성합니다.  
CREATE DIRECTORY emp\_dir AS '/emp\_dir' ;
2. 슬라이드에 표시된 CREATE TABLE 명령을 실행합니다.

**참고:** emp\_dir 디렉토리는 ORACLE\_LOADER를 사용하는 이전 예에서 생성된 것과 동일합니다.

다음 코드를 실행하여 External Table을 query할 수 있습니다.

```
SELECT * FROM emp_ext;
```

## 퀴즈

FOREIGN KEY 제약 조건을 사용할 경우 상위 키의 데이터가 삭제되면 해당 상위 키 값에 종속된 하위 테이블의 모든 행도 삭제됩니다.

1. 맞습니다.
2. 틀립니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2



## 퀴즈

**DROP TABLE 명령을 실행하면 항상 데이터베이스가 테이블의 이름을 바꾸고 테이블을 Recycle bin에 넣습니다. 나중에 FLASHBACK TABLE 문을 사용하여 Recycle bin에서 테이블을 recovery할 수 있습니다.**

- 1. 맞습니다.**
- 2. 틀립니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 제약 조건 추가
- 인덱스 생성
- CREATE TABLE 문을 사용하여 인덱스 생성
- 함수 기반(Function-based) 인덱스 생성
- 열 삭제 및 UNUSED 열로 설정
- FLASHBACK 작업 수행
- External Table 생성 및 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

이 단원에서는 스키마 객체 관리를 위해 다음 작업을 수행하는 방법을 배웠습니다.

- 테이블을 변경하여 열 또는 제약 조건을 추가하거나 수정합니다.
- CREATE INDEX 문을 사용하여 인덱스 및 함수 기반 인덱스를 생성합니다.
- unused 열을 삭제합니다.
- FLASHBACK 방식을 사용하여 테이블을 복원합니다.
- CREATE TABLE 문의 ORGANIZATION EXTERNAL 절을 사용하여 External Table을 생성합니다. External Table은 메타 데이터가 데이터베이스에 저장되지만 데이터는 데이터베이스 외부에 저장되는 읽기 전용 테이블입니다.
- 데이터를 데이터베이스에 먼저 로드하지 않고 External Table을 사용하여 데이터를 query합니다.
- CREATE TABLE 문으로 테이블을 생성할 때 PRIMARY KEY 열 인덱스의 이름을 지정합니다.

## 연습 2: 개요

이 연습에서는 다음 내용을 다룹니다.

- 테이블 변경
- 열 추가
- 열 삭제
- 인덱스 생성
- External Table 생성

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 2: 개요

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



# 3

## 데이터 디렉터리 뷰를 사용하여 객체 관리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 단원을 마치면 다음을 수행할 수 있습니다.**

- 데이터 디렉터리 뷰를 사용하여 객체의 데이터 검색
- 다양한 데이터 디렉터리 뷰 조회

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 데이터 디렉터리 뷰를 소개합니다. 데이터 디렉터리 뷰를 사용하여 메타 데이터를 검색하고 스키마 객체에 대한 보고서를 생성하는 방법을 배웁니다.

## 단원 내용

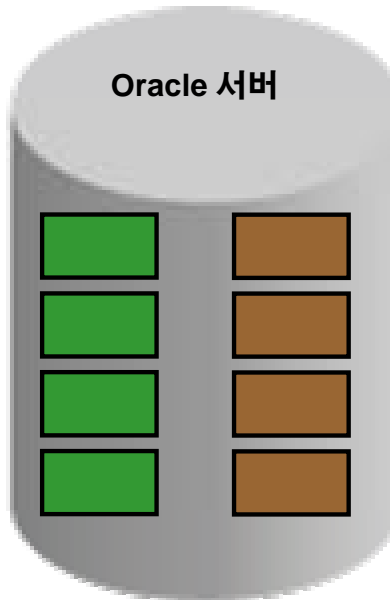
- 데이터 디렉터리 소개
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 테이블 정보
  - 열 정보
  - 제약 조건 정보
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 뷰 정보
  - 시퀀스 정보
  - 동의어 정보
  - 인덱스 정보
- 테이블에 주석을 추가하고 디렉터리 뷰를 query하여 주석 정보 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# 데이터 디렉터리

업무 데이터를  
포함하는 테이블:  
**EMPLOYEES**  
**DEPARTMENTS**  
**LOCATIONS**  
**JOB\_HISTORY**  
...



데이터 디렉터리 뷰:  
**DICTIONARY**  
**USER\_OBJECTS**  
**USER\_TABLES**  
**USER\_TAB\_COLUMNS**  
...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 데이터 디렉터리

EMPLOYEES와 같은 유저 테이블은 유저가 생성하는 테이블로 업무 데이터를 포함합니다. 오라클 데이터베이스에는 *데이터 디렉터리*로 알려진 또 다른 테이블 및 뷰 모음이 있습니다. 이 모음은 Oracle 서버에 의해 생성되고 유지 관리되며 데이터베이스에 대한 정보를 포함합니다. 데이터 디렉터리는 다른 데이터베이스 데이터처럼 테이블 및 뷰로 구조화됩니다. 데이터 디렉터리는 모든 오라클 데이터베이스의 핵심일 뿐만 아니라 일반 유저에서 응용 프로그램 설계자나 데이터베이스 관리자에 이르는 모든 유저에게 중요한 도구입니다.

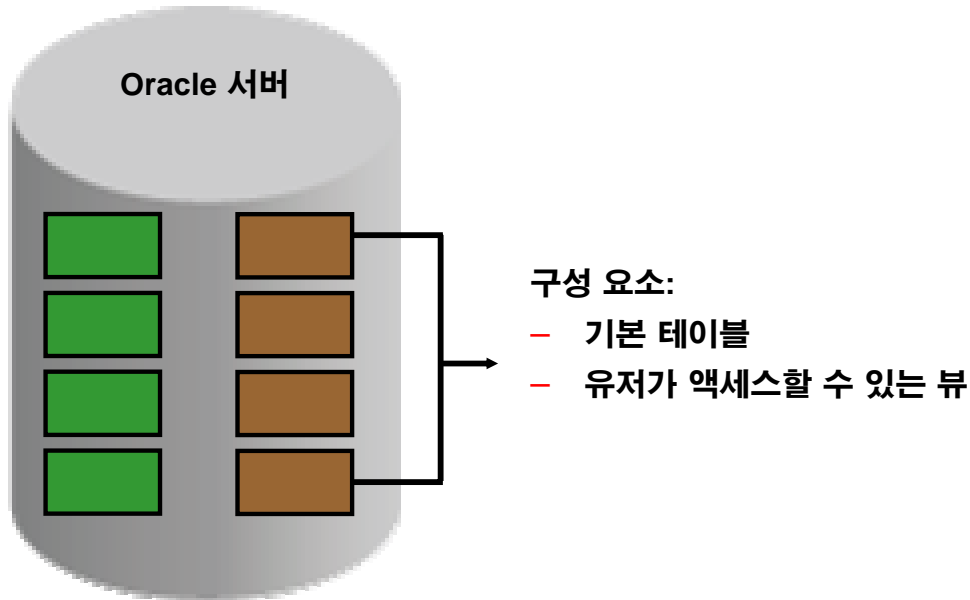
SQL 문을 사용하여 데이터 디렉터리에 액세스할 수 있습니다. 데이터 디렉터리는 읽기 전용이기 때문에 해당 테이블과 뷰에 대해 query만 실행할 수 있습니다.

디렉터리 테이블을 기반으로 하는 디렉터리 뷰를 query하여 다음과 같은 정보를 찾을 수 있습니다.

- 데이터베이스의 모든 스키마 객체(테이블, 뷰, 인덱스, 동의어, 시퀀스, 프로시저, 함수, 패키지, 트리거 등)의 정의
- 열의 기본값
- 무결성 제약 조건 정보
- Oracle 유저 이름
- 각 유저에게 부여된 권한 및 롤
- 기타 일반 데이터베이스 정보



# 데이터 디렉터리 구조



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 데이터 디렉터리 구조

기본 테이블은 연관된 데이터베이스에 대한 정보를 저장합니다. Oracle 서버만 이러한 테이블에 대해 읽기/쓰기를 수행할 수 있습니다. 사용자가 직접 테이블에 액세스하는 경우는 드뭅니다.

데이터 디렉터리의 기본 테이블에 저장된 정보를 요약하고 표시하는 여러 뷰가 있습니다. 이러한 뷰는 정보를 단순화하는 조인 및 WHERE 절을 사용하여 기본 테이블 데이터를 유저 또는 테이블 이름과 같은 유용한 정보로 해석합니다. 대부분의 유저는 기본 테이블이 아니라 뷰에 액세스합니다.

Oracle 유저 SYS는 데이터 디렉터리의 모든 기본 테이블과 유저가 액세스할 수 있는 뷰를 소유합니다. 데이터 무결성이 손상될 수도 있기 때문에 Oracle 유저는 SYS 스키마에 포함된 행이나 스키마 객체를 변경(UPDATE, DELETE 또는 INSERT)해서는 안됩니다.

# 데이터 디렉터리 구조

## 뷰 이름 지정 규칙:

뷰 접두어	목적
USER	유저의 뷰(유저의 스키마에 있는 내용, 유저가 소유한 내용)
ALL	확장된 유저의 뷰(유저가 액세스할 수 있는 내용)
DBA	데이터베이스 관리자의 뷰(모든 사람의 스키마에 있는 내용)
V\$	성능 관련 데이터

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 데이터 디렉터리 구조(계속)

데이터 디렉터리는 뷰 집합으로 구성됩니다. 대부분의 경우 집합은 유사한 정보를 포함하고 접두어를 통해 서로 구분되는 세 개의 뷰로 구성됩니다. 예를 들어, USER\_OBJECTS, ALL\_OBJECTS 및 DBA\_OBJECTS라는 뷰가 있을 수 있습니다.

이러한 세 개의 뷰는 범위가 다르다는 점을 제외하고 데이터베이스의 객체에 대한 유사한 정보를 포함합니다. USER\_OBJECTS는 유저가 소유하거나 생성한 객체에 대한 정보를 포함합니다. ALL\_OBJECTS는 유저가 액세스 권한이 있는 모든 객체에 대한 정보를 포함합니다. DBA\_OBJECTS는 모든 유저가 소유한 모든 객체에 대한 정보를 포함합니다. ALL 또는 DBA 접두어가 붙은 뷰의 경우 대개 객체를 소유한 사람을 식별하기 위해 뷰에 OWNER라는 추가 열이 있습니다.

v\$ 접두어가 붙은 뷰 집합도 있습니다. 이러한 뷰는 특성상 동적이며 성능에 대한 정보를 보유합니다. 동적 성능 테이블은 실제 테이블이 아니며 대부분의 유저가 액세스할 수 없어야 합니다. 그러나 데이터베이스 관리자는 테이블에서 뷰를 query하거나 생성하고 다른 유저에게 이러한 뷰에 대한 액세스 권한을 부여할 수 있습니다. 본 과정에서는 이러한 뷰에 대해 자세히 다루지 않습니다.

# 딕셔너리 뷰 사용 방법

DICTIONARY로 시작합니다. 딕셔너리 테이블 및 뷰의 이름과 설명을 포함합니다.

```
DESCRIBE DICTIONARY
```

Name	Null	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)
2 rows selected		

```
SELECT *  
FROM dictionary  
WHERE table_name = 'USER_OBJECTS';
```

TABLE_NAME	COMMENTS
USER_OBJECTS	Objects owned by the user

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 딕셔너리 뷰 사용 방법

딕셔너리 뷰에 익숙해지려면 DICTIONARY라는 딕셔너리 뷰를 사용하면 됩니다. 여기에는 액세스 권한이 있는 각 딕셔너리 뷰의 이름과 간단한 설명이 포함되어 있습니다.

특정 뷰 이름에 대한 정보를 검색하는 query를 작성하거나 COMMENTS 열에서 특정 단어나 구문을 검색할 수 있습니다. 위의 예제에는 DICTIONARY 뷰가 설명되어 있습니다. 이 뷰에는 두 개의 열이 있습니다. SELECT 문은 USER\_OBJECTS라는 딕셔너리 뷰에 대한 정보를 검색합니다. USER\_OBJECTS 뷰는 사용자가 소유한 모든 객체에 대한 정보를 포함합니다.

COMMENTS 열에서 특정 단어나 구문을 검색하는 query를 작성할 수 있습니다. 예를 들어, 다음 query는 COMMENTS 열에 *columns*라는 단어가 포함된 액세스 권한이 있는 모든 뷰의 이름을 반환합니다.

```
SELECT table_name  
FROM dictionary  
WHERE LOWER(comments) LIKE '%columns%';
```

**참고:** 데이터 딕셔너리의 이름은 대문자입니다.

## USER\_OBJECTS 및 ALL\_OBJECTS 뷰

### USER\_OBJECTS:

- **USER\_OBJECTS**를 query하여 자신이 소유한 모든 객체를 볼 수 있습니다.
- **USER\_OBJECTS**를 사용하면 유저의 스키마에 있는 모든 객체 이름 및 유형 리스트와 함께 다음 정보를 얻을 수 있습니다.
  - 생성된 날짜
  - 마지막 수정 날짜
  - 상태 (valid 또는 invalid)

### ALL\_OBJECTS:

- **ALL\_OBJECTS**를 query하여 액세스 권한이 있는 모든 객체를 봅니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_OBJECTS 및 ALL\_OBJECTS 뷰

USER\_OBJECTS 뷰를 query하여 유저의 스키마에 있는 모든 객체의 이름과 유형을 볼 수 있습니다. 이 뷰에는 다음과 같은 여러 열이 있습니다.

- **OBJECT\_NAME:** 객체 이름입니다.
- **OBJECT\_ID:** 객체의 디렉터리 객체 번호
- **OBJECT\_TYPE:** 객체의 유형(TABLE, VIEW, INDEX, SEQUENCE 등)
- **CREATED:** 객체가 생성된 시간 기록
- **LAST\_DDL\_TIME:** DDL(데이터 정의어) 명령으로 인해 객체가 마지막으로 수정된 시간 기록
- **STATUS:** 객체의 상태(VALID, INVALID 또는 N/A)
- **GENERATED:** 이 객체의 이름을 시스템이 생성했는지 여부 (Y|N)

**참고:** 여기에는 일부 열만 나열되어 있습니다. 전체 리스트는 *Oracle Database Reference*의 "USER\_OBJECTS"를 참조하십시오.

ALL\_OBJECTS 뷰를 query하여 액세스 권한이 있는 모든 객체 리스트를 볼 수도 있습니다.

## USER\_OBJECTS 뷰

```
SELECT object_name, object_type, created, status
FROM   user_objects
ORDER BY object_type;
```

	OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
1	LOC_COUNTRY_IX	INDEX	19-MAY-09	VALID

...

53	EMPLOYEES2	TABLE	22-MAY-09	VALID
54	SECURE_EMPLOYEES	TRIGGER	19-MAY-09	VALID
55	UPDATE_JOB_HISTORY	TRIGGER	19-MAY-09	VALID
56	EMP_DETAILS_VIEW	VIEW	19-MAY-09	VALID

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_OBJECTS 뷰

예제는 이 유저가 소유한 모든 객체의 이름, 유형, 생성 날짜 및 상태를 보여줍니다.

OBJECT\_TYPE 열은 TABLE, VIEW, SEQUENCE, INDEX, PROCEDURE, FUNCTION, PACKAGE 또는 TRIGGER 값 중 하나를 보유합니다.

STATUS 열은 VALID, INVALID 또는 N/A 값을 가집니다. 테이블은 항상 유효하지만 뷰, 프로시저, 함수, 패키지 및 트리거는 유효하지 않을 수도 있습니다.

### CAT 뷰

간단한 query 및 출력의 경우 CAT 뷰를 query할 수 있습니다. 이 뷰는 TABLE\_NAME 및 TABLE\_TYPE의 두 열만 포함합니다. 유저의 모든 INDEX, TABLE, CLUSTER, VIEW, SYNONYM, SEQUENCE 또는 UNDEFINED 객체의 이름을 제공합니다.

**참고:** CAT는 유저가 소유한 테이블, 뷰, 동의어 및 시퀀스를 나열하는 뷰인 USER\_CATALOG의 동의어입니다.

## 단원 내용

- 데이터 디렉터리 소개
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 테이블 정보
  - 열 정보
  - 제약 조건 정보
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 뷰 정보
  - 시퀀스 정보
  - 동의어 정보
  - 인덱스 정보
- 테이블에 주석을 추가하고 디렉터리 뷰를 query하여 주석 정보 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 테이블 정보

### USER\_TABLES:

```
DESCRIBE user_tables
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

...

```
SELECT table_name  
FROM user_tables;
```

TABLE_NAME
1 REGIONS
2 LOCATIONS
3 DEPARTMENTS
4 JOBS
5 EMPLOYEES
6 JOB_HISTORY

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 테이블 정보

USER\_TABLES 뷰를 사용하여 모든 테이블의 이름을 확인할 수 있습니다. USER\_TABLES 뷰는 유저의 테이블에 대한 정보를 포함합니다. 테이블 이름을 제공하는 것 외에도 저장 영역에 대한 자세한 정보를 포함합니다.

TABS 뷰는 USER\_TABLES 뷰의 동의어입니다. 이 뷰를 query하여 유저가 소유한 테이블 리스트를 볼 수 있습니다.

```
SELECT table_name  
FROM tabs;
```

**참고:** USER\_TABLES 뷰의 전체 열 리스트는 *Oracle Database Reference*의 "USER\_TABLES"를 참조하십시오.

ALL\_TABLES 뷰를 query하여 액세스 권한이 있는 모든 테이블 리스트를 볼 수도 있습니다.

## 열 정보

### USER\_TAB\_COLUMNS:

```
DESCRIBE user_tab_columns
```

Name	Null	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 열 정보

USER\_TAB\_COLUMNS 뷰를 query하면 테이블의 열에 대한 상세 정보를 찾을 수 있습니다. USER\_TABLES 뷰는 테이블 이름 및 저장 영역에 대한 정보를 제공하지만 자세한 열 정보는 USER\_TAB\_COLUMNS 뷰에서 찾을 수 있습니다.

이 뷰는 다음과 같은 정보를 포함합니다.

- 열 이름
- 열 데이터 유형
- 데이터 유형의 길이
- NUMBER 열의 전체 자릿수 및 소수점 이하 자릿수
- 널 허용 여부(열에 NOT NULL 제약 조건이 있는지 여부)
- 기본값

**참고:** USER\_TAB\_COLUMNS 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference*의 "USER\_TAB\_COLUMNS"를 참조하십시오.



## 열 정보

```
SELECT column_name, data_type, data_length,  
       data_precision, data_scale, nullable  
FROM   user_tab_columns  
WHERE  table_name = 'EMPLOYEES';
```

	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION
1	EMPLOYEE_ID	NUMBER	22	6
2	FIRST_NAME	VARCHAR2	20	(null)
3	LAST_NAME	VARCHAR2	25	(null)
4	EMAIL	VARCHAR2	25	(null)
5	PHONE_NUMBER	VARCHAR2	20	(null)
6	HIRE_DATE	DATE	7	(null)
7	JOB_ID	VARCHAR2	10	(null)
8	SALARY	NUMBER	22	8
9	COMMISSION_PCT	NUMBER	22	2
10	MANAGER_ID	NUMBER	22	6
11	DEPARTMENT_ID	NUMBER	22	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 열 정보(계속)

USER\_TAB\_COLUMNS 테이블을 query하면 열의 이름, 데이터 유형, 데이터 유형 길이, null 제약 조건 및 기본값 등 열에 대한 상세 정보를 찾을 수 있습니다.

위의 예제는 EMPLOYEES 테이블의 열, 데이터 유형, 데이터 길이 및 널 제약 조건을 표시합니다. 이 정보는 DESCRIBE 명령의 출력과 유사합니다.

unused로 설정된 열에 대한 정보를 보려면 USER\_UNUSED\_COL\_TABS 디렉터리 뷰를 사용하십시오.

**참고:** 데이터 디렉터리의 객체 이름은 대문자입니다.

## 제약 조건 정보

- **USER\_CONSTRAINTS**는 유저의 테이블에 있는 제약 조건 정의를 설명합니다.
- **USER\_CONS\_COLUMNS**는 유저가 소유하고 제약 조건에 지정된 열에 대해 설명합니다.

**DESCRIBE user\_constraints**

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG()
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 제약 조건 정보

제약 조건의 이름, 제약 조건의 유형, 제약 조건이 적용되는 테이블 이름, check 제약 조건에 대한 조건, Foreign key 제약 조건 정보, Foreign key 제약 조건에 대한 삭제 규칙, 상태 및 기타 다양한 유형의 제약 조건 정보를 찾을 수 있습니다.

**참고:** USER\_CONSTRAINTS 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference*의 "USER\_CONSTRAINTS"를 참조하십시오.

## USER\_CONSTRAINTS: 예제

```
SELECT constraint_name, constraint_type,
       search_condition, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	C...	SEARCH_CONDITION	R_CONSTR...	DELET...	STATUS
1	EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL	(null)	(null)	ENABLED
2	EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL	(null)	(null)	ENABLED
3	EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL	(null)	(null)	ENABLED
4	EMP_JOB_NN	C	"JOB_ID" IS NOT NULL	(null)	(null)	ENABLED
5	EMP_SALARY_MIN	C	salary > 0	(null)	(null)	ENABLED
6	EMP_EMAIL_UK	U	(null)	(null)	(null)	ENABLED
7	EMP_EMP_ID_PK	P	(null)	(null)	(null)	ENABLED
8	EMP_DEPT_FK	R	(null)	DEPT_ID_PK	NO ACTION	ENABLED
9	EMP_JOB_FK	R	(null)	JOB_ID_PK	NO ACTION	ENABLED
10	EMP_MANAGER_FK	R	(null)	EMP_EMP_ID_PK	NO ACTION	ENABLED

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_CONSTRAINTS: 예제

위의 예제에서는 USER\_CONSTRAINTS 뷰를 query하여 EMPLOYEES 테이블의 제약 조건에 대한 이름, 유형, check 조건, Foreign key가 참조하는 unique 제약 조건의 이름, Foreign key에 대한 삭제 규칙 및 상태를 찾습니다.

CONSTRAINT\_TYPE은 다음 값이 될 수 있습니다.

- C (테이블의 check 제약 조건 또는 NOT NULL)
- P (Primary key)
- U (Unique key)
- R (참조 무결성)
- V (뷰에서 check 옵션으로 사용)
- O (뷰에서 읽기 전용으로 사용)

DELETE\_RULE은 다음 값이 될 수 있습니다.

- **CASCADE:** 상위 레코드가 삭제되면 하위 레코드도 삭제됩니다.
- **SET NULL:** 상위 레코드가 삭제되면 해당 하위 레코드가 널로 변경됩니다.
- **NO ACTION:** 하위 레코드가 없는 경우에만 상위 레코드를 삭제할 수 있습니다.

STATUS는 다음 값이 될 수 있습니다.

- **ENABLED:** 제약 조건이 활성화됩니다.
- **DISABLED:** 제약 조건이 비활성화됩니다.

## USER\_CONS\_COLUMNS 조회

```
DESCRIBE user_cons_columns
```

Name	Null	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

```
SELECT constraint_name, column_name
FROM   user_cons_columns
WHERE  table_name = 'EMPLOYEES';
```

	CONSTRAINT_NAME	COLUMN_NAME
1	EMP_LAST_NAME_NN	LAST_NAME
2	EMP_EMAIL_NN	EMAIL
3	EMP_HIRE_DATE_NN	HIRE_DATE
4	EMP_JOB_NN	JOB_ID
5	EMP_SALARY_MIN	SALARY
6	EMP_EMAIL_UK	EMAIL

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_CONS\_COLUMNS 조회

제약 조건이 적용된 열 이름을 찾으려면 USER\_CONS\_COLUMNS 디렉터리 뷰를 query하십시오. 이 뷰는 객체 정의에서 제약 조건 소유자의 이름, 제약 조건의 이름, 제약 조건이 있는 테이블, 제약 조건을 가진 열의 이름 및 열이나 속성의 원래 위치를 알려줍니다.

**참고:** 제약 조건은 두 개 이상의 열에도 적용할 수 있습니다.

또한 USER\_CONSTRAINTS와 USER\_CONS\_COLUMNS 사이에 조인을 작성하여 두 테이블에서 커스터마이징된 출력을 생성할 수 있습니다.

## 단원 내용

- 데이터 디렉터리 소개
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 테이블 정보
  - 열 정보
  - 제약 조건 정보
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 뷰 정보
  - 시퀀스 정보
  - 동의어 정보
  - 인덱스 정보
- 테이블에 주석을 추가하고 디렉터리 뷰를 query하여 주석 정보 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 뷰 정보

### 1 DESCRIBE user\_views

Name	Null	Type
-----	-----	-----
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG()

### 2 SELECT view\_name FROM user\_views;

VIEW_NAME
1 EMP_DETAILS_VIEW

### 3 SELECT text FROM user\_views WHERE view\_name = 'EMP\_DETAILS\_VIEW';

TEXT
1 SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.co
...
AND c.region_id = r.region_id AND j.job_id = e.job_id WITH READ ONLY

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 뷰 정보

뷰를 생성한 후에 USER\_VIEWS라는 데이터 디렉터리 뷰를 query하여 뷰의 이름과 뷰 정의를 볼 수 있습니다. 뷰를 구성하는 SELECT 문의 텍스트는 LONG 열에 저장됩니다. LENGTH 열은 SELECT 문의 문자 수입니다. 기본적으로, LONG 열에서 선택할 때 열 값의 처음 80자만 표시됩니다. SQL\*Plus에서 80자 이상을 표시하려면 SET LONG 명령을 사용하십시오.

```
SET LONG 1000
```

슬라이드 예제 설명:

1. USER\_VIEWS 열이 표시됩니다. 이것은 부분 리스트입니다.
2. 뷰의 이름을 검색합니다.
3. 디렉터리에서 EMP\_DETAILS\_VIEW의 SELECT 문이 표시됩니다.

## 뷰를 사용하여 데이터 액세스

뷰를 사용하여 데이터에 액세스할 때 Oracle 서버는 다음 작업을 수행합니다.

- USER\_VIEWS 데이터 디렉터리 테이블에서 뷰 정의를 검색합니다.
- 뷰 기본 테이블의 액세스 권한을 검사합니다.
- 이렇게 하면 뷰 query는 기본 테이블의 해당 작업으로 변환됩니다. 즉, 기본 테이블에서 데이터를 검색하거나 기본 테이블을 갱신합니다.

## 시퀀스 정보

DESCRIBE user\_sequences

Name	Null	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 시퀀스 정보

USER\_SEQUENCES 뷰는 유저가 소유한 모든 시퀀스를 설명합니다. 시퀀스를 생성할 때 USER\_SEQUENCES 뷰에 저장되는 조건을 지정합니다. 이 뷰의 열은 다음과 같습니다.

- **SEQUENCE\_NAME:** 시퀀스의 이름
- **MIN\_VALUE:** 시퀀스의 최소값
- **MAX\_VALUE:** 시퀀스의 최대값
- **INCREMENT\_BY:** 시퀀스의 증분값
- **CYCLE\_FLAG:** 시퀀스가 제한에 도달하면 래핑되는지 여부
- **ORDER\_FLAG:** 시퀀스 번호가 순서대로 생성되는지 여부
- **CACHE\_SIZE:** 캐시할 시퀀스 번호 수
- **LAST\_NUMBER:** 디스크에 기록된 마지막 시퀀스 번호. 시퀀스가 캐시를 사용하는 경우 디스크에 기록된 번호는 시퀀스 캐시에 배치된 마지막 번호입니다. 이 번호는 대개 사용된 마지막 시퀀스 번호보다 큼니다.

## 시퀀스 확인

- **USER\_SEQUENCES 데이터 디렉터리 테이블에서 시퀀스 값을 확인합니다.**

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

R	SEQUENCE_NAME	R	MIN_VALUE	R	MAX_VALUE	R	INCREMENT_BY	R	LAST_NUMBER
1	DEPARTMENTS_SEQ		1		9990		10		280
2	EMPLOYEES_SEQ		1		999999999999999...		1		207
3	LOCATIONS_SEQ		1		9900		100		3300

- **NOCACHE가 지정된 경우 LAST\_NUMBER 열은 사용 가능한 다음 시퀀스를 표시합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 시퀀스 확인

시퀀스를 생성하면 데이터 디렉터리에 해당 시퀀스가 기록됩니다. 시퀀스는 데이터베이스 객체이므로 USER\_OBJECTS 데이터 디렉터리 테이블에서 식별할 수 있습니다.

USER\_SEQUENCES 데이터 디렉터리 뷰에서 시퀀스를 선택하여 시퀀스 설정을 확인할 수도 있습니다.

#### 시퀀스를 증분하지 않고 사용 가능한 다음 시퀀스 값 확인

NOCACHE로 시퀀스를 생성한 경우 USER\_SEQUENCES 테이블을 query하여 시퀀스를 증가시키지 않고도 사용 가능한 다음 시퀀스 값을 확인할 수 있습니다.



## 인덱스 정보

- **USER\_INDEXES**는 인덱스에 대한 정보를 제공합니다.
- **USER\_IND\_COLUMNS**는 인덱스로 구성된 열과 테이블의 인덱스 열에 대해 설명합니다.

```
DESCRIBE user_indexes
```

Name	Null	Type
-----		
INDEX_NAME	NOT NULL	VARCHAR2(30)
INDEX_TYPE		VARCHAR2(27)
TABLE_OWNER	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLE_TYPE		VARCHAR2(11)
UNIQUENESS		VARCHAR2(9)

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 인덱스 정보

USER\_INDEXES 뷰를 query하면 인덱스 이름, 인덱스가 생성된 테이블 이름 및 고유 인덱스인지 여부를 확인할 수 있습니다.

**참고:** USER\_INDEXES 뷰의 전체 열 리스트와 설명은 *Oracle Database Reference 11g Release 2(11.1)*의 "USER\_INDEXES"를 참조하십시오.

## USER\_INDEXES: 예제

**a** `SELECT index_name, table_name, uniqueness  
FROM user_indexes  
WHERE table_name = 'EMPLOYEES';`

	INDEX_NAME	TABLE_NAME	UNIQUENESS
1	EMP_EMAIL_UK	EMPLOYEES	UNIQUE
2	EMP_EMP_ID_PK	EMPLOYEES	UNIQUE
3	EMP_DEPARTMENT_IX	EMPLOYEES	NONUNIQUE
4	EMP_JOB_IX	EMPLOYEES	NONUNIQUE
5	EMP_MANAGER_IX	EMPLOYEES	NONUNIQUE
6	EMP_NAME_IX	EMPLOYEES	NONUNIQUE

**b** `SELECT index_name, table_name  
FROM user_indexes  
WHERE table_name = 'emp_lib';`

	INDEX_NAME	TABLE_NAME
1	SYS_C0011777	EMP_LIB

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_INDEXES: 예제

위 슬라이드 예제 **a**에서 USER\_INDEXES 뷰를 query하면 인덱스 이름, 인덱스가 생성된 테이블 이름 및 고유 인덱스인지 여부를 확인할 수 있습니다.

위 슬라이드 예제 **b**에서 Oracle 서버는 PRIMARY KEY 열에 대해 생성되는 인덱스에 일반적인 이름을 지정합니다. EMP\_LIB 테이블은 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE EMP_LIB  
(book_id NUMBER(6) PRIMARY KEY ,  
title VARCHAR2(25),  
category VARCHAR2(20));
```

CREATE TABLE succeeded.

## USER\_IND\_COLUMNS 조회

```
DESCRIBE user_ind_columns
```

Name	Null	Type
INDEX_NAME		VARCHAR2(30)
TABLE_NAME		VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
COLUMN_POSITION		NUMBER
COLUMN_LENGTH		NUMBER
CHAR_LENGTH		NUMBER
DESCEND		VARCHAR2(4)

```
SELECT index_name, column_name, table_name
FROM   user_ind_columns
WHERE  index_name = 'lname_idx';
```

	INDEX_NAME	COLUMN_NAME	TABLE_NAME
1	LNAME_IDX	LAST_NAME	EMP_TEST

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### USER\_IND\_COLUMNS 조회

USER\_IND\_COLUMNS 디렉터리 뷰는 인덱스 이름, 인덱스화된 테이블의 이름, 인덱스 내에 있는 열의 이름 및 인덱스 내에서 열의 위치 등의 정보를 제공합니다.

위 슬라이드 예제에서 emp\_test 테이블과 LNAME\_IDX 인덱스는 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE emp_test AS SELECT * FROM employees;
CREATE INDEX LNAME_IDX ON emp_test(Last_Name);
```

## 동의어 정보

```
DESCRIBE user_synonyms
```

Name	Null	Type
SYNONYM_NAME	NOT NULL	VARCHAR2(30)
TABLE_OWNER		VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
DB_LINK		VARCHAR2(128)

```
SELECT *  
FROM user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	TEAM2	ORA22	DEPARTMENTS	{null}

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 동의어 정보

USER\_SYNONYMS 디렉터리 뷰는 전용 (private) 동의어 (유저가 소유한 동의어) 에 대해 설명합니다.

이 뷰를 query하면 동의어를 찾을 수 있습니다. ALL\_SYNONYMS를 query하면 유저에게 제공되는 모든 동의어 이름 및 이러한 동의어가 적용되는 객체를 찾을 수 있습니다.

이 뷰의 열은 다음과 같습니다.

- **SYNONYM\_NAME:** 동의어의 이름
- **TABLE\_OWNER:** 동의어를 통해 참조되는 객체의 소유자
- **TABLE\_NAME:** 동의어를 통해 참조되는 테이블이나 뷰의 이름
- **DB\_LINK:** 데이터베이스 링크 참조의 이름(있는 경우)

## 단원 내용

- 데이터 디렉터리 소개
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 테이블 정보
  - 열 정보
  - 제약 조건 정보
- 디렉터리 뷰를 query하여 다음 정보 검색
  - 뷰 정보
  - 시퀀스 정보
  - 동의어 정보
  - 인덱스 정보
- 테이블에 주석을 추가하고 디렉터리 뷰를 query하여 주석 정보 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 테이블에 주석 추가

- **COMMENT 문을 사용하여 테이블이나 열에 주석을 추가할 수 있습니다.**

```
COMMENT ON TABLE employees  
IS 'Employee Information';
```

```
COMMENT ON COLUMN employees.first_name  
IS 'First name of the employee';
```

- **주석은 데이터 디렉터리 뷰를 통해 볼 수 있습니다.**
  - ALL\_COL\_COMMENTS
  - USER\_COL\_COMMENTS
  - ALL\_TAB\_COMMENTS
  - USER\_TAB\_COMMENTS

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 테이블에 주석 추가

COMMENT 문을 사용하여 열, 테이블, 뷰 또는 스냅샷에 대해 최대 4,000바이트의 주석을 추가할 수 있습니다. 주석은 데이터 디렉터리에 저장되며 COMMENTS 열의 다음 데이터 디렉터리 뷰 중 하나를 통해 볼 수 있습니다.

- ALL\_COL\_COMMENTS
- USER\_COL\_COMMENTS
- ALL\_TAB\_COMMENTS
- USER\_TAB\_COMMENTS

### 구문

```
COMMENT ON {TABLE table | COLUMN table.column}  
IS 'text';
```

이 구문에서 다음이 적용됩니다.

*table*    테이블의 이름입니다.  
*column*    테이블에 있는 열의 이름입니다.  
*text*    주석의 텍스트입니다.

주석을 빈 문자열('')로 설정하는 방식으로 데이터베이스에서 주석을 삭제할 수 있습니다.

```
COMMENT ON TABLE employees IS '';
```

## 퀴즈

딕셔너리 테이블을 기반으로 하는 딕셔너리 뷰에는 다음과 같은 정보가 들어 있습니다.

1. 데이터베이스의 모든 스키마 객체에 대한 정의
2. 열의 기본값
3. 무결성 제약 조건 정보
4. 각 유저에게 부여된 권한 및 롤
5. 위 항목 모두

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 5

## 요약

이 단원에서는 다음 디렉터리 뷰를 통해 객체에 대한 정보를 찾는 방법을 배웠습니다.

- **DICTIONARY**
- **USER\_OBJECTS**
- **USER\_TABLES**
- **USER\_TAB\_COLUMNS**
- **USER\_CONSTRAINTS**
- **USER\_CONS\_COLUMNS**
- **USER\_VIEWS**
- **USER\_SEQUENCES**
- **USER\_INDEXES**
- **USER\_SYNONYMS**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

이 단원에서는 유저에게 제공되는 일부 디렉터리 뷰에 대해 배웠습니다. 이러한 디렉터리 뷰를 사용하여 테이블, 제약 조건, 뷰, 시퀀스 및 동의어에 대한 정보를 찾을 수 있습니다.



## 연습 3: 개요

이 연습에서는 다음 내용을 다룹니다.

- 디셔너리 뷰를 query하여 테이블 및 열 정보 검색
- 디셔너리 뷰를 query하여 제약 조건 정보 검색
- 디셔너리 뷰를 query하여 뷰 정보 검색
- 디셔너리 뷰를 query하여 시퀀스 정보 검색
- 디셔너리 뷰를 query하여 동의어 정보 검색
- 디셔너리 뷰를 query하여 인덱스 정보 검색
- 테이블에 주석을 추가하고 디셔너리 뷰를 query하여 주석 정보 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 3: 개요

이 연습에서는 디셔너리 뷰를 query하여 스키마의 객체에 대한 정보를 찾습니다.



# 4

## 대형 데이터 집합 조작

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다중 테이블 INSERT의 기능 설명
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 subquery를 사용하여 오라클 데이터베이스의 데이터를 조작하는 방법에 대해 설명합니다. INSERT 문과 UPDATE 문에서 DEFAULT 키워드를 사용하여 기본 열 값을 식별하는 방법과 다중 테이블 INSERT 문, MERGE 문 및 데이터베이스의 변경 사항 추적에 대해 설명합니다.

## 단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only

# Subquery를 사용하여 데이터 조작

DML(데이터 조작어) 문의 subquery를 사용하여 다음 작업을 수행할 수 있습니다.

- 인라인 뷰에서 데이터 검색
- 테이블 간에 데이터 복사
- 다른 테이블의 값을 기반으로 테이블의 데이터 갱신
- 다른 테이블의 행을 기반으로 테이블에서 행 삭제

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Subquery를 사용하여 데이터 조작

Subquery를 사용하면 테이블에서 데이터를 검색하여 다른 테이블에서 INSERT에 대한 입력으로 사용할 수 있습니다. 이러한 방식으로 단일 SELECT 문을 사용하여 테이블 간에 대량의 데이터를 쉽게 복사할 수 있습니다. 마찬가지로 UPDATE 문과 DELETE 문의 WHERE 절에서 subquery를 사용하여 대량 갱신 및 삭제를 수행할 수 있습니다. 또한 SELECT 문의 FROM 절에서도 subquery를 사용할 수 있습니다. 이를 인라인 뷰라고 합니다.

**참고:** *Oracle Database 11g: SQL Fundamentals I* 과정에서 다른 테이블을 기반으로 행을 갱신하고 삭제하는 방법을 배웠습니다.

## Subquery를 소스로 사용하여 데이터 검색

```
SELECT department_name, city
FROM departments
NATURAL JOIN (SELECT l.location_id, l.city, l.country_id
               FROM loc l
               JOIN countries c
               ON(l.country_id = c.country_id)
               JOIN regions USING(region_id)
               WHERE region_name = 'Europe');
```

	DEPARTMENT_NAME	CITY
1	Human Resources	London
2	Sales	Oxford
3	Public Relations	Munich

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Subquery를 소스로 사용하여 데이터 검색

SELECT 문의 FROM 절에 subquery를 사용할 수 있으며 그 방법은 뷰 사용 방법과 비슷합니다. SELECT 문의 FROM 절에 있는 subquery를 *인라인 뷰*라고도 합니다. SELECT 문의 FROM 절에 있는 subquery는 해당하는 특정 SELECT 문과 해당 SELECT 문에 대한 데이터 소스를 정의합니다. 데이터베이스 뷰와 마찬가지로 subquery의 SELECT 문은 필요에 따라 간단하거나 복잡할 수 있습니다.

데이터베이스 뷰가 생성될 때 연관된 SELECT 문이 데이터 디렉터리에서 저장됩니다.

데이터베이스 뷰를 생성하는 데 필요한 권한이 없거나 SELECT 문의 뷰 적합성을 테스트하는 경우에 인라인 뷰를 사용할 수 있습니다.

인라인 뷰에서는 query를 지원하는 데 필요한 모든 코드를 제공합니다. 즉, 복잡하게 데이터베이스 뷰를 별도로 생성하지 않아도 됩니다. 슬라이드의 예제는 인라인 뷰를 사용하여 유럽의 부서 이름과 도시를 표시하는 방법을 보여줍니다. FROM 절의 subquery는 3개의 서로 다른 테이블을 조인하여 위치 ID, 도시 이름 및 국가를 패치(fetch)합니다. Inner query의 출력은 outer query에 대한 테이블로 간주됩니다. Inner query는 데이터베이스 뷰의 query와 유사하지만 물리적 이름이 없습니다.

슬라이드의 예제에서는 다음 명령문을 실행하여 loc 테이블을 생성합니다.

```
CREATE TABLE loc AS SELECT * FROM locations;
```

## subquery를 소스로 사용하여 데이터 검색(계속)

다음 두 단계를 수행하면 슬라이드의 예제와 동일한 결과를 얻을 수 있습니다.

1. 데이터베이스 뷰를 생성합니다.

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe';
```

2. EUROPEAN\_CITIES 뷰를 DEPARTMENTS 테이블과 조인합니다.

```
SELECT department_name, city
FROM   departments
NATURAL JOIN european_cities;
```

**참고:** *Oracle Database 11g: SQL Fundamentals I* 과정에서 데이터베이스 뷰를 생성하는 방법을 배웠습니다.



## Subquery를 대상으로 사용하여 삽입

```
INSERT INTO (SELECT l.location_id, l.city, l.country_id
              FROM   locations l
              JOIN   countries c
              ON(l.country_id = c.country_id)
              JOIN regions USING(region_id)
              WHERE region_name = 'Europe')
VALUES (3300, 'Cardiff', 'UK');
```

1 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Subquery를 대상으로 사용하여 삽입

INSERT 문의 INTO 절에서 테이블 이름 자리에 subquery를 사용할 수 있습니다. 이 subquery의 SELECT 리스트와 VALUES 절 열 리스트의 열 개수는 동일해야 합니다. INSERT 문의 성공적으로 실행되려면 기본 테이블 열의 모든 규칙을 따라야 합니다. 예를 들어, 중복된 위치 ID를 입력하거나 필수 NOT NULL 열의 값을 생략할 수 없습니다.

이러한 subquery를 사용하면 단지 INSERT를 수행하기 위해 뷰를 생성하지 않아도 됩니다.

슬라이드의 예제는 LOC 대신 subquery를 사용하여 새 유럽 도시에 대한 레코드를 생성합니다.

**참고:** 다음 코드를 사용하여 EUROPEAN\_CITIES 뷰에서 INSERT 작업을 수행할 수도 있습니다.

```
INSERT INTO european_cities
VALUES (3300, 'Cardiff', 'UK');
```

## Subquery를 대상으로 사용하여 삽입

결과를 확인합니다.

```
SELECT location_id, city, country_id
FROM   loc
```

	LOCATION_ID	CITY	COUNTRY_ID
20	2900	Geneva	CH
21	3000	Bern	CH
22	3100	Utrecht	NL
23	3200	Mexico City	MX
24	3300	Cardiff	UK

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Subquery를 대상으로 사용하여 삽입(계속)

슬라이드의 예제는 인라인 뷰를 통한 삽입으로 인해 기본 테이블 LOC에 새 레코드가 생성되었음을 보여줍니다.

다음 예제는 INSERT 문에서 테이블을 식별하는 데 사용된 subquery의 결과를 보여줍니다.

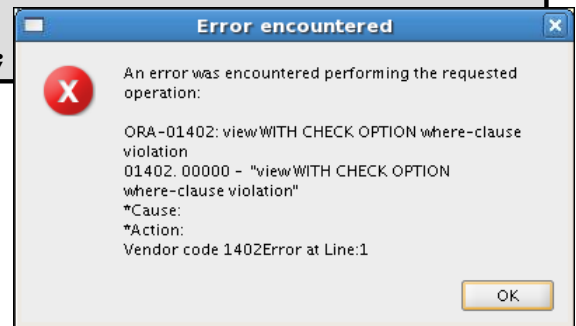
```
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN   regions USING(region_id)
WHERE  region_name = 'Europe'
```

	LOCATION_ID	CITY	COUNTRY_ID
6	2700	Munich	DE
7	2900	Geneva	CH
8	3000	Bern	CH
9	3100	Utrecht	NL
10	3300	Cardiff	UK

## DML 문에 WITH CHECK OPTION 키워드 사용

WITH CHECK OPTION 키워드를 사용하면 subquery에 없는 행은 변경할 수 없습니다.

```
INSERT INTO ( SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
                    (SELECT country_id
                     FROM countries
                     NATURAL JOIN regions
                     WHERE region_name = 'Europe' )
              WITH CHECK OPTION )
VALUES (3600, 'Washington', 'US');
```



Copyright © 2009, Oracle. All rights reserved.

### DML 문에 WITH CHECK OPTION 키워드 사용

INSERT, UPDATE, DELETE 문의 테이블 자리에 subquery가 사용되는 경우 해당 테이블에 대해 subquery에 포함되지 않은 행을 생성하도록 변경하지 못하게 하려면 WITH CHECK OPTION 키워드를 지정합니다.

슬라이드의 예제는 인라인 뷰를 WITH CHECK OPTION과 함께 사용하는 방법을 보여줍니다. INSERT 문은 유럽에 없는 도시에 대해 LOC 테이블에 레코드를 생성하지 못하게 합니다.

다음 예제는 VALUES 리스트의 변경 사항 때문에 성공적으로 실행됩니다.

```
INSERT INTO (SELECT location_id, city, country_id
              FROM   loc
              WHERE  country_id IN
                    (SELECT country_id
                     FROM countries
                     NATURAL JOIN regions
                     WHERE region_name = 'Europe' )
              WITH CHECK OPTION)
VALUES (3500, 'Berlin', 'DE');
```

## DML 문에 WITH CHECK OPTION 키워드 사용(계속)

인라인 뷰에 WITH CHECK OPTION을 사용하여 간단하게 테이블을 변경하지 못하게 할 수 있습니다.

비유럽 도시의 생성을 방지하기 위해 다음 단계를 수행하여 데이터베이스 뷰를 사용할 수도 있습니다.

1. 데이터베이스 뷰를 생성합니다.

```
CREATE OR REPLACE VIEW european_cities
AS
SELECT location_id, city, country_id
FROM   locations
WHERE  country_id in
      (SELECT country_id
       FROM countries
       NATURAL JOIN regions
       WHERE region_name = 'Europe')
WITH CHECK OPTION;
```

2. 데이터를 삽입하여 결과를 확인합니다.

```
INSERT INTO european_cities
VALUES (3400, 'New York', 'US');
```

두번째 단계를 수행하면 슬라이드에 표시된 것과 동일한 오류가 발생합니다.

## 단원 내용

- Subquery를 사용하여 데이터 조작
- **INSERT 및 UPDATE 문에서 명시적 기본값 지정**
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 명시적 기본값 기능의 개요

- 기본 열 값이 필요한 경우 **DEFAULT** 키워드를 열 값으로 사용합니다.
- 이 기능을 사용하면 사용자가 기본값을 데이터에 적용해야 하는 위치 및 시기를 제어할 수 있습니다.
- 명시적 기본값은 **INSERT** 문과 **UPDATE** 문에서 사용할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 명시적 기본값

**DEFAULT** 키워드는 **INSERT** 문과 **UPDATE** 문에서 기본 열 값을 식별하는 데 사용할 수 있습니다. 기본값이 존재하지 않는 경우 널 값이 사용됩니다.

**DEFAULT** 옵션이 도입됨에 따라 이전처럼 프로그램에서 기본값을 하드 코딩하거나 기본값을 찾기 위해 디렉터리를 query할 필요가 없어졌습니다. 기본값을 하드 코딩하면 기본값이 변경될 경우 그에 따라 코드도 변경해야 하기 때문에 문제가 됩니다. 디렉터리 액세스 기능은 대개 응용 프로그램에서 수행되지 않으므로 이 기능은 매우 중요한 기능입니다.

## 명시적 기본값 사용

- INSERT 문에서 DEFAULT 사용:

```
INSERT INTO deptm3
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

- UPDATE 문에서 DEFAULT 사용:

```
UPDATE deptm3
SET manager_id = DEFAULT
WHERE department_id = 10;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 명시적 기본값 사용

이전에 열에 대한 기본값으로 지정된 값으로 열을 설정하려면 DEFAULT를 지정합니다.

해당 열에 대해 기본값이 지정되지 않으면 Oracle 서버에서 열을 널로 설정합니다.

슬라이드의 첫번째 예제에서 INSERT 문은 MANAGER\_ID 열에 대해 기본값을 사용합니다. 열에 정의된 기본값이 없는 경우 대신 널 값이 삽입됩니다.

두번째 예제는 UPDATE 문을 사용하여 MANAGER\_ID 열을 부서 10에 대한 기본값으로 설정합니다. 열에 대해 기본값이 정의되지 않은 경우 널 값으로 변경됩니다.

**참고:** 테이블을 생성할 때 열에 대한 기본값을 지정할 수 있습니다. 이에 대한 자세한 내용은 *SQL Fundamentals I*을 참조하십시오.

## 다른 테이블에서 행 복사

- **INSERT 문을 subquery로 작성합니다.**

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

33 rows inserted

- **VALUES 절을 사용하지 마십시오.**
- **INSERT 절의 열 개수와 subquery의 열 개수를 일치시킵니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 다른 테이블에서 행 복사

INSERT 문을 사용하여 기존 테이블에서 파생된 값으로 테이블에 행을 추가할 수 있습니다. VALUES 절 자리에 subquery를 사용합니다.

#### 구문

```
INSERT INTO table [ column (, column) ] subquery;
```

이 구문에서 다음이 적용됩니다.

*table*           테이블 이름입니다.  
*column*         테이블에 채울 열의 이름입니다.  
*subquery*       테이블에 행을 반환하는 subquery입니다.

INSERT 절의 열 리스트에 나오는 열 개수 및 해당 데이터 유형은 subquery의 값 개수 및 해당 데이터 유형과 일치해야 합니다. 테이블 행의 복사본을 생성하려면 subquery에서 SELECT \*를 사용합니다.

```
INSERT INTO EMPL3
SELECT *
FROM employees;
```

**참고:** DML 문에서 LOG ERRORS 절을 사용하여 오류에 관계없이 DML 작업을 완료할 수 있습니다. 오라클은 사용자가 생성한 오류 로깅 테이블에 오류 메시지의 세부 정보를 기록합니다. 자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.



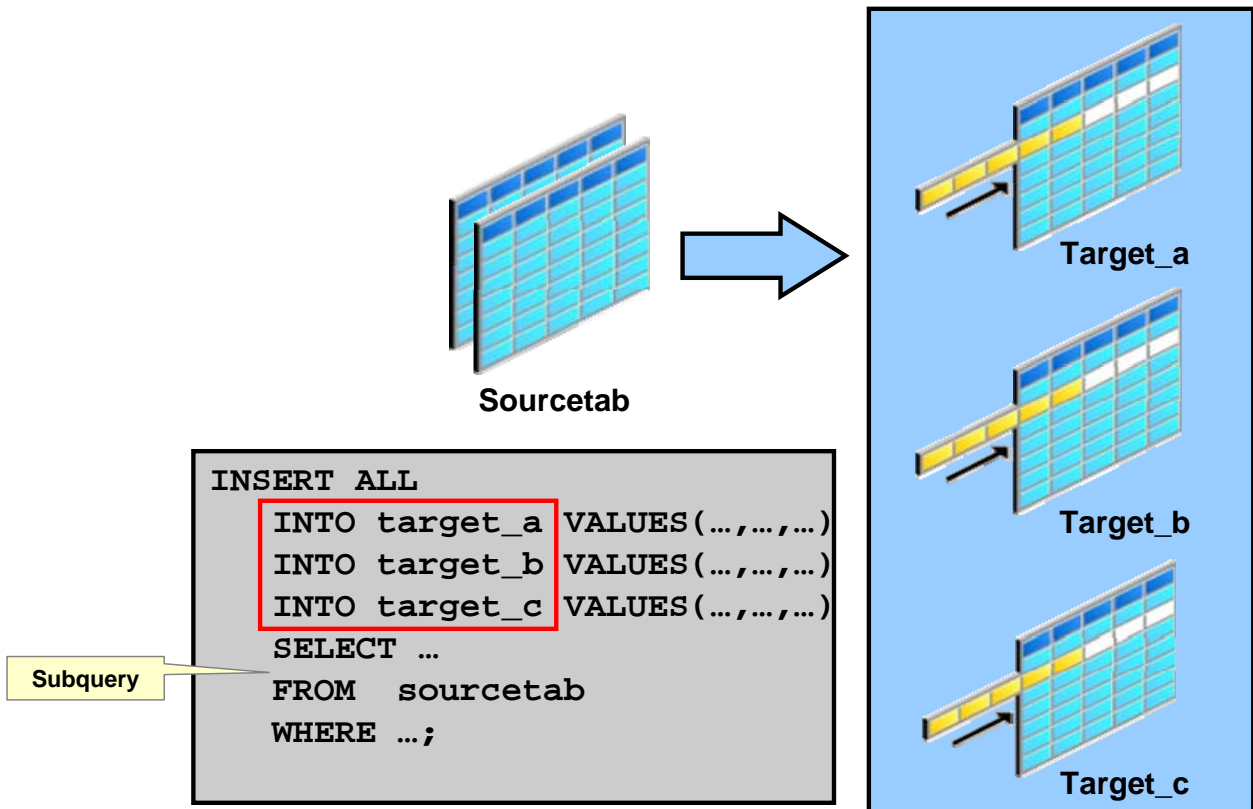
## 단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 다중 테이블 INSERT 문의 개요



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 다중 테이블 INSERT 문의 개요

Subquery 평가 결과 반환된 행에서 파생된 계산된 행을 다중 테이블 INSERT 문의 테이블에 삽입합니다.

다중 테이블 INSERT 문은 데이터 웨어하우스 시나리오에서 유용합니다. 업무 분석을 더욱 효율적으로 수행하려면 데이터 웨어하우스를 정기적으로 로드해야 합니다. 이렇게 하려면 하나 이상의 운영 시스템에서 데이터를 추출하여 웨어하우스에 복사해야 합니다. 소스 시스템에서 데이터를 추출하여 데이터 웨어하우스로 가져오는 과정을 일반적으로 ETL이라고 하며, 이는 추출(Extraction), 변형(Transformation) 및 로드>Loading)를 의미합니다.

추출 과정에서는 데이터베이스 시스템이나 응용 프로그램과 같은 여러 소스에서 원하는 데이터를 식별하여 추출해야 합니다. 추출이 끝나면 데이터를 추가로 처리할 수 있도록 대상 시스템이나 중개 시스템으로 데이터를 물리적으로 전송해야 합니다. 선택한 전송 방식에 따라 프로세스 도중에 일부 변형이 수행될 수 있습니다. 예를 들어 게이트웨이를 통해 원격 대상에 직접 액세스하는 SQL 문은 두 개 열을 SELECT 문의 일부로 연결할 수 있습니다.

데이터가 오라클 데이터베이스에 로드되면 SQL 작업을 통해 데이터 변형을 실행할 수 있습니다. 다중 테이블 INSERT 문은 SQL 데이터 변형을 구현하는 기술 중 하나입니다.

## 다중 테이블 INSERT 문의 개요

- **INSERT...ELECT 문을 사용하여 행을 단일 DML 문의 일부로 다중 테이블에 삽입합니다.**
- **데이터 웨어하우징 시스템에서 다중 테이블 INSERT 문을 사용하여 하나 이상의 운영 소스에서 대상 테이블 집합으로 데이터를 전송합니다.**
- **다음과 같은 비교를 통해 성능이 크게 향상됨을 확인할 수 있습니다.**
  - 단일 DML 문과 다중 INSERT...ELECT 문 비교
  - 단일 DML과 IF...THEN 구문을 사용하여 다중 삽입을 수행하는 프로시저 비교

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 다중 테이블 INSERT 문의 개요(계속)

다중 테이블 INSERT 문은 다중 테이블이 대상으로 포함된 경우 INSERT ... SELECT 문의 이점을 제공합니다. 다중 테이블 INSERT를 사용하지 않을 경우  $n$ 개의 독립적인 INSERT ... SELECT 문을 처리해야 하므로 동일한 소스 데이터를  $n$ 번 처리하고 변환 작업 로드도  $n$ 배 늘어납니다.

기존의 INSERT ... SELECT 문을 사용하는 경우와 마찬가지로, 새 명령문을 병렬화하고 직접 로드 방식과 함께 사용하여 성능을 향상시킬 수 있습니다.

이제 비관계형 데이터베이스 테이블과 같은 입력 스트림의 각 레코드를 관계형 데이터베이스 테이블 환경을 위한 다중 레코드로 변환할 수 있습니다. 다른 방식으로 이 기능을 구현하려면 다중 INSERT 문을 작성해야 했습니다.

## 다중 테이블 INSERT 문의 유형

다중 테이블 INSERT 문에는 다음과 같은 다양한 유형이 있습니다.

- 무조건 INSERT
- 조건부 INSERT ALL
- 피벗팅 INSERT
- 조건부 INSERT FIRST

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 다중 테이블 INSERT 문의 유형

다른 절을 사용하여 실행할 INSERT 유형을 나타낼 수 있습니다. 다중 테이블 INSERT 문에는 다음과 같은 유형이 있습니다.

- 무조건 **INSERT**: subquery에 의해 반환된 각 행에 대해 각 대상 테이블에 행이 삽입됩니다.
- 조건부 **INSERT ALL**: subquery에 의해 반환된 각 행에 대해 지정된 조건이 충족되는 경우 각 대상 테이블에 행이 삽입됩니다.
- 피벗팅 **INSERT**: 조건부 INSERT ALL의 특별한 경우입니다.
- 조건부 **INSERT FIRST**: subquery에 의해 반환된 각 행에 대해 조건이 충족되는 첫번째 대상 테이블에 행이 삽입됩니다.

## 다중 테이블 INSERT 문

- 다중 테이블 INSERT의 구문:

```
INSERT [conditional_insert_clause]
[insert_into_clause values_clause] (subquery)
```

- conditional\_insert\_clause:

```
[ALL|FIRST]
[WHEN condition THEN] [insert_into_clause values_clause]
[ELSE] [insert_into_clause values_clause]
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 다중 테이블 INSERT 문

슬라이드는 다중 테이블 INSERT 문의 일반 형식을 보여줍니다.

#### 무조건 INSERT: ALL into\_clause

무조건 다중 테이블 INSERT를 수행하려면 ALL 다음에 여러 개의 insert\_into\_clause를 지정합니다. Oracle 서버는 subquery에 의해 반환된 각 행에 대해 각 insert\_into\_clause를 한 번 실행합니다.

#### 조건부 INSERT: conditional\_insert\_clause

조건부 다중 테이블 INSERT를 수행하려면 conditional\_insert\_clause를 지정합니다. Oracle 서버는 해당 WHEN 조건을 통해 각 insert\_into\_clause를 필터링하여 해당 insert\_into\_clause가 실행되는지 여부를 판별합니다. 하나의 다중 테이블 INSERT 문은 최대 127개의 WHEN 절을 포함할 수 있습니다.

#### 조건부 INSERT: ALL

ALL을 지정하면 Oracle 서버는 다른 WHEN 절의 평가 결과에 관계없이 각 WHEN 절을 평가합니다. 조건이 true로 평가된 각 WHEN 절에 대해 Oracle 서버는 해당 INTO 절 리스트를 실행합니다.

## 다중 테이블 INSERT 문(계속)

### 조건부 INSERT: FIRST

FIRST를 지정하면 Oracle 서버는 각 WHEN 절을 명령문에 나타난 순서대로 평가합니다. 첫번째 WHEN 절이 True로 평가되면 Oracle 서버는 해당 INTO 절을 실행하고 주어진 행에 대해 다음 WHEN 절은 건너뛵니다.

### 조건부 INSERT: ELSE 절

주어진 행에 대해 True로 평가된 WHEN 절이 없는 경우:

- ELSE 절을 지정한 경우 Oracle 서버는 ELSE 절과 연관된 INTO 절 리스트를 실행합니다.
- ELSE 절을 지정하지 않은 경우 Oracle 서버는 해당 행에 대해 아무 작업도 수행하지 않습니다.

### 다중 테이블 INSERT 문에 대한 제한 사항

- 뷰 또는 Materialized View가 아닌 테이블에 대해서만 다중 테이블 INSERT 문을 수행할 수 있습니다.
- 원격 테이블에서는 다중 테이블 INSERT를 수행할 수 없습니다.
- 다중 테이블 INSERT를 수행할 때 테이블 컬렉션 표현식을 지정할 수 없습니다.
- 다중 테이블 INSERT에서 모든 insert\_into\_clause를 결합하여 999개를 초과하는 대상 열을 지정할 수 없습니다.

## 무조건 INSERT ALL

- EMPLOYEES 테이블에서 EMPLOYEE\_ID가 200보다 큰 사원의 EMPLOYEE\_ID, HIRE\_DATE, SALARY 및 MANAGER\_ID 값을 선택합니다.
- 다중 테이블 INSERT를 사용하여 SAL\_HISTORY 및 MGR\_HISTORY 테이블에 이 값을 삽입합니다.

```
INSERT ALL
  INTO sal_history VALUES(EMPID,HIREDATE,SAL)
  INTO mgr_history VALUES(EMPID,MGR,SAL)
  SELECT employee_id EMPID, hire_date HIREDATE,
         salary SAL, manager_id MGR
  FROM   employees
  WHERE  employee_id > 200;
```

12 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 무조건 INSERT ALL

슬라이드의 예제는 행을 SAL\_HISTORY 테이블과 MGR\_HISTORY 테이블에 모두 삽입합니다.

SELECT 문은 EMPLOYEES 테이블에서 사원 ID가 200보다 큰 사원의 사원 ID, 채용 날짜, 급여 및 관리자 ID의 세부 사항을 검색합니다. 사원 ID, 채용 날짜 및 급여의 세부 정보가 SAL\_HISTORY 테이블에 삽입됩니다. 사원 ID, 관리자 ID 및 급여 세부 사항을 MGR\_HISTORY 테이블에 삽입합니다.

이 INSERT 문은 SELECT 문으로 검색된 행에 더 이상의 제한이 적용되지 않으므로 무조건 INSERT라고 합니다. SELECT 문으로 검색된 모든 행은 SAL\_HISTORY와 MGR\_HISTORY라는 두 테이블에 삽입됩니다. INSERT 문의 VALUES 절은 SELECT 문에서 각 테이블에 삽입해야 하는 열을 지정합니다. SELECT 문에 의해 반환된 각 행은 SAL\_HISTORY 테이블과 MGR\_HISTORY 테이블에 각각 하나씩 삽입됩니다.

## 무조건 INSERT ALL(계속)

총 12개의 행이 선택되었습니다.

```
SELECT COUNT(*) total_in_sal FROM sal_history;
```

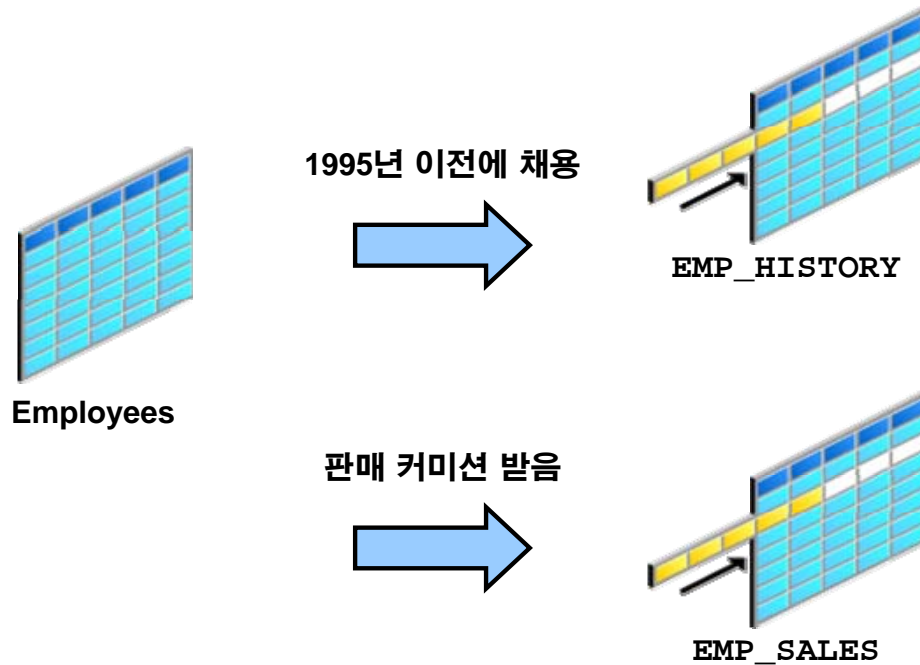
	A 2	TOTAL_IN_SAL
1		6

```
SELECT COUNT(*) total_in_mgr FROM mgr_history;
```

	A 2	TOTAL_IN_MGR
1		6



## 조건부 INSERT ALL: 예제



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 조건부 INSERT ALL: 예제

Employees 테이블의 모든 사원 중 1995년 이전에 채용된 사원의 레코드를 사원 기록에 삽입합니다. 사원이 판매 커미션을 받는 경우 EMP\_SALES 테이블에 레코드 정보를 삽입합니다. SQL 문은 다음 페이지에 표시됩니다.

## 조건부 INSERT ALL

```
INSERT ALL
  WHEN HIREDATE < '01-JAN-95' THEN
    INTO emp_history VALUES(EMPID,HIREDATE,SAL)
  WHEN COMM IS NOT NULL THEN
    INTO emp_sales VALUES(EMPID,COMM,SAL)
  SELECT employee_id EMPID, hire_date HIREDATE,
         salary SAL, commission_pct COMM
  FROM employees
```

48 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 조건부 INSERT ALL

슬라이드의 예제는 이전 슬라이드의 예제와 유사하게 EMP\_HISTORY 테이블과 EMP\_SALES 테이블에 모두 행을 삽입합니다. SELECT 문은 EMPLOYEES 테이블에서 모든 사원에 대해 사원 ID, 채용 날짜, 급여 및 커미션 비율과 같은 세부 정보를 검색합니다. 사원 ID, 채용 날짜, 급여 등의 세부 정보는 EMP\_HISTORY 테이블에 삽입됩니다. 사원 ID, 수수료율, 급여 등의 세부 정보는 EMP\_SALES 테이블에 삽입됩니다.

이 INSERT 문은 SELECT 문으로 검색된 행에 제한 사항이 추가로 적용되므로 조건부 INSERT ALL이라고 합니다. SELECT 문에 의해 검색된 행에서 채용 날짜가 1995년 이전인 행만 EMP\_HISTORY 테이블에 삽입됩니다. 마찬가지로, 수수료율의 값이 널이 아닌 행만 EMP\_SALES 테이블에 삽입됩니다.

```
SELECT count(*) FROM emp_history;
```

	COUNT(*)
1	13

```
SELECT count(*) FROM emp_sales;
```

	COUNT(*)
1	35

## 조건부 INSERT ALL(계속)

INSERT ALL 문과 함께 ELSE 절을 사용할 수도 있습니다.

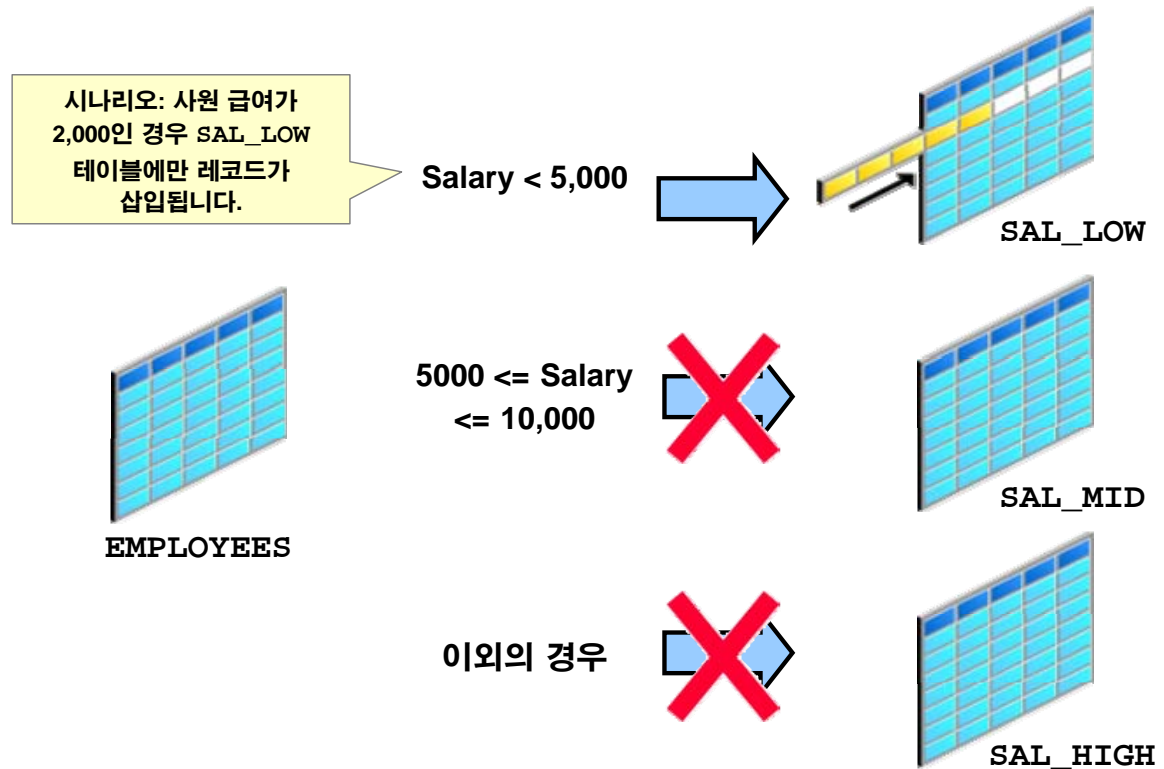
예제:

```
INSERT ALL
WHEN job_id IN
(select job_id FROM jobs WHERE job_title LIKE '%Manager%') THEN
  INTO managers2(last_name, job_id, SALARY)
VALUES (last_name, job_id, SALARY)
WHEN SALARY > 10000 THEN
  INTO richpeople(last_name, job_id, SALARY)
VALUES (last_name, job_id, SALARY)
ELSE
  INTO poorpeople VALUES (last_name, job_id, SALARY)
SELECT * FROM employees;
```

결과:

116개의 행이 삽입되었습니다.

## 조건부 INSERT FIRST: 예제



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 조건부 INSERT FIRST: 예제

EMPLOYEES 테이블의 모든 사원에 대해 조건을 충족하는 첫번째 대상 테이블에 사원 정보를 삽입합니다. 예제에서 사원의 급여가 2,000인 경우 SAL\_LOW 테이블에만 레코드가 삽입됩니다. SQL 문은 다음 페이지에 표시됩니다.

## 조건부 INSERT FIRST

```
INSERT FIRST
WHEN salary < 5000 THEN
    INTO sal_low VALUES (employee_id, last_name, salary)
WHEN salary between 5000 and 10000 THEN
    INTO sal_mid VALUES (employee_id, last_name, salary)
ELSE
    INTO sal_high VALUES (employee_id, last_name, salary)
SELECT employee_id, last_name, salary
FROM employees
```

107 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 조건부 INSERT FIRST

SELECT 문은 EMPLOYEES 테이블에서 모든 사원의 사원 ID, 성, 급여와 같은 세부 정보를 검색합니다. 각 사원 레코드는 조건을 충족하는 첫번째 대상 테이블에 삽입됩니다.

이 INSERT 문을 조건부 INSERT FIRST라고 합니다. WHEN salary < 5000 조건이 가장 먼저 평가됩니다. 이 첫번째 WHEN 절이 true로 평가되면 Oracle 서버가 해당 INTO 절을 실행하고 SAL\_LOW 테이블에 레코드를 삽입합니다. 이 행에 대해 후속 WHEN 절은 건너뛸니다.

행이 첫번째 WHEN 조건 (WHEN salary < 5000) 을 충족하지 않는 경우 다음 조건(WHEN salary between 5000 and 10000)이 평가됩니다. 이 조건이 true로 평가되면 SAL\_MID 테이블에 레코드가 삽입되고 마지막 조건을 건너뛸니다.

첫번째 조건 (WHEN salary < 5000)과 두번째 조건 (WHEN salary between 5000 and 10000)이 모두 true로 평가되지 않으면 Oracle 서버가 ELSE 절에 대해 해당 INTO 절을 실행합니다.

## 조건부 INSERT FIRST(계속)

총 20개의 행이 삽입되었습니다.

```
SELECT count(*) low FROM sal_low;
```

A Z LOW	
1	49

```
SELECT count(*) mid FROM sal_mid;
```

A Z MID	
1	43

```
SELECT count(*) high FROM sal_high;
```

A Z HIGH	
1	15

## 피벗팅 INSERT

비관계형 데이터베이스 테이블에서 판매 레코드 집합을 관계형 형식으로 변환합니다.

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000



Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 피벗팅 INSERT

피벗팅은 비관계형 데이터베이스 테이블과 같은 입력 스트림의 각 레코드를 관계형 데이터베이스 테이블 환경의 다중 레코드로 변환하는 등의 변형 작업을 수행하는 것을 말합니다.

다음 비관계형 데이터베이스 테이블에서 판매 레코드 집합을 검색한다고 가정합니다.

SALES\_SOURCE\_DATA에서 다음 형식의 판매 레코드 집합 검색:

EMPLOYEE\_ID, WEEK\_ID, SALES\_MON, SALES\_TUE, SALES\_WED,  
SALES\_THUR, SALES\_FRI

이러한 레코드를 다음과 같은 보다 일반적인 관계형 형식으로 SALES\_INFO 테이블에 저장하려고 합니다.

EMPLOYEE\_ID, WEEK, SALES

이 문제를 해결하려면 원래의 비관계형 데이터베이스 테이블 SALES\_SOURCE\_DATA의 각 레코드가 데이터 웨어하우스의 SALES\_INFO 테이블의 다섯 개의 레코드로 변환되도록 변형해야 합니다. 이러한 작업을 일반적으로 *피벗팅*이라고 합니다.

이 문제의 해결 방법은 다음 페이지에 나옵니다.

## 피벗팅 INSERT

```
INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sales_MON)
  INTO sales_info VALUES (employee_id, week_id, sales_TUE)
  INTO sales_info VALUES (employee_id, week_id, sales_WED)
  INTO sales_info VALUES (employee_id, week_id, sales_THUR)
  INTO sales_info VALUES (employee_id, week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR, sales_FRI
FROM sales_source_data;
```

5 rows inserted

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 피벗팅 INSERT(계속)

슬라이드의 예제에서 판매 데이터는 비관계형 데이터베이스 테이블 SALES\_SOURCE\_DATA에서 가져옵니다. 이 테이블은 판매 사원이 특정 주 ID의 주에 각 요일마다 판매한 판매 세부 정보입니다.

DESC SALES\_SOURCE\_DATA

Name	Null	Type
-----	-----	-----
EMPLOYEE_ID		NUMBER(6)
WEEK_ID		NUMBER(2)
SALES_MON		NUMBER(8,2)
SALES_TUE		NUMBER(8,2)
SALES_WED		NUMBER(8,2)
SALES_THUR		NUMBER(8,2)
SALES_FRI		NUMBER(8,2)



## 피버팅 INSERT(계속)

```
SELECT * FROM SALES_SOURCE_DATA;
```

	EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THUR	SALES_FRI
1	178	6	1750	2200	1500	1500	3000

```
DESC SALES_INFO
```

Name	Null	Type
-----	-----	-----
EMPLOYEE_ID		NUMBER(6)
WEEK		NUMBER(2)
SALES		NUMBER(8,2)

```
SELECT * FROM sales_info;
```

	EMPLOYEE_ID	WEEK	SALES
1	178	6	1750
2	178	6	2200
3	178	6	1500
4	178	6	1500
5	178	6	3000

피벗팅 INSERT를 사용하는 앞의 예제에서 SALES\_SOURCE\_DATA 테이블의 한 행은 관계형 테이블 SALES\_INFO의 레코드 다섯 개로 변환됩니다.

## 단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## MERGE 문

- 데이터베이스 테이블을 조건부로 갱신하거나 데이터를 삽입 또는 삭제하는 기능을 제공합니다.
- 행이 존재하는 경우 UPDATE를 수행하고, 새 행인 경우 INSERT를 수행합니다.
  - 별도의 갱신을 방지합니다.
  - 성능 및 사용 편의성이 향상됩니다.
  - 데이터 웨어하우징 응용 프로그램에서 유용합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### MERGE 문

Oracle 서버는 INSERT, UPDATE 및 DELETE 작업을 위한 MERGE 문을 지원합니다.

이 명령문을 사용하여 행을 조건부로 갱신하거나 테이블에 삽입하거나 테이블에서 삭제할 수 있으므로 다중 DML 문을 피할 수 있습니다. 대상 테이블에 대해 갱신, 삽입 또는 삭제 작업을 수행할지 여부는 ON 절의 조건에 따라 결정됩니다.

대상 테이블에 대한 INSERT 및 UPDATE 객체 권한과 소스 테이블에 대한 SELECT 객체 권한을 가져야 합니다. merge\_update\_clause의 DELETE 절을 지정하려면 대상 테이블에 대한 DELETE 객체 권한도 가져야 합니다.

MERGE 문은 결정적(deterministic) 명령문입니다. 동일한 MERGE 문에서 대상 테이블의 동일한 행을 여러 번 갱신할 수 없습니다.

다른 방법은 PL/SQL 루프와 다중 DML 문을 사용하는 것입니다. 그러나 MERGE 문은 사용하기 쉽고 간단하게 단일 SQL 문으로 표현됩니다.

MERGE 문은 다양한 데이터 웨어하우징 응용 프로그램에 적합합니다. 예를 들어, 데이터 웨어하우징 응용 프로그램에서 여러 소스의 데이터(이 데이터 중 일부는 중복될 수 있음)로 작업해야 하는 경우가 있습니다. MERGE 문을 사용하면 조건부로 행을 추가하거나 수정할 수 있습니다.

## MERGE 문 구문

MERGE 문을 사용하여 테이블에 행을 조건부로 삽입, 갱신 또는 삭제할 수 있습니다.

```
MERGE INTO table_name table_alias
  USING (table/view/sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col1_val,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 행 병합

MERGE 문을 사용하여 조건부로 기존 행을 갱신하고 새 행을 삽입할 수 있습니다. MERGE 문을 사용하여 테이블에서 행을 갱신하면서 동시에 불필요한 행을 삭제할 수 있습니다. 이렇게 하려면 MERGE 문의 구문에 자체 WHERE 절이 있는 DELETE 절을 포함시킵니다.

이 구문에서 다음이 적용됩니다.

INTO 절	갱신하거나 삽입할 대상 테이블을 지정합니다.
USING 절	갱신하거나 삽입할 데이터의 소스를 식별합니다. 테이블, 뷰 또는 subquery가 될 수 있습니다.
ON 절	MERGE 작업의 갱신 또는 삽입을 결정하는 조건입니다.
WHEN MATCHED   WHEN NOT MATCHED	조인 조건의 결과에 응답하는 방법을 서버에 지시합니다.

**참고:** 자세한 내용은 *Oracle Database 11g SQL Reference*를 참조하십시오.

## 행 병합: 예제

EMPLOYEES 테이블과 일치하도록 COPY\_EMP3 테이블에 행을 삽입하거나 갱신합니다.

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
...
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 행 병합: 예제

```
MERGE INTO copy_emp3 c
USING (SELECT * FROM EMPLOYEES ) e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
c.phone_number = e.phone_number,
c.hire_date = e.hire_date,
c.job_id = e.job_id,
c.salary = e.salary*2,
c.commission_pct = e.commission_pct,
c.manager_id = e.manager_id,
c.department_id = e.department_id
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```




## 행 병합: 예제(계속)

COPY\_EMP3 테이블은 다음 코드를 사용하여 생성됩니다.

```
CREATE TABLE COPY_EMP3 AS SELECT * FROM EMPLOYEES
WHERE SALARY<10000;
```

그런 다음 COPY\_EMP3 테이블을 query합니다.

```
SELECT employee_id, salary, commission_pct FROM COPY_EMP3;
```

	 EMPLOYEE_ID	 SALARY	 COMMISSION_PCT
1	198	5200	(null)
2	199	5200	(null)
3	200	8800	(null)
4	202	12000	(null)
5	203	13000	(null)

...

64	197	6000	(null)
65	162	10500	0.25
66	146	13500	0.3
67	150	10000	0.3

...

SALARY < 10000인 여러 명의 사원과 COMMISSION\_PCT를 가진 두 명의 사원이 있습니다.

슬라이드의 예제는 EMPLOYEES 테이블의 EMPLOYEE\_ID를 COPY\_EMP3 테이블의 EMPLOYEE\_ID와 일치시킵니다. 일치하는 행이 발견되면 COPY\_EMP3 테이블의 행이 EMPLOYEES 테이블의 행과 일치하도록 갱신되고 사원의 급여가 두 배가 됩니다.

COMMISSION\_PCT 열에 값이 있는 두 명의 사원 레코드가 삭제됩니다. 일치하는 행이 없으면 COPY\_EMP3 테이블에 행이 삽입됩니다.

## 행 병합: 예제

```
TRUNCATE TABLE copy_emp3;  
SELECT * FROM copy_emp3;  
0 rows selected
```

```
MERGE INTO copy_emp3 c  
USING (SELECT * FROM EMPLOYEES ) e  
ON (c.employee_id = e.employee_id)  
WHEN MATCHED THEN  
UPDATE SET  
c.first_name = e.first_name,  
c.last_name = e.last_name,  
...  
DELETE WHERE (E.COMMISSION_PCT IS NOT NULL)  
WHEN NOT MATCHED THEN  
INSERT VALUES(e.employee_id, e.first_name, ...
```

```
SELECT * FROM copy_emp3;  
107 rows selected.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 행 병합: 예제(계속)

슬라이드의 예제는 COPY\_EMP3 테이블이 비어있음을 보여줍니다. c.employee\_id = e.employee\_id 조건이 평가됩니다. 이 조건은 false를 반환합니다. 즉, 일치하는 항목이 없습니다. 논리는 WHEN NOT MATCHED 절로 분류되고 MERGE 명령은 EMPLOYEES 테이블의 행을 COPY\_EMP3 테이블에 삽입합니다. 이제 COPY\_EMP3 테이블은 EMPLOYEES 테이블과 정확히 동일한 데이터를 갖습니다.

```
SELECT employee_id, salary, commission_pct from copy_emp3;
```

	EMPLOYEE_ID	SALARY	COMMISSION_PCT
1	144	2500	(null)
2	143	2600	(null)
3	202	6000	(null)
4	141	3500	(null)
5	174	11000	0.3
...			
15	149	10500	0.2
16	206	8300	(null)
17	176	8600	0.2
18	124	5800	(null)
19	205	12000	(null)
20	178	7000	0.15

## 단원 내용

- Subquery를 사용하여 데이터 조작
- INSERT 및 UPDATE 문에서 명시적 기본값 지정
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- 일정 기간 동안의 데이터 변경 사항 추적

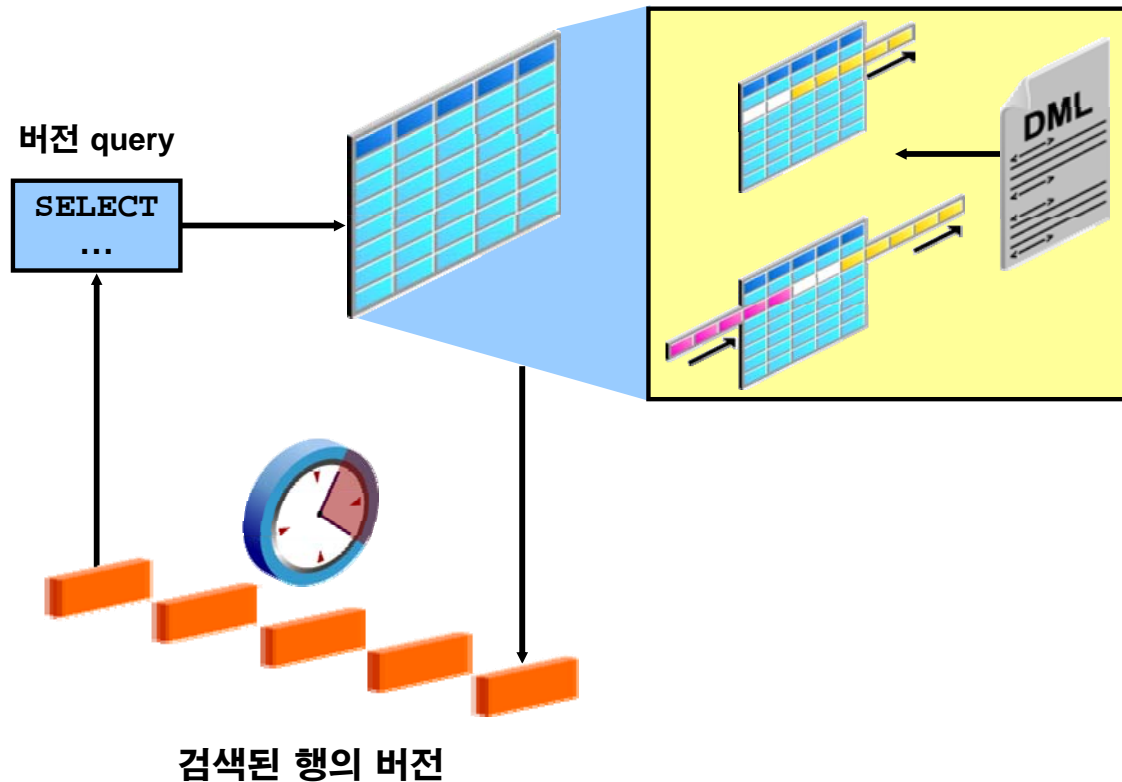
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Oracle Internal & Oracle Academy Use Only



## 데이터 변경 사항 추적



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 데이터 변경 사항 추적

테이블의 데이터가 부적절하게 변경된 사실을 발견하게 되는 경우가 있습니다. 이를 조사하기 위해 다중 flashback query를 사용하여 특정 시점의 행 데이터를 볼 수 있습니다. 보다 효과적인 방법으로는 flashback version query 기능을 사용하여 특정 기간의 행에 대해 모든 변경 사항을 볼 수 있습니다. 이 기능을 사용하면 SELECT 문에 SCN(시스템 변경 번호) 또는 행 값 변경 사항을 보려는 시간 기록 범위를 지정하는 VERSIONS 절을 추가할 수 있습니다. 또한 query에서 변경을 담당하는 트랜잭션과 같은 관련 메타 데이터를 반환할 수 있습니다.

뿐만 아니라 잘못된 트랜잭션을 식별한 후에 flashback transaction query 기능을 사용하여 해당 트랜잭션에서 수행한 다른 변경 사항을 식별할 수 있습니다. 그런 다음 Flashback Table 기능을 사용하여 변경되기 전의 상태로 테이블을 복원할 수 있습니다.

VERSIONS 절이 포함된 테이블 query를 사용하여 query가 실행된 시간과 현재 시간 전의 undo\_retention 초 사이에 존재하거나 존재했던 모든 행의 모든 버전을 생성할 수 있습니다. undo\_retention은 자동 튜닝 파라미터인 초기화 파라미터입니다. VERSIONS 절을 포함한 query를 version query라고 합니다. Version query의 결과는 WHERE 절이 행의 버전에 적용된 것처럼 동작합니다. Version query는 트랜잭션에서만 행 버전을 반환합니다.

**시스템 변경 번호(SCN):** Oracle 서버는 SCN을 할당하여 각각의 커밋된 트랜잭션에 대한 리두 레코드를 식별합니다.

## Flashback Version Query의 예

```
SELECT salary FROM employees3
WHERE employee_id = 107;
```

1

```
UPDATE employees3 SET salary = salary * 1.30
WHERE employee_id = 107;
```

2

```
COMMIT;
```

```
SELECT salary FROM employees3
  VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE employee_id = 107;
```

3

1

	SALARY
1	4200

3

	SALARY
1	5460
2	4200

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Flashback Version Query의 예

슬라이드의 예제에서 사원 107의 급여가 검색됩니다(1). 사원 107의 급여가 30% 인상되고 이 변경 사항이 커밋됩니다(2). 서로 다른 버전의 급여가 표시됩니다(3).

VERSIONS 절은 query 계획을 변경하지 않습니다. 예를 들어, 인덱스 액세스 방식을 사용하는 테이블 query를 실행하는 경우 VERSIONS 절을 사용하여 동일한 테이블에 동일한 query를 수행하면 계속 인덱스 액세스 방식을 사용합니다. Version query에서 반환되는 행 버전은 전체 트랜잭션에서의 행의 버전입니다. VERSIONS 절은 query의 트랜잭션 동작에 영향을 주지 않습니다. 이것은 VERSIONS 절이 포함된 테이블 query가 진행 중인 트랜잭션의 query 환경을 계속 상속한다는 것을 의미합니다.

기본 VERSIONS 절은 VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE로 지정될 수 있습니다.

VERSIONS 절은 query에만 사용할 수 있는 SQL 확장입니다. Subquery 내에서 VERSIONS 절을 사용하는 DML 및 DDL 작업을 가질 수 있습니다. 행 version query는 선택된 행의 모든 커밋된 버전을 검색합니다. 현재 활성 트랜잭션에 의해 수행된 변경 사항은 반환되지 않습니다. Version query는 행의 모든 incarnation을 검색합니다. 이것은 본질적으로 반환되는 버전에 삭제된 이후 재삽입된 행 버전이 포함됨을 의미합니다.

## Flashback Version Query의 예(계속)

Version query를 위한 행 액세스는 다음 두 가지 범주 중 하나로 정의될 수 있습니다.

- **ROWID 기반 행 액세스:** ROWID 기반 액세스의 경우 행 내용에 관계없이 지정된 ROWID의 모든 버전이 반환됩니다. 이는 본질적으로 ROWID로 나타난 블록에 있는 슬롯의 모든 버전이 반환됨을 의미합니다.
- **다른 모든 행 액세스:** 다른 모든 행 액세스의 경우 행의 모든 버전이 반환됩니다.

## VERSIONS BETWEEN 절

```
SELECT versions_starttime "START_DATE",
       versions_endtime   "END_DATE",
       salary
FROM   employees
       VERSIONS BETWEEN SCN MINVALUE
       AND MAXVALUE
WHERE  last_name = 'Lorentz';
```

	START_DATE	END_DATE	SALARY
1	18-JUN-09 05.07.10.000000000 PM	(null)	5460
2	(null)	18-JUN-09 05.07.10.000000000 PM	4200

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### VERSIONS BETWEEN 절

VERSIONS BETWEEN 절을 사용하여 query가 실행된 시간과 과거의 특정 시점 사이에 존재하거나 존재했던 행의 모든 버전을 검색할 수 있습니다.

Undo retention 시간이 BETWEEN 절의 하한값 시간 또는 SCN보다 작을 경우 query는 undo retention 시간까지만 버전을 검색합니다. BETWEEN 절의 시간 간격은 SCN 간격 또는 실제 시간 간격으로 지정될 수 있습니다. 이 시간 간격은 하한값 및 상한값에서 모두 닫힙니다.

예제에서 Lorentz의 급여 변경 사항이 검색됩니다. 첫번째 버전의 END\_DATE에 대한 NULL 값은 이 버전이 query 시점에 존재했던 버전임을 나타냅니다. 마지막 버전의 START\_DATE에 대한 NULL 값은 이 버전이 undo retention 시간 전에 생성되었음을 나타냅니다.

## 퀴즈

DEFAULT 키워드가 도입됨에 따라 INSERT 또는 UPDATE 명령을 사용할 때 프로그램에서 기본값을 하드 코딩하거나 기본값을 찾기 위해 디렉터리를 query할 필요가 없습니다.

1. 맞습니다.
2. 틀립니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- DML 문 사용 및 트랜잭션 제어
- 다중 테이블 INSERT의 기능 설명
- 다음과 같은 다중 테이블 INSERT 유형 사용
  - 무조건 INSERT
  - 피벗팅 INSERT
  - 조건부 INSERT ALL
  - 조건부 INSERT FIRST
- 테이블의 행 병합
- Subquery를 사용하여 데이터 조작
- 일정 기간 동안의 데이터 변경 사항 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

이 단원에서는 subquery를 사용하여 오라클 데이터베이스의 데이터를 조작하는 방법에 대해 설명했습니다. 또한 다중 테이블 INSERT 문, MERGE 문 및 데이터베이스의 변경 사항 추적에 대해 설명했습니다.

## 연습 4: 개요

이 연습에서는 다음 내용을 다룹니다.

- 다중 테이블 INSERT 수행
- MERGE 작업 수행
- 행 버전 추적

ORACLE

Copyright © 2009, Oracle. All rights reserved.





# 5

## 다른 시간대에서 데이터 관리

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 소수 표시 초를 저장하고 시간대를 추적하는 DATE와 유사한 데이터 유형 사용
- 두 datetime 값의 차이를 저장하는 데이터 유형 사용
- 다음 datetime 함수 사용:
  - CURRENT\_DATE
  - CURRENT\_TIMESTAMP
  - LOCALTIMESTAMP
  - DBTIMEZONE
  - SESSIONTIMEZONE
  - EXTRACT
  - TZ\_OFFSET
  - FROM\_TZ
  - TO\_TIMESTAMP
  - TO\_YMINTERVAL
  - TO\_DSINTERVAL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 소수 표시 초를 저장하고 시간대를 추적하는 DATE와 유사한 데이터 유형을 사용하는 방법을 배웁니다. 이 단원에서는 오라클 데이터베이스에서 사용할 수 있는 일부 datetime 함수에 대해 다룹니다.

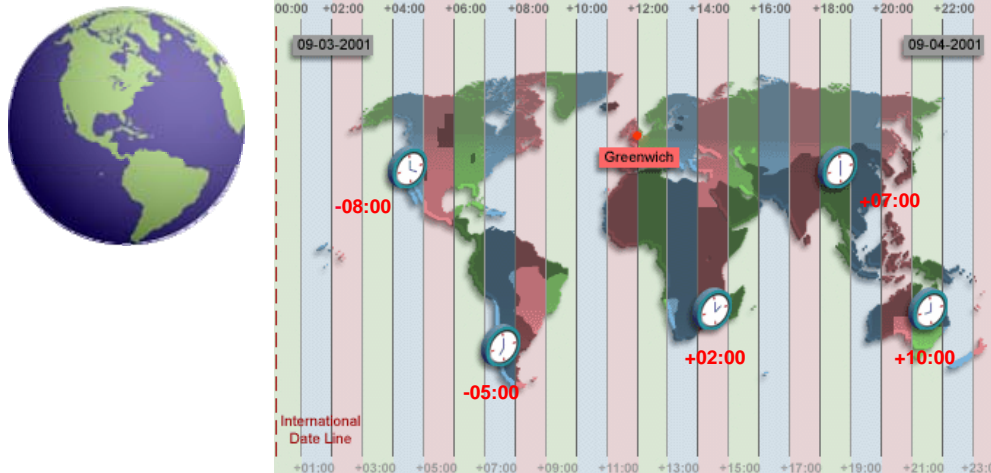
## 단원 내용

- **CURRENT\_DATE, CURRENT\_TIMESTAMP  
및 LOCALTIMESTAMP**
- **INTERVAL 데이터 유형**
- **다음 함수 사용:**
  - EXTRACT
  - TZ\_OFFSET
  - FROM\_TZ
  - TO\_TIMESTAMP
  - TO\_YMINTERVAL
  - TO\_DSINTERVAL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

# 시간대



**이 그림은 그리니치 표준시가 12:00일  
때 각 시간대의 시간을 나타냅니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 시간대

하루 중 시는 지구의 자전으로 측정됩니다. 특정한 순간의 하루 중 시간은 장소에 따라 달라집니다. 영국 그리니치시로 정오이면 날짜 변경선의 시간은 자정입니다. 지구는 24개의 시간대로 나뉘어져 있으며 각 시간대는 한 시간을 나타냅니다. 영국 그리니치의 본초 자오선 시간을 GMT (그리니치 표준시) 라고 합니다. GMT는 UTC(협정 세계 표준시)로 알려져 있습니다. UTC는 전세계의 다른 모든 시간대에서 참조의 기준으로 삼는 시간 표준입니다. GMT는 연중 동일하며 썬더 타임 또는 일광 절약 시간의 영향을 받지 않습니다. 자오선은 북극에서 남극까지 그어진 가상의 선입니다. 이 선을 경도 0도라고 하며 다른 모든 경선은 이 선에서부터 측정됩니다. 모든 시간은 UTC를 기준으로 측정되며 모든 위치는 위도 (적도에서 북쪽 또는 남쪽으로의 거리) 와 경도 (그리니치 자오선에서 동쪽 또는 서쪽으로의 거리) 를 가집니다.

## TIME\_ZONE 세션 파라미터

다음과 같이 TIME\_ZONE을 설정할 수 있습니다.

- 절대 오프셋
- 데이터베이스 시간대
- OS 로컬 시간대
- 명명된 지역

```
ALTER SESSION SET TIME_ZONE = '-05:00';  
ALTER SESSION SET TIME_ZONE = dbtimezone;  
ALTER SESSION SET TIME_ZONE = local;  
ALTER SESSION SET TIME_ZONE = 'America/New_York';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TIME\_ZONE 세션 파라미터

오라클 데이터베이스는 날짜 및 시간 데이터는 물론 소수 표시 초 단위로 시간대를 저장할 수 있도록 지원합니다. ALTER SESSION 명령을 사용하여 유저 세션의 시간대 값을 변경할 수 있습니다. 시간대 값은 절대 오프셋, 명명된 시간대, 데이터베이스 시간대 또는 로컬 시간대로 설정할 수 있습니다.

## CURRENT\_DATE, CURRENT\_TIMESTAMP 및 LOCALTIMESTAMP

- **CURRENT\_DATE:**
  - 유저 세션의 현재 날짜를 반환합니다.
  - DATE 데이터 유형입니다.
- **CURRENT\_TIMESTAMP:**
  - 유저 세션의 현재 날짜와 시간을 반환합니다.
  - TIMESTAMP WITH TIME ZONE 데이터 유형입니다.
- **LOCALTIMESTAMP:**
  - 유저 세션의 현재 날짜와 시간을 반환합니다.
  - TIMESTAMP 데이터 유형입니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CURRENT\_DATE, CURRENT\_TIMESTAMP 및 LOCALTIMESTAMP

CURRENT\_DATE 및 CURRENT\_TIMESTAMP 함수는 각각 현재 날짜 및 현재 시간 기록을 반환합니다. CURRENT\_DATE의 데이터 유형은 DATE입니다. CURRENT\_TIMESTAMP의 데이터 유형은 TIMESTAMP WITH TIME ZONE입니다. 반환되는 값은 함수를 실행하는 SQL 세션의 시간대 변위를 표시합니다. 시간대 변위는 현지 시간과 UTC 시간의 차이 (시 및 분으로 표시)입니다. TIMESTAMP WITH TIME ZONE 데이터 유형의 형식은 다음과 같습니다.

TIMESTAMP[ (fractional\_seconds\_precision) ] WITH TIME ZONE

여기서 fractional\_seconds\_precision은 선택적으로 SECOND datetime 필드의 소수 부분에 들어갈 자릿수를 지정하며 0~9 범위의 숫자가 될 수 있습니다. 기본값은 6입니다.

LOCALTIMESTAMP 함수는 세션 시간대의 현재 날짜 및 시간을 반환합니다.

LOCALTIMESTAMP와 CURRENT\_TIMESTAMP의 차이는 LOCALTIMESTAMP가 TIMESTAMP 값을 반환하는 반면 CURRENT\_TIMESTAMP는 TIMESTAMP WITH TIME ZONE 값을 반환한다는 것입니다.

이 두 함수는 NLS (국가별 언어 지원) 를 구분합니다. 즉, 결과는 현재 NLS 달력 및 datetime 형식이 됩니다.

**참고:** SYSDATE 함수는 현재 날짜와 시간을 DATE 데이터 유형으로 반환합니다.

Oracle Database 11g: SQL Fundamentals I 과정에서 SYSDATE 함수의 사용법을 배웠습니다.

## 세션의 시간대에서 날짜와 시간 비교

**TIME\_ZONE** 파라미터가 **-5:00**으로 설정된 다음 각 날짜 및 시간에 대해 **SELECT** 문이 실행되어 차이를 비교합니다.

```
ALTER SESSION
SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
ALTER SESSION SET TIME_ZONE = '-5:00';

SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL; ①

SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL; ②

SELECT SESSIONTIMEZONE, LOCALTIMESTAMP FROM DUAL; ③
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 세션의 시간대에서 날짜와 시간 비교

**ALTER SESSION** 명령은 세션의 날짜 형식을 'DD-MON-YYYY HH24:MI:SS'로 설정합니다. 이는 일(1-31)-월의 약어-4자리 연도 시(0-23):분(0-59):초(0-59)입니다.

슬라이드의 예제는 **TIME\_ZONE** 파라미터를 **-5:00**으로 설정하도록 세션이 변경됨을 나타냅니다. 그런 다음 형식의 차이점을 관찰하기 위해 **CURRENT\_DATE**, **CURRENT\_TIMESTAMP** 및 **LOCALTIMESTAMP**에 대해 **SELECT** 문이 실행됩니다.

**참고:** **TIME\_ZONE** 파라미터는 현재 **SQL** 세션에 대한 기본 로컬 시간대 변위를 지정합니다. **TIME\_ZONE**은 세션 파라미터일 뿐이며 초기화 파라미터가 아닙니다. **TIME\_ZONE** 파라미터는 다음과 같이 설정됩니다.

```
TIME_ZONE = '[+ | -] hh:mm'
```

형식 마스크 ([+ | -] hh:mm)는 UTC 이전 또는 이후의 시와 분을 나타냅니다.

## 세션의 시간대에서 날짜와 시간 비교

### Query 결과:

```
ALTER SESSION succeeded.
```

	SESSIONTIMEZONE	CURRENT_DATE
1	-05:00	23-JUN-2009 01:34:52

1

	SESSIONTIMEZONE	CURRENT_TIMESTAMP
1	-05:00	23-JUN-09 01.35.26.239882000 AM -05:00

2

	SESSIONTIMEZONE	LOCALTIMESTAMP
1	-05:00	23-JUN-09 01.36.21.811798000 AM

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 세션의 시간대에서 날짜와 시간 비교(계속)

이 경우에 `CURRENT_DATE` 함수는 세션의 시간대에서 현재 날짜를 반환하고 `CURRENT_TIMESTAMP` 함수는 세션의 시간대에서 현재 날짜 및 시간을 `TIMESTAMP WITH TIME ZONE` 데이터 유형의 값으로 반환하고 `LOCALTIMESTAMP` 함수는 세션의 시간대에서 현재 날짜와 시간을 반환합니다.



## DBTIMEZONE 및 SESSIONTIMEZONE

- 데이터베이스 시간대 값을 표시합니다.

```
SELECT DBTIMEZONE FROM DUAL;
```

DBTIMEZONE
1 +00:00

- 세션의 시간대 값을 표시합니다.

```
SELECT SESSIONTIMEZONE FROM DUAL;
```

SESSIONTIMEZONE
1 -05:00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DBTIMEZONE 및 SESSIONTIMEZONE

DBA는 CREATE DATABASE 문의 SET TIME\_ZONE 절을 지정하여 데이터베이스의 기본 시간대를 설정합니다. 이 절을 생략할 경우 기본 데이터베이스 시간대는 운영 체제의 시간대입니다. ALTER SESSION 문을 사용하여 세션에 대한 데이터베이스 시간대를 변경할 수 없습니다.

DBTIMEZONE 함수는 데이터베이스 시간대 값을 반환합니다. 반환 유형은 사용자가 가장 최근의 CREATE DATABASE 또는 ALTER DATABASE 문에서 데이터베이스 시간대 값을 지정한 방식에 따라 시간대 오프셋 ('[+|-]TZH:TZM' 형식의 문자 유형) 또는 시간대 지역 이름입니다. 슬라이드의 예제에서는 TIME\_ZONE 파라미터가 다음과 같은 형식으로 되어 있기 때문에 데이터베이스 시간대가 "-05:00"으로 설정됩니다.

```
TIME_ZONE = '[+ | -] hh:mm'
```

SESSIONTIMEZONE 함수는 현재 세션의 시간대 값을 반환합니다. 반환 유형은 사용자가 가장 최근의 ALTER SESSION 문에서 세션 시간대 값을 지정한 방식에 따라 시간대 오프셋 ('[+|-]TZH:TZM' 형식의 문자 유형) 또는 시간대 지역 이름입니다. 슬라이드의 예제는 세션 시간대가 UTC에 대해 -8시간 오프셋되었음을 보여줍니다. 데이터베이스 시간대는 현재 세션의 시간대와 다르다는 것을 확인합니다.

## TIMESTAMP 데이터 유형

데이터 유형	필드
TIMESTAMP	년, 월, 일, 시, 분, 초(소수 표시 초)
TIMESTAMP WITH TIME ZONE	TIMESTAMP 데이터 유형과 동일하며 또한 다음을 포함합니다. TIMEZONE_HOUR 및 TIMEZONE_MINUTE 또는 TIMEZONE_REGION
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMP 데이터 유형과 동일하며 또한 값에 시간대 오프셋을 포함합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TIMESTAMP 데이터 유형

TIMESTAMP 데이터 유형은 DATE 데이터 유형의 확장입니다.

**TIMESTAMP (fractional\_seconds\_precision)**

이 데이터 유형은 년, 월, 일 날짜 값과 시, 분, 초 시간 값을 포함하며 여기서 fractional\_seconds\_precision은 SECOND datetime 필드의 소수 부분 자릿수입니다.

fractional\_seconds\_precision에서 허용되는 값의 범위는 0~9이고 기본값은 6입니다.

**TIMESTAMP (fractional\_seconds\_precision) WITH TIME ZONE**

이 데이터 유형은 TIMESTAMP의 모든 값에 더해 시간대 변위 값도 포함합니다.

**TIMESTAMP (fractional\_seconds\_precision) WITH LOCAL TIME ZONE**

이 데이터 유형은 다음 예외 사항을 제외하고 TIMESTAMP의 모든 값을 포함합니다.

- 데이터는 데이터베이스에 저장될 때 데이터베이스 시간대로 정규화됩니다.
- 데이터가 검색될 때 유저는 세션 시간대의 데이터를 보게 됩니다.

## TIMESTAMP 필드

Datetime 필드	유효한 값
YEAR	-4712 ~ 9999(연도 0 제외)
MONTH	01 ~ 12
DAY	01 ~ 31
HOURL	00 ~ 23
MINUTE	00 ~ 59
SECOND	00 ~ 59.9(N). 여기서 9(N)은 자릿수
TIMEZONE_HOUR	-12 ~ 14
TIMEZONE_MINUTE	00 ~ 59

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TIMESTAMP 필드

각 datetime 데이터 유형은 이러한 여러 개의 필드로 구성됩니다. Datetime은 동일한 datetime 필드를 가진 경우에만 상호 비교 및 할당이 가능합니다.

# DATE와 TIMESTAMP의 차이

A

```
-- when hire_date is
of type DATE

SELECT hire_date
FROM employees;
```

	HIRE_DATE
1	21-JUN-99
2	13-JAN-00
3	17-SEP-87
4	17-FEB-96
5	17-AUG-97
6	07-JUN-94
7	07-JUN-94
8	07-JUN-94

B

```
ALTER TABLE employees
MODIFY hire_date TIMESTAMP;

SELECT hire_date
FROM employees;
```

	HIRE_DATE
1	21-JUN-99 12.00.00.000000000 AM
2	13-JAN-00 12.00.00.000000000 AM
3	17-SEP-87 12.00.00.000000000 AM
4	17-FEB-96 12.00.00.000000000 AM
5	17-AUG-97 12.00.00.000000000 AM
6	07-JUN-94 12.00.00.000000000 AM
7	07-JUN-94 12.00.00.000000000 AM
8	07-JUN-94 12.00.00.000000000 AM

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## TIMESTAMP 데이터 유형: 예제

슬라이드에서 예제 A는 hire\_date 열의 데이터 유형이 DATE인 경우 EMPLOYEES 테이블의 hire\_date 열에 있는 데이터를 보여줍니다. 예제 B에서 테이블이 변경되어 hire\_date 열의 데이터 유형이 TIMESTAMP로 되었습니다. 출력 결과가 다르게 표시되는 것을 확인할 수 있습니다. 열에 데이터가 있는 경우 DATE에서 TIMESTAMP로 변환할 수 있지만, 열이 비어 있지 않는 한 DATE 또는 TIMESTAMP에서 TIMESTAMP WITH TIME ZONE으로 변환할 수 없습니다.

TIMESTAMP에 대해 소수 표시 초의 자릿수를 지정할 수 있습니다. 이 예제와 같이 아무것도 지정되지 않으면 기본값 6으로 설정됩니다.

예를 들어 다음 명령문은 소수 표시 초의 자릿수를 7로 설정합니다.

```
ALTER TABLE employees
MODIFY hire_date TIMESTAMP(7);
```

**참고:** Oracle 날짜 데이터 유형은 기본적으로 이 예제에 표시된 것과 같이 나타납니다. 그러나 날짜 데이터 유형은 시간, 분, 초, AM, PM과 같은 추가 정보도 포함합니다. 이 형식으로 날짜를 구하려면 날짜 값에 형식 마스크나 함수를 적용하면 됩니다.

## TIMESTAMP 데이터 유형 비교

```
CREATE TABLE web_orders  
(order_date TIMESTAMP WITH TIME ZONE,  
 delivery_time TIMESTAMP WITH LOCAL TIME ZONE);
```

```
INSERT INTO web_orders values  
(current_date, current_timestamp + 2);
```

```
SELECT * FROM web_orders;
```

	ORDER_DATE	DELIVERY_TIME
1	23-JUN-09 01.56.39.000000000 AM -05:00	25-JUN-09 01.56.39.000000000 AM

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TIMESTAMP 데이터 유형 비교

슬라이드의 예제에서 TIMESTAMP WITH TIME ZONE 데이터 형식의 열과 TIMESTAMP WITH LOCAL TIME ZONE 데이터 형식의 열을 포함하는 새 테이블 web\_orders가 생성됩니다. 이 테이블은 web\_order가 발생할 때마다 채워집니다. 주문하는 유저의 시간 기록 및 시간대는 CURRENT\_DATE 값을 기준으로 삽입됩니다. 이 테이블은 주문이 발생할 때마다 CURRENT\_TIMESTAMP에 2일을 더한 값을 삽입하는 방식으로 채워집니다. 웹 기반 회사에서 배송을 보증하는 경우 이러한 방법으로 주문하는 사람의 시간대에 따라 배송 시간을 계산할 수 있습니다.

## 단원 내용

- CURRENT\_DATE, CURRENT\_TIMESTAMP  
및 LOCALTIMESTAMP
- INTERVAL 데이터 유형
- 다음 함수 사용:
  - EXTRACT
  - TZ\_OFFSET
  - FROM\_TZ
  - TO\_TIMESTAMP
  - TO\_YMINTERVAL
  - TO\_DSINTERVAL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## INTERVAL 데이터 유형

- INTERVAL 데이터 유형은 두 datetime 값의 차이를 저장하는 데 사용됩니다.
- 두 가지 간격 유형:
  - year-month
  - day-time
- 간격의 자릿수:
  - 간격을 구성하는 필드의 실제 부분 집합입니다.
  - 간격 수식자에 지정됩니다.

데이터 유형	필드
INTERVAL YEAR TO MONTH	년, 월
INTERVAL DAY TO SECOND	일, 시, 분, 초(소수 표시 초)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INTERVAL 데이터 유형

INTERVAL 데이터 유형은 두 datetime 값의 차이를 저장하는 데 사용됩니다. 간격에는 year-month 간격과 day-time 간격의 두 종류가 있습니다. year-month 간격이 YEAR 및 MONTH 필드의 연속된 부분 집합으로 구성되는 반면, day-time 간격은 DAY, HOUR, MINUTE, SECOND로 구성되는 필드의 연속된 부분 집합으로 구성됩니다. 간격을 구성하는 필드의 실제 부분 집합은 간격의 자릿수라고 하며 간격 수식자에 지정됩니다. 연도의 일수는 달력에 따라 결정되므로 year-month 간격은 NLS 종속적인 반면 day-time 간격은 NLS 독립적입니다.

간격 수식자는 선행 필드 또는 유일한 필드의 자릿수를 지정할 수도 있으며, 후행 필드가 SECOND인 경우 소수 표시 초의 자릿수(SECOND 값의 소수 부분 자릿수)를 지정할 수도 있습니다. 자릿수가 지정되지 않은 경우 선행 필드 자릿수의 기본값은 2이고 소수 표시 초 자릿수의 기본값은 6입니다.

## **INTERVAL 데이터 유형(계속)**

### **INTERVAL YEAR (year\_precision) TO MONTH**

이 데이터 유형은 년과 월로 기간을 저장하며, 여기서 year\_precision은 YEAR datetime 필드의 자릿수입니다. 허용되는 값의 범위는 0~9이고 기본값은 6입니다.

### **INTERVAL DAY (day\_precision) TO SECOND (fractional\_seconds\_precision)**

이 데이터 유형은 일, 시, 분, 초로 기간을 저장합니다. 여기서 day\_precision은 DAY datetime 필드의 최대 자릿수이고 (허용되는 값의 범위는 0~9, 기본값은 2), fractional\_seconds\_precision은 SECOND 필드의 소수 표시 부분의 자릿수입니다. 허용되는 값의 범위는 0~9이고 기본값은 6입니다.



## INTERVAL 필드

INTERVAL 필드	간격에 대해 유효한 값
YEAR	임의의 양의 정수 또는 음의 정수
MONTH	00 ~ 11
DAY	임의의 양의 정수 또는 음의 정수
HOURL	00 ~ 23
MINUTE	00 ~ 59
SECOND	00 ~ 59.9(N) 여기서 9(N)은 자릿수

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INTERVAL 필드

INTERVAL YEAR TO MONTH는 YEAR 및 MONTH 필드를 가질 수 있습니다.

INTERVAL DAY TO SECOND는 DAY, HOUR, MINUTE, SECOND 필드를 가질 수 있습니다.

간격 유형의 항목을 구성하는 필드의 실제 부분 집합은 간격 수식자에 의해 정의되며 이 부분 집합을 항목의 자릿수라고 합니다.

year-month 간격은 다른 year-month 간격과만 상호 비교 및 할당이 가능하고 day-time 간격은 다른 day-time 간격과만 상호 비교 및 할당이 가능합니다.

## INTERVAL YEAR TO MONTH: 예제

```
CREATE TABLE warranty
(prod_id number, warranty_time INTERVAL YEAR(3) TO
MONTH);

INSERT INTO warranty VALUES (123, INTERVAL '8' MONTH);
INSERT INTO warranty VALUES (155, INTERVAL '200'
YEAR(3));
INSERT INTO warranty VALUES (678, '200-11');
SELECT * FROM warranty;
```

	PROD_ID	WARRANTY_TIME
1	123	0-8
2	155	200-0
3	678	200-11

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INTERVAL YEAR TO MONTH 데이터 유형

INTERVAL YEAR TO MONTH는 YEAR 및 MONTH datetime 필드를 사용하여 기간을 저장합니다. INTERVAL YEAR TO MONTH를 다음과 같이 지정하십시오.

INTERVAL YEAR [(year\_precision)] TO MONTH

여기에서 year\_precision은 YEAR datetime 필드의 자릿수입니다. year\_precision의 기본값은 2입니다.

**제한 사항:** 선행 필드는 후행 필드보다 값이 커야 합니다. 예를 들어, INTERVAL '0-1' MONTH TO YEAR는 유효하지 않습니다.

#### 예제

- INTERVAL '123-2' YEAR(3) TO MONTH  
123년, 2개월의 간격을 나타냅니다.
- INTERVAL '123' YEAR(3)  
123년, 0개월의 간격을 나타냅니다.
- INTERVAL '300' MONTH(3)  
300개월의 간격을 나타냅니다.
- INTERVAL '123' YEAR  
기본 자릿수는 2인데 123은 세 자리이므로 오류를 반환합니다.

## INTERVAL YEAR TO MONTH 데이터 유형(계속)

오라클 데이터베이스는 INTERVAL YEAR TO MONTH 및 INTERVAL DAY TO SECOND의 두 가지 간격 데이터 유형과 열 유형, PL/SQL 인수, 변수를 지원하며 반환 유형은 두 가지 중 하나가 됩니다. 그러나 간격 리터럴의 경우 시스템이 INTERVAL '2' YEAR 또는 INTERVAL '10' HOUR와 같은 다른 ANSI 간격 유형을 인식합니다. 이 경우 각 간격은 지원되는 두 유형 중 하나로 변환됩니다.

위의 예제에서는 WARRANTY 테이블이 생성되는데, 이 테이블은 INTERVAL YEAR(3) TO MONTH 데이터 유형을 취하는 warranty\_time 열을 포함합니다. 다양한 제품에 대한 년과 월을 나타내기 위해 여러 다른 값이 테이블로 삽입됩니다. 이러한 행을 테이블에서 검색하면 년 값과 월 값이 (-)로 구분되어 표시됩니다.

## INTERVAL DAY TO SECOND

### 데이터 유형: 예제

```
CREATE TABLE lab
( exp_id number, test_time INTERVAL DAY(2) TO SECOND);

INSERT INTO lab VALUES (100012, '90 00:00:00');
INSERT INTO lab VALUES (56098,
    INTERVAL '6 03:30:16' DAY TO SECOND);
```

```
SELECT * FROM lab;
```

	EXP_ID	TEST_TIME
1	100012	90 0:0:0.0
2	56098	6 3:30:16.0

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### INTERVAL DAY TO SECOND 데이터 유형: 예제

슬라이드의 예제에서는 데이터 유형이 INTERVAL DAY TO SECOND인 test\_time 열을 가진 lab 테이블을 생성합니다. 그런 다음 90일, 0시간, 0분, 0초를 나타내는 '90 00:00:00' 값과 6일, 3시간, 30분, 16초를 나타내는 INTERVAL '6 03:30:16' DAY TO SECOND를 삽입합니다. SELECT 문은 이 데이터가 데이터베이스에서 어떻게 표시되는지 보여줍니다.

## 단원 내용

- CURRENT\_DATE, CURRENT\_TIMESTAMP  
및 LOCALTIMESTAMP
- INTERVAL 데이터 유형
- 다음 함수 사용:
  - EXTRACT
  - TZ\_OFFSET
  - FROM\_TZ
  - TO\_TIMESTAMP
  - TO\_YMINTERVAL
  - TO\_DSINTERVAL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## EXTRACT

- **SYSDATE에서 YEAR 구성 요소를 표시합니다.**

```
SELECT EXTRACT (YEAR FROM SYSDATE) FROM DUAL;
```

EXTRACT(YEARFROMSYSDATE)
2009

- **MANAGER\_ID가 100인 사원에 대해 HIRE\_DATE에서 MONTH 구성 요소를 표시합니다.**

```
SELECT last_name, hire_date,
       EXTRACT (MONTH FROM HIRE_DATE)
FROM employees
WHERE manager_id = 100;
```

	LAST_NAME	HIRE_DATE	EXTRACT(MONTHFROMHIRE_DATE)
1	Hartstein	17-FEB-1996 00:00:00	2
2	Kochhar	21-SEP-1989 00:00:00	9
3	De Haan	13-JAN-1993 00:00:00	1
4	Raphaely	07-DEC-1994 00:00:00	12
5	Weiss	18-JUL-1996 00:00:00	7

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### EXTRACT

EXTRACT 식은 datetime 또는 interval 값 표현식에서 지정된 datetime 필드의 값을 추출하여 반환합니다. 사용자는 EXTRACT 함수를 사용하여 다음 구문에서 언급된 임의의 구성 요소를 추출할 수 있습니다. EXTRACT 함수의 구문은 다음과 같습니다.

```
SELECT EXTRACT ([YEAR] [MONTH] [DAY] [HOUR] [MINUTE][SECOND]
               [TIMEZONE_HOUR] [TIMEZONE_MINUTE]
               [TIMEZONE_REGION] [TIMEZONE_ABBR]
FROM [datetime_value_expression] [interval_value_expression]);
```

TIMEZONE\_REGION 또는 TIMEZONE\_ABBR(약어) 을 추출할 때 반환된 값은 해당 시간대 이름이나 약어를 포함하는 문자열입니다. 다른 값을 추출할 때 반환되는 값은 그레고리력 형식의 날짜입니다. 시간대 값을 가진 datetime에서 추출할 경우 반환되는 값은 UTC 형식입니다.

슬라이드의 첫번째 예제에서 EXTRACT 함수는 SYSDATE에서 YEAR를 추출하는 데 사용됩니다. 슬라이드의 두번째 예제에서 EXTRACT 함수는 EMPLOYEE\_ID가 100인 관리자에게 보고하는 사원에 대해 EMPLOYEES 테이블의 HIRE\_DATE 열에서 MONTH를 추출하는 데 사용됩니다.

## TZ\_OFFSET

'US/Eastern' , 'Canada/Yukon' 및 'Europe/London' 시간대에 대한 시간대 오프셋을 표시합니다.

```
SELECT TZ_OFFSET('US/Eastern'),
       TZ_OFFSET('Canada/Yukon'),
       TZ_OFFSET('Europe/London')
FROM DUAL;
```

TZ_OFFSET('US/EASTERN')	TZ_OFFSET('CANADA/YUKON')	TZ_OFFSET('EUROPE/LONDON')
1 -04:00	-07:00	+01:00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TZ\_OFFSET

TZ\_OFFSET 함수는 입력된 값에 해당하는 시간대 오프셋을 반환합니다. 반환 값은 명령문이 실행되는 날짜에 따라 달라집니다. 예를 들어, TZ\_OFFSET 함수가 값 -08:00을 반환하는 경우 이 값은 명령이 실행된 시간대가 UTC로부터 8시간 이후임을 나타냅니다. 사용자는 적합한 시간대 이름, UTC로부터의 시간대 오프셋(자체를 반환함) 또는 키워드 SESSIONTIMEZONE 또는 DBTIMEZONE을 입력할 수 있습니다. TZ\_OFFSET 함수의 구문은 다음과 같습니다.

```
TZ_OFFSET ( [ 'time_zone_name' ] '[+ | -] hh:mm' ]
           [ SESSIONTIMEZONE ] [ DBTIMEZONE ]
```

Fold Motor Company는 US/Eastern 시간대에 속하는 USA Michigan에 본사가 있습니다. 이 회사의 대표 이사인 Fold 씨는 캐나다 지사의 부사장과 유럽 지사의 부사장과 함께 전화 회의를 하려고 합니다. 이 두 명의 부사장은 각각 Canada/Yukon 및 Europe/London 시간대에 있습니다. Fold 씨는 회사의 고위 경영진이 회의에 참석할 수 있는지 확인하기 위해 이 두 장소의 시간을 알아보려고 합니다. 그의 비서인 Scott은 예제에 표시된 query를 실행하여 다음과 같은 결과를 얻습니다.

- 'US/Eastern' 시간대는 UTC보다 4시간 늦습니다.
- 'Canada/Yukon' 시간대는 UTC보다 7시간 늦습니다.
- 'Europe/London' 시간대는 UTC보다 1시간 빠릅니다.

## TZ\_OFFSET(계속)

유효한 시간대 이름 값 리스트를 보려면 V\$TIMEZONE\_NAMES Dynamic Performance 뷰를 query하면 됩니다.

```
SELECT * FROM V$TIMEZONE_NAMES;
```

R 2	TZNAME	R 2	TZABBREV
1	Africa/Abidjan		LMT
2	Africa/Abidjan		GMT
3	Africa/Accra		LMT
4	Africa/Accra		GMT
5	Africa/Accra		CHST

...



## FROM\_TZ

'Australia/North' 시간대 지역에 대해 TIMESTAMP 값 '2000-03-28 08:00:00'을 TIMESTAMP WITH TIME ZONE 값으로 표시합니다.

```
SELECT FROM_TZ(TIMESTAMP
                '2000-07-12 08:00:00', 'Australia/North')
FROM DUAL;
```

FROM_TZ(TIMESTAMP'2000-07-1208:00:00','AUSTRALIA/NORTH')
1 12-JUL-00 08.00.00.000000000 AM AUSTRALIA/NORTH

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FROM\_TZ

FROM\_TZ 함수는 TIMESTAMP 값을 TIMESTAMP WITH TIME ZONE 값으로 변환합니다. FROM\_TZ 함수의 구문은 다음과 같습니다.

FROM\_TZ(TIMESTAMP timestamp\_value, time\_zone\_value)

여기서 time\_zone\_value는 'TZH:TZM' 형식의 문자열이거나 선택 사항인 TZD 형식의 TZR (시간대 지역) 로 문자열을 반환하는 문자 표현식입니다. TZD는 일광 절약 시간 정보를 포함하는 시간대 문자열의 약어입니다. TZR은 datetime 입력 문자열에서 시간대 지역을 나타냅니다. 예제에서는 'Australia/North'이고 미국/태평양 표준시의 경우 'PST', 미국/태평양 일광 절약 시간의 경우 'PDT' 등입니다.

슬라이드의 예제는 TIMESTAMP 값을 TIMESTAMP WITH TIME ZONE으로 변환합니다.

**참고:** TZR 및 TZD 형식 요소에 대해 유효한 값 리스트를 보려면 V\$TIMEZONE\_NAMES Dynamic Performance 뷰를 query합니다.

## TO\_TIMESTAMP

문자열 '2007-03-06 11:00:00'을 TIMESTAMP 값으로 표시합니다.

```
SELECT TO_TIMESTAMP ('2007-03-06 11:00:00',  
                      'YYYY-MM-DD HH:MI:SS')  
FROM DUAL;
```

```
TO_TIMESTAMP('2007-03-0611:00:00','YYYY-MM-DDHH:MI:SS')  
06-MAR-07 11.00.00.000000000
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TO\_TIMESTAMP

TO\_TIMESTAMP 함수는 CHAR, VARCHAR2, NCHAR, NVARCHAR2 데이터 유형의 문자열을 TIMESTAMP 데이터 유형의 값으로 변환합니다. TO\_TIMESTAMP 함수의 구문은 다음과 같습니다.

```
TO_TIMESTAMP (char,[fmt],[ 'nlsparam' ])
```

선택적 fmt는 char 형식을 지정합니다. fmt를 생략할 경우 문자열은 TIMESTAMP 데이터 유형의 기본 형식이어야 합니다. 선택 사항인 nlsparam은 월 및 일 이름과 약어가 반환되는 언어를 지정합니다. 이 인수는 다음과 같은 형식일 수 있습니다.

```
'NLS_DATE_LANGUAGE = language'
```

nlsparam을 생략할 경우 이 함수는 사용자 세션에 대해 기본 날짜 언어를 사용합니다.

슬라이드의 예제는 문자열을 TIMESTAMP의 값으로 변환합니다.

**참고:** TO\_TIMESTAMP\_TZ 함수는 CHAR, VARCHAR2, NCHAR, NVARCHAR2 데이터 유형의 문자열을 TIMESTAMP WITH TIME ZONE 데이터 유형의 값으로 변환합니다. 이 함수에 대한 자세한 내용은 *Oracle Database SQL Language Reference 11g Release 1 (11.1)*을 참조하십시오.

## TO\_YMINTERVAL

DEPARTMENT\_ID가 20인 부서에서 근무하는 사원에 대해 채용 날짜로부터 1년 2개월이 지난 날짜를 표시합니다.

```
SELECT hire_date,  
       hire_date + TO_YMINTERVAL('01-02') AS  
       HIRE_DATE_YMININTERVAL  
FROM   employees  
WHERE  department_id = 20;
```

	HIRE_DATE	HIRE_DATE_YMININTERVAL
1	17-FEB-1996 00:00:00	17-APR-1997 00:00:00
2	17-AUG-1997 00:00:00	17-OCT-1998 00:00:00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TO\_YMINTERVAL

TO\_YMINTERVAL 함수는 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2 데이터 유형의 문자열을 INTERVAL YEAR TO MONTH 데이터 유형으로 변환합니다. INTERVAL YEAR TO MONTH 데이터 유형은 YEAR 및 MONTH datetime 필드를 사용하여 기간을 저장합니다. INTERVAL YEAR TO MONTH의 형식은 다음과 같습니다.

INTERVAL YEAR [(year\_precision)] TO MONTH

여기에서 year\_precision은 YEAR datetime 필드의 자릿수입니다. year\_precision의 기본값은 2입니다. TO\_YMINTERVAL 함수의 구문은 다음과 같습니다.

TO\_YMINTERVAL (char)

여기에서 char은 변환될 문자열입니다.

슬라이드의 예제는 EMPLOYEES 테이블의 부서 20에서 근무하는 사원에 대해 채용 날짜로부터 1년 2개월이 지난 날짜를 계산합니다.

## TO\_DSINTERVAL

모든 사원에 대해 채용 날짜 이후 100일 10시간이 지난 날짜를 표시합니다.

```
SELECT last_name,  
       TO_CHAR(hire_date, 'mm-dd-yy:hh:mi:ss') hire_date,  
       TO_CHAR(hire_date +  
               TO_DSINTERVAL('100 10:00:00'),  
               'mm-dd-yy:hh:mi:ss') hiredate2  
FROM employees;
```

	LAST_NAME	HIRE_DATE	HIREDATE2
1	OConnell	06-21-99:12:00:00	09-29-99:10:00:00
2	Grant	01-13-00:12:00:00	04-22-00:10:00:00
3	Whalen	09-17-87:12:00:00	12-26-87:10:00:00
4	Hartstein	02-17-96:12:00:00	05-27-96:10:00:00
5	Fay	08-17-97:12:00:00	11-25-97:10:00:00
6	Mavris	06-07-94:12:00:00	09-15-94:10:00:00
7	Baer	06-07-94:12:00:00	09-15-94:10:00:00
8	Higgins	06-07-94:12:00:00	09-15-94:10:00:00

...

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### TO\_DSINTERVAL

TO\_DSINTERVAL은 CHAR, VARCHAR2, NCHAR 또는 NVARCHAR2 데이터 유형을 INTERVAL DAY TO SECOND 데이터 유형으로 변환합니다.

슬라이드의 예제에서는 채용 날짜로부터 100일 10시간이 지난 날짜를 구합니다.

## 일광 절약 시간

- **4월의 첫번째 일요일**
  - 시간이 01:59:59 AM에서 03:00:00 AM으로 건너뛰니다.
  - 02:00:00 AM에서 02:59:59 AM 사이의 값은 유효하지 않습니다.
- **10월의 마지막 일요일**
  - 시간이 02:00:00 AM에서 01:00:01 AM으로 건너뛰니다.
  - 01:00:01 AM에서 02:00:00 AM 사이의 값은 두 번 나타나기 때문에 모호해집니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 일광 절약 시간 (DST)

대부분의 서방 국가에서는 여름 몇 개월 동안 시계를 1시간 앞당깁니다. 이 기간을 일광 절약 시간이라고 합니다. 미국, 멕시코 및 캐나다의 경우 일광 절약 시간은 4월의 첫번째 일요일부터 10월의 마지막 일요일까지 지속됩니다. 유럽 연합 국가에서도 일광 절약 시간을 지키지만 썬머 타임 기간이라고 부릅니다. 유럽의 썬머 타임 기간은 북미 지역보다 한 주 먼저 시작하며 끝나는 시점은 동일합니다.

오라클 데이터베이스는 지정된 시간대 지역에서 일광 절약 시간이 시행되는지 여부를 자동으로 확인하고 그에 따라 로컬 시간을 반환합니다. Datetime 값은 경계 지역인 경우를 제외한 모든 경우에 오라클 데이터베이스가 지정된 지역에 일광 절약 시간이 적용되고 있는지 판단하기에 충분한 정보를 제공합니다. 경계 지역인 경우 일광 절약 시간이 적용되거나 해제되는 기간 동안 발생합니다. 예를 들어, 미국/동부 지역의 경우 일광 절약 시간이 적용되는 시점에서 시간이 01:59:59 AM에서 03:00:00 AM으로 변경됩니다. 02:00:00 AM과 02:59:59 AM사이의 1시간 간격은 존재하지 않습니다. 일광 절약 시간이 해제될 때는 시간이 02:00:00 AM에서 다시 01:00:01 AM로 변경되어 01:00:01 AM과 02:00:00 AM 사이의 1시간 간격이 반복됩니다.

## 일광 절약 시간 (DST)(계속)

### **ERROR\_ON\_OVERLAP\_TIME**

ERROR\_ON\_OVERLAP\_TIME은 겹치는 기간에 `datetime`이 발생했는데 기간을 구별하는 시간대 약어가 지정되지 않았을 경우 오류 발생을 통보하는 세션 파라미터입니다.

예를 들어, 일광 절약 시간이 10월 31일 02:00:01 AM에 끝나는 경우 겹치는 기간은 다음과 같습니다.

- 10/31/2004 01:00:01 AM to 10/31/2004 02:00:00 AM (EDT)
- 10/31/2004 01:00:01 AM to 10/31/2004 02:00:00 AM (EST)

이 두 기간 중 하나에서 발생하는 `datetime` 문자열을 입력하는 경우 시스템이 기간을 결정할 수 있도록 입력 문자열에 시간대 약어 (예: EDT 또는 EST) 를 지정해야 합니다. 이 시간대 약어가 없으면 시스템은 다음을 수행합니다.

ERROR\_ON\_OVERLAP\_TIME 파라미터가 FALSE인 경우 입력 시간이 표준 시간 (예: EST)이라고 가정합니다. 그렇지 않은 경우 오류가 발생합니다.

## 퀴즈

다음 중 TIME\_ZONE 세션 파라미터를 설정할 수 있는 시간대 값을 고르십시오.

1. 상대 오프셋
2. 데이터베이스 시간대
3. OS 로컬 시간대
4. 명명된 지역

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2, 3, 4

## 요약

이 단원에서는 다음 함수의 사용 방법에 대해 설명했습니다.

- CURRENT\_DATE
- CURRENT\_TIMESTAMP
- LOCALTIMESTAMP
- DBTIMEZONE
- SESSIONTIMEZONE
- EXTRACT
- TZ\_OFFSET
- FROM\_TZ
- TO\_TIMESTAMP
- TO\_YMINTERVAL
- TO\_DSINTERVAL

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 요약

이 단원에서는 오라클 데이터베이스에서 사용할 수 있는 일부 datetime 함수에 대해 설명했습니다.



## 연습 5: 개요

이 연습에서는 datetime 함수의 사용에 대해 다룹니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 5: 개요

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



# 6

## Subquery를 사용하여 데이터 검색

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

이 단원을 마치면 다음을 수행할 수 있습니다.

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- Correlated Subquery를 사용하여 행 갱신 및 삭제
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 SELECT 문의 여러 열 subquery 및 FROM 절의 subquery를 작성하는 방법에 대해 배웁니다. 또한 스칼라 subquery, correlated subquery 및 WITH 절을 사용하여 문제를 해결하는 방법을 설명합니다.

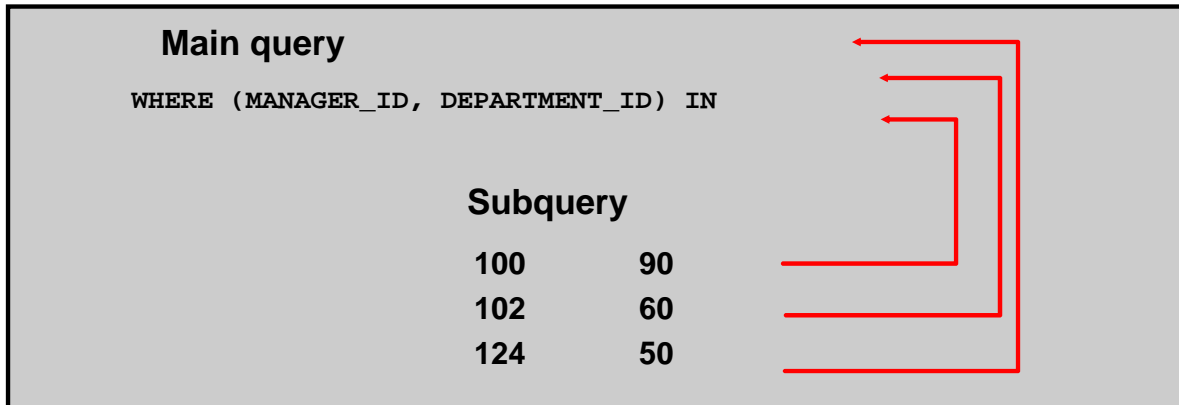
## 단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 여러 열 Subquery



**Main query의 각 행은 여러 행 및 여러 열 subquery의 값과 비교됩니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 여러 열 Subquery

지금까지는 inner SELECT 문에서 한 열만 반환되고 이 열을 사용하여 상위 SELECT 문의 표현식을 평가하는 단일 행 subquery 및 여러 행 subquery를 작성했습니다. 둘 이상의 열을 비교하려면 논리 연산자를 사용하여 WHERE 절을 작성해야 합니다. 여러 열 subquery를 사용하여 중복되는 WHERE 조건을 단일 WHERE 절로 조합할 수 있습니다.

#### 구문

```
SELECT      column, column, ...
FROM table
WHERE (column, column, ...) IN
      (SELECT column, column, ...
        FROM table
        WHERE condition);
```

슬라이드의 그림은 main query의 MANAGER\_ID 및 DEPARTMENT\_ID 값을 subquery에서 검색된 MANAGER\_ID 및 DEPARTMENT\_ID 값과 비교하여 보여줍니다. 비교되는 열 수가 두 개 이상이므로 이 예제는 여러 열 subquery로 적합합니다.

**참고:** 다음 몇 개의 슬라이드 예제를 실행하기 전에 lab\_06\_insert\_empdata.sql 파일을 사용하여 emp1\_demo 테이블을 생성하고 데이터를 채워야 합니다.

# 열 비교

Subquery를 포함한 여러 열 비교 유형:

- 비쌍 방식 비교
- 쌍 방식 비교

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 쌍 방식 비교와 비쌍 방식 비교

Subquery를 포함한 여러 열 비교는 비쌍방식 비교 또는 쌍방식 비교가 될 수 있습니다. "Daniel과 동일한 부서에서 근무하고 동일한 관리자의 감독을 받는 사원의 세부 정보를 표시합니다." 예제를 고려할 경우 다음 명령문을 사용하여 정확한 결과를 얻을 수 있습니다.

```
SELECT first_name, last_name, manager_id, department_id
FROM empl_demo
WHERE manager_id IN (SELECT manager_id
                     FROM empl_demo
                     WHERE first_name = 'Daniel')
AND department_id IN (SELECT department_id
                     FROM empl_demo
                     WHERE first_name = 'Daniel');
```

EMPL\_DEMO 테이블에는 한 명의 "Daniel"만 있습니다(사원 108의 감독을 받고 부서 100에서 근무하는 Daniel Faviat). 그러나 subquery가 두 개 이상의 행을 반환하는 경우 결과가 정확하지 않을 수도 있습니다. 예를 들어, 동일한 query를 실행하지만 "Daniel"을 "John"으로 대체하면 잘못된 결과를 얻을 수 있습니다. 그 이유는 department\_id와 manager\_id의 조합이 중요하기 때문입니다. 이 query에 대해 정확한 결과를 얻으려면 쌍방식 비교를 사용해야 합니다.

## 쌍 방식 비교 Subquery

이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원에 대한 세부 정보를 표시합니다.

```
SELECT employee_id, manager_id, department_id
FROM   empl_demo
WHERE  (manager_id, department_id) IN
      (SELECT manager_id, department_id
       FROM empl_demo
       WHERE first_name = 'John')
AND first_name <> 'John';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 쌍 방식 비교 Subquery

슬라이드의 예제는 EMPL\_DEMO 테이블에 있는 각 행의 MANAGER\_ID 열과 DEPARTMENT\_ID 열의 값 조합을 FIRST\_NAME이 "John"인 사원의 MANAGER\_ID 열 및 DEPARTMENT\_ID 열의 값과 비교합니다. 먼저 FIRST\_NAME이 "John"인 사원의 MANAGER\_ID 및 DEPARTMENT\_ID 값을 검색하는 subquery가 실행됩니다. 이 subquery는 다음을 반환합니다.

	MANAGER_ID	DEPARTMENT_ID
1	108	100
2	123	50
3	100	80



## 쌍방식 비교 Subquery(계속)

이러한 값들은 EMPL\_DEMO 테이블에 있는 각 행의 MANAGER\_ID 열 및 DEPARTMENT\_ID 열과 비교됩니다. 비교가 일치하면 행이 표시됩니다. 출력에서 FIRST\_NAME이 "John"인 사원의 레코드는 표시되지 않습니다. 다음은 슬라이드의 query를 출력한 결과입니다.

	EMPLOYEE_ID	MANAGER_ID	DEPARTMENT_ID
1	113	108	100
2	112	108	100
3	111	108	100
4	109	108	100
5	195	123	50
6	194	123	50
7	193	123	50
8	192	123	50
9	140	123	50
10	138	123	50
11	137	123	50
12	149	100	80
13	148	100	80
14	147	100	80
15	146	100	80

## 비쌍 방식 비교 Subquery

이름이 "John"인 사원과 동일한 관리자의 감독을 받고 동일한 부서에서 근무하는 사원의 세부 정보를 표시합니다.

```
SELECT  employee_id, manager_id, department_id
FROM    empl_demo
WHERE   manager_id IN
        (SELECT manager_id
         FROM  empl_demo
         WHERE first_name = 'John')
AND department_id IN
        (SELECT department_id
         FROM  empl_demo
         WHERE first_name = 'John')
AND first_name <> 'John';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 비쌍 방식 비교 Subquery

이 예제는 열의 비쌍방식 비교를 보여줍니다. 먼저 FIRST\_NAME이 "John"인 사원의 MANAGER\_ID 값을 검색하는 subquery가 실행됩니다. 마찬가지로, FIRST\_NAME이 "John"인 사원의 DEPARTMENT\_ID 값을 검색하는 두번째 subquery가 실행됩니다. MANAGER\_ID 및 DEPARTMENT\_ID 열의 검색된 값은 EMPL\_DEMO 테이블에 있는 각 행의 MANAGER\_ID 및 DEPARTMENT\_ID 열과 비교됩니다. EMPL\_DEMO 테이블에 있는 행의 MANAGER\_ID 열이 inner subquery에 의해 검색된 MANAGER\_ID 값과 일치하고, EMPL\_DEMO 테이블에 있는 행의 DEPARTMENT\_ID 열이 두번째 subquery에 의해 검색된 DEPARTMENT\_ID 값과 일치하면 레코드가 표시됩니다.

## 비쌍방식 비교 Subquery(계속)

다음은 이전 슬라이드의 query를 출력한 결과입니다.

R2	EMPLOYEE_ID	R2	MANAGER_ID	R2	DEPARTMENT_ID
1	109		108		100
2	111		108		100
3	112		108		100
4	113		108		100
5	120		100		50
6	121		100		50
7	122		100		50
8	123		100		50
9	124		100		50
10	137		123		50
11	138		123		50
12	140		123		50
13	192		123		50
14	193		123		50
15	194		123		50
16	195		123		50
17	146		100		80
18	147		100		80
19	148		100		80
20	149		100		80

이 query는 쌍방식 비교보다 많은 추가 행을 검색합니다. (manager\_id=100이고 department\_id=50 또는 80인 조합을 가진 이름이 "John"인 사원이 없는 경우에도 해당 조합을 가진 행을 검색합니다.)

## 단원 내용

- 여러 열 Subquery 작성
- **SQL에서 스칼라 Subquery 사용**
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 스칼라 Subquery 표현식

- 스칼라 subquery 표현식은 한 행에서 정확히 하나의 열 값을 반환하는 subquery입니다.
- 스칼라 subquery는 다음에서 사용할 수 있습니다.
  - DECODE 및 CASE의 조건 및 표현식 부분
  - GROUP BY를 제외한 SELECT의 모든 절
  - UPDATE 문의 SET 절 및 WHERE 절

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL의 스칼라 Subquery

한 행에서 정확히 하나의 열 값만 반환하는 subquery를 스칼라 subquery라고 합니다. 복합 WHERE 절과 논리 연산자를 사용하여 두 개 이상의 열을 비교하도록 작성된 여러 열 subquery는 스칼라 subquery가 아닙니다.

스칼라 subquery의 표현식 값은 subquery의 선택 리스트 항목의 값입니다. Subquery에서 반환되는 행이 없는 경우 스칼라 subquery 표현식의 값은 NULL입니다. Subquery가 두 개 이상의 행을 반환하면 Oracle 서버에서 오류를 반환합니다. Oracle 서버는 항상 SELECT 문에 스칼라 subquery를 사용할 수 있도록 지원합니다. 다음에서 스칼라 subquery를 사용할 수 있습니다.

- DECODE 및 CASE의 조건 및 표현식 부분
- GROUP BY를 제외한 SELECT의 모든 절
- UPDATE 문의 SET 절 및 WHERE 절

그러나 다음에서 스칼라 subquery는 유효한 표현식이 아닙니다.

- 열의 기본값과 클러스터의 해시 표현식
- DML(데이터 조작어) 문의 RETURNING 절
- 함수 기반 인덱스의 기준
- GROUP BY 절, CHECK 제약 조건 및 WHEN 조건
- CONNECT BY 절
- CREATE PROFILE과 같이 query와 관계없는 문

## 스칼라 Subquery: 예제

- CASE 표현식의 스칼라 Subquery:

```
SELECT employee_id, last_name,  
       (CASE  
         WHEN department_id = 20  
           (SELECT department_id  
            FROM departments  
            WHERE location_id = 1800)  
         THEN 'Canada' ELSE 'USA' END) location  
FROM employees;
```

- ORDER BY 절의 스칼라 Subquery:

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 스칼라 Subquery: 예제

슬라이드의 첫번째 예제는 스칼라 subquery를 CASE 표현식에 사용하는 방법을 보여줍니다. Inner query는 값 20을 반환합니다. 이 값은 위치 ID가 1800인 부서의 부서 ID입니다. Outer query의 CASE 표현식은 inner query의 결과를 사용하여 사원 ID, 성 및 Canada 또는 USA 값을 표시합니다. 이 위치 값은 outer query에 의해 검색된 레코드의 부서 ID가 20인지 아닌지 여부에 따라 결정됩니다.

다음은 슬라이드의 첫번째 예제의 결과입니다.

...

	EMPLOYEE_ID	LAST_NAME	LOCATION
1	198	OConnell	USA
2	199	Grant	USA
3	200	Whalen	USA
4	201	Hartstein	Canada
5	202	Fay	Canada
6	203	Mavris	USA

## 스칼라 Subquery: 예제(계속)

슬라이드의 두번째 예제는 스칼라 subquery를 ORDER BY 절에 사용하는 방법을 보여줍니다. 이 예제에서는 EMPLOYEES 테이블의 DEPARTMENT\_ID를 DEPARTMENTS 테이블의 DEPARTMENT\_ID와 일치시키고 DEPARTMENT\_NAME을 기준으로 출력을 정렬합니다. 이 비교는 ORDER BY 절의 스칼라 subquery에서 수행됩니다. 다음은 두번째 예제의 결과입니다.

	EMPLOYEE_ID	LAST_NAME
1	205	Higgins
2	206	Gietz
3	200	Whalen
4	100	King
5	101	Kochhar
6	102	De Haan
7	112	Urman
8	108	Greenberg
9	109	Faviet

...

두번째 예제에서는 correlated subquery가 사용됩니다. Correlated subquery에서 subquery는 상위 명령문에서 참조된 테이블의 열을 참조합니다. Correlated subquery는 이 단원의 뒷부분에서 설명합니다.

## 단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- **Correlated Subquery로 문제 해결**
- EXISTS 및 NOT EXISTS 연산자 사용
- WITH 절 사용

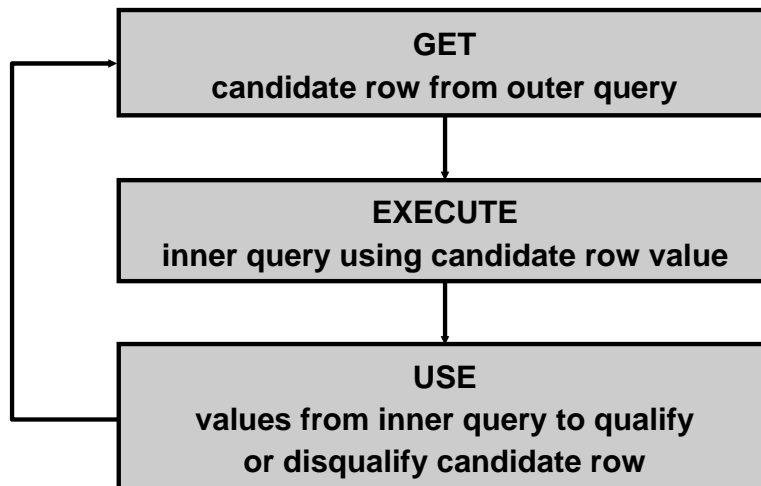
ORACLE

Copyright © 2009, Oracle. All rights reserved.



# Correlated Subquery

**Correlated subquery는 행 단위 처리에 사용됩니다. 각 subquery는 outer query의 모든 행에 대해 한번씩 실행됩니다.**



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated Subquery

Oracle 서버는 subquery가 상위 명령문에서 참조된 테이블의 열을 참조할 때 correlated subquery를 수행합니다. Correlated subquery는 상위 명령문에 의해 처리된 각 행에 대해 한 번 평가됩니다. 상위 명령문은 SELECT, UPDATE 또는 DELETE 문일 수 있습니다.

### Nested Subquery와 Correlated Subquery 비교

일반 nested subquery를 사용하면 inner SELECT query가 먼저 한 번 실행되어 main query에서 사용할 값을 반환합니다. 그러나 correlated subquery는 outer query에서 고려한 각 후보 행에 대해 한 번 실행됩니다. 즉, inner query를 outer query가 제어합니다.

### Nested Subquery 실행

- Inner query가 먼저 실행되어 값을 찾습니다.
- Outer query는 inner query의 값을 사용하여 한 번 실행됩니다.

### Correlated Subquery 실행

- Outer query에 의해 패치(fetch)된 후보 행을 가져옵니다.
- 후보 행의 값을 사용하여 inner query를 실행합니다.
- Inner query 결과 값을 사용하여 후보 행을 지정하거나 지정을 취소합니다.
- 남아 있는 후보 행이 없을 때까지 반복합니다.

# Correlated Subquery

Subquery는 상위 query에 있는 테이블의 열을 참조합니다.

```
SELECT column1, column2, ...  
FROM   table1 Outer_table  
WHERE  column1 operator  
        (SELECT column1, column2  
         FROM   table2  
         WHERE  expr1 =  
                Outer_table.expr2);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated Subquery(계속)

Correlated subquery는 테이블에 있는 모든 행을 읽고 관련 데이터에 대해 각 행의 값을 비교하는 방법입니다. 이 query는 subquery가 main query에서 고려된 각 후보 행에 대해 다른 결과 또는 결과 집합을 반환해야 할 때마다 사용됩니다. 즉, 상위 명령문에서 처리한 각 행의 값에 따라 응답이 달라지는 multipart 질문에 응답할 때 correlated subquery를 사용합니다.

Oracle 서버는 subquery가 상위 query의 테이블의 열을 참조할 때 correlated subquery를 수행합니다.

**참고:** correlated subquery에서는 ANY 및 ALL 연산자를 사용할 수 있습니다.

## Correlated Subquery 사용

자신의 부서의 평균 급여보다 많은 급여를 받는 사원을 모두 찾습니다.

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
      (SELECT AVG(salary)
       FROM   employees inner_table
       WHERE  inner_table.department_id =
              outer_table.department_id);
```

Outer query의 행이  
처리될 때마다  
inner query가  
평가됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Correlated Subquery 사용

슬라이드의 예제는 자신의 부서의 평균 급여보다 많은 급여를 받는 사원을 확인합니다. 여기에서 correlated subquery는 특히 각 부서의 평균 급여를 계산합니다.

Outer query와 inner query 모두 FROM 절에 EMPLOYEES 테이블을 사용하므로 명확한 구분을 위해 outer SELECT 문의 EMPLOYEES에 alias가 제공됩니다. Alias는 전체 SELECT 문을 보다 읽기 쉽게 만듭니다. 내부 명령문은 inner 테이블 열과 outer 테이블 열을 구분하지 못하기 때문에 alias가 없을 경우 query가 제대로 작동하지 않습니다.

# Correlated Subquery 사용

두 번 이상 직무가 바뀐 사원의 세부 사항을 표시합니다.

```
SELECT e.employee_id, last_name, e.job_id
FROM   employees e
WHERE  2 <= (SELECT COUNT(*)
             FROM job_history
             WHERE employee_id = e.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID
1	200	Whalen	AD_ASST
2	101	Kochhar	AD_VP
3	176	Taylor	SA_REP

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated Subquery 사용(계속)

슬라이드의 예제는 직무가 적어도 두 번 이상 변경된 사원의 세부 정보를 표시합니다.

Oracle 서버는 다음과 같이 correlated subquery를 평가합니다.

1. Outer query에 지정된 테이블에서 행을 선택합니다. 이 행이 현재 후보 행이 됩니다.
2. 이 후보 행의 subquery에서 참조된 열 값을 저장합니다. (슬라이드의 예제에서 subquery에서 참조된 열은 E.EMPLOYEE\_ID입니다.)
3. Outer query의 후보 행의 값을 참조하는 조건으로 subquery를 수행합니다. (슬라이드 예제에서 COUNT(\*) 그룹 함수는 2단계에서 얻은 E.EMPLOYEE\_ID 열의 값에 준하여 평가됩니다.)
4. 3단계에서 수행된 subquery 결과를 기반으로 outer query의 WHERE 절을 평가하고, 이에 따라 해당 후보 행의 출력 여부가 결정됩니다. (예제에서 subquery에 의해 평가된 사원의 직무 변경 횟수는 outer query의 WHERE 절에 있는 2와 비교됩니다. 조건이 충족되면 해당 사원 레코드가 표시됩니다.)
5. 테이블의 모든 행이 처리될 때까지 테이블의 다음 후보 행에 대해 이 과정을 반복합니다.

상관(correlation)은 subquery에서 outer query의 요소를 사용하여 설정됩니다. 이 예제에서 subquery에 있는 테이블의 EMPLOYEE\_ID를 outer query에 있는 테이블의 EMPLOYEE\_ID와 비교합니다.

## 단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- **EXISTS 및 NOT EXISTS 연산자 사용**
- WITH 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## EXISTS 연산자 사용

- **EXISTS 연산자는 subquery의 결과 집합에 행이 존재하는지 테스트합니다.**
- **Subquery 행 값이 있을 경우**
  - 검색이 inner query에서 계속 수행되지 않습니다.
  - 조건에는 TRUE로 플래그 지정됩니다.
- **Subquery 행 값이 없을 경우**
  - 조건은 FALSE로 플래그가 지정됩니다.
  - Inner query에서도 검색이 수행됩니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### EXISTS 연산자

중첩되는 SELECT 문에서는 모든 논리 연산자를 사용할 수 있으며 EXISTS 연산자를 사용할 수 있습니다. 이 연산자는 outer query로 검색된 값이 inner query로 검색된 값의 결과 집합에 존재하는지 여부를 테스트하기 위해 correlated subquery에서 자주 사용됩니다. Subquery가 적어도 한 행을 반환하면 이 연산자는 TRUE를 반환합니다. 값이 없으면 FALSE를 반환합니다. 따라서 NOT EXISTS는 outer query에 의해 검색된 값이 inner query에 의해 검색된 값의 결과 집합의 일부가 아닌지 여부를 테스트합니다.

## EXISTS 연산자 사용

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees outer
WHERE  EXISTS ( SELECT 'X'
                FROM employees
                WHERE manager_id =
                      outer.employee_id);
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	201	Hartstein	MK_MAN	20
2	205	Higgins	AC_MGR	110
3	100	King	AD_PRES	90
4	101	Kochhar	AD_VP	90
5	102	De Haan	AD_VP	90
6	103	Hunold	IT_PROG	60
7	108	Greenberg	FI_MGR	100
8	114	Raphaely	PU_MAN	30

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### EXISTS 연산자 사용

EXISTS 연산자를 사용하면 다음 조건을 가진 관리자와 사원 번호에 대해 일치하는 사원이 적어도 한 명 이상 있을 경우 inner query에서 검색이 중단됩니다.

WHERE manager\_id = outer.employee\_id.

Inner SELECT query는 특정 값을 반환할 필요가 없기 때문에 상수를 선택할 수 있습니다.

## 사원이 없는 부서 모두 찾기

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM employees
                  WHERE department_id = d.department_id);
```

DEPARTMENT_ID	DEPARTMENT_NAME
1	120 Treasury
2	130 Corporate Tax
3	140 Control And Credit
4	150 Shareholder Services
5	160 Benefits
6	170 Manufacturing
7	180 Construction

...

All Rows Fetched: 16

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### NOT EXISTS 연산자 사용

#### 대체 방법

아래 예제에 나오는 것처럼 NOT EXISTS 연산자 대신 NOT IN 생성자를 사용할 수 있습니다.

```
SELECT department_id, department_name
FROM departments
WHERE department_id NOT IN(SELECT department_id
                          FROM employees);
```

All Rows Fetched: 0

그러나 집합의 한 멤버라도 NULL 값이면 NOT IN은 FALSE로 평가됩니다. 따라서 departments 테이블에 WHERE 조건을 만족하는 열이 있더라도 query에서는 아무 행도 반환되지 않습니다.



# Correlated UPDATE

한 테이블의 행에 준하여 다른 테이블의 행을 갱신할 때 correlated subquery를 사용합니다.

```
UPDATE table1 alias1
SET    column = (SELECT expression
                  FROM    table2 alias2
                  WHERE   alias1.column =
                        alias2.column);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated UPDATE

UPDATE 문의 경우 한 테이블의 행에 준하여 다른 테이블의 행을 갱신할 때 correlated subquery를 사용할 수 있습니다.

## Correlated UPDATE 사용

- 부서 이름을 저장할 열을 추가하여 EMPL6 테이블을 비정규화합니다.
- Correlated update를 사용하여 테이블을 채웁니다.

```
ALTER TABLE empl6  
ADD(department_name VARCHAR2(25));
```

```
UPDATE empl6 e  
SET    department_name =  
        (SELECT department_name  
         FROM   departments d  
         WHERE  e.department_id = d.department_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Correlated UPDATE(계속)

슬라이드 예제에서는 부서 이름을 저장하기 위한 열을 추가하여 EMPL6 테이블을 비정규화한 다음 correlated update를 사용하여 테이블을 채웁니다.

다음은 correlated update에 대한 또 다른 예제입니다.

#### 문제

REWARDS 테이블에는 성과 기대치를 초과하는 사원 리스트가 있습니다. Correlated subquery를 사용하여 REWARDS 테이블의 행을 기반으로 EMPL6 테이블의 행을 갱신합니다.

```
UPDATE empl6  
SET    salary = (SELECT empl6.salary + rewards.pay_raise  
                 FROM   rewards  
                 WHERE  employee_id =  
                        empl6.employee_id  
                 AND    payraise_date =  
                        (SELECT MAX(payraise_date)  
                         FROM   rewards  
                         WHERE  employee_id = empl6.employee_id))  
WHERE  empl6.employee_id  
IN      (SELECT employee_id FROM rewards);
```

## Correlated UPDATE(계속)

이 예제에서는 REWARDS 테이블을 사용합니다. REWARDS 테이블에는 EMPLOYEE\_ID, PAY\_RAISE 및 PAYRAISE\_DATE 열이 있습니다. 사원의 급여가 인상될 때마다 사원 ID, 급여 인상폭 및 인상된 급여의 수령 날짜와 같은 세부 정보가 있는 레코드가 REWARDS 테이블에 삽입됩니다. REWARDS 테이블은 한 사원에 대해 두 개 이상의 레코드를 포함할 수 있습니다. PAYRAISE \_DATE 열은 사원의 가장 최근 급여 인상을 식별하는데 사용됩니다.

예제에서 EMPL6 테이블의 SALARY 열은 사원이 받은 최근의 급여 인상을 반영하도록 갱신됩니다. 이렇게 하려면 REWARDS 테이블의 인상 급여에 해당 사원의 현재 급여를 추가합니다.

# Correlated DELETE

Correlated subquery를 사용하면 한 테이블의 행에 준하여 다른 테이블의 행을 삭제할 수 있습니다.

```
DELETE FROM table1 alias1
WHERE column operator
      (SELECT expression
       FROM table2 alias2
       WHERE alias1.column = alias2.column);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated DELETE

DELETE 문의 경우 correlated subquery를 사용하면 다른 테이블에도 존재하는 행만 삭제할 수 있습니다. JOB\_HISTORY 테이블에서 마지막 네 개의 직무 기록 레코드만 관리하려면 사원이 직무를 다섯번째로 바꿨을 때 JOB\_HISTORY 테이블에서 해당 사원의 MIN(START\_DATE)를 찾아서 가장 오래된 JOB\_HISTORY 행을 삭제합니다. 다음 코드는 correlated DELETE를 사용하여 이 작업을 수행하는 방법을 보여줍니다.

```
DELETE FROM emp_history JH
WHERE employee_id =
      (SELECT employee_id
       FROM employees E
       WHERE JH.employee_id = E.employee_id
       AND START_DATE =
         (SELECT MIN(start_date)
          FROM job_history JH
          WHERE JH.employee_id = E.employee_id)
       AND 5 > (SELECT COUNT(*)
                FROM job_history JH
                WHERE JH.employee_id = E.employee_id
                GROUP BY EMPLOYEE_ID
                HAVING COUNT(*) >= 4));
```

# Correlated DELETE 사용

Correlated subquery를 사용하여 EMPL6 테이블에서  
EMP\_HISTORY 테이블에도 있는 행만 삭제합니다.

```
DELETE FROM empl6 E
WHERE employee_id =
      (SELECT employee_id
       FROM emp_history
       WHERE employee_id = E.employee_id);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Correlated DELETE(계속)

### 예제

이 예제에서는 두 개의 테이블이 사용됩니다. 각 테이블은 다음과 같습니다.

- EMPL6 테이블: 현재 모든 사원에 대한 세부 정보를 제공합니다.
- EMP\_HISTORY 테이블: 이전 사원의 세부 정보를 제공합니다.

EMP\_HISTORY는 이전 사원에 대한 데이터를 포함하므로 같은 사원의 레코드가 EMPL6 테이블과 EMP\_HISTORY 테이블에 모두 있을 경우 오류가 발생할 수 있습니다. 슬라이드에 표시된 correlated subquery를 사용하여 이러한 오류 레코드를 삭제할 수 있습니다.

## 단원 내용

- 여러 열 Subquery 작성
- SQL에서 스칼라 Subquery 사용
- Correlated Subquery로 문제 해결
- EXISTS 및 NOT EXISTS 연산자 사용
- **WITH 절 사용**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## WITH 절

- **WITH 절을 사용하면 복합 query 내에서 동일한 query 블록이 두 번 이상 발생하는 경우 그 블록을 SELECT 문에 사용할 수 있습니다.**
- **WITH 절은 query 블록 결과를 검색하여 유저의 임시 테이블스페이스에 저장합니다.**
- **WITH 절은 성능을 개선할 수 있습니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### WITH 절

WITH 절을 사용하여 query 블록을 정의한 다음 query에 사용할 수 있습니다. WITH 절은 일반적으로 `subquery_factoring_clause`라고도 하며, 이 절을 사용하면 복합 query 내에서 이 절이 두 번 이상 나타날 때 동일한 query 블록을 SELECT 문에 재사용할 수 있습니다. 이러한 방식은 특히 query에 동일 query 블록에 대한 참조가 여러 개 있고 조인 및 집계기가 있을 경우에 유용합니다.

WITH 절을 사용하면 query 블록의 평가 비용이 높고 복합 query 내에서 query 블록이 두 번 이상 나타나는 경우 동일한 query를 재사용할 수 있습니다. WITH 절을 사용할 경우 Oracle 서버는 query 블록의 결과를 검색하여 유저의 임시 테이블스페이스에 저장합니다. 이러한 방식으로 성능을 향상시킬 수 있습니다.

### WITH 절의 장점

- Query를 읽기 쉽게 만듭니다.
- Query에 절이 여러 번 나타나도 한 번만 평가합니다.
- 대부분의 경우 대용량 query에 대한 성능을 향상시킬 수 있습니다.

## WITH 절: 예제

**WITH 절을 사용하여 총 급여가 전체 부서의 평균 급여보다 큰 부서의 부서 이름 및 총 급여를 표시하는 query를 작성합니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### WITH 절: 예제

슬라이드의 문제를 해결하려면 아래의 중간 계산이 필요합니다.

1. 모든 부서의 총 급여를 계산하고 WITH 절을 사용하여 결과를 저장합니다.
2. 전체 부서의 평균 급여를 계산하고 WITH 절을 사용하여 결과를 저장합니다.
3. 첫번째 단계에서 계산된 총 급여와 두번째 단계에서 계산된 평균 급여를 비교합니다.  
특정 부서의 총 급여가 전체 부서의 평균 급여보다 많으면 부서 이름과 해당 부서의 총 급여를 표시합니다.

이 문제의 정답은 다음 페이지에 나옵니다.



## WITH 절: 예제

```
WITH
dept_costs AS (
    SELECT d.department_name, SUM(e.salary) AS dept_total
    FROM   employees e JOIN departments d
    ON     e.department_id = d.department_id
    GROUP BY d.department_name),
avg_cost AS (
    SELECT SUM(dept_total)/COUNT(*) AS dept_avg
    FROM   dept_costs)
SELECT *
FROM   dept_costs
WHERE  dept_total >
      (SELECT dept_avg
       FROM avg_cost)
ORDER BY department_name;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### WITH 절: 예제(계속)

슬라이드의 SQL 코드는 WITH 절을 사용하여 훨씬 간단하게 SQL을 작성하고 성능을 향상시킬 수 있는 상황에 대한 예제입니다. Query는 query 이름 DEPT\_COSTS 및 AVG\_COST를 생성하여 main query 본문에 사용합니다. 내부적으로 WITH 절은 인라인 뷰나 임시 테이블(Temporary Table)로 분석됩니다. 옵티마이저는 WITH 절의 결과를 임시로 저장하는 비용이나 이점에 따라 적절한 분석 방법을 선택합니다. 슬라이드에서 SQL 코드로 생성된 출력 결과는 다음과 같습니다.

R2	DEPARTMENT_NAME	R2	DEPT_TOTAL
1	Sales		304500
2	Shipping		156400

### WITH 절 사용 정보

- 이 절은 항상 SELECT 문과 함께 사용됩니다.
- Query 이름은 query 이름 다음에 정의된 모든 WITH 요소 query 블록(subquery 블록 포함)과 main query 블록 자체(subquery 블록 포함)에서 볼 수 있습니다.
- Query 이름이 기존 테이블 이름과 동일한 경우, 구문 분석기는 내부에서 검색을 시작하며 query 블록 이름이 테이블 이름보다 우선합니다.
- WITH 절은 두 개 이상의 query를 보유할 수 있습니다. 각 query는 쉼표로 구분됩니다.

# Recursive WITH 절

## Recursive WITH 절

- Recursive query의 공식화를 가능하게 합니다.
- Recursive WITH 요소 이름이라는 이름으로 query를 작성합니다.
- 앵커와 recursive라는 두 가지 유형의 query 블록 멤버를 포함합니다.
- ANSI와 호환됩니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Recursive WITH 절

Oracle Database 11g Release 2에서 recursive query를 공식화할 수 있도록 WITH 절이 확장되었습니다.

Recursive WITH는 이름, 즉 *Recursive WITH* 요소 이름으로 recursive query를 정의합니다. Recursive WITH 요소 정의에는 앵커 멤버와 recursive 멤버라는 두 개 이상의 query 블록이 포함되어야 합니다. 앵커 멤버는 여러 개 있을 수 있지만 recursive 멤버는 하나만 있을 수 있습니다.

Oracle Database 11g Release 2의 Recursive WITH 절은 부분적으로 ANSI를 따릅니다. Recursive WITH를 사용하여 조직도 등의 계층 데이터를 query할 수 있습니다.

## Recursive WITH 절: 예제

FLIGHTS 테이블

	SOURCE	DESTIN	FLIGHT_TIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8

1

```
WITH Reachable_From (Source, Destin, TotalFlightTime) AS
(
    SELECT Source, Destin, Flight_time
    FROM Flights
    UNION ALL
    SELECT incoming.Source, outgoing.Destin,
           incoming.TotalFlightTime+outgoing.Flight_time
    FROM Reachable_From incoming, Flights outgoing
    WHERE incoming.Destin = outgoing.Source
)
SELECT Source, Destin, TotalFlightTime
FROM Reachable_From;
```

2

	SOURCE	DESTIN	TOTALFLIGHTTIME
1	San Jose	Los Angeles	1.3
2	New York	Boston	1.1
3	Los Angeles	New York	5.8
4	San Jose	New York	7.1
5	Los Angeles	Boston	6.9
6	San Jose	Boston	8.2

3

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Recursive WITH 절: 예제

슬라이드의 예제 1은 두 도시 간의 항공편을 설명하는 FLIGHTS 테이블의 레코드를 보여줍니다.

예제 2의 query로 FLIGHTS 테이블을 query하여 출발지와 목적지 간의 총 비행 시간을 표시할 수 있습니다. 쿼리의 WITH Reachable\_From 절에는 두 개의 분기를 포함하는 UNION ALL query가 있습니다. 첫번째 분기는 Flights 테이블의 모든 행을 선택하는 앵커 분기이고, 두번째 분기는 recursive 분기입니다. Recursive 분기는 Reachable\_From의 내용을 Flights 테이블에 조인하여 이동 가능한 다른 도시를 찾아 Reachable\_From의 내용에 추가합니다. Recursive 분기에서 더 이상 행을 찾을 수 없을 때 작업이 끝납니다.

예제 3은 WITH 절 요소 Reachable\_From의 모든 내용을 선택하는 query의 결과입니다.

자세한 내용은 다음을 참조하십시오.

- Oracle Database SQL Language Reference 11g Release 2.0
- Oracle Database Data Warehousing Guide 11g Release 2.0

## 퀴즈

Correlated subquery를 사용하면 inner SELECT 문이 outer SELECT 문을 제어합니다.

1. 맞습니다.
2. 틀립니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 2

## 요약

이 단원에서는 다음 항목에 대해 설명했습니다.

- 여러 열 subquery는 두 개 이상의 열을 반환합니다.
- 여러 열 비교는 쌍방식이거나 비쌍방식일 수 있습니다.
- 여러 열 subquery는 SELECT 문의 FROM 절에 사용될 수도 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

여러 열 subquery를 사용하여 여러 WHERE 조건을 단일 WHERE 절에 결합할 수 있습니다.

여러 열 subquery의 열 비교는 쌍방식 비교이거나 비쌍방식 비교일 수 있습니다.

Subquery를 사용하여 포함 query로 처리할 테이블을 정의할 수 있습니다.

스칼라 subquery는 다음에서 사용할 수 있습니다.

- DECODE 및 CASE의 조건 및 표현식 부분
- GROUP BY를 제외한 SELECT의 모든 절
- UPDATE 문의 SET 절 및 WHERE 절

## 요약

- **Correlated subquery**는 subquery가 각 후보 행에 대해 항상 다른 결과를 반환해야 하는 경우에 유용합니다.
- **EXISTS** 연산자는 값이 존재하는지 여부를 테스트하는 부울 연산자입니다.
- **Correlated subquery**는 **SELECT**, **UPDATE** 및 **DELETE** 문과 함께 사용할 수 있습니다.
- 동일한 query 블록이 두 번 이상 나타날 경우 **WITH** 절을 사용하여 해당 query 블록을 **SELECT** 문에 사용할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약(계속)

Oracle 서버는 subquery가 상위 명령문에서 참조된 테이블의 열을 참조할 때 correlated subquery를 수행합니다. Correlated subquery는 상위 명령문에 의해 처리된 각 행에 대해 한 번 평가됩니다. 상위 명령문은 **SELECT**, **UPDATE** 또는 **DELETE** 문일 수 있습니다. **WITH** 절을 사용하면 query 블록을 재평가하는 데 비용이 많이 들 때와 복합 query 내에서 두 번 이상 이 절이 나타날 때 동일한 query를 재사용할 수 있습니다.

## 연습 6: 개요

이 연습에서는 다음 내용을 다룹니다.

- 여러 열 Subquery 작성
- Correlated Subquery 작성
- EXISTS 연산자 사용
- 스칼라 Subquery 사용
- WITH 절 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 6: 개요

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







## 정규식 지원

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 목표

**이 단원을 마치면 다음을 수행할 수 있습니다.**

- 정규식 사용 시 이점 나열
- 정규식을 사용하여 문자열 검색, 일치 및 대체

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 목표

이 단원에서는 정규식 지원 기능의 사용법에 대해 알아봅니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

## 단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP\_LIKE
  - REGEXP\_REPLACE
  - REGEXP\_INSTR
  - REGEXP\_SUBSTR
- 하위식 액세스
- REGEXP\_COUNT 함수 사용
- 정규식 및 Check 제약 조건

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 정규식이란?

- 정규식에서 표준 구문 규칙을 사용하여 문자열 데이터의 간단한 패턴 및 복잡한 패턴을 검색하고 조작할 수 있습니다.
- SQL 함수 및 조건 집합을 사용하여 SQL 및 PL/SQL에서 문자열을 검색하고 조작할 수 있습니다.
- 다음을 사용하여 정규식을 지정합니다.
  - 메타 문자: 검색 알고리즘을 지정하는 연산자
  - 리터럴: 검색 중인 문자

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식이란?

오라클 데이터베이스는 정규식을 지원합니다. 정규식의 구현은 ASCII 데이터와 일치하는 의미 및 구문을 위해 IEEE(Institute of Electrical and Electronics Engineers)가 제정한 POSIX(Portable Operating System for UNIX) 표준을 따릅니다. 오라클의 다중 언어 기능은 POSIX 표준을 넘어 연산자의 일치 기능을 확장합니다. 정규식은 검색 및 조작을 위해 간단한 패턴 및 복잡한 패턴을 설명하는 방식입니다.

문자열 조작 및 검색은 웹 기반 응용 프로그램 논리의 대부분을 차지합니다. 사용 범위는 지정된 텍스트에서 "San Francisco"란 단어를 찾는 간단한 검색부터, 텍스트에서 모든 URL을 추출하는 복잡한 검색과 두번째 문자가 모음인 모든 단어를 찾는 매우 복잡한 검색에 이르기까지 다양합니다.

고유의 SQL과 정규식을 함께 사용하면 오라클 데이터베이스에 저장된 임의의 데이터에서 매우 강력한 검색 및 조작 연산을 수행할 수 있습니다. 이 기능을 사용하면 복잡한 프로그래밍을 간단하게 해결할 수도 있습니다.

## 정규식 사용 시 이점

정규식을 사용하여 데이터베이스에서 복잡한 일치 논리를 구현하면 다음과 같이 이점이 있습니다.

- 오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.
- 서버측 정규식을 사용하여 제약 조건을 적용함으로써 클라이언트에서 데이터 검증 논리 코드를 작성할 필요가 없습니다.
- 내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 11g의 이전 버전보다 더욱 쉽고 강력하게 문자열을 조작할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식 사용 시 이점

정규식은 PERL 및 Java와 같은 프로그래밍 언어의 강력한 텍스트 처리 구성 요소입니다. 예를 들어, PERL 스크립트는 디렉토리의 각 HTML 파일을 처리하고 해당 내용을 단일 문자열로 스칼라 변수로 읽어들이는 다음 정규식을 사용하여 문자열에서 URL을 검색합니다. 많은 개발자가 PERL을 사용하는 한 가지 이유는 강력한 패턴 일치 기능이 있기 때문입니다. 개발자는 Oracle의 정규식 지원을 통해 데이터베이스에서 복잡한 일치 논리를 구현할 수 있습니다. 이 기법은 다음과 같은 이유로 유용합니다.

- 오라클 데이터베이스에서 일치 논리를 중앙화함으로써 middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다. SQL 정규식 함수는 처리 논리를 데이터에 더욱 근접하게 이동함으로써 더욱 효율적인 솔루션을 제공합니다.
- Oracle Database 10g 이전에는 개발자가 일반적으로 클라이언트에서 데이터 검증 논리 코드를 작성했으므로 여러 클라이언트에 대해 중복되는 동일한 검증 논리가 필요했습니다. 서버측 정규식을 사용하여 제약 조건을 적용하면 이러한 문제가 해결됩니다.
- 내장 SQL 및 PL/SQL 정규식 함수와 조건을 사용하여 Oracle Database 10g의 이전 버전보다 더욱 강력하면서도 간단하게 문자열을 조작할 수 있습니다.

## SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

함수 또는 조건 이름	설명
REGEXP_LIKE	LIKE 연산자와 유사하지만 간단한 패턴 일치(조건) 대신 정규식 일치를 수행합니다.
REGEXP_REPLACE	정규식 패턴을 검색하여 대체 문자열로 바꿉니다.
REGEXP_INSTR	정규식 패턴에 대해 문자열을 검색하고 일치이 발견된 위치를 반환합니다.
REGEXP_SUBSTR	지정된 문자열 내에서 정규식 패턴을 검색하고 일치하는 부분 문자열을 추출합니다.
REGEXP_COUNT	입력 문자열에서 패턴 일치이 발견되는 횟수를 반환합니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SQL 및 PL/SQL에서 정규식 함수 및 조건 사용

오라클 데이터베이스는 정규식을 사용하여 문자열을 검색 및 조작할 수 있는 일련의 SQL 함수를 제공합니다. 텍스트 리터럴, 바인드 변수 또는 CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2, VARCHAR2 (LONG 제외) 와 같은 문자 데이터를 포함하는 열에서 이러한 함수를 사용합니다. 정규식은 작은 따옴표로 묶어야 합니다. 이렇게 하면 SQL 함수가 전체 표현식을 해석하고 코드의 가독성을 높일 수 있습니다.

- REGEXP\_LIKE: 이 조건은 패턴에 대한 문자 열을 검색합니다. Query의 WHERE 절에서 이 조건을 사용하여 지정한 정규식과 일치하는 행을 반환합니다.
- REGEXP\_REPLACE: 이 함수는 문자 열에서 패턴을 검색하고 해당 패턴의 각 발생 값을 지정한 패턴으로 대체합니다.
- REGEXP\_INSTR: 이 함수는 문자열에서 지정한 정규식 패턴의 발생 값을 검색합니다. 찾고자 하는 발생 값과 검색 시작 위치를 지정합니다. 이 함수는 일치이 발견된 문자열의 위치를 나타내는 정수를 반환합니다.
- REGEXP\_SUBSTR: 이 함수는 지정한 정규식 패턴과 일치하는 실제 부분 문자열을 반환합니다.
- REGEXP\_COUNT: 이 함수는 입력 문자열에서 패턴 일치이 발견되는 횟수를 반환합니다.

## 단원 내용

- 정규식 소개
- **정규식에서 메타 문자 사용**
- 정규식 함수 사용:
  - REGEXP\_LIKE
  - REGEXP\_REPLACE
  - REGEXP\_INSTR
  - REGEXP\_SUBSTR
- 하위식 액세스
- REGEXP\_COUNT 함수 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 메타 문자란?

- 메타 문자는 대체 문자, 반복 문자, 일치하지 않는 문자 또는 일련의 문자와 같이 특별한 의미를 지닌 특수 문자입니다.
- 패턴 일치에서 여러 개의 미리 정의된 메타 문자 기호를 사용할 수 있습니다.
- 예를 들어, `^(f|ht)tps?:$` 정규식은 문자열 시작 부분에서 다음을 검색합니다.
  - 리터럴 `f` 또는 `ht`
  - `t` 리터럴
  - `p` 리터럴(선택적으로 다음에 `s` 리터럴이 나옴)
  - 문자열 끝부분의 `:"` 리터럴

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 메타 문자란?

슬라이드의 정규식은 `http:`, `https:`, `ftp:` 및 `ftps:` 문자열을 일치시킵니다.

**참고:** 정규식의 메타 문자 전체 리스트는

*Oracle Database Advanced Application Developer's Guide 11g Release 2*를 참조하십시오.



## 정규식에서 메타 문자 사용

구문	설명
.	지원되는 character set에서 NULL을 제외한 임의의 문자와 일치
+	한 번 이상 발생 수 일치
?	0 또는 1번 발생 수 일치
*	선행 하위식의 0번 이상 발생 수 일치
{m}	선행 표현식의 정확히 m번 발생 수 일치
{m, }	선행 하위식과 최소 m번 이상 발생 수 일치
{m,n}	선행 하위식의 최소 m번 이상, 최대 n번 이하 발생 수 일치
[...]	괄호 안의 리스트에 있는 임의의 단일 문자와 일치
	여러 대안 중 하나와 일치
( ... )	괄호로 묶인 표현식을 한 단위로 취급합니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식 함수에서 메타 문자 사용

임의의 문자, ".": **a.b**는 문자열 **abb, acb, adb**와 일치하지만 **acc**와는 일치하지 않습니다.

하나 이상, "+": **a+**는 문자열 **a, aa, aaa**와 일치하지만 **bbb**와는 일치하지 않습니다.

0 또는 1, "?": **ab?c**는 문자열 **abc, ac**와 일치하지만 **abbc**와는 일치하지 않습니다.

0 이상, "\*": **ab\*c**는 문자열 **ac, abc, abbc**와 일치하지만 **abb**와는 일치하지 않습니다.

정확한 숫자, "{m}": **a{3}**는 문자열 **aaa**와 일치하지만 **aa**와는 일치하지 않습니다.

최소 숫자, "{m,}": **a{3,}**은 문자열 **aaa, aaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

사이 숫자, "{m,n}": **a{3,5}**는 문자열 **aaa, aaaa, aaaaa**와 일치하지만 **aa**와는 일치하지 않습니다.

문자 리스트 일치, "[...]": **[abc]**는 문자열 **a11, bill, cold**의 첫번째 문자와 일치하지만 **doll**의 어떠한 문자와도 일치하지 않습니다.

또는 "|": **a|b**는 문자 **a** 또는 문자 **b**와 일치합니다.

하위식, "(...)": **(abc)?def**는 선택적 문자열 **abc** 다음에 **def**가 나옵니다. 이 표현식은 **abcdefghi** 및 **def**와 일치하지만 **ghi**와는 일치하지 않습니다. 하위식은 리터럴의 문자열이나 연산자를 포함한 복잡한 표현식이 될 수 있습니다.

## 정규식에서 메타 문자 사용

구문	설명
<b>^</b>	문자열 시작 부분과 일치
<b>\$</b>	문자열 끝부분과 일치
<b>\</b>	표현식에서 후속 메타 문자를 리터럴로 처리합니다.
<b>\n</b>	괄호 안에 그룹화된 <i>n</i> 번째 (1-9) 선행 하위식과 일치합니다. 괄호는 표현식이 기억되도록 만들고 backreference에서 표현식을 참조합니다.
<b>\d</b>	숫자 문자
<b>[ :class: ]</b>	지정된 POSIX 문자 클래스에 속한 임의의 문자와 일치
<b>[ ^:class: ]</b>	괄호 안의 리스트에 <i>없는</i> 임의의 단일 문자와 일치

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식 함수에서 메타 문자 사용(계속)

행 앵커의 시작/끝, "^" 및 "\$": **^def**는 문자열 **defghi**의 **def**와 일치하지만 **abcdef**의 **def**와는 일치하지 않습니다. **def\$**는 문자열 **abcdef**의 **def**와 일치하지만 문자열 **defghi**의 **def**와는 일치하지 않습니다.

이스케이프 문자 "\": **\+**는 **+**를 검색합니다. 문자열 **abc+def**의 더하기(+) 문자와 일치하지만 **Abcdef**와는 일치하지 않습니다.

Backreference, "**\n**": **(abc|def)xy\1**은 문자열 **abcxyabc** 및 **defxydef**와 일치하지만 **abcxydef** 또는 **abcxy**와는 일치하지 않습니다. backreference를 사용하면 실제 문자열을 미리 알고 있지 않아도 반복되는 문자열을 검색할 수 있습니다. 예를 들어, **^(.\*)\1\$** 표현식은 동일한 문자열의 2개의 인접한 instance로 구성된 행과 일치합니다.

숫자 문자, "**\d**": **^\d{3}\d{3}-\d{4}\$** 표현식은 **[650] 555-1212**와 일치하지만 **650-555-1212**와는 일치하지 않습니다.

문자 클래스, "**[ :class: ]**": **[ :upper: ]+**는 하나 이상의 연속되는 대문자를 검색합니다. 이 경우 문자열 **abcDEFghi**의 **DEF**와 일치하지만 문자열 **abcdefghijkl**와는 일치하지 않습니다.

비일치 문자 리스트 (또는 클래스), "**[ ^... ]**": **[ ^abc ]**는 문자열 **abcdef**의 문자 **d**와 일치하지만 문자 **a**, **b** 또는 **c**와는 일치하지 않습니다.

## 단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP\_LIKE
  - REGEXP\_REPLACE
  - REGEXP\_INSTR
  - REGEXP\_SUBSTR
- 하위식 액세스
- REGEXP\_COUNT 함수 사용

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 정규식 함수 및 조건: 구문

```
REGEXP_LIKE (source_char, pattern [,match_option]
```

```
REGEXP_INSTR (source_char, pattern [, position  
[, occurrence [, return_option  
[, match_option [, subexpr]]]])
```

```
REGEXP_SUBSTR (source_char, pattern [, position  
[, occurrence [, match_option  
[, subexpr]]]])
```

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence  
[, match_option]]]])
```

```
REGEXP_COUNT (source_char, pattern [, position  
[, occurrence [, match_option]]]])
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식 함수 및 조건: 구문

정규식 함수 및 조건의 구문은 다음과 같습니다.

- **source\_char**: 검색 값 역할을 하는 문자 표현식
- **pattern**: 정규식, 텍스트 리터럴
- **occurrence**: Oracle 서버가 검색해야 하는 **source\_char**에서 패턴의 발생 값을 나타내는 양의 정수. 기본값은 1입니다.
- **position**: Oracle 서버가 검색을 시작해야 하는 **source\_char**의 문자를 나타내는 양의 정수. 기본값은 1입니다.
- **return\_option**:
  - 0: 발생 값의 첫번째 문자의 위치를 반환합니다(기본값).
  - 1: 발생 값 다음의 문자 위치를 반환합니다.
- **Replacestr**: 패턴 대체 문자열
- **match\_parameter**:
  - "c": 대소문자를 구분하는 일치 사용(기본값)
  - "i": 대소문자를 구분하지 않는 일치 사용
  - "n": 임의의 문자 일치 연산자 허용
  - "m": 소스 문자열을 다중 행으로 처리
- **subexpr**: 괄호로 묶인 패턴 부분. 하위식은 이 단원 뒷부분에서 다룹니다.

## REGEXP\_LIKE 조건을 사용하여 기본 검색 수행

```
REGEXP_LIKE(source_char, pattern [, match_parameter ])
```

```
SELECT first_name, last_name  
FROM employees  
WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en$');
```

	FIRST_NAME	LAST_NAME
1	Steven	King
2	Steven	Markle
3	Stephen	Stiles

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_LIKE 조건을 사용하여 기본 검색 수행

REGEXP\_LIKE는 LIKE에서 수행하는 단순한 패턴 일치 대신 정규식 일치를 수행한다는 점을 제외하고 LIKE 조건과 일치합니다. 이 조건은 입력 character set에 의해 정의된 문자를 사용하여 문자열을 평가합니다.

#### REGEXP\_LIKE의 예제

EMPLOYEES 테이블에 대한 이 query에서는 first name에 Steven 또는 Stephen이 포함된 모든 사원이 표시됩니다. 다음은 query에 사용된 표현식 '^Ste(v|ph)en\$'에 대한 설명입니다.

- ^는 표현식의 시작을 나타냅니다.
- \$는 표현식의 끝을 나타냅니다.
- |는 둘 중 하나(또는)를 나타냅니다.

## REGEXP\_REPLACE 함수를 사용하여 패턴 대체

```
REGEXP_REPLACE(source_char, pattern [,replacestr  
[, position [, occurrence [, match_option]]])
```

```
SELECT REGEXP_REPLACE(phone_number, '\.','-') AS phone  
FROM employees;
```

원본

	LAST_NAME	PHONE
1	OConnell	650.507.9833
2	Grant	650.507.9844
3	Whalen	515.123.4444
4	Hartstein	515.123.5555

일부 결과

	LAST_NAME	PHONE
1	OConnell	650-507-9833
2	Grant	650-507-9844
3	Whalen	515-123-4444
4	Hartstein	515-123-5555

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_REPLACE 함수를 사용하여 패턴 대체

REGEXP\_REPLACE 함수를 사용하여 마침표(.) 구분자를 대시(-) 구분자로 대체하도록 전화 번호 형식을 다시 지정할 수 있습니다. 다음은 정규식 예제에서 사용되는 각 요소에 대한 설명입니다.

- phone\_number는 소스 열입니다.
- '\.'는 검색 패턴입니다.
  - 작은 따옴표(')를 사용하여 리터럴 문자 마침표(.)를 검색합니다.
  - 백슬래시(\)를 사용하여 일반적으로 메타 문자로 처리되는 문자를 검색합니다.
- '-'는 대체 문자열입니다.

## REGEXP\_INSTR 함수를 사용하여 패턴 찾기

```
REGEXP_INSTR (source_char, pattern [, position [,
occurrence [, return_option [, match_option]]])
```

```
SELECT street_address,
REGEXP_INSTR(street_address,'[[:alpha:]]') AS
    First_Alpha_Position
FROM locations;
```

	STREET_ADDRESS	FIRST_ALPHA_POSITION
1	1297 Via Cola di Rie	6
2	93091 Calle della Testa	7
3	2017 Shinjuku-ku	6
4	9450 Kamiya-cho	6

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_INSTR 함수를 사용하여 패턴 찾기

이 예제에서는 REGEXP\_INSTR 함수로 주소를 검색하여 대소문자 여부와 상관없이 첫번째 알파벳 문자의 위치를 찾습니다. `[<class>:]`는 문자 클래스를 나타내며 해당 클래스 내의 임의의 문자와 일치합니다. `[alpha:]`는 임의의 알파벳 문자와 일치합니다. 부분적인 결과가 표시됩니다.

다음은 query에 사용된 표현식 `'[alpha:]'`에 대한 설명입니다.

- `[`는 표현식을 시작합니다.
- `[alpha:]`는 알파벳 문자 클래스를 나타냅니다.
- `]`는 표현식을 종료합니다.

**참고:** POSIX 문자 클래스 연산자를 사용하면 특정 POSIX 문자 클래스의 멤버인 문자 리스트 내에서 표현식을 검색할 수 있습니다. 이 연산자를 사용하여 대문자와 같은 특정 형식을 검색하거나 자릿수 또는 구두점 문자 등의 특수 문자를 검색할 수 있습니다. 전체 POSIX 문자 클래스를 지원합니다. `[class:]` 구문을 사용합니다. 여기서 `class`는 검색할 POSIX 문자 클래스의 이름을 나타냅니다. 정규식 `[[:upper:]]+`는 하나 이상의 연속하는 대문자를 검색합니다.

## REGEXP\_SUBSTR 함수를 사용하여 부분 문자열 추출

```
REGEXP_SUBSTR (source_char, pattern [, position  
                [, occurrence [, match_option]])
```

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ]+ ' ) AS Road  
FROM locations;
```

	ROAD
1	Via
2	Calle
3	(null)
4	(null)
5	Jabberwocky

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_SUBSTR 함수를 사용하여 부분 문자열 추출

이 예제에서는 LOCATIONS 테이블에서 road name을 추출합니다. 이를 위해 REGEXP\_SUBSTR 함수를 사용하여 STREET\_ADDRESS 열에서 첫번째 공백 뒤에 있는 내용을 반환합니다. 다음은 query에 사용된 표현식 ' [^ ]+ '에 대한 설명입니다.

- [는 표현식을 시작합니다.
- ^는 NOT을 나타냅니다.
- 공백을 나타냅니다.
- ]는 표현식을 종료합니다.
- +는 1 이상을 나타냅니다.
- 공백을 나타냅니다.



## 단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP\_LIKE
  - REGEXP\_REPLACE
  - REGEXP\_INSTR
  - REGEXP\_SUBSTR
- 하위식 액세스
- REGEXP\_COUNT 함수 사용

ORACLE

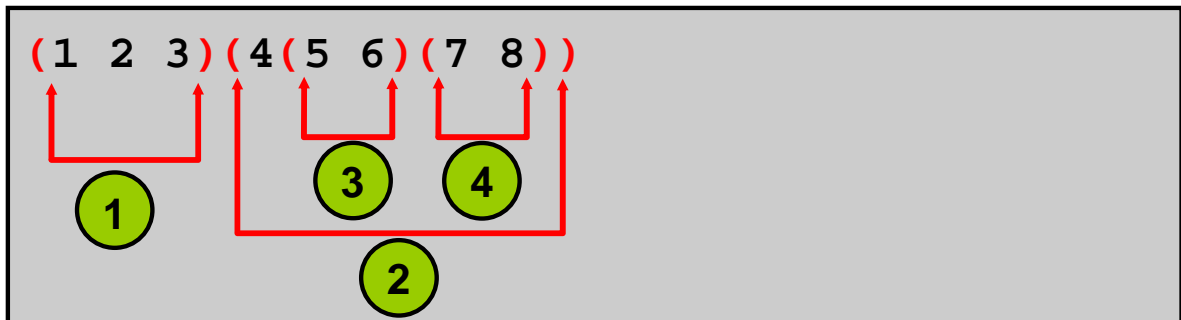
Copyright © 2009, Oracle. All rights reserved.

# 하위식

다음 표현식을 살펴보십시오.

( 1 2 3 ) ( 4 ( 5 6 ) ( 7 8 ) )

하위식은 다음과 같습니다.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

## 하위식

Oracle Database 11g는 하위식에 액세스할 수 있도록 정규식 지원 파라미터를 제공합니다. 슬라이드 예제에 숫자 문자열이 표시되어 있습니다. 괄호는 숫자 문자열 내의 하위식을 식별합니다. 왼쪽에서 오른쪽, 외부 괄호에서 내부 괄호 순서로 읽을 경우 숫자 문자열의 하위식은 다음과 같습니다.

1. 123
2. 45678
3. 56
4. 78

REGEXP\_INSTR 및 REGEXP\_SUBSTR 함수를 사용하여 이러한 하위식을 검색할 수 있습니다.

## 정규식을 지원하는 하위식 사용

```
SELECT
  REGEXP_INSTR
    ① ('0123456789',      -- source char or search value
    ② '(123)(4(56)(78))', -- regular expression patterns
    ③ 1,                  -- position to start searching
    ④ 1,                  -- occurrence
    ⑤ 0,                  -- return option
    ⑥ 'i',                -- match option (case insensitive)
    ⑦ 1)                 -- sub-expression on which to search
    "Position"
FROM dual;
```

Position	
1	2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식을 지원하는 하위식 사용

REGEXP\_INSTR 및 REGEXP\_SUBSTR에는 계산되는 정규식의 특정 부분 문자열을 대상으로 할 수 있는 선택적 SUBEXPR 파라미터가 있습니다.

슬라이드의 예제에서 하위식 리스트의 첫번째 하위식 패턴을 검색하려고 합니다. 제공된 예제는 REGEXP\_INSTR 함수에 대한 여러 파라미터를 식별합니다.

1. 검색 중인 문자열이 식별됩니다.
2. 하위식이 식별됩니다. 첫번째 하위식은 123이고 두번째는 45678, 세번째는 56, 네번째는 78입니다.
3. 세번째 파라미터는 검색을 시작할 위치를 식별합니다.
4. 네번째 파라미터는 찾으려는 패턴의 발생 값을 식별합니다. 1은 첫번째 발생 값을 찾는다는 의미입니다.
5. 다섯번째 파라미터는 반환 옵션입니다. 발생 값의 첫번째 문자의 위치입니다. 1을 지정하면 발생 값 다음의 문자 위치가 반환됩니다.
6. 여섯번째 파라미터는 검색에서 대소문자를 구분해야 하는지 여부를 식별합니다.
7. 마지막 파라미터는 Oracle Database 11g에서 추가된 파라미터입니다. 이 파라미터는 찾으려는 하위식을 지정합니다. 제공된 예제에서는 첫번째 하위식을 검색하며, 결과는 123입니다.

## *n*번째 하위식에 액세스하는 이유

- 더욱 실제적인 사용: DNA 시퀀싱
- 쥐의 DNA에서 면역에 필요한 단백질을 식별하는 특정 하위 패턴을 찾으려고 합니다.

```
SELECT
  REGEXP_INSTR('ccacctttccctccactcctcacgttctcacctgtaaagcgtccctc
cctcatcccatgcccccttaccctgcagggtagtaggctagaaaccagagagctccaagc
tccatctgtggagaggtgccatccttgggctgcagagagaggagaatttgcccaaagctgcc
tgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttca
ccctgccagcaggacactgcagcaccctaaagggctcccaggagtagggtgccctcaagag
gctcttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggc
atgtaggggctggggatgcgctctgctctcctctcctgaaccctgaaccctctggc
taccagagcacttagagccag',
    '(gtc(tcac)(aaag))',
    1, 1, 0, 'i',
    1) "Position"
FROM dual;
```

Position
1 195

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### *n*번째 하위식에 액세스하는 이유

생명 과학 분야에서 추가 처리를 위해 DNA 시퀀스로부터 하위식 일치의 오프셋을 추출해야 하는 경우가 있습니다. 예를 들어, gtc가 앞에 나오고 이어서 tcac, aaag가 나오는 DNA 시퀀스의 시작 오프셋과 같은 특정 단백질 시퀀스를 찾으려고 합니다. 이러한 결과를 얻으려면 일치가 발견된 위치를 반환하는 REGEXP\_INSTR 함수를 사용하면 됩니다.

슬라이드 예제에서는 첫번째 하위식 (gtc)의 위치가 반환됩니다. gtc는 DNA 문자열의 195 위치에서 시작되는 것으로 나타납니다.

슬라이드 예제를 수정하여 두번째 하위식 (tcac)를 검색하면 query 결과는 다음과 같이 출력됩니다. tcac는 DNA 문자열의 198 위치에서 시작되는 것으로 나타납니다.

Position
1 198

슬라이드 예제를 수정하여 세번째 하위식 (aaag)를 검색하면 query 결과는 다음과 같이 출력됩니다. aaag는 DNA 문자열의 202 위치에서 시작하는 것으로 나타납니다.

Position
1 202

## REGEXP\_SUBSTR: 예제

```
SELECT
  REGEXP_SUBSTR
  ① ('acgctgcactgca', -- source char or search value
  ② 'acg(.*?)gca',    -- regular expression pattern
  ③ 1,                -- position to start searching
  ④ 1,                -- occurrence
  ⑤ 'i',              -- match option (case insensitive)
  ⑥ 1)                -- sub-expression
  "Value"
FROM dual;
```

	Value
1	ctgcact

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_SUBSTR: 예제

슬라이드의 예제는 다음과 같이 설명됩니다.

1. acgctgcactgca는 검색할 소스입니다.
2. acg(.\*?)gca는 검색할 패턴입니다. acg 다음에 gca가 오는 문자열을 찾습니다. acg와 gca 사이에 다른 문자가 포함될 수 있습니다.
3. 소스의 첫번째 문자에서 검색을 시작합니다.
4. 패턴의 첫번째 발생 값을 검색합니다.
5. 소스에서 대소문자를 구분하지 않는 일치를 사용합니다.
6. 대상으로 지정할 *n*번째 하위식을 식별하는 음이 아닌 정수 값을 사용합니다. 이는 하위식 파라미터입니다. 이 예제에서 1은 첫번째 하위식을 나타냅니다. 0-9 사이의 값을 사용할 수 있습니다. 0은 하위식이 대상으로 지정되지 않음을 의미합니다. 이 파라미터에 대한 기본값은 0입니다.

## 단원 내용

- 정규식 소개
- 정규식에서 메타 문자 사용
- 정규식 함수 사용:
  - REGEXP\_LIKE
  - REGEXP\_REPLACE
  - REGEXP\_INSTR
  - REGEXP\_SUBSTR
- 하위식 액세스
- **REGEXP\_COUNT 함수 사용**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## REGEXP\_COUNT 함수 사용

```
REGEXP_COUNT (source_char, pattern [, position
              [, occurrence [, match_option]])
```

```
SELECT REGEXP_COUNT(
  'ccacctttccctccactcctcacgttctcacctgtaaagcgctccctccctcatcccatgcccccttaccctgcag
  ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggtgcagagagaggag
  aatttgcccaaagctgctgcagagcttcaccacccttagtctcacaagccttgagttcatagcatttcttgagtt
  ttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtc
  tgatggccacatcctggaattgtttcaagttgatggtcacagccctgaggcatgtagggcgctggggatgcgctctg
  ctctgctctcctctcctgaacccctgaacccctctggctaccccagagcacttagagccag' ,
  'gtc') AS Count
FROM dual;
```

	COUNT
1	4

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### REGEXP\_COUNT 함수 사용

REGEXP\_COUNT 함수는 입력 character set에서 정의된 문자를 사용하여 문자열을 평가합니다. 이 함수는 패턴의 발생 수를 나타내는 정수를 반환합니다. 일치 발견되지 않으면 이 함수는 0을 반환합니다.

슬라이드 예제에서는 REGEXP\_COUNT 함수를 사용하여 DNA 부분 문자열의 발생 수를 판별합니다.

다음 예제는 문자열 123123123123에서 패턴 123이 발생하는 횟수가 세 번임을 보여줍니다. 검색은 문자열의 두번째 위치에서 시작됩니다.

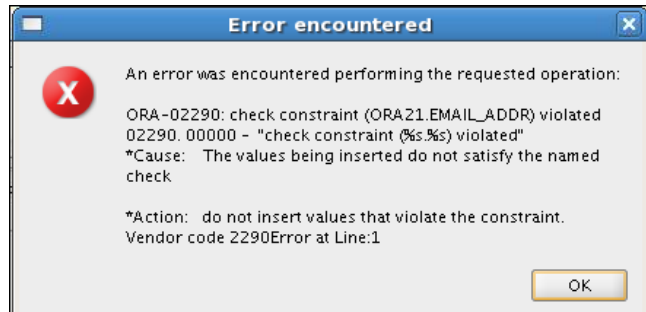
```
SELECT REGEXP_COUNT
  ('123123123123', -- source char or search value
   '123',          -- regular expression pattern
   2,              -- position where the search should start
   'i')            -- match option (case insensitive)
  As Count
FROM dual;
```

	COUNT
1	3

## 정규식 및 Check 제약 조건: 예제

```
ALTER TABLE emp8  
ADD CONSTRAINT email_addr  
CHECK(REGEXP_LIKE(email, '@')) NOVALIDATE;
```

```
INSERT INTO emp8 VALUES  
(500, 'Christian', 'Patel', 'ChrisP2creme.com',  
1234567890, '12-Jan-2004', 'HR_REP', 2000, null, 102, 40);
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 정규식 및 Check 제약 조건: 예제

CHECK 제약 조건에서도 정규식을 사용할 수 있습니다. 이 예제에서는 CHECK 제약 조건이 EMPLOYEES 테이블의 EMAIL 열에 추가되었습니다. 이렇게 하면 "@" 기호를 포함한 문자열만을 받아들입니다. 제약 조건을 테스트합니다. 전자 메일 주소에 필요한 필수 기호가 포함되어 있지 않으므로 CHECK 제약 조건을 위반했습니다. NOVALIDATE 절은 기존 데이터를 검사했는지 여부를 확인합니다.

슬라이드 예제에서는 다음 코드를 사용하여 emp8 테이블을 생성합니다.

```
CREATE TABLE emp8 AS SELECT * FROM employees;
```

**참고:** 슬라이드의 예제는 SQL Developer의 "Execute Statement" 옵션을 사용하여 실행합니다. "Run Script" 옵션을 사용하는 경우 출력 형식이 달라집니다.



## 퀴즈

SQL 및 PL/SQL에서 정규식을 사용할 경우 옳은 설명을 고르십시오.

1. middle-tier 응용 프로그램에 의한 SQL 결과 집합의 집중적인 문자열 처리를 방지할 수 있습니다.
2. 클라이언트에서 데이터 검증 논리를 작성할 필요가 없습니다.
3. 서버의 제약 조건을 적용할 수 있습니다.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

정답: 1, 2, 3

## 요약

**이 단원에서는 정규식을 사용하여 문자열을 검색, 일치 및 대체하는 방법을 배웁니다.**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 요약

이 단원에서는 정규식 지원 기능의 사용법을 배웁니다. 정규식 지원은 SQL 및 PL/SQL에서 모두 사용할 수 있습니다.

## 연습 7: 개요

이 연습에서는 정규식 함수를 사용하여 다음 작업을 수행하는 방법을 다룹니다.

- 데이터 검색, 대체 및 조작
- 새 CONTACTS 테이블을 생성하고 해당 전화 번호가 특정 표준 형식으로 데이터베이스에 입력되도록 p\_number 열에 CHECK 제약 조건을 추가
- 다양한 형식을 사용하여 p\_number 열에 일부 전화 번호 추가 테스트

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### 연습 7: 개요

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

