

[교재 제목]

ORACLE DATABASE 11g SQL

테스트

◆ 문서 작성자: 신상현

◆ 최종 수정일: 2017.07.13.

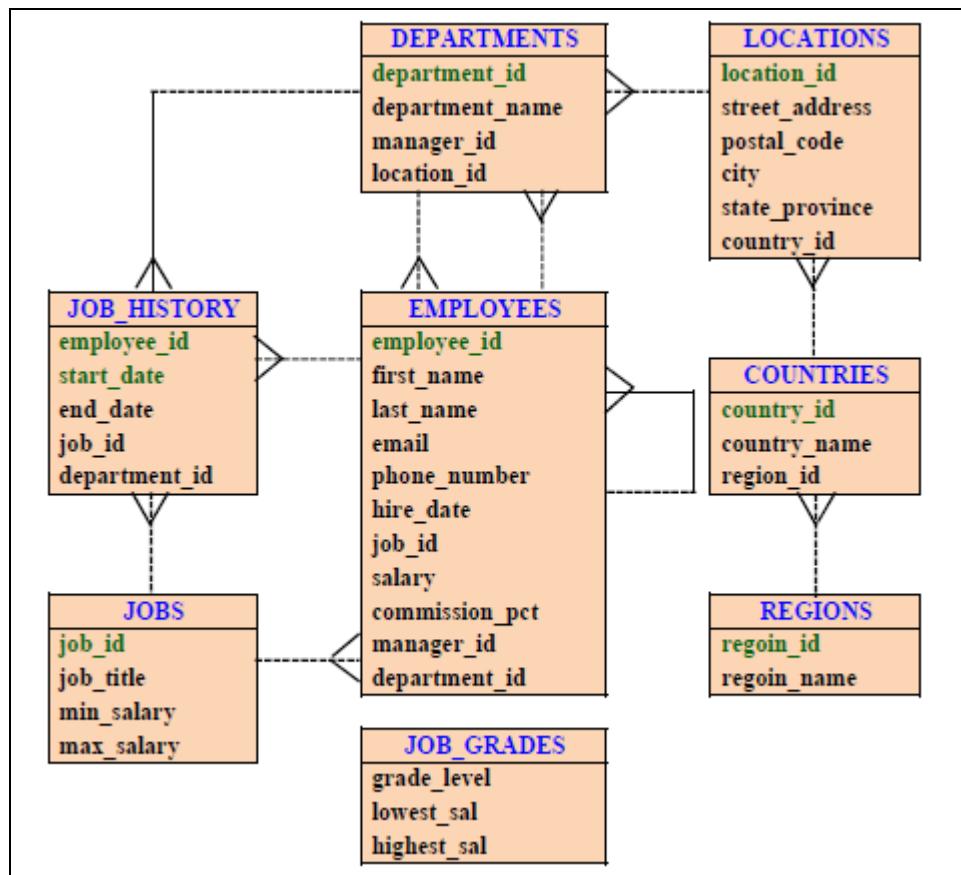
◆ 본 교재는 상업적인 용도로 사용하는 것을 절대로 금지하며, 학습을 위한 목적으로만 배포가 가능합니다.

문서를 배포할 때는 문서 작성자의 이름을 삭제하면 절대로 안됩니다.

◆ 본 교재의 내용을 복사하거나 본 문서에 주석을 삽입하는 것은 불가능합니다.

◆ 개요.

- 본 교재는, ORACLE SQL문에 대한 학습을 위해 작성되었습니다. 문서의 실습에서 ORACLE DATABASE 11g Release 2를 버전을 기준으로 내용이 설명되어 있습니다. 문서에서 설명하는 내용에 대한 주제는 문서의 목차를 참고하십시오.
- 본 교재에서 SQL문 예제들을 실습할 때, LINUX 운영체제에 "General Purpose or Transaction Processing" 옵션 및 [Sample Schema] 옵션을 선택하여 생성된 오라클 데이터베이스 서비스에서 기본으로 제공되는 실습용 계정들과 테이블의 데이터를 사용하며, 특히 HR 계정으로 오라클 데이터베이스에 접속하여 대부분의 실습을 수행합니다.
- 다음은 HR 계정이 소유한 테이블들의 이름과 컬럼들의 이름 및 각 테이블의 관계들을 표시한 그림입니다



- 위의 각 테이블이 저장하고 있는 정보는 다음과 같습니다.
- EMPLOYEES 테이블은 각 사원에 대한 정보를 포함합니다.
- DEPARTMENTS 테이블은 각 부서에 대한 정보를 포함하며, EMPLOYEES 테이블에 대한 기본키(기본키) 테이블이자, LOCATIONS 테이블의 하위 테이블입니다.
- LOCATIONS 테이블은 부서가 위치한 주소 정보를 포함하며, DEPARTMENTS 테이블에 대한 기본키 테이블입니다.
- COUNTRIES 테이블은 부서가 위치한 국가이름 정보를 포함하며, LOCATIONS 테이블에 대한 기본키 테이블입니다.
- REGIONS 테이블은 부서가 위치한 국가의 지역 정보를 포함하며, COUNTRIES 테이블에 대한 기본키 테이블입니다.
- JOB_HISTORY 테이블에는 사원의 과거 직무 기록이 저장됩니다.
- JOBS 테이블은 사원이 수행하는 직책과 관련된 정보와 각 직책에 대한 급여 범위를 포함합니다.
- JOB_GRADES 테이블은 직급별 급여 범위를 식별합니다. 급여 범위는 겹치지 않습니다.

- 본 교재의 실습에 자주 사용하는 테이블의 각 컬럼의 이름과 의미는 다음과 같습니다.

○ HR.EMPLOYEES 테이블에는 사원과 관련된 데이터가 저장되어 있으며, 다음과 같은 의미의 컬럼들로 구성됩니다.

컬럼이름	의미	컬럼이름	의미
EMPLOYEE_ID	사원의 사원번호(사번)	FIRST_NAME	사원의
LAST_NAME	사원의	EMAIL	사원의 이메일
PHONE_NUMBER	사원의 전화번호	HIRE_DATE	사원의 입사일
JOB_ID	사원의 직책코드	SALARY	사원의 급여
MANAGER_ID	사원의 관리자 사원번호	COMMISSION_PCT	영업부서 사원의 급여에 대한 커미션비율
DEPARTMENT_ID	사원의 근무부서 코드		

○ HR.DEPARTMENTS 테이블에는 부서와 관련된 데이터가 저장되어 있으며, 다음과 같은 의미의 컬럼들로 구성됩니다.

컬럼이름	의미	컬럼이름	의미
DEPARTMENT_ID	부서의 식별코드	DEPARTMENT_NAME	부서의 이름
MANAGER_ID	부서의 관리자 사원번호	LOCATION_ID	부서가 위치한 주소코드

○ HR.LOCATIONS 테이블에는 부서의 주소정보와 관련된 데이터가 저장되어 있으며, 다음과 같은 의미의 컬럼들로 구성됩니다.

컬럼이름	의미	컬럼이름	의미
LOCATIONS_ID	주소의 식별코드	STREET_ADDRESS	주소의 상세주소
POSTAL_CODE	주소의 우편번호	CITY	주소가 속한 도시명
STATE_PROVINCE	주소가 속한 주 (州)	COUNTRY_ID	주소가 속한 국가코드

○ HR.JOB_HISTORY 테이블에는 사원의 과거 직무와 관련된 데이터가 저장되어 있으며, 다음과 같은 의미의 컬럼들로 구성됩니다.

컬럼이름	의미	컬럼이름	의미
EMPLOYEE_ID	사원의 사원번호	START_DATE	과거 직책 시작일
END_DATE	과거 직책 종료일	JOB_ID	과거 직책코드
DEPARTMENT_ID	과거 근무 부서코드	-	-

- 본 교재는 다음의 문서들을 참고하여 작성되었습니다.

- Oracle® Database SQL Language Reference 매뉴얼.
- Oracle® Database Reference 매뉴얼.

- 다음은 SQL-학습 시에 도움을 주는 오라클 사이트의 URL-주소입니다.

홈페이지 이름	URL
○ 한국 Oracle 웹 사이트	http://www.oracle.com/kr/index.html
○ 오라클 데이터베이스 Documentation 메인-페이지 사이트	http://docs.oracle.com/en/database/database.html

◆ 문서 목차.

1. 데이터베이스 개요.
 2. 단순 SELECT문.
 3. SELECT문에서 WHERE절 및 ORDER BY절 사용하기.
 4. 단일-행 함수(SINGLE-ROW FUNCTION).
 5. 다중 행 함수(MULTIPLE-ROW FUNCTION, GROUP FUNCTION).
 6. 두 테이블의 행을 결합하여 데이터 조회하기(Join을 사용한 데이터 조회).
 7. 하위질의(SUBQUERY)의 활용.
 8. SET 연산자(SET OPERATOR)의 활용.
 9. 사용자 DATA 수정(DATA MANIPULATION LANGUAGE, DML).
 10. 테이블 생성(DATA DEFINITION LANGUAGE, DDL).
-
11. 테이블 이외의 오라클 데이터베이스 객체(VIEW, SEQUENCE, INDEX, SYNONYM).
 12. 오라클 데이터베이스 계정 생성 및 데이터베이스 계정에 대한 권한 설정.
 13. 스키마 객체 관리: ALTER TABLE 문장 사용.
 14. DATA DICTIONARY VIEW를 이용한 객체 정보 확인.

[부록]

[참고] SQL*Plus/SQL*Developer 툴의 COLUMN (줄여서 COL)명령어.

[참고] SQL*Plus/SQL*Developer 툴의 보고서 형식 출력기능 실습.

[참고] SQL*Plus/SQL*Developer 툴에서 치환변수(Substitution Variables)의 활용.

1 데이터베이스 개요

◆ 학습 목표.

- 웹-서버와 웹-애플리케이션-서버(WAS)의 차이점을 확인하여, 데이터베이스의 필요성을 이해합니다.
- Structural Query Language (SQL) 의 용도 및 특성에 대하여 확인합니다.
- 다음의 용어들에 대한 의미를 확인합니다.
 - 데이터(Data)
 - 정보(Information)
 - 테이블(Table)
 - 행(Row)
 - 컬럼(Column)
 - 레코드(Record)
 - 필드(Field)

1-1. 데이터베이스의 필요성: 웹 서버와 웹 어플리케이션 서버(Web Application Server, WAS)의 차이점

■ 웹-서버와 웹-어플리케이션-서버(WAS)의 차이점을 확인하면, 전산 시스템 구성에 있어서 데이터베이스가 필요한 이유를 이해할 수 있습니다.



- 웹 서버는 기본적으로 **HTML**로 작성된 웹 페이지 문서를, 사용자(CLIENT)에게 서비스하는 시스템입니다. 웹 서버를 이용하여 [회원 정보] 웹 페이지 문서를 서비스 하는 경우, 만약 사이트의 회원이 100 만 명이라면, 각 회원의 [회원 정보] 웹 페이지를 서비스 하기 위하여 100 만 개의 웹 페이지 문서를 작성해야 합니다. 또한 특정 회원이 탈퇴한 경우에는 해당 회원과 관련된 [회원 정보] 웹 페이지를 찾아서 삭제해 주어야 합니다. 즉, 회원 각각에 대하여 고유한 [회원 정보] 웹 페이지 문서를 작성 및 관리해야 합니다. 대단히 힘든 작업입니다.
- 이러한 [회원 정보] 웹 페이지를 살펴보면, 동일한 형식의 문서에 "표시되는 데이터만 차이"가 납니다. 이렇게, 동일한 형식의 문서에, 표시되는 데이터만 다른, 다수의 웹 페이지 문서들을 다음과 같은 방법을 이용하여 서비스를 구성할 경우, 웹 페이지 문서를 효율적으로 관리할 수 있습니다.
 1. 각 웹 페이지 문서에 표시되는 **데이터는 별도의 데이터베이스 서버에 저장**합니다.
 2. 웹 서버에 회원의 정보를 표시하는 형식만 정의된 한 개의 웹 페이지 문서를 작성합니다.
 3. 웹 서버에 존재하는 [회원 정보] 페이지가 각각의 사용자에 의해서 요청될 때, **특정한 기능의 프로그램**을 통하여 [웹 페이지에 표시할 데이터를 데이터베이스 서버로부터 가져와서] 해당 웹 페이지에 표시한 후, 데이터가 채워진 웹 페이지를 요청한 사용자에게 각각 전송합니다.
- 위의 세 번째 단계에서 **특정한 기능의 프로그램**은 다음의 기능이 포함됩니다.
 - (1) 데이터베이스 서비스에 접속합니다.
 - (2) SQL문을 이용하여 필요한 데이터를 가져와서 프로그램의 변수를 통해 웹 페이지에 표시합니다.
 - (3) 데이터베이스의 접속을 종료합니다.
- 위의 설명에서처럼 웹 서버 상에 형식이 정의된 웹 페이지 문서 하나를 작성 한 후, 페이지에 표시할 데이터를 [데이터베이스에 접속해서 SQL문을 통해 가져오는 프로그램]과 연동 시켜서 최종적으로 데이터가 표시된 웹 페이지를 서비스하면, 수 많은 동일한 형식의 웹 페이지를 작성할 필요가 없어집니다. 이렇게, 웹 서버에 웹 페이지와 연동된 프로그램이 같이 구성된 웹 서버를, "웹 어플리케이션 서버(WAS)" 라고 합니다.

- 웹 서비스를 구성하는 대부분의 경우에, 서비스 할 데이터를 체계적으로 관리하고 구성하기 위하여 데이터베이스 서버를 사용합니다.

- 데이터베이스 관리 시스템 (**Database Management System, DBMS**)은, 데이터를 개방적이고 포괄적이며 통합적으로 관리할 수 있는 소프트웨어입니다. 이러한 DBMS 소프트웨어 중에 하나가 오라클 사의 오라클 데이터베이스입니다.

- 오라클 DBMS 소프트웨어를 설치하고, 이를 이용하여 클라이언트의 데이터를 저장했다가 필요할 때 이를 제공해주는 서비스가 구축된 컴퓨터 시스템을 **오라클 데이터베이스 서버**라고 합니다.

1-2. Structural Query Language (SQL) 개요 및 SQL 문 분류

- SQL(구조적 질의어)은, 모든 프로그램 및 사용자가 데이터베이스의 데이터에 액세스하기 위하여 사용하는 일련의 명령문으로서, SQL을 이용하여, 다음의 작업을 포함한, 다양한 작업을 데이터베이스에 대하여 수행할 수 있습니다.

SQL 종류 및 해당 명령문	용도 설명
<input type="radio"/> Query-문 <ul style="list-style-type: none"> • SELECT문 	데이터 조회어(SELECT문)를 이용하면, 데이터베이스에서 데이터를 검색하여 사용자가 원하는 데이터를 확인 및 사용할 수 있습니다.
<input type="radio"/> DML문 <ul style="list-style-type: none"> • INSERT문 • DELETE문 • UPDATE문 • MERGE문 	데이터 조작어 (Data Manipulation Language)을 이용하면, 데이터베이스의 테이블에 새로운 행을 입력하고, 기존 행의 데이터를 변경하고, 테이블에서 불필요한 행을 제거할 수 있습니다.
<input type="radio"/> DDL-문 <ul style="list-style-type: none"> • CREATE문 • ALTER문 • DROP문 • TRUNCATE문 • COMMENT문 • RENAME문 	데이터 정의어 (Data Definition Language, DDL-문)를 이용하면, 데이터베이스에 테이블을 생성, 변경 또는 삭제하여, 새로운 데이터들(새로운 행)의 구조를 설정하고, 기존 데이터들의 구조를 변경 또는 제거할 수 있습니다. 또한 데이터베이스에서 사용되는 뷰, 시퀀스, 동의어 등의 객체를 생성하고, 기존 객체들에 설정된 내용을 변경하거나 필요 없는 객체를 삭제할 수 있습니다.
<input type="radio"/> DCL-문 <ul style="list-style-type: none"> • GRANT문 • REVOKE문 	데이터 제어어 (Data Control Language)를 이용하면, 데이터베이스에 접속하는 사용자에게, 데이터베이스와 그 안의 데이터 구조에 대한 액세스 권한을 부여 또는 제거할 수 있습니다.
<input type="radio"/> TCL-문 <ul style="list-style-type: none"> • COMMIT문 • SAVEPOINT문 • ROLLBACK문 	트랜잭션 제어어 (Transaction Control Language)를 이용하면, 데이터베이스에서 수행된 하나 이상의 DML문(들)으로 인한 변경 사항을 관리할 수 있습니다. 즉, 데이터베이스의 데이터에 대한 변경 사항(들)은 하나의 트랜잭션 단위로 논리적으로 그룹화할 수 있습니다.

- 오라클 SQL은 ANSI (American National Standards Institute) 표준 및 산업 표준을 모두 준수합니다.

1-3. 데이터베이스 기초 용어

- 데이터(Data)는 구체적으로 표현된 하나의 값을 의미합니다.

예를 들면, 아래에서 "홍길동"과 "01024567856"을 각각 "이름 데이터", "전화번호 데이터"라고 합니다.

- 정보(Information)는 사용자(Client)에게 특별한 의미를 가지는 데이터의 조합(Data-Group)을 의미합니다.

예를 들면, 아래에서 "홍길동"과 "01024567856"의 2 개의 데이터가 조합되어 "홍길동의 연락처 정보"라고 합니다.

이름 데이터	전화번호 데이터
홍길동	01024567856

→ 홍길동의 연락처 정보

- 테이블(Table)은 데이터베이스 내에 데이터들이 저장된 객체이며, 테이블을 통해 사용자는 필요할 때마다 데이터를 사용할 수 있습니다. 새로운 데이터를 저장하고, 저장된 데이터를 필요할 때마다 사용하려면, 먼저 데이터베이스에 접속하여, 테이블을 생성해야 합니다.

- 컬럼(Column)은 데이터베이스에 있는 특정 테이블에서, 동일한 의미와 동일한 속성을 가지는 값들을 의미합니다.

- 행(Row)은 특정 테이블에 정의된 모든 컬럼들의 값들로 구성된 데이터를 한 개의 묶음을 의미합니다.

- 필드(Field)는 행과 열의 교차점 또는 행에 있는 하나의 값의 위치를 의미합니다. 필드는 값이 없거나(NULL), 또는 해당되는 값이 있다면, 오직 하나의 값만 가질 수 있습니다.

- 레코드(Record)는 행에서, 하나 이상의 필드에 있는 값들의 조합을 의미합니다. 이러한 레코드가 사용자-프로그램에서 처리되는 값들의 조합으로 사용됩니다.

- 다음은 실습용 오라클 데이터베이스에 저장된 HR.DEPARTMENTS 테이블의 데이터들을 이용하여, 행, 컬럼, 필드 및 레코드를 간단히 표시한 그림입니다.

The diagram illustrates the structure of the HR.DEPARTMENTS table. It shows a grid of data with 14 rows and 4 columns. The columns are labeled: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, and LOCATION_ID. The data includes department names like Administration, Marketing, Purchasing, etc., along with their manager IDs and location IDs. A red box highlights the first three columns (DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID). A blue arrow labeled '← 행' points to the second row. A green box highlights the value '2500' in the LOCATION_ID column of the last row. A red arrow labeled '← 컬럼' points to the fourth column.

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700

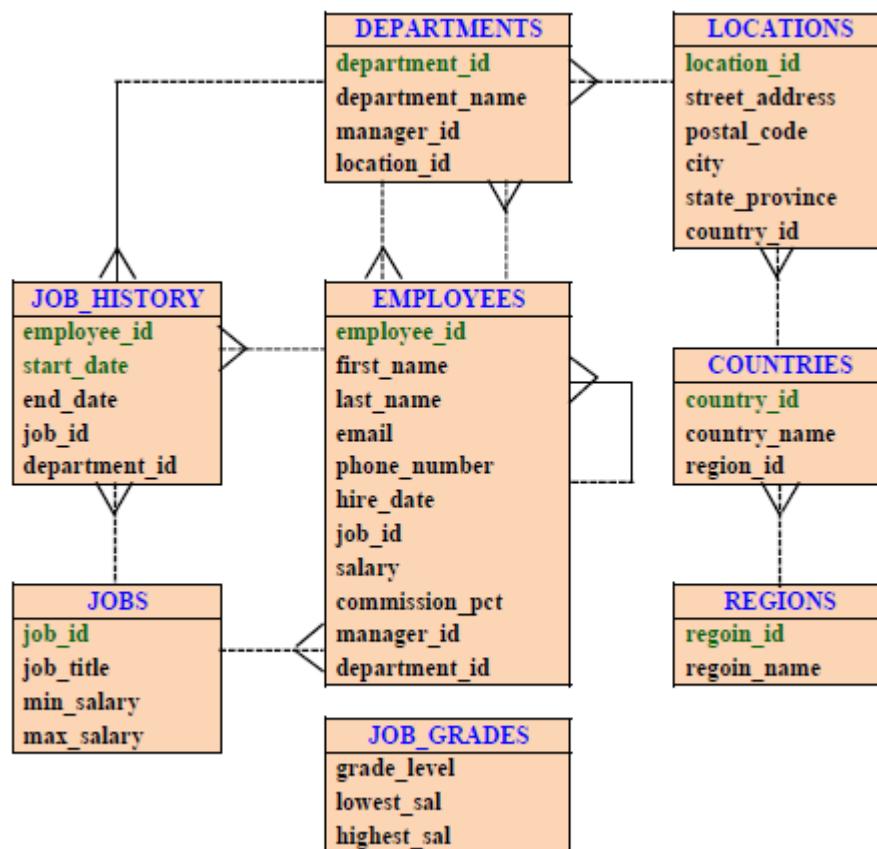
[참고] 데이터베이스 설계에 대한 간단한 이해

과거에는 사용자-프로그램이 처리하는 데이터 단위인 레코드를 근거로 테이블을 개발자가 생성했습니다. 예를 들어, 사원을 관리하기 위한 인사관리 업무가 10 개가 있다고 가정하면, 개발자는 10개의 인사관리 업무를 위한 프로그램 10 개를 제작하고, 이들 각각의 프로그램들마다 개별적인 테이블을 가지도록 데이터베이스에 10 개의 테이블을 생성했습니다.

이런 상황에서 만약, 신입사원이 새로 입사하게 되면, 10개의 인사 관리 프로그램에서 새로운 사원에 대한 인사관리 업무가 수행되도록 10 개의 테이블에 새로운 사원과 관련된 데이터를 각각 입력해야만 합니다. 이를 신속히 처리하기 위하여, 10 개의 테이블에 필요한 데이터를 자동으로 입력해주는 프로그램을 별도로 개발해야만 합니다.

이렇게, 프로그램의 데이터 처리단위인 레코드를 근거로 테이블을 생성하게 되면, 중복된 데이터가 입력되는 것을 피하기 어려울 뿐만 아니라 데이터 관리를 위한 별도의 프로그램까지 개발해야 되는 상황이 초래될 수 있습니다.

위와 같은 문제점을 때문에, 현재는, 전문가에 의하여 설계된 데이터 구조를 근거로 테이블을 생성합니다. 즉, 공통적인 목적으로 사용되는 프로그램들에서 사용되는 모든 데이터들을, 서로 관련된 데이터들을 특성에 따라 분류하여 그룹화시키고, 이 데이터들의 그룹단위로 테이블을 생성합니다. 이렇게 생성된 테이블들을, 동일한 목적의 프로그램들이 테이블에 저장된 데이터를 공유해서 사용하도록 합니다. 예를 들면, 10 개의 인사관리 업무를 위한 10 개의 인사관리 프로그램들에서 사용되는 모든 데이터들을 취합한 후, 이를 서로 관련된 데이터들끼리 분류하여, 사원관련 데이터, 부서관련 데이터 등으로 데이터들의 특성에 따라 분류 및 그룹화시키고, 이렇게 그룹화된 데이터들을, 데이터베이스에 사원테이블, 부서테이블 등으로 생성합니다. 만약, 10 개의 인사관리 프로그램들에서 사원과 관련되어 필요한 데이터들이 있다면, 인사 관리 프로그램들은, 사원 데이터들이 모두 저장된, 하나의 사원테이블로부터 필요한 데이터를 검색하여 사용하게 됩니다.

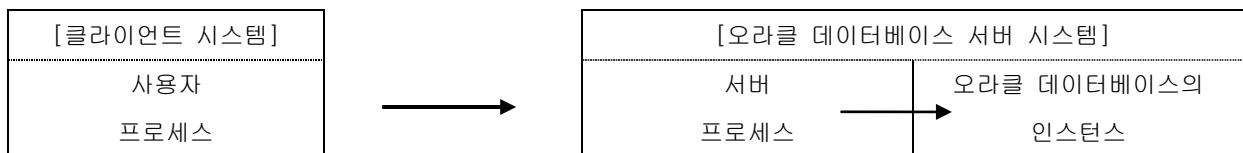


새로운 신입사원이 입사한 경우, 사원테이블에만 필요한 데이터를 입력해 놓으면, 10개의 인사관리 프로그램들이 이를 사용할 수 있게 됩니다. 이런 방법으로 데이터베이스의 테이블들을 구성하면, 중복된 데이터의 입력을 막고, 데이터 관리를 위한 별도의 프로그램을 개발할 필요도 없어지게 됩니다.

데이터베이스의 데이터 구조를 설계할 때, "개별적인 데이터가 가지는 의미"는 속성(Attribute)으로 정의되며, 서로 관계된 속성들은 사용 목적에 적합하도록 그룹화됩니다. 이렇게 "서로 관련된 그룹화된 속성들의 묶음"을 엔터티(Entity)라고 하며, 이러한 엔터티 및 엔터티들의 관계를 그림으로 표시한 것이 Entity Relationship Diagram (데이터 구조도, ERD)입니다. 위의 그림은 본 교재의 실습에 사용하는 데이터들의 의미 관계 및 의미들의 묶음을 표시한 그림이며, 이 그림을 바탕으로 최종적인 ERD가 완성됩니다. 이렇게 설계를 통해 완성된, 엔터티 및 속성들이 정의된 데이터-구조도(ERD)를 근거로 데이터베이스에 테이블을 생성하며, 이 때, 엔터티의 이름은 테이블이름으로, 또한 각 속성의 이름은 테이블을 구성하는 각 컬럼의 이름으로 구현됩니다.

[참고] 사용자프로세스(USER PROCESS), 서버프로세스(SERVER PROCESS), 및 세션 SESSION)에 대한 이해

- 사용자-프로세스(User process)는, 데이터베이스에 접속하여, SQL문을 요청하고, 결과를 받아 표시해주는 프로그램으로, 업무를 수행하기 위해 사용자(클라이언트)의 컴퓨터 또는 어플리케이션 서버에서 실행된 프로그램입니다. 예를 들면, SQL*Developer, SQL*Plus, RMAN, 또는 WAS의 웹-페이지와 연동된 프로그램 등등입니다.
- 서버-프로세스(Server process)는, 사용자-프로세스로부터 SQL문을 전달받아, SQL문을 처리하는 데이터베이스 서버의 프로세스입니다. 디풀트로 사용자-프로세스가 접속 시마다 개별적인 서버-프로세스가 생성됩니다.
- 세션(Session)은, 데이터 서비스를 받기 위하여 사용자가 허용된 데이터베이스 계정으로 데이터베이스 서버에 접속된 상태입니다. 즉, 사용자-프로세스와 연결(Connect)된 서버-프로세스가 데이터베이스 서버에 접속된 상태입니다.



- 오라클 데이터베이스의 인스턴스(Instance)는, 오라클 RDBMS에 의하여 실행되는, 데이터베이스-엔진 프로세스입니다.
- 실습환경에서 사용자 프로세스로 사용되는 프로그램으로, 오라클 사가 제공하는 SQL*Developer 그래픽 환경의 프로그램과 오라클 데이터베이스 서버 시스템에 설치된 명령줄 환경의 SQL*Plus를 사용합니다.

[참고] 명령줄 환경에서 사용되는 오라클 SQL*Plus 개발도구의 CONNECT 명령어와 SHOW USER 명령어

- 오라클 SQL*Plus의 CONNECT 명령어는, 실행 중인 오라클 SQL*Plus에서 데이터베이스에 접속할 때 사용됩니다. 또한 이미 접속된 상태에서 다른 데이터베이스 사용자 계정으로 접속할 때도 사용할 수 있습니다. 사용자 접속을 변경하면, 이전 사용자의 접속은 정상적으로 종료됩니다.
- 오라클 SQL*Plus의 SHOW USER 명령어는, 현재 오라클 데이터베이스 서버에 접속한 계정을 확인할 때 사용됩니다.

[실습] 명령줄 환경의 SQL*Plus의 CONNECT 명령어와 SHOW USER 명령어에 대한 사용 방법을 연습합니다.

```
C:\WINDOWS\system32>sqlplus /nolog — /nolog 옵션은 접속은 하지 않고, SQL*Plus 를만 실행시킵니다.
```

```
SQL*Plus: Release 11.2.0.1.0 Production on Thu Jan 1 19:56:16 2015
```

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

```
SQL> CONNECT / AS SYSDBA — CONNECT 명령어로 SYS 계정으로 접속합니다.
```

```
연결되었습니다.
```

```
SQL> SHOW USER — SHOW USER 명령어로 접속한 계정이름을 확인합니다.
```

```
USER is "SYS"
```

```
SQL> CONN hr/oracle4U — CONNECT를 간단히 CONN 으로 실행시켜 HR 계정으로 접속합니다.
```

```
연결되었습니다. — 이 때, 기존 SYS 계정의 접속은 해제됩니다.
```

```
SQL>
```

```
SQL> SHOW USER — SHOW USER 명령어로 접속한 계정이름을 확인합니다.
```

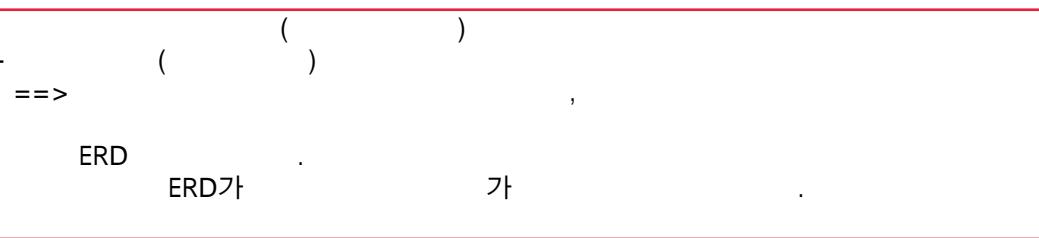
```
USER is "HR"
```

☞ 터미널에서 실행되는 오라클 SQL*Plus에서 CONNECT 명령어로 접속을 바꾸면, 이전의 접속은 해제됩니다.

☞ 단, 오라클 SQL*Developer에서는 CONNECT 명령어를 실행할 수 있지만, 실행이 완료되면, CONNECT 명령어에 의한 세션의 접속이 해제됩니다.

☞ 오라클 사의 SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어들은, 명령어 끝에 세미콜론(;)의 유무와 상관없이 정상적으로 처리됩니다.

☞ 본 교재에서는 "SQL문장"과 "SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어"를 쉽게 구분하기 위하여, SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어에 대해서는 세미콜론(;)을 명시하지 않습니다.



2 단순 SELECT문

◆ 학습 목표.

- 단순 SELECT문의 문법을 확인하고, 하나의 테이블로부터 데이터를 조회하는 기본적인 방법을 학습합니다.
- 숫자 및 날짜 데이터 컬럼에 대하여 산술연산자를 이용하여 계산된 결과를 표시하는 방법을 학습합니다.
- 테이블의 컬럼 및 레코드의 필드에 대한 NULL 상태에 대하여 이해합니다.
- 컬럼별칭(Column Alias)을 이용하여 표시결과의 머리글(Heading)을 변경하는 방법을 학습합니다.
- 리터럴 문자열을 이용하여 표시 결과에 상수를 포함시키는 방법을 학습하고, q 연산자의 기능 및 사용방법을 학습합니다.
- 연결연산자(||) 및 DISTINCT 키워드의 기능 및 사용방법을 학습합니다.

2-1. SELECT문 (쿼리) 기능

- SELECT문으로 질의하여, 데이터베이스에 저장된 데이터를, 사용자 프로그램에서 사용하기 위하여 가져올 수 있습니다. 즉, 프로그램에서 처리할 데이터를, 사용자 프로그램으로 가져오기 위하여, SELECT문을 사용합니다.

2-2. 단순 SELECT문의 기본 문법

- 단순 SELECT문의 기본 문법.

SELECT 컬럼1, 컬럼2, 컬럼-표현식	--옆에서 SELECT 키워드로 시작된 라인을 SELECT절이라고 합니다.
FROM 소유자명.테이블이름	--옆에서 FROM 키워드로 시작된 라인을 FROM절이라고 합니다.
WHERE 행을_선택할_조건	--옆에서 WHERE 키워드로 시작된 라인을 WHERE절이라고 합니다.
ORDER BY 정렬기준;	--옆에서 ORDER BY 키워드로 시작된 라인을 ORDER BY절이라고 합니다.

- 단순 SELECT문은 하나의 테이블에 대하여 질의할 때 사용하는 문장이며, 따라서, FROM 절에 테이블이름을 오직 하나만 명시합니다.
- 위에서 SELECT절에는, 원하는 데이터가 정의된 컬럼이름(들)이나 컬럼-표현식(들)을 콤마로 구분하여 명시합니다. 컬럼에 대하여 연산자나 함수 등으로 처리한 경우, 이를 "컬럼-표현식"이라고 합니다.
- 위에서 FROM절에는, SELECT절에 명시된 컬럼(들)이 정의된 테이블이름을 명시합니다. 이 때, 테이블이름 앞에 테이블을 소유한 계정이름(이를 스키마이름이라고 합니다)을 명시하는 것을 권장합니다.
- 위에서 WHERE절에는, SELECT문에 의하여 처리되는 행(들)을 선택할(SELECTION) 조건을 기술합니다.
- 위에서 ORDER BY절에는, 결과 레코드를 정렬할 기준을 명시합니다.

2-3. SQL문 작성 규칙

- SQL문에서 SQL 키워드, 컬럼이름, 테이블이름은 대소문자를 구분하지 않으며, 세미콜론(;)으로 끝납니다.
- 하나 이상의 여러 라인에 걸쳐(주로 절 단위) 하나의 문장을 작성할 수 있지만, 단어가 나눠질 수는 없습니다. 또한 문장을 쉽게 파악하도록 들여쓰기(Indent)를 활용할 수 있습니다.
- SQL*Plus 또는 SQL*Developer 툴에서 다음과 같은 방법을 이용하여 특정 내용을 주석으로 처리할 수 있습니다.
 - 라인 주석: 특정 라인에서 -- 를 명시하면 그 다음부터의 내용이 주석으로 처리됩니다.
 - 블록 주석: [/*] 기호부터 [*/] 기호 사이에 있는 [여러 줄] 또는 [라인의 일부] 내용이 주석으로 처리됩니다.

2-4. 단순 SELECT문을 이용한 대표적인 데이터 조회 실습.

[실습] HR.EMPLOYEES 테이블로부터, EMPLOYEE_ID, LAST_NAME, SALARY, EMAIL 컬럼의 데이터로 구성된 결과를 조회하시오.

```
SQL> SELECT employee_id, last_name, salary, email
      FROM hr.employees;
```

--SELECT 절에 원하는 컬럼이름을 명시하여,
--사용자가 원하는 데이터로 구성된 레코드를
--표시할 수 있습니다.

EMPLOYEE_ID	LAST_NAME	SALARY	EMAIL
198	OConnell	2600	DOCONNEL
199	Grant	2600	DGRANT
200	Whalen	4400	JWHALEN
201	Hartstein	13000	MHARTSTE
202	Fay	6000	PFAY
...			
107개의 행이 선택됨			

--WHERE 절이 누락된 경우,
--테이블의 모든 행에서 레코드가 추출됩니다.

☞ SELECT절에 사용자가 원하는 컬럼이름을 명시하여, 원하는 컬럼의 데이터를 표시할 수 있습니다.

[실습] HR.EMPLOYEES 테이블로부터, 부서 ID(DEPARTMENT_ID 컬럼)가 90인 부서에서 근무하는 사원들에 대한, 사원ID (EMPLOYEE_ID 컬럼), 사원의 이름(FIRST_NAME 컬럼), 급여(SALARY 컬럼), 급여에 0.05가 곱해진 세금, 급여에 100 이 더해진 연봉으로 구성된 레코드의 데이터를 조회하시오.

```
SQL> SELECT employee_id, first_name, salary, salary*0.05, 12*(salary+100)
      FROM hr.employees
      WHERE department_id=90;
```

--SALARY*0.05 및 12*(salary+100)을 표현식이라고 합니다.
--WHERE 절의 조건을 만족하는 행으로부터 레코드가 추출됩니다.

EMPLOYEE_ID	FIRST_NAME	SALARY	SALARY*0.05	12*(SALARY+100)
100	Steven	24000	1200	289200
101	Neena	17000	850	205200
102	Lex	17000	850	205200

--12*(salary+100)에서 괄호로 묶인 (salary+100)이 먼저 계산된 후에, 12가 곱해집니다.

☞ 컬럼의 데이터유형이 NUMBER 또는 DATE 형식일 때, 산술 연산자를 이용하여 계산된 결과를 얻을 수도 있습니다.

☞ 데이터유형에 따라 사용할 수 있는 산술연산자에 대하여 차이가 있습니다.

데이터유형	사용할 수 있는 산술연산자
NUMBER	+ (더하기), - (빼기), * (곱하기), / (나누기)
DATE	+ (더하기), - (빼기)

☞ SELECT문에 WHERE절이 포함되면, 테이블에서 WHERE절에 명시된 조건을 만족하는 행을 찾아서, SELECT절에 명시된 컬럼 및 컬럼-표현식으로 구성된 레코드가 추출되어 표시됩니다.

[실습] HR.DEPARTMENTS 테이블에 정의된 모든 컬럼의 데이터를 조회하시오.

```
SQL> SELECT * FROM hr.departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID	
10	Administration	200	1700	
20	Marketing	201	1800	
30	Purchasing	114	1700	-- WHERE 절이 누락된 경우,
40	Human Resources	203	2400	테이블의 모든 행에서
50	Shipping	121	1500	레코드가 추출됩니다.
60	IT	103	1400	
70	Public Relations	204	2700	
80	Sales	145	2500	
90	Executive	100	1700	
100	Finance	108	1700	
110	Accounting	205	1700	
...				
27 rows selected.				

☞ SELECT 절에 모든 컬럼이름 대신, * 을 명시하면, 테이블을 구성하는 모든 컬럼의 데이터가 표시됩니다.

☞ 사용자의 SQL문은, 오라클 데이터베이스 서비스에서, 테이블의 행 단위로 처리됩니다.

2-5. NULL 이란?

■ 테이블의 한 행에서, 데이터가 없는 컬럼의 경우, "이 컬럼은 NULL 이다" 라고 합니다. 즉, 컬럼이 NULL 이라는 것은 행의 컬럼에 데이터가 없는 상태를 의미합니다. 또는 아래의 예제처럼, SELECT 문의 표시된 결과 레코드에서 COMMISSION_PCT 필드에 표시된 데이터가 없을 경우. 이를 "COMMISSION_PCT 필드가 NULL 상태에 있다"라고 합니다.

[실습] HR.Employees 테이블로부터, 부서ID(DEPARTMENT_ID 컬럼)가 20인 부서에서 근무하는 사원들의, 사원ID(EMPLOYEE_ID 컬럼), 사원의 성(LAST_NAME 컬럼), 급여(SALARY 컬럼), 그리고, 급여에 대한 커미션 비율(COMMISSION_PCT 컬럼)로 구성된 레코드의 데이터를 조회하시오.

```
SQL> SELECT employee_id, last_name, salary, commission_pct
  FROM hr.employees
 WHERE department_id = 20 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	COMMISSION_PCT
201	Hartstein	13000	
202	Fay	6000	

--> COMMISSION_PCT 필드가 NULL 상태에 있다.

--> COMMISSION_PCT 필드가 NULL 상태에 있다.

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원들의 LAST_NAME, SALARY, COMMISSION_PCT 및 SALARY*(1+COMMISSION_PCT)*12 표현식으로 구성된 레코드의 데이터를 조회하시오.

```
SQL> SELECT last_name, salary, commission_pct, salary*12+salary*commission_pct*12
      FROM hr.employees ;
```

LAST_NAME	SALARY	COMMISSION_PCT	SALARY*12+Salary*Commission_Pct*12
...			
Matos	2600		
Vargas	2500		
Russell	14000	.4	235200
Partners	13500	.3	210600
Errazuriz	12000	.3	187200
...			

107개의 행이 선택됨 --WHERE 절이 누락된 경우, 테이블의 모든 행에서 레코드가 추출됩니다.

--> 표시되는 값이 없습니다.

☞ SELECT 문에서 산술 연산자가 포함된 표현식에 NULL인 컬럼이 포함되면 결과는 항상 NULL로 반환됩니다.

```
connect (conn)
edit(ed) , /
show user
describe(desc)
```

[참고] SQL*Plus 및 SQL*Developer 툴의 DESCRIBE 명령어.

■ SQL*Plus 및 SQL*Developer 툴의 DESCRIBE 명령어를 이용하여 테이블을 구성하는 컬럼이름과 데이터유형을 확인할 수 있습니다. DESCRIBE 명령어를 간단히 DESC로 줄여서 사용합니다.

[실습] DESCRIBE 명령어를 이용하여 HR.EMPLOYEES 테이블의 컬럼이름과 데이터유형을 확인하시오.

```
SQL> DESC hr.employees          --HR.EMPLOYEES 테이블에 정의된 컬럼이름과 데이터유형을 확인합니다.
      이름      널      유형
      -----
EMPLOYEE_ID    NOT NULL NUMBER(6)
FIRST_NAME           VARCHAR2(20)
LAST_NAME     NOT NULL VARCHAR2(25)   --VARCHAR2(25)는 문자 데이터유형으로서 최대 25 Bytes 길이의
EMAIL         NOT NULL VARCHAR2(25)   --문자 값을 처리할 수 있는 데이터유형입니다.
PHONE_NUMBER        VARCHAR2(20)
HIRE_DATE       NOT NULL DATE        --DATE는 날짜 데이터유형입니다.
JOB_ID         NOT NULL VARCHAR2(10)
SALARY           NUMBER(8,2)
COMMISSION_PCT        NUMBER(2,2)
MANAGER_ID        NUMBER(6)        --NUMBER(6)는 숫자(실수) 데이터유형으로서 최대 6글자 길이의
DEPARTMENT_ID        NUMBER(4)        --숫자 값을 처리할 수 있는 데이터유형입니다.
```

☞ DESCRIBE 명령어는 SQL문장이 아니고, 오라클 사의 SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어입니다.

2-6. 컬럼 별칭(Column Alias)으로 사용한 출력결과의 머리글(Heading) 변경하기.

- SELECT문에 의하여 출력된 결과에서, 레코드들 위에 표시된 각 필드의 제목을 머리글(Heading) 이라고 하고, SELECT절에 명시한 컬럼이름이나 컬럼표현식이 그대로 각 필드의 머리글(Heading)로 표시되며, 디폴트로 머리글은 대문자로 표시됩니다.
- SELECT절에서 컬럼이름이나 컬럼-표현식 다음에 아래의 방법으로 컬럼-별칭을 설정하여 SELECT문의 처리 결과에 표시되는 머리글(Heading)을 변경할 수 있습니다.

SELECT절에서 컬럼 별칭을 정의하는 방법	설명
[표준문법] 컬럼표현식 AS "원하는 별칭"	<ul style="list-style-type: none"> 큰따옴표[""]를 명시하면 둘 이상의 단어를 지정할 수 있습니다. 큰따옴표[""]를 명시하면 명[대/소]문자가 구분되어 표시됩니다.
[약식문법] 컬럼표현식 원하는_별칭	<ul style="list-style-type: none"> AS 키워드를 생략하고 간단히 빈칸을 이용합니다. 컬럼 별칭이 한 단어이면 큰따옴표[""]를 생략 할 수 있습니다. <p>☞ 큰 따옴표[""]를 생략하면 머리글이 대문자로 표시됩니다.</p>

[실습] HR.EMPLOYEES 테이블에서 모든 사원의 LAST_NAME, SALARY, SALARY*(1+COMMISSION_PCT)*12의 값들로 구성된 데이터를 조회하되, 결과의 머리글이 각각 Last_Name, SAL, ANN_SAL로 표시되도록 하시오.

```
SQL> SELECT last_name AS "Last Name", salary AS "SAL",
      salary*12+salary*commission_pct*12 ann_sal
     FROM hr.employees ;
```

Last Name	SAL	ANN_SAL
OConnell	2600	
Grant	2600	
Whalen	4400	
Hartstein	13000	
Fay	6000	
...		
107개의 행이 선택됨		

--Last Name 머리글은, 두 단어이므로 큰 따옴표로 감싸주었으며,
--따옴표로 감쌌으므로, 대소문자도 구분되어 표시되었습니다.
--ann_sal 머리글은 AS 도 생략되었고, 한 단어이므로,
--약식으로 큰 따옴표를 생략했으므로, 대문자로 표시되었습니다

☞ 위에서 SELECT 절에 명시된 ann_sal 컬럼별칭은 큰 따옴표를 생략했으므로, 대문자인 ANN_SAL로 표시됩니다.

2-7. 연결연산자인 사용.

- 연결연산자(||)는, 연산자 앞 뒤로 명시된 두 개의 문자열을 붙입니다.

[실습] HR.EMPLOYEES 테이블에서 EMPLOYEE_ID가 100 인 사원의 LAST_NAME, JOB_ID 의 값들로 구성된 데이터를 조회하되, 두 데이터를 하나의 값으로 합쳐서 표시하고, 표시 결과의 머리글이 RESULT로 표시되도록 하시오.

```
SQL> SELECT last_name || JOB_ID AS "RESULT"
   FROM hr.employees
 WHERE employee_id = 100;      --WHERE 절의 조건을 만족하는 행으로부터 레코드가 추출됩니다.

RESULT
-----
KingAD_PRES                  --||연산자 앞 뒤의 컬럼 중, 하나의 컬럼이 NULL인 경우에는,
                               --NULL이 아닌 다른 컬럼의 값이 출력됩니다.
                               --두 컬럼이 모두 NULL이면, 표시되는 값이 없습니다.
```

- 한쪽 컬럼이 NULL인 경우라도 NULL이 아닌 다른 컬럼의 값을 출력합니다.

2-8. 리터럴 문자열.

- SELECT문의 출력 결과 레코드에 같이 표시된 '문자(열)', '날짜', 숫자를 "리터럴 문자열"이라고 합니다.

- 문자(열)과 날짜 형식의 리터럴 문자열은 작은따옴표['']로 감싸 주어야 하며, 숫자 형식의 리터럴 문자열은 작은 따옴표 없이 그대로 표기합니다.

[실습] HR.EMPLOYEES 테이블에서 DEPARTMENT_ID가 90 인 사원의 LAST_NAME, JOB_ID 컬럼의 값들로 구성된 데이터를 조회하되, 두 값 사이에 'is a' 와 90 이 포함되도록 하시오.

```
SQL> SELECT last_name , 'is a' , 90 , job_id
   FROM hr.employees
 WHERE department_id = 90 ;

LAST_NAME          'IS A'        90 JOB_ID
-----
King               is a         90 AD_PRES
Kochhar            is a         90 AD_VP
De Haan            is a         90 AD_VP
```

- 위의 문장에서 'is a'와 90 이 리터럴 문자열입니다.

2-9. 대체 인용연산자 (q 연산자) - 10gNF, 오라클에서만 가능

- SELECT 절에 명시한 리터럴 문자열 안에, 작은 따옴표 같은, 처리될 수 없는 특수 문자 등이 포함된 경우, 오라클 10g 버전부터는, 대체 인용연산자(q 연산자)를 이용하여 오류 없이 처리할 수 있습니다.

- 기본문법: q 연산자 사용

```
q '[표시하고 싶은 리터럴 문자열]'      --표시하고 싶은 리터럴 문자열을 각진 괄호([ ])로 감싸주고,  
                                         --그것을 다시 작은따옴표(')로 감싸준 후, 앞에 q 를 명시합니다.
```

☞ 기본 문법에서, 각진 괄호 대신 {}, (), < >, !! 문자세트를 대신 사용할 수도 있습니다.

[실습] HR.DEPARTMENTS 테이블로부터, 부서ID(DEPARTMENT_ID 컬럼)가 10인 부서의 부서이름(DEPARTMENT_NAME 컬럼)과 부서 책임자의 사원ID(MANAGER_ID 컬럼)로 구성된 레코드를 조회하되, 두 컬럼의 값 사이에 ', it's assigned Manager id:' 리터럴 문자열이 포함되어 모두 하나의 값으로 표시되도록 하시오.
그리고 결과의 머리글이 "Department and Manager"로 표시되도록 하시오.

```
SQL> SELECT department_name          --리터럴 문자열 안에 작은 따옴표가 포함되어  
      || q'[, it's assigned Manager id:]'  --있기 때문에 q 연산자를 사용하지 않으면,  
      || manager_id                         --오류가 발생됩니다.  
      AS "Department and Manager"  
  FROM hr.departments                  --WHERE 절의 조건을 만족하는 행으로부터,  
 WHERE department_id=10 ;              --레코드가 추출됩니다.
```

Department and Manager

Administration, it's assigned Manager id:200

2-10. DISTINCT 키워드를 이용한 중복된 레코드 제거.

- SELECT 키워드 뒤에 DISTINCT 키워드를 명시하여, SELECT 절에 명시된 컬럼들로 구성된 레코드들을 표시할 때, 중복된 레코드를 "한 번만 출력"시킬 수 있습니다.
- SELECT문의 출력결과에 표시되는 레코드들은, 기본적으로 선택된 행으로부터 추출된 모든 레코드(중복된 레코드를 포함)가 표시됩니다(아래의 예제).

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID가 30인 사원의 DEPARTMENT_ID, JOB_ID 컬럼의 데이터를 조회하시오.

```
SQL> SELECT department_id, job_id FROM hr.employees WHERE department_id = 30 ;
```

```
DEPARTMENT_ID JOB_ID
```

```
-----  
30 PU_MAN  
30 PU_CLERK  
30 PU_CLERK  
30 PU_CLERK  
30 PU_CLERK  
30 PU_CLERK
```

6개의 행이 선택됨

- ☞ 위에서 DEPARTMENT_ID가 30인 조건을 만족하는 6개의 행으로부터 DEPARTMENT_ID, JOB_ID 컬럼으로 구성된 레코드가 추출되어 표시됩니다. 위에서, 5개의 레코드는 DEPARTMENT_ID가 30이고, JOB_ID가 PU_CLERK로서 동일합니다.
- ☞ 만약, 위에서 동일한 5개의 레코드를 한번만 표시하길 사용자가 원하면, 다음의 실습에서처럼 DISTINCT 키워드를 이용합니다.

[실습] HR.EMPLOYEES 테이블에서 DEPARTMENT_ID가 30인 사원에 대하여 DEPARTMENT_ID, JOB_ID 컬럼의 데이터를 조회하되, 중복된 레코드를 한 번만 표시되도록 하시오.

```
SQL> SELECT DISTINCT department_id, job_id FROM hr.employees WHERE department_id = 30 ;
```

```
DEPARTMENT_ID JOB_ID
```

```
-----  
30 PU_CLERK  
30 PU_MAN
```

- ☞ DISTINCT 키워드에 의하여, DEPARTMENT_ID, JOB_ID 컬럼으로 구성된 레코드를 표시할 때, 동일한 레코드의 경우, 한 번만 표시됩니다.
- ☞ 위의 문장에서는 DEPARTMENT_ID가 30이고, JOB_ID가 PU_CLERK인 5개의 동일한 레코드가 한번만 표시되었습니다.

[실습] SQL*Developer에서 한 번에 여러 개의 SQL문들을 실행하기.

- 다음의 실습을 통해 SQL*Developer에서 여러 SQL문들을 실행하는 방법을 학습합니다.

☞ 이 실습을 반드시 수행하여 앞으로 사용할 HR.JOB_GRADES 테이블을 생성하고 데이터를 입력합니다.

1> SQL*Developer에서 HR 계정으로 오라클 데이터베이스 서비스에 접속합니다.

2> [워크시트] 창에 아래의 SQL문들을 작성합니다.

```
CREATE TABLE HR.JOB_GRADES (
GRADE_LEVEL  VARCHAR2(2)  PRIMARY KEY,
LOWEST_SAL   NUMBER(10),
HIGHEST_SAL  NUMBER(10)  );
--HR.JOB_GRADES 테이블을 생성합니다.

INSERT INTO HR.JOB_GRADES VALUES ('A', 1000, 2999);          --6개의 행을 입력합니다.
INSERT INTO HR.JOB_GRADES VALUES ('B', 3000, 5999);
INSERT INTO HR.JOB_GRADES VALUES ('C', 6000, 9999);
INSERT INTO HR.JOB_GRADES VALUES ('D', 10000, 14999);
INSERT INTO HR.JOB_GRADES VALUES ('E', 15000, 24999);
INSERT INTO HR.JOB_GRADES VALUES ('F', 25000, 40000);
--데이터입력 작업단위를 종료합니다.

COMMIT;
```

3> 키보드의 [F5] 키를 눌러서, [워크시트]에 작성된 내용을 모두 실행시킵니다.

☞ 위에서 [워크시트]에 내용을 작성하는 대신, 원도우즈-운영체제의 [메모장]에서 위의 SQL문들을 작성하여 파일로 저장한 뒤, SQL*Developer에서 [파일열기] 메뉴를 통해 호출하여, [F5] 키로 실행시킬 수도 있습니다.

단 스크립트-파일을 [파일열기]로 호출하여 [F5]키로 실행 시에는, [접속선택] 창이 표시되며, 이 때 [확인]을 클릭하면, 기존접속을 통해 스크립트가 실행됩니다.

3 SELECT문에서 WHERE절 및 ORDER BY절 사용하기.**◆ 학습 목표.**

- WHERE 절의 기능 및 사용방법을 학습합니다.
- WHERE 절 작성 시에 사용할 수 있는 다음과 같은 연산자를 학습합니다.
 - 산술연산자
 - BETWEEN AND 연산자
 - IN () 연산자
 - LIKE 연산자 및 ESCAPE 식별자
 - 논리연산자
- ORDER BY 절의 기능 및 사용하는 방법을 학습합니다.
- ORDER BY 절에서 사용할 수 있는 NULLS FIRST, NULLS LAST 옵션에 대하여 학습합니다.

3-1. SQL문에서 WHERE절의 기능

- WHERE 절에는, 행을 선택(SELECTION)하기 위한 조건을 명시하며, 명시된 조건을 만족하는 행만 선택되어 처리됩니다.
- SELECT 문에서 WHERE 절은 FROM 절 다음에 옵니다.

■ 기본문법: 기본적인 WHERE 절 가술 방법

WHERE 컬럼이름 비교연산자 상수 --컬럼이름 대신 함수로 처리된 컬럼표현식을 기술할 수도 있습니다.

예) WHERE salary > 5000

3-2. WHERE절에서 조건을 기술할 때 상수를 명시하는 방법

- WHERE 절에서 숫자 데이터유형 컬럼과 비교되는 숫자 상수를 명시할 때는, 작은 따옴표 없이 기술합니다. 그리고, 문자 데이터유형 컬럼과 비교되는 문자 상수를 명시할 때는, 작은 따옴표(')로 감싸주어야 하며, 이 때, 문자상수가 영문자인 경우에는 [대/소]문자가 구분됩니다.

[실습] HR.EMPLOYEES 테이블로부터, LAST_NAME이 'Whalen'이고 DEPARTMENT_ID가 10인 사원의 LAST_NAME, JOB_ID, DEPARTMENT_ID 컬럼의 데이터를 조회하시오.

```
SQL> SELECT last_name, job_id, department_id FROM hr.employees
      WHERE last_name = 'Whalen'      --'Whalen' 문자상수는 작은 따옴표로 감싸주었고,
      AND department_id = 10;        --10 숫자상수는 작은 따옴표 없이 명시되었습니다.
```

LAST_NAME	JOB_ID	DEPARTMENT_ID
Whalen	AD_ASST	10

--WHERE 절에 명시된 문자상수('Whalen')와 LAST_NAME 필드에
--표시된 값의 철자 및 대소문자가 모두 일치합니다.

☞ 위의 문장에서 WHERE 절의 'Whalen' 값 대신, 모두 대문자로 바꾼 'WHALEN' 값으로 명시하면, 상수값과 컬럼의 값의 대소문자가 다르기 때문에, 아래처럼 "선택된 행 없음" 메시지가 표시됩니다.

```
SQL> SELECT last_name, job_id, department_id
      FROM hr.employees
      WHERE last_name = 'WHALEN'
      AND department_id = 10;
```

--'WHALEN' 상수값의 대소문자가 일치되는
--LAST_NAME 컬럼의 값이 없기 때문에,
--조건을 만족하는 행이 없습니다.

선택된 행 없음

- WHERE 절에서 날짜 데이터유형 컬럼과 비교되는 날짜 상수를 명시할 때는, 작은 따옴표(')로 감싸주며, 이 때, 날짜상수는 세션에서 표시되는 형식을 준수하여 명시해야만 합니다. 다음의 [참고] 내용을 확인하십시오.

[참고] 접속한 세션(CLIENT)에서의 DATE 데이터유형을 표시하는 형식.

- WHERE 절의 조건 작성 시에 [DATE 데이터유형의 컬럼]과 비교되는 상수값을 기술 시에는, 세션(즉, CLIENT)에서의 표시 형식(DISPLAY-FORMAT)을 지켜서 기술해야 합니다.

[실습] 접속한 오라클 데이터베이스 서비스에 SYSDATE 함수의 처리 결과를 요청하시오.

SQL> <code>SELECT SYSDATE FROM dual;</code>	--SYSDATE 함수는 오라클 데이터베이스 서비스가 구성된 운영체제로부터 현재의 날짜시간을 전달받아 DATE 데이터유형으로 표시합니다.
SYSDATE	--DUAL 테이블은 함수를 테스트 할 때 사용되며, 한 행을 반환합니다.

12/08/16	-- 현재 SQL*Developer 세션은 날짜가 RR/MM/DD 형식으로 출력됩니다.

- ☞ 오라클 데이터베이스 서비스에 접속한 SQL*Developer에서 위의 문장을 실행하면, 데이터베이스 서비스로부터 SYSDATE 함수의 처리 결과를, SQL*Developer가 전달받아, SQL*Developer에 설정된 DATE 데이터유형의 표시형식대로 현재 날짜 및 시간이 표시됩니다. 날짜의 표시형식은 오라클 데이터베이스에 접속한 프로그램이 실행된 지역에 따라 결정됩니다.
- ☞ 한글 Windows 운영체제에서 실행된 SQL*Developer를 이용하여 데이터베이스 서비스에 접속한 경우에는, DATE 데이터 유형의 기본 표시 형식은 RR/MM/DD 입니다. 이 때, 실행된 SELECT문의 WHERE절에서 날짜 상수를, YYYY/MM/DD, YYYYMMDD, YYYY-MM-DD, RR/MM/DD, RRMMD, 또는 RR-MM-DD의 표시형식으로 명시해도, 동일한 처리 결과를 얻을 수 있습니다. 단 SELECT문의 실행 결과에 표시되는 표시 형식은 RR/MM/DD 형식으로 표시됩니다.

[실습] HR.EMPLOYEES 테이블로부터, 입사일(HIRE_DATE 컬럼)이 2008년1월1일 이 후인 사원들의 EMPLOYEE_ID, LAST_NAME, HIRE_DATE 컬럼으로 구성된 데이터를 조회하시오.

SQL> <code>SELECT employee_id, last_name, hire_date FROM hr.employees</code>	--WHERE 절의 날짜 상수를 접속한 세션의 날짜 표시형식인
<code>WHERE hire_date >= '08/01/01';</code>	--RR/MM/DD 형식과 동일하게 명시하였으므로,
EMPLOYEE_ID LAST_NAME	--정상적으로 처리된 결과가 표시됩니다.

199 Grant	--HIRE_DATE의 날짜 값이 RR/MM/DD 형식으로 표시됩니다.
128 Markle	
136 Philtanker	
149 Zlotkey	
164 Marvins	
...	
11개의 행이 선택됨	

- ☞ WHERE 절에서 세션의 날짜 표시 형식과 다른 형식으로 날짜 형식의 상수 값을 기술한 경우에는 SELECT문이 정상적으로 처리되지 못하고 오류가 발생되거나 또는 잘못된 처리가 수행될 수 있습니다.

3-3. WHERE 절 작성 시에 사용되는 연산자 종류 및 기능.

■ WHERE절 작성 시에 다음과 같은 연산자들을 사용하여, 조건절을 작성할 수 있습니다.

- 단순 비교 연산자
- BETWEEN AND 연산자
- IN() 연산자
- LIKE 연산자
- 논리 연산자(AND/OR/NOT)

3-3-1. 단순 비교 연산자.

■ 아래의 연산자를 **단순 비교 연산자**라고 부르며, 컬럼의 값과 기술한 상수 값이 연산자의 기능에 의하여 비교됩니다.

연산자	연산자 기능	연산자	연산자 기능
=	같다	<> , != , ^=	같지 않다
>	크다	>=	크거나 같다
<	작다	<=	작거나 같다

☞ WHERE 절에서 행을 선택할 조건을 작성할 때, 일반적으로 [컬럼 연산자 상수값] 형식으로 적습니다. 위의 표에서 연산자 왼쪽에 있는 컬럼의 값이 연산자 오른쪽에 명시하는 상수값과 연산자에 의해서 비교되는 것으로 설명합니다.

[실습] HR.Employees 테이블로부터, SALARY 값이 3000보다 작거나 같은 모든 사원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary FROM hr.employees WHERE salary <= 3000;
```

LAST_NAME	SALARY
OConnell	2600
Grant	2600
Baida	2900
Tobias	2800
Himuro	2600
...	

26개의 행이 선택됨 --총 107개의 행 중, WHERE 절을 만족하는 26개 행으로부터 추출된 레코드가 표시되었습니다.

3-3-2. BETWEEN 값1 AND 값2 연산자.

■ 연산자 의미	컬럼의 값이 [값1] 보다 크거나 같고 [값2] 보다 작거나 같은지 비교합니다.
■ 사용 시 주의 사항	[값1], [값2]의 형식이 동일해야 하며, [값1]이 [값2] 보다 크면 안됩니다.

☞ 문자의 경우에는 국어사전에 명시된 순서대로, 뒤에 나올수록 큽니다. 영어의 경우, 소문자가 대문자 보다 큽니다.
날짜의 경우에는 최근 날짜 일수록 크며, 과거 날짜 일수록 작습니다.

[실습] HR.EMPLOYEES 테이블로부터 SALARY 값이 4100과 4500 사이에 있는 모든 사원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary FROM hr.employees WHERE salary BETWEEN 4100 AND 4500 ;
```

LAST_NAME	SALARY
Whalen	4400
Lorentz	4200
Sarchand	4200
Bull	4100

☞ WHERE 절에서 BETWEEN AND 연산자를 사용할 때는 AND의 앞에 명시한 값이 AND의 뒤에 명시한 값보다 크면 안됩니다
만약, AND의 앞에 명시한 값이 AND의 뒤에 명시한 값보다 크면, '선택된 행 없음' 메시지가 표시됩니다.

3-3-3. IN (값1, 값2, ...)연산자.

■ 연산자 의미	컬럼의 값이 명시된 값을 중 하나와 같은지 비교합니다.
■ 사용 시 주의 사항	IN 연산자에 명시된 모든 값의 데이터유형이 동일해야 합니다.

[실습] HR.EMPLOYEES 테이블로부터, MANAGER_ID 컬럼의 값이 100 또는 101 또는 201 인 모든 사원의 EMPLOYEE_ID, LAST_NAME, SALARY, MANAGER_ID 컬럼의 데이터를 표시하시오.

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

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
201	Hartstein	13000	100
101	Kochhar	17000	100
102	De Haan	17000	100
114	Raphaely	11000	100
...			

20개의 행이 선택됨

3-3-4. LIKE 연산자.

■ 연산자 의미	문자 데이터유형의 컬럼의 값이 명시된 문자열의 패턴과 동일한지 비교합니다. (예를 들면, A로 시작하는 또는 z로 끝나는 또는 oracle이 포함된 등등)
■ 사용 방법	LIKE 연산자를 통해 비교되는 상수를 명시할 때, 퍼센트 문자(%) 또는 언더스코어 문자(_)를 포함시켜 작성되며, 이 두 문자의 기능은 다음과 같습니다. ○ 퍼센트 문자(%)는 명시된 위치에 0 개 이상의 여러 문자가 존재할 수 있습니다. ○ 언더스코어 문자(_)는 명시된 위치에 반드시 1개의 문자가 존재해야 합니다.

[실습] HR.EMPLOYEES 테이블로부터, LAST_NAME 값이 'A'로 시작하는 모든 사원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary FROM hr.employees WHERE last_name LIKE 'A%' ;
```

LAST_NAME	SALARY
Abel	11000
Ande	6400
Atkinson	2800
Austin	4800

[실습] HR.EMPLOYEES 테이블로부터, LAST_NAME의 2번째 글자가 'b'이고 최소한 3글자 이상인 모든 사원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오

```
SQL> SELECT last_name, salary FROM hr.employees WHERE last_name LIKE '_b%' ;
```

LAST_NAME	SALARY
Abel	11000

[실습] LIKE 연산자 사용 예제, ESCAPE 식별자 사용 실습

- 테이블의 문자 데이터유형 컬럼의 값에 퍼센트 문자(%) 또는 언더스코어 문자(_)가 포함된 경우, LIKE 연산자를 통해 비교하는 상수를 적을 때, 값에 있는 퍼센트 문자(%) 또는 언더스코어 문자(_)를 식별하기 위하여 ESCAPE 식별자를 지정하여 사용합니다.

1> 테스트를 위한 HR.TEST01 테이블을 아래의 SQL문을 실행하여 생성합니다.

```
SQL> CREATE TABLE HR.TEST01( J_CODE VARCHAR2(30) ) ;
```

table HR.TEST01(가) 생성되었습니다.

2> 테스트용 데이터를 입력하고 트랜잭션을 커밋 합니다.

```
SQL> INSERT INTO HR.TEST01 VALUES ('ADPRES');
```

1개 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO HR.TEST01 VALUES ('AD__PRES'); --입력하는 값에 언더스코어 문자(_)가 2번 들어갑니다
```

1개 행 이(가) 삽입되었습니다.

```
SQL> COMMIT; --COMMIT 문으로, 변경 후 데이터 상태를 유지하면서, 데이터 변경 작업 단위를 끝냅니다.
```

커밋되었습니다.

3> J_CODE 컬럼의 데이터가 'AD__'로 시작하는 행만 표시하기 위하여 아래의 SELECT문을 작성하여 실행합니다.

```
SQL> SELECT *
  FROM  HR.TEST01
 WHERE J_CODE LIKE 'AD_%';
--상수 값에 명시된 _를, 데이터에 포함된 _ 문자가 아니라, LIKE 연산자와
--같이 사용되는 한 글자 치환 문자로 처리하기 때문에, ADPRES 를
--포함하여 모든 데이터가 결과에 표시됩니다.
```

J_CODE
ADPRES
AD__PRES

4> ESCAPE 식별자를 사용하여 J_CODE 컬럼의 데이터가 'AD__'로 시작하는 행만 표시되도록 합니다.

```
SQL> SELECT *
  FROM  HR.TEST01
 WHERE J_CODE LIKE 'AD\\_\\_%' ESCAPE '\\';
--ESCAPE 식별자로 지정한 문자인 \ 다음에 있는 _ 또는 % 문자는 값에 포함된
--문자로 처리됩니다.
```

J_CODE
AD__PRES

☞ LIKE 연산자 다음에 명시한 패턴에서, ESCAPE 식별자로 지정한 문자의 다음에 있는 _ 또는 % 문자는 값에 포함된 문자로 처리됩니다.

5> 실습용 테이블을 삭제합니다.

DROP TABLE

```
SQL> DROP TABLE HR.TEST01 PURGE;
```

table HR.TEST01이(가) 삭제되었습니다.

3-3-5. WHERE 절에서 IS NULL 사용.

- 오라클에서 NULL은 상태를 의미하기 때문에, WHERE 절에서 [컬럼=NULL]로 기술할 수 없습니다.
- 컬럼이 NULL인 상태에 있다는 것을 WHERE 절에 기술할 때는 [WHERE 컬럼 IS NULL]을 사용합니다.

[실습] HR.EMPLOYEES 테이블로부터, MANAGER_ID 컬럼이 NULL인 모든 사원의 LAST_NAME 데이터를 표시하시오.

```
SQL> SELECT last_name FROM hr.employees WHERE manager_id IS NULL ;
```

LAST_NAME

King

☞ 만약 WHERE 절에 [컬럼=NULL]로 기술하면, 오라클 RDBMS에서의 결과는, "선택된 행 없음" 메시지가 반환됩니다.

```
SQL> SELECT last_name FROM hr.employees WHERE manager_id = NULL ;
```

선택된 행 없음

3-3-6. AND 또는 OR 논리 연산자.

- WHERE 절에 둘 이상의 조건절을 AND 및 OR 연산자를 이용하여 다음처럼 평가하도록 기술할 수 있습니다.

기술 방법	의미
WHERE [조건1] AND [조건2]	[조건1] 과 [조건2]를 동시에 모두 만족하는 행만 처리합니다.
WHERE [조건1] OR [조건2]	[조건1] 또는 [조건2] 중 최소 하나의 조건을 만족하는 행만 처리합니다.

[실습] HR.EMPLOYEES 테이블로부터, MANAGER_ID 컬럼이 NULL 이거나 또는 SALARY가 10000 보다 큰 모든 사원의 LAST_NAME 및 SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary FROM hr.employees WHERE manager_id IS NULL OR salary > 10000;
```

LAST_NAME	SALARY
-----	-----
Hartstein	13000
Higgins	12008
King	24000
Kochhar	17000
De Haan	17000
...	

15개의 행이 선택됨

[실습] HR.EMPLOYEES 테이블로부터, MANAGER_ID 컬럼이 NULL 이고, SALARY가 10000보다 큰 모든 사원의 LAST_NAME 및 SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary FROM hr.employees WHERE manager_id IS NULL AND salary > 10000;
```

LAST_NAME	SALARY
King	24000

3-3-7. NOT 논리 연산자.

■ WHERE 절에서 NOT 연산자를 사용하여, 기술된 조건을 부정하도록 기술할 수 있습니다.

WHERE절에 NOT 연산자 기술 방법	사용 예
조건 자체를 부정하는 방법	WHERE NOT (manager_id IS NULL)
부정의 의미의 연산자를 사용하는 방법	WHERE manager_id IS NOT NULL

■ 연산자 및 해당 연산자에 대한 부정의 의미의 연산자

연산자	부정의 의미의 연산자	연산자	부정의 의미의 연산자
IN	NOT IN	BETWEEN AND	NOT BETWEEN AND
LIKE	NOT LIKE	IS NULL	IS NOT NULL
=	<> 또는 != 또는 ^=	>=	<
>	<=	<	>=

■ [NOT 연산자]는 아래의 원편에서처럼 조건절 앞에 명시하여 [조건절 자체를 부정] 하는 것보다는,
아래의 오른편에서처럼 연산자 앞에 NOT 연산자를 명시하여 [부정의 의미의 연산자]를 사용하는 방법으로 사용됩니다.

[실습] HR.EMPLOYEES 테이블로부터, MANAGER_ID 컬럼이 NULL이 아닌 모든 사원의 LAST_NAME 데이터를 표시하시오.

방법1: 조건절 자체를 부정하는 방법.	방법2: [부정의 의미의 연산자]를 사용하는 방법.
<pre>SQL> SELECT last_name FROM hr.employees WHERE NOT(manager_id IS NULL);</pre> <p>LAST_NAME</p> <p>Abel Ande Atkinson Austin Baer Baida ...</p> <p>106개의 행이 선택됨</p>	<pre>SQL> SELECT last_name FROM hr.employees WHERE manager_id IS NOT NULL;</pre> <p>LAST_NAME</p> <p>Abel Ande Atkinson Austin Baer Baida ...</p> <p>106개의 행이 선택됨</p>

3-4. ORDER BY 절을 이용한 결과 정렬.

■ ORDER BY 절은, 결과 레코드를 정렬해서 표시하고자 할 때 사용됩니다.

■ ORDER BY 절은, SELECT 문의 제일 마지막에 명시해야만 합니다.

■ ORDER BY 절에 레코드의 정렬기준을 명시할 때, 다음의 방법을 사용할 수 있습니다.

방법	설명
ORDER BY 컬럼이름	FROM 절에 명시된 테이블의 컬럼이름을 모두 명시 수 있지만, 결과의 해석이 곤란하므로, SELECT 절에 명시된 컬럼의 이름이 대부분 사용됩니다.
ORDER BY 컬럼별칭	SELECT 절에 명시된 표현식에 대한 컬럼별칭을 명시할 수 있습니다.
ORDER BY 위치에_따른_숫자	SELECT 절에 명시된 컬럼의 순서에 따라 1,2,3 의 숫자를 적을 수 있습니다.

■ ORDER BY 절에 ASC 또는 DESC 키워드를 같이 명시하여 정렬방식을 지정할 수 있습니다.

○ ASC를 명시하면, 작은 것부터 큰 것 순서로 정렬됩니다(오름차순).

○ DESC를 명시하면, 큰 것부터 작은 것 순서로 정렬됩니다(내림차순).

○ ASC 또는 DESC를 어느 것도 명시하지 않으면, 디폴트로 ASC 기준(오름차순)으로 정렬됩니다.

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 60 인 사원의 EMPLOYEE_ID, LAST_NAME, SALARY를 표시하시오.
단, SALARY 컬럼의 값을 기준으로 레코드가 오름차순으로 정렬되도록 하시오.

```
SQL> SELECT employee_id, last_name, salary FROM hr.employees WHERE department_id = 60
      ORDER BY salary ASC;
```

EMPLOYEE_ID	LAST_NAME	SALARY	---SELECT 절의 SALARY 컬럼을 기준으로 ---오름차순으로 정렬되어 출력결과가 표시됩니다.
107	Lorentz	4200	
105	Austin	4800	
106	Pataballa	4800	
104	Ernst	6000	
103	Hunold	9000	

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 60 인 사원의 EMPLOYEE_ID, LAST_NAME, SALARY를 표시하시오.
단, SALARY 컬럼의 값을 기준으로 레코드가 내림차순으로 정렬되도록 하시오.

```
SQL> SELECT employee_id, last_name, salary FROM hr.employees WHERE department_id = 60
      ORDER BY 3 DESC;          ---3은 SELECT 절의 3 번째 있는 SALARY 컬럼을 의미합니다.
```

EMPLOYEE_ID	LAST_NAME	SALARY	---SELECT 절의 3 번째 있는 SALARY 컬럼을 기준으로 ---내림차순으로 정렬되어 출력결과가 표시됩니다.
103	Hunold	9000	
104	Ernst	6000	
107	Lorentz	5460	
106	Pataballa	4800	
105	Austin	4800	

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 60 인 사원의 EMPLOYEE_ID, LAST_NAME, SALARY를 표시하시오.
단, SALARY 컬럼의 값을 기준으로 레코드가 내림차순으로 정렬되고, 같은 SALARY 값일 경우에는, EMPLOYEE_ID 컬럼의 값을 기준으로 내림차순으로 정렬되도록 하시오.

```
SQL> SELECT employee_id, last_name, salary FROM hr.employees WHERE department_id = 60
      ORDER BY salary DESC, employee_id DESC;
```

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
106	Pataballa	4800
105	Austin	4800
107	Lorentz	4200

--먼저, SALARY를 기준으로 내림차순으로 레코드를 정렬합니다.
--그 다음에 SALARY의 값이 동일한 레코드에 대하여
--EMPLOYEE_ID를 기준으로 내림차순으로 정렬합니다..

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 60 인 사원의 EMPLOYEE_ID, LAST_NAME, SALARY를 표시하시오.
단, SALARY 컬럼의 값을 기준으로 레코드가 오름차순으로 정렬되고, 같은 SALARY 값일 경우에는, EMPLOYEE_ID 컬럼의 값을 기준으로 내림차순으로 정렬되도록 하시오.

```
SQL> SELECT employee_id, last_name, salary FROM hr.employees WHERE department_id = 60
      ORDER BY salary, employee_id DESC; —SALARY 컬럼에 대한 ASC 옵션은 생략되었습니다.
```

EMPLOYEE_ID	LAST_NAME	SALARY
106	Pataballa	4800
105	Austin	4800
107	Lorentz	5460
104	Ernst	6000
103	Hunold	9000

--먼저, SALARY를 기준으로, 오름차순으로 레코드를 정렬합니다.
--그 다음에 SALARY의 값이 동일한 레코드에 대하여,
--EMPLOYEE_ID를 기준으로 내림차순으로 정렬합니다.

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 60 인 사원의 EMPLOYEE_ID, LAST_NAME, SALARY를 표시하시오.
단, 표시된 결과가 SALARY*12 표현식의 HEADING은 ANNSAL로 표시되어야 하며, SALARY*12 표현식의 값을 기준으로 레코드가 오름차순으로 정렬되어 표시되도록 하시오.

```
SQL> SELECT employee_id, last_name, salary*12 ANNSAL FROM hr.employees WHERE department_id=60
      ORDER BY annsal ASC; —SELECT 절에 선언된 컬럼별칭을 ORDER BY 절에서 사용했습니다.
```

EMPLOYEE_ID	LAST_NAME	ANNSAL
107	Lorentz	50400
105	Austin	57600
106	Pataballa	57600
104	Ernst	72000
103	Hunold	108000

-- ANNSAL을 기준으로, 오름차순으로 레코드가 정렬되었습니다.

[참고] WHERE 절에서는, 조건 작성 시에, 컬럼별칭을 사용하면, 오류가 발생됩니다.

- WHERE 절에서는 SELECT 절에 선언된 컬럼별칭을 사용할 수 없습니다.

☞ WHERE 절에 컬럼별칭을 사용할 경우, 다음과 같은 오류가 발생됩니다.

```
SQL> SELECT employee_id, last_name, salary*12 AS ANN_SAL
   FROM hr.employees
  WHERE ANN_SAL > 150000;      --WHERE 절에서는 SELECT 절에 선언된 컬럼별칭을 사용할 수 없습니다.

ORA-00904: "ANN_SAL": 부적합한 식별자
00904. 00000 - "%s: invalid identifier"
*Cause:
*Action:
3행, 7열에서 오류 발생
```

[참고] 정렬기준 컬럼이 NULL인 레코드의 정렬

- NULL이 포함된 컬럼을 정렬의 기준으로 사용하는 경우, 정렬 기준 컬럼이 NULL인 레코드들은, 오름차순으로 정렬되면 제일 마지막에, 내림차순으로 정렬되면 제일 처음에 각각 표시됩니다.

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID를 기준으로, 오름차순으로 표시하는 SELECT문과, 내림차순으로 표시하는 SELECT문을 각각 작성하시오.

```
SQL> SELECT employee_id, department_id
   FROM hr.employees
  ORDER BY 2 ASC; --2 번째 컬럼 기준, 오름차순

EMPLOYEE_ID DEPARTMENT_ID
----- -----
...
113          100
109          100
206          110
205          110
178          110
...           110
107개의 행이 선택됨
```

```
SQL> SELECT employee_id, department_id
   FROM hr.employees
  ORDER BY 2 DESC; --2 번째 컬럼 기준, 내림차순

EMPLOYEE_ID DEPARTMENT_ID
----- -----
178          110
205          110
206          110
112          100
109          100
...
107개의 행이 선택됨
```

--정렬기준 컬럼이 NULL인
--레코드가 제일 처음에
--표시됩니다.

[참고] ORDER BY 절에 NULLS FIRST 및 NULLS LAST 옵션 사용 실습

- ORDER BY 절에서 NULL이 포함된 컬럼을 정렬기준으로 사용하는 경우, **NULLS FIRST** 옵션이 명시되면, NULL 이 포함된 레코드들이 제일 먼저 표시됩니다.
- ORDER BY 절에서 NULL이 포함된 컬럼을 정렬기준으로 사용하는 경우, **NULLS LAST** 옵션이 명시되면, NULL 이 포함된 레코드들이 맨 마지막에 표시됩니다.

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID 를 기준으로, 오름차순으로 표시하는 SELECT문과, 내림차순으로 표시하는 SELECT문을 각각 작성하시오.
단, DEPARTMENT_ID 필드가 NULL인 레코드를 제일 처음에 표시하시오.

<pre>SQL> SELECT employee_id, department_id FROM hr.employees ORDER BY 2 ASC NULLS FIRST;</pre> <p>EMPLOYEE_ID DEPARTMENT_ID --오름차순</p> <hr/> <table border="1"> <tr><td>178</td><td>--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.</td></tr> <tr><td>200</td><td>10</td></tr> <tr><td>201</td><td>20</td></tr> <tr><td>202</td><td>20</td></tr> <tr><td>...</td><td></td></tr> </table> <p>107개의 행이 선택됨</p>	178	--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.	200	10	201	20	202	20	...		<pre>SQL> SELECT employee_id, department_id FROM hr.employees ORDER BY 2 DESC NULLS FIRST;</pre> <p>EMPLOYEE_ID DEPARTMENT_ID --내림차순</p> <hr/> <table border="1"> <tr><td>178</td><td>--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.</td></tr> <tr><td>205</td><td>110</td></tr> <tr><td>206</td><td>110</td></tr> <tr><td>112</td><td>100</td></tr> <tr><td>...</td><td></td></tr> </table> <p>107개의 행이 선택됨</p>	178	--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.	205	110	206	110	112	100	...	
178	--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.																				
200	10																				
201	20																				
202	20																				
...																					
178	--정렬기준 컬럼이 NULL인 레코드가 제일 처음에 표시됩니다.																				
205	110																				
206	110																				
112	100																				
...																					

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID를 기준으로, 오름차순으로 표시하는 SELECT문과, 내림차순으로 표시하는 SELECT문을 각각 작성하시오.
단, DEPARTMENT_ID 필드가 NULL인 레코드를 맨 마지막에 표시하시오.

<pre>SQL> SELECT employee_id, department_id FROM hr.employees ORDER BY department_id ASC NULLS LAST;</pre> <p>EMPLOYEE_ID DEPARTMENT_ID -- 오름차순</p> <hr/> <table border="1"> <tr><td>109</td><td>100</td></tr> <tr><td>206</td><td>110</td></tr> <tr><td>205</td><td>110</td></tr> <tr><td>178</td><td>--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.</td></tr> <tr><td>...</td><td></td></tr> </table> <p>107개의 행이 선택됨</p>	109	100	206	110	205	110	178	--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.	...		<pre>SQL> SELECT employee_id, department_id FROM hr.employees ORDER BY department_id DESC NULLS LAST;</pre> <p>EMPLOYEE_ID DEPARTMENT_ID -- 내림차순</p> <hr/> <table border="1"> <tr><td>202</td><td>20</td></tr> <tr><td>201</td><td>20</td></tr> <tr><td>200</td><td>10</td></tr> <tr><td>178</td><td>--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.</td></tr> <tr><td>...</td><td></td></tr> </table> <p>107개의 행이 선택됨</p>	202	20	201	20	200	10	178	--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.	...	
109	100																				
206	110																				
205	110																				
178	--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.																				
...																					
202	20																				
201	20																				
200	10																				
178	--정렬기준 컬럼이 NULL인 레코드가 제일 마지막에 표시됩니다.																				
...																					

4 단일-행 함수(SINGLE-ROW FUNCTION).

◆ 학습 목표.

- SQL문에서 사용할 수 있는 다음과 같은, 대표적인 오라클 단일 행 함수에 대하여 학습합니다.

기능 또는 분류	함수이름
대소문자 변환 함수	UPPER(), LOWER(), INITCAP()
문자열 수정 함수	CONCAT(), SUBSTR(), REPLACE(), TRIM(), LPAD(), RPAD()
문자열 길이 확인 함수	LENGTH(), LENGTHB()
문자열 검사 함수	INSTR()
숫자 데이터 처리 함수	MOD(), ROUND(), TRUNC()
날짜 데이터 처리 함수	SYSDATE, CURRENT_DATE, ADD_MONTHS(), MONTHS_BETWEEN(), NEXT_DAY(), LAST_DAY(), ROUND(), TRUNC()
날짜시간 데이터 처리 함수	SYSTIMESTAMP, CURRENT_TIMESTAMP, LOCALTIMESTAMP
데이터유형 변환 함수	TO_DATE(), TO_CHAR(), TO_NUMBER()
일반 함수	NVL(), DECODE(), NVL2(), NULLIF(), COALESCE()

- CASE 표현식에 대하여 학습합니다.

- ☞ 함수들은 데이터베이스 제품에 따라 함수의 이름 및 사용법이 차이가 날 수 있습니다. 즉, 오라클 제품의 함수가 다른 회사의 데이터베이스 제품에는 다른 이름으로 제공될 수도 있습니다.
- ☞ 본 단원에서는 오라클 데이터베이스 서비스에서 제공하는 대표적인 단일 행 함수들을 설명합니다. 교재에서 설명하지 않은 단일 행 함수들에 대해서는 Oracle SQL Reference 매뉴얼을 참고하십시오.

4-1. SQL 함수 개요

- 함수는, 하나 이상의 값을 처리하여, 처리된 결과값을 반환합니다.
- 함수가 처리하는 값은 다음 중 하나가 될 수 있습니다.
 - 상수값
 - 컬럼이름 또는 표현식
 - 변수
- 다음과 같은 2 가지 유형의 함수가 있습니다.
 - 단일 행 함수(Single Row Function)는, 반환되는 행마다 한 번씩 실행되어, 해당 하나의 결과를 반환합니다.
 - 다중 행 함수(Multiple Row Function)는, 반환되는 행 집합마다 한 번씩 실행되어, 행 집합당 하나의 결과를 반환합니다. 다중 행 함수를 그룹함수라고도 합니다.

☞ 본 단원에서는 단일 행 함수를 설명하며, 다중 행 함수에 대해서는, 다음 단원(단원 5)에서 설명합니다.

[실습] 다음의 실습을 통해, 단일 행 함수와 다중 행 함수의 처리 방법의 차이를 확인합니다.

1> HR.EMPLOYEES 테이블에서, DEPARTMENT_ID 컬럼이 90인 모든 사원의 저장된 성(LAST_NAME 컬럼) 및, 대문자로 표시되는 이름으로 구성된 레코드의 데이터를 표시하시오. 단, 표시되는 이름의 머리글은 FIRST_NAME으로 표시되도록 하시오.

```
SQL> SELECT last_name, UPPER(first_name) FIRST_NAME --UPPER() 함수는 영어 문자열을 대문자로 변환하는
   FROM hr.employees                                --단일 행 함수입니다.
   WHERE department_id = 90;

LAST_NAME      FIRST_NAME
-----
King           STEVEN
Kochhar        NEENA
De Haan        LEX
--UPPER() 함수는 WHERE 절의 조건을 만족하는 행이 반환될 때마다,
--한 번씩 실행되어, 각 해당 하나의 결과가 반환됩니다.
--3개의 행이 반환되었으므로, 총 3번이 실행되었습니다.
```

2> HR.EMPLOYEES 테이블에서, DEPARTMENT_ID 컬럼이 90인 모든 사원의 급여 합계를 표시하시오.

```
SQL> SELECT SUM(salary)                            --SUM() 함수는 합계를 구하는 다중 행 함수입니다.
   FROM hr.employees
   WHERE department_id = 90;
--SUM() 함수는 WHERE 절의 조건을 만족하는 모든 행 집합에 대하여
--한 번 실행되어, 하나의 결과가 반환됩니다.

SUM(SALARY)
-----
58000
--반환되는 3개 행의 SALARY 값을 모두 구한 후, 한 번 실행되어,
--합산된 하나의 결과가 반환되었습니다
```

4-2. 단일 행 함수 분류.

- 함수가 처리하는 값의 데이터유형(DATA-TYPE)에 따라 다음처럼 분류됩니다.

함수의 분류	특징
문자 함수(CHARACTER FUNCTION)	처리되는 데이터가 문자-유형 데이터입니다.
숫자 함수(NUMBER FUNCTION)	처리되는 데이터가 숫자-유형 데이터입니다.
날짜 함수(DATE FUNCTION)	처리되는 데이터가 날짜-유형 데이터입니다.
일반 함수(GENERAL FUNCTION))	처리되는 데이터가 데이터-유형에 대한 제한이 없습니다.
형-변환 함수(CONVERSION FUNCTION)	처리되는 데이터가 SQL문 처리 중에 다른 데이터유형으로 변경할 필요가 있을 때 사용됩니다. <ul style="list-style-type: none"> ○ [숫자, 날짜] 데이터를 문자 데이터로 변환합니다. ○ [문자] 데이터를 날짜 데이터로 변환합니다. ○ [문자] 데이터를 숫자 데이터로 변환합니다.

[참고] 오라클 DUAL 테이블

- 오라클 데이터베이스 서비스에 접속하는 모든 계정이 사용할 수 있습니다.
- DUAL 테이블에 대하여 SELECT 문을 실행시키면, 항상 하나의 레코드가 반환됩니다.
- 함수의 테스트를 위해 주로 사용됩니다.

4-3. 문자 함수-1: 대/소문자 변환 함수.

- 대소문자 변환 함수에는 `UPPER('문자열')` 함수, `LOWER('문자열')` 함수, `INITCAP('문자열')` 함수가 있으며, 명시된 문자열 또는 테이블의 문자열 컬럼의 값을 각각 [대문자]/[소문자]/[첫 글자 대문자 나머지 소문자]로 변환합니다.

[실습] 'abcd efg high' 문자열 값을 모두 대문자, 모두 소문자, 단어마다 첫-글자 대문자 나머지는 소문자로 변환하여 각각 표시하는 하나의 SELECT 문을 작성하시오.

```
SQL> SELECT UPPER('abcd efg high') "UPPER",      -- 모든 영어문자들을 대문자로 변환
      LOWER('abcd efg high') "LOWER",      -- 모든 영어문자들을 소문자로 변환
      INITCAP('abcd efg high') "InitCap"  -- 영어 단어마다 첫 글자 대문자 나머지 소문자로 변환
   FROM dual;
```

UPPER	LOWER	InitCap
ABCD EFG HIGH	abcd efg high	Abcd Efg High

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID 컬럼이 20인 모든 사원에 대하여, 성(LAST_NAME)은 모두 대문자, 직책코드(JOB_ID)는 모두 소문자로 표시하되, 표시결과가 The job id for 성 is 직책코드 형태로 표시되도록 하시오. 표시결과 상의 머리글은 EMP_DETAILS로 표시하시오.

```
SQL> SELECT 'The job id for '||UPPER(last_name)||' is '||LOWER(job_id) AS "EMP_DETAILS"
  FROM hr.employees
 WHERE department_id = 20;

EMP_DETAILS
-----
The job id for HARTSTEIN is mk_man
The job id for FAY is mk_rep
```

[실습] HR.EMPLOYEES 테이블로부터, 성(LAST_NAME)이 Higgins 인 사원의, EMPLOYEE_ID, LAST_NAME, DEPARTMENT_ID로 구성된 조회결과를 표시하시오.

```
SQL> SELECT employee_id, last_name, department_id
  FROM employees
 WHERE last_name = 'Higgins';

EMPLOYEE_ID LAST_NAME      DEPARTMENT_ID
-----
205 Higgins          110
```

☞ 오라클 데이터베이스 서비스에서는, 위의 문장의 WHERE 절에서 비교되는 상수값이, LAST_NAME 컬럼에 저장된 값과 대소문자가 다르면, 아래처럼 다른 결과가 표시됩니다.

```
SQL> SELECT employee_id, last_name, department_id
  FROM hr.employees
 WHERE last_name = 'higgins';
--기술한 상수값의 대소문자가 달라서,
--표시되는 결과가 없습니다.

선택된 행 없음
```

[실습] HR.EMPLOYEES 테이블의 LAST_NAME 컬럼에 입력된 값이, 대소문자에 상관없이, 철자가 'higgins' 인 사원의 EMPLOYEE_ID 컬럼, LAST_NAME 컬럼, DEPARTMENT_ID 컬럼으로 구성된 레코드를 표시하시오.

```
SQL> SELECT employee_id, last_name, department_id
  FROM hr.employees
 WHERE LOWER(last_name) = 'higgins';
--컬럼에 입력된 영어 문자들에 대한 대소문자를
--정확히 모르는 경우, WHERE 절의 조건에서
--컬럼을 대소문자 변환함수로 처리합니다.

EMPLOYEE_ID LAST_NAME      DEPARTMENT_ID
-----
205 Higgins          110
--WHERE 절의 조건에서 컬럼을 대소문자 변환함수로
--처리하는 경우, 비교되는 영어 상수를, 사용된 함수에
--따라 대소문자를 적당하게 명시할 수 있습니다.
```

☞ 영어 문자열이 입력되는 컬럼에 대하여, 대소문자 구분이 중요하지 않다면, SELECT 문의 WHERE 절의 조건에서 대소문자 변환함수를 사용하는 것보다는, INSERT문에서 대소문자 변환함수를 사용하는 것을 권장합니다.

4-4. 문자 함수-2: 문자열 수정 함수.

■ 문자값을 처리하는, 대표적인 오라클 단일 행 문자함수들의 사용방법 및 기능을 요약하면 다음과 같습니다.

문자함수 이름 및 사용방법	기능
CONCAT('문자열', '문자열')	두 문자값을 붙입니다.
SUBSTR('문자열', 시작 위치, 추출할 문자 수)	문자값에서 원하는 문자(열)을 추출합니다.
SUBSTR('문자열', 시작 위치)	추출할 문자 수를 생략하면, 끝까지 추출합니다.
REPLACE ('문자열', '찾을 문자', '대체 문자')	문자값에서, 특정문자(열)를 다른 문자(열)로 변경합니다.
TRIM(BOTH '문자' FROM '문자열')	문자열로부터, 앞 및 뒤(또는 앞, 또는 뒤)에 있는 지정된 문자를 모두(연속된 경우) 삭제합니다.
TRIM(LEADING '문자' FROM '문자열')	
TRIM(TRAILING '문자' FROM '문자열')	
LPAD('문자열', Byte길이, '빈칸을 채울 문자')	문자열을, 지정된 바이트 공간에 표시하고, 왼쪽 또는 오른쪽에 남은 빈 공간을 지정된 문자로 채웁니다.
RPAD('문자열', Byte길이, '빈칸을 채울 문자')	
LENGTH ('문자열'), LENGTHB ('문자열')	문자열의 길이를 문자 수 또는 바이트 수로 반환합니다.
INSTR('문자열', '찾을 문자(열)')	문자열에서 지정된 문자(열)을 찾아 위치를 반환합니다.

[실습] 다음의 실습을 통해, 문자값을 처리하는 오라클 단일 행 문자함수들의 사용방법을 확인합니다.

1> CONCAT('문자열', '문자열') 함수는, 두 문자값을 붙입니다.

다음의 CONCAT() 함수는, 'HELLO'와 'WORLD' 문자열을 붙인 결과를 표시합니다.

```
SQL> SELECT CONCAT('HELLO', 'WORLD') FROM dual;
```

```
CONCAT('HELLO', 'WORLD')
```

```
-----
```

```
HELLOWORLD
```

2> SUBSTR('문자열', 시작 위치, 추출할 문자 수) 함수는, 데이터로부터 원하는 문자(열)를 추출해 줍니다.

다음은, '20080815' 문자열로부터 각각 지정된 문자(열)을 추출하는 예입니다.

```
SQL> SELECT SUBSTR('20080815', 1, 4), -- 1 번째 글자부터 4 글자 추출
      SUBSTR('20080815', 5, 2), -- 5 번째 글자부터 2 글자 추출
      SUBSTR('20080815', 7), -- 7 번째 글자부터 끝까지 추출
      SUBSTR('20080815', -2) -- 뒤에서 2번째 글자부터 끝까지 추출
   FROM dual;
```

```
SUBSTR('20080815',1,4) SUBSTR('20080815',5,2) SUBSTR('20080815',7) SUBSTR('20080815',-2)
```

```
-----
```

2008	08	15	15
------	----	----	----

☞ SUBSTR() 함수 사용 시에, 추출할 문자 수는 생략할 수 있습니다. 생략된 경우, 맨 끝자리까지 추출됩니다.

3> LENGTH ('문자열') 함수 및 LENGTHB ('문자열') 함수는, 문자열의 길이를 각각 글자 수와 바이트 수로 반환합니다.

다음의 LENGTH() 및 LENGTHB() 함수는, '신상현' 문자열의 길이를 각각, 글자수 및 바이트 수로 반환합니다.

```
SQL> SELECT LENGTH('신상현'), LENGTHB('신상현') FROM dual;
```

```
LENGTH('신상현') LENGTHB('신상현')
```

3	9
---	---

☞ 오라클 데이터베이스에서 한글을 처리하는 코드규칙(문자셀이라고 함)을 유니코드(AL32UTF8 또는 UTF8)로 설정한 경우, 한글 한 글자는 3바이트 길이로 처리됩니다. 코드규칙을 K016MSWIN949로 설정한 경우, 한글 한 글자는 2 바이트 길이로 처리됩니다.

4> REPLACE ('문자열', '찾을 문자1', '대체 문자2') 함수는, 문자열에서 문자(열)1을 찾아서 문자(열)2로 교체합니다.

```
SQL> SELECT REPLACE('JACK and JUE', 'J', 'BL'),      —'JACK and JUE' 문자열에서 'J' 를 'BL' 로 교체
      REPLACE('Java Oracle', 'Ja', 'Ora')      —'Java Oracle' 문자열에서 'Ja' 를 'Ora' 로 교체
      FROM dual;
```

```
REPLACE('JACKANDJUE','J','BL') REPLACE('JAVAORACLE','JA','ORA')
```

BLACK and BLUE	Orava Oracle
----------------	--------------

5> TRIM('문자' FROM '문자열') 함수는, 문자열의 양 끝의 연속된 문자를 삭제합니다. 단, 중간에 있는 문자는 삭제되지 않습니다.

```
SQL> SELECT TRIM(BOTH 'S' FROM 'SSMISTHSS') AS "RESULT1",      — 양 끝의 연속된 'S' 를 삭제.
      TRIM.LEADING 'S' FROM 'SSMISTHSS') AS "RESULT2",      — 앞에 있는 연속된 'S' 를 삭제.
      TRIM.TRAILING 'S' FROM 'SSMISTHSS') AS "RESULT3"      — 뒤에 있는 연속된 'S' 를 삭제.
      FROM dual;
```

```
RESULT1 RESULT2 RESULT3
```

MISTH	MISTHSS	SSMISTH
-------	---------	---------

☞ TRIM() 함수 사용 시에, BOTH 옵션은 생략할 수 있습니다.

6> LPAD('문자열', 바이트-길이, '빈칸을 채울 문자') 함수 및 RPAD('문자열', 바이트-길이, '빈칸을 채울 문자') 함수는, 문자열을, 지정된 바이트 공간에 표시하고, 각각 왼쪽(L) 및 오른쪽(R)에 남은 빈 공간을 지정된 문자로 채웁니다.

```
SQL> SELECT LPAD('SMITH',10,'*') AS "RESULT1", RPAD('SMITH',10,'*') AS "RESULT2" FROM dual;
```

RESULT1	RESULT2
---------	---------

— LPAD()함수는 'SMITH'를 10-BYTE 공간에 표시하고, 남은 공간을 왼쪽부터 '*'로 채움.
 — RPAD()함수는 'SMITH'를 10-BYTE 공간에 표시하고, 남은 공간을 오른쪽부터 '*'로 채움.

```
*****SMITH SMITH*****
```

7> **INSTR('문자열', '찾는 문자(열)'**, 검색을 시작할 자리 수, 존재 횟수) 함수는, 찾는 문자(열)이 있으면 그 문자(열)이 시작된 자리 수를 반환하며, 찾는 문자(열)이 없으면 0을 반환합니다. 주로, 입력되는 데이터에 명시해야 하는 문자가 빠졌거나, 명시하면 안 되는 문자가 존재하는지를 검증 할 때 사용됩니다.

```
SQL> SELECT INSTR('HELLO ORACLE','L',1,1) "RESULT1",      —1번째 글자부터 1번째 나오는 L의 위치.
          INSTR('HELLO ORACLE','L') "RESULT2",           —1번째 글자부터 1번째 나오는 L의 위치.
          INSTR('HELLO ORACLE','L',4,2) "RESULT3",       —4번째 글자부터 2번째 나오는 L의 위치.
          INSTR('HELLO ORACLE','L',4,3) "RESULT4",       —4번째 글자부터 3번째 나오는 L의 위치.
          INSTR('HELLO ORACLE','L',-8,2) "RESULT5",      —뒤에서 8번째 글자부터 2번째 나오는 L의 위치.
          INSTR('HELLO ORACLE','ORACLE') "RESULT6"        —1번째 글자부터 1번째 나오는 ORACLE의 위치.

          FROM dual;
```

RESULT1	RESULT2	RESULT3	RESULT4	RESULT5	RESULT6
3	3	11	0	3	7

☞ INSTR() 함수 사용 시에, 검색을 시작할 자리 수와 존재 회수는 각각 생략할 수 있습니다. 생략된 경우, 각각 1 번째 문자부터 검색을 시작하며, 1 번째 나오는 문자의 위치를 반환합니다.

8> HR.EMPLOYEES 테이블로부터, JOB_ID 컬럼의 4번째 문자부터 끝까지 추출한 문자열을, 소문자로 변환한 값이 'rep'인 조건을 만족하는 모든 사원에 대하여, 사원ID(EMPLOYEE_ID컬럼), 이름(FIRST_NAME 컬럼)과 성(LAST_NAME 컬럼)을 불인값, 직책ID(JOB_ID 컬럼), 성의 글자수, 성에 a 문자가 있는 위치를 표시하는 값들로 구성된 레코드를 표시하시오. 단, 출력결과의 머리글은 임의로 정해서 표시하시오.

```
SQL> SELECT employee_id, CONCAT(first_name, last_name) AS "Name" ,job_id, LENGTH(last_name),
          INSTR(last_name, 'a') AS "Conatains 'a'?"
          FROM hr.employees
          WHERE LOWER(SUBSTR(job_id, 4)) = 'rep';
```

EMPLOYEE_ID	Name	JOB_ID	LENGTH(LAST_NAME)	Conatains 'a'?
202	PatFay	MK_REP	3	2
203	SusanMavris	HR_REP	6	2
204	HermannBaer	PR_REP	4	2
150	PeterTucker	SA_REP	6	0
151	DavidBernstein	SA_REP	9	0
152	PeterHall	SA_REP	4	2
153	ChristopherOlsen	SA_REP	5	0
154	NanetteCambrault	SA_REP	9	2
155	OliverTuvault	SA_REP	7	4
156	JanetteKing	SA_REP	4	0
...				
33개의 행이 선택됨				

4-5. 단일 행 함수의 중첩

- 단일 행 함수는 제한 없이 여러 번 중첩할 수 있습니다.
- 함수가 중첩된 경우, 안쪽에 있는 함수가 먼저 처리되어 그 결과를 바깥쪽의 함수에게 전달하면, 바깥쪽의 함수가 안쪽 함수로부터 전달받은 처리결과를 이용하여 처리됩니다. 따라서, 안쪽부터 바깥쪽으로, 순서대로 중첩된 함수들이 처리됩니다.

[실습] HR.EMPLOYEES 테이블로부터, DEPARTMENT_ID가 10인 사원에 대하여, LAST_NAME컬럼과, LAST_NAME 컬럼에서 추출된 처음 8 글자와 그 다음에 _US 문자열이 붙어서, 모두 대문자로 표시된 값으로 구성된 레코드를 표시하시오.

```
SQL> SELECT last_name, UPPER(CONCAT(SUBSTR(LAST_NAME, 1, 8), '_US'))
   FROM hr.employees
 WHERE department_id = 10;
```

LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME, 1, 8), '_US'))
Whalen	WAHLEN_US

☞ 위의 SELECT 절에서, SUBSTR(), CONCAT(), UPPER() 함수 순서로, 중첩된 함수가 처리됩니다

☞ SUBSTR() 함수 사용 시에 추출한 글자수가 값의 글자수보다 큰 경우에는. 추출을 시작하는 위치부터 끝까지 전체가 표시됩니다. 위의 예에서, LAST_NAME 컬럼의 값인 Whalen의 글자수가, 추출할 글자수인 8 글자 보다 작으므로, 값 전체가 표시되었습니다.

4-6. 숫자 함수

- 숫자값을 처리하는, 대표적인 오라클 단일 행 숫자함수들의 사용방법 및 기능을 요약하면 다음과 같습니다.

문자함수 이름 및 사용방법	기능
MOD(숫자1, 숫자2)	숫자1을 숫자2로 나누고 남은 나머지를 반환합니다.
ROUND (숫자, 양의_정수로_된_자리_수)	숫자를, 반올림하여, 소수점 이하 유효 자리까지만 표시합니다.
ROUND (숫자, 음의_정수로_된_자리_수)	숫자의 정수부를, 명시된 자리에서 반올림하여, 표시합니다.
TRUNC (숫자, 양의_정수로_된_자리_수)	숫자를, 절삭하여, 소수점 이하 유효 자리까지만 표시합니다.
TRUNC (숫자, 음의_정수로_된_자리_수)	숫자의 정수부를, 명시된 자리에서 절삭하여, 표시합니다.

[실습] 다음의 실습을 통해, 문자값을 처리하는 오라클 단일 행 숫자함수들의 사용방법을 확인합니다.

1> 다음의 MOD() 함수는, 1600을 300으로 나누고, 남은 나머지인 100을 반환합니다.

```
SQL> SELECT MOD(1600, 300) FROM dual ;
```

```
MOD(1600,300)
-----
100
```

2> 다음의 ROUND() 함수의 처리 결과를 각각 확인합니다.

```
SQL> SELECT ROUND(1745.9265,2) RESULT1, --소수점 이하, 3번째 자리에서 반올림하여, 2번째 자리까지 표시됨.
      ROUND(1745.9465,0) RESULT2, --소수점 이하, 1번째 자리에서 반올림하여, 정수부만 표시됨.
      ROUND(1745.9265) RESULT3, --소수점 이하, 1번째 자리에서 반올림하여, 정수부만 표시됨.
      ROUND(1745.9265,-1) RESULT4 --정수부의, 1번째 자리에서 반올림하여 표시됨.
      FROM dual ;
```

RESULT1	RESULT2	RESULT3	RESULT4
1745.93	1746	1746	1750

3> 다음의 TRUNC() 함수의 처리 결과를 각각 확인합니다.

```
SQL> SELECT TRUNC(1745.9265,2) RESULT1, --소수점 이하, 3번째 자리부터 절삭된 후, 2 번째 자리까지 표시됨.
      TRUNC(1745.9465,0) RESULT2, --소수점 이하가 모두 절삭되어 정수부만 표시됨.
      TRUNC(1745.9265) RESULT3, --소수점 이하가 모두 절삭되어 정수부만 표시됨.
      TRUNC(1745.9265,-1) RESULT4 --정수부 1번째 자리에서 절삭하여, 표시됨.
      FROM dual ;
```

RESULT1	RESULT2	RESULT3	RESULT4
1745.92	1745	1745	1740

```
SQL> SELECT floor(1745.345), ceil(1745.345) FROM dual ;
```

```
FLOOR(1745.345) CEIL(1745.345)
```

```
-----
```

```
1745       1746
```

```
floor(n): n      가      ,n  
ceil(n) :n      가      ,n
```

4> HR.EMPLOYEES 테이블로부터, JOB_ID 컬럼을 소문자로 변환한 값이 'sa_rep' 인 조건을 만족하는 모든 사원에 대하여, 사원의 성(LAST_NAME 컬럼), 급여(SALARY 컬럼), 급여를 5000으로 나누고 소수부분을 절삭한 값, 및 급여를 5000으로 나누고 남은 나머지 값으로 구성된 레코드를 표시하시오.

```
SQL> SELECT last_name, salary, TRUNC(salary/5000,0), MOD(salary, 5000)
   FROM hr.employees
  WHERE LOWER(job_id) = 'sa_rep';
```

LAST_NAME	SALARY	TRUNC(SALARY/5000,0)	MOD(SALARY,5000)
Tucker	10000	2	0
Bernstein	9500	1	4500
Hall	9000	1	4000
Olsen	8000	1	3000
Cambrault	7500	1	2500
...			
30개의 행이 선택됨			

151 ~ 152 .

4-7. SYSDATE, SYSTIMESTAMP, CURRENT_TIMESTAMP, LOCALTIMESTAMP, CURRENT_DATE 함수

- SYSDATE 함수와 SYSTIMESTAMP 함수는, 오라클 데이터베이스 서비스가 실행 중인 운영체제의 날짜시간 및 시간대를 각각 DATE 데이터유형과 TIMESTAMP WITH TIME ZONE 데이터유형으로 반환합니다.
- CURRENT_DATE 함수와 CURRENT_TIMESTAMP 함수는, 현재 접속한 세션의 날짜시간 및 시간대를 각각 DATE 데이터유형과 TIMESTAMP WITH TIME ZONE 데이터유형으로 반환합니다.
- LOCALTIMESTAMP 함수는 현재 접속한 세션의 날짜시간을 TIMESTAMP 데이터유형으로 반환합니다.

[실습] 오라클 데이터베이스 서버 시스템의 현재 날짜와 현재 날짜시간 및 시간대를 확인하고, 접속한 세션의 현재 날짜와 현재 날짜시간 및 시간대를 확인하시오.

```
SQL> COL CURRENT_TIMESTAMP FORMAT A40 ;
SQL> SELECT SYSDATE, SYSTIMESTAMP(2), LOCALTIMESTAMP(2), CURRENT_TIMESTAMP, CURRENT_DATE
   FROM dual ;
```

SYSDATE	SYSTIMESTAMP(2)	LOCALTIMESTAMP(2)
CURRENT_TIMESTAMP	CURRENT_	
17/07/05 17:07/05 16:41:32.940000000 +09:00	17/07/05 16:41:32.940000000	
17/07/05 16:41:32.942000000 ASIA/SEOUL	17/07/05	

☞ DATE 데이터유형에 대한 표시형식을 설정하지 않았으므로, SYSDATE 및 CURRENT_DATE의 결과에 날짜만 표시됩니다.

- ☞ SYSTIMESTAMP 함수, CURRENT_TIMESTAMP 함수, LOCALTIMESTAMP 함수는 함수 뒤에, 표시되는 밀리초의 글자수를 명시할 수 있습니다. 만약, 밀리초 글자수 생략되면, 디플트로 밀리초가 6글자가 처리되어 표시됩니다.
- ☞ ~~TIMESTAMP~~ 또는 **TIMESTAMP WITH TIME ZONE** 또는 **TIMESTAMP WITH LOCAL TIME ZONE** 데이터유형이 사용된 데이터베이스 테이블의 컬럼에, 날짜시간 값을 입력할 때에는 LOCALTIMESTAMP 함수, CURRENT_TIMESTAMP, SYSTIMESTAMP 함수 중 하나를 사용하십시오. SYSDATE 또는 CURRENT_DATE 함수는 날짜시간 값에 잘못된 처리가 발생될 수 있습니다.

4-8. 데이터유형 변환 함수

- 데이터유형 변환 함수에, 형식모델을 이용하여, 상수 또는 컬럼의 값을 원하는 데이터유형을 변환하는 것을, "명시적인 데이터유형 변환"이라고 합니다.
- 오라클 단일 행 데이터유형 변환 함수들의 사용방법 및 기능을 요약하면 다음과 같습니다.

데이터유형 변환 함수 이름 및 사용방법	기능
TO_DATE('날짜문자열','날짜 형식 모델') TO_DATE('날짜문자열','날짜 형식 모델','NLS-매개변수')	날짜문자열의 표시형식을, 형식 모델로 지정하여, 날짜문자열을 날짜(DATE) 데이터유형으로 변환합니다.
TO_CHAR('날짜표현식','날짜 형식 모델') TO_CHAR('날짜표현식','날짜 형식 모델','NLS-매개변수')	원하는 표시형식을, 형식 모델로 지정하여, 날짜값 또는 숫자값을 문자(VARCHAR2) 데이터유형으로 변환합니다.
TO_CHAR('숫자표현식','날짜 형식 모델') TO_CHAR('숫자표현식','날짜 형식 모델','NLS-매개변수')	숫자문자열의 표시형식을, 형식 모델로 지정하여, 숫자문자열을 숫자(NUMBER) 데이터유형으로 변환합니다.
TO_NUMBER('숫자문자열','숫자 형식 모델') TO_NUMBER('숫자문자열','숫자 형식 모델','NLS-매개변수')	숫자문자열의 표시형식을, 형식 모델로 지정하여, 숫자문자열을 숫자(NUMBER) 데이터유형으로 변환합니다.

4-8-1. 데이터유형 변환 함수에서 사용되는 대표적인 형식 모델(FORMAT MODEL).

- 숫자문자열 또는 숫자 유형 데이터에 대한 대표적인 형식 모델은 다음과 같습니다.

형식모델	의미 (사용 예)	형식모델	의미 (사용 예)
9	자리 수 (예, 999,999)	\$	달러 통화 단위 (예, \$999,999)
0	자리 수 (예, 099,999)	L	지역 통화 단위 (예, L999,999)
.	소수점 (예, 999,999.99)	,	천 단위 구분 기호 (예, L999,999)
.	소수점 (예, 999,999.99)	,	천 단위 구분 기호 (예, L999,999)
D	소수점 호출 (예, 999,999D99)	G	천 단위 구분 기호 호출 (예, L999G999)

- ☞ 9의 개수가 처리하려는 숫자값의 글자수 보다 부족하면, #####으로 표시됩니다.

- 날짜문자열 또는 날짜 유형 데이터에 대한 대표적인 형식 모델은 다음과 같습니다.

형식모델	의미	형식모델	의미
YYYY	4자리 숫자 연도 (예, 2010)	DD	숫자로 된 달의 일.
YY	2자리 숫자 연도 (예, 10)	DAY	요일 (한국어: 금요일, 영어: FRIDAY)
RR	2자리 숫자 연도 (예, 10)	DY	3자리 약어로 표시한 요일 (한국어: 금, 영어: FRI)
YEAR	영어로 표시한 연도	HH24	숫자로 된 시간(24시간 표기법)
MM	숫자로 된 월 (예, 12)	HH, HH12	숫자로 된 시간(12시간 표기법)
MONTH	달 이름(전체) (한국어: 4월, 영어: APRIL)	MI	숫자로 된 분
		SS	숫자로 된 초
MON	3자리 약어로 된 달 이름 (한국어: 4월, 영어: APR)	AM	오전/오후 (한국어: 오전/오후, 영어: AM/PM)

SSXFF : (Timestamp
(DATE))

[참고] 년도를 2자리로 표시할 때 사용하는 RR-형식모델과 YY-형식모델의 차이

- 세기 없이 년도만 YY 형식으로 표시되면, 세기는 항상 현재 세기와 동일합니다.

- 세기 없이 연도만 RR 형식으로 표시되면, 세기는 아래의 기준으로 고려됩니다.

현재의 세기가 중반(40 ~ 59 연도)에 이르면, RR 동작이 예상과 다를 수도 있습니다.	값에 표시된 2자리 년도	
	0 ~ 49	50 ~ 99
현재의 2자리 년도	0 ~ 49	현재세기
	50 ~ 99	현재세기 + 1

- RR은, 2 자리로 표시된 연도에 대하여, 위의 표에 있는 공식으로 다른 세기를 지정할 수 있습니다. 이를 통해, 사람이 느끼는 세기를 고려할 수 있습니다. 단, 현재의 세기가 중반(40 ~ 59 연도)에 이르면, RR 동작이 예상과 다를 수도 있습니다.

- 다음은 지정된 2 자리 연도가 YY, RR 형식일 때, 해석되는 방법을 나타낸 예입니다.

현재 연도	지정된 2자리 연도	지정된 2자리 연도가 YY 형식일 때 4자리 연도	지정된 2자리 연도가 RR 형식일 때 4자리 연도
1988	95	1995	1995
1992	17	1917	2017
2008	17	2017	2017
1949 (세기 중반)	50	1950	1850 (예상과 다름)
1950 (세기 중반)	49	1949	2049 (예상과 다름)

- RR과 YY의 세기 처리방식의 차이 및 RR 자체의 문제는, 2 자리 연도 대신, 세기와 연도를 모두 포함한 YYYY 형식을 사용하면 모두 해결됩니다.

[실습] 다음은 RR 및 YY의 차이를 이해하기 위한 예제 문장입니다.

```
SQL> SELECT TO_CHAR(TO_DATE('94/01/11','RR/MM/DD')-7, 'YYYY/MM/DD') AS "RR" ,
      TO_CHAR(TO_DATE('94/01/11','YY/MM/DD')-7, 'YYYY/MM/DD') AS "YY"
      FROM dual ;
```

RR	YY
1994/01/04	2094/01/04

4-8-2. 데이터유형 변환 함수를 이용한 명시적 데이터유형 변환

- **TO_DATE('날짜문자열', '날짜 형식 모델'[, 'NLS-매개변수'])** 함수는, 날짜문자열의 표시형식을, 형식 모델로 지정하여, 날짜문자 값을 날짜 데이터유형의 값으로 변환합니다.
 - 날짜문자 값에 언어로 표시된 형식 요소가 있을 때, NLS_DATE_LANGUAGE 매개변수로 사용된 언어를 설정해야 합니다.
 - 날짜문자 값이 숫자 형식 요소로만 이루어져 있으면, NLS-매개변수를 사용할 필요가 없습니다.
- **TO_CHAR('날짜 표현식', '날짜 형식 모델'[, 'NLS-매개변수'])** 함수는, 형식 모델을 사용하여, 날짜 또는 숫자 데이터 유형의 값을 VARCHAR2 데이터유형의 문자열로 변환합니다. 숫자 및 날짜시간 값을 사용자가 표시하고 싶은 형식대로 지정할 때 사용됩니다.
- **TO_NUMBER('숫자문자열', '숫자 형식 모델', 'NLS-매개변수')** 함수는, 숫자문자열에 해당하는 형식모델을 명시하여, 문자값을 숫자 데이터유형으로 변환합니다.

[실습] '01-SEP-95' 날짜의 3일 후의 날짜를 표시하시오.

```
SQL> SELECT TO_DATE('01-SEP-95','DD-MON-RR', 'NLS_DATE_LANGUAGE=AMERICAN')+3 AS RESULT
      FROM dual;
```

RESULT	--날짜 상수에 언어가 포함된 경우, TO_DATE() 함수에 사용된 언어를 지정하는 NLS_DATE_LANGUAGE --매개변수를 설정해야 합니다. 날짜 상수가 숫자로만 된 경우에는, NLS_DATE_LANGUAGE 매개변수를 --지정할 필요가 없습니다.
95/09/04	

☞ SQL문에, 위에서처럼 날짜-시간 상수를 명시하는 경우에는, 반드시 TO_DATE() 함수로 처리해야만 세션의 표시 형식에 관계 없이 정상적인 처리결과를 얻을 수 있습니다.

[실습] SYSDATE 함수의 처리 결과가, 'YYYY/MM/DD HH24:MI:SS AM DAY' 형식으로 표시되도록 하시오.

```
SQL> SELECT SYSDATE, TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS AM DAY') FROM dual;
```

SYSDATE	TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS AM DAY')
11/07/25 2011/07/25 18:02:19 오후 월요일	--TO_CHAR()에서 지정한 형식대로 표시되었습니다.

[실습] SYSDATE 함수의 처리 결과가, 'YYYY/MM/DD HH24:MI:SS AM DAY' 형식으로 표시되도록 하시오.

단, 요일 및 오전/오후가 영어로 표시되도록 하시오.

```
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS AM DAY', 'NLS_DATE_LANGUAGE=AMERICAN')
  FROM dual;
```

```
TO_CHAR(SYSDATE, 'YYYY/MM/DDHH24:MI:SSAMDAY', 'NLS_DATE_LANGU
-----
2011/07/25 18:02:19 AM MONDAY
```

--NLS_DATE_LANGUAGE 매개변수를 이용하여
표시되는 언어를 영어로 설정했으므로
오전/오후 및 요일이 영어로 표시되었습니다.

☞ TO_CHAR() 함수로, DATE 데이터유형을 변환할 때, NLS_DATE_LANGUAGE 매개변수를 사용하여, 표시되는 언어를 지정할 수 있습니다.

[실습] SYSDATE 함수의 처리 결과가, 'YYYY/MM/DD HH24:MI:SS AM DAY' 형식으로 표시되도록 하시오.

단, 날짜 요소 구분자를 / 및 : 이 아닌, 년, 월, 일, 시, 분, 초로 표시하시오.

```
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY"년" MM"월" DD"일" HH24"시" MI"분" SS"초") FROM dual;
```

TO_CHAR()

```
TO_CHAR(SYSDATE, 'YYYY"년" MM"월" DD"일" HH24"시" MI"분" SS"초")
-----
2016년 01월 16일 19시 01분 02초
```

-- TO_DATE() 함수로 날짜값을 처리 시에, 형식모델 안에
큰따옴표로 감싼 문자열(년,월,일 등)을 기호(/ - , . 빈칸)
대신 결과에 표시하도록 지정했습니다.

TO_CHAR()

☞ TO_DATE() 함수로 날짜값을 처리할 때, 날짜와 함께 표시하고 싶은 문자열을 큰따옴표로 감싸서 형식 모델 안에

명시하면, 처리결과에 포함되어 표시됩니다.

```
SELECT employee_id, last_name, TO_CHAR(hire_date, 'YYYY/MM') FROM hr.employees
WHERE TO_CHAR(hire_date, 'YYYY') = '2008';
==> WHERE
      WHERE hire_date BETWEEN TO_DATE('2008/01/01', 'YYYY/MM/DD')
      AND TO_DATE('2008/12/31', 'YYYY/MM/DD');
```

[실습] SYSDATE 함수의 처리 결과가, 'YYYY/MON/DD HH24:MI:SS AM DAY' 형식으로 표시되도록 하시오.

단, 월, 요일 및 오전/오후가 영어로 표시되고, 월은 첫 글자 대문자 나머지 소문자, 오전/오후는 모두 소문자,
요일은 모두 대문자로 표시되도록 하시오.

```
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY-Mon-DD HH24:MI:SS am DAY', 'NLS_DATE_LANGUAGE=AMERICAN')
  FROM dual;
```

```
TO_CHAR(SYSDATE, 'YYYY-MON-DDHH24:MI:SSAMDAY', 'NLS_DATE_LANGUAGE=AMERI
-----
2013-Apr-26 13:52:27 pm FRIDAY
```

☞ 영어로 표시하는 형식 모델(MONTH, MON, DAY, DY, AM, 등)를 사용할 때, 대소문자를 섞어서 명시하면,

다음의 규칙대로 대소문자가 구분되어 표시됩니다.

형식모델 표시 방법	표시될 때, 적용되는 규칙 및 결과
첫 글자가 소문자 (예, mONTH)	전체가 소문자로 표시됨 (april)
첫 글자 및 두 번째 글자가 대문자 (예, MOnth)	전체가 대문자로 표시됨 (APRIL)
첫 글자는 대문자, 두 번째 글자는 소문자 (예, MoNTH)	첫 글자만 대문자 나머지 소문자로 표시됨 (April)

[실습] '07/05/01 02:00:04' 날짜가, 'RR-MM-DD HH24:MI:SS' 형식으로 표시되도록 하시오.

단, 년, 월, 일에 대하여 사용하지 않은 자리에 0이 표시되지 않고. 시, 분, 초는 0이 표시되도록 하시오.

```
SQL> SELECT TO_CHAR(TO_DATE('07/05/01 02:00:04', 'RR/MM/DD HH24:MI:SS'),
    'fmRR-MM-DD fmHH24:MI:SS') AS RESULT
  FROM dual;
```

RESULT	-- 첫 번째 fm에 의하여 RR-MM-DD까지는 0이 채워지지 않습니다. 두 번째 fm에 의하여 첫 번째 fm의 기능이 해제되어, HH24:MI:SS는 다시 0이 채워집니다.
	7-5-7 02:00:04

- ☞ 숫자로 표시되는 2 자리 년도, 월, 일, 시, 분, 초에서 두 자리 중에 한 자리만 사용한 경우, 기본으로 남은 빈 자리를 0으로 채웁니다(예, 2010/03/02 04:06:04).
- ☞ 형식 모델 요소의 제일 앞에 fm 요소를 한 번만 명시하면, fm이 명시된 위치부터, 두 자리 중, 사용하지 않은 자리를 0으로 채우지 않습니다(예, 2010/3/2 4:6:4 PM WEDNESDAY).
- ☞ fm 요소가 명시되어, 0을 채우지 않는 기능이 설정되었다가, 다시 fm 요소를 한 번 더 명시하면, 설정된 기능이 해제됩니다.

[실습] SYSDATE 함수의 처리 결과가, 'YYYY/MON/DD HH24:MI:SS AM DAY' 형식으로 표시되도록 하시오.

단, 년, 월, 일은 숫자를 영어로 표시되도록 하고, 특히 월과 일은 영어서수로 표시되도록 하시오.

```
SQL> SELECT TO_CHAR(SYSDATE, 'yYYYSP/Mmsspth/DDspth') AS RESULT FROM dual;
```

RESULT	--SP 요소에 의하여 yYY, Mm, 및 DD 요소가 숫자가 아닌 영어로 표시되었습니다.
	two thousand sixteen/Second/SECOND

- ☞ SP 요소는 숫자요소를 영어로 표시합니다. TH 요소는 숫자요소에 TH를 붙여 서수로 표시합니다. SPTH를 같이 사용하면, 숫자가 영어 서수로 표시됩니다.
- ☞ yYYYSP에서 yYY 요소의 첫 글자가 소문자이므로, 연도가, 영어 소문자로 표시되었습니다
- ☞ Mmsspth에서 Mm 요소의 첫 글자가 대문자이고 다음 글자가 소문자이므로, 첫 글자 대문자 나머지는 소문자로 결과가 표시되었습니다.
- ☞ DDspth에서 DD 요소의 첫 두 글자가 대문자이므로, 일이 영어 대문자로 표시되었습니다.

[실습] 현재 날짜에서 년도, 월, 일, 요일 및 시각을 추출하여 각각 표시하시오.

```
SQL> SELECT sysdate, TO_CHAR(sysdate,'YYYY') AS "YEAR", TO_CHAR(sysdate,'MM') AS "MONTH",
    TO_CHAR(sysdate,'DD') AS "DAY", TO_CHAR(sysdate,'DAY') AS "WEEKDAY" ,
    TO_CHAR(sysdate,'HH24:MI:SS') AS "TIME"
  FROM dual ;
```

SYSDATE	YEAR	MO	DA	WEEKDAY	TIME	--SYSDATE 함수가 반환하는 데이터베이스 서버의 현재 날짜 및 시간 --값으로부터 각각 TO_CHAR() 함수를 이용하여, 년도, 월, 일, 요일, 13/04/26 2013 04 26 금요일 23:09:54 --시간이 추출되어 문자 값으로 표시됩니다.
---------	------	----	----	---------	------	---

[실습] HR.EMPLOYEES 테이블로부터, SALARY 컬럼의 값에 달러(\$) 또는 원(₩) 통화단위가 표시되고, 숫자 값 사이에 천 단위 구분자로 콤마(,)가 표시되도록 처리한 SELECT문 예제입니다.

```
SQL> SELECT salary, TO_CHAR(salary, '$9G999G999D99'), TO_CHAR(salary, 'L0,999,999.99')
   FROM hr.employees
  WHERE employee_id=100 ;

SALARY TO_CHAR(SALARY TO_CHAR(SALARY, 'L99,
-----
24000 $24,000.00 ₩000,024,000.00
```

- ☞ 형식모델에서 정수부로 7 자리, 소수점 이하 2자리까지 표시되도록 9를 정수부에 7개, 소수점 이하 2개 명시했습니다.
- ☞ \$와 L 에 의하여 통화단위가 앞에 표시되었습니다. 단 \$ 와 L 형식모델은 동시에 사용될 수 없습니다.
- ☞ 첫 글자를 9 대신 0으로 표시하면, 사용되지 않은 자리들을 앞에서부터 0으로 채웁니다.

[실습] MI(음수일 때 -를 뒤에 표시) 및 PR (음수를 <>로 감쌈) 형식모델 사용 실습

```
SQL> SELECT TO_CHAR(0-salary, '99,999.99MI'), TO_CHAR(0-salary, '99,999.99PR')
   FROM hr.employees
  WHERE employee_id=100 ;

TO_CHAR(0-SALARY, '99,999.99MI') TO_CHAR(0-SALARY, '99,999.99PR') --단, MI 및 PR 요소는 형식모델에서 뒤에
--명시하며, 동시에 사용될 수 없습니다.
-----
24,000.00- <24,000.00>
```

[주의] 형식으로 지정한 9의 개수가 숫자 데이터의 정수 부분의 자리 수 보다 작으면, #####로 표시됩니다.

```
SQL> SELECT salary, TO_CHAR(salary, '9,999.99') FROM hr.employees WHERE employee_id=100 ;

SALARY TO_CHAR(SALARY, '9,999.99')

-----
24000 ##### --9 를 몇 개 더 적어서 형식의 글자수를 늘려주면, 값이 정상적으로 표시됩니다.
```

[실습] HR.EMPLOYEES 테이블로부터, 사원ID가 121 인 사원의 입사일과 급여로 구성된 레코드를 표시하시오.

단, 입사일은 YYYY/MM/DD DAY 형식으로, 급여는 통화단위를 앞에 붙이고 천 단위 구분자와 소수점을 명시하시오.

```
SQL> SELECT TO_CHAR(HIRE_DATE, 'YYYY/MM/DD DAY') "입사일",
      TO_CHAR(salary, 'L999G999D00') "급여"
    FROM hr.employees
   WHERE employee_id=121;

입사일           급여
-----
2005/04/10 일요일 ₩8,200.00
```

```
SELECT TO_CHAR(HIRE_DATE, 'YYYY" "MM" "DD" " DAY') "",
      TO_CHAR(salary, 'L999G999D00'||' ' ' ') "
    FROM hr.employees
   WHERE employee_id=121;
```

--점속 세션의 국가가 한국이므로,
--요일이 한국어로 표시됩니다.

[실습] '₩123,678.23' 문자열, '123,678.23원' 문자열 및 'KRW123,000' 문자열을 숫자로 변환하시오

```
SQL> SELECT TO_NUMBER('₩123,678.23', 'L999,999,999.9999') RESULT1,
      TO_NUMBER('123,678.23원', '999G999G999D9999L', 'NLS_CURRENCY='원') RESULT2
     FROM dual;
```

RESULT1	RESULT2
123678.23	123678.23

- ☞ 숫자문자열의 정수 및 소수의 숫자들이 처리될 수 있도록 형식모델에, 충분한 개수의 9 요소를 명시합니다.
- ☞ 숫자문자열의 정수뿐만 아니라, 소수에서도 9 요소가 부족하면, 오류가 발생됩니다.
- ☞ 숫자문자열에 포함된 소수점은 마침표(.) 및 D 요소로, 그룹구분자는 콤마(,) 및 G 요소로 처리됩니다.
- ☞ 숫자문자열에 포함된 \$ 또는 ₩는 각각 \$ 요소와 L 요소로 처리됩니다.
- ☞ TO_NUMBER() 함수로 숫자문자열을 처리할 때, NLS_CURRENCY = '원' NLS-매개변수를 사용하여, 통화기호로 사용된 기호나 문자를 지정할 수 있습니다. 작은 따옴표로 2번 감싸서, 값을 설정해야 합니다.

[참고] 암시적 데이터유형 변환 (자동 데이터유형 변환).

■ 데이터유형 변환함수를 사용하지 않고, 다음과 같은 경우에, 값의 데이터유형이 자동으로 변환하는 것을 "암시적 데이터유형 변환"이라고 합니다.

처리 전 데이터유형	처리 후 데이터유형	비고
VARCHAR2 or CHAR	NUMBER	숫자로 된 문자만 숫자로 자동 변환됩니다.
VARCHAR2 or CHAR	DATE	세션의 DATE 표시 형식과 일치하는 경우에만 자동 변환됩니다.
NUMBER	VARCHAR2	숫자는 문자로 항상 자동 변환됩니다.
DATE	VARCHAR2	날짜는 세션에서 표시되는 형식 그대로 문자로 자동 변환됩니다.

- ☞ SQL문의 처리 중에, 위의 설명처럼 특정 경우에 해당되면, 데이터유형이 다른 데이터유형으로 자동으로 변환됩니다. 그렇지만, 자동 데이터유형 변환은, 해당되는 경우를 벗어날 때는, 잘못된 처리나 오류가 발생될 수 있기 때문에, 데이터유형 변환함수를 사용하여, 명시적으로 데이터유형을 변환하는 것을 권장합니다.

[실습] 다음은 자동 데이터유형 변환을 이해하기 위한 실습입니다.

```
SQL> SELECT '1' + '1' AS "RESULT1", 1 || 1 AS "RESULT2" FROM dual;
```

RESULT1	RESULT2
2 11	2 11

- ☞ RESULT1은 산술연산자(+) 때문에 문자 '1' 이 자동으로 숫자 데이터유형으로 변환되어 연산자가 처리되었습니다.
- ☞ RESULT2는 || 연산자에 의하여 숫자 1 이 문자 '1'로 자동으로 데이터유형 변환이 수행 되었습니다.

4-8-3. WHERE 절에서 TO_DATE() 함수를 이용한 날짜 상수 처리.

- 날짜 상수를 WHERE절에 명시할 때, 세션의 날짜 표시형식과 일치하지 않으면, 오류나 잘못된 결과가 반환됩니다.
따라서, WHERE절에 날짜 상수를 직접 기술할 때는, 반드시 TO_DATE() 변환 함수로 날짜 상수를 처리합니다.

[실습] HR.EMPLOYEES 테이블로부터, 입사일(HIRE_DATE 컬럼)이 2008년2월1일 이 후인 사원들의 성(LAST_NAME 컬럼) 및 입사일로 구성된 레코드의 데이터를 조회하시오. 단 날짜상수를 'RR/MM/DD' 형식으로 명시하시오.

```
SQL> SELECT last_name, hire_date FROM hr.employees WHERE hire_date > '08/02/01';
```

LAST_NAME	HIRE_DATE	--세션의 표시형식과 동일한 형식으로 WHERE절에 날짜 상수를 명시했으므로, --오류 없이 결과가 표시됩니다.
Markle	08/03/08	
Philtanker	08/02/06	--한글 Windows에 설치된 SQL*Developer는 기본적으로, --언어는 'KOREAN'으로, 날짜 표시형식은 'RR/MM/DD'로 설정됩니다.
Lee	08/02/23	
Ande	08/03/24	
Banda	08/04/21	
Kumar	08/04/21	
Geoni	08/02/03	

7개의 행이 선택됨

- ☞ 위의 SELECT문의 WHERE절에 날짜 상수를 다음처럼, 세션의 표시형식과 다른 형식으로 기술하면, 오류가 발생됩니다.

```
SQL> SELECT last_name, hire_date FROM hr.employees WHERE hire_date > '08/2월/01';
```

명령의 1 행에서 시작하는 중 오류 발생 - --세션의 표시형식과 다른 RR/MON/DD 형식으로 상수가 명시되었으므로,
SELECT last_name, hire_date --오류가 발생되었습니다.
FROM hr.employees

WHERE hire_date > '08/2월/01'

오류 보고 -

ORA-01861: 리터럴이 형식 문자열과 일치하지 않음

[실습] HR.EMPLOYEES 테이블로부터, 2002년6월15일 보다 전에 입사(HIRE_DATE 컬럼)한 사원들의 성(LAST_NAME 컬럼) 및

입사일로 구성된 레코드의 데이터를 조회하시오. 단, 날짜상수를 'YYYY-MM-DD' 형식을 사용하고,

세션의 표시형식과 상관없이 항상 정상적으로 처리되도록 하시오.

```
SQL> SELECT last_name, hire_date FROM hr.employees  
      WHERE hire_date < TO_DATE('2002-06-15', 'YYYY-MM-DD');
```

LAST_NAME	HIRE_DATE	--2002-06-15' 날짜문자값을 TO_DATE() 함수로 'YYYY-MM-DD' 날짜 형식모델을 설정해서 --처리했으므로, 세션의 기본 날짜 표시형식에 상관없이 DATE 데이터유형으로 변환되어, --정상적인 처리결과를 얻을 수 있습니다.
Mavris	02/06/07	
Baer	02/06/07	
Higgins	02/06/07	
Gietz	02/06/07	
De Haan	01/01/13	

☞ SQL문에서 날짜 상수를 명시할 때는 반드시 TO_DATE() 함수로 처리합니다!

[참고] 날짜 데이터유형의 값에 대한 산술 연산 처리

- 오라클 데이터베이스 서비스에서는, 세기, 년, 월, 일, 시, 분, 초로 이루어진 내부 숫자형식으로 날짜시간 값을 저장합니다.

- 날짜 데이터유형의 값에 대하여 다음과 같은 산술연산이 가능합니다.

날짜에 대한 산술연산	사용 예	사용 예의 의미
날짜 + 숫자	SYSDATE + 3	3일 후
	SYSDATE + 3/24	3시간 후
날짜 - 숫자	SYSDATE - 3	3일 전
	SYSDATE - 30/1440	30분 전
날짜 - 날짜	TO_DATE('2009/10/03', 'YYYY/MM/DD') - TO_DATE('2009/10/02', 'YYYY/MM/DD')	1
	TO_DATE('2009/10/02', 'YYYY/MM/DD') - TO_DATE('2009/10/03', 'YYYY/MM/DD')	-1

[실습] HR.EMPLOYEES 테이블로부터, 근무부서(DEPARTMENT_ID 컬럼)가 90인 모든 사원의 성(LAST_NAME 컬럼)과, 입사일(HIRE_DATE 컬럼)부터 오늘까지 근무한 주(WEEK)의 수(소수점 이하 부분은 절삭)로 구성된 레코드를 표시하시오. 단, 출력결과의 머리글은 임의로 지정하시오. DATE 형식 데이터에 산술 연산 실습.

```
SQL> SELECT last_name, TRUNC((SYSDATE-hire_date)/7,0) "Weeks"  
      FROM hr.employees  
     WHERE department_id=90 ;
```

LAST_NAME	Weeks
King	602
Kochhar	484
De Haan	728

☞ 위의 SELECT문의 실행 후에 표시되는 결과는 본 문서와 다를 수 있습니다.

☞ 두 날짜를 더할 수는 없습니다.

```
SQL> SELECT SYSDATE + SYSDATE FROM dual;  
SELECT SYSDATE + SYSDATE FROM dual  
      *  
ERROR at line 1:  
ORA-00975: date + date not allowed
```

4-9. DATE 데이터유형을 기본으로 여러 가지 날짜 함수.

■ 날짜값을 처리하는, 대표적인 오라클 단일 행 문자함수들의 사용방법 및 기능을 요약하면 다음과 같습니다.

문자함수 이름 및 사용방법	기능
ADD_MONTHS('날짜', 숫자)	날짜의 달 수에 숫자를 더한 날짜를 반환합니다.
MONTHS_BETWEEN('날짜1', '날짜2')	두 날짜 사이의 달 수를 반환합니다.
NEXT_DAY('날짜', '요일')	날짜를 기준으로 다음에 오는 요일의 날짜를 반환합니다.
LAST_DAY('날짜')	날짜가 포함된 달의 맨 마지막 날짜를 반환합니다.
ROUND('날짜', '형식 요소')	지정된 형식 요소의 다음 하위요소에서 반올림 된 날짜를 반환합니다.
TRUNC('날짜', '형식 요소')	지정된 형식 요소의 다음 하위요소부터 절삭된 날짜를 반환합니다.

☞ 위의 날짜 함수 사용 시에, 명시되는 날짜값은 반드시 TO_DATE()함수로 처리합니다.

[실습] 다음의 실습을 통해, 문자값을 처리하는 오라클 단일 행 문자함수들의 사용방법을 확인합니다.

1> **ADD_MONTHS('날짜', 숫자)** 함수는, 날짜의 달 수에 숫자를 더한 날짜를 반환합니다.

```
SQL> SELECT ADD_MONTHS(TO_DATE('1994/01/11','YYYY/MM/DD'), 6) FROM dual ;
ADD_MONTHS(TO_DATE('1994/01/11','YYYY/MM/DD'),6)
-----
94/07/11
```

2> **NEXT_DAY ('날짜', '요일')** 함수는, 날짜를 기준으로 다음에 오는 요일의 날짜를 반환합니다.

<pre>SQL> SELECT NEXT_DAY(TO_DATE('08/11/11','RR/MM/DD'), '금요일') FROM dual ;</pre> <pre>NEXT_DAY(TO_DATE('08/11/11','RR/MM/DD'), '금요일') ----- 08/11/14</pre> <pre>SQL> SELECT NEXT_DAY (TO_DATE('08/11/11','RR/MM/DD'), 6) FROM dual ;</pre> <pre>NEXT_DAY(TO_DATE('08/11/11','RR/MM/DD'),6) ----- 08/11/14</pre>	<p>• 2008년 11월 달력</p> <table border="1"> <thead> <tr> <th>일</th> <th>월</th> <th>화</th> <th>수</th> <th>목</th> <th>금</th> <th>토</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1</td> </tr> <tr> <td>2</td> <td>3</td> <td>4</td> <td>5</td> <td>6</td> <td>7</td> <td>8</td> </tr> <tr> <td>9</td> <td>10</td> <td>11</td> <td>12</td> <td>13</td> <td>14</td> <td>15</td> </tr> <tr> <td>16</td> <td>17</td> <td>18</td> <td>19</td> <td>20</td> <td>21</td> <td>22</td> </tr> <tr> <td>23</td> <td>24</td> <td>25</td> <td>26</td> <td>27</td> <td>28</td> <td>29</td> </tr> <tr> <td>30</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>	일	월	화	수	목	금	토														1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30						
일	월	화	수	목	금	토																																																			
						1																																																			
2	3	4	5	6	7	8																																																			
9	10	11	12	13	14	15																																																			
16	17	18	19	20	21	22																																																			
23	24	25	26	27	28	29																																																			
30																																																									

☞ 2 번째 실습 문장은, 접속 세션에서 사용하는 언어가 다르면, 요일을 언어로 명시했을 때, 오류가 발생되므로, NEXT_DAY() 함수에 요일을 언어가 아닌 숫자로 표시한 예제입니다.

☞ 오라클 데이터베이스 서비스에서, 각 요일에 대하여, 다음처럼, 숫자가 지정됩니다.

일요일은 1번, 월요일은 2번, 화요일은 3번, 수요일은 4번, 목요일은 5번, 금요일은 6번, 토요일은 7번 입니다.

3> **MONTHS_BETWEEN ('날짜1', '날짜2')** 함수는, 두 날짜 사이의 달 수를 정확히 알려 줍니다.

```
SQL> SELECT MONTHS_BETWEEN( TO_DATE('1995/09/01','YYYY/MM/DD'),
                                TO_DATE('1994/01/11','YYYY/MM/DD') ) AS RESULT
          FROM dual ;

RESULT
-----
19.6774194 --두 날짜 사이의 달수 및 일 수에 대한 차이도 계산되어 소수로 표시됩니다.
```

4> **LAST_DAY ('날짜')** 함수는, 날짜가 포함된 달의 맨 마지막 날짜를 반환합니다.

```
SQL> SELECT LAST_DAY (TO_DATE('2000-02-15','YYYY-MM-DD')) FROM dual ;

LAST_DAY(TO_DATE('2000-02-15','YYYY-MM-DD'))
-----
00/02/29
```

☞ 2000년 2월 15일 이 포함된 달의 마지막 날(말일)은 2000년 02월 29일입니다.

☞ LAST_DAY() 함수에 표시하는 날짜는 1일부터 28일 사이의 날짜를 사용하십시오. 29일, 30일, 31일을 사용하는 경우, 다음과 같은 오류가 발생될 수 있습니다.

```
SQL> SELECT LAST_DAY (TO_DATE('1995-02-29','YYYY-MM-DD')) FROM dual ;
명령의 1 행에서 시작하는 중 오류 발생 -
SELECT LAST_DAY (TO_DATE('1995-02-29','YYYY-MM-DD')) --1995년 2월은 28일까지만 있고, 없는 날짜인 29일을 함수에
      FROM dual --사용하였기 때문에 오류가 발생되었습니다.

오류 보고 -
SQL 오류: ORA-01839: 지정된 월에 대한 날짜가 부적합합니다
01839. 00000 - "date not valid for month specified"
*Cause:
*Action:
```

5> **TRUNC('날짜', '유효 형식 요소')** 함수는, 날짜를, 명시된 형식요소의 다음 하위요소부터 절삭된 날짜를 반환합니다.

유효 형식 요소가 생략된 경우, 날짜는 가장 가까운 일로 절삭됩니다.

```
SQL> SELECT SYSDATE,
           TRUNC(SYSDATE, 'YEAR') T_YEAR,
           TRUNC(SYSDATE, 'MONTH') T_MONTH,
           TRUNC(SYSDATE, 'DD') T_DD1,
           TRUNC(SYSDATE) T_DD2
          FROM dual;

SYSDATE   R_YEAR   R_MONTH   R_DD1   R_DD2
-----
07/11/17 07/01/01 07/11/01 07/11/17 07/11/17
```

--서버가 실행 중인 운영체제의 날짜가 표시됩니다.
--연도를 유효한 단위로 설정하여, 월(MONTH)에서 절삭됩니다.
--월을 유효한 단위로 설정하여, 일(DD)에서 절삭됩니다.
--일을 유효한 단위로 설정하여, 시(HH)에서 절삭됩니다.
--형식요소가 생략되면, 기본으로 DD 요소가 사용됩니다.

☞ 유효 형식 요소로서, HH(시), MI(분), DAY(요일)도 사용될 수 있습니다.

6> **ROUND('날짜', '유효 형식 요소')** 함수는, 날짜를, 명시된 형식 요소의 다음 하위요소에서 반올림 된 날짜를 반환합니다. 유효 형식 요소가 생략된 경우, 날짜는 가장 가까운 일로 반올림됩니다.

```
SQL> SELECT SYSDATE,
          ROUND(SYSDATE, 'YEAR') R_YEAR,
          ROUND(SYSDATE, 'MONTH') R_MONTH,
          ROUND(SYSDATE, 'DD') R_DD1,
          ROUND(SYSDATE) R_DD2
     FROM dual;
```

SYSDATE	R_YEAR	R_MONTH	R_DD1	R_DD2	
07/05/07	07/01/01	07/05/01	07/05/07	07/05/07	-- 각 유효 날짜단위의 하위단위가 반을 넘지 않았을 때,
07/11/17	08/01/01	07/12/01	07/11/18	07/11/18	-- 각 유효 날짜단위의 하위단위가 반을 넘었을 때,

☞ 유효 형식 요소로서, HH(시), MI(분), DAY(요일)도 사용될 수 있습니다.

4-10. 일반 함수(GENERAL FUNCTION)와 CASE 표현식

- 오라클 일반 함수는, 처리하는 값의 데이터유형에 제한 받지 않습니다.
- CASE 표현식은, 대부분의 RDBMS 제품들에서 지원되며, IF-THEN-ELSE 로직을 통해, 조건부 조회를 수행합니다.
- 대표적인 오라클 단일 행 일반함수들의 사용방법 및 기능을 요약하면 다음과 같습니다.

문자함수 이름 및 사용방법	기능
NVL(컬럼표현식, 대체값_표현식)	컬럼표현식에 값이 있으면, 그 값을 반환하고, 컬럼표현식이 NULL이면, 대체값_표현식의 값을 반환합니다.
DECODE() 함수	단순 CASE 표현식과 기능이 동일하며, IF-THEN-ELSE 로직을 수행하여 조건부 조회를 편리하게 수행합니다.
NVL2(컬럼표현식, 대체값_표현식1, 대체값_표현식2)	컬럼표현식에 값이 있으면, 대체값_표현식1의 값을 반환하고, 컬럼표현식이 NULL이면, 대체값_표현식2의 값을 반환합니다.
NULLIF(표현식1, 표현식2)	표현식1과 표현식2를 비교하여, 두 표현식의 값이 같으면, NULL이 반환되고, 값이 다르면, 표현식1의 값이 반환됩니다. 단, 표현식1에 NULL 리터럴을 지정할 수 없습니다.
COALESCE(표현식1,표현식2,...,표현식n)	명시된 리스트에서 NULL이 아닌 첫 번째 표현식의 값을 반환합니다.

4-10-1. CASE 표현식

■ IF-THEN-ELSE 로직을 수행하여 조건부 조회를 편리하게 수행하도록 합니다.

■ CASE 표현식은 대부분의 RDBMS 제품들에서 지원됩니다.

■ CASE 표현식에는 다음 두 가지 표현식이 있습니다.

○ 단순 CASE 표현식(Basic CASE Expression)

○ 검색된 CASE 표현식(Searched CASE Expression)

■ CASE 표현식 기본 문법

○ 단순 CASE 표현식

<pre>CASE 컬럼표현식 WHEN 비교값1 THEN 반환값_표현식1 WHEN 비교값2 THEN 반환값_표현식2 ... WHEN 비교값n THEN 반환값_표현식n ELSE 기본_반환값_표현식 END</pre>	--컬럼표현식과 비교값은 데이터유형이 동일해야 합니다. --컬럼표현식은, 함수나 연산자로 처리된 컬럼입니다. --반환값_표현식은, 결과에 표시될 값입니다. --비교값에 NULL을 지정할 수 없습니다. --모든 반환값_표현식은 데이터유형이 동일해야 합니다.
---	--

☞ 단순 CASE 표현식은 다음처럼 작동됩니다.

- 컬럼표현식의 값이 명시된 WHEN 절의 비교값과 순서대로 비교됩니다.
- 컬럼표현식의 값과 동일한 비교값에 해당하는 첫 번째 WHEN-THEN 절의 반환값_표현식의 값이 반환됩니다.
- 컬럼표현식의 값과 동일한 비교값에 해당하는 WHEN-THEN 절이 없는 경우,
 - ELSE 절이 명시되어 있으면, ELSE 절에 명시된 기본_반환값_표현식의 값이 반환됩니다.
 - ELSE 절이 생략되면, 해당 행에 대하여 NULL이 반환됩니다.

○ 검색된 CASE 표현식

<pre>CASE WHEN 조건1 THEN 반환값_표현식1 WHEN 조건2 THEN 반환값_표현식2 ... WHEN 조건n THEN 반환값_표현식n ELSE 기본_반환값_표현식 END</pre>	--WHEN-THEN 절의 조건은 서로 중복되지 않아야 합니다. --모든 반환값_표현식은 데이터유형이 동일해야 합니다.
--	---

☞ 검색된 CASE 표현식은 다음처럼 작동됩니다.

- WHEN 절에 명시된 조건이 순서대로 평가됩니다
- 첫 번째 조건을 만족하는 WHEN-THEN 절의 반환값_표현식의 값이 반환됩니다.
- WHEN 절의 어떤 조건도 만족하는 WHEN-THEN 절이 없는 경우,
 - ELSE 절이 명시되어 있으면, ELSE 절에 명시된 기본_반환값_표현식의 값이 반환됩니다.
 - ELSE 절이 생략되면, 해당 행에 대하여 NULL이 반환됩니다.

- [실습] HR.EMPLOYEES 테이블로부터, 사원의 성(LAST_NAME 컬럼), 직책코드(JOB_ID 컬럼), 급여(SALARY 컬럼) 및 인상된 급여로 구성된 레코드의 데이터를 표시하시오. 단, 인상된 급여는, 다음과 같이 계산됩니다.
- 직책코드가 IT_PROG 이면, 급여가 10% 증가됩니다.
 - 직책코드가 ST_CLERK 이면, 급여가 15% 증가됩니다.
 - 직책코드가 SA REP 이면 급여가 20% 증가됩니다.
 - 다른 모든 직책코드에 대해서는 급여 인상이 없습니다.

```
SQL> SELECT last_name, job_id, salary, (CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
                                              WHEN 'ST_CLERK' THEN 1.15*salary
                                              WHEN 'SA REP' THEN 1.20*salary
                                              ELSE salary
                                         END) AS "REVISED_SALARY"
      FROM hr.employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY	---컬럼의 값과 상수값이 동일한만 확인하므로, ---단순 CASE 표현식을 사용하였습니다.
OConnell	SH_CLERK	2600	2600	
Grant	SH_CLERK	2600	2600	
Whalen	AD_ASST	4400	4400	
Hartstein	MK_MAN	13000	13000	
Fay	MK_REP	6000	6000	
...				
Hunold	IT_PROG	9000	9900	
Ernst	IT_PROG	6000	6600	
Austin	IT_PROG	4800	5280	
...				
Nayer	ST_CLERK	3200	3680	
Mikkilineni	ST_CLERK	2700	3105	
Landry	ST_CLERK	2400	2760	
...				
Tucker	SA REP	10000	12000	
Bernstein	SA REP	9500	11400	
Hall	SA REP	9000	10800	
...				
107개의 행이 선택됨				

☞ JOB_ID 컬럼의 값에 따라, 변경된 급여가 다르게 표시되도록 단순 CASE 표현식이 사용되었으며,

위의 SELECT 문에 사용된 CASE 표현식은 각 행에 대하여 다음처럼 작동합니다.

- JOB_ID 컬럼 값이 'IT_PROG' 이면, 1.10*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG' 가 아니고, 'ST_CLERK' 이면, 1.15*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG' 및 'ST_CLERK'가 아니고, 'SA REP'이면, 1.10*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG', 'ST_CLERK' 및 'SA REP'가 모두 아니면, SALARY를 표시합니다.

[실습] HR.EMPLOYEES 테이블로부터, 사원의 성(LAST_NAME 컬럼), 급여(SALARY 컬럼) 및 급여에 따른 등급으로 구성된 레코드의 데이터를 표시하시오. 단, 급여에 따른 등급은, 급여가 5000 미만이면 Low, 급여가 5000 보다 크거나 같고 1000 미만이면 Medium, 급여가 10000 보다 크거나 같고 20000 미만이면 Good, 급여가 20000 보다 크거나 같으면 Excellent가 각각 표시되도록 하시오.

```
SQL> SELECT last_name, salary, (CASE WHEN salary < 5000 THEN 'Low'
                                         WHEN salary < 10000 THEN 'Medium'
                                         WHEN salary < 20000 THEN 'Good'
                                         ELSE 'Excellent'
                                     END) AS "QUALIFIED_SALARY"
   FROM hr.employees;
```

LAST_NAME	SALARY	QUALIFIED_SALARY
OConnell	2600	Low
Grant	2600	Low
Whalen	4400	Low
Hartstein	13000	Good
Fay	6000	Medium
Mavris	6500	Medium
Baer	10000	Good
Higgins	12008	Good
Gietz	8300	Medium
King	24000	Excellent
Kochhar	17000	Good
...		

--WHEN-THEN 절의 조건에서 '<' 대신, '>'을 사용하면,
--범위가 겹쳐서 항상 Low 만 표시됩니다.
--따라서, 검색된 CASE 표현식은 WHEN 절에 명시된
--조건의 범위나 값이 중복되지 않도록 조심합니다

107개의 행이 선택됨

- ☞ 급여가 특정 범위에 포함되는 유무에 따라 등급이 다르게 표시되도록, 조건을 사용할 수 있는 검색된 CASE 표현식이 사용되었으며, 위의 SELECT 문에 사용된 CASE 표현식은 각 행에 대하여 다음처럼 작동합니다.
- SALARY 컬럼 값이 5000 보다 작으면, 'Low'가 표시됩니다.
 - SALARY 컬럼 값이 5000 보다 크거나 같고, 10000 보다 작으면, 'Medium'이 표시됩니다.
 - SALARY 컬럼 값이 10000 보다 크거나 같고, 20000 보다 작으면, 'Good'이 표시됩니다.
 - SALARY 컬럼 값이 20000 보다 크거나 같으면, 'Excellent'가 표시됩니다.

4-10-2. 일반 함수의 활용

- 일반함수는, 함수가 처리하는 값의 데이터유형에 제한 받지 않습니다.

```
SELECT last_name, salary, commission_pct,
       (salary*12 + salary*
        (CASE WHEN commission_pct IS NULL THEN 0
              ELSE commission_pct END)*12) "ANN_SAL2"
  FROM hr.employees;
```

4-10-2-1. NVL() 함수와 NVL2() 함수

- NVL(컬럼표현식, 대체값_표현식) 함수는, 컬럼표현식에 값이 있으면, 그 값을 반환하고, 컬럼표현식이 NULL이면, 대체값_표현식의 값을 반환합니다.
- NVL2(컬럼표현식, 대체값_표현식1, 대체값_표현식2) 함수는, 컬럼표현식에 값이 있으면, 대체값_표현식1의 값을 반환하고, 컬럼표현식이 NULL이면, 대체값_표현식2의 값을 반환합니다.
- NVL 함수에서, 컬럼표현식과 대체값_표현식의 데이터유형은 반드시 일치해야 합니다.
- NVL2 함수에서, 대체값_표현식1과 대체값_표현식2의 데이터유형은 반드시 일치해야 합니다.

[실습] HR.EMPLOYEES 테이블로부터, 사원의 성(LAST_NAME 컬럼), 급여(SALARY 컬럼), 커미션율(COMMISSION_PCT 컬럼), 연봉으로 구성된 레코드를 표시하시오. 단, 연봉에는 급여 및 커미션율을 모두 고려하시오. 커미션을 지급받지 못하는 사원에 대해서는 커미션율이 없습니다(NULL).

CASE .

```
SQL> SELECT last_name, salary, commission_pct, salary*12+salary*commission_pct*12 ANN_SAL1,
      (salary*12 + salary*NVL(commission_pct, 0)*12) "ANN_SAL2"
    FROM hr.employees;
```

LAST_NAME	SALARY	COMMISSION_PCT	ANN_SAL1	ANN_SAL2
King	24000			288000
Kochhar	17000			204000
...				--산술연산이 수행되는 표현식의 컬럼이
Russell	14000	.4	235200	235200 --NULL이면, 표현식의 결과는 NULL입니다
Partners	13500	.3	210600	210600
...				
107개의 행이 선택됨				

☞ 연봉을 계산하는 표현식에 COMMISSION_PCT 컬럼 대신, NVL(COMMISSION_PCT,0)가 사용되어, COMMISSION_PCT 컬럼이 NULL인 사원에 대해서도 연봉이 표시되었습니다.

☞ 만약, 위에서 연봉표현식을 NVL 함수로 처리하지 않으면(ANN_SAL1), 커미션을 받는 사원들만 연봉이 표시되고, 커미션을 받지 않는 사원은 연봉이 표시되지 않습니다(NULL).

[실습] HR.EMPLOYEES 테이블로부터, 20 및 80 부서에 근무하는(DEPARTMENT_ID 컬럼) 사원에 대하여, 사원의 성(LAST_NAME 컬럼), 커미션율(COMMISSION_PCT 컬럼), 및 커미션의 수령여부를 확인할 수 있는 값으로 구성된 레코드를 표시하시오. 단, 커미션수령여부는 커미션을 받으면, '커미션_수령'이 표시되고, 커미션을 받지 않으면, '커미션_미수령'이 표시되도록 하시오.

```
SQL> SELECT last_name, commission_pct,
      NVL2(commission_pct, '커미션_수령', '커미션_미수령') AS "커미션수령여부"
    FROM hr.employees
   WHERE department_id IN (20,80);

LAST_NAME           COMMISSION_PCT 커미션수령여부
-----              -----
Hartstein          커미션_미수령
Fay                커미션_미수령
Russell            .4 커미션_수령
Partners           .3 커미션_수령
Errazuriz          .3 커미션_수령
Cambrault          .3 커미션_수령
...
36개의 행이 선택됨
```

```
SELECT last_name, commission_pct,
      (CASE WHEN commission_pct is null
            THEN ''
            ELSE '' END) AS ""
    FROM hr.employees
   WHERE department_id IN (20,80);
```

4-10-2-2. DECODE() 함수

■ DECODE() 함수는, 단순 CASE 표현식과 기능이 동일하며, IF-THEN-ELSE 로직을 통해 조건부 조회를 수행합니다.

■ 기본문법: DECODE() 함수

DECODE(컬럼표현식,	
비교값1, 반환값_표현식1,	--컬럼표현식과 비교값은 데이터유형이 동일해야 합니다.
비교값2, 반환값_표현식2,	--컬럼표현식은, 함수나 연산자로 처리된 컬럼입니다.
...	--반환값_표현식은, 결과에 표시될 값입니다.
비교값n, 반환값_표현식n,	--모든 반환값_표현식은 데이터유형이 동일해야 합니다.
기본_반환값_표현식)	--DECODE() 함수는, 비교값에 NULL을 지정할 수 있습니다.

■ DECODE() 함수는 다음처럼 작동됩니다.

- 컬럼표현식의 값이 명시된 비교값과 순서대로 비교됩니다.
- 컬럼표현식의 값과 동일한 비교값에 해당하는 첫 번째 반환값_표현식의 값이 반환됩니다.
- 만약, 컬럼표현식의 값과 동일한 비교값이 없는 경우,
 - 기본_반환값_표현식이 명시되어 있으면, 기본_반환값_표현식의 값이 반환됩니다.
 - 기본_반환값_표현식이 없으면, 해당 행에 대하여 NULL이 반환됩니다.

[실습] HR.EMPLOYEES 테이블로부터, 80번 부서에 근무하는 사원의 성(LAST_NAME 컬럼), 급여(SALARY 컬럼) 및 급여에 따른 세율을 표시하시오. 단, 급여에 따른 세율은 다음과 같습니다.

- 월 급여 범위가 0.00 ~ 2,999.99 이면, 세율은 0% 입니다.
- 월 급여 범위가 3,000.00 ~ 5,999.99 이면, 세율은 10% 입니다.
- 월 급여 범위가 6,000.00 ~ 8,999.99 이면, 세율은 20% 입니다.
- 월 급여 범위가 9,000.00 ~ 11,999.99 이면, 세율은 30% 입니다.
- 월 급여 범위가 12,000.00 ~ 14,999.99 이면, 세율은 40% 입니다.
- 월 급여 범위가 15,000.00 이상이면, 세율은 45% 입니다.

```
SQL> SELECT last_name, salary,
      DECODE (TRUNC(salary/3000), 0, 0.00, 1, 0.10, 2, 0.20, 3, 0.30, 4, 0.40, 0.45) "세율"
    FROM hr.employees
   WHERE department_id = 80;
```

LAST_NAME	SALARY	세율	설명
Russell	14000	.4	--세율 대신, 지불되는 세금이나 또는 세금이 공제된 실수령 급여를 표시하는 SQL문도 스스로 작성해 보시기 바랍니다. 또한
Partners	13500	.4	--DECODE() 함수를 CASE 표현식으로 바꿔서 SQL문을 작성해 보시기 바랍니다.
Errazuriz	12000	.4	CASE TRUNC(salary/3000) WHEN 0 THEN 0.00 WHEN 1 THEN 0.10 WHEN 2 THEN 0.20 WHEN 3 THEN 0.30 WHEN 4 THEN 0.40 ELSE 0.45 END
Cambrault	11000	.3	
Zlotkey	10500	.3	
Tucker	10000	.3	
Bernstein	9500	.3	
Hall	9000	.3	
Olsen	8000	.2	
...			
34개의 행이 선택됨			

[실습] HR.EMPLOYEES 테이블로부터, 사원의 성(LAST_NAME 컬럼), 직책코드(JOB_ID 컬럼), 급여(SALARY 컬럼) 및 인상된 급여로 구성된 레코드의 데이터를 표시하시오. 단, 인상된 급여는, 다음과 같이 계산됩니다.

- 직책코드가 IT_PROG 이면, 급여가 10% 증가되고, 직책코드가 ST_CLERK 이면, 급여가 15% 증가되고, 직책코드가 SA REP 이면 급여가 20% 증가됩니다. 그리고, 다른 모든 직책코드에 대해서는 급여 인상이 없습니다.

```
SQL> SELECT last_name, job_id, salary,
      DECODE(job_id, 'IT_PROG', 1.10*salary, 'ST_CLERK', 1.15*salary,
             'SA REP', 1.20*salary, salary ) AS "REVISED_SALARY"
    FROM hr.employees;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
OConnell	SH_CLERK	2600	2600
...			
Hunold	IT_PROG	9000	9900
...			
Nayer	ST_CLERK	3200	3680
...			
107개의 행이 선택됨			

☞ JOB_ID 컬럼의 값에 따라, 변경된 급여가 다르게 표시되도록, DECODE() 함수가 사용되었으며,

위의 SELECT 문에 사용된 DECODE() 함수는, 각 행에 대하여 다음처럼 작동합니다.

- JOB_ID 컬럼 값이 'IT_PROG' 이면, 1.10*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG' 가 아니고, 'ST_CLERT' 이면, 1.15*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG' 및 'ST_CLERT'가 아니고, 'SA REP'이면, 1.10*SALARY 값이 표시됩니다.
- JOB_ID 컬럼 값이 'IT_PROG', 'ST_CLERT' 및 'SA REP'가 모두 아니면, SALARY를 표시합니다.

4-10-2-3. NULLIF() 함수

■ NULLIF(표현식1, 표현식2) 함수는, 표현식1과 표현식2를 비교하여, 두 표현식의 값이 같으면, NULL이 반환되고, 값이 다르면, 표현식1의 값이 반환됩니다. 단, 표현식1에 NULL 리터럴을 지정할 수 없습니다.

■ 표현식1 및 표현식2의 데이터유형이 일치해야 합니다.

[실습] HR.EMPLOYEES 테이블로부터, 10 및 20 부서에 근무하는(DEPARTMENT_ID 컬럼) 사원에 대하여, 사원의 이름(FIRST_NAME 컬럼), 사원의 성(LAST_NAME 컬럼), 이름의 글자수, 성의 글자수, 및 이름의 글자수와 성의 글자수가 같으면 1을, 다르면 이름의 글자수가 반환되는 값들로 구성된 레코드를 표시하시오.

```
SQL> SELECT first_name,
           last_name,
           LENGTH(first_name) F_NAME_LENGTH,
           LENGTH(last_name) L_NAME_LENGTH,
           NULLIF(LENGTH(first_name), LENGTH(last_name)) COMP_LENGTH
      FROM hr.employees
     WHERE department_id IN (10,20);
```

FIRST_NAME	LAST_NAME	F_NAME_LENGTH	L_NAME_LENGTH	COMP_LENGTH
Jennifer	Whalen	8	6	8
Michael	Hartstein	7	9	7
Pat	Fay	3	3	3

```
SELECT first_name, last_name, LENGTH(first_name) F_NAME_LENGTH,
       LENGTH(last_name) L_NAME_LENGTH,
       (CASE WHEN LENGTH(first_name) = LENGTH(last_name)
             THEN NULL
             ELSE LENGTH(first_name) END) AS COMP_LENGTH
  FROM hr.employees
 WHERE department_id IN (10,20);
```

4-10-2-4. COALESCE() 함수

■ COALESCE(표현식1, 표현식2, ... 표현식n) 함수는, 명시된 리스트에서 NULL이 아닌 첫 번째 표현식의 값을 반환합니다.

■ COALESCE() 함수가 테이블의 행을 처리할 때 작동방식은 다음과 같습니다.

- 표현식1이 NULL이 아니면, 표현식1의 값을 반환하고 실행이 종료됩니다.
- 표현식1이 NULL이고, 표현식2가 NULL이 아니면, 표현식2의 값을 반환하고 실행이 종료됩니다.
- 앞의 모든 표현식이 NULL이고 표현식n이 NULL이 아니면, 표현식n의 값을 반환하고 실행이 종료됩니다.
- 모든 표현식이 NULL이면 NULL을 반환하고 실행이 종료됩니다.

■ COALESCE() 함수에 명시되는 모든 표현식의 데이터유형은 일치되어야 합니다. 만약, 표현식들의 데이터유형이 일치하지 않으면, 다음과 같은 오류가 발생됩니다.

```
SQL> SELECT last_name, COALESCE(last_name, commission_pct, -1) RESULT FROM hr.employees;
ORA-00932: 일관성 없는 데이터유형: CHAR01(가) 필요하지만 NUMBER임
00932. 00000 - "inconsistent datatypes: expected %s got %s"
*Cause:
*Action:
1행, 39열에서 오류 발생 -- last_name이 문자유형이지만, commission_pct가 숫자유형이므로 오류가 발생됨.
```

[실습] COALESCE ()함수 기능 이해를 위한 실습

1> COALESCE ()함수 테스트를 위해 HR.COALESCE_TEST 테이블을 생성하고, 필요한 데이터를 입력합니다.

```
CREATE TABLE HR.COALESCE_TEST (
    EID NUMBER(4) PRIMARY KEY,
    ENAME VARCHAR2(30),
    MGRID NUMBER(4),
    COMM  NUMBER (3,2) ;
    --COALESCE () 함수 테스트를 위한 데이터입력 및 트랜잭션 커밋
    INSERT INTO HR.COALESCE_TEST VALUES (200, '홍길동', 100, 0.1);   --NULL인 컬럼 없음
    INSERT INTO HR.COALESCE_TEST VALUES (201, '홍길순', NULL, 0.2);  --MGRID 컬럼만 NULL
    INSERT INTO HR.COALESCE_TEST VALUES (202, '홍길용', NULL, NULL); --MGRID, COMM 컬럼이 모두 NULL
    COMMIT;                --입력 후의 상태를 유지하면서, 데이터 입력작업을 종료.
```

2> HR.COALESCE_TEST 테이블에 입력된 모든 데이터를 확인합니다.

```
SQL> SELECT * FROM HR.COALESCE_TEST ;
```

EID	ENAME	MGRID	COMM
200	홍길동	100	.1
201	홍길순		.2
202	홍길용		

3> HR.COALESCE_TEST 테이블을 이용하여, EID 컬럼, ENAME 컬럼, 및 MGRID 컬럼과, COMM 컬럼 순서로 확인하여, 최초로 NULL 이 아닌 컬럼의 값을 표시하되, 두 컬럼이 모두 널일 경우, '매니저와 커미션이 없음' 으로 표시되는 RESULT로 구성되는 레코드의 데이터를 표시하시오.

```
SQL> SELECT eid, ename,
      COALESCE(TO_CHAR(mgrid), TO_CHAR(comm), '매니저와 커미션이 없음') "RESULT"
    FROM hr.coalesce_test ;
```

EID	ENAME	RESULT
200	홍길동	100
201	홍길순	.2
202	홍길용	매니저와 커미션이 없음

--COALESCE() 함수에 명시된 표현식들의 데이터유형을 일치시키기 위하여
TO_CHAR() 함수가 사용되었습니다.

4> 실습에 사용한 테이블을 삭제합니다.

```
SQL> DROP TABLE HR.COALESCE_TEST ;
```

Table HR.COALESCE_TEST이(가) 삭제되었습니다.

[실습] HR.Employees 테이블로부터, 사원의 성(LAST_NAME 컬럼), 현재급여(SALARY 컬럼), 커미션율(COMMISSION_PCT 컬럼), 인상된 급여액으로 구성된 레코드의 데이터를 표시하시오.

단, 인상된 급여는 다음처럼 계산됩니다.

- 커미션을 받지 않는 사원에게는 기존 급여에 2000을 더함.
- 커미션을 받는 사원에게는 기존 급여에 커미션이 추가됨.

```
SQL> SELECT last_name, commission_pct, salary AS "OLD SALARY",
      COALESCE((salary+(salary*commission_pct)), (salary+2000), 0) AS "NEW SALARY"
    FROM hr.employees ;
```

LAST_NAME	COMMISSION_PCT	OLD SALARY	NEW SALARY
Patel		2500	4500
Rajs		3500	5500
Davies		3100	5100
Matos		2600	4600
Vargas		2500	4500
Russell	.4	14000	19600
Partners	.3	13500	17550
Errazuriz	.3	12000	15600
Cambrault	.3	11000	14300
Zlotkey	.2	10500	12600

--커미션 수령여부에 따라 다르게 인상된 급여가
--레코드에서 동일한 컬럼에 표시되어야 합니다.

--이를 위하여 COALESCE 함수를 사용합니다.

--커미션을 받는 사원들은 계산은 정상적으로 수행되지만
--커미션을 받지 않는 사원들은 COMMISSION_PCT 컬럼이
--포함된 산술표현식이 NULL이므로, 커미션을 받는 사원의
--새로운 급여를 계산하는 표현식을 첫 번째에 위치시키고,

--커미션을 받지 않는 사원은 첫 번째 표현식이 NULL이므로,
--두 번째 위치에 새로운 급여 계산을 위치 시킵니다.

--맨 끝의 0은 첫 번째, 두 번째 표현식이
--모두 NULL 일 때를 고려하여 추가한 것입니다.

107개의 행이 선택됨

5 다중 행 함수(MULTIPLE-ROW FUNCTION, 그룹함수, GROUP FUNCTION).

◆ 학습 목표.

- SQL에서 사용할 수 있는 다음과 같은 오라클 그룹함수의 사용법에 대하여 학습합니다.
 - SUM
 - AVG
 - MAX
 - MIN
 - COUNT
- GROUP BY 절을 사용하여, 더 작은 집합으로 그룹화된 결과를 산출하는 방법에 대하여 학습합니다.
- HAVING 절을 사용하여, 그룹화된 결과로부터, 필요한 결과만 선택하여 표시하는 방법을 학습합니다.

5-1. 그룹함수(GROUP FUNCTION) 개요.

- 그룹함수는, 기본적으로 선택된 행들을 하나의 그룹으로 그룹화하여, 그룹화된 행들로부터 가져온 값의 집합에 대하여, 집합연산(합계, 평균 등)을 수행하여, 하나의 처리 결과를 표시하는 함수입니다.

- 기본문법: 그룹함수 사용 형식

그룹함수(컬럼표현식) --선택된 행들로부터 반환된 컬럼표현식의 값을 모두를,

== 그룹함수(ALL 컬럼표현식) --한 번에 그룹함수로 처리합니다.

그룹함수(DISTINCT 컬럼표현식) --선택된 행들로부터 반환된 컬럼표현식의 값을 중, 동일한 값을

--한 번씩만 고려하여, 그룹함수로 처리합니다.

☞ 컬럼표현식 앞에 ALL 및 DISTINCT 키워드를 명시하지 않으면, ALL 키워드가 디폴트로 사용됩니다.

☞ 선택된 행의 컬럼표현식이 NULL을 반환하면, 해당 행들은 무시되며, 그룹함수 처리에서 고려되지 않습니다.

☞ 만약, 그룹함수가 NULL을 무시하지 않도록, 값으로 치환하려면, 컬럼표현식을 NVL(), NVL2(), COALESCE(), DECODE() 함수 또는 CASE 표현식으로 처리합니다.

- 다음은, 본 단원에서 학습하는 대표적인 그룹함수들입니다.

그룹함수	기능
SUM 및 AVG	각각 합계 및 평균을 반환하며, NUMBER 데이터유형에 대하여 사용됩니다.
MAX 및 MIN	각각 최대값 및 최소값을 반환하며, LONG 및 CLOB 데이터유형을 제외한 문자, 숫자, 날짜 데이터유형에 대하여 사용됩니다.
COUNT	개수를 반환하며, 모든 데이터유형에 대하여 사용됩니다.
COUNT(*)	행 카운트 함수라고 부르며, 중복 행 및 컬럼에 NULL이 포함된 행을 포함하여, 선택된 모든 행의 개수를 반환합니다.

☞ CLOB는 최대 (4GB-1)*(LOB 스토리지의 CHUNK 매개변수 설정값) 바이트 길이까지 가능한 문자 데이터유형이며, LONG 데이터유형은 최대 2GB 길이까지 가능한 문자 데이터유형입니다. 이 두 데이터유형은, 값의 길이가 너무 크기 때문에, MAX, 및 MIN 함수에 대하여 사용할 수 없습니다.

☞ 오라클 데이터베이스 서버에서 제공하는, 모든 그룹함수들의 목록 및 각 그룹함수들에 대한 자세한 내용은 Oracle Database SQL Language Reference 매뉴얼을 참고하세요.

5-2. 그룹함수에서 ALL, DISTINCT 옵션의 의미 및 NULL 무시에 대한 설명

- 그룹함수에 옵션으로 DISTINCT 또는 ALL 키워드를 사용할 수 있습니다. 또한 그룹함수 실행 시 데이터가 없으면(즉, NULL 상태의 필드), 그 필드는 없는 것으로 간주합니다.
- 아래와 같은 데이터가 저장된 [scores 테이블]을 가지고 그룹함수 사용방법을 설명합니다. 여포의 SCORE 컬럼은 NULL 상태입니다.

STU_NO	NAME	SCORE
01	관우	50
02	유비	50
03	장비	100
04	조조	100
05	여포	

- 위의 테이블에 대하여 다음의 SELECT문을 수행하여 점수의 합계를 구한다고 합시다.

```
SELECT SUM(score), SUM(ALL score), SUM(DISTINCT score) FROM scores;
```

- 위의 문장에서 SELECT 절에 명시된 각 그룹함수 처리 결과는 다음과 같습니다.

사용된 그룹함수 표현식	그룹함수에 의하여 처리 후 표시되는 값
SUM(score) = SUM(ALL score)	300
SUM(DISTINCT score)	150

- ALL 키워드를 사용하면, 합계 시에 해당 컬럼의 모든 값이 처리됩니다.
- DISTINCT 키워드를 사용하면, 합계 시에 같은 값들은 한 번만 처리됩니다(대표값 합계).
- ALL이나 DISTINCT를 명시하지 않으면, ALL 방식으로 처리됩니다.

- 위의 테이블에 대하여 다음의 SELECT문으로 점수의 평균 및 개수를 구한다고 합시다.

```
SELECT AVG(score), AVG(NVL(score, 0)), COUNT(score), COUNT(NVL(score, 0)), COUNT(*)
FROM scores ;
```

- 위의 문장에서 SELECT 절에 명시된 각 그룹함수 처리 결과는 다음과 같습니다.

사용된 그룹함수 표현식	그룹함수에 의하여 처리 후 표시되는 값
AVG(score)	300/4
AVG(NVL(score, 0))	300/5
COUNT(score)	4
COUNT(NVL(score, 0)) = COUNT(*)	5

- 그룹함수 처리 시에 해당 컬럼에 데이터가 없는(NULL 상태임) 경우, 그 행은 무시하고 고려하지 않습니다.
- 그룹함수 처리 시에 NULL 상태의 필드도 그룹함수 처리에 포함해야 한다면, NVL() 함수로 해당 필드에 NULL을 대신할 적절한 값을 지정하여 그룹함수가 처리하도록 해줍니다.

5-3. 그룹함수 사용 실습.

[실습] HR.EMPLOYEES 테이블로부터, 사원의 입사일 중, 가장 오래된 입사일과 가장 최근의 입사일을 표시하시오.

```
SQL> SELECT MIN(hire_date), MAX(hire_date) FROM hr.employees;
```

```
MIN(HIRE_DATE) MAX(HIRE_DATE)
----- -----
```

```
01/01/13      08/04/21
```

[실습] HR.EMPLOYEES 테이블로부터, 회사의 모든 사원수 및 커미션을 지급받는 사원수를 표시하시오.

```
SQL> SELECT COUNT(*) ALL_PERSONS, COUNT(commission_pct) COMM_PERSONS FROM hr.employees;
```

```
ALL_PERSONS COMM_PERSONS      --COUNT(*) 함수에는 ALL 및 DISTINCT 키워드를 사용할 수 없습니다.
----- -----
```

```
107          35
```

[실습] HR.EMPLOYEES 테이블로부터, 사원이 근무하고 있는 부서의 개수 및 근무 부서가 지정된 사원의 수를 구하시오.

```
SQL> SELECT COUNT(DISTINCT department_id) DEPTS, COUNT(ALL department_id) DEPT_PERSONS
   FROM hr.employees;
```

```
DEPTS DEPT_PERSONS
----- -----
```

```
11          106      --department_id 컬럼의 값이 NULL 상태인 EMPLOYEES 테이블의 한 행이 제외됩니다.
```

[실습] HR.EMPLOYEES 테이블로부터, 커미션을 지급받는 사원들만 고려하여 commission의 평균을 구하고, 전체 사원을 대상으로 한(커미션을 받는 사원 및 받지 않는 사원을 모두 포함한 전체 사원) 커미션의 평균을 구하시오.

```
SQL> SELECT AVG(commission_pct) COMM_AVG_COMM_PERSONS,
           AVG(NVL(commission_pct, 0)) AS COMM_AVG_ALL_PERSONS
      FROM hr.employees;
```

```
COMM_AVG_ONLY_COMM_PERSONS COMM_AVG_ALL_PERSONS
----- -----
```

```
.222857143      .0728971963
```

☞ COMM_AVG_ONLY_COMM_PERSONS에 표시된 값은 COMMISSION_PCT 컬럼이 NULL이 아닌 것만 고려합니다(즉, 35로 나눔).

☞ COMM_AVG_ALL_PERSONS에 표시된 값은 COMMISSION_PCT 컬럼이 NULL인 것도 포함하여 고려합니다(즉, 107로 나눔).

5-4. 그룹함수와 같이 사용하는 SELECT 문의 키워드: GROUP BY 와 HAVING.

- 예를 들어 10번, 20번 부서에 근무하는 사원의 부서별 평균 임금을 알고자 할 때, GROUP BY를 사용하지 않는다면, 아래처럼 SELECT문을 작성하여 사용할 수도 있습니다.

<pre>SELECT 10 AS DEPTNO, AVG(salary) AS AVG_SAL FROM hr.employees WHERE department_id = 10 UNION ALL SELECT 20, AVG(salary) FROM hr.employees WHERE department_id = 20 ;</pre>	<p>[department_id=10 인 행을 찾아서 평균 임금을 구하는 SELECT 문]</p> <p>UNION ALL</p> <p>[department_id=20 인 행을 찾아서 평균 임금을 구하는 SELECT 문] ;</p> <p>[참고] UNION ALL 연산자는 두 개의 SELECT 문을 처리한 결과 레코드를 합쳐서 표시해줍니다.</p>						
<table border="1"> <thead> <tr> <th>DEPTNO</th> <th>AVG_SAL</th> </tr> </thead> <tbody> <tr> <td>10</td> <td>4400</td> </tr> <tr> <td>20</td> <td>9500</td> </tr> </tbody> </table>	DEPTNO	AVG_SAL	10	4400	20	9500	
DEPTNO	AVG_SAL						
10	4400						
20	9500						

- 만약 30 번 부서까지 포함한 부서별 평균 임금을 구해야 한다면, 위의 SELECT문에 아래처럼 department_id=30 인 행을 찾아서 평균 임금을 구하는 SELECT 문을 [UNION ALL] 과 함께 기존 문장의 맨 끝에 추가해 주어야 합니다.

```
UNION ALL
SELECT 30, AVG(salary)
FROM hr.employees
WHERE department_id = 30
```

- GROUP BY 절을 이용하면, 위의 요구사항에 대하여 간단히 조회할 수 있습니다. 다음의 실습을 참고하십시오.

[실습] HR.EMPLOYEES 테이블로부터, 각 부서별 사원들의 평균봉급과 각 부서별 인원수를 표시하시오.

```
SQL> SELECT department_id, AVG(salary) AS "AVG_SAL_PER_DEPT" , COUNT(*) AS Persons
  FROM hr.employees
 GROUP BY department_id ;
```

DEPARTMENT_ID	AVG_SAL_PER_DEPT	PERSONS	--GROUP BY 절에 명시된 컬럼의 값이 같은 행들을 대상으로 --그룹함수가 실행됩니다.			
100	8601.33333	6	GROUP BY 가 SQL ORDER BY ..			
30	4150	6				
	7000	1				
20	9500	2				
70	10000	1				
90	19333.3333	3				
110	10154	2				
50	3475.55556	45				
40	6500	1				
80	8955.88235	34				
10	4400	1				
60	5760	5				

12개의 행이 선택됨

5-5. 둘 이상의 컬럼에 의한 행들의 그룹 처리

- GROUP BY 절에 둘 이상의 컬럼을 명시했을 때, 처리되는 방식을 확인하기 위하여 다음의 실습을 수행합니다.

[실습] 다음은 HR.EMPLOYEES 테이블에서 department_id가 30인 행들의 DEPARTMENT_ID, JOB_ID, SALARY로 구성된 레코드들입니다.

```
SQL> SELECT department_id, job_id, salary
  FROM hr.employees
 WHERE department_id =30
 ORDER BY 1, 2 ;
```

DEPARTMENT_ID	JOB_ID	SALARY
30	PU_CLERK	3100
30	PU_CLERK	2500
30	PU_CLERK	2900
30	PU_CLERK	2800
30	PU_CLERK	2600
30	PU_MAN	11000

6개의 행이 선택됨

- ☞ GROUP BY 뒤에 컬럼이름이 두 개 이상 명시된 경우, 행들은 GROUP BY 뒤에 명시된 컬럼들에 의해 구성되는 레코드가 같은 행들끼리 GROUPING 됩니다.

[실습] 30번 부서에 근무하는 사원을 대상으로, (동일한 부서 및 동일한 직책코드) 별로 근무하는 사원들에 대한 급여의 평균과 인원수를 표시하시오.

```
SQL> SELECT department_id, job_id, AVG(salary) AS "AVG_SAL", count(*) as persons
  FROM hr.employees
```

```
 WHERE department_id=30
```

```
 GROUP BY department_id, job_id ;
```

DEPARTMENT_ID	JOB_ID	AVG_SAL	PERSONS
30	PU_CLERK	2780	5
30	PU_MAN	11000	1

[문법] SELECT 절에서 그룹함수와 같이 명시된 컬럼이름들은 반드시 GROUP BY 절에 모두 명시되어야 합니다.

```
SELECT col1, col2, col3, col4, col5, col6, col7, col8, col9, col10, AVG(col11)
FROM table1
GROUP BY col1, col2, col3, col4, col5, col6, col7, col8, col9, col10 ;
```

- ☞ 위에서 그룹함수와 col1~col10까지 10개의 컬럼이 SELECT 절에 명시되었다면, GROUP BY 절에서 하나도 누락되는 것 없이 col1~col10까지 모든 컬럼을 명시해 주어야 합니다.
- ☞ SELECT 절에서 그룹함수와 같이 명시된 컬럼이름이 GROUP BY 절에서 누락되거나, 또는 GROUP BY 절이 누락되면 다음처럼 오류가 발생됩니다.

○ SELECT 절에서 그룹함수와 같이 사용된 컬럼 명이 GROUP BY 절에서 누락된 경우.	○ SELECT 절에서 그룹함수와 컬럼 명이 같이 명시되었는데, GROUP BY 절이 누락된 경우.
SQL> <pre>SELECT department_id, job_id, AVG(salary) FROM hr.employees WHERE department_id in (10,20,30,40) GROUP BY department_id ;</pre> <p>ORA-00979: GROUP BY 표현식이 아닙니다. 00979. 00000 - "not a GROUP BY expression"</p> <p>*Cause: *Action: 1행, 22열에서 오류 발생</p>	SQL> <pre>SELECT department_id, COUNT(last_name) FROM hr.employees;</pre> <p>ORA-00937: 단일 그룹의 그룹 함수가 아닙니다 00937. 00000 - "not a single-group group function"</p> <p>*Cause: *Action: 1행, 8열에서 오류 발생</p>

[참고] WHERE 절에는, 그룹함수를 사용하여 조건절을 작성할 수 없습니다.

- WHERE 절은 SELECT문의 처리되는 행 제한하기 위하여 사용됩니다. 따라서, WHERE 절과 GROUP BY 절이 같이 사용된 경우 처리되는 순서는 다음과 같습니다
 - (1) WHERE절에 명시된 조건을 만족하는 행을 먼저 선택(SELECTION)한 후에
 - (2) 선택된 행들에서 GROUP BY 키워드에 명시된 컬럼의 값이 같은 행들을 찾아서 GROUPING을 한 후,
 - (3) 그 행들의 그룹에서 그룹함수가 처리됩니다.
- 따라서, ~~위에서처럼~~ 행을 선택하는 기능을 위한 WHERE절에는 훨씬 뒤에 처리되는 그룹함수가 명시될 수 없습니다.

■ WHERE 절에 그룹함수를 사용하면, 다음처럼 오류가 발생됩니다.

```
SQL> 

```
SELECT department_id, SUM(salary)
 FROM hr.employees
 WHERE SUM(salary) > 7000
 GROUP BY department_id ;
```



ORA-00934: 그룹 함수는 허가되지 않습니다  
00934. 00000 - "group function is not allowed here"



*Cause:  
*Action:  
3행, 14열에서 오류 발생


```

5-6. HAVING 절 사용.

■ **HAVING 절은** GROUP BY 절과 함께 사용되어, **그룹함수가 포함된 조건절을 작성할 수 있습니다.**

■ HAVING 절의 기능을 확인하기 위하여 다음의 실습을 수행합니다.

[실습] 다음은 HR.EMPLOYEES 테이블에서 job_id 컬럼의 값에 REP가 포함된 행들의 일부 컬럼들로 구성된 레코드입니다.

```
SQL> SELECT department_id, salary FROM hr.employees WHERE job_id LIKE '%REP%' ORDER BY 1 ;
```

DEPARTMENT_ID	SALARY
20	6000
40	6500
70	10000
80	8000
80	7500
80	7000
80	10000
80	9500
...	
	7000

33개의 행이 선택됨

[실습] HR.EMPLOYEES 테이블로부터, job_id에 REP가 포함된 사원에 대해서만 부서별 임금의 합계, 부서별 평균 임금 및 부서별 근무 인원수를 구하시오.

```
SQL> SELECT department_id, SUM(salary), AVG(salary), COUNT(*)
   FROM hr.employees
  WHERE job_id LIKE '%REP%'
  GROUP BY department_id ;
```

DEPARTMENT_ID	SUM(SALARY)	AVG(SALARY)	COUNT(*)
20	7000	7000	1
20	6000	6000	1
70	10000	10000	1
40	6500	6500	1
80	243500	8396.55172	29

[실습] HR.EMPLOYEES 테이블로부터, job_id에 REP가 포함된 사원에 대해서만 고려하여, 이때 부서별 임금의 합계가 7000 보다 큰 부서에 대해서만, 부서별 임금의 합계, 부서별 평균 임금 및 부서별 근무 인원수를 구하시오.

```
SQL> SELECT department_id, SUM(salary), AVG(salary), count(*)
   FROM hr.employees
  WHERE job_id LIKE '%REP%'      --(1)
  GROUP BY department_id        --(2)
  HAVING SUM(salary) > 7000    --(3)
  ORDER BY 2 ;                 --(4)

DEPARTMENT_ID SUM(SALARY) AVG(SALARY) COUNT(*)
----- ----- -----
          70      10000     10000       1
         80     243500    8396.55172     29
```

☞ HAVING 절을 이용하면 그룹함수에 제한을 걸어서 최종 결과를 선별할 수 있습니다. 이 때, 처리 순서는 다음과 같습니다.

- (1) WHERE절에 명시된 조건을 만족하는 행을 선택(SELECTION)한 후,
- (2) 선택된 행들에서 GROUP BY 키워드에 명시된 컬럼의 값이 같은 행들을 찾은 후(즉, GROUP을 찾은 후),
- (3) 해당 그룹의 행들을 대상으로 HAVING 절의 그룹함수의 조건을 검사하여, 조건을 만족하면 해당 그룹에 대하여 원하는 레코드를 구성합니다.
- (2), (3) 과정을 반복하여 WHERE 절을 만족하는 모든 행들의 각 그룹들을 모두 처리한 후,
- (4) 최종 결과 레코드를 정렬(ORDER BY 절)하여, 구문을 요청한 사용자에게 전달합니다.

5-7. 그룹함수의 중첩.

■ 그룹함수는 2번 까지만 중첩할 수 있습니다. 왜냐하면, 아래처럼 그룹함수를 2번 중첩하면 무조건 하나의 결과가 되기 때문입니다.

■ HR.EMPLOYEES 테이블을 이용하여, 부서별 임금의 합계 금액 중에서, 가장 큰 부서별 합계임금 값을 표시하시오.

```
SQL> SELECT MAX(SUM(salary)) AS Result
   FROM hr.employees
  GROUP BY department_id;

RESULT
-----
304500
```

```
[ 1) ]HR.EMPLOYEES      hire_date(      )
      2)             SELECT
                      SELECT
                        (WW, IW, W)
  1)           SELECT TO_CHAR(hire_date,'YYYY') AS "      ", count(*) AS "
  FROM hr.employees GROUP BY TO_CHAR(hire_date,'YYYY') ORDER BY 1;
  2)           SELECT TO_CHAR(hire_date,'YYYYMM') AS "      ", count(*) AS "
  FROM hr.employees
  GROUP BY TO_CHAR(hire_date,'YYYYMM')
  ORDER BY 1;
```

☞ SELECT 절에 중첩된 그룹함수를 사용하면, 테이블의 컬럼을 SELECT 절에 명시할 수 없습니다.

```
SELECT
  , (count(*))
```

6 두 테이블의 행을 결합하여 데이터 조회하기 (Join을 사용한 데이터 조회).**◆ 학습 목표.**

■ 아래의 다양한 ANSI-표준 조인(JOINING) 방법을 이용하여 두 테이블의 데이터를 조회하는 방법을 학습합니다.

- EQUI-INNER JOIN
- NON-EQUI-INNER JOIN
- OUTER JOIN
- CROSS JOIN

- NATURAL JOIN: EQUI-INNER JOIN
- SELF-JOIN: INNER JOIN OUTER JOIN

6-1. JOIN 개념.

■ JOIN은, 조인-조건을 기준으로 두 테이블의 각 행들을 합친 후, 원하는 데이터의 레코드를 가져오는 방법입니다.

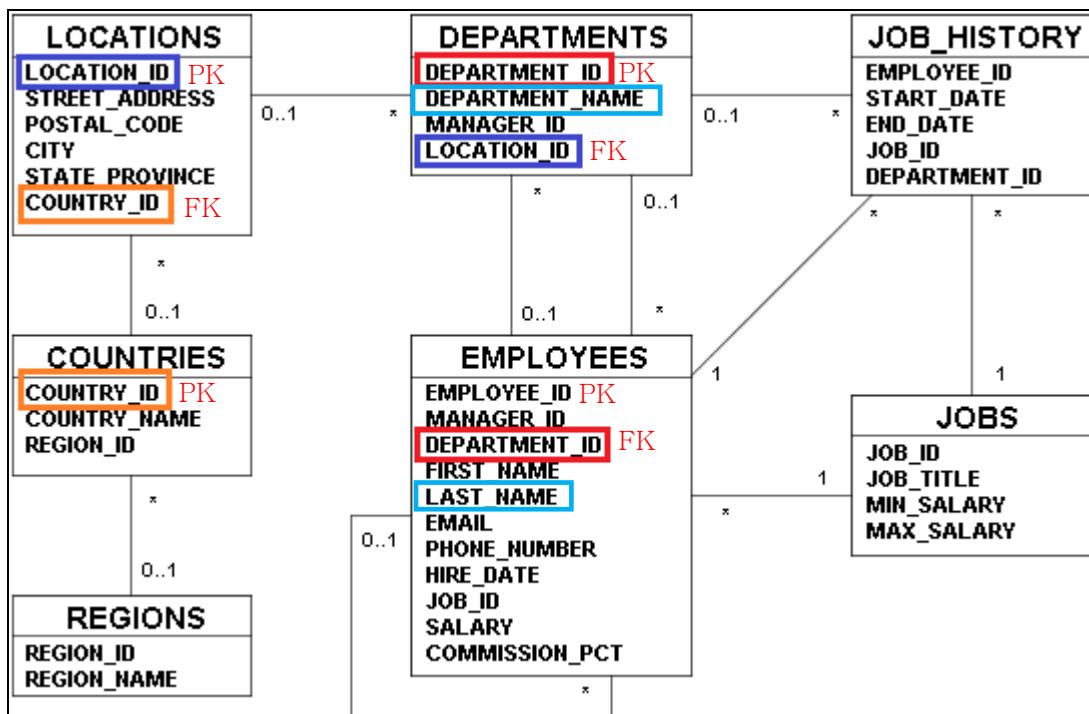
■ ANSI 표준 JOIN 중, 많이 사용되는 대표적인 3가지 방법은 다음과 같습니다.

INNER JOIN	EQUI-INNER JOIN
	NONEQUI-INNER JOIN
OUTER JOIN	
CROSS JOIN	

■ INNER-JOIN과 OUTER-JOIN 방법에서, [=] 연산자를 조인조건에 사용하는 경우 [ON-절] 또는 [USING-절]을 이용하는 2 가지 방법이 있습니다.

☞ 위의 조인방식 외에, Natural Join 방법도 있으나, 이는 EQUI-INNER JOIN으로 대체되어 사용될 수 있으므로 본 교재에서는 설명을 생략합니다.

[참고] HR (Human Resources) SCHEMA DATA-SET.



- 위의 HR DATA-SET 그림에서, 사원의 사번(EMPLOYEES.EMPLOYEE_ID 컬럼값)과 봉급(EMPLOYEES.SALARY 컬럼값) 및 그 사원이 근무하는 부서의 이름(DEPARTMENTS.DEPARTMENT_NAME 컬럼값)으로 구성된 레코드를 구성하려고 합니다. 즉, 두 개(또는 그 이상)의 테이블에 저장된 데이터로부터 구성된 레코드를 반환할 수 있는 SELECT문을 하려고 합니다. 이를 위하여 사용되는 데이터 조회방법이 JOIN입니다.

[참고] 기본키 (PRIMARY KEY, PK) 및 외래키 (FOREIGN KEY, FK) 제약조건

- 기본키 제약조건은 테이블에 저장된 각 행을 고유하게 식별할 수 있는 데이터 또는 데이터들의 조합입니다.

따라서, 행을 고유하게 식별해야 하므로 기능상 데이터도 반드시 있어야 하며, 중복되면 안됩니다.

- 컬럼1, 컬럼2로 구성된 테이블의 컬럼1에 기본키 제약조건을 정의한 경우, 행 입력 시에 다음처럼 기본키 제약조건이 데이터에 대한 검사를 수행합니다.

컬럼1	컬럼2
10	A

20 A ← INSERT(1)은, 기존의 행에 입력된 컬럼1의 값과 중복되지 않으므로 행이 입력됩니다.

10 B ← INSERT(2)는 기존의 행에 입력된 컬럼1의 값과 중복되므로 오류가 발생됩니다.

B ← INSERT(3)은 입력되는 행의 컬럼1이 데이터가 없으므로(NULL) 오류가 발생됩니다.

- 외래키 제약조건은 기본키 제약조건을 참조하는 제약조건입니다. 즉, 외래키가 정의된 컬럼의 값은 반드시 기본키가 정의된 컬럼의 값을 중 하나이어야 합니다.

- 사원테이블의 부서코드에 외래키 제약조건이 정의되지 않으면, 부서테이블의 부서코드와 상관없이, 사원테이블의 부서코드 컬럼에 아무 값(아래 그림에서 부서코드가 3인 행)이나 삽입될 수 있습니다.
그렇지만, 사원테이블의 부서코드 컬럼에 외래키 제약조건이 정의되어 있다면, 사원테이블에 행을 삽입할 때, 부서_테이블의 부서코드 컬럼에 없는 값을 가지는 행이 사원_테이블에 삽입될 수 없습니다.

- 다음처럼, 부서테이블과 사원테이블이 구성되어 있고, 사원테이블의 부서코드 컬럼에, 부서테이블의 기본키 컬럼인 부서코드 컬럼을 참조하는 외래키 제약조건이 정의되어 있다면, 사원테이블에 데이터를 입력할 때, 다음처럼 외래키 제약조건이 데이터에 대한 검사를 수행합니다.



- INSERT(1)은, 부서코드에 입력되는 값 20이 부서테이블의 부서코드 컬럼에 있는 값이므로 행이 입력됩니다.
- INSERT(2)는 부서코드에 입력되는 값 3은 부서_테이블의 부서코드 컬럼에 없는 값이므로, 오류가 발생됩니다.
- INSERT(3)은 부서코드에 검사할 데이터가 없는 NULL 이므로 검사하지 않고 행이 입력됩니다.

6-2. EQUI-INNER JOIN: ON 절을 이용한 레코드 추출.

- EQUI-INNER JOIN이란? [ON] 절을 이용하여 조인 조건을 기술할 때, 조건에 "=" 연산자를 사용합니다.

- 구문 이해를 위해 아래의 HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 데이터를 참고 합니다.

HR.EMPLOYEES 테이블 데이터	HR.DEPARTMENTS 테이블 데이터	
EMPLOYEE_ID LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID DEPARTMENT_NAME
200 Whalen	10	10 Administration
201 Hartstein	20	20 Marketing
202 Fay	20	30 Purchasing
114 Raphaely	30	...
119 Colmenares	30	120 Treasury
115 Khoo	30	130 Corporate Tax
116 Baida	30	140 Control And Credit
117 Tobias	30	...
118 Himuro	30	240 Government Sales
...		250 Retail Sales
113 Popp	100	260 Recruiting
109 Faviet	100	270 Payroll
206 Gietz	110	27개의 행이 선택됨
205 Higgins	110	
178 Grant		
107개의 행이 선택됨		

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터 사원의 사번, 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오.

```
SQL> SELECT hr.employees.employee_id, hr.employees.last_name, hr.employees.department_id,
       hr.departments.department_name
  FROM hr.employees INNER JOIN hr.departments
  ON (hr.employees.department_id=hr.departments.department_id)
 ORDER BY 3 ;
```

EMPLOYEE_ID LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200 Whalen	10	Administration
201 Hartstein	20	Marketing
202 Fay	20	Marketing
114 Raphaely	30	Purchasing
115 Khoo	30	Purchasing
116 Baida	30	Purchasing
117 Tobias	30	Purchasing
118 Himuro	30	Purchasing
119 Colmenares	30	Purchasing
203 Mavris	40	Human Resources
120 Weiss	50	Shipping
...		
106개의 행이 선택됨		

■ EQUI-INNER JOIN 구문설명

ON 절 표시된 조인 조건에 따라 EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼의 데이터와 DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼의 데이터가 같은 두 테이블의 행들을 합친 후(예, EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼의 데이터가 10인 행과 DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼의 데이터가 10인 행을 합친 후), SELECT 절에 명시된 컬럼들을 두 테이블로부터 추출하여 레코드를 구성합니다.

이 때 두 테이블의 행을 합치기 위하여 조인 조건에 = 연산자가 사용되었고, 조인-조건을 만족하는 경우에만 두 테이블의 행을 합치기 때문에 EQUI-INNER JOIN 이라고 합니다. 즉, 조인 조건을 만족하지 않는 행들은 합쳐 지지 않으며 결과에도 표시되지 않습니다.

EMPLOYEES 테이블에서 EMPLOYEE_ID가 178인 행은 DEPARTMENT_ID 컬럼에 값이 없기 때문에(NULL이기 때문에) 조인-조건에 의해 합칠 수가 없고, DEPARTMENTS 테이블에서 DEPARTMENT_ID가 120부터 270번까지 행들은 EMPLOYEES 테이블에 해당 값이 없기 때문에 조인 조건을 만족할 수 없어서 레코드로 표시되지 않습니다.

6-3. 컬럼이름 앞에 테이블의 이름을 접두어로 명시할 때 장점.

- 컬럼이름 앞에 테이블 접두어를 명시하면, 코드 해석이 쉬워지고 성능이 조금 개선되며, 아래의 오류도 해결됩니다.

```
SQL> SELECT employee_id, employees.last_name, department_id
   FROM hr.employees INNER JOIN hr.departments
  ON (employees.department_id = departments.department_id) ;
ORA-00918: 열의 정의가 애매합니다
00918. 00000 - "column ambiguously defined"
*Cause:
*Action:
1행, 42열에서 오류 발생
```

- 위의 문장에서 SELECT절에 있는 department_id 컬럼에, 테이블 접두어가 없고, 컬럼이 두 테이블에 모두 존재하기 때문에, "column Ambiguously defined" 오류가 발생되었습니다.

6-4. 테이블 별칭 사용.

- 컬럼이름 앞에 테이블이름을 접두어를 사용하면, 구문이 길어집니다. FROM 절에서 테이블이름 다음에 최대한 간단히 기술된 테이블별칭을 선언할 수 있습니다. 선언된 테이블별칭을, SQL문에서 테이블이름 대신 사용하여 보다 간결한 구문 작성이 가능해집니다.

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터 사원의 사번, 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오.

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.department_name
   FROM hr.employees e JOIN hr.departments d ON (e.department_id = d.department_id)
   ORDER BY 3 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	Administration
201	Hartstein	20	Marketing
202	Fay	20	Marketing
114	Raphaely	30	Purchasing
119	Colmenares	30	Purchasing
115	Khoo	30	Purchasing
116	Baida	30	Purchasing
...			
106개의 행이 선택됨			

☞ JOIN 키워드 앞에 JOIN 형식을 지정하는 키워드를 생략하면, INNER-JOIN 을 의미합니다.

6-5. 3 개 테이블의 조인 (3-Way Join).

- FROM 절에서, 3개의 테이블을 조인하는 것을 3-Way 조인이라고 합니다.

[실습] HR.EMPLOYEES 테이블, HR.DEPARTMENTS 테이블, 및 HR.LOCATIONS 테이블로부터, 사번이 100,150,200 인 사원의 성 (EMPLOYEES.LAST_NAME 컬럼)과 그 사원이 출근하는 부서가 있는 도시이름(LOCATIONS.CITY 컬럼)을 표시하시오.

```
SQL> SELECT e.last_name, l.city
   FROM hr.employees e JOIN hr.departments d ON (e.department_id = d.department_id)
   JOIN hr.locations l ON (d.location_id = l.location_id)
   WHERE e.employee_id IN (100,150,200);
```

LAST_NAME	CITY
King	Seattle
Tucker	Oxford
Whalen	Seattle

- 3 개 이상의 테이블과 조인하는 것을 N-Way 조인이라고 하며, N-Way 조인 구문 작성 시에 JOIN 키워드 다음에 선언된 테이블은, 앞에 선언된 테이블하고만 조인될 수 있습니다. 즉, 뒤에 선언된 테이블과는 조인될 수 없습니다. 따라서, 구문 작성 시에 다음의 오류를 조심하시기 바랍니다.

```
SQL> SELECT e.employee_id, e.last_name, l.city, d.department_name
   FROM hr.employees e JOIN hr.locations l ON (d.location_id =l.location_id)
                      JOIN hr.departments d ON (e.department_id = d.department_id)
 WHERE e.employee_id IN (100,150,200);
ORA-00904: "D"."LOCATION_ID": 부적합한 식별자
00904. 00000 - "%s: invalid identifier"
*Cause:
*Action:
2행, 53열에서 오류 발생
```

- ☞ 위의 문장에서 2번째 라인에 있는 d.location_id 에서 d가 해석될 수 없기 때문에 오류가 발생되었습니다. 즉, 조인 조건에 의하여 locations 테이블이 뒤에 있는 departments 테이블과 조인이 되도록 작성되었고, 이 때, d는 다음 라인의 departments 테이블의 Table-Alias로서, locations 테이블 다음에서 선언되었기 때문에 해석될 수 없어서 오류가 발생된 것입니다.

6-6. EQUI-INNER JOIN: USING 절을 이용한 레코드 추출.

- EQUI-INNER JOIN 문장에서 조인 조건 작성 시에 컬럼이름이 동일하고 데이터유형이 일치되는 두 테이블의 특정 컬럼을, USING 절에 명시하여 EQUI-INNER 조인의 조인조건을 명시할 수 있습니다

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터, 사원의 사번, 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오.

```
SQL> SELECT e.employee_id, e.last_name, department_id, d.department_name
   FROM hr.employees e INNER JOIN departments d USING (department_id) ;
EMPLOYEE_ID LAST_NAME          DEPARTMENT_ID DEPARTMENT_NAME
-----  -----
 200 Whalen                  10 Administration
 201 Hartstein                20 Marketing
 202 Fay                       20 Marketing
 114 Raphaely                 30 Purchasing
 119 Colmenares               30 Purchasing
 115 Khoo                      30 Purchasing
 116 Baida                     30 Purchasing
...
106개의 행이 선택됨
```

[실습] HR.EMPLOYEES 테이블, HR.DEPARTMENTS 테이블, 및 HR.LOCATIONS 테이블로부터, 사번이 100,150,200 인 사원의 성(EMPLOYEES.LAST_NAME 컬럼)과 그 사원이 출근하는 부서가 있는 도시이름(LOCATIONS.CITY 컬럼)을 표시하시오.

```
SQL> SELECT e.last_name, l.city
  FROM hr.employees e JOIN hr.departments d USING(department_id)
    JOIN hr.locations l USING(location_id)
 WHERE e.employee_id IN (100,150,200);
```

LAST_NAME	CITY
King	Seattle
Tucker	Oxford
Whalen	Seattle

☞ 컬럼이름이 같더라도 데이터유형이 다르면 오류가 발생되며, USING 절에 명시된 컬럼에는 **테이블-별칭을 명시하면 안됩니다.** USING 절에 정의된 컬럼에 테이블별칭을 사용하면, 아래처럼 오류가 발생됩니다.

```
SQL> SELECT e.employee_id, e.last_name, d.department_id, d.department_name
  FROM hr.employees e INNER JOIN departments d USING (department_id) ;
ORA-25154: USING 절의 열 부분은 식별자를 가질 수 없음
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause: Columns that are used for a named-join (either a NATURAL join
        or a join with a USING clause) cannot have an explicit qualifier.
*Action: Remove the qualifier.
1행, 36열에서 오류 발생
```

6-7. NON-EQUI-INNER JOIN.

■ INNER JOIN 구문에서 조인 조건을 기술할 때, 조건에 "**= 이 아닌 다른 연산자**"가 사용된 경우, 이를 **NON-EQUI-INNER JOIN**이라고 합니다.

☞ 단원의 실습을 위하여, 21페이지의 내용을 수행하여, HR.JOB_GRADES 테이블을 생성하고, 데이터를 입력합니다.

■ 구문 이해를 위해 아래의 EMPLOYEES 테이블과 JOB_GRADES 테이블의 데이터를 참고 합니다.

EMPLOYEES 테이블 데이터		JOB_GRADES 테이블 데이터			
EMPLOYEE_ID	LAST_NAME	SALARY	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
132	Olson	2100	A	1000	2999
136	Philtanker	2200	B	3000	5999
...			C	6000	9999
195	Jones	2800	D	10000	14999
116	Baida	2900	E	15000	24999
134	Rogers	2900	F	25000	40000
190	Gates	2900			
197	Feeney	3000			
...					
168	Ozer	11500			
147	Errazuriz	12000			
...					

- ☞ JOB_GRADES 테이블은 사원들이 지급받는 급여가 포함되는 금액 범위에 따른 등급 정보를 제공하며, 등급에 대한 급여의 범위가 누락되거나 겹치지 않습니다. 그리고, 사원의 급여는 등급의 급여범위 중 한 범위에 포함됩니다. 따라서 사원의 급여와 급여 등급은 [다-대-일] 관계에 있습니다.

[실습] HR.EMPLOYEES 테이블과 HR.JOB_GRADES 테이블로부터, 모든 사원에 대하여 사원의 성(EMPLOYEES.LAST_NAME 컬럼)과 사원의 급여(EMPLOYEES.SALARY 컬럼) 및 해당 급여에 대한 등급(JOB_GRADES.GRADE_LEVEL 컬럼)을 표시하시오.

```
SQL> SELECT e.last_name, e.salary, j.grade_level
  FROM hr.employees e INNER JOIN hr.job_grades j
    ON (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

LAST_NAME	SALARY	GR	---조인조건에 = 연산자가 아닌 BETWEEN AND 연산자가 사용되었습니다.
Olson	2100	A	--조인조건에서, EMPLOYEES 테이블에 있는 한 행의 SALARY 컬럼의 값은,
Philtanker	2200	A	--JOB_GRADES 테이블에 있는, 각 등급의 하나의 LOWEST_SAL과 HIGHEST_SAL
...			--범위에만 포함됩니다. 따라서, EMPLOYEES 테이블의 SALARY 컬럼의 값과
Feeeney	3000	B	--JOB_GRADES 테이블의 GRADE_LEVEL 컬럼은 [다 대 일] 관계로 합쳐집니다.
Cabrio	3000	B	
...			
Kumar	6100	C	
Banda	6200	C	
...			
Tucker	10000	D	
Vishney	10500	D	
...			
Kochhar	17000	E	
King	24000	E	

107개의 행이 선택됨

6-8. OUTER JOIN.

- INNER JOIN 결과도 표시되고, [INNER JOIN]에서 조인 조건을 만족시키지 못하지만, 표시되길 원하는 레코드를 추가적으로 표시해 줍니다. 다음의 3 가지 방법이 가능합니다.

(1) **RIGHT OUTER JOIN**은, INNER JOIN 결과도 출력되고, JOIN 키워드의 오른편에 명시된 테이블로부터, 조인 조건을 만족시키지 못하지만, 표시되길 원하는 정보가 추가적으로 표시됩니다.

(2) **LEFT OUTER JOIN**은, INNER JOIN 결과도 출력되고, JOIN 키워드의 왼편에 명시된 테이블로부터, 조인 조건을 만족시키지 못하지만, 표시되길 원하는 정보가 추가적으로 표시됩니다.

(3) **FULL OUTER JOIN**은, INNER JOIN 결과도 출력되고, 조인되는 모든 테이블에서 조인 조건을 만족시키지 못하지만, 표시되길 원하는 정보가 추가적으로 표시됩니다.

- 구문 이해를 위해 아래의 EMPLOYEES 테이블과 DEPARTMENTS 테이블의 데이터를 참고 합니다.

EMPLOYEES 테이블 데이터		DEPARTMENTS 테이블 데이터	
EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	10 Administration
201	Hartstein	20	20 Marketing
202	Fay	20	30 Purchasing
114	Raphaely	30	...
119	Colmenares	30	120 Treasury
115	Khoo	30	130 Corporate Tax
116	Baida	30	140 Control And Credit
117	Tobias	30	150 Shareholder Services
118	Himuro	30	160 Benefits
203	Mavris	40	170 Manufacturing
...			180 Construction
110	Chen	100	190 Contracting
108	Greenberg	100	200 Operations
111	Sciarra	100	210 IT Support
112	Urman	100	220 NOC
113	Popp	100	230 IT Helpdesk
109	Faviet	100	240 Government Sales
206	Gietz	110	250 Retail Sales
205	Higgins	110	260 Recruiting
178	Grant		270 Payroll
107개의 행이 선택됨		27개의 행이 선택됨	

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터 사원의 사번, 사원의 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오. 단, 근무부서가 지정되지 않은 사원에 대한 정보도 표시되어야 합니다.

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.department_name
   FROM hr.employees e LEFT OUTER JOIN departments d ON (e.department_id=d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	Administration
202	Fay	20	Marketing
201	Hartstein	20	Marketing
119	Colmenares	30	Purchasing
118	Himuro	30	Purchasing
117	Tobias	30	Purchasing
...			
108	Greenberg	100	Finance
206	Gietz	110	Accounting
205	Higgins	110	Accounting
178	Grant		
107개 행이 선택되었습니다.			

☞ LEFT OUTER JOIN을 이용하여, JOIN 키워드의 왼편에 명시된 EMPLOYEES 테이블로부터, 조인 조건을 만족시키지 못하지만 표시되길 원하는 정보가 추가적으로 더 표시되었습니다.

■ OUTER JOIN 처리 방식 설명

INNER JOIN의 결과도 표시되고, INNER JOIN에서 조인 조건을 만족하지 않아서 표시되지 않았던 행들(EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 NULL 인 행과 DEPARTMENTS 테이블에서는 DEPARTMENT_ID가 120부터 270번까지 행들)의 레코드도 같이 표시됩니다.

조인조건을 만족하지 못하는 행들도 표시하기 위하여 OUTER JOIN 사용 시 JOIN 키워드의 (1) 왼쪽(LEFT)에 명시된 테이블로부터 조인조건을 만족하지 못하는 행들도 표시하려는 경우에는 LEFT OUTER JOIN을, (2) 오른쪽(LEFT)에 명시된 테이블로부터 조인조건을 만족하지 못하는 행들도 표시하려는 경우에는 RIGHT OUTER JOIN을, (3) 양쪽 테이블 모두로부터 조인조건을 만족하지 못하는 행들도 표시하려는 경우에는 FULL OUTER JOIN을 각각 사용합니다.

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터 사원의 사번, 사원의 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오. 단, 근무하는 사원이 없는 부서의 부서코드 및 부서이름도 결과에 표시되도록 합니다.

```
SQL> SELECT e.employee_id, e.last_name, d.department_id, d.department_name
  FROM hr.employees e RIGHT OUTER JOIN departments d ON (e.department_id=d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
200	Whalen	10	Administration
201	Hartstein	20	Marketing
202	Fay	20	Marketing
...			
		120	Treasury
		130	Corporate Tax
		140	Control And Credit
		150	Shareholder Services
		160	Benefits
		170	Manufacturing
		180	Construction
		190	Contracting
		200	Operations
		210	IT Support
		220	NOC
		230	IT Helpdesk
		240	Government Sales
		250	Retail Sales
		260	Recruiting
		270	Payroll

122개 행이 선택되었습니다.

☞ RIGHT OUTER JOIN을 이용하여, JOIN 키워드의 오른편에 명시된 DEPARTMENTS 테이블로부터, 조인 조건을 만족시키지 못하지만 표시되길 원하는 정보가 추가적으로 더 표시되었습니다.

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블로부터 사원의 사번, 사원의 성과 사원이 근무하는 부서의 부서코드 및 부서이름을 모든 사원에 대하여 표시하시오. 단, 근무부서가 지정되지 않은 사원에 대한 정보도 표시하며, 근무하는 사원이 없는 부서의 부서코드 및 부서이름도 결과에 표시되도록 합니다.

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.department_name
   FROM hr.employees e FULL OUTER JOIN departments d ON (e.department_id=d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	DEPARTMENT_NAME
198	OConnell	50	50	Shipping
199	Grant	50	50	Shipping
200	Whalen	10	10	Administration
201	Hartstein	20	20	Marketing
202	Fay	20	20	Marketing
203	Mavris	40	40	Human Resources
...				
178	Grant	80	80	Sales
179	Johnson			
...				
189	Dilly	50	50	Shipping
190	Gates	50	50	Shipping
191	Perkins	50	50	Shipping
192	Bell	50	50	Shipping
193	Everett	50	50	Shipping
194	McCain	50	50	Shipping
195	Jones	50	50	Shipping
196	Walsh	50	50	Shipping
197	Feeney	50	50	Shipping
		220	NOC	
		170	Manufacturing	
		240	Government Sales	
		210	IT Support	
		160	Benefits	
		150	Shareholder Services	
		250	Retail Sales	
		140	Control And Credit	
		260	Recruiting	
		200	Operations	
		120	Treasury	
		270	Payroll	
		130	Corporate Tax	
		180	Construction	
		190	Contracting	
		230	IT Helpdesk	

123개의 행이 선택됨

☞ FULL OUTER JOIN을 이용하여, FROM 절에 명시된 모든 테이블로부터, 조인 조건을 만족시키지 못하지만 표시되길 원하는 정보가 추가적으로 더 표시되었습니다.

6-9. 테이블 JOIN 시에 추가적인 조건 적용하기.

- INNER-JOIN에서는 추가적인 조건을 기술할 때 ON 절 다음에 [AND절]을 사용하거나 또는 FROM절 다음에 [WHERE절]을 사용할 수 있습니다.

- EMPLOYEES 및 DEPARTMENTS 테이블을 이용하여, 90번 부서에 근무하는 사원의 성과 근무하는 부서이름을 표시하시오.

--추가적인 조건을 AND절을 이용하여 명시한 경우.

```
SQL> SELECT e.last_name, d.department_name
  FROM hr.employees e JOIN hr.departments d ON (e.department_id = d.department_id )
    AND e.department_id = 90 ;
```

LAST_NAME	DEPARTMENT_NAME
King	Executive
Kochhar	Executive
De Haan	Executive

--추가적인 조건을 WHERE절을 이용하여 명시한 경우.

```
SQL> SELECT e.last_name, d.department_name
  FROM hr.employees e JOIN hr.departments d ON (e.department_id = d.department_id )
 WHERE e.department_id = 90 ;
```

LAST_NAME	DEPARTMENT_NAME
King	Executive
Kochhar	Executive
De Haan	Executive

- ☞ INNER-JOIN의 경우에는 추가적인 조건을 ON-절 다음에 [AND-절]을 사용하거나 별도의 [WHERE절]을 사용하거나 상관없이 동일한 표시결과를 얻을 수 있습니다.

- ☞ OUTER JOIN의 경우에는 추가적인 조건을 AND절을 사용하여 명시할 때와 WHERE절을 사용하여 명시할 때, 다음과 같은 차이가 있으므로, 서로 다른 결과가 표시될 수 있습니다.

- 추가적인 조건을 AND절을 사용하는 경우에는, 조인조건을 명시하는 ON 절의 조건이 2 가지 조건으로 처리됩니다.

즉, 2개의 조인조건을 검사하여, 두 조건을 모두 만족할 때만 두 테이블의 행이 합쳐지며, 둘 중 하나의 조건만 만족하는 경우에는 두 테이블의 행이 합쳐지지 않고, OUTER JOIN의 유형에 따라 추가적으로 데이터가 표시되며, [일 대 다] 관계에서 [다] 쪽 테이블의 모든 행이 처리되어 표시됩니다.

- 추가적인 조건을 WHERE절을 사용하는 경우에는, 테이블로부터 WHERE절을 만족하는 행에 대해서만 조인조건이 평가됩니다. 즉, 테이블의 행 중, WHERE절을 만족하는 행에 대해서만 이 후 처리가 진행됩니다. WHERE절을 만족하지 않는 행은 처리대상에서 제외됩니다.

6-10. CROSS JOIN.

■ Cartesian product에 대하여

JOIN 문장에서 (1) 조인-조건이 누락되었거나 (2) 조인-조건이 유효하지 않은 경우에, 첫 번째 테이블의 각 행들이 두 번째 테이블의 모든 행들과 조인될 수도 있습니다. 이런 현상을 Cartesian product라고 하며, 조인 사용시 Cartesian product가 발생되지 않도록 주의해야 합니다.

■ CROSS JOIN은 위에서 설명한 Cartesian product를 구현한 JOIN 방법으로, INNER JOIN 또는 OUTER JOIN으로 합쳐 질 수 없는 행들을 합쳐야 할 때 사용할 수 있습니다.

[실습] EMPLOYEES 및 DEPARTMENTS 테이블 테이블로부터, 90번 부서에 근무하는 사원의 사번, 사원의 성과 사원이 근무하는 부서의 부서코드를 표시하되 표시되는 각 레코드마다 10번 부서의 부서코드 및 부서이름이 같이 표시되도록 하시오.

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.department_name
   FROM hr.employees e CROSS JOIN departments d
 WHERE e.department_id = 90 AND d.department_id = 10 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	DEPARTMENT_NAME
100	King	90	10	Administration
101	Kochhar	90	10	Administration
102	De Haan	90	10	Administration

☞ CROSS JOIN을 이용하여 EMPLOYEES 테이블의 부서코드가 90인 행과 DEPARTMENTS 테이블의 부서코드가 10인 행을 합쳐서 결과 레코드가 표시됩니다.

☞ 잘못된 CROSS JOIN 사용: 다음처럼, WHERE절 없이 CROSS JOIN을 사용하면, EMPLOYEES 테이블의 각 행들이 DEPARTMENTS 테이블의 모든 행들과 조인되어 2889 개의 레코드가 표시됩니다.

```
SQL> SELECT e.employee_id, e.last_name, d.department_name
   FROM employees e CROSS JOIN departments d ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_NAME
198	OConnell	Administration
199	Grant	Administration
200	Whalen	Administration
201	Hartstein	Administration
202	Fay	Administration
203	Mavris	Administration
204	Baer	Administration
206	Gietz	Administration

2,889개의 행이 선택됨

☞ CROSS JOIN 사용 시에는, WHERE 절 및 AND 을 이용하여, 합쳐야 하는 테이블의 행수를 제한하는 조건을 명시합니다.

[참고] 오라클-식 조인 문법.

■ 오라클 9i 버전부터는 위에서 소개한 ANSI-표준 조인 문법을 사용할 수 있지만, 오라클 8i 버전에서는 위에서 설명한 ANSI-표준 조인을 지원하지 않았기 때문에, 다음과 같은 조인 문법을 이용하였습니다.

■ 오라클-식 조인 문법을 사용하여 문장을 작성할 때, 방법은 다음과 같습니다.

- 테이블이름들을 FROM 절에 콤마(,)로 구분하여 나열합니다.
- 조인조건은 WHERE 절에 다른 조건과 마찬가지로 명시합니다.
- 오라클 OUTER 조인에서는 조인조건에서 (+) 기호가 명시되지 않은 컬럼이 정의된 테이블 데이터를 표시해줍니다.

■ 오라클 EQUI-JOIN (ANSI EQUI-INNER 조인)

```
SQL> SELECT e.last_name, d.department_name ,d.department_id
      FROM hr.employees e, hr.departments d
     WHERE e.department_id = 90
       AND e.department_id = d.department_id ;
```

■ 오라클 OUTER-JOIN

```
SQL> SELECT e.last_name, d.department_name ,d.department_id
      FROM hr.employees e, hr.departments d
     WHERE e.department_id (+) = d.department_id ;
```

```
SQL> SELECT e.last_name, d.department_name ,d.department_id
      FROM hr.employees e, hr.departments d
     WHERE e.department_id = d.department_id (+);
```

☞ 오라클 OUTER-JOIN에서 양쪽 모두에 (+) 를 사용하면, 다음처럼 오류가 발생됩니다.

```
SQL> SELECT e.last_name, d.department_name ,d.department_id
      FROM hr.employees e, hr.departments d
     WHERE e.department_id (+) = d.department_id (+) ;
```

ERROR at line 3:
ORA-01468: a predicate may reference only one outer-joined table

■ 오라클 NON-EQUIJOIN

```
SQL> SELECT e.last_name, e.salary, j.grade_level
      FROM hr.employees e, hr.job_grades j
     WHERE e.salary BETWEEN j.lowest_sal AND j.highest_sal ;
```

■ 오라클 3-WAY JOIN

```
SQL> SELECT e.last_name, l.city
  FROM hr.employees e, hr.locations l, hr.departments d
 WHERE e.department_id = d.department_id
   AND d.location_id = l.location_id
   AND e.department_id = 90 ;
```

■ 오라클 카르테시언 프로덕트 처리

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.department_name
  FROM hr.employees e, hr.departments d
 WHERE e.department_id = 90 AND d.department_id = 10 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	DEPARTMENT_NAME
100	King	90	10	Administration
101	Kochhar	90	10	Administration
102	De Haan	90	10	Administration

7 하위질의 (SUBQUERY)를 사용한 데이터 조회.

◆ 학습 목표.

- SELECT문의 [WHERE]절과 [HAVING]절에서 하위질의를 사용하는 기본적인 방법 및 주의할 사항을 학습합니다.

7-1. 하위질의 (SUBQUERY) 개념.

- EMPLOYEES 테이블을 이용하여 last_name이 'Abel' 인 사원이 받는 봉급보다 많이 받는 사원들의 사번, 성, 급여를 구하고 싶습니다. 그런데, 만약 'Abel'의 salary를 모른다면, 위의 요구사항에 해당하는 레코드를 알아내기 위하여 먼저 SELECT-2를 실행하여 'Abel'의 salary를 구한 후, SELECT-2의 WHERE 절에 상수를 입력하여 실행해야 합니다.

SELECT문-1	SELECT문-2
<pre>SQL> SELECT employee_id, last_name, salary FROM hr.employees WHERE salary > ???; --??? 대신 11000</pre>	<pre>SQL> SELECT salary FROM hr.employees WHERE last_name = 'Abel' ; ----- 11000</pre>

- 앞의 SELECT문-1과 SELECT문-2 를, 다음처럼 하나로 합쳐서 작성할 수 있습니다. 아래의 WHERE절에서 괄호 안에 기술된 SELECT문을 "하위질의"라고 하며, 하위질의가 포함된 전체 문장을 **메인쿼리**(또는 Outer-Query)라고 부릅니다.

```
SQL> SELECT employee_id, last_name, salary
      FROM hr.employees
     WHERE salary > (SELECT salary FROM hr.employees WHERE last_name = 'Abel') ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
201	Hartstein	13000
205	Higgins	12008
100	King	24000
101	Kochhar	17000
102	De Haan	17000
108	Greenberg	12008
145	Russell	14000
146	Partners	13500
147	Errazuriz	12000
168	Ozer	11500

10개의 행이 선택됨

- ☞ SQL문에 하위질의가 포함되면, 대부분의 경우, 하위질의의 처리 결과를 메인쿼리에서 전달받아, 최종적으로 메인쿼리가 처리됩니다.
- ☞ 본 단원에서는 WHERE절 및 HAVING절의 조건에서 상수 대신 하위질의를 사용하는 방법을 설명합니다. 오라클 데이터베이스 서비스에서는 테이블이름 대신 하위질의를 사용할 수 있습니다.

7-2. WHERE절 또는 HAVING절에서 상수 대신 하위질의를 사용할 때 주의사항.

- 비교 조건에서 하위질의를 연산자의 오른편에 괄호로 감싸서 위치시킵니다.
- WHERE 절에 기술된 하위질의 안에는 ORDER BY 절을 적으면 안됩니다.

☞ WHERE절 또는 HAVING절에 사용된 하위질의 안에 [ORDER BY절]을 사용하면, 다음처럼 오류가 발생됩니다.

```
SQL> SELECT employee_id, last_name, salary FROM hr.employees
      WHERE salary > (SELECT salary FROM hr.employees WHERE last_name='Abel' ORDER BY 1);
ORA-00907: 누락된 우괄호
00907. 00000 - "missing right parenthesis"
*Cause:
*Action:
6행, 26열에서 오류 발생
```

[실습] HR.EMPLOYEES 테이블로부터, 사원의 성이 'Abel' 인 사원이 받는 봉급보다 많이 받는 사원들의 사번, 성, 급여를 표시하시오. 단 표시결과를 사원의 사번을 기준으로 오름차순으로 정렬하시오.

```
SQL> SELECT employee_id, last_name, salary
      FROM hr.employees
      WHERE salary > (SELECT salary FROM hr.employees WHERE last_name='Abel')
      ORDER BY 1 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
100	King	24000
101	Kochhar	17000
102	De Haan	17000
108	Greenberg	12008
145	Russell	14000
146	Partners	13500
147	Errazuriz	12000
168	Ozer	11500
201	Hartstein	13000
205	Higgins	12008

10개의 행이 선택됨

☞ [ORDER BY절]은 메인쿼리의 제일 마지막에 명시합니다. 하위질의 안에 ORDER BY절을 명시하면 됩니다.

7-3. WHERE절 또는 HAVING절에서 상수 대신 사용된 하위질의의 유형

- 하위질의 앞에 위치한 연산자에 따라서 다음의 2가지 형식의 하위질의로 분류됩니다.

하위질의 형식	설명
단일 행 하위질의 (SINGLE ROW SUBQUERY)	하위질의가 오직 하나의 행에서 값을 반환합니다.
다중 행 하위질의 (MULTIPLE ROW SUBQUERY)	하위질의가 하나 이상의 행에서 값을 반환합니다.

7-3-1. 단일 행 하위질의.

- WHERE절 또는 HAVING절에서 다음의 단순 비교 연산자 뒤에 사용된 하위질의를 단일-행 하위질의 유형이라고 하며, 반드시 하나의 행에서 하나의 값이 반환되어야 오류가 발생되지 않습니다.

☞ 단순 비교 연산자의 다중-행 연산자의 종류 및 기능

같다	크다	크거나 같다	작다	작거나 같다	같지 않다
=	>	>=	<	<=	\neq , !=, ^=

- WHERE절 또는 HAVING절에 사용된 단일 행 하위질의가, 둘 이상의 행으로부터 값을 메인쿼리로 반환하는 경우에는 아래처럼 오류가 발생됩니다.

```
SQL> SELECT employee_id, last_name, salary
   FROM hr.employees
  WHERE salary > (SELECT salary
                  FROM hr.employees
                 WHERE last_name = 'King');

ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
```

```
SQL> SELECT salary
   FROM hr.employees
  WHERE last_name = 'King' ;
-----+
          SALARY
-----+
          10000
          24000
```

```
SQL> SELECT last_name, salary
   FROM hr.employees
  WHERE salary = (SELECT MIN(salary)
                  FROM hr.employees
                 GROUP BY department_id) ;
-- 하위질의가 반환하는 레코드가 12개입니다.

ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.
01427. 00000 - "single-row subquery returns more than one row"
*Cause:
*Action:
```

- 위의 2 번째 SELECT문에서 = 연산자를 IN 연산자로 변경하면, 문장은 오류 없이 실행될 수 있습니다.

단, = 연산자 또는 IN 연산자를 사용하는 결정은, SELECT문의 결과를 사용하는 사용자 입장에서 결정해 주어야 합니다.

- WHERE절 또는 HAVING절에 사용된 단일 행 하위질의가, 메인쿼리로 반환하는 것이 없으면(하위질의의 조건을 만족하는 행이 없거나 또는 행이 있지만 해당 컬럼이 NULL 상태인 경우) 메인쿼리의 결과는 항상 no rows selected 메시지가 표시됩니다.

```
SQL> SELECT last_name, job_id
  FROM hr.employees
 WHERE job_id = (SELECT job_id
                  FROM hr.employees
                 WHERE last_name = 'Haas') ;
선택된 행 없음
```

```
SQL> SELECT job_id
  FROM hr.employees
 WHERE last_name = 'Haas' ;
선택된 행 없음
```

- ☞ 위에서 하위질의에 명시된 조건을 만족하는 행이 없으므로, 메인쿼리로 반환하는 값이 없습니다. 따라서, 최종 결과가 [선택된 행 없음]으로 표시됩니다.

- 하위질의가 메인쿼리로 반환하는 값이 없을 때, 아래와 같은 사용자의 요구사항에 따라, 문제가 될 수도 있습니다.

○ 회사에 사원이 106명이 근무하고 있고, 올해 신입사원 1명을 뽑았는데, 이 신입사원의 근무부서가 아직 결정되지 않았습니다. 즉, 사원테이블에 해당 신입사원에 대한 근무 부서코드에 입력된 데이터가 없습니다(NULL). 사용자가, 사번을 입력했을 때, 해당 사번의 사원과 같은 부서에 근무하는 사원들의 이름과 직책코드를 확인하는 SELECT문이 포함된 프로그램을, 필요하다고 요청했습니다. 단, 신입사원의 사번을 입력했을 때도, 신입사원 1명에 대한 정보가 표시되어야 한다고 요구했습니다.

```
SQL> SELECT last_name, job_id
  FROM hr.employees
 WHERE department_id = (SELECT department_id FROM hr.employees WHERE employee_id=178) ;
선택된 행 없음
```

○ 위에서 사번 178번이 신입사원의 사번입니다. 그런데, 해당사원의 근무 부서코드 값이 없으므로(NULL)으로, 메인쿼리의 최종결과가 [선택된 행 없음]으로 표시됩니다. 신입사원에 대한 성과 직책코드가 표시되어야 하는 사용자의 요구사항에 부합되지 않습니다.

- ☞ 하위질의 사용 시에, 요구될 수 있는 위와 같은 상황을 해결하기 위한 하나의 방법으로, 사원테이블에 신입사원에 대한 데이터(행)를 입력할 때, **근무하는 부서코드 컬럼을 NULL 상태로 입력하지 않고, 데이터유형에 따라 '-' 값이나 0 값을 입력해주면, 사용자가 원하는 결과를 표시할 수 있습니다.**

```
SELECT last_name, job_id
FROM hr.employees
WHERE (CASE WHEN department_id IS NULL THEN 0
           ELSE department_id END) =
      (SELECT (CASE
                  WHEN department_id IS NULL THEN 0
                  ELSE department_id END)
        FROM hr.employees
       WHERE employee_id=178);
```

- ☞ 다른 방법으로는, 아래처럼 NVL()함수를 이용한 SQL문을 작성할 수도 있습니다.

```
SQL> SELECT last_name, job_id
  FROM hr.employees
 WHERE NVL(department_id,0) = (SELECT NVL(department_id,0) FROM hr.employees
                                WHERE employee_id=178);
```

LAST_NAME	JOB_ID	--이 방법은 데이터의 양에 따라 많은 양의 데이터가 처리될 수 있으므로
Grant	SA_REP	--데이터베이스에 부하가 될 수도 있습니다.

[실습] HR.EMPLOYEES 테이블로부터, 90번 부서에서 급여가 가장 작은 사원의 급여와 동일한 급여를 받는 사원의 성과 급여를 표시하시오.

```
SQL> SELECT last_name, salary
   FROM hr.employees
 WHERE salary=(SELECT MIN(salary) FROM hr.employees WHERE department_id=90);

LAST_NAME          SALARY
-----  -----
Kochhar            17000
De Haan            17000
```

[실습] HR.EMPLOYEES 테이블로부터, 부서코드 별 사원의 최소급여를 표시하시오. 단, 부서의 최소급여가 50번 부서에서 근무하는 사원들에 대한 최소급여 보다 커야 합니다.

```
SQL> SELECT department_id, MIN(salary)
   FROM hr.employees
 GROUP BY department_id
 HAVING MIN(salary)>(SELECT MIN(salary) FROM employees WHERE department_id=50);

DEPARTMENT_ID MIN(SALARY)
-----  -----
      100      6900
       30      2500
       20      6000
       70     10000
       90     17000
      110      8300
       40      6500
       80      6100
       10      4400
       60      4200

11개의 행이 선택됨
```

[실습] HR.EMPLOYEES 테이블로부터, 부서코드 별 평균급여가, 가장 큰 부서코드와 그 부서의 평균급여를 표시하시오.

```
SQL> SELECT department_id, AVG(salary)
   FROM hr.employees
 GROUP BY department_id
 HAVING AVG(salary)=(SELECT MAX(AVG(salary)) FROM hr.employees GROUP BY department_id);

DEPARTMENT_ID AVG(SALARY)
-----  -----
      90    19333.3333
```

[실습] HR.EMPLOYEES 테이블로부터, 직책코드 별 평균급여가 가장 작은 직책코드와 그 직책코드의 평균급여를 표시하시오.

```
SQL> SELECT job_id, AVG(salary)
   FROM hr.employees
  GROUP BY job_id
 HAVING AVG(salary)=(SELECT MIN(AVG(salary)) FROM hr.employees GROUP BY job_id);

JOB_ID      AVG(SALARY)
----- -----
PU_CLERK          2780
```

7-3-2. 다중 행 하위질의.

- WHERE절 또는 HAVING절에서, 다음의 ANY, ALL, IN 같은 비교 연산자 뒤에 사용된 하위질의를, 다중 행 하위질의 유형이라고 하며, 하나 이상의 행으로부터 하나 이상의 값이 반환될 수 있습니다.

☞ 다중-행 연산자의 종류 및 기능

여러 값 중 하나	모든 값	여러 값 중 하나와 같다
ANY	ALL	IN

- 위에서 IN 연산자만 단독으로 사용되며, ANY와 ALL 연산자는 단순 비교 연산자와 조합되어 사용됩니다.

- 다음의 표는, WHERE절 및 HAVING절의 조건에서 하위질의와 함께 사용된 다중 행 연산자의 의미를 설명합니다.

조건에서 하위질의와 ANY/ALL 다중 행 연산자	의미
컬럼 <ANY (하위질의)	가장 큰 값보다 작음.
컬럼 >ANY (하위질의)	가장 작은 값보다 큼.
컬럼 <ALL (하위질의)	가장 작은 값보다 작음.
컬럼 >ALL (하위질의)	가장 큰 값보다 큼.

- ☞ 다중 행 하위질의가 반환하는 값들이 100, 200, 300이라고 가정하고, 위의 의미를 스스로 확인해 봅니다.

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 사번, 성, 급여를 표시하시오. 단, 해당 사원의 급여가 [10, 20, 30] 부서에 근무하는 사원들에 대한 근무 부서코드 별 평균급여] 모두 보다 작아야 합니다. 표시되는 결과가 급여를 기준으로 오름차순으로 정렬되어야 합니다.

```
SQL> SELECT employee_id, last_name, salary
   FROM hr.employees
  WHERE salary < ALL (SELECT AVG(salary)
   FROM hr.employees
  WHERE department_id IN (10,20,30)
 GROUP BY department_id)

 ORDER BY 3;
```

EMPLOYEE_ID	LAST_NAME	SALARY
132	Olson	2100
128	Markle	2200
136	Philtanker	2200
135	Gee	2400
127	Landry	2400
140	Patel	2500
119	Colmenares	2500
182	Sullivan	2500
...		

44개의 행이 선택됨

☞ 위의 SELECT문은, 사용된 다중 행 연산자의 의미를 이용하여, 다음처럼 단일 행 하위질의를 사용한 SELECT문으로 작성될 수 있습니다. AVG() 함수 처리결과가 MIN() 함수로 다시 처리되었습니다.

```
SQL> SELECT employee_id, last_name, salary
   FROM hr.employees
  WHERE salary < (SELECT MIN(AVG(salary))
   FROM hr.employees
  WHERE department_id IN (10,20,30)
 GROUP BY department_id )

 ORDER BY 3 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
132	Olson	2100
136	Philtanker	2200
128	Markle	2200
135	Gee	2400
127	Landry	2400
140	Patel	2500
119	Colmenares	2500
131	Marlow	2500
...		

44개의 행이 선택됨

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 사번, 성, 급여를 표시하시오. 단, 해당 사원의 급여가 [10, 20, 30] 부서에 근무하는 사원들에 대한 근무 부서코드 별 평균급여] 중 하나보다 작거나 같아야 합니다. 표시되는 결과가 급여를 기준으로 오름차순으로 정렬되어야 합니다.

```
SQL> SELECT employee_id, last_name, salary
  FROM hr.employees
 WHERE salary <= ANY (SELECT AVG(salary)
  FROM hr.employees
 WHERE department_id IN (10,20,30)
 GROUP BY department_id )
 ORDER BY 1 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
132	Olson	2100
136	Philtanker	2200
128	Markle	2200
127	Landry	2400
135	Gee	2400
191	Perkins	2500
119	Colmenares	2500
140	Patel	2500
144	Vargas	2500
...		

87개의 행이 선택됨

☞ 위의 SELECT문은, 사용된 다중 행 연산자의 의미를 이용하여, 다음처럼 단일 행 하위질의를 사용한 SELECT문으로 작성될 수 있습니다. AVG() 함수 처리결과가 MAX() 함수로 다시 처리되었습니다.

```
SQL> SELECT employee_id, last_name, salary
  FROM hr.employees
 WHERE salary <= (SELECT MAX(AVG(salary))
  FROM hr.employees
 WHERE department_id IN (10,20,30)
 GROUP BY department_id )
 ORDER BY 1 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
104	Ernst	6000
105	Austin	4800
106	Pataballa	4800
107	Lorentz	4200
109	Faviet	9000
110	Chen	8200
111	Sciarra	7700
112	Urman	7800
...		

87개의 행이 선택됨

[실습] HR.EMPLOYEES 테이블로부터, 모든 사원의 사번, 성, 급여를 표시하시오. 단, 해당 사원의 급여가 [10, 20, 30 부서에 근무하는 사원들에 대한 근무 부서코드 별 평균급여] 중 하나와 동일해야 합니다.

```
SQL> SELECT employee_id, last_name, salary
   FROM hr.employees
 WHERE salary IN (SELECT AVG(salary)
   FROM hr.employees
 WHERE department_id IN (10,20,30)
 GROUP BY department_id ) ;
```

EMPLOYEE_ID	LAST_NAME	SALARY
200	Whalen	4400
151	Bernstein	9500
157	Sully	9500
163	Greene	9500

[실습] HR.EMPLOYEES 테이블로부터, 다른 사원의 관리자로 근무하는 모든 사원의 사번, 성, 급여를 표시하시오. 표시되는 결과가 사번을 기준으로 오름차순으로 정렬되어야 합니다.

```
SQL> SELECT employee_id, last_name, salary
   FROM hr.employees
 WHERE employee_id IN (SELECT manager_id FROM hr.employees)
 ORDER BY 1;
```

EMPLOYEE_ID	LAST_NAME	SALARY
100	King	24000
101	Kochhar	17000
102	De Haan	17000
103	Hunold	9000
108	Greenberg	12008
114	Raphaely	11000
120	Weiss	8000
121	Frapp	8200
122	Kaufling	7900
123	Vollman	6500
124	Mourgos	5800
145	Russell	14000
146	Partners	13500
147	Errazuriz	12000
148	Cambrault	11000
149	Zlotkey	10500
201	Hartstein	13000
205	Higgins	12008

18개의 행이 선택됨

- WHERE절 또는 HAVING절에서, NOT IN 연산자 뒤에 다중 행 하위질의가 사용된 경우, 하위질의가 메인쿼리로 반환하는 값에 NULL이 포함되면, 메인쿼리의 최종 처리결과가 [선택된 행 없음]으로 표시되므로 주의해야 합니다. 다음의 실습을 참고합니다.

[실습] HR.EMPLOYEES 테이블로부터, 다른 사원의 관리자로 근무하지 않는 모든 사원의 사번, 성, 급여 및 관리자 사번을 표시하시오.

```
SQL> SELECT employee_id, last_name, salary, manager_id
  FROM hr.employees
 WHERE employee_id NOT IN (SELECT manager_id FROM hr.employees) ;
```

선택된 행 없음

☞ 위와 같이 처리결과가 표시된 이유는, 사번이 100인 사원의 MANAGER_ID 컬럼에 입력된 값이 없기 때문에(NULL), 하위질의가 반환하는 값에 NULL도 포함되어 있기 때문입니다.

☞ 이를 해결하기 위하여, 메인쿼리의 WHERE절 또는 HAVING절에서 NOT IN 연산자와 하위질의가 같이 사용되는 경우에는, 하위질의가 반환하는 컬럼에 대하여, 하위질의에 [반환컬럼 IS NOT NULL] 조건을 WHERE절에 추가하여, 하위질의가 반환하는 값들로부터 NULL을 배제시킵니다.

[실습] HR.EMPLOYEES 테이블로부터, 다른 사원의 관리자로 근무하지 않는 모든 사원의 사번, 성, 급여 및 관리자 사번을 표시하시오.

```
SQL> SELECT employee_id, last_name, salary, manager_id
  FROM hr.employees
 WHERE employee_id NOT IN (SELECT manager_id FROM hr.employees
                           WHERE manager_id IS NOT NULL)
 ORDER BY 1 ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID	—NOT IN 연산자 다음에 사용된 하위질의가 값을
198	OConnell	2600	124	—반환하는 컬럼(예제에서 MANAGER_ID 컬럼)에
199	Grant	2600	124	—대하여, 컬럼정의를 확인했을 때, NOT NULL
200	Whalen	4400	101	—제약조건이 없으면, 하위질의에 WHERE절을
202	Fay	6000	201	—사용하여 [반환컬럼 IS NOT NULL] 조건을
203	Mavris	6500	101	—포함시킵니다.
204	Baer	10000	101	
206	Gietz	8300	205	
104	Ernst	6000	103	
105	Austin	4800	103	
106	Pataballa	4800	103	
107	Lorentz	4200	103	
109	Faviet	9000	108	
110	Chen	8200	108	
...				

89개의 행이 선택됨

8 SET 연산자 (SET OPERATOR)의 활용.

◆ 학습 목표.

■ 아래의 SET 연산자를 이용하여 두 개의 SELECT문들의 결과 레코드들을 처리하는 방법을 학습합니다.

- UNION 및 UNION ALL
- INTERSECT
- MINUS

[참고] 본 단원에 실습에 사용되는 테이블 소개.

HR.EMPLOYEES 테이블	HR.JOB_HISTORY 테이블
<pre>SQL> desc hr.employees Name Null? Type ----- ----- ----- EMPLOYEE_ID NOT NULL NUMBER(6) FIRST_NAME VARCHAR2(20) LAST_NAME NOT NULL VARCHAR2(25) EMAIL NOT NULL VARCHAR2(25) PHONE_NUMBER VARCHAR2(20) HIRE_DATE NOT NULL DATE JOB_ID NOT NULL VARCHAR2(10) SALARY NUMBER(8,2) COMMISSION_PCT NUMBER(2,2) MANAGER_ID NUMBER(6) DEPARTMENT_ID NUMBER(4)</pre>	<pre>SQL> desc hr.job_history Name Null? Type ----- ----- ----- EMPLOYEE_ID NOT NULL NUMBER(6) START_DATE NOT NULL DATE END_DATE NOT NULL DATE JOB_ID NOT NULL VARCHAR2(10) DEPARTMENT_ID NUMBER(4)</pre>
<pre>SELECT employee_id, job_id, department_id FROM hr.employees WHERE employee_id in (101,102,114,122,176,200,201) ORDER BY 1;</pre>	<pre>SELECT employee_id, job_id, department_id FROM hr.job_history ORDER BY 1;</pre>
<pre>EMPLOYEE_ID JOB_ID DEPARTMENT_ID ----- ----- ----- 101 AD_VP 90 102 AD_VP 90 114 PU_MAN 30 122 ST_MAN 50 176 SA_REP 80 200 AD_ASST 10 201 MK_MAN 20</pre>	<pre>EMPLOYEE_ID JOB_ID DEPARTMENT_ID ----- ----- ----- 101 AC_ACCOUNT 110 101 AC_MGR 110 102 IT_PROG 60 114 ST_CLERK 50 122 ST_CLERK 50 176 SA_REP 80 176 SA_MAN 80 200 AD_ASST 90 200 AC_ACCOUNT 90 201 MK_REP 20</pre>

☞ HR.EMPLOYEES 테이블에는 현재 근무하고 있는 사원들의 현재직책에 대한 정보를 제공합니다.

☞ HR.JOB_HISTORY 테이블에는 현재 근무하는 사원들의 과거직책에 대한 정보를 제공합니다.

8-1. SET 연산자 개요.

■ UNION ALL, UNION, INTERSECT, MINUS 의 4 종류의 SET 연산자가 존재합니다.

■ SET 연산자는 **두 개의 SELECT문 사이에 위치합니다.**

■ 각각의 SET 연산자는 다음의 설명처럼 동작합니다.

SET 연산자	동작 방법
UNION	UNION 연산자의 앞뒤에 명시된 2개의 SELECT문을 처리하여 결과레코드 집합을 각각 구한 후, 결과레코드 집합들을 하나로 합칩니다. 그리고, 합쳐진 결과레코드들을 첫 번째 필드를 기준으로 정렬한 뒤, 중복된 레코드 제거하여, 최종적인 결과레코드들을 표시합니다.
UNION ALL	UNION ALL 연산자의 앞뒤에 명시된 2개의 SELECT문을 처리하여 결과레코드 집합을 각각 구한 후, 이 결과레코드 집합들을 하나로 합쳐서 그대로 표시합니다. UNION 연산자와는 다르게, 정렬작업이 수행되지 않으며, 중복된 레코드들도 제거되지 않습니다.
INTERSECT	INTERSECT 연산자의 앞뒤에 명시된 2개의 SELECT문들을, 첫 번째 필드를 기준으로 정렬시키면서 결과레코드를 각각 구한 후, 2 개의 SELECT문 각각의 정렬된 결과레코드 집합들을 비교하여, 두 결과코드에 공통적으로 포함된 결과레코드만 추출하여 한 번만 표시합니다.
MINUS	MINUS 연산자의 앞뒤에 명시된 2개의 SELECT문들을, 첫 번째 필드를 기준으로 정렬시키면서 결과코드를 각각 구한 후, 2 개의 SELECT문 각각의 정렬된 결과레코드 집합들을 비교하여, 앞의 결과코드로부터 뒤의 결과코드에 공통적으로 포함된 결과레코드를 제거하여, 최종적으로 앞의 SELECT문의 결과에만 고유하게 포함된 결과레코드만 표시합니다.

☞ UNION ALL 연산자를 제외하고, 나머지 SET연산자는 처리 중에 정렬 작업(SORT)이 발생하므로 처리할 레코드의 양에 따라서 메모리 소모가 많을 수 있습니다.

8-2. SET 연산자 사용 시 고려사항.

■ SET 연산자의 앞뒤에 있는 SELECT문에 명시된 컬럼의 개수와 위치에 따른 데이터유형이 모두 일치해야 합니다.

☞ 두 SELECT 문의 컬럼개수와 데이터유형이 일치 하지 않을 경우 아래처럼 오류가 발생됩니다.

```
SQL> SELECT department_id, hire_date FROM hr.employees
      UNION
      SELECT department_id, location_id FROM hr.departments;
```

ORA-01790: 대응하는 식과 같은 데이터유형이어야 합니다
 01790. 00000 - "expression must have same datatype as corresponding expression"
 *Cause:
 *Action:
 1행, 23열에서 오류 발생

- 표시된 결과 상의 머리글(HEADING)은 항상 첫 번째 SELECT문의 컬럼이름이나 표현식이 표시됩니다. 따라서, 컬럼 별칭을 정의하여 머리글을 변경하려면, 첫 번째 SELECT문에 컬럼별칭을 기술해야 합니다.
- 3 개 이상의 SELECT문들을 여러 개의 SET 연산자를 이용하여 처리할 때, 위에서부터 아래로 순서대로 처리됩니다. 이러한 기본 처리 순서는, 괄호를 이용하여 처리 순서가 변경될 수 있습니다.
- SET-연산자가 포함된 문장에서 [ORDER BY절]은 전체 문장의 맨 마지막에 기술해야 하며, 이 때, 첫 번째 SELECT문에 명시된 컬럼표현식, 컬럼별칭 또는 SELECT절의 컬럼 순서를 숫자로 명시한 위치 표기법을 사용합니다.

☞ 아래처럼, [ORDER BY절]을 전체 문장의 맨 마지막이 아닌 다른 위치에 기술하면 오류가 발생합니다.

```
SQL> SELECT employee_id, last_name FROM hr.employees ORDER BY last_name
      UNION ALL
      SELECT employee_id, job_id FROM hr.job_history ;
```

ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다
 00933. 00000 - "SQL command not properly ended" --ORDER BY 절이 UNION ALL 연산자의 앞에 있는
 *Cause: --SELECT문에 명시되었습니다.
 *Action:

2행, 6열에서 오류 발생

```
SQL> SELECT employee_id, job_id FROM hr.employees WHERE department_id=90
      UNION
      (  SELECT employee_id, job_id FROM hr.job_history
      MINUS
      SELECT employee_id, job_id FROM hr.employees WHERE department_id = 90 ORDER BY 1);
```

ORA-00907: 누락된 우괄호
 00907. 00000 - "missing right parenthesis" --ORDER BY 절이 UNION 연산자의 뒤에 있는 SELECT문의 괄호 안에
 *Cause: --명시되었습니다.
 *Action:
 11행, 6열에서 오류 발생

[실습] 다음의 요구조건에 적합한 SELECT문을 SET 연산자를 이용하여 작성하시오.

- (1) [HR.EMPLOYEES 테이블로부터, 90번 부서에 근무하는 사원들의 사번과 직책코드로 구성된 레코드],
- (2) [HR.JOB_HISTORY 테이블로부터, 직책이 바뀐 모든 사원의 사번과 과거 직책코드로 구성된 레코드],
- (3) [HR.EMPLOYEES 테이블로부터, 90번 부서에 근무하는 사원들의 사번과 직책코드로 구성된 레코드]
- (4) [단계(1)과 단계(2)의 합쳐진 결과에서 중복된 레코드가 제거된 처리레코드]

위의 결과레코드들에 대하여 단계(4)의 결과레코드로부터 단계(3)의 결과레코드와 동일한 레코드를 제거된 최종레코드를 첫 번째 필드를 기준으로 오름차순으로 정렬해서 표시하시오.

```
SQL> SELECT employee_id, job_id FROM hr.employees WHERE department_id=90
      UNION
      SELECT employee_id, job_id FROM hr.job_history
      MINUS
      SELECT employee_id, job_id FROM hr.employees WHERE department_id = 90
      ORDER BY 1;
```

EMPLOYEE_ID JOB_ID

101	AC_ACCOUNT	--첫 번째 SELECT 문과 두 번째 SELECT문에 대한 UNION 연산자가 처리된 후
101	AC_MGR	--세 번째 SELECT 문과 MINUS 연산자 처리가 수행됩니다.
102	IT_PROG	--최종 결과레코드들이 첫 번째 필드를 가지고 정렬되어 표시됩니다.
114	ST_CLERK	
122	ST_CLERK	
176	SA_MAN	
176	SA_REP	
200	AC_ACCOUNT	
200	AD_ASST	
201	MK_REP	

10개의 행이 선택됨

[실습] 다음의 요구조건에 적합한 SELECT문을 SET 연산자를 이용하여 작성하시오.

- (1) [HR.EMPLOYEES 테이블로부터, 90번 부서에 근무하는 사원들의 사번과 직책코드로 구성된 레코드],
- (2) [HR.JOB_HISTORY 테이블로부터, 직책이 바뀐 모든 사원의 사번과 과거 직책코드로 구성된 레코드],
- (3) [HR.EMPLOYEES 테이블로부터, 90번 부서에 근무하는 사원들의 사번과 직책코드로 구성된 레코드]
- (4) [단계(2)의 결과레코드로부터 단계(3)의 결과레코드 중 동일한 레코드가 제거된 레코드]

위의 결과레코드들에 대하여 단계(1)과 단계(4)의 합쳐진 결과에서 중복된 레코드가 제거된 최종레코드를, 첫 번째 필드를 기준으로 오름차순으로 정렬해서 표시하시오.

```
SQL> SELECT employee_id, job_id FROM hr.employees WHERE department_id=90
      UNION
      ( SELECT employee_id, job_id FROM hr.job_history
        MINUS
        SELECT employee_id, job_id FROM hr.employees WHERE department_id = 90 )
      ORDER BY 1 ;
```

EMPLOYEE_ID JOB_ID

100 AD_PRES	--두 번째 SELECT문과 세 번째 SELECT문의 MINUS 연산자 처리가 먼저 수행된 후,
101 AC_ACCOUNT	--첫 번째 SELECT문과의 UNION 처리가 수행됩니다.
101 AC_MGR	--최종 결과레코드들이 첫 번째 필드를 가지고 정렬되어 표시됩니다.
101 AD_VP	
102 AD_VP	
102 IT_PROG	
114 ST_CLERK	
122 ST_CLERK	
176 SA_MAN	
176 SA REP	
200 AC_ACCOUNT	
200 AD_ASST	
201 MK_REP	

13개의 행이 선택됨

[실습] (1) HR.EMPLOYEES 테이블로부터, 사번이 101, 176, 200, 201인 사원의 사번과 직책코드로 구성된 레코드와
 (2) HR.JOB_HISTORY 테이블로부터, 사번이 101, 176, 200, 201인 사원의 사번과 과거 직책코드로 구성된 레코드를
 이용하여, (1) 및 (2) 결과레코드가 모두 합쳐진 결과에서, 중복된 레코드가 제거된 최종레코드를 표시하시오.

```
SQL> SELECT employee_id, job_id FROM hr.employees WHERE employee_id IN (101,176,200,201)
      UNION
      SELECT employee_id, job_id FROM hr.job_history WHERE employee_id IN (101,176,200,201);

EMPLOYEE_ID JOB_ID
-----
101 AC_ACCOUNT
101 AC_MGR
101 AD_VP
176 SA_MAN
176 SA_REP
200 AC_ACCOUNT
200 AD_ASST
201 MK_MAN
201 MK_REP

9개의 행이 선택됨
```

[실습] (1) HR.EMPLOYEES 테이블로부터, 사번이 101, 176, 200, 201인 사원의 사번과 직책코드로 구성된 레코드와
 (2) HR.JOB_HISTORY 테이블로부터, 사번이 101, 176, 200, 201인 사원의 사번과 과거 직책코드로 구성된 레코드를
 이용하여, (1) 및 (2) 결과레코드를 합쳐서 표시하되, 첫 번째 값을 기준으로 정렬된 최종레코드를 표시하시오.

```
SQL> SELECT employee_id, job_id FROM hr.employees WHERE employee_id IN (101,176,200,201)
      UNION ALL
      SELECT employee_id, job_id FROM hr.job_history WHERE employee_id IN (101,176,200,201)
      ORDER BY 1 ;

EMPLOYEE_ID JOB_ID
-----
101 AC_ACCOUNT
101 AD_VP
101 AC_MGR
176 SA_MAN
176 SA_REP
176 SA_REP
176 SA_REP
200 AD_ASST
200 AC_ACCOUNT
200 AD_ASST
201 MK_MAN
201 MK_REP

11개의 행이 선택됨
```

[실습] HR.EMPLOYEES 테이블과 HR.JOB_HISTORY 테이블로부터, 현재 근무하는 사원 중, 과거에 수행했던 직책코드를 현재 다시 수행하고 있는 사원의 사번과 해당 직책코드를 표시하시오.

```
SQL> SELECT employee_id, job_id FROM hr.employees
      INTERSECT
      SELECT employee_id, job_id FROM hr.job_history ;

EMPLOYEE_ID JOB_ID
----- -----
    176 SA_REP
    200 AD_ASST
```

[실습] HR.EMPLOYEES 테이블과 HR.JOB_HISTORY 테이블로부터, 현재 근무하는 사원들 중, 입사한 후 한 번도 직책이 바뀌지 않은 모든 사원의 사번을 표시하시오.

```
SQL> SELECT employee_id FROM hr.employees
      MINUS
      SELECT employee_id FROM hr.job_history;

EMPLOYEE_ID
-----
    100
    103
    104
    105
    106
    107
    108
    109
    110
...
100개의 행이 선택됨
```

```
SELECT employee_id, job_id, salary, department_id
FROM hr.employees
WHERE employee_id IN (SELECT employee_id FROM hr.employees
                      MINUS
                      SELECT employee_id FROM hr.job_history) ;
```

- [실습] (1) HR.EMPLOYEES 테이블로부터, 사원의 근무부서와 입사일로 구성된 레코드와
 (2) HR.DEPARTMENTS 테이블로부터, 부서코드와 위치코드로 구성된 레코드를
 이용하여, (1) 및 (2) 결과레코드를 합쳐서 표시하되, 중복된 레코드를 제거한 최종레코드를 표시하시오.

```
SQL> SELECT department_id, TO_NUMBER(null) AS "LOCATION_ID", hire_date FROM hr.employees
UNION
SELECT department_id, location_id, TO_DATE(null) FROM hr.departments;
```

DEPARTMENT_ID	LOCATION_ID	HIRE_DATE
10	1700	03/09/17
10	1800	04/02/17
20	1700	05/08/17
30	1700	02/12/07
30	1700	03/05/18
30	1700	05/07/24
30	1700	05/12/24
30	1700	06/11/15
30	1700	07/08/10
40	2400	02/06/07
40	2400	02/06/07
...		

130개의 행이 선택됨

--첫 번째 SELECT문에는, 두 번째 SELECT문에 있는 LOCATION_ID 컬럼에
--해당하는 숫자 데이터유형의 컬럼이 없으므로,
--이를 TO_NUMBER(NULL) 표현식을 이용하여 일치시켰습니다.
--두 번째 SELECT문에는, 첫 번째 SELECT문에 있는 HIRE_DATE 컬럼에
--해당하는 날짜 데이터유형의 컬럼이 없으므로,
--이를 TO_DATE(NULL) 표현식을 이용하여 일치시켰습니다.
--UNION 연산자가 사용되었으므로, 최종 결과레코드가 첫 번째 필드를
--기준으로 정렬되어 표시됩니다.
--첫 번째 SELECT문에 TO_NUMBER(NULL)에 대한 컬럼별칭이 명시되어,
--결과레코드의 머리글에 표시되었습니다.

- ☞ SET 연산자 사용 시, TO_CHAR(NULL), TO_NUMBER(NULL), TO_DATE(NULL) 함수를 사용하여, 두 SELECT문의 컬럼 개수 및
 데이터유형 불일치를 해결할 수 있습니다. 103

- ☞ 위의 문장에서 TO_CHAR(NULL), TO_NUMBER(NULL), TO_DATE(NULL) 함수 대신 다음처럼 NULL 키워드를 명시하여,
 동일한 결과를 표시할 수 있습니다.

```
SQL> SELECT department_id, NULL AS "LOCATION_ID", hire_date FROM hr.employees
UNION
SELECT department_id, location_id, NULL FROM hr.departments;
```

DEPARTMENT_ID	LOCATION_ID	HIRE_DATE
10	1700	03/09/17
10	1800	04/02/17
20	1700	05/08/17
30	1700	02/12/07
30	1700	03/05/18
30	1700	05/07/24
30	1700	05/12/24
30	1700	06/11/15
30	1700	07/08/10
40	2400	02/06/07
40	2400	02/06/07
...		

130개의 행이 선택됨

[실습] (1) HR.EMPLOYEES 테이블로부터, 사원의 사번, 직책코드 및 급여 구성된 레코드와
 (2) HR.JOB_HISTORY 테이블로부터, 사번, 과거 직책코드로 구성된 레코드를
 이용하여, (1) 및 (2) 결과레코드를 합쳐서 표시하되, 중복된 레코드를 제거한 최종레코드를 표시하시오.

```
SQL> SELECT employee_id, job_id, salary FROM hr.employees
      UNION
      SELECT employee_id, job_id, 0 FROM hr.job_history;
```

EMPLOYEE_ID	JOB_ID	SALARY
100	AD_PRES	24000
101	AC_ACCOUNT	0
101	AC_MGR	0
101	AD_VP	17000
102	AD_VP	17000
102	IT_PROG	0
103	IT_PROG	9000
104	IT_PROG	6000
105	IT_PROG	4800
106	IT_PROG	4800
107	IT_PROG	4200
108	FI_MGR	12008
109	FI_ACCOUNT	9000
110	FI_ACCOUNT	8200
111	FI_ACCOUNT	7700
112	FI_ACCOUNT	7800
113	FI_ACCOUNT	6900
114	PU_MAN	11000
114	ST_CLERK	0
115	PU_CLERK	3100
116	PU_CLERK	2900
117	PU_CLERK	2800
118	PU_CLERK	2600
119	PU_CLERK	2500
120	ST_MAN	8000
121	ST_MAN	8200
...		

117개의 행이 선택됨

☞ SET 연산자 사용 시, 적절한 상수를 이용하여 SELECT 문들 간, 컬럼의 개수 및 데이터유형 불일치를 해결할 수도 있습니다.

9 사용자 데이터의 수정(DATA MANIPULATION LANGUAGE, DML).**◆ 학습 목표.**

- 다음의 SQL문을 이용하여 테이블에 데이터를 수정하는 방법을 학습합니다.

- INSERT INTO VALUES문**
- 하위질의를 이용한 데이터입력
- UPDATE문**
- DELETE문**
- MERGE문
- 다중 테이블 데이터 삽입
- TRUNCATE문(DDL문)

- **트랜잭션의 의미**와 오라클 데이터베이스 서비스에서의 트랜잭션 처리 특징에 대하여 학습합니다.

가

9-1. DML문 개요 및 트랜잭션에 대한 이해

- 오라클 데이터베이스의 테이블에 아래의 작업을 수행하는 문들을 모두 DML문이라고 합니다.

INSERT문	새로운 행(ROW) 또는 행들을 테이블에 입력합니다.
UPDATE문	테이블에 저장된 행 또는 행들의 컬럼(들)의 데이터를 수정합니다.
DELETE문	테이블에 저장된 행 또는 행들을 삭제합니다.
MERGE문	UPDATE 및 INSERT를 동시에 수행합니다.
다중 테이블 행 입력	한 번에 여러 테이블에 행 또는 행들을 입력합니다.

가

[참고] 트랜잭션의 의미 및 트랜잭션을 정상적으로 종료하는 COMMIT과 ROLLBACK 문.

- 다음의 설명을 통해 트랜잭션의 의미를 확인합니다.

은행 데이터베이스에서, 은행 고객이 예금 계좌에서 결제 계좌로 현금을 이체하는 경우, 트랜잭션은, 예금 계좌 감소와 결제 계좌 증가 및 로그-저널에 로그-기록이라는 세 가지의 개별적인 데이터 수정 작업(DML문들)들로 구성됩니다. 데이터베이스에서 세 개의 DML문이 수행될 때, 계좌 잔액이 정확히 유지되어야 합니다. 특정 문제로 인해 트랜잭션을 구성하는 DML문 중 하나가 실행되지 못하면, 트랜잭션을 구성하는 다른 DML문에 의한 데이터 수정 작업을 포함하여, 전체 데이터 수정작업이, 수행되기 전의 상태로 데이터를 다시 변경한 후 작업이 완료되어야 합니다(이를, 룰백 또는 UNDO라고 함). 즉, 트랜잭션을 구성하는 모든 DML문들이 모두 성공적으로 완료되었을 때만, 데이터 수정 후의 상태가 유지되면서 하나의 작업단위인 트랜잭션이 완료되어야 합니다. 이렇게 DML문(들)들로 구성된 데이터 수정의 논리적 작업 단위 하나를 하나의 트랜잭션(Transaction)이라고 합니다.

- 다음은 트랜잭션을 정상적으로 종료하기 위하여 사용자가 사용하는 두 가지의 SQL문입니다.

COMMIT 문	DML문장 완료 후, 변경 후의 데이터 상태를 유지하면서 트랜잭션을 종료합니다.
ROLLBACK 문	DML문장 완료 후, 변경 전의 데이터 상태로 되돌려 놓고 트랜잭션을 종료합니다.

9-2. 데이터베이스의 특정 테이블에 한 행을 삽입하기

- `INSERT INTO...` `VALUES` 문을 이용하여, 한 번에 하나의 행을 하나의 테이블에 입력할 수 있습니다.

MySQL/Maria VALUES 2
VALUES ,

- 기본 문법

INSERT INTO 테이블이름(컬럼1, 컬럼2,...) **--VALUES** 절은 **INSERT INTO** 절 다음에 위치해야 합니다.

VALUES (값1, '값2', ...);

- INSERT INTO 절에는, 행이 입력되는 테이블이름 및 괄호 안에 콤마로 구분하여 컬럼이름들을 명시합니다. 괄호 안에 컬럼이름들은 생략될 수 있으며, 컬럼이름들이 생략되면, 테이블에 정의된 컬럼의 순서대로, 모든 컬럼에 값들을 입력해야 합니다.

- VALUES절에는, 괄호 안에, INSERT INTO절에 명시한 컬럼의 순서대로, 입력하는 값을 콤마로 구분하여 명시합니다. VALUES절에 값을 명시할 때, 숫자형식의 값은 그대로 기술하고, 문자형식의 값은 작은 따옴표(')로 감싸줍니다. DATE 데이터유형의 값은 TO_DATE() 함수로 처리하여 입력합니다.

[실습] HR.DEPARTMENTS 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.departments(department_id, department_name, manager_id, location_id)
      VALUES (310, 'Oracle SQL', 100, 1700);
```

1개 행 이(가) 삽입되었습니다.

- ☞ INSERT INTO...VALUES 문을 이용하여 테이블에 한 행을 입력할 때는, 테이블에 정의된 모든 컬럼을 고려하고, 테이블에 정의된 컬럼의 순서대로 값을 입력하는 것을 권장합니다.

[실습] HR.DEPARTMENTS 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.departments VALUES (320, 'Oracle Administration', 100, 1700);
```

1개 행 이(가) 삽입되었습니다.

- ☞ INSERT INTO절에 컬럼이름이 생략되었습니다. 테이블의 모든 컬럼을 대상으로 값이 입력되어야 합니다.

HR.DEPARTMENTS 테이블에 정의된 컬럼의 순서는, [describe 테이블이름] 명령어로 확인할 수 있습니다.

[실습] HR.DEPARTMENTS 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.departments (department_id, department_name)
      VALUES (330, 'Oracle Maintenance');
```

1개 행 이(가) 삽입되었습니다.

- ☞ INSERT INTO절에서, 테이블이름 다음에 컬럼이름을 일부만 명시하여 행을 입력하면, 명시되지 않는 컬럼(들)은 NULL 상태로 한 행이 입력됩니다.

[실습] HR.DEPARTMENTS 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.departments VALUES (340, 'Oracle PLSQL', NULL, NULL);
```

1개 행 이(가) 삽입되었습니다.

- ☞ VALUES절에 값 대신 NULL 키워드를 명시하면, 해당 컬럼은 값이 없는 상태로 행이 입력됩니다. NULL 키워드 대신, 작은따옴표(')를 연달아 2번 표기하여, 값이 없는 상태로 입력할 수 있습니다.

- ☞ 사용자가 일부 컬럼에 값을 지정하지 않은 경우, NULL 키워드를 이용하여 테이블에 행을 입력하는 것을 권장합니다.

[실습] HR.EMPLOYEES 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.employees (employee_id, first_name, last_name, email, phone_number,
   hire_date, job_id, salary, commission_pct, manager_id, department_id)
   VALUES (301, INITCAP('LOUIS'), INITCAP('POPP'), 'LPOPP1', '515.124.4567',
   SYSDATE, 'AC_ACCOUNT', 6900, NULL, 205, 100);
```

1개 행 이(가) 삽입되었습니다.

TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH LOCAL TIME ZONE

SYSTIMESTAMP,
LOCALTIMESTAMP,
CURRENT_TIMESTAMP

☞ FIRST_NAME, LAST_NAME, HIRE_DATE, 컬럼에 대하여, 함수를 이용하여 각각의 값을 처리한 후, 함수처리 결과로 반환된 값을 입력합니다.

```
TIMESTAMP/TIMESTAMP WITH LOCAL TIME ZONE: TO_TIMESTAMP('2020-11-20 09:45:55.435',
   'YYYY-MM-DD HH24:MI:SSXFF')
TIMESTAMP WITH TIME ZONE: TO_TIMESTAMP_TZ('2020-11-20 09:45:56.234 +09:00',
   'YYYY-MM-DD HH24:MI:SSXFF TZR')
```

[실습] HR.EMPLOYEES 테이블에 대하여 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.employees VALUES
   (302, UPPER('Den'), UPPER('Raphealy'), 'hong@yahoo.co.kr', '515.127.4561',
   TO_DATE('2월 3, 1999', 'MON DD, YYYY', 'NLS_DATE_LANGUAGE=KOREAN'), 'AC_ACCOUNT',
   11000, NULL, 100, 30);
```

1개 행 이(가) 삽입되었습니다.

☞ 테이블의 DATE 데이터유형의 컬럼에 직접 날짜 형식의 값을 입력하는 경우에는 반드시 **TO_DATE()** 함수를 이용하여, 날짜형식의 값을 처리하여 값을 입력해야 합니다. DATE 데이터유형의 값을 **TO_DATE()** 함수로 처리하지 않은 경우, 세션의 날짜 표시 형식이, 입력되는 날짜의 형식과 다르면, 아래처럼 오류가 발생되거나 잘못된 값이 입력될 수 있습니다.

```
SQL> INSERT INTO hr.employees VALUES (303, UPPER('Den'), UPPER('Raphealy'),
   'hong@yahoo.co.kr', '515.127.4561', '2월 3 1999', 'AC_ACCOUNT', 11000, NULL, 100, 30);
```

명령의 12 행에서 시작하는 중 오류 발생 -

```
INSERT INTO hr.employees VALUES (303, UPPER('Den'), UPPER('Raphealy'),
   'hong@yahoo.co.kr', '515.127.4561', '2월 3 1999', 'AC_ACCOUNT', 11000, NULL, 100, 30)
```

오류 보고 -

SQL 오류: ORA-01861: 리터럴이 형식 문자열과 일치하지 않음

01861. 00000 - "literal does not match format string"

*Cause: Literals in the input must be the same length as literals in the format string (with the exception of leading whitespace). If the "FX" modifier has been toggled on, the literal must match exactly, with no extra whitespace.

*Action: Correct the format string to match the literal.

[실습] 지금까지 수행한 데이터 입력작업들을 룰백(UNDO)시키고, 트랜잭션을 종료합니다.

```
SQL> ROLLBACK ;
```

룰백이 완료되었습니다.

☞ ROLLBACK 문을 실행시키면, 지금까지, 접속 세션에서 수행했던 모든 데이터 입력 작업들로 구성된 하나의 트랜잭션을, 행들이 입력되기 전의 상태로 변경시킨 후, 접속 세션에서 수행한 트랜잭션을 정상적으로 종료(트랜잭션-룰백)합니다.

[참고] 다음과 같은 경우, INSERT문 처리 시에, 오류가 발생됩니다.

- NOT NULL 제약조건이 지정된 컬럼에, [값이 없는 상태]로 입력을 시도하면, 오류가 발생됩니다.
- UNIQUE, 기본키 제약조건을 위반하는, 중복된 값을 입력하려고 시도하면, 오류가 발생됩니다.
- CHECK 제약조건을 위반하는 값을 입력하려고 시도하면, 오류가 발생됩니다.
- 외래키 제약조건을 위반하는 값을 입력하려고 시도하면, 오류가 발생됩니다.
- 컬럼과 데이터유형이 다른 값을 입력하려고 시도하면, 오류가 발생됩니다..
- 컬럼에 설정된 값의 최대 길이보다 길이가 긴 값을 입력하려고 시도하면, 오류가 발생됩니다..

 제약조건(CONSTRAINT)에 대해서는 제10장에서 자세히 설명합니다.

9-3. 하위질의를 이용한 데이터 입력

- INSERT문장에서, VALUES절 대신, 하위질의를 사용하여, 다른 테이블의 데이터를 복사하여, 테이블에 다수의 행을 입력할 수 있습니다. 즉, 하위질의를 사용하여, 하위질의의 처리결과로 반환된 0 개 이상의 행들의 값들을, 테이블에 입력할 수 있습니다.

[실습] 다음에 있는 일련의 실습을 통해 하위질의를 이용한 데이터 입력을 실습합니다.

1> 다음의 CREATE TABLE문을 실행하여, HR.SALES_REPS 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.sales_reps (
    id NUMBER(6) PRIMARY KEY,
    name VARCHAR2(30) NOT NULL,
    salary NUMBER(8,2),
    comm NUMBER(8,2),
    email VARCHAR2(30) UNIQUE );
```

table HR.SALES_REPS이(가) 생성되었습니다.

2> 하위질의를 이용하여 HR.EMPLOYEES 테이블의 데이터를 복사하여 HR.SALES_REPS 테이블에 입력합니다..

```
SQL> INSERT INTO hr.sales_reps (id, name, salary, comm, email)
    SELECT employee_id, last_name, salary, commission_pct, email
    FROM hr.employees
    WHERE job_id LIKE '%REP%' ;
```

33개 행 이(가) 삽입되었습니다.

- ☞ 하위질의의 WHERE절을 만족하는 행에서 추출된 레코드가 테이블에 행으로 입력됩니다. 하위질의의 WHERE절을 만족하는 행이 없는 경우에는, [0개 행 이(가) 삽입되었습니다.]라는 메시지가 표시됩니다.
- ☞ 하위질의의 SELECT절에 기술된 컬럼 개수와 순서가, INSERT INTO 절의 테이블이름 다음에 기술된 컬럼 개수 및 순서와 동일해야 합니다.

3> HR.SALES_REPS 테이블에 대한 데이터 입력작업의 트랜잭션을 커밋합니다.

SQL> **COMMIT ;**

커밋되었습니다.

- ☞ COMMIT 문을 실행시키면, HR 접속 세션에서 수행된 데이터 입력 작업(들)의 트랜잭션을, 행들이 입력된 후의 상태를 유지하면서 정상적으로 종료([트랜잭션-커밋](#))합니다.

9-4. UPDATE문을 이용한 테이블의 기존 행 데이터 수정

- UPDATE...SET...WHERE 문을 이용하여, WHERE 절을 만족하는 테이블의 행들의 값을 수정할 수 있습니다.

■ 기본 문법

UPDATE 소유자명.테이블이름 --SET절은 UPDATE절 다음에 위치해야 하며, WHERE절은 SET절 다음에 명시합니다.

SET 컬럼1=값, 컬럼2=값,...

WHERE 조건;

- UPDATE절에는, 테이블이름을 명시합니다.
- SET절에, 콤마로 구분하여, 컬럼=값 형태로, 값이 수정될 컬럼과 값을 필요한 만큼 명시합니다. 명시되는 값이 숫자인 경우, 값을 그대로 기술하고, 문자형식의 값은 작은 따옴표(')로 감싸줍니다. DATE 데이터유형의 값은 TO_DATE() 함수로 처리하여 기술합니다.
- WHERE절을 만족하는 모든 행의 값이 수정됩니다. WHERE절은 생략될 수 있으며, 생략된 경우, 테이블의 모든 행이 수정됩니다.

[실습] 다음에 있는 일련의 실습을 통해 UPDATE문을 사용하는 방법을 학습합니다.

1> 다음의 CREATE TABLE문을 실행하여, HR.COPY_EMP 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.copy_emp (
  eid NUMBER(6) PRIMARY KEY,
  ename VARCHAR2(60) NOT NULL,
  esal NUMBER(8,2),
  ehiredate DATE,
  ejobid VARCHAR2(20),
  edeptid NUMBER(4) );
```

table HR.COPY_EMP이(가) 생성되었습니다.

2> 하위질의 입력을 이용하여, HR.EMPLOYEES 테이블의 데이터를 복사하여, HR.COPY_EMP 테이블에 입력하고,

해당 트랜잭션을 커밋합니다.

```
SQL> INSERT INTO hr.copy_emp(eid, ename, esal, ehiredate, ejobid, edeptid)
  SELECT employee_id, UPPER(last_name), salary, hire_date, job_id, department_id
  FROM hr.employees;
```

107개 행 이(가) 삽입되었습니다.

```
SQL> COMMIT;
```

커밋되었습니다.

3> HR.COPY_EMP 테이블에 대하여, 다음의 UPDATE문을 수행하여, eid가 113인 행을 값들을 수정합니다.

```
SQL> UPDATE hr.copy_emp
  SET esal=8500,
      ename=UPPER('Shin'),
      ehiredate=TO_DATE('FEB 3, 1999', 'MON DD, YYYY', 'NLS_DATE_LANGUAGE=AMERICAN'),
      edeptid=NULL
  WHERE eid = 113;
--DATE 데이터유형의 컬럼을 수정할 때, 명시된 날짜 값이 UPDATE문을 요청한
--세션의 날짜 표시 형식과 다를 때, 오류가 발생되는 것을 방지하기 위하여
--날짜 값을 TO_DATE() 함수로 처리하십시오.
```

1개 행 이(가) 업데이트되었습니다.

```
SQL> COMMIT;
```

커밋되었습니다.

☞ HR.COPY_EMP 테이블에 저장된, EID 컬럼의 값이 113 인 행에서, ESAL 컬럼의 값을 8500으로, ENAME 컬럼의 값은 UPPER('SHIN') 함수로 처리된 'SHIN' 으로, EHIREDATE 컬럼의 값은 '1999/02/03' 날짜 값으로, EDEPTID 컬럼은 NULL 상태로 각각 변경되어 저장됩니다.

4> 앞의 UPDATE문에 의하여 변경된 데이터를 조회합니다.

```
SQL> SELECT * FROM hr.copy_emp WHERE eid = 113;
```

EID	ENAME	ESAL	EHIREDATE	EJOBID	EDEPTID
113	SHIN	8500	99/02/03	FI_ACCOUNT	

5> HR.COPY_EMP 테이블에 대하여, 다음의 UPDATE문을 실행합니다.

```
SQL> UPDATE hr.copy_emp SET edeptid = 110;
```

107개 행 이(가) 업데이트되었습니다. --WHERE절이 없으므로, 테이블의 모든 행의 edeptid 컬럼값이 수정됩니다.

```
SQL> ROLLBACK; --수정 작업을, 수정하기 전의 상태로 되돌린 후, 트랜잭션을 정상적으로 종료합니다.
```

롤백이 완료되었습니다.

9-5. DELETE문을 이용한 테이블의 기존 행 삭제

■ **DELETE FROM...WHERE** 문을 이용하여, WHERE 절을 만족하는 테이블의 기존 행들을 삭제할 수 있습니다.

■ 기본 문법

```
DELETE FROM 소유자명.테이블이름
```

```
WHERE 조건;
```

○ DELETE FROM 절에는, 테이블이름을 명시합니다. FROM 키워드는 생략할 수 있습니다

○ WHERE절을 만족하는 모든 기존 행이 삭제됩니다. WHERE절은 생략될 수 있으며, 생략된 경우, 테이블의 모든 행이 삭제됩니다.

[실습] HR.COPY_EMP 테이블에 대하여 다음의 DELETE문을 실행합니다.

```
SQL> DELETE FROM hr.copy_emp WHERE edeptid=30; --EDEPTID 컬럼의 값이 30 인 모든 행이 삭제됩니다.
```

6개 행 이(가) 삭제되었습니다.

```
SQL> COMMIT ;
```

커밋되었습니다.

[실습] HR.COPY_EMP 테이블에 대하여 다음의 DELETE문을 실행합니다.

```
SQL> DELETE FROM hr.copy_emp; -- WHERE절이 없으므로, 테이블의 모든 행이 삭제됩니다.
```

101개 행이(가) 삭제되었습니다.

```
SQL> ROLLBACK; -- 수정된 데이터를 수정하기 전의 상태로 되돌리고 트랜잭션을 종료합니다.
```

롤백이 완료되었습니다.

☞ UPDATE-문에서 [WHERE 절]을 누락하면, 테이블의 모든 행이 삭제됩니다.

9-6. TRUNCATE TABLE 문을 이용한 테이블 절단

- TRUNCATE문은 테이블의 모든 데이터를 제거한다는 점에서는, WHERE절 없는 DELETE문을 실행하고, 커밋을 수행한 것과 동일합니다. 단, TRUNCATE문은, 기본적으로 테이블의 데이터가 저장되었던 저장공간을 테이블이 처음 생성될 때의 상태로 초기화(테이블 절단) 시켜서, 테이블의 데이터를 제거합니다.
- TRUNCATE문은, DML문이 아니라, DDL(데이터 정의어)문이므로, 실행과 동시에 자동으로 커밋됩니다. 즉, 실행되면 둘백 또는 언두 할 수 없습니다.

[실습] HR.SAELS_REPS 테이블을 TRUNCATE 시키시오.

```
SQL> TRUNCATE TABLE hr.sales_reps ;
```

table HR.SALES_REPS이(가) 절렸습니다.

☞ HR.SALES_REPS 테이블의 저장공간이 처음 생성될 때의 상태로 초기화되어 테이블의 모든 데이터가 제거됩니다.

☞ 다른 테이블에 정의된 외래키 제약조건에 의하여 참조되는, 부모-테이블은 절단(TRUNCATE) 될 수 없습니다.

9-7. DML문에서 하위질의의 사용

[실습] HR.COPY_EMP 테이블에 저장된 EID=200 인 행의 EJOBID 컬럼과 ESAL 컬럼의 값을 다음처럼 수정하시오.

- EID가 206 값인 행의 EJOBID 컬럼의 값으로, EJOBID 컬럼을 수정
- EID가 205 값인 행의 ESAL 컬럼의 값으로, ESAL 컬럼을 수정

```
SQL> UPDATE hr.copy_emp
      SET ejobid=(SELECT ejobid FROM hr.copy_emp WHERE eid=206),
          esal=(SELECT esal FROM hr.copy_emp WHERE eid=205)
      WHERE eid=200;
```

1개 행 이(가) 업데이트되었습니다.

```
SQL> COMMIT ;
```

커밋되었습니다.

☞ UPDATE문의 SET절에, 하위질의를 이용하여 컬럼의 변경값을 지정할 수 있습니다. 단 이 때 사용되는 하위질의는 반드시 하나의 행에서 값을 반환하는 단일 행 하위질의이어야만 합니다.

[실습] 다음의 요구사항 대로 HR.COPY_EMP 테이블의 행(들)의 데이터를 수정하고, 수정된 데이터를 확인하시오.

- HR.EMPLOYEES 테이블의 EMPLOYEE_ID가 201 인 행의 JOB_ID와 동일한 EJOBID를 가지는 HR.COPY_EMP 테이블의 행(들)을 찾아서, EDEPTID 컬럼의 값을, HR.EMPLOYEES 테이블의 EMPLOYEE_ID가 100 인 행의 DEPARTMENT_ID 컬럼의 값으로 수정하시오.

```
SQL> UPDATE hr.copy_emp
      SET edeptid=(SELECT department_id FROM hr.employees WHERE employee_id=100)
      WHERE ejobid=(SELECT job_id FROM hr.employees WHERE employee_id=201);
```

1개 행 이(가) 업데이트되었습니다.

```
SQL> COMMIT ;
```

커밋되었습니다.

```
SQL> SELECT * FROM hr.copy_emp
      WHERE ejobid=(SELECT job_id FROM hr.employees WHERE employee_id=201);
```

EID	ENAME	ESAL	EHIREDATE	EJOBID	EDEPTID
201	Hartstein	13000	04/02/17	MK_MAN	90

☞ DML문에 하위질의를 이용하여 다른 테이블의 데이터로, 테이블의 데이터를 수정할 수 있습니다.

[실습] HR.DEPARTMENTS 테이블의 DEPARTMENT_NAME에 'Administration'이 포함된 행(들)의 DEPARTMENT_ID와 동일한 EDEPTID 값을 가지는 HR.COPY_EMP 테이블의 행(들)을 삭제하시오.

```
SQL> DELETE FROM hr.copy_emp
  WHERE edeptid = (SELECT department_id
                    FROM hr.departments
                   WHERE department_name LIKE '%Administration%');
```

1개 행 이(가) 삭제되었습니다. —위의 문장에서 하위질의의 WHERE절에 사용된 LIKE 연산자와 하위질의 앞에 있는 --등호(=)연산자는 조합이 좋지 못합니다.

```
SQL> COMMIT;
```

커밋되었습니다.

9-8. MERGE 문

- MERGE 문을 사용하여, 대상테이블(데이터의 수정이 수행되는 테이블)에 대하여, INSERT, UPDATE 및 DELETE 작업을 한 번에 수행할 수 있습니다. 단, DELETE 작업은 UPDATE와 같이 수행되어야 합니다.
- MERGE 문을 이용하여, 데이터를 입력, 수정, 삭제하려면, 대상테이블에 대한 INSERT, UPDATE, DELETE 객체 권한을 적절히 가져야 하며, 소스 테이블에 대하여 SELECT 객체 권한을 가져야 합니다.
- 동일한 MERGE 문에서 대상 테이블의 행은 한 번만 UPDATE될 수 있습니다.

■ 기본문법

MERGE INTO 스키마이름.대상테이블이름 테이블별칭	--MERGE가 수행되는 대상테이블
USING (테이블이름 또는 뷰-이름 또는 하위질의) 별칭	--입력 및 수정 시에 필요한 데이터의 소스
ON (조인 조건)	--소스와 대상 테이블의 행들을 선별한 조건으로
WHEN MATCHED THEN	--소스테이블의 행 중 ON 조건을 만족할 때
UPDATE SET	--대상테이블의 행을 UPDATE
대상컬럼1 = 소스컬럼표현식,	--형식: 대상테이블의 컬럼 = 소스의 컬럼
대상컬럼2 = 소스컬럼표현식	
...	
DELETE WHERE (소스에 대한 조건)	--대상테이블을 DELETE
WHEN NOT MATCHED THEN	-- 소스테이블의 행 중, ON 조건을 만족하지 않을 때
INSERT (대상테이블의_컬럼리스트)	-- 대상테이블에 행이 입력됩니다.
VALUES (소스의_컬럼리스트);	

- ON 절의 조건은, 소스테이블 대 대상테이블의 행이, [일 대 다 또는 일 대 일] 관계가 되는 조건이어야 합니다.
ON 절에 기술된 검색의 기준이 되는 컬럼은 UPDATE SET 부분에 명시하면 안됩니다.

- 목적테이블과 대상테이블이 꼭 같을 필요는 없습니다. 대상 및 소스테이블에 대하여 테이블 별칭을 지정해야 합니다.
- 대상테이블에 삽입 및 갱신되는 컬럼들은 소스테이블의 해당 컬럼과 동일한 데이터유형이어야 합니다.
특히 ON 절에 기술되는 컬럼들은 반드시 동일한 데이터유형 이어야 합니다.

[실습] 다음의 실습을 통해, MERGE 문의 기능 및 사용방법을 확인합니다.

1> 다음의 CREATE TABLE문을 실행하여, 대상테이블로 사용될 HR.EMP_MERGE_TEST 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.EMP_MERGE_TEST
  AS SELECT *
    FROM HR.EMPLOYEES
   WHERE SALARY < 10000;
```

Table HR.EMP_MERGE_TEST(가) 생성되었습니다.

2> 생성된 HR.EMP_MERGE_TEST 테이블의 데이터를 확인합니다.

```
SQL> SELECT COUNT(*) TOTAL_ROWS
  FROM HR.EMP_MERGE_TEST;

TOTAL_ROWS
-----
88      --테이블의 모든 행의 개수는 88 개.

SQL> SELECT 'NOT NULL' COMM_PCT, count(*) "ROWS"
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NOT NULL
UNION ALL
SELECT 'NULL', count(*)
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NULL;

COMM_PCT      ROWS
----- -----
NOT NULL      24      --COMMISSION_PCT 컬럼이 NULL이 아닌 행수는 24개
NULL          64      --COMMISSION_PCT 컬럼이 NULL인 행수는 64개
```

3-1> 다음의 MERGE문을 실행하여 HR.EMP_MERGE_TEST 테이블의 데이터를 수정합니다.

```
SQL> MERGE INTO hr.emp_merge_test T
  USING (SELECT * FROM hr.employees) S --소스데이터로 하위질의가 사용됨.
  ON (T.employee_id = S.employee_id) --소스와 대상테이블의 행을 찾을 조인조건.
  WHEN MATCHED THEN
    UPDATE SET
      T.first_name = UPPER(S.first_name), --UPDATE문의 SET절에는 컬럼=값 형식이므로
      T.last_name = UPPER(S.last_name), -- = 연산자의 왼편에 대상테이블의 컬럼을 명시함.
      T.email = LOWER(S.email), -- = 연산자의 오른편에 소스테이블의 컬럼을 명시함.
      T.phone_number = S.phone_number,
      T.hire_date = S.hire_date, --UPDATE가 정상적으로 수행되려면,
      T.job_id = S.job_id, --ON절을 만족하는 [대상테이블의 행수]와 "소스의 행수]의
      T.salary = S.salary*2, --관계가 [일:일 또는 다:일]관계가 되어야 함.
      T.commission_pct = S.commission_pct,
      T.manager_id = S.manager_id,
      T.department_id = S.department_id
  DELETE WHERE (S.commission_pct IS NOT NULL); --DELETE WHERE 절에는 소스에 대한 조건을 기술함.
```

88개 행 이(가) 병합되었습니다. --88개 행에 대한 UPDATE가 수행됨.

- ☞ 오라클 10g 버전부터, WHEN MATCHED THEN 블록이나 WHEN NOT MATCHED THEN 블록 중 하나만 선택적으로 사용할 수 있습니다.
- ☞ WHEN MATCHED THEN 블록만 명시하여, ON절을 만족시키는 행을 소스테이블에서 찾은 경우, 해당되는 대상테이블의 행의 값을 소스의 값으로 수정하고, ON절을 만족하는 소스테이블의 행이 DELETE WHERE절의 조건을 만족하는 경우, 소스의 행에 해당하는 대상테이블의 행을 삭제합니다.
- ☞ ON절에 명시된 컬럼은 UPDATE SET 절에 명시하면 안되며, DELETE WHERE 절에 명시하는 조건은, 소스테이블의 컬럼에 대한 조건을 명시해야 합니다.
- ☞ WHEN MATCHED THEN 블록을 기술할 때, UPDATE SET 절은 반드시 기술해야 하며, DELETE WHERE 절은 선택적입니다. 즉, DELETE WHERE 절은 생략할 수 있습니다.
- ☞ 대상테이블과 소스테이블에 대하여 테이블 별칭을 지정해야 편리합니다.
- ☞ 값으로 사용되는 소스의 컬럼에 대하여 연산처리 및 함수 처리가 수행될 수 있습니다.

3-2> 작업 후, HR.EMP_MERGE_TEST 테이블의 데이터를 확인합니다.

```
SQL> SELECT COUNT(*) TOTAL_ROWS FROM HR.EMP_MERGE_TEST;

TOTAL_ROWS
-----
64      --테이블의 행의 개수는 64 개. MERGE 작업전의 행수는 88개, 따라서, 24개 행이 삭제됨.

SQL> SELECT 'NOT NULL' AS COMM_PCT, count(*) AS "ROWS"
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NOT NULL
UNION ALL
SELECT 'NULL', count(*)
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NULL;

COMM_PCT      ROWS
----- -----
NOT NULL          0      --COMMISSION_PCT 컬럼이 NULL이 아닌 행수는 0개, 처음 24 개 행이 삭제됨.
NULL              64      --COMMISSION_PCT 컬럼이 NULL인 행수는 64개
```

☞ 수정 및 삭제 작업만 수행된 것이 확인됩니다.

3-3> 다음 실습을 위하여 MERGE문에 의한 데이터 변경의 작업의 트랜잭션을 롤백 합니다.

```
SQL> ROLLBACK;
```

롤백 완료.

4-1> 다음의 MERGE문을 실행하여 HR.EMP_MERGE_TEST 테이블의 데이터를 수정합니다.

```
SQL> MERGE INTO hr.emp_merge_test T
  USING (SELECT * FROM hr.employees ) S
  ON (T.employee_id = S.employee_id)
WHEN NOT MATCHED THEN
  INSERT
    (T.employee_id,T.last_name,T.email,T.hire_date,T.job_id,T.salary,T.commission_pct)
  VALUES
    (S.employee_id,S.last_name,S.email,S.hire_date,S.job_id,S.salary,S.commission_pct);
```

19개 행 이(가) 병합되었습니다.

☞ WHEN NOT MATCHED THEN 블록만 명시하여, ON절을 만족시키지 못하는 소스테이블의 행이 대상테이블에 입력됩니다.

☞ INSERT 키워드 다음에 컬럼이름을 명시하여, 명시된 컬럼에만 데이터가 입력되도록 할 수 있습니다.

4-2> 작업 후, HR.EMP_MERGE_TEST 테이블의 데이터를 확인합니다.

```
SQL> SELECT COUNT(*) TOTAL_ROWS FROM HR.EMP_MERGE_TEST;
```

```
TOTAL_ROWS
```

```
-----  
107    --테이블의 행의 개수는 107 개. MERGE 작업전의 행수는 88개, 따라서, 19개 행이 입력됨.
```

4-3> 다음 실습을 위하여 MERGE문에 의한 데이터 변경의 작업의 트랜잭션을 룰백 합니다.

```
SQL> ROLLBACK;
```

```
루백 완료.
```

5-1> 다음의 MERGE문을 실행하여 HR.EMP_MERGE_TEST 테이블의 데이터를 수정합니다.

```
SQL> MERGE INTO hr.emp_merge_test T
      USING (SELECT * FROM hr.employees) S
      ON (T.employee_id = S.employee_id)
      WHEN MATCHED THEN
        UPDATE SET
          T.first_name = UPPER(S.first_name),
          T.last_name = UPPER(S.last_name),
          T.email = LOWER(S.email),
          T.phone_number = S.phone_number,
          T.hire_date = S.hire_date,
          T.job_id = S.job_id,
          T.salary = S.salary*2,
          T.commission_pct = S.commission_pct,
          T.manager_id = S.manager_id,
          T.department_id = S.department_id
        DELETE WHERE (S.commission_pct IS NOT NULL)
      WHEN NOT MATCHED THEN
        INSERT VALUES (
          S.employee_id, S.first_name, S.last_name, S.email, S.phone_number, S.hire_date,
          S.job_id, S.salary, S.commission_pct, S.manager_id, S.department_id );
```

107개 행 이(가) 병합되었습니다.

5-2> 작업 후, HR.EMP_MERGE_TEST 테이블의 데이터를 확인합니다.

```
SQL> SELECT COUNT(*) TOTAL_ROWS FROM HR.EMP_MERGE_TEST;

TOTAL_ROWS
-----
83

SQL> SELECT 'NOT NULL' AS COMM_PCT, count(*) AS "ROWS"
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NOT NULL
UNION ALL
SELECT 'NULL', count(*)
  FROM HR.EMP_MERGE_TEST WHERE commission_pct IS NULL;

COMM_PCT      ROWS
----- -----
NOT NULL          11    -- COMMISSION_PCT 컬럼이 NULL이 아닌 행수는 11개
NULL              72    -- COMMISSION_PCT 컬럼이 NULL인 행수는 72개
```

☞ 삭제 및 입력이 수행되어 행의 수가 변경된 것이 확인됩니다.

6> MERGE문에 의한 데이터 변경의 작업의 트랜잭션을 커밋 합니다.

SQL> COMMIT;	MERGE MariaDB (Primary Key Unique Key)가 UPDATE 가 INSERT . column_name1 PK UK 커밋 완료.
	INSERT INTO table_name1 (column_name1, column_name2, column_name3, ...) SELECT column_name1, column_name2, column_name3, ... FROM table_name2 ON DUPLICATE KEY UPDATE column_name2 = , column_name3 = ;

9-9. 다중 테이블 데이터 삽입 개요.

- 하나의 데이터베이스로부터 데이터를 추출하여, 다른 데이터베이스로 가져오는 전체 과정을, 일반적으로 ETL이라고 하며, 이는 추출(Extraction), 변형(Transformation) 및 로드>Loading)를 의미합니다. 이 중 변형은, SQL 작업을 통해 입력하려는 값에 함수처리나, 연산처리를 수행하는 것을 의미합니다.
- 이러한 ETL 작업 시에, 오라클의 다중 테이블 INSERT문을 이용하여, 하위질의의 실행 결과로 반환된 행에서 얻어진 계산된 행들을, 여러 테이블에 한 번에 입력할 수 있습니다. 이 방법을 사용하면, 각 테이블 별로 동일한 데이터 소스를 여러 번에 걸쳐서 데이터 삽입을 수행하지 않아도 됩니다.
- 다중 테이블 INSERT문은 SQL 데이터 변형을 구현하는 기술 중 하나입니다.
- 동일한 데이터 소스를 한 번만 읽어서 여러 테이블에 삽입하기 때문에 작업에 소요되는 시간도 절약될 수 있습니다.

- 다음의 4가지 다중 테이블 데이터 삽입 방법이 가능합니다.

다중 테이블 데이터 삽입 방법	특징
무조건 INSERT ALL (UNCONDITIONAL INSERT ALL)	하위절의에 의해 반환된 행을 문장에 명시된 모든 테이블에 삽입합니다.
조건부 INSERT ALL (CONDITIONAL INSERT ALL)	하위절의에 의해 반환된 행에 대하여 특정 조건(WHEN절)을 검사한 후, 조건을 만족하는 모든 테이블에 행이 삽입됩니다.
조건부 INSERT FIRST (CONDITIONAL INSERT FIRST)	하위절의에 의해 반환된 행에 대하여 특정 조건(WHEN절)을 검사한 후, 조건을 만족하는 첫 번째 테이블에만 행이 삽입됩니다.
피벗팅 INSERT (PIVOT INSERT)	하위절의에 의해 반환된 행을 동일한 테이블에 반복적으로 삽입합니다. 문법적으로는 무조건 INSERT ALL 방법과 동일합니다.

- 다중 테이블 INSERT문을 사용할 때 다음의 제한 사항을 주의합니다.

- 뷰 또는 구체화된 뷰가 아닌, 테이블에 대해서만 다중 테이블 INSERT문을 수행할 수 있습니다.
- 원격 테이블에서는 다중 테이블 INSERT를 수행할 수 없습니다.
- 다중 테이블 INSERT를 수행할 때 테이블 컬렉션 표현식을 지정할 수 없습니다.
- 다중 테이블 INSERT문을 이용하여 총 999개의 컬럼에만 값을 입력할 수 있습니다.

9-9-1. 다중 테이블 데이터 삽입 실습

[실습] 다음의 실습을 통해 다중 테이블 INSERT문의 기능 및 사용방법을 확인합니다.

- 1> 다음의 CREATE TABLE문을 실행하여 실습을 위해 필요한 3개의 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.SAL_HISTORY
      (EMPID NUMBER(6), HIREDATE DATE, SAL NUMBER);
```

Table HR.SAL_HISTORY이(가) 생성되었습니다.

```
SQL> CREATE TABLE HR.MGR_HISTORY
      (EMPID NUMBER(6), MGR NUMBER(6), SAL NUMBER);
```

Table HR.MGR_HISTORY이(가) 생성되었습니다.

```
SQL> CREATE TABLE HR.ELSE_TABLE
      (EMPID NUMBER(6), HIREDATE DATE, SAL NUMBER(10), MGR NUMBER(6));
```

Table HR.ELSE_TABLE이(가) 생성되었습니다.

2> HR.EMPLOYEES 테이블에 다음의 한 행을 입력합니다.

```
SQL> INSERT INTO hr.employees VALUES
  (800, 'SH', 'SHIN', 'email@1234', NULL, sysdate, 'AC_MGR', 30000, NULL, 205, 110);
```

1 행 이(가) 삽입되었습니다. -- 입력한 행은 SALARY 컬럼의 값이 10000 보다 크고,
-- MANAGER_ID 컬럼의 값이 200 보다 큰 조건을 모두 만족합니다.

```
SQL> COMMIT ;
```

커밋 완료.

3> 다중 테이블 INSERT문으로 입력할 HR.EMPLOYEES 테이블의 소스데이터를 확인합니다.

```
SQL> SELECT employee_id, salary, manager_id FROM hr.employees WHERE employee_id > 200;
```

EMPLOYEE_ID	SALARY	MANAGER_ID
201	13000	100
202	6000	201
203	6500	101
204	10000	101
205	12008	101
206	8300	205
800	30000	205

7개의 행이 선택됨

4> HR.EMPLOYEES 테이블에서, 사원ID(EMPLOYEE_ID) 컬럼의 값이 200 보다 큰 직원에 대하여, 사원ID(EMPLOYEE_ID컬럼),
급여(SALARY컬럼), 관리자ID(MANAGER_ID컬럼), 입사일(HIRE_DATE컬럼)로 구성된 행들을, HR.SAL_HISTORY 테이블 및
HR.MGR_HISTORY 테이블에 모두 삽입하시오

```
SQL> INSERT ALL    -- 무조건 INSERT ALL을 사용합니다.
      INTO hr.sal_history      -- 대상테이블이름만 명시하면, 모든 컬럼에 값을 입력함.
      VALUES (employee_id, hire_date, EMPSAL) -- 하위질의의 컬럼이름 또는 컬럼별칭을 명시.
      INTO hr.mgr_history (EMPID, MGR, SAL)
      VALUES (employee_id, manager_id, EMPSAL*0.1)
      SELECT employee_id, hire_date, salary AS "EMPSAL", manager_id
      FROM hr.employees
      WHERE employee_id >200;          --하위질의에서 컬럼에 대한 컬럼별칭이 선언된 경우,
                                         --VALUES절에 반드시 컬럼별칭을 사용해야만 합니다.
```

14개 행 이(가) 삽입되었습니다.

- 기본문법: 무조건 INSERT ALL 문(피벗팅 INSERT문도 문법은 동일합니다)

```
INSERT ALL
```

```
INTO 대상테이블이름1 VALUES (컬럼1, 컬럼2,...) --VALUES 절에 명시하는 컬럼은, 하위질의에 표현된  
INTO 대상테이블이름2 VALUES (컬럼1, 컬럼2,...) --컬럼이름, 컬럼표현식 또는 컬럼별칭이 될 수 있습니다.  
...  
INTO 대상테이블이름n VALUES (컬럼1, 컬럼2,...)
```

하위질의:

- 맨 끝에 기술된 하위질의의 실행 결과로 반환된 행으로부터 계산되어 추출된 행들이, INTO절들에 기술된 대상테이블에 입력되며, 입력하기 원하는 테이블의 개수만큼, **INTO 대상테이블이름 VALUES** 절을 반복해서 기술합니다.
- INTO절에서, 대상테이블의 이름 뒤에, 괄호로 감싸서, 콤마로 구분된 컬럼이름들을 명시할 수 있습니다.
- **VALUES**절에, 입력할 값에 해당하는 하위질의의 컬럼이름을 기술합니다. 만약, INTO절에 대상테이블의 이름만 명시되면, 대상테이블의 모든 컬럼에 값이 입력될 수 있도록 하위질의의 컬럼이름을 기술합니다. INTO절에 컬럼이름도 명시되면, 명시된 컬럼에 해당하는 하위질의의 컬럼이름을 기술합니다. 단, 하위질의에 컬럼별칭이 명시되면, VALUES 절에도 컬럼별칭을 명시해야 합니다.

5> HR.SAL_HISTORY 테이블 및 HR.MGR_HISTORY 테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.sal_history; --하위질의로 선택된 행들이 모두 테이블에 입력되었습니다.
```

EMPID	HIREDATE	SAL
201	04/02/17	13000
202	05/08/17	6000
203	02/06/07	6500
204	02/06/07	10000
205	02/06/07	12008
206	02/06/07	8300
800	15/12/29	30000

7개의 행이 선택됨

```
SQL> SELECT * FROM hr.mgr_history; --하위질의로 선택된 행들이 모두 테이블에 입력되었습니다.
```

EMPID	MGR	SAL
201	100	1300
202	201	600
203	101	650
204	101	1000
205	101	1200.8
206	205	830
800	205	3000

7개의 행이 선택됨

6> 다음 실습을 위해서, 다중테이블 INSERT 작업의 트랜잭션을 롤백 합니다.

```
SQL> ROLLBACK ;
```

롤백이 완료되었습니다.

7> HR.EMPLOYEES 테이블에서, 사원ID(EMPLOYEE_ID) 컬럼의 값이 200 보다 큰 직원의, 사원ID(EMPLOYEE_ID컬럼), 100원 인상된 급여(SALARY컬럼), 관리자ID(MANAGER_ID컬럼), 입사일(HIRE_DATE컬럼)로 구성된 행들을 찾아서, 급여가 10100 보다 큰 조건을 만족하는 행들은 HR.SAL_HISTORY 테이블에 입력되고, 관리자ID가 200 보다 큰 조건을 만족하는 행들은 HR.MGR_HISTORY 테이블에 입력되도록 하시오. 두 조건을 모두 만족하는 행들은, 2개의 테이블에 모두 입력되어야 합니다. 그리고, 두 조건을 모두 만족하지 못하는 행들은 HR.ELSE_TABLE에 입력되도록 하시오.

```
SQL> INSERT ALL      -- 조건부 INSERT ALL 문을 사용합니다.
```

```
WHEN EMPSAL > 10100      -- WHEN 절에는 하위질의의 행을 선택할 조건을 작성합니다.
THEN INTO hr.sal_history
VALUES (employee_id, hire_date, EMPSAL)

WHEN manager_id > 200
THEN INTO hr.mgr_history (EMPID, MGR, SAL)
VALUES (employee_id, manager_id, EMPSAL)

ELSE INTO hr.else_table (EMPID, HIREDATE, SAL, MGR)
VALUES (employee_id, hire_date, EMPSAL, manager_id)

SELECT employee_id, hire_date, salary+100 AS "EMPSAL", manager_id
FROM hr.employees          -- 하위질의에서 컬럼에 대한 컬럼별칭이 선언된 경우,
WHERE employee_id > 200;    -- VALUES절에 반드시 컬럼별칭을 사용해야만 하며,
                           -- WHEN절의 조건에도 컬럼별칭이 선언된 컬럼이 사용되면
                           -- 컬럼별칭으로 조건을 기술해야 합니다.
```

8개 행 이(가) 삽입되었습니다.

■ 기본문법: 조건부 INSERT ALL 문

```
INSERT ALL      -- 하위질의에 의해 반환된 행에 대하여 조건을 만족하는 모든 테이블에 행이 삽입됩니다.
WHEN 조건1 THEN INTO 대상테이블이름1 VALUES (컬럼1, 컬럼2,...)
WHEN 조건2 THEN INTO 대상테이블이름2 VALUES (컬럼1, 컬럼2,...)
...
WHEN 조건n THEN INTO 대상테이블이름n VALUES (컬럼1, 컬럼2,...)
ELSE INTO 대상테이블이름 VALUES (컬럼1, 컬럼2,...)
하위질의;
```

- 각각의 INTO 키워드 앞에 WHEN 조건 THEN 을 각각 기술합니다.

- 조건부 INSERT ALL 문을 사용하면, 하위질의의 실행 결과로 반환된 행들이 WHEN절에 기술된 조건을 만족하는 모든 대상테이블에 입력되며, 각 WHEN 절에 대하여 동일한 하위질의의 결과가 계속 검사됩니다.

- WHEN절에 명시하는 조건은, 하위질의의 SELECT 절에 명시된 컬럼에 대하여 작성되어야 합니다.
- INTO절에서, 행이 입력되는 대상테이블의 이름 뒤에, 괄호로 감싸서, 테이블의 하나 이상의 컬럼이름(콤마로 구분)을 명시할 수 있습니다.
- VALUES절에, 입력할 값에 해당하는 하위질의의 컬럼이름을 기술합니다. 만약, INTO절에 대상테이블의 이름만 명시되면, 대상테이블의 모든 컬럼의 수만큼 하위질의의 컬럼이름을 기술합니다. INTO절에 대상테이블의 이름과 컬럼이름을 명시했으면, 명시된 대상테이블의 컬럼의 수만큼 하위질의의 컬럼이름을 기술합니다.
- ELSE INTO 절은 생략될 수 있습니다.
- 하위질의로부터 추출된 행들 중, 각각의 WHEN절에 기술된 조건들을 모두 만족하지 않았기 때문에. 명시된 어떤 대상테이블들에도 입력되지 않은 행들은, ELSE INTO 절에 대상테이블을 명시했을 때만, 해당 테이블에 입력됩니다. ELSE INTO 절이 명시되지 않으면, 서버는 해당 행에 대해 아무 작업도 수행하지 않습니다.
- 하나의 조건부 다중 테이블 INSERT문은 최대 127개의 WHEN 절을 포함할 수 있습니다.

8> HR.SAL_HISTORY 테이블, HR.MGR_HISTORY 테이블 및 HR.ELSE_TABLE에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.sal_history;
```

EMPID	HIREDATE	SAL
201	04/02/17	13100
205	02/06/07	12108
800	16/01/04	30100

-- 첫 번째 WHEN절의 EMPSAL > 10100 조건을
-- 만족하는 행들이 입력되었습니다.

```
SQL> SELECT * FROM hr.mgr_history;
```

EMPID	MGR	SAL
202	201	6100
206	205	8400
800	205	30100

-- 첫 번째 WHEN절의 조건에서 사용된 동일한 하위질의의 반환 행들이
-- 두 번째 WHEN절의 MANAGER_ID > 200 조건에 대해서도,
-- 검사되어, 조건을 만족하는 행들이 입력되었습니다.

-- 하위질의로부터 선택된 행이, 조건을 만족하는 테이블 모두에
-- 입력되어 있습니다.

```
SQL> SELECT * FROM hr.else_table;
```

EMPID	HIREDATE	SAL	MGR
203	02/06/07	6600	101
204	02/06/07	10100	101

-- WHEN절의 어떤 조건도 만족하지 않아서, 어느 테이블에 입력되지
-- 못한 행들이, ELSE INTO 절에 명시된 테이블에 입력되었습니다.

9> 다음 실습을 위해서, 다중테이블 INSERT 작업의 트랜잭션을 롤백 합니다.

```
SQL> ROLLBACK ;
```

롤백이 완료되었습니다.

10> HR.EMPLOYEES 테이블에서, 사원ID(EMPLOYEE_ID) 컬럼의 값이 200 보다 큰 직원의, 사원ID(EMPLOYEE_ID컬럼), 급여(SALARY컬럼), 관리자ID(MANAGER_ID컬럼), 입사일(HIRE_DATE컬럼)로 구성된 행들을 찾아서, 급여가 10000 보다 큰 조건을 만족하는 행들은 HR.SAL_HISTORY 테이블에 입력되고, HR.SAL_HISTORY 테이블에 입력된 행들을 제외한, 나머지 행들 중, 관리자ID가 200 보다 큰 조건을 만족하는 행들이 HR.MGR_HISTORY 테이블에 입력되도록 하시오. 그리고, 두 조건을 모두 만족하지 못하는 행들은 아무 작업도 수행하지 마시오.

```
SQL> INSERT FIRST      -- 조건부 INSERT FIRST 문을 사용합니다.
```

```
WHEN salary > 10000      -- WHEN 절에는 하위질의의 행을 선택할 조건을 작성합니다.

THEN INTO hr.sal_history
VALUES (employee_id, hire_date, salary)

WHEN manager_id > 200

THEN INTO hr.mgr_history (EMPID, MGR, SAL)      --ELSE INTO절과 관련된 VALUES절이 생략되면
VALUES (employee_id, manager_id, salary)          --조건을 만족하지 않은 행들에 대해서는
SELECT employee_id, hire_date, salary, manager_id  --아무 작업도 수행되지 않습니다.

FROM hr.employees

WHERE employee_id > 200;
```

5개 행 이(가) 삽입되었습니다.

■ 기본문법: 조건부 INSERT FIRST 문

```
INSERT FIRST      -- 하위질의가 반환된 행에 대하여, 조건을 만족하는 첫 번째 테이블에만 해당 행이 삽입됩니다.

WHEN 조건1 THEN INTO 대상테이블이름1 VALUES (컬럼1, 컬럼2,...)

WHEN 조건2 THEN INTO 대상테이블이름2 VALUES (컬럼1, 컬럼2,...)

...

WHEN 조건n THEN INTO 대상테이블이름n VALUES (컬럼1, 컬럼2,...)

ELSE INTO 대상테이블이름 VALUES (컬럼1, 컬럼2,...)

하위질의 :
```

- 각각의 INTO 키워드 앞에 WHEN 조건 THEN 을 각각 기술합니다
- 조건부 INSERT FIRST 문을 사용하면, 하위질의로부터 반환된 행들이, WHEN절에 기술된 조건을 만족하는 첫 번째 대상테이블에 입력되면, 일단 입력된 행들은 하위질의 결과로부터 사라지고, 남은 행을 가지고, 다음 WHEN절의 조건을 검사해서 입력에 사용합니다. 즉, WHEN절을 만족하는 행이 테이블에 입력될 때마다 소스로부터 해당 행은 사라집니다.
- INTO절에서, 행이 입력되는 대상테이블의 이름 뒤에, 괄호로 감싸서, 테이블의 하나 이상의 컬럼이름(콤마로 구분)을 명시할 수 있습니다.

- VALUES절에, 입력할 값에 해당하는 하위질의의 컬럼이름을 기술합니다. 만약, INTO절에 대상테이블의 이름만 명시되면, 대상테이블의 모든 컬럼의 수만큼 하위질의의 컬럼이름을 기술합니다. INTO절에 대상테이블의 이름과 컬럼이름을 명시했으면, 명시된 대상테이블의 컬럼의 수만큼 하위질의의 컬럼이름을 기술합니다.
- ELSE INTO 절은 생략될 수 있습니다.
- 하위질의로부터 추출된 행들 중, 각각의 WHEN절에 기술된 조건들을 모두 만족하지 않았기 때문에. 명시된 어떤 대상테이블들에도 입력되지 않은 행들은, ELSE INTO 절에 대상테이블을 명시했을 때만, 해당 테이블에 입력됩니다. ELSE INTO 절이 명시되지 않으면, 서버는 해당 행에 대해 아무 작업도 수행하지 않습니다.
- WHEN절에 명시하는 조건은, 하위질의의 SELECT 절에 명시된 컬럼에 대하여 작성되어야 합니다.
- 하나의 조건부 다중 테이블 INSERT문은 최대 127개의 WHEN 절을 포함할 수 있습니다.

11> HR.SAL_HISTORY 테이블, HR.MGR_HISTORY 테이블 및 HR.ELSE_TABLE에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.sal_history;
```

EMPID	HIREDATE	SAL	
201	04/02/17	13000	-- 첫 번째 WHEN절의 SALARY > 10000 조건을
205	02/06/07	12008	-- 만족하는 행들이 입력되었습니다.
800	15/12/29	30000	-- 첫 번째 WHEN절의 조건을 만족해서 이미 입력된 행들을 제외하고,

-- 나머지 행들이 두 번째 WHEN절의 MANAGER_ID > 200 조건에 대하여
-- 검사됩니다.

```
SQL> SELECT * FROM hr.mgr_history;
```

EMPID	MGR	SAL	
202	201	6000	-- MANAGER_ID > 200 조건을 만족한 행들이 입력됩니다.
206	205	8300	

12> 입력작업에 대하여 커밋을 수행합니다.

```
SQL> COMMIT ;
```

커밋되었습니다.

9-9-2. 피벗-INSERT 실습

- 피벗은 비-관계형 데이터베이스 테이블의 각 레코드를 관계형 데이터베이스 테이블 환경의 다중 레코드로 변환하는 등의 변형 작업을 수행하는 것을 말합니다.

[실습] 다음의 실습을 통해, 피벗 INSERT문의 기능 및 사용방법을 확인합니다.

1> 다음의 CREATE TABLE문을 실행하여, 비-관계형 테이블로 사용되는 HR.SALES_SOURCE_DATA 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.SALES_SOURCE_DATA (
    EMPLOYEE_ID NUMBER(6),      --판매직원 사원ID
    WEEK_ID NUMBER(2),          --주ID(1년 중 몇 번째 주)
    SALES_MON NUMBER(6),        --월요일 판매금액
    SALES_TUE NUMBER(6),        --화요일 판매금액
    SALES_WED NUMBER(6),        --수요일 판매금액
    SALES_THU NUMBER(6),        --목요일 판매금액
    SALES_FRI NUMBER(7) );     --금요일 판매금액
```

Table HR.SALES_SOURCE_DATA이(가) 생성되었습니다.

2> HR.SALES_SOURCE_DATA 테이블에 다음의 행들을 입력합니다.

```
SQL> INSERT INTO HR.SALES_SOURCE_DATA VALUES (176, 6, 2000, 3000, 4000, 5000, 6000);
```

1 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO HR.SALES_SOURCE_DATA VALUES (177, 6, 5000, 4000, 3000, 2000, 1000);
```

1 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO HR.SALES_SOURCE_DATA VALUES (178, 6, 3000, 2000, 1000, 2000, 6000);
```

1 행 이(가) 삽입되었습니다.

```
SQL> COMMIT;
```

커밋되었습니다.

3> 비-관계형 테이블인 SALES_SOURCE_DATA 테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM HR.SALES_SOURCE_DATA;
```

EMPLOYEE_ID	WEEK_ID	SALES_MON	SALES_TUE	SALES_WED	SALES_THU	SALES_FRI
176	6	2000	3000	4000	5000	6000
177	6	5000	4000	3000	2000	1000
178	6	3000	2000	1000	2000	6000

☞ 위의 테이블을 NON-RELATIONAL TABLE 이라고 하는 이유?

5개의 SALES_MON, SALES_TUE, SALES_WED, SALES_THUR, SALES_FRI 컬럼들은, 요일만 다르고, 모두 판매금액을 나타내는 컬럼들입니다. 즉, 속성이 동일한 컬럼들입니다.

4> 다음의 CREATE TABLE문을 실행하여, 관계형 테이블로 사용되는 HR.SALES_INFO 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.SALES_INFO (
    EMPLOYEE_ID NUMBER(6),      --판매직원 사원ID
    WEEK_ID NUMBER(2),          --주ID(1년 중 몇 번째 주)
    SALES NUMBER(6) );         --판매금액
```

Table HR.SALES_INFO이(가) 생성되었습니다.

- 비-관계형 형식의 SALES_SOURCE_DATA 테이블에 저장된 판매 레코드 집합을, EMPLOYEE_ID, WEEK, SALES 컬럼으로 구성된, 보다 일반적인 관계형 형식의 SALES_INFO 테이블로 저장하려면, HR.SALES_SOURCE_DATA 테이블의 각 레코드를 SALES_INFO 테이블에 적합하도록 변형해야 합니다. 이러한 일련의 변환 및 입력 작업을 일반적으로 **피벗**이라고 합니다. 이러한 피벗 작업을 수행 시에, **피벗 INSERT문을 사용하지 않으면, 다음과 같이, 하위질의를 이용한 입력작업을 5번 수행해야 합니다.**

```
INSERT INTO hr.sales_info (employee_id, week_id, sales)
SELECT employee_id, week_id, sales_mon FROM HR.SALES_SOURCE_DATA ;

INSERT INTO hr.sales_info (employee_id, week_id, sales)
SELECT employee_id, week_id, sales_tue FROM HR.SALES_SOURCE_DATA ;

INSERT INTO hr.sales_info (employee_id, week_id, sales)
SELECT employee_id, week_id, sales_wed FROM HR.SALES_SOURCE_DATA ;

INSERT INTO hr.sales_info (employee_id, week_id, sales)
SELECT employee_id, week_id, sales_thu FROM HR.SALES_SOURCE_DATA ;

INSERT INTO hr.sales_info (employee_id, week_id, sales)
SELECT employee_id, week_id, sales_fri FROM HR.SALES_SOURCE_DATA ;
```

5> 피벗 INSERT문을 이용하여, 비-관계형 HR.SALES_SOURCE_DATA 테이블의 모든 행들을 관계형 HR.SALES_INFO 테이블로 입력하시오. 단, HR.SALES_SOURCE_DATA 테이블의 SALES_MON, SALES_TUE, SALES_WED, SALES_THUR, SALES_FRI 컬럼들의 값이 HR.SALES_INFO 테이블의 SALES 컬럼에 입력되도록 합니다.

```
SQL> INSERT ALL -- 무조건 INSERT ALL문을 사용합니다. 단, INTO절에 테이블이름이 모두 동일합니다.

  INTO hr.sales_info(employee_id, week_id, sales) VALUES(employee_id, week_id, sales_mon)
  INTO hr.sales_info(employee_id, week_id, sales) VALUES(employee_id, week_id, sales_tue)
  INTO hr.sales_info(employee_id, week_id, sales) VALUES(employee_id, week_id, sales_wed)
  INTO hr.sales_info(employee_id, week_id, sales) VALUES(employee_id, week_id, sales_thu)
  INTO hr.sales_info(employee_id, week_id, sales) VALUES(employee_id, week_id, sales_fri)

  SELECT employee_id, week_id, sales_mon, sales_tue, sales_wed, sales_thu, sales_fri
  FROM HR.SALES_SOURCE_DATA ;
```

15개 행이(가) 삽입되었습니다.

☞ 무조건 INSERT ALL 문에서, 각 INTO절에 기술된 대상테이블의 이름들이 같으면, 피벗 INSERT문이 됩니다.

7> HR.SALES_INFO 테이블에 입력된 행들을 확인 후, 입력작업의 트랜잭션을 커밋합니다.

```
SQL> SELECT * FROM hr.sales_info;
```

EMPLOYEE_ID	WEEK_ID	SALES
176	6	2000
177	6	5000
178	6	3000
176	6	3000
177	6	4000
178	6	2000
176	6	4000
177	6	3000
178	6	1000
176	6	5000
177	6	2000
178	6	2000
176	6	6000
177	6	1000
178	6	6000

15개의 행이 선택됨

```
SQL> COMMIT ;
```

커밋되었습니다.

9-10. 트랜잭션(TRANSACTION) 개요.

- 트랜잭션을 제어하는 Transaction Control Language(TCL) 문장인, COMMIT문과 ROLLBACK문에 대하여 학습합니다.

■ 트랜잭션 개념.

- "논리적으로 하나의 단위로 처리되어야 하는 하나 이상의 DML 문장(들)"을 묶어서 "하나의 트랜잭션" 이라고 합니다.
- 앞에서 설명한 은행의 "계좌-이체"를 다시 생각해 보세요.

[참고] 트랜잭션의 의미에 대하여 이해하여 봅시다.

은행 데이터베이스를 생각해 봅시다. 은행 고객이 예금 계좌에서 결제 계좌로 현금을 이체하는 경우, 트랜잭션은 예금 계좌 감소, 결제 계좌 증가, 로그-저널에 로그-기록이라는 세 가지 별도 작업으로 구성됩니다. 오라클-서버에서 세 개의 DML문을 수행할 때 계좌 잔액이 정확히 유지되어야 합니다. 특정 문제로 인해 트랜잭션을 구성하는 DML문들 중 하나가 실행되지 못하면 트랜잭션을 구성하는 다른 DML문들도 모두 롤백(또는 UNDO)되어야 합니다

■ 트랜잭션을 정상적으로 종료하기 위하여 사용되는 SQL문들.

COMMIT문	DML문장 완료 후, 변경 후의 데이터 상태를 유지하면서 트랜잭션을 정상적으로 종료합니다.
ROLLBACK문	DML문장 완료 후, 변경 전의 데이터 상태로 되돌려 놓고 트랜잭션을 정상적으로 종료합니다.

■ 트랜잭션이 종료되는 방법.

- 세션 사용자가 수행한 DML문(들)로 구성된 트랜잭션은, 세션의 사용자가 직접 COMMIT문 또는 ROLLBACK문을 실행하여 세션의 트랜잭션을 명시적으로 종료해야 합니다. DB가 .
- 다음과 같은 경우에 서버에 의하여 자동으로 커밋이 수행됩니다.
 - 세션 사용자가 하나의 DDL문 또는 하나의 DCL문을 실행시킨 경우, 오라클-서버가 자동으로 커밋을 수행합니다.
 - 세션 사용자가 DML문(들)을 수행한 후, COMMIT문 또는 ROLLBACK문을 명시적으로 실행하지 않은 채, 오라클-서버에의 접속을 정상적으로 종료한 경우, 해당 DML문(들)으로 구성된 트랜잭션은 자동으로 커밋됩니다.
- 다음과 같은 경우에 서버에 의하여 자동으로 롤백이 수행됩니다.
 - 세션 사용자가 DML문(들)을 수행한 후, COMMIT문 또는 ROLLBACK문을 명시적으로 실행하지 않은 채, 오라클-서버에의 접속이 비정상적으로 종료된 경우, 해당 DML문(들)으로 구성된 트랜잭션은 자동으로 롤백됩니다.

■ 오라클 데이터베이스 서버에서의 트랜잭션 특징.

- 오라클-서버에서는 트랜잭션을 시작하는 별도의 명령어가 없으며, 접속한 세션에서 DML문 하나가 처음 실행되면, 오라클-서버는 자동으로 해당 세션의 트랜잭션을 관리하기 시작합니다.
- 특정 세션에서 수행되는 하나의 트랜잭션은, 1 개 이상의 DML문(들) 또는 하나의 DDL 또는 하나의 DCL 문장으로 구성되며, SELECT문은 트랜잭션과 관계가 없습니다.

■ 오라클 데이터베이스 서버에서 진행 중인 트랜잭션(DML 작업이 완료되었고, 커밋/롤백이 되기 전의 상태) 특징.

- 필요한 경우, ROLLBACK문을 실행하여 이전의 데이터 상태를 복구할 수 있습니다.
- DML을 수행한 현재 세션의 유저는, SELECT문을 사용하여 DML문의 작업 결과를 확인할 수 있습니다.
- 현재 세션의 유저가 실행한 DML 문의 결과를 다른 세션의 유저는 볼 수 없습니다.
- 수정되는 행이 잠기므로(ROW-LOCK), 다른 유저는 수정 중인 행의 데이터를 변경할 수 없습니다.

■ COMMIT문 실행 완료 후의 데이터 상태

- 데이터에 대한 변경-후의 사항이 데이터베이스에 적용되며, 모든 유저가 변경-후의 결과를 확인할 수 있습니다.
- FLASHBACK-기술이 아닌 일반적인 SELECT문을 이용하여 변경-전의 데이터를 사용할 수 없습니다.
- DML문의 대상 행에 대한 잠금이 해제되고, 다른 유저가 해당 행에 대하여 새로운 변경 작업을 수행할 수 있습니다.
- 모든 저장점(Savepoint)가 지워집니다.

■ ROLLBACK문 실행 완료 후의 데이터 상태

- DML문에 의하여 수행된 데이터의 변경 사항이 실행 취소되고, 변경되기 전의 데이터 상태가 복원됩니다.
- 영향 받는 행의 잠금이 해제됩니다.

[실습] 다음의 실습을 통해 트랜잭션의 의미에 대하여 확인합니다.

1> 실습을 위하여 HR.EMPS8 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.emps8 (
      id VARCHAR2(6), salary NUMBER(8,2), name VARCHAR2(30));
table HR.EMPS801(가) 생성되었습니다.
```

2> 다음의 하위질의를 이용한 INSERT를 수행합니다. 세션에 대하여 트랜잭션이 시작됩니다.

```
SQL> INSERT INTO hr.emps8(id, salary, name)
      SELECT employee_id, salary, last_name
      FROM hr.employees
     WHERE department_id = 90 ;
3개 행 이(가) 삽입되었습니다.
```

[참고] 이 세션의 상태는 다음과 같습니다.

- DML 작업(하위질의를 이용한 INSERT 작업)은 완료되었습니다.
- COMMIT문 또는 ROLLBACK문을 실행시키지 않았으므로, 트랜잭션은 진행 중입니다.
- 현재까지 트랜잭션은 하나의 INSERT문으로 구성됩니다.

☞ 트랜잭션이 진행되는 동안 세션에 대하여 변경되는 행은 잠김 상태가 됩니다(ROW-LOCK). 이러한 LOCK은 DML이 끝날 때가 아니라 트랜잭션이 끝날 때 해제됩니다.

3> INSERT1 이름의 저장점(Savepoint)을 설정합니다.

```
SQL> SAVEPOINT insert1 ;  
SAVEPOINT insert1
```

4) 동일한 세션에서 하위질의를 이용한 INSERT를 한번 더 수행합니다.

```
SQL> INSERT INTO hr.emps8(id, salary, name)  
      SELECT employee_id, salary, last_name  
        FROM hr.employees  
       WHERE department_id = 60 ;  
5개 행 이(가) 삽입되었습니다.
```

[참고] 세션의 상태는 다음과 같습니다.

- DML 작업들(2 개의 하위질의를 이용한 INSERT 작업)은 완료되었습니다.
- COMMIT문 또는 ROLLBACK문을 실행시키지 않았으므로, 트랜잭션은 진행 중입니다.
- 이 트랜잭션은 2 개의 INSERT문으로 구성됩니다.

☞ 트랜잭션이 진행되는 동안 세션에 대하여 변경되는 행은 잠김 상태가 됩니다(ROW-LOCK). 이러한 LOCK은 DML이 끝날 때가 아니라 트랜잭션이 끝날 때 해제됩니다.

5> 단계3에서 생성한 저장점까지 트랜잭션의 일부 DML 작업을 룰백합니다.

```
SQL> ROLLBACK TO SAVEPOINT insert1 ;  
룰백이 완료되었습니다.
```

☞ 트랜잭션이 종료된 것은 아닙니다. 트랜잭션을 구성하는 DML 작업들 중, 저장점부터 마지막까지의 DML 작업만 룰백됩니다. 그리고, 저장점은 삭제됩니다.

☞ 위의 문장 대신 ROLLBACK 문을 실행하면, 모든 DML 작업들이 룰백되며, 트랜잭션이 종료되며, ROLLBACK문 실행 전에 생성한 모든 저장점들은 삭제됩니다. 일단 룰백된 트랜잭션의 작업을 다시 자동으로 생성시킬 수는 없습니다.

6> COMMIT을 실행하여 트랜잭션을 종료합니다..

```
SQL> COMMIT ;  
커밋되었습니다.
```

☞ 이 세션에서 처음 수행한 INSERT문의 작업만 변경 후의 상태로 유지되면서 트랜잭션이 종료됩니다. 또한, COMMIT문 실행 전에 생성한 모든 저장점들은 삭제됩니다. 일단 커밋된 트랜잭션의 작업을 다시 룰백할 수는 없습니다.

[참고] DML문에 의해 변경된 행에 대한 행-잠금(ROW-LOCK).

- 모든 RDBMS 제품에서, 하나의 세션에서 수행된 DML작업으로 수정 중인 행에 대하여 다른 세션에서의 수정작업을 방지하기 위하여, 락(LOCK)을 이용합니다. 오라클 데이터베이스에서는 수정되는 행에만 해당 세션에 대하여 LOCK을 설정합니다.
- 하나의 세션에서 수정되는 행에 설정된 락은 트랜잭션이 종료되었을 때, 해제됩니다.

[실습] 다음의 실습을 통해 LOCK의 기능과 LOCK을 해제하는 방법을 확인합니다.

1> 2개의 터미널이나 CMD창에서 HR 계정으로 접속합니다.

2> 다음의 번호 순서에 맞춰서 다음의 DML문들을 실행하며 실습을 수행합니다.

2-1> HR-접속 세션들에서 다음의 UPDATE-문을 각각 실행합니다.

HR-접속 세션1	HR-접속 세션2
1> department_id 가 270 인 행에 UPDATE 수행. SQL> <code>UPDATE hr.departments SET department_name= 'Oracle' WHERE department_id = 270 ;</code> 1 row updated	2> department_id 가 260 인 행에 UPDATE 수행. SQL> <code>UPDATE hr.departments SET department_name= 'DB2' WHERE department_id = 260 ;</code> 1 row updated.

☞ 각 세션에서 서로 다른 행에 UPDATE를 수행했으며, 각 세션에서 변경되는 행에 각각 LOCK이 걸려 있습니다.

☞ 두 세션 모두 UPDATE문 실행은 완료했지만, 트랜잭션을 종료시키지는 않았습니다.

2-2> HR-접속 세션2에서만 다음의 UPDATE-문을 실행합니다.

	HR-접속 세션2
	3> department_id 가 270 인 행에 UPDATE 수행. SQL> <code>UPDATE hr.departments SET department_name= 'MS-SQL' WHERE department_id = 270 ;</code> _← 커서가 깜박이는 상태(대기 상태)가 지속됩니다.

☞ 세션1에서 수행한 UPDATE문에 의하여 해당 행에 걸려 있는 LOCK이 트랜잭션을 끝내지 않아서 풀려 지지 않았기 때문에, 세션2에서는 UPDATE 작업에 의해 LOCK을 요청하고 획득될 때까지 UPDATE-문 처리가 대기하게 됩니다.

2-3> HR-접속 세션1에서만 다음의 COMMIT문을 실행합니다.

HR-접속 세션1	HR-접속 세션2
<p>4> COMMIT을 수행하여 TRANSACTION을 끝냅니다.</p> <p>SQL> COMMIT ;</p> <p>Commit complete.</p>	<p>5> 자동으로 "1 row updated" 메시지가 표시됩니다.</p> <p>6> 수행한 트랜잭션을 롤백합니다.</p> <p>SQL> ROLLBACK;</p> <p>Rollback complete.</p>

☞ 세션1에서 COMMIT을 수행하여 트랜잭션을 끝내면, 해당 행에 걸려 있던 LOCK이 해제되고, 이 행에 대하여 DML문을 시도한 세션2로 LOCK이 전달되어 대기 상태에 있던 세션2의 UPDATE문이 실행 완료됩니다.

☞ 이렇게 LOCK을 이용하여, 같은 행에 대하여 서로 다른 세션들이 동시에 DML이 수행되는 것을 방지할 수 있습니다.

2-4> 이 후 과정의 실습을 위하여 변경된 데이터를 다음을 수행하여 원래의 값으로 변경합니다.

```
SQL> UPDATE hr.departments SET department_name= 'Payroll' WHERE department_id = 270;
1 row updated.

SQL> COMMIT ;

Commit complete.
```

- DB SELECT	LOCK	.
- DB SELECT	LOCK	.

9-11 SELECT문에 [FOR UPDATE-절] 사용.

가

■ 오라클-서버에서는 일반적인 SELECT문에 의하여 액세스되는 행에는 LOCK이 걸리지 않습니다. 하지만 상황에 따라서는

SELECT문에 의하여 액세스되는 행에 LOCK을 걸어야 할 경우도 필요합니다.

■ SELECT문에 [FOR UPDATE-절]을 이용하면 SELECT문으로 액세스되는 행(들)에 LOCK을 걸 수 있습니다.

이 때, [SELECT...FOR UPDATE]문에 의해 식별된 모든 행(들)에는 배타적 행 레벨 잠금(exclusive row-level lock)이 문장을 실행시킨 세션에게 자동으로 획득되며, 해당 세션의 사용자가 COMMIT문 또는 ROLLBACK문을 수행하기 전까지 다른 세션의 사용자는 행(들)을 변경할 수 없습니다.

[실습] HR.EMPLOYEES 테이블로부터 90번 부서에 근무하는 직원들의 사번, 급여, 커미션비율 및 직책코드를, 사번을 기준으로 오름차순으로 정렬하여 표시하시오. 단, SELECT문으로 액세스되는 행에 락을 설정하시오.

```
SQL> SELECT employee_id, salary, commission_pct, job_id
  FROM hr.employees
 WHERE department_id=90
 FOR UPDATE
 ORDER BY employee_id ;
```

EMPLOYEE_ID	SALARY	COMMISSION_PCT	JOB_ID
100	24000		AD_PRES
101	17000		AD_VP
102	17000		AD_VP

☞ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼의 값이 90 인 행들이 잠기며, 이 세션에서 사용자가 COMMIT문 또는 ROLLBACK문을 실행하는 경우에만 위의 행들에 걸린 LOCK이 해제됩니다.

☞ 위의 행들(LOCK이 걸린 행들)에 대하여 다른 세션에서 [SELECT..FOR UPDATE]문 또는 [DML문]으로 잠그려고 하면, 데이터베이스는 해당 행의 LOCK이 해제될 때까지 기다린 다음, 다른 세션의 [SELECT..FOR UPDATE]문 또는 [DML문] 작업들을 수행하게 됩니다.

[실습] SELECT문에 사용된 [FOR UPDATE-절]의 기능을 이해하기 위한 실습: JOIN-SELECT문에 [FOR UPDATE-절] 사용1

```
SQL> SELECT e.employee_id, e.salary, e.commission_pct, d.department_name, d.department_id
  FROM hr.employees e INNER JOIN hr.departments d ON e.department_id=d.department_id
 WHERE job_id = 'FI_ACCOUNT' AND location_id = 1700
 FOR UPDATE
 ORDER BY 1 ;
```

EMPLOYEE_ID	SALARY	COMMISSION_PCT	DEPARTMENT_NAME	DEPARTMENT_ID
109	9000		Finance	100
110	8200		Finance	100
111	7700		Finance	100
112	7800		Finance	100
113	6900		Finance	100

☞ HR.EMPLOYEES 테이블 및 HR.DEPARTMENTS 테이블에서 문장이 액세스되는 행이 잠깁니다.

[실습] SELECT문에 사용된 [FOR UPDATE-절]의 기능을 이해하기 위한 실습: JOIN-SELECT문에 [FOR UPDATE-절] 사용2

```
SQL> SELECT e.employee_id, e.salary, e.commission_pct, d.department_name, d.department_id
   FROM employees e INNER JOIN departments d
  ON e.department_id=d.department_id
 WHERE job_id = 'FI_ACCOUNT'
   AND location_id = 1700
 FOR UPDATE OF e.salary
 ORDER BY 1 ;
```

EMPLOYEE_ID	SALARY	COMMISSION_PCT	DEPARTMENT_NAME	DEPARTMENT_ID
109	9000		Finance	100
110	8200		Finance	100
111	7700		Finance	100
112	7800		Finance	100
113	6900		Finance	100

☞ JOIN이 사용된 SELECT 문에 [FOR UPDATE OF 컬럼이름]절을 사용하면, 명시된 컬럼이 포함된 테이블의 행만 잠깁니다.

위의 예제에서는 HR.Employees 테이블의 행만 잠기고, HR.Departments 테이블의 행은 잠기지 않습니다.

[참고] FOR UPDATE 절에 [WAIT n, n은 정수] 또는 [NOWAIT] 옵션의 사용(11gNF).

한 세션에서 [SELECT ... FOR UPDATE]문으로 액세스하려는 테이블의 행들을, 다른 사용자의 세션에서 먼저 잠근(LOCK) 경우, [SELECT ... FOR UPDATE]문을 요청한 세션이 해당 테이블의 행을 사용할 수 있을 때까지, 즉 다른 사용자의 세션에서 COMMIT 또는 ROLLBACK 명령을 실행하여 LOCK을 해제해 줄 때까지 대기상태가 유지됩니다.
만약, [FOR UPDATE-절]에 [NOWAIT] 옵션 또는 [WAIT n, n은 정수] 옵션을 추가하면(아래의 예제 참조), 위와 동일한 상황에서 [SELECT ... FOR UPDATE]문을 요청한 세션이 대기하지 않거나, 또는 다른 세션의 LOCK이 해제될 때까지 n초만 대기상태를 유지하게 됩니다.

[실습] SELECT문에 사용된 [FOR UPDATE-절]의 기능을 이해하기 위한 실습: WAIT 옵션의 사용

```
SQL> SELECT employee_id, salary, commission_pct, job_id
   FROM employees
 WHERE job_id = 'FI_ACCOUNT'
 FOR UPDATE WAIT 5
 ORDER BY employee_id;
```

EMPLOYEE_ID	SALARY	COMMISSION_PCT	JOB_ID
109	9000		FI_ACCOUNT
110	8200		FI_ACCOUNT
111	7700		FI_ACCOUNT
112	7800		FI_ACCOUNT
113	6900		FI_ACCOUNT

☞ 앞의 [SELECT..FOR UPDATE]문으로 액세스되는 행(들)이, 다른 세션에 의한 LOCK이 걸린 상태로 사용 중일 경우,

위의 문장은 5 초간 대기 상태가 됩니다. 다른 세션의 LOCK이 5초 내에 해제될 경우, 문장이 정상적으로 처리되고,

5초가 지나도록 다른 세션의 LOCK이 해제되지 않으면, 해당 세션에서 아래의 오류가 발생되며, 문장이 중지됩니다.

명령의 1 행에서 시작하는 중 오류 발생 -

```
SELECT employee_id, salary, commission_pct, job_id  
FROM employees  
WHERE job_id = 'SA_REP'  
FOR UPDATE WAIT 5  
ORDER BY employee_id
```

오류 보고 -

SQL 오류: ORA-30006: 리소스 사용 중. WAIT 시간 초과로 획득이 만료됨
30006. 00000 - "resource busy; acquire with WAIT timeout expired"
*Cause: The requested resource is busy.
*Action: Retry the operation later.

[실습] SELECT문에 사용된 [FOR UPDATE-절]을 사용해서 액세스되는 행에 걸린 잠김(LOCK)은 다음처럼, COMMIT문 또는 ROLLBACK문을 실행하여 해제시킵니다.

```
SQL> COMMIT ;
```

커밋되었습니다.

10 테이블 생성(DATA DEFINITION LANGUAGE, DDL).

◆ 학습 목표.

- 테이블을 생성하는 방법을 학습합니다.
- 대표적인 데이터유형(DATATYPE)에 대하여 학습합니다.
- 테이블에 정의할 수 있는 제약조건에 대하여 학습합니다.

10-1. 테이블 및 제약조건 개요.

- 데이터베이스에서 테이블과 제약조건을 사용하는 목적은 다음과 같습니다.

객체종류	설명
테이블	데이터가 저장된 저장공간을 가진 객체이며, 사용자가 원하는 데이터를 액세스(SELECT, DML문장)하기 위한 데이터의 의미와 속성(테이블이름, 컬럼이름, 컬럼의 데이터유형 등)가 정의된 객체입니다.
제약조건	데이터베이스 테이블에 입력(또는 수정 및 삭제)되는 데이터가 지켜야 하는 규칙을 의미합니다.

- 데이터베이스에 접속한 계정이, 자신의 스키마에 테이블을 생성하기 위해서는 다음의 조건이 충족되어야 합니다.

- 접속한 데이터베이스 계정에게 CREATE TABLE 시스템 권한이 부여되어 있어야 합니다.
- 접속한 데이터베이스 계정에게 저장 공간에 대한 권한(TABLESPACE 상의 QUOTA)이 설정되어 있어야 합니다.

[참고] 스키마(SCHEMA)란?

- 소유권(OWNERSHIP)이 동일한 객체들의 그룹을 의미합니다. 예를 들어, HR 계정이 소유한 모든 객체들을 그룹화하여 HR 스키마라고 합니다.

```
DB
CREATE TABLE . . .
 1 ( . . . ),
 2
) TABLESPACE ;
```

10-2. 기본적인 테이블 생성 문법.

- 테이블을 생성하기 위한 최소한의 문장은 다음의 문법에 맞도록 작성되어야 합니다.

```
CREATE TABLE 스키마이름.테이블이름 ( --스키마이름을 생략하면, 접속계정의 소유로 테이블이 생성됩니다.
  컬럼이름1 데이터유형(최대길이),
  컬럼이름2 데이터유형, -- 일부 데이터유형은 데이터의 최대 길이를 지정하지 않기도 합니다.
  , ... );
```

☞ CREATE TABLE 문장으로 테이블 생성 시에 최소한 테이블이름, 컬럼이름, 컬럼의 데이터유형 및 허용된 데이터의 최대 길이는 반드시 명시해야 합니다. 선택적으로, 컬럼에 대하여 기본값을 설정할 수 있습니다.

☞ 테이블 생성 시에, 테이블이름 및 컬럼이름을 기술할 때 다음의 사항을 준수해야 합니다.

- 오라클에서는 테이블(또는 객체)이름 및 컬럼이름이 30 BYTES 길이를 넘을 수 없습니다.

- A-Z, a-z, 0-9, _, \$, # 문자만 포함될 수 있으며, 숫자로 시작하면 안됩니다.

- 테이블(또는 객체) 이름 및 컬럼이름에 오라클 데이터베이스 서버의 예약어는 사용될 수 없습니다.

- 테이블(또는 객체) 이름은 같은 사용자에 의해 소유된 다른 객체의 이름과 중복될 수 없습니다.

☞ 이미 다른 객체가 사용 중인 이름으로, 테이블을 생성하려고 시도하면 아래와 같은 오류가 발생됩니다.

```
SQL> CREATE TABLE hr.departments (
    department_id NUMBER(4),
    department_name VARCHAR2(30),
    manager_id NUMBER(6) );
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
CREATE TABLE hr.departments (
    department_id NUMBER(4),
    department_name VARCHAR2(30),
    manager_id NUMBER(6) )
```

오류 발생 명령행: 1 열: 17

오류 보고 -

SQL 오류: ORA-00955: 기존의 객체가 이름을 사용하고 있습니다.

00955. 00000 - "name is already used by an existing object"

*Cause:

*Action:

[참고] 다른 데이터베이스 접속 계정이 소유한 테이블 사용하기.

■ HR 계정이 소유한 EMPLOYEES 테이블을 DBA 계정인 SYS 계정이 사용하려고 할 때, SYS 계정은 EMPLOYEES 테이블이 자신의 스키마 객체가 아니므로, HR.EMPLOYEES (스키마_이름.테이블_이름)라고 SQL문에서 명시해야 합니다.

■ 다음은 테이블이름에 스키마이름을 명시할 때와 명시하지 않을 때의 의미 차이를 설명한 것입니다.

SELECT문	의미
SELECT * FROM EMPLOYEES ;	접속한 계정의 스키마에 있는 EMPLOYEES 테이블의 데이터를 조회합니다.
SELECT * FROM HR.EMPLOYEES ;	HR의 스키마에 있는 HR.EMPLOYEES 테이블의 데이터를 조회됩니다.

[참고] 테이블의 컬럼에 대하여 DEFAULT 옵션을 이용한 컬럼의 기본값 설정.

■ 테이블에 값을 입력할 때, DEFAULT 옵션으로 기본값이 설정된 컬럼에 입력되는 값이 없을 경우, DEFAULT 옵션으로 설정된 기본값이 자동으로 입력됩니다. 필요한 경우, 테이블의 컬럼 각각에 대하여 DEFAULT 옵션을 이용하여 기본값을 설정할 수 있습니다.

■ 컬럼의 데이터유형과 일치한 상수(Literal) 값, 표현식, SQL 함수들을 컬럼의 DEFAULT 값으로 설정할 수 있습니다.
단, 다른 컬럼의 이름이나 ROWNUM, NEXTVAL, CURRVAL 같은 PSEUDO-컬럼이 기본값으로 설정될 수 없습니다.

■ UPDATE문과 INSERT문에서 컬럼에 설정된 기본값을 DEFAULT 키워드로 호출하여 사용할 수 있습니다.

[실습] 다음의 실습으로 테이블을 생성하고, 테이블의 컬럼에 설정된 DEFAULT옵션의 기능을 확인합니다.

1> 다음의 문장을 실행하여 HR.CUST 테이블 생성합니다.

```
SQL> CREATE TABLE hr.cust (
    cid NUMBER(4),
    name VARCHAR2(25),
    city VARCHAR2(10) DEFAULT 'SEOUL',
    rdate DATE DEFAULT SYSDATE ) ;
```

table HR.CUST이(가) 생성되었습니다.

2> HR.CUST 테이블에 다음의 데이터를 입력합니다.

```
SQL> INSERT INTO hr.cust VALUES (10, 'SHSHIN', NULL, TO_DATE('20100321','YYYYMMDD')) ;
```

1개 행 이(가) 삽입되었습니다.

☞ 사용자가 직접 값을 지정하거나 NULL 키워드를 직접 지정하면, 컬럼의 기본값과 상관없이 행이 입력됩니다.

3> HR.CUST 테이블에 다음의 데이터를 입력합니다.

```
SQL> INSERT INTO hr.cust(cid) VALUES (20) ;
```

1개 행 이(가) 삽입되었습니다.

☞ 기본값이 설정된 컬럼에 대하여, 입력된 데이터도 없고, NULL로 지정하지 않은 경우, 기본값이 자동으로 입력됩니다.

기본값이 지정되지 않은 컬럼은 NULL 상태로 입력됩니다.

4> HR.CUST 테이블에 다음의 데이터를 입력합니다.

```
SQL> INSERT INTO hr.cust VALUES (30, DEFAULT, DEFAULT, DEFAULT) ;
```

1개 행 이(가) 삽입되었습니다.

☞ DEFAULT 키워드를 이용하여, 기본값을 직접 호출할 수 있습니다. 이 때, 기본값이 설정되지 않은 컬럼들은(예, name 컬럼) NULL 상태로 입력되고, 기본값이 설정된 컬럼은 기본값이 자동으로 입력됩니다.

4> HR.CUST 테이블에 입력된 데이터 조회

```
SQL> SELECT * FROM hr.cust ;
```

CID	NAME	CITY	RDATE
10	SHSHIN		10/03/21
20		SEOUL	15/01/04
30		SEOUL	15/01/04

10-3. 오라클 데이터베이스의 대표적인 데이터유형들.

10-3-1. VARCHAR2와 CHAR 데이터유형

CLOB (가 4GB 가)

- VARCHAR2 및 CHAR 데이터유형은 오라클 데이터베이스에서 문자 데이터를 처리하는 데이터유형입니다.
- 최대 길이를 명시할 때 [BYTE] 또는 [CHAR]을 옵션을 명시하여 처리 단위를 각각 바이트 또는 문자수로 지정할 수 있으며, [BYTE] 또는 [CHAR]을 명시하지 않으면, 기본적으로 [BYTE] 단위로 데이터의 최대 길이가 처리됩니다.

가 (, , , ,) VARCHAR2(size)	가 (, , , ,) CHAR(size)
<ul style="list-style-type: none"> 최대 4000 BYTE 길이까지만 처리 가능합니다. 반드시 최대 허용 길이를 명시해야 합니다. 입력되는 데이터의 실제 길이만큼 저장 공간을 사용하는 가변-길이 문자 데이터유형입니다. 만약 VARCHAR2(30)로 지정된 컬럼에 'ORACLE' 값을 입력하면, 6BYTE 공간을 사용하여 저장됩니다. <p style="text-align: center;">6 ORACLE</p>	<ul style="list-style-type: none"> 최대 2000 BYTE 길이까지만 처리 가능합니다. 최대 길이를 명시하지 않으면, 1 BYTE가 지정됩니다. 지정된 최대 길이를 무조건 저장 공간으로 사용하는 고정-길이 문자 데이터유형으로, 데이터가 차지하고 남은 공간은 빈칸으로 채워서 저장됩니다. 만약 CHAR(30)로 지정된 컬럼에 'ORACLE' 값을 입력하면, 무조건 30 BYTE 공간을 사용하며, 사용 안 된 빈 공간을 빈칸으로 채우면서 저장됩니다. <p style="text-align: center;">30 ORACLE</p> <ul style="list-style-type: none"> CHAR 데이터유형은 길이가 일정한 문자 데이터에만 사용하는 것을 권장합니다.

[참고] 테이블의 행이 블록에 저장될 때의 행-구조(ROW-STRUCTURE)

INSERT문으로 테이블에 한 행이 입력될 때, 오라클-서버에 의하여, 다음과 같은 행-구조(ROW-STRUCTURE)의 형식으로 테이블의 세그먼트를 구성하는 하나의 데이터-블록에 저장됩니다.

데이터유형 VARCHAR2(5)		데이터유형 VARCHAR2(30)		데이터유형 VARCHAR2(30)	
행 해더	저장 길이	값	저장 길이	값	저장 길이
		2	10	6	ORACLE

10-3-2. DATE 데이터유형.

Timestamp(0): ()

- 날짜 데이터를 처리하는 데이터유형으로, 사용 시에 길이를 명시하지 않습니다.
- 7 BYTE의 저장 공간을 사용하여 [세기/년도/월/일/시/분/초]를 내부 숫자 형식으로 저장합니다.
- 유효한 DATE 범위는 BC 4712년 01월 01일부터 AD 9999년 12월 31일까지입니다.
- DATE 데이터유형이 세션(CLIENT)에서 기본적으로 표시되는 형식은 사용자가 NLS_DATE_FORMAT 세션 설정에 의하여 명시적으로 정해 주거나 NLS_TERRITORY 세션 설정에 의하여 암시적으로 정해집니다.
- 위의 세션 설정을 명시적으로 선언해서 설정하지 않은 경우에는 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라, 자동으로 설정됩니다.

10-3-3. NUMBER (PRECISION, SCALE) 데이터유형.

- NUMBER 데이터유형은 0을 포함하여, 1.0×10^{-130} 부터 1.0×10^{126} (1.0×10^{126} 은 포함되지 않음)까지 절대값(ABSOLUTE VALUES)을 가지는 양(POSITIVE, 양수) 및 음(NEGATIVE, 음수)의 FIXED NUMBER를 저장합니다.
- NUMBER 데이터유형의 산술 표현식(Expression)의 절대값(ABSOLUTE VALUE)이 1.0×10^{126} 보다 크거나 같으면, 오류가 반환됩니다.
- 각 NUMBER 형식 값은 1부터 22 BYTES를 필요로 합니다.
- 저장되는 NUMBER는 2 자리당 1 BYTE 저장 공간을 사용합니다.
- NUMBER 데이터-형식 사용 예제 및 설명

number(5,2)
number(5) ==> number(5,0)

- NUMBER(5, 2) 데이터유형에 NUMBER 데이터 입력 시 정상 처리 유무.

100	입력됩니다.
1000	입력되지 못합니다(정수부 자리 수 제한을 넘음).
999.99457	999.99로 반올림되어 입력됩니다.
999.995	반올림하면 1000이 되므로 입력되지 못합니다.

- NUMBER(3)과 NUMBER(3, 0)은 동일하며, 전체 자리 수가 3자리이고 소수점 이하 자리 수가 0 자리인 실수입니다.

즉, 정수부가 3자리인 실수를 의미합니다. 즉, 3자리 정수가 아닙니다.

100.5 가 101
100.3 100

10-3-4. TIMESTAMP 계열 데이터유형.

Date

- 오라클 9i 버전부터 TIMESTAMP, [TIMESTAMP WITH TIME ZONE](#), 또는 [TIMESTAMP WITH LOCAL TIME ZONE](#) 데이터유형을 이용하여 DATETIME을 처리할 수 있습니다.
- [TIMESTAMP WITH TIME ZONE](#) 및 [TIMESTAMP WITH LOCAL TIME ZONE](#) 데이터유형을 사용하여, 시간대를 처리할 수 있습니다.
- 사용 시에 밀리초의 자리 수를 최소 0 자리부터 최대 9 자리까지 지정할 수 있습니다. 명시하지 않은 경우, 디폴트로 6 자리까지 밀리 초가 표시됩니다.

• **TIMESTAMP 계열 데이터유형들에 대한 기본적인 특징.**

TIMESTAMP (fractional_seconds)	<ul style="list-style-type: none"> ○ [세기/년/월/일/시/분/초/밀리초] 형식으로 구성되어 날짜시간 데이터를 처리합니다. 단, 시간대는 포함되지 않습니다. ○ 이 데이터유형이 세션에서 표시되는 형식은 사용자가 NLS_TIMESTAMP_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 자동으로 정해집니다. ○ 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다.
TIMESTAMP (fractional_seconds) WITH TIME ZONE	<ul style="list-style-type: none"> ○ TIMESTAMP 데이터유형의 값에 시간대가 추가된 데이터유형입니다. ○ 이 데이터유형이 세션에서 표시되는 형식은 사용자가 NLS_TIMESTAMP_TZ_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 자동으로 정해집니다. ○ 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다. ○ 이 데이터유형에서의 시간대는 접속한 세션의 시간대가 고려됩니다.
TIMESTAMP (fractional_seconds) WITH LOCAL TIME ZONE	<ul style="list-style-type: none"> ○ 데이터베이스 서버의 시간대를 기준으로 서버와 CLIENT 세션의 시간대와의 시차가 계산되어 반영된 TIMESTAMP 데이터유형입니다. ○ 이 데이터유형이 세션에서 기본적으로 표시되는 형식은 사용자가 NLS_TIMESTAMP_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 암시적으로 정해집니다. ○ 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다. ○ 이 데이터유형에서의 시간대는 접속한 세션의 시간대가 고려됩니다.

[참고] DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE 의 차이.

- [+9:00] 시간대의 오라클-서버에, [+7:00] 시간대의 사용자가 접속하여 '2010/06/18/ 20:00:00'의 클라이언트의 날짜-시간 데이터를 입력하는 경우, 테이블의 데이터유형에 따라 처리하는 방법이 아래처럼 차이가 납니다.

TIMESTAMP 또는 DATE 데이터유형의 컬럼	사용자가 입력한 값이 그대로 오라클-서버에 저장되며, 다른 시간대의 사용자가 접속하여 조회하면, 입력된 상태 그대로 표시됩니다('2010/06/18/ 20:00:00'). 즉, 서버와 세션 사이의 시간대에 대한 차이가 전혀 고려되지 않습니다.
TIMESTAMP WITH TIME ZONE 데이터유형의 컬럼	사용자가 입력한 값에 사용자의 시간대인 +7:00가 자동으로 추가되어 서버에 저장되며, 모든 시간대의 세션에서, 입력-세션의 시간대가 포함된 동일한 형태로 표시됩니다('2010/06/18/ 20:00:00 +7:00'). 따라서, 시간대 오류를 극복할 수 있습니다.
TIMESTAMP WITH LOCAL TIME ZONE 데이터유형의 컬럼	오라클-서버의 시간대 [+9:00]를 기준으로 사용자 세션의 시간대 [+7:00]와의 차이가 반영된 값('2010/06/18/ 22:00:00')이, 서버에 저장되고, 다른 시간대의 세션에 대해서, 서버의 시간대를 기준으로 처리된 날짜시간이 표시됩니다.

■ SYSTIMESTAMP 함수, LOCALTIMESTAMP 함수와 CURRENT_TIMESTAMP 함수.

SYSTIMESTAMP	데이터베이스 서버가 운영 중인 시스템의 날짜와 시간을 TIMESTAMP WITH TIME ZONE 데이터유형으로 반환합니다.
CURRENT_TIMESTAMP	세션(CLIENT)의 시간대에서 현재 날짜와 시간을 TIMESTAMP WITH TIME ZONE 데이터유형으로 반환합니다.
LOCALTIMESTAMP (밀리초_자리수)	세션(CLIENT)의 시간대에서 현재 날짜와 시간을 TIMESTAMP 데이터유형으로 반환합니다.

- ☞ TIMESTAMP 데이터유형은 시간대를 표시하지 못하므로, 시간대를 표시해야 하는 경우라면, TIMESTAMP 데이터유형은 올바른 선택이 아닙니다.
- ☞ TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE 데이터유형이 지정된 테이블의 컬럼에 시간대를 고려해야 하는 DATETIME 데이터를 입력할 때, SYSDATE 함수를 절대로 사용하지 마십시오.
- ☞ TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE 데이터유형이 지정된 테이블의 컬럼에 값을 입력할 때는 SYSTIMESTAMP 함수나 LOCALTIMESTAMP 함수, 또는 CURRENT_TIMESTAMP 함수를 사용하시기 바랍니다.

10-3-5. INTERVAL 계열 데이터유형.

- 9i 버전부터 기간(INTERVAL)을 처리하는 INTERVAL YEAR TO MONTH 및 INTERVAL DAY TO SECOND 데이터유형이 새로 추가되었습니다.

■ INTERVAL 계열 데이터유형에 대한 기본적인 특징

INTERVAL YEAR (연도 최대-자릿수) TO MONTH	<ul style="list-style-type: none"> ○ 기간 데이터를 [n 년]-[n 개월] 형식으로 처리합니다. ○ [연도 최대-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 처리되는 연도의 최대 자릿수를 지정합니다. ○ [연도 최대-자릿수]를 명시하지 않으면, 2로 지정됩니다.
---------------------------------------	--

INTERVAL DAY (일 최대-자릿수) TO SECOND(밀리초-자릿수)	<ul style="list-style-type: none"> ○ 기간 데이터를 [n일]-[n시간]-[n분]-[n초] 형식으로 처리합니다. ○ [일 최대-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 처리되는 일의 최대 자릿수를 지정합니다. ○ [일 최대-자릿수]를 명시하지 않으면, 2로 지정됩니다. ○ [밀리초-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 표시하고 싶은 밀리초의 자릿수를 지정합니다. ○ [밀리초-자릿수]를 명시하지 않으면, 6으로 지정됩니다.
---	---

[실습] 다음의 실습을 수행하여 INTERVAL 계열 데이터유형의 컬럼에 데이터를 입력하는 방법을 학습합니다.

1> 데이터베이스 서버에 접속한 SQL*Plus 세션의 TIMESTAMP 데이터유형의 표시 형식을 변경합니다.

```
SQL> ALTER SESSION SET NLS_TIMESTAMP_FORMAT = 'YYYY/MM/DD HH24:MI:SS' ;
```

session SET이(가) 변경되었습니다.

2> 실습용 hr.test_interval 테이블 생성

```
SQL> CREATE TABLE hr.test_interval (
    t_id NUMBER(3),
    iytm INTERVAL YEAR(3) TO MONTH,          -- YEAR(3): 최대 999년
    idts INTERVAL DAY(3) TO SECOND(2);      -- DAY(3): 최대 999일
                                                -- SECOND(2): 밀리초 2자리
```

table HR.TEST_INTERVAL이(가) 생성되었습니다.

☞ 위의 테이블 생성 문장에서 YEAR(3)과 DAY(3)에서 (3)을 명시하지 않고, 각각 YEAR, DAY만 적으면, 자동으로 YEAR(2), DAY(2)로 각각 설정됩니다.

3> INTERVAL 데이터유형 컬럼에 데이터 입력 실습: 데이터 입력 시 주의하시기 바랍니다.

```
SQL> INSERT INTO hr.test_interval VALUES (1, '999-11', '999 23:59:59.99');
```

1개 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO hr.test_interval VALUES (2, INTERVAL '10' YEAR, INTERVAL '24' DAY);
```

1개 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO hr.test_interval VALUES (3, INTERVAL '60' MONTH, INTERVAL '240' MINUTE);
```

1개 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO hr.test_interval VALUES (4, NULL, INTERVAL '240:30' HOUR TO MINUTE);
```

1개 행 이(가) 삽입되었습니다.

```
SQL> INSERT INTO hr.test_interval VALUES (5, NULL, INTERVAL '5 23:59' DAY TO MINUTE);
```

1개 행 이(가) 삽입되었습니다.

☞ 데이터 입력 시에 YEAR 다음에 자리 수 (3)을 누락하면, 기본적으로 최대 2자리까지만 허용되므로
100년은 3자리이기 때문에 오류가 발생됩니다.

```
SQL> INSERT INTO hr.test_interval VALUES (8, INTERVAL '100' YEAR, INTERVAL '240' DAY(3));
```

명령의 13행에서 시작하는 중 오류 발생 -

```
INSERT INTO hr.test_interval VALUES (8, INTERVAL '100' YEAR, INTERVAL '240' DAY(3))
```

오류 발생 명령행: 13 열: 50

오류 보고 -

SQL 오류: ORA-01873: 간격의 선행 정밀도가 너무 작습니다

01873. 00000 - "the leading precision of the interval is too small"

*Cause: The leading precision of the interval is too small to store the specified interval.

*Action: Increase the leading precision of the interval or specify an interval with a smaller leading precision.

4> 입력된 INTERVAL 형식의 데이터 확인 및 현재 세션의 날짜시간에 기간데이터 활용

```
SQL> SELECT T_ID, IYTM, IDTS, localtimestamp(0) "LOCALTIMESTAMP",
       localtimestamp(0)+IYTM "+IYTM", localtimestamp(0)+IDTS "+IDTS"
  FROM hr.test_interval ;
```

T_ID	IYTM	IDTS	LOCALTIMESTAMP	+IYTM	+IDTS
1	+999-11	999 23:59:59.99	2017/07/09 21:24:04	3017/06/09 21:24:04	2020/04/04 21:24:03
2	+10-00	24 0:0:0.0	2017/07/09 21:24:04	2027/07/09 21:24:04	2017/08/02 21:24:04
3	+05-00	0 4:0:0.0	2017/07/09 21:24:04	2022/07/09 21:24:04	2017/07/10 01:24:04
4		10 0:30:0.0	2017/07/09 21:24:04		2017/07/19 21:54:04
5		5 23:59:0.0	2017/07/09 21:24:04		2017/07/15 21:23:04

☞ 현재 세션의 날짜시간 데이터에 기간데이터가 더해진 날짜시간 데이터를 확인할 수 있습니다.

5> 실습에 사용한 HR.TEST_INTERVAL 테이블을 삭제합니다.

```
SQL> DROP TABLE hr.test_interval PURGE ;
table HR.TEST_INTERVAL이(가) 삭제되었습니다.
```

10-4. 제약조건 (CONSTRAINT) 개념.

- 제약조건(Data Integrity Constraint)을 사용하는 목적.

제약 조건(Data Integrity Constraint 또는 CONSTRAINT)은 사용자가 요구하는 **업무규칙(Business Rule)**을 데이터에 대하여 구현한 것으로, 데이터베이스에 저장되는 데이터가 준수해야 하는 규칙입니다. 즉, 지정된 규칙을 준수하는 데이터만 데이터베이스에 저장되어서 사용될 수 있습니다.

- 데이터베이스에 가능한 제약조건(CONSTRAINT) 5가지.

제약조건의 타입	설명
NOT NULL	지정된 컬럼에 반드시 값이 입력되어야 합니다.
UNIQUE	테이블에 있는 기존 행의 컬럼에 입력된 값과 같은 값(중복된 값)이 입력될 수 없습니다.
PRIMARY KEY	테이블에 있는 각 행을 고유하게 식별할 수 있는 데이터 또는 데이터 조합입니다.
FOREIGN KEY	다른 테이블(참조되는 테이블)의 컬럼에 없는 값은 지정된 컬럼에 입력될 수 없습니다.
CHECK	명시된 조건을 위반하는 데이터는 입력될 수 없습니다.

10-4-1. NOT NULL 제약조건.

- 한 행 입력 시에 NOT NULL 제약조건이 지정된 컬럼에는 반드시 값이 입력되어야 합니다.

- 예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼2에 NOT NULL 제약조건을 정의했다면,

컬럼1	컬럼2	행 입력
A	10	← INSERT(1)은 컬럼 2에 데이터가 있으므로 행이 정상적으로 입력됩니다.
B		← INSERT(2)는 컬럼 2에 데이터가 없는 NULL 상태 이므로 행이 입력되지 않고 오류가 발생됩니다.

10-4-2. 고유키 (UNIQUE) 제약조건.

- 테이블에 있는 모든 행에 대하여 지정된 컬럼(또는 컬럼들)에 입력된 값과 같은 값(중복된 값)이 입력될 수 없습니다. 또한 입력되는 값이 없으면, 검사를 수행하지 않습니다.

- 예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼2에 UNIQUE 제약조건을 정의했다면,

컬럼1	컬럼2	행 입력
A	10	
B	20	← INSERT(1)은 기존의 행에 입력된 컬럼2의 값과 중복되지 않으므로 행이 입력됩니다.
C	10	← INSERT(2)는 기존의 행의 컬럼2의 값과 중복되므로, 입력 시에 오류가 발생됩니다.
D		← INSERT(3)은 입력되는 행의 컬럼2가 NULL 이므로 검사하지 않고 행이 입력됩니다.
E		← INSERT(4)은 입력되는 행의 컬럼2가 NULL 이므로 검사하지 않고 행이 입력됩니다.

10-4-3. 기본키 (PRIMARY KEY) 제약조건.

- 테이블에 저장된 각 행을 고유하게 식별할 수 있는 사용자 데이터 또는 데이터의 조합을 의미합니다.
- 각 행을 대표하는 의미로 사용되는 데이터는 테이블에 하나만 있으면 되므로, 다른 제약조건과는 달리 테이블에 오직 하나의 기본키 제약조건만 정의할 수 있습니다. 또한 행을 고유하게 식별해야 하므로 기능상 데이터도 반드시 있어야 하며(NOT NULL) 값이 다른 행의 값과 중복되면 안됩니다(UNIQUE).

- 예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼1에 기본키 제약조건을 정의했다면,

컬럼1	컬럼2	행 입력
10	A	
20	A	← INSERT(1)은, 기존의 행에 입력된 컬럼1의 값과 중복되지 않으므로 행이 입력됩니다.
10	B	← INSERT(2)는 기존의 행에 입력된 컬럼1의 값과 중복되므로 오류가 발생됩니다.
	B	← INSERT(3)은 입력되는 행의 컬럼1이 데이터가 없으므로(NULL) 오류가 발생됩니다.

10-4-4. 외래키 (FOREIGN KEY) 제약조건.

- 외래키 제약조건은 기본키 제약조건을 참조하는 제약조건입니다. 즉, 외래키가 정의된 컬럼의 값은 반드시 기본키가 정의된 컬럼의 값을 중 하나이어야 합니다.

가 , UNIQUE
 , UNIQUE NOT NULL
- 사원테이블의 부서코드에 외래키 제약조건이 정의되지 않으면, 부서테이블의 부서코드와 상관없이, 사원테이블의 부서코드 컬럼에 아무 값(아래 그림에서 부서코드가 3인 행)이나 삽입될 수 있습니다.
 그렇지만, 사원테이블의 부서코드 컬럼에 외래키 제약조건이 정의되어 있다면, 사원테이블에 행을 삽입할 때,

부서_테이블의 부서코드 컬럼에 없는 값을 가지는 행이 사원_테이블에 삽입될 수 없습니다.

- 다음처럼, 부서테이블과 사원테이블이 구성되어 있고, 사원테이블의 부서코드 컬럼에, 부서테이블의 기본키 컬럼인 부서코드 컬럼을 참조하는 외래키 제약조건이 정의되어 있다면, 사원테이블에 데이터를 입력할 때, 다음처럼 외래키 제약조건이 데이터에 대한 검사를 수행합니다.



- **INSERT(1)**은, 부서코드에 입력되는 값 200이 부서테이블의 부서코드 컬럼에 있는 값이므로 행이 입력됩니다.
- **INSERT(2)**는 부서코드에 입력되는 값 3은 부서_테이블의 부서코드 컬럼에 없는 값이므로, **오류가 발생됩니다.**
- **INSERT(3)**은 부서코드에 검사할 데이터가 없는 NULL 이므로 검사하지 않고 행이 입력됩니다.

- 외래키 제약조건이 정의되는 테이블을 **CHILD-테이블**이라고 하며, 외래키 제약조건에 의해 참조되는 테이블(기본키 제약조건이 정의된 테이블)을 **PARENT-테이블**이라고 합니다.

- 기본키-외래키 관계에 있는 두 테이블에 대하여, DML 수행 시에 다음의 오류가 발생될 수 있습니다.

☞ HR.EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼에는, 외래키 제약조건이 정의되어 있으며, 이 제약조건은 HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼에 정의된 기본키 제약조건을 참조하는 있습니다.

☞ HR.EMPLOYEES 테이블의 외래키 제약조건 때문에 다음의 DML 작업 시에 오류가 발생될 수 있습니다.

HR.EMPLOYEES 테이블에 대한 INSERT/UPDATE	HR.DEPARTMENTS 테이블에 대한 DELETE/UPDATE
<pre>SQL> UPDATE hr.employees SET department_id = 55 WHERE department_id = 110; UPDATE hr.employees * ERROR at line 1: ORA-02291: integrity constraint (HR.EMP_DEPT_FK) violated - parent key not found</pre>	<pre>SQL> DELETE FROM hr.departments WHERE department_id = 60; DELETE FROM hr.departments * ERROR at line 1: ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated - child record found</pre>
HR.DEPARTMENTS.DEPARTMENT_ID에 값 55가 없기 때문에 발생된 오류입니다.	HR.EMPLOYEES.DEPARTMENT_ID에 60인 행이 존재하기 때문에 발생된 오류입니다.

10-4-5. CHECK 제약조건.

- 명시된 조건을 위반하는 데이터는 입력될 수 없습니다.

- 조건 기술 시에 다음의 표현식들은 기술하면 안됩니다.

- CURRVAL, NEXTVAL, LEVEL, ROWNUM 같은 PSEUDO 컬럼은 사용할 수 없습니다.
- SYSDATE, SYSTIMESTAMP, CURRENT_TIMESTAMP, LOCALTIMESTAMP, CURRENT_DATE, UID, USERENV 함수들을 사용할 수 없습니다.
- 다른 행에 있는 다른 값을 참조하는 SELECT 문(즉, 하위질의)을 사용할 수 없습니다.

- 예를 들어 컬럼2에 [컬럼2 > 0]인 CHECK 제약조건을 정의했다면,

컬럼1	컬럼2	행 입력
A	10	
A	20	← INSERT(1)은 값 20이 0 보다 크므로 조건을 만족하기 때문에 행이 입력됩니다.
A	0	← INSERT(2)는 값이 0 이므로 조건을 만족하지 않기 때문에, 오류가 발생됩니다.
A		← INSERT(3)은 입력되는 행의 컬럼2가 NULL 이므로 검사하지 않고 행이 입력됩니다.

10-4-6. 테이블 생성 시에 제약조건을 정의하는 방법.

- 제약조건을 정의할 때, CONSTRAINT 제약조건 이름 옵션을 명시하여 이름을 지정하는 것을 권장합니다. 만약 제약조건을 정의할 때, 이름을 명시하지 않으면, [SYS_Cnnnnnn] 형식(n은 정수)으로 된 이름이 자동으로 지정됩니다.
- 제약조건은 테이블 생성 시에 정의할 수 있으며, 또는 테이블 생성 후에 추가할 수도 있습니다.
- 고유키/기본키 제약조건은, 중복된 데이터를 빠르게 찾기 위하여 반드시 인덱스를 사용해야 합니다. 따라서, 사용할 수 있는 인덱스가 없으면, 고유키/기본키 제약조건이 정의될 때, 제약조건이 정의된 컬럼(들)을 인덱스-키로 가지는 인덱스가 제약조건과 동일한 이름으로 자동으로 생성됩니다.
- 제약 조건을 정의하는 방법.

컬럼-레벨로 정의	<ul style="list-style-type: none"> ○ 컬럼 정의 구문 안에서 정의합니다. ○ NOT NULL 제약조건은 컬럼-레벨 방식으로만 정의해야 합니다.
테이블-레벨로 정의	<ul style="list-style-type: none"> ○ 컬럼 정의 구문 밖에서 (즉, 콤마(,)로 구분하여) 따로 정의합니다. ○ 기본키, 고유키 및 외래키 제약조건은 필요한 경우, 둘 이상의 컬럼을 조합하여, 하나의 제약조건을 정의(복합키 제약조건)할 수 있으며, 이 때는 반드시 테이블-레벨 방식으로 정의합니다. ○ CHECK 제약조건의 경우에는 위의 예제처럼 2개의 컬럼에 2개의 조건을 AND로 묶어서 정의한 경우에는 테이블-레벨로 정의해야 합니다.

[실습] 다음의 테이블 생성 구문을 실행하여, 테이블 생성 시에 제약조건을 정의하는 방법을 실습합니다.

```
SQL> CREATE TABLE hr.emps91 (
    eid NUMBER(4),
    name VARCHAR2(4),
    address VARCHAR2(30) DEFAULT 'SEOUL',
    salary NUMBER(8),
    jumin VARCHAR2(13) CONSTRAINT nn_jname_emps91 NOT NULL,
    deptid NUMBER(4),
    CONSTRAINT pk_eid_name_emps91 PRIMARY KEY(eid, name),
    CONSTRAINT uk_jumin_emps91 UNIQUE(jumin),
    CONSTRAINT ck_sal_addr_emps91 CHECK(salary > 0 and address is not null),
    CONSTRAINT fk_emps_dept91 FOREIGN KEY(deptid)
        REFERENCES hr.departments(department_id) ON DELETE SET NULL );
```

table HR.EMPS91이(가) 생성되었습니다.

```
SQL> CREATE TABLE hr.emps92 (
    empid NUMBER(6) PRIMARY KEY,
    f_name VARCHAR2(20),
    l_name VARCHAR2(25) NOT NULL,
    email VARCHAR2(25) NOT NULL UNIQUE,
    phone_no VARCHAR2(20),
    hire_date DATE NOT NULL,
    job_id VARCHAR2(10) NOT NULL,
    salary NUMBER(8,2) CHECK (salary>0),
    comm_pct NUMBER(2,2),
    manager_id NUMBER(6),
    dept_id NUMBER(4) REFERENCES hr.departments (department_id) ON DELETE CASCADE );
```

table HR.EMPS92이(가) 생성되었습니다.

10-5. 외래키 제약조건 기술 시에 사용되는 키워드의 의미 및 주의점.

- 외래키 제약조건을 정의할 때, 컬럼 레벨 문법을 사용하는 경우에는 FOREIGN KEY(컬럼명)를 명시하면 안됩니다.
FOREIGN KEY(컬럼명)은, 테이블 레벨 방법으로 기술 시에만 명시합니다.
- REFERENCES 키워드 다음에는 참조되는 테이블이름(컬럼이름)을 명시하고, 이 때 참조되는 컬럼에는 기본키 또는 고유키 제약조건이 반드시 설정되어 있어야만 합니다.

- 부서_테이블의 부서코드에 기본키 제약조건이 정의되어 있고, 이 기본키 제약조건을 참조하는 외래키 제약조건이 **ON DELETE CASCADE** 또는 **ON DELETE SET NULL** (옵션이 없이) 사원_테이블의 부서코드에 정의되어 있을 때, 부서테이블의 참조되는 키 값을 가지는 행을 삭제하려고 시도하면, 오류가 발생됩니다.

DELETE(1): 사원_테이블의 부서코드 컬럼에 10 인 행에 의하여 부서_테이블의 부서코드 컬럼이 10인 행이 참조되고 있으므로 행이 삭제되지 않고 오류가 발생됩니다.

DELETE(2): 사원_테이블의 부서코드 컬럼에 30 인 행이 없기 때문에 참조되지 않으므로 행이 삭제됩니다.

부서_테이블		사원_테이블		
부서코드 (PK)	부서명	사번 (PK)	이름	부서코드 (FK)
10	오라클	100	관우	10
20	시스템	101	유비	20
30	개발	102	장비	10
		103	조조	10
		104	여포	20

- 만약 **사원_테이블의 외래키 제약조건을 정의할 때 ON DELETE CASCADE 옵션을 명시하면, DELETE(1)을 실행 시에 사원_테이블의 부서코드 10인 행이 먼저 삭제된 후, 부서_테이블에서도 행이 삭제됩니다.**
- 만약 **사원_테이블의 외래키 제약조건을 정의할 때 ON DELETE SET NULL 옵션을 명시하면, DELETE(1)을 실행 시에 사원_테이블의 해당 행에서 부서코드 10을 NULL로 수정한 후, 부서_테이블에서 행이 삭제됩니다.**

10-6. 하위질의를 이용하여 테이블 생성

- 하위질의를 이용해서 기존 테이블의 컬럼 정의 및 데이터를 복사하면서 새로운 테이블을 생성할 수 있습니다.
- NOT NULL 제약조건만 복사되며, 기본키, 외래키, UNIQUE, CHECK 제약조건은 복사되지 않습니다.**
- CLOB, BLOB 데이터유형 컬럼은 복사되지만, LONG, LONG RAW 데이터유형 컬럼을 복사되지 않습니다.

[실습] 다음의 문장을 실행하여, 하위질의를 이용하여 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.dept70
      AS SELECT employee_id, last_name, salary*12 ANNSAL, hire_date
        FROM hr.employees
       WHERE department_id = 70 ;
```

table HR.DEPT70이(가) 생성되었습니다.

```
SQL> CREATE TABLE hr.dept80 (empno, ename, annsal, hiredate)
      AS SELECT employee_id, last_name, salary*12, hire_date
        FROM hr.employees
       WHERE department_id = 80 ;
```

table HR.DEPT80이(가) 생성되었습니다.

☞ 하위질의에서 SELECT 절에 명시된 컬럼이 함수로 처리되거나 산술 연산이 포함된 표현식으로 기술된 경우, 반드시 컬럼 Alias로 처리해야 합니다. 처리 하지 않은 경우 다음처럼 오류가 발생됩니다.

```
SQL> CREATE TABLE hr.dept801
      AS SELECT employee_id, UPPER(last_name) as last_name, salary*12, hire_date
        FROM hr.employees WHERE department_id = 80 ;
명령의 1 행에서 시작하는 중 오류 발생 -
CREATE TABLE hr.dept801
      AS SELECT employee_id, UPPER(last_name) as last_name, salary*12, hire_date
        FROM hr.employees WHERE department_id = 80
오류 발생 명령행: 2 열: 68
오류 보고 -
SQL 오류: ORA-00998: 이 식은 열의 별명과 함께 지정해야 합니다
00998. 00000 - "must name this expression with a column alias"
*Cause:
*Action:
```

10-7. ALTER TABLE 문장.

- ALTER TABLE 문장을 이용하여 다음과 같은 작업을 수행할 수 있습니다.

- 컬럼 **추가**/컬럼 **삭제**/컬럼 **수정**이 가능합니다.
- 새로 추가된 컬럼에 디폴트 값을 정의할 수 있습니다.
- 컬럼의 **이름 변경**
- 제약조건(CONSTRAINTS)을 **추가**/**삭제**/DISABLE/ENABLE.
- 읽기 전용 상태로 테이블 변경

☞ 위와 관련된 내용은 본 교재의 이 후의 단원에서 설명합니다.

[참고] 읽기 전용 테이블 설정(11gNF)

- 11g 버전부터 사용자가 원하는 경우, 테이블에 대하여 READ ONLY를 지정하여 테이블을 읽기 전용 모드로 변경할 수 있습니다.
- READ-ONLY 모드인 테이블에 대해서는 DML문 또는 SELECT ...FOR UPDATE 문을 실행할 수 없습니다.
- READ-ONLY 모드인 테이블에 대해서 테이블의 데이터를 수정하지 않는 DDL-문은 실행할 수 있습니다. 예를 들어 테이블과 연결된 인덱스에 대한 작업은 테이블이 READ ONLY 모드일 때도 가능합니다. 단, 테이블의 데이터를 변경시키는 DDL-문은 실행될 수 없습니다.
- READ ONLY 모드인 테이블을 삭제할 수 있습니다.

[실습] HR.DEPARTMENTS 테이블을 READ ONLY 모드 및 READ WRITE 모드로 변경해 봅니다.

1> HR.DEPARTMENTS 테이블을 READ ONLY 모드로 설정합니다.

```
SQL> ALTER TABLE hr.departments READ ONLY ;
```

table HR.DEPARTMENTS이(가) 변경되었습니다.

2> HR.DEPARTMENTS 테이블을 READ WRITE 모드로 설정합니다.

```
SQL> ALTER TABLE hr.departments READ WRITE ;
```

table HR.DEPARTMENTS이(가) 변경되었습니다.

10-8. 테이블 삭제(DROP TABLE 문장).

- 테이블을 삭제하면, 데이터베이스에 정의된 테이블의 정의(테이블이름, 컬럼이름, 컬럼의 데이터유형)를 삭제합니다.
- 테이블을 삭제하면, 테이블과 관련된 INDEX, CONSTRAINT, TRIGGER 객체도 모두 삭제됩니다.
단, VIEW 객체는 삭제되지 않고 사용할 수 없는 상태로 변경됩니다.
- 삭제되는 테이블에게 할당된 디스크 상의 저장 공간(SEGMENT)은 사용이 끝난 공간으로 관리 됩니다.
- DROP TABLE 문장은 DDL 문장(실행 완료 시 자동으로 COMMIT 됨)이기 때문에, 룰백이 되질 않습니다.
- 10g-버전부터는 휴지통(RECYCLEBIN) 기능에 의하여, 테이블을 삭제하면, 관련된 시스템 정의가 삭제되지 않고 휴지통

정보로 관리됩니다. 휴지통에 대해서는 뒤의 과정에서 설명합니다.

[실습] 특정 배치 작업을 위해 하위질의를 이용하여 생성한 테이블은 작업 후에 더 이상 필요 없는 경우,

아래의 구문으로 삭제하는 것을 권장합니다.

```
SQL> DROP TABLE hr.dept80 ;
```

table HR.DEPT80이(가) 삭제되었습니다.

)

DROP TABLE

CASCADE CONSTRAINTS;

==>

[실습] DROP TABLE 구문을 이용하여 본 과정에서 실습 시에 생성한 테이블을 모두 삭제해 봅시다.

```
SQL> DROP TABLE hr.dept70 PURGE ;
```

table HR.DEPT70이(가) 삭제되었습니다.

```
SQL> DROP TABLE hr.cust PURGE ;
```

table HR.CUST이(가) 삭제되었습니다.

```
SQL> DROP TABLE hr.emps91 PURGE ;
```

table HR.Emps91이(가) 삭제되었습니다.

```
SQL> DROP TABLE hr.emps92 PURGE ;
```

table HR.Emps92이(가) 삭제되었습니다.

☞ 위에서처럼 PURGE 옵션(10gNF)을 추가하면, 삭제되는 테이블의 시스템 정보가 휴지통에서 관리되지 못하고 데이터베이스에서 모두 삭제됩니다. PURGE 옵션은 가급적 사용을 자제하시기를 권장합니다.

만약 테이블을 삭제 및 절단해야 한다면, 객체단위 백업(테이블 익스포트)을 반드시 수행한 후,
테이블 삭제 및 절단을 수행하십시오.

(119

)

```
TRUNCATE TABLE ;
```

11 테이블 외의 오라클 데이터베이스 객체(VIEW, SEQUENCE, INDEX, SYNONYM).**◆ 학습 목표.**

- **뷰(VIEW) 객체**를 사용하는 목적과 뷰 객체를 생성하는 방법을 학습합니다.
- **시퀀스(SEQUENCE) 객체**의 기능과 시퀀스 객체를 생성하는 방법을 학습합니다.
- **인덱스(INDEX) 객체**를 사용하는 목적과 인덱스 객체를 생성하는 방법을 학습합니다.
- 동의어(SYNONYM) 객체의 기능과 동의어 객체를 생성하는 방법을 학습합니다.

11-1. VIEW, SEQUENCE, INDEX, SYNONYM 객체에 대한 개요.

객체-형식(TYPE)	용도
VIEW	하나 또는 둘 이상의 테이블로부터 논리적으로 구성되는 레코드 단위를 표시합니다. 주로 데이터에 대한 Access를 제한하기 위하여 사용합니다.
SEQUENCE	디폴트로 고유한 숫자 값을 생성하여 사용자 대신 테이블에 대한 데이터 입력 값으로 사용하기 위하여 사용합니다.
INDEX	DISK I/O를 줄여서 테이블의 행을 빠르게 찾아내어 쿼리의 처리 성능을 올리기 위하여 사용합니다.
SYNONYM	데이터베이스 테이블에 대한 대체-이름(동의어)을 정의합니다.

11-2. 뷰(VIEW).

- 뷰는, 테이블의 일부 데이터를 조회하는 SELECT 문이 정의된 객체입니다. 따라서, 저장 공간(STORAGE)에 저장된 데이터를 가지는 테이블과는 달리 VIEW는 사용자 데이터를 저장하지 않습니다.

[실습] HR.EMPLOYEES 테이블로부터 department_id가 30인 부서에 근무하는 사원의 employee_id, first_name, salary를 조회하는 HR.SAL30vw 이름의 뷰를 생성합니다.

```
SQL> CREATE VIEW hr.sal30vw
      AS
        SELECT employee_id, first_name, salary
        FROM hr.employees
       WHERE department_id = 30 ;
```

View created.

☞ 뷰에 정의된 SELECT 문에 명시된 테이블 (위의 예에서 HR.EMPLOYEES)을 뷰에 대한 BASE TABLE이라고 합니다.

☞ 뷰를 생성하기 위해서는 CREATE VIEW 시스템 권한이 필요합니다.

[실습] 생성된 HR.SAL30vw 뷰를 통해 조회할 수 있는 HR.EMPLOYEES 테이블의 컬럼을 확인해 봅니다.

```
SQL> DESC hr.sal30vw
Name          Null?    Type
-----        -----
EMPLOYEE_ID   NOT NULL NUMBER(6)
FIRST_NAME    VARCHAR2(20)
SALARY        NUMBER(8,2)
```

[실습] 생성된 HR.SAL30VW 뷰를 이용하여, HR.EMPLOYEES 테이블의 데이터를 조회해 봅니다.

```
SQL> col first_name format a15
SQL> SELECT * FROM hr.sal30vw ;
```

EMPLOYEE_ID	FIRST_NAME	SALARY
114	Den	11000
115	Alexander	3100
116	Shelli	2900
117	Sigal	2800
118	Guy	2600
119	Karen	2500

6 rows selected.

11-2-1. 뷰를 사용하는 목적.

- 사용자들이 테이블의 일부 데이터에만 ACCESS하도록 **데이터에 대한 ACCESS를 제한해야 할 필요성이 있을 때 주로 VIEW를 사용**합니다. 즉, 다른 데이터베이스 사용자들에게 hr.employees 테이블에 대한 사용 권한을 부여하지 않고, hr.sal30vw VIEW에 대한 사용 권한만을 부여한다면, hr 계정 이 외의 다른 데이터베이스 사용자들은 VIEW에 정의된 SELECT문의 결과 레코드에 해당하는 데이터만을 조회할 수 있습니다.

11-2-2. VIEW 생성 실습-1.

- HR.EMPLOYEES 테이블에서 department_id가 50인 사원들의 employee_id, 대문자로 표시되는 last_name, 연봉을 표시하는 salvu50 뷰를 생성하고 뷰를 통해 테이블의 데이터를 조회하시오.

```
SQL> CREATE VIEW hr.salvu50
      AS SELECT employee_id, UPPER(last_name) AS NAME, salary*12 AS ANN_SAL
        FROM hr.employees
       WHERE department_id = 50;
```

View created.

```
SQL> SELECT * FROM hr.salvu50 ;
```

EMPLOYEE_ID	NAME	ANN_SAL
198	O'CONNELL	31200
199	GRANT	31200
120	WEISS	96000
...		

45 rows selected.

[실습] HR.EMPLOYEES 테이블의 데이터를 이용하여 department_id별 salary의 합계를 구하는 뷰를 생성하고, 생성된 뷰를 통해 테이블의 데이터를 조회하시오.

```
SQL> CREATE VIEW hr.sum_sal_dept_vu
  AS SELECT department_id, SUM(salary) sumsal
    FROM hr.employees
   GROUP BY department_id ;
```

View created.

```
SQL> SELECT * FROM hr.sum_sal_dept_vu ;
```

DEPARTMENT_ID	SUMSAL
100	51600
30	24900
	7000
20	19000
70	10000
90	58000
110	20300
50	156400
40	6500
80	304500
10	4400
60	28800

[실습] HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 데이터를 이용하여 department_name 별 사원의 최소 salary, 최대 salary, 평균 salary를 표시하는 hr.dept_sal_vu 뷰를 생성하고, 생성된 뷰를 통해 테이블의 데이터를 조회하시오.

```
SQL> CREATE VIEW hr.dept_sal_vu (NAME, MINSAL, MAXSAL, AVGSAL)
  AS SELECT UPPER(d.department_name), MIN(e.salary), MAX(e.salary), AVG(e.salary)
    FROM hr.employees e INNER JOIN hr.departments d
   ON (e.department_id = d.department_id)
  GROUP BY d.department_name;
```

View created.

```
SQL> SELECT * FROM hr.dept_sal_vu ;
```

NAME	MINSAL	MAXSAL	AVGSAL
ADMINISTRATION	4400	4400	4400
ACCOUNTING	8300	12000	10150
HUMAN RESOURCES	6500	6500	6500
PUBLIC RELATIONS	10000	10000	10000
EXECUTIVE	17000	24000	19333.3333
IT	4200	9000	5760
PURCHASING	2500	11000	4150
SHIPPING	2100	8200	3475.55556
FINANCE	6900	12000	8600
SALES	6100	14000	8955.88235
MARKETING	6000	13000	9500

11 rows selected.

- 위에서처럼 하위질의에 명시하는 컬럼Alias를 VIEW의 이름 옆에 명시하여 VIEW를 생성할 수도 있습니다.
- VIEW 생성 시에 SELECT 절에 명시된 컬럼이 함수로 처리되거나 연산 처리가 된 경우, 컬럼Alias를 명시해야 합니다. 그렇지 않으면, 아래처럼 오류가 발생됩니다.

```
SQL> CREATE VIEW hr.salvu50_2
      AS SELECT employee_id, UPPER(last_name) AS NAME, salary*12
            FROM hr.employees
           WHERE department_id = 50;
      AS SELECT employee_id, UPPER(last_name) AS "NAME", salary*12
                           *
ERROR at line 2:
ORA-00998: must name this expression with a column alias
```

11-2-3. VIEW 생성 실습-2: WITH READ ONLY 옵션이 포함된 VIEW 생성.

- VIEW를 통한 DML 수행을 방지하기 위하여 WITH READ ONLY를 사용하여 VIEW를 생성합니다.

, empvu10 DML

[실습] 아직 생성되지 않은 HR.EMPLOYEES 테이블에 대하여 department_id가 80 인 사원의 employee_id, last_name, job_id 컬럼들로 구성된 사원정보를 표시하는 HR.EMPVU10 뷰를 생성하시오.

```
SQL> CREATE VIEW hr.empvu10 (employee_number, employee_name, job_title)
      AS SELECT employee_id, last_name, job_id
            FROM hr.employees
           WHERE department_id = 80
             WITH READ ONLY ;
```

View created.

11-2-4. VIEW 생성 실습-3: WITH CHECK OPTION 옵션이 포함된 VIEW 생성.

- 뷰의 WHERE 절에 WITH CHECK OPTION을 명시하면, 뷰를 통해 BASE-테이블에 DML 시에, 뷰의 WHERE 절에 명시된 조건을 모두 만족할 때만 뷰를 통하여 뷰의 BASE-테이블에 정상적인 DML이 수행됩니다.

[실습] 다음의 실습을 통해 WITH CHECK OPTION이 포함된 뷰를 사용하는 방법을 확인합니다.

1> HR.EMPLOYEES 테이블을 이용하여, department_id가 20인 사원의 모든 정보를 표시하는 hr.empvu20 뷰를 생성하시오.

단, WITH CHECK OPTION을 이용하여 DML시에 department_id가 20인지 검사하도록 하시오.

```
SQL> CREATE VIEW hr.empvu20
  AS SELECT *
    FROM hr.employees
   WHERE department_id = 20 WITH CHECK OPTION ;
```

View created.

2> 생성된 VIEW를 이용하여 hr.employees 테이블에 다음의 입력 및 행 삭제를 시도합니다.

```
SQL> INSERT INTO hr.empvu20
      VALUES (901, 'SH','SHIN','ks7009',NULL ,sysdate,'AC_MGR', 30000,NULL ,205, 20);

1 row created. <-- 입력 성공
```

```
SQL> INSERT INTO hr.empvu20
      VALUES (902, 'SH','SHIN','ks7005',NULL ,sysdate,'AC_MGR', 30000,NULL ,205, 50);
INSERT INTO hr.empvu20
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation <-- 입력 오류
```

```
SQL> DELETE FROM hr.empvu20 WHERE department_id = 50 ;
```

0 rows deleted. <-- 삭제 되지 않음.

☞ 뷰의 WHERE 절의 조건을 만족하지 않은 경우에는, 입력 시 오류가 발생되고, DELETE 도 되지 않습니다.

[참고] 뷰를 통해서 테이블의 데이터를 INSERT/UPDATE 시에, DEFAULT 키워드를 사용하면 아래와 같은 오류가 발생됩니다.

```
SQL> INSERT INTO hr.empvu20
      VALUES (901, 'SH','SHIN','ks7009',DEFAULT ,sysdate,'AC_MGR', 30000, DEFAULT, 205, 20) ;
VALUES (901, 'SH','SHIN','ks7009',DEFAULT ,sysdate,'AC_MGR', 30000, DEFAULT, 205, 20)
*
ERROR at line 2:
ORA-32575: Explicit column default is not supported for modifying views
```

11-2-5. VIEW에 정의된 SELECT문 수정.

■ 기존 VIEW에 정의된 SELECT문을 수정하고 싶을 때, [CREATE OR REPLACE VIEW] 구문을 이용합니다.

■ VIEW에 정의된 권한 설정이 유지되기 때문에 [VIEW를 삭제하고 다시 생성하는 것]보다 편리합니다.

[실습] 앞에서 생성한 HR.SALVU50 뷰로 표시되는 데이터에서 (1) NAME에 first_name과 last_name 값을 붙여서 표시하고, (2) 연봉 대신 salary로, (3) department_id 컬럼도 표시하도록 뷰의 SELECT문을 수정하시오.

```
SQL> CREATE OR REPLACE VIEW hr.salvu50
      AS SELECT employee_id AS ID_NUMBER, first_name || ' ' || last_name AS NAME,
                 salary AS SAL, department_id
            FROM hr.employees
           WHERE department_id = 50 ;
```

View created.

☞ 오라클에서는 **ALTER VIEW** 문장으로 뷰에 정의된 SELECT문을 수정할 수 없습니다.

11-2-7. VIEW에 대해서 DML 수행 시에 규칙.

■ 뷰에 정의된 SELECT 문이 아래의 경우에 해당되면, 뷰를 통해서는 BASE-테이블의 데이터를 DELETE 할 수 없습니다.

- 그룹함수가 SELECT절에 사용됨.
- GROUP BY 절이 포함됨.
- SELECT 절에 DISTINCT 또는 ROWNUM 키워드가 사용됨.

■ 뷰에 정의된 SELECT 문이 아래의 경우에 해당되면, 뷰를 통해서는 BASE-테이블의 데이터를 UPDATE 할 수 없습니다.

- 그룹함수가 SELECT절에 사용됨.
- GROUP BY 절이 포함됨.
- SELECT 절에 DISTINCT 또는 ROWNUM 키워드가 사용됨.
- 컬럼이 표현식으로 처리된 경우

■ 뷰에 정의된 SELECT 문이 아래의 경우에 해당되면, 뷰를 통해서는 BASE-테이블에 데이터를 INSERT 할 수 없습니다.

- 그룹함수가 SELECT절에 사용됨.
- GROUP BY 절이 포함됨.
- SELECT 절에 DISTINCT 또는 ROWNUM 키워드가 사용됨.
- 컬럼이 표현식으로 처리된 경우.
- NOT NULL 제약조건이 정의된 BASE TABLE의 컬럼이 VIEW의 SELECT 절에 누락된 경우.

[참고] VIEW에 대하여 DML 수행 시에 규칙 확인을 위한 실습.

[실습] HR.SUM_SAL_DEPT_VU 뷰를 통하여 department_id가 10인 행을 HR.EMPLOYEES 테이블에서 삭제해 보시오.

```
SQL> DELETE FROM hr.sum_sal_dept_vu WHERE department_id =10 ;
DELETE FROM hr.sum_sal_dept_vu WHERE department_id =10
*
ERROR at line 1:
ORA-01732: data manipulation operation not legal on this view
```

☞ VIEW에 정의된 SELECT 문에 GROUP BY가 사용된 경우, VIEW를 통해 표시되는 데이터를 DELETE/UPDATE 하거나 INSERT하여 BASE TABLE에 데이터 조작을 할 수 없습니다.

[실습] HR.SALVU50 뷰를 통하여 employee_id가 197인 사원의 NAME을 'SHSHIN'으로 변경해보시오.

```
SQL> UPDATE hr.salvuo50 SET name = 'SHSHIN' WHERE employee_id= 197 ;
UPDATE hr.salvuo50 SET name = 'SHSHIN' WHERE employee_id= 197
*
ERROR at line 1:
ORA-01733: virtual column not allowed here
```

☞ HR.SALVU50 뷰의 NAME 컬럼은 실제로 HR.EMPLOYEES 테이블의 first_name 컬럼과 last_name 컬럼이 결합된 표현식이기 때문에 UPDATE 시에 위와 같은 오류가 발생됩니다.

[실습] HR.SAL30VW 뷰를 통하여 HR.EMPLOYEES 테이블에 employee_id가 300, first_name이 'SHSHIN', salary가 30000인 사원정보를 입력해보시오.

```
SQL> INSERT INTO hr.sal30vw VALUES (300, 'SHSHIN', 30000) ;
INSERT INTO hr.sal30vw VALUES (300, 'ORACLE', 30000)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("HR"."EMPLOYEES"."LAST_NAME")
```

☞ HR.SAL30VW 뷰를 사용하여 HR.EMPLOYEES 테이블에 대한 데이터 액세스를 제어하는 목적 때문에, 뷰의 SELECT문에 HR.EMPLOYEES 테이블의 NOT NULL이 정의된 컬럼 모두가 포함되지 못했으므로, 입력 시 오류가 발생됩니다.

☞ 다음은 HR.EMPLOYEES 테이블에 대하여 DESCRIBE 명령어를 수행한 결과입니다.

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

11-2-8. VIEW 삭제.

- DROP VIEW 구문을 이용하여, VIEW 객체를 삭제(VIEW에 정의된 SELECT 문을 삭제)합니다.

[실습] 앞에서 생성한 HR.SAL30VM 뷰를 삭제하시오.

```
SQL> DROP VIEW hr.sal30vw ;
```

```
View dropped.
```

11-3. 시퀀스(SEQUENCE).

- 자동으로 고유한 숫자를 생성하는 객체입니다.

- [주문 번호] 데이터처럼, 사용자가 데이터를 직접 입력하는 대신, SEQUENCE 객체가 생성해 준 고유한 숫자 값을 이용하여 사용자가 값을 입력할 필요가 없이 프로그램적으로 데이터를 만들어서 INSERT문에 대신 넣어줄 때 사용되는 객체입니다.
- SEQUENCE 객체를 생성하기 위해서는 CREATE SEQUENCE 시스템 권한이 필요합니다.

- SEQUENCE를 생성하는 기본 문법.

CREATE SEQUENCE 스키마 이름.시퀀스이름

```
START WITH n      INCREMENT BY n      MAXVALUE n 또는 NOMAXVALUE      MINVALUE n 또는 NOMINVALUE  
CYCLE 또는 NOCYCLE      CACHE n 또는 NOCACHE ;
```

[옵션에 대한 간단한 설명]

옵션	기능
START WITH n	숫자 값이 생성되는 첫 시작-값을 지정합니다. 명시하지 않으면, 디폴트로 1이 설정됩니다.
INCREMENT BY n	숫자 값이 생성될 때, 값 사이의 간격을 지정합니다. 명시하지 않으면, 디폴트로 1이 설정됩니다.
MAXVALUE n 또는 NOMAXVALUE	생성될 수 있는 최대값을 지정합니다. 명시하지 않으면, 디폴트로 NOMAXVALUE로 설정되며, 최대 10 ²⁷ 까지 숫자가 생성됩니다.
MINVALUE n	CYCLE 옵션에 의하여 다시 이전의 값을 생성할 때, 시작할 값을 명시합니다. MAXVALUE n 또는 NOMAXVALUE가 설정된 경우, MINVALUE n 옵션을 명시하지 않으면, 디폴트로 1이 설정됩니다.
CYCLE 또는 NOCYCLE	CYCLE 옵션은 MAXVALUE에 도달 후, 다음 값을 생성할 때, MINVALUE n에 지정된 값부터 다시 시작합니다. 명시하지 않으면 NOCYCLE로 설정되며, MAXVALUE에 도달되면 오류가 발생됩니다.
CACHE n 또는 NOCACHE	CACHE n으로 설정하면, 지정된 개수에 해당하는 끝 수를 메모리에 CACHE 시킵니다. 명시하지 않으면 CACHE 20으로 설정됩니다. NOCACHE를 설정하면, 메모리 캐시 기능을 사용하지 않습니다.

[참고] 오라클의 시퀀스 객체는 CACHE 옵션을 이용하여, 시퀀스의 사용할 값을 오라클-서버의 메모리에 캐시하여, 시퀀스에 대한 다음 값의 요청을 빠르게 처리할 수 있습니다.

[실습] 다음의 3개의 시퀀스를 생성합니다.

1> 모든 옵션이 DEFAULT 값을 사용하는 HR.DEPT_DEPTID_SEQ1 시퀀스를 생성하시오.

```
SQL> CREATE SEQUENCE hr.dept_deptid_seq1 ;  
Sequence created.
```

☞ 일반적으로 시퀀스를 생성할 때 사용하는 방법입니다.

2> 메모리에 캐싱 기능을 사용하지 않으면서, 500부터 시작해서 10씩 간격을 두고 최대 9999까지 숫자를 생성하며, 최대값에 도달되었을 경우 자동으로 최소값부터 다시 시작하는 시퀀스를 생성하시오.

```
SQL> CREATE SEQUENCE hr.dept_deptid_seq2  
      START WITH 500  
      INCREMENT BY 10  
      MAXVALUE 9999  
      MINVALUE 500  
      NOCACHE  
      CYCLE ;  
  
Sequence created.
```

3> 메모리에 10개에 해당하는 끝수가 캐싱되면서, 500부터 시작해서 10씩 간격을 두고 최대 9999까지 숫자를 생성하며, 최대값에 도달되었을 경우 오류를 발생시키는 시퀀스를 생성하시오.

```
SQL> CREATE SEQUENCE hr.dept_deptid_seq3  
      START WITH 500  
      INCREMENT BY 10  
      MAXVALUE 9999  
      --MINVALUE 500  
      CACHE 10  
      NOCYCLE ;  
  
Sequence created.
```

11-3-1. 특정 시퀀스에 대한 NEXTVAL 및 CURRVAL 가상컬럼(Pseudo-column).

- 시퀀스에 대한 NEXTVAL 및 CURRVAL 가상컬럼(Pseudo-column)에 대하여 학습합니다.

사용방법	의미
시퀀스이름.NEXTVAL	해당 시퀀스의 다음에 사용할 값을 호출합니다. 오라클-서버에서 관리됩니다.
시퀀스이름.CURRVAL	세션에서 사용한 시퀀스의 마지막 값으로, 만약 세션에서 시퀀스이름.NEXTVAL을 한 번도 호출하지 않은 경우에는 시퀀스를 사용한 적이 없기 때문에, 시퀀스에 대한 CURRVAL 값은 정의되어 있지 않습니다. 따라서, 이러한 상태에서 시퀀스이름.CURRVAL을 호출하면 아래처럼 오류가 발생됩니다. SQL> SELECT DEPT_DEPTID_SEQ1.CURRVAL FROM dual ; SELECT DEPT_DEPTID_SEQ1.CURRVAL FROM dual * ERROR at line 1: ORA-08002: sequence DEPT_DEPTID_SEQ1.CURRVAL is not yet defined in this session 테이블에 한 행의 데이터를 입력할 때, 시퀀스이름.NEXTVAL로 하나의 컬럼에 입력한 값을 같은 행의 다른 컬럼에서 사용하고 싶을 때 시퀀스이름.CURRVAL을 이용합니다.

[참고] 시퀀스에 대한 NEXTVAL 및 CURRVAL 사용 규칙.

다음과 같은 상황에서 NEXTVAL 및 CURRVAL을 사용할 수 있습니다.

- 하위질의의 일부가 아닌 **SELECT문의 SELECT 리스트**. SELECT
 - INSERT문에서 테이블이름 대신 사용된 하위질의의 SELECT 리스트
 - **INSERT문의 VALUES 절**
 - **UPDATE 문의 SET 절**

다음과 같은 상황에서는 NEXTVAL 및 CURRVAL을 사용할 수 없습니다.

- **뷰의 SELECT 리스트**
 - **DISTINCT 키워드가 있는 SELECT 문**
 - **GROUP BY, HAVING 또는 ORDER BY 절이 있는 SELECT 문**
 - SELECT, DELETE 또는 UPDATE 문의 테이블이름 대신 사용된 하위질의
 - **CREATE TABLE 또는 ALTER TABLE 문의 DEFAULT 식**

, 12c DEFAULT
 .NEXTVAL (12cNF)

11-3-2. 시퀀스의 사용.

- 시퀀스의 다음 값을 호출하여 사용하려면, SQL문에서 시퀀스이름.NEXTVAL(예, DEPT_DEPTID_SEQ1.NEXTVAL)을 이용합니다.

[실습] HR.DEPARTMENTS 테이블에 department_name, location_id, manager_id 컬럼은 각각 'Oracle', 2500, 100 값으로 사용자가 직접 입력한 값을 사용하고, department_id 컬럼은 DEPT_DEPTID_SEQ1 시퀀스가 자동으로 생성하는 값을 이용하여 한 행의 데이터를 입력하시오.

```
SQL> INSERT INTO hr.departments(department_id, department_name, location_id, manager_id)
      VALUES (DEPT_DEPTID_SEQ1.NEXTVAL,'Oracle', 2500, 100);
1 row created.
```

- ☞ 이 INSERT 작업의 트랜잭션을 롤백시키면, 데이터는 입력되기 전 상태가 되지만, 사용된 시퀀스의 값은 다시 반환되지 않습니다. 즉. 사용된 값은 사라지게 됩니다.

, ROLLBACK .

[참고] 다음과 같은 경우에 시퀀스를 통해서 입력된 데이터의 값을 사이에 차이(GAP)가 발생될 수 있습니다.

- 접속한 세션이 비정상적으로 접속이 해제되어 롤백(ROLLBACK)이 발생한 경우.
- 데이터베이스 서버가 비정상적으로 중지(System-Crash)된 경우.
- 하나의 시퀀스가 다른 테이블에서도 같이 사용되는 경우.

11-3-3. 시퀀스의 수정.

- 시퀀스의 수정은 시퀀스의 소유자이거나 시퀀스에 대한 ALTER 객체 권한을 가진 사용자만 수행할 수 있습니다.
- 현재 시퀀스의 상태나 또는 시퀀스의 값을 이용하여 입력되는 컬럼의 데이터 상태에 따라 시퀀스의 옵션을 변경할 수 있습니다. 즉, 옵션을 변경 시에 일부 유효성 검사가 수행됩니다.
- 단, START WITH 옵션에 설정된 값은 변경할 수 없습니다.

[실습] HR.DEPT_DEPTID_SEQ1 시퀀스의 CACHE 수를 1,000,000 으로 늘리시오.

```
SQL> ALTER SEQUENCE DEPT_DEPTID_SEQ1 CACHE 1000000 ;
Sequence altered.
```

11-3-4. 시퀀스 삭제.

[실습] 필요 없는 HR.DEPT_DEPTID_SEQ1 시퀀스를 삭제하시오.

```
SQL> DROP SEQUENCE hr.dept_deptid_seq1;
```

```
Sequence dropped.
```

가 가 .
 ,
 - ,
 .

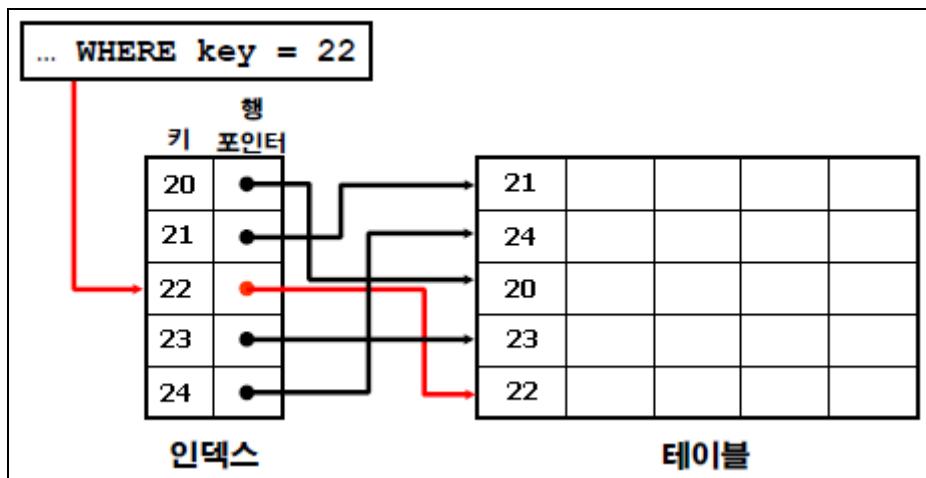
11-4. 인덱스(INDEX).

■ 테이블에 대한 전체 테이블 스캔 방법과 인덱스 스캔 방법 개요.

데이터베이스 서버로 전달된 사용자의 SQL문이 서버-프로세스에 의하여 처리되어 최종적인 결과를 표시하기 위해서는 디스크에 저장된 데이터를 서버의 메모리로 로드 해야 합니다. 이 때, 디스크 상에 저장된 데이터를 메모리로 로드하는 방법으로 (1) 디스크에 구성된 테이블의 저장 공간(세그먼트)에 저장된 모든 데이터 블록을 메모리로 로드하여, 모든 데이터를 처리하는 **전체 테이블 스캔(Full-Table Scan) 방법**과 (2) 인덱스를 이용하여 SQL문의 처리 대상이 되는 행이 저장된 테이블의 데이터 블록만을 디스크로부터 메모리로 로드 하는 **인덱스 스캔(Index-Scan) 방법**의 두 가지가 있습니다.

테이블에 저장된 매우 적은 수의 행을 찾아서 원하는 결과를 제공하는 SQL문(주로 SELECT 문)의 경우에는, **인덱스 객체를 이용하여 원하는 행이 있는 디스크 상의 데이터 블록만을 메모리에 로드 하기 때문에 디스크에 대한 액세스 횟수(이를 I/O 횟수라고 합니다)를 줄일 수 있습니다.** 따라서, 빠르게 SQL문을 처리할 수 있기 때문에 SQL문의 처리 성능을 향상시킬 수 있습니다.

■ 인덱스 개요: 인덱스를 이용하여 테이블의 행을 찾는 과정에 대한 개요.



- 인덱스는 스키마 객체이며, 오라클-서버에서 포인터를 사용하여 행 검색 속도를 높이는데 사용됩니다.
- 신속한 경로 액세스 방식을 사용하여 데이터를 빠르게 찾아 디스크 I/O(입/출력)를 줄일 수 있습니다.

- 인덱스는 테이블의 종속 객체이긴 하지만 테이블과 구분된 저장공간을 가지는 객체입니다.
- 오라클-서버에 의하여 자동으로 사용되고 유지-관리됩니다.

11-4-1. 인덱스를 생성하는 가이드라인.

- 테이블에 저장된 행이 매우 많고 대부분의 쿼리들에서 그 테이블에 저장된 행의 2% ~ 4% 미만의 데이터를 추출할 것으로 기대될 때, 해당 테이블에 꼭 필요한 인덱스를 생성할 것을 권장합니다.
- 주로 SQL문의 WHERE절 또는 조인 조건에 자주 나오는 컬럼을 인덱스-키로 가지는 인덱스를 생성합니다.
- 테이블에 저장된 행이 매우 극소수이고 대부분의 쿼리들에서 그 테이블에 저장된 행의 2% ~ 4% 를 초과하는 다수의 데이터를 추출할 것으로 예상되면, 인덱스를 사용하지 않는 것이 권장됩니다.
- 쿼리의 WHERE절에 그다지 자주 사용되지 않는 컬럼을 인덱스-키로 사용하여 인덱스를 생성하지 마십시오.
- WHERE절에 사용되지만, "표현식의 일부"로서 참조되는 컬럼을 인덱스-키로 사용하여 인덱스를 생성하지 마십시오.
- 자주 UPDATE되어 값이 변경되는 컬럼을 인덱스-키로 사용하여 인덱스를 생성하지 마십시오.
- 인덱스-키의 속성에 따라, 인덱스는 UNIQUE 인덱스와 NON-UNIQUE 인덱스가 있습니다.

UNIQUE 인덱스	인덱스-키의 데이터가 중복되지 않는 인덱스입니다. 기본키 또는 UNIQUE 제약조건을 정의할 때 자동으로 생성되는 인덱스가 UNIQUE 인덱스입니다.
NON-UNIQUE 인덱스	인덱스 키의 데이터가 중복이 허용되는 인덱스입니다. UNIQUE 옵션을 사용하지 않은 CREATE INDEX 문장으로 인덱스를 생성하면, 디폴트로 NON-UNIQUE 인덱스가 생성됩니다.

11-4-2. 인덱스 생성.

"

"가

.

[실습] HR.EMPLOYEES 테이블의 last_name 컬럼의 데이터를 인덱스-키로 가지는 NON-UNIQUE 인덱스를 생성하시오.

```
SQL> CREATE INDEX hr.idx_lname_emp ON hr.employees(last_name) ; Balence-Tree (B-Tree )  
Index created.
```

```
execute dbms_stats.gather_index_stats('HR','idx_lname_emp');
```

☞ 위의 생성실습에서 HR.EMPLOYEES.LAST_NAME 컬럼이 HR.IDX_LNAME_EMP 인덱스의 인덱스-키 입니다.

```
CREATE INDEX hr.EMP_NAME_IX ON hr.employees(last_name, first_name) ;  
CREATE UNIQUE INDEX hr.EMP_EMAIL_UK ON hr.employees(email) ;--<
```

☞ WHERE 절 작성시에 조건절에서 인덱스-키 컬럼을 어떻게 명시하나에 따라 SQL문이 처리될 때, 생성된 인덱스가 사용될 수도 있고, 사용되지 않을 수 있습니다.

○ 아래의 SELECT 문의 경우, 인덱스-키 컬럼에 아무 처리를 하지 않았으므로 인덱스를 이용하여 처리됩니다.

```
SQL> SELECT * FROM hr.employees WHERE last_name='King' ;
```

○ 아래의 SELECT 문의 경우, 인덱스-키 컬럼을 UPPER() 함수로 처리하여 인덱스 객체에 저장된 인덱스-키 정보를 사용할 수 없기 때문에 인덱스를 사용하지 않습니다.

```
SQL> SELECT * FROM hr.employees WHERE UPPER(last_name)='KING' ;
```

[참고] WHERE 절에 명시된 컬럼을 아래처럼 작성하여 정의된 인덱스를 사용하지 못하도록 지정할 수 있습니다.

• 문자 데이터유형 컬럼	컬럼 ''
• 숫자 또는 DATE 데이터유형 컬럼	컬럼 + 0

11-4-3. 인덱스 삭제.

[실습] 앞에서 생성한 HR.IDX_LNAME_EMP 인덱스를 삭제하시오.

```
SQL> DROP INDEX hr.idx_lname_emp ;
```

```
Index dropped.
```

☞ 기본키 및 UNIQUE 제약조건이 사용하는 인덱스는 기본키 및 UNIQUE 제약조건을 먼저 삭제하거나

DISABLE 시켜야지만 인덱스를 삭제할 수 있습니다. 만약, 기본키 및 UNIQUE 제약조건이 사용 중인 인덱스를 삭제하려고 시도하면 다음의 오류가 발생합니다.

```
SQL> DROP INDEX hr.emp_emp_id_pk ;
```

```
ERROR at line 1:
ORA-02429: cannot drop index used for enforcement of unique/PRIMARY KEY
```

```
] db_file_multiblock_read_count      :  
]  
) CREATE INDEX hr.EMP_NAME_IX ON hr.employees(last_name, first_name);
```

11-5. 동의어(SYNONYM).

- **(테이블/뷰) 다른 동의어의 이름에 대한 대체어(동의어)를 정의한 객체입니다.**
- 주로 DATABASE-LINK 객체(둘 이상의 데이터베이스 사이에서 데이터를 액세스하기 위해 사용하는 객체)와 조합된 이름을 간편히 사용하려고 동의어를 생성합니다.

11-5-1. 동의어(SYNONYM) 티입.

NORMAL SYNONYM (일반 동의어)	<ul style="list-style-type: none"> 데이터베이스에 로그인 한 계정이 자신이 소유한 스키마 객체로서 생성하며, 소유자 계정만 사용하는 동의어입니다. 로그인 계정이 NORMAL SYNONYM을 생성하려면 CREATE SYNONYM 시스템 권한을 가지고 있어야 합니다.
PUBLIC SYNONYM (공용 동의어)	<ul style="list-style-type: none"> DBA(SYS, SYSTEM)가 생성하며, 데이터베이스의 모든 로그인 계정이 사용하는 동의어입니다. 로그인 계정이 PUBLIC SYNONYM을 생성하려면 CREATE PUBLIC SYNONYM 시스템 권한을 가지고 있어야 합니다.

11-5-2. 동의어 생성: 접속 계정이 자신의 스키마 객체로서 사용하는 동의어 생성.

[실습] HR 계정으로 접속한 세션에서 HR.EMPLOYEES 테이블에 대한 EMP_SYN 이름의 NORMAL SYNONYM 객체를 생성합니다.

```
SQL> CREATE SYNONYM hr.emp_syn FOR hr.employees ;
synonym HR.EMP_SYN이(가) 생성되었습니다.
```

[실습] 생성한 EMP_SYN 동의어를 이용하여 HR.EMPLOYEES 테이블의 데이터를 조회합니다.

```
SQL> SELECT last_name
   FROM hr.emp_syn
  WHERE department_id = 90 ;

LAST_NAME
-----
King
Kochhar
De Haan
```

11-5-3. 동의어 생성: PUBLIC-SYNONYM(공용 동의어) 생성.

[실습] SYS 계정으로 접속한 세션에서 HR.DEPARTMENTS 테이블에 대한 DEPT_PUSYN 이름의 공용 동의어 객체를 생성합니다.

```
SQL> CREATE PUBLIC SYNONYM dept_pusyn FOR hr.departments ; [ ]  
public synonym DEPT_PUSYN01(가) 생성되었습니다. PUBLIC SYNONYM . SYNONYM
```

[참고] CREATE PUBLIC SYNONYM 시스템 권한이 없는 HR 계정으로 접속하여 PUBLIC SYNONYM을 생성하려고 시도하면 다음의 오류가 발생됩니다.

```
SQL> CREATE PUBLIC SYNONYM dept_pusyn FOR hr.departments ;  
명령의 1 행에서 시작하는 중 오류 발생 -  
CREATE PUBLIC SYNONYM dept_pusyn FOR hr.departments  
오류 발생 명령행: 1 열: 1  
오류 보고 -  
SQL 오류: ORA-01031: 권한이 불충분합니다  
01031. 00000 - "insufficient privileges"  
*Cause: An attempt was made to perform a database operation without  
the necessary privileges.  
*Action: Ask your database administrator or designated security  
administrator to grant you the necessary privileges
```

11-5-4. SYNONYM 삭제.

[실습] HR 계정으로 접속한 세션에서, HR 스키마액체인 EMP_SYN 동의어를 삭제하시오.

```
SQL> DROP SYNONYM hr.emp_syn ;  
Synonym dropped.
```

☞ 일반 동의어는 해당 동의어의 소유자 계정 또는 데이터베이스 관리자 계정이 삭제할 수 있습니다.

[실습] SYS 계정으로 접속한 세션에서 dept_pusyn 공용동의어를 삭제하시오.

```
SQL> DROP PUBLIC SYNONYM dept_pusyn ;  
public synonym DEPT_PUSYN01(가) 삭제되었습니다.
```

☞ 공용 동의어는 DROP PUBLIC SYNONYM 권한을 가진 계정 또는 데이터베이스 관리자 계정이 삭제할 수 있습니다.

12 오리클 데이터베이스 계정 생성 및 데이터베이스 계정에 대한 권한 설정.

◆ 학습 목표.

- 데이터베이스 계정(데이터베이스 사용자)을 생성하는 방법을 학습합니다.
 - 오라클 데이터베이스 서버의 대표적인 시스템 권한(System Privileges)과 객체 권한(Object Privileges)에 대하여 이해합니다.
 - 데이터베이스 계정에게 시스템 권한과 객체 권한을 부여하고 철회하는 방법을 학습합니다.
 - 풀(ROLE)의 정의와 필요성에 대하여 이해하고, 풀을 이용하여 권한을 쉽게 관리하는 방법을 학습합니다.

1. 가 ()
2. 가 ,

| -->
==> CREATE SESSION ,

3. ()
=> CREATE SESSION

[.]
, ALTER USER hr1 ACCOUNT lock PASSWORD EXPIRE ;

=> , DBA가

12-1. 사용자 액세스 제어 개요.

- 데이터베이스 관리자(DBA)는 사용자들의 데이터베이스에 대한 액세스를 제한적으로 제어해야 합니다.
- 데이터베이스에 접속하는 사용자의 액세스를 제어하는 가장 기본적인 방법에는 다음의 작업이 포함됩니다.
 - [CREATE USER]문을 이용하여 고유한 계정이름과 패스워드를 설정하여 오라클 데이터베이스 서버의 계정을 생성합니다.
 - [GRANT]문을 이용하여, 오라클-서버의 접속-계정에게 역할에 필요한 권한만을 부여합니다.
또한 필요한 경우, [REVOKE]문을 이용하여 부여된 권한을 철회하여 권한을 제어합니다.

[참고] SYS 계정과 SYSTEM 계정.

- 오라클 데이터베이스 서버를 구성하면, SYS 계정 및 SYSTEM 계정이 오라클 데이터베이스 서버의 관리자(DBA) 계정으로서 자동으로 생성됩니다. 이 계정들을 삭제하거나 계정들의 권한 설정을 변경하지 마십시오.

12-2. 오라클 데이터베이스 서버의 권한: 시스템 권한과 객체 권한에 대한 이해.

12-2-1. 시스템 권한(SYSTEM PRIVILEGES).

- 데이터베이스에 대한 접속과 스키마 객체 생성 같이, 데이터베이스에 대한 액세스 및 데이터베이스의 특정 작업을 수행할 수 있는 권한이며, **오라클 데이터베이스 서버의 관리자 계정(DBA, 디폴트로 SYS 계정 및 SYSTEM 계정)만 시스템 권한을 부여 및 철회 작업을 수행할 수 있습니다.**
- 다음은 오라클 데이터베이스 관리 업무를 수행하는 관리자(DBA) 계정들만 사용해야 하는 대표적인 시스템 권한입니다.

시스템 권한이름	해당 시스템 권한이 부여되었을 때, 가능한 작업
CREATE USER	데이터베이스 계정을 생성할 수 있습니다.
DROP USER	데이터베이스 계정 및 스키마를 삭제할 수 있습니다.
BACKUP ANY TABLE	exp/expdp 툴을 이용하여 다른 스키마의 테이블을 백업할 수 있습니다.
SELECT ANY TABLE	다른 스키마에 속한 테이블의 데이터를 조회(SELECT)할 수 있습니다.
CREATE ANY TABLE	다른 스키마에 속하는 테이블을 생성할 수 있습니다.
DROP ANY TABLE	다른 스키마에 속한 테이블을 삭제할 수 있습니다.

☞ ANY가 이름에 포함된 시스템권한은, 다른 스키마에 대하여 작업을 수행할 수 있는 시스템권한입니다.

- 다음은 개발자들이 사용하는 데이터베이스 계정에게 부여되는 오라클 데이터베이스의 대표적인 시스템 권한입니다.

시스템 권한이름	해당 시스템 권한이 부여되었을 때, 가능한 작업
CREATE SESSION	오라클 데이터베이스에 접속할 수 있습니다.
ALTER SESSION	일부 세션의 설정을 [ALTER SESSION SET]문을 이용하여 변경할 수 있습니다.
CREATE TABLE	계정의 스키마에 테이블 객체를 생성할 수 있습니다.
CREATE SEQUENCE	계정의 스키마에 시퀀스 객체를 생성할 수 있습니다.
CREATE VIEW	계정의 스키마에 뷰 객체를 생성할 수 있습니다.
CREATE DATABASE LINK	계정의 스키마에 데이터베이스-링크 객체를 생성할 수 있습니다.
CREATE PROCEDURE	계정의 스키마에 프로시저, 함수, 패키지 같은 내장프로그램을 생성할 수 있습니다.
CREATE TRIGGER :	

☞ 오라클 데이터베이스 계정에게 CREATE TABLE, CREATE SEQUENCE, CREATE VIEW, CREATE DATABASE LINK, CREATE PROCEDURE 시스템권한이 부여되면, 해당 계정이, 자신이 소유하는 스키마 객체를 구현할 수 있습니다.

[참고] 스키마(SCHEMA)란?

- 소유권이 동일한 객체들의 집합(Collection)을 의미하며, 데이터베이스 계정이 스키마를 소유합니다.
- 스키마의 이름은 계정의 이름과 동일하며, 스키마 이름은 객체의 이름 앞에 명시하여 사용됩니다.

12-2-2. 객체 권한(OBJECT PRIVILEGES).

- 특정 스키마 객체를 액세스할 수 있는 권한을 객체 권한이라고 합니다. 예를 들면, HR.EMPLOYEES 테이블에 대하여 SELECT문으로 데이터를 조회할 수 있는 권한을 HR.EMPLOYEES 테이블에 대한 SELECT 객체 권한이라고 합니다.

액체 권한의 부여 및 철회는 데이터베이스 관리자 및 객체의 소유자가 수행할 수 있지만, 대부분의 경우에, 데이터베이스 관리자가 객체권한을, 권한이 필요한 데이터베이스 계정에게 부여합니다.

- 대표적인 스키마-액체의 유형에 따른 오라클 데이터베이스에서의 객체 권한은 다음과 같습니다.

액체 유형	액체 권한의 이름
TABLE	SELECT, INSERT, DELETE, UPDATE, ALTER, INDEX, REFERENCES
VIEW	SELECT, INSERT, DELETE, UPDATE
SEQUENCE	SELECT, ALTER
PROCEDURE	EXECUTE

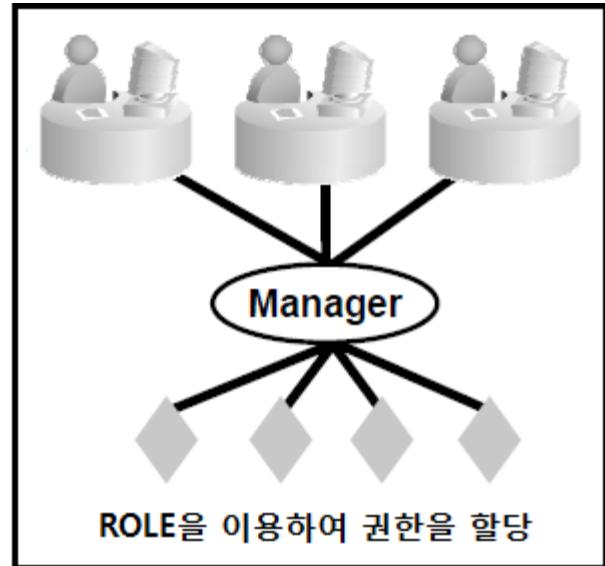
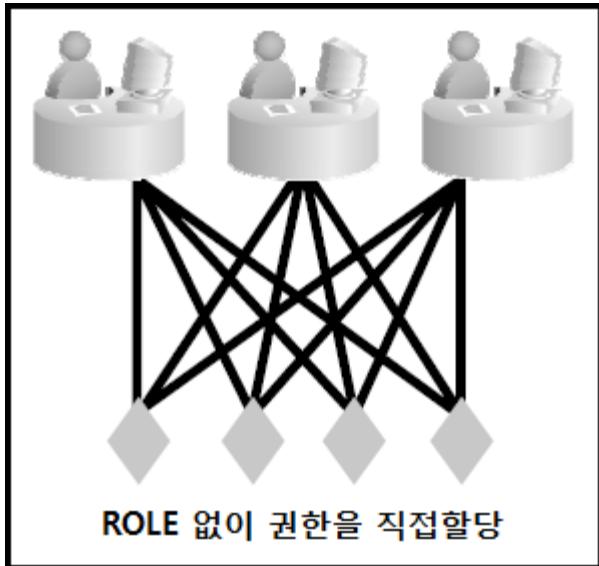
☞ TABLE에 대한 객체 권한 중 UPDATE와 REFERENCES 객체 권한은 특정 컬럼에 국한하여 부여될 수도 있습니다.

☞ 모든 스키마 객체에 대하여 표시한 것이 아닙니다. 일부 객체에 대하여만 표시합니다.

12-3. 룰(ROLE)의 정의 및 룰(ROLE)을 사용하는 목적.

■ 룰(ROLE)은 권한(들)이 부여된, [권한의-그룹](#)이며, 룰을 사용하여 계정에 대한 권한 부여 및 철회 등의 권한 관리를 쉽게 수행할 수 있습니다.

■ 다음의 그림을 이용하여 룰을 사용하는 목적을 이해해 봅니다.



- 그림의 왼쪽처럼 4개의 권한을 직접 3명의 데이터베이스 사용자들에게 부여합니다(복잡해 보이지요).
- 그림의 오른쪽처럼 (1) MANAGER라는 이름의 룰을 생성 한 후, (2) 4개의 권한을 생성한 MANAGER룰에게 부여합니다.
(3) 그리고 이 MANAGER룰을 3명의 데이터베이스 사용자에게 부여합니다(보다 단순해 보입니다).
- 만약 모든 계정들에게 새로운 객체에 대한 객체권한을 부여하거나 또는 이미 부여된 권한을 모든 계정들로부터 철회하려는 경우, 룰의 사용유무에 따라 다음과 같은 작업수행에 차이가 있습니다.
 - 룰을 사용하지 않은 경우에는, 모든 데이터베이스 계정들의 권한 설정을 변경해야 합니다(그림 왼쪽).
 - 룰을 사용하는 경우에는 룰에만 권한의 변경사항을 적용하면, 해당 룰을 사용하는 모든 계정들에게 권한의 변경사항이 간단히 적용될 수 있습니다.
- 이렇게 룰을 사용하면, 사용자들에게 권한의 부여 또는 철회가 쉽게 수행될 수 있습니다.
- 오라클 데이터베이스를 생성하면, DBA 이름의 룰이 생성됩니다. SYSTEM 계정이 DBA룰을 통해 데이터베이스 관리와 관련된 202 개의 시스템 권한을 사용할 수 있습니다. **개발자들이 사용하는 데이터베이스 계정에게는 절대로 DBA 룰을 부여하지 않도록 하십시오.**

[참고] **다이렉트-권한(Direct Privilege)**과 **롤-권한(Role Privilege)**의 차이.

- 데이터베이스 계정이나 룰에게, 권한(Privileges)이 룰을 통하지 않고 직접 부여되면, 이를 다이렉트-권한(Direct Privileges) 이라고 합니다. 데이터베이스 계정이나 룰에게, 룰을 통하지 않고 권한을 직접 부여하거나 철회하면, 변경사항이 이 후에 접속하는 세션 및 이미 데이터베이스에 접속한 사용자 세션에도 바로 적용됩니다.
- 데이터베이스 계정이나 룰에게, 권한이 룰을 통해서 부여되면, 이를 룰-권한(Role-Privileges) 이라고 합니다. 룰-권한(또는 간단히 룰)을 데이터베이스 계정이나 룰에게 부여하거나 철회하면, 변경한 이 후에 접속하는 세션부터 적용되며, 변경 전에 데이터베이스에 접속한 사용자 세션에는 적용되지 않습니다.

☞ 룰을 이용하여 사용자의 권한을 관리하는 보다 다양한 방법에 대해서는 Oracle Database Administrator's Guide 매뉴얼을 참조하십시오.

```
[ ] CONNECT- , RESOURCE- DBA-
      , DBA, CONNECT, RESOURCE
CONNECT CREATE SESSION , RESOURCE
      (CREATE VIEW, CREATE SYNONYM ) . DBA
      가 가 202
      DBA
```

12-4. 오라클 데이터베이스 계정 생성 및 시스템 권한과 객체 권한 부여 실습 및 기본 문법.

[실습] 다음의 실습을 수행하면서, 계정을 생성하고, 권한을 부여하는 방법을 학습합니다.

1> SQL*Developer를 이용하여 SYS 계정으로 접속합니다.

2> [CREATE USER]문을 이용하여 패스워드가 oracle4U로 설정된 HR1 계정과 HR2 계정을 생성하시오.

```
SQL> CREATE USER hr1 IDENTIFIED BY oracle4U;
```

user HR1이(가) 생성되었습니다.

```
SQL> CREATE USER hr2 IDENTIFIED BY oracle4U;
```

user HR2이(가) 생성되었습니다.

```
SELECT tablespace_name, file_id, file_name
FROM dba_data_files
ORDER BY 2 ;
```

```
--  
SELECT tablespace_name, contents  
FROM dba tablespaces  
WHERE contents = 'PERMANENT'  
AND tablespace_name NOT IN ('SYSTEM', 'SYSAUX');
```

```
--  
CREATE user devuser01  
IDENTIFIED BY oracle4U  
DEFAULT TABLESPACE example  
QUOTA unlimited ON example  
QUOTA unlimited ON users  
PASSWORD expire ;
```

■ 데이터베이스 계정을 생성할 때, [CREATE USER]문을 사용합니다.

■ 기본문법

```
CREATE USER 계정이름 IDENTIFIED BY 패스워드;
```

- [CREATE USER]문으로 계정을 생성할 때, 다음의 사항들에 대하여 유의하십시오.
 - 계정이름과 패스워드는 **30Bytes 길이를 초과할 수 없습니다.**
 - 계정이름은 숫자로 시작할 수 없으며, 영문 대/소문자, 숫자, _, \$, # 의 특수 기호만 포함될 수 있습니다.
 - 계정이름에 대하여 영문의 [대/소]문자를 구분하지 않습니다.
 - 10g 버전까지는 패스워드에 대하여 영문의 [대/소문자]를 구분하지 않습니다.
 - **11g 버전부터는 패스워드에 대하여 영문의 [대/소문자]를 구분합니다.**
- 데이터베이스 서버에 대한 계정을 생성할 때, 고려해야 하는 옵션들이 많습니다. 데이터베이스 계정 생성과 관련된 다른 옵션들에 대해서는 Oracle Database Administrator's Guide 매뉴얼을 참고하십시오.

☞ 생성된 계정은 CREATE SESSION 시스템 권한이 부여된 후에 데이터베이스에 접속할 수 있습니다.

데이터베이스 계정이 CREATE SESSION 시스템 권한이 없는 상태에서 데이터베이스 접속을 시도하면 아래처럼 오류가 발생합니다.



3> [GRANT]문을 이용하여 HR1 계정과 HR2 계정에게, 데이터베이스 서버에 접속하여, 테이블, 뷰, 시퀀스, 동의어를 생성할 수 있는 권한과 세션의 설정을 변경할 수 있는 권한을 부여하시오.

```
SQL> GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE SEQUENCE, CREATE SYNONYM,
      ALTER SESSION
      TO HR1 , HR2;
```

GRANT를(를) 성공했습니다.

■ 시스템 권한은 오직 데이터베이스 관리자(DBA)만 데이터베이스 계정 또는 뷰에게 부여할 수 있습니다.

■ 시스템권한을 부여할 때, GRANT 문을 사용합니다.

■ 기본문법

GRANT 시스템권한이름, 시스템권한이름,...

TO 계정이름, 계정이름;

- GRANT 문으로 시스템권한을 부여할 때, 다음의 사항들에 대하여 유의하십시오.
 - GRANT절에 부여할 시스템권한의 이름을 명시하며, 부여할 시스템권한이 여러 개일 경우, 콤마(,)로 구분하여 명시합니다.
 - TO절에 시스템권한이 부여되는 데이터베이스 계정의 이름이나 룰 이름을 명시하며, 시스템권한이 부여되는 계정이나 룰의 이름이 여러 개일 경우, 콤마(,)로 구분해서 명시합니다.

- 테이블 생성하기 위해서는 CREATE TABLE 시스템 권한 이 외에도 저장영역에 대한 권한(QUOTA)가 데이터베이스 계정에게 설정되어 있어야만 합니다.

☞ 데이터베이스의 테이블스페이스 저장공간 및 데이터베이스 계정에 대한 저장공간(테이블스페이스)에 대한 QUOTA 설정과 관련된 자세한 내용은 Oracle Database Administrator's Guide 매뉴얼을 참고하십시오.

4> [REVOKE]문을 이용하여 HR1 계정으로부터 CREATE VIEW, CREATE SYNONYM 시스템 권한을 철회하시오

```
SQL> REVOKE CREATE VIEW, CREATE SYNONYM FROM HR1;
```

REVOKE를(를) 성공했습니다.

- 시스템 권한은 오직 데이터베이스 관리자(DBA)만 데이터베이스 계정 또는 룰로부터 철회할 수 있습니다.

- 시스템권한을 철회할 때, REVOKE 문을 사용합니다.

■ 기본문법

REVOKE 시스템권한이름, 시스템권한이름,...

FROM 계정이름, 계정이름;

○ REVOKE 문으로 시스템권한을 철회할 때, 다음의 사항들에 대하여 유의하십시오.

- REVOKE절에 철회할 시스템 권한이름을 명시하며, 철회할 시스템 권한이 여러 개일 경우, 콤마(,)로 구분하여 명시합니다.
- FROM절에 시스템권한이 철회되는 데이터베이스 계정의 이름(또는 룰 이름)을 명시하며, 시스템권한이 철회되는 계정이나 룰의 이름이 여러 개일 경우, 콤마(,)로 구분해서 명시합니다.

5> 모든 데이터베이스 계정이 CREATE SESSION 권한을 사용할 수 있게끔 설정하시오.

```
SQL> GRANT CREATE SESSION TO PUBLIC ;
```

GRANT를(를) 성공했습니다.

☞ TO절에 데이터베이스 계정 이름이나 룰 이름 대신 PUBLIC 키워드를 명시하면, 부여되는 시스템 권한이나 룰을 데이터베이스의 모든 계정이 사용할 수 있도록 설정됩니다.

6> 모든 데이터베이스 계정에게 사용하도록 설정된 CREATE SESSION 권한에 대한 설정을 해제하시오.

```
SQL> REVOKE CREATE SESSION FROM PUBLIC ;
```

REVOKE를(를) 성공했습니다.

☞ FROM절에 데이터베이스 계정 이름이나 를 이름 대신 PUBLIC 키워드를 명시하면, 해당 시스템 권한(들)이나 를 통하여 모든 데이터베이스 계정들이 사용하도록 설정된 것이 해제됩니다.

7> HR1 계정 및 HR2 계정에게 HR.LOCATIONS 테이블의 데이터에 대하여 조회, 입력, 수정, 삭제, 및 테이블의 정의를 변경하고, 테이블을 참조할 수 있고, 인덱스를 생성할 수 있는 객체 권한을 부여하시오.

```
SQL> GRANT SELECT, INSERT, DELETE, ALTER, INDEX, UPDATE, REFERENCES  
      ON HR.LOCATIONS  
      TO HR1, HR2 ;
```

GRANT를(를) 성공했습니다.

■ 특정 객체에 대한 객체권한은, DBA 또는 객체를 소유한 계정이, 다른 계정이나 를에게 부여할 수 있습니다.

■ 기본문법

```
GRANT 객체권한이름, 객체권한이름,...  
ON 스키마이름.객체이름  
TO 계정이름, 계정이름;
```

○ GRANE 문으로 객체권한을 부여할 때, 다음의 사항들에 대하여 유의하십시오.

- GRANT절에 부여되는 객체권한의 이름을 명시하며, 객체권한이 여러 개일 경우, 콤마(,)로 구분하여 명시합니다.
- **ON절에 해당 객체의 이름을 명시합니다. 단, 오직 하나의 객체 이름만을 명시할 수 있습니다.**
- TO절에 객체권한이 부여되는 데이터베이스 계정의 이름(또는 를 이름)을 명시하며, 객체권한이 부여되는 계정이나 를의 이름이 여러 개일 경우, 콤마(,)로 구분해서 명시합니다.

○ ON절 다음에 객체의 이름을 2개 이상 적으면 다음과 같은 오류가 발생됩니다.

```
SQL> GRANT SELECT, INSERT, DELETE, UPDATE  
      ON HR.EMPLOYEES , HR.DEPARTMENTS  
      TO HR1, HR2 ;
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
GRANT SELECT, INSERT, DELETE, UPDATE
```

```
ON HR.EMPLOYEES , HR.DEPARTMENTS
```

```
TO HR2
```

오류 보고 -

SQL 오류: ORA-00905: 누락된 키워드

00905. 00000 - "missing keyword"

***Cause:**

***Action:**

8> HR2 계정에게, HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼에 대한 외래키를 생성할 수 있는 객체권한과 DEPARTMENT_NAME 컬럼의 데이터만 수정할 수 있는 객체 권한을 부여하시오.

```
SQL> GRANT UPDATE(department_name), REFERENCES(department_id)
  ON HR.DEPARTMENTS
  TO HR2 ;
```

GRANT을(를) 성공했습니다.

☞ TABLE에 대한 객체 권한 중, UPDATE와 REFERENCES 객체권한에 대해서만, 권한이름 뒤에 컬럼이름(들)을 명시하여 특정 컬럼에 국한된 객체 권한을 부여할 수 있습니다. 단, 다른 객체권한의 이름 뒤에는 컬럼이름을 명시할 수 없습니다.

☞ REFERENCES 객체권한 부여 시에, 만약 컬럼이름을 명시하는 경우에, 명시된 컬럼에는 반드시 기본키 또는 UNIQUE 제약조건이 정의되어 있어야 합니다.

9> HR1 계정 및 HR2 계정에게, HR.EMPLOYEES 테이블에 대한 모든 객체 권한을 부여하시오.

```
SQL> GRANT ALL ON HR.EMPLOYEES TO HR1, HR2 ;
```

GRANT을(를) 성공했습니다.

☞ GRANT절에 객체권한 이름 대신 ALL 키워드를 사용하면, 해당 객체에 대한 모든 객체 권한을 부여할 수 있습니다.

☞ 단, 테이블이나 뷰에 대해서는 ALL 옵션을 이용하여, 객체 권한을 부여하지 않도록 주의하십시오. 즉, 테이블 또는 뷰가 아닌 다른 유형의 객체에 대해서 ALL 옵션을 사용하십시오.

10> HR.DEPARTMENTS 테이블의 데이터를 SELECT 할 수 있는 객체 권한을 모든 데이터베이스 계정이 사용하도록 설정하시오.

```
SQL> GRANT SELECT ON HR.DEPARTMENTS TO PUBLIC ;
```

GRANT을(를) 성공했습니다.

☞ TO절에 계정이름이나 뷰-이름 대신 PUBLIC 키워드를 사용하면, 해당 객체 권한이 모든 계정에 의하여 사용될 수 있도록 설정됩니다.

11> HR1 계정에게 부여된 HR.LOCATIONS 테이블에 대한 SELECT 객체권한을 철회하시오.

```
SQL> REVOKE SELECT ON HR.LOCATIONS FROM HR1 ;
```

REVOKE을(를) 성공했습니다.

■ 객체권한은, DBA 또는 객체를 소유한 계정이, 다른 계정이나 뷰로부터, REVOKE 문을 사용하여 철회할 수 있습니다.

■ 기본문법

```
REVOKE 객체권한이름, 객체권한이름, ...
ON 스키마이름.객체이름
FROM 계정이름, 계정이름;
```

○ REVOKE 문으로 객체권한을 철회할 때, 다음의 사항들에 대하여 유의하십시오.

- REVOKE절에 철회할 객체권한의 이름을 명시하며, 철회할 객체권한이 여러 개일 경우, 콤마(,)로 구분해서 명시합니다.
- ON절에 해당 객체의 이름을 명시합니다. 단, 오직 하나의 객체 이름만을 명시할 수 있습니다.
- FROM절에 객체권한이 철회되는 데이터베이스 계정의 이름(또는 룰 이름)을 명시하며, 객체권한이 철회되는 계정이나 룰의 이름이 여러 개일 경우, 콤마(.)로 구분해서 명시합니다.

12> HR1 계정에게 부여된 HR.EMPLOYEES 테이블에 대한 모든 객체권한을 철회하시오.

```
SQL> REVOKE ALL ON HR.EMPLOYEES FROM HR1 ;
```

REVOKE를(를) 성공했습니다.

☞ REVOKE절에 객체권한 이름 대신 ALL 키워드를 사용하면, 계정이나 룰에게 부여된 모든 객체 권한을 철회할 수 있습니다.

13> 모든 데이터베이스 계정이 사용하도록 설정된 HR.DEPARTMENTS 테이블의 SELECT 객체 권한을 설정해제 하시오.

```
SQL> REVOKE SELECT ON HR.DEPARTMENTS FROM PUBLIC ;
```

REVOKE를(를) 성공했습니다.

☞ FROM절에 계정이름이나 룰-이름 대신 PUBLIC 키워드를 명시하면, 모든 데이터베이스 계정이 사용하도록 설정된 객체권한이 해제됩니다.

12-5. 객체 권한의 부여와 철회 실습: REFERENCES 객체 권한의 부여 및 철회 실습 및 기본문법.

[실습] 테이블의 REFERENCE 객체권한의 기능과 REFERENCE 객체 권한을 부여 철회하는 방법을 학습합니다.

1> SQL*Developer를 이용하여 SYS 계정으로 접속합니다. 이미 SQL*Developer를 이용하여 SYS계정으로 접속된 경우에는, 기존의 SYS 계정의 접속 세션을 그대로 이용합니다.

2> HR1 계정 소유의 테이블이 생성될 수 있도록 USERS 저장공간(테이블스페이스)에 대한 QUOTA를 다음처럼 설정합니다.

```
SQL> ALTER USER HR1
      QUOTA UNLIMITED ON USERS; --USERS 저장공간에 구성된 용량을 제한 없이 사용하도록 설정됩니다.

user HR1이(가) 변경되었습니다.
```

☞ 이 작업은 디풀트로 데이터베이스 관리자(SYS 또는 SYSTEM 계정)로 접속한 세션에서만 수행될 수 있습니다.

☞ 테이블스페이스(TABLESPACE)와 QUOTA에 대해서는 Oracle Database Administrator's Guide 매뉴얼을 참고하십시오.

3> SQL*Developer로 SYS 계정으로 접속한 세션에서, HR1.EMPS1 테이블을 생성하기 위하여 다음의 [CREATE TABLE]문을 실행합니다. 오류가 발생됩니다.

```
SQL> CREATE TABLE hr1.emps1 (
      e_id          NUMBER(6),
      e_name        VARCHAR2(30),
      e_salary       NUMBER(8),
      department_id  NUMBER(4),
      CONSTRAINT fk_emps_departments FOREIGN KEY(department_id)
      REFERENCES hr.departments(department_id) ;
      REFERENCES hr.departments(department_id) )
      *
ERROR at line 7:
ORA-00942: table or view does not exist
```

☞ 앞의 SQL문을 실행하면, HR1 계정은 HR.DEPARTMENTS테이블에 대한 REFERENCES 객체 권한이 없기 때문에, 외래키 제약조건을 정의할 때, HR.DEPARTMENTS 객체가 액세스되지 못하므로 오류가 발생됩니다.

4> SQL*Developer로 SYS 계정으로 접속한 세션에서, HR.DEPARTMENTS 테이블의 기본키 제약조건에 대한 REFERENCES 객체 권한을 HR1 계정에게 부여합니다.

```
SQL> GRANT REFERENCES ON HR.DEPARTMENTS TO HR1 ;

GRANT을(를) 성공했습니다.
```

☞ 테이블에 대한 REFERENCES 객체권한을 부여할 때, 컬럼이름을 명시하지 않은 경우, 해당 테이블의 기본키 제약조건이 정의된 컬럼에 대한 객체권한이 자동으로 부여됩니다.

5> SQL*Developer로 SYS 계정으로 접속한 세션에서, HR1.EMPS1 테이블을 생성하기 위하여 다음의 [CREATE TABLE]문을 다시 실행합니다.

```
SQL> CREATE TABLE hr1.emps1 (
    e_id          NUMBER(6),
    e_name        VARCHAR2(30),
    e_salary      NUMBER(8),
    department_id NUMBER(4),
    CONSTRAINT fk_emps_departments FOREIGN KEY(department_id)
    REFERENCES hr.departments(department_id) );
```

Table created. —REFERENCES 객체권한 부여 후에는 테이블이 정상적으로 생성됩니다.

6> SQL*Developer로 SYS 계정으로 접속한 세션에서, HR1 계정에게 부여된 DEPARTMENTS 테이블의 기본키에 대한 REFERENCES 객체 권한을 철회하기 위하여, 다음의 REVOKE 문을 실행합니다. 오류가 발생됩니다

```
SQL> REVOKE REFERENCES ON HR.DEPARTMENTS FROM HR1;
명령의 1 행에서 시작하는 중 오류 발생 -
REVOKE REFERENCES ON HR.DEPARTMENTS FROM HR1
오류 보고 -
SQL 오류: ORA-01981: 현 권한취소를 수행하려면 CASCADE CONSTRAINTS가 지정되어야 합니다
01981. 00000 - "CASCADE CONSTRAINTS must be specified to perform this revoke"
*Cause: During this revoke some FOREIGN KEY constraints will be removed.
        In order to perform this automatically, CASCADE CONSTRAINTS must
        be specified.
*Action: Remove the constraints or specify CASCADE CONSTRAINTS.
```

☞ HR1.EMPS1.DEPARTMENT_ID 컬럼에 설정된 외래키 제약조건이, HR1 계정에게 부여된 HR.DEPARTMENTS.DEPARTMENT_ID 컬럼에 대한 REFERENCES 객체 권한을 기반하여 정의되어 있기 때문에, REFERENCES 객체권한을 철회할 수 없습니다.

,	REFERENCES	(FOREIGN KEY)
,		

8> SQL*Developer를 이용하여 데이터베이스 서버에 SYS 계정으로 접속한 세션에서, HR1 계정에게 부여된 HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼에 대한 REFERENCES 객체 권한을 CASCADE CONSTRAINTS 옵션을 이용하여 철회하시오.

```
SQL> REVOKE REFERENCES ON HR.DEPARTMENTS FROM HR1
      CASCADE CONSTRAINTS;
      —CASCADE CONSTRAINTS 옵션은 계정에게 부여된 REFERENCES 객체권한을
      --철회 시 사용되며, 다른 객체권한에 대해서는 사용되지 않습니다.
REVOKE를(를) 성공했습니다.
```

☞ 계정에게 부여된 테이블의 REFERENCES 객체 권한을 철회하는 [REVOKE]문에 CASCADE CONSTRAINTS 옵션을 명시하면, 먼저, 부여된 REFERENCES 객체권한을 이용하여 구성된 다른 테이블(들)의 외래키 제약조건들을 모두 삭제한 후, 계정에게 부여된 REFERENCES 객체권한이 철회됩니다.

12-6. 객체 권한의 부여와 철회 실습: [WITH GRANT OPTION] 옵션의 사용.

- [WITH GRANT OPTION] 옵션은 시스템 권한 또는 룰을 부여할 때는 사용될 수 없으며, **액체권한을 부여할 때만 사용**할 수 있습니다.

[]

WITH ADMIN OPTION

- 객체권한을 부여할 때, [WITH GRANT OPTION] 옵션을 명시하면, 해당 객체 권한을 부여 받은 계정이, DBA나 해당 객체를 소유한 계정이 아니더라도, 다른 데이터베이스의 계정에게 해당 객체 권한을 부여할 수 있게 됩니다.

- 계정으로부터, [WITH GRANT OPTION] 옵션을 명시하여 부여된 객체권한을 철회하면, 그 계정에 의해서 해당 객체권한을 부여 받은 모든 다른 계정들로부터도 해당 객체 권한이 동시에 철회됩니다.

[실습] 다음의 실습을 수행하여, [WITH GRANT OPTION] 옵션의 기능을 확인합니다.

1> SQL*Developer를 이용하여 SYS 계정으로 접속합니다. 이미 SQL*Developer를 이용하여 SYS계정으로 접속된 경우에는, 기존의 SYS 계정의 접속 세션을 그대로 이용합니다.

2> HR1 계정에게 HR.DEPARTMENTS 상의 SELECT 객체 권한을 [WITH GRANT OPTION-절]을 명시하여 부여합니다.

```
SQL> GRANT SELECT ON HR.DEPARTMENTS TO HR1 WITH GRANT OPTION;
```

GRANT을(를) 성공했습니다.

☞ HR1 계정도 HR.DEPARTMENTS 테이블에 대한 SELECT 객체권한을 사용할 수 있고, 또한, [WITH GRANT OPTION] 옵션을 명시하여 권한이 부여되었으므로, HR1 계정이 HR.DEPARTMENTS 상의 SELECT 객체권한을 다른 계정에게 부여할 수 있습니다.

3> 관리자 권한으로 실행된 명령프롬프트에서, SQL*Plus를 이용하여 HR1 계정으로 ORCL 데이터베이스의 인스턴스에 로컬접속합니다.

```
C:\WINDOWS\system32>set ORACLE_SID=orcl
```

```
C:\WINDOWS\system32>sqlplus hr1/oracle4U
```

```
SQL*Plus: Release 11.2.0.1.0 Production on 화 7월 11 00:30:59 2017
```

```
Copyright (c) 1982, 2010, Oracle. All rights reserved.
```

다음에 접속됨:

```
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

4> 터미널의 SQL*Plus로 데이터베이스에 HR1 계정으로 접속한 세션에서, HR2 계정에게 HR.DEPARTMENTS 상의 SELECT 객체권한을 부여합니다.

```
SQL> GRANT SELECT ON HR.DEPARTMENTS TO HR2 ;
```

Grant succeeded.

☞ HR1 계정은 HR.DEPARTMENTS 객체를 소유한 계정이나 DBA가 아니지만, HR.DEPARTMENTS 객체의 SELECT 객체 권한을 [WITH GRANT OPTION] 옵션으로 부여 받았으므로, HR2 계정에게 부여할 수 있습니다.

5> HR1 계정으로 접속한 SQL*Plus의 명령프롬프트에서, CONN 명령어를 이용하여 HR2 계정으로 접속한 후, HR.DEPARTMENTS 에 대하여 SELECT를 수행해 봅니다.

```
SQL> conn HR2/oracle4U
```

연결되었습니다.

```
SQL> SELECT * FROM HR.DEPARTMENTS WHERE DEPARTMENT_ID=10 ;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700

☞ HR1 계정으로부터 부여 받은 권한을 이용하여 HR2 계정도 HR.DEPARTMENTS 테이블에 대하여 조회할 수 있습니다.

6> SQL*Developer를 이용하여 데이터베이스 서버에 SYS 계정으로 접속한 세션에서, HR1 계정에게 부여된 HR.DEPARTMENTS 테이블에 대한 SELECT 객체권한을 철회합니다.

```
SQL> REVOKE SELECT ON HR.DEPARTMENTS FROM HR1;
```

REVOKE를(를) 성공했습니다.

☞ HR1 계정으로부터 [WITH GRANT OPTION] 옵션으로 부여된 HR.DEPARTMENTS 테이블에 대한 SELECT 객체권한을 철회하면, HR1 계정에 의하여 HR.DEPARTMENTS 테이블에 대한 SELECT 객체 권한을 부여 받은 모든 다른 계정(HR2) 들도 해당 객체 권한이 철회됩니다.

7> HR2 계정으로 접속한 SQL*Plus의 명령프롬프트에서, HR.DEPARTMENTS 테이블에 대하여 SELECT문을 수행해 봅니다.

```
SQL> SELECT * FROM HR.DEPARTMENTS WHERE DEPARTMENT_ID=10 ;
```

```
FROM HR.DEPARTMENTS
```

```
*
```

```
ERROR at line 1:
```

```
ORA-01031: insufficient privileges
```

☞ HR1 계정뿐만 아니라, HR2 계정도 HR.DEPARTMENTS 테이블에 대한 SELECT 객체 권한이 같이 철회되었으므로 위와 같이 권한 부족 오류가 발생됩니다.

12-7. ROLE을 사용한 권한 관리 실습.

1> SQL*Developer를 이용하여 SYS 계정으로 접속합니다. 이미 SQL*Developer를 이용하여 SYS계정으로 접속된 경우에는, 기존의 SYS 계정의 접속 세션을 그대로 이용합니다.

2> 룰을 통한 권한 관리 실습을 위하여 HR1 계정에게 부여된 CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, ALTER SESSION 시스템 권한 및 HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블 상의 모든 객체권한을 철회합니다.

```
SQL> REVOKE CREATE SESSION, CREATE TABLE, CREATE SEQUENCE, ALTER SESSION FROM HR1;
```

REVOKE를(를) 성공했습니다.

```
SQL> REVOKE ALL ON HR.DEPARTMENTS FROM HR1 ;
```

REVOKE를(를) 성공했습니다.

```
SQL> REVOKE ALL ON HR.EMPLOYEES FROM HR1 ;
```

REVOKE를(를) 성공했습니다.

☞ CREATE SESSION 권한도 철회하였으므로, HR1 계정으로는 데이터베이스 서버에 접속할 수 없습니다.

3> EMPROLE1 이름의 룰을 생성합니다.

```
SQL> CREATE ROLE EMROLE1 ; — 암호 없이 룰을 생성합니다.
```

role EMROLE1이(가) 생성되었습니다.

■ CREATE ROLE 문을 이용하여 데이터베이스에 룰(ROLE)을 생성하며, **CREATE ROLE 시스템 권한**을 부여 받으면 룰을 생성할 수 있지만, 일반적으로 룰은 DBA가 생성합니다.

■ 기본문법

CREATE ROLE 룰이름

IDENTIFIED BY 패스워드; —룰을 생성할 때, IDENTIFIED BY 절을 이용하여 패스워드를 설정할 수 있습니다

4> 생성된 EMROLE1-룰에 CREATE SESSION, CREATE TABLE 시스템 권한을 부여합니다.

```
SQL> GRANT CREATE SESSION, CREATE TABLE TO emrole1 ;
```

GRANT를(를) 성공했습니다.

5> 생성된 EMPROLE1-룰에 HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블 상의 SELECT 객체권한을 부여합니다.

```
SQL> GRANT SELECT ON HR.EMPLOYEES TO emprole1;
```

GRANT를(를) 성공했습니다.

```
SQL> GRANT SELECT ON HR.DEPARTMENTS TO emprole1;
```

GRANT를(를) 성공했습니다.

☞ 룰에 시스템 권한과 객체 권한을 부여 및 철회하는 방법은 데이터베이스 계정에 대하여 권한을 부여 및 철회하는 방법과 동일합니다.

☞ 룰에 객체권한을 부여하는 것은 DBA 및 해당 객체를 소유한 계정도 수행할 수 있습니다.

☞ 위의 예제처럼, 다른 룰을 통하지 않고 룰에 권한이 직접 부여되면(다이렉트-권한), 데이터베이스 계정이나 다른 룰에 이 룰이 부여된 경우, 변경사항이, 이미 데이터베이스에 접속한 사용자 세션에도 바로 적용됩니다.

6> EMPROLE1-룰을 SCOTT, HR1, HR2 계정에게 부여합니다.

```
SQL> GRANT EMROLE1 TO HR1, HR2, RESOURCE;
```

GRANT를(를) 성공했습니다.

☞ 룰을 데이터베이스 계정이나 다른 룰에게 부여하는 문법은, 계정에게 시스템 권한을 부여하는 문법과 동일합니다.

☞ 시스템 권한 및 룰은 DBA만 다른 데이터베이스 계정에게 부여할 수 있습니다.

☞ 계정에게 룰을 부여하면, 해당 변경사항은, 변경 후에 접속한 세션부터 적용됩니다.

7> 관리자 권한으로 실행된 명령프롬프트에서, SQL*Plus를 이용하여 HR1 계정으로 orcl 데이터베이스의 인스턴스에 로컬 접속합니다.

```
C:\WINDOWS\system32>sqlplus hr1/oracle4U
```

SQL*Plus: Release 11.2.0.1.0 Production on 화 7월 11 00:30:59 2017

Copyright (c) 1982, 2010, Oracle. All rights reserved.

다음에 접속됨:

Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

☞ EMPROLE1에 부여된 CREATE SESSION 권한이 사용되어, HR1 계정이 데이터베이스 서버에 접속할 수 있습니다.

8> HR1 계정으로 접속한 세션에서 MYDEPT 이름으로 HR.DEPARTMENTS에 대한 SYNONYM(동의어)을 생성해 봅니다.

```
SQL> CREATE SYNONYM mydept FOR HR.DEPARTMENTS ;
CREATE SYNONYM mydept FOR HR.DEPARTMENTS
*
ERROR at line 1:
ORA-01031: insufficient privileges.
```

☞ CREATE SYNONYM 시스템 권한이 없기 때문에 SYNONYM(동의어)를 생성할 수 없어서 오류가 발생됩니다.

9> SYS 계정으로 접속한 SQL*Developer 세션에서, CREATE SYNONYM 시스템 권한을 EMPROLE1 를에 부여합니다.

```
SQL> GRANT CREATE SYNONYM TO emprole1 ;

GRANT을(를) 성공했습니다.
```

☞ 를에게 권한을 부여 및 철회하면, 를이 권한을 부여 및 철회하기 전에 오라클 데이터베이스 서버에 접속한 세션에도 변경사항이 적용됩니다.

10> 터미널에서 HR1 계정으로 접속한 SQL*Plus 세션에서, MYDEPT 이름으로 HR.DEPARTMENTS에 대한 동의어를 생성합니다.

```
SQL> CREATE SYNONYM mydept FOR HR.DEPARTMENTS ;

synonym MYDEPT1이(가) 생성되었습니다.
```

☞ 시스템 권한이 부여된 후에는 동의어가 정상적으로 생성됩니다.

11> SYS 계정으로 접속한 SQL*Developer 세션에서, HR1 계정에게 부여된 EMPROLE1-를을 철회합니다

```
SQL> REVOKE EMROLE1 FROM HR1 ;

REVOKE를(를) 성공했습니다.
```

☞ 시스템 권한 및 를은 DBA만 다른 데이터베이스 계정으로부터 철회할 수 있으며, 를을 철회하면, 해당 변경사항은, 변경 후에 접속한 세션부터 적용됩니다.

12> 터미널에서 HR1 계정으로 접속한 SQL*Plus 세션에서, HR1 계정으로 아래의 CREATE TABLE 구문을 실행해 봅니다.

```
SQL> CREATE TABLE HR1.TEST_DEPT1
      AS SELECT * FROM HR.DEPARTMENTS ;

table HR1.DEPT2110이(가) 생성되었습니다.
```

☞ EMROLE1-를을 HR1 계정으로부터 철회했지만, 를을 철회하기 전에 접속된 세션에서는 여전히 EMPROLE-를에 부여된 권한을 사용할 수 있습니다. 데이터베이스 계정에게 를 자체를 부여하거나 철회하면, 계정에 대한 를의 부여 및 철회

사항은, 변경 작업 후에 접속하는 세션부터 적용되며, 기존 세션에 대해서는 적용되지 않기 때문입니다.

12> SQL*Plus로 HR1 계정으로 접속한 명령프롬프트에서, CONNECT 명령어로 다시 접속을 시도해 봅니다.

```
SQL> CONN HR1/oracle4U
ERROR:
ORA-01045: 사용자 HR1는 CREATE SESSION 권한을 가지고 있지 않음; 로그온이
거절되었습니다
```

```
Warning: You are no longer connected to ORACLE.
```

☞ 다시 접속을 시도 하면, EMPROLE1-를이 철회된 변경사항이 적용되어 접속이 되지 않습니다.

☞ 앞의 실습(실습7)에서 HR1 계정은 EMPROLE1-를에 부여되었던 CREATE SESSION 권한을 사용하여 접속했었지만, EMPROLE1-를이 철회된 후에는 CREATE SESSION 권한을 사용할 수 없으므로 접속이 수행되지 못합니다.

12-8. 데이터베이스 계정의 암호 변경 실습.

1> 관리자 권한으로 실행된 명령프롬프트에서, SQL*Plus를 이용하여 HR2 계정으로 orcl 데이터베이스의 인스턴스에 로컬 접속합니다.

```
C:\WINDOWS\system32>sqlplus hr2/oracle4U

SQL*Plus: Release 11.2.0.1.0 Production on 화 7월 11 00:30:59 2017

Copyright (c) 1982, 2010, Oracle. All rights reserved.
```

다음에 접속됨:

```
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

2> SQL*Plus로 HR1 계정으로 접속한 명령프롬프트에서, HR2 계정의 암호를 oracle5U로 변경하시오.

```
SQL> ALTER USER HR2 IDENTIFIED BY oracle5U ;
[      ] ( , HR2)
User altered.
```

☞ 데이터베이스 사용자는 데이터베이스 서버에 접속된 상태에서, [ALTER USER]문을 이용하여, 자신의 계정에 대한 패스워드를 변경할 수 있습니다.

3> SQL*Developer를 이용하여 SYS 계정으로 접속합니다. 이미 SQL*Developer를 이용하여 SYS계정으로 접속된 경우에는, 기존의 SYS 계정의 접속 세션을 그대로 이용합니다.

4> SYS 계정으로 접속한 SQL*Developer 세션에서, HR2 계정의 패스워드를 다시 oracle4U로 변경하시오.

```
SQL> ALTER USER HR2 IDENTIFIED BY oracle4U ;
```

user HR01(가) 변경되었습니다.

☞ DBA는 [ALTER USER]문을 이용하여, 데이터베이스의 다른 사용자의 패스워드를 변경할 수 있습니다.

☞ SQL문장으로 데이터베이스에 저장된 계정의 암호를 변경하면, SQL*Developer의 기존 접속 항목에 설정된 암호를 동일하게 변경해주어야 합니다.

[참고] [DROP USER]문을 이용한 데이터베이스 계정 삭제.

- 데이터베이스 계정이 소유한 스키마 객체가 없는 경우, 데이터베이스 관리자는 [DROP USER 계정이름:]문을 이용하여 데이터베이스 계정을 삭제할 수 있습니다.
- 데이터베이스 계정이 스키마 객체를 소유한 경우, 데이터베이스 관리자는 [DROP USER 계정이름 CASCADE:]문을 이용하여 데이터베이스 계정을 삭제할 수 있습니다. CASCADE 옵션을 명시해서 계정을 삭제하면, 삭제되는 계정이 소유한 모든 객체가 먼저 삭제된 후, 계정이 삭제됩니다.
- 소유한 객체가 있는 계정을 CASCADE 옵션 없이 삭제하려고 시도하면, 다음의 오류가 발생됩니다.

명령의 1 행에서 시작하는 중 오류 발생 -

DROP USER hr1

오류 보고 -

SQL 오류: ORA-01922: 'HR1'(을)를 삭제하려면 CASCADE를 지정하여야 합니다

01922. 00000 - "CASCADE must be specified to drop '%s'"

*Cause: Cascade is required to remove this user from the system. The user own's object which will need to be dropped.

*Action: Specify cascade.

☞ 단, 데이터베이스 계정을 삭제하는 것은 매우 위험한 작업이므로, 가급적, 데이터베이스 계정을 삭제하지 않는 것을 권장합니다. 데이터베이스 계정을 삭제해야만 하는 경우라면, 스키마-레벨에서 데이터를 익스포트 받은 후에 계정을 삭제하는 것을 고려하십시오.

☞ 본 단원에서 생성한 HR1, HR2 계정을 삭제하지 마십시오. 이 후 단원의 실습에서 필요합니다.

13 스키마 객체 관리: ALTER TABLE 문장 사용.**◆ 학습 목표.**

- [ALTER TABLE]문을 이용하여 테이블에 대하여 다음과 같은 작업을 수행하는 방법을 학습합니다.

- 테이블에 새로운 **컬럼을 추가**.
- 테이블의 기존 **컬럼을 수정**.
- 테이블의 기존 **컬럼을 삭제**.
- 테이블의 기존 컬럼을 UNUSED로 설정 후, 이를 UNUSED-컬럼을 삭제.
- **테이블 및 컬럼의 이름 변경**.
- 테이블에 **제약조건을 추가**.
- 테이블에 정의된 제약조건을 비활성화 및 활성화.
- 테이블에 **제약조건을 삭제**.
- 제약조건의 이름 변경.
- 제약조건의 지연(DEFERRED) 기능 소개 및 사용방법.
- CASCADE CONSTRAINTS 절

- **CREATE TABLE** 문에서 **USING INDEX 절**을 이용한 인덱스 생성.

- 함수 기반 인덱스 **인덱스** 생성

- 임시 테이블(Temporary Table)을 생성하고, 사용하는 방법을 학습합니다.

- **FLASHBACK DROP** 수행.

13-1. 테이블의 구조 및 설정 변경.

- 테이블을 생성한 후에, 필요한 경우, 새로운 컬럼을 추가하거나, 기존 컬럼의 정의를 변경 또는 제거하여, 테이블 구조를 변경해야 할 수 있습니다. 이러한 작업 시에 ALTER TABLE 문을 사용합니다.
- 필요한 경우, ALTER TABLE 문을 이용하여, 테이블의 기존 컬럼(들)을 UNUSED로 설정하여, 컬럼이 삭제된 것과 유사하게, 컬럼의 데이터에 대한 액세스를 막을 수 있습니다.
- ALTER TABLE 문을 이용하여, 테이블의 이름을 변경(10gNF)하거나 컬럼의 이름을 변경(9iNF)할 수 있습니다.
- ALTER TABLE 문은 DDL문입니다. 따라서, 실행과 동시에 커밋 되며, 따라서 룰백 될 수 없습니다.
- 다음의 실습을 수행하여, 컬럼 추가, 수정, 삭제를 포함한, ALTER TABLE문의 사용방법을 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 실습에 사용할, HR.EMP_APP_TEST2 테이블을 하위질의를 이용하여 다음과 같이 생성합니다.

```
SQL> CREATE TABLE HR.EMP_APP_TEST2
      AS SELECT EMPLOYEE_ID AS "EMPNO", LAST_NAME, SALARY AS "SAL", SALARY*12 AS "ANNSAL",
                 COMMISSION_PCT AS "COMM RATE", DEPARTMENT_ID AS "DEPTNO"
              FROM HR.EMPLOYEES
             WHERE DEPARTMENT_ID=80;
```

Table HR.EMP_APP_TEST2이(가) 생성되었습니다.

3> 생성된 HR.EMP_APP_TEST2 테이블의 데이터를 확인합니다.

```
SQL> SELECT * FROM HR.EMP_APP_TEST2 ;
```

EMPNO	LAST_NAME	SAL	ANNSAL	COMM RATE	DEPTNO
145	Russell	14000	168000	.4	80
146	Partners	13500	162000	.3	80
147	Errazuriz	12000	144000	.3	80
148	Cambrault	11000	132000	.3	80
149	Zlotkey	10500	126000	.2	80
...					

34개의 행이 선택됨

4> HR.EMP_APP_TEST2 테이블에, 다음의 컬럼을 추가하시오.

- 이름이 JOB_ID 이고, 데이터유형이 VARCHAR2(9) 인 컬럼.
 - 이름이 HIREDATE 이고, 데이터유형이 DATE 인 컬럼 .
 - DEFAULT 값은 JOB_ID 컬럼에만 '-' 값을 지정하고, HIREDATE 컬럼에는 DEFAULT 값을 지정하지 않음.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2  
      ADD (JOB_ID VARCHAR2(9) DEFAULT '-' , HIREDATE DATE );
```

Table HR.EMP_APP_TEST201(가) 변경되었습니다.

■ 기본문법: 컬럼추가

```
ALTER TABLE 스키마이름.테이블이름  
ADD (컬럼이름 데이터유형, -- DEFAULT 값을 설정하지 않고, 컬럼추가(NULL상태로 추가됨).  
      컬럼이름 데이터유형 DEFAULT 값표현식....); -- DEFAULT 값을 설정하면서 컬럼추가.
```

■ 테이블에 컬럼을 추가할 때. 다음의 사항을 고려하십시오.

- 추가된 컬럼은 항상 테이블의 맨 마지막에 추가됩니다.
 - 컬럼 추가 시에 DEFAULT 옵션을 지정하지 않으면, 이미 저장된 행에 대하여, 추가된 컬럼은 NULL 상태입니다.
 - 컬럼 추가 시에 DEFAULT 옵션을 지정하면, 이미 저장된 행들에 대하여, 추가된 컬럼의 값을, DEFAULT로 설정한 값으로, 모든 행에 대하여 수정하는 작업이 발생됩니다.
 - 테이블에 컬럼을 추가한 후에는, 테이블에 행을 입력할 때 추가된 컬럼에 대하여 고려해야 합니다.
 - DEFAULT 옵션이 없이 테이블에 추가된 컬럼에 대하여,
적절한 값으로 NULL 상태를 수정해 주는 것을 고려해야 합니다.
 - 테이블에 이미 저장된 행이 있는 경우, 컬럼 추가 시에 NOT NULL 제약조건을 같이 지정하려면, 반드시 DEFAULT 옵션으로 기본 값을 지정해야만 합니다. 테이블에 저장된 행이 하나도 없는 경우에는, DEFAULT 옵션으로 기본값을 지정하지 않고도 NOT NULL 제약조건을 명시하여, 컬럼을 추가할 수 있습니다.

5> JOB_ID 컬럼 추가 후, HR.EMP_APP_TEST2 테이블의 데이터를 확인합니다.

```
SQL> SELECT * FROM HR.EMP_APP_TEST2;
```

EMPNO	LAST_NAME	SAL	ANNSAL	COMMRATE	DEPTNO	JOB_ID	HIREDATE	-- DEFAULT 값이 지정된 -- JOB_ID 컬럼에 '-' 가 -- 표시됩니다.
145	Russell	14000	168000	.4	80	-		
146	Par tners	13500	162000	.3	80	-		
147	Errazuriz	12000	144000	.3	80	-		-- DEFAULT 값을 지정하지
148	Cambrault	11000	132000	.3	80	-		-- 않은 HIREDATE컬럼은
149	Zlotkey	10500	126000	.2	80	-		-- NULL 상태입니다.

34개의 행이 선택됨

☞ 새로이 추가된 컬럼은 테이블의 맨 마지막에, ALTER TABLE .. ADD 문에서 명시한 순서대로 추가됩니다.

6> HR.EMP_APP_TEST2 테이블에, 다음의 컬럼을 추가하시오(오류가 발생되는 것을 확인합니다).

- 이름이 ADDR이고, 데이터유형이 VARCHAR2(1000)인 컬럼.
- DEFAULT 값은 지정하지 않음.
- NOT NULL 제약조건을 명시함.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2  
ADD ( ADDR VARCHAR2(1000) NOT NULL );
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.EMP_APP_TEST2  
ADD ( ADDR VARCHAR2(1000) NOT NULL )
```

오류 보고 -

SQL 오류: ORA-01758: 테이블은 필수 열을 추가하기 위해 (NOT NULL) 비어 있어야 합니다.

01758. 00000 - "table must be empty to add mandatory (NOT NULL) column"

*Cause:

*Action:

☞ 테이블에 DEFAULT 값 설정 없이 컬럼을 추가하면, 이미 저장된 행에 대하여 NULL 상태로 추가됩니다.

따라서, 추가되는 컬럼에 NOT NULL 제약조건을 같이 명시하면, 위와 같은 오류가 발생됩니다.

☞ 컬럼을 추가할 때 위의 오류를 피하려면, NOT NULL 제약조건을 명시하지 않고 컬럼을 추가하거나,

만약, NOT NULL 제약조건을 명시해야 하는 경우라면, 테이블에 행이 전혀 없거나 또는

적절한 DEFAULT 값을 명시하여 컬럼을 추가해야 합니다.

7> HR.EMP_APP_TEST2 테이블에 대하여, 다음의 조건대로 **컬럼의 정의를 변경**하시오.

- LAST_NAME 컬럼의 최대길이를 30Byte로 늘림.
- HIREDATE 컬럼에 SYSDATE 함수를 DEFAULT 값으로 설정.
- JOB_ID 컬럼에 설정된 DEFAULT 값을 해제.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2
  MODIFY (LAST_NAME VARCHAR2(30),
          HIREDATE DEFAULT SYSDATE, -- DEFAULT 값 설정만 변경 시에는,
          JOB_ID DEFAULT '');      -- 데이터유형을 명시할 필요가 없습니다.
```

Table HR.EMP_APP_TEST2이(가) 변경되었습니다.

- ☞ DEFAULT 값 설정 시에, 숫자 상수값은 그대로 명시하면 되고, 문자 상수값은 작은 따옴표로 감싸주어야 하며, 날짜 상수값은 TO_DATE() 또는 TO_TIMESTAMP() 함수로 적절하게 데이터유형을 변환시켜주어야 합니다.
만약, DEFAULT 값으로 함수를 설정할 때는 해당 함수를 작은 따옴표로 감싸지 않습니다.
- ☞ 데이터유형에 상관없이, 작은 따옴표(')를 연달아 2번 명시하여 DEFAULT 값을 지정하면, 설정된 DEFAULT 값이 해제됩니다.

■ 기본문법: 컬럼수정

```
ALTER TABLE 스키마이름.테이블이름
MODIFY (컬럼이름 DEFAULT 값표현식, -- 컬럼의 DEFAULT 값 설정 및 변경
         컬럼이름 데이터유형(길이), -- 데이터의 최대 길이 변경
         컬럼이름 데이터유형, ...); -- 데이터유형 변환
```

■ 테이블에 있는 기존 컬럼의 정의를 수정할 때, 다음의 사항을 고려하십시오.

- 숫자 또는 문자 컬럼의 최대길이를 늘릴 수 있습니다.
- 컬럼에 DEFAULT 값을 지정하거나 변경할 수 있습니다.
 - ☞ 컬럼의 DEFAULT 값을 변경하면, 테이블에 이미 저장된 행에 대해서는 적용되지 않으며, 이 후에 입력되는 행부터 적용됩니다.
 - ☞ 기본키 또는 UNIQUE 제약조건이 적용된 컬럼에는 DEFAULT 값을 설정할 수 없습니다.
- 다음과 같은 경우에는 컬럼의 최대길이를 줄일 수 있습니다.
 - 컬럼이 모든 기존 행에 대하여 NULL 상태인 경우.
 - 테이블에 행이 전혀 없는 경우.
 - 컬럼의 감소된 최대길이가, 해당 컬럼의 모든 행에 대한 기존 값의 최대길이 보다 작지 않은 경우.
- 이미 저장된 행들에 대하여 컬럼이 NULL 상태일 때만 데이터유형이 변환될 수 있습니다.
 단, 예외적으로 데이터유형을 CHAR에서 VARCHAR2로 변환하는 것은, 기존 행의 해당 컬럼에 값이 있어도 수행될 수 있습니다.
- 이미 저장된 행들에 대하여 컬럼이 NULL 상태일 이거나, 컬럼의 최대길이가 같이 변경되지 않을 때는, CHAR에서 VARCHAR2로 또는, VARCHAR2에서 CHAR로 컬럼의 데이터유형이 변환될 수 있습니다.

8> HR.EMP_APP_TEST2 테이블의 컬럼 정의를 수정한 후, HR.EMP_APP_TEST2 테이블의 데이터를 확인합니다.

```
SQL> SELECT * FROM HR.EMP_APP_TEST2;
```

EMPNO	LAST_NAME	SAL	ANNSAL	COMMRATE	DEPTNO	JOB_ID	HIREDATE
145	Russell	14000	168000	.4	80	-	
146	Partners	13500	162000	.3	80	-	
147	Errazuriz	12000	144000	.3	80	-	
148	Cambrault	11000	132000	.3	80	-	
149	Zlotkey	10500	126000	.2	80	-	
...							

34개의 행이 선택됨

☞ HR.EMP_APP_TEST2 테이블의 기존 행에는, HIREDATE 컬럼에 설정된 DEFAULT 값인 SYSDATE 함수가 적용되지 않으므로, 여전히 NULL 상태입니다.

☞ HR.EMP_APP_TEST2 테이블의 기존 행에는, JOB_ID 컬럼에 수행된 DEFAULT 값 설정 해제가 적용되지 않으므로, 여전히 '-'가 표시됩니다.

9> HR.EMP_APP_TEST2 테이블의 JOB_ID 컬럼을 삭제하시오.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2
      DROP COLUMN JOB_ID ;    -- COLUMN 키워드를 명시했으므로, 하나의 컬럼만 삭제할 수 있습니다.
                                -- 컬럼이름을 괄호로 감싸면 안됩니다.
Table HR.EMP_APP_TEST2이(가) 변경되었습니다.
```

■ 기본문법: 컬럼삭제(2 가지 문법)

ALTER TABLE 스키마이름.테이블이름	-- 명시된 한 개 이상의 컬럼들이 삭제됩니다.
DROP(컬럼이름, 컬럼이름,...) CASCADE CONSTRAINTS;	-- CASCADE CONSTRAINTS 옵션은 생략될 수 있습니다. -- 컬럼이름 전체를 괄호로 감싸주어야 합니다.
ALTER TABLE 스키마이름.테이블이름	-- COLUMN 키워드를 명시하면, 한 번에 하나의 컬럼만 삭제됩니다.
DROP COLUMN 컬럼이름 CASCADE CONSTRAINTS;	-- CASCADE CONSTRAINTS 옵션은 생략될 수 있습니다. -- COLUMN 키워드를 명시하면, 컬럼이름을 괄호로 감싸면 안됩니다.

■ 테이블에 있는 기존 컬럼의 정의를 삭제할 때, 다음의 사항을 고려하십시오.

- 컬럼 삭제 시에, 이미 저장된 행들에 대하여 삭제되는 컬럼의 값을 모두 삭제하는 작업이 발생됩니다.
따라서, 데이터의 유실이 발생될 수 있으며, 테이블의 행이 많은 경우, 시간이 오래 소요될 수 있습니다.
또한, 컬럼 삭제가 완료되기 전까지는, 테이블의 모든 행에 대하여 배타적인 락(LOCK)이 걸리므로,
다른 세션에서의 테이블에 대한 행의 입력, 수정, 삭제가 수행될 수 없습니다.
- 컬럼 삭제는 수행완료와 동시에 커밋 되므로, 롤백 될 수 없습니다.
즉, 컬럼이 삭제된 후에는 복구할 수 없습니다.
- 테이블의 컬럼을 삭제한 후에, 최소 하나 이상의 컬럼이 테이블에 남아 있어야 합니다.
- 삭제하려는 컬럼이 제약 조건의 일부이거나 인덱스 키의 일부인 경우,
CASCADE CONSTRAINTS 옵션을 명시해야만 컬럼이 삭제될 수 있습니다.
- 삭제하려는 컬럼이 파티션 테이블의 파티션-키의 일부를 구성하는 컬럼이거나,
인덱스-구성-테이블(Index-Organized Table)의 기본키 제약조건의 일부를 구성하는 컬럼인 경우,
해당 컬럼은 삭제할 수 없습니다.
☞ 파티션 테이블과 인덱스 구성 테이블에 대한 자세한 내용은 Oracle Database Administrator's Guide를
참조하십시오.

10> HR.EMP_APP_TEST2 테이블의 HIREDATE 컬럼을 UNUSED로 설정하시오.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2 -- HIREDATE 컬럼에 대하여 데이터조회, 컬럼정보확인이 불가능합니다.
      SET UNUSED (HIREDATE) ;
```

Table HR.EMP_APP_TEST2이(가) 변경되었습니다.

- ☞ [ALTER TABLE ... SET UNUSED] 문은 DDL 문장이므로 ROLLBACK이 되질 않으며,
일단 UNUSED로 설정된 컬럼을 다시 사용할 수 있도록, UNUSED 설정을 해제하는 방법은 없습니다.

■ 기본문법: 컬럼을 UNUSED로 설정(2 가지 문법)

ALTER TABLE 스키마이름.테이블이름

```
SET UNUSED(컬럼이름, 컬럼이름,...) ; -- 명시된 한 개 이상의 기존 컬럼들이 UNUSED로 설정됩니다.
                                         -- 반드시 컬럼이름들 전체를 괄호로 감싸주어야 합니다.
```

ALTER TABLE 스키마이름.테이블이름

```
SET UNUSED COLUMN 컬럼이름 ; -- COLUMN 키워드를 명시하면, 한 번에 하나의 컬럼만 UNUSED로 설정됩니다.
                                 -- COLUMN 키워드를 명시하면, 컬럼이름을 괄호로 감싸면 안됩니다.
```

■ 테이블에 있는 기존 컬럼을 UNUSED로 설정할 때, 다음의 사항을 고려하십시오.

- 일단, 컬럼을 UNUSED로 설정하면, 이를 해제할 수 없습니다.
 - 컬럼에 대하여 UNUSED 상태로 지정해도, 대상 컬럼의 데이터가 테이블의 각 행에서 실제로 제거되는 것은 아니기 때문에, 컬럼을 삭제하는 것보다 완료시간이 빨라집니다.
 - UNUSED-컬럼은 테이블의 행으로부터 실제로 컬럼의 데이터가 삭제된 것은 아니지만,
삭제된 것으로 간주됩니다. 따라서, 컬럼이 UNUSED로 표시된 후에는, 해당 컬럼에 액세스할 수 없습니다.
즉, SELECT * 문이나, DESCRIBE문으로 컬럼의 데이터나 정의가 표시되지 않습니다.
 - UNUSED-컬럼에 대해서는, 해당 컬럼을 삭제하는 것만 가능합니다.
 - UNUSED-컬럼을 테이블로부터 삭제하지 않더라도, 동일한 이름을 가진 새 컬럼을 테이블에 추가할 수 있습니다.
 - USER_UNUSED_COL_TABS 딕셔너리 뷰를 통해, 테이블에 UNUSED로 설정된 컬럼의 개수만 확인할 수 있습니다.
- ☞ 데이터 딕셔너리 뷰에 대해서는, 이 후 단원에서 자세히 설명합니다.

11> HR.EMP_APP_TEST2 테이블에 대하여, UNUSED-컬럼이 몇 개 있는지 확인하시오.

```
SQL> SELECT * FROM USER_UNUSED_COL_TABS; -- UNUSED-컬럼이 테이블에 몇 개가 있는지만 확인할 수 있으며,
      -- UNUSED-컬럼의 이름 등 다른 정보는 확인할 수 없습니다.
```

TABLE_NAME	COUNT
DEPT_APP_TEST1	1

12> HR.EMP_APP_TEST2 테이블에 있는 모든 UNUSED-컬럼을 삭제하시오.

```
SQL> ALTER TABLE HR.EMP_APP_TEST2  
      DROP UNUSED COLUMNS ;
```

Table HR.EMP APP TEST20이(가) 변경되었습니다.

☞ 테이블에 대하여 UNUSED로 설정된 컬럼은, 위의 문장으로 삭제하는 것만 가능하며, 이 때, 테이블의 UNUSED-컬럼들 모두가 삭제됩니다. 즉, UNUSED-컬럼 중 일부만 삭제하는 것은 불가능합니다.

■ 기본문법: UNUSED-컬럼 삭제

ALTER TABLE 스키마이름.테이블이름

DROP UNUSED COLUMNS ; — 현재 UNUSED로 표시된 모든 컬럼을 테이블에서 제거합니다.

-- 테이블에 UNUSED로 표시된 컬럼이 없어도 오류를 반환하지 않습니다.

■ 위의 문장을 실행하면, 테이블에 존재하는 UNUSED-컬럼에 대한 실제 데이터가 삭제되고, 저장공간이 회수됩니다.

13> HR.EMP_APP_TEST2 테이블의 이름을 DEPT_APP_TEST3로 변경하시오.

```
SQL> ALTER TABLE HR.EMP APP TEST2
```

RENAME TO EMP APP TEST3 ; — DEPT_APP_TEST3 앞에 스키마이름을 명시하면 안됩니다.

Table HR.EMP_APP_TEST201(가) 변경되었습니다.

■ 기본문법: 테이블이름 변경(2 가지 문법, 10gNF).

ALTER TABLE 스키마이름.기존_테이블이름

RENAME TO 새로운_테이블이름 : -- 새로운_테이블이름 앞에 스키마이름을 명시할 수 없습니다.

`RENAME` 기존_테이블이름 — 기존_테이블이름 및 새로운_테이블이름 앞에 스키마이름을 명시할 수 없습니다.

TO 새로운 테이블 이름: — 이 문장은 기존 테이블을 소유한 계정만 실행할 수 있습니다.

```
RENAME          1 : , , PRIVATE , .  
TO           2 , , , , , , , , , , , ,
```

14> HR.EMP_APP_TEST3 테이블의 LAST_NAME 컬럼의 이름을 LASTNAME으로 변경하시오.

```
SQL> ALTER TABLE HR.EMP_APP_TEST3  
      RENAME COLUMN LAST_NAME TO LASTNAME ;
```

Table HR.EMP_APP_TEST301(가) 변경되었습니다.

■ 기본문법: 컬럼이름 변경(9iNF)

ALTER TABLE 스키마이름.테이블이름

```
RENAME COLUMN 기존컬럼이름 TO 새로운_컬럼이름 ;
```

13-2. 제약조건 추가, 삭제, 활성화, 비활성화 및 제약조건의 이름변경.

- 기존 테이블에 대하여 필요한 경우, 제약조건을 추가, 삭제, 활성화, 비활성화 하거나 제약조건의 이름을 변경해야 할 수 있습니다. 이러한 작업 시에도 ALTER TABLE 문을 사용합니다.
- 단, 이미 존재하는 제약조건에 대하여 제약 조건의 타입, 제약조건이 적용되는 컬럼을 변경할 수는 없습니다. 특히, CHECK 제약조건의 조건식을 변경할 수 없습니다. 따라서, 이런 작업이 필요한 경우에는, 기존 제약조건을 삭제 후, 원하는 제약조건을 다시 추가해 주어야 합니다.
- 다음의 실습을 수행하여, 제약조건 추가, 삭제를 포함한, ALTER TABLE문의 사용방법을 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 실습에 사용할, HR.EMP_APP_TEST2 테이블을 하위질의를 이용하여 다음과 같이 생성합니다.

```
SQL> CREATE TABLE HR.DEPT_CONS_TEST1
      AS SELECT * FROM HR.DEPARTMENTS; -- WHERE 절이 없으므로 모든 행이 복사됩니다.
```

Table HR.DEPT_CONS_TEST1(가) 생성되었습니다.

```
SQL> CREATE TABLE HR.EMP_CONS_TEST1
      AS SELECT * FROM HR.EMPLOYEES
      WHERE DEPARTMENT_ID=10; -- 이 조건을 만족하는 하나의 행만 복사됩니다.
```

Table HR.EMP_CONS_TEST1(가) 생성되었습니다.

☞ 원본테이블로부터 하위질의를 이용하여 새로운 테이블을 생성하면, 데이터 및 컬럼 정의가 복사되며, 제약조건의 경우에는, NOT NULL 제약조건만 복사되고, 다른 타입의 제약조건은 복사되지 않습니다.

3> DESCRIBE 명령어를 이용하여 생성된 HR.DEPT_CONS_TEST1 테이블의 컬럼 정의를 확인합니다.

```
SQL> DESC HR.DEPT_CONS_TEST1          -- HR.EMP_CONS_TEST1 테이블에 대하여 DESCRIBE 명령어를 실행하면
      -- 복사된 NOT NULL 제약조건을 표시결과로부터 확인할 수 있습니다.
  이름          널          유형
  -----
DEPARTMENT_ID      NUMBER(4)
DEPARTMENT_NAME NOT NULL VARCHAR2(30) -- 복사된 NOT NULL 제약조건입니다.
MANAGER_ID        NUMBER(6)
LOCATION_ID       NUMBER(4)
```

4> HR.DEPT_CONS_TEST1 테이블에, ALTER TABLE...MODIFY 문을 이용하여, 다음의 제약조건을 추가하시오.

- DEPARTMENT_ID 컬럼에 PK_DEPTID_DEPT_CONS_TEST1 이름의 기본키 제약조건을 추가하시오.

```
SQL> ALTER TABLE HR.DEPT_CONS_TEST1
      MODIFY department_id CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1 PRIMARY KEY ;
```

Table HR.DEPT_CONS_TEST1(가) 변경되었습니다.

■ 기본문법: 제약조건 추가

-- ALTER TABLE...ADD 문 사용: 테이블레벨로 제약조건 추가:.

ALTER TABLE 스키마이름.테이블이름

ADD CONSTRAINT 제약조건이름 CONSTRAINT_TYPE (컬럼이름(들));

-- ALTER TABLE...MODIFY 문 사용: 컬럼레벨로 제약조건 추가

ALTER TABLE 스키마이름.테이블이름

MODIFY 컬럼이름 CONSTRAINT 제약조건이름 CONSTRAINT_TYPE ; -- NOT NULL 제약조건은 이 방법만 가능합니다.

- NOT NULL 제약조건은 위의 방법 중, ALTER TABLE ... MODIFY 문으로만 추가할 수 있습니다.

- NOT NULL 이 외의 제약조건은, 위의 두 가지 방법을 모두 사용할 수 있습니다.

- 제약조건은 테이블레벨 또는 컬럼레벨에서 추가될 수 있습니다. 단, 다음의 제한사항이 있습니다.

- NOT NULL 제약조건은, 컬럼레벨로만 추가될 수 있으며, 테이블에 행이 하나도 없거나 또는 모든 행에 대하여 해당 컬럼의 값이 모두 있는 경우에만, 추가됩니다.
- 기본키, UNIQUE 또는 외래키 제약조건이, 2 개 이상의 컬럼을 조합된 형태로 추가될 때는 테이블레벨로만 추가될 수 있습니다.

- 제약조건의 이름을 명시하는 [CONSTRAINT 제약조건이름] 구문은 생략될 수 있습니다.

만약 생략되면, 데이터베이스 서버가 자동으로 제약조건의 이름을 지정합니다.

■ 테이블에 새로운 제약조건을 추가할 때, 다음의 사항을 고려하십시오.

- 기본키, UNIQUE 제약조건은 추가될 때, 제약조건의 컬럼 구성과 동일한 인덱스 키를 가지는,

사용 가능한 인덱스가 있어야 합니다.

- 만약 사용할 수 있는 인덱스가 있으면, 자동으로 기존 인덱스를 사용합니다.
- 만약 사용할 수 있는 인덱스가 없으면, 자동으로 새로운 인덱스를 생성하여 사용합니다.

- 테이블에 이미 행들이 존재하면, 제약조건 추가 시에, 기존 데이터에 대하여,

추가되는 제약조건의 검사가 수행됩니다.

5> HR.EMP_CONS_TEST1 테이블에, ALTER TABLE...ADD 문을 이용하여, 다음의 제약조건을 추가하시오.

- EMPLOYEE_ID 컬럼에 PK_EMPID_EMP_CONS_TEST1 이름의 기본키 제약조건을 추가합니다.
- DEPARTMENT_ID 컬럼에 DEPT_CONS_TEST1 테이블의 기본키 컬럼을 참조하는 FK_DEPTID_EMP_CONS_TEST1 이름의 외래키 제약조건을 추가하시오. 단, 외래키에 대하여 ON DELETE SET NULL 옵션을 명시하시오.
- SALARY > 0 인 조건을 검사하는 CK_SAL_EMP_CONS_TEST1 이름의 CHECK 제약조건을 추가하시오.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      ADD CONSTRAINT PK_EMPID_EMP_CONS_TEST1 PRIMARY KEY (EMPLOYEE_ID) ;
```

Table HR.EMP_CONS_TEST1이(가) 변경되었습니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      ADD CONSTRAINT FK_DEPTID_EMP_CONS_TEST1 FOREIGN KEY (DEPARTMENT_ID)
      REFERENCES HR.DEPT_CONS_TEST1(DEPARTMENT_ID) ON DELETE SET NULL ;
```

Table HR.EMP_CONS_TEST1이(가) 변경되었습니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      ADD CONSTRAINT CK_SAL_EMP_CONS_TEST1 CHECK (SALARY > 0) ;
```

Table HR.EMP_CONS_TEST1이(가) 변경되었습니다.

☞ 외래키 제약조건을 추가할 때, 맨 끝에 ON DELETE SET NULL 또는 ON DELETE CASCADE 절을 명시하지 않으면, 하위테이블(HR.EMP_CONS_TEST1)에 종속 행이 있는, 상위테이블(HR.DEPT_CONS_TEST1)의 참조된 키 값을 가지는 행을 삭제할 수 없습니다.

☞ 외래키 제약조건을 추가할 때, 맨 끝에 ON DELETE SET NULL 또는 ON DELETE CASCADE 절을 명시하여, 상위테이블의 참조된 상위키 값을 가지는 행을 삭제할 때, 하위 테이블의 종속 행에 대하여 참조 무결성을 처리하는 방법을 지정할 수 있습니다.

- ON DELETE SET NULL 절: 상위테이블의 참조된 상위키 값을 가지는 행이 삭제될 때, 상위키 값과 동일한 값을 가지는 하위테이블의 종속된 모든 행의 하위키 값을 NULL로 변환합니다.
- ON DELETE CASCADE 절: 상위테이블의 참조된 상위키 값을 가지는 행이 삭제될 때, 상위키 값과 동일한 값을 가지는 하위테이블의 종속된 모든 행을 삭제합니다.

☞ 위에서 추가한 FK_DEPTID_EMP_CONS_TEST1 외래키 제약조건을 ALTER TALBE...MODIFY 문을 이용하여 작성하면, 다음과 같습니다. ALTER TALBE...MODIFY 문과 비교하면, 외래키 절을 명시하지 않습니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      MODIFY DEPARTMENT_ID CONSTRAINT FK_DEPTID_EMP_CONS_TEST1
      REFERENCES HR.DEPT_CONS_TEST1(DEPARTMENT_ID) ON DELETE SET NULL ;
```

6> HR.EMP_CONS_TEST1 테이블에, 다음의 제약조건을 추가하시오.

- FIRST_NAME 컬럼에 NN_FNAME_EMP_CONS_TEST1 이름의 NOT NULL 제약조건을 추가합니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1  
      MODIFY FIRST_NAME CONSTRAINT NN_FNAME_EMP_CONS_TEST1 NOT NULL ;
```

Table HR.EMP_CONS_TEST1(가) 변경되었습니다.

☞ NOT NULL 제약조건을 추가할 때는 ALTER TABLE...MODIFY 문을 사용해야만 합니다.

☞ 만약, ALTER TABLE...ADD 문을 사용하여 NOT NULL 제약조건을 추가하면, 다음의 오류가 발생됩니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1  
      ADD CONSTRAINT NN_FNAME_EMP_CONS_TEST1 NOT NULL (FIRST_NAME);
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.EMP_CONS_TEST1
```

```
ADD CONSTRAINT NN_FNAME_EMP_CONS_TEST1 NOT NULL (FIRST_NAME)
```

오류 보고 -

SQL 오류: ORA-00904: : 부적합한 식별자

```
00904. 00000 - "%s: invalid identifier"
```

*Cause:

*Action:

7> 앞에서 추가한 다음의 제약조건을 비활성화 시키시오.

- HR.EMP_CONS_TEST1 테이블의 PK_EMPID_EMP_CONS_TEST1 이름의 기본키 제약조건
- HR.DEPT_CONS_TEST1 테이블의 PK_DEPTID_DEPT_CONS_TEST1 이름의 기본키 제약조건

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1          -- 제약조건이 비활성화 되면, DML 수행 시에,
      DISABLE CONSTRAINT PK_EMPID_EMP_CONS_TEST1 ;  -- 해당 제약조건에 대한 검사가 수행되지 않습니다.
```

Table HR.EMP_CONS_TEST1(가) 변경되었습니다.

```
SQL> ALTER TABLE HR.DEPT_CONS_TEST1          -- CASCADE절에 의해, 관련된 외래키
      DISABLE CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1 CASCADE;  -- 제약조건도 함께 비활성화 됨.
```

Table HR.DEPT_CONS_TEST1(가) 변경되었습니다.

- 제약조건이 비활성화 되면, DML 수행 시에, 해당 제약조건에 대한 검사가 수행되지 않습니다.

- 기본문법: 제약조건 비활성화

```
ALTER TABLE 스키마이름.테이블이름
DISABLE CONSTRAINT 제약조건이름 CASCADE ; -- CASCADE 옵션은 생략될 수 있습니다.
```

```
ALTER TABLE 스키마이름.테이블이름  -- 기본키 제약조건은, 테이블에 하나만 정의되므로, 이 문장으로
DISABLE PRIMARY KEY CASCADE ;      -- 기본키 제약조건의 이름을 올라도 비활성화시킬 수 있습니다.
```

- CASCADE절은, 다른 테이블의 외래키 제약조건에 의하여 참조되는, 기본키 또는 UNIQUE 제약조건을 비활성화 시킬 때 사용되며, 다른 테이블에 정의된 관련된 모든 외래키 제약조건(들)이 먼저 비활성화된 후, 해당 기본키 또는 UNIQUE 제약조건이 비활성화됩니다.
 - CASCADE절을 사용하지 않고, 외래키 제약조건으로 참조되는 기본키 제약조건을 비활성화 시키려고 시도하면, 다음과 같은 오류가 발생됩니다.

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.DEPT_CONS_TEST1
DISABLE CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1
오류 보고 -
SQL 오류: ORA-02297: 제약 조건(HR.PK_DEPTID_DEPT_CONS_TEST1)을 사용 안함으로 설정 불가 - 종속성이 존재
합니다.
02297. 00000 - "cannot disable constraint (%s.%s) - dependencies exist"
*Cause:   an alter table disable constraint failed because the table has
          foreign keys that are dependent on this constraint.
*Action:  Either disable the FOREIGN KEY constraints or use disable cascade
```

- 제약조건을 비활성화 시킬 때, 다음의 사항을 고려하십시오.

- 자동으로 생성된 인덱스를 사용하는 기본키, UNIQUE 제약조건을 비활성화 시키면, 인덱스가 자동으로 삭제됩니다.
- 사용자가 생성한 인덱스를 사용하는 기본키, UNIQUE 제약조건을 비활성화 시키면, 인덱스는 삭제되지 않습니다.

8> 앞에서 비활성화 시킨 다음의 제약조건을 활성화 시키시오.

- HR.EMP_CONS_TEST1 테이블의 PK_EMPID_EMP_CONS_TEST1 이름의 기본키 제약조건
- HR.EMP_CONS_TEST1 테이블의 FK_DEPTID_EMP_CONS_TEST1 이름의 외래키 제약조건
- HR.DEPT_CONS_TEST1 테이블의 PK_DEPTID_DEPT_CONS_TEST1 이름의 기본키 제약조건

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      ENABLE CONSTRAINT PK_EMPID_EMP_CONS_TEST1;    -- 제약조건을 활성화 시키면,
                                                     -- DML 수행 시에 제약조건의 검사가 수행됩니다.
```

Table HR.EMP_CONS_TEST1(가) 변경되었습니다.

```
SQL> ALTER TABLE HR.DEPT_CONS_TEST1          -- 기본키 및 UNIQUE 제약조건을 활성화
      ENABLE CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1;  -- 시킬 때는, CASCADE절을 사용할 수 없습니다.
```

Table HR.DEPT_CONS_TEST1(가) 변경되었습니다.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1
      ENABLE CONSTRAINT FK_DEPTID_EMP_CONS_TEST1;
```

Table HR.EMP_CONS_TEST1(가) 변경되었습니다.

■ 기본문법: 제약조건 활성화

```
ALTER TABLE 스키마이름.테이블이름   -- 기본키 및 UNIQUE 제약조건을 활성화 시에는, 비활성화 시와 달리,
ENABLE CONSTRAINT 제약조건이름 ;    -- CASCADE절을 사용할 수 없습니다.
```

```
ALTER TABLE 스키마이름.테이블이름   -- 기본키 제약조건은, 테이블에 하나만 정의되므로, 이 문장으로
ENABLE PRIMARY KEY ;                -- 기본키 제약조건의 이름을 올라도 활성화시킬 수 있습니다.
```

■ 제약조건을 활성화 시킬 때, 다음의 사항을 고려하십시오.

- 기본키, UNIQUE 제약조건은 활성화될 때, 제약조건의 컬럼 구성과 동일한 인덱스 키를 가지는, 사용 가능한 인덱스가 있어야 합니다.
 - 만약 사용할 수 있는 인덱스가 있으면, 자동으로 기존 인덱스를 사용합니다.
 - 만약 사용할 수 있는 인덱스가 없으면, 자동으로 새로운 인덱스를 생성하여 사용합니다.
- 테이블에 이미 행들이 존재하면, 제약조건 활성화 시에, 기존 데이터에 대하여, 제약조건의 검사가 수행됩니다.
- **기본키 제약 조건이 비활성화 될 때, CASCADE 옵션으로 함께 비활성화 된 외래키 제약조건은,**
 - 기본키 제약 조건이 활성화될 때, 함께 활성화시킬 수 없습니다. 따라서, 종속하는 외래키 제약조건은 별도로 활성화 시켜야만 합니다.**
- UNIQUE KEY 또는 기본키 제약 조건을 활성화하려면
 - 테이블에 인덱스를 생성하는 데 필요한 권한을 가지고 있어야 합니다.

9> HR.DEPT_CONS_TEST1 테이블의 PK_DEPTID_DEPT_CONS_TEST1 이름의 기본키 제약조건을 삭제하시오

```
SQL> ALTER TABLE HR.DEPT_CONS_TEST1                                     -- CASCADE절에 의해, 관련된 외래키
      DROP CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1 CASCADE; -- 제약조건도 함께 삭제됩니다.
```

Table HR.DEPT_CONS_TEST1이(가) 변경되었습니다.

■ 기본문법: 제약조건 삭제

ALTER TABLE 스키마이름.테이블이름 — 이 문장을 사용하여 모든 제약조건을 삭제할 수 있습니다.

DROP CONSTRAINT 제약조건이름 CASCADE; — 외래키 제약조건으로 참조되는 기본키 및 UNIQUE

— 제약조건을 삭제할 때만 CASCADE 옵션이 사용됩니다.

ALTER TABLE 스키마이름.테이블이름

— 기본키 제약조건만 삭제할 수 있습니다.

DROP PRIMARY KEY CASCADE;

— 외래키 제약조건으로 참조되는 PRIMARY 제약조건을

— 삭제할 때만 CASCADE 옵션이 사용됩니다.

ALTER TABLE 스키마이름.테이블이름

— UNIQUE 제약조건만 삭제할 수 있습니다.

DROP UNIQUE (컬럼이름) CASCADE;

— UNIQUE 제약조건의 영향을 받는 컬럼의 이름입니다.

— 외래키 제약조건으로 참조되는 UNIQUE 제약조건을

— 삭제할 때만 CASCADE 옵션이 사용됩니다.

- CASCADE절은, 다른 테이블의 외래키 제약조건에 의하여 참조되는, 기본키 또는 UNIQUE 제약조건을

삭제할 때 사용되며, 다른 테이블에 있는 관련된 외래키 제약조건(들)이 먼저 삭제된 후, 해당 PRIMARY KEY 또는 UNIQUE 제약조건이 삭제됩니다.

- CASCADE절을 사용하지 않고, 외래키 제약조건으로 참조되는 기본키 제약조건을 삭제하려고 시도하면, 다음과 같은 오류가 발생됩니다.

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.DEPT_CONS_TEST1
DROP CONSTRAINT PK_DEPTID_DEPT_CONS_TEST1
```

오류 보고 -

SQL 오류: ORA-02273: 고유/기본 키가 외부 키에 의해 참조되었습니다

02273. 00000 - "this unique/PRIMARY KEY is referenced by some FOREIGN KEYS"

*Cause: Self-evident.

*Action: Remove all references to the key before the key is to be dropped.

■ 기본키 및 UNIQUE 제약조건을 삭제할 때, 다음의 사항을 고려하십시오.

- 자동으로 생성된 인덱스를 사용하는 기본키, UNIQUE 제약조건이 삭제되면,

인덱스도 자동으로 삭제됩니다.

- 사용자가 생성한 인덱스를 사용하는 기본키, UNIQUE 제약조건이 삭제되면

인덱스는 삭제되지 않습니다.

■ 제약조건 삭제는, 제약조건의 컬럼구성, 제약조건 유형, CHECK 제약조건의 조건식을 변경해야 할 경우에,

변경이 안되므로, 제약조건을 삭제 후, 다시 추가합니다. 이 때, 제약조건 삭제 방법이 사용됩니다.

10> HR.EMP_CONS_TEST1 테이블의 NN_FNAME_EMP_CONS_TEST1 제약조건의 이름을, FNAME_EMP_CONS_TEST1_NN으로 변경하시오.

```
SQL> ALTER TABLE HR.EMP_CONS_TEST1  
      RENAME CONSTRAINT NN_FNAME_EMP_CONS_TEST1 TO FNAME_EMP_CONS_TEST1_NN ;
```

Table HR.EMP_CONS_TEST1(가) 변경되었습니다.

■ 기본문법: 제약조건의 이름 변경

ALTER TABLE 스키마이름.테이블이름

— 제약조건의 이름 앞에 스키마이름을 명시할 수 없습니다.

RENAME CONSTRAINT 기존_제약조건이름 TO 새로운_제약조건이름;

- 구축 초기에 테이블에 제약조건을 추가하면서, CONSTRAINT 절을 이용하여, 이름을 명시하지 않으면, 서버에 의해서 자동으로 이름이 지정됩니다. 이 자동으로 지정된 이름이, 제약조건 관리나 사용에 불편한 경우에, ALTER TABLE ... RENAME CONSTRAINT 문을 이용하여, 자동으로 지정된 기존이름을 사용자가 관리하기 쉬운 이름으로 변경할 수 있습니다.

13-3. 제약조건-검사의 지연

- 제약조건의 유효성 검사는 기본적으로, 각각의 DML문 처리 중에 수행됩니다. 제약조건-검사의 지연이란, 각 DML문을 처리하는 중에 제약조건의 유효성 검사를 수행하지 않고, 트랜잭션에 대한 COMMIT 요청이 왔을 때 제약조건의 유효성 검사를 수행하도록, 제약조건의 유효성 검사를 지연시키는 것입니다. 만약 제약조건의 유효성 검사를 위반하는 데이터가 발견되면, 해당 DML문이 포함된 전체 트랜잭션이 롤백 됩니다.
- 제약조건-검사의 지연 기능은, DEFERRABLE절이 명시되어 정의된 제약조건에만 사용될 수 있습니다.
- 다음과 같은 제약조건의 속성을 지정하는 SQL절을 이용하여, 해당 제약조건에 대한 제약조건-검사 지연 기능의 사용유무 및 기본 거동을 지정할 수 있습니다.
 - NOT DEFERRABLE 또는 DEFERRABLE 속성(SQL절)
 - NOT DEFERRABLE 속성(절)은, DEFERRABLE절 없이 정의된 제약조건의 속성이며, 무조건 각각의 DML문 수행 시에, 제약조건의 유효성 검사가 수행됩니다. 즉, 제약조건-검사 지연 기능을 사용할 수 없습니다.
 - DEFERRABLE 속성(절)은, DEFERRABLE절이 명시되어 정의된 제약조건의 속성이며, 필요한 경우에 제약조건-검사 지연 기능을 사용 또는 사용하지 않도록 지정할 수 있습니다.
 - INITIALLY IMMEDIATE 또는 INITIALLY DEFERRED 속성(절)은 DEFERRABLE 속성을 가진 제약조건의 추가적인 속성(절)입니다. 이 절들 중 하나도 명시하지 않으면, 디폴트로 INITIALLY IMMEDIATE 속성이 지정됩니다.
 - INITIALLY IMMEDIATE 속성(절)은, 기본적으로 각각의 DML문 수행 시에 제약조건의 유효성 검사를 수행합니다. 필요한 경우, 트랜잭션의 COMMIT 요청이 왔을 때까지 제약조건-검사를 지연시킬 수 있습니다.
 - INITIALLY DEFERRED 속성(절)은, 기본적으로 트랜잭션의 COMMIT 요청이 왔을 때까지 제약조건-검사를 지연시킵니다. 필요한 경우, 각 DML문 수행 시에 제약조건의 유효성 검사를 수행시킬 수 있습니다.
 - DEFERRABLE 속성의 기본키 또는 UNIQUE 제약조건을 생성하려면, 해당 제약조건에 대한 중복된 인덱스-키 값이 허용되는 비고유 인덱스(NONUNIQUE INDEX)를 생성해야 합니다.
 - 위의 제약조건 속성에 따라, 제약조건 검사가 수행되는 순서는 다음과 같습니다
 - NOT DEFERRABLE 및 DEFERRABLE INITIALLY IMMEDIATE 속성의 제약조건이 먼저 검사됩니다.
 - DEFERRABLE 속성이 지정된 제약조건 중, 필요에 의하여, 지연기능이 활성화 된 제약조건이 검사됩니다.
- 외래키 제약조건의 ON DELETE CASCADE 절의 기능으로, 상위테이블의 상위키 행이 삭제될 때, 하위테이블의 하위키 행이 함께 삭제되는 작업이 발생하면, 부모테이블의 기본키 제약조건을 참조하는 하위테이블의 외래키 제약 조건이 DEFERRED인지 또는 IMMEDIATE인지에 관계없이, 항상 해당 작업을 발생시킨 명령문의 일부로 작업이 실행됩니다.
- 제약조건의 속성을 이용하여, 제약조건-검사 지연기능을 사용하는 기본적인 방법 외에도, 세션레벨에서 명령문으로 설정하여, 트랜잭션-단위 또는 세션-단위로 제약조건-검사 지연기능을 사용할 수 있습니다.

- 다음의 실습을 수행하여, 제약조건-검사 지연 기능을 사용하는 방법을 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 실습에 사용할, HR.EMP_APP_TEST2 테이블을 하위질의를 이용하여 다음과 같이 생성합니다.

```
SQL> CREATE TABLE HR.CONS_CHK_TEST1 (
    EMPNO NUMBER(6),
    SALARY NUMBER(8,2),
    BONUS NUMBER(8,2),
    CONSTRAINT PK_EMPNO_CONS_CHK_TEST1 PRIMARY KEY (EMPNO),
    CONSTRAINT CK_SAL_CONS_CHK_TEST1 CHECK (SALARY > 100) DEFERRABLE,
    CONSTRAINT CK_BONUS_CONS_CHK_TEST1 CHECK (BONUS > 0 ) DEFERRABLE INITIALLY DEFERRED );
```

Table HR.CONS_CHK_TEST1이(가) 생성되었습니다.

- ☞ PK_EMPNO_CONS_CHK_TEST1 제약 조건에는 DEFERRABLE절이 없습니다. 따라서, 무조건 각각의 DML문 실행 시에 제약조건 유효성 검사가 수행되며, 제약조건-검사 지연 기능을 사용할 수 없습니다.
- ☞ CK_SAL_CONS_CHK_TEST1 제약 조건에는 DEFERRABLE절만 명시되어 있으므로, 디폴트로 INITIALLY IMMEDIATE 속성이 지정됩니다. 따라서, 기본적으로 각각의 DML문 실행 시에 제약조건 유효성 검사가 수행되며, 필요한 경우, 제약조건-검사 지연 기능을 사용할 수 있습니다.
- ☞ CK_BONUS_CONS_CHK_TEST1 제약 조건에는 DEFERRABLE절과 INITIALLY DEFERRED절이 명시되어 있습니다, 따라서, 기본적으로 트랜잭션에 대한 COMMIT 요청이 올 때, 제약조건의 유효성 검사가 수행되며, 필요한 경우, 각각의 DML 문이 실행될 때, 제약조건의 유효성 검사가 수행되도록 지정할 수 있습니다.

3> 위의 테이블에 다음의 한 행을 입력합니다.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,90, 5); -- DEFERRABLE INITIALLY IMMEDIATE 속성의
-- HR.OK_SAL_CONS_CHK_TEST1 제약조건 위반
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO hr.cons_chk_test1 VALUES (100,90, 5)
오류 보고 -
SQL 오류: ORA-02290: 체크 제약조건(HR.OK_SAL_CONS_CHK_TEST1)이 위배되었습니다
02290. 00000 - "check constraint (%s.%s) violated"
*Cause: The values being inserted do not satisfy the named check
*Action: do not insert values that violate the constraint.
```

- ☞ CK_SAL_CONS_CHK_TEST1 체크 제약조건의 DEFERRABLE INITIALLY IMMEDIATE 속성에 의하여, 각 DML 수행 시에 제약조건의 유효성 검사가 수행되면서, 조건을 위반했으므로, 위의 오류가 발생됩니다.
- ☞ CK_BONUS_CONS_CHK_TEST1 체크 제약조건은 검사가 수행되지 않았습니다.

4> 위의 테이블에 다음의 한 행을 입력합니다.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,190, 0) ; -- DEFERRABLE INITIALLY DEFERRED 속성의
-- HR.CK_BONUS_CONS_CHK_TEST1 제약조건
1 행 이(가) 삽입되었습니다.
-- 검사 하지 않음
```

- ☞ PK_EMPNO_CONS_CHK_TEST1 제약 조건과 CK_SAL_CONS_CHK_TEST1 제약 조건은 DML 문 수행 중에 제약조건의 유효성 검사가 수행되었습니다. 제약 조건을 만족했습니다.
- ☞ CK_BONUS_CONS_CHK_TEST1 체크 제약조건의 DEFERRABLE INITIALLY DEFERRED 속성으로, 제약조건-검사가 지연되었으므로, BONUS 컬럼에 입력되는 값이 조건을 위반했지만, INSERT문 수행 시에 제약조건의 유효성 검사가 수행되지 않았기 때문에, 행이 정상적으로 입력되었습니다.

5> COMMIT문을 실행하여 트랜잭션을 끝냅니다. 다음과 같은 오류가 발생됩니다.

```
SQL> COMMIT ; -- 트랜잭션을 구성하는 DML 문에 관련된 지연된 제약조건이 검사됩니다.

명령의 1 행에서 시작하는 중 오류 발생 -
COMMIT
오류 보고 -
SQL 오류: ORA-02091: 트랜잭션이 룰백되었습니다
ORA-02290: 체크 제약조건(HR.CK_BONUS_CONS_CHK_TEST1)이 위배되었습니다
02091. 00000 - "transaction rolled back"
*Cause: Also see error 2092. If the transaction is aborted at a remote
site then you will only see 2091; if aborted at host then you will
see 2092 and 2091.
*Action: Add rollback segment and retry the transaction.
```

- ☞ CK_BONUS_CONS_CHK_TEST1 체크 제약조건의 DEFERRABLE INITIALLY DEFERRED 속성에 의하여, COMMIT이 요청될 때까지 지연되었던 제약조건-검사가 수행됩니다. BONUS 컬럼에 입력되는 값이 조건을 위반했으므로, INSERT문이 포함된 트랜잭션이 룰백됩니다.

6> SET CONSTRAINT 문을 이용하여, CK_SAL_CONS_CHK_TEST1 제약조건이, 제약조건-검사를 트랜잭션의 COMMIT이 요청될 때 수행하도록 설정하시오.

```
SQL> SET CONSTRAINT HR.CK_SAL_CONS_CHK_TEST1 DEFERRED; -- 특정 제약조건의 속성을 변경합니다.
```

Constraint CK_SAL_CONS_CHK_TEST1(를) 성공했습니다.

■ 기본문법:

SET CONSTRAINT 제약조건이름 DEFERRED; --DEFERRABLE 속성의 지정된 제약조건을 DEFERRED 속성으로 변경합니다.

SET CONSTRAINT 제약조건이름 IMMEDIATE; --DEFERRABLE 속성의 지정된 제약조건을 IMMEDIATE 속성으로 변경합니다.

SET CONSTRAINTS ALL DEFERRED; --가능한 모든 DEFERRABLE 속성의 제약조건을 DEFERRED 속성으로 변경합니다.

SET CONSTRAINTS ALL IMMEDIATE; --가능한 모든 DEFERRABLE 속성의 제약조건을 IMMEDIATE 속성으로 변경합니다.

○ SET CONSTRAINT 문을 이용하여, 제약조건의 기본 속성과 상관없이 유효성 검사가 수행되는 시점을 변경할 수 있습니다. 단, DEFERRABLE 속성의 제약조건에 대해서만 SET CONSTRAINT 문을 사용할 수 있습니다.

NOT DEFERRABLE 속성의 제약조건에 대해서는 사용될 수 없습니다.

- DEFERRED 속성으로 설정되면, COMMIT이 요청될 때까지 제약조건-검사가 지연됩니다.
- IMMEDIATE 속성으로 설정되면, 각 DML 문이 실행될 때마다 제약조건-검사가 수행됩니다.

○ SET CONSTRAINT 문으로 설정된 속성들은 하나의 트랜잭션에서만 유효합니다.

즉, 설정 후, 트랜잭션이 커밋되거나 롤백되면, SET CONSTRAINT 문으로 설정한 속성은 해제됩니다.

7> SET CONSTRAINT 문으로 CK_SAL_CONS_CHK_TEST1 제약조건의 속성을 DEFERRED로 설정한 세션에서, 3> 스텝에서 오류를 발생시킨 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,90, 5); -- DEFERRABLE INITIALLY IMMEDIATE 속성의
-- HR.CK_SAL_CONS_CHK_TEST1 제약조건 위반
1 행 이(가) 삽입되었습니다.
```

☞ PK_EMPNO_CONS_CHK_TEST1 제약 조건은 DML 문 수행 중에 제약조건의 유효성 검사가 수행되었고, 검사 조건을 만족했습니다.

☞ CK_SAL_CONS_CHK_TEST1 제약조건의 속성(INITIALLY IMMEDIATE)과 상관없이, SET CONSTRAINT 문으로 DEFERRED로 설정되어, 제약조건 검사가 지연되었으므로, SALARY 값이 100 보다 작은 90 이지만, INSERT문 수행 시에 제약조건의 유효성 검사가 수행되지 않았기 때문에 행이 정상적으로 입력되었습니다.

☞ CK_BONUS_CONS_CHK_TEST1 체크 제약조건은 기본속성에 의해 제약조건-검사가 지연되었으므로, INSERT문 수행 시에 제약조건의 유효성 검사가 수행되지 않았습니다.

8> COMMIT문을 실행하여 트랜잭션을 끝냅니다. 다음과 같은 오류가 발생됩니다.

```
SQL> COMMIT ; -- 트랜잭션을 구성하는 DML 문에 관련된 지연된 제약조건이 검사됩니다.
```

명령의 1 행에서 시작하는 중 오류 발생 -

COMMIT

오류 보고 -

SQL 오류: ORA-02091: 트랜잭션이 롤백되었습니다

ORA-02290: 체크 제약조건(HR.CK_SAL_CONS_CHK_TEST1)이 위배되었습니다

02091. 00000 - "transaction rolled back"

*Cause: Also see error 2092. If the transaction is aborted at a remote site then you will only see 2091; if aborted at host then you will see 2092 and 2091.

*Action: Add rollback segment and retry the transaction.

☞ 트랜잭션의 커밋이 요청될 때, 지연되었던 CK_SAL_CONS_CHK_TEST1, 제약조건과 CK_BONUS_CONS_CHK_TEST1 제약조건의 검사가 수행됩니다. 이 때, CK_SAL_CONS_CHK_TEST1, 제약조건을 입력된 값이 위반하였으므로 트랜잭션이 롤백됩니다.

☞ 트랜잭션이 커밋 되었으므로, SET CONSTRAINT 문으로 설정된 속성들도 모두 해제되었습니다.

[참고] 다음은 DEFERRABLE 속성(절)이 지정되지 않은 PK_EMPNO_CONS_CHK_TEST1 제약조건에 대하여 SET CONSTRAINT 문을 실행했을 때, 발생된 오류입니다.

```
SQL> SET CONSTRAINT HR.PK_EMPNO_CONS_CHK_TEST1 IMMEDIATE;-- NOT DEFERRABLE 속성의 제약조건은
-- 제약조건 검사 속성을 변경할 수 없습니다.
```

명령의 1 행에서 시작하는 중 오류 발생 -

SET CONSTRAINT HR.PK_EMPNO_CONS_CHK_TEST1 IMMEDIATE

오류 보고 -

SQL 오류: ORA-02447: 지연이 가능하지 않은 제약조건을 지연할 수 없습니다

02447. 00000 - "cannot defer a constraint that is not deferrable"

*Cause: An attempt was made to defer a nondeferrable constraint

*Action: Drop the constraint and create a new one that is deferrable

9> SET CONSTRAINT 문을 이용하여, 가능한 모든 제약조건에 대하여, 각 DML 문이 수행될 때 제약조건-검사를 수행하도록 설정하시오.

```
SQL> SET CONSTRAINTS ALL IMMEDIATE ;          -- 가능한 모든 DEFERRABLE 속성의 제약조건을
                                         -- IMMEDIATE 속성으로 변경합니다.
Constraints ALL을(를) 성공했습니다.
```

☞ NOT DEFERRABLE 속성이 설정된 HR.PK_EMPNO_CONS_CHK_TEST1 제약 조건에는 SET CONSTRAINTS 문의 설정이 적용되지 않습니다. 단, NOT DEFERRABLE은 기본적으로 각각의 DML 문 수행 중에 제약조건의 유효성 검사가 수행됩니다.

☞ CK_SAL_CONS_CHK_TEST1 제약조건과 HR.CK_BONUS_CONS_CHK_TEST1 제약조건 모두 각각의 DML이 수행될 때, 제약조건-검사를 수행하는 IMMEDIATE 속성으로 설정됩니다.

10> SET CONSTRAINTS 문으로 가능한 모든 제약조건의 속성을 IMMEDIATE로 설정한 세션에서, 4> 스텝에서 사용한 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,190, 0) ;          -- DEFERRABLE INITIALLY DEFERRED 속성의
                                         -- CK_BONUS_CONS_CHK_TEST1 제약조건 위반
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO hr.cons_chk_test1 VALUES (100,190, 0)
오류 보고 -
SQL 오류: ORA-02290: 체크 제약조건(HR.CK_BONUS_CONS_CHK_TEST1)이 위배되었습니다
02290. 00000 - "check constraint (%s.%s) violated"
*Cause:    The values being inserted do not satisfy the named check
*Action:   do not insert values that violate the constraint.
```

☞ PK_EMPNO_CONS_CHK_TEST1 제약 조건과 CK_SAL_CONS_CHK_TEST1 제약 조건은 DML 문 수행 중에 제약조건의 유효성 검사가 수행되었습니다. 검사 조건을 만족했습니다.

☞ CK_BONUS_CONS_CHK_TEST1 체크 제약조건도 SET CONSTRAINTS 문으로 지정한 IMMEDIATE 속성 때문에, DML 문 수행 시에 제약조건의 유효성 검사가 수행되어, 오류가 발생되었습니다.

☞ DML 수행 시에 오류가 발생된 것이고, 아직 COMMIT 문을 실행하지 않았으므로, 트랜잭션은 끝나지 않았습니다. 따라서, SET CONSTRAINTS 문으로 설정한 제약조건의 속성이 계속 유지됩니다.

☞ SET CONSTRAINTS 문으로 설정한 제약조건의 속성을 강제로 끝내고 싶으면, COMMIT문이나 ROLLBACK문을 사용하여 트랜잭션을 종료하십시오.

11> 현재 SET CONSTRAINTS 문으로 설정한 제약조건의 속성을 강제로 끝내시오.

```
SQL> COMMIT ;          -- COMMIT문이나 ROLLBACK문 중 하나를 실행할 수 있습니다.
```

커밋 완료.

12> 다음의 ALTER SESSION 문장을 이용하여, 가능한 모든 제약조건을, COMMIT이 요청될 때 제약조건 검사가 수행되도록 설정하시오.

```
SQL> ALTER SESSION SET CONSTRAINTS=DEFERRED ; -- 가능한 모든 DEFERRABLE 속성의 제약조건을
-- DEFERRED 속성으로 변경합니다.
Constraints ALL을(를) 성공했습니다.
```

■ 기본문법:

```
ALTER SESSION SET CONSTRAINTS=DEFERRED ; -- 가능한 모든 DEFERRABLE 속성의 제약조건을
-- DEFERRED 속성으로 변경합니다.
```

```
ALTER SESSION SET CONSTRAINTS=IMMEDIATE ; -- 가능한 모든 DEFERRABLE 속성의 제약조건을
-- IMMEDIATE 속성으로 변경합니다.
```

- ALTER SESSION SET CONSTRAINTS 문을 이용하여, 제약조건의 기본 속성과 상관없이 유효성 검사가 수행되는 시점을 변경할 수 있습니다. 단, DEFERRABLE 속성의 제약조건에 대해서만 ALTER SESSION SET CONSTRAINT 문을 사용할 수 있습니다. NOT DEFERRABLE 속성의 제약조건에 대해서는 사용될 수 없습니다.
 - DEFERRED 속성으로 설정되면, COMMIT이 요청될 때까지 제약조건-검사가 지연됩니다.
 - IMMEDIATE 속성으로 설정되면, 각 DML 문이 실행될 때마다 제약조건-검사가 수행됩니다.
- ALTER SESSION SET CONSTRAINTS 문으로 설정된 속성들은, 다시 변경하지 않는 한, 기본적으로 세션이 종료될 때까지 유지됩니다. 즉, 설정 후, 해당 세션이 유지되는 동안, 트랜잭션이 커밋되거나 롤백 되더라도, 설정된 사항은 계속 유지됩니다.

13> ALTER SESSION SET CONSTRAINTS 문으로 가능한 모든 제약조건의 속성을 DEFERRED로 설정한 세션에서, 3> 스텝에서 오류를 발생시킨 다음의 INSERT문을 실행하시오.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,90, 5) ; -- DEFERRABLE INITIALLY IMMEDIATE 속성의
-- CK_SAL_CONS_CHK_TEST1 제약조건 위반
1 행 이(가) 삽입되었습니다.
```

☞ PK_EMPNO_CONS_CHK_TEST1 제약 조건은 DML 문 수행 중에 제약조건의 유효성 검사가 수행되었고, 검사 조건을 만족했습니다.

☞ CK_SAL_CONS_CHK_TEST1 및 CK_BONUS_CONS_CHK_TEST1 제약조건 모두 검사가 지연되었으므로, INSERT문 실행 중에 제약조건의 검사가 수행되지 않습니다.

14> COMMIT문을 실행하여 트랜잭션을 끝냅니다. 다음과 같은 오류가 발생됩니다.

```
SQL> COMMIT ; -- 트랜잭션을 구성하는 DML 문에 관련된 지연된 제약조건이 검사됩니다.
```

명령의 1 행에서 시작하는 중 오류 발생 -

COMMIT

오류 보고 -

SQL 오류: ORA-02091: 트랜잭션이 룰백되었습니다

ORA-02290: 체크 제약조건(HR.CK_SAL_CONS_CHK_TEST1)이 위배되었습니다

02091. 00000 - "transaction rolled back"

*Cause: Also see error 2092. If the transaction is aborted at a remote site then you will only see 2091; if aborted at host then you will see 2092 and 2091.

*Action: Add rollback segment and retry the transaction.

☞ 트랜잭션의 커밋이 요청될 때, 지연되었던 CK_SAL_CONS_CHK_TEST1, 제약조건과 CK_BONUS_CONS_CHK_TEST1 제약조건의 검사가 수행됩니다. 이 때, CK_SAL_CONS_CHK_TEST1, 제약조건을 입력된 값이 위반하였으므로 트랜잭션이 룰백됩니다.

☞ 앞의 13, 14 스텝을 다시 수행합니다. ALTER SESSION SET CONSTRAINTS 문으로 세션에 설정한 DEFERRED 설정이 유지되어 있으므로, 다음과 같은 동일한 실행결과가 표시됩니다.

```
SQL> INSERT INTO hr.cons_chk_test1 VALUES (100,90, 5) ; -- DEFERRABLE INITIALLY IMMEDIATE 속성의 -- CK_SAL_CONS_CHK_TEST1 제약조건 위반
```

1 행 이(가) 삽입되었습니다.

```
SQL> COMMIT ; -- 트랜잭션을 구성하는 DML 문에 관련된 지연된 제약조건이 검사됩니다.
```

명령의 1 행에서 시작하는 중 오류 발생 -

COMMIT

오류 보고 -

SQL 오류: ORA-02091: 트랜잭션이 룰백되었습니다

ORA-02290: 체크 제약조건(HR.CK_SAL_CONS_CHK_TEST1)이 위배되었습니다

02091. 00000 - "transaction rolled back"

*Cause: Also see error 2092. If the transaction is aborted at a remote site then you will only see 2091; if aborted at host then you will see 2092 and 2091.

*Action: Add rollback segment and retry the transaction.

15> 다음의 ALTER SESSION 문장을 이용하여, 가능한 모든 제약조건을, DML 수행 시에 제약조건 검사가 수행되도록 설정하시오.

```
SQL> ALTER SESSION SET CONSTRAINTS=IMMEDIATE ; -- 가능한 모든 DEFERRABLE 속성의 제약조건을 -- DEFERRED 속성으로 변경합니다.
```

Session이(가) 변경되었습니다.

16> 10, 11 번 스텝을 반복하시오. ALTER SESSION SET CONSTRAINTS 문으로 설정된 속성이 유지되는 것 이외는 동일합니다.

17> SQL*Developer의 HR 접속을 해제하시오.

13-4. 제약조건 연쇄화(Constraints-Cascading).

- 제약조건의 연쇄화는, 특정 작업 시에 종속관계가 있는 정의들을, CASCADE 절이나 CASCADE CONSTRAINTS 절을 이용하여 한 단위로 묶어서 작업을 수행하는 것을 의미합니다. 예를 들어, 기본키 제약조건을 삭제하려고 할 때, 해당 PRIMARY KEY 제약조건을 참조하는 외래키 제약조건을 CASCADE 절을 이용하여, 함께 삭제할 수 있습니다. 간단히, 제약조건과 관련되어 작업 중에 CASCADE절 또는 CASCADE CONSTRAINTS 절을 사용하는 것을 제약조건 연쇄화를 사용한다고 합니다.
- CASCADE절을 사용하는 제약조건 연쇄화는 외래키로 참조되는 또는 기본키 또는 UNIQUE 제약조건을 삭제, 또는 비활성화 시킬 때 사용됩니다.
- CASCADE CONSTRAINT 절을 사용하는 제약조건 연쇄화는 컬럼삭제, 테이블 삭제 시에 사용됩니다.
만약, 컬럼 삭제 시에 CASCADE CONSTRAINTS 절을 사용하면 제약조건 연쇄화에 의하여 다음의 작업이 발생됩니다.
 - 삭제된 컬럼에 정의된 기본키 또는 UNIQUE 제약조건을 참조하는 모든 참조 무결성 제약조건(외래키)도 함께 삭제됩니다.
 - 삭제된 컬럼에 정의된 모든 다른 컬럼 제약 조건을 삭제합니다.
- 다음의 실습을 수행하여, CASCADE CONSTRAINTS 절을 사용하는 제약조건 연쇄화에 대하여 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 실습에 사용할, HR.CASCADE_TEST1 테이블을 다음과 같이 생성합니다.

```
SQL> CREATE TABLE HR.CASCADE_TEST1 (
    COL1 NUMBER PRIMARY KEY,
    COL2 NUMBER,
    COL3 NUMBER,
    COL4 NUMBER,
    FOREIGN KEY (COL2) REFERENCES HR.CASCADE_TEST1(COL1),
    CONSTRAINT CK_COL1_COL3_CHK CHECK (COL1 > 0 AND COL3 > 0),
    CHECK (COL4 > 0));
```

Table HR.CASCADE_TEST1이(가) 생성되었습니다.

☞ HR.CASCADE_TEST1 테이블에 다음과 같은 제약조건들이 정의되어 있습니다.

- COL1에 테이블의 기본키 제약조건이 정의되어 있습니다.
- COL2에 COL1의 기본키 제약조건을 참조하는 외래키 제약조건이 정의되어 있습니다.
- COL1과 COL3 이 체크 제약조건에 의하여 연관되어 있습니다.
- COL4에 체크 제약조건이 정의되어 있습니다.

3> CASCADE절 없이 HR.CASCADE_TEST1 테이블의 기본키 제약조건을 삭제해보시오.

```
SQL> ALTER TABLE HR.CASCADE_TEST1 DROP PRIMARY KEY;
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.CASCADE_TEST1 DROP PRIMARY KEY
```

오류 보고 -

SQL 오류: ORA-02273: 고유/기본 키가 외부 키에 의해 참조되었습니다

02273. 00000 - "this unique/PRIMARY KEY is referenced by some foreign keys"

*Cause: Self-evident.

*Action: Remove all references to the key before the key is to be dropped.

☞ 외래키 제약조건에 의하여 참조되고 있기 때문에 오류가 발생됩니다.

☞ 이 오류는 위의 문장 끝에 CASCADE 절을 명시하면, 제약조건 연쇄화에 의하여, 기본키 제약조건 및 해당 기본키 제약조건을 참조하는 외래키 제약조건을 모두 삭제합니다.

4> CASCADE CONSTRAINT절 없이 HR.CASCADE_TEST1 테이블의 COL1 컬럼을 삭제해보시오.

```
SQL> ALTER TABLE HR.CASCADE_TEST1 DROP (COL1);
```

명령의 1 행에서 시작하는 중 오류 발생 -

```
ALTER TABLE HR.CASCADE_TEST1 DROP (COL1)
```

오류 보고 -

SQL 오류: ORA-12992: 부모 키 열을 삭제할 수 없습니다

12992. 00000 - "cannot drop parent key column"

*Cause: An attempt was made to drop a parent key column.

*Action: Drop all constraints referencing the parent key column, or specify CASCADE CONSTRAINTS in statement.

☞ COL1 삭제 시에, 먼저 COL1에 정의된 기본키 제약조건을 삭제하려고 시도합니다. 이 기본키 제약조건을 외래키 제약조건이 참조하고 있기 때문에 위의 발생됩니다.

☞ 만약, COL1, COL2, COL3을 삭제한다면, CASCADE CONSTRAINTS 절이 없어도 정상적으로 컬럼삭제가 수행됩니다.

```
ALTER TABLE HR.CASCADE_TEST1 DROP (COL1, COL2, COL3);
```

☞ 삭제되는 열에 정의된 제약 조건에서 참조하는 모든 열도 삭제되는 경우 CASCADE CONSTRAINTS는 필요하지 않습니다. 위의 예에서, COL2의 외래키가 삭제되고, COL1과 COL3이 연관된 체크 제약조건이 삭제되고, COL1의 기본키 제약조건이 삭제되면, 의존성이 존재하는 제약조건이 없으므로 COL1, COL2, COL3을 삭제할 수 있습니다.

5> CASCADE CONSTRAINT절을 명시하여, HR.CASCADE_TEST1 테이블의 COL1 컬럼을 삭제하시오.

```
SQL> ALTER TABLE HR.CASCADE_TEST1 DROP (COL1) CASCADE CONSTRAINTS;
```

Table HR.CASCADE_TEST1이(가) 변경되었습니다.

☞ 위의 문장을 실행하면, 제약조건 연쇄화에 의하여 다음의 작업이 수행됩니다.

- COL1에 정의된 기본키 제약조건을 의존하는 외래키 제약조건을 삭제합니다.
- COL1에 정의된 기본키 제약조건을 삭제합니다.
- COL1이 관련된 CK_COL1_COL3_CHK 체크 제약조건을 삭제합니다
- COL1을 삭제합니다.

13-5. USING INDEX 절 사용(9iNF)

- 사용자는 필요한 경우, CREATE INDEX 문을 이용하여, 인덱스-키가 중복될 수 있는 비고유-인덱스를 생성할 수 있습니다.
- 기본키 또는 UNIQUE 제약조건을 정의할 때, 해당 제약조건이 사용할 수 있는 인덱스가 없으면, 오라클 서버에 의하여, 인덱스-키가 중복되지 않는 고유-인덱스가 자동으로 생성되며. 이 때 인덱스의 이름은 제약조건의 이름과 동일합니다.
- 테이블에 기본키 또는 UNIQUE 제약조건을 정의 시에, USING INDEX 절을 이용하여, 사용자가 직접 생성한 인덱스를, 기본키 또는 UNIQUE 제약조건이 사용하도록 지정할 수 있습니다. 이렇게 사용자가 지정한 인덱스를 기본키 또는 UNIQUE 제약조건이 사용하면, 해당 제약조건을 비활성화 및 삭제하더라도 인덱스가 삭제되지 않습니다.
- USING INDEX 절을 사용하여, 다음을 수행할 수 있습니다.
 - 기본키 또는 UNIQUE 제약조건이 정의될 때 자동으로 인덱스가 생성되는 것을 막을 수 있습니다.
즉, 해당 제약조건이 사용할 인덱스를 사용자가 지정할 수 있습니다.
 - 기본키 또는 UNIQUE 제약조건이 정의될 때, 해당 제약조건의 이름과 다르게 인덱스의 이름을 지정할 수 있습니다.
 - 테이블 저장공간과 인덱스 저장공간을 각각 지정할 수 있습니다.

- 다음의 실습을 수행하여, USING INDEX 절을 사용하는 방법을 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 다음의 CREATE TABLE 문으로, HR.EMP_TEST3 테이블을 생성하면서, 기본키 제약조건을 정의될 때, USING INDEX 절의 사용방법을 확인합니다.

```
SQL> CREATE TABLE HR.EMP_TEST3 ( -- CREATE TABLE문 내에서 CREATE INDEX 문을 USING INDEX절에 포함시켜
    EMPNO NUMBER(6),           -- 테이블 생성 시에 인덱스가 함께 생성됩니다.
    ENAME VARCHAR2(60),
    ESAL NUMBER(8,2),          -- DEFERRABLE 절은 USING INDEX 절 앞에 사용해야 합니다.
    EMAIL VARCHAR2(100),
    CONSTRAINT PK_EMPID_EMP_TEST3 PRIMARY KEY (EMPNO) DEFERRABLE
    USING INDEX (
        CREATE INDEX HR.IDX_EMPID_EMP_TEST3_PK -- 사용자가 생성한 인덱스가 됩니다.
        ON HR.EMP_TEST3 (EMPNO)
    );

```

Table HR.EMP_TEST3이(가) 생성되었습니다.

☞ 기본키 제약조건을 정의하면서, USING INDEX 절을 이용하여 CREATE INDEX 문을 포함시켰습니다.

3> HR.EMP_TEST3 테이블에 있는 PK_EMPID_EMP_TEST3 이름의 기본키 제약조건을 비활성화 했다가, 다시 활성화하시오. 활성화 시킬 때, USING INDEX절을 이용하여, 기본키 제약조건이 HR.IDX_EMPID_EMP_TEST3_PK 인덱스를 사용하도록 지정하시오.

```
-- PK_EMPID_EMP_TEST3 제약조건 비활성화
SQL> ALTER TABLE HR.EMP_TEST3
    DISABLE CONSTRAINT PK_EMPID_EMP_TEST3 CASCADE ; -- 기본키 제약조건이 사용하던 인덱스가
                                                    -- 삭제되지 않습니다.
```

Table HR.EMP_TEST3이(가) 변경되었습니다.

```
-- PK_EMPID_EMP_TEST3 제약조건 활성화
```

```
SQL> ALTER TABLE HR.EMP_TEST3
    ENABLE CONSTRAINT PK_EMPID_EMP_TEST3 USING INDEX HR.IDX_EMPID_EMP_TEST3_PK ;
```

Table HR.EMP_TEST3이(가) 변경되었습니다.

☞ 제약조건을 활성화 시킬 때도 USING INDEX 절에 인덱스 이름을 명시하여, 기본키 및 UNIQUE 제약조건이 사용할 인덱스를 지정할 수 있습니다.

4> 다음의 CREATE TABLE 문을 실행하여 HR.EMP_TEST4 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.EMP_TEST4 (
    EMPNO NUMBER(6),
    ENAME VARCHAR2(60),
    ESAL NUMBER(8,2),
    EMAIL VARCHAR2(100)
) ;
```

Table HR.EMP_TEST4(가) 생성되었습니다.

5> HR.EMP_TEST4 테이블의 EMAIL 컬럼을 인덱스-키로 사용하는 HR.IDX_EMAIL_EMP_TEST4 비고유 인덱스를 생성하시오.

```
SQL> CREATE INDEX HR.IDX_EMAIL_EMP_TEST4
    ON HR.EMP_TEST4(EMAIL);
```

Index HR.IDX_EMAIL_EMP_TEST4(가) 생성되었습니다.

6> HR.EMP_TEST4 테이블의 EMAIL 컬럼에 UK_EMAIL_EMP_TEST4 이름의 UNIQUE 제약조건을 추가하시오. 단, 추가되는 UNIQUE 제약조건이 HR.IDX_EMAIL_EMP_TEST4 비고유 인덱스를 사용하도록 지정하시오.

```
SQL> ALTER TABLE HR.EMP_TEST4
    ADD CONSTRAINT UK_EMAIL_EMP_TEST4 UNIQUE(EMAIL) DEFERRABLE
        USING INDEX HR.IDX_EMAIL_EMP_TEST4;
```

Table HR.EMP_TEST4(가) 변경되었습니다.

☞ 기본키 및 UNIQUE 제약조건을 추가할 때, USING INDEX 절을 이용하여, 해당 제약조건이 사용할 인덱스를 지정할 수 있습니다.

7> HR.EMP_TEST4 테이블의 EMPNO 컬럼에 PK_EMPNO_EMP_TEST4 이름의 기본키 제약조건을 추가하시오.

단 HR.IDX_EMPNO_EMP_TEST4_PK라는 비고유 인덱스 생성하고, 제약조건이 이 인덱스를 사용하도록 지정하시오.

```
SQL> ALTER TABLE HR.EMP_TEST4
    ADD CONSTRAINT PK_EMPNO_EMP_TEST4 PRIMARY KEY (EMPNO)DEFERRABLE
        USING INDEX (CREATE INDEX HR.IDX_EMPNO_EMP_TEST4_PK
            ON HR.EMP_TEST4(EMPNO));
```

Table HR.EMP_TEST4(가) 변경되었습니다.

☞ 기본키 및 UNIQUE 제약조건을 테이블에 추가할 때, USING INDEX 절에 CREATE INDEX 문을 명시하여, 사용자가 원하는 인덱스를 생성하고, 해당 제약조건이 생성된 인덱스를 사용하도록 할 수 있습니다.

13-6. 함수 기반 인덱스(FUNCTION-BASED INDEX) 사용하기.

- 함수 기반 인덱스는 인덱스-키로, 컬럼 대신, 컬럼에 대한 표현식을 사용하는 인덱스입니다.
- 생성된 함수 기반 인덱스가 SQL문 처리 시에 사용되려면, QUERY_REWRITE_ENABLED 초기화 파라미터가 TRUE(디폴트:TRUE)로 설정되어야 합니다.
- 다음의 실습을 수행하여 함수 기반 인덱스를 생성하는 방법을 학습합니다.

1> SQL*Developer를 이용하여 데이터베이스 서버에 HR 계정으로 접속합니다.

2> HR.EMPLOYEES 테이블의 LAST_NAME 컬럼에 대하여 UPPER(LAST_NAME) 표현식을 인덱스 키로 사용하는 LNAME_EMP_FUNC_IDX 이름의 함수 기반 인덱스를 생성합니다.

```
SQL> CREATE INDEX HR.LNAME_EMP_FUNC_IDX ON HR.EMPLOYEES(UPPER(LAST_NAME)) ;
```

Index HR.LNAME_EMP_FUC_IDX이(가) 생성되었습니다. -- 인덱스 키로 UPPER(LAST_NAME) 표현식이 사용되었습니다.

☞ 다음의 SQL문을 비교합니다.

문1>	<code>SELECT * FROM HR.EMPLOYEES WHERE LAST_NAME='King' ;</code>
문2>	<code>SELECT * FROM HR.EMPLOYEES WHERE UPPER(LAST_NAME)='KING' ;</code>

☞ 문1과 문2 중, 문2가 처리될 때, 생성된 함수 기반 인덱스가 사용됩니다.

많은 SQL문의 WHERE절에 UPPER(LAST_NAME) 표현식이 사용된다면, 행이 빠르게 선택될 수 있도록, UPPER(LAST_NAME) 표현식을 인덱스-키로 가지는 함수 기반 인덱스를 사용할 수 있습니다.

☞ 다음은 문2가 처리될 때, 선택된 실행계획을 표시한 것입니다.

Execution Plan

Plan hash value: 3914982809

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT		1	69	2 (0) 00:00:01	
1 TABLE ACCESS BY INDEX ROWID EMPLOYEES	1	69	2 (0) 00:00:01		
* 2 INDEX RANGE SCAN	LNAME_EMP_FUNC_IDX	1	1 (0) 00:00:01		

Predicate Information (identified by operation id):

2 - access(UPPER("LAST_NAME")='KING')

[참고] 표현식 통계정보 수집(11gNF)

○ 함수 기반 인덱스가 SQL문 처리 시에 실행계획 상에서 적절히 사용될 수 있도록, 함수 기반 인덱스의 인덱스-키로 사용된, 컬럼 표현식에 대한 옵티マイ저-통계정보를 수집하는 기능이 오라클 11g 버전부터 제공됩니다.

○ 위에서 생성한 HR.LNAME_EMP_FUNC_IDX 인덱스의 인덱스-키로 사용된 UPPER(LAST_NAME) 표현식에 대한 옵티マイ저-통계정보를 수집하려면, DBMS_STATS.GATHER_TABLE_STATS 내장패키지를 다음처럼 실행합니다.

-- UPPER(LAST_NAME) 표현식에 대한 옵티マイ저 통계정보를 수집합니다.

```
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS('HR','EMPLOYEES',METHOD_OPT=>'FOR COLUMNS (UPPER(LAST_NAME))')
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

-- 수집된 UPPER(LAST_NAME) 표현식에 대한 옵티マイ저 통계정보를 기반으로,

UPPER(LAST_NAME)을 인덱스-키로 사용하는 HR.LNAME_EMP_FUNC_IDX 인덱스의 옵티マイ저 통계정보를 수집합니다.

```
SQL> EXEC DBMS_STATS.GATHER_INDEX_STATS('HR','LNAME_EMP_FUNC_IDX')
```

PL/SQL 프로시저가 성공적으로 완료되었습니다.

3> 생성한 LNAME_EMP_FUNC_IDX 인덱스를 삭제합니다.

```
SQL> DROP INDEX HR.LNAME_EMP_FUNC_IDX;
```

Index HR.LNAME_EMP_FUNC_IDX이(가) 삭제되었습니다.

■ 기본문법: 인덱스 삭제

```
DROP INDEX 인덱스이름 ;
```

○ 인덱스를 삭제하려면 인덱스의 소유자이거나 DROP ANY INDEX 시스템권한이 있어야 합니다.

○ 인덱스의 인덱스-키 구성을 수정할 수 없습니다. 인덱스의 인덱스-키 구성을 변경하려면, 인덱스를 삭제한 다음, 원하는 인덱스-키 구성으로 다시 인덱스를 생성해야 합니다.

○ 테이블을 삭제하면, 해당 테이블에 종속적인 인덱스, 제약 조건 및 트리거는 자동으로 삭제되지만 뷰와 시퀀스는 남아 있습니다.

○ 기본키 및 UNIQUE 제약조건이 사용 중인 인덱스는 삭제할 수 없습니다.

13-7 임시 테이블(GLOBAL TEMPORARY TABLE)

- 세션이 종료되거나 또는 트랜잭션이 종료될 때 자동으로 입력된 데이터가 데이터베이스 서버 상에서 TRUNCATE 되어 내용이 사라지는 테이블입니다.
- 임시 테이블은 온라인 쇼핑몰의 카트와 같은 시나리오에서 유용하게 사용될 수 있습니다.
즉, 온라인 응용 프로그램의 쇼핑 카트의 각 항목은 임시 테이블의 한 행으로 표현될 수 있습니다. 온라인 상점에서 쇼핑하는 동안 카트에서 계속 항목을 추가하거나 제거할 수 있습니다. 세션에서 이 작업이 수행되는 동안, 이 카트 데이터는 해당 세션에만 국한된 데이터입니다. 상품에 대하여 결제한 후에는 선택한 카트의 행이 영구 테이블로 이동됩니다. 쇼핑을 마치고 세션이 끝날 때, 임시 테이블에 있는 카트의 데이터가 자동으로 삭제됩니다.
- 동일한 테이블 정의를 공유하여 사용하지만, 각각의 세션에 고유한 데이터를 입력하여 작업할 수 있습니다.
- 각 세션 별로, 서버프로세스의 메모리 영역인 PGA 와 임시 저장영역(임시 세그먼트)를 이용해서 데이터가 유지되며, 다른 테이블처럼 물리적인 영구 저장영역이 구성되지 않습니다.
- 임시 테이블에 근거한 인덱스, 뷰, 트리거 객체를 생성할 수 있습니다.
- 다음의 실습을 수행하여 임시테이블을 생성하는 방법을 학습하고, 임시테이블의 작동방식을 이해합니다.

1> SQL*Developer를 이용하여 데이터베이스 서버에 HR 계정으로 접속합니다.

2> 다음의 CREATE 문을 실행하여 HR.EMP_TEMP01 임시테이블을 생성합니다.

```
SQL> CREATE GLOBAL TEMPORARY TABLE HR.EMP_TEMP01
  (EMPNO NUMBER(6) PRIMARY KEY,
   EMPNAME VARCHAR2(60) NOT NULL,
   EMPSAL NUMBER(8,2)
  ) ON COMMIT DELETE ROWS ; -- 트랜잭션이 종료될 때, 내용을 지우는 임시테이블
```

Global temporary TABLE이(가) 생성되었습니다.

- 기본문법: 트랜잭션이 종료될 때, 내용을 지우는 임시테이블

CREATE GLOBAL TEMPORARY TABLE 스키마이름.테이블이름 -- CREATE GLOBAL TEMPORARY TABLE 문을 사용합니다. (컬럼이름 데이터유형, ...) ON COMMIT DELETE ROWS ;

- ON COMMIT DELETE ROWS 절을 사용하면, 임시테이블에 입력된 사용자 데이터는, 트랜잭션 종료 시에 자동으로 TRUNCATE 되어 사라집니다.

3> 생성된 HR.EMP_TEST01 임시 테이블에 하위질의를 이용하여 HR.EMPLOYEES 테이블의 데이터 일부를 입력합니다.

```
SQL> INSERT INTO hr.emp_temp01
      SELECT employee_id, last_name, salary
        FROM  hr.employees
       WHERE  department_id = 10;
```

1 행 이(가) 삽입되었습니다.

4> HR.EMP_TEMP01 임시테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_temp01 ;
```

EMPNO	EMPNAME	EMPSAL
200	Whalen	4400

☞ 이 세션에서만 사용되는 데이터입니다.

5> COMMIT을 수행하여 데이터 입력 트랜잭션을 종료합니다.

```
SQL> COMMIT ;
```

커밋 완료.

6> 다시 HR.EMP_TEMP01 임시테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_temp01 ;
```

선택된 행 없음

☞ 트랜잭션이 종료됨과 동시에 임시로 저장된 데이터가 TRUNCATE 되었기 때문에 표시되는 데이터가 없습니다.

7> 다음의 CREATE 문을 실행하여 HR.EMP_TEMP02 임시테이블을 생성합니다.

```
SQL> CREATE GLOBAL TEMPORARY TABLE hr.emp_temp02
  (empno NUMBER(6) PRIMARY KEY,
  empname VARCHAR2(60) NOT NULL,
  empsal NUMBER(8,2)
  ) ON COMMIT PRESERVE ROWS ;           — 트랜잭션이 종료될 때, 내용을 유지되는 임시테이블
                                         -- 단, 세션이 종료되면, 내용이 삭제됩니다.
Global temporary TABLE이(가) 생성되었습니다.
```

■ 기본문법: 트랜잭션이 종료될 때, 내용을 지우는 임시테이블

```
CREATE GLOBAL TEMPORARY TABLE 스키마이름.테이블이름 -- CREATE GLOBAL TEMPORARY TABLE 문을 사용합니다.
(컬럼이름 데이터유형, ...)
ON COMMIT PRESERVE ROWS ;
```

- ON COMMIT PRESERVE ROWS 절을 사용하면, 임시테이블에 입력된 사용자 데이터는 트랜잭션 종료 시에는 계속 유지됩니다. 단, 사용자가 접속을 종료(세션 종료)하면, 입력된 내용이 자동으로 TRUNCATE 되어 사라집니다.

8> 생성된 HR.EMP_TEST02 임시 테이블에 하위질의를 이용하여, HR.EMPLOYEES 테이블의 데이터 일부를 입력합니다.

```
SQL> INSERT INTO hr.emp_temp02
  SELECT employee_id, last_name, salary
  FROM   hr.employees
  WHERE  department_id = 20;
```

2개 행 이(가) 삽입되었습니다.

9> HR.EMP_TEMP02 임시테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_temp02 ;
```

EMPNO	EMPNAME	EMPSAL
201	Hartstein	13000
202	Fay	6000

☞ 이 세션에서만 사용되는 데이터입니다.

10> COMMIT을 수행하여 데이터 입력 트랜잭션을 종료합니다.

```
SQL> COMMIT ;
```

커밋 완료.

11> 커밋 후, 다시 HR.EMP_TEMP02 임시테이블에 입력된 데이터를 확인합니다.

```
SQL> SELECT * FROM hr.emp_temp02 ;
```

EMPNO	EMPNAME	EMPSAL
201	Hartstein	13000
202	Fay	6000

☞ 이 세션에서만 사용되는 데이터입니다.

☞ 트랜잭션이 종료되어도, 임시 테이블에 저장된 데이터가 유지됩니다.

☞ 사용자가 접속을 종료하기 전까지 임시 테이블의 데이터는 계속 유지됩니다.

[참고] 임시 테이블에 대하여 임시 저장공간(임시 테이블스페이스) 지정(11gNF) 지정.

- 오라클 10g 버전까지, 계정에게 설정된 임시 테이블스페이스(저장공간)을 사용하여, 임시 테이블의 데이터가 처리 중에 임시로 저장되었습니다. 따라서, 계정이 다수의 임시 테이블을 사용할 경우, 모든 임시 테이블들이 동일한 임시 저장공간을 사용할 수 밖에 없었습니다.
- 오라클 11g 버전부터, 임시 테이블마다 고유한 임시 테이블스페이스(저장공간)를 지정할 수 있게 개선 되었습니다.

■ 기본문법

```
CREATE GLOBAL TEMPORARY TABLE 스키마이름.테이블이름
(컬럼이름 데이터유형, ...)
ON COMMIT DELETE ROWS
TABLESPACE 임시테이블스페이스이름 : -- 오직 임시테이블스페이스 이름만 명시할 수 있습니다.
```

```
CREATE GLOBAL TEMPORARY TABLE 스키마이름.테이블이름
(컬럼이름 데이터유형, ...)
ON COMMIT PRESERVE ROWS
TABLESPACE 임시테이블스페이스이름 : -- 오직 임시테이블스페이스 이름만 명시할 수 있습니다.
```

13-8 오라클 FLASHBACK-DROP 기능을 이용하여 삭제된 테이블 복구 (10gNF).

- 10g 버전부터 테이블을 삭제(DROP)하면, 테이블과 관련된 시스템 정의가 삭제되지 않고 RECYCLEBIN 정보로서 유지되며, 이렇게 유지된 RECYCLEBIN 정보를 기반한 FLASHBACK-DROP 기능을 이용하여 쉽게 복구할 수 있습니다.
- RECYCLEBIN은 실제로는 삭제된 객체에 대한 정보가 담긴 데이터 딕셔너리 테이블입니다. 삭제된 테이블 및 삭제된 테이블에 정의되었던 인덱스 및 제약 조건 및 중첩 테이블 등과 같은 모든 종속 객체들의 정보가 RECYCLEBIN에서 유지되며, 또한 삭제된 테이블 및 종속된 인덱스의 저장공간(세그먼트)도 삭제되지 않고 유지됩니다.
- 삭제된 테이블의 저장공간이, 다른 테이블에 의하여 사용되면, 삭제된 테이블을 복원될 수 없습니다.
- 데이터베이스 관리자가 데이터베이스 계정을 삭제할 때, 같이 삭제되는, 해당 계정에 속한 모든 객체는 RECYCLEBIN에 저장되지 않습니다. 또한 계정 삭제 전에 해당 계정과 관련된 RECYCLEBIN 정보도 모두 삭제됩니다.
- 다음의 실습을 수행하여, FLASHBACK-DROP 기능의 사용방법과 관련 SQL문을 학습합니다.

1> SQL*Developer를 이용하여, 오라클 데이터베이스 서버에 HR계정으로 접속합니다.

2> 이전 실습에서 실행된 DROP TABLE 문으로 저장되어 있는 기존 휴지통 정보를 PURGE RECYCLEBIN 문으로 삭제합니다.

SQL> **PURGE RECYCLEBIN;** -- SQL문이므로 반드시 뒤에 세미콜론(:)을 명시해야 합니다.

PURGE RECYCLEBIN

☞ HR 계정이 접속하여 실행했으므로, HR 계정의 RECYCLEBIN 정보만 모두 삭제됩니다.

3> 앞의 USING INDEX 절 실습 시(30페이지)에 생성한 HR.EMP_TEST4 테이블을 삭제합니다.

SQL> **DROP TABLE HR.EMP_TEST4;** -- PURGE 키워드가 명시되어 있지 않습니다.

Table HR.EMP_TEST4(가) 삭제되었습니다. -- RECYCLEBIN 정보로 저장됩니다.

4> 삭제된 HR.EMP_TEST4 테이블의 휴지통(RECYCLEBIN) 정보를 확인합니다.

SQL> **SELECT ORIGINAL_NAME, OPERATION, DROPTIME, CAN_UNDROP FROM RECYCLEBIN ;**

ORIGINAL_NAME	OPERATION	DROPTIME	CAN
IDX_EMAIL_EMP_TEST4	DROP	2012-01-21:00:55:56	NO --테이블 삭제 시에 같이 삭제된 인덱스 정보
IDX_EMPNO_EMP_TEST4_PK	DROP	2012-01-21:00:55:56	NO --테이블 삭제 시에 같이 삭제된 인덱스 정보
EMP_TEST4	DROP	2012-01-21:00:55:56	YES --삭제된 테이블 정보

[참고] SHOW RECYCLEBIN SQL*Developer 명령어를 이용하여 삭제된 테이블 정보를 확인할 수도 있습니다.

SQL> **SHOW RECYCLEBIN** -- SQL*Developer 명령어이므로 뒤에 세미콜론(;)을 명시하지 않아도 됩니다.

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT TYPE	DROP TIME
IDX_EMAIL_EMP_TEST4	BIN\$KGZ3H4lw6ongUKjAbgoUfQ==\$0	INDEX	2012-01-21:00:55:56
IDX_EMPNO_EMP_TEST4_PK	BIN\$KGZ3H4lx6ongUKjAbgoUfQ==\$0	INDEX	2012-01-21:00:55:56
EMP_TEST4	BIN\$KGZ3H4ly6ongUKjAbgoUfQ==\$0	TABLE	2012-01-21:00:55:56

☞ CMD-창이나, 터미널에서 접속한 세션에서는 테이블 정보만 표시됩니다. 오라클 SQL*Developer에서는, 삭제된 테이블과 테이블에 종속된 인덱스 정보가 같이 표시됩니다.

5> 삭제된 HR.EMP_TEST4 테이블을 FLASHBACK-DROP 기능을 이용하여 복구합니다.

SQL> **FLASHBACK TABLE HR.EMP_TEST4 TO BEFORE DROP;** -- 삭제된 테이블이 복원됩니다.

Flashback을(를) 성공했습니다.

6> DESCRIBE 명령어로 HR.EMP_TEST4 테이블의 컬럼 정보를 확인하여, 테이블이 복구되었는지 확인합니다.

SQL> **DESC hr.emp_test4** -- 복원된 HR.EMP_TEST4 테이블의 컬럼 정보가 표시됩니다.

이름	널	유형
EMPNO		NUMBER(6)
ENAME		VARCHAR2(60)
ESAL		NUMBER(8,2)
EMAIL		VARCHAR2(100)

7> HR.EMP_TEST4 테이블을 PURGE절을 명시하여 다시 삭제합니다.

SQL> **DROP TABLE HR.EMP_TEST4 PURGE;** -- PURGE 키워드에 의하여 HR.EMP_TEST4 테이블의 시스템 정보가
-- RECYCLEBIN에 관리되지 않고 바로 삭제됩니다.

Table HR.EMP_TEST4(가) 삭제되었습니다.

☞ DROP TABLE 문에 PURGE절을 사용하면, 삭제되는 테이블의 시스템 정보가 RECYCLEBIN 정보로 관리되지 않고, 바로 삭제됩니다.

8> 삭제된 HR.EMP_TEST4 테이블의 휴지통(RECYCLEBIN) 정보를 확인합니다.

SQL> **SELECT ORIGINAL_NAME, OPERATION, DROPTIME, CAN_UNDROP**
FROM RECYCLEBIN; -- 삭제된 테이블의 시스템 정보가 휴지통 정보로 관리되지 않고,
-- 바로 삭제되었으므로 표시되는 내용이 없습니다.

선택된 행 없음

14 DATA DICTIONARY VIEW■ 이용한 객체 정보 확인.**◆ 학습 목표.**

- 데이터 딕셔너리 뷰에 대하여 소개합니다.
- 다음과 같은, 기본적인 데이터 딕셔너리 뷰를 조회하는 방법을 학습합니다.
 - USER_OBJECT
 - USER_TABLES
 - USER_TAB_COLUMNS
 - USER_CONSTRAINTS
 - USER_CONS_COLUMNS
 - USER_VIEWS
 - USER_SEQUENCES
 - USER_INDEXES
 - USER_IND_COLUMNS
 - USER_SYNONYMS
 - ALL_SYNONYMS
 - NLS_SESSION_PARAMETERS
 - NLS_DATABASE_PARAMETERS
 - SYSTEM_PRIVILEGE_MAP
 - USER_SYS_PRIVS
 - USER_ROLE_PRIVS
 - ROLE_SYS_PRIVS
 - ROLE_ROLE_PRIVS
 - ROLE_TAB_PRIVS
 - USER_TAB_PRIVS_REC
 - USER_COL_PRIVS_REC
 - USER_TAB_PRIVS_MADE
 - SESSION_PRIVS
 - SESSION_ROLES
 - USER_TAB_COMMENTS
 - USER_COL_COMMENTS

☞ 이 단원에서 실습 시에 표시되는 결과는, 이전 단원의 실습유무에 따라 다를 수 있습니다. 교재에 표시된 결과보다 해당 결과를 표시하는 데이터 딕셔너리 뷰에 대한 SELECT문을 주의해서 학습합니다.

14-1. DATA DICTIONARY 개념.

■ 데이터베이스에 접속한 사용자가 다음과 같은 데이터베이스의 시스템 정보를 알고 싶을 때 어떻게 하면 될까요?

- 현재 접속한 계정이 사용할 수 있는 테이블 또는 뷰는 어떤 것들이 있을까?
- 특정 뷰에 정의된 SELECT 문의 내용은 무엇일까?
- 테이블의 컬럼에 설정된 디폴트 값은 무엇일까?
- 테이블에 정의된 제약조건과 인덱스는 어떤 것이 있을까?
- 현재 접속한 계정이 사용할 수 있는 권한은 무엇일까?
- 기타 등등...

■ 오라클 데이터베이스 서버에 접속한 사용자들이 위에서 언급한 오라클 데이터베이스 내에 정의된 시스템 정보가 필요할 때, 사용하는 객체가 데이터 딕셔너리(**DATA-DICTIONARY**)입니다.

■ 이러한 데이터 딕셔너리 객체들은, **SYS** 스키마 내에 구성되어 있으며, 데이터베이스가 구성될 때 자동으로 생성되는 데이터 딕셔너리 테이블과 별도의 스크립트를 실행하여 생성되는 데이터 딕셔너리 뷰 및 관련 동의어(**SYNONYM**)들로 구성되어 있습니다.

■ 사용자들은 데이터 딕셔너리 뷰에 대한 동의어를 사용하는 것이지만, 일반적으로 데이터 딕셔너리 뷰를 사용한다고 합니다. 필요한 경우, 오라클 데이터베이스 사용자들은 데이터 딕셔너리 뷰를 이용하여 필요한 시스템 정보를 확인할 수 있습니다.

☞ 일반적으로 데이터 딕셔너리 테이블에 저장된 정보는, 사용자가 요청한 SQL문을 처리하는 서버프로세스가, SQL문에 명시된 이름(컬럼이름, 테이블이름, 스키마이름)들을 확인할 때 사용되며, 이 확인 작업을 빠르게 수행하기 위하여, 데이터 딕셔너리 테이블에 저장된 데이터는 코드화된 형태로 저장되어 있습니다.

14-2. 데이터 딕셔너리 뷰 구조 (데이터 딕셔너리 뷰의 이름 체계).

14-2-1. STATIC DATA DICTIONARY VIEW

```

SELECT owner, table_name
FROM all_tables
WHERE owner IN ('          ');
SELECT owner, table_name
FROM all_tables
WHERE owner IN ('HR','OE','PM','SH');

```

- STATIC DATA DICTIONARY VIEW를 통해서 제공되는 시스템 정보는,

디스크상에 특정 데이터 딕셔너리 테이블에 저장되어 있습니다.

- 사용자들은 아래와 같은 형식의 접두어로 시작하는 뷰를 이용하여 필요한 정보를 조회할 수 있습니다.

접두어	설명
USER_TABLES	현재 로그인한 계정(Current User)이 소유한 테이블 관련 시스템 정보를 제공합니다.
ALL_TABLES	현재 로그인한 계정이 액세스 할 수 있는 테이블 관련 시스템 정보를 제공합니다. (접속한 계정이 소유한 것 + 다른 계정이 소유한 객체의 객체권한이 부여된 것)
DBA_TABLES	데이터베이스 전체에 존재하는 모든 테이블의 시스템 정보를 제공합니다. DBA(SYS, SYSTEM계정)만 조회할 수 있습니다.

☞ 테이블과 관련된 시스템 정보를 제공하는 데이터 딕셔너리 뷰를 예를 들어 접두어를 설명합니다.

14-2-2. DYNAMIC PERFORMANCE VIEW (V\$-VIEW).

- DYNAMIC PERFORMANCE VIEW를 통해서 제공되는 시스템 데이터 정보는,

메모리에만 존재합니다. 즉, 저장되는 정보가 아닙니다.

오라클 데이터베이스 서버에 의해서 실시간으로 변경되기 때문에 저장되지 않습니다.

- 오라클 데이터베이스 서버가 기동된 이 후 현재까지, 성능 관련 누적 데이터 및 현재 OPEN 상태로 운영중인

오라클 데이터베이스 서버의 서비스 상태(현재 접속한 계정이 누구 ? 등등)를 나타내는 시스템 정보를 제공합니다.

- DYNAMIC PERFORMANCE VIEW는, V\$DATABASE, V\$SESSION, V\$INSTANCE, V\$SYSSTAT, V\$SYSTEM_EVENT 와 같이

대부분 V\$로 접두어로 시작됩니다.

☞ 본 단원에서는 USER_ 접두어로 시작하는 데이터 딕셔너리 뷰를 사용하여, 시스템 정보를 확인하는 방법을 설명합니다.

☞ 본 단원의 실습 시에, SQL*Developer를 이용하여, HR 계정으로 접속하여 수행하십시오.

다른 계정으로 접속하는 경우에는, 내용 중에 명시하겠습니다.

[참고] DICTIONARY 데이터 딕셔너리 뷰.

- DICTIONARY 데이터 딕셔너리 뷰는 오라클 데이터베이스에 있는 데이터 딕셔너리 뷰들의 이름 및 뷰에 대한 간단한 설명을 제공합니다. 주로 데이터 딕셔너리 뷰의 이름을 확인할 때 사용됩니다.

1> SQL*Developer를 이용하여, HR 계정으로 데이터베이스에 접속합니다.

2> DICTIONARY 데이터 딕셔너리 뷰에 구성된 컬럼 정보를 확인하시오.

```
SQL> DESC dictionary
```

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

3> 현재 접속한 사용자가 사용할 수 있는 동의어(SYNONYM) 정보를 제공하는 데이터 딕셔너리 뷰의 이름을 확인하시오.

```
SQL> SELECT table_name
  FROM dictionary
 WHERE table_name LIKE '%SYNONYM%' ; --비교되는 상수값이 대문자 입니다.

TABLE_NAME
-----
ALL_SYNONYMS
USER_SYNONYMS
```

☞ 데이터 딕셔너리에 저장된 시스템 데이터 중, 이름 데이터(테이블이름, 컬럼이름, 객체이름,...)는 모두 "대문자"입니다.

☞ HR 계정으로 접속해서 DICTIONARY 뷰에 조회했으므로, HR 계정이 조회할 수 있는 데이터 딕셔너리 뷰 이름만 표시됩니다. DBA_SYNONYMS는 HR 계정은 조회할 수 있는 권한이 없기 때문에 표시되지 않습니다.

4> USER_OBJECT 데이터 딕셔너리 뷰가 제공하는 정보에 대한 요약된 내용을 확인하시오.

```
SQL> SELECT *
  FROM dictionary
 WHERE table_name = 'USER_OBJECTS' ; --비교되는 상수값이 대문자 입니다.

TABLE_NAME      COMMENTS
-----
USER_OBJECTS    Objects owned by the user
```

- ☞ 앞의 쿼리들에서 데이터 딕셔너리 뷰와 관련된 내용을 조회할 때 조건절 작성 시에 소문자로 작성하면 안됩니다. 아래처럼 아무것도 내용이 표시되지 않습니다.

```
SQL> SELECT *
  FROM dictionary
 WHERE table_name = 'user_objects'; --비교되는 상수값이 소문자입니다.

no rows selected
```

[참고] 오라클 DOCUMENTS 사이트에서 제공하는 매뉴얼 활용하기.

- 인터넷을 통해 오라클 사가 제공하는 Documentation(Manual)을 온라인으로 활용할 수 있습니다.
이들 문서들 중에 REFERENCE 메뉴얼에 DATA DICTIONARY VIEW에 관한 정보를 확인할 수 있습니다.

- 온라인 오라클 Documentation 사용하는 방법.

- 1> 웹-브라우저를 이용하여 한국 오라클 홈페이지 [<http://www.oracle.com/kr/index.html>]에 접속합니다.
 - 2> 상단의 [지원(Support)]메뉴에 마우스 위치시킨 후, 표시된 내용에서 [Product Documentation] 링크를 클릭합니다.
 - 3> 표시된 [Oracle Help Center]페이지에서 [Database] 항목을 클릭합니다.
[Oracle Help Center의 Oracle Database Documentation] 페이지가 표시됩니다.
 - 4> [Oracle Database Documentation] 페이지에서 [Oracle Database] 섹션의 밑에 있는
[All Oracle Database documentation] 링크를 클릭합니다.
[Oracle Database] 페이지가 표시됩니다.
 - 5> [Oracle Database] 탭 페이지에서, 원하는 버전(예, Oracle Database 11g Release 2(11.2))을 선택한 후,
약 5초 후에 [Common Task]섹션에서 [Get Started] 링크를 클릭합니다.
[주의] 링크를 너무 빨리 클릭하면 12.1 버전의 [Get Started] 페이지가 표시될 수 있습니다.
 - 6> [Oracle Database Online Documentation 11g Release 2 (11.2) Getting Started] 페이지가 표시됩니다.
원편에는 매뉴얼들을 주제별로 분류한 이름이 표시됩니다.
페이지의 중간에 있는 [Supporting Documentation] 섹션에서 Reference 링크를 클릭합니다.
 - 7> 11gR2 버전의 [Database Reference] 매뉴얼 페이지가 표시됩니다.
데이터 딕셔너리 뷰들과 구성된 컬럼들이 제공하는 정보에 대하여 확인할 수 있습니다.
- ☞ 표시된 페이지에서 원하는 항목의 매뉴얼 제목을 클릭하여 참조자료로 사용합니다.

14-3. 계정이 소유한 모든 객체의 요약 정보 확인!

- **USER_OBJECTS** 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 객체의 이름, 객체 유형, 현재 상태 정보를 확인할 수 있습니다.

- **USER_OBJECTS**: 데이터 딕셔너리 뷰의 대표적인 컬럼이름 및 제공되는 정보

컬럼이름	제공되는 정보
OBJECT_NAME	객체 이름이 표시됩니다.
OBJECT_ID	객체의 딕셔너리 객체 번호가 표시됩니다.
OBJECT_TYPE	객체의 유형(TABLE, VIEW, INDEX, SEQUENCE 등)이 표시됩니다.
CREATED	객체가 생성된 날짜와 시간이 표시됩니다.
LAST_DDL_TIME	객체에 대하여 수행된 마지막 DDL문에 의하여 수정된 시간이 표시됩니다.
STATUS	객체의 상태(VALID, INVALID 또는 N/A)가 표시됩니다.
GENERATED	객체의 이름을 시스템이 생성했는지 여부 (Y N)가 표시됩니다.

1> HR 계정으로 접속한 세션에서 HR이 소유한 객체들의 이름 및 타입과 상태를 확인하시오.

```
SQL> SELECT object_name, object_type, status
  FROM user_objects
  ORDER BY object_type ;
```

OBJECT_NAME	OBJECT_TYPE	STATUS
...		
LOCATIONS_SEQ	SEQUENCE	VALID
DEPARTMENTS_SEQ	SEQUENCE	VALID
EMPLOYEES_SEQ	SEQUENCE	VALID
EMP_SYN	SYNONYM	VALID
JOB_HISTORY	TABLE	VALID
COPY_EMP	TABLE	VALID
REGIONS	TABLE	VALID
JOB_GRADES	TABLE	VALID
SALES_REPS	TABLE	VALID
JOB	TABLE	VALID
EMPLOYEES	TABLE	VALID
DEPARTMENTS	TABLE	VALID
COUNTRIES	TABLE	VALID
LOCATIONS	TABLE	VALID
SECURE_EMPLOYEES	TRIGGER	VALID
UPDATE_JOB_HISTORY	TRIGGER	-- VALID: 객체가 사용 가능한 정상적인 상태
EMP_DETAILS_VIEW	VIEW	-- INVALID: 객체가 사용 할 수 없는 상태

42개의 행이 선택됨

```
[DBA]
SELECT owner, object_name, object_type
FROM dba_objects
WHERE status ='INVALID';
```

14-4. 테이블 정보.

- USER_TABLES 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 테이블에 대한 설정 정보를 확인할 수 있습니다.

1> HR 계정으로 접속한 세션에서 HR 계정이 소유한 테이블의 이름을 확인하시오.

```
SQL> SELECT table_name FROM user_tables ;
```

TABLE_NAME
JOBS
JOB_HISTORY
LOCATIONS
EMPLOYEES
COUNTRIES
SALES_REPS
COPY_EMP
JOB_GRADES
DEPARTMENTS
REGIONS

```
JOBS
JOB_HISTORY
LOCATIONS
EMPLOYEES
COUNTRIES
SALES_REPS
COPY_EMP
JOB_GRADES
DEPARTMENTS
REGIONS
```

10개의 행이 선택됨 -- 실습 내용에 따라, 표시되는 테이블이름이 다를 수 있습니다.

- ☞ USER_TABLES 데이터 딕셔너리 뷰에 대하여 TABS 동의어를 사용할 수 있습니다.

[참고] TAB 동의어

```
SQL> SELECT * FROM tab ORDER BY 2, 1;
```

TNAME	TABTYPE	CLUSTERID
SYN1	SYNONYM	
COUNTRIES	TABLE	
DEPARTMENTS	TABLE	
EMPLOYEES	TABLE	
JOBS	TABLE	
JOB_GRADES	TABLE	
JOB_HISTORY	TABLE	
LOCATIONS	TABLE	
REGIONS	TABLE	
EMP_DETAILS_VIEW	VIEW	

- ☞ TAB 동의어를 사용하여, 현재 접속한 계정이 소유한 TABLE, VIEW, SYNONYM 의 이름 및 객체유형 정보를 확인할 수 있습니다.

14-5 테이블의 컬럼 정보.

- USER_TAB_COLUMNS 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 테이블을 구성하는 각 컬럼의 설정 정보를 확인할 수 있습니다(특히, DATA_DEFAULT).

1> HR 계정으로 접속한 세션에서, 다음의 CREATE TABLE 문장을 실행하여 HR.EMPS_APP_TEST11 테이블을 생성합니다.

```
SQL> CREATE TABLE HR.EMPS_APP_TEST11 (
    EMPID NUMBER(4),
    EMPNAME VARCHAR2(30) CONSTRAINT NN_ENAME_EMPS_APP_TEST11 NOT NULL,
    ADDRESS VARCHAR2(30) DEFAULT 'SEOUL',      -- DEFAULT 값이 설정되었습니다.
    SALARY NUMBER(8) NULL,
    EMAIL VARCHAR2(100) NOT NULL,
    DEPTID NUMBER(4),
    HIREDATE DATE DEFAULT SYSDATE,      -- DEFAULT 값이 설정되었습니다.
    CONSTRAINT PK_EMPID_EMPS_APP_TEST11 PRIMARY KEY(EMPID),
    CONSTRAINT UK_EMAIL_EMPS_APP_TEST11 UNIQUE(EMAIL),
    CONSTRAINT CK_SAL_EMPS_APP_TEST11 CHECK(SALARY > 0 AND ADDRESS IS NOT NULL),
    CONSTRAINT FK_DEPTID_EMPS_APP_TEST11 FOREIGN KEY(DEPTID)
        REFERENCES HR.DEPARTMENTS(DEPARTMENT_ID) ON DELETE SET NULL );
```

Table HR.EMPS_APP_TEST11이(가) 생성되었습니다.

☞ 일부 컬럼에 대하여 DEFAULT 값이 설정되어 있습니다.

2> HR.EMPS_APP_TEST11 테이블의 컬럼에 설정된 DEFAULT 값을 확인하시오.

```
SQL> SELECT column_name, data_default
  FROM user_tab_columns
 WHERE table_name = 'EMPS_APP_TEST11' AND DATA_DEFAULT IS NOT NULL ;
```

COLUMN_NAME	DATA_DEFAULT
ADDRESS	'SEOUL'
HIREDATE	SYSDATE

☞ 컬럼과 관련된 일반적인 정보(컬럼의 이름, 데이터유형, 데이터의 최대 길이, NULL 허용 유무)는 SQL*Developer 또는 SQL*Plus의 명령어인 DESCRIBE(줄여서, DESC) 명령어를 이용하는 것이 편리합니다

14-6. 제약조건 정보.

■ **USER_CONSTRAINTS** 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 제약조건에 대한 설정 정보를 확인할 수 있습니다.

■ **USER_CONS_COLUMNS** 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 제약조건들이 정의된 테이블의 컬럼이름을 확인할 수 있습니다.

○ **USER_CONSTRAINT** 데이터 딕셔너리 뷰의 대표적인 컬럼이름 및 제공되는 정보.

컬럼이름	제공되는 정보
OWNER	제약조건의 소유자 이름이 표시됩니다.
CONSTRAINT_NAME	제약조건의 이름이 표시됩니다.
CONSTRAINT_TYPE	제약조건의 유형이 표시됩니다.
STATUS	제약조건의 현재 상태(ENABLE/DISABLE)가 표시됩니다.
R_OWNER	외래키 제약조건이 참조하는 기본키 또는 유일키 제약조건의 소유자가 표시됩니다.
R_CONSTRAINT_NAME	외래키 제약조건이 참조하는 기본키 또는 유일키 제약조건의 이름이 표시됩니다.
DELETE_ROLE	외래키 제약조건에 ON DELETE CASCADE/ON DELETE SET NULL 절 적용유무가 표시됩니다.
SEARCH_CONDITION	체크 제약조건에 정의된 조건을 표시합니다. NOT NULL 제약조건에 대해서 "컬럼" IS NOT NULL 조건이 항상 표시됩니다.

○ **USER_CONS_COLUMNS** 데이터 딕셔너리 뷰의 대표적인 컬럼이름 및 제공되는 정보

컬럼이름	제공되는 정보
CONSTRAINT_NAME	제약조건의 이름이 표시됩니다.
COLUMN_NAME	제약조건이 정의된 컬럼의 이름이 표시됩니다.
POSITION	기본키, UNIQUE, 외래키 제약조건이, 두 개 이상의 컬럼으로 구성된 복합키-제약조건으로 생성되었을 때, 컬럼 순서를 표시합니다.

1> HR 계정으로 접속한 세션에서, HR.EMPLOYEES 테이블에 정의된 제약조건의 소유자, 제약조건의 이름, 제약조건 타입, 제약조건의 상태를 확인하시오.

```
SQL> SELECT owner, constraint_name, constraint_type, status
  FROM user_constraints
 WHERE table_name = 'EMPLOYEES';
```

OWNER	CONSTRAINT_NAME	C STATUS	
HR	EMP_MANAGER_FK	R ENABLED	--STATUS 컬럼의 값이
HR	EMP_JOB_FK	R ENABLED	--ENABLE이면, 제약조건이 활성화된 상태
HR	EMP_DEPT_FK	R ENABLED	
HR	EMP_SALARY_MIN	C ENABLED	--DISABLE이면, 제약조건이
HR	EMP_LAST_NAME_NN	C ENABLED	--비활성화 된 상태
HR	EMP_EMAIL_NN	C ENABLED	
HR	EMP_HIRE_DATE_NN	C ENABLED	
HR	EMP_JOB_NN	C ENABLED	
HR	EMP_EMAIL_UK	U ENABLED	
HR	EMP_EMP_ID_PK	P ENABLED	

10개의 행이 선택됨

○ USER_CONSTRAINTS 데이터 딕셔너리 뷰의 CONSTRAINT_TYPE 컬럼에 표시되는 값의 설명

값	의미
C	CHECK 제약조건 또는 NOT NULL 제약조건
P	기본키 제약조건
U	UNIQUE 제약조건
R	REFERENTIAL INTEGRITY CONSTRAINT 즉, 외래키 제약조건
V	뷰에 정의된 WITH CHECK OPTION 제약조건
O	뷰에 정의된 WITH READ ONLY 제약조건

☞ USER_CONSTRAINTS 데이터 딕셔너리 뷰의 TABLE_NAME 컬럼에, 테이블 또는 뷰의 이름이 표시됩니다.

2> HR.EMPLOYEES 테이블에 정의된 외래키 제약조건의 이름 및 외래키가 참조하는 기본키

제약조건의 이름을 확인하시오. 또한 ON DELETE CASCADE, ON DELETE SET NULL 사용 유무도 확인하시오.

SQL> SELECT owner, constraint_name, r_owner, r_constraint_name, delete_rule FROM user_constraints WHERE table_name = 'EMPLOYEES' AND constraint_type = 'R' ;				
OWNER	CONSTRAINT_NAME	R_OWNER	R_CONSTRAINT_NAME	DELETE_RULE
HR	EMP_MANAGER_FK	HR	EMP_EMP_ID_PK	NO ACTION
HR	EMP_JOB_FK	HR	JOB_ID_PK	NO ACTION
HR	EMP_DEPT_FK	HR	DEPT_ID_PK	NO ACTION

— R_CONSTRAINT_NAME컬럼에는
— 해당 외래키 제약조건이
— 참조하는 기본키 또는 UNIQUE
— 제약조건의 이름이 표시됩니다.

☞ 외래키 제약조건 정보를 확인하고 싶으면, USER_CONSTRAINTS 뷰의 OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, R_OWNER, R_CONSTRAINT_NAME, DELETE_RULE 컬럼을 모두 조회해야 합니다.

3> 위의 결과에서 EMP_DEPT_FK 제약조건이 참조하는 기본키 제약조건이 정의된 테이블의 이름을 확인하시오.

SQL> SELECT owner, table_name, constraint_type FROM user_constraints WHERE constraint_name = 'DEPT_ID_PK' ;		
OWNER	TABLE_NAME	CONSTRAINT_TYPE
HR	DEPARTMENTS	P

○ USER_CONSTRAINTS 딕셔너리 뷰의 DELETE_RULE 컬럼에 표시되는 값의 설명

값	의미
NO ACTION	ON DELETE CASCADE, ON DELETE SET NULL 옵션이 사용되지 않음.
CASCADE	ON DELETE CASCADE 옵션이 사용됨.
SET NULL	ON DELETE SET NULL 옵션이 사용됨.

4> HR.EMPLOYEES 테이블에 정의된 CHECK 및 NOT NULL 제약조건을 확인하시오.

SQL> SELECT owner, constraint_name, search_condition-- SEARCH_CONDITION컬럼에는 체크제약조건의 FROM user_constraints -- 조건식이 표시됩니다.			
OWNER	CONSTRAINT_NAME	SEARCH_CONDITION	
HR	EMP_LAST_NAME_NN	"LAST_NAME" IS NOT NULL	--NOT NULL 제약조건
HR	EMP_EMAIL_NN	"EMAIL" IS NOT NULL	--NOT NULL 제약조건
HR	EMP_HIRE_DATE_NN	"HIRE_DATE" IS NOT NULL	--NOT NULL 제약조건
HR	EMP_JOB_NN	"JOB_ID" IS NOT NULL	--NOT NULL 제약조건
HR	EMP_SALARY_MIN	salary > 0	--CHECK 제약조건

5> HR.EMPS_APP_TEST11 테이블에 설정된 각각의 제약조건이 정의된 컬럼이름을 확인하시오.

```
SQL> SELECT constraint_name, column_name, position  
      FROM user_cons_columns  
     WHERE table_name = 'EMPS_APP_TEST11'  
     ORDER BY 1, 3 ;
```

CONSTRAINT_NAME	COLUMN_NAME	POSITION
CK_SAL_EMPS_APP_TEST11	SALARY	
CK_SAL_EMPS_APP_TEST11	ADDRESS	
FK_DEPTID_EMPS_APP_TEST11	DEPTID	1
NN_ENAME_EMPS_APP_TEST11	EMPNAME	
PK_EMPID_EMPS_APP_TEST11	EMPID	1
SYS_C0011056	EMAIL	
UK_EMAIL_EMPS_APP_TEST11	EMAIL	1

7개의 행이 선택됨

☞ USER_CONS_COLUMNS 뷰의 POSITION 컬럼에는 기본키, UNIQUE, 외래키 제약조건이, 두 개 이상의 컬럼으로 구성된 복합-제약조건으로 생성되었을 때, 컬럼 순서를 표시합니다.

14-7. 뷰 정보

- **USER_VIEWS** 데이터 딕셔너리 뷰를 이용하여, 현재 접속한 계정이 소유한 뷰와 관련된 시스템 정보를 제공합니다.
- 주로 뷰에 정의된 SELECT문의 내용(TEXT 컬럼)을 확인하기 위하여 조회합니다.

1> HR로 접속한 세션에서, HR. EMP_DETAILS_VIEW 뷰에 정의된 SELECT문을 확인하시오.

- [F5] 키를 눌러서, 스크립트 실행 방법으로 아래의 문장을 실행합니다.

그러면, 뷰 생성 시에 작성한 SELECT문이 그대로 작성된 그대로 표시됩니다.

- [Ctrl]키와 [Enter] 키를 동시에 눌러서 실행하면, SELECT문이 한 줄로 표시됩니다. 불편합니다.

```
SQL> SET LONG 1000          -- LONG/CLOB 데이터유형이 표시되는 공간을 1000글자로 늘림.
SQL> SET PAGESIZE 500
SQL> SELECT text
      FROM user_views
      WHERE view_name = 'EMP_DETAILS_VIEW' ;
TEXT
-----
SELECT
  e.employee_id,
  e.job_id,
  e.manager_id,
  e.department_id,
  d.location_id,
  l.country_id,
  e.first_name,
  e.last_name,
  e.salary,
  e.commission_pct,
  d.department_name,
  j.job_title,
  l.city,
  l.state_province,
  c.country_name,
  r.region_name
FROM
  employees e,
  departments d,
  jobs j,
  locations l,
  countries c,
  regions r
WHERE e.department_id = d.department_id
  AND d.location_id = l.location_id
  AND l.country_id = c.country_id
  AND c.region_id = r.region_id
  AND j.job_id = e.job_id
WITH READ ONLY
```

14-8. 시퀀스 정보

■ USER_SEQUENCES 데이터 딕셔너리 뷰를 통해, 현재 접속한 계정이 소유한 시퀀스와 관련된 정보를 확인할 수 있습니다.

1> HR 계정으로 접속한 세션에서, HR.DEPARTMENTS_SEQ 시퀀스에 정의된 설정 사항을 확인하시오.

```
SQL> SELECT SEQUENCE_NAME, MIN_VALUE, MAX_VALUE, INCREMENT_BY, CYCLE_FLAG, CACHE_SIZE
      ,LAST_NUMBER
  FROM user_sequences
 WHERE sequence_name = 'DEPARTMENTS_SEQ' ;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	CACHE_SIZE	LAST_NUMBER
DEPARTMENTS_SEQ	1	9990	10 N	0	280

○ USER_SEQUENCES 데이터 딕셔너리 뷰의 대표적인 컬럼이름 및 제공되는 정보

컬럼명	제공되는 정보
SEQUENCE_NAME	시퀀스의 이름을 표시합니다.
MIN_VALUE	시퀀스에 설정된 최소값을 표시합니다.
MAX_VALUE	시퀀스에 설정된 최대값을 표시합니다.
INCREMENT_BY	시퀀스에 설정된 시퀀스의 증분 값을 표시합니다.
CYCLE_FLAG	시퀀스가 제한(MAX_VALUE)에 도달하면, 처음부터 재사용되는지 여부를 표시합니다.
CACHE_SIZE	캐시 할 시퀀스 번호 개수를 표시합니다.
LAST_NUMBER	데이터 딕셔너리에 저장되어 있는, 바로 다음에 사용할 시퀀스 번호입니다. 시퀀스가 캐시를 사용하는 경우, LAST_NUMBER 컬럼의 값은 실제로 사용하는 다음 시퀀스 번호가 아니라, 시퀀스 캐시에 배치된 마지막 번호의 다음 번호입니다.

☞ 시퀀스를 생성할 때 NOCACHE 옵션을 사용하면, CACHE_SIZE 컬럼이 0으로 표시되며, 이러한 시퀀스는 USER_SEQUENCES.LAST_NUMBER 컬럼에 표시된 값이 다음에 사용할 값입니다. 시퀀스의 다음 숫자를 세션에서 호출할 때마다, 데이터 딕셔너리를 액세스 해야 하므로, 처리 성능이 좋지 못합니다.

☞ 시퀀스를 생성할 때 CACHE 10 옵션을 사용했다면, CACHE_SIZE 컬럼에 10으로 표시되며, 이러한 시퀀스가 NEXTVAL 가상컬럼에 의하여 호출될 때, 메모리에 캐싱 된 시퀀스 값이 없으면, 딕셔너리의 LAST_NUMBER의 값부터 시작하여 10 개의 숫자를 캐싱하고, 딕셔너리의 LAST_NUMBER 값은 캐싱 된 마지막 숫자의 다음 숫자가 저장됩니다. 캐싱 된 시퀀스의 숫자를 모두 사용할 때까지 데이터 딕셔너리에 대한 액세스를 하지 않게 됩니다.

☞ 시퀀스의 캐싱 되는 숫자의 개수를 변경하려고 할 때, 다음의 SQL문을 사용합니다.

```
ALTER SEQUENCE 시퀀스이름 CACHE 캐시_되는_숫자의_개수 ;
```

14-9. 인덱스 정보

■ USER_INDEXES 데이터 딕셔너리 뷰를 통해, 현재 접속한 사용자가 소유한 인덱스 관련 설정 정보를 확인할 수 있습니다.

■ USER_IND_COLUMNS 데이터 딕셔너리 뷰를 통해, 현재 접속한 사용자가 소유한 인덱스를 구성하는 컬럼이름 및 컬럼 순서를 확인할 수 있습니다.

1> HR 계정으로 접속한 세션에서, HR.EMPLOYEES 테이블에 정의된 인덱스 이름 및 UNIQUENESS를 확인하시오.

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, UNIQUENESS
  FROM user_indexes
 WHERE table_name = 'EMPLOYEES' ;
```

INDEX_NAME	INDEX_TYPE	UNIQUENESS
EMP_NAME_IX	NORMAL	NONUNIQUE
EMP_MANAGER_IX	NORMAL	NONUNIQUE
EMP_JOB_IX	NORMAL	NONUNIQUE
EMP_DEPARTMENT_IX	NORMAL	NONUNIQUE
EMP_EMP_ID_PK	NORMAL	UNIQUE
EMP_EMAIL_UK	NORMAL	UNIQUE

6개의 행이 선택됨

2> HR.EMPLOYEES 테이블에 정의된 인덱스들이 정의된 컬럼이름을 확인하시오.

```
SQL> SELECT index_name, column_name, column_position
  FROM user_ind_columns
 WHERE table_name = 'EMPLOYEES' ;
```

INDEX_NAME	COLUMN_NAME	COLUMN_POSITION	
EMP_EMAIL_UK	EMAIL	1	-- EMP_NAME_IX 인덱스는 복합-키 형식으로
EMP_EMP_ID_PK	EMPLOYEE_ID	1	-- 인덱스가 구성되었음을 의미합니다
EMP_DEPARTMENT_IX	DEPARTMENT_ID	1	-- 즉, 다음의 문장으로 생성되었습니다.
EMP_JOB_IX	JOB_ID	1	-- CREATE INDEX HR.EMP_NAME_IX
EMP_MANAGER_IX	MANAGER_ID	1	-- ON HR.EMPLOYEES(LAST_NAME, FIRST_NAME) ;
EMP_NAME_IX	LAST_NAME	1	
EMP_NAME_IX	FIRST_NAME	2	

7개의 행이 선택됨

14-10. 동의어 정보.

- USER_SYNONYMS 데이터 딕셔너리 뷰를 통해, 현재 접속한 사용자가 소유한 동의어 객체에 대한 정보를 확인할 수 있습니다.
- ALL_SYNONYMS 데이터 딕셔너리 뷰를 통해, 현재 접속한 사용자가 액세스 할 수 있는 동의어 객체에 대한 정보를 확인할 수 있습니다.
- 주로 사용자들은, 데이터베이스 관리자가 생성한 공용 동의어(PUBLIC SYNONYM)의 정보를 함께 확인하기 위하여, [ALL_SYNONYMS 딕셔너리 뷰를 통해](#), 현재 접속한 사용자가 사용 가능한 동의어 정보를 확인합니다.

1> HR 계정으로 접속한 세션에서, HR, SH, OE, PM 계정이 소유한 테이블에 대하여, HR 계정이 사용할 수 있는 동의어를 확인하시오.

```
SQL> SELECT owner, synonym_name, table_owner, table_name
      FROM all_synonyms
     WHERE TABLE_OWNER IN ('HR', 'SH', 'OE' , 'PM') ;
```

OWNER	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME
OE	COUNTRIES	HR	COUNTRIES
OE	DEPARTMENTS	HR	DEPARTMENTS
OE	EMPLOYEES	HR	EMPLOYEES
OE	JOB_HISTORY	HR	JOB_HISTORY
OE	LOCATIONS	HR	LOCATIONS

6개의 행이 선택됨

14-11 현재 접속한 세션 및 데이터베이스에 대한 국가별 언어 지원과 관련된 설정 확인.

- NLS_SESSION_PARAMETERS 데이터 딕셔너리 뷰를 통하여, 현재 접속한 세션의 국가별 언어 지원과 관련된 세션의 설정사항을 확인할 수 있습니다.
- NLS_DATABASE_PARAMETERS 뷰를 통하여, 현재 접속한 오라클 데이터베이스 서버에 설정된 국가별 언어 지원과 관련된 설정을 확인할 수 있습니다.

1> HR 계정으로 접속한 세션에서, 현재 접속한 세션의 국가별 언어 지원과 관련된 세션 설정을 확인하시오.

```
SQL> SELECT * FROM NLS_SESSION_PARAMETERS ;
```

PARAMETER	VALUE
NLS_LANGUAGE	KOREAN
NLS_TERRITORY	KOREA
NLS_CURRENCY	₩
NLS_ISO_CURRENCY	KOREA
NLS_NUMERIC_CHARACTERS	.,,
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	RR/MM/DD
NLS_DATE_LANGUAGE	KOREAN
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH24:MI:SSXFF
NLS_TIMESTAMP_FORMAT	RR/MM/DD HH24:MI:SSXFF
NLS_TIME_TZ_FORMAT	HH24:MI:SSXFF TZR
NLS_TIMESTAMP_TZ_FORMAT	RR/MM/DD HH24:MI:SSXFF TZR
NLS_DUAL_CURRENCY	₩
NLS_COMP	BINARY
NLS_LENGTH_SEMANTICS	BYTE
NLS_NCHAR_CONV_EXCP	FALSE

17개의 행이 선택됨

```
ALTER SESSION SET nls_date_format='YYYY/MM/DD HH24:MI:SS';
ALTER SESSION SET nls_timestamp_format='YYYYMMDD HH:MI:SSXFF AM';
ALTER SESSION SET nls_timestamp_tz_format='YYYY/MM/DD HH24:MI:SSXFF TZR';
ALTER SESSION SET nls_timestamp_tz_format='YYYY/MM/DD HH24:MI:SSXFF TZH:TZM';

ALTER SESSION SET nls_territory=korea;
ALTER SESSION SET nls_lang=korean;

[      ]
(For Oracle Tools)
NLS_LANG=KOREAN_KOREA.UTF8; export NLS_LANG
NLS_DATE_FORMAT='YYYY/MM/DD HH24:MI:SS'; export NLS_DATE_FORMAT
NLS_TIMESTAMP_FORMAT='YYYYMMDD HH:MI:SSXFF AM'; export NLS_TIMESTAMP_FORMAT
NLS_TIMESTAMP_TZ_FORMAT='YYYY/MM/DD HH24:MI:SSXFF TZR'; export NLS_TIMESTAMP_TZ_FORMAT
NLS_TIMESTAMP_TZ_FORMAT='YYYY/MM/DD HH24:MI:SSXFF TZH:TZM'; export NLS_TIMESTAMP_TZ_FORMAT
```

2> 현재 계정이 접속한, 오라클 데이터베이스 서버의 국가별 언어 지원과 관련된 설정을 확인하시오.

```
SQL> SELECT * FROM NLS_DATABASE_PARAMETERS ;
```

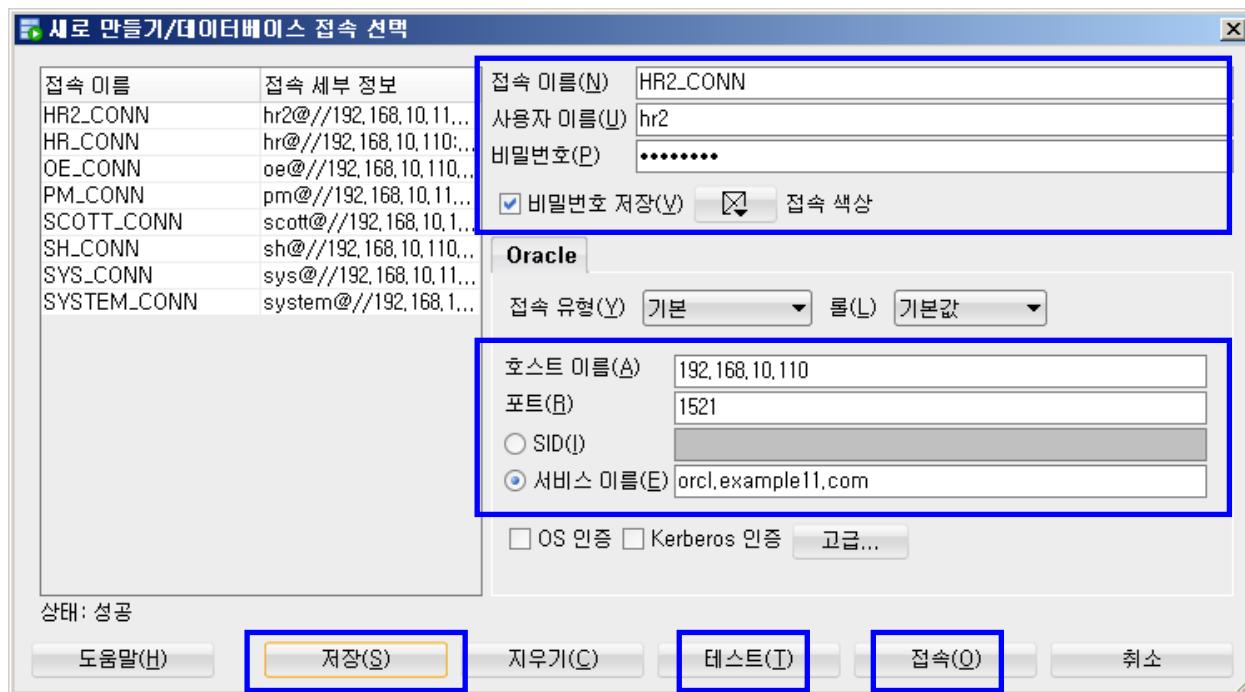
PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	\$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.,,
NLS_CHARACTERSET	AL32UTF8
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	DD-MON-RR
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT	DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT	HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT	DD-MON-RR HH.MI.SSXFF AM TZR
NLS_DUAL_CURRENCY	\$
NLS_COMP	BINARY
NLS_LENGTH_SEMANTICS	BYTE
NLS_NCHAR_CONV_EXCP	FALSE
NLS_NCHAR_CHARACTERSET	AL16UTF16
NLS_RDBMS_VERSION	11.2.0.1.0

20개의 행이 선택됨

14-12 계정에게 부여된 권한 정보 확인

- 다음의 실습을 통해, 사용자 생성 단원에서 실습으로 생성한 HR2 계정에게 부여된, 시스템 권한, 객체 권한 및 룰과 관련된 데이터 딕셔너리 정보를 확인해 봅니다.

1> SQL*Developer를 실행한 후, HR2로 접속할 수 있도록 [새 접속]을 생성합니다.



- 다음의 지시대로 각 항목에 적절한 값을 입력합니다.
 - 접속이름: HR2_CONN
 - 사용자이름: hr2
 - 비밀번호: oracle4U
 - [비밀번호 저장] 항목 체크
 - 호스트이름: 192.168.10.110 --접속할 데이터베이스 서버의 IP주소입니다.
 - 포트: 1521 --접속요청을 보낼 리스너 포트 번호입니다.
 - 서비스 이름: orcl.example11.com --접속할 데이터베이스 서비스 이름입니다.
- [테스트]를 클릭합니다. [도움말] 위에, 상태: 성공 이라고 표시되어야 합니다.
- [저장]을 클릭합니다. 왼쪽 창에 생성한 HR2_CONN 접속 이름이 표시됩니다.
- [접속]을 클릭하여, SQL*Developer로 오라클 데이터베이스에 HR2 계정으로 접속합니다.

2> HR2 계정으로 접속한 세션에서, SYSTEM_PRIVILEGE_MAP 딕셔너리 뷰를 이용하여, 시스템 권한 중 SYNONYM 객체와 관련된 시스템 권한의 이름을 확인하시오.

```
SQL> SELECT NAME
   FROM SYSTEM_PRIVILEGE_MAP
  WHERE NAME LIKE '%SYNONYM%' ;  — 데이터 딕셔너리에 저장된 시스템 권한의 이름은 대문자입니다.

NAME
-----
DROP PUBLIC SYNONYM      ↗ 공용동의어를 삭제할 수 있는 시스템 권한입니다.
CREATE PUBLIC SYNONYM    ↗ 공용동의어를 생성할 수 있는 시스템 권한입니다.
DROP ANY SYNONYM          ↗ 다른 계정이 소유한 동의어를 삭제할 수 있는 시스템 권한입니다.
CREATE ANY SYNONYM        ↗ 다른 계정 소유의 동의어를 생성할 수 있는 시스템 권한입니다.
CREATE SYNONYM            ↗ 자신이 소유하는 동의어를 생성할 수 있는 시스템 권한입니다.
```

- ☞ SYSTEM_PRIVILEGE_MAP 딕셔너리 뷰를 통해, 오라클-서버에 구성된 시스템 권한들의 이름을 확인할 수 있습니다.
- ☞ 오라클 데이터베이스 11g Release 2에서는 208개의 시스템 권한이 존재합니다.

3> 현재 접속한 HR2 계정에게 직접 부여된 시스템 권한의 이름을 확인하시오.

```
SQL> SELECT PRIVILEGE      -- USER_SYS_PRIVS 뷰를 이용하여 현재 접속한 계정에게 직접 부여된
   FROM USER_SYS_PRIVS;    -- 시스템 권한의 이름을 확인할 수 있습니다

PRIVILEGE
-----
CREATE SYNONYM
CREATE SEQUENCE
CREATE SESSION
CREATE VIEW
ALTER SESSION
CREATE TABLE

6개의 행이 선택됨
```

4> 현재 접속한 HR2 계정에게 부여된 룰 이름(룰-권한)을 확인하시오.

```
SQL> SELECT GRANTED_ROLE    -- USER_ROLE_PRIVS 뷰를 이용하여 현재 접속한 계정에게 직접 부여된
   FROM USER_ROLE_PRIVS;   -- 룰의 이름을 확인할 수 있습니다.

GRANTED_ROLE
-----
EMPROLE1
```

5> 현재 접속한 HR2 계정에게 부여된 EMPROLE1-를에 부여된 시스템 권한의 이름을 확인하시오.

```
SQL> SELECT PRIVILEGE          -- ROLE_SYS_PRIVS 뷰를 이용하여 현재 접속한 계정에게 부여된 를에
      FROM ROLE_SYS_PRIVS      -- 설정된 시스템 권한의 이름을 확인할 수 있습니다.
      WHERE ROLE='EMPROLE1' ;
```

PRIVILEGE

```
CREATE TABLE
CREATE SESSION
CREATE SYNONYM
```

☞ 접속한 계정에게 부여된 를에 대해서만 설정된 시스템 권한을 확인할 수 있습니다.

6> 현재 접속한 HR2 계정에게 부여된 EMPROLE1-를에 부여된 다른 를의 이름을 확인하시오.

```
SQL> SELECT GRANTED_ROLE        -- ROLE_ROLE_PRIVS 뷰를 이용하여 현재 접속한 계정에게 직접 부여된 를에
      FROM ROLE_ROLE_PRIVS      -- 설정된 다른 를의 이름을 확인할 수 있습니다.
      WHERE ROLE='EMPROLE1' ;
```

선택된 행 없음 -- EMPROLE1에 부여된 다른 를은 없습니다.

7> 현재 접속한 HR2 계정에게 부여된 EMPROLE-를에 설정된 객체권한의 이름을 확인하시오.

```
SQL> SELECT owner, table_name, column_name, privilege    -- ROLE_TAB_PRIVS 뷰를 이용하여
      FROM ROLE_TAB_PRIVS
      WHERE ROLE='EMPROLE1' ;                                -- 현재 접속한 계정에게 를을 통해 설정된
                                                               -- 객체권한의 이름을 확인할 수 있습니다.
```

OWNER	TABLE_NAME	COLUMN_NAME	PRIVILEGE	
HR	EMPLOYEES		SELECT	-- TABLE_NAME에 테이블을 포함하여 -- 다른 유형의 객체들의 이름이 -- 표시됩니다.
HR	DEPARTMENTS		SELECT	

8> 현재 접속한 HR2계정에게 직접 부여된 객체 권한을 확인하시오.

```
SQL> SELECT owner, table_name, privilege, grantor      -- USER_TAB_PRIVS_REC 뷰를 이용하여 현재 접속한
      FROM USER_TAB_PRIVS_REC ;
                                                               -- 계정에게 직접 부여된 객체권한에 대한 정보를
                                                               -- 확인할 수 있습니다.
```

OWNER	TABLE_NAME	PRIVILEGE	GRANTOR	
HR	LOCATIONS	ALTER	HR	-- TABLE_NAME에 테이블을 포함하여
HR	LOCATIONS	SELECT	HR	-- 다른 유형의 객체들의 이름이 표시됩니다.
HR	LOCATIONS	INSERT	HR	
HR	LOCATIONS	INDEX	HR	
HR	LOCATIONS	DELETE	HR	

9> 현재 접속한 HR 계정에게 직접 부여된 객체 권한 중, 컬럼에 제한이 걸린 객체권한만 조회 확인하시오.

```
SQL> SELECT owner, table_name, column_name, privilege, grantor, grantable
   FROM user_col_privs_recd ;
```

OWNER	TABLE_NAME	COLUMN_NAME	PRIVILEGE	GRANTOR	GRANTABLE
HR	DEPARTMENTS	DEPARTMENT_ID	REFERENCES	HR	NO
HR	DEPARTMENTS	DEPARTMENT_NAME	UPDATE	HR	NO

☞ USER_COL_PRIVS_RECD 뷰를 이용하여, 현재 접속한 계정에게 컬럼의 이름을 명시하여 UPDATE 및 REFERENCES 객체 권한이 직접 부여되었을 때, 설정된 컬럼이름을 제공합니다.

10> SQL*Developer를 이용하여, HR 계정으로 데이터베이스에 접속합니다.

11> HR.EMPLOYEES 테이블에 대하여 다른 계정에게 부여된 객체 권한을 확인하시오.

```
SQL> SELECT table_name, privilege, grantee ,grantor -- USER_TAB_PRIVS_MADE 뷰를 이용하여
   FROM user_tab_privs_made
   WHERE table_name='EMPLOYEES'; -- 현재 접속한 계정이 소유한 객체에 대하여
                                 -- 다른 계정이나 룰에게 부여된 객체 권한에 대한
                                 -- 정보를 확인할 수 있습니다.
```

TABLE_NAME	PRIVILEGE	GRANTEE	GRANTOR
EMPLOYEES	SELECT	OE	HR
EMPLOYEES	REFERENCES	OE	HR
EMPLOYEES	SELECT	EMPROLE1	HR

12> 현재 접속한 HR 계정이 사용 가능한 모든 시스템 권한의 이름을 확인하시오.

```
SQL> SELECT * FROM SESSION_PRIVS ;
PRIVILEGE
-----
```

-- SESSION_PRIVS 딕셔너리 뷰를 이용하여, 현재 접속한 계정에게
-- 직접 또는 룰을 통해서 부여된 시스템 권한들 중에서
-- 현재 세션에서 사용할 수 있는 모든 시스템 권한의 이름을
-- 확인할 수 있습니다

```
CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
CREATE PROCEDURE
CREATE TRIGGER
CREATE TYPE
CREATE OPERATOR
CREATE INDEXTYPE
```

14개의 행이 선택됨

13> 현재 접속한 HR 계정이 사용 가능한 모든 룰의 이름을 확인하시오.

```
SQL> SELECT * FROM SESSION_ROLES ;
ROLE
-----
RESOURCES
EMPROLE1
```

-- SESSION_ROLES 딕셔너리 뷰를 이용하여 현재 접속한 계정에게
 -- 직접 또는 룰을 통해서 부여된 룰들 중에서
 -- 현재 세션에서 사용할 수 있는 모든 룰의 이름을 확인할 수 있습니다.

[참고] DBA_SYS_PRIVS, DBA_TAB_PRIVS, DBA_ROLES, DBA_ROLE_PRIVS 딕셔너리 뷰

- DBA는 DBA_SYS_PRIVS 딕셔너리 뷰를 이용하여, 모든 계정 및 모든 룰에게 설정된 객체 권한 정보를 확인할 수 있습니다.
- DBA는 DBA_TAB_PRIVS 딕셔너리 뷰를 이용하여, 모든 계정과 모든 룰에게 설정된 객체 권한 정보를 확인할 수 있습니다.
- DBA는 DBA_ROLE_PRIVS 딕셔너리 뷰를 이용하여, 모든 계정 및 모든 룰에게 설정된 룰 정보를 확인할 수 있습니다.
- DBA는 DBA_ROLES 딕셔너리 뷰를 이용하여, 데이터베이스에 존재하는 모든 룰의 이름을 확인할 수 있습니다.

[참고] 권한 및 룰 정보를 제공하는 데이터 딕셔너리 뷰.

데이터 딕셔너리 뷰	제공되는 정보
USER_SYS_PRIVS	현재 접속한 계정에게 직접 부여된 시스템 권한 정보를 제공합니다.
USER_ROLE_PRIVS	현재 접속한 계정에게 직접 부여된 룰에 대한 정보를 제공합니다.
ROLE_SYS_PRIVS	현재 접속한 계정에게 부여되어 있는 룰에 부여된 시스템 권한 정보를 제공합니다.
ROLE_ROLE_PRIVS	현재 접속한 계정에게 부여되어 있는 룰에 부여된 룰 정보를 제공합니다.
ROLE_TAB_PRIVS	현재 접속한 계정에게 부여되어 있는 룰에 부여된 객체 권한 정보를 제공합니다.
USER_TAB_PRIVS_MADE	현재 접속한 계정의 스키마 객체에 대하여 다른 계정에게 부여된 객체 권한 정보를 제공합니다.
USER_TAB_PRIVS_REC'D	현재 접속한 계정이 다른 계정 소유의 스키마 객체에 대하여 부여 받은 객체 권한 정보를 제공합니다.
USER_COL_PRIVS_MADE	현재 접속한 계정의 스키마 객체에 대하여 다른 계정에게 UPDATE/REFERENCES 객체 권한이 부여될 때, 컬럼이름이 지정된 경우, 해당 컬럼을 확인할 수 있습니다.
USER_COL_PRIVS_REC'D	현재 접속한 계정이 다른 계정 소유의 스키마 객체에 대하여 UPDATE/REFERENCES 객체 권한을 부여 받을 때 컬럼이 지정된 경우, 해당 컬럼을 확인할 수 있습니다..
SESSION_PRIVS	현재 접속한 세션에서 사용할 수 있도록 설정된 시스템 권한의 이름을 표시합니다.
SESSION_ROLES	현재 접속한 세션에서 사용할 수 있도록 설정된 룰의 이름을 표시합니다.

14-13. 테이블 및 컬럼에 주석 추가 및 확인.

- 사용자 테이블 및 각 컬럼에 사용자의 주석을 딕셔너리 정보로 저장한 후, 필요할 때마다 테이블의 의미 및 컬럼의 의미를 조회하여 확인할 수 있습니다.

1> SQL*Developer를 이용하여, HR 계정으로 데이터베이스에 접속합니다.

2> 앞의 컬럼 정보 조회 실습에서 생성한 HR.EMPS_APP_TEST11 테이블에 '사원정보 테이블' 주석을 추가하시오.

```
SQL> COMMENT ON TABLE HR.EMPS_APP_TEST11
      IS '사원정보 테이블' ;
```

Comment on table hr.emps_app_test11 '사원정보 테이블'을(를) 성공했습니다.

3> HR.EMPS_APP_TEST11 테이블의 EMPID 컬럼에 '사원 식별 아이디'라는 컬럼 주석을 추가하시오.

```
SQL> COMMENT ON COLUMN HR.EMPS_APP_TEST11.EMPID
      IS '사원 식별 아이디' ;
```

Comment on column hr.emps_app_test11.empid '사원 식별 아이디'을(를) 성공했습니다.

4> HR.EMPS_APP_TEST11 테이블에 설정된 테이블 주석을 확인해 보시오.

```
SQL> SELECT comments
      FROM user_tab_comments
     WHERE table_name = 'EMPS_APP_TEST11';
-- USER_TAB_COMMENTS 딕셔너리 뷰를 이용하여,
-- 테이블에 설정된 주석을 확인할 수 있습니다.

COMMENTS
-----
사원정보 테이블
```

5> HR.EMPS_APP_TEST11 테이블의 EMPID 컬럼에 설정된 컬럼 주석을 확인해 보시오.

```
SQL> SELECT comments
      FROM user_col_comments
     WHERE table_name = 'EMPS_APP_TEST11'
       AND column_name = 'EMPID';
-- USER_COL_COMMENTS 딕셔너리 뷰를 이용하여,
-- 테이블에 설정된 주석을 확인할 수 있습니다.

COMMENTS
-----
사원 식별 아이디
```

6> HR.EMPS_APP_TEST11 테이블에 설정된 테이블 주석을 삭제하시오.

```
SQL> COMMENT ON TABLE HR.EMPS_APP_TEST11  
IS '' ;
```

Comment on table hr.emps_app_test11 ''을(를) 성공했습니다.

☞ 주석 삭제는 작은 따옴표(')를 연속해서 2번 입력하면, 설정된 주석이 삭제됩니다.

7> HR.EMPS_APP_TEST11 테이블의 EMPID 컬럼에 설정된 컬럼 주석을 삭제하시오.

```
SQL> COMMENT ON COLUMN HR.EMPS_APP_TEST11.EMPID  
IS '' ;
```

Comment on column hr.emps_app_test11.empid ''을(를) 성공했습니다.

☞ 주석 삭제는 작은 따옴표(')를 연속해서 2번 입력하면, 설정된 주석이 삭제됩니다.

■ 지금까지 데이터 딕셔너리 뷰에 직접 조회하여 시스템 정보를 확인하는 방법을 설명했습니다.

오라클 데이터베이스에 접속하는 개발자, 또는 관리자가 SQL*Developer를 이용하는 경우에는 딕셔너리에 대한 쿼리를 작성해서 실행하지 않고, 마우스 클릭으로 지금까지 배운 시스템 정보를 확인할 수 있습니다.

[참고] SQL*Plus/SQL*Developer 툴에서 치환변수(Substitution Variables)의 활용.

- 치환변수는 SQL 언어의 기능이 아니며, 오라클 SQL*Plus 또는 SQL*Developer 툴에서 제공되는 기능입니다.

- 치환변수를 이용하여 동일한 SQL문을 재작성 하지 않고도 일부 상수값을 변경해가며 실행시킬 수 있습니다.

- SQL*Plus 또는 SQL*Developer 툴에서 치환변수를 선언하고 값을 지정하는 방법.

(1) SQL*Plus 또는 SQL*Developer 툴의 DEFINE 명령어를 이용하여 선언 및 값의 지정도 같이 수행합니다.

(2) SQL문에서 변수이름 앞에 &를 명시하여 선언하고 SQL문 실행 시에 값을 지정 합니다.

(3) SQL문에서 변수이름 앞에 &&를 명시하여 선언하고 SQL문 실행 시에 값을 지정 합니다.

- SQL문에서 변수이름 앞에 &&를 명시하거나 DEFINE 명령어를 이용하여 선언하는 경우에는,

접속 세션에 선언된 치환변수가 계속 유지됩니다.

- SQL문에서 변수이름 앞에 &를 명시하여 선언하는 경우에는 접속 세션에 선언된 치환변수가 유지되지 않습니다.

- SQL문에서 & 또는 &&를 이용하여 치환변수를 선언하는 경우, 해당 세션에서 같은 이름의 치환변수가 이미 선언되어 값이 지정되어 있으면, 값의 입력을 요구하지 않고, 세션에 이미 지정된 값으로 그대로 사용하고, 미리 지정된 값이 없으면 입력을 요구합니다.

[실습]: 치환변수 선언 및 값 지정하기(동일한 접속 세션에서 수행합니다).

- 오라클-서버가 구성된 리눅스 가상머신의 터미널에서 SQL*Plus를 실행하여 HR 계정으로 접속합니다.

```
$ sqlplus hr/oracle4U
```

- DEFINE 명령어로 세션에 eid1 치환변수를 선언하고 100을 지정합니다.

```
SQL> DEFINE eid1=100
```

```
SQL>
```

- [DEFINE 치환변수이름]명령어를 이용하여 세션에 유지되어 있는 지정된 치환변수의 값을 확인할 수 있습니다.

```
SQL> DEFINE eid1
```

```
DEFINE EID1          = "100" (CHAR)
SQL>
```

4. SELECT문 작성 시에 &, &&을 변수이름 앞에 명시하여 eid2 및 did 치환변수를 각각 선언합니다.

```
SQL> SELECT employee_id, last_name, department_id
  FROM hr.employees
 WHERE employee_id=&eid2          -- &를 명시하여 eid2 치환변수를 선언합니다.
   OR department_id=&&did ;        -- &&를 명시하여 did 치환변수를 선언합니다.

Enter value for eid2: 101          -- SQL문 실행 시에 eid2 치환변수에 지정할 값을 물어옵니다.
old 3:      WHERE employee_id=&eid2
new 3:      WHERE employee_id=101
Enter value for did: 90           -- SQL문 실행 시에 did 치환변수에 지정할 값을 물어옵니다.
old 4:      OR      department_id=&&did
new 4:      OR      department_id=90

EMPLOYEE_ID LAST_NAME  DEPARTMENT_ID
-----  -----  -----
    100 King            90
    101 Kochhar         90
    102 De Haan         90
```

- 위의 SELECT문을 수행하면 명시된 치환변수가 다음처럼 사용됩니다.

- (1) eid2 치환변수는 &을 사용했으므로 지정된 값이 세션에 유지되지 않습니다.
- (2) did 치환변수는 &&을 사용했으므로 지정된 값이 세션에 계속 유지됩니다.

5. 다음의 문장을 실행하여 앞의 문장과 실행될 때를 비교해 봅니다.

```
SQL> SELECT employee_id, last_name, department_id
  FROM hr.employees
 WHERE employee_id=&&eid1
   OR department_id=&did ;
old 3:      WHERE employee_id=&&eid1
new 3:      WHERE employee_id=100
old 4: OR      department_id=&did
new 4: OR      department_id=90

EMPLOYEE_ID LAST_NAME  DEPARTMENT_ID
-----  -----  -----
    100 King            90
    101 Kochhar         90
    102 De Haan         90
```

- 위의 SELECT문을 실행하면, 다음과 같은 이유 때문에 치환변수에 값을 지정하도록 입력을 요구하지 않습니다.

- (1) eid1 치환변수는 이미 앞에서 DEFINE 명령어에 의해 변수가 선언되고 값이 지정되어 세션에 유지되어 있습니다.
- (2) did 치환변수는 처음 실행된 SELECT 문에서 && 기호에 의하여 지정된 값이 유지되어 있습니다.

- UNDEFINE 명령어를 이용하여 현재 세션에 유지되어 있는 치환변수를 해제할 수 있습니다.

```
SQL> UNDEFINE eid1 did
SQL> DEFINE eid1 did
SQL> -- 지정된 치환 변수가 해제되어 아무것도 표시되지 않습니다.
```

- DEFINE 명령어를 이용하여 현재 세션에 유지되어 있는 모든 치환변수를 확인할 수 있습니다.

```
SQL> DEFINE
DEFINE _DATE          = "2013/01/21" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "orcl" (CHAR)
DEFINE _USER           = "HR" (CHAR)
DEFINE _PRIVILEGE     = "" (CHAR)
DEFINE _SQLPLUS_RELEASE = "1002000100" (CHAR)
DEFINE _EDITOR         = "vi" (CHAR)
DEFINE _O_VERSION      = "Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options" (CHAR)
DEFINE _O_RELEASE       = "1002000100" (CHAR)
SQL>
```

- [SET VERIFY OFF] 명령어를 이용하면, 치환변수 사용 시에 값이 변경되는 과정이 화면에 표시되지 않습니다.

```
SQL> SET VERIFY OFF

SQL> SELECT employee_id, last_name, department_id
   FROM hr.employees
  WHERE employee_id=&eid2
    OR department_id=&did ;
Enter value for eid2: 101
Enter value for did: 90

EMPLOYEE_ID LAST_NAME  DEPARTMENT_ID
-----  -----
      100 King            90
      101 Kochhar          90
      102 De Haan          90
```

[참고] SQL*Plus/SQL*Developer 툴의 COLUMN (줄여서 COL) 명령어.

- SQL*Plus/SQL*Developer 툴에서 제공하는 column 명령어를 포함한 여러 명령어를 이용하여 SELECT 문의 최종 결과 물을 보고서처럼 출력할 수 있습니다.

- 표시 결과 출력 시에 같이 사용하면 도움이 되는 SQL*Plus 명령어.

COL last_name FORMAT A15	출력 결과의 Heading이 LAST_NAME인 문자 데이터를 15바이트 공간에 출력.
COL hire_date FOR A20	출력 결과의 Heading이 HIRE_DATE인 날짜 데이터를 20글자 공간에 출력.
COL salary FOR 99999	출력 결과의 Heading이 SALARY인 숫자 데이터를 5글자 공간에 출력.
SET PAGESIZE 999	한 페이지를 999 개 라인으로 사용하며, 999 라인마다 Heading이 출력.
SET LINESIZE 200	한 라인의 길이를 200 Byte로 제한.
TTITLE 'Employee Report'	각 페이지마다 'Employee Report'를 두 줄에 제목으로 표시.
BTITLE 'Confidential'	각 페이지마다 'Confidential'을 마지막에 표시.
SET FEEDBACK OFF	SELECT문의 결과레코드 마지막에 표시되는 메시지를 표시하지 않음.
BREAK ON JOB_ID	출력결과의 Heading이 JOB_ID인 컬럼에서 연속적으로 표시되는 동일 데이터를 처음 한번만 표시. (주의) BREAK ON 설정된 Heading은 (예, JOB_ID) 반드시 SELECT 문에서 SELECT절과 ORDER BY절에 명시되어야 합니다.
COLUMN job_id CLEAR	출력결과의 Heading이 JOB_ID에 설정된 사항을 모두 지움.
CLEAR BREAK	설정된 BREAK를 모두 지움.
TTITLE OFF	페이지 마다 제목으로 표시하도록 설정된 내용을 지움.
BTITLE OFF	페이지 마다 마지막에 표시하도록 설정된 내용을 지움.

- COLUMN 명령어 사용-고급

SQL> **COLUMN last_name HEADING 'EMPLOYEE|NAME' JUSTIFY right FORMAT a15**

(의미) 출력결과에서 LAST_NAME Heading을 'EMPLOYEE|NAME'으로 2줄에 표시하고
오른쪽 맞춤 방식으로 문자 데이터를 15글자 공간에 출력합니다.

SQL> **COL salary HEADING 'Salary' JUSTIFY center FORMAT \$999,999,999.9**

(의미) 출력결과에서 SALARYE 해딩을 'Salary'로 가운데 맞춤으로 표시하고, 레코드에 표시되는
숫자 데이터를 최대 정수부 9자리 소수점 이하 1자리 그리고 앞에 \$를 붙여서 표시합니다.

SQL> **COLUMN commission_pct NULL "No_Commission"**

(의미) 출력되는 레코드에서 해딩이 COMMISSION_PCT인 필드에 데이터가 없는 NULL 상태인 경우
빈 여백을 출력하는 대신 "No_Commission"을 표시합니다.

[참고] SQL*Plus/SQL*Developer 툴의 보고서 형식 출력기능 실습.

(1) 오라클-서버가 구성된 리눅스 가상머신에서 터미널을 실행합니다.

(2) vi 편집기로 /home/oracle/emp_report.sql 파일을 생성합니다.

```
$ vi /home/oracle/emp_report.sql
```

(2-1) 실행된 vi 편집기에서 [i 키]를 눌러 [—끼워넣기—]모드로 변경합니다.

(2-2) 아래 박스의 내용을 작성합니다.

```
SET pagesize 50
SET linesize 62
SET feedback OFF
TTITLE 'Employee|Report'
BTITLE 'Confidential'
COLUMN job_id HEADING 'Job|Category' JUSTIFY CENTER FORMAT a10
COL last_name HEADING 'EMPLOYEE|NAME' JUSTIFY CENTER FOR a20
COL salary HEADING '|Salary' JUSTIFY LEFT FOR $999,999,999.9
COL commission_pct NULL "No_Commission"
BREAK ON job_id
SPOOL /home/oracle/report_result.txt /* SQL*Developer로 원도우에서 수행 시에는 경로설정만 변경 */
-- SELECT statement
SELECT job_id, last_name, salary, commission_pct
FROM hr.employees
WHERE salary < 15000
ORDER BY job_id, last_name
/
SPOOL OFF
REM clear all formatting commands ...
SET FEEDBACK 6
COL job_id CLEAR
COL last_name CLEAR
COL salary CLEAR
COL commission_pct CLEAR
CLEAR BREAK
TTITLE OFF
BTITLE OFF
```

(2-3) 명령모드로 나온 후, 편집내용을 저장하고 vi 편집기를 종료합니다.

```
[Esc] 키를 누른 후, :wq 를 입력한 뒤 [Enter]
```

(3) 터미널에서 SQL*Plus를 실행하여 HR 계정으로 데이터베이스에 접속합니다.

```
$ sqlplus hr/oracle4U
```

(4) 생성된 스크립트를 실행하여 보고서를 화면에 출력합니다.

```
SQL> @/home/oracle/emp_report.sql
```

(참고) 다음은 화면에 표시된 결과입니다.

Tue Nov 22		page	1
	Employee Report		
Job Category	EMPLOYEE NAME	Salary	COMMISSION_PCT
AC_ACCOUNT	Gietz	\$8,300.0	No_Commission
AC_MGR	Higgins	\$12,000.0	No_Commission
AD_ASST	Whalen	\$4,400.0	No_Commission
FI_ACCOUNT	Chen	\$8,200.0	No_Commission
	Faviet	\$9,000.0	No_Commission
	Popp	\$6,900.0	No_Commission
	Sciarra	\$7,700.0	No_Commission
	Urman	\$7,800.0	No_Commission
FI_MGR	Greenberg	\$12,000.0	No_Commission
HR_REP	Mavris	\$6,500.0	No_Commission
IT_PROG	Austin	\$4,800.0	No_Commission
	Ernst	\$6,000.0	No_Commission
	Hunold	\$9,000.0	No_Commission
	Lorentz	\$4,200.0	No_Commission
	Pataballa	\$4,800.0	No_Commission
(총략)...	Hall	\$9,000.0	.25
		Confidential	
Tue Nov 22		page	2
(총략)...			