[문서 제목]

# ORACLE DATABASE 11g SQL

테스트

◈ 문서 작성자: 신상현

◈ 최초 작성일: 2006.11.24.

◈ 최종 수정일: 2015.11.22.

◈ 본 문서는 ORACLE SQL 교육과정을 수강하는 학생들의 학습에 도움을 주기 위한 목적으로 작성되었습니다.

◈ 본 문서는 상업적인 용도로 사용하는 것을 절대로 금지하며, 학습을 위한 목적으로만 배포가 가능합니다.
문서를 배포할 때는 문서 작성자의 이름을 삭제하면 절대로 안됩니다.

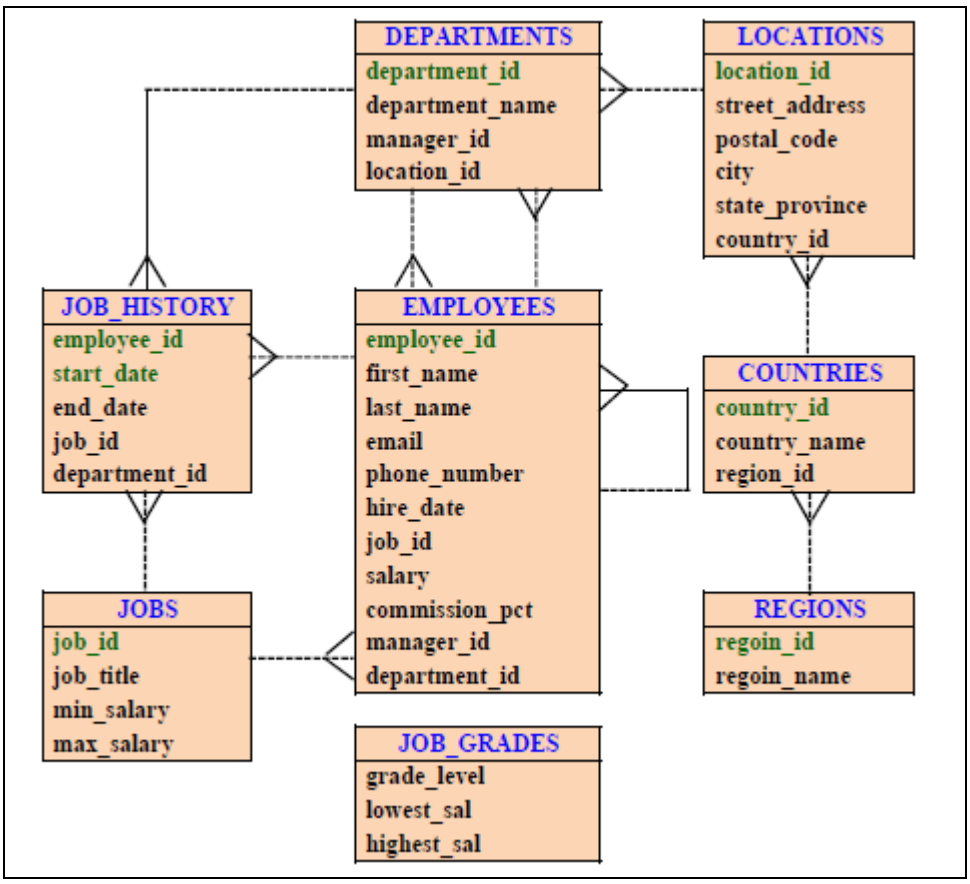
◈ 본 문서의 내용을 복사하거나 본 문서에 주석을 삽입하는 것은 불가능합니다.

◈ 문서 개요.

○ 본 문서는 ORACLE WDP 교육 코스에 포함된 SQL Fundamentals I, II 과목에 대한 교육 내용이 설명되어 있습니다. 문서에서 설명하는 내용에 대한 주제는 문서의 목차를 참고하십시오.

○ 본 문서에서 사용하는 예제들은, 오라클 데이터베이스 서버를 "General Purpose or Transaction Processing" 옵션으로 오라클 데이터베이스 서버를 생성하고, 이 때, [Sample Schema] 옵션을 선택했을 때 제공되는 계정들과 테이블들의 데이터를 이용하며, HR 계정으로 오라클 데이터베이스 접속 계정하여 대부분의 실습을 수행합니다.

○ HR 계정이 소유한 테이블들의 이름과 컬럼들의 이름 및 각 테이블의 관계들은 다음과 같습니다.




○ 위의 각 테이블이 저장하고 있는 정보는 다음과 같습니다.

- EMPLOYEES 테이블은 각 사원에 대한 정보를 포함하며, DEPARTMENTS 테이블의 하위 테이블입니다.
- DEPARTMENTS 테이블은 각 부서에 대한 정보를 포함하며, EMPLOYEES 테이블에 대한 Primary Key 테이블입니다.
- LOCATIONS 테이블은 부서가 위치한 주소 정보를 포함하며, DEPARTMENTS 테이블에 대한 Primary Key 테이블이자 COUNTRIES 테이블의 하위 테이블입니다.
- COUNTRIES 테이블은 부서가 위치한 국가이름 정보를 포함하며, LOCATIONS 테이블에 대한 Primary Key 테이블입니다.

- REGIONS 테이블은 부서가 위치한 국가의 지역 정보를 포함하며, COUNTRIES 테이블에 대한 Primary Key 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 문서는 다음의 문서들을 참고하여 작성되었습니다.

- Oracle® Database SQL Language Reference 메뉴얼.
- Oracle® Database Reference 메뉴얼.
- Oracle Database: SQL Fundamentals I Student Guide.
- Oracle Database: SQL Fundamentals II Student Guide.

○ 다음은 SQL-학습 시에 도움을 주는 오라클 사이트의 URL-주소 입니다.

| 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 SELECT-문.

3. SELECT-문에서 WHERE-절 및 ORDER BY-절 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. 서브쿼리(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).

[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.

■ 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)

■ 웹-서버와 웹-어플리케이션-서버(WAS)의 차이점을 확인하여, 데이터베이스의 필요성을 이해합니다.

## 1-1. Structural Query Language (SQL) 소개.

■ 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 |
|---|---|
| ○ Query-문<br>• SELECT-문 | 데이터 조회어(SELECT문)를 이용하면, 데이터베이스에서 데이터를 검색하여 사용자가 원하는 데이터를 확인 및 사용할 수 있습니다. |
| ○ DML-문<br>• INSERT-문<br>• DELETE-문<br>• UPDATE-문<br>• MERGE-문 | 데이터 조작어 (Data Manipulation Language)을 이용하면, 데이터베이스의 테이블에 새로운 행을 입력하고, 기존 행의 데이터를 변경하고, 테이블에서 불필요한 행을 제거할 수 있습니다. |
| ○ DDL-문<br>• CREATE-문<br>• ALTER-문<br>• DROP-문<br>• TRUNCATE-문<br>• COMMENT-문<br>• RENAM-문 | 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. |
| ○ DCL-문<br>• GRANT-문<br>• REVOKE-문 | 데이터 제어어 (Data Control Language)를 이용하면, 데이터베이스에 접속하는 사용자에게, 데이터베이스와 그 안의 데이터 구조에 대한 액세스 권한을 부여 또는 제거할 수 있습니다. |
| ○ TCL-문<br>• COMMIT-문<br>• SAVEPOINT-문<br>• ROLLBACK-문 | 트랜잭션 제어어 (Transaction Control Language)를 이용하면, 데이터베이스에서 수행된 하나 이상의 DML-문(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.

## 1-2. 데이터베이스와 관련된 기본 용어: 데이터, 정보, 테이블, 행, 컬럼, 레코드, 필드

■ **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 데이터들 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.

| 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 |

← 행

←필드

← 컬럼

**[참고] 데이터베이스 설계에 대한 간단한 이해**

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

이런 상황에서 만약, 신입사원이 새로 입사하게 되면, 10개의 인사 관리 프로그램에서 새로운 직원에 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.

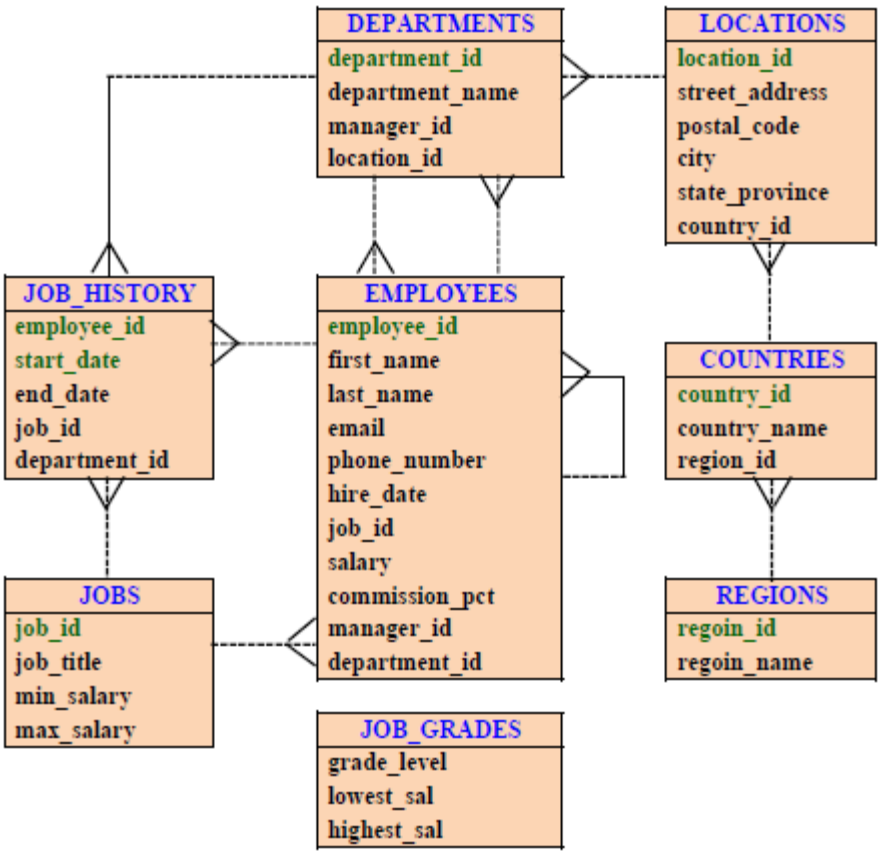
이렇게, 프로그램의 데이터 처리단위인 레코드를 근거로 테이블을 생성하게 되면, 중복된 데이터가 입력되는 것을 피하기 어려울 뿐만 아니라 데이터 관리를 위한 별도의 프로그램까지 개발해야 되는 상황이 초래될 수 있습니다.

위와 같은 문제점들 때문에, 현재는 전문가에 의하여 설계된 데이터 구조를 근거로 테이블을 생성합니다. 즉, 공통적인 목적으로 사용되는 프로그램들에서 사용되는 모든 데이터들을 데이터들의 특성에 따라 분류하여 그룹화시키고, 이 그룹의 단위로 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 사원관련 데이터, 부서관련 데이터 등으로 데이터들의 특성에 따라 분류하여 그룹화시키고, 이렇게 그룹화된 데이터들의 조합을 근거로 데이터베이스에 사원테이블, 부서테이블 등을 생성합니다. 만약, 10 개의 인사관리 프로그램들에서 사원과 관련되어 필요한 데이터들이 있다면, 해당 인사 관리 프로그램들은, 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 관련된 속성들은 사용 목적에 적합하도록 그룹화됩니다. 이렇게 "그룹화된 속성들의 묶음"을 **엔터티(Entity)**라고 하며, 이러한 엔터티 및 엔터티들의 관계를 그림으로 표시한 것이 Entity Relationship Diagram (데이터 구조도, ERD)입니다.
다음은 실습에 사용하는 데이터들의 의미 관계 및 의미들의 묶음을 표시한 그림이며, 이 그림을 바탕으로 최종적인 ERD가 완성됩니다.




이렇게 설계를 통해 완성된 데이터-구조도(ERD)의 엔터티 및 속성들의 정의를 근거로 데이터베이스-테이블을 생성하며, 이 때, 엔터티의 이름은 테이블 이름으로, 또한 각 속성의 이름은 테이블을 구성하는 각 컬럼의 이름으로 구현됩니다.

## 1-3. 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 HTML로 작성된 웹 페이지 문서를, 사용자(CLIENT)에게 서비스하는 서버입니다. 웹 서버를 이용하여 [회원 정보] 웹 페이지 문서를 서비스 하는 경우, 만약 사이트의 회원이 100 만 명이라면, 각 회원의 [회원 정보] 웹 페이지 서비스를 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 작업 입니다.

○ 이 때, [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 저장합니다.
2. 웹 서버에 회원의 정보를 표시하는 형식만 정의된 한 개의 웹 페이지 문서를 작성합니다.
3. 웹 서버에 존재하는 [회원 정보] 페이지가 각각의 사용자(Client)에 의해서 요청될 때, 특정한 기능의 프로그램을 통하여 [웹 페이지에 표시할 데이터를 데이터베이스 서버로부터 가져와서] 해당 웹 페이지에 표시한 후, 데이터가 채워진 웹 페이지를 요청한 사용자(Client)에게 각각 전송합니다.

○ 위의 세 번째 단계에서 특정한 기능의 프로그램은 다음의 기능이 포함됩니다.
(1) 데이터베이스에 접속하여,
(2) SQL-문을 이용하여 필요한 데이터를 가져와서 프로그램의 변수를 통해 웹 페이지에 표시한 후,
(3) 데이터베이스의 접속을 종료합니다.

○ 위의 설명에서처럼 웹 서버 상에 형식이 정의된 웹 페이지 문서 하나를 작성 한 후, 페이지에 표시할 데이터를 [데이터베이스에 접속해서 SQL-문을 통해 가져오는 프로그램]과 연동 시켜서 최종적으로 데이터가 표시된 웹 페이지를 서비스하면, 수 많은 웹 페이지를 작성할 필요가 없어집니다. 이렇게, 웹 서버에 웹 페이지와 연동된 프로그램이 같이 구성된 웹 서버를 [웹 어플리케이션 서버]라고 합니다.

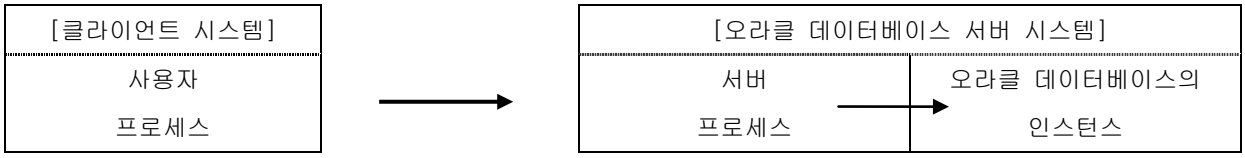
○ 웹 서비스를 구성하는 대부분의 경우에, 서비스 할 데이터를 체계적으로 관리하고 구성하기 위하여 데이터베이스 서버를 사용합니다.

## [참고] 사용자-프로세스(USER PROCESS), 서버-프로세스(SERVER PROCESS), 세션(SESSION) 이란?

○ **사용자-프로세스(User process)**는, 데이터베이스에 접속하여, SQL-문을 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 SQL-문을 전달받아, SQL-문을 처리하는 데이터베이스 서버의 프로세스입니다. 디폴트로 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-서버가 기동되어 운용 중일 때, 시스템의 저장장치에 구성된 오라클 데이터베이스에 대한, 시스템의 메모리에 구성되는 메인-프로세스(또는 서비스-프로세스) 입니다.

○ 실습환경에서 사용자 프로세스로 사용되는 프로그램으로, 오라클 사가 제공하는 SQL*Developer와 오라클 서버 시스템에 설치된 SQL*Plus를 사용합니다.

## [참고] SQL*Plus 툴의 CONNECT 명령어와 SHOW USER 명령어

○ SQL*Plus의 CONNECT 명령어는, 실행 중인 SQL*Plus에서 데이터베이스에 접속할 때 사용됩니다.
또한 이미 접속된 상태에서 다른 사용자 계정으로 접속할 때도 사용할 수 있습니다. 사용자 접속을 변경하면, 이전 사용자의 접속은 정상적으로 종료됩니다.

○ SQL*Plus의 SHOW USER 명령어는, 현재 오라클 데이터베이스 서버에 접속한 계정을 확인할 때 사용됩니다.

■ 오라클 데이터베이스가 구성된 LINUX 가상머신에서 터미널에서 다음을 수행하여, SQL*Plus의 CONNECT 명령어와 SHOW USER 명령어에 대한 사용 방법을 연습합니다.

```
[oracle@ora112:orcl:~]$ sqlplus /nolog   — /nolog 옵션은 접속은 하지 않고, SQL*Plus 툴만 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>
SQL> CONNECT / AS SYSDBA          — CONNECT 명령어로 SYS 계정으로 접속합니다.
Connected.
SQL>
SQL> SHOW USER                    — SHOW USER 명령어로 접속한 계정이름을 확인합니다.
USER is "SYS"
SQL>
SQL> CONN hr/oracle4U             — CONNECT를 간단히 CONN 으로 실행시켜 HR 계정으로 접속합니다.
Connected.                        — 이 때, 기존 SYS 계정의 접속은 해제됩니다.
SQL>
SQL> SHOW USER                    — SHOW USER 명령어로 접속한 계정이름을 확인합니다.
USER is "HR"
SQL>
```

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

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

# 2 단순 SELECT-문.

◈ 학습 목표.

■ 단순 SELECT-문의 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-캐랙터-스트링을 이용하여 표시 결과에 상수를 포함시키는 방법을 학습하고, q 연산자의 기능 및 사용방법을 학습합니다.

■ Concatenation 연산자(||) 및 DISTINCT 키워드의 기능 및 사용방법을 학습합니다.

## 2-1. SELECT-문 (쿼리) 사용 목적.

○ 데이터베이스에 저장된 데이터를 **사용자 프로그램에서 사용**하기 위하여 **SELECT-문을 이용하여 질의**할 수 있습니다.
즉, 프로그램에서 처리할 데이터를 사용자 프로그램으로 가져오기 위하여, SELECT-문을 사용합니다.

## 2-2. 단순 SELECT-문의 기본 문법.

○ 단순 SELECT-문의 기본 문법.

```
SELECT 컬럼1, 컬럼2, 컬럼-표현식   -- 옆에서 SELECT 키워드로 시작된 라인을 SELECT-절 이라고 합니다.
FROM 소유자명.테이블이름           -- 옆에서 FROM 키워드로 시작된 라인을 FROM-절 이라고 합니다.
WHERE 행을 선택할 조건식           -- 옆에서 WHERE 키워드로 시작된 라인을 WHERE-절 이라고 합니다.
ORDER BY 정렬기준 ;                -- 옆에서 ORDER BY 키워드로 시작된 라인을 ORDER BY-절 이라고 합니다.
```

☞ **단순** SELECT-문은 하나의 테이블에 대하여 질의할 때 사용하는 문장이며, 따라서, FROM 절에 테이블이름을 오직 하나만 명시합니다.

☞ 위에서 SELECT-절에는, 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 FROM-절에는, SELECT-절에 명시된 컬럼(들)이 정의된 테이블이름을 명시합니다. 이 때, 테이블이름 앞에 테이블을 소유한 계정이름(이를 스키마이름 이라고 합니다)을 명시하는 것을 권장합니다.

☞ 위에서 WHERE-절에는, SELECT-문에 의하여 처리되는 행(들)을 선택할(SELECTION) 조건을 기술합니다.

☞ 위에서 ORDER BY-절에는, 결과 레코드를 정렬할 기준을 명시합니다.

## 2-3. SQL-문 작성 규칙.

○ SQL-문에서 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 SELECT-문을 이용한 대표적인 데이터 조회 방법.

■ HR.EMPLOYEES 테이블에서 EMPLOYEE_ID, LAST_NAME, SALARY, EMAIL 컬럼의 데이터로만 구성된 결과를 조회하시오.

```
SQL> SELECT employee_id, last_name, salary, email        --SELECT 절에 원하는 컬럼 이름을 명시하여,
     FROM hr.employees;                                  --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        --WHERE 절이 누락된 경우,
        202 Fay                            6000 PFAY              테이블의 모든 행에서 레코드가 추출됩니다.
...

107개의 행이 선택됨
```

☞ SELECT-절에 사용자가 원하는 컬럼 이름을 명시하여, 원하는 컬럼의 데이터를 표시할 수 있습니다.

☞ 사용자의 SQL-문은 오라클-서버 상에서, 테이블의 행 단위로 처리됩니다.

■ HR.DEPARTMENTS 테이블에 정의된 모든 컬럼의 데이터를 조회하시오.

```
SQL> SELECT *            -- SELECT 절에 * 을 명시하면, 테이블을 구성하는 모든 컬럼의 데이터가 표시됩니다
     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 컬럼 이름 대신, * 을 명시하면, 테이블을 구성하는 모든 컬럼의 데이터가 표시됩니다.

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

○ SQL*Plus 및 SQL*Developer 툴의 **DESCRIBE** 명령어를 이용하여 테이블을 구성하는 컬럼이름과 데이터 유형을 확인할 수 있습니다. DESCRIBE 명령어를 간단히 **DESC** 로 줄여서 사용합니다.

■ 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 데이터 유형으로서 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 데이터 유형입니다.
JOB_ID          NOT NULL VARCHAR2(10)
SALARY                   NUMBER(8,2)
COMMISSION_PCT           NUMBER(2,2)
MANAGER_ID               NUMBER(6)      -- NUMBER(6)는 숫자(실수) 데이터 유형으로서 최대 6글자 길이의
DEPARTMENT_ID            NUMBER(4)         숫자 값을 처리할 수 있는 데이터유형입니다.
```

☞ DESCRIBE 명령어는 SQL-문장이 아닙니다. 오라클 사의 SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어입니다. 따라서, 다른 회사의 데이터베이스 제품에서는 제공되지 않습니다.

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

☞ 본 문서에서는 SQL문장과 SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어를 쉽게 구분하기 위하여, SQL*Developer 및 SQL*Plus 프로그램에서 제공하는 명령어에 대해서는 세미콜론(;)을 명시하지 않습니다.

## 2-5. NUMBER 또는 DATE 데이터 유형 컬럼에 대한 산술 연산자 처리.

○ 컬럼의 데이터 유형이 NUMBER 또는 DATE 형식일 때, 산술 연산자를 이용하여 계산된 결과를 얻을 수도 있습니다.

○ 데이터 유형에 따라 사용할 수 있는 산술연산자에 대하여 차이가 있습니다.

| 데이터 유형 | 사용할 수 있는 산술연산자 |
|---|---|
| NUMBER | [ 더하기, +] [ 빼기, - ] [ 곱하기, * ] [ 나누기, / ] |
| DATE | [ 더하기, +] [ 빼기, - ] |

■ HR.EMPLOYEES 테이블을 이용하여, 부서ID(DEPARTMENT_ID 컬럼)가 90인 부서에서 근무하는 직원들의, 직원ID (EMPLOYEE_ID 컬럼), 직원의 이름(FIRST_NAME 컬럼), 급여(SALARY 컬럼) 및 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                    --SALARY*0.05 및 12*(salary+100)을 표현식이라고 합니다.
     WHERE department_id=90;              --WHERE 절의 조건을 만족하는 행으로부터 레코드가 추출됩니다.

EMPLOYEE_ID FIRST_NAME              SALARY SALARY*0.05 12*(SALARY+100)  --12*(salary+100)에서 괄호로 묶인
----------- -------------------- ---------- ----------- ---------------    (salary+100)이 먼저 계산된 후에,
        100 Steven                    24000        1200          289200    12가 곱해집니다.
        101 Neena                     17000         850          205200
        102 Lex                       17000         850          205200
```

☞ SELECT-문에 WHERE-절이 포함되면, 테이블에서 WHERE-절에 명시된 조건을 만족하는 행을 찾아서, SELECT-절에 명시된 컬럼 및 컬럼-표현식으로 구성된 레코드가 추출되어 사용자 프로그램으로 전달되어 표시됩니다.

## 2-6. 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.

```
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.

■ HR.EMPLOYEES 테이블에서 모든 직원들의 LAST_NAME, SALARY, 및 SALARY*(1+COMMISSION_PCT)*12 표현식으로 구성된 레코드의 데이터를 조회하시오.

```
SQL> SELECT last_name, salary, commission_pct, salary*(1+commission_pct)*12
     FROM hr.employees ;

LAST_NAME                   SALARY COMMISSION_PCT SALARY*(1+COMMISSION_PCT)*12
------------------------ ---------- -------------- ----------------------------
...
Matos                         2600
Vargas                        2500
Russell                      14000             .4                       235200
Partners                     13500             .3                       210600
Errazuriz                    12000             .3                       187200
...
107개의 행이 선택됨
```

<-- 표시되는 값이 없습니다.

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

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

## 2-7. 컬럼 별칭(Column Alias)의 사용한 출력결과의 머리글(Heading) 변경하기.

○ SELECT-문에 의하여 출력된 결과에서, 레코드들 위에 표시된 각 필드의 제목을 머리글(Heading) 이라고 하고, SELECT-절에 명시한 컬럼이름이나 컬럼표현식이 그대로 각 필드의 머리글(Heading)로 표시되며, 디폴트로 머리글은 대문자로 표시됩니다.

○ SELECT-절에서 컬럼이름이나 컬럼-표현식 다음에 아래의 방법으로 컬럼-별칭을 설정하여 SELECT-문의 처리 결과에 표시되는 머리글(Heading)을 변경할 수 있습니다.

| SELECT-절에서 컬럼 별칭을 정의하는 방법 | 설명 |
|---|---|
| [표준문법]<br>**컬럼표현식 AS "원하는 별칭"** | • 큰따옴표["]를 명시하면 둘 이상의 단어를 지정할 수 있습니다.<br>• 큰따옴표["]를 명시하면 명[대/소]문자가 구분되어 표시됩니다. |
| [약식문법]<br>**컬럼표현식 원하는_별칭** | • AS 키워드를 생략하고 간단히 빈칸을 이용합니다.<br>• 컬럼 별칭이 한 단어이면 큰따옴표["]를 생략 할 수 있습니다.<br>☞ 큰 따옴표["]를 생략하면 머리글이 대문자로 표시됩니다. |

■ HR.EMPLOYEES 테이블에서 모든 직원의 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*(1+commission_pct)*12 Ann_Sal
     FROM hr.employees ;

Last Name                      SAL    ANN_SAL
------------------------ ---------- ----------  --Last Name 머리글은, 두 단어이므로 큰 따옴표로 감싸주었으며,
OConnell                       2600               큰 따옴표로 감쌌으므로, 대소문자도 구분되어 표시되었습니다.
Grant                          2600
Whalen                         4400             --Ann_Sal 머리글은 AS 도 생략되었고, 한 단어이므로,
Hartstein                     13000               약식으로 큰 따옴표를 생략했으므로, 대문자로 표시되었습니다
Fay                            6000
...

107개의 행이 선택됨
```

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

## 2-8. Concatenation 연산자 (||)의 사용.

○ Concatenation 연산자(||)는 연산자 앞 뒤로 명시된 두 개의 문자열을 붙입니다.

■ HR.EMPLOYEES 테이블에서 EMPLOYEE_ID가 100 인 직원의 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                            --||연산자 앞 뒤의 컬럼 중, 하나의 컬럼이 NULL인 경우에는,
----------------------------------  NULL이 아닌 다른 컬럼의 값이 출력됩니다.
KingAD_PRES                         두 컬럼이 모두 NULL이면, 표시되는 값이 없습니다.
```

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

## 2-9. 리터럴 캐릭터 스트링(Literal Character Strings).

○ SELECT문의 출력 결과 레코드에 같이 표시된 '문자(열)', '날짜', 숫자를 "**리터럴 캐릭터 스트링**"이라고 합니다.

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

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID가 90 인 직원의 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                 'ISA'         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 캐릭터 스트링 입니다.

## 2-10. q 연산자 (10gNF, 오라클에서만 가능).

○ SELECT절에, 명시한 리터럴 캐릭터 스트링 안에 처리될 수 없는 특수 문자 (앞의 실습에서 작은 따옴표(') 같은) 등이 포함된 경우, 오라클 10g 버전부터는 **q 연산자**를 이용하여 오류 없이 처리할 수 있습니다.

○ q 연산자의 사용방법을 다음의 예제를 통하여 살펴봅니다.

■ HR.DEPARTMENTS 테이블에서 DEPARTMENT_ID가 10인 부서의 DEPARTMENT_NAME, MANAGER_ID 컬럼의 데이터를 조회하되, q 연산자를 사용하여, 두 값 사이에 ', it's assigned Manager id:'가 포함되어 모두 하나의 값으로 표시되도록 하시오. 그리고 결과의 HEADING이 "Department and Manager"로 표시되도록 하시오.

```
SQL> SELECT department_name || q'[, it's assigned Manager id:]' ||manager_id
           AS "Department and Manager"
     FROM hr.departments
     WHERE department_id=10 ;

Department and Manager
-----------------------------------------------------------------------------------------------
Administration, it's assigned Manager id:200
```

☞ 표기 방법은 다음처럼 리터럴 캐릭터 스트링을 각진 괄호([ ])로 감싸주고, 그것을 다시 작은 따옴표(')로 감싸준 후, 앞에 q를 명시합니다.

```
q '[사용자가 표시하고 싶은 리터널 캐렉터 스트링]'
```

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

## 2-11. DISTINCT 키워드를 이용한 중복된 레코드 제거.

○ SELECT-문의 출력결과에 표시되는 레코드들은, 기본적으로 선택된 행으로부터 추출된 모든 레코드(중복된 레코드를 포함)가 표시됩니다(아래의 예제).

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID가 30인 직원에 대하여 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, DISTINCT 키워드를 이용합니다. 즉, SELECT 키워드 뒤에 DISTINCT 키워드를 명시하여, 중복된 레코드를 "한 번만 출력"시킬 수 있습니다.

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID가 30인 직원에 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.

**[참고] SQL*Developer에서 한번에 여러 개의 SQL-문들을 실행하는 방법.**

○ SQL*Developer에서 여러 SQL-문들을 실행하는 방법을 학습합니다.

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

1> SQL*Developer에서 HR 계정으로 접속합니다.

2> [워크시트] 창에 아래의 SQL-문들을 작성합니다.

```
CREATE TABLE HR.JOB_GRADES
(GRADE_LEVEL  VARCHAR2(2)  PRIMARY KEY
,LOWEST_SAL   NUMBER(10)
,HIGHEST_SAL  NUMBER(10) ) ;
INSERT INTO HR.JOB_GRADES VALUES ('A',  1000,  2999) ;
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) ;
COMMIT ;
```

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

☞ 위에서 [워크시트]에 내용을 작성하는 대신, 윈도우즈-운영체제의 [메모장]에서 위의 SQL-문들을 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 SELECT-문에서 WHERE-절 및 ORDER BY-절 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. SQL-문에서 WHERE-절을 사용하는 목적.

■ 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. WHERE-절에서 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, 상수가 영문자인 경우에는 [대/소]문자가 구분됩니다.

■ HR.EMPLOYEES 테이블에서 LAST_NAME이 'Whalen' 이고 DEPARTMENT_ID가 10 인 직원의 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  --WHERE 절에 명시된 문자상수('Whalen')와 LAST_NAME 필드에
-------------------- ---------- -------------    표시된 값의 철자 및 대소문자가 모두 일치합니다.
Whalen               AD_ASST               10
```

○ 위의 실습에서 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      --'WHALEN' 상수값의 대소문자가 일치되는
     FROM hr.employees                            --LAST_NAME 컬럼의 값이 없기 때문에,
     WHERE last_name = 'WHALEN'                   --조건을 만족하는 행이 없습니다.
       AND department_id = 10 ;

선택된 행 없음
```

○ WHERE 절에서 날짜 데이터유형 컬럼과 비교되는 날짜 상수를 명시할 때는 '작은 따옴표'로 감싸주며, 이 때, 날짜상수는 데이터베이스에 접속한 세션에서 표시되는 형식을 준수하여 명시해야만 합니다.

○ 다음의 [참고] 내용을 확인하십시오.

**[참고] 접속한 세션(CLIENT)에서의 DATE 데이터 유형을 표시하는 형식.**

○ WHERE 절의 조건 작성 시에 [DATE 데이터 유형의 컬럼]과 비교되는 상수값을 기술 시에는, 세션(즉, CLIENT)에서의 표시 형식(DISPLAY-FORMAT)을 지켜서 기술해야 합니다. 만약 WHERE 절에 기술된 DATE 데이터 유형의 상수 값이 세션의 표시 형식과 다를 경우, 잘못된 처리가 수행되거나 또는 문장 실행 중에 오류가 발생될 수 있습니다.

■ 현재 접속한 데이터베이스 서버에 대하여 SYSDATE 함수 처리 결과를 확인하시오.

```
SQL> SELECT sysdate FROM dual ;

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

☞ 오라클의 SYSDATE 함수는 데이터베이스 서버의 현재 날짜 및 시간을 DATE 데이터유형으로 반환합니다.

☞ 데이터베이스에 접속한 SQL*Developer 세션에서 위의 문장을 실행하면, 데이터베이스의 현재 날짜 및 시간을, 접속한 SQL*Developer가 전달받아 SQL*Developer에 설정된 DATE 데이터 유형의 표시 형식대로 현재 날짜 및 시간이 표시됩니다.

☞ 한글 Windows 환경에 구성된 SQL*Developer에서 데이터베이스 서버에 접속한 경우에는 DATE 데이터 유형의 표시 형식은 RR/MM/DD 입니다.

■ HR.EMPLOYEES 테이블을 이용하여, 입사일(HIRE_DATE 컬럼)이 2008년1월1일 이 후인 직원들의 EMPLOYEE_ID, LAST_NAME, HIRE_DATE 컬럼으로 구성된 데이터를 조회하시오.

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

EMPLOYEE_ID LAST_NAME            HIRE_DATE   --WHERE 절의 날짜 상수를 접속한 세션의 날짜 표시형식인
----------- -------------------- ---------     RR/MM/DD 형식과 동일하게 명시하였으므로,
        199 Grant                08/01/13      정상적으로 처리된 결과가 표시됩니다.
        128 Markle               08/03/08
        136 Philtanker           08/02/06
        149 Zlotkey              08/01/29
        164 Marvins              08/01/24
        165 Lee                  08/02/23   -- HIRE_DATE의 날짜값도 RR/MM/DD 형식으로 표시됩니다.
        166 Ande                 08/03/24
        167 Banda                08/04/21
        173 Kumar                08/04/21
        179 Johnson              08/01/04
        183 Geoni                08/02/03

11개의 행이 선택됨
```

☞ 한글 윈도우에 설치된 SQL*Developer를 이용하여 데이터베이스에 접속하여 실행한 SELECT-문의 WHERE-절에 명시하는 날짜 상수를, YYYY/MM/DD, YYYYMMDD, YYYY-MM-DD, RR/MM/DD, RRMMDD, 또는 RR-MM-DD의 표시형식으로 명시해도 동일한 처리 결과를 얻을 수 있습니다. 단 SELECT-문의 실행 결과에 표시되는 표시 형식은 RR/MM/DD 형식으로 표시됩니다.

☞ 만약, WHERE 절에서 **세션의 날짜 표시 형식과 다른 형식으로 날짜 형식의 상수 값을 기술한 경우**에는 SELECT-문이 정상적으로 처리되지 못하고 오류가 발생되거나 또는 잘못된 처리가 수행될 수 있습니다.

**[참고] WHERE 절에서 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
```

## 3-3. WHERE 절 작성 시에 사용되는 연산자 종류.

○ 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 연산자라고 부르며, 컬럼의 값과 기술한 상수 값이 비교됩니다.

| 연산자 | 연산자의 의미 |
|---|---|
| = | 같다 |
| <> , != , ^= | 같지 않다 |
| > | 크다 |
| < | 작다 |
| >= | 크거나 같다 |
| <= | 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.

■ HR.EMPLOYEES 테이블에서 SALARY 값이 3000보다 작거나 같은 모든 직원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary
     FROM hr.employees
     WHERE salary <= 3000;      -- 총 107개의 행 중, WHERE 절을 만족하는 26개 행으로부터
                                   추출된 레코드가 표시되었습니다.
LAST_NAME                  SALARY
------------------------ ----------
OConnell                     2600
Grant                        2600
Baida                        2900
Tobias                       2800
Himuro                       2600
...

26개의 행이 선택됨
```

## 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. 또한 영어의 소문자가 대문자 보다 큽니다. 날짜의 경우에는 최근 날짜 일수록 크며, 과거 날짜 일수록 작습니다.

■ HR.EMPLOYEES 테이블에서 SALARY 값이 4000과 4500 사이에 있는 모든 직원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오.

```
SQL> SELECT last_name, salary
     FROM hr.employees
     WHERE salary BETWEEN 4000 AND 4500 ;

LAST_NAME                    SALARY
------------------------ ----------
Whalen                         4400
Lorentz                        4200
Sarchand                       4200
Bull                           4100
Bell                           4000
```

☞ 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, 다음처럼 '선택된 행 없음' 메시지가 표시됩니다.

```
SQL> SELECT last_name, salary
     FROM   hr.employees
     WHERE  salary BETWEEN 4500 AND 4000 ;

선택된 행 없음
```

## 3-3-3. IN (값1, 값2,..)연산자.

| | |
|---|---|
| ○ 연산자 의미 | 컬럼의 값이 명시된 값들 중 하나와 같은지 비교합니다. |
| ○ 사용 시 주의 사항 | IN 연산자에 명시된 모든 값의 데이터 유형이 동일해야 합니다. |

■ HR.EMPLOYEES 테이블에서 MANAGER_ID 컬럼의 값이 100 또는 101 또는 201 인 모든 직원의 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
        120 Weiss                          8000        100
...

 20개의 행이 선택됨
```

## 3-3-4. LIKE 연산자.

| | |
|---|---|
| ○ 연산자 의미 | 문자 데이터유형의 컬럼의 값이 명시된 문자열의 패턴과 동일한지 비교합니다.<br>(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.<br>• 퍼센트 문자(%)는 명시된 위치에 0 개 이상의 여러 문자가 존재할 수 있습니다.<br>• 언더스코어 문자(_)는 명시된 위치에 반드시 1개의 문자가 존재해야 합니다. |

■ HR.EMPLOYEES 테이블에서 LAST_NAME 값이 'A'로 시작하는 모든 직원의 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
```

■ HR.EMPLOYEES 테이블에서 LAST_NAME의 2번째 글자가 'b'이고 최소한 3글자 이상인 모든 직원의 LAST_NAME, SALARY 컬럼의 데이터를 표시하시오

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

LAST_NAME                    SALARY
------------------------ ----------
Abel                          11000
```

## [참고] LIKE 연산자 사용 시에, ESCAPE 식별자를 같이 사용해야 하는 경우.

○ 테이블의 문자 데이터 유형 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.

■ 다음의 일련의 실습을 통해 LIKE 연산자를 사용할 때 ESCAPE 식별자를 명시하는 방법을 확인해 봅시다.

1> 실습용 테이블을 생성합니다.

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

table HR.TEST01이(가) 생성되었습니다.
```

2> 실습용 데이터를 입력하고 트랜잭션을 커밋합니다.

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

1개 행 이(가) 삽입되었습니다.

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

1개 행 이(가) 삽입되었습니다.

SQL> INSERT INTO HR.TEST01 VALUES ('AD_%dfkml') ;

1개 행 이(가) 삽입되었습니다.

SQL> COMMIT ;

커밋되었습니다.
```

3> J_CODE 컬럼의 데이터가 'AD_'로 시작하는 행만 표시하기 위하여 아래의 SELECT-문을 작성하여 실행합니다.

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

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 '\';

J_CODE
------------------------------
AD_PRES
AD_%dfkml
```

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 '\';

J_CODE
------------------------------
AD_%dfkml
```

☞ ESCAPE 식별자로 지정한 문자가, 패턴에 표시한 위치 다음에 있는 _ 또는 % 문자는 값에 포함된 문자로 처리됩니다.

5> 실습용 테이블을 삭제합니다.

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

table HR.TEST01이(가) 삭제되었습니다.
```

## 3-3-5. WHERE 절에서 IS NULL 키워드 사용하기.

○ 오라클에서 NULL은 상태를 의미하기 때문에, WHERE 절에서 [컬럼=NULL]로 기술할 수 없습니다. 만약 WHERE 절을 [WHERE 컬럼=NULL]로 기술하면 결과는 항상 "선택된 행 없음 또는 no rows selected"라는 메시지가 반환됩니다.

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

선택된 행 없음
```

○ 컬럼이 NULL 인 상태에 있다는 것을 WHERE 절에 기술할 때는 [WHERE 컬럼 IS NULL]을 사용합니다.

■ HR.EMPLOYEES 테이블에서 MANAGER_ID 컬럼이 NULL 인 모든 직원의 LAST_NAME 데이터를 표시하시오.

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

LAST_NAME
-------------------------
King
```

## 3-3-6. AND 또는 OR 논리 연산자.

○ WHERE 절에 둘 이상의 조건절을 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. |

■ HR.EMPLOYEES 테이블에서 MANAGER_ID 컬럼이 NULL 이고, SALARY가 10000보다 큰 모든 직원의 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
```

■ HR.EMPLOYEES 테이블에서 MANAGER_ID 컬럼이 NULL 이거나 또는 SALARY가 10000 보다 큰 모든 직원의 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
```

## 3-3-7. NOT 논리 연산자.

○ WHERE 절에서 기술된 조건을 부정하도록 기술할 수 있습니다.

| 기술 방법 | 의미 |
|---|---|
| WHERE NOT (조건1) | [조건1]을 만족하지 않는 행만 처리합니다. |

○ [NOT 연산자]는 아래의 왼편에서처럼 조건절 앞에 명시하여 [조건절 자체를 부정] 하는 것보다는,
아래의 오른편에서처럼 연산자 앞에 NOT 연산자를 명시하여 [부정의 연산자를 사용]하는 방법으로 주로 사용됩니다.

■ HR.EMPLOYEES 테이블에서 MANAGER_ID 컬럼이 NULL이 아닌 모든 직원의 LAST_NAME 데이터를 표시하시오.

방법1: 조건절 자체를 부정하는 방법.

```
SQL> SELECT  last_name
     FROM   hr.employees
     WHERE  NOT (manager_id IS NULL) ;

LAST_NAME
------------------------
Abel
Ande
Atkinson
Austin
Baer
Baida
Banda
Bates
Bell
Bernstein
Bissot
Bloom
...

 106개의 행이 선택됨
```

방법2: 조건절에서 부정의 연산자를 사용하는 방법.

```
SQL> SELECT  last_name
     FROM   hr.employees
     WHERE  manager_id IS NOT NULL ;

LAST_NAME
------------------------
Abel
Ande
Atkinson
Austin
Baer
Baida
Banda
Bates
Bell
Bernstein
Bissot
Bloom
...

 106개의 행이 선택됨
```

[참고] WHERE절에서 연산자에 대하여 NOT 연산자의 사용 예.

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

## 3-4. ORDER BY 절.

■ 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,<br>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.

1> HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 60 인 직원의 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
```

2> HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 60 인 직원의 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
```

3> HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 60 인 직원의 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   --먼저, SALARY를 기준으로 내림차순으로 레코드를 정렬합니다.
----------- -------------------- ----------     그 다음에 SALARY의 값이 동일한 레코드에 대하여
        103 Hunold                     9000     EMPLOYEE_ID를 기준으로 내림차순으로 정렬합니다.,
        104 Ernst                      6000
        106 Pataballa                  4800
        105 Austin                     4800
        107 Lorentz                    4200
```

4> HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 60 인 직원의 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   --먼저, SALARY를 기준으로, 오름차순으로 레코드를 정렬합니다.
----------- -------------------- ----------     그 다음에 SALARY의 값이 동일한 레코드에 대하여,
        106 Pataballa                  4800     EMPLOYEE_ID를 기준으로 내림차순으로 정렬합니다.
        105 Austin                     4800
        107 Lorentz                    5460
        104 Ernst                      6000
        103 Hunold                     9000
```

5> HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 60 인 직원의 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 AS 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 -- ANNSAL을 기준으로, 오름차순으로 레코드가 정렬되었습니다.
        105 Austin                      57600
        106 Pataballa                   57600
        104 Ernst                       72000
        103 Hunold                     108000
```

[참고] 정렬기준 컬럼이 NULL 인 레코드의 정렬

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

1> HR.EMPLOYEES 테이블에서 모든 직원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID를 기준으로 오름차순 및 내림차순으로 각각 표시하시오

```
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                --정렬기준 컬럼이 NULL인
                           레코드가 제일 마지막에
 107개의 행이 선택됨       표시됩니다.
```

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

EMPLOYEE_ID DEPARTMENT_ID
----------- -------------
        178                --정렬기준 컬럼이 NULL인
        205           110   레코드가 제일 처음에
        206           110   표시됩니다.
        112           100
        109           100
...

107개의 행이 선택됨
```

[참고] ORDER BY 절에 NULLS FIRST 및 NULLS LAST 절 사용

■ ORDER BY 절에서 NULL이 포함된 컬럼을 정렬기준으로 사용하는 경우, NULLS FIRST 절이 명시되면, NULL 이 포함된 레코드들이 제일 먼저 표시됩니다.

■ ORDER BY 절에서 NULL이 포함된 컬럼을 정렬기준으로 사용하는 경우, NULLS LAST 절이 명시되면, NULL 이 포함된 레코드들이 맨 마지막에 표시됩니다.

1> HR.EMPLOYEES 테이블에서 모든 직원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID를 기준으로 오름차순 및 내림차순으로 각각 표시하시오. 단, DEPARTMENT_ID 필드가 NULL인 레코드를 제일 처음에 표시하시오.

```
SQL> SELECT employee_id, department_id
     FROM hr.employees
     ORDER BY 2 ASC NULLS FIRST; --오름차순

EMPLOYEE_ID DEPARTMENT_ID
----------- -------------
        178               --정렬기준 컬럼이 NULL인
        200            10   레코드가 제일 처음에
        201            20   표시됩니다.
        202            20
...

 107개의 행이 선택됨
```

```
SQL> SELECT employee_id, department_id
     FROM hr.employees
     ORDER BY 2 DESC NULLS FIRST; --내림차순

EMPLOYEE_ID DEPARTMENT_ID   --내림차순
----------- -------------
        178               --정렬기준 컬럼이 NULL인
        205           110   레코드가 제일 처음에
        206           110   표시됩니다.
        112           100
...

 107개의 행이 선택됨
```

2> HR.EMPLOYEES 테이블에서 모든 직원의 EMPLOYEE_ID와 DEPARTMENT_ID로 구성된 레코드를 DEPARTMENT_ID를 기준으로 오름차순 및 내림차순으로 표시하시오. 단, DEPARTMENT_ID 필드가 NULL인 레코드를 맨 마지막에 표시하시오.

```
SQL> SELECT employee_id, department_id
     FROM hr.employees
     ORDER BY department_id  ASC NULLS LAST;

EMPLOYEE_ID DEPARTMENT_ID  -- 오름차순
----------- -------------
...
        109           100
        206           110
        205           110
        178               --정렬기준 컬럼이 NULL인
                            레코드가 제일 마지막에
                            표시됩니다.
 107개의 행이 선택됨
```

```
SQL> SELECT employee_id, department_id
     FROM hr.employees
     ORDER BY department_id DESC NULLS LAST;

EMPLOYEE_ID DEPARTMENT_ID   -- 내림차순
----------- -------------
...
        202            20
        201            20
        200            10
        178               --정렬기준 컬럼이 NULL인
                            레코드가 제일 마지막에
 107개의 행이 선택됨        표시됩니다.
```

# 4 단일-행 함수(SINGLE-ROW FUNCTION).

## ◈ 학습 목표.

■ SQL-문에서 사용할 수 있는 다양한 오라클 단일 행 함수에 대하여 학습합니다.

■ 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.

## 4-1. 단일 행 함수(SINGLE ROW FUNCTION) 개요.

○ 아래의 UPPER() 함수는 WHERE절을 만족하는 행에서 레코드가 구성될 때마다 행의 last_name 컬럼의 데이터에 대하여 한번씩 실행되어 각 레코드 마다 함수 처리 결과를 반환합니다.

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 90인 모든 직원의 저장된 그대로의 LAST_NAME, 과 대문자로 표시되는 LAST_NAME 및 SALARY로 구성된 레코드를 표시하시오. 단, 표시된 결과에서 대문자로 표시되는 LAST_NAME의 HEADING은 UPPER_NAME으로 표시되도록 하시오.

```
SQL> SELECT last_name, UPPER(last_name) AS "UPPER_NAME", salary
     FROM hr.employees
     WHERE department_id = 90 ;

LAST_NAME                 UPPER_NAME                   SALARY
------------------------- ------------------------- ----------
King                      KING                          24000
Kochhar                   KOCHHAR                       17000
De Haan                   DE HAAN                       17000
```

## 4-2. 다중 행 함수 (MULTIPLE ROW FUNCTION, GROUP FUNCTION) 개요.

○ 아래의 SUM() 함수는 WHERE절을 만족하는 각각의 행으로부터 SALARY 값을 모두 구한 후 한번 실행되어 추출된 모든 SALARY 데이터들을 합산하여 하나의 결과를 반환합니다.

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 90인 모든 직원의 월급의 합계를 표시하시오.

```
SQL> SELECT SUM(salary)
     FROM hr.employees
     WHERE department_id = 90;

SUM(SALARY)
-----------
      58000
```

☞ 다중 행 함수에 대해서는 다음 레슨에서 설명합니다.

## 4-3. 단일 행 함수 분류.

○ 함수가 처리하는 값의 데이터 유형(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 SQL-문 처리 중에 다른 데이터 유형으로 변경할 필요가 있을 때 사용됩니다.<br>• [숫자, 날짜] 데이터를 문자 데이터로 변환합니다.<br>• [문자] 데이터를 날짜 데이터로 변환합니다.<br>• [문자] 데이터를 숫자 데이터로 변환합니다. |

☞ 오라클-서버에 접속하는 모든 계정이 함수의 테스트 등을 위하여 사용할 수 있도록 DUAL 테이블이 제공됩니다. DUAL 테이블에 대하여 SELECT-문을 실행시키면, 항상 하나의 레코드가 반환됩니다.

```
SQL> DESC  DUAL

이름    널 유형
----- - ----------
DUMMY   VARCHAR2(1)
```

## 4-4. 문자 함수-1 : [대/소]문자 변환 함수.

○ [대/소]문자 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.

■ 'abcd efg high' 문자열 값을 각각 모두 대문자, 소문자, 첫-글자 대문자 나머지는 소문자로 표시하시오.

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

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

■ HR.EMPLOYEES 테이블에서 DEPARTMENT_ID 컬럼이 20인 모든 직원의 LAST_NAME을 모두 대문자, JOB_ID는 소문자로 표시하되, 표시결과가 The job id for LAST_NAME is job_id 형태로 표시되도록 하시오. 결과 상의 HEADING은 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
```

■ HR.EMPLOYEES 테이블에서 LAST_NAME이 Higgins 인 직원의 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
```

[참고] SELECT-문에서 대/소문자 변환함수의 사용 예,

○ 문자 데이터유형의 컬럼에, 저장된 영어로 된 데이터에 대하여 [대/소]문자 입력 규칙을 지정하지 않아서 [대/소]문자의 위치가 불규칙하게 섞여 있는 컬럼을 WHERE 절의 조건에서 사용하는 경우에 컬럼에 대하여 [대/소]문자 변환함수의 사용 유무에 따라서 다른 결과 레코드가 반환될 수 있습니다.

○ 예를 들어, 아래처럼 사원테이블의 이름컬럼에 아래처럼 [대/소]문자가 불규칙한 동일한 철자의 값들이 있다고 합시다.

```
이름컬럼
---------
SHIN
shin
Shin
```

• WHERE 절에서 이름컬럼을 대/소문자 변환 함수로 처리하지 않으면 SHIN 인 행만 선택됩니다.

```
SQL> SELECT 이름컬럼
     FROM 사원테이블
     WHERE 이름컬럼 = 'SHIN';

이름컬럼
---------------
SHIN
```

• WHERE 절에서 이름컬럼을 대/소문자 변환 함수로 처리하면 [대/소]문자에 상관없이 철자가 SHIN 인 모든 행이 선택됩니다.

```
SQL> SELECT 이름컬럼
     FROM 사원테이블
     WHERE UPPER(이름컬럼) = 'SHIN';

이름컬럼
---------------
SHIN
shin
Shin
```

■ HR.EMPLOYEES 테이블에서 LAST_NAME이 [대/소]문자에 상관없이 'higgins' 인 직원의 EMPLOYEE_ID, LAST_NAME, DEPARTMENT_ID로 구성된 레코드를 표시하시오.

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

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

☞ 만약 위의 문장에서 WHERE 절에서 LAST_NAME 컬럼을 LOWER() 함수로 처리하지 않으면, [대/소]문자가 다르기 때문에 아래와 같이 결과 메시지가 표시됩니다.

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

선택된 행 없음
```

☞ 사용자가 영문 데이터를 입력할 때, 필요한 경우 (1) 대/소문자 규칙을 따르도록 지침을 내리거나 또는 (2) 사용자가 입력한 영문 값을 프로그램의 INSERT-문에서 UPPER() 또는 LOWER() 함수로 처리하면, SELECT-문의 WHERE-절에 조건을 기술 할 때, 컬럼에 대한 [대/소]문자 변환 함수의 사용을 방지할 수 있습니다.

## 4-5. 문자 함수-2: 문자열 수정 함수.

■ CONCAT('문자열', '문자열') 함수: 두 문자열을 붙입니다 [참고: || 연산자]

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

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

■ SUBSTR('문자열', 시작 위치, [추출할 문자 수]) 함수: 데이터로부터 원하는 문자(열)를 추출해 줍니다.

```
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
```

■ LENGTH ( '문자열' ) 함수 : 문자열의 길이를 **글자 수**로 반환합니다.

LENGTHB ( '문자열' ) 함수: 문자열의 길이를 **바이트 수**로 반환합니다.

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

LENGTH('신상현') LENGTHB('신상현')
---------------- -----------------
               3                 9
```

■ REPLACE ('문자열', '찾을 문자', '대체 문자') 함수: 문자열에서 문자(열)을 찾아서 문자(열)로 교체

```
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
```

■ TRIM('문자' FROM '문자열') 함수: 문자열의 양 끝의 연속된 문자를 삭제합니다. 단, 중간에 있는 문자는 삭제되지 못합니다.

```
SQL> SELECT  TRIM(BOTH 'S' FROM 'SSMISTHSS') AS "RESULT1"      — 양 끝의 연속된 'S' 를 삭제.
           , TRIM('S' FROM 'SSMISTHSS') AS "RESULT2"           — 양 끝의 연속된 'S' 를 삭제.
           , TRIM(LEADING 'S' FROM 'SSMISTHSS') AS "RESULT3"   — 앞에 있는 연속된 'S' 를 삭제.
           , TRIM(TRAILING 'S' FROM 'SSMISTHSS') AS "RESULT4" -- 뒤에 있는 연속된 'S' 를 삭제.
     FROM DUAL;

RESULT1 RESULT2 RESULT3 RESULT4
------- ------- ------- -------
MISTH   MISTH   MISTHSS SSMISTH
```

[참고] TRIM 함수 대신에 LTRIM(), RTRIM() 함수를 이용하여 동일한 결과를 얻을 수도 있습니다.

```
SQL> SELECT  LTRIM('SSMISTHSS', 'S') AS "RESULT1"              — 왼쪽에 있는 연속된 'S' 를 삭제.
           , RTRIM('SSMISTHSS', 'S') AS "RESULT2"              — 오른쪽에 있는 연속된 'S' 를 삭제.
           , LTRIM(RTRIM('SSMISTHSS','S'),'S') AS "RESULT3"    — 양 끝의 연속된 'S' 를 삭제.
     FROM DUAL;

RESULT1 RESULT2 RESUL
------- ------- -----
MISTHSS SSMISTH MISTH
```

☞ 단, 양쪽 모두에서 명시된 문자를 삭제하려면, LTRIM(), RTRIM() 함수를 중첩해서 사용해야 하므로, TRIM() 함수를 사용하는 것을 권장합니다.

■ LPAD('문자열', 출력 시 사용되는 Byte길이,'남은 빈칸을 채울 문자') 함수 및
RPAD('문자열', 출력 시 사용되는 Byte길이,'남은 빈칸을 채울 문자') 함수

○ 문자열을 출력 시 사용되는 Byte 길이의 공간에 문자열을 표시하고, 사용되지 않고 남은 길이의 Byte를 지정된 문자로 채웁니다. 이 때, LPAD()는 왼쪽을, RPAD()는 오른쪽을 지정된 문자로 채웁니다.

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

RESULT
----------
*****SMITH  -- 'SMITH'를 10-BYTE 공간에 표시하고, 남은 공간을 왼쪽부터 '*'로 채움.
```

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

RESULT
----------
SMITH*****   -- 'SMITH'를 10-BYTE 공간에 표시하고, 남은 공간을 오른쪽부터 '*'로 채움.
```

■ INSTR('문자열','찾을 문자(열)'[, 검색을 시작할 자리 수, 존재 횟수]) 함수

○ 찾을 문자나 문자열이 있으면 그 문자나 문자열이 시작된 자리 수를 반환하며, 찾을 문자나 문자열이 없으면 0 을 반환합니다. 주로, 입력되는 데이터에 명시해야 하는 문자가 빠졌거나, 명시하면 안 되는 문자가 존재하는 지를 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
```

■ 문자열 수정 함수가 사용된 SELECT-문 예제.

```
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
```

## 4-6. 숫자 함수.

■ MOD(숫자1, 숫자2) 함수: 숫자1을 숫자2로 나누고 남은 나머지를 반환합니다.

| | |
|---|---|
| SQL> **SELECT MOD(1600, 300)**<br>**FROM DUAL ;**<br><br>MOD(1600,300)<br>----------<br>100 | 1600을 300으로 나눈 후, 나머지를 반환. (100) |

■ ROUND (숫자, [소수점 이하 유효 자리 수]) 함수: 소수점 이하 유효 자리 수의 다음 자리에서 반올림합니다.

| | |
|---|---|
| SQL> **SELECT**<br>**ROUND(1745.9260, 4) RESULT1, --(1)**<br>**ROUND(1745.9460, 1) RESULT2, --(2)**<br>**ROUND(1745.9260, 0) RESULT3, --(3)**<br>**ROUND(1745.9260) RESULT4 --(4)**<br>**FROM DUAL ;**<br><br>RESULT1 RESULT2 RESULT3 RESULT4<br>---------- ---------- ---------- ----------<br>1745.926 1745.9 1746 1746 | • (1) 유효 소수점 자리 수를 초과하여 처리 데이터가 그대로 표시됩니다. (1745.926)<br><br>• (2) 소수점 이하 1자리까지 유효하며, 2번째 자리에서 반올림됩니다. (1745.9)<br><br>• (3)와 (4)는 동일한 의미이며, 소수점 이하 0자리까지 유효하며, 1번째 자리에서 반올림됩니다. (1746) |
| SQL> **SELECT**<br>**ROUND(1745.9260, -1) RESULT1, --(1)**<br>**ROUND(1765.9260, -2) RESULT2, --(2)**<br>**ROUND(1445.9260, -3) RESULT3, --(3)**<br>**ROUND(1745.9260, -4) RESULT4 --(4)**<br>**FROM DUAL ;**<br><br>RESULT1 RESULT2 RESULT3 RESULT4<br>---------- ---------- ---------- ----------<br>1750 1800 1000 0 | 반올림 자리 수가 음수인 경우에는, 정수부 쪽으로 반올림 자리 수가 고려됩니다.<br><br>• (1) 정수부 일 자리에서 반올림됩니다. (1750)<br><br>• (2) 정수부 십 자리에서 반올림됩니다. (1800)<br><br>• (3) 정수부 백 자리에서 반올림됩니다. (1000)<br><br>• (4) 정수부 반올림 단위를 초과하여 0 이 반환됩니다. |

■ TRUNC (숫자, [소수점 이하 유효 자리 수]) 함수: 소수점 이하 유효 자리 수 이하를 절삭(버림)합니다.

```
SQL> SELECT
     TRUNC(1745.9260, 4) RESULT1,  --(1)
     TRUNC(1745.9260, 2) RESULT2,  --(2)
     TRUNC(1745.9260, 0) RESULT3,  --(3)
     TRUNC(1745.9260)    RESULT4   --(4)
     FROM  DUAL ;

   RESULT1    RESULT2    RESULT3    RESULT4
---------- ---------- ---------- ----------
  1745.926    1745.92       1745       1745
```

- (1) 유효 소수점 자리 수를 초과하여 처리 데이터가 그대로 표시됩니다. (1745.926)
- (2) 소수점 이하 2자리까지 유효하며, 3번째 자리부터 절삭(버림)됩니다. (1745.92)
- (3)과 (4)는 동일한 의미이며, 소수점 이하 모두가 절삭(버림)됩니다. (1745)

```
SQL> SELECT
     TRUNC(1745.9260, -1) RESULT1,  --(1)
     TRUNC(1765.9260, -2) RESULT2,  --(2)
     TRUNC(1445.9260, -3) RESULT3,  --(3)
     TRUNC(1745.9260, -4) RESULT4   --(4)
     FROM  DUAL ;

   RESULT1    RESULT2    RESULT3    RESULT4
---------- ---------- ---------- ----------
      1740       1700       1000          0
```

절삭 자리 수가 음수인 경우에는, 정수부 쪽으로 절삭 자리 수가 고려됩니다.

- (1) 정수부 일 자리에서 절삭(버림)됩니다. (1740)
- (2) 정수부 십 자리에서 절삭(버림)됩니다. (1700)
- (3) 정수부 백 자리에서 절삭(버림)됩니다. (1000)
- (4) 정수부 버림 단위를 초과하여 0 이 반환됩니다.

## ■ 숫자 함수가 사용된 SELECT-문 예제

```
SQL> SELECT last_name, salary, TRUNC(salary/5000,0), MOD(salary, 5000)
     FROM  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
Tuvault                      7000                    1             2000
King                        10000                    2                0
Sully                        9500                    1             4500
...

30개의 행이 선택됨
```

## 4-7. 날짜 함수-1: SYSDATE 함수.

• SYSDATE 함수는 오라클-서버가 운영되는 운영체제의 날짜와 시간을 DATE 데이터유형으로 반환합니다.

```
SQL> SELECT SYSDATE FROM DUAL ;

SYSDATE
--------
11/05/10
```

## 4-8. WHERE 절에 날짜 형식 상수를 사용 시에 주의사항.

○ 날짜 형식의 상수를 SQL-문에 명시할 경우에 세션에 표시되는 형식과 동일하지 않을 경우에 에러 또는 잘못된 처리가 발생됩니다.

☞ 한글 Windows 에 설치된 SQL*Developer는 디폴트로 언어는 'KOREAN'으로, 날짜 표시 형식은 'RR/MM/DD'를 사용하도록 설정되어 있습니다.

1> 세션에 설정된 날짜 표시 형식과 동일한 형식으로 WHERE절에 날짜 상수를 명시하여 데이터 조회를 수행합니다.

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

LAST_NAME                 HIRE_DATE
------------------------- ---------
Markle                    08/03/08
Philtanker                08/02/06
Lee                       08/02/23
Ande                      08/03/24
Banda                     08/04/21
Kumar                     08/04/21
Geoni                     08/02/03

 7개의 행이 선택됨
```

☞ 에러가 없이 정상적으로 처리 결과가 표시됩니다.

2> 세션에 설정된 날짜 표시 형식과 다르게 WHERE절에 날짜 상수를 명시한 경우에는 다음처럼 에러가 발생됩니다.

```
SQL> SELECT last_name, hire_date
     FROM hr.employees
     WHERE hire_date > '01-FEB-08';
ORA-01858: 숫자가 있어야 하는 위치에서 숫자가 아닌 문자가 발견되었습니다.
01858. 00000 -  "a non-numeric character was found where a numeric was expected"
*Cause:    The input data to be converted using a date format model was
           incorrect.  The input data did not contain a number where a number was
           required by the format model.
*Action:   Fix the input data or the date format model to make sure the
           elements match in number and type.  Then retry the operation.
```

## 4-9. WHERE 절에서 TO_DATE() 함수를 이용한 날짜 상수 처리.

○ 날짜 상수는 세션의 날짜 표시 형식에 민감하므로, SQL-문에서 날짜 상수를 직접 기술 시에는 반드시 TO_DATE() 함수로 날짜 상수를 처리해야 SQL-문이 정상적으로 처리됩니다.

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

LAST_NAME                 HIRE_DATE
------------------------- ---------
Mavris                    02/06/07
Baer                      02/06/07
Higgins                   02/06/07
Gietz                     02/06/07
De Haan                   01/01/13
```

• SQL-문에서 사용되는 날짜 형식의 상수는 반드시 TO_DATE() 함수로 처리합니다!

## 4-10. DATE 형식의 데이터에 산술 연산하기.

| | | |
|---|---|---|
| 날짜 + 숫자 | TO_DATE('2009/10/03', 'YYYY/MM/DD') + 3 | 2009/10/06 |
| | TO_DATE('2009/10/03', 'YYYY/MM/DD') + 48/24 | 2009/10/05 |
| 날짜 - 숫자 | TO_DATE('2009/10/03', 'YYYY/MM/DD') - 3 | 2009/09/30 |
| 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 |

■ 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 SELECT-문의 실행 후에 표시되는 결과는 본 문서와 다를 수 있습니다.

[주의] 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
```

## 4-11. 날짜 함수-2: DATE 데이터유형을 기반한 여러 가지 날짜 함수.

■ **MONTHS_BETWEEN ( '날짜1' , '날짜2' ) 함수**: 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
```

☞ 위의 예제처럼, 날짜 상수에 언어로 달(Month) 이름을 명시한 경우, TO_DATE() 함수에 NLS_DATE_LANGUAGE 옵션을 이용하여, 달(Month) 이름에서 사용한 언어를 명시해야 합니다. 달(Month)를 숫자로 명시하면, NLS_DATE_LANGUAGE 옵션을 사용할 필요가 없습니다.

■ **NEXT_DAY ( '날짜' , '요일' ) 함수**: 날짜를 기준으로 다음에 오는 요일의 날짜를 반환합니다.

```
SQL> SELECT NEXT_DAY (TO_DATE('08/11/11','RR/MM/DD'),'금요일')
     FROM dual ;

NEXT_DAY(TO_DATE('08/11/11','RR/MM/DD'),'금요일')
-------------------------------------------------
08/11/14
```

```
SQL> SELECT NEXT_DAY (TO_DATE('08/11/11','RR/MM/DD'), 6)
     FROM dual ;

NEXT_DAY(TO_DATE('08/11/11','RR/MM/DD'),6)
------------------------------------------
08/11/14
```

• 2008년 11월 달력

| 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 | | | | | | |

• [참고] 오라클 데이터베이스에서 요일을 숫자로 표시할 수 있습니다.
일요일: 1번, 월요일: 2번, 화요일: 3번, 수요일: 4번, 목요일: 5번, 금요일: 6번, 토요일: 7번 입니다.

■ **ADD_MONTHS( '날짜' , 숫자 ) 함수**: 숫자를 날짜의 달 수에 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
```

■ LAST_DAY ( '날짜' ) 함수: 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 29일 입니다.

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

☞ 1995년 2월은 28일까지만 있고, 없는 날짜인 29일을 함수에 사용하였기 때문에 위의 오류가 발생되었습니다.

## 4-12. 날짜 함수-3: 날짜 데이터에 대한 ROUND() 함수(반올림) 및 TRUNC()함수(버림).

• 세션의 날짜 표시 형식을 아래와 같이 설정한 후, ROUND(), TRUNC() 함수 문장을 실습합니다.

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT='YYYY/MM/DD' ;

session SET이(가) 변경되었습니다.
```

☞ 데이터베이스에 접속한 상태에서 ALTER SESSION 문장으로 설정된 사항은 접속을 해제하면 설정된 사항이 해제됩니다.

| ■ ROUND( '날짜' , '형식 요소' ) | ■ TRUNC( '날짜' , '형식 요소' ) |
|---|---|
| SQL> **SELECT sysdate, ROUND(sysdate, 'YEAR')**<br>**FROM dual ;**<br><br>SYSDATE ROUND(SYSDATE,'YEAR')<br>---------- ----------<br>2007/11/07 2008/01/01<br><br>SYSDATE ROUND(SYSDATE,'YEAR')<br>---------- ----------<br>2007/05/07 2007/01/01 | SQL> **SELECT sysdate, TRUNC(sysdate, 'YEAR')**<br>**FROM dual ;**<br><br>SYSDATE TRUNC(SYSDATE,'YEAR')<br>---------- ----------<br>2007/11/07 2007/01/01 |
| SQL> **SELECT sysdate, ROUND(sysdate, 'MONTH')**<br>**FROM dual ;**<br><br>SYSDATE ROUND(SYSDATE,'MONTH')<br>---------- ----------<br>2007/11/16 2007/12/01<br><br>SYSDATE ROUND(SYSDATE,'MONTH')<br>---------- ----------<br>2007/11/07 2007/11/01 | SQL> **SELECT sysdate, TRUNC(sysdate, 'MONTH')**<br>**FROM dual ;**<br><br>SYSDATE TRUNC(SYSDATE,'MONTH')<br>---------- ----------<br>2007/11/07 2007/11/01 |

☞ ROUND() 및 TRUNC() 함수에 대하여 DD, HH, MI 형식요소도 사용하여 반올림 및 버림을 수행할 수 있습니다.

## 4-13. 자동 데이터 유형 변환 (IMPLICIT DATA TYPE CONVERSION).

| FROM | TO | 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. |

☞ SQL문에서 위의 설명처럼 하나의 데이터유형이 다른 데이터유형으로 처리 중에 자동으로 데이터유형이 변환되어 처리될 수 있습니다. 그렇지만, 위의 설명처럼 해당되는 경우를 벗어난 경우에는 잘못된 처리나 오류가 발생될 수 있기 때문에, 데이터 유형 자동 변환보다는 데이터유형 변환함수(TO_CHAR(), TO_DATE(), TO_NUMBER())를 사용하여 처리하는 것을 권장합니다.

■ 다음은 자동 데이터 유형 변환을 이해하기 위한 예제 구문입니다.

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

   RESULT1 RESULT2
---------- -------
         2 11
```

☞ RESULT1은 산술연산자(+) 때문에 문자 '1' 이 자동으로 숫자 데이터 유형으로 변환되어 연산자가 처리되었습니다.

☞ RESULT2는 || 연산자에 의하여 숫자 1 이 문자 '1'로 자동으로 데이터 유형 변환이 수행 되었습니다.

## 4-14. 명시적인 데이터 유형 변환 (EXPLICIT DATA TYPE CONVERSION).

• 명시적인 데이터 유형 변환이란, TO_CHAR(), TO_DATE(), TO_NUMBER() 함수에서 다음 섹션에서 설명하는 형식모델을 이용하여 상수 또는 컬럼의 값을 직접 원하는 데이터 유형으로 변환하는 것을 의미합니다.

## 4-15. 날짜에 대한 대표적인 형식 모델(FORMAT MODEL).

| 형식 | 의미 | 형식 | 의미 |
|---|---|---|---|
| YYYY | 4자리 숫자 연도 (예, 2010) | DD | 숫자로 된 달의 일. |
| YY | 2자리 숫자 연도 (예, 10) | DAY | 요일 (영어: FRIDAY, 한국어: 금요일) |
| RR | 2자리 숫자 연도 (예, 10) | DY | 3자리 약어로 표시한 요일<br>(영어: FRI, 한국어: 금) |
| 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)<br>(영어: APRIL , 한국어: 4월) | MI | 숫자로 된 분 |
| | | SS | 숫자로 된 초 |
| MON | 3자리 약어로 된 달 이름<br>(영어: APR, 한국어: 4월) | AM | 오전/오후<br>(영어: AM/PM, 한국어: 오전/오후) |

## [참고] RR-FORMAT과 YY-FORMAT 차이.

• 세기 없이 년도만 표시했을 때, 만약 표시 형식이 YY로 지정된 경우에는, 세기는 항상 현재 세기와 동일합니다.

• 세기 없이 년도만 표시했을 때, 만약 표시 형식이 RR로 지정된 경우에는, 세기는 아래의 기준을 따릅니다.

| | | 알기를 원하는 2자리 연도 | |
|---|---|---|---|
| | | 0 ~ 49 | 50 ~ 99 |
| 현재년도 | 0 ~ 49 | 현재세기 | 현재세기 - 1 |
| | 50 ~ 99 | 현재세기 + 1 | 현재세기 |

☞ 단, 위의 표에 명시된 RR 기준은 현재 년도가 1999년 이 후 이므로, 1950 년부터 1999년 사이의 데이터를 처리할 때만 유용합니다.

☞ 현재의 년도가 2012년 일 경우에 YY 형식으로 표시되는 94 두 자리 년도의 경우에는, 이를 세기와 같이 표시했을 때 2094년이 됩니다.

■ 다음은 RR 및 YY의 차이를 이해하기 위한 위한 예제 구문입니다.

```
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
```

☞ 이러한 RR과 YY의 차이는 세기 및 년도를 YYYY 형식으로 모두 표시하면 쉽게 극복됩니다.

## 4-16. 숫자에 대한 대표적인 형식 모델(FORMAT MODEL).

| 형식 | 의미 | 사용 예 | 설명 |
|---|---|---|---|
| 9 ( 또는 0) | 자리 수 | 999,999 | 숫자를 최대 만 단위까지 표시할 수 있습니다. |
| 0 | 자리 수 | 099,999 | 사용되지 않은 자리를 0으로 채워 표시합니다. |
| $ | 달러 | $99,999 | 숫자 앞에 $를 표시합니다. |
| L | 그 지역 통화 단위 | L99,999 | 해당 지역의 통화 단위를 앞에 표시합니다. |
| . | 소숫점 | 999.99 | 소숫점을 표시합니다. |
| , | 천 단위 구분자 | 99,999 | 숫자에 천 단위 구분자를 표시합니다. |

## 4-17. 데이터 유형-변환 함수(CONVERSION FUNCTION).

■ **TO_DATE( '날짜문자열' , '날짜 형식 모델' [ , 'NLS_PARAM'] ) 함수** : 날짜처럼 표시된 문자 값에 해당하는 형식모델을 명시하여, 문자 값을 날짜 데이터 유형으로 데이터유형을 변환합니다.

○ '01-SEP-95' 날짜와 '1994/01/11' 날짜 사이의 달수를 MONTHS_BETWEEN() 함수를 이용하여 구하시오.

```
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
```

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

■ **TO_CHAR( '날짜 또는 숫자 표현식' , '날짜 또는 숫자 형식 모델' [ , 'NLS_PARAM'] ) 함수** : 날짜 또는 숫자 표현식을, 명시한 날짜 또는 숫자의 형식 모델의 표시형식대로 문자 데이터유형으로 표시합니다.

☞ TO_CHAR() 함수는 날짜 또는 숫자 유형의 데이터에 대하여 사용자가 원하는 출력형식을 지정하여, 지정된 형식으로 표시(DISPLAY)하거나, 또는 지정된 형식을 이용하여 다른 처리를 수행하고 싶을 때 사용합니다.

☞ 이 때 처리 결과는 문자 데이터유형으로 데이터유형이 변경됩니다.

■ TO_CHAR() 함수의 기능을 이해하기 위한 실습.

1> SYSDATE 함수의 처리 결과를 세션의 표시형식이 아닌, 사용자가 원하는 표시형식으로 표시되도록 지정합니다.

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

SYSDATE  TO_CHAR(SYSDATE,'YYYY/MON/DDHH24:MI:SSAMDAY')
-------- ----------------------------------------------
11/07/25 2011/7월 /25 18:02:19 오후 월요일
```

☞ SYSDATE 처리 결과가 접속 세션의 'RR/MM/DD' 표시형식이 아닌, 사용자가 지정한 'YYYY/MON/DD HH24:MI:SS AM DAY' 형식으로 표시됩니다.

☞ TO_CHAR() 함수에서는 사용자가 원하는 형식 모델을 모두 사용할 수 있습니다. 즉, [AM]과 [HH24] 형식 모델을 동시에 사용할 수 있습니다.

2> 날짜 요소 구분자를 [ . / - 빈칸 , ] 문자가 아닌 [년, 월, 일, 시, 분, 초] 같은 말로 표시할 수 있습니다.

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

TO_CHAR(SYSDATE,'YYYY"년"MM"월"DD"일"HH24"시"MI"분"SS"초"AMDAY')
-----------------------------------------------------------------------------------
2013 년  04월 26일 13시  44분 26 초  오후 금요일
```

☞ 날짜 형식 모델을 지정할 때 위의 예제처럼 큰 따옴표(")로 감싸서 "년", "월", "일", "시", "분", "초"로 구분된 결과를 표시할 수 있습니다.

3> 영어 문자로 표시하는 형식 모델(MONTH, MON, DAY, DY, AM, 등)를 사용할 때, [대/소]문자를 섞어서 명시하면, 아래의 규칙대로 [대/소]문자가 구분되어 표시됩니다.

```
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
```

☞ 위에서 설명한 영어로 표시되는 형식모델에 대한 [대/소]문자 표시규칙을 MONTH 형식모델로 설명합니다.

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

4> 디폴트로 숫자로 표시되는 년도, 달, 날, 시간, 분, 초 에서 두 자리 중에 한 자리만 사용한 경우,
남은 빈 자리를 0[Zero]로 채웁니다(예, 2010/03/02 04:06:04 PM WEDNESDAY).
형식 모델 요소의 제일 앞에 fm을 한 번 명시하면, 두 자리 중 사용하지 않은 자리를 0[Zero]으로 채우지 않습니다.
(예, 2010/3/2 4:6:4 PM WEDNESDAY).

```
SQL> SELECT TO_CHAR(sysdate, 'fmYYYY-MM-DD fmHH24:fmMI:fmSS AM Day')
     FROM dual ;

TO_CHAR(SYSDATE,'FMYYYY-MM-DDFMHH24:FMMI:FMSSAMDAY')
-----------------------------------------------------
2007-5-7 02:0:04 오후 수요일
```

☞ 위의 예제에서 YYYY 앞의 첫 번째 fm에 의하여 YYYY-MM-DD까지는 사용하지 않은 자리가 0으로 채워지지 않습니다.

☞ HH24 앞의 두 번째 fm에 의하여 HH24는 0으로 다시 채워집니다. 즉, 첫 번째 fm에 의한 기능이 해제됩니다.

☞ MI 앞의 세 번째 fm에 의하여 MI부터는 사용하지 않은 자리가 0으로 채워지지 않습니다.

☞ SS 앞의 네 번째 fm에 의하여 SS부터는 0으로 다시 채워집니다. 즉, 두 번째 fm에 의한 기능이 해제됩니다.

5> SP 옵션(숫자를 영문으로 표시), TH 옵션(숫자를 영어의 서수(th) 형태로 표시)를 사용하여 숫자를 문자로 표시할 수 있습니다.

```
SQL> SELECT TO_CHAR(sysdate, 'yYyySP-Mmspth-DDspth HH24:MI:SS AM Day')
     FROM dual ;

TO_CHAR(SYSDATE,'YYYYSPTH-MMSP-DDSPTHHH24:MI:SSAMDAY')
-------------------------------------------------------------------------------------------
two thousand ten-Third-TENTH 20:21:40 오후 수요일
```

☞ yYyy의 첫 글자가 소문자이므로, 연도가 모두 소문자로 표시되었습니다.

☞ MM의 첫 글자가 대문자이고 다음 글자가 소문자이므로, 첫 글자 대문자 나머지는 소문자로 결과가 표시되었습니다.

☞ DD의 처음 2글자가 대문자이므로 모두 대문자로 결과가 표시되었습니다.

☞ th 옵션에 의하여 달과 일이 서수형식으로 각각 Third 및 TENTH로 표시되었습니다.

6> 다음은 TO_CHAR() 함수를 이용하여 HR.EMPLOYEES 테이블의 SALARY 컬럼의 값에 달러($) 또는 원(₩) 통화단위를 표시되고, 숫자 값 사이에 천 단위 구분자로 콤마(,)가 표시되도록 처리한 SELECT문 예제입니다.

```
SQL> SELECT salary, TO_CHAR(salary, '$9,999,999.99'), TO_CHAR(salary, 'L9,999,999.99')
     FROM hr.employees
     WHERE employee_id=100 ;

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

☞ 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.

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

```
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')
------------------------------- -------------------------------
24,000.00-                      <24,000.00>
```

☞ 단, MI와 PR 형식모델은 동시에 사용될 수 없습니다.

8> 9 및 0 형식 모델 사용 시 표시결과의 차이.

```
SQL> SELECT TO_CHAR(salary, 'L999,999,999.99'), TO_CHAR(salary, 'L099,999,999.99')
     FROM hr.employees
     WHERE employee_id = 100 ;

TO_CHAR(SALARY,'L999,999,999.99') TO_CHAR(SALARY,'L099,999,999.99')
--------------------------------- ---------------------------------
                       ₩24,000.00                   ₩000,024,000.00
```

☞ 표시 결과에서처럼 자리 수를 표시할 때, 첫 글자를 9 대신 0으로 표시하면, 사용되지 않은 자리 들을 앞에서부터 0으로 채웁니다.

[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 형식 자리수의 수를 늘려주면, 값이 정상적으로 표시됩니다.

■ 현재 날짜에서 년도, 월, 일, 요일을 추출하여 표시하시오.

```
SQL> SELECT sysdate,   -- 세션의 날짜 표시형식대로 서버의 날짜와 시간이 표시됩니다.
            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
-------- ---- -- -- ------------ --------
13/04/26 2013 04 26 금요일       23:09:54
```

☞ SYSDATE 함수가 반환하는 데이터베이스 서버의 현재 날짜 및 시간 값으로부터 각각 TO_CHAR() 함수를 이용하여 년도, 월, 일, 요일, 시간이 추출되어 문자 값으로 표시됩니다.

■ **TO_NUMBER( '숫자문자열' , '숫자 형식 모델' ) 함수** : 명시된 숫자문자열에 해당하는 형식모델을 명시하여, 문자 값을 숫자 데이터유형으로 변환합니다.

○ '$12,000' 문자열 값에 0.1을 곱한 값을 표시하시오.

```
SQL> SELECT TO_NUMBER('$12,000','$999,999')*0.1
     FROM dual ;

    TO_NUMBER('$12,000','$999,999')*0.1
---------------------------------------
                                  1200
```

☞ 9의 개수는 문자열의 숫자 자리 수에 맞추지 않아도 되며, 원하는 만큼 충분한 개수를 표시해줍니다.

## 4-18. 단일 행 함수의 중첩(NESTING FUNCTION).

○ 단일 행 함수는 제한 없이 여러 번 중첩할 수 있습니다.

■ HR.EMPLOYEES 테이블의 데이터를 이용하여 DEPARTMENT_ID가 10인 직원의 LAST_NAME과 대문자로 LAST_NAME의 처음 8 글자와 그 다음에 _US가 표시되는 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                   WHALEN_US
```

☞ LAST_NAME 컬럼의 값이 8 글자 이하인 경우에는 LAST_NAME 전체가 표시됩니다.

## 4-19. 일반 함수(GENERAL FUNCTION).

• 데이터 유형(DATA-TYPE)에 제한 받지 않으며, NVL(), DECODE() 함수와 CASE 표현식(CASE EXPRESSION)은 모든 버전의 오라클 데이터베이스 서버 제품에서 사용할 수 있습니다.

• NVL2(), NULLIF(), COALESCE() 함수는 오라클 데이터베이스 9i 버전부터 사용할 수 있습니다.

### 4-19-1. NVL(컬럼명, 대체값) 함수.

• 처리되는 컬럼에 입력된 데이터가 존재하면 그 데이터가 사용되고, 처리되는 컬럼이 NULL 상태에 있으면, 뒤에 명시된 대체값으로 변경하여 처리됩니다.

1> NVL() 함수로 COMMISSION_PCT 컬럼을 처리하지 않은 경우.

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

LAST_NAME                    SALARY COMMISSION_PCT    ANN_SAL
------------------------ ---------- -------------- ----------
OConnell                       2600
Grant                          2600
...
Russell                       14000             .4     235200
Partners                      13500             .3     210600
...

 107개의 행이 선택됨
```

← COMMISSION_PCT, ANN_SAL에 아무것도 표시되지 않습니다.

2> NVL() 함수로 COMMISSION_PCT 컬럼을 처리해 준 경우.

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

LAST_NAME                    SALARY COMMISSION_PCT    ANN_SAL
------------------------ ---------- -------------- ----------
OConnell                       2600                    31200
Grant                          2600                    31200
...
Russell                       14000             .4     235200
Partners                      13500             .3     210600
...

 107개의 행이 선택됨
```

## 4-19-2. DECODE(컬럼, 값1, 표현식1, 값2, 표현식2, . . . , 값n, 표현식n, [표현식x]) 함수.

- 컬럼의 값이 값1 이면, 표현식1을 표시하고, 값2 이면, 표현식2를 표시하고,. . ., 값n이면, 표현식n을 표시하고, 컬럼의 값이 명시된 값들에 해당되지 않으면, 표현식x를 표시합니다.

- 아래의 문장에 사용된 DECODE 함수는 JOB_ID 컬럼의 값이 'IT_PROG' 이면, 1.10*salary 값을 표시하고, 'ST_CLEARK' 이면, 1.15*salary 값을 표시하고, 'SA_REP' 이면, 1.20*salary 값을 표시하고, 이 외의 다른 값일 경우에는 salary를 표시합니다.

```
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
Grant                     SH_CLERK         2600           2600
...
Hunold                    IT_PROG          9000           9900
Ernst                     IT_PROG          6000           6600
...
Nayer                     ST_CLERK         3200           3680
Mikkilineni               ST_CLERK         2700           3105
...
Tucker                    SA_REP          10000          12000
Bernstein                 SA_REP           9500          11400
...

107개의 행이 선택됨
```

[참고] 위의 예제에서 DECODE 함수 안의 내용에서 맨 마지막 salary를 명시하지 않으면, JOB_ID 컬럼의 값이 'IT_PROG', 'ST_CLERK', 'SA_REP' 가 아닐 경우에는 NULL 로 표시됩니다.

## 4-19-3. CASE 표현식

• CASE 표현식은 대부분의 데이터베이스 제품들에서 지원됩니다.

• 단순 CASE-표현식은 오라클 데이터베이스의 DECODE-함수와 동일한 기능을 제공합니다.

• 확장 CASE-표현식은 오라클 데이터베이스의 DECODE-함수가 할 수 없는 처리 방법을 제공합니다.

1> Basic CASE Expression

```
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
------------------------- ---------- ---------- --------------
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
Mavris                    HR_REP           6500           6500
...
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
```

[참고] 위의 예제에서 CASE 표현식의 내용에서 맨 마지막 ELSE salary를 명시하지 않으면, JOB_ID 컬럼의 값이 'IT_PROG', 'ST_CLERK', 'SA_REP' 가 아닐 경우에는 NULL 로 표시됩니다.

2> Extended Case Expression

```
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
...

 107개의 행이 선택됨
```

## 4-19-4. NVL2 (컬럼 이름, 대체값1, 대체값2) 함수(9iNF).

- 컬럼의 데이터가 존재하면 대체값1을 표시하고 컬럼이 NULL상태이면 대체값2를 표시합니다.

- 대체값1과 대체값2가 같은 데이터유형의 값이어야 합니다.

```
SQL> SELECT last_name, commission_pct, NVL2(commission_pct,'OK','NO') AS "GET_O_X"
     FROM hr.employees
     WHERE department_id IN (20,80);

LAST_NAME                COMMISSION_PCT GET_O_X
------------------------ -------------- -------
Hartstein                               NO
Fay                                     NO
Russell                            .4 OK
Partners                           .3 OK
Errazuriz                          .3 OK
...

 37개의 행이 선택됨
```

## 4-19-5. NULLIF( 컬럼1 , 컬럼2 ) 함수(9iNF).

• 컬럼1과 컬럼2의 값이 다르면, 무조건 컬럼1을 표시하고, 두 값이 같으면, NULL을 반환합니다.

• 컬럼1과 컬럼2의 데이터 유형이 동일해야 합니다.

```
SQL> SELECT first_name, LENGTH(first_name) expr1, last_name,  LENGTH(last_name)  expr2,
           NULLIF(LENGTH(first_name), LENGTH(last_name)) result
     FROM hr.employees
     WHERE department_id IN (10,20);

FIRST_NAME               EXPR1 LAST_NAME                    EXPR2     RESULT
-------------------- ---------- ------------------------- ---------- ----------
Jennifer                      8 Whalen                             6          8
Michael                       7 Hartstein                          9          7
Pat                           3 Fay                                3
```

## 4-19-6. COALESCE (컬럼expr1 , 컬럼expr2 , … , 컬럼exprn, [상수]) 함수(9iNF).

• 함수 내에 명시된 컬럼 값을 확인해서 최초로 NULL이 아닌 컬럼expr의 값을 표시합니다.

• 함수 안에 명시된 컬럼-표현식들의 데이터 유형이 동일해야 하며, 데이터 유형이 다르면 아래처럼 에러가 발생됩니다.

```
SQL> SELECT last_name, COALESCE(last_name, commission_pct, -1) comm FROM hr.employees ;
ORA-00932: 일관성 없는 데이터 유형: CHAR이(가) 필요하지만 NUMBER임
00932. 00000 -  "inconsistent datatypes: expected %s got %s"
*Cause:
*Action:
1행, 39열에서 오류 발생
```

• 아래처럼 데이터가 구성된 사원 테이블이 있다고 가정합니다.

| 사번 | 이름 | 관리자 | 커미션 | 각 행에 대한 설명 |
|---|---|---|---|---|
| 100 | 홍길동 | 10 | 0.1 | ← 관리자 및 커미션 컬럼이 모두 NULL이 아님 |
| 101 | 홍길순 | | 0.1 | ← 관리자 컬럼은 NULL, 커미션 컬럼이 NULL이 아님 |
| 102 | 홍길용 | | | ← 관리자, 커미션 컬럼 모두 NULL. |

위의 사원 테이블에 다음의 SQL-문을 수행했을 때 결과는 다음과 같습니다.

```
SELECT 이름, COALESCE(관리자, 커미션, -1) Result1
FROM   사원 ;

이름   RESULT1
------ -------
홍길동      10     ← 관리자 컬럼이 NULL이 아님
홍길순     0.1     ← 관리자 컬럼은 NULL, 커미션 컬럼이 NULL이 아님
홍길용      -1     ← 관리자, 커미션 컬럼 모두 NULL, 상수 -1 을 표시
```

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

## ◈ 학습 목표.

■ SQL-문에서 사용할 수 있는 다양한 오라클 다중행 함수(집합함수)에 대하여 학습합니다.

■ 집합 함수와 같이 사용되는 [GROUP BY]절 및 [HAVING]절에 대하여 학습합니다.

## 5-1. 집합함수(GROUP FUNCTION) 개요.

■ 집합함수의 종류

| 집합 함수 이름 | 기능 |
|---|---|
| SUM() | 합계 |
| AVG() | 평균 |
| MAX() | 최대값 |
| MIN() | 최소값 |
| COUNT() | 개수 |

■ 각 집합함수가 처리할 수 있는 데이터 유형

| | |
|---|---|
| SUM(), AVG() | 숫자 데이터 유형에만<br>사용할 수 있습니다. |
| MAX(), MIN(), COUNT() | 문자, 숫자, 날짜 데이터유형 모두에서<br>사용할 수 있습니다. |

☞ MAX(), MIN() 집합함수는 LONG, CLOB 데이터유형에는 **사용할 수 없습니다**.

[참고] LONG, CLOB 데이터 유형에 대한 간단 설명

| | |
|---|---|
| LONG | 최대 2GB 길이까지 가능한 **문자** 데이터 유형 |
| CLOB | 최대 4GB 길이까지 가능한 **문자** 데이터 유형 |

## 5-2. 집합함수 사용 시 고려할 사항.

• 집합함수에 옵션으로 DISTINCT 또는 ALL 키워드를 사용할 수 있습니다. 또한 집합함수 실행 시 데이터가 없으면(즉, NULL 상태의 필드), 그 필드는 없는 것으로 간주합니다.

◦ 아래와 같은 데이터가 저장된 [scores 테이블]을 가지고 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 다음의 SELECT문을 수행하여 점수의 합계를 구한다고 합시다.

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

◦ 문장의 SELECT 절에 명시된 각 집합함수 처리 결과는 다음과 같습니다.

| 사용된 집합함수 표현식 | 집합 함수에 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).

☞ ALL 이나 DISTINCT를 명시하지 않으면, ALL 방식으로 처리됩니다.

◦ 위의 테이블에 대하여 다음의 SELECT-문으로 점수의 평균 및 개수를 구한다고 합시다.

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

◦ 문장의 SELECT 절에 명시된 각 집합함수 처리 결과는 다음과 같습니다.

| 사용된 집합함수 Expression | 집합 함수에 의하여 처리 후 표시되는 값 |
|---|---|
| AVG (score) | 300/4 |
| AVG (NVL(score, 0)) | 300/5 |
| COUNT(score) | 4 |
| COUNT(NVL(score, 0)) = COUNT(*) | 5 |

☞ 집합함수 처리 시에 해당 컬럼에 데이터가 없는(NULL 상태임) 경우, 그 행은 무시하고 고려하지 않습니다.

☞ 집합함수 처리 시에 NULL 상태의 필드도 집합함수 처리에 포함해야 한다면, NVL() 함수로 해당 필드에 NULL을 대신할 적절한 값을 지정하여 집합함수가 처리하도록 해줍니다.

☞ COUNT(*) 함수는 별도로 **행-카운트 함수**라고 합니다.

## 5-3. GROUP FUNCTION 사용 연습.

■ EMPLOYEES 테이블에서 입사일이 가장 오래된 날짜와 입사일이 가장 최근인 날짜를 구하시오.

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

MIN(HIRE_DATE) MAX(HIRE_DATE)
-------------- --------------
01/01/13       08/04/21
```

■ EMPLOYEES 테이블에서 전체 직원의 인원수와 commission을 지급받는 직원의 인원수를 구하시오.

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

ALL_PERSONS COMM_PERSONS
----------- ------------
        107           35
```

■ EMPLOYEES 테이블에서 현재 직원이 근무하고 있는 부서의 개수 및 현재 근무 부서가 지정된 직원수를 구하시오.

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

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

■ EMPLOYEES 테이블에서 커미션을 지급받는 직원들만 고려하여 commission의 평균을 구하고, 또한 전체 직원을 대상으로 한(커미션을 받는 직원 및 받지 않는 직원을 모두 포함한 전체 직원) commission의 평균을 구하시오.

```
SQL> SELECT AVG(commission_pct) AS COMM_AVG_ONLY_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_PERSONS 에 표시된 값은 COMMISSION_PCT 컬럼이 NULL 이 아닌 것만 고려합니다(즉, 35로 나눔).

☞ COMM_AVG_ALL_PERSONS 에 표시된 값은 COMMISSION_PCT 컬럼이 NULL 인 것도 포함하여 고려합니다(즉, 107로 나눔).

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

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

```
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 ;

DEPTNO      AVG_SAL
------- -----------
     10        4400
     20        9500
```

[department_id=10 인 행을 찾아서 평균 임금을 구하는 SELECT 문]

UNION ALL

[department_id=20 인 행을 찾아서 평균 임금을 구하는 SELECT 문] ;

[참고] UNION ALL 연산자는 두 개의 SELECT 문을 처리한 결과 레코드를 합쳐서 표시해줍니다.

• 만약 30 번 부서까지 포함한 부서별 평균 임금을 구해야 한다면, 위의 SELECT-문에 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
```

■ 다음은 **부서별 평균 임금**을 구하기 위하여, GROUP BY 키워드를 사용한 SELECT문을 이용한 경우입니다.

```
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
------------- ---------------- ----------
          100       8601.33333          6
           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 GROUPING.

• 다음은 실제 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 컬럼 이름이 두 개 이상 명시된 경우, 행들은 GROUP BY 뒤에 명시된 컬럼들에 의해 구성되는 레코드가 같은 행들끼리 GROUPING 됩니다.

```
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
```

[주의] SELECT 절에서 집합함수와 같이 명시된 컬럼 이름들은 반드시 GROUP BY 다음에 모두 명시해야만 합니다.

• 만약 아래처럼 집합함수와 col1~col10까지 10개의 컬럼명이 명시되었다면 GROUP BY 절에서 하나도 누락되는 것 없이 col1~col10까지 모든 컬럼명을 명시해 주어야 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 ;
```

• SELECT 절에서 집합함수와 같이 명시된 컬럼이름이 GROUP BY 절에서 누락되거나, 또는 GROUP BY 절이 누락되면 다음처럼 에러가 발생됩니다.

| 1> SELECT 절에서 집합함수와 같이 사용된 컬럼 명이 GROUP BY 절에서 누락된 경우. | 2> SELECT 절에서 집합함수와 컬럼 명이 같이 명시되었는데, GROUP BY 절이 누락된 경우. |
|---|---|
| SQL> **SELECT department_id,job_id,AVG(salary)**<br>**FROM hr.employees**<br>**WHERE department_id in (10,20,30,40)**<br>**GROUP BY department_id ;**<br>ORA-00979: GROUP BY 표현식이 아닙니다.<br>00979. 00000 - "not a GROUP BY expression"<br>*Cause:<br>*Action:<br>1행, 22열에서 오류 발생 | SQL> **SELECT department_id, COUNT(last_name)**<br>**FROM hr.employees;**<br>ORA-00937: 단일 그룹의 그룹 함수가 아닙니다<br>00937. 00000 - "not a single-group group function"<br>*Cause:<br>*Action:<br>1행, 8열에서 오류 발생 |

**[주의] WHERE 절에는 GROUP FUNCTION을 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 집합함수가 처리됩니다.

따라서, 위에서처럼 행을 선택하는 기능을 위한 WHERE절에는 훨씬 뒤에 처리되는 집합함수가 명시될 수 없습니다.

```
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.

• GROUP BY 가 포함된 문장에 "집합함수가 포함된 조건절을 HAVING 절에 작성하여" 최종 결과를 선별할 수 있습니다.

• 다음은 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
```

■ EMPLOYEES 테이블에서 job_id에 REP가 포함된 직원에 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(*)
------------- ----------- ----------- ----------
                     7000        7000          1
           20        6000        6000          1
           70       10000       10000          1
           40        6500        6500          1
           80      243500  8396.55172         29
```

■ EMPLOYEES 테이블에서 job_id에 REP가 포함된 직원에 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
```

• 위에서처럼 HAVING 절을 이용하면 집합함수에 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 집합함수의 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. 집합함수의 중첩.

• 집합함수는 2번 까지만 중첩할 수 있습니다. 왜냐하면, 아래처럼 집합함수를 2번 중첩하면 무조건 하나의 결과가 되기 때문입니다.

■ EMPLOYEES 테이블을 이용하여, 부서별 임금의 합계 금액 중에서, 가장 큰 부서별 합계임금을 구하시오.

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

    RESULT
----------
    304500
```

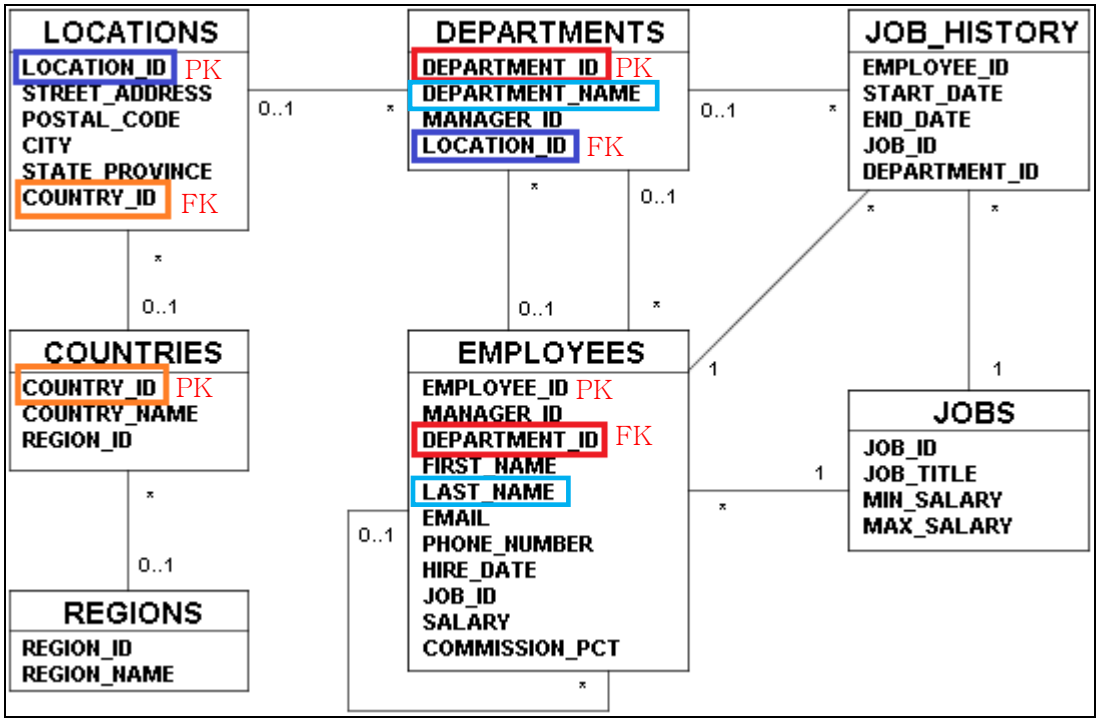
# 6 두 테이블의 행을 결합하여 데이터 조회하기(Join을 사용한 데이터 조회).

## ◈ 학습 목표.

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

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

## ▣ HR(Human Resources) SCHEMA DATA-SET.




- 위의 HR DATA-SET 그림에서, 직원의 사번(EMPLOYEES.EMPLOYEE_ID 컬럼값), 과 봉급(EMPLOYEES.SALARY 컬럼값) 및 그 직원이 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 SELECT-문을 하려고 합니다. 이를 위하여 사용되는 데이터 조회방법이 JOIN 입니다.

## 6-1. JOIN 개요.

• JOIN이란? 조인-조건을 기준으로 **두 테이블의 각 행들을 합친 후**, 원하는 데이터의 레코드를 가져오는 방법입니다.

• ANSI 표준 JOIN 형식: 다음의 4가지 (총 5 가지)조인 방법이 가능합니다.

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

• 조인 조건에 [=] 연산자를 사용하는 경우의 INNER-JOIN과 OUTER-JOIN 방법에서 조인 조건을 표시할 때, [ON-절] 또는 [USING-절]을 이용하는 2 가지 방법이 있습니다.

## 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 EMPLOYEES 테이블과 DEPARTMENTS 테이블의 데이터를 참고 합니다.

**EMPLOYEES 테이블 데이터**

```
EMPLOYEE_ID LAST_NAME                 DEPARTMENT_ID
----------- ------------------------- -------------
        200 Whalen                               10
        201 Hartstein                            20
        202 Fay                                  20
        114 Raphaely                             30
        119 Colmenares                           30
        115 Khoo                                 30
        116 Baida                                30
        117 Tobias                               30
        118 Himuro                               30
...
        113 Popp                                100
        109 Faviet                              100
        206 Gietz                               110
        205 Higgins                             110
        178 Grant

107개의 행이 선택됨
```

**DEPARTMENTS 테이블 데이터**

```
DEPARMENT_ID DEPARTMENT_NAME
------------- ------------------------------
           10 Administration
           20 Marketing
           30 Purchasing
...
          120 Treasury
          130 Corporate Tax
          140 Control And Credit
...
          240 Government Sales
          250 Retail Sales
          260 Recruiting
          270 Payroll

 27개의 행이 선택됨
```

■ EQUI-INNER JOIN 구문 예제

```
SQL> SELECT  hr.employees.employee_id, hr.employees.last_name, hr.employees.department_id
           , hr.departments.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_ID DEPARTMENT_NAME
----------- ------------------------- ------------- ------------- ------------------------------
        200 Whalen                               10            10 Administration
        201 Hartstein                            20            20 Marketing
        202 Fay                                  20            20 Marketing
        114 Raphaely                             30            30 Purchasing
        119 Colmenares                           30            30 Purchasing
        115 Khoo                                 30            30 Purchasing
        116 Baida                                30            30 Purchasing
        117 Tobias                               30            30 Purchasing
        118 Himuro                               30            30 Purchasing
        203 Mavris                               40            40 Human Resources
        198 OConnell                             50            50 Shipping
        199 Grant                                50            50 Shipping
        120 Weiss                                50            50 Shipping
        121 Fripp                                50            50 Shipping
        122 Kaufling                             50            50 Shipping
        123 Vollman                              50            50 Shipping
        124 Mourgos                              50            50 Shipping
        125 Nayer                                50            50 Shipping
        126 Mikkilineni                          50            50 Shipping
        127 Landry                               50            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 **= 연산자**가 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. 컬럼 이름 앞에 테이블의 이름을 접두어로 명시할 때 잇점.

```
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
```

- 컬럼 이름 앞에 테이블 접두어를 명시하면 코드 해석이 쉬워 지고 성능이 조금 개선됩니다.

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

## 6-4. TABLE-ALIAS(QUALIFIER) 사용하기.

```
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
        117 Tobias                               30 Purchasing
        118 Himuro                               30 Purchasing
...

106개의 행이 선택됨
```

- 컬럼이름 앞에 테이블 이름을 접두어를 사용하면, 구문이 길어 집니다. 위의 문장처럼 FROM 절에 명시되는 테이블 이름 옆에 [테이블 alias]를 선언하고, 이를 테이블 이름 대신 사용하여 문장을 기술하면, 보다 간결한 구문 작성이 가능해 집니다.

[참고] JOIN 키워드 앞에 JOIN 형식을 지정하는 키워드를 생략한 경우에는 무조건 INNER-JOIN 을 의미합니다.

## 6-5. 3 개 테이블의 조인(3-Way Join).

```
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
```

- 위의 문장처럼 3개의 테이블을 조인하는 것을 3-Way 조인이라고 합니다.

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

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

• EQUI-INNER JOIN 문장에서 조인 조건 작성 시에 컬럼 이름이 동일하고 데이터 유형이 일치되는 두 테이블의 특정 컬럼을 ON 절 대신 USING 절로 아래와 같이 명시할 수 있습니다.

```
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
```

■ EMPLOYEES, DEPARTMENTS, LOCATIONS 테이블을 이용하여, 사번이 100,150,200 인 직원의 성(EMPLOYEES.LAST_NAME 컬럼)과 그 직원이 출근하는 부서가 있는 도시이름(LOCATIONS.CITY 컬럼)을 조회하시오.

```
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
```

## 6-7. USING 절 사용 시의 주의 사항.

• 컬럼 이름이 같더라도 데이터 유형이 다르면 에러가 발생되며, USING 절에 명시된 컬럼에는 테이블-ALIAS(QUALIFIER)를 명시하면 안됩니다. USING 절에 정의된 컬럼에 테이블-ALIAS(QUALIFIER)를 사용하면, 아래의 에러가 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
```

## 6-8. 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.

• 실습을 위하여, 23페이지의 내용을 수행하여, JOB_GRADES 테이블을 생성하고, 필요한 데이터를 입력합니다.

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

EMPLOYEES 테이블 데이터

```
EMPLOYEE_ID LAST_NAME                     SALARY
----------- ------------------------ ----------
        132 Olson                          2100
        136 Philtanker                     2200
...
        195 Jones                          2800
        116 Baida                          2900
        134 Rogers                         2900
        190 Gates                          2900
        197 Feeney                         3000
...
        168 Ozer                          11500
        147 Errazuriz                     12000
...

 107개의 행이 선택됨
```

JOB_GRADES 테이블 데이터

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

 6개의 행이 선택됨
```

- JOB_GRADES 테이블은 직원들이 지급받는 급여가 포함되는 금액 범위에 따른 등급 정보를 제공합니다.

■ NON-EQUI-INNER JOIN 구문 예제

```
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
------------------------ ---------- --
Olson                          2100 A
Philtanker                     2200 A
...
Feeney                         3000 B
Cabrio                         3000 B
...
Kumar                          6100 C
Banda                          6200 C
...
Tucker                        10000 D
Vishney                       10500 D
...
Kochhar                       17000 E
King                          24000 E

 107개의 행이 선택됨
```

## 6-9. OUTER JOIN.

• INNER JOIN 결과도 출력되고, [INNER JOIN]에서 조인 조건을 만족시키지 못해서 표시되지 않는 레코드를 추가적으로 더 표시해 줍니다. 다음의 3 가지 방법이 가능합니다.

(1) RIGHT OUTER JOIN

INNER JOIN 결과도 출력되고, JOIN 키워드의 오른편에 있는 테이블에서 조인 조건을 만족시키지 못해서 표시되지 않는 레코드를 더 표시해줍니다.

(2) LEFT OUTER JOIN

INNER JOIN 결과도 출력되고, JOIN 키워드의 왼편에 있는 테이블에서 조인 조건을 만족시키지 못해서 표시되지 않는 레코드를 더 표시해줍니다.

(3) FULL OUTER JOIN

INNER JOIN 결과도 출력되고, 조인되는 모든 테이블에서 조인 조건을 만족시키지 못해서 표시되지 않는 레코드를 더 표시해줍니다.

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

EMPLOYEES 테이블 데이터

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|---|---|
| 200 | Whalen | 10 |
| 201 | Hartstein | 20 |
| 202 | Fay | 20 |
| 114 | Raphaely | 30 |
| 119 | Colmenares | 30 |
| 115 | Khoo | 30 |
| 116 | Baida | 30 |
| 117 | Tobias | 30 |
| 118 | Himuro | 30 |
| 203 | Mavris | 40 |
| ... | | |
| 110 | Chen | 100 |
| 108 | Greenberg | 100 |
| 111 | Sciarra | 100 |
| 112 | Urman | 100 |
| 113 | Popp | 100 |
| 109 | Faviet | 100 |
| 206 | Gietz | 110 |
| 205 | Higgins | 110 |
| 178 | Grant | |

107개의 행이 선택됨

DEPARTMENTS 테이블 데이터

| DEPARTMENT_ID | DEPARTMENT_NAME |
|---|---|
| 10 | Administration |
| 20 | Marketing |
| 30 | Purchasing |
| ... | |
| 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 |

27개의 행이 선택됨

■ LEFT OUTER JOIN 구문 예제

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, d.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_ID DEPARTMENT_NAME
----------- ------------------------- ------------- ------------- ------------------------------
        200 Whalen                               10            10 Administration
        202 Fay                                  20            20 Marketing
        201 Hartstein                            20            20 Marketing
        119 Colmenares                           30            30 Purchasing
        118 Himuro                               30            30 Purchasing
        117 Tobias                               30            30 Purchasing
...
        108 Greenberg                           100           100 Finance
        206 Gietz                               110           110 Accounting
        205 Higgins                             110           110 Accounting
        178 Grant

 107개의 행이 선택됨
```

■ RIGHT OUTER JOIN 구문 예제

```
SQL> SELECT e.employee_id, e.last_name, e.department_id, 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_ID DEPARTMENT_NAME
----------- ------------------------- ------------- ------------- ------------------------------
        200 Whalen                               10            10 Administration
        201 Hartstein                            20            20 Marketing
        202 Fay                                  20            20 Marketing
        114 Raphaely                             30            30 Purchasing
        119 Colmenares                           30            30 Purchasing
        115 Khoo                                 30            30 Purchasing
...
        205 Higgins                             110           110 Accounting
                                                              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

 122개의 행이 선택됨
```

■ FULL OUTER JOIN 구문 예제

```
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
        179 Johnson                              80            80 Sales
...
        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
```

■ OUTER JOIN 구문설명

결과를 보면, 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.

## 6-10. 테이블 JOIN 시에 추가적인 조건 적용하기.

▣ INNER-JOIN에서는 추가적인 조건을 기술할 때 ON 절 다음에 [AND-절] 또는 [WHERE-절**]을 이용할 수 있습니다.

■ EMPLOYEES 및 DEPARTMENTS 테이블을 이용하여, 90번 부서에 근무하는 직원의 성과 근무하는 부서이름을 조회하시오.

1> 추가적인 조건을 WHERE-절을 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
```

2> 추가적인 조건을 AND-절을 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-JOING의 경우에는 추가적인 조건을 ON-절 다음에 [AND-절]을 사용하거나 별도의 [WHERE-절]을 사용하거나 상관없이 동일한 표시결과를 얻을 수 있습니다.

## 6-11. 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.

■ CROSS JOIN 구문 예제

```
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
```

■ 잘못된 CROSS JOIN 사용: 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
        100 King                      Administration
...

 2,889개의 행이 선택됨
```

☞ CROSS JOIN 사용 시에는 행을 합쳐야 하는 테이블의 행수를 극소수로 제한하는 조건절을 WHERE 절 및 AND 을 이용하여 명시하십시오.

## [참고] ORACLE JOIN 문법.

(1) 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 ;
```

(2) 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 (+) 사용 시 에러가 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
```

(3) 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  ;
```

(4) 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  ;
```

(5) 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 서브쿼리(SUBQUERY)를 사용한 데이터 조회.

◈ 학습 목표.

■ SELECT-문의 [WHERE]절과 [HAVING]절에서 서브쿼리를 사용하는 기본적인 방법 및 주의할 사항을 학습합니다.

## 7-1. 서브쿼리(SUBQUERY) 개요.

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

| SELECT-1 | SELECT-2 |
|---|---|
| SQL> **SELECT employee_id, last_name, salary**<br>**FROM hr.employees**<br>**WHERE salary > ???;** --??? 대신 11000 | SQL> **SELECT salary**<br>**FROM hr.employees**<br>**WHERE last_name = 'Abel' ;**<br><br>SALARY<br>----------<br>11000 |

• 앞의 두 SELECT-문(SELECT-1과 SELECT-2)들을 다음처럼 하나로 합쳐서 작성할 수 있습니다. 아래에서 ( ) 안에 기술된 SELECT 문을 "서브쿼리"라고 하며, 서브쿼리가 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
```

## 7-2. WHERE 절 또는 HAVING 절에서 서브쿼리(Subquery) 사용에 대한 가이드라인.

• 비교 조건에서 서브쿼리를 연산자의 오른편에 괄호로 감싸서 위치시킵니다.

• WHERE 절에 기술된 서브쿼리 안에는 ORDER BY 절을 적으면 안됩니다.

• WHERE/HAVING 절에 사용하는 서브쿼리 안에 [ORDER BY-절]을 사용하면, 아래처럼 에러가 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
```

• 다음은 WHERE절 또는 HAVING 절에 서브쿼리가 사용된 경우, 결과 레코드를 정렬해서 표시하기 위하여 [ORDER BY-절]을 명시하는 올바른 방법에 대한 예제입니다. 아래의 예제에서처럼 [ORDER BY-절]은 메인쿼리의 제일 마지막에 명시합니다.

```
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
```

## 7-3. [WHERE 절] 또는 [HAVING 절]에 적는 서브-쿼리의 형식.

• 서브쿼리 앞에 위치한 연산자에 따라서 2가지 형식의 서브쿼리로 분류됩니다.

| 서브쿼리 형식 | 설명 |
|---|---|
| (1) 단일-행 서브쿼리(SINGLE-ROW SUBQUERY) | 서브쿼리가 오직 하나의 행에서 값을 반환합니다. |
| (2) 다중-행 서브쿼리(MULITPLE-ROW SUBQUERY) | 서브쿼리가 하나 이상의 행에서 값을 반환합니다. |

## 7-4. WHERE/HAVING 절에서 단순 비교 연산자를 사용한 단일-행 서브쿼리 및 주의할 점.

• 서브쿼리가 메인 쿼리의 [WHERE/HAVING]절에 사용될 때, 서브쿼리 앞에 연산자가 **단순 비교 연산자**가 사용되면, **서브쿼리는 무조건 하나의 행으로부터 원하는 레코드를 반환**해야 합니다. 이를 단일-행 서브쿼리라고 합니다.

• 단순 비교 연산자(SINGLE ROW OPERATOR)의 종류

| 같다 | 크다 | 크거나 같다 | 작다 | 작거나 같다 | 같지않다 |
|---|---|---|---|---|---|
| = | > | >= | < | <= | <> , != , ^= |

• 만약 단일-행 서브쿼리가 둘 이상의 행으로부터 레코드를 메인쿼리로 반환하는 경우에는 아래처럼 에러가 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 ) ;      -- 서브쿼리가 반환하는 레코드가 12개 입니다.
ORA-01427: 단일 행 하위 질의에 2개 이상의 행이 리턴되었습니다.
01427. 00000 -  "single-row subquery returns more than one row"
*Cause:
*Action:
```

☞ 위의 2번째 SELECT-문에서 =을 IN으로 변경하면, 문장은 에러없이 실행될 수 있습니다.
단, = 또는 IN 연산자를 사용하는 결정은 SELECT 문의 결과를 사용하는 사용자 입장에서 결정해 주어야 합니다.

## 7-5. WHERE 또는 HAVING 절에 적는 단일-행 서브쿼리에서 집합 함수의 사용.

■ EMPLOYEES 테이블의 데이터를 이용하여, 90번 부서에서 salary가 가장 작은 직원의 salary와 같은 salary를 받는 직원의 LAST_NAME과 SALARY를 구하시오.

```
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
```

■ EMPLOYEES 테이블의 데이터를 이용하여, department_id가 50인 부서의 최소 salary 보다 [부서의 최소 salary가 큰] 각 부서별 최소 salary를 구하시오.

```
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
                     7000
           20        6000
           70       10000
           90       17000
          110        8300
           40        6500
           80        6100
           10        4400
           60        4200

 11개의 행이 선택됨
```

■ EMPLOYEES 테이블의 데이터를 이용하여, department_id별 평균salary가 가장 큰 department_id와 평균 salary를 구하시오.

```
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
```

■ EMPLOYEES 테이블의 데이터를 이용하여, JOB_ID별 평균salary가 가장 작은 JOB_ID와 평균salary를 구하시오.

```
SQL> SELECT   job_id, AVG(salary)
     FROM     employees
     GROUP BY job_id
     HAVING   AVG(salary) = (SELECT   MIN(AVG(salary))
                             FROM     employees
                             GROUP BY job_id);

JOB_ID     AVG(SALARY)
---------- -----------
PU_CLERK          2780
```

## 7-6. WHERE 또는 HAVING 절에 적는 단일-행 서브쿼리 사용 시의 주의점.

• 서브쿼리가 메인쿼리로 반환하는 것이 없으면(서브쿼리의 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
```

• 위와 같은 경우가 문제가 되는 상황을 고려해 봅시다.

만약 회사에 직원이 1000명이 근무하고 있고, 올해 신입사원 3명을 뽑았는데, 이 신입사원의 근무부서가 아직 결정되지 않았습니다. 즉, EMPLOYEES 테이블에 해당 신입사원에 대한 DEPARTMENT_ID는 NULL 상태입니다.

이 때 개발자가 사번을 입력했을 때, 해당 사번의 직원과 같은 부서에 근무하는 직원들의 이름과 업무코드를 확인하는 SELECT 문이 포함된 프로그램을 만들어서 현장부서에 사용하도록 전달했습니다.

```
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
```

이 때, 현장부서 사용자는 신입사원의 사번을 명시한 경우에, 신입사원 3명의 데이터가 출력되기를 원하지만 결과는 위에서처럼 **선택된 행 없음** 으로 아무것도 표시되지 않습니다.

- 서브쿼리 사용 시, 발생될 수 있는 위와 같은 문제점들을 해결하기 위한 하나의 방법으로 테이블에 데이터를 입력 시에, EMPLOYEES.DEPARTMENT_ID 컬럼을 NULL 상태로 입력하지 않고 문자 데이터유형인 경우 '-' 값을, 숫자 데이터 유형 컬럼은 0 같은 대체값을 미리 입력해 주면, 결과가 표시될 수 있습니다.

- 다른 방법으로는 아래처럼 NVL()함수를 이용하여 SQL-문을 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
------------------------- ----------
Grant                     SA_REP
```

단, 이 방법은 데이터의 양이나 인덱스-키 값의 구성 방법에 따라, 데이터베이스에 부하가 될 수도 있습니다.

## 7-7. MULTIPLE-ROW 연산자 (ANY, ALL, IN)를 사용한 다중-행 서브쿼리.

• 메인 SELECT-문의 WHERE 절이나 HAVING 절에서 ANY, ALL, IN 같은 연산자 뒤에 명시된 서브쿼리를 **다중 행 서브쿼리** 라고 합니다.

• 다중-행 연산자의 종류 및 의미

| 여러 값 중 하나 | 모든 값 | 여러 값 중 하나와 같다 |
|---|---|---|
| ANY | ALL | IN |

☞ [IN]연산자만 단독으로 사용되며, [ANY 또는 ALL]연산자는 단순 비교 연산자와 조합되어 사용됩니다.

• 아래의 표는 WHERE 절에 사용된 다중 행 연산자의 의미를 설명합니다.

| ANY/ALL 다중행 연산자 | 의미 |
|---|---|
| 컬럼 <ANY (서브쿼리) | 가장 큰 값보다 작음. |
| 컬럼 >ANY (서브쿼리) | 가장 작은 값보다 큼. |
| 컬럼 <ALL (서브쿼리) | 가장 작은 값보다 작음. |
| 컬럼 >ALL (서브쿼리) | 가장 큰 값보다 큼. |

☞ 다중 행 서브쿼리가 반환하는 값들이 100, 200, 300 이라고 가정하고 각 연산자의 의미를 스스로 확인해 봅니다.

## 7-8. MULTIPLE-ROW 연산자(ANY, ALL, IN)를 사용한 다중-행 서브쿼리 사용하는 실습.

■ EMPLOYEES 테이블의 데이터를 이용하여 department_id가 10, 20, 30 인 직원들의 department_id 별 평균salary 모두 보다 적은 salary를 받는 직원의 employee_id, last_name, salary를 구하시오.

```
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
```

• 메인-쿼리의 WHERE 절에 다중행 서브 쿼리가 사용된 앞의 예제 문장은 단순 비교 연산자를 이용한 단일행 서브쿼리로 변환하면 아래와 같습니다(MIN() 함수로 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
```

■ EMPLOYEES 테이블의 데이터를 이용하여 department_id가 10, 20, 30 인 직원들의 department_id 별 평균salary 중 하나보다 작거나 같은 salary를 받는 직원의 employee_id, last_name, salary를 구하시오.

```
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
        182 Sullivan                       2500
...

 87개의 행이 선택됨
```

• 메인-쿼리의 WHERE 절에 다중행 서브 쿼리가 사용된 앞의 예제 문장은 단순 비교 연산자를 이용한 단일행 서브쿼리로 변환하면 아래와 같습니다(MIN() 함수로 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
        113 Popp                           6900
...

87개의 행이 선택됨
```

■ EMPLOYEES 테이블의 데이터를 이용하여 department_id가 10, 20, 30 인 직원들의 department_id 별 평균salary 중 하나와 같은 salary를 받는 직원의 employee_id, last_name, salary를 구하시오.

```
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
```

## 7-9. 메인-쿼리의 WHERE/HAVING 절에 [IN 또는 NOT IN] 연산자와 다중-행 서브쿼리가 함께 사용된 경우.

• HR.EMPLOYEES 테이블에 있는 MANAGER_ID 컬럼에는 해당 직원의 직속 상관의 사번이 저장되어 있습니다.

■ EMPLOYEES 테이블의 데이터를 이용하여, 다른 직원의 상관으로 근무하는 직원의 employee_id, last_name, salary를 구하시오.

```
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 Fripp                        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
```

■ EMPLOYEES 테이블의 데이터를 이용하여, 회사에서 다른 직원의 상관이 아닌(즉, 부하직원이 없는) 직원들의 employee_id, last_name, salary, manager_id를 구하시오.

```
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
```

- 부하직원이 없는 직원들의 데이터가 존재함에도 결과가 no rows selected로 표시됩니다. 왜냐하면, 메인쿼리의 WHERE절에서 서브쿼리 앞에 NOT IN 연산자 사용될 경우에 서브쿼리로부터 반환되는 결과에 NULL이 포함되어 있으면, 메인쿼리의 결과는 항상 no rows selected로 표시됩니다.

- 따라서, 메인쿼리의 WHERE절에서 서브쿼리 앞에 NOT IN 연산자 사용될 경우에 주의해야 합니다.

- 이를 해결하기 위하여 메인쿼리의 WHERE 절 또는 HAVING 절에서 NOT IN 연산자와 서브쿼리가 같이 사용되는 경우에는 서브쿼리에 IS NOT NULL 조건절을 추가하여, 서브쿼리에 NULL을 배제시켜야 합니다(아래의 구문).

```
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
----------- ------------------------ ---------- ----------
        198 OConnell                       2600        124
        199 Grant                          2600        124
        200 Whalen                         4400        101
        202 Fay                            6000        201
        203 Mavris                         6500        101
        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
        111 Sciarra                        7700        108
        112 Urman                          7800        108
...

 89개의 행이 선택됨
```

[참고] 서브쿼리의 SELECT 컬럼(위의 예제에서 EMPLOYEES.MANAGER_ID 컬럼)에 NOT NULL 규칙이 없으면, 서브쿼리의 WHERE 절에 컬럼 IS NOT NULL 조건을 포함시킵니다.

# 8 SET 연산자(SET OPERATOR)의 활용.

◈ 학습 목표.

■ 아래의 SET 연산자를 이용하여 두 개의 SELECT-문들의 결과 레코드들을 처리하는 방법을 학습합니다.

- UNION 및 UNION ALL
- INTERSECT
- MINUS

[참고] 본 레슨의 실습 테이블 소개.

HR.EMPLOYEES 테이블

```
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)
```

```
SELECT employee_id, job_id, department_id
FROM hr.employees
WHERE employee_id in
              (101,102,114,122,176,200,201)
ORDER BY 1;

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

 7개의 행이 선택됨
```

HR.JOB_HISTORY 테이블

```
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)
```

```
SELECT employee_id, job_id, department_id
FROM hr.job_history
ORDER BY 1;

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
```

- HR.EMPLOYEES 테이블에는 현재 근무하고 있는 직원들의 현재-업무에 대한 정보를 제공합니다.
- HR.JOB_HISTORY 테이블에는 현재 근무하는 직원들의 과거-업무에 대한 정보를 제공합니다.

## 8-1. SET 연산자 개요.

■ **SET 연산자 종류**

• UNION ALL, UNION, INTERSECT, MINUS 의 4개의 SET 연산자가 있습니다.

■ **SET 연산자 사용 위치**: 두 SELECT 문 사이에 위치합니다.

| |
|---|
| SELECT-1 SET연산자 SELECT-2 ; |

■ **4 개의 SET 연산자의 기능.**

| SET 연산자 | 처리 방법 |
|---|---|
| UNION | SELECT-1 문장을 처리하여 결과-레코드 집합를 구하고, SELECT-2 문장을 처리하여 결과-레코드 집합을 구한 뒤, 이 결과-레코드 집합들을 하나로 합칩니다. 그리고, 합쳐진 결과-레코드들을 첫 번째 필드를 기준으로 정렬하여 중복된 레코드 제거하여, 최종적인 결과-레코드들을 표시합니다. |
| UNION ALL | SELECT-1 문장을 처리하여 결과-레코드 집합를 구하고, SELECT-2 문장을 처리하여 결과-레코드 집합을 구한 뒤, 이 결과-레코드 집합들을 하나로 합쳐서 그대로 표시합니다. UNION 과는 달리 중복된 레코드들을 제거하지 않습니다. 따라서 정렬작업이 수행되지 않습니다. |
| INTERSECT | SELECT-1 문장을 처리할 때, 첫 번째 필드를 기준으로 정렬시키면서 결과-레코드 집합를 구하고, SELECT-2 문장을 처리할 때, 첫 번째 필드를 기준으로 정렬시키면서 결과-레코드 집합를 구한 후, 두 SELECT-문 각각의 정렬된 결과-레코드 집합으로부터 공통된 결과-레코드들만 추출하며, 한번만 표시합니다. |
| MINUS | SELECT-1 문장을 처리할 때, 첫 번째 필드를 기준으로 정렬시키면서 결과-레코드 집합를 구하고, SELECT-2 문장을 처리할 때, 첫 번째 필드를 기준으로 정렬시키면서 결과-레코드 집합를 구한 후, SELECT-1의 결과-레코드 집합으로부터 [SELECT-1과 SELECT-2의 공통된 결과 레코드들]를 제거하여, SELECT-1에만 있는 결과-레코드들을 표시합니다. |

☞ UNION ALL 연산자를 제외하고, 나머지 SET연산자는 처리 중에 정렬(SORT)이 발생하므로 처리할 레코드의 양에 따라서 메모리 소모가 많을 수 있습니다.

## 8-2. SET 연산자 사용 시 가이드라인.

• SET 연산자의 대상이 되는 SELECT-문장들에서 각 문장들에 명시된 컬럼의 개수와 위치에 따른 데이터유형이 모두 일치해야 합니다.

• 표시된 결과 상의 해딩(HEADING)은 항상 첫 번째 SELECT-문장에 기술된 해딩이 사용됩니다. 따라서, 컬럼-별칭을 정의하여 해딩을 변경하려면, 첫 번째 SELECT-문장에 컬럼-별칭을 기술해야 합니다.

• 3 개 이상의 SELECT-문들을 여러 개의 SET-연산자를 이용하여 처리할 때, 위에서부터 아래로 처리되면서 순서대로 처리됩니다(예제-1). 이러한 기본 처리 순서는 괄호()를 이용하여 변경할 수 있습니다(예제-2).

[예제-1]

```
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
        101 AC_MGR
        102 IT_PROG
        114 ST_CLERK
        122 ST_CLERK
        176 SA_MAN
        176 SA_REP
        200 AC_ACCOUNT
        200 AD_ASST
        201 MK_REP

 10개의 행이 선택됨
```

[예제-2]

```
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
        101 AC_ACCOUNT
        101 AC_MGR
        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
```

☞ [예제-1]에서는 먼저 처음 두 개의 SELECT-문장들의 레코드들을 UNION 연산자로 처리한 후, 그 결과 레코드들과 세 번째 SELECT-문장의 결과-레코드를 MINUS 연산자로 처리합니다.

☞ [예제-2]에서는 먼저 두 번째와 세 번째 SELECT-문장들의 레코드들을 MINUS 연산자로 처리한 후, 그 결과 레코드들과 첫 번째 SELECT-문장의 그 결과 레코드를 UNION 연산자로 처리합니다.

• SET-연산자가 포함된 문장에서 [ORDER BY-절]은 전체 문장의 맨 마지막에 기술해야 하며,
이 때, 첫 번째 SELECT-문장에 명시된 해딩 또는 숫자로 명시한 위치 표기법을 사용합니다.

[참고] 아래의 예제에서처럼, [ORDER BY-절]을 전체 문장의 맨 마지막이 아닌 다른 위치에 기술하면 에러가 발생합니다.

| [예제-1] | [예제-2] |
|---|---|
| SQL> **SELECT employee_id, last_name**<br>**FROM hr.employees**<br>**ORDER BY last_name**<br>**UNION ALL**<br>**SELECT employee_id, job_id**<br>**FROM hr.job_history ;**<br><br>ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다<br>00933. 00000 - "SQL command not properly ended"<br>*Cause:<br>*Action:<br>4행, 6열에서 오류 발생 | SQL> **SELECT employee_id, job_id**<br>**FROM hr.employees**<br>**WHERE department_id=90**<br>**UNION**<br>**( SELECT employee_id, job_id**<br>**FROM hr.job_history**<br>**MINUS**<br>**SELECT employee_id, job_id**<br>**FROM hr.employees**<br>**WHERE department_id = 90**<br>**ORDER BY 1) ;**<br><br>ORA-00907: 누락된 우괄호<br>00907. 00000 - "missing right parenthesis"<br>*Cause:<br>*Action:<br>11행, 6열에서 오류 발생 |

☞ [예제-1]에서는 [ORDER BY-절] 위치가 첫 번째 SELECT-문의 끝에 있어서 에러가 발생되었습니다.

☞ [예제-2]에서는 [ORDER BY-절] 위치가 괄호로 묶여진 세번째 SELECT-문장의 끝에 있어서 에러가 발생되었습니다.

## 8-3. SET 연산자 사용 실습.

■ UNION 및 UNION ALL 연산자의 사용.

```
SQL> SELECT employee_id, job_id
     FROM hr.employees
     WHERE employee_id in
               (101,102,114,122,176,200,201)
     UNION
     SELECT employee_id, job_id
     FROM hr.job_history ;

EMPLOYEE_ID JOB_ID
----------- ----------
        101 AC_ACCOUNT
        101 AC_MGR
        101 AD_VP
        102 AD_VP
        102 IT_PROG
        114 PU_MAN
        114 ST_CLERK
        122 ST_CLERK
        122 ST_MAN
        176 SA_MAN
        176 SA_REP
        200 AC_ACCOUNT
        200 AD_ASST
        201 MK_MAN
        201 MK_REP

 15개의 행이 선택됨
```

```
SQL> SELECT employee_id, job_id
     FROM hr.employees
     WHERE employee_id in
               (101,102,114,122,176,200,201)
     UNION ALL
     SELECT employee_id, job_id
     FROM hr.job_history
     ORDER BY 1 ;

EMPLOYEE_ID JOB_ID
----------- ----------
        101 AC_ACCOUNT
        101 AD_VP
        101 AC_MGR
        102 AD_VP
        102 IT_PROG
        114 PU_MAN
        114 ST_CLERK
        122 ST_CLERK
        122 ST_MAN
        176 SA_MAN
        176 SA_REP
        176 SA_REP
        200 AD_ASST
        200 AC_ACCOUNT
        200 AD_ASST
        201 MK_MAN
        201 MK_REP

 17개의 행이 선택됨
```

☞ 위의 UNION 연산자를 사용한 예제의 결과에서 중복된 레코드가 제거되었고, ORDER BY-절을 사용하지 않았지만, 첫 번째 필드를 기준으로 최종적인 결과레코드가 정렬되어 있습니다.

☞ 위의 UNION ALL 연산자를 사용한 예제의 결과에서는 SELECT-문장들의 결과-레코드를 그대로 합쳐서 표시합니다. 동일한 레코드의 제거가 발생되지 않습니다.

■ INSERTSECT와 MINUS 연산자의 사용.

```
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
```

```
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
```

## 8-4. SET 연산자 사용 시 도움이 되는 팁.

[참고] 두 SELECT 문의 컬럼개수와 데이터 유형이 일치 하지 않을 경우 아래처럼 에러가 발생됩니다.

```
SQL> SELECT department_id, hire_date
     FROM hr.employees
     UNION
     SELECT department_id, location_id
     FROM hr.departments;

ORA-01790: 대응하는 식과 같은 데이터 유형이어야 합니다
01790. 00000 -  "expression must have same datatype as corresponding expression"
*Cause:
*Action:
1행, 23열에서 오류 발생
```

- 이 때, 이렇게 SET 연산자 사용 시에 컬럼 개수나 데이터 유형이 일치 하지 않을 때, (1) TO_CHAR(NULL), TO_NUMBER(NULL), TO_DATE(NULL) 함수를 이용하거나 (2) NULL 키워드를 이용하거나 또는 (3) 데이터 유형에 따른 적절한 상수를 이용하여 이를 해결할 수 있습니다.

■ SET 연산자 사용 시 TO_CHAR(NULL), TO_NUMBER(NULL), TO_DATE(NULL) 함수를 사용하여 SELECT 문들 간, 컬럼의 데이터유형 불일치를 해결하는 실습.

```
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
           10             03/09/17
           20        1800
           20             04/02/17
           20             05/08/17
           30        1700
           30             02/12/07
           30             03/05/18
           30             05/07/24
           30             05/12/24
           30             06/11/15
           30             07/08/10
           40        2400
           40             02/06/07
...

 130개의 행이 선택됨
```

■ SET 연산자 사용 시, NULL 키워드를 이용하여 SELECT 문들 간, 컬럼의 데이터유형 불일치를 해결하는 실습

```
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
           10             03/09/17
           20        1800
           20             04/02/17
           20             05/08/17
           30        1700
           30             02/12/07
           30             03/05/18
           30             05/07/24
           30             05/12/24
           30             06/11/15
           30             07/08/10
           40        2400
           40             02/06/07
...

 130개의 행이 선택됨
```

■ SET 연산자 사용 시, 적절한 상수를 이용하여 SELECT 문들 간, 컬럼의 데이터유형 불일치를 해결하는 실습

```
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
```

# 9 사용자 데이터의 수정(DATA MANIPULATION LANGUAGE, DML).

## ◈ 학습 목표.

■ 테이블에 데이터를 수정하는 방법을 학습합니다.

■ 트랜잭션이 무엇인지 그리고 오라클 제품의 트랜잭션 처리 특징에 대하여 학습합니다.

■ 오라클의 UNDO 구조에 대하여 소개합니다.

## 9-1. 대표적인 DML-문장 종류.

• 데이터베이스 테이블에 아래의 작업을 수행하는 문장을 DML-문장이라고 합니다.

| | |
|---|---|
| INSERT-문장 | 새로운 행(ROW) 또는 행들을 테이블에 입력합니다. |
| UPDATE-문장 | 테이블에 입력된 행 또는 행들의 컬럼(들)의 데이터를 수정합니다. |
| DELETE-문장 | 테이블에 저장된 행 또는 행들을 삭제합니다. |

## [참고] 트랜잭션을 정상적으로 종료하는 SQL-문: COMMIT-문 및 ROLLBACK-문.

• 논리적 작업 단위를 형성하는 DML-문장(들)의 모음을 트랜잭션(Transaction)이라고 합니다.

• 트랜잭션을 정상적으로 종료하기 위하여 사용자가 기본적으로 사용하는 SQL-문은 다음의 2 가지가 있습니다.

| | |
|---|---|
| COMMIT ; | DML-문장 완료 후, 변경 후의 데이터 상태를 유지하면서 트랜잭션을 종료합니다. |
| ROLLBACK ; | DML-문장 완료 후, 변경 전의 데이터 상태로 되돌려 놓고 트랜잭션을 종료합니다. |

**[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. Oracle 서버에서 세 개의 DML문을 수행할 때 계좌 잔액이 정확히 유지되어야 합니다. 특정 문제로 인해 트랜잭션을 구성하는 DML-명령문 중 하나가 실행되지 못하면 트랜잭션을 구성하는 다른 DML-명령문도 모두 롤백(또는 UNDO)되어야 합니다

## 9-2. 데이터베이스의 특정 테이블에 한 행을 삽입하기.

• [INSERT INTO ... VALUES] 문장을 이용하면, 한 번에 하나의 행을 입력할 수 있습니다.

• 아래의 예제 실습을 통하여 [INSERT INTO ... VALUES] 문장의 기본적인 문법 및 사용방법을 익혀봅니다.

■ HR.DEPARTMENTS 테이블에 한 행을 삽입하는 예제-1: 테이블의 모든 컬럼에 값을 입력.

```
SQL> INSERT INTO hr.departments(department_name, department_id, location_id, manager_id)
     VALUES ('Oracle SQL', 310, 1700, 100);

1개 행 이(가) 삽입되었습니다.
```

☞ [INSERT INTO 절]에 행을 입력하려는 테이블-이름과 컬럼명들을 명시하고, [VALUES-절]의 괄호 안에 명시된 컬럼들의 순서대로 값들을 명시합니다. [VALUES-절]에 값들을 명시할 때, 숫자형식의 값은 그대로 기술하고, 문자형식의 값은 작은 따옴표(')로 감싸줍니다. DATE-데이터유형의 값은 TO_DATE() 함수로 처리하여 입력합니다.

☞ 위의 INSERT-문에서 HR.DEPARTMENTS 테이블에 정의된 컬럼순서가 바뀌었지만, 테이블에 정의된 컬럼의 순서대로 INSERT-문이 수정되어 데이터가 입력됩니다.

☞ INSERT INTO ..VALUES 문장을 이용하여 테이블에 한 행을 입력할 때는, 테이블에 정의된 모든 컬럼을 고려하고 또한, 테이블에 정의된 컬럼의 순서대로 값을 입력하는 것을 권장합니다.

[참고] DESCRIBE 명령어를 이용하여, HR.DEPARTMENTS 테이블에 정의된 컬럼의 순서를 확인합니다.

```
SQL> desc hr.departments
 Name                    Null?    Type
 ----------------------- -------- ----------------
 DEPARTMENT_ID           NOT NULL NUMBER(4)
 DEPARTMENT_NAME         NOT NULL VARCHAR2(30)
 MANAGER_ID                       NUMBER(6)
 LOCATION_ID                      NUMBER(4)
```

■ HR.DEPARTMENTS 테이블에 한 행을 삽입하는 예제-2: 테이블의 모든 컬럼에 값을 입력.

```
SQL> INSERT INTO hr.departments
     VALUES (320, 'Oracle Administration', 100, 1700);

1개 행 이(가) 삽입되었습니다.
```

☞ [INSERT INTO-절]에서 테이블의 이름 다음에 컬럼명들이 생략되면, 테이블의 모든 컬럼을 대상으로 행이 입력됩니다.

■ HR.DEPARTMENTS 테이블에 한 행을 삽입하는 예제-3: 테이블의 일부 컬럼에만 값을 입력.

```
SQL> INSERT INTO hr.departments (department_id, department_name)
     VALUES (330, 'Oracle Maintenance');

1개 행 이(가) 삽입되었습니다.
```

☞ [INSERT INTO-절]에서 테이블의 이름 다음에 컬럼명들을 일부만 명시하면, 명시된 컬럼에는 값이 지정되고, 명시되지 않는 컬럼(들)은 NULL 상태로 한 행이 입력됩니다. 위의 예제에서 HR.DEPARTMENTS 테이블에 한 행을 입력할 때, DEPARTMENT_ID 컬럼과 DEPARTMENT_NAME 컬럼에만 값이 지정되고, MANAGER_ID 컬럼과 LOCATION_ID 컬럼은 NULL 상태로 한 행이 입력됩니다.

■ HR.DEPARTMENTS 테이블에 한 행을 삽입하는 예제-4: 테이블의 일부 컬럼에만 값을 입력.

```
SQL> INSERT INTO hr.departments
     VALUES (340, 'Oracle PLSQL', NULL, NULL);

1개 행 이(가) 삽입되었습니다.
```

☞ 값을 지정하지 않은 컬럼에 대하여 NULL 키워드를 명시하여 한 행을 입력합니다.

☞ 일부 컬럼에만 값을 지정하여 한 행을 입력하는 경우에는, 이 방법을 권장합니다.

■ HR.DEPARTMENTS 테이블에 한 행을 삽입하는 예제-5: 테이블의 일부 컬럼에만 값을 입력.

```
SQL> INSERT INTO hr.departments
     VALUES (350, 'Oracle Modeling', '', '');

1개 행 이(가) 삽입되었습니다.
```

☞ 값을 지정하지 않은 컬럼에 대하여 NULL-키워드 대신, 작은따옴표(')를 연달아 2번 표기하여 한 행을 입력합니다. 작은따옴표(')를 연달아 2번 표기하면. 길이가 0 인 값은 없기 때문에 NULL 상태로 입력됩니다.

■ HR.EMPLOYEES 테이블에 한 행을 삽입하는 예제-1: 함수처리결과를 값으로 입력.

```
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.
```

☞ FIRST_NAME, LAST_NAME, HIRE_DATE, 컬럼에 대하여, INITCAP() 함수, INITCAP() 함수, SYSDATE 함수를 이용하여 각각의 값을 처리한 후, 함수처리 결과로 반환된 값을 이용하여 한 행을 입력합니다.

■ HR.EMPLOYEES 테이블에 한 행을 삽입하는 예제-1: 함수처리결과를 값으로 입력.

```
SQL> INSERT INTO hr.employees
     VALUES (302, UPPER('Den'), UPPER('Raphealy'),'hong@yahoo.co.kr', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY', 'NLS_DATE_LANGUAGE=american'),
             'AC_ACCOUNT', 11000, NULL, 100, 30) ;

1 row created.
```

☞ 테이블의 DATE-데이터유형의 컬럼에 직접 날짜 형식의 값을 입력하는 경우에는 반드시 TO_DATE() 함수를 이용하여, 날짜형식의 값을 처리한 후, 함수처리 결과로 반환된 값을 이용하여 한 행을 입력해야 합니다.

[주의] 테이블의 DATE-데이터유형의 컬럼에 TO_DATE() 함수를 이용하지 않고, 직접 날짜 형식의 값을 입력하는 경우, 만약 세션의 날짜 표시 형식이 입력되는 날짜의 형식과 다르면, 아래처럼 에러가 발생됩니다.

```
SQL> INSERT INTO hr.employees
     VALUES (303, UPPER('Den'), UPPER('Raphealy'), 'hong@yahoo.co.kr', '515.127.4561',
             'FEB 3 1999', 'AC_ACCOUNT', 11000, NULL, 100, 30);


[SQL*Developer 에서 표시되는 에러 메시지]
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO hr.employees
     VALUES (303, UPPER('Den'), UPPER('Raphealy'), 'hong@yahoo.co.kr', '515.127.4561',
             'FEB 3 1999', 'AC_ACCOUNT', 11000, NULL, 100, 30)
오류 보고 -
SQL 오류: ORA-01841: 년은 영이 아닌 -4713 과 +4713 사이의 값으로 지정해야 합니다.
01841. 00000 -  "(full) year must be between -4713 and +9999, and not be 0"
*Cause:    Illegal year entered
*Action:   Input year in the specified range

[SQL*Plus 에서 표시되는 에러 메시지]
'FEB 3 1999', 'AC_ACCOUNT', 11000, NULL, 100, 30)
*
ERROR at line 3:
ORA-01858: a non-numeric character was found where a numeric was expected
```

**[참고] INSERT-문 처리 시에 에러가 발생되는 경우.**

- NOT NULL 제약-조건이 지정된 컬럼에 [값이 없는 상태]로 입력을 시도하면 에러가 발생됩니다.
- UNIQUE, PRIMARY KEY 제약-조건을 위반하는 중복된 값을 입력하려고 시도하면 에러가 발생됩니다.
- CHECK 제약-조건을 위반하는 값을 입력하려고 시도하면 에러가 발생됩니다.
- FOREIGN KEY 제약-조건을 위반하는 값을 입력하려고 시도하면, 에러가 발생됩니다.
- 컬럼과 데이터 유형이 다른 데이터를 입력하려고 시도하면 에러가 발생됩니다.
- 컬럼에 설정된 최대 길이보다 길이(WIDTH)가 긴 데이터를 입력하려고 시도하면 에러가 발생됩니다..

☞ 제약조건(CONSTRAINT)에 대해서는 제10장에서 자세히 설명합니다.

■ 지금까지 완료된 데이터 입력작업을 취소하고 롤백(UNDO)시킵니다.

```
SQL> ROLLBACK ;

롤백이 완료되었습니다.
```

☞ ROLLBACK 문을 실행시키면, 지금까지 접속한 HR-세션에서 수행했던 모든 데이터 입력 작업들로 구성된 트랜잭션을 행들이 입력되기 전의 상태로 변경시킨 후. HR-세션에서 수행한 트랜잭션을 종료(트랜잭션-롤백)합니다.

## 9-3. 서브쿼리를 이용하여, 다른 테이블의 행들을 복사해서 특정 테이블에 입력하기.

• INSERT-문장에서 [VALUES-절] 대신, 서브쿼리를 사용하여, 테이블에 0개 이상의 행을 입력할 수 있습니다.

[준비실습] 실습을 위하여 사용할 HR.SALES_REPS 실습-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.
```

■ 서브쿼리를 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.
```

☞ 서브쿼리의 SELECT-절에 기술된 컬럼 개수와 순서가 [INSERT INTO-절]의 테이블 이름 다음에 기술된 컬럼 개수 및 순서와 동일해야 합니다.

☞ 서브쿼리의 WHERE-절을 만족하는 행에서 추출된 레코드를 대상-테이블에 입력합니다.

☞ 서브쿼리의 [WHERE-절]을 만족하는 행이 없는 경우에는, [0개 행 이(가) 삽입되었습니다.] 메시지가 표시됩니다.

■ HR.SALES_REPS 테이블에 대하여 서브쿼리를 이용하여 데이터를 입력한 트랜잭션을 커밋합니다.

```
SQL> COMMIT ;

커밋되었습니다.
```

☞ COMMIT 문을 실행시키면, 접속한 HR-세션에서 수행했던 데이터 입력 작업(들)로 구성된 트랜잭션을 행들이 입력된 후의 상태를 유지하면서 종료(트랜잭션-커밋)합니다.

## 9-4. UPDATE-문을 이용하여 행의 데이터를 수정하기.

• UPDATE-문은 [WHERE-절]의 조건을 어떻게 작성하느냐에 따라서 0개 이상의 행을 업데이트 할 수 있습니다.

[준비실습] 실습을 위하여 사용할 HR.COPY_EMP 실습-테이블을 생성하고 필요한 데이터를 입력한 후 커밋합니다.

```
SQL> CREATE TABLE hr.copy_emp
     (id              NUMBER(6)    PRIMARY KEY
     ,name           VARCHAR2(30) NOT NULL
     ,salary         NUMBER(8,2)
     ,hiredate        DATE
     ,jobid          VARCHAR2(10)
     ,department_id  NUMBER(4)
     ) ;

table HR.COPY_EMP이(가) 생성되었습니다.


SQL> INSERT INTO hr.copy_emp(id, name, salary, hiredate, jobid, department_id)
       SELECT employee_id, last_name, salary, hire_date, job_id, department_id
       FROM hr.employees ;

107개 행 이(가) 삽입되었습니다.


SQL> COMMIT ;

커밋되었습니다.
```

■ HR.COPY_EMP 테이블에 대하여 [UPDATE]를 수행하는 예제-1: 특정 행의 일부 컬럼 값을 변경.

```
SQL> UPDATE hr.copy_emp
     SET  salary = 8500
         ,name = INITCAP('SHSHIN')
         ,hiredate = TO_DATE('FEB 3, 1999', 'MON DD, YYYY', 'NLS_DATE_LANGUAGE=AMERICAN')
         ,DEPARTMENT_ID = NULL
     WHERE id = 113;

1개 행 이(가) 업데이트되었습니다.


SQL> COMMIT ;

커밋되었습니다.
```

☞ HR.COPY_EMP 테이블에 저장된 ID 컬럼의 값이 113 인 행에서, SALARY 컬럼의 값을 8500 으로, NAME 컬럼의 값은 INITCAP('SHSHIN") 함수로 처리된 'Shshin' 으로, HIREDATE 컬럼의 값은 'FEB 3, 1999' 날짜 값으로, DEPARTMENT_ID 컬럼은 NULL 상태로 각각 변경됩니다.

☞ UPDATE 문에 의하여 변경되는 컬럼의 데이터유형이 DATE 인 경우, 명시된 날짜 값이 UPDATE-문이 요청된 세션의 날짜 표시 형식과 다를 때, 에러가 발생되는 것을 방지하기 위하여 날짜 값을 TO_DATE() 함수로 처리하십시오.

■ 앞의 UPDATE-문장에 의하여 변경된 데이터를 조회합니다.

```
SQL> SELECT *
     FROM hr.copy_emp
     WHERE id = 113;

        ID NAME                           SALARY HIREDATE JOBID      DEPARTMENT_ID
---------- ------------------------------ ------ -------- ---------- -------------
       113 Shshin                          8500 99/02/03 FI_ACCOUNT
```

■ HR.COPY_EMP 테이블에 대하여 [UPDATE]를 수행하는 예제-2: [WHERE-절]의 누락.

```
SQL> UPDATE hr.copy_emp
     SET department_id = 110;

107개 행 이(가) 업데이트되었습니다.


SQL> ROLLBACK ; -- 수정된 데이터를 수정하기 전의 상태로 되돌리고 트랜잭션을 종료합니다.

롤백이 완료되었습니다.
```

☞ UPDATE-문에서 [WHERE 절]을 누락하면, 모든 행의 해당 컬럼이 모두 동일한 값으로 변경됩니다.

## 9-5. DELETE-문을 이용하여 테이블의 행을 삭제하기.

• DELETE-문은 [WHERE-절]의 조건을 어떻게 작성하느냐에 따라서 0개 이상의 행을 삭제할 수 있습니다.

■ HR.COPY_EMP 테이블에 대하여 행을 삭제(DELETE)하는 예제-1: 특정 조건을 만족하는 행을 삭제.

```
SQL> DELETE FROM hr.copy_emp
     WHERE department_id = 30 ;

6개 행 이(가) 삭제되었습니다.


SQL> COMMIT ;

커밋되었습니다.
```

☞ HR.COPY_EMP 테이블에서 DEPARTMENT_ID 컬럼의 값이 30 인 모든 행이 삭제됩니다.

■ HR.COPY_EMP 테이블에 대하여 행을 삭제(DELETE)하는 예제-2: [WHERE-절]의 누락.

```
SQL> DELETE FROM hr.copy_emp ;

101개 행 이(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 문장을 이용한 테이블 절단.

• TRUNCATE 문은 테이블의 모든 데이터를 제거한다는 점에서는 [WHERE-절 없이 DELETE-문을 실행하고, 커밋을 수행한 것]과 동일합니다.

• TRUNCATE 문은 DML-문이 아니라, DDL(데이터 정의어)-문이므로, 실행과 동시에 자동으로 커밋됩니다.
즉, 한 번 실행되면 DML-문처럼 롤백 또는 언두할 수 없습니다.

• 기본적으로 테이블의 데이터가 저장되었던 저장공간을 테이블이 처음 생성될 때의 상태로 초기화 시킵니다.

■ HR.SAELS_REPS 테이블에 대하여 테이블-절단(TABLE TRUNCATE)을 수행하는 예제

```
SQL> TRUNCATE TABLE hr.sales_reps ;

table HR.SALES_REPS이(가) 잘렸습니다.
```

☞ HR.SALES_REPS 테이블의 구조는 그대로 유지되지만, 저장공간이 처음 생성될 때의 상태로 초기화되어 테이블의 모든 데이터가 제거됩니다.

☞ 다른 테이블에 정의된 FOREIGN KEY 제약조건에 의하여 참조되는 부모-테이블을 절단(TRUNCATE) 할 수 없습니다.

## 9-7. DML-문에서 서브쿼리를 사용하는 예제.

### 9-7-1. UPDATE-문에서의 서브쿼리 사용.

■ HR.COPY_EMP 테이블에 저장된 ID=200 인 행의 JOBID 컬럼과 SALARY 컬럼의 값을 각각 (ID가 206 값인 행의 JOBID 컬럼의 값으로), 그리고 (ID가 205 값인 행의 SALARY 컬럼의 값으로) 변경하시오.

```
SQL> UPDATE hr.copy_emp
     SET jobid = (SELECT jobid
                  FROM hr.copy_emp
                  WHERE id = 206)
        ,salary = (SELECT salary
                   FROM hr.copy_emp
                   WHERE id = 205)
     WHERE id = 200 ;

1개 행 이(가) 업데이트되었습니다.


SQL> COMMIT ;

커밋되었습니다.


SQL> SELECT * FROM hr.copy_emp WHERE id=200 ;

        ID NAME                           SALARY HIREDATE JOBID      DEPARTMENT_ID
---------- ------------------------------ ------ -------- ---------- -------------
       200 Whalen                          12008 03/09/17 AC_ACCOUNT            10
```

☞ UPDATE-문장의 [SET-절]에서 서브쿼리를 이용하여 컬럼에 대하여 변경하는 값을 지정할 수 있습니다. 단 이 때 사용되는 서브쿼리는 반드시 하나의 행에서 값을 반환하는 단일-행 서브쿼리 이어야만 합니다.

■ 다음의 요구사항 대로 HR.COPY_EMP 테이블의 행(들)의 일부 데이터를 수정하시오.

[요구사항]

HR.EMPLOYEES 테이블의 사번(EMPLOYEE_ID 컬럼의 값)이 201 인 행의 업무코드(JOB_ID 컬럼의 값)와 동일한 업무코드(COPY_EMP.JOBID 컬럼의 값)를 가지는 HR.COPY_EMP 테이블의 행(들)의 근무부서(DEPARTMENT_ID 컬럼의 값) 값을, [HR.EMPLOYEES 테이블의 사번(EMPLOYEE_ID 컬럼의 값)이 100 인 행의 근무부서(DEPARTMENT_ID 컬럼의 값)으로 변경하시오.

```
SQL> UPDATE hr.copy_emp
     SET department_id = (SELECT department_id
                          FROM hr.employees
                          WHERE employee_id = 100)
     WHERE jobid = (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 jobid = (SELECT job_id
                    FROM hr.employees
                    WHERE employee_id = 201);

        ID NAME                           SALARY HIREDATE JOBID      DEPARTMENT_ID
---------- ------------------------------ ------ -------- ------ -------------
       201 Hartstein                       13000 04/02/17 MK_MAN             90
```

☞ DML-문에 서브쿼리를 이용하여 다른 테이블의 데이터를 이용하여 특정 테이블의 데이터를 수정할 수 있습니다.

### 9-7-2. DELETE-문에서의 서브쿼리 사용.

■ 다음의 요구사항 대로 HR.COPY_EMP 테이블의 행(들)을 삭제하시오.

[요구사항]
HR.DEPARTMENTS 테이블의 부서이름(DEPARTMENT_NAME 컬럼의 값)에 'Administration'이 포함된 행(들)의 부서코드(DEPARTMENT_ID 컬럼의 값)와 동일한 부서코드(COPY_EMP. DEPARTMENT_ID) 값을 가지는 HR.COPY_EMP 테이블의 행(들)을 삭제하시오.

```
SQL> DELETE FROM hr.copy_emp
     WHERE department_id = (SELECT department_id
                            FROM hr.departments
                            WHERE department_name LIKE '%Administration%');

1개 행 이(가) 삭제되었습니다.

SQL> COMMIT ;

커밋되었습니다.
```

## 9-8. DML 문장에서 WITH CHECK OPTION이 포함된 서브쿼리 사용하기(오라클에서만 됩니다).

- INSERT INTO ... VALUES () 문장에서 [WITH CHECK OPTION을 가진 WHERE 절이 포함된 서브쿼리]를 사용하여 사용자가 입력하는 데이터에 대한 조건 검사를 구현할 수 있습니다.

- 예를 들어 아래와 같은 INSERT-문으로 HR.COPY_EMP 테이블에 한 행을 입력하려고 합니다.

```
SQL> INSERT INTO hr.copy_emp(id, name, salary, hiredate, jobid, department_id)
     VALUES (99999,'Taylor',5000, TO_DATE('28-06-99','DD-MM-RR'),'ST_CLERK', 70) ;
```

위의 INSERT-문으로 한 행을 입력할 때, DEPARTMENT_ID 컬럼의 값이 30 보다 작지 않으면, 입력이 되지 못하도록 하고 싶습니다. 이를 위하여, 아래처럼, [WITH CHECK OPTION이 명시된 [WHERE 절]을 포함하는 서브쿼리를 [INSERT INTO-절]에서 테이블이름 대신 사용합니다.

■ [WITH CHECK OPTION]이 명시된 서브쿼리를 사용한 INSET INTO 문장 예제1

```
SQL> INSERT INTO (SELECT id, name, salary, hiredate, jobid, department_id
                  FROM hr.copy_emp
                  WHERE department_id < 30 WITH CHECK OPTION )
     VALUES (99998,'Taylor',5000, TO_DATE('28-06-99','DD-MM-RR'),'ST_CLERK', 10) ;

1개 행 이(가) 삽입되었습니다.
```

☞ DEPARTMENT_ID 컬럼에 지정된 값이 WITH CHECK OPTION 이 명시된 WHERE 절의 조건을 만족(30 보다 작음)하므로, 위의 입력 작업은 정상적으로 처리됩니다.

■ [WITH CHECK OPTION]이 명시된 서브쿼리를 사용한 INSET INTO 문장 예제2

```
SQL> INSERT INTO (SELECT *
                  FROM hr.copy_emp
                  WHERE department_id < 30 WITH CHECK OPTION )
     VALUES (99997,'Taylor',5000, TO_DATE('28-06-99','DD-MM-RR'),'ST_CLERK', 50) ;
명령의 1 행에서 시작하는 중 오류 발생 -
INSERT INTO (SELECT *
                  FROM hr.copy_emp
                  WHERE department_id < 30 WITH CHECK OPTION )
     VALUES (99997,'Taylor',5000, TO_DATE('28-06-99','DD-MM-RR'),'ST_CLERK', 50)
오류 보고 -
SQL 오류: ORA-01402: 뷰의 WITH CHECK OPTION의 조건에 위배 됩니다
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
*Cause:
*Action:
```

☞ DEPARTMENT_ID 컬럼에 지정된 값이, WITH CHECK OPTION 이 명시된 WHERE 절의 조건을 만족하지 않음으로 에러가 발생되고, 행이 입력되지 않습니다.

## 9-9. 트랜잭션(TRANSACTION) 개요.

• 트랜잭션을 제어하는 Transaction Control Language(TCL) 문장인, COMMIT-문과 ROLLBACK-문에 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 DML-문들 중 하나가 실행되지 못하면 트랜잭션을 구성하는 다른 DML-문들도 모두 롤백(또는 UNDO)되어야 합니다

### ■ 트랜잭션을 정상적으로 종료하기 위하여 사용되는 SQL-문들.

| | |
|---|---|
| COMMIT-문 | DML-문장 완료 후, 변경 후의 데이터 상태를 유지하면서 트랜잭션을 정상적으로 종료합니다. |
| ROLLBACK-문 | DML-문장 완료 후, 변경 전의 데이터 상태로 되돌려 놓고 트랜잭션을 정상적으로 종료합니다. |

■ 트랜잭션이 종료되는 방법.

- 세션 사용자가 수행한 DML-문(들)로 구성된 트랜잭션은, 세션의 사용자가 직접 COMMIT-문 또는 ROLLBACK-문을 실행하여 세션의 트랜잭션을 명시적으로 종료해야 합니다.

- 다음과 같은 경우에 서버에 의하여 자동으로 커밋이 수행됩니다.
  - 세션 사용자가 하나의 DDL-문 또는 하나의 DCL문을 실행시킨 경우, 오라클-서버가 자동으로 커밋을 수행합니다.
  - 세션 사용자가 DML-문(들)을 수행한 후, COMMIT-문 또는 ROLLBACK-문을 명시적으로 실행하지 않은 채, 오라클-서버에의 접속을 정상적으로 종료한 경우, 해당 DML-문(들)으로 구성된 트랜잭션은 자동으로 커밋됩니다.

- 다음과 같은 경우에 서버에 의하여 자동으로 롤백이 수행됩니다.
  - 세션 사용자가 DML-문(들)을 수행한 후, COMMIT-문 또는 ROLLBACK-문을 명시적으로 실행하지 않은 채, 오라클-서버에의 접속이 비정상적으로 종료된 경우, 해당 DML-문(들)으로 구성된 트랜잭션은 자동으로 롤백됩니다.

■ 오라클 데이터베이스 서버에서의 트랜잭션 특징.

- 오라클-서버에서는 트랜잭션을 시작하는 별도의 명령어가 없으며, 접속한 세션에서 DML-문 하나가 처음 실행되면, 오라클-서버는 자동으로 해당 세션의 트랜잭션을 관리하기 시작합니다.
- 특정 세션에서 수행되는 하나의 트랜잭션은, 1 개 이상의 DML-문(들) 또는 하나의 DDL 또는 하나의 DCL 문장으로 구성되며, SELECT-문은 트랜잭션과 관계가 없습니다.

■ 오라클 데이터베이스 서버에서 진행 중인 트랜잭션(DML 작업이 완료되었고, 커밋/롤백이 되기 전의 상태) 특징.

- 필요한 경우, ROLLBACK-문을 실행하여 이전의 데이터 상태를 복구할 수 있습니다.
- DML을 수행한 현재 세션의 유저는, SELECT-문을 사용하여 DML-문의 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.

■ COMMIT-문 실행 완료 후의 데이터 상태

- 데이터에 대한 변경-후의 사항이 데이터베이스에 적용되며, 모든 유저가 변경-후의 결과를 확인할 수 있습니다.
- FLASHBACK-기술이 아닌 일반적인 SELECT-문을 이용하여 변경-전의 데이터를 사용할 수 없습니다.
- DML-문의 대상 행에 대한 잠금이 해제되고, 다른 유저가 해당 행에 대하여 새로운 변경 작업을 수행할 수 있습니다.
- 모든 저장점(Savepoint)가 지워집니다.

■ ROLLBACK-문 실행 완료 후의 데이터 상태

- DML-문에 의하여 수행된 데이터의 변경 사항이 실행 취소되고, 변경되기 전의 데이터 상태가 복원됩니다.
- 영향 받는 행의 잠금이 해제됩니다.

## 9-10. 트랜잭션 이해를 위한 실습.

1> 실습을 위하여 HR.EMPS8 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.emps8
     (id      VARCHAR2(6),
     salary NUMBER(8,2),
     name    VARCHAR2(30)) ;

table HR.EMPS8이(가) 생성되었습니다.
```

2> 서브쿼리를 이용한 INSERT를 수행하여, 세션에 대한 트랜잭션을 시작시킵니다.

| | |
|---|---|
| SQL> **INSERT INTO hr.emps8(id, salary, name)**<br>**SELECT employee_id, salary, last_name**<br>**FROM hr.employees**<br>**WHERE department_id = 90 ;**<br><br>3개 행 이(가) 삽입되었습니다. | [참고] 세션의 상태는 다음과 같습니다.<br>• DML 작업(서브쿼리를 이용한 INSERT 작업)은 완료되었습니다.<br>• COMMIT-문 또는 ROLLBACK-문을 실행시키지 않았으므로, 트랜잭션은 진행 중입니다.<br>• 현재까지 트랜잭션은 1 개의 INSERT-문으로 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 서브쿼리를 이용한 INSERT를 한번 더 수행합니다.

| | |
|---|---|
| SQL> **INSERT INTO hr.emps8(id, salary, name)**<br>**SELECT employee_id, salary, last_name**<br>**FROM hr.employees**<br>**WHERE department_id = 60 ;**<br><br>5개 행 이(가) 삽입되었습니다. | [참고] 세션의 상태는 다음과 같습니다.<br>• DML 작업들(2 개의 서브쿼리를 이용한 INSERT 작업)은 완료되었습니다.<br>• COMMIT-문 또는 ROLLBACK-문을 실행시키지 않았으므로, 트랜잭션은 진행 중입니다.<br>• 이 트랜잭션은 2 개의 INSERT-문으로 구성됩니다. |

☞ 트랜잭션이 진행되는 동안 세션에 대하여 변경되는 행은 잠김 상태가 됩니다(ROW-LOCK). 이러한 LOCK은 DML이 끝날 때가 아니라 트랜잭션이 끝날 때 해제됩니다.

5> COMMIT을 실행하여 트랜잭션을 종료합니다..

```
SQL> COMMIT ;

커밋되었습니다.
```

☞ 이 세션에서 2 개의 INSERT-문들로 구성된 트랜잭션이 변경 후의 데이터 상태를 유지하면서 종료됩니다.
일단 커밋된 트랜잭션의 작업을 다시 롤백할 수는 없습니다.

☞ COMMIT-문 실행 전에 생성한 모든 저장점(Savepoint)들은 삭제됩니다.

**[참고] 하나의 트랜잭션 롤백**

앞의 트랜잭션(전체 DML작업들)이 잘못된 경우, ROLLBACK-문을 실행하면, [전체 DML 작업들이 수행되기 이전 상태로] 데이터의 상태가 변경됩니다. 즉, 하나의 트랜잭션을 구성하는 모든 DML작업들이 ROLLBACK됩니다.

```
SQL> ROLLBACK ;

롤백이 완료되었습니다.
```

☞ 이 세션에서 2 개의 INSERT-문들로 구성된 트랜잭션이 변경 전의 데이터 상태로 언두(UNDO)되면서 종료됩니다.
일단 롤백된 트랜잭션의 작업을 다시 자동으로 생성시킬 수는 없습니다.

☞ ROLLBACK-문 실행 전에 생성한 모든 저장점(Savepoint)들은 삭제됩니다.

**[참고] 특정 저장점(Savepoint)까지 트랜잭션의 일부를 롤백.**

만약, 위의 실습에서 첫 번째 서브쿼리를 이용한 INSERT 후에 INSERT1 이라는 이름으로 저장점을 생성한 경우, 다음의 구문을 이용하여, INSERT1 저장점 이 후에 수행한 두 번째 서브쿼리를 이용한 INSERT 작업만 변경 작업 이전의 상태로 롤백 할 수 있습니다.

```
SQL> ROLLBACK TO SAVEPOINT insert1 ;

Rollback complete.
```

☞ 트랜잭션이 종료된 것은 아닙니다. 트랜잭션을 구성하는 DML-문들 중 저장점(Savepoint) 이 후부터의 마지막까지의 DML-문의 작업들이 롤백됩니다.

## 9-11. LOCK의 기능을 이해하기 위한 실습.

• 2개의 터미널에서 HR 계정으로 접속한 후, 번호 순서에 맞춰서 다음의 DML-문들을 실행하며 실습을 수행합니다.

■ HR-접속 세션들에서 다음의 UPDATE-문을 각각 실행합니다.

| 터미널1: HR-접속 세션1 | 터미널2: HR-접속 세션2 |
|---|---|
| 1> department_id 가 270 인 행에 UPDATE 수행.<br><br>SQL> **UPDATE hr.departments**<br>**SET department_name= 'Oracle'**<br>**WHERE department_id = 270 ;**<br><br>1 row updated | 2> department_id 가 260 인 행에 UPDATE 수행.<br><br>SQL> **UPDATE hr.departments**<br>**SET department_name= 'DB2'**<br>**WHERE department_id = 260 ;**<br><br>1 row updated. |

☞ 각 세션에서 서로 다른 행에 UPDATE를 수행했으며, 각 세션에서 변경되는 행에 각각 LOCK이 걸려 있습니다.

☞ 두 세션 모두 UPDATE-문 실행은 완료했지만, 트랜잭션을 종료시키지는 않았습니다.

■ [터미널2: HR-접속 세션2]에서만 다음의 UPDATE-문을 실행합니다.

| 터미널1: HR-접속 세션1 | 터미널2: HR-접속 세션2 |
|---|---|
| | 3> department_id 가 270 인 행에 UPDATE 수행.<br>SQL> **UPDATE hr.departments**<br>**SET department_name= 'MS-SQL'**<br>**WHERE department_id = 270 ;**<br>_← 커서가 깜박이는 상태(대기 상태)가 지속됩니다. |

☞ 세션1에서 수행한 UPDATE 문에 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.

■ [터미널1: HR-접속 세션1]에서만 다음의 COMMIT-문을 실행합니다.

| 터미널1: HR 세션1 | 터미널2: HR 세션2 |
|---|---|
| 4> COMMIT을 수행하여 TRANSACTION을 끝냄.<br><br>SQL> **COMMIT ;**<br><br>Commit complete. | 5> 자동으로 "1 row updated "메세지가 표시됩니다.<br><br>6> 수행된 트랜잭션을 롤백시킵니다.<br><br>SQL> **ROLLBACK ;**<br><br>Rollback complete. |

☞ 세션1에서 COMMIT을 수행하여 트랜잭션을 끝내면, 해당 행에 걸려 있던 LOCK이 해제되고, 이 행에 대하여 DML-문을 시도한 세션2로 LOCK이 전달되어 대기 상태에 있던 세션2의 UPDATE-문이 비로서 실행 완료됩니다.

☞ 이렇게 LOCK 을 이용하여, 같은 행에 대하여 서로 다른 세션들이 동시에 DML이 수행되는 것을 방지할 수 있습니다.

[참고] 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.
```

## 9-12 SELECT-문에^1 [FOR UPDATE-절] ^사용.

- 오라클-서버에서는 일반적인 SELECT-문에 의하여 액세스되는 행에는 LOCK이 걸리지 않습니다. 하지만 상황에 따라서는 SELECT-문에 의하여 액세스되는 행에 LOCK을 걸어야 할 경우도 필요합니다.

- SELECT-문에 [FOR UPDATE-절]을 이용하면 SELECT-문으로 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 COMMIT-문 또는 ROLLBACK-문을 수행하기 전까지 다른 세션의 사용자는 행(들)을 변경할 수 없습니다.

■ SELECT-문에 [FOR UPDATE-절]을 사용한 예제1

```
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 COMMIT-문 또는 ROLLBACK-문을 실행하는 경우에만 위의 행들에 걸린 LOCK이 해제됩니다.

☞ 위의 행들(LOCK이 걸린 행들)에 대하여 다른 세션에서 [SELECT..FOR UPATE]문 또는 [DML-문]으로 잠그려고 하면, 데이터베이스는 해당 행의 LOCK이 해제될 때까지 기다린 다음, 다른 세션의 [SELECT..FOR UPATE]문 또는 [DML-문] 작업들을 수행하게 됩니다.

■ SELECT-문에 [FOR UPDATE-절]을 사용한 예제2: JOIN이 사용된 SELECT문에 [FOR UPDATE-절] 사용

```
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
     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 액세스하는 행이 잠깁니다.

■ SELECT-문에 [FOR UPDATE-절]을 사용한 예제2: JOIN이 사용된 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.

■ SELECT-문에 [FOR UPDATE-절]을 사용한 예제3: 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.
```

# 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(TABLE) 및 제약조건(CONSTRAINT) 개요.

• 데이터베이스에서 테이블과 제약조건을 사용하는 목적은 다음과 같습니다.

| 객체종류 | 용도 |
|---|---|
| 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. |

## 10-2. 데이터베이스에 접속한 계정이 테이블을 생성하기 위한 조건.

• 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.

## 10-3. 기본적인 테이블 생성 문법.

• 테이블을 생성하기 위한 최소한의 문장은 다음의 문법에 맞도록 작성되어야 합니다.

```
CREATE TABLE 스키마이름.테이블이름
( 컬럼이름1   데이터유형(최대길이)
, 컬럼이름2   데이터유형          -- 일부 데이터유형은 데이터의 최대 길이를 지정하지 않기도 합니다.
[, ...]
 ) ;
```

☞ '**스키마이름.**'을 생략하면, CREATE TABLE 문장을 실행하는 접속 계정의 스키마 객체로 테이블이 생성됩니다.

☞ CREATE TABLE 문장으로 테이블 생성 시에 최소한 테이블 이름, 컬럼 이름, 컬럼의 데이터유형 및 허용된 데이터의 최대 길이는 반드시 명시해야 합니다.

## 10-4. 테이블 생성 시에, 테이블의 이름 및 컬럼의 이름을 기술할 때 지켜야 되는 규칙.

• 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 에러가 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:
```

## 10-5. 테이블 생성 실습-1: DEFAULT 옵션을 사용한 테이블 생성.

■ 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.
```

### [참고] DEFAULT 옵션 사용 시에 고려할 사항과 DEFAULT 옵션의 기능.

- 컬럼의 데이터 유형과 일치한 상수(Literal) 값, 표현식, SQL 함수들을 컬럼의 DEFAULT 값으로 설정할 수 있습니다.
  단, 다른 컬럼의 이름이나 ROWNUM, NEXTVAL, CURRVAL 같은 PSEUDO-컬럼이 DEFAULT 값으로 설정될 수 없습니다.

- 필요한 경우, 테이블의 컬럼 하나 하나에 대하여 DEFAULT 옵션을 이용하여 디폴트로 사용될 값을 각각 정의합니다.

- 테이블에 한 행을 입력할 때, DEFAULT 옵션이 설정된 컬럼에 대하여, 컬럼에 입력되는 값이 있으면 그 값이 입력되고, 입력 값이 없으면 DEFAULT 옵션으로 설정된 값이 자동으로 입력됩니다.

- UPDATE문과 INSERT문에서 DEFAULT 키워드를 이용하여 데이터입력 및 수정을 할 수 있습니다.

### [참고] 테이블의 컬럼에 설정된 DEFAULT 옵션을 이용한 데이터입력.

1> HR.CUST 테이블에 데이터 입력-1.

```
SQL> INSERT INTO hr.cust
     VALUES (10, 'SHSHIN', NULL, TO_DATE('20100321','YYYYMMDD')) ;

1개 행 이(가) 삽입되었습니다.
```

☞ 사용자가 직접 값을 지정하거나 NULL 키워드를 지정하면, DEFAULT 옵션과 상관없이 행이 입력됩니다.

2> HR.CUST 테이블에 데이터 입력-2

```
SQL> INSERT INTO hr.cust(cid)
     VALUES (20) ;

1개 행 이(가) 삽입되었습니다.
```

☞ DEFAULT 옵션이 지정된 컬럼에 대하여 입력된 데이터도 없고, NULL로 지정하지 않은 경우 DEFAULT로 지정된 값이 입력됩니다. DEFAULT 옵션이 지정되지 않은 컬럼은 NULL 상태로 입력됩니다.

3> HR.CUST 테이블에 데이터 입력-3 및 트랜잭션 커밋.

```
SQL> INSERT INTO hr.cust
     VALUES (30, DEFAULT, DEFAULT, DEFAULT) ;

1개 행 이(가) 삽입되었습니다.

SQL> COMMIT ;

커밋되었습니다.
```

☞ DEFAULT 키워드를 이용하여, 디폴트 값을 직접 호출할 수 있습니다. 이 때, DEFAULT로 설정된 값이 없으면(예, name 컬럼) NULL 상태로 입력되고, DEFAULT 옵션으로 설정된 값이 있으면, 그 값이 대신 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
```

**[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 SQL-문에서 명시해야 합니다.

• 다음은 테이블이름에 스키마이름을 명시할 때와 명시하지 않을 때의 의미 차이를 설명한 것입니다.

| SELECT-문 | 의미 |
|---|---|
| SELECT *<br>FROM EMPLOYEES ; | 접속한 계정의 스키마에 있는 EMPLOYEES 테이블의 데이터를 조회합니다. |
| SELECT *<br>FROM HR.EMPLOYEES ; | HR의 스키마에 있는 HR.EMPLOYEES 테이블의 데이터를 조회됩니다. |

**[참고] 스키마(SCHEMA)란 ?**

소유권(OWNERSHIP)이 동일한 객체들의 묶음(GROUP)을 의미합니다.

## 10-6. 오라클 데이터베이스의 대표적인 데이터 유형(DATA-TYPE)들.

### ◈ VARCHAR2 및 CHAR 데이터 유형.

- VARCHAR2 및 CHAR 데이터 유형은 오라클 데이터베이스에서 문자 데이터를 처리하는 데이터 유형입니다.

- 최대 길이를 명시할 때 [BYTE] 또는 [CHAR]을 옵션을 명시하여 처리 단위를 각각 바이트 또는 문자수로 지정할 수 있으며, [BYTE] 또는 [CHAR]를 명시하지 않으면, 기본적으로 [BYTE] 단위로 데이터의 최대 길이가 처리됩니다.

| VARCHAR2 ( size [BYTE\|CHAR] ) | CHAR [ ( size [BYTE\|CHAR] ) ] |
|---|---|
| • 최대 4000 BYTE 길이까지만 처리 가능합니다.<br>• 반드시 최대 허용 길이를 명시해야 합니다.<br>• 입력되는 데이터의 실제 길이만큼 저장 공간을 사용하는 **가변-길이 문자 데이터 유형**입니다.<br>• 만약 VARCHAR2(30)로 지정된 컬럼에 'ORACLE' 값을 입력하면, 6 BYTE 공간을 사용하여 데이터가 저장됩니다.<br>6 \| ORACLE | • 최대 2000 BYTE 길이까지만 처리 가능합니다.<br>• 최대 길이를 명시하지 않으면, 1 BYTE가 지정됩니다.<br>• 지정된 최대 길이를 무조건 저장 공간으로 사용하는 **고정-길이 문자 데이터 유형**으로, 실제 데이터가 차지하고 남은 공간은 빈칸으로 채워서 저장됩니다.<br>• 만약 CHAR(30)로 지정된 컬럼에 'ORACLE' 값을 입력하면, 무조건 30 BYTE 공간을 사용하며, 사용 안 된 빈 공간을 빈칸으로 채우면서 데이터가 저장됩니다.<br>30 \| ORACLE<br>• CHAR 데이터 유형은 길이가 일정한 문자 데이터에만 사용하는 것을 권장합니다. |

**[참고] 테이블의 행이 블록에 저장될 때의 행-구조(Row-Structure)**

INSERT-문으로 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) | |
|---|---|---|---|---|---|---|
| | 2 | 10 | 6 | ORACLE | 6 | SHSHIN |
| 행 해더 | 저장 길이 | 값 | 저장 길이 | 값 | 저장 길이 | 값 |

## ◈ DATE 데이터 유형.

- 날짜 데이터를 처리하는 데이터 유형으로, 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 31일까지 입니다.

- DATE 데이터 유형이 세션(CLIENT)에서 기본적으로 표시되는 형식은 사용자가 NLS_DATE_FORMAT 세션 설정에 의하여 명시적으로 정해 주거나 NLS_TERRITORY 세션 설정에 의하여 암시적으로 정해 집니다.

- 위의 세션 설정을 명시적으로 선언해서 설정하지 않은 경우에는 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라, 자동으로 설정됩니다.

## ◈ NUMBER [ ( RRECISION [, SCALE] ) ] 데이터 유형.

- NUMBER 데이터유형은 0을 포함하여, $1.0 \times 10^{-130}$ 부터 $1.0 \times 10^{126}$($1.0 \times 10^{126}$은 포함되지 않음)까지 절대 값 (ABSOLUTE VALUES)을 가지는 양(POSITIVE, 양수) 및 음(NEGATIVE, 음수)의 FIXED NUMBER를 저장합니다.

- NUMBER 데이터 유형의 산술 표현식(Expression)의 절대값(ABSOLUTE VALUE)이 $1.0 \times 10^{126}$ 보다 크거나 같으면, 에러가 반환됩니다.

- 각 NUMBER 형식 값은 1 부터 22 BYTES를 필요로 합니다.

- 저장되는 NUMBER는 2 자리당 1 BYTE 저장 공간을 사용합니다.

- NUMBER 데이터-형식 사용 예제 및 설명

| | |
|---|---|
| NUMBER(5, 2) | 전체 자리 수(PRECISION)가 5자리이고 소수점 이하 자리 수(SCALE)가 2인 실수를 의미하며, **정수부가 최대 3자리까지**라는 의미가 내포되어 있습니다. |

- NUMBER(5, 2) 데이터 유형에 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.

◈ **TIMESTAMP 계열 데이터 유형.**

- 오라클 9i 버전부터 TIMESTAMP, TIMESTAMP WITH TIME ZONE, 또는 TIMESTAMP WITH LOCAL TIME ZONE 데이터 유형을 이용하여 DATETIME을 처리할 수 있습니다.

- TIMESTAMP WITH TIME ZONE 및 TIMESTAMP WITH LOCAL TIME ZONE 데이터 유형을 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 데이터 유형들에 대한 기본적인 특징.**

| | |
|---|---|
| TIMESTAMP [ (fractional_seconds) ] | • [세기/년/월/일/시/분/초/밀리초]형식으로 구성되어 날짜시간 데이터를 처리합니다. 단, 시간대는 포함되지 않습니다.<br>• 이 데이터유형이 세션에서 표시되는 형식은 사용자가 NLS_TIMESTAMP_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 자동으로 정해집니다.<br>• 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다. |

| | |
|---|---|
| TIMESTAMP [ (fractional_seconds) ] WITH TIME ZONE | • TIMESTAMP 데이터유형의 값에 시간대가 추가된 데이터 유형입니다.<br>• 이 데이터유형이 세션에서 표시되는 형식은 사용자가 NLS_TIMESTAMP_TZ_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 자동으로 정해 집니다.<br>• 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다.<br>• 이 데이터유형에서의 시간대는 접속한 세션(CLIENT)의 시간대가 고려됩니다. |

| | |
|---|---|
| TIMESTAMP [ (fractional_seconds) ] WITH LOCAL TIME ZONE | • 데이터베이스 서버의 시간대를 기준으로 서버와 CLIENT 세션의 시간대와의 시차가 계산되어 반영된 TIMESTAMP 데이터 유형입니다.<br>• 이 데이터유형이 세션에서 기본적으로 표시되는 형식은 사용자가 NLS_TIMESTAMP_FORMAT 설정으로 명시적으로 정해 주거나 NLS_TERRITORY 설정에 의하여 암시적으로 정해 집니다.<br>• 이 데이터유형에 대한 표시형식을 세션에서 설정하지 않은 경우에는, 클라이언트의 프로그램이 실행되는 운영체제의 언어 및 지역에 따라 자동으로 설정됩니다.<br>• 이 데이터유형에서의 시간대는 접속한 세션(CLIENT)의 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 데이터 유형에 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. |

[참고] SYSTIMESTAMP 함수, LOCALTIMESTAMP 함수와 CURRENT_TIMESTAMP 함수.

| | |
|---|---|
| SYSTIMESTAMP | 데이터베이스 서버가 운영 중인 시스템의 날짜와 시간을 TIMESTAMP WITH TIME ZONE 데이터 유형으로 반환합니다. |
| CURRENT_TIMESTAMP | 세션(CLIENT)의 시간대에서 현재 날짜와 시간을 TIMESTAMP WITH TIME ZONE 데이터 유형으로 반환합니다. |
| LOCALTIMESTAMP [(밀리초_자리수)] | 세션(CLIENT)의 시간대에서 현재 날짜와 시간을 TIMESTAMP 데이터 유형으로 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.

## ◈ INTERVAL 계열 데이터 유형.

• 9i 버전부터 기간(INTERVAL)을 처리하는 INTERVAL YEAR TO MONTH 및 INTERVAL DAY TO SECOND 데이터 유형이 새로 추가되었습니다.

• **데이터 유형에 대한 기본적인 특징**

| | |
|---|---|
| INTERVAL YEAR [ (연도 최대-자릿수) ]<br>TO MONTH | • 기간 데이터를 [n 년]-[n 개월] 형식으로 처리합니다.<br>• [연도 최대-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 처리되는 연도의 최대 자릿수를 지정합니다.<br>• [연도 최대-자릿수]를 명시하지 않으면, 2 로 지정됩니다. |

| | |
|---|---|
| INTERVAL DAY [ (일 최대-자릿수) ]<br>TO SECOND[(밀리초-자릿수)] | • 기간 데이터를 [n일]-[n시간]-[n분]-[n초] 형식으로 처리합니다.<br>• [일 최대-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 처리되는 일의 최대 자릿수를 지정합니다.<br>• [일 최대-자릿수]를 명시하지 않으면, 2로 지정됩니다.<br>• [밀리초-자릿수]에는 0부터 9까지 범위의 숫자를 명시하여 표시하고 싶은 밀리초의 자릿수를 지정합니다.<br>• [밀리초 자릿수]를 명시하지 않으면, 6으로 지정됩니다. |

## [참고] INTERVAL 계열 데이터 유형의 컬럼에 데이터 입력 및 조회 실습.

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 altered.
```

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 created.
```

☞ 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 데이터 유형 컬럼에 데이터 입력 실습: 데이터 입력 시 주의하시기 바랍니다.

```
SQL> INSERT INTO hr.test_interval
     VALUES (1, '999-11', '999 23:59:59.99') ;

1 row created.


SQL> INSERT INTO hr.test_interval
     VALUES  (2, INTERVAL '999-11' YEAR(3) TO MONTH
               , INTERVAL '999 23:59:59.99' DAY(3) TO SECOND(2)) ;

1 row created.


SQL> INSERT INTO hr.test_interval
     VALUES (3, INTERVAL '100' YEAR(3), INTERVAL '24' DAY) ;

1 row created.


SQL> INSERT INTO hr.test_interval
     VALUES (4, INTERVAL '30' MONTH, INTERVAL '240' MINUTE) ;

1 row created.


SQL> INSERT INTO hr.test_interval
     VALUES (5, INTERVAL '10' YEAR, INTERVAL '240:30' HOUR TO MINUTE) ;

1 row created.


SQL> INSERT INTO hr.test_interval
     VALUES (6, INTERVAL '24' MONTH, INTERVAL '5 23:59' DAY TO MINUTE);

1 row created.
```

[주의] 데이터 입력 시에 YEAR 다음에 자리 수 (3)을 누락하면, 기본적으로 최대 2자리까지만 허용되므로
100 년은 3 자리이기 때문에 에러가 발생됩니다.

```
SQL> INSERT INTO hr.test_interval VALUES (8, INTERVAL '100' YEAR, INTERVAL '240' DAY(3)) ;
INSERT INTO hr.test_interval VALUES (8, INTERVAL '100' YEAR, NULL)
                                                             *
ERROR at line 1:
ORA-01873: the leading precision of the interval is too small
```

4> 입력된 INTERVAL 형식의 데이터 확인

```
SQL> SELECT T_ID, localtimestamp(0) AS "LOCALTIMESTAMP"
           ,IYTM, localtimestamp(0) + IYTM "+IYTM"
           ,IDTS, localtimestamp(0) + IDTS "+IDTS"
     FROM hr.test_interval ;

     T _ID LOCALTIMESTAMP       IYTM     +IYTM                IDTS               +IDTS
---------- -------------------- -------- -------------------- ------------------ --------------------
         1 2013/07/29 18:02:52  +999-11  3013/06/29 18:02:52  +999 23:59:59.99   2016/04/24 18:02:51
         2 2013/07/29 18:02:52  +999-11  3013/06/29 18:02:52  +999 23:59:59.99   2016/04/24 18:02:51
         3 2013/07/29 18:02:52  +100-00  2113/07/29 18:02:52  +024 00:00:00.00   2013/08/22 18:02:52
         4 2013/07/29 18:02:52  +002-06  2016/01/29 18:02:52  +000 04:00:00.00   2013/07/29 22:02:52
         5 2013/07/29 18:02:52  +010-00  2023/07/29 18:02:52  +010 00:30:00.00   2013/08/08 18:32:52
         6 2013/07/29 18:02:52  +002-00  2015/07/29 18:02:52  +005 23:59:00.00   2013/08/04 18:01:52

7 rows selected.
```

☞ INTERVAL 형식 데이터가 특정 날짜에 산술 연산으로 처리된 날짜가 표시됩니다.

5> 실습 정리: 실습에 사용한 HR.TEST_INTERVAL 테이블을 삭제합니다.

```
SQL> DROP TABLE hr.test_interval PURGE ;

Table dropped.
```

## 10-7. 제약조건 (CONSTRAINT) 개요.

• 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 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. |
| 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. |

## 10-7-1. NOT NULL 제약조건.

• 한 행 입력 시에 NOT NULL 제약조건이 지정된 컬럼에는 반드시 값이 입력되어야 합니다.

| 컬럼1 | 컬럼2 | 한 행 입력 |
|---|---|---|
| A | 10 | ← INSERT(1) |
| B | | ← INSERT(2) |

예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼2에 NOT NULL 제약조건을 정의했다면,

• INSERT(1)은 컬럼 2에 데이터가 있으므로 행이 정상적으로 입력됩니다.

• INSERT(2)는 컬럼 2에 데이터가 없는 NULL 상태 이므로 행이 입력되지 않고 에러가 발생됩니다.

## 10-7-2. 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.

| 컬럼1 | 컬럼2 | 한 행 입력 |
|---|---|---|
| A | 10 | ← INSERT(1) |
| B | 20 | ← INSERT(2) |
| C | 10 | ← INSERT(3) |
| D | | ← INSERT(4) |
| E | | ← INSERT(5) |

예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼2에 UNIQUE 제약조건을 정의했다면,

- INSERT(1,2)은 기존의 행에 입력된 컬럼2의 값과 중복되지 않으므로 행이 입력됩니다.

- INSERT(3)는 기존의 행에 입력된 컬럼2의 값과 중복되므로 행이 입력되지 않고 에러가 발생됩니다.

- INSERT(4, 5)는 입력되는 행의 컬럼2가 검사할 데이터가 없는 NULL 이므로 검사하지 않고 행이 입력됩니다.

## 10-7-3. PRIMARY KEY 제약조건.

- 테이블에 저장된 각 행을 고유하게 식별할 수 있는(즉, 대표할 수 있는) 사용자 데이터 또는 데이터의 조합입니다.

- 각 행을 대표하는 의미로 사용되는 데이터는 테이블에 하나만 있으면 되므로 다른 제약조건과는 달리 테이블에 오직 하나의 PRIMARY KEY 제약조건만 정의할 수 있습니다. 또한 행을 고유하게 식별해야 하므로 기능상 데이터도 반드시 있어야 하며(NOT NULL) 중복되면 안됩니다(UNIQUE).

| 컬럼1 | 컬럼2 | 한 행 입력 |
|---|---|---|
| 10 | A | |
| 20 | A | ← INSERT(1) |
| 10 | B | ← INSERT(2) |
| | B | ← INSERT(3) |

예를 들어, 컬럼1, 컬럼2로 구성된 테이블에서 컬럼1에 PRIMARY KEY 제약조건을 정의했다면,

- INSERT(1)은 기존의 행에 입력된 컬럼1의 값과 중복되지 않으므로 행이 입력됩니다.
- INSERT(2)는 기존의 행에 입력된 컬럼1의 값과 중복되므로 행이 입력되지 않고 에러가 발생됩니다.
- INSERT(3)은 입력되는 행의 컬럼1이 데이터가 없는 NULL 이기 때문에 행이 입력되지 않고 에러가 발생됩니다.

## 10-7-4. FOREIGN KEY 제약조건.

- 다음의 제약조건이 정의된 부서_테이블과 사원_테이블을 생성합니다.

(1) 부서_테이블의 행을 고유하게 식별하는 데이터로서 부서코드 컬럼에 기본키(PRIMARY KEY)를 정의합니다.
(2) 사원_테이블의 행을 고유하게 식별하는 데이터로서 사번 컬럼에 기본키(PRIMARY KEY)를 정의합니다.

부서_테이블

| 부서코드 (PK) | 부서명 |
|---|---|
| 10 | 오라클 |
| 20 | 시스템 |
| 30 | 개발 |

사원_테이블

| 사번 (PK) | 이름 | 부서코드 (FK) | |
|---|---|---|---|
| 100 | 관우 | 10 | 한 행 입력 |
| 101 | 유비 | 20 | ←INSERT(1) |
| 102 | 장비 | 3 | ←INSERT(2) |
| 103 | 조조 | | ←INSERT(3) |

- 사원_테이블의 부서코드에 FOREIGN KEY 제약조건이 정의되지 않으면, 부서_테이블의 부서코드와 상관없이 사원_테이블에 데이터가 INSERT 됩니다(INSERT(1),(2),(3)). 그렇지만, 사원_테이블에 입력된 행 중에서 부서코드가 3인 행은 부서_테이블에 부서코드가 3 인 부서가 존재하지 않으므로 잘못된 데이터지만 정상적인 것처럼 에러 없이 사원_테이블에 INSERT 되었습니다(INSERT(2)). 즉, 의미 없는 데이터가 사원_테이블에 저장되어 있습니다. 이러한 INSERT를 방지하기 위하여 필요한 것이 사원_테이블에 정의하는 FOREIGN KEY 제약조건입니다.

즉, FOREIGN KEY 제약조건이 명시되어 있다면, 사원_테이블에 행을 INSERT할 때, 부서_테이블의 부서코드 컬럼에 없는 값이 사원_테이블의 부서코드 컬럼에 INSERT 될 수 없도록 할 수 있습니다.

사원_테이블의 부서코드에 FOREIGN KEY 제약조건이 정의되어 있으면,

- INSERT(1)은 사원_테이블의 부서코드에 입력되는 값 20은 부서_테이블의 부서코드 컬럼에 있는 값이므로 행이 입력됩니다.

- INSERT(2)는 사원_테이블의 부서코드에 입력되는 값 3은 부서_테이블의 부서코드 컬럼에 없는 값이므로 행이 입력되지 않고 에러가 발생됩니다.

- INSERT(3)은 사원_테이블의 부서코드에 검사할 데이터가 없는 NULL 이므로 검사하지 않고 행이 입력됩니다.

- FOREIGN KEY 제약조건은 사원_테이블에 정의하며, 이 때 FOREIGN KEY 제약조건에 의해 참조되는 부서_테이블의 부서코드 컬럼에는 반드시 PRIMARY KEY(또는 UNIQUE) 제약조건이 정의되어 있어야만 합니다.

- FOREIGN KEY 제약조건이 정의되는 테이블을 CHILD-테이블이라고 하며, FOREIGN KEY 제약조건에 의해 참조되는 테이블(PRIMARY KEY 제약조건이 정의된 테이블)을 PARENT-테이블이라고 합니다.

**[참고] FOREIGN KEY 제약조건 관련 에러.**

- HR.EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼에는 FOREIGN KEY 제약조건이 정의되어 있으며, 이 제약조건은 HR.DEPARTMENTS 테이블의 DEPARTMENT_ID 컬럼에 정의된 PRIMARY KEY 제약조건을 참조하는 있습니다.

- HR.EMPLOYEES 테이블의 FOREIGN KEY 제약조건 때문에 다음의 DML 작업 시에 에러가 발생될 수 있습니다.

| HR.EMPLOYEES 테이블에 대한 INSERT/UPDATE | HR.DEPARTMENTS 테이블에 대한 DELETE/UPDATE |
|---|---|
| SQL> **UPDATE hr.employees**<br>**SET department_id = 55**<br>**WHERE department_id = 110;**<br>UPDATE hr.employees<br>*<br>ERROR at line 1:<br>ORA-02291: integrity constraint (HR.EMP_DEPT_FK) violated - parent key not found | SQL> **DELETE FROM hr.departments**<br>**WHERE department_id = 60;**<br>DELETE FROM hr.departments<br>*<br>ERROR at line 1:<br>ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated - child record found |
| HR.DEPARTMENTS.DEPARTMENT_ID에 값 55가 없기 때문에 발생된 에러입니다. | HR.EMPLOYEES.DEPARTMENT_ID에 60인 행이 존재하기 때문에 발생된 에러입니다. |

## 10-7-5. 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, USER, USERENV 함수들을 사용할 수 없습니다.
• **다른 행에 있는 다른 값을 참조하는 SELECT 문(즉, 서브쿼리)을 사용할 수 없습니다.**

| 컬럼1 | 컬럼2 | 한 행 입력 |
|---|---|---|
| A | 10 | ← 기존 행 |
| A | 20 | ←INSERT(1) |
| A | 0 | ←INSERT(2) |
| A | | ←INSERT(3) |

예를 들어 컬럼2에 [컬럼2 > 0]인 CHECK 제약조건을 정의했다면

• INSERT(1)은 값 20이 0 보다 크므로 조건을 만족하기 때문에 행이 입력됩니다.

• INSERT(2)는 값이 0 이므로 조건을 만족하지 않기 때문에 행이 입력되지 않고 에러가 발생됩니다.

• INSERT(3)은 입력되는 행의 컬럼2가 검사할 데이터가 없는 NULL 이므로 검사하지 않고 행이 입력됩니다.

## 10-8. 테이블 생성 실습-2: 제약조건 정의를 포함하여 테이블 생성.

• 아래의 구문을 연습하면서, 구문에 명시된 각 제약조건을 정의하는 문법을 암기하기 바랍니다.

```
SQL> CREATE TABLE hr.emps91
     (eid       NUMBER(4)
     ,name      VARCHAR2(4) CONSTRAINT nn_name_emps91 NOT NULL       --(1)
     ,address   VARCHAR2(30) DEFAULT 'SEOUL'
     ,salary    NUMBER(8)
     ,jumin     VARCHAR2(13) NOT NULL                               --(2)
     ,deptid    NUMBER(4)
     ,CONSTRAINT pk_eid_name_emps91 PRIMARY KEY(eid, name)         --(3)
     ,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)                 --(4)
               REFERENCES hr.departments(department_id) ON DELETE SET NULL );

Table created.
```

```
SQL> CREATE TABLE hr.emps92
     ( empid       NUMBER(6) CONSTRAINT emp92_emp_id PRIMARY KEY
     , f_name      VARCHAR2(20)
     , l_name      VARCHAR2(25) CONSTRAINT emp92_last_name_nn NOT NULL
     , email       VARCHAR2(25) CONSTRAINT emp92_email_nn NOT NULL
                                CONSTRAINT emp92_email_uk UNIQUE
     , phone_no    VARCHAR2(20)
     , hire_date    DATE CONSTRAINT emp92_hire_date_nn NOT NULL
     , job_id      VARCHAR2(10) CONSTRAINT emp92_job_nn NOT NULL
     , salary      NUMBER(8,2) CONSTRAINT emp92_salary_ck CHECK (salary>0)
     , comm_pct    NUMBER(2,2)
     , manager_id  NUMBER(6)
     , dept_id     NUMBER(4) CONSTRAINT emp92_dept_fk
                             REFERENCES hr.departments (department_id) ON DELETE CASCADE);

Table created.
```

## 10-9. 제약조건을 사용하기 위한 가이드 라인.

- 제약조건을 정의할 때는 [CONSTRAINT 제약조건이름]을 명시하여 이름을 지정하는 것을 권장합니다. --(1)
  만약 제약조건을 정의할 때, [CONSTRAINT 제약조건이름]을 명시하지 않으면, 데이터베이스 서버가 [SYS_Cnnnnnn] 형식으로 된 이름을 자동으로 부여합니다. --(2)

- 제약 조건을 정의하는 방법.

| | |
|---|---|
| 컬럼-레벨로 정의 | • 컬럼 정의 구문 안에서 정의합니다.<br>• NOT NULL 제약조건은 컬럼-레벨 방식으로만 정의해야 합니다. --(1) |
| 테이블-레벨로 정의 | • 컬럼 정의 구문 밖에서 (즉, 콤마(,)로 구분하여) 따로 정의합니다. --(4)<br>• 둘 이상의 컬럼을 조합하여 하나의 제약조건을 정의(Composite Constraint)할 때는 테이블-레벨 방식으로만 정의합니다. --(3)<br>• PRIMARY KEY, UNIQUE, FOREIGN KEY 제약조건을 Composite Constraint로 정의할 수 있습니다.<br>• CHECK 제약조건의 경우에는 위의 예제처럼 2개의 컬럼에 2개의 조건을 AND로 묶어서 정의한 경우에는 테이블-레벨로 정의해야 합니다. |

- UNIQUE, PRIMARY KEY 제약조건을 정의하면, 중복된 데이터를 빠르게 찾기 위하여 반드시 인덱스를 사용해야 합니다. 따라서, 사용할 수 있는 인덱스가 없으면 UNIQUE, PRIMARY KEY 제약조건이 정의될 때 자동으로 인덱스가 생성됩니다.

- 제약조건을 정의하는 시기

| |
|---|
| • 테이블 생성 시에 정의할 수 있습니다.<br>• 테이블 생성 후에 추가할 수 있습니다. (SQL2 과정에서 설명합니다) |

- 테이블에 정의된 제약조건은 데이터 딕셔너리(DATA DICTIONARY)를 이용하여 정보를 확인할 수 있습니다.

```
CONSTRAINT_NAME                TABLE_NAME
------------------------------ ------------------------------
NN_NAME_EMPS91                 EMPS91
SYS_C005423                    EMPS91
CK_SAL_ADDR_EMPS91             EMPS91
PK_EID_NAME_EMPS91             EMPS91
UK_JUMIN_EMPS91                EMPS91
FK_EMPS_DEPT91                 EMPS91
```

## 10-10. FOREIGN KEY 제약조건 기술 시에 사용되는 키워드의 의미 및 주의점.

- FOREIGN KEY 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 테이블 이름(컬럼이름)**을 명시하고, 이 때 참조되는 컬럼에는 PRIMARY KEY 또는 UNIQUE 제약조건이 반드시 설정되어 있어야만 합니다.

- 부서_테이블의 부서코드에 PRIMARY KEY 제약조건이 정의되어 있고, 이 PRIMARY KEY 제약조건을 참조하는 FOREIGN KEY 제약조건이 ON DELETE CASCADE 또는 ON DELETE SET NULL 옵션이 없이 사원_테이블의 부서코드에 정의되어 있을 때, 부서_테이블의 참조되는 키 값을 가지는 행을 삭제하려고 시도하면, 에러가 발생됩니다.

  DELETE(1): 사원_테이블의 부서코드 컬럼에 10 인 행에 의하여 부서_테이블의 부서코드 컬럼이 10인 행이 참조되고 있으므로 행이 삭제되지 않고 에러가 발생됩니다.

  DELETE(2): 사원_테이블의 부서코드 컬럼에 30 인 행이 없기 때문에 참조되지 않으므로 행이 삭제됩니다.

| 부서_테이블 | | |
|---|---|---|
| 부서코드 (PK) | 부서명 | |
| 10 | 오라클 | ←DELETE(1) |
| 20 | 시스템 | |
| 30 | 개발 | ←DELETE(2) |

| 사원_테이블 | | |
|---|---|---|
| 사번 (PK) | 이름 | 부서코드 (FK) |
| 100 | 관우 | 10 |
| 101 | 유비 | 20 |
| 102 | 장비 | 10 |
| 103 | 조조 | 10 |
| 104 | 여포 | 20 |

- 만약 **사원_테이블의** FOREIGN KEY **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** FOREIGN KEY **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.

## 10-11. 테이블 생성 실습-3: 서브쿼리를 이용하여 테이블 생성.

- 서브쿼리를 이용해서 기존 테이블의 컬럼 정의 및 데이터를 복사하면서 새로운 테이블을 생성할 수 있습니다.

- NOT NULL 제약조건만 복사되며, PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK 제약조건은 복사되지 않습니다.

- CLOB, BLOB 데이터 유형 컬럼은 복사되지만, LONG, LONG RAW 데이터 유형 컬럼을 복사되지 않습니다.

1> 서브쿼리를 이용하여 테이블을 생성합니다.

```
SQL> CREATE TABLE hr.dept70
     AS
       SELECT employee_id, last_name as ENAME, salary*12 ANNSAL, hire_date hiredate
       FROM hr.employees
       WHERE department_id = 70 ;

Table created.


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 created.
```

2> DESCRIBE 명령어를 이용하여 컬럼 정보 확인

```
SQL> desc hr.dept70 ;
 Name                  Null?    Type
 --------------------- -------- --------------
 EMPLOYEE_ID                    NUMBER(6)
 ENAME                 NOT NULL VARCHAR2(25)
 ANNSAL                         NUMBER
 HIREDATE              NOT NULL DATE
```

```
SQL> desc hr.dept80
 Name                  Null?    Type
 --------------------- -------- --------------
 EMPNO                          NUMBER(6)
 ENAME                 NOT NULL VARCHAR2(25)
 ANNSAL                         NUMBER
 HIREDATE              NOT NULL DATE
```

3> 테이블의 데이터 확인

```
SQL> SELECT  * FROM hr.dept70 ;

EMPLOYEE_ID ENAME                        ANNSAL HIREDATE
----------- ------------------------ ---------- ------------------
        204 Baer                         120000 07-JUN-02
```

[참고] 서브쿼리에서 SELECT 절에 명시된 컬럼이 함수로 처리되거나 산술 연산이 포함된 표현식으로 기술된 경우, 반드시 컬럼 Alias로 처리해야 합니다. 처리 하지 않은 경우 다음처럼 에러가 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 ;
AS  SELECT employee_id, UPPER(last_name) as last_name, salary*12, hire_date
                                                        *
ERROR at line 2:
ORA-00998: must name this expression with a column alias
```

## 10-12. 테이블 절단(TRUNCATE TABLE)을 통한 데이터 삭제.

• 테이블을 절단하면, 테이블이 처음 생성될 때의 상태가 되며 따라서, 테이블의 데이터가 모두 유실됩니다.

• FOREIGN KEY 로 참조를 당하는 테이블(PARENT TABLE)을 TRUNCATE 할 수 없습니다.

• 테이블 절단은 아래와 같은 이유 때문에 WHERE절 없는 DELETE-문으로 모든 데이터를 삭제하는 것보다 훨씬 빠르게 수행됩니다.

| | |
|---|---|
| TRUNCATE 문 | UNDO-데이터가 구성되지 않습니다. |
| WHERE 절 없는 DELETE 문 | UNDO-데이터가 구성되면서 데이터가 삭제됩니다. |

• TRUNCATE 문장은 DDL 문장이기 때문에 자동으로 COMMIT 됩니다. 즉, ROLLBACK되지 않습니다. 따라서, 중요한 데이터가 있는 테이블을 TRUNCATE 해야 한다면, 작업 전에 EXPORT 받는 것을 권장합니다.

### ◈ TRUNCATE TABLE 명령어 실습.

```
1> 실습용 HR.EMPLOYEES3 테이블을 서브쿼리를 이용하여 생성합니다.
SQL> CREATE TABLE employees3
     AS
        SELECT *
        FROM hr.employees ;

Table created.


2> HR.EMPLOYEES33 테이블을 절단(TRUNCATE) 합니다.
SQL> TRUNCATE TABLE hr.employees3 ;

Table truncated.
```

## 10-13. ALTER TABLE 문장.

• ALTER TABLE 문장을 이용하여 다음과 같은 작업을 수행할 수 있습니다.

• 컬럼 추가/컬럼 삭제/컬럼 수정이 가능합니다.
• 새로 추가된 컬럼에 디폴트 값을 정의할 수 있습니다.
• 컬럼의 이름 변경
• 제약조건(CONSTRAINTS)을 추가/삭제/DISABLE/ENABLE.
• 읽기 전용 상태로 테이블 변경

**[참고] 읽기 전용 테이블 설정(11gNF)**

• 11g 버전부터 사용자가 원하는 경우, 테이블에 대하여 READ ONLY를 지정하여 테이블을 읽기 전용 모드로 둘 수 있습니다.

• READ-ONLY 모드인 테이블에 대해서는 DML-문 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.

■ 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.
```

☞ [ALTER TABLE]문에 대한 보다 자세한 내용은 [SQL-2] 과정을 참조하십시오.

## 10-14. 테이블 삭제(DROP TABLE 문장).

• 테이블을 삭제하면, 데이터베이스에 정의된 테이블의 정의(테이블 이름, 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, 롤백이 되질 않습니다.

• 10g-버전부터는 휴지통(RECYCLEBIN) 기능에 의하여, 테이블을 삭제하면, 관련된 시스템 정의가 삭제되지 않고 휴지통 정보로 관리됩니다. 휴지통에 대해서는 뒤의 과정에서 설명합니다.

• 특정 배치 작업을 위해 서브쿼리를 이용하여 생성한 테이블은 작업 후에 더 이상 필요 없는 경우,
아래의 구문으로 삭제하는 것을 권장합니다.

```
SQL> DROP TABLE hr.dept80 ;

Table dropped.
```

• DROP TABLE 구문을 이용하여 본 과정에서 실습 시에 생성한 테이블을 모두 삭제해 봅시다.

```
SQL> DROP TABLE hr.dept70 PURGE ;

Table dropped.

SQL> DROP TABLE hr.cust PURGE ;

Table dropped.

SQL> DROP TABLE hr.emps91 PURGE ;

Table dropped.

SQL> DROP TABLE hr.emps92 PURGE ;

Table dropped.
```

☞ 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.

# 11 테이블 외의 오라클 데이터베이스 객체(VIEW, SEQUENCE, INDEX, SYNONYM).

## ◈ 학습 목표.

■ 뷰(VIEW) 객체를 사용하는 목적과 뷰 객체를 생성하는 방법을 학습합니다.

■ 시퀀스(SEQUENCE) 객체의 기능과 시퀀스 객체를 생성하는 방법을 학습합니다.

■ 인덱스(INDEX) 객체를 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.<br>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 IO를 줄여서 테이블의 행을 빠르게 찾아내어 쿼리의 처리 성능을 올리기 위하여 사용합니다. |
| SYNONYM | 데이터베이스 테이블에 대한 대체-이름(동의어)를 정의합니다. |

## 11-2. 뷰(VIEW).

• 아래의 실습을 수행하면서, 뷰가 어떤 객체인지, 그리고 무슨 목적으로 뷰를 사용하는지 설명합니다.

■ HR.EMPLOYEES 테이블로부터 department_id가 30인 부서에 근무하는 직원의 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.
```

• 위의 실습에서 알 수 있듯이 뷰는, 테이블의 일부 데이터를 조회하는 SELECT 문이 정의된 객체입니다.
따라서, 저장 공간(STORAGE)에 저장된 데이터를 가지는 테이블과는 달리 VIEW는 사용자 데이터를 저장하지 않습니다.

■ 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)
```

■ 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 사용합니다. 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.

- 뷰에 정의된 SELECT 문에 명시된 테이블 (위의 예에서 HR.EMPLOYEES)을 뷰에 대한 BASE TABLE이라고 합니다.

- 뷰를 생성하기 위해서는 CREATE VIEW 시스템 권한이 필요합니다.

## 11-2-2. VIEW 생성 실습-1.

■ HR.EMPLOYEES 테이블에서 department_id가 50인 직원들의 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 OCONNELL             31200
        199 GRANT                31200
        120 WEISS                96000
        121 FRIPP                98400
...

45 rows selected.
```

■ 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
```

■ HR.EMPLOYEES 테이블과 HR.DEPARTMENTS 테이블의 데이터를 이용하여 department_name 별 직원의 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 서브쿼리에 명시하는 컬럼-Alias를 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 에러가 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 ONLY을 사용하여 VIEW를 생성합니다.

■ 아직 생성되지 않은 HR.EMPLOYEES111 테이블에 대하여 department_id가 10 인 직원의 employee_id, last_name, job_id 컬럼들로 구성된 직원정보를 표시하는 HR.EMPVU10 뷰를 생성하시오.

```
SQL> CREATE FORCE VIEW hr.empvu10 (employee_number, employee_name, job_title)
     AS SELECT employee_id, last_name, job_id
        FROM hr.employees111
        WHERE department_id = 10
        WITH READ ONLY ;

Warning: View created with compilation errors.
```

### [참고] NOFORCE(디폴트)/FORCE 키워드

VIEW 사용 시에 FORCE 키워드를 사용하면, SELECT문에 명시된 테이블이 데이터베이스에 존재하지 않더라도 VIEW를 생성할 수 있습니다. NOFORCE/FORCE를 명시하지 않은 경우에는 디폴트로 NOFORCE 키워드를 사용한 것으로 간주되며 SELECT 문에 명시된 테이블이 데이터베이스에 존재할 때만 VIEW가 생성됩니다.

위의 실습에서 데이터베이스에 employees111 테이블이 없지만, FORCE 옵션이 때문에 VIEW가 생성되긴 하지만, 컴파일 경고 메세지가 표시됩니다.

## 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.

[참고] WITH CHECK OPTION을 명시된 서브쿼리를 INSERT-문에서 사용하는 문장.

```
SQL> INSERT INTO (SELECT *
                  FROM hr.copy_emp
                  WHERE department_id < 30 WITH CHECK OPTION )
     VALUES (99999,'Taylor',5000, TO_DATE('28-06-99','DD-MM-RR'),'ST_CLERK', 50) ;
            FROM hr.copy_emp
                 *
ERROR at line 2:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

■ HR.EMPLOYEES 테이블을 이용하여, department_id가 20인 직원의 모든 정보를 표시하는 hr.empvu20 뷰를 생성하시오. 단, WITH CHECK OPTION을 이용하여 DML시에 department_id가 20인지 검사하도록 하시오.

```
1> WITH CHECKPOINT을 이용하여 VIEW를 생성

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 행 입력 시도.

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 에러


SQL> DELETE FROM hr.empvu20 WHERE department_id = 50 ;

0 rows deleted. <-- 삭제 되지 않음.
```

☞ 앞에서 뷰의 WHERE 절의 조건을 만족하지 않은 경우에는, 입력 시 에러가 발생되고, DELETE 도 되지 않습니다.

[참고] 뷰를 통해서 테이블에 데이터를 INSERT/UPDATE 시에, DEFAULT 키워드를 사용하면 아래와 같은 에러가 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 SELECT-문 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.

■ 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 SELECT-문을 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 대하여 DML 수행 시에 규칙.

▣ 뷰에 정의된 SELECT 문이 아래의 경우에 해당되면, 뷰를 통해서는 BASE-테이블의 데이터를 DELETE 할 수 **없습니다.**

- 집합함수가 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.

- 집합함수가 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.

- 집합함수가 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.

■ 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.

■ HR.SALVU50 뷰를 통하여 employee_id가 197인 직원의 NAME을 'SHSHIN' 으로 변경해보시오.

```
SQL> UPDATE hr.salvu50 SET name = 'SHSHIN' WHERE employee_id= 197 ;
UPDATE hr.salvu50 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 에러가 발생됩니다.

■ HR.SAL30VW 뷰를 통하여 HR.EMPLOYEES 테이블에 employee_id가 300, first_name이 'SHSHIN', salary가 30000인 직원정보를 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 SELECT-문에 HR.EMPLOYEES 테이블의 NOT NULL이 정의된 컬럼 모두가 포함되지 못했으므로, 입력 시 에러가 발생됩니다.

[참고] 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.

■ 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 INSERT-문에 대신 넣어줄 때 사용되는 객체 입니다.

- SEQUENCE 객체를 생성하기 위해서는 CREATE SEQUENCE 시스템 권한이 필요합니다.

### 11-3-1. SEQUENCE 생성 문법 및 각 옵션이 생략되었을 경우에 설정되는 디폴트 값.

- SEQUENCE를 생성하는 기본적인 문법은 다음과 같습니다.

```
CREATE SEQUENCE [스키마 이름.]시퀀스이름
    [START WITH n]
    [INCREMENT BY n]
    [{MAXVALUE n | NOMAXVALUE}]
    [{MINVALUE n | NOMINVALUE}]
    [{CYCLE | NOCYCLE}]
    [{CACHE n | NOCACHE}]
;
```

[옵션에 대한 간단한 설명]

| 옵션 | 기능 |
|---|---|
| START WITH n | 숫자 값이 생성되는 첫 시작-값을 지정합니다. 명시하지 않으면, 디폴트로 1 이 설정됩니다. |
| INCREMENT BY n | 숫자 값이 생성될 때, 값 사이의 간격을 지정합니다. 명시하지 않으면, 디폴트로 1 이 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 1 이 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 에러가 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, 세션의 시퀀스에 대한 다음 값의 요청을 빠르게 처리할 수 있습니다.

## 11-3-2. SEQUENCES 생성.

• 아래의 3개의 시퀀스를 생성해 봅니다.

■ 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.

■ 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.
```

■ 메모리에 10개에 해당하는 끝수가 캐싱되면서, 500부터 시작해서 10씩 간격을 두고 최대 9999까지 숫자를 생성하며, 최대값에 도달되었을 경우 에러를 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.
```

## 11-3-3. 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.<br><br>SQL> **SELECT DEPT_DEPTID_SEQ1.CURRVAL FROM dual ;**<br>SELECT DEPT_DEPTID_SEQ1.CURRVAL FROM dual<br>  *<br>ERROR at line 1:<br>ORA-08002: sequence DEPT_DEPTID_SEQ1.CURRVAL is not yet defined in this session<br><br>테이블에 한 행의 데이터를 입력할 때, 시퀀스이름.NEXTVAL 로 하나의 컬럼에 입력한 값을 같은 행의 다른 컬럼에서 사용하고 싶을 때 시퀀스이름.CURRVAL을 이용합니다.<br><br>INSERT INTO 테이블이름<br>VALUES (시퀀스이름.NEXTVAL, TO_CHAR(SYSDATE,' YYYYMMDD' )\|\|' _' \|\|시퀀스이름.CURRVAL) ; |

**[참고] 시퀀스에 대한 NEXTVAL 및 CURRVAL 사용 규칙.**

| |
|---|
| 다음과 같은 상황에서 NEXTVAL 및 CURRVAL을 사용할 수 있습니다.<br>• 서브쿼리의 일부가 아닌 SELECT-문의 SELECT 리스트.<br>• INSERT 문에서 테이블이름 대신 사용된 서브쿼리의 SELECT 리스트<br>• INSERT 문의 VALUES 절<br>• UPDATE 문의 SET 절 |
| 다음과 같은 상황에서는 NEXTVAL 및 CURRVAL을 사용할 수 없습니다.<br>• 뷰의 SELECT 리스트<br>• DISTINCT 키워드가 있는 SELECT 문<br>• GROUP BY, HAVING 또는 ORDER BY 절이 있는 SELECT 문<br>• SELECT, DELETE 또는 UPDATE 문의 테이블이름 대신 사용된 서브쿼리<br>• CREATE TABLE 또는 ALTER TABLE 문의 DEFAULT 식 |

## 11-3-4. 시퀀스의 사용.

• 테이블에 한 행의 데이터를 입력할 때, 특정 컬럼의 값을 시퀀스를 이용하여 입력하는 방법을 학습합니다.

• 시퀀스의 다음 값을 호출하여 사용하려면, SQL-문에서 시퀀스이름.NEXTVAL(예, DEPT_DEPTID_SEQ1.NEXTVAL) 을 이용합니다.

■ 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.

## 11-3-5. 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.

■ HR.DEPT_DEPTID_SEQ1 시퀀스의 CACHE 수를 1,000,000 으로 늘리시오.

```
SQL> ALTER SEQUENCE DEPT_DEPTID_SEQ1 CACHE 1000000 ;

Sequence altered.
```

**[참고] 다음과 같은 경우에 시퀀스를 통해서 입력된 데이터의 값들 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.

## 11-3-6. 시퀀스 삭제.

■ 필요 없는 HR.DEPT_DEPTID_SEQ1 시퀀스를 삭제하시오.

```
SQL> DROP SEQUENCE hr.dept_deptid_seq1;

Sequence dropped.
```

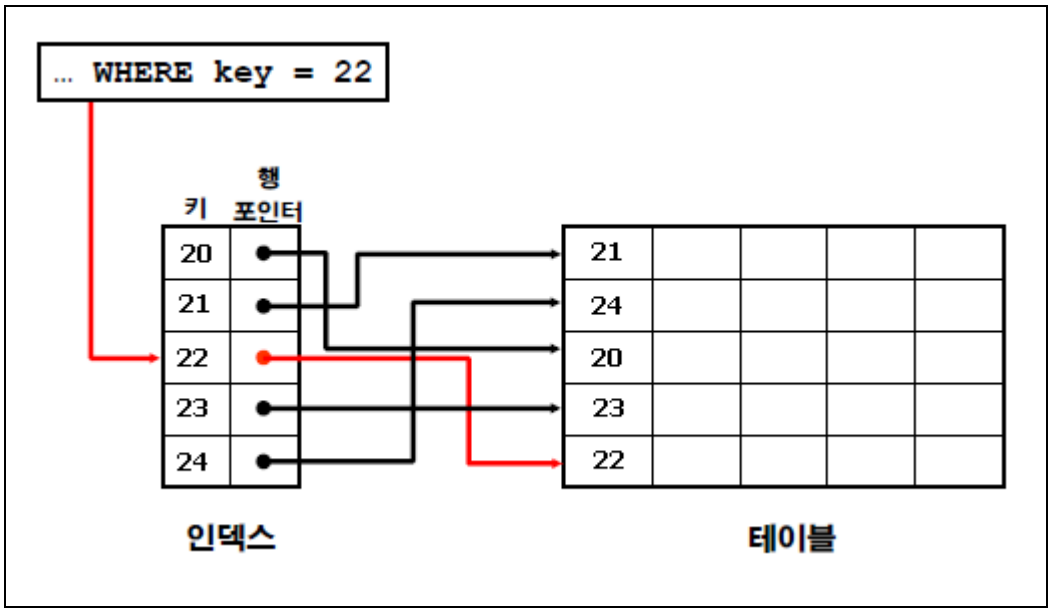
## 11-4. 인덱스(INDEX).

**[참고] 테이블에 대한 전체 테이블 스캔 방법과 인덱스 스캔 방법 개요.**

데이터베이스 서버로 전달된 사용자의 SQL-문이 서버-프로세스에 의하여 처리되어 최종적인 결과를 표시하기 위해서는 디스크에 저장된 데이터를 서버의 메모리로 로드 해야 합니다. 이 때, 디스크 상에 저장된 데이터를 메모리로 로드 하는 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 SQL-문의 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 SQL-문(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 SQL-문의 처리 성능을 향상시킬 수 있습니다.

**[참고] 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 SQL-문의 **WHERE절 또는 조인 조건에 자주 나오는 컬럼**을 **인덱스-키**로 가지는 인덱스를 생성합니다.

- 테이블에 **저장된 행이 매우 극소수**이고 대부분의 쿼리들에서 그 테이블에 저장된 행의 2% ~ 4% **를 초과하는 다수의 데이터를 추출할 것으로 예상되면**, **인덱스를 사용하지 않는 것이 권장**됩니다.

- 쿼리의 WHERE절에 그다지 자주 **사용되지 않는 컬럼**을 **인덱스-키로 사용하여 인덱스를 생성하지 마십시오**.

- WHERE절에 사용되지만, **" 표현식의 일부" 로서 참조되는 컬럼**을 **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. PRIMARY KEY 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.

■ 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) ;

Index created.
```

- 위의 생성실습에서 HR.EMPLOYEES.LAST_NAME 컬럼이 HR.IDX_LNAME_EMP 인덱스의 인덱스-키 입니다.

- WHERE 절 작성시에 조건절에서 인덱스-키 컬럼을 어떻게 명시하냐에 따라 SQL-문이 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 데이터 유형 컬럼 | 컬럼 \|\| '' |
| • 숫자 또는 DATE 데이터 유형 컬럼 | 컬럼 + 0 |

## 11-4-3. 인덱스 삭제.

■ 앞에서 생성한 HR.IDX_LNAME_EMP 인덱스를 삭제하시오.

```
SQL> DROP INDEX hr.idx_lname_emp ;

Index dropped.
```

• PRIMARY KEY 및 UNIQUE 제약조건이 사용하는 인덱스는 PRIMARY KEY 및 UNIQUE 제약조건을 먼저 삭제하거나 DISABLE 시켜야지만 인덱스를 삭제할 수 있습니다. 만약, PRIMARY KEY 및 UNIQUE 제약조건이 사용 중인 인덱스를 삭제하려고 시도하면 다음의 에러가 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
```

## 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) 타입.

| | |
|---|---|
| NORMAL SYNONYM<br>(일반 동의어) | • 데이터베이스에 로그인 한 계정이 자신이 소유한 스키마 객체로서 생성하며, 소유자 계정만 사용하는 동의어입니다.<br>• 로그인 계정이 NORMAL SYNONYM을 생성하려면 CREATE SYNONYM 시스템 권한을 가지고 있어야 합니다. |
| PUBLIC SYNONYM<br>(공용 동의어) | • DBA(SYS, SYSTEM)가 생성하며, 데이터베이스의 모든 로그인 계정이 사용하는 동의어입니다.<br>• 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.

■ 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.
```

■ 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.

■ 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_PUSYN이(가) 생성되었습니다.
```

[참고] CREATE PUBLIC SYNONYM 시스템 권한이 없는 HR 계정으로 접속하여 PUBLIC SYNONYM을 생성하려고 시도하면 다음의 에러가 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.

■ HR 계정으로 접속한 세션에서, HR 스키마객체인 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.

■ SYS 계정으로 접속한 세션에서 dept_pusyn 공용동의어를 삭제하시오.

```
SQL> DROP PUBLIC SYNONYM dept_pusyn ;

public synonym DEPT_PUSYN이(가) 삭제되었습니다.
```

☞ 공용 동의어는 DROP PUBLIC SYNONYM 권한을 가진 계정 또는 데이터베이스 관리자 계정이 삭제할 수 있습니다.

## [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** | SELECT-문의 결과레코드 마지막에 표시되는 메시지를 표시하지 않음. |
| **BREAK ON JOB_ID** | 출력결과의 Heading이 JOB_ID인 컬럼에서 연속적으로 표시되는 동일 데이터를 처음 한번만 표시.<br>(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*Develper로 윈도우에서 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           EMPLOYEE
 Category          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
(중략)...
           Hall                        $9,000.0            .25
                         Confidential

Tue Nov 22                                          page     2
(중략)...
```

**[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 SQL-문을 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) SQL-문에서 변수이름 앞에 &를 명시하여 선언하고 SQL문 실행 시에 값을 지정 합니다.
(3) SQL-문에서 변수이름 앞에 &&를 명시하여 선언하고 SQL문 실행 시에 값을 지정 합니다.

• SQL-문에서 변수이름 앞에 &&를 명시하거나 DEFINE 명령어를 이용하여 선언하는 경우에는, 접속 세션에 선언된 치환변수가 계속 유지됩니다.

• SQL-문에서 변수이름 앞에 &를 명시하여 선언하는 경우에는 접속 세션에 선언된 치환변수가 유지되지 않습니다.

• SQL-문에서 & 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).

1. 오라클-서버가 구성된 리눅스 가상머신의 터미널에서 SQL*Plus를 실행하여 HR 계정으로 접속합니다.

```
$ sqlplus hr/oracle4U
```

2. DEFINE 명령어로 세션에 eid1 치환변수를 선언하고 100을 지정합니다.

```
SQL> DEFINE eid1=100
SQL>
```

3. [DEFINE 치환변수이름]명령어를 이용하여 세션에 유지되어 있는 지정된 치환변수의 값을 확인할 수 있습니다.

```
SQL> DEFINE eid1
DEFINE EID1          = "100" (CHAR)
SQL>
```

4. SELECT-문 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              -- SQL-문 실행 시에 eid2 치환변수에 지정할 값을 물어옵니다.
old   3:       WHERE  employee_id=&eid2
new   3:       WHERE  employee_id=101
Enter value for did: 90                -- SQL-문 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 SELECT-문을 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 SELECT-문을 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
```