

# 데이터베이스 기초에서 설계 까지

발행일 : 2023.01.01  
변경일 : 2023.01.01  
저자 : 박수민  
이메일 : [foxman12@hanmail.net](mailto:foxman12@hanmail.net)  
발행처 : 휴먼교육

# 목차

<b>목차</b>	<b>2</b>
<b>01. 데이터베이스의 시작</b>	<b>4</b>
> 01. 관계형 데이터베이스 이해	4
> 02. 오라클 설치 및 계정생성	7
> 03. 자료형	13
> 04. 테이블 만들어 조작해 보기	17
> 05. sql developer 설치하기	30
> 06. select문 사용하기	47
> 07. distinct로 중복제거하기	48
> 08. select의 where절	51
> 09. null 검색하기	53
> 10. 컬럼 연산	54
> 11. 컬럼 별명 as	54
> 12. Concatenation(  )를 이용한 컬럼 합치기	55
> 13. and or not 논리 연산자	56
> 14. between 연산자	58
> 15. null 연산과 null값 변경 함수	59
> 16. decode와 case 문법	61
> 17. in 연산자	63
> 18. like 연산자	65
> 19. order by 절	67
> 20. insert,update,delete문 사용하기	69
> 21. 중간 문제 풀이	71
> 22. 함수	76
> 23. 오라클에서 시간계산 하기	79
> 24. 그룹함수	82
> 25. group by 절	84
> 26. having 절과 sql실행 순서	87
> 27. 중간 문제풀이	88
> 28. join	90
> 29. sub쿼리	111
> 30. exists 함수	114
> 31. any, some, all 사용법	116
> 32. 무결성 제약 조건	122
> 33. view와 sequence 사용하기	132
> 34. index 사용하기	134
> 35. 트랜잭션 이해하기	137
> 36. HR 테이블 이해하기	140
<b>02. 데이터베이스 모델링</b>	<b>149</b>
> 01. ER 다이어그램	149

> 02. ERDCloud	158
> 03. HR 데이터베이스 살펴보기	160
> 04. Northwind 데이터베이스 살펴보기	171
> 05. 게시판 데이터베이스 설계하기	198
> 06. 정규형	201

# 01. 데이터베이스의 시작

---

## > 01. 관계형 데이터베이스 이해

---

데이터베이스는 현실 세계의 데이터를 컴퓨터에 저장하는 가장 대표적인 방법이다.

컴퓨터에 데이터를 저장하는 방법으로 파일 시스템이 있는데 컴퓨터에 파일로 저장하는 방법 이여서 혼자 보기는 쉬우나 여럿이 보기 어렵다. 차후 이를 개선하여 여러 사람이 함께 데이터를 쉽게 사용할 수 있도록 데이터 베이스를 사용 하였다.

파일 시스템은 컴퓨터에 데이터를 파일로 저장하는 시스템이다. 한글, 워드에 글을 작성하고 파일로 저장하면 원할때 불러와 저장한 데이터를 확인할 수 있고 이런 방식이 파일 시스템이다.

파일은 혼자 사용하기 편하나 여럿이 사용하는데 불편함이 있어 여러 사람이 24시간 함께 데이터를 공유하고 사용할 수 있는 데이터베이스를 만들어 사용하게 되었다.

데이터베이스란? 데이터를 컴퓨터에 저장해 놓고 매일 24시간 여러 사람과 함께 데이터를 사용하는 시스템이다. 현실 세계의 데이터를 컴퓨터에 저장하는 방법을 고민 하다 파일 시스템을 사용하게 되었고 여러 사람이 같은 파일을 사용하면서 생기는 문제점을 개선하여 데이터베이스 시스템을 만들었다. 데이터 베이스에 저장된 데이터는 컴퓨터를 꺼도 사라지지 않고 나중에 다시 컴퓨터를 켜서 데이터를 사용할 수 있다.

누군가가 데이터 베이스를 설치한 컴퓨터를 날마다 쉬지 않고 운영한다면 여러 사람들이 데이터를 공유하면서 원하는 시점에 데이터를 사용할 수 있다. 이렇게 24시간 데이터를 제공해 주는 컴퓨터를 데이터 베이스 서버라 한다.

네이버, 다음 같은 웹사이트들이 24시간 운영되면서 지금 우리가 배울 데이터 베이스에서 데이터를 가져다가 여러 사용자에게 브라우저를 통해서 원하는 데이터 정보를 제공해 주는 일을 하고 있다. 이런 웹사이트를 만드는 작업을 웹 프로그램이라 한다.

이때 데이터를 제공하는 컴퓨터를 서버 사용하는 컴퓨터를 클라이언트라 한다.

한글, 워드와 같이 문서 제작 소프트웨어가 여러개 있듯이 데이터 베이스 소프트웨어에는 오라클 mssql SQLite mysql MariaDB 등이 있으며, 이 중 책에서 다룬 데이터 베이스는 오라클이다. 다른 소프트웨어 처럼 컴퓨터에 오라클을 설치하면 본인 컴퓨터를 데이터 베이스로 사용 할 수 있게 된다. 데이터 베이스를 사용하게 되면 간단하게 메모장을 열어 데이터를 컴퓨터에 저장하는 방법을 뛰어 넘어 여러 사람이 함께 저장한 데이터를 쉽게 가져다 사용 할 수 있게 된다. 결국에 데이터 베이스는 여러 사람이 언제 어디서 누구나 데이터를 저장하고 가져다 쓸 수 있도록 관리하는 프로그램이다.

대부분의 데이터베이스는 관계형 데이터베이스 이론을 사용하여 만들어졌다. 우리가 배울 오라클도 관계형 데이터베이스이다. 관계형 데이터베이스는 아래와 같은 표 형태로 데이터를 읽어 오고 저장하는 데이터베이스를 의미한다. 우리는 이미 현실 세계에서 표를 사용해 데이터를 기록하는 것 처럼 관계형 데이터 베이스와 같은 방식으로 데이터를 사용하고 있다.

관계형 데이터 베이스에서 사용하는 용어를 확인해 보자.

교실에서 학생들의 정보를 종이에 기록한다면 다음과 같은 표를 그리고 학생들의 정보를 입력할 것입니다. 이런 방식으로 데이터를 저장하는 데이터베이스를 관계형 데이터베이스라고 한다.(RDBMS) Relational Database Management System 표형태로 데이터를 저장하는 데이터베이스를 관계형 데이터베이스 라고 한다.

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

새로운 모임에 모여 있고 사람들이 사람정보를 만들 계획이라고 생각해 보자. 대표로 한 사람이 종이를 꺼내 직사각형 모양의 표(table)를 그리고 상단에 기록 할 데이터 속성 중 저장할 속성의 column(컬럼, 열)를 작성 할 것이다. 속성, 컬럼, 열을 예로 들자면 이름, 나이, 키, 생일 등등이 있다. 컬럼 작성 후 학생 데이터를 하나씩 입력 하는 작업을 할 것이고 이를 로우(row), 행, 데이터 입력이라 한다. 이렇게 사람들이 많이 사용 하는 표 형태 저장 방법을 응용해서 만든 데이터베이스를 관계형 데이터베이스라고 한다.

데이터베이스에서 현실세계의 데이터를 저장하는 과정은 객체를 테이블 이름으로 만들고 객체에서 저장할 속성을 뽑아서 컬럼으로 만들고 테이블에 저장할 데이터를 테이블안에 넣으면 된다.

객체는 현실 세계에서 식별 가능한 존재하는 모든 것을 의미한다

속성은 사물의 상태나 특징을 기술할 것을 의미 한다.

row는 테이블에 입력된 데이터를 의미 한다 .

컬럼은 데이터의 여러 속성중 저장하기 위해 선택된 속성이고 로우는 실제 저장할 대상의 속성값에 해당한다. 상위 예를 가지고 설명해 보자.

현실 세계의 데이터를 데이터베이스에 넣기 위해서 표(테이블)을 사용한다. 현실 세계에서

종이에 표를 그리고 데이터 기록 할 목록 (컬럼)을 기술하는 작업을 데이터 베이스에서는 테이블 생성이라 이야기 하고 명령으로 `create`를 사용 한다. 테이블(표)을 생성 하는 것을 `create table`, 테이블을 삭제하는 것을 `drop table`, 변경하는것을 `alter table`이라고 한다. 상위 예를 가지고 설명해 보자.

그 다음에 해야 할 작업은 테이블에 데이터를 채워넣는 작업이다. 테이블에 데이터를 넣는 작업을 `insert`라고 한다. 테이블에 들어 있는 원하는 데이터를 찾는 명령어를 `select`라 한다. 표에 들어 있는 데이터를 수정 할 때 사용하는 명령어를 `update`라 한다. 데이터를 삭제할 때 사용하는 명령어를 `delete`라 한다. 정리하면 `insert` 입력, `select` 검색, `update` 수정, `delete` 삭제에 해당한다. CRUD란? `create` `read` `update` `delete`의 의미로 `insert` `select` `update` `delete` 작업을 줄여서 표시한다.

상의 키워드들은 sql에 사용 된다. sql이란? 질의어라 하는데 여러 사람이 데이터베이스를 사용 할 때 사용하는 약속된 언어이다. Sql를 이용하여 원하는 데이터를 데이터베이스에 요청 하면 요청한 데이터를 제공해 준다.

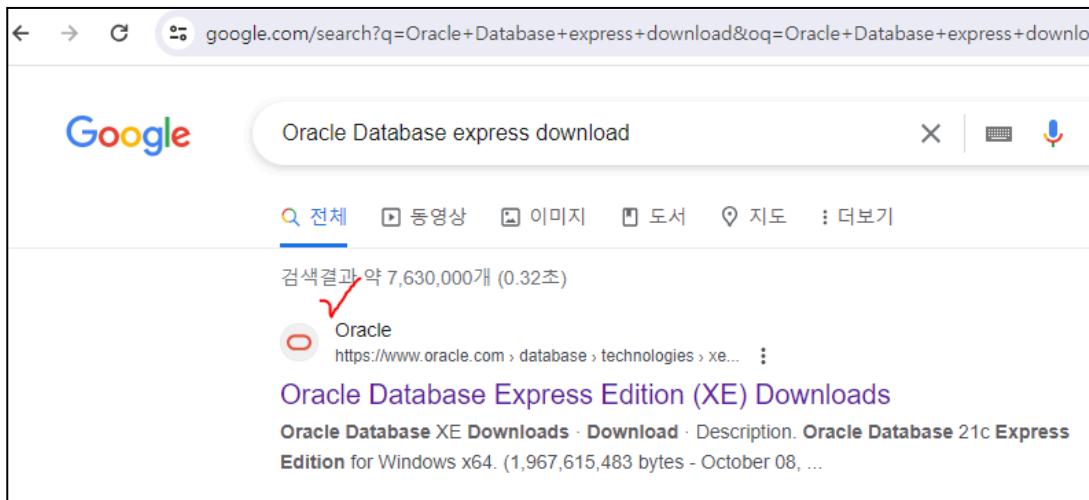
연습문제들을 날마다 확인해서 익숙해 질수 있도록 답해 보자.

#### 연습문제

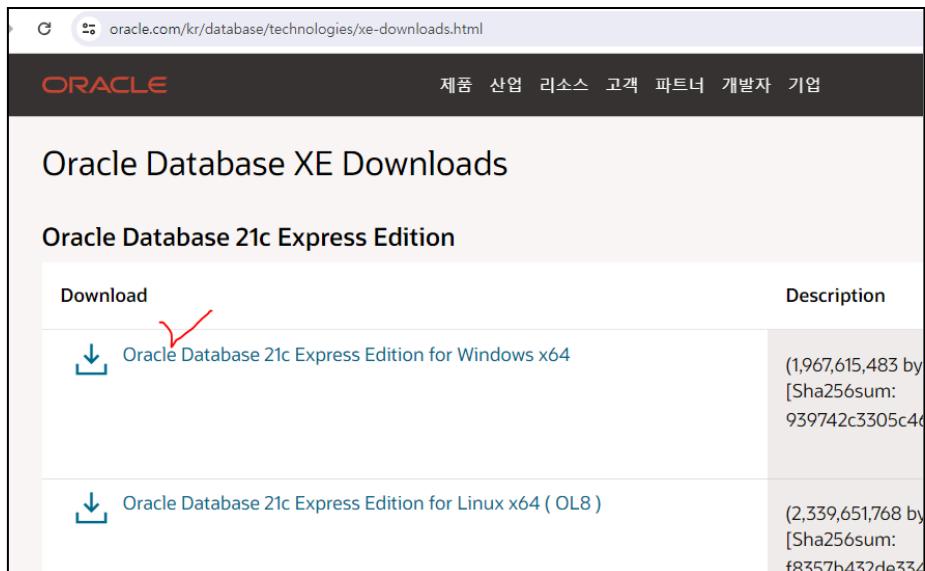
- 데이터베이스란?
- 파일시스템이란?
- 서버 클라이언트 란?
- 데이터베이스는 서버와 클라이언트 중 보통 어디에 설치되는가?
- 관계형 데이터베이스에 대해서 설명하시오.
- 객체 속성 로우 테이블 컬럼 `row` 각각에 대해 설명하시오.
- 테이블 조작할 때 사용되는 명령어 세계에 대해서 설명하시오.
- 테이블 안에 들어갈 입력데이터 조작 방법 네 가지 와 각각의 용도를 설명하시오.
- sql에 대해서 설명하시오.
- crud란?

## > 02. 오라클 설치 및 계정생성

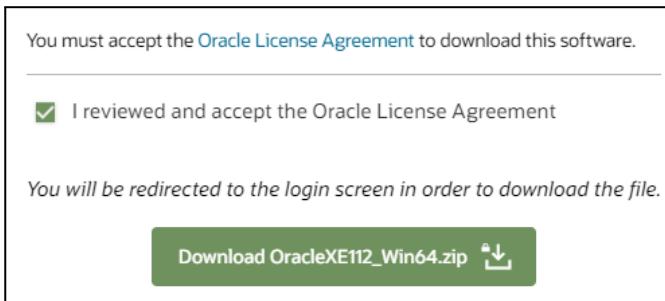
오라클 데이터베이스를 설치하여 보자. [www.oracle.com](http://www.oracle.com) 사이트에 들어가 상단  
돋보기 모양 아이콘을 클릭하고 검색부분에 Oracle Database express download 를  
입력하고 검색 하거나 크롬브라우저에서 검색하면 다음과 같이 체크된 부분의 경로를 얻을  
수 있다. 아래 이미지는 크롬 브라우저로 검색한 결과이다. 버전이 업 될때 마다 검색  
결과가 달라지니 잘 찾을 수 있는 방법을 선택하자.



시간이 지날때 마다 결과 화면들이 달라질수 있으니 잘 검색해 보자. 클릭하고 들어 가면  
다음과 같은 화면이 보이는데 windows x64환경을 클릭해서 다운로드 받을 수 있다.



최신 버전이 발매 되어 다를 수 있으나 책과 같은 버전이 존재 한다면 되도록 책과 같은  
버전을 이용하고 없을 때만 최신 버전을 설치해서 따라해 보자.



왼쪽 중간에 체크 박스를 클릭하고 하단에 download 버튼을 클릭하면 오라클 데이터베이스를 다운로드 받을 수 있다.  
회원가입과 로그인을 해야 가능하다.

로그인시 종종 문제가 발생하는데 문제 해결 방법은 로그인 시도 전 새로 고침하고 다시 로그인 시도를 천천히 여러번

시도해 보자. 로그인 후에도 다운로드가 잘 안된다면 브라우저를 모두 닫은 후 다시 시도해 보자. 로그인 성공 후에도 잘못된 페이지로 이동할 수 있다. 사이트 자체에 버그가 많다.

검색과 설치가 잘 안되면 웹에 oracle 21c로 검색해서 설치 방법을 확인하여 설치해 보자. 이미지가 18버전으로 되어 있는데 21c와 차이 없이 동일하다.



다운로드가 끝나고 압축을 풀면 setup.exe 파일이 있는데 클릭하면 오라클을 설치할 수 있다. 왼쪽 이미지 처럼 창이 뜨면 라디오 버튼(동의함)을 클릭하고 next 버튼을 눌러 진행을 계속한다. 이후 질문에 긍정적인 답변으로 next를 누르며 설치를 계속 이어 나가면 된다.

다음은 system 계정의 암호를 설정하는 창이다. 설치후 아이디 system 비밀번호 human 으로 로그인 할 수 있도록 설정 할 예정이다.

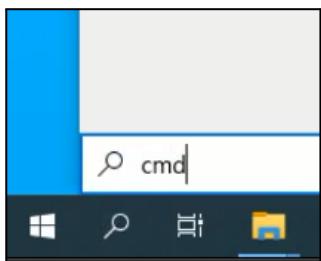


왼쪽 이미지는 관리자 계정의 암호를 입력하는 부분이다. human/human을 입력해서 암호를 human으로 설정하고 설치를 이어 나가자.

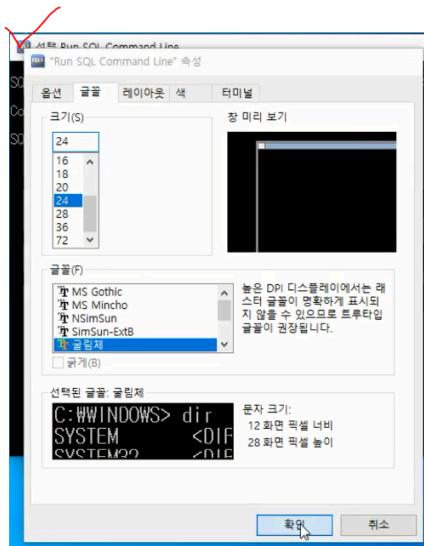
설치 중간에 browser 버튼을 눌러서 새로운 설치 경로를 지정 할 수 있다. 되도록 기본 위치에 설치하고 다른 드라이브에 설치 하려면 팝업으로 뜬 창에 설치할 폴더 위치를 기술하고 확인 버튼을 누르면 된다.

설치 중간에 네트워크 관련 팝업창이 뜨는데 반드시 허용하는 형태로 클릭 해야 이후 작업이

문제 없이 진행된다.



설치가 종료되고 윈도우키+R를 누르고 cmd를 입력하면 검정색 커맨드 창이 뜨는데 콘솔창이라고 부른다. sqlplus를 입력하고 엔터를 치면 우리가 설치한 데이터베이스 프로그램과 연결할 수 있는 sqlplus 프로그램이 실행된다. 윈도우 하단 돋보기 모양을 클릭하고 cmd를 입력하여도 된다.



왼쪽 이미지 처럼 빨간색 체크부분 왼쪽 상단 이미지를 클릭하고 속성을 선택하면 글꼴이나 폰트 크기를 설정할 수 있다. 변경후 cmd창을 닫은 다음에 다시 열어야 설정한 내용이 적용된다. 검정색 콘솔창에서 작업하는 일이 쉽지 않으면 메모장에 기술하였다가 복사 붙여넣기 해서 사용하자.

메모장에서 원하는 문자열을 작성 한 다음 shift+방향키로 작성한 문자열을 선택한 다음 ctrl+c(복사)로 선택한 문자열을 복사 한 다음 콘솔 창을 클릭한 후 ctrl+v(붙여넣기)하면 메모장에 작성한 문자열을 콘솔창에 붙여넣기 할 수 있다.

cmd 콘솔창에서 이전에 입력한 문자열을 다시 사용하고 싶다면 방향키를 이용하여 이전에 입력한 문자열을 찾을 수 있고 찾은 문자열을 방향키를 이용해서 편집할 수도 있다. 위쪽 방향키를 누르면 이전에 입력하여 실행한 문자열이 콘솔창에 생성되고 왼쪽 오른쪽 화살표를 이용하여 원하는 위치를 찾아가 수정할 수 있다.

sqlplus는 설치한 오라클 데이터베이스에 sql를 전달해서 데이터베이스를 조작하고 원하는 데이터를 사용할 수 있도록 제공해 주는 프로그램이다. 이 프로그램을 통해서 나중에 배울 sql로 데이터베이스에 테이블을 만들어 데이터를 저장하고 읽어올 수 있다.

sqlplus 자체가 데이터 베이스가 아니고 데이터베이스에게 사용자가 원하는 명령어를 편집해서 전달해주고 값을 받아 올 수 있는 db 관리 프로그램이다. 오라클이 설치 되면 오라클에 직접 접근 할 수 없고 관리 도구를 이용해서 접근해야 한다. sqlplus는 여러 관리 도구중 하나이다.

웹사이트에서 물건을 구매한다고 생각해 보자. 웹사이트 쇼핑몰에서 원하는 물건을 요청하면 해당 웹사이트에서 물류 창고에 있는 물건을 본인에게 전달해 준다. 여기서 물류 창고가 데이터베이스 서버에 해당되고 쇼핑몰이 sqlplus에 해당 된다.

sqlplus가 실행되면 사용자 아이디와 암호를 입력해 로그인 해야 데이터 베이스 오라클을 사용 할 수 있다. 아이디가 없거나 아이디와 암호를 모른다면 다음 아이디 없이 로그인 한 다음 아이디 계정을 만들 수 있는 권한인 sys 권한으로 로그인한 다음 새로운 계정을 만들어 새로운 계정을 사용할 수 있다. 다음을 따라서 계정을 만들어 데이터베이스를 사용해 보자.

1. 아래 처럼 명령 프롬프트가 실행된 상태에서 sqlplus /nolog를 입력하고 엔터를 치면 로그인 없이 sqlplus에 들어가 보자.

```
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\foxman12>sqlplus /nolog
```

```
Copyright (c) 1982, 2018
SQL> conn sys as sysdba
비밀번호 입력:
연결되었습니다.
SQL> show user;
USER은 "SYS"입니다
SQL>
```

2. sqlplus에 로그인 하지 않고 접속한 다음 왼쪽 처럼 conn sys as sysdba를 입력하면 sys계정으로 로그인할 수 있다. 중간에 password를 물으면 그냥 엔터를 치면 connected라는 메시지와 함께 로그인 된다. as sysdba를 통해서 이미 암호를 입력한 상태이기 때문이다. 제대로 로그인 되었는지 확인하고 싶다면 show user를 입력 하고 엔터를 치면 현재 유저가 sys라는 메시지를 확인할 수 있다. sys계정은 데이터 베이스 최상위 관리자 계정이다. 새로운 계정을 만들기 위해서 sys계정으로 로그인 하였다. 혹시 sys as sysdba로 로그인이 불가능 하다면 오라클 설치할 때 넣었던 암호가 system의 암호이다. 아이디 system 비번 human으로 sys대신 system계정을 사용해서 작업을 진행 할 수 있다.

3. sys계정으로 로그인 하였다면 create user c##다음에 원하는 계정명(c##human) identified by 암호(human)를 입력 하여 아이디 c##human, 암호 human인 계정을 만들어 보자. c##은 오라클 12g 버전이후 부터 반드시 넣도록 되어 있다. 모든 사용자 계정 앞에 c##를 붙여야 사용할 수 있다.

```
SQL> create user c##human identified by human;
```

같은 이름으로 하나의 계정만 만들 수 있다. 같은 이름으로 또 만들려고 하면 사용자 를 이름과 상충됩니다. 라는 에러 메시지가 나온다.

```
SQL> create user c##human identified by human;
create user c##human identified by human
*
1행에 오류:
ORA-01920: 사용자명 'C##HUMAN'(이)가 다른 사용자나 를 이름과 상충
됩니다
```

4. 정상적으로 작업을 하였는데 만든 계정으로 로그인을 하면 에러 메시지를 확인 할 수 있다. 로그인 할 수 없는 이유는 계정만 만들었을 뿐 사용 권한을 주지 않아서이다. c##human계정에 connect 로그인 연결을 할 수 있는 권한, resource 데이터베이스 자원을 사용할 수 있는 권한, dba 데이터베이스 관리 할 수 있는 권한을 부여하여야 한다.

grant를 사용하여 사용자에게 권한을 부여 할 수 있다

sys로 로그인 한 다음 grant 권한 to 계정을 이용해서 human계정에 권한을 부여하고 다시 로그인 하면 다음처럼 c##human으로 로그인 된것을 확인 할 수 있다.

grant connect,resource,dba to c##human; ,(쉼표) .(마침표) :(콜론) ;(세미콜론)이 잘 식별이 되지 않는다. 자세히 구분해 식별해 보자.

```
SQL> grant connect,resource,dba to c##human;  
권한이 부여되었습니다.
```

sys계정에서 왼쪽처럼 권한을 부여 해야 c##human 계정으로 로그인 할 수 있다.

```
SQL> conn  
사용자명 입력: c##human  
비밀번호 입력:  
연결되었습니다.▶  
SQL>
```

5. conn c##human/human으로 로그인 하여 보자. 성공적으로 계정이 만들어 진것을 확인할 수 있다. 한번에 입력할 수 있고 왼쪽처럼 conn 엔터 c##human 엔터 human처럼 엔터를 치면서 순서대로 하나씩 입력 하여도 된다.

```
SQL> conn human/human  
ERROR:  
ORA-01017: 사용자명/비밀번호가 부적합.  
  
경고: 이제는 ORACLE에 연결되어 있지 않습니다.  
SQL> show user:  
USER은 ""입니다
```

만약에 로그인에 실패하거나 명령어에 실패하면 자동 로그 아웃 된다. 입력을 잘못 하였다면 show user를 입력하여 결과를 확인해 보면 아까는 sys였는데 지금은 “ ”으로 로그아웃 되어 있는 것을 확인 할 수 있다. 로그인후 사용하다가 문제가 발생하면 자동으로 로그아웃 되니 정상적으로 실행한

명령어가 동작하지 않으면 show user를 사용하여 로그인 상태인지 확인해 로그아웃 상태이면 다시 로그인 해야 한다.

```
SQL> select * from tab;  
선택된 레코드가 없습니다.
```

6. 로그인이 완료되면 select \* from tab;를 입력하고 엔터를 쳐서 왼쪽과 같이 나오면 새로운 계정을 만들어 실행하는 작업이 완료된 것이다. select \* from tab;은 만든 테이블을 모두 출력하라는 내용이다. 계정을 새로 만들면 새로운 테이블이 없어서 레코드가 없다고 뜬것이다.

혹시 sql를 종료하고 싶다면 exit를 입력하면 된다.

콘솔창은 컴퓨터 환경에서 사용자와 시스템 간 상호작용을 할 수 있는 텍스트 기반의 인터페이스를 제공하는 창 또는 화면입니다. 명령프롬프트(cmd)와 sqlplus 같은 검은 형태 창을 콘솔창이라고 한다.

이전에 만든 계정의 데이터 베이스의 암호가 만기되어 로그인 할 수 없으면 alter user 계정명 identified by 새비밀번호로 비밀 번호를 변경하면 해결 된다.

ex) alter user c##hr identified by hr -- c##hr 계정의 비밀번호를 hr로 변경하였다.

계정에 문제가 발생하면

오라클을 설치하고 계정을 만들어 로그인하는 방법을 다음과 같이 정리하였다.

1. [www.oracle.com](http://www.oracle.com) 사이트에 방문해서 oracle ex를 다운로드 받아 설치한다.
2. cmd창에 sqlplus /nolog를 입력해서 로그인 없이 sqlplus에 접속한다.
3. 계정을 만들수 있는 권한이 있는 sys계정으로 로그인한다. conn sys as sysdba;
4. 만들고 싶은 계정을 생성한다. create user c##human identified by human;
5. 만든 계정에 권한을 설정한다. grant connect,resource,dba to c##human;
6. 만든 계정으로 로그인한다. conn c##human/human;
7. 로그인 되었는지 확인해본다. show user;
8. 정상적으로 작업이 되었는지 확인을 위해서 존재하는 테이블이 있는지 확인해 본다.  
select \* from tab;

### 연습문제

1. 본인 이름의 이니셜로 계정을 만드는 실습을해 보자.
  2. 로그인 없이 sqlplus에 접속하는 옵션은 무엇인가?
  3. 특정 계정에 권한 부여하는 명령어는 무엇인가?
  4. 새로운 계정으로 다시 로그인 할때 사용하는 명령어는 무엇인가?
  5. 테이블 만들 때 사용한 명령어와 계정 만들 때 사용하는 명령어는 동일하다. 계정을 삭제하고자 한다면 어떤 명령어를 사용해야 하는가?
  6. sqlplus와 oracle db와 관계를 설명해 보자.
- 명확하게 기억이 나지 않을 때 키워드를 웹에서 검색하는 습관을 기르자.

---

## > 03. 자료형

---

자료형은 자료의 형태를 의미한다. 현실세계의 데이터를 데이터베이스에 저장 하려면 자료형마다 정방 방법이 다르기 때문에 어떤 자료형을 저장 할 것인지 미리 알아야 데이터베이스에 문제 없이 데이터를 저장할 수 있다. 그래서, 테이블을 만들 때 특정 컬럼에 데이터가 어떤 자료 형인지 기술 해주어야 한다.

자료형이란? 현실세계의 데이터를 컴퓨터에 어떻게 저장 해야 효율적인지 미리 정해 놓은 약속된 자료 형태로 데이터 베이스 테이블을 만들때 선언하여 사용 한다. 컴퓨터에 저장 할 현실 세계의 대표적인 데이터 종류는 정수, 실수, 문자열 시간 등이 있다. 숫자는 크게 정수와 실수가 있는데 정수는 소수점이 없는 숫자이고 실수는 소수점이 있는 숫자이다. 문자열은 우리가 실생활에 사용하는 모든 기호를 글로 표현한 것을 의미 한다. 시간은 현실 세계의 시간을 의미한다.

오라클에서 숫자를 저장하고 싶으면 number을 사용하면 된다. number은 정수 실수 다 저장할 수 있는데 정수나 실수 38 자리 숫자를 저장할 수 있다. 예) 정수 9999 실수 9999.99 둘다 저장할 수 있다.

정수만 저장 하고 싶을 때 number(n)을 사용하면 된다. 실수는 저장할 수 없다. n의 의미는 최대 n 자리 정수를 저장할 수 있다는 의미이다. 예) number(3) -999 ~ +999사이의 정수를 저장 할 수 있다.

number(n,m)은 실수를 저장할 수 있다. 자세히 보면 n,m은 .점이 아니고 ,쉼표이다. 소수점 이하를 포함해서 총 자리 수는 n이고 그 중 소수점의 자리 수는 m 이라는 이야기다. 정수 및 실수를 넣을 수 있는데 3을 넣으면 3.0 실수로 들어간다. 예) number(5,2)는 -999.99~+999.99 까지 저장할 수 있다.

문자열을 저장하는 방법에는 nchar(4000) nvarchar2(2000)이 있다. char(8000) varchar(8000) 는 과거에 사용하던 방식이고 최근에는 nchar(4000) nvarchar2(2000)를 사용 한다. 괄호 안에 있는 숫자는 최대 저장할 수 있는 문자 개수를 의미한다. nchar(4000)과 같이 선언하면 2,000개의 문자를 저장할 수 있다는 의미이다. 최대 2천이어서 그 이상의 문자열을 저장하려면 다른 방법을 사용해야 한다.

nchar 와 nvarchar2의 차이는 nchar는 고정길이이고 nvarchar2는 가변길이 이다. 둘다 저장 가능한 크기를 2000으로 정했다면 2000보다 큰 데이터는 저장 할 수 없지만 작은 데이터에 대해서 고정 길이 nchar는 데이터 크기와 상관없이 저장공간이 2000 소모되고, 가변 길이 데이터는 2000이 아닌 실제 저장할 데이터 크기 만큼 저장 공간을 잡는다.

장단점을 살펴보면 고정 길이는 데이터 길이가 모두 같아 다음 데이터 위치가 명확하여 찾을 때 속력이 빨라지는 반면 저장 공간이 많이 필요하고 가변 길이는 찾을 때 속력이 떨어지는 반면에 저장 공간이 적게 사용 된다.

같은 크기 서류에서 원하는 서류를 찾는 것이 다른 크기 서류에서 원하는 문서를 찾는 것이 더 쉽다. nchar가 같은 크기서류 nvarchar2가 다른 크기 서류라 생각하면 된다. 모든 데이터 크기가 같으면 다음 데이터 위치를 쉽게 알 수 있어 검색 속력이 더 빨라진다. nchar는 그대신 저장공간이 많이 낭비된다. 실무에서 아이디 패스워드 같은 데이터 크기가 비슷하고 빠른 읽기 능력을 요구하니 nchar가 좋고 글내용은 내용의 크기가 일정치 않고 크기가 비교적 큰 데이터도 있으니 varchar2를 사용하는 것이 좋다.

시간을 저장하는 방법은 date가 있다. 내부적으로는 숫자로 저장되기 때문에 실제 데이터를 기술해서 저장하는 것이 불가능하여 시간 문자열을 to\_date()라는 메소드를 사용하여 데이터 베이스에서 사용할 수 있는 시간 데이터로 변경하여 DB에 저장하고 저장된 시간 데이터를 받아와 to\_char()메소드로 문자로 출력해야 한다.

속성	설명
nchar(n)	고정길이 문자 n개
nvarchar2(n)	가변길이 문자 n개 최대 2000까지 가능
number	+ - 38자리 정수와 실수
number(n)	n자리 정수
number(m,n)	총 m 자리인데 소수점이 n자리인 실수
date	날짜와 시간을 저장('YYYYMMDDHH24MISS') 현재시간: sysdate

데이터베이스에서 특정 자료형의 데이터를 표현하기 위해서 기술하는 방법을 확인해 보자. 사용자가 직접 다양한 자료형 데이터를 기술하여 표현한 것을 상수라고 한다.

데이터베이스에서 다양한 자료형 상수 표기법은 다음과 같다.

숫자를 기술 할때 아무 기호 없이 기술하면 된다. 1, 2, 3, 4, 2021, 14.23

```
select 1 from dual;
```

```
select 1,2,14.5 from dual;
```

문자열을 기술할 때 작은 따옴표로 묶어서 기술한다. '1', '2020', '안녕', 'hello'

```
select '문자열' from dual;
```

시간 데이터를 저장 하려면 사용자 입력 문자열을 가지고 시간 데이터로 변경하면 된다.  
이때 사용하는 것이 to\_date이고 사용방법은 다음과 같다.

```
to_date('1977:05:06 14:05:06', 'YYYY:MM:DD HH24:MI:SS')
```

```
to_date('1977/05/06 14-05-06', 'YYYY/MM/DD HH24-MI-SS')
```

```
select to_date('1977:05:06 14:05:06', 'YYYY:MM:DD HH24:MI:SS') from dual;
```

상위 코드는 데이터베이스 쿼리나 날짜 및 시간 조작을 지원하는 프로그래밍 언어에서 날짜 개체로의 변환을 위해 to\_date 함수관련 설명이다.

각 표현을 자세히 살펴보자.

to\_date('1977:05:06 14:05:06', 'YYYY:MM:DD HH24:MI:SS'):

'1977:05:06 14:05:06'은 입력 날짜 문자이고

형식 문자열 'YYYY:MM:DD HH24:MI:SS'은 입력 문자열의 예상 형식이다.

예상 형식에 맞춰서 입력 날짜 문자를 기술해서 원하는 시간 데이터를 얻을 수 있다.

YYYY: 네 자리의 연도 (예: 1977).

MM: 두 자리의 월 (예: 05는 5월).

DD: 두 자리의 일 (예: 06).

HH24: 24시간 형식의 두 자리 시간 (예: 오후 2시는 14).

MI: 두 자리의 분 (예: 05).

SS: 두 자리의 초 (예: 06).

제공된 날짜 문자열을 구문 분석하고 지정된 형식 문자열에 따라 연, 월, 일, 시, 분 및 초 정보로 변환하여 날짜 개체로 만드는 것입니다.

형식 문자열의 / 또는 -와 같은 문자는 입력 날짜 문자열에서 해당 문자와 일치하도록 사용되며 날짜 및 시간 구성 요소의 표현에 유연성을 제공합니다.

결국, 소괄호 안의 두 문자열을 비교하여 YYYY 부분에 년, MM 부분에 월, DD 일, HH24부분에 시간, MI부분에 분, SS부분에 초를 기술하고 이외의 문자인 : 나 스페이스는 동일하게 기술하면 된다. / 이면 /로 - 이면 -으로 기술하면 된다.

시간 데이터를 원하는 형태의 문자열로 출력 하고자 할 때 to\_char를 사용한다. 원래는 숫자 형태로 저장되어 있어서 해당 데이터를 출력하면 숫자가 나와야 하지만 내부적으로 자동으로 시간 문자열로 처리되게 되어 있다. 만약 본인이 원하는 형태로 시간 객체를 출력하고 싶다면 좀더 복잡한 형태로 출력하면 된다.

`sysdate`는 데이터베이스에서 제공하는 현재 시간을 저장하고 있는 시간 데이터이다.  
`select sysdate from dual;`이라 입력하면 현재 시간이 출력되는 것을 확인할 수 있다.  
`dual`테이블은 어떤 실행 결과를 화면에 찍을때 오라클에서 제공하는 테이블 이다. 화면에  
출력하고자 하는 내용이 있다면 ‘`select` 출력하고자하는내용 `from dual;`’과 같이 기술  
하면 된다.

`select 5+6 from dual; -- 5+6한 결과를 화면에 찍기`

원하는 시간 문자열로 출력하고 싶으면 다음과 같이 하면 된다.

```
select to_char(sysdate, 'YYYY/MM:dd HH24:MI:SS') from dual;
```

`TO_CHAR(SYSDATE, 'YYYY/MM:dd HH24:MI:SS')`: `TO_CHAR` 함수는 날짜나 숫자를 지정된  
형식의 문자열로 변환합니다. 여기서 `SYSDATE`는 현재 시스템 날짜와 시간을 나타냅니다.  
`'YYYY/MM:dd HH24:MI:SS'`는 날짜 및 시간 형식을 지정하는 형식 문자열이다.

```
SQL> select '안녕하세요' from dual;
'안녕하세요'
-----
안녕하세요
SQL> select sysdate from dual;
SYSDATE
-----
21/10/21
```

`sysdate`라는 현재 시간 데이터를 다음에 오는  
형태의 문자열로 변경해서 출력되는 것을 확인 할  
수 있다. `dual`은 오라클이 제공해 주는 데이터  
1개 들어 있는 테이블이다. `select` 다음에 오는  
문자열을 출력하고 싶을때 사용하는 오라클에서  
지원해 주고는 테이블이다.

다음을 실습해 보자.

아래 처럼시간 객체를 만들어서 찍어 보면 `77/05/06` 처럼 시분초가 잘려서 나오는데 시간  
객체를 오라클에서 기본적으로 연월일만 출력되게 되어 있어서 그런 거지 실제로는 시분초도  
가지고 있다. 원하는 시간 모양을 출력하고 싶다면 다음 예제 처럼 `to_char`를 사용하여  
문자열을 만들어 출력해야 한다.

```
SQL> select to_date('1977:05:06 14:05:06', 'YYYY:MM:DD HH24:MI:SS') from dual;
TO_DATE(
-----
77/05/06
```

```
SQL> select to_char(sysdate, 'YYYY:MM:DD HH24:MI:SS') from dual;
TO_CHAR(SYSDATE, 'YYYY:MM:DD HH24:MI:SS')
-----
2021:10:21 11:14:32
```

연습문제

1. 데이터베이스에서 자료형을 사용하는 이유는 무엇인가?
2. 현실 세계의 다양한 데이터 정수, 실수, 문자열, 시간을 데이터베이스에 저장 할 때 어떤 자료형으로 선언 하는지 설명하시오.
3. 4 가지 자료형을 오라클에서 사용하는 상수로 기술 해 보자.
4. `select` 문을 이용해서 4가지 자료형을 출력해 보자.
5. ‘YYYYMMDDHH24MISS’문자열에서 년월일시분초를 표현하기 문자열을 뽑아보자.
6. `to_date`, `to_char` 용도와 사용 방법을 설명 하시오.

---

## > 04. 테이블 만들어 조작해 보기

---

기본적으로 sql은 대소문자를 구분하지 않지만 테이블에 들어갈 데이터는 대소문자를 구분한다.

SQL은 "Structured Query Language"의 약자로, 데이터베이스에서 데이터를 관리하기 위한 표준화된 언어입니다. SQL은 관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터를 정의, 조작, 제어하는 데 사용됩니다. RDBMS는 데이터를 테이블 형태로 저장하고 관리하는 시스템을 의미합니다.

SQL은 다음과 같은 주요 작업을 수행할 수 있는 명령어로 구성되어 있습니다:

데이터 정의 언어 (DDL - Data Definition Language):

데이터베이스의 구조를 정의하는 명령어로, 테이블, 인덱스, 뷰 등을 생성, 수정 또는 삭제합니다.

주요 명령어: CREATE, ALTER, DROP.

데이터 조작 언어 (DML - Data Manipulation Language):

데이터를 쿼리하고 조작하는 명령어로, 테이블에 데이터를 삽입, 수정, 삭제하고 조회합니다.

주요 명령어: SELECT, INSERT, UPDATE, DELETE.

데이터 제어 언어 (DCL - Data Control Language):

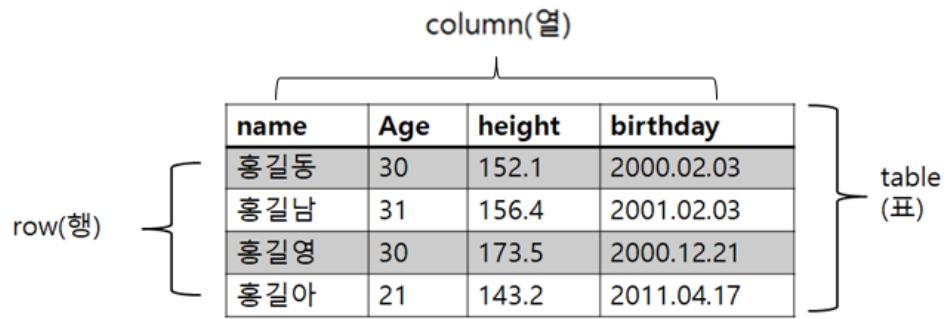
데이터베이스의 접근 권한을 관리하는 명령어로, 사용자에게 권한을 부여하거나 회수합니다.

주요 명령어: GRANT, REVOKE.

SQL은 다양한 데이터베이스 시스템에서 사용되며, 대표적으로 Oracle, MySQL, PostgreSQL, Microsoft SQL Server 등이 있습니다. SQL을 사용하면 데이터베이스와 상호 작용하여 데이터를 효과적으로 관리하고 조회할 수 있습니다.

sql 질의어를 이용하여 다음 페이지에 있는 테이블을 조작해 보면서 이해해 보자.

sqlplus에서 조작하기 어려울 수 있으니 메모장에서 작업한 다음 **ctrl+c**로 복사한 다음 콘솔창에서 **ctrl+v**를 이용해서 붙여 넣기 해서 사용해 보자.



## 테이블 만들기(create)

테이블명은 식별자여서 같은 이름이 있거나 sql에서 사용하는 키워드를 사용하면 안된다.

```
CREATE TABLE 테이블명 (
    컬럼1명 자료형1,
    컬럼2명 자료형2,
    컬럼3명 자료형3,
    -- 추가 컬럼들
);
```

테이블명: 생성할 테이블의 이름을 나타냅니다. 테이블명은 중복되지 않는 고유한 이름이어야 합니다.

컬럼1명, 컬럼2명, 컬럼3명: 테이블에 포함될 각 열의 이름을 나타냅니다. 열의 이름은 해당 테이블 내에서 고유해야 합니다.

자료형1, 자료형2, 자료형3: 각 컬럼의 데이터 형식을 나타냅니다. 숫자, 문자열, 날짜 등 다양한 자료형을 사용할 수 있습니다.

예를 들어, 실제로 사용할 수 있는 예제는 다음과 같습니다:

```
CREATE TABLE employees (
    employee_id NUMBER(6),
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    hire_date DATE,
    salary NUMBER(10, 2)
);
```

위의 예제에서는 "employees"라는 테이블을 생성하며, 다섯 개의 열을 가지고 있습니다. 이 열들은 각각 사원 ID, 이름, 성, 입사일, 급여를 나타내며 각각의 데이터 형식에 맞게 정의되어 있습니다.

상위 이미지의 테이블명은 사람 정보를 저장할 예정이므로 human으로 하자. 다음 sql을 sqlplus에 출력해서 결과를 확인해 보자.

```

create table human(
    name nvarchar2(30),
    age number(3),
    height number(4,1),
    birthday date
);

--현재 데이터베이스에 있는 테이블정보 출력하여 방금 만든 테이블을 확인해 보자. 내가 만들지 않은 테이블이 있을 수 있는데 시스템이 관리하는 테이블이라고 생각하고 무시하자.

select * from tab;

desc human;           --human 테이블 상세 내용 출력

```

### 데이터 입력하기(insert)

`INSERT INTO` 문은 데이터베이스 테이블에 새로운 레코드(행, row)를 추가하는 데 사용됩니다. 다음은 `INSERT INTO` 문의 일반적인 구조입니다

`INSERT INTO` 테이블명 (컬럼1, 컬럼2, 컬럼3, ...)

`VALUES (값1, 값2, 값3, ...);`

테이블명: 데이터를 삽입할 대상 테이블의 이름

(컬럼1, 컬럼2, 컬럼3, ...): 데이터를 삽입할 테이블의 컬럼 목록. 중괄호 안에 여러 컬럼을 콤마로 하여 기술 한다.

`VALUES (값1, 값2, 값3, ...)`: 각 컬럼에 삽입될 실제 값들의 목록입니다. 이 목록도 괄호 안에 값을 콤마로 구분한다.

예제:

```

INSERT INTO employees (employee_id, first_name, last_name, hire_date, salary)

VALUES (1, 'John', 'Doe', '2022-01-06', 50000.00);

```

위의 예제에서는 "employees" 테이블에 새로운 레코드를 추가하고 있습니다. 각 컬럼에는 새로운 직원의 정보가 들어가게 됩니다. 이 때, 데이터 유형에 주의하여 값을 삽입해야 합니다. 날짜 형식이나 숫자 형식 등을 잘 맞춰주어야 합니다.

다음은 human 테이블에 데이터를 넣는 방법이다.

```
insert into human(name,age,height,birthday) values ('홍길동', 30, 152.1,  
to_date('2000:02:03 00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

- human(name,age,height,birthday) 테이블에 전체데이터를 넣는다면 다음처럼 생략이 가능하다.

```
insert into human values ('홍길남', 31, 156.4, to_date('2001:02:03 00:00:00',  
'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길영', 30, 173.5, to_date('2000:12:21 00:00:00',  
'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길아', 21, 143.2, to_date('2011:04:17 00:00:00',  
'YYYY:MM:DD HH24:MI:SS'));
```

commit; --모든 insert 작업이 끝나면 반드시 commit를 실행해 주어야 한다. 일단은 모든 작업을 적용시키고 마무리 한다는 의미이고, 일단 그냥 써주어야 한다고 생각하면 된다.

## 데이터 검색하기(select)

SELECT 문을 사용하여 테이블에서 원하는 정보를 검색하는 작업을 한다.

사용 방법은 select 다음에 검색 하고자 하는 컬럼명을 기술하고 form 다음에 검색할 테이블 이름을 기술하고 where절에 검색 조건을 기술하면 해당 테이블의 원하는 데이터가 출력 된다.

모든 컬럼을 출력하고 싶다면 \*을 사용하면 된다. 특정 컬럼만 출력하고 싶다면 컬럼명 위치에다 출력 하고자 하는 컬럼명을 ,로 연결하여 기술하면 된다. 전체 row(행) 데이터가 아닌 특정 row(행) 데이터만 출력하고 싶다면 where절을 사용한다.

-- 테이블의 모든 컬럼을 조회

```
SELECT * FROM human;
```

-- 특정 컬럼들만 선택하여 조회

```
SELECT 컬럼1, 컬럼2, 컬럼3
```

```
FROM 테이블명;
```

-- WHERE 절을 사용하여 특정 데이터만 조회

```
SELECT * FROM 테이블명 WHERE 조건식;
```

-- and나 or로 연결하여 여러 조건을 조합하여 조회할 수 있다.

```
SELECT * FROM 테이블명 WHERE 조건1 AND/OR 조건2;
```

컬럼1, 컬럼2, 컬럼3: 조회하고자 하는 특정 컬럼들을 ,로 연결해서 나열한다.

\* : 모든 컬럼을 출력하고자 할때 사용한다.

WHERE: 특정 조건을 지정하여 특정 데이터만을 조회할 수 있습니다. 예를 들어, WHERE salary > 50000은 급여가 50000 이상인 데이터만을 조회합니다.

AND, OR: 여러 조건을 결합하여 사용할 수 있습니다.

```
WHERE salary>50000 and first_name='John'; --두 조건 모두 만족하면 데이터가 출력
```

```
WHERE salary>50000 or first_name='John'; --둘 중 하나의 조건을 만족하면 데이터 출력
```

다양한 예제를 확인해 보자.

-- "employees" 테이블에서 "John"이라는 이름을 가진 직원의 모든 정보를 조회

```
SELECT * FROM employees WHERE first_name = 'John';
```

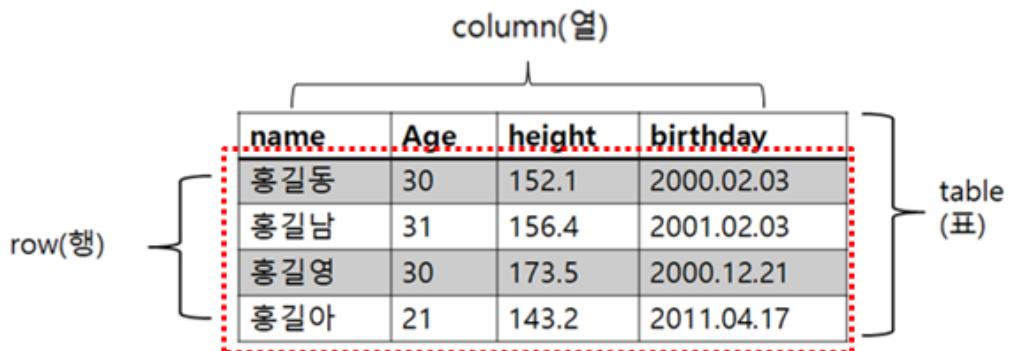
위의 예제에서는 "employees" 테이블에서 "John"이라는 이름을 가진 직원의 모든 정보를 조회하고 있습니다.

-- "employees" 테이블에서 이름과 급여 정보만 조회

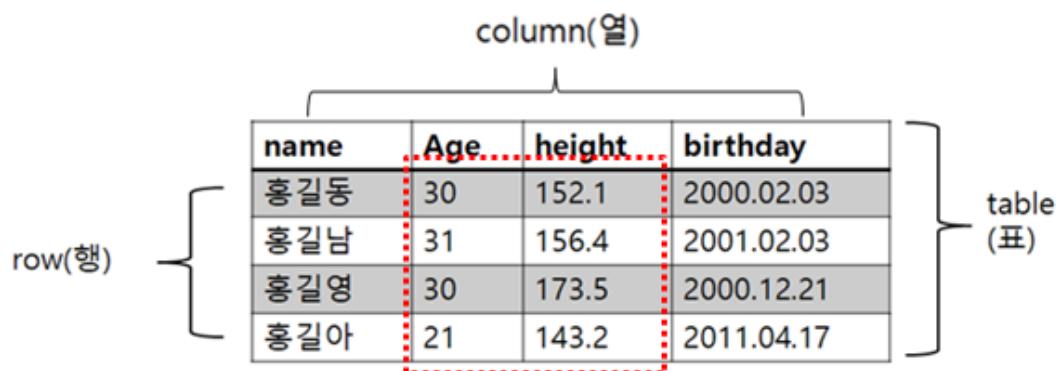
```
SELECT first_name, last_name, salary  
FROM employees;
```

하나의 데이터가 한줄에 출력 할 수 없다면 다음과 같이 한줄에 출력하는 글자수를 늘려 보기 편하게 만들 수 있다.

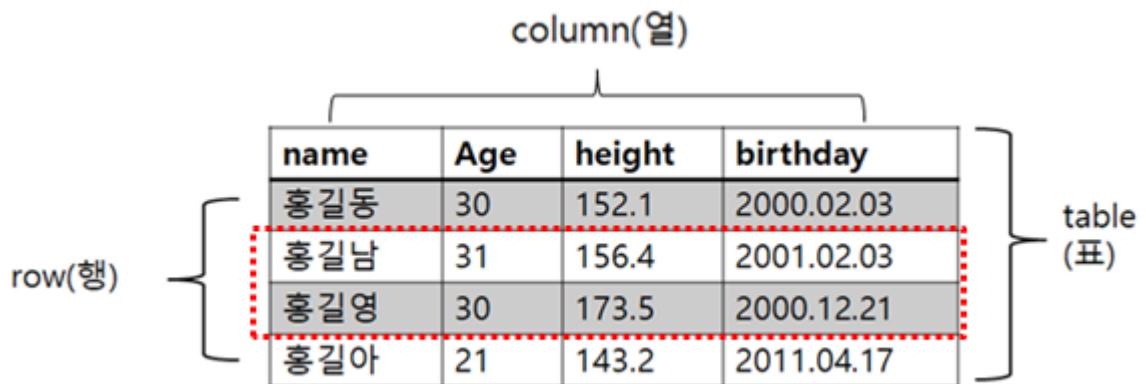
```
set linesize 180 --과 같이 화면 크기를 적절히 조절해서 보기 편하게 만들 수 있다.
```



- 모두 출력하고 싶다면 select 문옆에 \*를 기술한다. `select * from human`

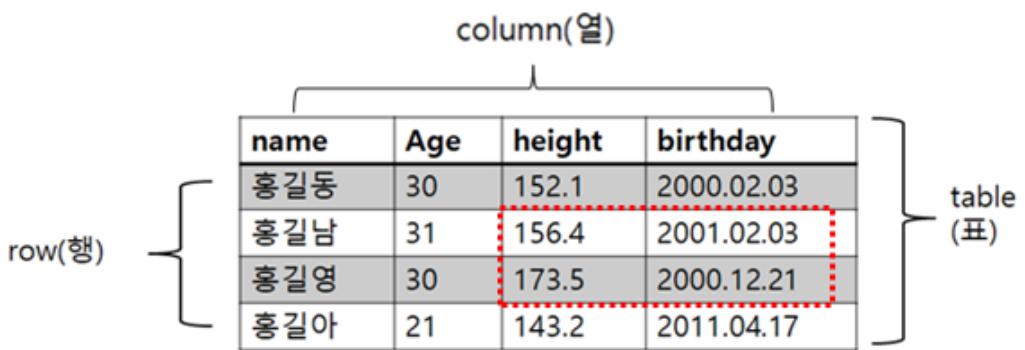


- 특정 컬럼의 데이터만 출력하고 싶다면 select 옆에 출력하고자 하는 컬럼을 ,로 연결하여 출력한다. `select age,height from human`



- 특정 row만 출력하고 싶다면 where 절을 사용한다.

```
select * from human where height > 155;
```



- 특정 컬럼의 특정 row를 출력하고 싶다면 select 다음 특정 컬럼을 ,로 연결하여 기술한 다음 where 절을 사용하여 원하는 row를 선택하여야 한다.

```
select height,birthday from human where height > 155;
```

다음을 확인해 보자.

이름과 나이만 출력하고 싶다면 다음과 같이 기술하면 된다.

```
select name,age from human;
```

생일만 출력하고 싶다면 select birth from human; 과 같이 기술하면 된다.

원하는 데이터만(row)을 출력하고 싶다면 where 절을 사용한다.

홍길남만 출력하고 싶을때 select \* from human where name='홍길남';

나이가 25보다 큰 사람들의 정보를 출력하고 싶다면 다음과 같이 하면 된다.

```
select * from human where age>25;
```

키가 150보다 작은 사람의 이름과 생일을 출력해 보자.

```
select name,birthday from human where height<150;
```

## 데이터 변경하기(update)

UPDATE 문은 데이터베이스 테이블에서 기존의 레코드를 수정할 때 사용됩니다. 기본 구조는 다음과 같다

UPDATE 테이블명

SET 컬럼1 = 변경값1, 컬럼2 = 변경값2, ...

WHERE 조건;

테이블명: 수정하고자 하는 테이블의 이름을 나타냅니다.

SET: 수정할 컬럼과 해당하는 값을 설정하는 키워드입니다.

컬럼1 = 변경값1, 컬럼2 = 변경값2, ...: 변경하고자 하는 각 컬럼과 그에 해당하는 새로운 값을 설정합니다.

WHERE: 어떤 레코드를 수정할 것인지를 결정하는 조건을 지정하는 키워드입니다. 이 부분이 없으면 테이블의 모든 레코드가 변경됩니다.

예를 들어, "employees" 테이블에서 사원의 급여를 60000으로 변경하고자 할 때:

```
UPDATE employees SET salary = 60000 WHERE employee_id = 1;
```

이렇게 하면 "employees" 테이블에서 employee\_id가 1인 사원의 급여가 60000으로 변경됩니다.

또한 ,로 여러 개의 컬럼을 동시에 변경할 수 있으며, WHERE 절을 사용하여 특정 조건을 만족하는 레코드만을 수정할 수 있습니다. 이를 통해 정확한 대상 레코드를 지정하여 원하는 부분만 수정할 수 있습니다. where절을 사용하지 않으면 선택된 테이블의 모든 데이터가 변경된다.

human 테이블의 age를 100으로 변경해보자.

```
update human set age=100; -- human테이블의 모든 데이터가 100으로 변경됨  
select * from human;를 실행 시키면 변경된 모든 데이터를 확인 할 수 있다. 특정  
데이터만 변경하고 싶다면 where절을 사용하면 된다.
```

rollback; --commit;은 작업한 내용 적용 rollback; 작업한 내용을 취소한다. rollback을 입력하여 작업을 취소하자.

모든 age컬럼의 값이 100으로 변경된 상태에서 commit;를 하면 적용되지만 rollback;를 통해서 age 컬럼을 100으로 변경하기 전으로 복구할 수 있다.

rollback; 후 select \* from human;를 실행시켜 원래 데이터로 복구 되었는지 확인해 보자.

만약 rollback;전에 commit;를 하고 나면 데이터가 모두 적용되어 rollback;를 하더라도 age가 100일때 이전 데이터로 돌아 갈 수 없다. rollback;은 이전 commit;를 기준으로 이전 데이터로 복귀 한다.

특정 row만 변경하고 싶으면 where절을 사용해야 한다.

홍길남의 나이를 100으로 변경해 보자.

```
update human set age=100 where name= '홍길남';
```

홍길아의 나이와 키를 60,170으로 변경해 보자.

```
update human set age=60,height=170 where name = '홍길아';
```

작업을 취소 하려면 rollback; 적용하려면 commit;를 입력하자.

sqlplus에서 제대로 변경되고 있는지 select문을 이용하여 그때 그때 확인해 보자.

## 삭제하기(delete)

원하는 행(row)을 삭제할 때 사용한다. `delete from` 다음에 삭제할 데이터가 들어 있는 테이블명을 기입하면 테이블의 모든 데이터가 삭제된다. 특정 데이터만 삭제하고 싶다면 `where` 절을 사용한다.

`DELETE` 문의 기본 구조는 다음 2개의 예제와 같다

```
DELETE FROM 테이블명 WHERE 삭제조건;
```

```
DELETE 테이블명 WHERE 삭제조건; -- from를 생략 할 수 있다.
```

테이블명: 삭제 하고자 하는 테이블의 이름입니다.

WHERE: 어떤 레코드를 삭제할 것인지를 결정하는 조건을 지정하는 키워드입니다. 이 부분이 없으면 테이블의 모든 레코드가 삭제됩니다.

예를 들어, "human" 테이블에서 모든 데이터를 삭제하고자 할 때:

```
DELETE FROM human; 또는 DELETE human;
```

예를 들어, "human" 테이블에서 이름이 "John"인 행을 삭제하고자 한다면:

```
DELETE FROM human WHERE 이름 = 'John';
```

`DELETE` 문은 사용할 때 신중하게 사용해야 합니다. `WHERE` 절을 사용하지 않으면 테이블의 모든 레코드가 삭제되므로 주의가 필요합니다.

human 테이블에 모든 데이터를 삭제해 보자.

```
delete human; -- 혹은 delete from human;
```

`select * from human;`를 통해서 모든 데이터가 삭제되었는지 확인해 보자. 확인이 완료되었으면 `rollback;`를 입력하여 이전 데이터로 돌아가 보자. `rollback;`은 작업 내용을 취소하여 이전 데이터로 되돌릴 때 사용한다. 제대로 `rollback` 되었는지 `select` 문을 이용해서 확인해 보자. `where` 절에 삭제할 데이터를 선택하면 선택된 데이터가 삭제된다.

나이가 30보다 큰 사람의 데이터를 삭제해 보자. `delete human where age>30;`

데이터 관련 작업이 마무리되면 반드시 `commit;`이나 `rollback;` 중 하나를 선택해서 실행해야

한다. 적용하고 싶으면 `commit;` 취소하고 싶으면 `rollback;`를 실행하면 된다.

## 테이블 삭제

`delete`는 테이블에 들어 있는 데이터를 삭제하는 것이고 테이블 삭제는 테이블 자체를 지우는 것이다.

테이블을 삭제하고 싶다면 `drop table` 다음에 삭제하고자 하는 테이블 이름을 기술 하면 된다. 테이블이 삭제되면 당연히 해당 테이블의 데이터는 접근 할 수 없다.

`drop table 테이블명;` --데이터 베이스에서 해당 테이블을 삭제한다.

`drop table human;`

`select * from tab;` -- human테이블이 삭제되어 존재하지 않는다.

`rollback`은 `insert, update, delete`에서만 쓸수 있다 테이블을 `drop`하면 롤백할 수 없다.

SQL 주석은 SQL 문장에서 설명이나 메모를 추가하는 데 사용됩니다. 주석은 SQL 엔진에 의해 무시되며 쿼리 실행에 영향을 미치지 않습니다. SQL 주석은 코드를 이해하기 쉽게 만들고 코드의 일부를 임시로 비활성화하는 데 유용합니다.

-- 이것은 한 줄 주석입니다.

/\*

이것은 다중 줄

주석입니다.

\*/

null이란? 데이터가 없을때 데이터가 없다는 의미로 `null`를 사용한다.

`insert into human values ('', null, 0, NULL);`

`null`, `NULL`, `''`, 은 `null`을 의미 한다. ,, 처럼 ,, 사이에 값을 기술해야 할때 아무것도 기술하지 않으면 `null`로 처리되는 데이터베이스도 있지만 오라클에서는 `null`로 처리하지 않는다.

`null`은 데이터가 없음을 의미하고 ‘`null`’, ‘없음’, `0` 은 데이터가 없다는 의미가 아니고 문자열 `null`, 없음, 숫자 `0` 데이터가 있다는 의미이다 헷갈리지 않도록 조심하자.

테이블명이나 컬럼명에 영문을 제외한 다른 것은 사용하지 않는 것이 좋다.

2개이상의 단어로 테이블이나 컬럼을 만들때에는 \_를 사용한다. mybag인경우 my\_bag으로 이름을 짓는다. 이런 방식을 뱀이 지나가는 것 같다고 해서 스네이크 케이스라고 한다.

문제1) 다음중 잘못된 부분을 찾아서 올바르게 변경해보자.

```
insert into human values ('홍길남',31,156.4,to-date(2001:02:03  
00:00:00','YYYY:MM:DD HH24:MI:SS'));
```

문제2) drop table human; 와 같은 sql 작업후 rollback; 가능한가?

rollback은 insert update delete 같은 데이터를 조작할때 사용 가능하다.

문제3) 다음 데이터를 저장할 수 있는 테이블을 만들고 CRUDE 작업 가능한 sql를 만들어 보자. 테이블, 컬럼명을 한글이 아닌 영어로 만들어 사용하자.

테이블명: 화학물질 재고 현황

화학물질명	재고량(kg)	최소재고량(kg)	입고일자
아세트산	500	100	2023-03-15
수산화나트륨	300	50	2023-04-20
황산	200	80	2023-05-10
염화칼슘	400	120	2023-06-05
염산	600	200	2023-07-08

문제4) 다음 데이터를 저장할 수 있는 테이블을 만들고 CRUDE 작업 가능한 sql를 만들어 보자.

테이블명: 도서 대출 기록

도서명	저자	대출일	반납예정일	실제반납일	연체료(원)
죄와 벌	톨스토이	2023-03-01	2023-03-15	2023-03-20	500
해리포터와 마법사의 돌	J.K.롤링	2023-04-10	2023-04-24	2023-04-23	0

반지의 제왕	J.R.R.톨킨	2023-05-15	2023-05-29	2023-05-29	0
빨간 머리 앤	L.M.몽고메리	2023-06-20	2023-07-04	-	-
산삼	박완서	2023-07-10	2023-07-24	-	-
-	-	-	-	-	-

문제5) 다음을 테이블로 만들어 보자.

테이블명: 고객 주문 기록

주문번호	고객명	주문일	상품명	수량	상품가격(원)	주문금액(원)
2023001	홍길동	2023-03-05	노트북	1	1,200,000	1,200,000
2023002	김영희	2023-03-08	스마트폰	2	800,000	1,600,000
2023003	박철수	2023-03-15	헤드폰	3	150,000	450,000
2023004	이지훈	2023-03-20	마우스	1	30,000	30,000
2023005	김민지	2023-03-25	키보드	1	50,000	50,000

문제6) 다음 이미지를 보고 데이터 베이스를 만들어 보자. 힌트는 새로운 데이터로 이미지의 내용을 변경하였을때 바뀌어야 하는 부분과 바뀌지 않는 부분을 확인하여 바뀌는 부분을 데이터 베이스로 만들면된다.



문제7) 다음 이미지를 보고 데이터 베이스를 만들어 보자.

## 앨범 정보



[EP]  
NewJeans 2nd EP 'Get Up'  
NewJeans

발매일 2023.07.21 ★★★★☆ 4.6 5,497명 >  
장르 댄스 댓글 8,808개 >  
발매사 YG PLUS 공유  
기획사 ADOR

♡ 35,253 └ 앨범다운 > └ FLAC앨범다운 > └ 선물하기 >

문제8) 우리 주변의 객체중 우리가 배운 자료형을 모두 사용하는 속성을 가진 객체를 선택해서 테이블을 만들어 데이터를 넣어 보자.

## > 05. sql developer 설치하기

SQL Developer는 SQLPlus와 같은 작업을 하는 프로그램인데 GUI 그래픽 기반의 프로그램으로, 직관적이고 시각적으로 데이터베이스와 상호 작용할 수 있는 도구입니다.

sql developer 를 다운로드하여 프로그램을 설치해보고 sqlplus와 sql developer를 이용하여 scott계정과 hr계정을 사용 할 수 있도록 세팅해 보자.



[www.oracle.com](http://www.oracle.com)에 방문해서 sql developer download 를 검색어로 넣고 검색한 결과를 아래로 드래그 하면 sql Developer를 다운로드 받을 수 있는 링크가 제공 된다.

다음 이미지 처럼 제공 되는 링크를 클릭해서 들어가면 sql developer를 다운로드 받을 수 있는 버튼을 확인 할 수 있다.

Oracle SQL Developer Downloads  
<https://www.oracle.com/database/sqldeveloper/technologies/download/>  
Oracle SQL Developer Downloads



링크를 클릭해 들어가 본인에 맞는 sql developer 를 다운로드해 보자.

우리가 필요한 제품은 windows 64 JDK 11 included 버전이다. 해당 제품 download 버튼을 눌러 다운로드 하자.

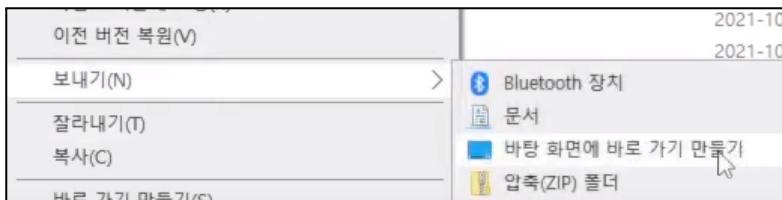
다운로드 완료후 완료된 파일에 마우스를 올려놓고

마우스 오른쪽 버튼을 눌러 압축 풀기를 하면 다음과 같은 폴더안에 파일을 확인 할 수 있다.

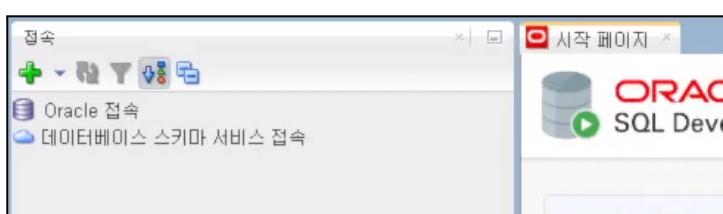


실행방법은 압축 품 폴더안의 `sqldeveloper.exe`를 클릭하여 실행 시키면 `sqldeveloper`를 실행시킬 수 있다. 실행 도중에 나오는 메시지 박스들을 계속 진행시키면 문제 없이 실행이 될 것이다. 바로 가기를 바탕 화면에 만들고 사용하자.

`sqldeveloper.exe` 파일 위에서 마우스 오른쪽 클릭 >> 보내기 >> 바탕화면에 바로가기 만들기를 클릭하여 바로가기 아이콘을 바탕화면에 만들어 사용하자.



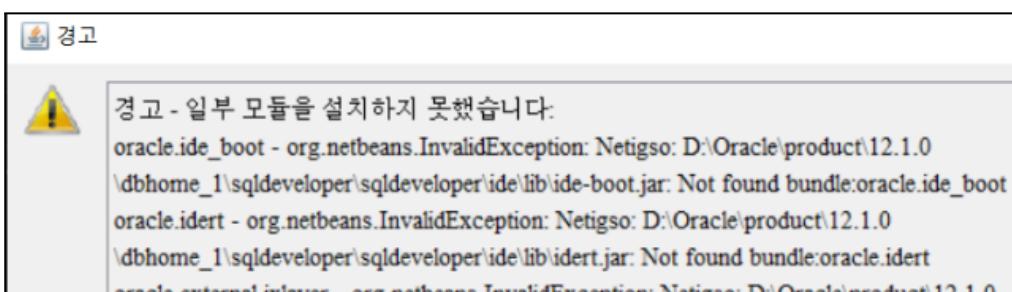
아이콘을 더블 클릭하여 실행하고 나면 다음 이미지 처럼 접속창이 보일 것이다. 접속창이 없다면 메뉴에 보기>>접속을 클릭하면 접속창이 열린다.



접속창 상단에 녹색 + 버튼을 누르면 오라클 데이터베이스에 접근할 수 있는 계정 등록하는 새로 만들기 창이 뜬다.

만약 실행시 다음과 같은 화면이 나오고 실행되지 않을 때는

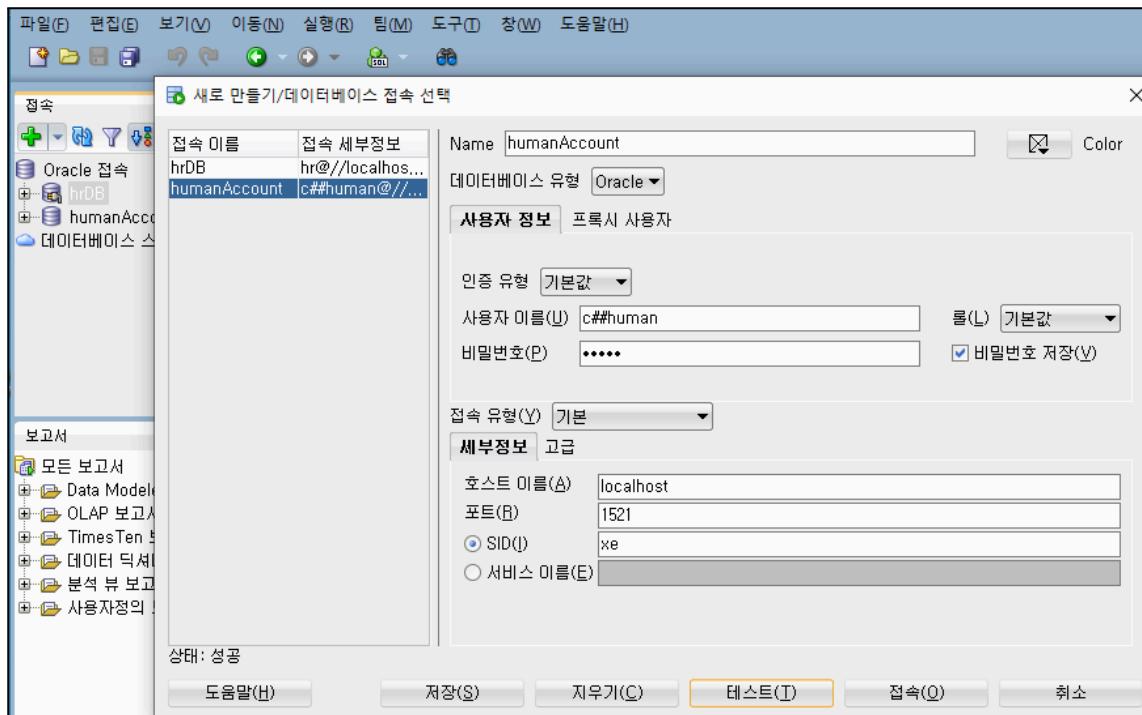
`C:\Users\사용자명\AppData\Roaming\SQL Developer` 폴더 삭제 및 하위 파일을 제거하고 다시 실행해 보자.



다음 페이지에 있는 이미지 처럼 새로 만들기 창이 뜨면 ‘human 계정 정보’를 name, 사용자이름, 비밀번호, 비밀번호저장에 입력 한 다음 접속 버튼을 눌러서 데이터베이스 human 계정 사용자를 등록 할 수 있다.

이전에 `sqlplus`에서 만든 `human`계정을 기반으로 `sql developer`에 등록 하는 예제를 따라해 보면 다음과 같다. 입력 부분 Name은 프로그램에서 식별 할때 사용 할 식별자를 넣는다. 보통 계정이랑 같은 이름으로 만든다. 여기서는 `humanAccount`로 만들었고 이름과 비밀번호를 `c##human/human`으로 넣는다. 비밀번호 저장을 체크한다. 호스트이름에는 `oracle`이 설치된 ip주소를 입력한다. 현재 같은 컴퓨터에 `oracle`이 설치되어 있다면

localhost를 입력하고 포트는 기본으로 설치 하였다면 1521이고 변경하였다면 변경한 포트번호를 입력하면 된다. sid는 본인이 변경하지 않았다면 xe이고 2개 이상 설치 하거나 변경 하였다면 변경한 이름을 넣자. 하단 테스트 버튼을 눌러 확인해 보면 문제가 없으면 성공이라는 문자열이 왼쪽 하단 부분 상태창 부분에 출력 된다. 문제가 발생해서 에러 메시지가 생기면 해당 에러 메시지를 웹에 검색해서 문제점을 확인하여 문제를 없애 보자. 모두 완료되었다면 접속 버튼을 누르자.



#### 호스트 이름 (Host Name):

호스트 이름은 컴퓨터나 장치를 식별하는 데 사용되는 읽기 쉬운 문자열입니다.

예를 들어, "www.example.com"은 웹 서버의 호스트 이름입니다.

#### IP 주소 (Internet Protocol Address):

IP 주소는 컴퓨터나 네트워크 장치를 고유하게 식별하는 숫자입니다.

IPv4 주소는 보통 "xxx.xxx.xxx.xxx" 형식으로 표현되며, 각 부분은 0부터 255까지의 숫자입니다.

예를 들어, "192.168.0.1"은 일반적인 로컬 네트워크에서 사용되는 IP 주소입니다.

"localhost"는 현재 사용 중인 컴퓨터를 가리키는 호스트 이름입니다.

자기 자신을 가리키기 위해 사용되며, 일반적으로 루프백 주소인 "127.0.0.1"로 매핑됩니다. 매핑이란? 데이터를 한 형태에서 다른 형태로 변환하는 것을 나타낼 수 있습니다. 변환전과 변환후를 1:1로 매핑한다고 이야기한다.

상위 호스트 이름 입력부분에 localhost 대신에 IP주소를 넣어도 된다.

DNS(도메인 네임 시스템)은 컴퓨터 네트워크에서 사용되는 서비스로, 사람이 이해하기 쉬운 도메인 이름(예: www.example.com)을 컴퓨터가 이해할 수 있는 IP 주소(예: 192.168.0.1)로 변환하는 역할을 합니다. 네이버 사이트 접속시 124.12.132.12 이라 입력하지 않고 주소를 넣어도 처리되는 이유는 dns서버가 [www.naver.com](http://www.naver.com)이라는 문자열을 ip주소로 변환하여 상대방 컴퓨터를 찾아 갈 수 있다.

#### 포트 (Port):

포트는 네트워크 연결에서 서비스를 식별하기 위한 숫자입니다.

ip로 상대방 컴퓨터까지 찾아오면 컴퓨터 안에 네트워크를 사용하는 여러 응용프로그램이 있다. 여러 응용프로그램중 누구에게 데이터를 줄것인가를 포트로 결정할 수 있다.

택배 보내면 주소와 받을 사람 이름을 쓰는데 주소는 ip, 포트는 받을 사람 이름이라고 생각하면 된다.

컴퓨터가 여러 응용 프로그램들이 동작 할 때, 각 응용 프로그램은 고유한 포트 번호를 사용합니다.

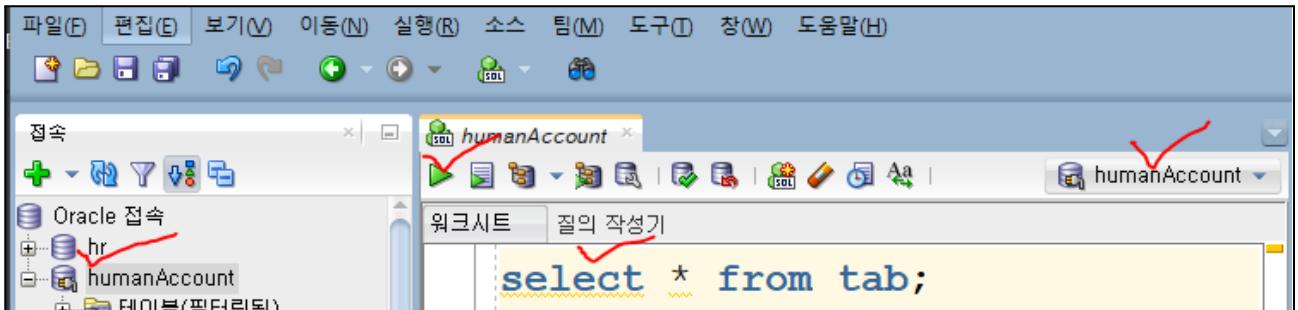
예를 들어, 웹 서버는 일반적으로 80번 포트를 사용하며, 보안 연결에는 443번 포트, 파일서버는 21, 오라클은 1521를 사용한다.

#### SID (System ID):

SID는 Oracle 데이터베이스를 한 컴퓨터에 여러개 설치 하였을때 각각의 데이터베이스를 식별하는 데 사용되는 고유한 이름 또는 식별자입니다. 본인이 변경하지 않았다면 일반적으로 xe라는 이름을 가지고 같은 이름으로 2개 이상 설치 할 수 없다. 설치시 마다 다른 이름으로 설치하여야 한다.

따라서, 호스트 이름, IP 주소, 포트, 그리고 SID는 데이터베이스 연결에 필요한 정보입니다. 이 정보를 사용하여 데이터베이스 클라이언트나 도구에서 데이터베이스 서버에 연결하여 sql을 이용해 원하는 데이터 처리가 가능하게 된다.

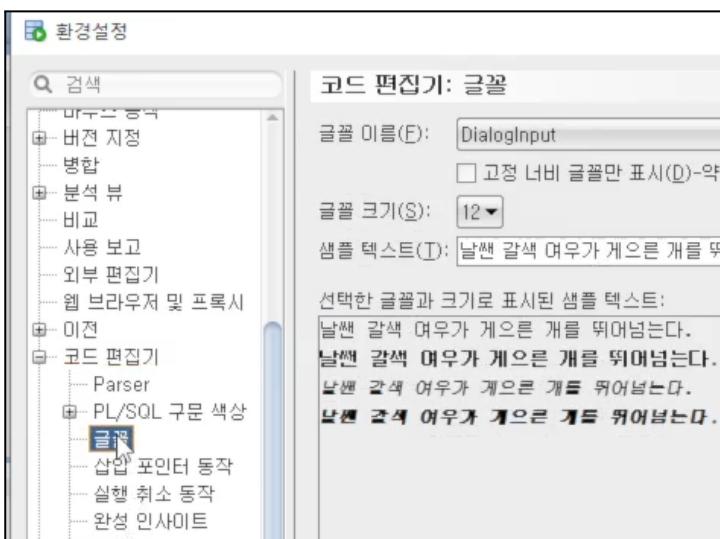
등록이 완료되면 아래 이미지 처럼 접속창에 등록된 아이디가 생긴다. 생성한 human Account를 더블 클릭하여 아이디와 패스워드를 입력 하면 sql을 작성 할 수 있는 창이 생긴다. 원하는 sql을 기술하고 기술한 내용을 실행시키고 싶다면 해당 sql을 마우스로 드래그한 다음 상단 화살표 모양인 실행 버튼을 누르면 선택한 부분의 쿼리문만 실행 된다.



그냥 녹색 화살표 모양 실행 버튼을 누르면 작성된 모든 sql이 실행된다. 실행하고 싶은 sql에 커서를 위치시키고 **ctrl+엔터**를 누르면 커서가 위치한 sql이 실행되는데 앞뒤 sql문이 제대로 종료되어 있지 않으면 문제가 발생한다. 가장 많이 발생하는 예라는 이전 sql이나 다음 sql 끝에 ;를 찍지 않고 sql를 실행 시킬때 발생한다. 예러가 발생하면 앞뒤 문맥에 문제가 없는지 확인하자.

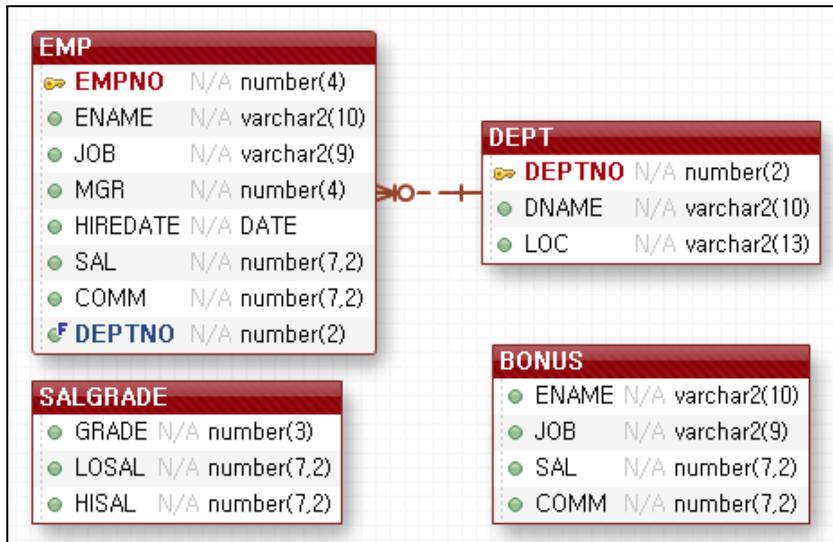
오른쪽 상단에 접속된 사용자가 뜨고 사용자를 변경해서 작성한 sql을 실행하고 싶다면 클릭해서 사용자를 변경하면 된다.

메뉴에서 파일 >> 저장 을 선택하거나 메뉴 밑에 디스크 모양의 저장아이콘을 선택해서 작성한 sql문을 원하는 위치에 저장 하였다가 다음에 불러와서 사용할 수 있다. 이전에 sqlplus로 하던 human 테이블 만드는 작업을 sql developer 에서 해보자.



도구>>환경설정>>글꼴에서 원하는 폰트와 크기를 설정할 수 있다.

다음을 따라해서 c##scott/tiger 데이터베이스를 만들어 보자. 해당 계정의 테이블들은 회사에서 사원을 관리하는 테이블이다.



c##scott 계정은 다음과 같은 테이블들로 구성되어 있다.

사원 관리 데이터베이스이며 사원, 부서, 보너스, 연봉등급과 같은 데이터를 저장 할 수 있는 테이블을 가지고 있다.

아래에 테이블명과 해당 테이블의 설명이 표로 만들어져 있다. 읽어보고 이해해 보자.

Table_ID	EMP	
테이블 명	사원 테이블	
테이블 설명	회사사원 정보를 저장	
Column_ID	컬럼명	자료형
empno	사원번호	number(4)
ename	사원이름	varchar2(10)
job	사원직업	nvarchar2(9)
mgr	사원상사번호	number(4)
hiredate	사원고용일	date
sal	사원월급	number(7, 2)
comm	사원커미션	number(7, 2)
deptno	사원부서넘버	number(2)

다음 표를 참고하여 c##scott/tigger 계정을 만들고 4개의 테이블과 데이터를 만들어 보자.

커미션은 연봉이외에 추가로 받는 돈을 의미한다.

Table_ID	DEPT	
테이블 명	부서 테이블	
테이블 설명	부서 정보 저장	
Column_ID	컬럼명	자료형
deptno	부서 번호	number(2)
dname	부서 이름	varchar2(15)
loc	부서 지역	varchar2(13)

Table_ID	BONUS	
테이블 명	보너스 테이블	
테이블 설명	보너스 받은사람 정보	
Column_ID	컬럼명	자료형
ename	받는 사람 이름	nvarchar2(10)
job	받는 사람 직업	nvarchar2(9)
sal	받는 금액	number
comm	받는 추가 금액	number

Table_ID	SALGRADE	
테이블 명	연봉등급	
테이블 설명	연봉등급정보	
Column_ID	컬럼명	자료형
grade	연봉 등급 정보	number
losal	최저 금액	number
hisal	최고 금액	number

다음은 sql 결과물이다. 될 수 있으면 스스로 해보고 잘 안되면 보고 해 보자.

1. c##scott/tiger 계정 만들기는 sqlplus로 작업하는 것이다.

```
sqlplus /nolog --로그인 없이 접속
conn sys as sysdba;      --sys 계정 sysdba 비번으로 로그인
show user;               -- 현재 로그인한 계정 확인
create user c##scott identified by tiger; -- c##scott 아이디 tiger 비밀번호
계정 생성
grant connect, resource, dba to c##scott; -- c##scott 계정에 권한 부여
conn c##scott/tiger -- c##scott 계정에 tiger 비밀번호로 로그인함
show user                -- 현재 로그인 계정 확인
```

2. 테이블들을 만들어 보자. 이미 존재할 수 있으니 drop한 다음에 테이블을 만들어 볼 것이다. 지면상 한줄에 여러 컬럼을 기술하고 있지만 실제 입력 할 때는 한줄에 한 컬럼식 입력 하자.

```
DROP TABLE EMP;--이미 만들어진 테이블이 있으면 지우고 만들려고 추가함
CREATE TABLE EMP(
    EMPNO NUMBER(4),    ENAME VARCHAR2(10),      JOB VARCHAR2(9),
    MGR NUMBER(4),      HIREDATE DATE,          SAL NUMBER(7,2),   COMM NUMBER(7,2),
    DEPTNO NUMBER(2)
);
DROP TABLE DEPT;
CREATE TABLE DEPT(
    DEPTNO NUMBER(2),  DNAME VARCHAR2(14) ,     LOC VARCHAR2(13)
) ;
DROP TABLE BONUS;
CREATE TABLE BONUS(
    ENAME VARCHAR2(10),    JOB VARCHAR2(9),    SAL NUMBER, COMM NUMBER
);
DROP TABLE SALGRADE;
CREATE TABLE SALGRADE(
    GRADE NUMBER,        LOSAL NUMBER,       HISAL NUMBER
);
```

3. 상위에 기술한 데이터를 입력하면 c##scott 계정 데이터베이스는 완료가 된다.

다음은 dept 테이블의 데이터이다.

```
INSERT INTO DEPT VALUES(10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES(20, 'REASEARCH', 'DALLAS');
INSERT INTO DEPT VALUES(30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES(40, 'OPERATIONS', 'BOSTON');
```

다음은 emp 테이블의 데이터입니다.

```
INSERT INTO EMP VALUES(7369, 'SMITH', 'CLERK', 7902,to_date('1980:12:17
00:00:00','YYYY:MM:DD HH24:MI:SS'),800,NULL,20);
INSERT INTO EMP VALUES(7499, 'ALLEN', 'SALESMAN', 7698,to_date('1981:02:20
00:00:00','YYYY:MM:DD HH24:MI:SS'),1600,300,30);
```

```

INSERT INTO EMP VALUES(7521,'WARD','SALESMAN',7698,to_date('1981:02:22
00:00:00','YYYY:MM:DD HH24:MI:SS'),1250,500,30);
INSERT INTO EMP VALUES(7566,'JONES','MANAGER',7839,to_date('1981:04:02
00:00:00','YYYY:MM:DD HH24:MI:SS'),2975,NULL,20);
INSERT INTO EMP VALUES(7654,'MARTIN','SALESMAN',7698,to_date('1981:09:28
00:00:00','YYYY:MM:DD HH24:MI:SS'),1250,1400,30);
INSERT INTO EMP VALUES(7698,'BLAKE','MANAGER',7839,to_date('1981:05:01
00:00:00','YYYY:MM:DD HH24:MI:SS'),2850,NULL,30);
INSERT INTO EMP VALUES(7782,'CLARK','MANAGER',7839,to_date('1981:06:09
00:00:00','YYYY:MM:DD HH24:MI:SS'),2450,NULL,10);
INSERT INTO EMP VALUES(7788,'SCOTT','ANALYST',7566,to_date('1987:07:13
00:00:00','YYYY:MM:DD HH24:MI:SS'),3000,NULL,20);
INSERT INTO EMP VALUES(7839,'KING','PRESIDENT',NULL,to_date('1981:11:17
00:00:00','YYYY:MM:DD HH24:MI:SS'),5000,NULL,10);
INSERT INTO EMP VALUES(7844,'TURNER','SALESMAN',7698,to_date('1981:09:08
00:00:00','YYYY:MM:DD HH24:MI:SS'),1500,0,30);
INSERT INTO EMP VALUES(7876,'ADAMS','CLERK',7788,to_date('1987:07:13
00:00:00','YYYY:MM:DD HH24:MI:SS'),1100,NULL,20);
INSERT INTO EMP VALUES(7900,'JAMES','CLERK',7698,to_date('1981:12:03
00:00:00','YYYY:MM:DD HH24:MI:SS'),950,NULL,30);
INSERT INTO EMP VALUES(7902,'FORD','ANALYST',7566,to_date('1981:12:03
00:00:00','YYYY:MM:DD HH24:MI:SS'),3000,NULL,20);
INSERT INTO EMP VALUES(7934,'MILLER','CLERK',7782,to_date('1982:01:23
00:00:00','YYYY:MM:DD HH24:MI:SS'),1300,NULL,10);

```

다음은 연봉등급 테이블 데이터입니다.

```

INSERT INTO SALGRADE VALUES (1,700,1200);
INSERT INTO SALGRADE VALUES (2,1201,1400);
INSERT INTO SALGRADE VALUES (3,1401,2000);
INSERT INTO SALGRADE VALUES (4,2001,3000);
INSERT INTO SALGRADE VALUES (5,3001,9999);

```

다음은 oracle 21에서 hr db 사용하는 방법이다. 확인해서 hr 데이터 베이스를 만들어 보자.

1. hr데이터베이스 설치를 위한 파일을 다운로드 해 보자. hr데이터 베이스를 21c 버전은 문제가 있어 문제 없이 설치되는 18c 버전으로 설치해 볼 예정이다.

Database 18c Sample Schemas를 구글에서 검색해서 github를 찾아 들어가 보자. 다음 페이지의 이미지를 확인해 보자.

Google Database 18c Sample Schemas

검색결과 약 84,600개 (0.25초)

도움말: 이 검색을 영어 검색 결과로 제한합니다. 언어별 필터링에 대해 자세히 알아보기

**GitHub** <https://github.com/oracle-samples/releases>

**Releases · oracle-samples/db-sample-schemas**

Use these scripts to create the Oracle Database 18c Sample Schemas referenced in the documentation and examples. The scripts install in Oracle Database 12c and ...

새로 열린 페이지에서 아래로 드래그해서 18c 예제를 찾아 assets를 클릭해서 나타나는 source code zip 파일을 다운로드 한다.

## Oracle Database Sample Schemas 18c

Use these scripts to create the Oracle Database 18c Sample Schemas referenced in the [documentation](#) and examples. The scripts install in Oracle Database 12c and upwards, including Oracle Database Cloud Services.

**Assets** 2

<a href="#">Source code (zip)</a>	Apr 6, 2018
<a href="#">Source code (tar.gz)</a>	Apr 6, 2018

1 person reacted

db-sample-sche... > db-sample-schemas-18c

이름	유형
bus_intelligence	파일 폴더
human_resources	파일 폴더
info_exchange	파일 폴더
order_entry	파일 폴더
product_media	파일 폴더
sales_history	파일 폴더

demo > schema > human\_resources

- 이름
  - hr\_analz.sql
  - hr\_code.sql
  - hr\_comnt.sql
  - hr\_cre.sql
  - hr\_drop.sql
  - hr\_drop\_new.sql
  - hr\_idx.sql
  - hr\_main.sql

왼쪽 이미지는 다운로드한 zip 파일 압축을 풀었을 때 이미지이고 오른쪽 이미지는 human\_resource 폴더 안에 들어 있는 파일들이다.

2. sys계정은 데이터 베이스를 관리하는 최상위 계정이다. sys계정을 사용하면 데이터베이스에서 할 수 있는 모든 작업을 제한 없이 할 수 있어 계정을 만들고 권한을 줄 때 sys계정을 사용한다.

```

사용자명 입력: sys as sysdba
비밀번호 입력:

다음에 접속됨:
Oracle Database 18c Express Edition Version 18.4.0.0.0

SQL> -

```

1. sys 계정으로 로그인 한다.

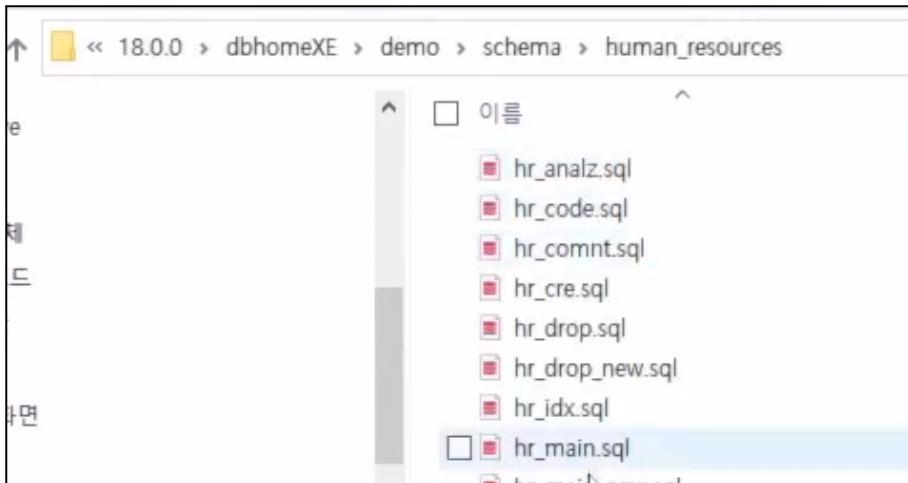
```

sqlplus /nolog
conn sys as sysdba;

```

2. 압축을 풀어 human\_resource 내부 파일들을

C:\app\foxman12\product\21.0.0\dbhomeXE\demo\schema\human\_resources 폴더에



복사한다. 경로는 오라클 버전이나 사용자 윈도우 계정에 따라 다를 수 있으니 잘 확인해 보고 human\_resource 폴더가 없으면 직접 만들어 파일들을 복사하자. 상위 경로에서 foxman12는 계정명이고 21.0.0은 오라클 버전에 따라

달라진다.

```

*hr_main.sql - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
ALTER SESSION SET NLS_LANGUAGE=American;
ALTER SESSION SET NLS_TERRITORY=America;
-- create tables, sequences and constraint
-- 
-- 
@/_SUB_CWD_ /human_resources/hr_popul
-- 
-- create indexes
-- 

```

3. human\_resource 폴더안에 hr\_main.sql를 열면 왼쪽 이미지처럼 @\_SUB\_CWD\_ 부분의 문자열이 여러개 있는데 이부분을 @?/demo/schema로 모두 변경 후 저장하고 다음 설치를 이어 나가자.

이후에 다음 스크립트를 사용하게 되는데 사용 이유는 다음과 같다.

```

alter session set
"_ORACLE_SCRIPT"=true;

```

과거에 계정을 만들때에 계정 이름에 c## 없이도 만들 수 있지만, 최신 오라클은 계정은

이름에 c## 없으면 계정을 만들 수 없다. 우리가 다운로드 받은 설치 sql은 과거 버전 기준으로 되어 있어서 계정을 hr로 만드는 것으로 되어 있어, 최신 버전 21c에서 실행시 계정에 c##이 붙어 있지 않다는 이유로 계정 에러가 난다. 이런 문제를

해결하기 위해서 alter session ... 질의는 c## 없이 계정을 만들수 있도록 oracle 설정을 변경한 것 이어서, 해당 작업 이후에는 c##hr이 아닌 c##이 없는 hr 계정을 만들어 사용할 수 있게 된다.

```
SQL> alter session set "_ORACLE_SCRIPT"=true;  
Session altered.  
SQL> @?/demo/schema/human_resources/hr_main.sql
```

4. 다운로드한 sql파일을 이용해서 hr테이블 만드는 방법은 다음과 같다.

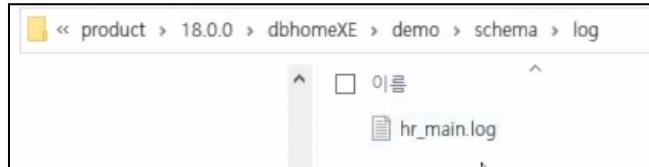
@?/demo/schema/human\_resources/hr\_main.sql 또는  
@C:\app\foxman12\product\21.0.0\dbhomeXE\demo\schema\human\_resources\hr\_main.sql 를 입력하여 hr계정과 테이블등을 생성해 보자. @는 해당 sql파일을 실행하라는 의미이고 ?는 c의 루트 폴더 부터 (루트 폴더는 c드라이브 시작 폴더를 의미 한다.) dbhomeXE폴더 까지의 전체 경로를 의미한다. @다음에 실행할 sql파일 경로와 이름을 넣어서 해당 경로의 hr\_main.sql 파일을 실행 할 계획이다. 입력하고 엔터를 치면 다음 이미지처럼 사용자 입력을 받을 수 있다.

```
SQL> @C:\app\foxman12\product\21c\dbhomeXE\demo\schema\human_resources\hr_main.sql  
specify password for HR as parameter 1:  
1의 값을 입력하십시오: hr  
specify default tablespace for HR as parameter 2:  
2의 값을 입력하십시오: users  
specify temporary tablespace for HR as parameter 3:  
3의 값을 입력하십시오: temp  
specify password for SYS as parameter 4:  
4의 값을 입력하십시오: sysdba  
specify log path as parameter 5:  
5의 값을 입력하십시오: C:\app\foxman12\product\21c\dbhomeXE\demo\schema\log  
specify connect string as parameter 6:  
6의 값을 입력하십시오: localhost:1521/xe
```

1. 입력 hr
2. 입력 users
3. 입력 temp
4. 입력 sysdba
5. 입력 C:\app\foxman12\product\21.0.0\dbhomeXE\demo\schema\log

6. 입력 localhost:1521/xe

5. 혹시 문제가 발생하면 오류가 있는지 확인하고 아래 이미지와 같은 schema 경로 안에 log 폴더가 있는지 확인해보자.



6. 만든 계정으로 로그인 해보자. conn hr / hr;

7. select \* from tab;를 입력하여 8개가 나오는데 7개의 테이블과 1개의 뷰가 있는지 확인해 보자.

8. desc employees;를 응용하여 검색된 7개 테이블들의 세부 내용을 확인해 보자.

9. sqlDeveloper에 hr데이터베이스를 사용할 수 있도록 등록해 보자.

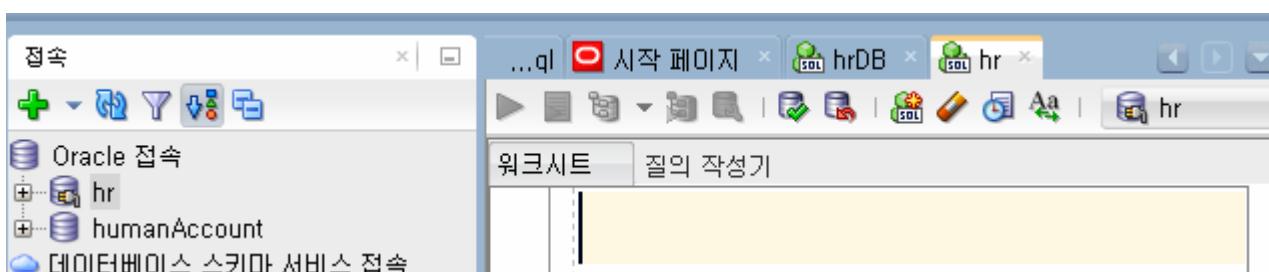
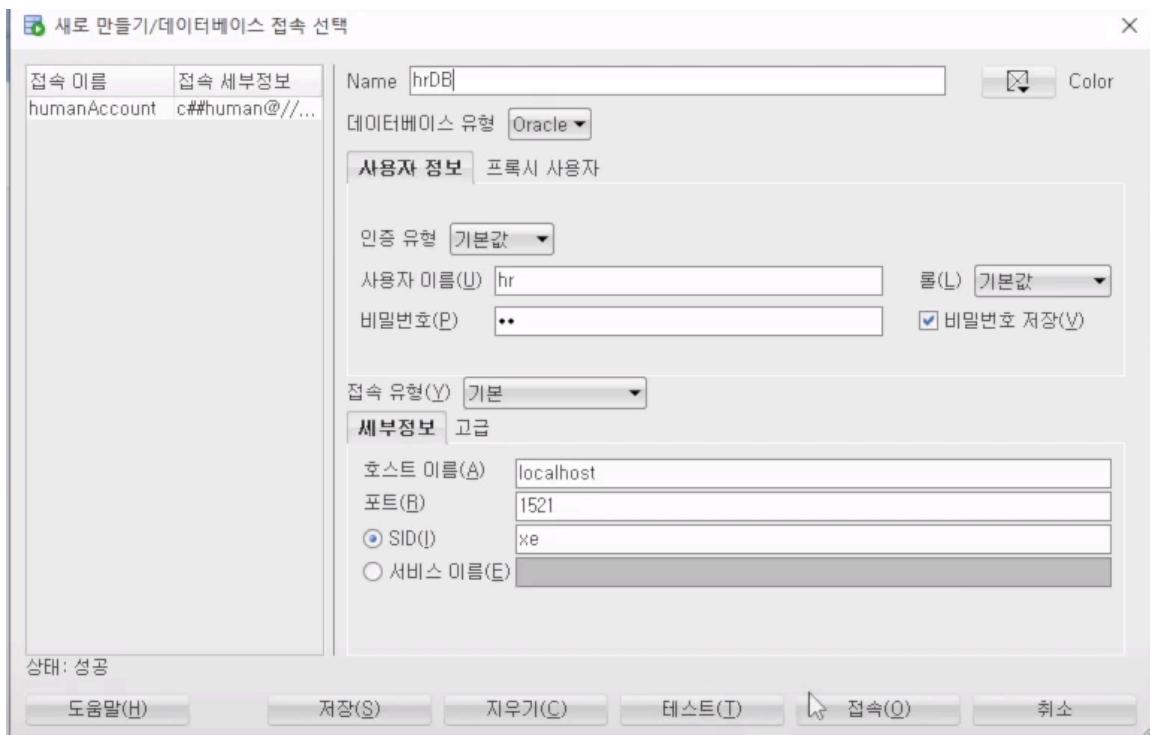
이후 나오는 과정에서 c##scott와 hr테이블을 이용해서 sql 기본 문법을 공부할 예정이다.

특정 계정에 table를 확인 하려면 해당 계정으로 로그인 하여 select \* from tab; 를 입력하면 해당 계정의 테이블 정보가 리턴 된다.

9. sqldeveloper에 hr과 c##scott계정을 등록해 보자. 아래 이미지에서 왼쪽 상단의 녹색 + 버튼을 클릭하면 등록 할 수 있는 창이 뜬다.

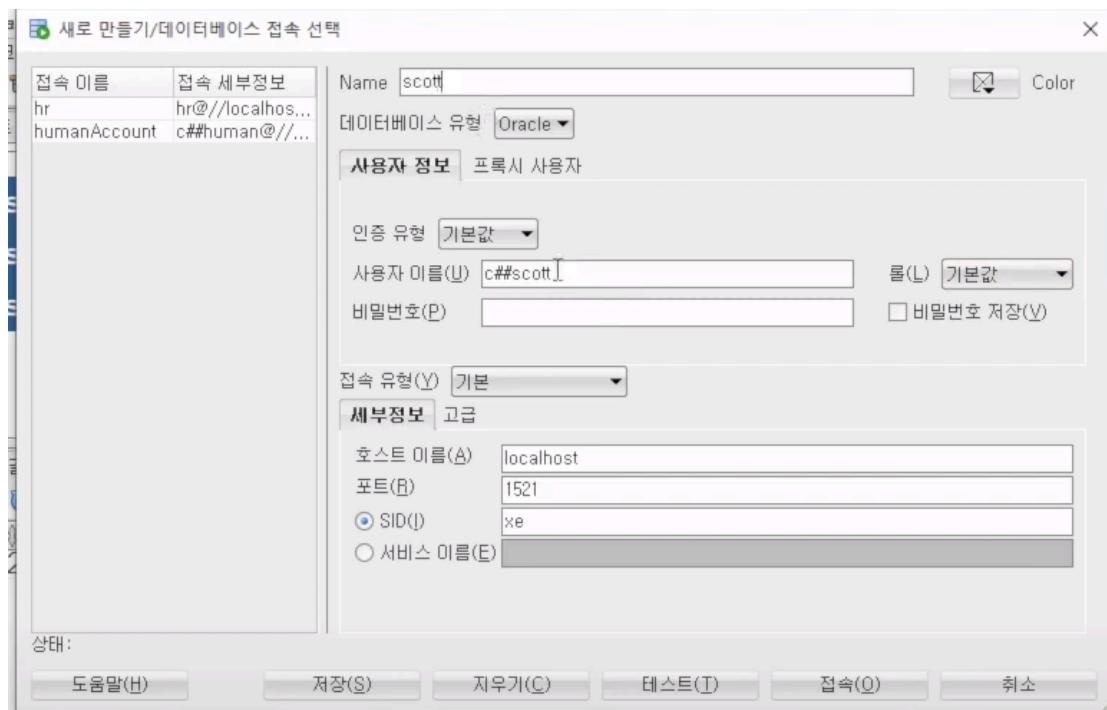


녹색 + 부분을 누르고 다음과 같이 hr디비를 등록한 다음 접속 버튼을 누르면 등록된다.



10. 등록이 마무리 되면 처럼 왼쪽 접속 창에서 원하는 db를 클릭하면 해당 계정에 접근 할 수 있는 sql문서가 열리고 오른쪽에 메모장 처럼 사용할 수 있는 편집기가 열리고 그곳에 sql 를 작성 하면된다.
11. 접속할 계정을 변경 하고 싶으면 접속 창에서 원하는 db를 더블 클릭하거나 현재 작성중인 파일의 오른쪽 상단에 원하는 계정을 선택하면 된다.
12. 작업이 완료되거나 새로운 문서를 열고 싶다면 저장후 새로 문서를 열어서 원하는 db를 선택한후 작업을 하면 된다. 상단 파일 메뉴에 파일 저장과 새로 만들기 관련 메뉴가 있다.

다음 이미지에서 처럼 c##scott계정도 등록해 보자. 비밀번호는 tiger이다.



왼쪽 이미지 처럼 오른쪽 상단에 어떤 DB가 선택되어 있으나에 따라 select \* from tab; 의 실행 결과가 달라진다. scott계정에서는 scott관련 정보가 hr계정에서는 hr관련 정보가 출력된다.

제대로 설치 되었는지 hr계정과 scott계정에서 다음과 같이 출력해 보자.

TNAME	TABTYPE	CLUSTERID
COUNTRIES	TABLE	(null)
DEPARTMENTS	TABLE	(null)
EMPLOYEES	TABLE	(null)
EMP DETAILS	VIEW	(null)
JOB DETAILS	VIEW	(null)
JOB HISTORY	TABLE	(null)
LOCATIONS	TABLE	(null)
REGIONS	TABLE	(null)

TNAME	TABTYPE	CLUSTERID
BONUS	TABLE	(null)
DEPT	TABLE	(null)
EMP	TABLE	(null)
SALGRADE	TABLE	(null)

특정 table구조를 확인 할 때 ‘desc 테이블명’ 을 사용한다.

-hr계정에서 사원과 사원의 과거 업무 기록을 저장하는 테이블 구조를 출력해 보자.

```
desc employees; desc job_history;
```

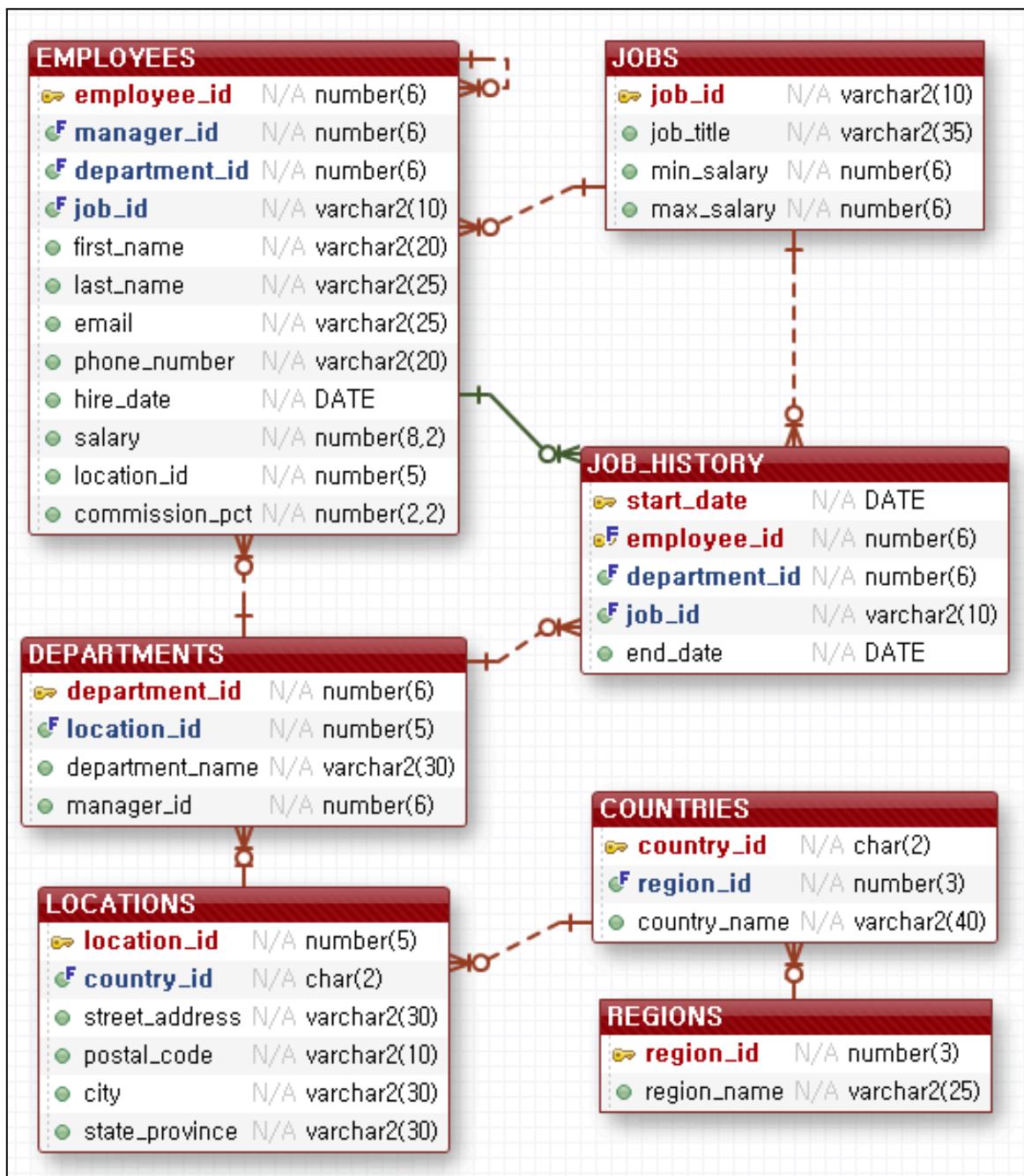
-hr계정에서 부서가 위치한 대륙 정보를 저장하는 테이블 구조를 출력해 보자.

```
desc regions;
```

다음 문제들을 scott 데이터 베이스에서 구해보자.

-scott계정에서 사원 테이블의 테이블 구조를 출력해 보자. desc emp;

-scott계정에서 부서 테이블의 테이블 구조를 출력해 보자. desc dept;



상위 ERD는 HR 테이블 관련 내용이다. hr 계정은 회사에서 사원을 관리하는 데이터 베이스이다. hr의 부서는 전세계에 걸쳐서 위치하고 있다.

**Employees** 테이블에 사원관련 정보를 저장하고 **jobs**테이블에는 사원이 맡은 직무(직종)를 기록하고 **job\_history**테이블에는 사원의 직무 이력을 기록한다. 직무 이력 내용으로 해당 직무의 시작일, 종료일, 사원id, 직무id, 부서 id관련 이력이 저장된다.

**departments**테이블에는 부서 정보를 저장한다. **locations** 테이블은 부서의 위치 정보를 가지고 있다. **countries**테이블에는 부서가 위치한 나라 정보를 저장한다. **regions** 테이블에는 부서가 위치한 대륙 정보를 저장 한다.

Table_ID	COUNTRIES				작성일자	
테이블 명	국가				작성자	
테이블 설명	각 부서가 위치한 국가 정보 테이블					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
country_id	국가 코드	CHAR	Y	P		
country_name	국가명	VARCHAR2				
region_id	대륙 코드	NUMBER		F	regions(region_id) 참조	

Table_ID	EMPLOYEES				작성일자	
테이블 명	사원				작성자	
테이블 설명	사원 정보 테이블					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
employee_id	사번	NUMBER(6)	Y	P		
first_name	이름	VARCHAR2				
last_name	성	VARCHAR2	Y			
email	이메일	VARCHAR2	Y	U		
phone_number	전화번호	VARCHAR2				
hire_date	입사일	DATE	Y			
job_id	직무코드	VARCHAR2	Y	R	jobs(job_id) 참조	
salary	급여	NUMBER			salary > 0	
commission_pct	추가금	NUMBER				
manager_id	담당자 사번	NUMBER		R	employees(employee_id) 참조	
department_id	부서 번호	NUMBER		R	departments(department_id)참조	

Table_ID	DEPARTMENTS				작성일자	
테이블 명	부서				작성자	
테이블 설명	부서 정보 테이블					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
department_id	부서 코드	NUMBER		P		
department_name	부서 이름	VARCHAR2	Y			
manager_id	담당자 사번	NUMBER		F	employees (employee_id) 참조	
location_id	지역 코드	NUMBER		F	locations (location_id) 참조	

Table_ID	JOBS				작성일자	
테이블 명	직무				작성자	
테이블 설명	직무, 급여 정보					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
job_id	직무 코드	VARCHAR2	Y	P		
job_title	직무 이름	VARCHAR2	Y			
min_salary	최소 급여	NUMBER				
max_salary	최대 급여	NUMBER				

Table_ID	JOB_HISTORY				작성일자	
테이블 명	계약기록				작성자	
테이블 설명	해당 사번의 과거 계약기록					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
employee_id	사번	NUMBER	Y	P,F	employees(employee_id) 참조	
start_date	입사일	DATE	Y	P		
end_date	퇴사일	DATE	Y		end_date > start_date	
job_id	직무 코드	VARCHAR2	Y	F	jobs(job_id) 참조	
department_id	부서 코드	NUMBER		F	departments(department_id) 참조	

Table_ID	LOCATIONS				작성일자	
테이블 명	부서위치				작성자	
테이블 설명	부서 위치 정보 테이블					
Column_ID	컬럼명	자료형	Not NULL	key	비 고	
location_id	지역 코드	NUMBER	Y	P	regions(region_id)	
street_address	주소	VARCHAR2				
postal_code	우편번호	VARCHAR2				
city	도시	VARCHAR2	Y			
state_province	주	VARCHAR2				
country_id	국가 코드	CHAR		F		

추가로 regions 테이블이 존재하는데 테이블의 내용은 대륙 정보를 가지고 있는 테이블로 대륙을 식별하는데 필요한 Region\_id와 대륙의 이름을 저장하는 region\_name으로 이루어져 있다. 상위 표들을 숙지해 보자.

## > 06. select문 사용하기

select문은 데이터베이스의 테이블에 들어있는 데이터를 검색하는데 사용하는 sql 질의어이다. 사용 방법은 다음과 같다.

**select 컬럼선택 from 테이블선택**

모든 컬럼을 검색하고 싶다면 컬럼 선택하는 부분에 \*을, 몇몇 특정 컬럼의 내용을 출력하고 싶다면 컬럼명을 ,로 구분하여 기술하면 된다.

select * from countries;		
질의 결과 x		
□ COUNTRY_ID	□ COUNTRY_NAME	□ REGION_ID
1 AR	Argentina	2
2 AU	Australia	3
3 BE	Belgium	1
4 BR	Brazil	2
5 CA	Canada	2
6 CH	Switzerland	1
7 CN	China	3
8 DE	Germany	1
9 DK	Denmark	1
10 EG	Egypt	4

이후 과정에서 일반 예제는 hr계정을 사용해서 실습하면 된다. 이전에 hr 관련 ERD가 있는 페이지에 전체 테이블 그림과 설명을 보고 원하는 데이터를 찾아서 문제를 해결해 보자.

1. 회사와 관련된 모든 나라 정보를 출력해 보자.  
왼쪽 이미지 처럼 sql developer를 실행시킨 다음 sql문을 작성하자.
2. 회사와 관련있는 대륙 정보를 모두 출력해 보자. 대륙이란? 아시아, 유럽 등을 의미 한다.  
아래 이미지를 참조해 보자.

3. 회사와 관련있는 부서가 존재하는 모든 지역정보를 출력해 보자.

select * from locations;			
의 결과 x			
□ LOCATION_ID	□ STREET_ADDRESS	□ POSTAL_CODE	□ CITY
1 10001297	Via ...	00989	Roma
2 110093091	Cal...	10934	Venice
3 12002017	Shin...	1689	Tokyo
4 13009450	Kami...	6823	Hiroshima
5 14002014	Jabb...	26192	Southlake
6 15002011	Inte...	99236	South San
7 16002007	Zago...	50090	South Bru
8 17002004	Char...	98199	Seattle
9 1800147	Spadi...	M5V 2L7	Toronto

select * from regions;	
의 결과 x	
□ REGION_ID	□ REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

4. 하나의 컬럼을 출력하고 싶다면 컬럼 선택 부분에 하나의 컬럼명을 기술 하면 된다.

--회사관련 건물들의 거리 주소만 출력해 보자.

```
select street_address from locations;
```

--회사에 존재하는 모든 작업 아이디만 출력해보자.

```
select job_id from jobs;
```

--회사에 존재하는 모든 (직업)직무에 이름을 출력해보자.

```
select job_title from jobs;
```

3. 여러개의 컬럼을 출력하고 싶다면 컬럼 선택 부분에 ,를 이용하여 여러개 기술 하면 된다.

--모든 사원의 이름과 성을 출력해 보자.

```
select first_name, last_name from employees;
```

--사원의 사번과 이메일을 출력해 보자.

```
select employee_id, email from employees;
```

--직업 이력에 있는 사원들의 사번, 직업시작일, 직업종료일을 출력해 보자.

```
select employee_id, start_date, end_date from job_history;
```

-- 직업별 직업명,최소,최대 급여를 출력해 보자.

```
select job_title, min_salary, max_salary from jobs;
```

## > 07. distinct로 중복제거하기

이전 sql문들은 중복 데이터가 있으면 여러개 출력이 된다. 중복 데이터를 없애고 싶으면 `distinct`를 사용하면 된다. `distinct`가 설정되면 검색된 컬럼의 모든 데이터가 중복될 때 생략된다.

```
select employee_id from job_history;
```

The screenshot shows a SQL query results window with the following data:

EMPLOYEE_ID
101
101
102
114
122
176
176
200
200
201

`select`문은 실제 데이터에 영향을 주지 않고 변형된 데이터를 제공한다. `distinct`를 사용하면 원본 데이터가 변경되지 않고 새로 가공된 데이터를 보여 준다.

--회사 업무 이력이 있는 사원의 사번을 모두 출력해 보면 왼쪽과 같이 총 10개 인것을 확인할 수 있다. 자세히 보면 중복된 값 101, 176, 200 이 있는 것을 확인 할 수 있다.

중복없이 출력하고 싶다면 `distinct` 를 사용하면 된다. `distinct` 추가한 결과는 다음 이미지와 같다.

총 10개의 데이터중 중복된 데이터 3개를 제외하고 7개가 출력 되었다.

```
select distinct employee_id from job_history;
```

The screenshot shows a SQL query results window with the following data:

EMPLOYEE_ID
101
102
114
122
176
200
201

job\_history테이블은 총 10개의 데이터가 들어 있다. 다음 쿼리를 실행 시켜서 총 몇개의 데이터가 출력 되는지 확인해 보자.

`select distinct employee_id from job_history;` --7개가 출력 된다.

`select employee_id from job_history;` --10개가 출력 된다.

`select * from job_history;` --10개가 출력 된다.

```
select distinct * from job_history; --10개가 출력 된다. distinct는 출력된 모든 컬럼 값이 동일할 때 하나만 출력 된다. 일부분만 동일 하면 출력이 된다. 따라서, 모든 컬럼의 값이 동일한 데이터가 없으므로 10개의 데이터 모두가 출력 되었다.
```

```
select distinct employee_id,department_id from job_history; --7개가 출력 된다.  
해당 테이블의 총 데이터 개수가 10개이므로 employee_id,department_id 컬럼의 출력값에 중복되는 데이터가 3개 존재한다는 의미다. employee_id,department_id 컬럼 둘다 중복되어야 중복된 데이터로 본다.
```

```
select distinct employee_id,job_id from job_history; --10개가 출력 된다. 출력 결과 중복되는 데이터가 없다.
```

--사원의 업종(업무)ID를 중복없이 출력해 보자.

```
select distinct * from employees; --107개  
select distinct job_id from employees; --19개
```

--사원을 관리하는 관리자 사원 번호를 중복 없이 출력해보자.

```
select distinct manager_id from employees;
```

-사원의 직업아이디(업종)과 급여를 중복없이 출력해 보고 중복을 허용 하였을 경우와 개수 차이를 확인해 보자.

```
select job_id,salary from employees; --107  
select distinct job_id,salary from employees; --82
```

## > 08. select의 where절

where절의 조건문을 사용하여 테이블의 원하는 데이터만 출력할 수 있다.

```
select 컬럼선택 from 테이블선택 where 조건선택
```

지금까지 sql문은 테이블에 있는 모든 데이터(row)가 출력 되었다. where절에 비교 연산자를 사용한 조건문을 작성하여 해당 조건에 맞는 데이터만 출력 할 수 있다.

예를 들면 테이블에 나이를 저장하는 age 컬럼명이 있고 age가 50이상인 데이터만 출력하고 싶다면 where절에 where age>50 이라고 입력 하여 전체 데이터중 age가 50보다 큰 데이터만 출력할 수 있다.

문자의 비교는 사전순으로 먼저나온 문자가 작고, 나중에 나온 문자가 크다.

시간은 과거의 시간보다 현재의 시간이 크고 현재 시간보다 미래 시간이 크다.

sql은 대소문자를 구분하지만 테이블에 들어 있는 데이터는 데소문자를 구분한다.

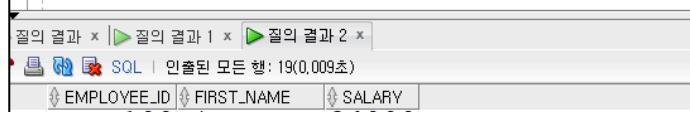
a >= 'a' and a <= 'b' b를 제외한 b로 시작하는 문자열 banana는 검색되지 않는다.

다음은 오라클에서 사용하는 비교 연산자이다. 참고해 보자.

속성	설명
=, !=, <>	=은 같을때 나머지는 다를 때 참(true)이다.
>, >= ,< ,<=	왼쪽이 클때, 왼쪽이 크거나 작을때, 오른쪽이 클때, 오른쪽이 크거나 같을때 true이다.

```
select employee_id,first_name,salary  
from employees  
where salary>=10000;
```

-봉급이 10000이상인 사원의 아이디와 이름과 봉급을 출력 하시오.



```
select employee_id, first_name, salary from employees      where  
salary>=10000;
```

--부서번호가 50인 사원 정보를 출력 하시오.

```
select * from employees where department_id=50;
```

--직업 아이디가 'AC\_MGR'인 직업정보를 출력해보자.

```
select * from jobs where job_id='AC_MGR';      - 'ac_mgr'로 입력하면 아무것도  
출력되지 않는것을 확인할 수 있다. 데이터는 대소문자를 구분한다.
```

--작업아이디가 'FI\_ACCOUNT'이 아닌 작업아이디 정보를 출력하시오.

```
select * from jobs where job_id != 'FI_ACCOUNT';
```

--2003년 9월 17일 이후에 입사한 사원의 사번,고용일, 전화번호를 출력하시오.

```
select * from employees where hire_date > '2003-09-17';
```

```
select * from employees where hire_date > to_date('2003:09:17 00:00:00',  
'YYYY:MM:DD HH24:MI:SS');
```

---

## > 09. null 검색하기

---

특정 컬럼에 null이 들어 있는 데이터를 찾을때 = 연산자를 사용해서 찾을 수 없다.

특정 컬럼에 null이 들어 있는 데이터를 찾고 싶다면 ‘where 컬럼명 is null’  
null이 아닌 데이터를 찾고 싶다면 ‘where 컬럼명 is not null’를 사용하면 된다.

--관리자가 없는 사원의 정보를 출력해보자.

```
select * from employees where manager_id=null;           --동작하지 않음  
select * from employees where manager_id is null;       -- null인경우 선택  
select * from employees where manager_id is not null;  -- null이 아닌경우 선택
```

-커미션이 있는 사원의 정보를 출력해 보자.

```
select * from employees where commission_pct is not null;
```

-부서를 배정받지 못한 사원을 출력해보자.

```
select * from employees where department_id is null;
```

-커미션이 있는 사원중 봉급이 10000이상인 사원의 정보를 출력해보자. where 절에 and 를 사용하여 두조건이 모두 맞을때만 동작하게 할 수 있다.

```
select * from employees  
where commission_pct is not null and salary>=10000;
```

---

## > 10. 컬럼 연산

---

컬럼을 연산자로 연산하여 새로운 결과를 얻어낼 수 있다. 컬럼+컬럼, 컬럼+숫자, 숫자+숫자

--다음을 확인해 보자. `select salary,salary+50, 10+10, '출력' from employees;`

--사원의 봉급을 50증가한 형태로 출력해 보자.

```
select salary,salary+50 from employees;
```

--사원의 현봉급보다 10%증가한 형태로 출력해 보자.

```
select salary,salary*1.1 from employees;
```

--사원의 현봉급보다 2배증가한 형태로 출력해 보자.

```
select salary, salary+salary, salary*2 from employees;
```

---

## > 11. 컬럼 별명 as

---

연산된 컬럼명은 복잡 해서 별명을 이용해서 컬럼명을 변경 해야 사용하기 쉽다.

'`select 컬럼명 as 별명`' 일반 컬럼 명이나 연산된 컬럼명 다음에 `as`를 이용해서 별명을 기술 할 수 있다. 여기서 `as`를 생략할 수 있다.

--다음을 확인해 보자. `select 10+10 as sum, '출력' str from employees;`

--사원의 봉급을 50증가한 형태로 컬럼명을 `upgradeSalary`로 출력해 보자.

```
select salary,salary+50 as upgradeSalary from employees;
```

```
select salary,salary+50 upgradeSalary from employees;
```

--사원의 현봉급 보다 10%증가한 형태로 컬럼명을 upgradeSalary로 출력해 보자.

```
select salary,salary*1.1 as upgradeSalary from employees;
```

---

## > 12. Concatenation(||)를 이용한 컬럼 합치기

||은 컬럼과 문자열을 합쳐서 새로운 컬럼과 데이터를 만들 수 있다.

| 키보드 엔터 위에 존재한다.

Concatenation(||)를 사용하여 컬럼에 결과물을 결합하여 하나의 컬럼을 만들 수 있다.

```
'select 컬럼명 || '과' || 컬럼명 as 합친칼럼명 from 테이블'
```

-사원의 성과 이름을 합쳐서 fullname이라는 컬럼명으로 출력해보자.

```
select first_name||last_Name as fullname from employees;
```

```
select first_name||' '||last_Name as fullname from employees;
```

## > 13. and or not 논리 연산자

where 절에 논리 연산을 이용해서 여러 비교 조건을 이용하여 한번에 검색할 수 있다.  
and 연산은 여러 조건들이 모두 참일때 해당 데이터를 읽어온다.  
or 연산은 여러 조건들 중에 하나라도 참이면 해당 데이터를 읽어온다.  
not 연산은 조건에 맞지 않는 경우의 데이터가 출력된다.

where  $a > 0$  and  $b > 0$  and  $c > 0$  과 같이 기술한다면 3가지 조건이 모두 참일 때만 해당 데이터가 출력이 된다. 나이가 50이면서 남자인 데이터 두 조건이 만족해야 데이터를 출력하고 싶을 때 and를 사용한다. `age=50 and gender= 'man'`

where  $a > 0$  or  $b > 0$  or  $c > 0$  와 같이 기술한다면 3가지 조건 중 하나라도 참이 있으면 해당 데이터가 출력이 된다. 나이가 50이거나 남자인 데이터 두 조건 중 하나만 만족하면 해당 데이터가 출력된다. `age=50 or gender= 'man'`

where  $a \neq 0$ , where not  $a=0$  a컬럼의 값이 0이 아닌 경우에 데이터가 출력 된다.

다음은 진리표이다 boolean의 논리 연산을 표로 만든 것이다.

a	b	$a \&\&b$	$a     b$	$!a$	c	$a     b     c$	$a \&\&b \&\&c$	$a     b \&\&c$
true	true	true	true	false	f	true	false	true
true	false	false	true	false	f	true	false	true
false	true	false	true	true	f	true	false	false
false	false	false	false	true	f	false	false	false

- 월급이 8000 초과 이면서 10000 미만인 사원의 정보를 출력해 보자.

```
select * from employees where salary>8000 and salary<10000;
```

- 월급이 7000이하 이거나 10000초과인 사원의 정보를 출력해 보자.

```
select * from employees where salary<=7000 or salary>10000;
```

-부서가 80이 아닌 사원 정보를 출력해 보자.

```
select * from employees where department_id!=80; -- <> ^=
```

```
select * from employees where not department_id=80;
```

-커미션을 받으면서 2008년 이전에 입사한 사원 정보를 출력해 보자.

```
select * from employees where hire_date < '2008-01-01' and commission_pct is  
not null; --2008년을 포함하고 싶으면 where hire_date < '2009-01-01' 과 같아  
하면된다.
```

- 대륙 아이디가(REGION\_ID) 1, 3, 4인 나라 정보를 출력해보자.

```
select * from countries where region_id=1 or region_id=2 or region_id=4;
```

---

## > 14. between 연산자

between은 특정 컬럼의 특정 범위에 해당하는 값을 출력 하고 싶을때 사용한다.

between은 특정 컬럼의 범위 데이터를 쉽게 구하기 위해서 사용 한다.

where 컬럼 between a and b a,b를 포함한 사이값을 찾는다 사용한다.

where 컬럼 not between a and b a,b를 포함한 사이값을 제외한 모든 데이터를 찾는다.

--연봉이 8000보다 같거나 많고 10000보다 같거나 작은 사원 정보를 출력해 보자.

```
select * from employees where salary between 8000 and 10000;
```

```
select * from employees where salary >= 8000 and salary<=10000;
```

--연봉이 7000 미만 이거나 10000초과인 사원 정보를 출력해 보자.

```
select * from emp where sal not between 7000 and 10000;
```

```
select * from employees where salary < 7000 or salary>10000;
```

--사원의 이름이 E~G로 시작하는 사원정보를 출력해 보자.

```
select * from employees where first_name between 'E' and 'H'
```

```
and first_name != 'H';
```

```
select * from employees where first_name >= 'E' and first_name <'H' ;
```

-- 사원의 입사일이 2004년 5월 20일 ~ 2007년 10월 9일 이전 까지의 사원 정보를 출력해 보자.

```
select * from employees where hiredate between '2004-05-20' and '2007-10-10';
```

## > 15. null 연산과 null값 변경 함수

모든 사원에게 커미션 1%씩 추가해서 화면에 출력한다고 생각해 보자.

사원 테이블의 커미션 컬럼에 +1하면 될것 같지만 커미션이 없는 사람은 데이터가 null이 들어가 있어 + 1 해도 null이 된다. 현실 세계에서 커미션 없던 사람에게 1% 커미션을 추가해준다면 null이 아닌 1이 되어야 한다. 이문제를 해결하기 위해서 null인 데이터를 다른 데이터로 여기서는 0으로 변경한후 연산하여야 한다. 이때 사용하는 null값을 변경하는 함수가 nvl, nvl2, decode가 있다.

null값을 가지고 연산을 하면 결과값은 null이 되어 생각과 다른 결과가 나오는 경우가 있다. null값을 다른 값으로 변경해서 원하는 결과가 나올 수 있도록 할 수 있는데 이때 사용하는 함수로 nvl, nvl2, decode가 있다. 컬럼을 사용할 위치에 해당 메소드를 사용하여 null값을 처리할 수 있다.

nvl(컬럼명,0) 컬럼명이 null일때 0를 출력 한다.

nvl2(컬럼명, 'notNull', 'null') 컬럼이 null이 아니면 'notNull'이 null이면 'null'이 출력 된다.

decode(컬럼명,null,0,컬럼명) 컬럼명이 가지고 있는 값이 null이면 0 아니면 컬럼명이 가지고 있는 값이 출력된다.

다음 sql을 실행 결과를 확인해서 sql를 이해해 보자.

```
select * from employees;  
  
select commission_pct from employees;  
  
select commission_pct+1 from employees;  
  
select nvl(commission_pct,0)+1 from employees;  
  
select nvl2(commission_pct,'notNull','strNull') from employees;  
  
select nvl2(commission_pct,commission_pct,0)+1 as addOne from employees;  
  
select decode(commission_pct,null,0,commission_pct)+1 addOne from employees;
```

--사원의 봉급과 커미션 비율을 이용하여 사원이 1년 동안 받는 전체 금액을 구해보자. 상위 3가지 방법을 이용해서 풀어보자. 봉급 salary는 월급에 해당하고 커미션은 1년간 받은 월급에 커미션 비율로 받는다.

```
select salary,commission_pct,salary*12+salary*12*commission_pct from employees;  
--실행해보면 null이 있어서 원하는 결과를 얻지 못함
```

```
select salary,nvl(commission_pct,0),salary*12+salary*12*nvl(commission_pct,0)  
from employees; --null 를 0으로 변경하여 원하는 결과를 얻을 수 있다.
```

-- 다음은 다른 방법으로 사원이 1년 동안 받는 전체 금액에 해당한다.

```
select salary,nvl2(commission_pct,commission_pct,0),salary*12+  
salary*12*decode(commission_pct,null,0,commission_pct) from employees;
```

--커미션을 받으면 yes 없으면 no가 출력 되도록 isCommition컬럼을 만들어 보자.상위 3가지 방법을 이용해서 풀어보자.

```
select nvl(commission_pct,'no') as isCommition from employees; --결과 자료형이 섞여 있어서 문제가 발생하여 동작하지 않음 나머지는 문제없이 동작
```

```
select nvl2(commission_pct,'no','yes') as isCommition from employees;  
select decode(commission_pct,null,'yes','no') isCommition from employees;
```

## > 16. decode와 case 문법

decode와 case는 기준값을 다른 값으로 변경할 때 사용한다. decode같은 경우 1:1 맵핑이 가능할 때 사용하고 case 같은 경우 범위 조회일 때 사용한다.

```
decode(컬럼명, 값1, 변경값1,
       값2, 변경값2..., '나머지경우변경값');
decode 다음에 오는 컬럼의 값이 값1이면 해당 컬럼의 값은 값1으로 변경되고 값2이면 값2로 변경되고 맵핑되는 값이 없을 때에는 나머지 경우 변경값이 출력 된다.
```

다음 예제는 job\_id 컬럼의 값이 'IT\_PROG'이면 'Developer'로, 'SA REP'이면 'Sales Representative'로, 그 외의 값이면 'Other Job'으로 변환됩니다.

```
SELECT
    employee_id,
    job_id,
    DECODE(job_id,
        'IT_PROG', 'Developer',
        'SA REP', 'Sales Representative',
        'Other Job') AS job_title
FROM employees;
```

-- 사원 테이블에서 10, 20, 30 부서는 숫자+ '부서입니다.' 나머지 부서는 '그밖에 부서입니다.'가 출력 되도록 쿼리를 구현해 보자.

```
select distinct department_id from employees; 를 실행해 보면 50 40 110 90 30 70 등과 같은 부서번호들이 들어 있다.
```

```
select decode(department_id,
              10, '10번 부서',
              20, '20번 부서',
              30, '30번 부서', department_id || '나머지부서') as 부서번호
from employees;
```

```
SELECT    컬럼명
, CASE WHEN 컬럼명 조건식1 THEN 값1
      WHEN 컬럼명 조건식2 THEN 값2
      WHEN 컬럼명 조건식3 THEN 값3
      ELSE 값4 END AS 별명
from employees;
```

case에서 end 까지가 하나의 컬럼이 되고, 컬럼명이 길어서 as로 별명을 사용하였다.

decode 같은 경우 1:1 맵핑이지만 case 같은 경우 범위 맵핑이 가능하다. case문은 when

다음에 오는 조건식에 만족하는 값으로 대체되고 없을 경우 else 부분의 값으로 대체된다. case에서 시작해서 end에서 끝난다. 보통 컬럼명이 복잡해서 별명을 사용한다.

-사원의 커미션이 0.1이하면 하 0.2이하면 중 0.2초과면 상, null이면 'null이다' 그 밖의 경우에는 '선택받지 못한데이터' 을 출력해보자.

```
select first_name,commission_pct,
       case
           when commission_pct <=0.1 then '하'
           when commission_pct <=0.2 then '중'
           when commission_pct > 0.2 then '상'
           when commission_pct is null then 'null이다'
           else '선택받지 못한데이터'
       end as PCT
from employees;
```

다음과 같이 기술하면 '하'가 출력되지 않는다. 이유는 case문은 많은 조건중 하나만 실행되기 때문에 조건에 만족하는 when절을 찾으면 이후 작업이 종료되어 0.1같은 경우 commission\_pct <=0.2 에 만족하므로 해당 when절이 실행되고 이후 commission\_pct <=0.1 부분과 같이 조건에 맞는 부분이 있어도 더이상 실행되지 않는다.

```
select first_name,commission_pct,
       case
           when commission_pct <=0.2 then '중' --변경부분
           when commission_pct <=0.1 then '하'
           when commission_pct > 0.2 then '상'
           when commission_pct is null then 'null이다'
           else '선택받지 못한데이터'
       end as PCT
from employees;
```

문제1) departments 테이블에서 department\_name, location\_id, department\_city를 출력하자. department\_city 컬럼은 location\_id가 1500,1700,2400 일때 SanFrancisco, Seattle ,London 도시명으로 이외의 도시는 'ETC CITY'로 출력 하는 커럼이다.

문제2) employees 테이블에서 employee\_id,job\_id,salary,upgrade\_salary를 출력해 보자. upgrade\_salary은 salary의 값이 10000보다 크면 'HIGH' 작으면 'LOW' 값을 변환하여 출력하는 쿼리를 작성 해보자.

문제3) employees 테이블에서 employee\_id, hire\_date, upgrade\_date를 출력해 보자. upgrade\_date컬럼은 각 직원의 hire\_date 입사한 년도를 검색하여, 2005년 기준 '2005년이전입사'와 '2005년이후입사' , '2005년 입사' 값중 하나의 값을 반환하는 쿼리를 작성하세요. EXTRACT(YEAR FROM hire\_date) 를 출력하면 입사년이 정수로 생성 된다.

```
-- select EXTRACT(YEAR FROM hire_date) from employees;
```

```
select department_name, location_id,
decode(location_id,
      1500, 'SanFrancisco',
      1700, 'Seattle',
      2400, 'London',
      'ETC CITY') as department_city
from departments;

select employee_id,job_id,salary,
case
    when salary >10000 then 'High'
    else 'Low'
end as 'upgrade_salary'
from employees;

select EXTRACT(YEAR FROM hire_date) from employees;

select employee_id,job_id,salary,
case
    when salary >10000 then 'High'
    else 'Low'
end as upgrade_salary
from employees;
```

---

## > 17. in 연산자

---

in 연산자는 여러 개의 값 중 하나의 값이 포함되어 있으면 해당 데이터를 선택한다.

IN 및 NOT IN 조건은 SQL에서 특정 값의 존재 여부를 확인하는데 사용됩니다. 이들은 일반적으로 WHERE 절에서 사용되어 특정 조건을 만족하는 행을 필터링합니다.

예를 들어, IN을 사용하여 특정 값을 가진 행을 선택할 수 있습니다:

```
SELECT *  
FROM 테이블명  
WHERE 컬럼명 IN (값1, 값2, 값3);  
이 쿼리는 컬럼명이 값1, 값2, 값3 중 하나를 가진 행을 선택합니다.
```

반대로, NOT IN을 사용하여 특정 값을 가진 행을 제외할 수 있습니다:

```
SELECT *  
FROM 테이블명  
WHERE 컬럼명 NOT IN (값1, 값2, 값3);  
이 쿼리는 컬럼명이 값1, 값2, 값3 중 어느 하나도 아닌 행을 선택합니다.
```

예를 들어, 다음은 employees 테이블에서 job\_id가 'IT\_PROG', 'SA REP', 'HR REP' 중 하나인 행을 선택하는 쿼리입니다:

```
SELECT *  
FROM employees  
WHERE job_id IN ('IT_PROG', 'SA REP', 'HR REP');  
이와 반대로, NOT IN을 사용하여 job_id가 해당 값들 중 어느 하나도 아닌 행을 선택하는 쿼리는 다음과 같습니다:  
SELECT *  
FROM employees  
WHERE job_id NOT IN ('IT_PROG', 'SA REP', 'HR REP');
```

컬럼 in (값1,값2,값3) 해당 컬럼의 값으로 값1,값2,값3중 하나를 가지고 있으면 출력된다.

컬럼 not in (값1,값2,값3) 해당 해당 컬럼의 값으로 값1,값2,값3중 하나를 가지고 있으면 출력되지 않는다.

--커비션 비율이 0.1,0.25,0.2,0.3 갑중 하나를 가진 사원정보를 출력해 보자.

```
select * from employees where commission_pct =0.1 or commission_pct =0.25 or  
commission_pct=0.2 or commission_pct=0.3;--26  
select * from employees where commission_pct  in (0.1,0.25,0.2,0.3);--26
```

--커비션 비율이 0.1,0.25,0.2,0.3가 아닌 사원정보를 출력해 보자.

```
select * from employees where commission_pct !=0.1 and commission_pct !=0.25  
and commission_pct!=0.2 and commission_pct!=0.3; --9 null은 제외되지 않음  
select * from employees where commission_pct not in (0.1,0.25,0.2,0.3); --9  
null은 출력되지 않음
```

--출력 데이터 개수를 확인하여 기술하면 잘못된 데이터가 기술 된다.

```
select * from employees; --107
```

```
select * from employees where commission_pct !=0.1 or commission_pct !=0.25 or  
commission_pct!=0.2 or commission_pct!=0.3; -- 35 모든 데이터가 선택 될 것 같지만  
null값은 선택되지 않는다. null이 포함된 컬럼은 항상 조심하자.  
select * from employees where commission_pct is not null;--35
```

-대륙 아이디가(REGION\_ID) 1, 3, 4가 아닌 나라 정보를 출력해 보자.

```
select * from countries where region_id not in(1,3,4);
```

-부서가 80,50,null인 사원정보를 출력해 보자.

```
select * from employees where department_id in (50,80,null); --null 출력 안됨
```

```
select * from employees where department_id in (50,80)
```

```
or department_id is null;
```

```
select * from employees where nvl(department_id,0) in (50,80,0); --null을  
포함하여 출력하고 싶을때 null처리 함수를 사용한다.
```

---

## > 18. like 연산자

---

= 연산자로 문자열을 비교하면 동일한 문자열만 비교할 수 있다. like를 이용해서 부분 문자열을 비교할 수 있다.

LIKE는 SQL에서 문자열 패턴 매칭을 수행하는데 사용되는 조건 연산자입니다. LIKE를 사용하면 특정 패턴을 가진 문자열을 검색할 수 있습니다. 주로 WHERE 절에서 문자열 조건을 지정할 때 사용됩니다.

LIKE 연산자의 기본 구문은 다음과 같습니다:

SELECT \* FROM 테이블명 WHERE 열명 LIKE '패턴';

테이블명: 데이터를 검색할 테이블의 이름입니다.

열명: 검색할 열의 이름입니다.

'패턴': 검색하려는 문자열 패턴입니다. %와 \_와 같은 와일드카드 문자를 사용하여 패턴을 지정할 수 있습니다.

와일드카드 문자의 사용 예시:

%: 0개 이상의 문자를 나타냅니다.

\_: 정확히 한 개의 문자를 나타냅니다.

예를 들어, "employees" 테이블에서 이름이 "John"으로 시작하는 모든 직원을 검색하려면 다음과 같이 작성할 수 있습니다:

`SELECT * FROM employees WHERE employee_name LIKE 'John%';`

이렇게 하면 "John"으로 시작하는 모든 이름을 가진 직원들이 검색됩니다.

또 다른 예로, "employees" 테이블에서 성이 "Smith"이고 이름이 5글자인 모든 직원을 찾으려면 다음과 같이 작성할 수 있습니다:

```
SELECT * FROM employees WHERE employee_name LIKE 'Smith____';
```

여기서 \_는 하나의 문자를 나타내므로, "Smith"로 시작하고 다섯 글자인 모든 이름을 찾을 수 있습니다.

like연산자는 부분 문자열을 이용해서 원하는 문자열을 찾을 수 있다.

부분문자열이란 찾고자 하는 전체 문자열중 일부분으로 원하는 데이터를 찾을때 사용하는 문자열이다.

= 연산자로는 같은 문자열을 찾을수 있지만 부분열자열을 찾을 수는 없다.

사용방법은 다음과 같다. where 컬럼명 like '부분 문자열'  
부분문자열 안에 %는 문자가 없거나 하나 이상의 어떤 문자가 와도 상관 없다는 의미로 사용되고 밑줄 \_ 은 반드시 하나의 어떤 문자가 와야 할때 사용된다.

'myHi', 'myhi', '-hi', 'hi', 'hi-' 데이터에서 검색되는 경우를 다음 부분 문자열에서 찾아보자.

'hi' 해당 컬럼 문자열이 hi인 데이터만 검색

'hi%' hi로 시작하는 모든 문자열

'%hi' hi로 끝나는 모든 문자열

'%hi%' hi가 들어간 모든 문자열

'%%' 모든 문자열

'\_\_' 글자수가 2개인 모든 문자열 'a' (x)

'\_hi' 3글자인데 hi로 끝나는 모든 문자열

'hi\_\_' 3글자인데 hi로 시작하는 모든 문자열

'\_\_hi' 4문자이고 두번째위치에 hi가 들어간 모든 문자열

'\_\_% 두글자 이상의 모든 데이터 ('hi')(0)

where 컬럼명 not like '부분 문자열' 부분 문자열에 해당하지 않는 문자열을 출력한다.

--전화번호가 011로 시작하는 사원정보를 출력해 보자.

```
select * from employees where phone_number like '011%';
```

-- 이메일 주소 앞에서 3번째 문자가 E인 사원정보를 출력해 보자.

```
select * from employees where email like '__E%';
```

--job title에 'Account'가 들어가 있는 직업정보를 출력해 보자.

```
select * from jobs where job_title like '%Account%';
```

--전화번호에 .1343. 이 들어가지 않은 모든 사용자를 출력해 보자.

```
select * from employees where phone_number not like '%.1343.%';
```

--이름에 알파벳 a,A가 들어 있는 사원정보

```
select * from employees where first_name like '%A%' or first_name like '%a%';
```

---

## > 19. order by 절

order by 절은 검색 결과를 정렬할 때 사용한다. 데이터베이스는 기본적으로 검색 결과에 순서를 보장하지 않는다. 그동안 정렬된 것처럼 보였지만 항상 정렬을 보장하지 않는다. 검색 결과에 정렬을 보장 받고 싶다면 반드시 order by 절을 이용해서 정렬 해야 한다.

ORDER BY는 SQL 쿼리 결과를 정렬하는데 사용되는 구문입니다. ORDER BY를 사용하면 특정 열을 기준으로 결과를 오름차순(기본) 또는 내림차순으로 정렬할 수 있습니다.

기본적인 ORDER BY 구문은 다음과 같습니다:

```
SELECT 열1, 열2, ... FROM 테이블명  
ORDER BY 열1 [ASC|DESC], 열2 [ASC|DESC], ...;
```

열1, 열2, ...: 결과에 표시할 열들의 목록입니다.

테이블명: 데이터를 가져올 테이블의 이름입니다.

열1 [ASC|DESC], 열2 [ASC|DESC], ...: 정렬할 열의 목록과 정렬 방식입니다. ASC는 오름차순(기본값), DESC는 내림차순을 나타냅니다.

예를 들어, "employees" 테이블에서 이름으로 정렬하여 모든 직원을 가져오고 싶다면 다음과 같이 작성할 수 있습니다:

```
SELECT * FROM employees ORDER BY first_name;
```

만약 내림차순으로 정렬하려면 다음과 같이 작성할 수 있습니다:

```
SELECT * FROM employees ORDER BY first_name DESC;
```

또한 여러 열을 기준으로 정렬할 수 있습니다. 예를 들어, "employees" 테이블에서 부서(department)로 먼저 정렬하고 그 다음에는 봉급(salary)로 정렬하려면 다음과 같이 작성할 수 있습니다:

```
SELECT * FROM employees ORDER BY department_id asc, salary desc;
```

이렇게 하면 먼저 부서로 정렬되고, 동일한 부서 내에서는 직급으로 정렬된 결과가 반환됩니다.

```
select 컬럼선택 from 테이블선택 where 조건선택 order by 컬럼1 asc, 컬럼2 desc
```

order by 절은 검색 결과를 정렬할 때 사용한다. 일단 컬럼1으로 정렬하고 컬럼1 데이터가 같을 경우 컬럼2로 정렬 한다. 정렬을 보장받을 필요가 있다면 반드시 order by문을 사용해야 정렬이 보장 된다. asc는 오름차순 정렬이고 desc는 내림차순 정렬이다. 둘다 기술하지 않으면 asc로 정렬 된다. 오름차순이란 점점 숫자가 커지는 것을 의미하고 내림차순이란 점점 숫자가 작아지는 것을 의미한다. asc의 경우 숫자는 작은것, 문자는 사전순, 과거의 시간이 먼저 정렬 된다. null의 경우 asc일때 먼저 나오고 desc일때 나중에 나온다.

--부서번호가 오름차순으로 정렬 되도록 부서 정보를 출력해 보자.

```
select * from departments order by department_id; --기본값 asc 생략 가능  
select * from departments order by department_id asc;--상위와 동일한 출력
```

-봉급을 내림차 순으로 사원정보를 출력해 보자.

```
select * from employees order by salary desc;
```

-알파벳 오름차 순으로 국가 이름을 정렬하여 국가정보를 출력해 보자.

```
select * from countries order by country_name asc;
```

-퇴사일 오름차순으로 퇴사이력 데이터들을 출력해 보자. 퇴사일이 같을 때에는 입사일을 내림차순으로 정렬 하자.

```
select * from job_history order by end_date asc,start_date desc;
```

-봉급을 내림차순으로 봉급이 같으면 커미션 비율을 오름차 순으로 정렬하여 사원정보를 출력해 보자.

```
select * from employees order by salary desc,commission_pct asc;
```

지금까지 데이터베이스의 내용을 select 하는 방법을 공부하였다. 다음은 insert, update, delete문 사용 방법을 알아볼 예정이다.

## > 20. insert,update,delete문 사용하기

테이블에 값을 넣는 방법은 다음과 같이 하면된다.

```
insert into 테이블명(컬럼1,컬럼2,...) values (값1,값2,...);
```

테이블의 모든 데이터를 입력할때는 (컬럼1,컬럼2,...) 부분을 생략할 수 있다.

```
insert into 테이블명 values (값1,값2,..);
```

테이블을 만드는 `create table` 명령어를 알아보자.

```
create table 만들테이블명 as select문;을 이용하여 select문의 결과로 테이블을 만들 수 있다.
```

--employees 테이블과 동일한 구조와 데이터를 가지는 `copyTable`를 만들어 보자.

```
create table copyTable as select * from employees;
```

```
select * from copyTable; -- 데이터가 제대로 만들어져 있는지 확인
```

```
desc copyTable; -- 테이블구조가 동일한지 확인
```

-- 모든 컬럼 대신에 부분 컬럼을 테이블로 만들고 싶다면 다음과 같이 하면 된다.

```
create table copyTable2 as select employee_id,salary from employees;
```

-- 모든 데이터 대신에 특정 범위의 데이터를 출력하고 싶다면 다음과 같이 하면 된다.

다음을 실행하면 특정일 이전 고용된 사원의 데이터를 담는 새로운 테이블이 생긴다.

```
create table copyTable3 as select * from employees
```

```
where hire_date < '2004-01-01';
```

-- 데이터 없이 구조만 뽑고 싶다면 다음과 같이 기술하면 된다.

```
create table copyTable3 as select * from employees where 1<>1;
```

```
insert into 테이블명(컬럼1,컬럼2) select문; select문의 결과가 기존 테이블에 들어간다
```

`insert into copyTable select * from employees;` --`copyTable`에 `employees`의 모든 데이터가 추가로 들어 간다.

```
select * from copyTable; -- 들어간 데이터를 확인해 보자.
```

테이블의 데이터를 수정하는 방법을 확인해 보자.

```
update 테이블명 set 컬럼1=값1,컬럼2=값2 where 변경조건 where절에 있는 조건을 만족하는 모든 데이터들을 set에 설정한 형태로 변경한다. where절이 없으면 모든 데이터가 변경된다.
```

```
-- 모든 사원의 봉급을 10%인상해 보자. copyTable를 이용해서 질의 문을 만들어 보자.  
select * from copyTable;  
update copyTable set salary=salary*1.1;  
  
--2004년 이전에 들어온 모든 사원의 봉급을 10%인상해 보자.  
update copyTable set salary=salary*1.1 where hire_date<'2004-01-01';  
  
--여러개 업데이트 : 2004년 이전에 들어온 모든 사원의 봉급을 10%인상, 전화번호  
01011112222로 변경  
update copyTable set salary=salary*1.1,phone_number='01011112222' where  
hire_date<'2004-01-01';  
  
--사번이 100번인 사원의 봉급을 10%인상하고 전화번호 변경  
update employees set salary=salary*1.1,phone_number='01011112222' where  
employee_id=100;  
commit; -- copyTable를 전체를 삭제할 예정 이어서 commit 하여 실제로 적용하였다.
```

delete from 테이블명 where 조건식; 를 사용하여 원하는 데이터를 삭제할 수 있다.  
여기서 from은 생략할 수 있다.

--사원번호가 110 미만인 사원을 삭제해보자.

```
delete copyTable where employee_id<110;  
select * from copyTable;
```

--copyTable의 모든 데이터를 사원을 삭제해보자.

```
delete copyTable;
```

drop table 테이블명; 을 이용해서 테이블을 삭제할 수 있다.  
insert, update, delete는 commit이나 rollback을 사용해야 하지만 drop table은 다른  
곳에서 데이터 조작이 있었다면 자동으로 commit되어 적용되는 조심해서 사용하자.

-copyTable, copyTable2 테이블을 삭제해보자.

```
drop table copyTable; drop table copyTable2; --테이블 관련된 것 rollback안됨
```

---

## > 21. 중간 문제 풀이

---

다음 문제를 풀어보자. (scott 계정을 사용해서 풀어보면 된다.)

- 사원, 부서 테이블의 구조를 출력해 보자.
- 모든 부서정보를 출력해보자.
- 모든 사원정보를 출력해보자.
- 모든 사원의 이름을 출력해보자.
- 회사의 부서번호를 출력해 보자.
- 사원의 관리자, 월급, 커미션을 출력해 보자.
- 부서의 이름과 지역을 출력해 보자.
- 연봉등급과 최저 연금을 출력해 보자.
- 회사에 존재하는 사원의 작업을 출력해보자.
- 사원이 존재하는 부서 번호를 중복없이 출력해보자.
- 관리하는 사원이 존재하는 관리자 아이디를 중복없이 출력해보자.
- 연봉 등급이 3일때 받을 수 있는 최대 최소 연봉을 출력해보자.
- 연봉이 2100이상인 사원의 정보를 출력하시오.
- 부서가 20인 곳에서 일하는 사원들의 사원 정보를 출력해 보자.
- 관리자가 없는 사원의 정보를 출력해보자.
- 커미션이 없는 사원의 정보를 출력해보자.
- 사원의 봉급을 50증가한 형태로 출력해 보자.
- 사원의 현봉급보 10%증가한 형태로 출력해 보자.
- 사원의 봉급을 50증가한 형태로 컬럼명을 upgradeSalary로 출력해 보자.
- ‘xxxx 사번의 사원은 관리자가 xxx사번 입니다.’의 형태로 결과를 출력해 보자.

- 관리자가 없는 사원이 사장이다. 사장의 관리자 번호를 9999로 출력 되도록 sql를 nvl, nv12, decode 함수를 이용해서 각각 구현해 보자.
- 추가로 지급되는 돈(commission)이 없으면 0으로 출력 하도록 구현해 보자.
- 연봉이 800보다 같거나 많고 1000보다 같거나 작은 사원 정보를 출력해 보자.
- 부서 번호가 20이면서 직업이 MANAGER인 사원 정보를 출력해 보자.
- 부서 번호가 20이거나 직업이 MANAGER인 사원 정보를 출력해 보자.
- 직업이 Manager가 아닌 사원 정보를 출력해 보자.
- 커미션이 0, 500, 1400와 같은 사원정보를 출력해 보자.
- 연봉이 800보다 같거나 많고 1000보다 같거나 작은 사원 정보를 출력해 보자.
- 연봉이 700보다 작거나 1000보다 큰 사원 정보를 출력해 보자.
- 사원의 이름이 SCOTT인 사원정보를 출력해 보자.
- 사원 이름이 'A'로 시작하는 사원 정보를 출력해 보자.
- 이름에 S가 들어가는 사원 정보를 출력해 보자.
- 이름에 L이 들어가지 않은 사원 정보를 출력해 보자.
- 이름에 세번째 문자가 I 인 사원 정보를 출력해 보자.
- 사원의 이름이 E~G로 시작하는 사원정보를 출력해 보자.
- 사원의 입사일이 2004년 5월 20일~2007년 10월 10일까지의 사원 정보를 출력해 보자.
- 입사일이 83년 이후이거나 job이 SALESMAN인 모든사원을 출력해 보자.
- 20번 부서가 아닌 모든 사원 정보를 출력해 보자.
- 급여가 600에서 3000사이가 아닌 사원의 정보를 출력해보자.
- 상사가 없는 사원을 출력해 보자.
- 매니저가 7782, 2902, 2698, 7566인 사원 정보를 출력해 보자.
- 부서번호가 40, 10, 20이 아닌 사원정보를 출력해 보자.
- 사원 이름이 5 글자인 사원정보를 출력해 보자.
- 직업이 N으로 끝나는 사원정보를 출력해 보자.

- 사원 이름에 A가 들어간 사원정보를 출력해 보자.
- 이름이 S로 시작하는 사원정보를 출력해 보자.
- 관리자번호를 내림차 순으로 정렬하여 사원 정보를 출력해 보자.
- 급여가 많은 순으로 사원 정보를 내림차순 출력해보자.
- 사번, 이름, 관리자 번호, 봉급 정보가 출력되고 관리자 번호를 오름차순, 부서번호는 오름차순으로 정렬하여 사원정보가 출력되도록 구현해 보자.
- 사원 정보를 부서는 알파벳순으로 정렬하고 같은 부서 사원은 사원번호 오름차순으로 정렬해 보자.
- 직급이 ‘SALESMAN’이면 15%, ‘MANAGER’이면 10%, 이외의 직종은 5% 급여를 인상하여 사원정보를 출력해 보자.

다음은 답안이다.

```
-- desc emp;    desc dept;
-- select * from dept;
-- select * from emp;
-- select ename from emp;
-- select deptno from dept;
-- select mgr, sal, comm from emp;
-- select dname, loc from dept;
-- select grade, losal from salgrade;
-- select job from emp;
-- select DISTINCT deptno from emp;

-- select DISTINCT mgr from emp where mgr is not null;
-- select hisal, losal from salgrade where grade=3;
-- select * from emp where sal>=2100;
```

```

-- select * from emp where deptno=20;

-- select * from emp where mgr is null;

-- select * from emp where comm is null;

-- select sal+50 from emp;

-- SELECT sal*1.1 from emp;

-- select sal +50 as upgradeSalary from emp;

-- select sal, comm, sal+nvl(comm,0) upgradeSalary from emp; 다른 형태로 구현
-- 할 경우 null때문에 데이터 손실이 생길수 있으니 조심하자.

-- select empno||'사번의 사원은 관리자가'|| mgr||'사번 입니다.' FROM emp;

-- select nvl(mgr,9999) from emp;

select nvl2(mgr,mgr,9999) from emp;

select decode(mgr,null,9999,mgr) from emp;

-- select nvl(comm,0) from emp;

-- select * from emp where sal>800 and sal<1000;

-- select * from emp where deptno=20 and job='MANAGER';

-- select * from emp where deptno=20 or job='MANAGER';

-- select * from emp where not job='MANAGER';

-- select * from emp where comm in(0,500,1400);

-- select * from emp where sal between 800 and 1000;

-- select * from emp where not sal between 700 and 1000;

-- select * from emp where ename='SCOTT';

-- select * from emp where ename like 'A%';

-- select * from emp where ename like '%S%';

-- select * from emp where ename not like '%L%';

-- select * from emp where ename like '__I%';

```

```
-- select * from emp where ename between 'E' and 'H' and ename!= 'H';

-- select * from emp where hiredate between '1980.05.20' and '1981.10.10';

-- select * from emp where hiredate>'1983.01.01' or job='SALESMAN';

-- select * from emp where not deptno=20;

-- select * from emp where sal not between 600 and 3000;

-- select * from emp where mgr is null;

-- select * from emp where mgr in (7782,2902,2698,7566);

-- select * from emp where comm in (0,500,1400);

-- select * from emp where deptno not in (40,10,20);

-- select * from emp where ename like '____';

-- select * from emp where job like '%N';

-- select * from emp where ename like '%A%';

-- select * from emp where ename like 'S%';

-- select * from emp order by mgr desc;

-- select * from emp ORDER BY sal asc;

-- select empno, ename, mgr, sal from emp ORDER BY mgr asc, deptno asc;

-- select * from emp order by deptno asc, empno asc;

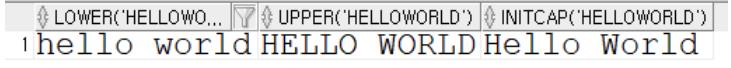
-- select      case      when job='SALESMAN' then sal*1.15
--                  when job='MANAGER' then sal*1.1
--                  else sal*1.05 end as upgradeSalary
-- from emp;
```

## > 22. 함수

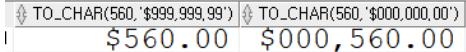
함수는 입력에 대해 특정한 규칙에 따라 출력을 내보내는 "관계 규칙"이라고 생각할 수 있습니다. 함수는 특정 컬럼의 데이터를 가공하여 원하는 결과를 얻을 때 사용한다. select 다음에 컬럼 명을 기술하면 해당 컬럼의 데이터가 출력 되는데 select 다음에 함수와 컬럼을 이용해서 함수에 따른 변경된 데이터를 기술 할 수 있다.

select 함수(컬럼명) from 테이블명; 과 같이 사용할 수 있고 where절 같은 컬럼이 사용될 수 있는 곳이면 어디서든 함수와 함께 사용할 수 있다.

다음 함수를 이해해 보자. 예제에서는 상수를 이용했지만 특정 테이블에 컬럼을 이용 할 수 있다.

함수	내용	예제
abs	절대값 출력	select -10,abs(-10) from dual; -10과 절대값 10이 출력 된다.
floor	소수점 아래를 버림	select floor(11.123) from dual; 소수점을 잘라낸 11 출력
round	소수점 반올림	select round(11.123),round(11.129,2) from dual; 11과 소수점 3째자리에서 반올림한 11.13 출력
trunc	소수점 버림	select trunc(123.129),trunc(123.129,-1) from dual; 123과 1의 자리에서 버림한 120 출력
mod	나머지구하는 함수	select mod(7,2),mod(5,3) from dual; 7을 2로 나눈 나머지 1과 5를 3으로 나눈 나머지 2가 출력된다.
lower upper initcap	소문자 대문자 시작부분만	select lower('HELLO world'),upper('HELLO world'),initcap('HELLO world') from dual; 

	대문자로 변경	
concat	문자열 합치기	<pre>select concat('he','llo') from dual;</pre> <p>두컬럼의 문자열을 합칠수 있다. hello 출력</p>
substr	부분문자열 추출	<pre>select substr('안녕 HELLO world',2,4) from dual;</pre> <p>실행결과 넝 HE 이출력된다. 문자열에서 2번째 인덱스부터 4개의 문자가 출력된다.</p>
length	문자열 개수	<pre>select length('안녕 HELLO world') from dual;</pre> <p>14가 출력 된다.</p>
ltrim rtrim trim	왼쪽공백 삭제 오른쪽공백 삭제 양쪽공백 삭제	<pre>select ' '    trim(' hello ')  ' ',ltrim(' hello '),rtrim(' hello ')  ' from dual;</pre> <p>해당 공백이 제거된다. 'hello' 같은 중간에 있는 공백은 제거되지 않는다.</p>
instr	특정 문자열의 위치를 찾음	<pre>select instr(upper('wo 안녕 HELLO world'),'WO',1,2) from dual;</pre> <p>실행결과 13</p> <p>첫번째 매개변수 문자열에서 두번째 문자열을 찾음, 세번째 매개변수는 첫번째 문자열에서 찾을 때 시작할 위치, 네번째 매개변수 찾은 동일한 문자열의 위치 중 해당번째 문자열의 위치 인덱스를 리턴</p>
months_between	두날짜의 월차를 구함	<pre>select months_between(sysdate,sysdate+60) from dual;</pre> <p>결과값 -2.03225...</p>
add_months	월을 더함	<pre>select add_months(sysdate,2) from dual;</pre> <p>결과값은 오늘 날짜보다 2달 증가한 값이 출력</p>
next_day	다음 요일의 날짜를 구함	<pre>select next_day(sysdate,7) from dual;</pre> <p>1은 일요일 2는 월요일 7은 토요일에 해당하는 다음 날짜를 출력한다.</p>

last_day	달의 마지막일을 구함	select last_day(sysdate) from dual; 해당일에 마지막 일이 출력된다.
to_char	문자로 변환	<pre>select to_char(sysdate, 'YYYY/MM:dd HH24:MI:SS') from dual;</pre> <pre>select to_char(560, '\$999,999.99'), to_char(560, '\$000,000.00') from dual;</pre>  <pre>select to_cahr(50) from dual;</pre> <p>웹에서 오라클 숫자 format으로 검색해 보자.</p>
to_date	날짜형으로 변환	<pre>select to_date('1977:05:06 14:05:06', 'YYYY:MM:DD HH24:MI:SS') from dual</pre> <p>시간 포맷에서 대소문자 구분을 안함</p>
to_number	숫자형 변환	select to_number('1') from dual

문제 1)

concat으로 3개의 문자열 ‘a’ ‘b’ ‘c’ 를 합쳐보자.

- select concat(concat('a','b'),'c') from dual;

문제 2) salary에 10000을 빼고 음수일때는 양수로 출력해 보자.

- select abs(salary -10000) from employees;

문제 3) 상위 모든 함수들을 사용해서 employees 테이블에서 사용해 보자.

---

## > 23. 오라클에서 시간계산 하기

---

시간 계산 방법을 확인해 보자. 시간관련 date자료형은 연산을 통해서 시간 연산을 할 수 있다. 현재 시간은 sysdate이며 정수 1은 시간연산 할때 하루를 의미 한다. 따라서, sysdate+1은 하루 지난 시간을 의미하고 sysdate-1은 하루 이전 시간을 의미한다.

select sysdate+1 from dual; -- 현재시간 + 1은 다음날이 된다.

select sysdate-1 from dual; -- 현재시간 - 1은 전날이 된다.

1이 하루를 의미하므로 1를 24로 나누면 1/24이 되고 1시간이 된다.

```
select to_char(sysdate, 'YYYY:MM:dd HH24:MI:SS') ,to_char(sysdate-1/24,  
'YYYY:MM:dd HH24:MI:SS') from dual;
```

```
select to_char(sysdate+1/24, 'YYYY:MM:dd HH24:MI:SS') ,to_char(sysdate-1/24,  
'YYYY:MM:dd HH24:MI:SS') from dual;
```

-- 현재시간 + 1/24은 1시간 후이다. 현재시간 - 1/24은 1시간 전이 된다.

1/24/60은 1분 1/24/60/60은 1초를 의미한다.

월이나 년 관련 데이터를 계산하려면 add\_months()함수를 사용하여야 한다.

select add\_months(sysdate,1) from dual; 1달후를 의미 하고

select add\_months(sysdate,-1) from dual; 1달전을 의미 한다.

select add\_months(sysdate,12) from dual; 1년후를 의미한다.

select add\_months(sysdate,-12) from dual; 1년전을 의미한다.

1달후를 30을 더해서 계산하면 문제가 발생할 수 있다. 1달이 31 29 28인 경우가 있다.  
1년후를 365일을 더하면 문제가 발생 할 수 있다. 1년이 366일 때가 있다.

```
select last_day(sysdate) from dual; 오늘의 마지막 달의 일자가 출력 된다. 1월이면  
31 2월이면 28이나 29 3월이면 31이 출력 된다.
```

사원이 회사에서 일한 총일수를 구하려면 입사날에서 현재 시간을 뺀다.

다음은 scott계정에 만들어져있는 emp 테이블을 사용한 것이다.

```
select ename, sysdate-hiredate from emp;
```

시간 데이터 빼기 시간 데이터는 두시간의 차가 1.5가 리턴 되며 1일하고 12시간 차이나는 것이 된다.

```
select next_day(sysdate,'월요일') from dual;  
select next_day(sysdate,1) from dual;-- 1은 일요일, 3는 화요일, 7토요일
```

현재 시간의 요일을 짹는 방법은 다음과 같다.

```
select to_char(sysdate,'DAY') from dual;
```

두시간의 월차 구하기

```
SELECT MONTHS_BETWEEN(TO_DATE('2023-07-27', 'YYYY-MM-DD'),  
TO_DATE('2022-03-15', 'YYYY-MM-DD')) AS months_difference  
FROM DUAL;
```

EXTRACT를 이용한 각 시간 출력 시분초를 처리할때 date는 관련 데이터가 없을 수 있어서 TIMESTAMP로 변환하여 사용한다.

```
SELECT hire_date,  
       EXTRACT(YEAR FROM CAST(hire_date AS TIMESTAMP)) AS 고용년도,  
       EXTRACT(MONTH FROM CAST(hire_date AS TIMESTAMP)) AS 고용월,  
       EXTRACT(DAY FROM CAST(hire_date AS TIMESTAMP)) AS 고용일,  
       EXTRACT(HOUR FROM CAST(hire_date AS TIMESTAMP)) AS 고용시,  
       EXTRACT(MINUTE FROM CAST(hire_date AS TIMESTAMP)) AS 고용분,  
       EXTRACT(SECOND FROM CAST(hire_date AS TIMESTAMP)) AS 고용초  
FROM employees;
```

입사일이 2003년 이후인 사원

```
SELECT last_name, employee_id, hire_date  
FROM employees  
WHERE EXTRACT(YEAR FROM hire_date) > 2003  
ORDER BY hire_date;
```

## 두 시간의 차를 구하는 방법

```
select sysdate,hire_date,  
trunc(months_between(sysdate,hire_date)/12,0) as year,  
mod(trunc(months_between(sysdate,hire_date),0),12) as month,  
--extract (day from (sysdate - add_Months(hire_date,12*20+1)) day to second)  
day, -- 이코드의 고정값 20과 1를 컬럼으로 연산되 가변 값으로 변경한 것이 다음 예제  
이다.  
  
extract (day from (sysdate - add_Months(hire_date,  
trunc(months_between(sysdate,hire_date),0))) day to second) day,  
extract (day from (sysdate - hire_date) day to second) day_all,  
extract (Hour from (sysdate - hire_date) day to second) Hour,  
extract (minute from (sysdate - (hire_date)) day to second) minute,  
extract (second from (sysdate - hire_date) day to second) second  
from employees;
```

extract (second from (sysdate - hire\_date) day to second) second는 sysdate와 hire\_date 간의 시간 차이를 초(second)로 추출하는 Oracle SQL의 표현 방법입니다.

sysdate: 현재 날짜와 시간을 나타내는 함수입니다.

hire\_date: 직원의 고용일자를 나타내는 컬럼(또는 값)입니다.

두 날짜 사이의 시간 차이를 초(second)로 추출하는 방법은 다음과 같이 두 단계로 이루어집니다:

sysdate - hire\_date: 먼저 sysdate에서 hire\_date를 빼서 두 날짜 사이의 시간 차이를 일(day)부터 초(second)까지의 형식으로 나타냅니다.

extract (second from ...) : 그리고 extract 함수를 사용하여 일(day)부터 초(second)까지의 시간 간격에서 초(second)만 추출합니다.

결과적으로, extract (second from (sysdate - hire\_date) day to second) second는 sysdate와 hire\_date 사이의 시간 차이를 초(second)로 추출하는 SQL 구문입니다. 이 방법은 시간 차이를 "일:시:분:초" 형식으로 추출한 후, 그 중에서 초(second) 부분만 따로 추출하는 방식입니다.

## > 24. 그룹함수

SQL에서 그룹 함수는 데이터베이스 테이블의 그룹에 대한 집계 연산을 수행하는 함수를 의미합니다. 주로 그룹화된 데이터에서 평균, 합계, 최댓값, 최솟값 등의 연산을 수행할 때 사용됩니다.

그룹함수는 특정 컬럼의 값들 중 하나의 대표 값을 얻을 때 사용하는 함수를 의미한다. 다음 그룹 함수를 확인해 보자.

연산자	의미
sum	검색된 총합을 반환 select sum(salary) from employees; 전체 사원의 급여합
avg	검색된 평균을 반환 select avg(salary) from employees; 전체 사원의 급여 평균
count	검색된 총 개수를 반환 select count(salary) from employees; 급여를 받는 전체 사원의 수
max	검색 결과중 가장 큰수 반환 select max(salary) from employees; 전체 사원중 급여를 가장 많이 받는 사원의 급여
min	검색 결과중 가장 작은수 반환 select min(salary) from employees; 전체 사원중 급여를 가장 적게 받는 사원의 급여

일반 컬럼과 함께 사용하면 출력할 데이터 개수가 달라져서 출력에 문제가 발생하여 사용할 수 없다.

`select sum(salary), salary from employees;` - 동작하지 않음

그룹함수는 null 때문에 문제가 발생 할 수 있으니 항상 머리속에 null을 생각하며 사용하자.

`select count(*), count(commission_pct) from employees;` 이 쿼리를 실행시켜 보면 다른 결과 107 35가 나오는데 null를 카운팅하지 않아서 그렇다.

`select count(*), count(nvl(commission_pct,0)) from employees;`

--nvl(commission\_pct,0)은 commission\_pct컬럼의 값이 null일때 값이 0으로 바뀌어서 null도 셀 수 있게 되어 결과가 107 107이 된다.

```
select avg(commission_pct),avg(nvl(commission_pct,0)) from employees;  
--null 때문에 다른 결과가 나온다.
```

## > 25. group by 절

SQL의 GROUP BY 절은 특정열을 그룹화하여, 그룹화한 열의 대표 값을 출력하는데 사용한다.

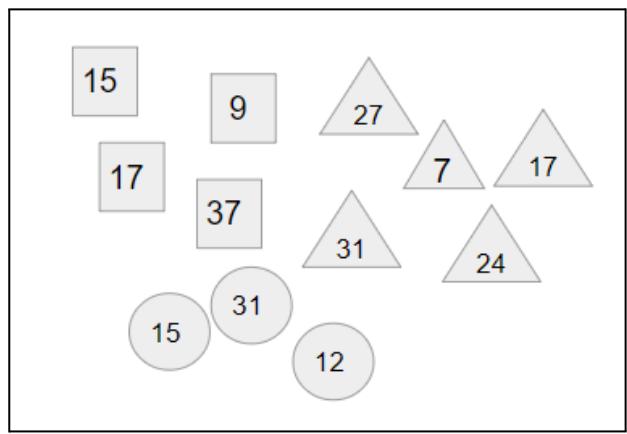
예를 들어, 직원 테이블에서 부서별로 평균 급여를 구하고자 할 때: 부서별이 그룹화한 열에 해당하고, 그룹별 평균 급여는 그룹화한 열의 대표값에 해당한다.

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

예를 들면 1번부서, 2번부서, 3번부서의 각각 부서별 평균을 원할 때 사용한다. 1번부서 평균 30, 2번부서 평균 35, 3번부서 평균 44 그룹화할 컬럼을 group by 옆에 쓰고 그룹화할 컬럼과 그룹별 대표값을 그룹 함수를 select 문에 기술한다.

위의 쿼리에서 GROUP BY department는 부서별로 데이터를 그룹화하며, AVG(salary)는 각 부서 내의 급여의 평균을 계산합니다. GROUP BY를 사용하지 않으면 전체 데이터가 하나의 그룹으로 간주되어 전체 평균이 계산됩니다.

GROUP BY를 사용하여 특정 열을 기준으로 데이터를 그룹화하고, 그룹 단위로 집계 함수를 적용할 수 있습니다.



왼쪽에 도형과 숫자 데이터로 이루어진 데이터가 있다. 모든 데이터의 숫자의 합은 이전에 배운 그룹함수를 이용해서 구할 수 있다. 그렇다면 도형별 그룹의 합을 구하는 방법이 가능한가? 그룹함수와 group by를 이용해서 가능하다.

왼쪽 이미지를 기반으로 테이블 컬럼을 확인해 보면 어떤 모양의 도형인지를 나타내는 도형 컬럼과 숫자 컬럼으로 구성되어 있다.

원 31, 사각형 37, 삼각형 31과 같은 데이터를 구하려면 도형을 기준으로 도형마다 가장 큰 숫자를 출력해야 한다.

출력 방법은 그룹 지우길 원하는 컬럼과 그룹별 원하는 그룹함수를 select에 넣고 (select 도형, max(숫자)) 그룹 지우기 원하는 그룹을 group by에 넣는다. (group by

도형. (select 도형, max(숫자) from 테이블 group by 도형)

group by 절을 이용해서 기준 컬럼이 가지고 있는 값의 종류별 그룹을 대표하는 특정 값을 얻을 수 있다. 다음은 기준 컬럼인 반 컬럼이 가지고 있는 값의 종류 1, 2, 3에 해당하는 그룹에서 대표하는 특정 값을 뽑은 것이다. 특정 값의 그룹별 대표값은 가장 높은 학생의 점수일 때 52, 55, 99, 가장 낮은 학생의 점수 일 때 23, 49, 19에 해당한다.

각 반 학생 점수	각 반 점수가 가장 높은 학생의 점수	각 반 점수가 가장 낮은 학생의 점수	각 반 학생수
반 점수	반 점수	반 점수	반 점수
1 52	1 52	1 23	1 3
1 23	2 55	2 49	2 2
1 34	3 99	3 19	3 2
2 55			
2 49			
3 19			
3 99			

학생 중 가장 점수가 높은 학생 : select max(점수) from 학생; - 99

학생 중 가장 점수가 낮은 학생 : select min(점수) from 학생; - 19

전체 학생수 : select count(점수) from 학생; - 7

각 반별 점수가 가장 높은 학생의 점수 : select 반, max(점수) from 학생 group by 반;

각 반별 점수가 가장 낮은 학생의 점수 : select 반, min(점수) from 학생 group by 반;

각 반 학생수 : select 반, count(점수) from 학생 group by 반;

select 기준컬럼, 그룹함수사용컬럼 from 테이블명 where 조건  
group by 기준컬럼 order by 컬럼 ;  
group by 다음에 오는 기준 컬럼에 들어 있는 값들의 종류를 기준으로 그룹지를 때 사용한다.

여러 개의 데이터가 하나로 합쳐져 그룹으로 표시 되므로 select 문에는 기준으로 선정된 컬럼과 그룹 함수만 올 수 있다.

다음 예제를 읽고 따라해 보자.

DEPARTMENT_ID
1
2
3
4
5
6
7
8
9
10
11
12
100

왼쪽은 employees 테이블에서 department\_id를 출력한 결과인데 부서번호가 같은 사원이 여럿 존재하는 것을 확인할 수 있고 부서 번호가 90, 60, 100인 사원 그룹에 대한 봉급 평균 즉 부서별 봉급 평균, 부서별 사원의 수, 부서별 봉급 합 등을 알고 싶을 때가 있을 것이다. 이때 사용하는 것이 group by 절이다. 여기에서는 department\_id 기준으로 그룹 지으면 group by department\_id라 기술하면 된다.

부서번호를 기준으로 부서별 사원수 부서별 평균 봉급을 출력해 보자.

```
select department_id, count(department_id), avg(salary)
from employees
group by department_id;
```

여기서 avg(salary)의 결과는 모든 사원에 대한 평균을 이야기하는 것이 아니라 널 값을 제외한 사원의 평균 값이다

널때문에 문제가 발생할 수 있으니 다음 sql과 비교해 보자.

```
select count(department_id), count(nvl(department_id, 0))
from employees
group by department_id;
```

현재 salary 컬럼에 널 값이 없지만 만약 널이 있는 컬럼이라면 널처리를 해주어야 한다  
부서별 커미션 평균을 구해 보자.

그룹과 관련 없거나 그룹함수가 아닌 컬럼은 select 옆에 기술할 수 없다.

```
select department_id, first_name, count(department_id)
from employees
group by department_id;
```

department\_id를 기준으로 그룹 지으면 first\_name은 여러 개의 결과를 가지게 되어 출력할 수 없게 된다. 상위 코드는 잘못 제작된 sql문이다.

부서번호가 100보다 작은 데이터중 부서번호로 그룹지어 그룹별 사원 평균봉급을 출력해 보자.

```
select department_id, avg(salary)
from employees
where department_id < 100
group by department_id;
```

## > 26. having 절과 sql실행 순서

HAVING 절은 SQL에서 그룹 함수와 함께 사용되며, 그룹화된 결과에 대한 조건을 지정할 때 활용됩니다. HAVING 절은 GROUP BY 절 다음에 나오며, 일반적으로 그룹 함수로 계산한 결과에 대한 조건을 정의할 때 사용됩니다.

예를 들어, 부서별로 그룹화한 후에 급여 평균이 50000 이상인 부서만 선택하고자 할 때:

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) >= 50000;
```

위의 쿼리에서 HAVING AVG(salary) >= 50000는 부서별로 그룹화된 결과 중에서 평균 급여가 50000 이상인 부서만을 선택합니다. HAVING 절은 집계 함수로 계산한 결과에 대한 조건을 지정하는 데 사용된다. 일반적으로 WHERE 절에서 사용하는 조건과는 차이가 있습니다. HAVING은 그룹 함수의 결과에 대한 조건을 검사하는 반면, WHERE은 개별 레코드의 조건을 검사합니다.

차이를 정리하자면 where절은 where절에 만족하지 않는 데이터를 제외하고 그룹을 진다. having 절은 그룹진 결과를 가지고 조건에 만족하지 않는 그룹을 제외 한다.

having 절은 group by로 그룹진 컬럼의 대표값을 비교하는 구절이다. 그룹 함수의 결과가 조건에 맞을 때 출력 된다.  
다음 내용을 확인해서 having절이 오는 위치를 확인하자.

```
select 컬럼 from 테이블명 where 조건 group by 컬럼 having 조건 order by 컬럼;
```

부서번호가 100보다 작은 데이터중 부서번호로 그룹지어 부서 그룹별 사원 평균 봉급이 8000이상인 그룹의 정보를 내림차순으로 출력해 보자.

```
select department_id,avg(salary) avg_salary
from employees
where department_id <100
group by department_id
having avg(salary)>=8000
order by avg_salary desc;
```

문제1) 다음 sql를 작성해보자. 급여가 1000이하 1억 이상인 모든 사원을 제외하고 부서 그룹별 사원 평균봉급중 8000이상인 그룹의 정보를 내림차순으로 출력해 보자.

select 문을 사용하여 다음과 같이 만들면 됩니다. 숫자는 실행 순서이다.

```
select 그룹 결과를 가지는 컬럼 --5
from 테이블명 --1
where 조건 --2
group by 컬럼 --3 where절에 위해서 걸러진 데이터만 그룹진다.
having 조건 --4
order by 컬럼 --6
```

---

## > 27. 중간 문제풀이

---

다음 문제를 scott데이터 베이스를 이용해서 출력해 보자.

--각 부서이름의 길이를 출력하시오.

--사원 이름의 길이가 6이상인 사원의 수를 출력해보자.

--사원의 이름을 소문자로 출력하시오.

--다음날 날짜를 출력해보자.

--사원의 근무 년수, 월수, 일수를 출력해 보자.

--사원들의 입사일로 부터 1년 6개월이 지난 날을 출력해 보자.

--입사일 달의 마지막 날을 출력해보자.

--오늘부터 3개월후 돌아오는 토요일 날짜를 출력해 보자.

--현재 시간 객체를 ‘XXXX-xx-xx xx:xx:xx’은 문자열로 출력해 보자.

--‘2020-6-7 22:12:11’를 시간 데이터로 만들어 출력해 보자.

--20번 부서의 커미션을 받는 사원의수를 출력해 보자.

--job이 CLERK인 사원의 평균 급여를 출력해 보자.

--사원들이 받는 커미션의 총합을 출력해보자.

--사원들이 받는 최대 최소 월급을 출력해 보자.

--부서별 급여 평균을 구해보자.

--급여별 커미션을 받는 사원의 수를 출력해보자.

--급여가 2000이상인 부서별 사원수를 출력해 보자.

--부서별 평균 급여가 1700이상인 부서번호와 평균급여를 출력하시오.

--1200이상의 급여를 받는 사원들의 부서별 평균 급여가 1900이상인 부서번호와 평균급여를 출력하시오.

--직책이 PRESIDENT 가 아닌 사원에 대한 부서별 급여 합이 6000이상인 부서와 급여합을 출력 하시오.

--20과 30 부서에서 최대 급여를 받는 사람의 최대 급여를 출력해 보자.

--select length(dname) from dept;

```
--select count(ename) from emp where length(ename)>=6;
--select lower(ename) from emp;
--select sysdate+1 from dual;
--select sysdate-hiredate as days from emp;
--select MONTHS_BETWEEN(SYSDATE,hiredate) as months from emp;
--select floor(MONTHS_BETWEEN(SYSDATE,hiredate)) as months from emp;
--select MONTHS_BETWEEN(SYSDATE,hiredate)/12 as years from emp;
--select add_months(hiredate,18) from emp;
--select last_day (hiredate) from emp;
--select next_day ( add_months(sysdate,3) ,7) from dual;
--select to_char(sysdate, 'YYYY-MM-DD HH24:MI:SS') from dual;
--select to_date('2020-6-7 22:12:11', 'YYYY-MM-DD HH24:MI:SS') from dual;
--시분초가 안 들어가 있는 것처럼 보일 수 있지만, 시분초를 포함하는 문자열로 변환하여
--찍으면 들어 있는것을 확인할 수 있다.
--select count(*) from emp where deptno = 20;
--select avg(sal) from emp where job = 'CLERK';
--select sum(nvl(comm,0)) from emp;
--select max(sal), min(sal) from emp;
--select deptno, avg(sal) from emp group by deptno order by deptno asc;
--select deptno, count(*) from emp group by deptno order by deptno;
--select deptno, count(*) from emp where sal>=2000 group by deptno order by
deptno;
--select deptno, avg(sal) from emp group by deptno having avg(sal)>=1700 order
by deptno;
--select deptno, avg(sal) from emp where sal>=1200 group by deptno having
avg(sal)>=1900 order by deptno;
--select deptno, sum(sal) from emp where job != 'PRESIDENT' group by deptno
having sum(sal)>=6000 order by deptno;
--select deptno, max(sal) from emp where deptno in(20,30) group by deptno order
by deptno asc;
```

---

## > 28. join

---

두개의 테이블을 합치는 작업을 조인 이라고 한다. 일반 적으로 데이터 베이스에서 원하는 데이터를 얻고자 할때 여러개의 테이블을 조인해야 한다. 왜 데이터를 하나의 테이블에 저장하지 않고 여러개의 테이블에 저장하는지 이유를 확인해 보자.

홍길동은 취미가 없고 홍길남은 음악, 축구가 취미이고, 홍길영은 야구가 취미이다. 사람의 취미를 저장하는 테이블을 만들어 보면 다음과 같이 생각 할 수 있다.

no	name	age	height	birthday	hobby
1	홍길동	41	169.4	2000.03.05	
2	홍길남	22	172.1	2001.02.05	음악,축구
3	홍길영	31	175.4	2002.01.21	야구

큰 문제없는 테이블 처럼 보이지만 기본적으로 관계형 데이터베이스에서 하나의 컬럼은 하나의 데이터를 가져야 하는데 hobby 컬럼의 데이터는 없을 수도 있고 데이터가 여러 개일 수도 있다. 관계형 데이터 베이스에서 이런 형태의 테이블 저장은 잘못된 형태로 본다. 전문 용어로 원자성에 위배된다.라고 한다.

원자성이란 하나의 컬럼은 쪼갤수 없는 하나의 데이터를 가진다는 것을 의미한다. 위배는 약속을 어기는 것을 의미한다.

문제를 해결하기 위해서 하나의 컬럼에 하나의 데이터가 오도록 테이블을 변경해 보면 다음과 같이 생각 할 수 있다.

no	name	age	height	birthday	hobby1	hobby2
1	홍길동	41	169.4	2000.03.05		
2	홍길남	22	172.1	2001.02.05	음악	축구
3	홍길영	31	175.4	2002.01.21	야구	

null이 들어 있는 데이터를 위쪽에 배치시키면 들어 있는 데이터가 L(엘)자 모양 처럼 된다고 해서 L자형 데이터 테이블이라고 부른다.

문제가 해결된 것처럼 보이지만 L자형 테이블에 데이터가 많아지면 null이 데이터 베이스에 많이 저장되어 저장 공간을 차지하는 문제가 발생하고, 만약 취미가 5개인 사람이 나타나면 테이블을 다시 구성해야 한다. 이런 문제를 어떻게 해결할 것인지 고민하다가 null이 생기는 컬럼과 null 이 생기지 않는 컬럼을 분리하여 두개의 테이블로 나누어 저장하기 시작했다.

humanNo	name	age	height	birthday	hobbyNo	humanNo	hobby
1	홍길동	41	169.4	2000.03.05	1	2	축구
2	홍길남	22	172.1	2001.02.05	2	2	음악
3	홍길영	31	175.4	2002.01.21	3	3	야구

왼쪽 테이블의 이름은 `human`으로 사람의 정보를 담고 있고 오른쪽 테이블은 `hobby`로 사람의 취미 정보를 담고 있다.

`human` 테이블의 `humanNo` 컬럼은 `human` 테이블 데이터 식별용으로 사용하는 컬럼이고, 테이블 `hobby` 테이블을 확인해 보면 `hobbyNo`는 `hobby` 테이블 데이터 식별용으로 사용하는 컬럼이고, `humanNo`는 해당 취미를 가진 사람 식별 번호이고, `hobby` 컬럼은 `humanNo` 컬럼에 있는 사람의 취미 정보이다.

이전 한 테이블에 `null`이 많이 생긴 이유는 하나의 테이블에 2개의 객체(사람, 취미)를 담으려고 해서 생긴 문제이다. 문제를 해결하기 위해서 `human` 객체와 `hobby` 객체를 나눠서 테이블에 저장해야 한다.

이렇게 분리된 2개의 테이블 데이터를 원래의 합쳐진 하나의 테이블 데이터로 얻기 위해서 조인을 사용한다.

왼쪽 테이블에 저장되어 있는 `name`이 홍길동 `humanNo`가 1인 데이터는 오른쪽 테이블에 `humanNo` 컬럼에 1 값이 존재하지 않으므로 1번 사람의 취미는 존재하지 않는다.

왼쪽 테이블에 저장되어 있는 `name`이 홍길남인 `humanNo`가 2인 데이터는 오른쪽 테이블에 `humanNo`가 2인 데이터는 `humanNo` 2에 `hobby` 축구인 데이터와 `humanNo` 2에 `hobby` 음악인 데이터 2개가 존재 하므로 2번 홍길남의 취미는 축구와 음악 2개가 존재한다.

왼쪽 테이블에 저장되어 있는 `name`이 홍길영인 `humanNo`가 3인 데이터는 오른쪽 테이블에 `humanNo` 3에 `hobby`가 야구인 데이터 한개 존재하므로 3번 홍길영의 취미 데이터는 1개 존재한다.

왼쪽 `human` 테이블의 하나의 데이터는 오른쪽 `hobby` 테이블의 여러개의 테이블과 관계를 가진다.

오른쪽 `hobby` 테이블의 하나의 데이터는 왼쪽 `human` 테이블의 하나의 데이터와 관계를 가진다.

왼쪽 테이블에서 `humanNo`와 오른쪽 테이블에서 `hobbyNo` 컬럼은 각 테이블에 들어 있는 값을 식별하는 용도로 같은 데이터가 들어있지 않다. 이렇게 값을 식별하기 위해서 같은 값이 들어가면 안되는 컬럼을 기본키 `primary key`라 한다. 오른쪽 테이블의 `humanNo` 컬럼은 왼쪽 테이블에 있는 `humanNo` 컬럼의 값 중 하나의 값을 가진다. 이런 컬럼을 외래키 `foreign key`라고 한다.

PK: Primary key(기본 키)는 테이블에서 각 데이터를 고유하게 식별하는 컬럼

FK: Foreign key(외래 키)는 다른 테이블의 기본 키의 값 중 하나의 값을 가지는 컬럼

BNo	BKind	BName	BArea	GNo	GPrice
1	왕포도	김명천	번동	1	30000
2	청포도	김진우	홍일동	3	20000
3	청포도	김태수	쌍문동	2	25000
4	왕포도	박지민	상계동	2	25000
5	청포도	김명천	평창동	1	30000
6	왕포도	김진우	오류동	3	20000
7	왕포도	김태수	대림동	1	30000
8	청포도	김태수	청담동	2	25000

다음 다른 예제를 통해서 테이블이 쪼개진 이유를 다른 방법으로 확인해 보자.

조인은 여러 개의 테이블을 하나의 테이블로 만들어 원하는 결과를 얻는 쿼리이다.

테이블을 나누는 이유는 하나의 테이블은 하나의

객체 정보를 데이터로 담아야 문제가 없다. 하나의 테이블에 여러 객체의 정보를 담으면 삽입이상, 간신이상, 삭제 이상이 발생 한다.

상위 테이블은 사용자가 구매한 포도박스와 포도등급 정보를 저장하는 테이블이다. 이후 사용하는 약자 B는 Box, G는 Grade를 의미한다.

테이블의 첫번째 데이터 하나를 해석해 본다면 1등급 3만원짜리 1번 박스 왕포도를 번동사는 김명천이 구매하였다는 정보를 가지고 있다. 해당 테이블은 포도 구매 관련 정보를 저장한 테이블이고 컬럼 BNo는 판매 포도의 고유식별 번호이고 BKind는 해당 박스에 들어있는 포도의 종류이고 BName은 박스를 구매한 구매자의 이름이고 BArea는 배송지역이고 GNo는 포도의 등급이고 GPrice는 등급별 포도 가격이다. 포도 구매 정보를 상위와 같은 형태로 만들어 사용하면 큰문제 없이 사용할 수 있는 것처럼 보이지만 사용하다 보면 많은 문제가 발생한다. 상위 테이블에서 발생할 수 있는 대표적인 3가지 문제점 삭제이상, 삽입이상, 변경이상을 확인해 보자.

상위 테이블에서 판매되는 모든 포도 등급을 출력 하려면 다음과 같은 쿼리가 필요 하다.  
`select distinct gno, gprice from grapeTable;` 실행결과 확인 가능한 데이터는 1등급 30000원, 2등급 25000원, 3등급 20000원이고 이는 이 가게에서 취급하는 등급에 해당한다. 어떠 경우라도 이 가게에서 취급하는 등급정보를 데이터 베이스 테이블로 부터 취득 가능 하여야 하지만, 테이블에 데이터를 삽입, 삭제, 변경하면서 원래의 정보인 포도등급 정보를 유지하지 못하는 경우가 생긴다. 다음에서 삭제, 삽입, 변경 이상을 살펴볼 예정이다.

삭제이상은 가지고 있던 데이터를 삭제하면서 발생하는 문제이다.

상위 테이블에서 bno 3,4,8을 삭제하면 포도등급 2와 관련된 데이터들이 모두 지워져 테이블을 통해서 포도등급 2의 정보를 얻어올 수 있는 방법이 없어진다.

데이터를 삭제하면 보존해야 하는 데이터가 사라지는 문제가 발생하는데 이를 삭제 이상이라고 한다.

삽입이상은 테이블에 새로운 데이터를 삽입하면서 발생하는 문제이다.

이 테이블에 포도등급 4, 가격이 15000인 포도 등급 데이터를 추가하려면 4등급 관련 제품을 구매한 사용자가 없어 gno,gprice를 제외한 모든 컬럼에 null이 들어가게 된다.

만약에 등급을 전문적으로 구분해서 관리 해야 한다면 이 테이블은 null로 가득차게 되어 저장 공간을 낭비하게 된다. 이런 현상을 삽입이상이라 한다.

### 변경이상

bno가 4인 데이터의 gprice 포도가격 컬럼의 값을 28000원으로 변경 하면 포도 가격이 포도 등급에 위해서 결정되는 상황에서 bno가 4인 포도등급 2, 포도가격 28000원 데이터와 bno가 3인 포도등급 2, 포도 가격 25000원의 데이터가 동시에 존재하여 등급은 같으나 가격이 다른 데이터가 생성된다. 현 상태에서 등급별 포도 가격을 동일하게 유지 하려면 이미 들어 있는 2등급 25000원의 데이터를 일일이 2등급 28000원의 데이터로 변경 해야 한다. 특정 데이터를 변경해서 다른데이터에 영향을 주는 이런 현상을 변경 이상이라 한다. 이런 많은 문제들을 해결하기 위해서 테이블을 쪼개서 저장하게 되었다.

BNo	BKind	BName	BArea
1	왕포도	김명천	번동
2	청포도	김진우	홍일동
3	청포도	김태수	쌍문동
4	왕포도	박지민	상계동
5	청포도	김명천	평창동
6	왕포도	김진우	오류동
7	왕포도	김태수	대림동
8	청포도	김태수	청담동

GNo	GPrice
1	30000
2	25000
3	20000

다음은 테이블을 쪼개는 방법을 확인해 보겠다.  
이전 이미지에 테이블을  
왼쪽 그림과 같이 2개의  
상자로 테이블을 분리한  
다음 오른쪽 상자에서  
반복되는 부분을 제거하면  
왼쪽 이미지 처럼 된다.  
이런 형태로 테이블을  
분리하면 원래의 데이터로  
돌아 갈 수 있는 방법이

없어 다음과 같이 공통 컬럼을 넣어서 분리 해야 한다.

BNo	BKind	BName	BArea	GNo	GPrice
1	왕포도	김명천	번동	1	30000
2	청포도	김진우	홍일동	3	20000
3	청포도	김태수	쌍문동	2	25000
4	왕포도	박지민	상계동	2	25000
5	청포도	김명천	평창동	1	30000
6	왕포도	김진우	오류동	3	20000
7	왕포도	김태수	대림동	1	30000
8	청포도	김태수	청담동	2	25000

왼쪽 이미지 처럼  
중복되는 부분을  
포함하여 테이블을  
분리하면 아래와 같이  
분리 된다.

BNo	BKind	BName	BArea	GNo	GNo	GPrice
1	왕포도	김명천	번동	1	1	30000
2	청포도	김진우	홍일동	3	2	25000
3	청포도	김태수	쌍문동	2	3	20000
4	왕포도	박지민	상계동	2		
5	청포도	김명천	평창동	1		
6	왕포도	김진우	오류동	3		
7	왕포도	김태수	대림동	1		
8	청포도	김태수	청담동	2		

이렇게 중복된 부분을 넣어서 만들면 왼쪽 테이블에서 gno가 1인 row를 오른쪽 테이블에서 gno가 1인 row와 연결하여 원래의 데이터로 돌아갈 수 있다. 이렇게 분리된 테이블을 선택된 중복된 컬럼과 연결하면 원래의 데이터로 돌아갈 수 있다. 이렇게 분리된 테이블을 합쳐서 원래 테이블의 데이터들을 만드는 과정을 조인이라고 한다.

상위 테이블을 만들고 데이터를 입력해 보자.

```
drop table BTable;
create table BTable(
    BNO number(10),
    BKind nvarchar2(30),
    BName nvarchar2(30),
    BArea nvarchar2(30),
    GNo number(10)
);
drop table GTable;
create table GTable(
    GNo number(10),
    GPrice number(10)
);
insert into BTable values (1,'왕포도','김명천','번동',1);
insert into BTable values (2,'청포도','김진우','홍일동',3);
insert into BTable values (3,'청포도','김태수','쌍문동',2);
insert into BTable values (4,'왕포도','박지민','상계동',2);
insert into BTable values (5,'청포도','김명천','평창동',1);
insert into BTable values (6,'왕포도','김진우','오류동',3);
insert into BTable values (7,'왕포도','김태수','대림동',1);
insert into BTable values (8,'청포도','김태수','청담동',2);
insert into GTable values (1,'30000');
insert into GTable values (2,'25000');
insert into GTable values (3,'20000');
commit;
```

```
select * from BTable;
select * from GTable;
```

테이블을 분리하였으면 두테이블을 통해서 원래의 데이터를 얻어오는 방법이 존재해야 하고 이런 방법을 조인이라고 한다. 두 테이블의 데이터를 합쳐 하나의 데이터를 만드는 여러 가지 방법을 이후 하나씩 확인해 볼 예정이다.

## 1. 크로스 조인

**cross Join( 크로스 조인)**은 두 테이블이 가지고 있는 모든 데이터를 합쳐서 만들 수 있는 모든 데이터를 만들어 보여주는 작업이다. 다음은 2개의 테이블을 크로스 조인하는 예제이다.

```
select * from BTable,Gtable;
```

BTable			GTable							
...	BName	GNo		GNo	GPrice	...	BName	GNo	GNo	GPrice
...	전영호	1		1	30000	...	전영호	1	1	30000
...	김기홍	3		2	25000	...	전영호	1	2	25000
...	전영호	2		3	20000	...	김기홍	3	1	30000
...	김동민	2				...	김기홍	3	2	25000
						...	김기홍	3	3	20000
						...	전영호	2	1	30000
						...	전영호	2	2	25000
						...	전영호	2	3	20000
						...	김동민	2	1	30000
						...	김동민	2	2	25000
						...	김동민	2	3	20000

왼쪽 테이블을 BTable 오른쪽 테이블을 GTable이라고 할때 합치는 방법은 상위 처럼 BTable 각각의 모든 데이터를 GTable 각각의 모든 데이터와 일일이 하나씩 합친 모든 결과를 얻는 방법이다. 크로스 조인이라 한다. 상위 이미지를 확인해 보자.

`select * from BTable,Gtable;` 이렇게 하면 두 테이블에서 데이터를 가지고 합칠 수 있는 모든 데이터와 컬럼이 출력 된다.

두 테이블을 합친 결과 데이터는 두 테이블에 있는 모든 컬럼을 하나의 데이터로 표현할 수 있어야 하기 때문에 왼쪽 테이블의 컬럼수가 5개이고, 오른쪽 테이블의 컬럼수가 2개라면 양쪽 테이블의 컬럼수를 더한 7이 되어야 한다.

두 테이블의 모든 데이터를 합쳐 나올 수 있는 모든 데이터는 왼쪽 테이블에 데이터가 4개 오른쪽 테이블 3개가 있다면 실행 결과 총 데이터 수는 12개가 된다. 아까 만든 데이터를 이용해서 직접 확인해 보자.

다음 예제를 통해서 크로스 조인을 이해해 보자.

```
select * from employees;--107
```

```

select * from departments;--27
select * from employees,departments;
select count(*) from employees,departments;--2889 = 107*27
전체 데이터 개수는 employees테이블 107개 departments테이블 27개 크로스 조인한
결과는 2889 = 107*27 이 된다.

```

## 2. 동등조인

equi Join(이퀴 조인) 특정 컬럼 값이 일치되는 데이터(row)만 합쳐서 출력하는 방법이다. 동등조인이라고도 한다.

크로스 조인은 두 테이블을 가지고 만들수 있는 모든 데이터를 만들어 보여주지만  
이퀴조인은 크로스 조인으로 만들어진 모든 데이터중 특정 컬럼이 같은 데이터만 보여준다.  
 다음 이미지에서 크로스 조인결과 데이터중에서 GNo컬럼 2개를 확인해 값이 같은 데이터만 출력 하였다. 두 테이블을 gno 컬럼으로 equi join한 결과이다.

BTable				GTable	
...	BNo	BName	GNo	GNo	GPrice
...	5	전영호	1	1	30000
...	1	김기홍	3	2	25000
...	5	전영호	2	3	20000
...	2	김동민	2		

...	BNo	BName	GNo	GNo	GPrice
...	5	전영호	1	1	30000
...	1	김기홍	3	3	20000
...	5	전영호	2	2	25000
...	2	김동민	2	2	25000

select \* from BTable,GTable 과 같이 크로스 조인하여 만들수 있는 모든 데이터를 만든 다음 where BTable.GNo = GTable.Gno 과 같이 두 테이블에서 특정 컬럼이 같은 데이터만 뽑아서 출력하면 된다. select \* from BTable,GTable where BTable.GNo = GTable.Gno;  
 다음과 같이 기술하면 두 테이블의 GNo가 같은 데이터만 얻을 수 있다. 실행 결과 4개의 데이터가 출력된다.

...	BName	GNo	GNo	GPrice
...	전영호	1	1	30000
...	전영호	1	2	25000
...	전영호	1	3	20000

왼쪽 테이블에서 GNo컬럼의 값이 1인 하나의 데이터가 클로스 조인하면 오른쪽 테이블의 GNo컬럼의 값 1,2,3 3개의 값이랑 매핑되어 3개의 데이터가 만들어 진다음 이중에서 왼쪽 GNo컬럼과 오른쪽 GNo 컬럼 값이 같은 첫번째 데이터만 출력이 된다.

원하는 컬럼만 출력하고 싶다면 select 문 다음에 다음과 같이 테이블명.컬럼명 형태로 ,로 구분하여 여러개 기술하면 된다.

```

select BTable.* , GTable.GNo, GTable.GPrice from BTable,GTable
where BTable.GNo = GTable.Gno;

```

다음과 같이 별명을 사용할 수 있다.

```
select B.* , G.GNo , G.GPrice from BTable B , GTable G  
where B.GNo = G.Gno;
```

```
insert into BTable values (9 , '청포도' , '김태수' , '청담동' , 4);
```

만약 왼쪽 테이블에 상위와 같은 데이터를 넣고 조인을 시도한다면 dno컬럼에 매칭되는 4 데이터가 오른쪽 테이블에 없어서 동등 조인결과 해당 데이터 관련 데이터는 출력되지 않는다.

다음은 사원의 이름과 부서 이름을 출력하는 쿼리이다.

```
select employees.first_name , department_name  
from employees , departments  
where employees.department_id = departments.department_id;  
출력 개수를 확인해보면 사원수가 107인데 조인 결과 106개가 된것을 확인할 수 있다.  
select count(*) from employees , departments  
where employees.department_id = departments.department_id;
```

employees.department\_id 부서 아이디가 null인 사원이 존재하여 106개의 데이터가 된 것이니 걱정하지 말자.

```
select * from employees  
where employees.department_id is null;
```

다음 문제를 확인해 보자.

-모든 사원정보와 해당 사원의 부서 정보를 모두 출력해 보자.

```
select * from employees , departments  
where employees.department_id = departments.department_id;  
-다음과 같은 방법으로 접근 가능 하다. sql은 대소문자 구분을 하지 않지만 데이터는  
구분한다. sql 문법에 사용되는 ValueS 와 values는 같으나 데이터의 값으로 사용되는  
문자열 'ValueS' 와 'values'는 다르다.
```

```
select employees.employee_id , d.*  
from employees , departments D  
where employees.department_id = D.department_id;  
--부서가 100인 사원의 이름과, 부서이름을 출력해보자.  
select employees.department_id , employees.first_name , department_name  
from employees , departments  
where employees.department_id = departments.department_id  
and employees.department_id = 100;  
--급여가 10000이하인 사원의 사번,봉급과 부서이름을 사번으로 정렬 하여 출력해 보자.  
select employees.employee_id , employees.salary , departments.department_name  
from employees , departments  
where employees.department_id = departments.department_id  
and employees.salary <= 10000
```

```

order by employee_id;
--국가별(countries) 대륙(region)정보를 출력해 보시오.
select * from countries;
select * from regions;
select * from countries,regions
where countries.region_id=regions.region_id;
--사원의 사번,봉급,직종 이름, 해당 직종의 최소급여, 최대급여 정보를 출력해 보자.
select * from employees;           --사원테이블
select * from jobs;                --직종테이블
select employees.employee_id, employees.salary, jobs.job_Title,
jobs.min_salary, jobs.max_salary
from jobs,employees
where jobs.job_id=employees.job_id;

```

-ansi 표준 형태로 조인문을 작성하면 다음과 같다. 지금까지 설명한 방식은 오라클에서 사용할 수 있는 조인 방식이다. 다른 제품의 DB에서는 동작할 수도 있고 하지 않을 수도 있다. 다른 모든 DB에서 동작하게 하려면 ansi 형태의 sql로 작성할 수 있다. 모든 종류의 데이터베이스에서 돌아간다는 장점이 있지만 복잡해서 잘 사용하지 않는다.

동등조인을 ansi 표준 형태로 다음과 같이 기술한다.

```

select employees.department_id,employees.first_name,department_name
from employees join departments
on employees.department_id =departments.department_id;

```

### 3. self join

self join은 같은 테이블을 조인하여 원하는 결과를 얻는 것을 의미한다.

회사에 사원으로 입사했다고 생각해 보자. 본인 상사도 있고 부하직원도 있을 것이다. 회사의 사원은 부하직원이자 상사이다. 상사도 부하직원도 결국 사원이다. 따라서, 둘다 사원 테이블에 저장된다. 사원 테이블의 영어이름은 employees이다.

사원 정보를 얻고 싶을때 사원 테이블을 검색한다.

```
select * from employees;
```

```
select * from employees;
```

질의 결과 x  
SQL | 50개의 행이 인출됨(0.007초)

	EMPLOYEE_ID	FIRST_NAME	EMAIL	PH...	HIRE_DATE	JOB_ID	SALARY	MANAGER_ID	DEPARTMENT_ID
1	198	Donald	.......	07.....	2600.	124	50		
2	199	Douglas	.......	08.....	2600.	124	50		
3	200	Jennifer	.......	03.....	4400.	101	10		
4	201	Michael	.......	04.....	1.....	100	20		
5	202	Pat	.......	05.....	6000.	201	20		
6	203	Susan	.......	02.....	6500.	101	40		
7	204	Hermann	.......	02.....	1.....	101	70		
8	205	Shelley	.......	02.....	1.....	101	110		
9	206	William	.......	02.....	8300.	205	110		
10	100	Steven	.......	03.....	2.....	(null)	90		
11	101	Neena	.......	05.....	1.....	100	90		

부하직원 정보를 얻고 싶을 때 사원 테이블을 검색한다. 결국 사원 테이블이 부하직원 테이블이다.

select \* from employees 부하직원;

```
select * from employees 부하직원;
```

질의 결과 x  
SQL | 50개의 행이 인출됨(0.007초)

	EMPLOYEE_ID	FIRST_NAME	EMAIL	PH...	HIRE_DATE	JOB_ID	SALARY	MANAGER_ID	DEPARTMENT_ID
1	198	Donald	.......	07.....	2600.	124	50		
2	199	Douglas	.......	08.....	2600.	124	50		
3	200	Jennifer	.......	03.....	4400.	101	10		
4	201	Michael	.......	04.....	1.....	100	20		
5	202	Pat	.......	05.....	6000.	201	20		
6	203	Susan	.......	02.....	6500.	101	40		
7	204	Hermann	.......	02.....	1.....	101	70		
8	205	Shelley	.......	02.....	1.....	101	110		
9	206	William	.......	02.....	8300.	205	110		
10	100	Steven	.......	03.....	2.....	(null)	90		
11	101	Neena	.......	05.....	1.....	100	90		

상사 정보를 얻고 싶다면 상사 테이블을 검색 해야 한다.

select \* from employees 상사;

```
select * from employees 상사;
```

질의 결과 x  
SQL | 50개의 행이 인출됨(0.007초)

	EMPLOYEE_ID	FIRST_NAME	EMAIL	PH...	HIRE_DATE	JOB_ID	SALARY	MANAGER_ID	DEPARTMENT_ID
1	198	Donald	.......	07.....	2600.	124	50		
2	199	Douglas	.......	08.....	2600.	124	50		
3	200	Jennifer	.......	03.....	4400.	101	10		
4	201	Michael	.......	04.....	1.....	100	20		
5	202	Pat	.......	05.....	6000.	201	20		
6	203	Susan	.......	02.....	6500.	101	40		
7	204	Hermann	.......	02.....	1.....	101	70		
8	205	Shelley	.......	02.....	1.....	101	110		
9	206	William	.......	02.....	8300.	205	110		
10	100	Steven	.......	03.....	2.....	(null)	90		
11	101	Neena	.......	05.....	1.....	100	90		

사원 테이블이 상사 테이블도 되고 부하직원 테이블도 된다.

사원 테이블을 부하사원 테이블로 검색을 하면 검색된 데이터의 employee\_id는 부하사원 번호가 되고 manager\_id는 부하직원의 상사 번호가 된다. 이때 상사 번호만 있지 상사이름은 알 수 없다.

```
SELECT 부하직원.employee_id AS 부하직원번호, 부하직원.manager_id AS 부하의상사번호
FROM employees 부하직원;
```

```
SELECT 부하직원.employee_id AS 부하직원번호, 부하직원.manager_id AS 부하의상사번호
FROM employees 부하직원;
```

질의 결과 x  
SQL | 50개의 행이 인출됨(0.008초)

부하직원번호	부하의상사번호
198	124
199	124
200	101
201	100

사원 테이블을 상사 테이블로 검색을 하면 employee\_id는 상사들의 아이디가 되고 first\_name은 상사들의 이름이 된다.

```
SELECT employee_id AS 상사번호, first_name AS 상사이름
FROM employees 상사;
```

```
SELECT employee_id AS 상사번호, first_name AS 상사이름
FROM employees 상사;
```

질의 결과 x  
SQL | 50개의 행이 인출됨(0.014초)

상사번호	상사이름
174	Ellen
166	Sundar
130	Mozhe
105	David
204	Hermann

174번 사원의 상사번호가 149일때 사원 테이블에서 174번 부하사원의 상사번호를 출력하는 방법은 다음과 같다.

```
SELECT employee_id AS 부하사원사번, manager_id AS 상사사번
FROM employees 부하직원
WHERE employee_id = 174;
```

174,149의 실행 결과를 얻는다.

```
SELECT employee_id AS 부하사원사번, manager_id AS 상사사번
FROM employees 부하직원
WHERE employee_id = 174;
```

질의 결과 x  
SQL | 인출된 모든 행: 1(0.002초)

부하사원사번	상사사번
174	149

149인 상사의 이름을 출력하려면 다음과 같다.

```
SELECT employee_id AS 상사사원번호, first_name AS 상사이름
FROM employees 상사
WHERE employee_id = 149;
```

```

SELECT employee_id AS 상사사원번호, first_name AS 상사이름
FROM employees 상사
WHERE employee_id = 149;

```

질의 결과 x  
SQL | 인출된 모든 행: 1(0,004초)

상사사원번호	상사이름
149	Eleni

부하직원 테이블에서 174 부하직원아이디의 상사아이디 149이란 데이터에서 부하직원 번호는 부하직원.employee\_id, 상사아이디는 부하직원.manager\_id에 해당한다.  
 부하직원.manager\_id가 149이라면 해당 부하직원의 상사가 149이므로 상사 테이블에서 상사.employee\_id가 174번 부하직원의 149인 데이터가 상사의 정보가 된다.  
 부하직원.employee\_id= 상사.employee\_id = 149이라는 이야기이다.

만약, 상사와 부하직원 간에 조인 관계가 같은 테이블에 존재하면 조인하여 원하는 결과를 얻어야 하고 이를 셀프 조인이라 부른다.

모든 사원으로 상사와 부하직원 관계로 짹지를수 있는 모든 데이터를 출력해 보자.

```

select * from employees 상사,employees 부하직원;
한 사원의 모든 짹이 출력 된다.

SELECT 부하직원.employee_id AS 부하직원_ID, 부하직원.first_name AS 부하직원_이름,
       상사.employee_id AS 상사_ID, 상사.first_name AS 상사_이름
  FROM employees 부하직원, employees 상사
 order by 부하직원.employee_id,상사.employee_id;

```

```

SELECT 부하직원.employee_id AS 부하직원_ID, 부하직원.first_name AS 부하직원_이름,
       상사.employee_id AS 상사_ID, 상사.first_name AS 상사_이름
  FROM employees 부하직원, employees 상사
 order by 부하직원.employee_id,상사.employee_id;

```

질의 결과 x  
SQL | 50개의 행이 인출됨(0,008초)

	부하직원_ID	부하직원_이름	상사_ID	상사_이름
1	100	Steven	100	Steven
2	100	Steven	101	Neena
3	100	Steven	102	Lex
4	100	Steven	103	Alexander
5	100	Steven	104	Bruce
6	100	Steven	105	David
7	100	Steven	106	Valli
8	100	Steven	107	Diana
9	100	Steven	108	Nancy
10	100	Steven	109	Daniel
11	100	Steven	110	John
12	100	Steven	111	Ismael
13	100	Steven	112	Jose Manuel
14	100	Steven	113	Luis
15	100	Steven	114	Pen

상위 결과중 실제로 부하직원.manager\_id = 상사.employee\_id; 인 데이터만 뽑으면, 부하직원과 상사의 모든 짹에서 부하직원과 상사의 데이터 상의 실제 짹만 남게된다.

```
select * from employees 상사, employees 부하직원  
WHERE 부하직원.manager_id = 상사.employee_id;
```

결론적으로 모든 부하직원의 상사이름을 출력하려면 다음과 같이 하면되고 같은 테이블을 조인해야 하므로 셀프조인이라고 한다.

```
SELECT 부하직원.employee_id AS 부하직원_ID, 부하직원.first_name AS 부하직원_이름,  
       상사.employee_id AS 상사_ID, 상사.first_name AS 상사_이름  
  FROM employees 부하직원, employees 상사  
 WHERE 부하직원.manager_id = 상사.employee_id  
order by 부하직원.employee_id, 상사.employee_id;
```

The screenshot shows a SQL query execution interface. The query is:

```
SELECT 부하직원.employee_id AS 부하직원_ID, 부하직원.first_name AS 부하직원_이름,  
       상사.employee_id AS 상사_ID, 상사.first_name AS 상사_이름  
  FROM employees 부하직원, employees 상사  
 WHERE 부하직원.manager_id = 상사.employee_id  
order by 부하직원.employee_id, 상사.employee_id;
```

The results table has four columns: 부하직원\_ID, 부하직원\_이름, 상사\_ID, and 상사\_이름. The data is as follows:

	부하직원_ID	부하직원_이름	상사_ID	상사_이름
1	101	Neena	100	Steven
2	102	Lex	100	Steven
3	103	Alexander	102	Lex
4	104	Bruce	103	Alexander
5	105	David	103	Alexander
6	106	Valli	103	Alexander
7	107	Diana	103	Alexander
8	108	Nancy	101	Neena
9	109	Daniel	108	Nancy
10	110	John	108	Nancy
11	111	Ismael	108	Nancy
12	112	Jose Manuel	108	Nancy
13	113	Luis	108	Nancy
14	114	Den	100	Steven
15	115	Alexander	114	Den
16	116	Shelli	114	Den

```
SELECT 부하직원.employee_id AS 부하직원_ID, 부하직원.first_name AS 부하직원_이름,  
       상사.employee_id AS 상사_ID, 상사.first_name AS 상사_이름  
  FROM employees 부하직원, employees 상사  
 WHERE 부하직원.manager_id = 상사.employee_id;
```

이렇게 함으로써, 'employees' 테이블에서 셀프 조인을 사용하여 부하직원과 상사 간의 관계를 파악할 수 있습니다. 이를 통해 각 사원의 상사 이름을 얻을 수 있게 됩니다.

self join(셀프 조인)은 하나의 테이블을 가지고 두개의 테이블 처럼 조인하는 것을 의미한다.

다음을 따라해보자.

1. 사원의 a.employee\_id,a.first\_name,a.manager\_id 정보를 출력해보자.

```
select a.employee_id,a.first_name,a.manager_id  
from employees a
```

2. 모든 사원이 짹을 이루었을때 발생할수 있는 데이터를 모두 출력하려면 사원 테이블을 하나더 만들어서 사원과 사원 테이블을 클로스 조인하면된다.

```
select a.employee_id,a.first_name,a.manager_id ,  
b.employee_id,b.first_name,b.manager_id  
from employees a,employees b
```

3. 107\*107개의 모든 사원들이 짹을 이룬 데이터중 a.manager\_id와 b.employee\_id가 같은 데이터만 남기면 회사의 모든 사원 짹중 사원과 관리자로 이루어진 짹만 남겨진다.

```
select a.employee_id,a.first_name,a.manager_id ,  
b.employee_id,b.first_name,b.manager_id  
from employees a,employees b  
where a.manager_id=b.employee_id;
```

4. manager\_id가 null인 사원이 있어서 106개의 데이터가 출력된다.

5. 조인 작업을 같은 테이블을 가지고 하기 때문에 셀프 조인이라고 한다.

6. 최종 목표는 사원의 관리자 이름을 출력하는 것이다.

- 사원번호(employee\_id)와 사원번호에 해당하는 관리자이름(first\_name)를 출력 하는 sql를 작성한다고 생각해 보자.

사원번호와 해당 관리자 이름을 하나의 테이블을 사용해서 처리할수 있을 것 같지만 잘 생각해 보면 불가능하다.

```
select first_name, employee_id, manager_id, (관리자의 이름은?)  
from employees;
```

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	S.KING	SALESMAN	24000	0.15	(null)	90
2	101	Neena	Kochhar	N.KOCHHAR	SALESMAN	17000	0.15	100	90
3	102	Lex	De Haan	L.DEHAN	SALESMAN	17000	0.15	100	90
4	103	Alexander	Hunold	A.HUNOLD	SALESMAN	9000	0.15	102	60
5	104	Bruce	Ernst	B.ERNST	SALESMAN	6000	0.15	103	60
6	105	David	Austin	D.AUSTIN	SALESMAN	4800	0.15	103	60
7	106	Valli	Patahalia	V.PATHALIA	SALESMAN	4800	0.15	103	60

상위 이미지에서 선택된 데이터의 관리자 이름은 Lex인데 선택된 데이터 만으로 관리자 이름을 출력할 수 없다. 만약 아래와 이미지와 같은 테이블이 하나 더 있다면 상위 테이블 MANAGER\_ID와 아래 테이블의 EMPLOYEE\_ID를 조인하여 103번 사원의 관리자 102번 사원의 이름 LEX를 얻을 수 있다.

	EMPLOYEE_ID	MANAGER_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER
1	100	(null)	Steven	King	SKING	515.123.4567
2	101	100	Neena	Kochhar	NKOCHHAR	515.123.4568
3	102	100	Lex	De Haan	LDEHAAN	515.123.4569
4	103	102	Alexander	Hunold	AHUNOLD	590.423.4567
5	104	103	Bruce	Ernst	BERNST	590.423.4568
6	105	103	David	Austin	DAUSTIN	590.423.4569

상위 두 테이블을 잘 생각해 보면 같은 테이블이다. 관리자 이름으로 올수 있는 컬럼이 없었지만, 잘 생각해 보면 관리자도 사원이므로 manager\_id의 값은 employee\_id컬럼의 값중 하나의 값을 가진다. 결론은, 사원의 관리자 이름은 사원테이블의 관리자 번호를 찾은 다음 찾은 관리자 번호에 해당하는 사원테이블의 사원번호와 조인해서 관리자 번호에 해당하는 사원 번호의 사원이름을 출력하면 된다.

하나의 테이블에서 직접 manager\_id에 해당하는 관리자의 이름을 출력할 수는 없지만 사원 테이블을 관리자 테이블이라고 생각하고 사원 테이블의 관리자 아이디를 관리자 테이블의 사원 아이디와 동등조인 하면 원하는 결과를 얻을 수 있다.

1. `select * from employees` 결과에서 2번째 데이터를 뽑아 왔다면 사원번호 101번 사원의 이름은 neena 관리자 번호는 100이라는 정보는 뽑을 수 있지만 101번의 관리자 이름이 steven이라는 정보는 알수 없다. 관리자 이름 정보를 알려면 사원 테이블에서 관리자 번호 100를 이용해서 사원테이블에서 사번이 100인 사원이름을 뽑아와야 관리자 100번의 이름 Steven를 알 수 있다.

2. 이전에 사번이 101인 사원의 매니저 번호가 100이라는 것을 확인 하였다. 다음과 같이 얻을 수 있다.

```
select e1.employee_id ,e1.manager_id          -- 101,100출력 (100 == manager_id)
from employees e1
where e1.employee_id=101;
```

3. 매니저도 사원 이어서 사원 테이블에서 찾으면 정보를 얻을 수 있다.  
매니저 사번 (manager\_id) 100를 가지고 사원 테이블(employees)의 사원번호 (employee\_id)에서 100을 찾으면 검색된 결과에서 매니저의 이름인 100번 사원의 이름을 찾을 수 있다.

```
select e2.employee_id, e2.first_name -- 100, steven 출력
from employees e2
where e2.employee_id=100;
```

4. 잘 생각해보면 e1테이블에서는 사원 정보를 얻을수 있어서 e1을 사원테이블 이라 생각하고 e2 테이블에서 매니저 정보를 얻을 수 있어서 e2를 매니저 테이블 이라고 생각하면, e1테이블과 e2테이블을 조인하여 사원과 관리자 정보를 뽑아 낼 수 있을 것이다.

```
select e1.*, e2.* , e1.employee_id, e2.first_name
from employees e1, employees e2
where e1.manager_id=e2.employee_id;
```

아래 이미지를 확인해 보면 같은 테이블에서 각각 다른 데이터를 뽑아서 두 테이블(e1,e2)을 만들고 이 두 테이블을 합쳐 새로운 데이터를 얻으려 하고 있다.

점선 화살표가 두 테이블을 연결하여 얻고자 하는 데이터이다. 왼쪽 테이블의 관리자ID (MANAGER\_ID)와 오른쪽 테이블의 사원ID (EMPLOYEE\_ID) 를 짹지어 두 테이블이 합쳐진다. 왼쪽 결과물을 사원 정보 테이블로 사원정보를 오른쪽은 관리자 테이블로 관리자 정보를 출력할 수 있다.

왼쪽 테이블의 2번째 데이터 사원 아이디 101과 관리자 아이디 100 데이터를 오른쪽 테이블의 사원 아이디 100번과 이름 steven 정보를 조인을 통해서 사번 101의 관리자 사번 100번과 100번의 사원이름 steven 즉 101번사원의 관리자 이름 steven을 얻어 낼 수 있다.

select e1.employee_id, e1.manager_id from employees e1;		select e2.employee_id, e2.first_name from employees e2;																																																							
<table border="1"> <thead> <tr> <th></th><th>EMPLOYEE_ID</th><th>MANAGER_ID</th></tr> </thead> <tbody> <tr> <td>1</td><td>100</td><td>(null)</td></tr> <tr> <td>2</td><td>101</td><td>100</td></tr> <tr> <td>3</td><td>102</td><td>100</td></tr> <tr> <td>4</td><td>103</td><td>102</td></tr> <tr> <td>5</td><td>104</td><td>103</td></tr> <tr> <td>6</td><td>105</td><td>103</td></tr> <tr> <td>7</td><td>106</td><td>103</td></tr> <tr> <td>8</td><td>107</td><td>103</td></tr> <tr> <td>9</td><td>108</td><td>101</td></tr> </tbody> </table>			EMPLOYEE_ID	MANAGER_ID	1	100	(null)	2	101	100	3	102	100	4	103	102	5	104	103	6	105	103	7	106	103	8	107	103	9	108	101	<table border="1"> <thead> <tr> <th></th><th>EMPLOYEE_ID</th><th>FIRST_NAME</th></tr> </thead> <tbody> <tr> <td>56</td><td>156</td><td>Janette</td></tr> <tr> <td>53</td><td>100</td><td>Steven</td></tr> <tr> <td>54</td><td>101</td><td>Neena</td></tr> <tr> <td>55</td><td>173</td><td>Sundita</td></tr> <tr> <td>56</td><td>137</td><td>Renske</td></tr> <tr> <td>57</td><td>127</td><td>James</td></tr> <tr> <td>58</td><td>165</td><td>David</td></tr> </tbody> </table>			EMPLOYEE_ID	FIRST_NAME	56	156	Janette	53	100	Steven	54	101	Neena	55	173	Sundita	56	137	Renske	57	127	James	58	165	David
	EMPLOYEE_ID	MANAGER_ID																																																							
1	100	(null)																																																							
2	101	100																																																							
3	102	100																																																							
4	103	102																																																							
5	104	103																																																							
6	105	103																																																							
7	106	103																																																							
8	107	103																																																							
9	108	101																																																							
	EMPLOYEE_ID	FIRST_NAME																																																							
56	156	Janette																																																							
53	100	Steven																																																							
54	101	Neena																																																							
55	173	Sundita																																																							
56	137	Renske																																																							
57	127	James																																																							
58	165	David																																																							

생각해 보면 사원의 사번과 관리자의 사번은 결국 사원번호여서 우리가 원하는 값을 얻으려면 같은 테이블을 조인해야 얻을 수 있고 이를 셀프 조인이라고 한다.

employees테이블을 셀프 조인하여 사원 정보와 관리자 정보를 출력한 sql이 다음과 같다.

```

select e1.employee_id, e1.manager_id, e2.employee_id, e2.first_name
from employees e1, employees e2
where e1.manager_id=e2.employee_id;

```

질의 결과 x  
SQL | 인출된 모든 행: 106(0.073초)

EMPLOYEE_ID	MANAGER_ID	EMPLOYEE_ID_1	FIRST_NAME
52	114	100	Steven
53	102	100	Steven
54	101	100	Steven
55	205	101	Neena
56	204	101	Neena
57	203	101	Neena
58	200	101	Neena
59	108	101	Neena
60	199	124	Kevin
61	198	124	Kevin
62	197	124	Kevin
63	196	124	Kevin
64	144	124	Kevin

#### 4. outer join

##### outer join(외부 조인)

두 테이블을 이퀴 조인할때 선택된 두 컬럼에 일치하는 값이 들어 있지 않다면 이퀴 조인 결과 매칭되는 데이터가 없으므로 출력 되지 않는다. 만약 한쪽 테이블에만 있는 데이터를 사라지지 않고 출력을 원한다면 없는쪽 테이블 컬럼의 데이터에 null을 넣어 출력해야 하는데 이렇게 이퀴조인시 일치하는 데이터가 없을때 한쪽 커럼에 null을 넣어 출력하는 방법을 외부 조인이라고 한다.

다음과 같은 사원, 부서 테이블이 있다고 생각해 보자.

사원테이블	
사원번호	부서번호
100	10
101	11
102	12

부서테이블	
부서번호	부서이름
10	A
11	B
13	C

상위 두 테이블을 일반적인 이퀴 조인을 하게되면 사원테이블에 부서번호 12와 매핑되는 데이터가 없어서 다음과 같은 데이터만 확인할 수 있다.

사원번호	부서번호	부서번호	부서이름
100	10	10	A
101	11	11	B

하지만, 사원 테이블의 부서번호에 해당하는 부서 테이블의 부서번호가 없더라도 데이터를 출력 하고 싶으면 다음과 같이 출력하게 하면 될것이다.

사원번호	부서번호	부서번호	부서이름
100	10	10	A
101	11	11	B
102	12	null	null

사원테이블의 부서번호 12에 해당하는 부서테이블 데이터가 없어서 데이터 null를 입력하였다. 이런 형태의 결과를 얻기 희망할때 사용하는 조인 방식이 left outer 조인이다. left outer join은 왼쪽 테이블(사원테이블)를 기준으로 왼쪽 테이블의 모든 데이터가 출력되고 오른쪽 테이블(부서테이블)의 없는 데이터를 null을 채워 넣어 보여준다.

사원들이 속한 부서 이름을 출력한다고 생각해 보자. 동등 조인으로 조인하면 부서에 속해있지 않는 사원 데이터는 사라져서 어떤 사원이 사라졌는지 확인할 수 없다. 외부 조인을 사용해서 부서정보가 없는 사원도 null을 채워 출력하면 부서에 속해있지 않은 데이터를 확인할 수 있다.

다음은과 같이 출력하고 싶을 때도 있을 것이다.

사원번호	부서번호	부서번호	부서이름
100	10	10	A
101	11	11	B
null	null	13	C

부서 테이블의 부서번호 13번은 사원테이블의 부서번호에 존재하지 않아서 이퀴조인의 경우 사라지게 된다. 사원 테이블에 없는 데이터를 null을 넣어서 상위처럼 출력하는 방식을 right outer join이라고 한다. right outer join은 오른쪽 테이블(부서테이블)를 기준으로 왼쪽 테이블의 모든 데이터가 출력되고 왼쪽 테이블(사원테이블)에 없는 데이터를 null을 넣어 채워 넣는다.

사원번호	부서번호	부서번호	부서이름
100	10	10	A
101	11	11	B
102	12	null	null
null	null	13	C

상위와 같은 데이터를 얻고 싶을때도 있을 것이다. 이를 full outer join 풀 아우터 조인이라고 한다. full outer join은 left,right outer join으로 발생한 모든 데이터를 생성해서 보여주는 방식이다.

다음 예제를 통해서 아우터 조인을 이해해 보자.

1. employees테이블의 데이터 개수를 확인해 보자.

```
select count(*) from employees;--107
```

2. 사원의 부서정보를 출력하기 위해서 employees테이블과 departments테이블을 동등조인한 데이터 개수를 확인해보자.

```
select count(*) from employees,departments --106  
where employees.department_id=departments.department_id;
```

사원의 수가 107인데 사원중 한명의 부서번호 데이터에 null이 들어있어서 조인결과 106명의 사원정보만 출력 되었다. 매핑되는 부서가 없는 사원을 출력 하고 싶다면 부서정보에 null를 넣어 출력하면 되고 이럴때 사원 테이블을 기준으로 outer 조인을 사용하면 된다.

3. 다음은 left outer join 이다. 왼쪽 테이블을 기준으로 조인 결과가 없더라도 반대쪽 테이블 데이터에 null를 넣어 모든 데이터가 출력 된다. (+)이 추가된 쪽의 테이블 컬럼들에 null값이 들어 간다.

```
select employees.department_id,employees.first_name,department_name  
from employees,departments  
where employees.department_id =departments.department_id(+);
```

부서가 없는 employees테이블의 사원이 부서정보에 null를 넣어 출력되는 것을 확인 할 수 있다. 조인 결과 데이터가 총 107개 출력되는것을 확인할 수 있다.

```
select count(*)  
from employees,departments  
where employees.department_id =departments.department_id(+);
```

4. ansi표준 형태로 left outer 조인 sql를 작성하면 다음과 같고 107개가 출력되어 부서 정보가 없는 사원도 출력되고 있다는것을 확인 할 수 있다.

```
select employees.department_id,employees.first_name,department_name  
from employees left outer join departments  
on employees.department_id =departments.department_id;
```

5. 부서를 기준으로 사원이 한명도 존재하지 않는 부서 정보가 있을 때 사원 정보에 null을 넣어 출력하고 싶다면 Right outer join를 사용하면 된다. 오른쪽 departments 테이블이 기준이 되고 왼쪽 employees 테이블에 관련 부서의 직원이 없으면 직원 정보에 null이 채워져 출력될 것이다.

```
select employees.department_id,employees.first_name,department_name  
from employees,departments  
where employees.department_id(+) =departments.department_id;  
right outer join 결과 122개의 데이터의 검색 결과를 얻을 수 있다는 것을 알 수 있다. 왜 122개가 나왔는지 다음을 확인해 보자.
```

6. 다음 sql문을 통해 사원은 11개의 부서와 1개의 null로 구성되어 있다는 것을 알수 있다.

```
select distinct(employees.department_id) from employees; --12
```

7. 회사에 존재하는 전체 부서는 다음과 같다.

```
select distinct(department_id) from departments; --27
```

8. 회사에 존재하는 총 부서는 27개이므로 사원이 존재하지 않는 부서는 전체부서-사원이존재하는부서 이다.  $27-11=16$ 이라는 사실을 파악할 수 있다. 사원이 존재하지 않는 부서가 16개라는 소리이다.

11개의 부서에 존재하는 사원은 106명이고 사원이 없는 부서는 16개 이므로 사원이 없는 부서도 사원값에 null를 넣어서 출력한다면 부서가 존재하는 사원 데이터 106개 + 부서만 존재하는 데이터 16개가 화면에 출력되어  $106+16=122$ 가되어 122개의 데이터가 right outer join 결과로 출력된다.

9. ansi표준형태의 right outer join문을 sql를 작성하면 다음과 같다.

```
select employees.department_id,employees.first_name,department_name  
from employees right outer join departments  
on employees.department_id =departments.department_id;
```

10. 부서를 배정 받지 못한 사원 1명, 사원이 없는 부서 16개를 null을 포함해서 모두 출력 하려면 full outer join을 사용하면 된다. join를 통해 부서와 사원 정보가 모두 있는데 데이터 106개, 부서 정보가 없는 사원정보 1개, 사원이 없는 부서 정보 16개를 모두 합한 123개의 데이터가 출력된다.

다음 sql과 같은 방법의 full outer join 방식은 제공해 주지 않아서 ansi형태의 sql를 사용해야 한다.

```
select employees.department_id,employees.first_name,department_name  
from employees,departments  
where employees.department_id(+) =departments.department_id(+) --불가능
```

다음과 같은 ansi형태의 full outer join만 가능하다.

```
select employees.department_id,employees.first_name,department_name  
from employees full outer join departments --full outer join  
on employees.department_id = departments.department_id;
```

여러 개의 데이터 베이스를 조인 하려면 다음과 같이 from절과 where절에 여러개 기술하면 된다.

```
select * from regions,countries,locations  
where regions.region_id=countries.region_id  
and countries.country_id=locations.country_id;
```

잘 확인해 보면 기차처럼 모든 테이블이 연결 되도록 각 테이블의 컬럼들을 연결 하면 된다.

-11/11

```
select count(*) from employees;--107
```

```
select count(*) from employees,departments --106  
where employees.department_id=departments.department_id;
```

```

select * from employees;--null확인

select count(*) from employees,departments --107
where employees.department_id=departments.department_id(+);

--ansi
select employees.department_id,employees.first_name,department_name
from employees left outer join departments
on employees.department_id =departments.department_id;

select count(*) from departments; --27 전체 부서 개수

select distinct employees.department_id from employees,departments --11 직원이
있는부서는 11개 없는 부서는 16
where employees.department_id=departments.department_id;

select distinct departments.department_id from employees,departments --11 직원이
있는부서는 11개 없는 부서는 16
where employees.department_id(+) =departments.department_id; --27 11+16

select * from employees,departments --11 직원이 있는부서는 11개 없는 부서는 16
where employees.department_id(+) =departments.department_id;
--ansi
select employees.department_id,employees.first_name,department_name
from employees right outer join departments
on employees.department_id =departments.department_id;
--ansi where employees.department_id(+) =departments.department_id(+);
select employees.department_id,employees.first_name,department_name
from employees full outer join departments      --full outer join
on employees.department_id = departments.department_id;

--사원에 부서명을 출력하시오 부서가 없는 사원은 대기중으로 출력
--1. 외부조인 사용 outer join
--2. nvl
select employees.* ,departments.* , nvl(departments.department_name, '대기중')
from employees,departments
where employees.department_id=departments.department_id(+);

```

## > 29. sub쿼리

sub 쿼리는 다음 sql처럼 sql안에 sql문이 중복되어 존재 하는 것을 의미한다.

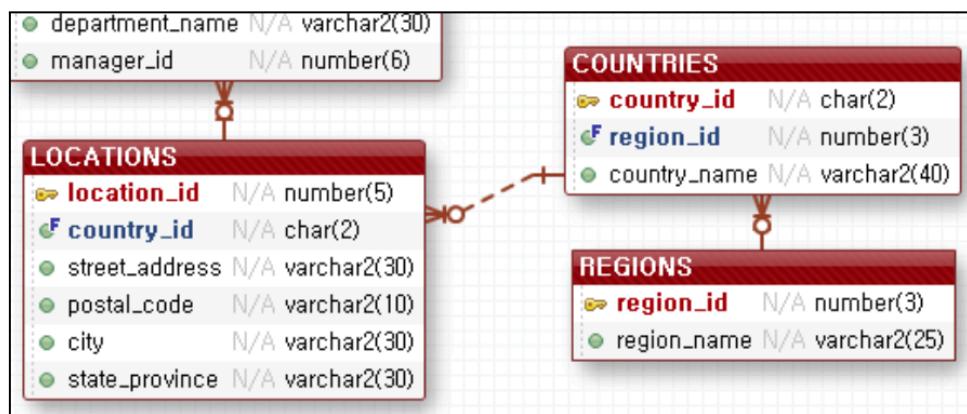
안쪽 소괄호로 묶인 sql이 실행된 결과를 가지고 바깥쪽 sql를 실행한다.

대륙아이디가 2인 지역도시 정보를 출력해 보자.

```
select * from locations where country_id in (
    select country_id from countries where region_id=2
);
```

2번 대륙에 존재하는 도시를 알고 싶어서 안쪽 sql에서는 2번 대륙에 존재하는 나라 정보를 바깥쪽 sql에서는 안쪽에서 검색된 나라에 존재하는 도시정보를 출력하였다.

다음 예제를 이용하여 서브 쿼리를 이해해 보자. regions는 대륙 정보이고 countries는 대륙에 존재하는 나라 정보이고, locations는 나라에 존재하는 도시 정보이다. 2번 대륙 아이디가 만약 아시아라면 아시아 대륙에 속한 나라들 한국, 일본에 있는 지역도시를 출력하면 된다. 상위 코드에서 안쪽 쿼리에 2번대륙의 나라 정보 한국, 일본등 를 찾고 바깥쪽 sql 문에서 찾은 나라에 해당하는 지역도시를 출력 하였다.



1. region\_id가 2인 대륙에 존재하는 부서의 지역 정보를 출력해 보자. 지역정보는 locations 테이블에 있다. 하지만, 대륙 정보를 가지는 region\_id는 존재하지 않는다.  
select \* from locations; 으로 region\_id가 있는지 확인해 보자.

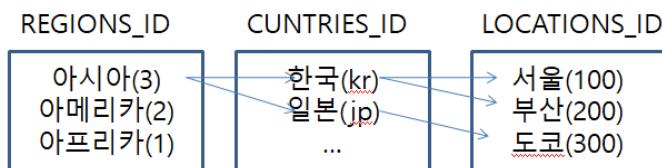
2. 부서 지역 정보를 가지고 있는 locations 테이블에서 2번 대륙을 where절에서 찾아 select 하면 될 것 같지만 locations테이블에는 대륙 아이디(region\_id)가 2인 데이터를 식별할 수 있는 컬럼이 없고 국가 아이디를 식별할 수 있는 country\_id컬럼 밖에 없다.

3. locations 테이블만으로 2번 대륙에 존재하는 country\_id정보를 확인하는 것은 불가능하다.

4. countries 테이블에서 region\_id를 이용해서 2번 대륙에 존재하는 나라들의 country\_id를 구할 수 있다.

```
select country_id from countries where region_id=2;
```

대륙아이디(region\_id)를 통해서 해당 대륙의 여러 나라아이디(country\_id)를 얻을 수 있고 여러 나라 아이디(country\_id)를 통해서 해당 나라의 부서지역(location\_id)을 얻을 수 있다.



countries테이블에서 region\_id가 3일때의 country\_id를 구한 다음, 구한 country\_id들과 동일한 값을 가지는 모든 country\_id를 locations 테이블에서 찾으면, region\_id가 3인 대륙에 존재하는 모든 부서 지역 정보를 얻을 수 있다.

다음 예제를 통해서 2번 대륙의 모든 부서 지역 정보를 출력해 보자.

1. 2번대륙에 있는 나라 아이디를 검색하면 검색 결과는 다음 이미지와 같다.

COUNTRY_ID
AR
BR
CA
MX
US

```
select country_id from countries where region_id=2;
```

2. 나라 아이디가 AR, BR, CA, MX, US에 위치한 부서정보를 출력해 보자.  
대문자로 기술해야 찾을 수 있다.

```
select * from locations where country_id in  
( 'AR', 'BR', 'CA', 'MX', 'US' );
```

3. 상황에 따라 2번 대륙에 있는 부서정보는 변경될 수 있다. 상위 sql문에서 in부분에 'AR', 'BR', 'CA', 'MX', 'US' 대신에 2번 대륙의 나라 아이디를 검색하는 select country\_id from countries where region\_id=2를 넣어 주면 대륙에 존재하는 나라 정보가 변경되더라도 2번 대륙에 위치한 부서 정보를 출력 할 수 있다.

결과적으로 2번 대륙에 있는 부서 정보를 출력한 sql은 다음과 같다.

```
select * from locations where country_id in (  
    select country_id from countries where region_id=2  
) ;
```

4. 3번 대륙에 있는 부서 정보를 출력해보자.

```
select * from locations where country_id in (
    select country_id from countries where region_id=3
);
```

5. Asia 대륙에 있는 모든 부서 도시 이름 출력

-아시아 대륙 아이디가 출력됨 3

```
select REGION_ID from regions where REGION_NAME = 'Asia'; --region_id 구하기
```

-국가아이디 정보가 출력됨 - country\_id = JP CN IN AU SG ML

```
select country_id from countries where region_id=(
    select REGION_ID from regions where REGION_NAME = 'Asia'-- region_id=3
);
```

--최종 지역 정보가 출력됨

```
select * from locations where country_id in (
    select country_id from countries where region_id=(
        select REGION_ID from regions where REGION_NAME = 'Asia'
    )
);
```

다음 서브쿼리 문제를 확인해 보자.

- 지역 아이디가 1700인 부서들에서 일하는 사원 정보를 출력해 보자.
- 지역 아이디가 1700인 부서들에서 일하지 않는 사원 정보를 출력해 보자.

```
SELECT * FROM employees
WHERE department_id NOT IN (
    SELECT department_id FROM departments WHERE location_id = 1700
)
ORDER BY employee_id;
```

이렇게 sql안에 sql를 정의 해서 원하는 결과를 얻는 것을 서브 쿼리라고 한다.

- 최대급여를 받는 사원의 정보를 출력해 보자.

```
select * from employees where salary = (
    select max(salary) from employees
);
```

- 평균보다 급여를 많이 받는 사원의 정보를 출력해 보자.

```
select * from employees where salary > (
    select avg(salary) from employees
);
```

## > 30. exists 함수

exists() 함수는 특정 sql 실행 결과가 있는지 없는지에 따라 실행 여부를 결정할 수 있는 함수이다.

exists()는 괄호안에 오는 sql를 실행한 결과 데이터가 있으면 true, 없으면 false를 생성한다. not exists는 sql를 실행한 결과 데이터가 있으면 false 없으면 true를 생성한다.

다음 예제를 확인하여 exists에 대해서 이해하여야 한다.

select 1 from dual;를 실행하면 항상 1이 실행되므로 실행 결과 항상 데이터가 존재한다. 다음 쿼리를 실행하면 select 1 from dual의 실행 결과가 존재 하므로 employees의 모든 데이터가 항상 출력이된다.

```
select * from employees where exists(select 1 from dual);
```

select 1 from dual where 1!=1;를 실행하면 실행 결과가 항상 없으므로 다음 쿼리를 실행하면 아무 데이터도 출력되지 않는다.

```
select * from employees where exists(select 1 from dual where 1!=1);
select * from employees where not exists(select 1 from dual);
```

- 10번 부서가 존재하면 모든 사원을 출력해 보자.

```
select * from employees where exists(
    select * from departments where department_id=10
)
```

- 10번 부서가 존재하면 20번 부서의 사원을 출력해 보자.

```
select * from employees where department_id=20 and exists(
    select * from departments where department_id=10
)
```

- 사원이 존재하는 부서 정보만 출력해보자.

```
select * from departments d
where exists(
    select * from employees e where e.department_id=d.department_id
);
```

상위 밑줄친 부분을 잘 확인해 보자. 서브 쿼리이면서 바깥쪽 테이블 (departments d)이 안쪽 테이블(employees e)과 관계를 가지면 반복문처럼 바깥쪽 테이블의 여러 데이터(row)에서 하나씩 데이터가 반복되어 안쪽 데이터(row)들과 하나씩 1:1 연결한(join) 다음 연결 되었을 때의 데이터 값은 가지고 안쪽 select문에 데이터가 검색되면 바깥쪽 테이블의 데이터(row)가 선택되어 출력되고 검색되지 않으면 출력되지 않는다. 다음 페이지에 좀더 자세히 확인해 보자.

안쪽 select 문과 바깥쪽 select문 데이터를 연결 하였을때 선택된 값으로 안쪽 select문에서 데이터가 생성되면 exists안에 데이터가 존재하여 바깥쪽에 선택된 데이터가 출력이되고 안쪽에 데이터가 생성되지 않으면 exists의 조건을 만족하지 못해서 해당 데이터는 출력되지 못한다. 상위 쿼리에서는 두 select문에 있는 department\_id값이 같으면 해당 데이터(row)가 선택되어 출력된다.

department		employees	
부서이름	department_id	department_id	사원번호
A	10	→ 10	100
B	11	→ 11	101
C	13	→ 10	102

department.department\_id가 10일때 employees.department\_id 10,11,10를 차례대로 확인해 select \* from employees e where e.department\_id=d.department\_id 를 만족하는 employees.department\_id컬럼의 데이터가 10인 2개의 데이터가 검색되고 exists안에 2개의 검색된 데이터가 생성되므로 바깥쪽 select문에서 department.department\_id가 10일때의 데이터(row) 값이 선택되어 출력 된다.

department.department\_id가 11일때 employees.department\_id 10,11,10를 차례대로 확인해 select \* from employees e where e.department\_id=d.department\_id 를 만족하는 데이터 1개 가 검색되고 exists안에 1개의 검색된 데이터가 생성되므로 바깥쪽 select문에서 department.department\_id가 11일때의 데이터(row) 값이 선택되어 출력 된다.

department.department\_id가 13일때 employees.department\_id 10,11,10를 차례대로 확인해 select \* from employees e where e.department\_id=d.department\_id 를 만족하는 데이터 0개 가 검색되고 exists안에 검색된 데이터가 없으므로 바깥쪽 select문에서 department.department\_id가 13일때의 데이터(row) 값은 선택되지 못해 출력되지 않는다.

따라서 최종 실행 결과는 사원이 존재하는 부서만 출력되는 서브 쿼리가 된다.

## > 31. any, some, all 사용법

any, some은 여러 비교 대상중 하나 이상이 일치하면 결과를 출력한다.  
any와 some은 같아서 둘중 하나를 사용하면 된다.

all은 여러 비교 대상중 모두가 일치하면 결과를 출력한다.

왼쪽과 같은 테이블이 있다고 생각해 보자. 다음 sql를 실행시키면 any나 some의 경우 어떤 결과가 나오는지 확인해 보자.

Salary
50
100
150
200

```
select salary from employees where salary<= any (
    30,90,170
);
```

상위와 같이 기술 하였다면 salary 컬럼의 값 50,100,150은 170일때 조건에 만족하여 30,90,170의 값중 적어도 하나 이상의 값에 만족하여 출력 되지만 salary 컬럼의 값 200의 경우 30,90,170의 값 모두에 만족하지 않아 출력되지 않는다. 따라서, 실행 결과는 50,100,150이 된다.

다음 예제를 확인해 보자.

```
select salary from employees where salary > some (
    30,90,170
);
```

50,100,150,200 모두 출력이 된다.

all의 경우 다음과 같이 동작된다.

```
select salary from employees where salary>= all (
    30,90,170
);
```

상위와 같이 기술 하였다면 salary 컬럼의 값 200일때 만 30,90,170의 모든 값을 만족하여 200이 출력 되지만 salary 컬럼의 값 50,100,150의 경우 30,90,170의 값 모두에 만족하지 않아 출력되지 않는다. 따라서, 실행 결과는 200이 된다.

다음 예제를 확인해 보자.

```
select salary from employees where salary< all (
    30,90,170
);
```

다음 예제들을 살펴보고 이해도를 높여보자.

-부서가 50인 사원들중 한명 보다 급여를 적게 받는 사람을 출력해 보자.

```
select * from employees where salary <= any (
    select salary from employees where department_id=50
);
select * from employees where salary <= some (
    select salary from employees where department_id=50
);
```

-부서가 50인 사원들중 한명 보다 급여를 적게 받는 사람을 부서 50인 사람을 제외하고 출력해 보자. -부서가 50인 사원들 모두 보다 월급을 더 받는 사원을 출력해 보자.

```
select * from employees where salary > all (
    select salary from employees where department_id=50
);
```

non-equal join(범위 조인)는 where절에 범위를 사용하여 테이블을 조인하는 방법이다.

등급 범위 테이블(R)			등급 테이블(G)	조인 결과 테이블
min	max	name	grade	grade name
1	3	상	1	1 상
4	6	중	8	8 하
7	9	하	5	5 중
			4	4 중
			5	5 중
			2	2 상

범위 조인은 조인할 때 범위값을 사용해서 원하는 데이터를 얻는 방법이다. 상위 등급 범위 테이블 R을 확인해 보면 등급별 이름과 해당 등급의 최소 최대값 정보를 가지고 있고, 등급 테이블 G는 여러 등급정보를 가지고 있어 두 테이블의 조인을 통해서 G테이블의 등급별 이름 데이터를 얻을 수 있다.

G테이블의 각각의 등급 데이터들이 R테이블의 min, max 범위에 맞는 이름을 찾아 조인하면 원하는 결과를 얻을 수 있고, 이는 1:1 조인이 아닌 등급(grade)이 min보다 크고 max보다 작을 때의 범위에 해당하는 등급이름을 찾아야 하기 때문에 범위 조회를 해야 한다.

```

select G.grade, R.name from R,G
where G.grade>=R.min and G.grade<=R.max;

```

다음을 실습하고 좀더 이해해 보자.

사원의 연봉이 AD\_PRES, SA\_MAN, IT\_PROG 중 어느 직종의 범위와 같은지 출력하는 sql를 작성해보자.

1. jobs테이블에는 직종별 직종 아이디, 해당 직종의 최소, 최대 연봉이 들어 있다.

```
select * from jobs;
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1 AD PRES	President	20080	40000
2 AD VP	Administration Vice President	15000	30000
3 AD ASST	Administration Assistant	3000	6000
4 FI MGR	Finance Manager	8200	16000
5 FI ACCOUNT	Accountant	4200	9000
6 AC MGR	Accounting Manager	8200	16000
7 AC ACCOUNT	Public Accountant	4200	9000

2. job 테이블에서 AD\_PRES, SA\_MAN, IT\_PROG의 최대 최소급여 정보를 출력해 보자.

```
select * from jobs where job_id in ('AD_PRES', 'SA_MAN', 'IT_PROG')
order by min_salary;
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1 IT PROG	Programmer	4000	10000
2 SA MAN	Sales Manager	10000	20080
3 AD PRES	President	20080	40000

3. employee테이블을 select 하여 사원의 봉급범위가 어느 job\_id 직종과 같은지 확인해 보자.

```
select * from employees;
```

EMPLOYEE_ID	FIRST_NAME	SALARY	LAST_NAME
1 100	Steven	24000	King
2 101	Neena	17000	Kochhar
3 102	Lex	17000	De Haan
4 103	Alexander	9000	Hunold
5 104	Bruce	6000	Ernst
6 105	David	4800	Austin
7 106	Valli	4800	Pataballa
8 107	Diana	4200	Lorentz
9 108	Nancy	12008	Greenberg
10 109	Daniel	9000	Faviet

job\_range컬럼은 사원이 AD\_PRES, SA\_MAN, IT\_PROG의 직종중 어느 직종이 사원의 봉급과 같은 범위에 있는지 출력하는 컬럼이다.

salary가 24000인 사원은 AD\_PRES 직종의 연봉범위와 일치한다. 해당 사원의 job\_range컬럼을 추가해서

AD\_PRES가 들어가도록 만들어 보자.

17000 이 연봉인 사람은 SA\_MAN 직종의 연봉범위와 동일하다. 해당 데이터의 job\_range

컬럼의 값은 SA\_MAN이 된다.

4. 사원별 job\_range를 구하기 위해서 jobs의 부분 테이블 j와 employees테이블을 크로스 조인을 통해 두 테이블에서 나올수 있는 모든 데이트를 출력해 보았다.

```
select employees.employee_id,employees.salary,j.* ,j.job_id as job_range  
from employees,(  
    select * from jobs  
    where job_id in ('AD_PRES', 'SA_MAN', 'IT_PROG')  
    order by min_salary) j  
order by employees.employee_id,min_salary;
```

EMPLOYEE_ID	SALARY	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY	JOB_RANGE
100	24000	IT PROG	Programmer	4000	10000	IT PROG
100	24000	SA MAN	Sales Manager	10000	20080	SA MAN
100	24000	AD PRES	President	20080	40000	AD PRES
101	17000	IT PROG	Programmer	4000	10000	IT PROG
101	17000	SA MAN	Sales Manager	10000	20080	SA MAN
101	17000	AD PRES	President	20080	40000	AD PRES
102	17000	IT PROG	Programmer	4000	10000	IT PROG
102	17000	SA MAN	Sales Manager	10000	20080	SA MAN

5. 상위 이미지에서 크로스 조인된 100번 사원의 데이터 3개중 연봉데이터와 일치하는 데이터만 출력한다면 우리가 원하던 데이터가 된다. 범위 조회를 이용해서 문제를 풀어 보면 다음과 같다.

```
select employees.employee_id,employees.salary,j.* ,j.job_id as job_range  
from employees,(  
    select * from jobs  
    where job_id in ('AD_PRES', 'SA_MAN', 'IT_PROG')  
    order by min_salary) j  
where employees.salary >= j.min_salary and employees.salary < j.max_salary  
order by employees.employee_id,min_salary;
```

6. 최종 결과 데이터를 살펴보면 전체 사원은 107명인데 실행 결과는 65명이다.  
'AD\_PRES', 'SA\_MAN', 'IT\_PROG' 직종의 급여 범위에 벗어나는 사원이 많이 존재 한다.

다음은 employees 테이블과 jobs 테이블을 이용하여 사원의 급여가 해당 사원의 job의 최소급여와 최대급여 사이에 속하는 직원들만 검색하는 예제입니다.

```
SELECT e.employee_id,e.first_name, e.last_name, j.job_title, e.salary  
FROM employees e  
JOIN jobs j ON e.job_id = j.job_id  
WHERE e.salary BETWEEN j.min_salary AND j.max_salary  
ORDER BY e.salary;
```

위 쿼리문에서는 employees와 jobs 테이블을 조인하여 각 직무의 최소급여(min\_salary)와 최대급여(max\_salary) 사이에 속하는 employees의 레코드들을 검색합니다. 결과는 employees 테이블의 first\_name, last\_name, salary 컬럼과 jobs 테이블의 job\_title 컬럼을 보여줍니다.

이 예제에서는 BETWEEN 연산자를 사용하여 범위를 비교하였습니다. BETWEEN 연산자는 해당 컬럼의 값이 두 개의 지정된 값 사이에 속하는 레코드들을 검색하는 데 유용한 연산자입니다.

다음은 HR 스키마에서 departments, jobs, employees 테이블을 이용하여 부서별 최소급여와 최대급여 사이에 속하는 직원들을 검색하는 예제입니다.

```
SELECT e.first_name, e.last_name, d.department_name, e.salary  
FROM employees e  
JOIN jobs j ON e.job_id = j.job_id  
JOIN departments d ON e.department_id = d.department_id  
WHERE e.salary BETWEEN j.min_salary AND j.max_salary  
ORDER BY e.salary;
```

위 쿼리문에서는 employees, jobs, departments 테이블을 조인하여 부서별 최소급여(min\_salary)와 최대급여(max\_salary) 사이에 속하는 employees의 레코드들을 검색합니다. 결과는 employees 테이블의 first\_name, last\_name, salary 컬럼과 departments 테이블의 department\_name 컬럼을 보여줍니다.

이 예제에서도 BETWEEN 연산자를 사용하여 범위를 비교하였습니다. 이 외에도 JOIN 연산자를 사용하여 여러 개의 테이블을 조인하여 검색하는 것이 핵심입니다.

다음 문제를 scott에서 찾아 풀어 보자. 다음을 join형태로 풀보자.

- 사원들이 이름과 사번을 출력해보자.
- job이 CLERK인 사원들의 이름과 부서명을 출력해 보자.
- 연봉이 2600이상인 사원의 이름과 부서명을 출력해 보자.
- 사원 이름과 급여와 급여등급(호봉)을 출력해 보자.
- 사원의 봉급과 상사의 봉급을 출력해 보자.
- 사원번호와 관리자의 사원번호를 출력해보자. 단, 관리자가 없어도 출력해 보자.
- 사원이름과 사원이 다니는 부서명을 출력해보자. 단, 사원이 존재하지 않는 부서는 부서명만 출력해 보자.
- 부서번호가 20인 사원의 사번,부서번호,부서위치를 출력해 보자.

다음을 서브 쿼리로 풀어보자.

- JAMES 사원의 부서이름을 출력해보자.
- 평균급여보다 많이 받는 사원의 이름과 급여를 이름과 급여순으로 내림차순 정렬 하시오.
- 10번 부서의 최대 급여와 같은 급여를 받는 사원번호와 이름 출력
- 20번 부서의 최대 급여보다 많이 받는 사원 번호와 이름을 출력해 보자.
- BLAKE를 상사로가지는 사원정보를 출력하시오.
- 부하직원이 있는 사원 정보를 출력하시오.
- 부하직원이 없는 사원 정보를 출력하시오.

정답풀이

```
--select dept.dname, emp.deptno from emp;
--select emp.ename, dept.dname from emp, dept where emp.deptno=dept.deptno and
emp.job = 'CLERK';
--select emp.ename, dept.dname from emp, dept where emp.deptno=dept.deptno and
emp.sal >= 2600;
--select emp.ename, emp.sal, salgrade.grade from emp, salgrade where emp.sal
between salgrade.losal and salgrade.hisal;
--select e1.sal, e2.sal from emp e1, emp e2 where e1.mgr = e2.empno;
--select e1.empno, e2.empno from emp e1, emp e2 where e1.mgr = e2.empno(+);
--select emp.ename, dept.dname from emp, dept where emp.deptno(+) =
dept.deptno;
--select emp.empno, emp.deptno, dept.loc from emp, dept where dept.deptno =
emp.deptno and emp.deptno = 20;
--
--select ename, sal from emp where sal>(select avg(sal) from emp) order by
ename desc, sal desc;
--select empno, ename from emp where sal=(select max(sal) from emp where
deptno=10) ;
--select empno, ename from emp where sal>(select max(sal) from emp where
deptno=20);
--select * from emp where mgr=(select empno from emp where ename='BLAKE');
--select * from emp where empno=any(select mgr from emp);
--select * from emp where empno!=all(select mgr from emp where mgr is not
null);
```

## > 32. 무결성 제약 조건

무결성 잘못된 데이터가 테이블에 들어오는 것을 막아 테이블안에 올바른 데이터가 유지되도록 하는 것을 의미한다.

데이터 무결성 제약 조건은 테이블에 잘못된 데이터가 들어가지 못하도록 하기 위해 사용하는 제약 조건이다. 다음과 같은 것들이 존재하고 아래 5개중 선택된 제약조건을 테이블 제작시 컬럼에 설정하여 적용 한다.

1.primary key	테이블에 들어있는 데이터를 구분하는데 사용하는 대표컬럼으로 사용하고 싶을때 컬럼에 적용한다. 해당 컬럼의 값으로 null과 중복된값을 허용하지 않는다. 기본키라 한다. 설정된 컬럼은 중복되지 않는 유일한 값을 가지고 있어서 테이블에 데이터를 식별하는 용도로 사용한다. 유일한 값을 가진다. 테이블에 한개만 존재하고 중복된 컬럼을 하나의 키로 사용할 수 있다.
2.foreign key	설정된 컬럼은 다른 테이블의 primary key 컬럼에 존재하는 값중 하나의 값을 값으로 가지는 컬럼 이다. 외래키라 한다.
3.not null	null을 허용하지 않는다.
4.unique	중복된 값을 허용하지 않는다. 같은 값이 들어갈수 없다.
5.check	선택된 컬럼에 설정한 범위의 값만 허용한다.

1. primary key 가 필요한 이유에 대해서 설명해 볼 예정이다.

BNo	PNo	PName	PArea	BKind
1	1	김명천	번동	왕포도
2	2	김진우	홍일동	청포도
3	3	김태수	쌍문동	청포도
4	4	박지민	상계동	왕포도
5	1	김명천	번동	청포도
6	2	김진우	홍일동	왕포도
8	3	김태수	쌍문동	청포도
8	3	김태수	쌍문동	청포도

왼쪽 이미지 빨간선 부분을 확인해보면 테이블 안에 같은 데이터가 들어가 있는 것을 확인할 수 있다. 테이블에 같은 데이터가 들어 있으면 데이터로 특정 데이터를 조작하는 데이터베이스 특성상 두 데이터중 하나의 데이터를 선택해서 delete, update, select 할 수 있는 방법이 없다. 따라서, 같은 테이블에 모든 컬럼의 값이 같은 데이터가 여러개 들어간 경우 쓸데없이 공간만 차지하는 형태가 된다. 그러므로, 원천적으로

같은 데이터가 들어 가지 못하도록 컬럼에 기본키(primary key)를 설정하면 해당 컬럼에 사용자가 중복된 데이터를 넣으면 제약조건에 위배된다는 메시지와 함께 테이블에 데이터가 들어가지 않게 된다. 어떤 컬럼에 primary key를 설정하는 것이 좋은지 생각해 보면 테이블에 한개만 설정할 수 있으므로 테이블을 대표하는 컬럼중에 null이 포함되지 않고 중복된 값을 가지지 않는 컬럼을 선택해야 한다.

primary key 생성하는 방법은 다음과 같다.

-기본키가 하나의 컬럼으로 이루어져 있을 때는 다음과 같이 하면 된다.

테이블 이름이 pkTable1이고 col1,col2,col3를 컬럼으로 가지고 col1 컬럼이 기본키(pk)인 테이블을 만들면 다음과 같다.

```

create table pkTable1(
    col1 number primary key, --컬럼 col1은 기본키가 된다.
    col2 number,
    col3 number
);

```

만든 테이블에 왼쪽과 같은 데이터를 넣는다면 첫번째 두번째 데이터는 문제 없이 들어가지만 3번째 데이터는 pk로 설정되어 있는 col1에 같은 값이 들어 가게 되므로 제약조건에 위배된다는 메시지가 발생한다. 다음 insert문을 실행시켜서 확인해 보자.

col1	col2	col3
1	1	1
2	1	1
2	3	4

```

insert into pkTable1 values(1,1,1);
insert into pkTable1 values(2,1,1); --col1컬럼에 2값이 없어서 들어 간다.
insert into pkTable1 values(2,3,4); --col1컬럼에 2값이 있어서 안 들어 간다.
commit;

```

--두개의 컬럼을 하나의 primary key로 사용할 수 있다. col1,col2컬럼 둘다 동시에 같은 값을 넣으면 제약조건에 위배 된다.

```

create table pkTable2(
    col1 number,
    col2 number,
    col3 number,
    primary key(col1,col2) --col1,col2 2개의 컬럼이 합쳐져서 하나의 키가 된다.
)

```

col1	col2	col3
1	1	1
2	1	1
2	3	4
2	3	6
1	1	6

1~3번째 데이터는 문제없이 테이블에 들어가지만 4~5번째 데이터는 중복된 기본키로 설정되어 있는 col1,col2에 이미 들어가 있는 값과 겹쳐서 무결성 제약 조건을 위배하여 데이터가 들어가지 않는다. 다음 insert문을 확인해보자.

```

insert into pkTable2 values(1,1,1);
insert into pkTable2 values(2,1,1);
insert into pkTable2 values(2,3,4);
insert into pkTable2 values(2,3,6); -col1,col2컬럼에 2,3이 있어서 안 들어간다.
insert into pkTable2 values(1,1,6); --col1,col2컬럼에 1,1이 있어서 안 들어간다.
insert into pkTable2 values(1,2,4); --1,2는 col1,col2컬럼에 없어서 들어가 진다.
commit;

```

테이블을 만들고 차후 pk를 끌어 넣을 수도 있다.

```
create table pkTable3(
    col1 number,
    col2 number,
    col3 number
);
alter table pkTable3 add primary key(col1,col2);
```

2. foreign key가 필요한 이유를 설명할 예정이다.

BNo	BKind	BName	BArea	GNo	GPrice
1	왕포도	김명천	번동	1	30000
2	청포도	김진우	홍일동	3	25000
3	청포도	김태수	쌍문동	2	20000
4	왕포도	박지민	상계동	2	
5	청포도	김명천	평창동	1	
6	왕포도	김진우	오류동	3	
7	왕포도	김태수	대림동	1	
8	청포도	김태수	청담동	2	

FK 컬럼은 다른 테이블의 PK 컬럼의 값 중 하나를 값으로 가지는 컬럼이다.

상위처럼 2개의 테이블이 존재 할 때 왼쪽 테이블을 box 오른쪽 테이블을 grade라 한다면 box 테이블에서 pk는 bno컬럼이고 fk는 gno컬럼이다. grade 테이블에서 pk는 gno이다.

foreign key로 설정할 box테이블의 gno 컬럼은 다른 테이블 grade테이블의 pk인 gno컬럼의 값 1, 2, 3 중 하나의 값을 값으로 가지는 컬럼이다. 만약 fk가 설정된 컬럼에 다른 테이블의 pk가 가지고 있지 않는 값 4를 넣으려 한다면 제약 조건에 위배 된다는 메시지가 출력된다. box 테이블의 gno 컬럼에 fk를 설정하면 grade테이블의 gno컬럼이 가지고 있는 값 중 하나의 값을 반드시 box 테이블의 gno 컬럼의 값으로 사용하여야 한다.

왼쪽 테이블의 GNo는 오른쪽 테이블의 primary key인 Gno컬럼의 값 중 하나의 값을 가진다. 만약에 오른쪽 테이블 GNo 컬럼에 없는 값이 왼쪽 테이블 GNo에 들어 있다면 두 테이블 조인할 경우 짹을 찾을 수 없어 사라지게 된다. 이런 잘못된 문제가 발생하지 않도록 데이터를 왼쪽 테이블 GNo에 오른쪽 기본키 GNo 컬럼에 없는 값을 넣지 못하게 하는 방법이 왼쪽 테이블 GNo컬럼에 외래키(foreign key)를 설정 하는 방법이다.

기본키를 설정하면 오른쪽 테이블 GNo컬럼이 가지고 있는 값을 제외한 다른 값이 왼쪽 테이블 GNo컬럼에 들어오지 못하게 해준다. 만약 들어 있지 않은 값을 외래키 설정 컬럼에 넣으려 한다면 제약조건에 위배된다라는 에러 메시지를 확인할 수 있다.

외래키를 만드는 것은 다음과 같다. 외래키를 설정하고자 하는 컬럼 자료형 옆에 references라 기술하고 참조 할 테이블명 (참조 할 컬럼명)을 기술하면 된다.

다음은 fkTable2 테이블의 col3 컬럼을 참조하는 fkTable1테이블의 col3 컬럼을 sql로 작성한 것이다. 참조할 테이블 fkTable2테이블을 먼저 만들어야 한다.

fkTable2 테이블의 기본키 col3 컬럼을 fkTable1 테이블에 col3 컬럼의 외래키로 설정하면 다음과 같다.

```
create table fkTable2(
    col1 number,
    col2 number,
    col3 number primary key --컬럼 col3은 기본키가 된다.
);
```

```
create table fkTable1(
    col1 number primary key, --컬럼 col1은 기본키가 된다.
    col2 number,
    col3 number references fkTable2(col3 )
);
```

테이블을 만들때 순서를 지켜야한다. fkTable1를 먼저만들면 문제가 발생한다. 이유는 fkTable1 테이블의 col3컬럼에서 fkTable2테이블의 col3컬럼을 참조하고 있어 없는 테이블을 참조할 수 없으므로 fkTable2테이블의 col3컬럼을 먼저 만들어야 한다.

다음은 테이블만 만든후 키를 추가하는 방법이다. pk\_fktable2\_col과 비슷한 것들은 생성한 키를 식별하는 키의 별명이여서 겹치지 않게 마음데로 사용하면 된다.

```
create table fkTable2(
    col1 number,
    col2 number,
    col3 number --컬럼 col3은 기본키가 된다.
);
```

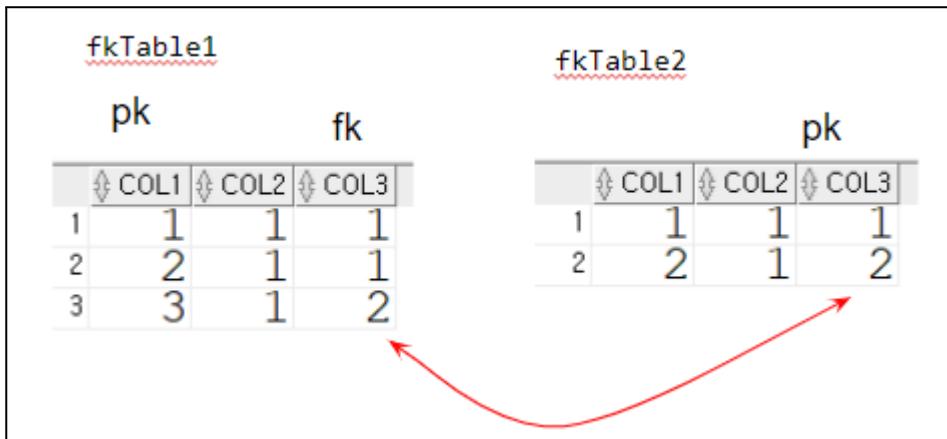
```
create table fkTable1(
    col1 number, --컬럼 col1은 기본키가 된다.
    col2 number,
    col3 number
);
- fkTable2 테이블의 기본키 추가
ALTER TABLE fkTable2
ADD CONSTRAINT pk_fkTable2_col3 PRIMARY KEY (col3);
```

```
-- fkTable1 테이블의 기본키 및 외래키 추가
ALTER TABLE fkTable1
ADD CONSTRAINT pk_fkTable1_col1 PRIMARY KEY (col1);

ALTER TABLE fkTable1
ADD CONSTRAINT fk_fkTable1_col3 FOREIGN KEY (col3)
```

```
REFERENCES fkTable2(col3);
```

self 조인이 필요한 경우는 만든 다음에 추가하는 방법을 사용해야 한다.



만든 테이블에 데이터를 넣을 때도 두 테이블이 관계를 가지고 있으므로 fkTable2테이블의 primary key인 col3에 들어 있는 값이어야 fkTable1테이블의 col3 foreign key 컬럼에 넣을 수 있다. 따라서 입력할 때도 반드시 fkTable2테이블에 먼저 데이터를 입력한 다음에 fkTable1 테이블에 데이터를 입력해야 한다.

다음 sql를 실습해 보자.

```
insert into fkTable2 values(1,1,1);
insert into fkTable2 values(2,1,2);
insert into fkTable2 values(2,1,2); --col3이 pk여서 제약조건 위배로 실행되지 않음
```

```
insert into fkTable1 values(1,1,1);
insert into fkTable1 values(2,1,1);
```

```
insert into fkTable1 values(2,1,1); -- pk 때문에 들어가지 않는다.
```

```
insert into fkTable1 values(3,1,5); -- fk 설정되어 있어서 들어가지 않는다.
```

--fkTable2테이블에 col3컬럼에는 1과 2만 들어 있어서 5를 넣을 수 없다.

```
select col3 from fkTable2; --의 결과는 1,2 만들어 있다.
```

```
insert into fkTable1 values(3,1,2); -- fk 설정되어 있어서 들어간다.
```

```
insert into fkTable1 values(4,1,5); 와 같은 데이터를 넣으려면 fkTable2 테이블에 다음 데이터를 넣어야 들어 갈 수 있다.
```

```
insert into fkTable2 values(2,1,5);
```

데이터 삭제 방법은 반드시 fk가 들어 있는 테이블의 데이터를 먼저 삭제해야 한다.

```
delete fkTable1; -- 반드시 fkTable1을 먼저 삭제해야 한다.
```

```
delete fkTable2; - fkTable2를 먼저 삭제하면 무결성에 위배되어 에러가 발생한다.
```

무결성에 위배되는 이유는 fkTable2의 데이터를 삭제하면 삭제된 데이터를 참조하는 데이터가 fkTable1에 존재하기 때문이다. 결론은 foreign key가 설정되어 있어서 fkTable2를 먼저 삭제하면 무결성에 위배된다.

```
drop table fkTable1;    drop table fkTable2;
```

두 테이블 사이에 관계가 설정되어 있으면 데이터 삭제 순서를 반드시 지켜야 삭제가 가능하다.

다음 sql처럼 foreign key를 설정해 놓으면 fkTable2의 primary key 가 아닌 컬럼을 참조하고 있으므로 문제가 발생한다. foreign key는 다른 테이블의 pk만 참조할 수 있다.

```
create table fkTable1(
    col1 number primary key, --컬럼 col1은 기본키가 된다.
    col2 number,
    col3 number references fkTable2(col2) -- col2는 pk가 아니다.
);
```

fk와 pk는 두 테이블을 조인으로 합칠 때 보통 기준이 되는 컬럼이다. 하나의 테이블을 굳이 테이블을 쪼개서 만든 이유는 테이블을 합쳐서 만들면 삽입, 삭제, 갱신 이상 문제가 발생하기 때문이다. 그래서 관련 있는 컬럼끼리 나누어 테이블을 만든다.

조인의 종류를 정리해 보면 크로스조인(동등조인) 아우터 조인 셀프 조인 등이 있다. 크로스 조인은 두 테이블에 있는 모든 데이터를 가지고 만들 수 있는 모든 데이터를 만드는 방법이다. 이퀴 조인은 크로스 조인으로 만든 어진 모든 데이터 중 각각의 테이블에서 특정 컬럼 값이 같은 데이터만 출력하는 방법이다. 아우터 조인은 이퀴조인으로 조인할 경우 공통 컬럼에 같은 값이 존재하지 않으면 출력되지 않는다. 아우터 조인은 공통된 값이 존재하지 않더라도 존재하는 값은 원래의 값을 출력하고 존재하지 않는 값은 null을 넣어 출력하는 방법이다.

데이터베이스에서 여러 테이블을 사용하다 보면 의도한 바와 다르게 잘못된 데이터가 들어가는 경우 무결성 제약 조건에 위배된다고 이야기한다.

다음 이미지를 참조해서 box와 grade 테이블의 제약 조건을 적용하여 sql로 테이블과 데이터를 만들어 넣어 보자. 스스로 만들어 보고 이후 내용을 읽어 보자.

BNo	BKind	BName	BArea	GNo
1	왕포도	김명천	번동	1
2	청포도	김진우	홍일동	3
3	청포도	김태수	쌍문동	2
4	왕포도	박지민	상계동	2
5	청포도	김명천	평창동	1
6	왕포도	김진우	오류동	3
7	왕포도	김태수	대림동	1
8	청포도	김태수	청담동	2

GNo	GPrice
1	30000
2	25000
3	20000

스스로 작성해 보고 다음 sql을 확인해 보자.

```

drop table Grade; --기존에 테이블이 있을수 있어서 추가함
create table Grade(
    GNo number(3) primary key,--primary key 추가
    GPrice number(10)
);
drop table Box;
create table Box(
    BNo number(3) primary key,
    BKind nvarchar2(5),
    BName nvarchar2(5),
    BArea nvarchar2(5),
    GNo number(3) references Grade(GNo) --foreign key추가
);

```

-상위 테이블들에 데이터를 입력해 본후 다시 삭제해 보자. 테이블도 삭제해 보자.

-cascade옵션을 이용해서 관련이 있는 테이블을 한번에 삭제할 수 있지만 테이블들은 대부분 복잡하게 연결되어 있어서 예상과 다르게 테이블이 지워질수 있으니 되도록 사용하지 않는 것이 좋다.

```
DROP TABLE parent_table CASCADE CONSTRAINTS;
```

-신규 프로젝트 진행시 키설정을 하지말고 차후 추가하는 방식을 사용하자. 이유는 제약이 많으면 프로그램짜는데 많은 시간과 노력이 들어간다. 제약조건 없이 테스트한후 차후 추가하는 것이 신규 개발하기 편하다.

다음은 나머지 제약 조건에 대한 설명이다.

```

create table test4(
    a1 number unique,           --해당 컬럼에 같은 값을 넣지 못한다.
    a2 number not null,         --null을 허용하지 않음.
    a3 number not null unique, --유일한 값을 가지고 null을 허용하지 않음.
    a4 number check (a4>0),    --0보다 큰수만 가진다. 도메인
    a5 number default 1        --입력시 값이없으면 1로 들어간다.
);

```

테이블을 만들어 보고 실제로 값을 넣어서 설명한것 처럼 동작하는지 확인해보자.

```

--default값 : a5컬럼에 값을 넣지 않으면 default값이 들어감
insert into test4(a1,a2,a3,a4) values(2,2,2,2); -- default값 1이 들어감
--일반적인 입력
insert into test4(a1,a2,a3,a4,a5) values(1,1,1,1,null); --a5 컬럼은 null이 들어감 --값을 넣지 않아야 1이 들어감 null을 넣으면 null들어감;

```

보통 컬럼에 값을 넣지 않으면 null이 들어 가지만 컬럼에 default 값을 설정해 놓으면 null 대신 default값이 들어 간다.

--check (a4>0) : a4컬럼에 들어갈 값의 범위를 설정할 수 있다.

```
insert into test4(a1,a2,a3,a4,a5) values(-2,-2,-2,-2,-2); --안 들어감
```

--a4 컬럼에 check 범위가 양수로 되어 있어서 음수를 넣을수 없음

```
insert into test4(a1,a2,a3,a4,a5) values(-2,-2,-2,1,-2); -- 들어감
```

--not null unique a3컬럼에 이미들어 있는 값이나 null을 넣으면 안들어간다. 이미 들어가 있는 데이터와 중복되지 않은 다른 값을 넣어야 동작한다.

```
insert into test4(a1,a2,a3,a4,a5) values(3,3,null,3,3); -- null이어서 안들어감
```

```
insert into test4(a1,a2,a3,a4,a5) values(3,3,1,3,3); -- 이전에 a3컬럼에 1값을 넣어서 안들어감
```

```
insert into test4(a1,a2,a3,a4,a5) values(3,3,3,3,3); -- 들어감
```

--not null : a2컬럼에 null을 넣지 못하도록 설정하였다.

```
insert into test4(a1,a2,a3,a4,a5) values(4,null,4,4,4); -- 안들어감
```

```
insert into test4(a1,a2,a3,a4,a5) values(4,4,4,4,4); -- 들어감
```

--a1컬럼에 unique 가 설정되어 있다.

```
insert into test4(a1,a2,a3,a4,a5) values(null,5,5,5,5); -- 들어감
```

```
insert into test4(a1,a2,a3,a4,a5) values(null,6,6,6,6); -- 들어감
```

--unique설정된 컬럼에서 null은 여러개 들어간다.

-unique설정이 되어 있어 다음 데이터는 안들어간다.

```
insert into test4(a1,a2,a3,a4,a5) values(4,4,4,4,4); -- 안 들어감
```

--다른 데이터는 들어감

```
insert into test4(a1,a2,a3,a4,a5) values(7,7,7,7,7);
```

```
select * from test4;
```

제약 조건 사용 방법을 간단히 살펴 보았다. 자세히 공부해 보고 싶다면 나중에 필요할때 웹에서 검색해서 확인해 보자.

user\_constraints테이블을 이용해서 해당 테이블이 가지고 있는 정보를 확인 할 수 있다.

```
select constraint_type,user_constraints.* from user_constraints
```

```
where table_name='TEST4';
```

TEST4테이블에 제약조건을 확인 할 수 있다. 대문자로 기술하여야 한다. 아래와 같은 약어로 저장되어 있다.

p: primary key R: foreign key U: unique C:check,not null

다음을 실습해서 employees테이블에 새로운 데이터를 넣어보자.

EMPLOYEES		
employee_id	N/A	number(6)
manager_id	N/A	number(6)
department_id	N/A	number(6)
job_id	N/A	varchar2(10)
first_name	N/A	varchar2(20)

이전 챕터에서 hr테이블 제작시 hr 전체

관계도를 확인해 보자.

`insert`문을 알고 있더라도 `employees`테이블에 데이터를 넣는 것은 쉬운 작업이 아니다.

일단 다음 순으로 테이블을 확인해 보아야 한다.

1. `null`을 허용하는가? 허용하지 않는다면 데이터 입력시 반드시 값이 있어야 한다.

2. 기본키가 설정되어 있는가? 기본키가 설정되어 있다면 해당 컬럼에 같은 값을 넣을 수 없다.

3. 외래키가 설정되어 있는가? 외래키가 설정되어 있다면 해당 테이블의 컬럼에 들어 있는 값 중 하나의 값만을 넣어야 한다.

4. `check` 속성을 사용해서 들어갈 값의 범위가 설정되어 있는가?

일단 제약조건이 설정되어 있는 `employees`테이블에 데이터를 넣어보자.

--사원 테이블에 데이터를 넣어 보자. 다음 `select` 문을 이용해서 `employee_id` `job_id` `manager_id` `department_id` 컬럼들의 위치와 들어 있는 값을 확인해 보자.

`employees`테이블에 데이터를 입력할 때 다음 조건들을 만족해야 정상적으로 데이터가 입력된다.

--`employee_id` 현재 테이블의 해당 컬럼에 이미 들어가 있는 동일한 값을 다시 넣으면 안된다. 해당 컬럼에 존재하지 않는 값 중 하나를 넣자. 이런 컬럼을 `primary key` (기본키)라 한다. pk는 해당 컬럼에 들어 있는 값을 잘 살펴보면 동일한 값 없이 모두 다른 값을 가지고 있다.

--`job_id` `jobs`테이블 안에 있는 `job_id` 컬럼의 값 중 하나를 넣어야 한다. 이런 컬럼을 `foreign key` (외래키)라고 한다. fk가 설정되어 있는 컬럼은 특정 컬럼이 가지고 있는 값 중 하나의 값을 가진다.

--`manager_id` 사원 테이블 안에 있는 `employee_id` 중 하나를 넣자. `self join`을 통한 `foreign key`이다.

--`department_id` `departments`테이블에 있는 부서 아이디 값 중 하나를 넣자. 외래키.

--우리가 직접 만든 테이블에 마음대로 데이터를 넣을 수 있지만, 상위 테이블은 우리가 만든 테이블이 아니고, 제약 조건이라는 문법을 이용해서 특정 데이터만 들어 가게 만들어 놓았다. 상위 조건들을 만족하게 데이터를 넣지 않으면 문제가 발생해서 데이터를 넣을 수 없다. 다음 `sql`과 같이 조건에 맞춰서 넣어야 데이터가 들어간다.

--이메일 컬럼은 유니크한 값을 가지도록 제약 조건이 걸려 있다. 이미 들어 있는 같은 이메일 주소를 넣을려고 하면 제약 조건에 위배된다는 글이 뜬다.

```
insert into employees(employee_id, first_name, last_name, email, phone_number,
```

```
hire_date, job_id, salary, commission_pct, manager_id, department_id)
values(250,'sumin','park','foxman12@hanmail.net','010-2211-0000',sysdate,
'MK_REP',10000,0.2,100,140);
```

--모든 컬럼에 데이터를 넣을 때 컬럼명을 생략 할수 있지만 일부 컬럼에 데이터를 넣을 때는 컬럼명을 생략할 수 없다.

```
insert into employees values(251,'yeji','shin','asdf@naver.com',
'010-1234-5678',sysdate,'MK_REP',10000,0.2,100,140);
```

일반적으로 테이블의 형에 맞는 데이터를 넣으면 문제 없이 데이터가 들어가지만 제약 조건이 설정되어 있는 테이블이 있으면 제약 조건에 맞는 데이터만 들어 간다.

```
select * from employees; --입력한 데이터가 있는지 확인해 보자.
```

```
rollback; --commit은 적용 rollback; 취소
```

-- commit하면 적용 되지만 rollback하면 취소 된다. select를 통해서 제대로 들어갔는지 확인한 다음 값이 변경되면 이후 작업에 혼돈이 올 수 있으니 rollback하여 원래의 상태로 돌려 놓자.

--desc employees;로 확인해보면 not null인 컬럼이 뜨는데 not null인 컬럼은 null값을 허용하지 않아서 생략 불가능 하다 반드시 값을 가지고 있어야 한다.

다음은 null값을 허용하지 않는 컬럼에만 값을 넣어 새로운 데이터를 넣어본 것이다.

```
insert into employees(employee_id,last_name,email,hire_date,job_id)
values(250,'shin','asdf@naver.com',sysdate,'MK_REP');
select * from employees;
rollback; -작업 후 확인하여 취소 하였다.
```

다음 문제를 풀어보자.

HR데이터베이스에 다음 데이터를 적절한 데이터를 포함하여 DB에 넣어보자.

- 다음 sql를 만들어 데이터베이스에 넣어보고 삭제해 보자. 오세아니아 대륙의 하와이에 신규 입사한 홍길동 사원이 프로그램으로 입사하였다.
- 입력한 데이터는 차후 문제가 발생할 수 있기 때문에 삭제 하자.
- 참조 관계를 무시하고 데이터 베이스를 조작 할 수 있는 방법으로 cascade를 사용해 보자.

---

## > 33. view와 sequence 사용하기

---

view는 기존 존재하는 여러 테이블이나 뷰를 조합하여 새로운 가상 테이블로 만든 것을 view라 한다.

view는 복잡한 내용의 sql문이나 특정 사용자에게 보여주고 싶지 않은 내용을 감추기 위해서 사용 한다.

view를 만드는 방법은 `create view` 뷰이름 `as` 쿼리 형태로 뷰로 만들고 만들어진 뷰의 내용은 뷰를 만들 때 기술한 쿼리를 기술 하면 된다.

view 자체가 실제 데이터를 가지고 있는 것은 아니지만 테이블과 같다고 생각하고 동일하게 사용하면된다.

테이블의 데이터가 변경되면 이를 사용하는 view의 데이터도 변경된다.

view를 통해서 CRUD작업이 가능하지만 되도록 `select`할때만 사용하자.

1. 사원의 이름과 해당 사원이 일하는 부서 이름을 출력해 보자.

```
select employees.first_name,departments.department_name from  
employees,departments  
where employees.department_id=departments.department_id;
```

2. 상위 복잡한 쿼리의 결과 와 같은 결과를 가지는 테이블을 만들 필요성이 있다면 view로 만들어 테이블처럼 사용할 수 있다. 아래 예제를처럼 view를 만들어 보자.

```
create view view_ed as  
select employees.first_name,departments.department_name from  
employees,departments  
where employees.department_id=departments.department_id;
```

3. 앞에서 만든 view를 이용해서 사원의 이름과 해당 사원이 일하는 부서 이름을 출력해 보자.

```
select first_name,department_name from view_ed;  
select * from view_ed;
```

view를 삭제하고 싶으면 `drop view view이름;` 하면된다.

`view`를 이용해서 데이터 입력이나 여러 작업들을 사용할 수 있으나 복잡하게 사용하려면 `view`보다는 프로시저를 사용하는 것이 나을 것 같다. 이 책에서 프로시저관련 내용은 다루고 있지 않으니 검색해서 공부해 보자.

시퀀스는 스스로 하나씩 증가하는 카운터를 관리하는 객체이다.  
`create sequence` 시퀀스명; 으로 시퀀스를 생성할 수 있고 삭제하고 싶다면  
`drop sequence` 시퀀스명;을 사용하면된다.  
시퀀스 객체는 다음과 같은 값을 가진다.  
.nextval 시퀀스가 가지고 있는 현재 카운트수에서 하나를 증가시켜 반납한다.  
.currval 현재 시퀀스가 가지고 있는 카운트수를 리턴한다.

1. 시퀀스를 `testSequence`라는 이름으로 만들어 보자.

```
create sequence testSequence;
```

2. `testSequence`가 가지고 있는 카운트를 하나 증가시켜 찍어보자.

```
select testSequence.nextval from dual;
```

상위 쿼리를 여러번 실행시켜 카운터가 증가하는 것을 확인해보자.

현재 카운트를 하나 증가시킨 후 증가된 값을 출력한다.

3. `testSequence`가 가지고 있는 현재 카운트를 찍어보자.

```
select testSequence.currval from dual;
```

상위 쿼리를 여러번 실행시켜 현재 카운트가 지속적으로 출력되는 것을 확인해 보자.

카운터의 현재 값을 출력한다.

3. `testSequence`를 삭제해 보자.

```
drop sequence testSequence;
```

시퀀스는 간단히 카운트를 할 때 사용하는 것이지 안전성을 보장받아 카운트하는 용도로는 사용하면 안된다.

테이블을 만들어 카운터를 이용하여 데이터를 넣어보자.

```
create sequence ctable_seq;
select ctable_seq.nextval from dual;
select * from ctable;
select ctable_seq.currval from dual;
create table ctable(
    c_count number,
    c_comment nvarchar2(10)
);
```

```

insert into ctable values (ctable_seq.nextval,'시작');
insert into ctable values (ctable_seq.nextval,'2번째 데이터');
insert into ctable values (ctable_seq.nextval,'마지막');
commit;
drop sequence ctable_seq;

```

## > 34. index 사용하기

인덱스는 우리나라 말로 색인이라 하는데 데이터를 좀 더 빠르게 검색 하기 위해서 사용 한다. 원본 데이터에서 검색에 필요한 부분을 뽑아서 인덱스로 만든다.

책을 예로 들자면 목차가 있으면 원하는 내용을 좀 더 빠르게 찾을 수 있다. 책의 목차가 데이터베이스에서는 인덱스에 해당한다.

오라클 데이터베이스의 경우 primary key, 외래키, unique를 사용하면 자동으로 인덱스가 생성 된다.

인덱스는 크게 nonunique와 unique인덱스가 있는데 unique인덱스는 중복값이 없는 컬럼에 nonunique 인덱스는 중복값을 허용하는 컬럼에 사용한다. 만드는 방법은 다음과 같다.

nonunique인덱스 create index 인덱스이름 on 인덱스를 설정할 테이블(컬럼);

unique인덱스 create unique index 인덱스이름 on 인덱스를 설정할 테이블(컬럼);

primary key, unique에는 unique인덱스가 외래키에는 nonunique인덱스가 사용된다.

오라클에서는 매우 다양한 인덱스가 존재하는데 잘못 사용하면 오히려 성능저하의 원인이 되니 본인이 잘모르면 사용하지 않는 것이 낫다. 상의 2가지 인덱스 정도만 잘 알고 있어도 성능 향상에 큰 도움이 될 것이다.

```

drop table test7;
create table test7(
    a1 number,      a2 number,      a3 number,      a4 number
);

```

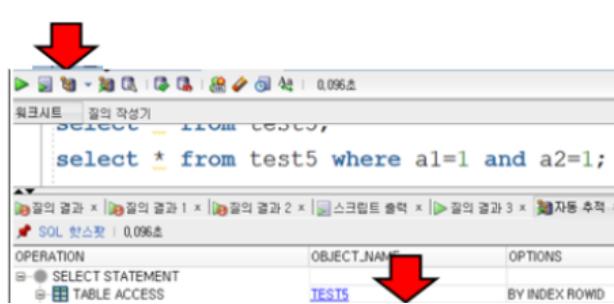
a1,a2 컬럼에 nonunique 인덱스 설정하기

```
create index idx_test7_a1a2 on test7(a1,a2);
```

a3 컬럼에 unique 인덱스 설정하기

```
create unique index idx_test7_a3 on test7(a3);
```

idx\_test7\_a1a2, idx\_test7\_a3은 인덱스의 이름이다 중복되지 않게 본인이 정하면 된다.



인덱스 사용 여부를 확인 하려면 왼쪽 이미지처럼 실행하고자 하는 sql를 선택한 다음에 상위 빨간색 부분의 버튼을 누르면

하단 빨간색 화살표 부분 처럼 사용한 인덱스가 표시된다.

인덱스를 만들었다고 항상 성능이 향상되는 것이 아니다. 잘못 사용하면 오히려 성능이 저하된다. 인덱스를 만들면 테이블에 데이터를 넣어보고 실행시간과 인덱스 사용 여부를 확인해 보자.

다음 쿼리를 실행시켜보고 인덱스가 사용되었는지 확인해보자.

```
select * from test7;          -- 인덱스를 사용하지 않음
select * from test7 where a1=1;    -- 인덱스 사용
select * from test7 where a1=1 and a2=1; -- 인덱스 사용
select * from test7 where a2=1;          -- 인덱스 사용하지 않음
select * from test7 where a3=1;          -- 인덱스 사용
select * from test7 where a4=1;          -- 인덱스를 사용하지 않음
select a3 from test7 where a3=1;        -- 인덱스를 사용한다.
```

1. `select * from test7;` -- 인덱스를 사용하지 않음

생각해보자. 모든 데이터를 다 출력해야 한다면 어차피 처음부터 끝까지 다 검색해야하기 때문에 굳이 인덱스를 사용할 필요가 없다. 책으로 예를 들자면 처음부터 끝까지 다확인해 볼 예정인데 굳이 목차를 볼 필요는 없을 것이다.

2. `select * from test7 where a1=1;` -- 인덱스 사용

a1컬럼에 인덱스가 설정되어 있고 a1=1인 부분만 찾아서 일부분만 출력하기 때문에 인덱스를 사용하고 성능이 향상된다. 책으로 예를 들자면 목차에서 a=1인 부분을 찾아서 그부분만 확인 하기 때문에 목차를 확인해서 책에서 그부분만 확인 할 수 있다.

a1, a2 컬럼을 이용해서 인덱스를 만들었지만 부분적이나마 인덱스를 사용한다.

3. `select * from test7 where a1=1 and a2=1;` -- 인덱스 사용

4. `select * from test7 where a3=1;` -- 인덱스 사용

3,4번도 2번과 같은 상황이다.

5. `select * from test7 where a2=1;` 의 경우 인덱스 이지만 a1으로 정렬한 다음 그안에서 a2로 정렬하기 때문에 a2컬럼만으로 인덱스는 정렬되어 있지 않아 인덱스를 사용하지 않는다.

6. `select * from test7 where a4=1;` -- 인덱스를 사용하지 않음

a4컬럼 같은 경우 인덱스를 적용하지 않았기 때문에 인덱스를 사용하지 않는다. 책으로 예를 들자면 목차에 없는 내용은 목차로 사용할 수 없다.

7. `select a3 from test7 where a3=1;` -- 인덱스를 사용한다.

상위 쿼리중 가장 빠른 쿼리이다. 인덱스 만으로 원하는 결과를 얻을 수 있어 검색이 빠르다. 책으로 예를 들자면 찾고자 하는 내용을 목차로 확인 가능하면 굳이 내용을 확인하지 않아도 원하는 데이터를 얻을 수 있다.

인덱스에 사용하는 컬럼으로만 쿼리를 작성해야 인덱스를 사용할 수 있다. 상위 쿼리중 마지막 쿼리가 인덱스 설정 된 쿼리로 작성되어 있어서 가장 빠른 결과를 얻을 수 있다. 인덱스는 삽입 삭제 변경이 많은 테이블에서는 성능이 떨어져 사용하지 않는 것이 낫다. 책으로 생각해보자 책의 내용이 바뀌면 목차도 변경해야 한다.

인덱스는 여러개 생성할 수 있으나, 인덱스는 많은 공간을 차지하므로 꼭 필요한 인덱스를 최소한으로 만들어 사용하여야 한다.

같은 데이터가 많이 들어있는 컬럼은 인덱스로 만들지 않는것이 좋다.

외래키에 nonunique 인덱스를 사용하는 것이 좋다. 상위 모든 경우는 일반적인 경우이지 항상 맞는 것은 아닌다. 꼭 필요할 때만 인덱스를 사용 하자.

인덱스를 삭제하려면 다음과 같이 하면된다.

```
drop index idx_test7_a1a2;
```

idx\_test7\_a1a2은 삭제하고자하는 인덱스의 이름이다.

지금까지 만든 test 테이블들에 primary key들이 있어서 다음과 같은 sql문으로 이미 만들어져 있는 index 확인이 가능하다.

```
select * from user_indexes;  
select * from user_indexes where table_name='TEST7';
```

오라클에서는 기본적으로 오름차순(ASC)으로 인덱스를 생성합니다. 그러나 필요에 따라 내림차순(DESC)으로 인덱스를 생성할 수 있습니다.

```
CREATE INDEX 인덱스이름 ON 테이블이름(컬럼1 ASC, 컬럼2 DESC);
```

오름 차순으로 저장되어 있는데 내림차순으로 검색하면 속력은 더욱 떨어 질 것이다. 해당 컬럼이 큰수를 검색하는 일이 많다면 내림차순으로 인덱스를 만들어야 속력이 높아지고 작은 수를 검색하는 일이 많다면 오름차 순으로 저장되어 있어야 성능이 향상된다.

---

## > 35. 트랜잭션 이해하기

---

트랜잭션이란? 쪼갤 수 없는 실행 단위를 의미한다. 쪼갤 수 없어서 원자성 실행 결과가 일정해서 일관성을 가진다.

트랜잭션은 여러 개의 sql문 작업을 하나의 작업 처럼 만드는 방법이다.

본인이 원하는 어떤 작업이 있는데 작업중 다른 사람이 사용해서 문제가 발생될 수 있다면 내가 사용하는 동안 다른 사람이 사용하지 못하도록 막아야 한다. 트랜잭션으로 만들어 도중에 다른 사람이 사용하지 못하게 만들어야 한다. 원자성을 의미한다.

-- 계좌 1에서 200원을 출금

```
UPDATE 계좌 SET 잔액 = 잔액 - 200 WHERE 계좌번호 = 1;
```

-- 계좌 2로 200원을 입금

```
UPDATE 계좌 SET 잔액 = 잔액 + 200 WHERE 계좌번호 = 2;
```

상위와 같은 이체작업은 반드시 둘다 실행 되거나 둘다 실행되지 않아야 문제가 발생하지 않는다. 2개의 sql이 독립적인 sql이 되어 둘중에 하나만 실행되면 문제가 된다. 둘다 실행되거나 둘다 실행되지 않아야 한다. 이렇게 2개가 독립적이지 않은 하나의 작업(원자성)으로 만들려면 트랜잭션을 사용해야 한다.

오라클의 경우 sql작업이 들어가면 자동으로 트랜잭션이 시작되어 작업중 다른 사람이 해당 데이터베이스를 사용하지 못하게 된다. 또한 작업중 여러 문제가 발생하면 그동안 적용한 sql이 모두 롤백(취소) 된다. 정상적으로 여러 sql작업이 완료된 후 지금 까지의 작업을 적용하려면 `commit;` 해야 하고 취소 하려면 `rollback;` 를 해야 한다. 우리는 내용을 몰랐을 뿐 이미 트랜잭션을 사용해 오고 있었다.

다음은 추가 설명이다. 하나의 `insert` 문을 실행 시키면 이것은 하나의 작업이고 실행에 성공하면 데이터베이스에 데이터가 들어가고 작업에 실패하면 데이터가 들어가 있지 않을 것이다.

`insert`문 2개를 실행 시킬 때 하나는 맞고 하나는 틀렸을 경우 맞은 데이터는 데이터베이스에 들어가고 틀린 데이터는 데이터베이스에 들어가지 않을 것이다. 하지만, 둘중에

하나만 실행되면 안되는 상황이라면 둘다 맞을 때만 데이터 베이스에 둘다 넣고, 둘중 하나라도 실행되지 않은 데이터가 있을때 둘다 실행되지 않게 해야 한다. 이럴 때 2개의 insert문을 하나로 묶어 하나의 실행 단위가 되도록 하면 되는데 이런 작업들을 트랜잭션이라고 한다. 트랜잭션은 여러개의 작업을 쪼갤수 없는 하나의 작업으로 만드는 것을 의미한다.

트랜잭션 용어중 commit과 rollback이 있는데 commit은 지금까지 작업한 내용을 적용한다는 의미이고 rollback은 모두 취소 한다는 이야기이다.

오라클에서 sql를 실행하게 되면 바로 트랜잭션이 시작되고 트랜잭션이 종료되려면 commit이나 rollback를 적용해야하고 적용 후에는 새로운 트랜잭션이 시작된다.

한명이 다른 사람에게 입금해 주는 프로그램에서 본인 계좌에서 다른 사람 계좌로 보내려면 2개 이상의 sql이 필요하고 둘다 실행 되거나 실행 되지 않는다면 문제가 발생 할 것이다. 이때 트랜잭션을 사용하여 2개의 작업을 묶는다면 둘다 실행되거나 실행되지 않거나 하지 둘중 하나만 실행 되는 일은 절대로 발생하지 않는다.

다음 코드는 트랜잭션을 사용하여 2개의 데이터를 DB에 넣는 작업을 하는 코드이다. 트랜잭션으로 묶었기 때문에 둘다 실행되거나 둘다 실행되지 않는 경우는 있을 수 있어도 둘중 하나만 실행되는 경우는 있을 수 없다.

다음 예를 살펴보자. 재고 관리하는 프로그램에서 주문 10개를 받는다면 재고 테이블과 주문 테이블에서 다음과 같은 작업이 이루어 진다.

재고 테이블에 재고 수량 100개 가 있고, 주문 10개를 받으면 주문 테이블에 해당 물품을 10개를 추가 하고, 재고 테이블의 재고 수량을 90개로 만든다.

상위 작업을 SQL로 작성하려면 2개의 SQL이 사용된다.

1. `INSERT INTO 주문(주문수량) VALUES(10);`
2. `UPDATE 제품 SET 재고수량=90; --100개의 데이터를 90개로 변경한다.`

주문 10개를 처리 하려면 상위 2개의 쿼리가 모두 실행 되어야 한다. 만약에 1번만 실행이되고 2번은 실행되지 않으면 주문 10, 재고 100이되어 물품 수량에 문제가 발생할 것이다. 이 문제를 해결 하려면 1,2번을 하나의 실행 단위로 묶어서 둘다 실행 되든지 실행되지 않게 만들어야 한다. 이때 사용하는 방법이 트랜잭션이다.

오라클에서는 사용자가 데이터베이스를 사용하기 위해서 로그인 하는 순간부터 트랜잭션의 시작 부분이 되고 여러 데이터와 관련된 sql를 실행하다가 적용하기 위해서 트랜잭션을 끝내려면 `commit;`를 입력하고 그동안 데이터 조작 내용을 취소 하려면 `rollback;`를 입력 한다. `commit`이나 `rollback`이 실행되면 현재 트랜잭션은 종료되고 자동으로 다음 트랜잭션이 시작된다.

`insert,update,select,delete`와 같이 데이터를 조작하는 sql에 한해서 트랜잭션을 적용할 수 있다. `create,drop,alter`와 같이 구조 조작과 관련된 sql를 사용하면 기존 작업한 sql 내용은 자동으로 `commit`되고 새로운 트랜잭션이 시작 되므로 주의해서 사용하자.

한 유저가 데이터베이스를 접속해서 데이터를 조작하고 있다면 트랜잭션 중이어서 다른 유저가 해당 데이터를 조작하는 것에 지장을 줄수 있고, 한 사용자가 영원히 트랜잭션 종료를 하지 않으면 다른 사용자도 영원히 데이터베이스를 사용할 수 없게 된다. 다른 유저의 사용을 무한정 기다리는 상태를 교착 상태라고 한다.

종종 sql developer에서 데이터를 넣은 다음 프로그램에서 데이터베이스에 접근하여 데이터를 읽어오면 프로그램에서 sql developer를 통해서 입력한 데이터를 확인 못한다. 교착상태여서 그런 것이다. sql developer 사용자가 데이터를 조작하고 트랜잭션을 끝내지 않으면 다른 데이터베이스 사용자인 프로그램에서 확인하는 것은 불가능하다. 교착상태가 발생한 것이다. sql developer에서 데이터 조작후 사용을 완료 짓는 commit, rollback를 사용하여 트랜잭션 작업을 완료 해야만 다른 사람이 사용할 수 있다. sql developer 작업후 잊지 말고 commit, rollback를 실행해서 꼭 트랜잭션을 완료하자.

---

## > 36. HR 테이블 이해하기

---

다음 문제를 HR 데이터베이스를 풀어 보자.

1. 2000년에 입사해서 최고임금이 10000이 넘는 직종에서 일하고 있는 사원을 출력하시오.

```
select first_name, job_id, salary, hire_date  
from employees  
where to_char(hire_date, 'yyyy') = 2000  
and job_id in ( select job_id from jobs where max_salary > 10000)
```

2. 매니저의 이름이 MICHAEL인 부서를 출력하시오.

```
select * from departments where manager_id in  
(select employee_id  
from employees where upper(first_name) like '%MICHAEL%')
```

3. 직종의 최저임금이 사번 105번 사원의 임금보다 적은 직종을 출력하시오.

```
select * from jobs where min_salary <  
(select salary from employees where employee_id = 105)
```

4. 메일주소에 밑줄표시가 있는 사원을 출력하시오.

```
select * from employees where email like '%\_%' ESCAPE '\'  
_은 반드시 문자하나를 포함한다는 의미인데 실제 _를 검색에 사용하고 싶으면 escape  
'\'에서 정의한 \ 다음에 오는 문자 _은 하나의 문자를 포함하라는 의미가 아닌 데이터에  
들어 있는 _를 찾으라는 의미가 된다.
```

5. 사원 이름과 해당 사원의 매니저 이름을 출력하시오.

```
select e1.first_name Employee, e2.first_name Manager  
from employees e1, employees e2 where e1.manager_id = e2.employee_id
```

6. 30번 부서에 입사한 사원의 수를 연도별로 출력하시오.

```
select to_char(hire_date, 'yyyy'), count(*) from employees  
where department_id = 30 group by to_char(hire_date, 'yyyy');
```

7. 각 부서의 직종별로 사원들의 총 봉급합을 구해보자.

```
select department_id department, job_id job, sum(salary) TotalSalary  
from employees group by department_id, job_id  
order by department_id, job_id;
```

8. 업무별 최저임금과 최고임금 사이의 봉급을 가지는 사원의 이름과 업무명을 출력하시오.

```
select first_name, job_title from employees e, jobs j  
where salary between min_salary and max_salary order by first_name;
```

9. 커미션을 받은 사원과 받지 않은 사원이 각각 몇명인지 출력하시오.

```
select count(commission_pct) NoEmployeesWithCommission,  
count(*) - count(commission_pct) NoEmployeesWithoutCommission  
from employees;
```

10. 2월 28일에 입사한 사원의 이름, 업무명, 부서명을 출력하시오.

```
select first_name, job_title, department_name  
from employees e, jobs j, departments d  
where e.job_id = j.job_id and e.department_id = d.department_id  
and to_char(hire_date,'ddmm') = '2802';
```

11. 최저임금이 10000보다 높은 업무의 세부사항을 출력하시오.

```
SELECT * FROM JOBS WHERE MIN_SALARY > 10000
```

12. 2002년~2005년 사이에 입사한 사원의 이름과 입사일을 출력하시오.

```
SELECT FIRST_NAME, HIRE_DATE FROM EMPLOYEES  
WHERE TO_CHAR(HIRE_DATE, 'YYYY') BETWEEN 2002 AND 2005 ORDER BY HIRE_DATE;
```

13. IT Programmer이거나 Sales Man인 사원의 이름과 입사일을 출력하시오.

```
SELECT FIRST_NAME, HIRE_DATE  
FROM EMPLOYEES WHERE JOB_ID IN ('IT_PROG', 'SA_MAN');
```

14. 2008년 1월 1일 이후에 입사한 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES where hire_date > '01-jan-2008';
```

15. 사번이 150,160번인 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID in (150,160);
```

16. 임금이 10000보다 적은 사원의 이름, 임금, 커미션, 입사일을 출력하시오.

```
SELECT FIRST_NAME, SALARY, COMMISSION_PCT, HIRE_DATE  
FROM EMPLOYEES WHERE SALARY < 10000;
```

17. 최고 임금이 10000~20000인 업무의 업무명, 최저임금과 최고임금의 차액을 출력하시오.

```
SELECT JOB_TITLE, MAX_SALARY-MIN_SALARY DIFFERENCE FROM JOBS WHERE MAX_SALARY
```

```
BETWEEN 10000 AND 20000;
```

18. 사원 이름과 임금을 출력하되 임금을 1000단위로 출력하시오.

```
SELECT FIRST_NAME, SALARY, ROUND(SALARY, -3) FROM EMPLOYEES;
```

19. 업무명을 기준으로 하여 내림차순으로 업무 세부사항을 출력하시오.

```
SELECT * FROM JOBS ORDER BY JOB_TITLE desc;
```

20. 성이나 이름이 S로 시작하는 사원을 출력하시오.

```
SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE FIRST_NAME LIKE 'S%' OR  
LAST_NAME LIKE 'S%';
```

21. 5월에 입사한 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE, 'MON')= 'MAY';
```

--64page↓

22. 커미션을 받지 않고, 임금이 5000~10000이면서 30번 부서에 소속된 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES WHERE COMMISSION_PCT IS NULL AND SALARY BETWEEN 5000  
AND 10000 AND DEPARTMENT_ID=30;
```

23. 사원들의 이름과 첫 월급날을 출력하시오. 매월 1일이 월급 날이다.

```
SELECT FIRST_NAME, HIRE_DATE, LAST_DAY(HIRE_DATE)+1 FROM EMPLOYEES;
```

24. 사원의 이름과 근속연수를 출력하시오.

```
SELECT FIRST_NAME, HIRE_DATE, FLOOR((SYSDATE-HIRE_DATE)/365)FROM EMPLOYEES;
```

25. 2001년에 입사한 사원의 이름을 출력하시오.

```
SELECT FIRST_NAME, HIRE_DATE FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE,  
'YYYY' )=2001;
```

26. 첫 글자는 대문자로, 나머지 글자는 소문자로 변환하여 성과 이름을 출력하시오.  
initcap 사용

```
SELECT INITCAP(FIRST_NAME), INITCAP(LAST_NAME) FROM EMPLOYEES;
```

27. 업무명의 첫 단어를 출력하시오.

```
SELECT JOB_TITLE, SUBSTR( JOB_TITLE, 1, INSTR(JOB_TITLE || ' ', ' ') - 1 ) FROM  
JOBS;
```

28. 성의 3번째 문자 이후에 'b'가 포함된 사원의 이름의 길이를 출력하시오.

```
SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEES WHERE INSTR(LAST_NAME, 'B') > 3
```

29. 이름과 이메일주소가 대소문자 상관없이 같은 사원의 이름을 대문자로, 이메일주소는 소문자로 출력하시오.

```
SELECT UPPER(FIRST_NAME), LOWER(EMAIL) FROM EMPLOYEES WHERE UPPER(FIRST_NAME)=UPPER(EMAIL);
```

30. 올해 입사한 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE, 'YYYY')=TO_CHAR(SYSDATE, 'YYYY');
```

31. 현재 날짜와 2011년 1월 1일이 며칠 차이인지 출력하시오.

```
SELECT SYSDATE - to_date('01-jan-2011') FROM DUAL;
```

32. 올해 각 달마다 몇 명의 사원이 입사하였는지 출력하시오.

```
SELECT TO_CHAR(HIRE_DATE, 'MM'), COUNT(*) FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE, 'YYYY')= TO_CHAR(SYSDATE, 'YYYY') GROUP BY TO_CHAR(HIRE_DATE, 'MM');
```

33. 매니저 사번과 그 매니저가 총 몇명을 관리하는지 출력하시오.

```
SELECT MANAGER_ID, COUNT(*) FROM EMPLOYEES GROUP BY MANAGER_ID; --e1.mgr = e2.empid 이렇게 어렵게 쓸 필요 없음!
```

34. 사원 사번과 이전 업무를 종료한 날을 출력하시오.

```
SELECT EMPLOYEE_ID, MAX(END_DATE) FROM JOB_HISTORY GROUP BY EMPLOYEE_ID;
```

35. 입사일이 n월 '15일' 이후인(몇월이든 월말에 입사한) 사원의 수를 출력하시오.

```
SELECT COUNT(*) FROM EMPLOYEES WHERE TO_CHAR(HIRE_DATE, 'DD') > 15;
```

36. 국가번호와 해당 국가에 있는 도시들의 개수를 출력하시오.

```
SELECT COUNTRY_ID, COUNT(*) FROM LOCATIONS GROUP BY COUNTRY_ID;
```

37. 각 부서별로 커미션을 받은 사원들의 평균임금을 출력하시오.

```
SELECT DEPARTMENT_ID, AVG(SALARY) FROM EMPLOYEES WHERE COMMISSION_PCT IS NOT NULL GROUP BY DEPARTMENT_ID;
```

38. 업무별로 업무명, 사원의 수, 임금의 합, 최고임금과 최저임금의 차액을 출력하시오.

```
SELECT JOB_ID, COUNT(*), SUM(SALARY), MAX(SALARY)-MIN(SALARY) SALARY FROM EMPLOYEES GROUP BY JOB_ID;
```

39. 평균임금이 10000을 넘는 업무의 업무ID와 평균임금을 출력하시오.

```
SELECT JOB_ID, AVG(SALARY) FROM EMPLOYEES
```

```
GROUP BY JOB_ID HAVING AVG(SALARY)>10000;
```

40. 10명이 넘는 사원이 입사한 연도를 출력하시오.

```
SELECT TO_CHAR(HIRE_DATE, 'YYYY') FROM EMPLOYEES  
GROUP BY TO_CHAR(HIRE_DATE, 'YYYY') HAVING COUNT(EMPLOYEE_ID) > 10;
```

41. 커미션을 받은 사원이 5명 이상인 부서를 출력하시오.

```
SELECT DEPARTMENT_ID FROM EMPLOYEES WHERE COMMISSION_PCT IS NOT NULL  
GROUP BY DEPARTMENT_ID HAVING COUNT(COMMISSION_PCT)>5;
```

42. 과거에 하나가 넘는(하나 이상 말고 초과)의 업무를 하였던 사원의 사번을 출력하시오.

```
SELECT EMPLOYEE_ID FROM JOB_HISTORY GROUP BY EMPLOYEE_ID HAVING COUNT(*) > 1;
```

43. 3명이 넘는 사원이 100일이 넘게 근무한 업무ID를 출력하시오.

```
SELECT JOB_ID FROM JOB_HISTORY WHERE END_DATE-START_DATE > 100  
GROUP BY JOB_ID HAVING COUNT(*)>3;
```

44. 부서 번호, 연도, 연도별 입사한 사원의 수를 조인하여 출력하시오.

```
SELECT DEPARTMENT_ID, TO_CHAR(HIRE_DATE, 'YYYY'), COUNT(EMPLOYEE_ID)  
FROM EMPLOYEES  
GROUP BY DEPARTMENT_ID, TO_CHAR(HIRE_DATE, 'YYYY') ORDER BY DEPARTMENT_ID;
```

45. 임의의 매니저가 5명 이상의 사원을 관리하고 있는 부서를 출력하시오.

```
SELECT DISTINCT DEPARTMENT_ID --any manager이므로 distinct 하여 한번만 출력  
FROM EMPLOYEES  
GROUP BY DEPARTMENT_ID, MANAGER_ID HAVING COUNT(EMPLOYEE_ID) > 5;
```

46. 사번 115번 사원의 현재 임금이 6000미만일 경우 8000으로 변경하시오.

```
UPDATE EMPLOYEES SET SALARY = 8000 WHERE EMPLOYEE_ID = 115 AND SALARY < 6000;
```

47. 사원 테이블에 모든 데이터를 입력하여 새로운 사원을 추가하시오.

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL,  
PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, DEPARTMENT_ID)  
VALUES (207, 'ANGELA', 'SNYDER', 'ANGELA', '215 253 4737', SYSDATE, 'SA_MAN',  
12000, 80);
```

48. 20번 부서를 삭제하시오.

```
DELETE FROM DEPARTMENTS WHERE DEPARTMENT_ID=20;
```

49. 사번 110번 사원이 10번 부서에 속해있고 현재 업무ID가 IT로 시작하지 않으면 해당 사원의 업무ID를 'IT\_PROG'로 변경하시오.

```
UPDATE EMPLOYEES SET JOB_ID= 'IT_PROG'  
WHERE EMPLOYEE_ID=110 AND DEPARTMENT_ID=10 AND NOT JOB_ID LIKE 'IT%';
```

50. 부서 테이블에 매니저ID가 120이고 장소ID는 도쿄 시의 임의의 장소로 하는 행을 추가하시오.

```
INSERT INTO DEPARTMENTS (150,'SPORTS',120,1200);
```

51.부서 이름과 해당 부서의 사원들 수를 출력하시오.

```
SELECT DEPARTMENT_NAME, COUNT(*) FROM EMPLOYEES JOIN DEPARTMENTS USING  
(DEPARTMENT_ID) GROUP BY DEPARTMENT_NAME;
```

52.업무기록에서 30번 부서의 모든 업무에 대하여 업무명, 사번, 시작일과 종료일 간 근무일수를 출력하시오.

```
SELECT EMPLOYEE_ID, JOB_TITLE, END_DATE-START_DATE DAYS  
FROM JOB_HISTORY NATURAL JOIN JOBS WHERE DEPARTMENT_ID=30;
```

53. 부서명과 매니저 이름을 출력하시오.

```
SELECT DEPARTMENT_NAME, FIRST_NAME FROM DEPARTMENTS D JOIN EMPLOYEES E ON  
(D.MANAGER_ID=E.EMPLOYEE_ID);
```

54.부서명과 매니저 이름, 도시를 출력하시오.

```
SELECT DEPARTMENT_NAME, FIRST_NAME, CITY FROM DEPARTMENTS D JOIN EMPLOYEES E ON  
(D.MANAGER_ID=E.EMPLOYEE_ID) JOIN LOCATIONS L USING (LOCATION_ID);
```

55.국가명, 도시, 부서명을 출력하시오.

```
SELECT COUNTRY_NAME, CITY, DEPARTMENT_NAME  
FROM COUNTRIES JOIN LOCATIONS USING (COUNTRY_ID)  
JOIN DEPARTMENTS USING (LOCATION_ID);
```

56.2000~2005년 사이의 모든 업무에 대하여 업무명, 부서명, 사원의 성, 업무시작일을 출력하시오.

```
SELECT JOB_TITLE, DEPARTMENT_NAME, LAST_NAME, START_DATE  
FROM JOB_HISTORY JOIN JOBS USING (JOB_ID) JOIN DEPARTMENTS  
USING (DEPARTMENT_ID) JOIN EMPLOYEES USING (EMPLOYEE_ID)  
WHERE TO_CHAR(START_DATE,'YYYY') BETWEEN 2000 AND 2005;
```

57.업무명과 사원들의 평균임금을 출력하시오.

```
SELECT JOB_TITLE, AVG(SALARY) FROM EMPLOYEES  
NATURAL JOIN JOBS GROUP BY JOB_TITLE;
```

58.업무명, 사원 이름, 해당 사원의 임금과 해당 업무의 최고임금 간의 차액을 출력하시오.

```
SELECT JOB_TITLE, FIRST_NAME, MAX_SALARY-SALARY DIFFERENCE FROM EMPLOYEES
```

```
NATURAL JOIN JOBS;
```

59. 커미션을 받고 30번 부서에 속한 사원의 성, 업무명을 출력하시오.

```
SELECT JOB_TITLE, FIRST_NAME, MAX_SALARY-SALARY DIFFERENCE FROM EMPLOYEES  
NATURAL JOIN JOBS WHERE DEPARTMENT_ID = 30;
```

60. 현재 임금이 15000 이상인 사원이 지금까지 수행했던 업무들을 출력하시오.

```
SELECT JH.*  
FROM JOB_HISTORY JH JOIN EMPLOYEES E ON (JH.EMPLOYEE_ID = E.EMPLOYEE_ID)  
WHERE SALARY > 15000;
```

61.5년 이상 근무한 모든 매니저들의 부서명, 매니저 이름, 매니저의 임금을 출력하시오.

```
SELECT DEPARTMENT_NAME, FIRST_NAME, SALARY  
FROM DEPARTMENTS D JOIN EMPLOYEES E ON (D.MANAGER_ID=E.MANAGER_ID)  
WHERE (SYSDATE-HIRE_DATE) / 365 > 5;
```

62. 자신의 매니저보다 먼저 입사한 사원 이름을 출력하시오.

```
SELECT FIRST_NAME FROM EMPLOYEES E1 JOIN EMPLOYEES E2 ON  
(E1.MANAGER_ID=E2.EMPLOYEE_ID)  
WHERE E1.HIRE_DATE < E2.HIRE_DATE;
```

63. 사원이 6개월 미만으로 근무한 업무에 대하여 사원 이름, 업무명을 출력하시오.

```
SELECT FIRST_NAME, JOB_TITLE FROM EMPLOYEES E JOIN JOB_HISTORY JH ON  
(JH.EMPLOYEE_ID = E.EMPLOYEE_ID)  
JOIN JOBS J ON( JH.JOB_ID = J.JOB_ID) WHERE  
MONTHS_BETWEEN(END_DATE,START_DATE) < 6;
```

64. 사원의 이름과 그 사원이 근무하는 국가를 출력하시오.

```
SELECT FIRST_NAME, COUNTRY_NAME FROM EMPLOYEES JOIN DEPARTMENTS  
USING(DEPARTMENT_ID)  
JOIN LOCATIONS USING( LOCATION_ID)  
JOIN COUNTRIES USING ( COUNTRY_ID);
```

65. 부서명, 평균임금, 부서 내에서 커미션을 받은 사원의 수를 출력하시오.

```
SELECT DEPARTMENT_NAME, AVG(SALARY), COUNT(COMMISSION_PCT)  
FROM DEPARTMENTS JOIN EMPLOYEES USING (DEPARTMENT_ID)  
GROUP BY DEPARTMENT_NAME;
```

66. 시애틀에 위치한 부서 어디든 사원이 5명 이상 입사한 달을 출력하시오.

```
SELECT TO_CHAR(HIRE_DATE,'MON-YY')  
FROM EMPLOYEES JOIN DEPARTMENTS USING (DEPARTMENT_ID) JOIN LOCATIONS USING  
(LOCATION_ID)
```

```
WHERE CITY = 'Seattle' GROUP BY TO_CHAR(HIRE_DATE, 'MON-YY')
HAVING COUNT(*) > 5;
```

67. 최고임금이 10000이 넘는 부서의 세부사항을 출력하시오.

```
SELECT * FROM DEPARTMENTS WHERE DEPARTMENT_ID IN
( SELECT DEPARTMENT_ID FROM EMPLOYEES
GROUP BY DEPARTMENT_ID HAVING MAX(SALARY)>10000);
```

68. 'Smith'에 의해 관리되는 부서의 세부사항을 출력하시오.

```
SELECT * FROM DEPARTMENTS WHERE MANAGER_ID IN
(SELECT EMPLOYEE_ID FROM EMPLOYEES WHERE FIRST_NAME='SMITH');
```

69. 올해 입사한 사원의 업무 세부사항을 출력하시오.

```
SELECT * FROM JOBS WHERE JOB_ID IN
(SELECT JOB_ID FROM EMPLOYEES WHERE
TO_CHAR(HIRE_DATE, 'YYYY')=TO_CHAR(SYSDATE, 'YYYY'));
```

70. 과거에 다른 어떠한 업무도 수행하지 않은 사원을 출력하시오.

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID NOT IN
(SELECT EMPLOYEE_ID FROM JOB_HISTORY);
```

71. 과거에 다른 업무를 한 적이 있는 사원의 업무와 평균임금을 출력하시오.

```
SELECT JOB_TITLE, AVG(SALARY) FROM JOBS NATURAL JOIN EMPLOYEES
GROUP BY JOB_TITLE
WHERE EMPLOYEE_ID IN
(SELECT EMPLOYEE_ID FROM JOB_HISTORY);
```

72. 5명 이상의 사원이 속한 부서만 선정하여 부서가 있는 국가명, 도시, 총 부서의 수를 출력하시오.

```
SELECT COUNTRY_NAME, CITY, COUNT(DEPARTMENT_ID)
FROM COUNTRIES JOIN LOCATIONS USING (COUNTRY_ID) JOIN DEPARTMENTS USING
(LOCATION_ID)
WHERE DEPARTMENT_ID IN
(SELECT DEPARTMENT_ID FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING COUNT(DEPARTMENT_ID)>5)
GROUP BY COUNTRY_NAME, CITY;
```

73. 5명 이상의 사원을 관리하는 매니저의 세부사항을 출력하시오.

```
SELECT FIRST_NAME FROM EMPLOYEES
WHERE EMPLOYEE_ID IN
```

```
(SELECT MANAGER_ID FROM EMPLOYEES  
GROUP BY MANAGER_ID HAVING COUNT(*)>5);
```

74. 커미션을 받지않은 사원들의 사원 이름, 과거 업무명, 업무시작일, 업무종료일을 출력하시오.

```
SELECT FIRST_NAME, JOB_TITLE, START_DATE, END_DATE  
FROM JOB_HISTORY JH JOIN JOBS J USING (JOB_ID) JOIN EMPLOYEES E ON (  
JH.EMPLOYEE_ID = E.EMPLOYEE_ID)  
WHERE COMMISSION_PCT IS NULL;
```

75. 지난 2년 간 사원이 새로 들어오지 않은 부서를 출력하시오.

```
SELECT * FROM DEPARTMENTS  
WHERE DEPARTMENT_ID NOT IN  
(SELECT DEPARTMENT_ID FROM EMPLOYEES WHERE FLOOR((SYSDATE-HIRE_DATE)/365) < 2);
```

76. 과거에 다른 업무를 수행했던 사원들 중에 최고임금이 10000 이상인 사원이 속한 부서의 세부사항을 출력하시오.

```
SELECT * FROM DEPARTMENTS  
WHERE DEPARTMENT_ID IN  
(SELECT DEPARTMENT_ID FROM EMPLOYEES  
WHERE EMPLOYEE_ID IN (SELECT EMPLOYEE_ID FROM JOB_HISTORY)  
GROUP BY DEPARTMENT_ID HAVING MAX(SALARY) >10000);
```

77. 과거에 IT Programmer로 일했던 사원들의 현재 업무를 출력하시오.

```
SELECT * FROM JOBS WHERE JOB_ID IN  
(SELECT JOB_ID FROM EMPLOYEES WHERE EMPLOYEE_ID IN  
(SELECT EMPLOYEE_ID FROM JOB_HISTORY WHERE JOB_ID='IT_PROG'));
```

78. 각 부서에서 가장 높은 임금을 받는 사원을 출력하시오.

```
SELECT DEPARTMENT_ID, FIRST_NAME, SALARY FROM EMPLOYEES OUTER WHERE SALARY =  
(SELECT MAX(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID =  
OUTER.DEPARTMENT_ID);
```

79. 사번이 105번인 사원이 있는 도시를 출력하시오.

```
SELECT CITY FROM LOCATIONS WHERE LOCATION_ID =  
(SELECT LOCATION_ID FROM DEPARTMENTS WHERE DEPARTMENT_ID =  
(SELECT DEPARTMENT_ID FROM EMPLOYEES WHERE EMPLOYEE_ID=105)  
);
```

80. 모든 사원들 중 3번째로 높은 임금은 얼마인지 출력하시오.

```
select salary from employees main
```

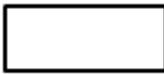
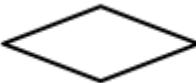
```
where 2 = (select count( distinct salary )
from employees where salary > main.salary);
```

## 02. 데이터베이스 모델링

### > 01. ER 다이어그램

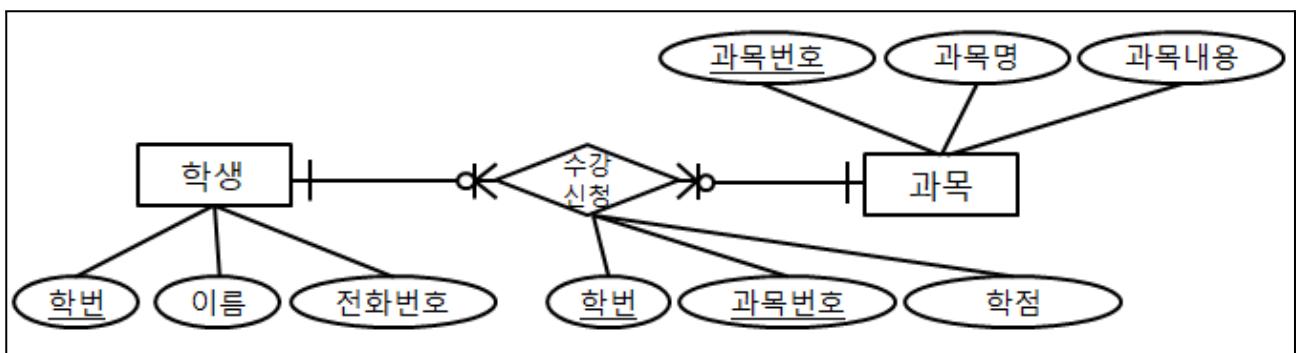
ER다이어그램(ERD)란? 현실세계의 객체를 데이터베이스의 테이블에 넣는 과정에서 여러 사람이 알아보기 쉽게 기호로 그려 표현한 것이다. 테이블은 사각형 컬럼은 동그라미 기본키는 동그라미에 밑줄로 표시한다.

ER다이어그램은 개체 속성 관계 등을 다음과 같은 기호들을 사용하여 그림으로 표현한다. 다음 표를 보고 각각의 기호를 이해해 보자.

기호	설명
	<p>객체라 부르고 현실세계의 데이터를 데이터베이스의 테이블로 넣을 때 사용한다.</p> <p>개체(entities)는 현실 세계에 존재하는 식별 가능한 모든 것을 의미하고 프로그램에서 이들은 class로 데이터 베이스에서는 table로 만들어 사용 한다. 객체는 학생, 학점, 과목, 등과 같이 독립된 의미를 가진다.</p>
 	<p>속성이라 부르고 테이블의 컬럼을 표현 할 때 사용한다.</p> <p>속성(attribute)은 개체가 가지고 있는 특성, 개체를 이루는 구성 요소들을 의미한다. 예를 들면 학생은 학번, 이름, 전화번호 등의 속성으로 이루어져 있다. 과목은 과목번호, 과목명, 과목내용 등의 속성으로 이루어져 있다. 학점은 학번, 과목 번호, 수업시간 등으로 이루어져 있다. 속성은 프로그램에서는 클래스의 필드 데이터베이스에서는 테이블의 컬럼이 된다.</p> <p>속성 기호에 밑줄이 들어간 경우 해당 속성은 개체에 들어 있는 데이터들을 구분하는 식별자 기본키에 해당 한다.</p>
	<p>관계(relationship)는 두 개체간의 관계를 통해 생성된 새로운 개체를 의미한다. 예를 들면 학생, 과목 객체가 있다면 학생은</p>

수업을 수강하게 될 것이고 두 객체 사이에는 수강이라는 새로운 관계가 만들어 진다. 프로그램에서는 관계를 통해서 클래스간 의존관계가 생성되고 데이터 베이스에서는 두 테이블의 관계를 통해서 새로운 테이블로 만들어 표현한다. 관계는 결국 새로운 테이블로 만들 수 있어서 ERD에서 종종 관계를 객체 형태로 그린다.

다음은 현실 세계의 수강 신청을 ERD로 옮긴 것이다.

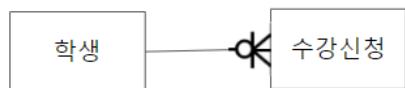


관계는 보통 테이블로 변환되기 때문에 몇몇 툴에서는 다이아 몬드가 아닌 사각형으로 표현하고 객체(테이블)처럼 사용한다.

두 테이블 간의 관계를 시각적으로 나타낼 때 선으로 연결하고, 선 끝에 기호를 표시하여 관련 데이터의 수 있다. 선 끝에 0이면 해당 관계에서 데이터가 없음을 의미하고, 1은 1개의 데이터와 관계가 있고, 새발은 여러개의 데이터와 관계가 있음을 나타낸다.

다음 학생과 수강신청에 대한 관계를 살펴보자. 왼쪽은 학생테이블 오른쪽은 수강신청 테이블이다.

select name, phone, student_id from student;			select * from register;		
결과 x SQL   인출된 모든 행: 3(0,003초)			결과 x SQL   인출된 모든 행: 4(0,017초)		
NAME PHONE STUDENT_ID 홍길동 01099999999 1 홍길남 01088888888 2 홍길순 01077777777 3			STUDENT_ID COURSE_ID GRADE 1 1 A 1 2 B 1 3 C 2 1 C		



하나의 학생은 수강 신청하지 않을 수 도 있고, 하나만 할 수 도 있고, 여려개 할 수도 있다. 그림으로 그리면 왼쪽과 같다.

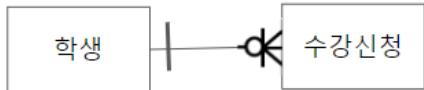
The image shows two database interfaces. On the left, a table named 'student' is displayed with columns 'NAME', 'PHONE', and 'STUDENT\_ID'. It contains three rows: '홍길동' (01099999999), '홍길남' (01088888888), and '홍길순' (01077777777). On the right, a table named 'register' is shown with columns 'STUDENT\_ID', 'COURSE\_ID', and 'GRADE'. It contains four rows: (1, 1, 'A'), (1, 2, 'B'), (1, 3, 'C'), and (2, 1, 'C'). Arrows from the 'STUDENT\_ID' column in the 'register' table point to the corresponding 'STUDENT\_ID' values in the 'student' table.

select name, phone, student_id from student;			select * from register;		
결과 x			결과 x		
SQL   인출된 모든 행: 3(0,003초)			SQL   인출된 모든 행: 4(0,017초)		
NAME	PHONE	STUDENT_ID	STUDENT_ID	COURSE_ID	GRADE
홍길동	01099999999	1	1	1	A
홍길남	01088888888	2	1	2	B
홍길순	01077777777	3	1	3	C
			2	1	C

하나의 수강신청은 반드시 하나의 학생만 수강신청을 할 수 있다.



최종적으로 두 테이블간의 관계를 왼쪽처럼 표시한다.



두 테이블 간의 데이터 관계를 새발 표기법으로 나타내는 방법은 다음과 같다 왼쪽 테이블과 오른쪽 테이블이 있다고 생각하고 상위 그림을 보고 이해해 보자.

### 1. 왼쪽 테이블의 데이터가 오른쪽 테이블의 데이터와의 관계:

- 왼쪽 테이블의 각 하나하나의 데이터에 대해 오른쪽 테이블의 데이터와의 관계를 나타내기 위해 오른쪽 사각형 옆에 새다리 모양으로 선을 그립니다.
- 이때 왼쪽 사각형 옆에 새다리관련 모양은 아무 영향을 주지 않는다.
- 선 끝에 표시된 숫자는 해당 관계에서의 데이터 수를 나타냅니다. 0은 관계가 없음, 1은 1개, 새 다리는 여러개의 데이터와의 관계를 의미합니다.

### 2. 오른쪽 테이블의 데이터가 왼쪽 테이블과의 관계:

- 반대로, 오른쪽 테이블의 각 데이터에 대해 왼쪽 테이블과의 관계를 나타내기 위해 왼쪽 사각형 옆에 새다리 모양으로 선을 그립니다.
- 이때 오른 사각형 옆에 새다리관련 모양은 아무 영향을 주지 않는다.
- 선 끝에 표시된 숫자는 해당 관계에서의 데이터 수를 나타냅니다. 0은 관계가 없음, 1은 1개, 새 다리는 여러개의 데이터와의 관계를 의미합니다.

관계가 있다는 이야기는 조인을 할 수 있다는 이야기이다. 테이블 간에 관계가 생성되면 테이블에 들어 있는 하나의 데이터는 다른 테이블에 들어 있는 데이터들과 관계를 가진다.

새발 표기법은 두 테이블 안에 들어 있는 데이터의 관계를 그림으로 그리는 방법이다.

두 테이블에 들어 있는 데이터의 관계를 그림으로 그리면 왼쪽과 같이 새다리 모양으로 그린다. 영어로는 CROW'S FOOT 표기법 까마귀발 표기법이라 한다.

테이블의 하나의 데이터가 다른 테이블의 데이터와 관계를 가지지 않는다면 동그라미 모양을 사용하여 표현 한다. 관계를 가지지 않는다는 의미는 연결되는 데이터가 없다는 의미이다. 이를 1대0 관계라고 한다. 이 경우 해당되는 테이블의 하나의 데이터는 조인 할 데이터가 없어 이 퀴 조인(동등 조인)을 하게 되면 연결할 데이터가 없으니 사라지게 된다.

예를 들어 보면 학생과 수강 테이블에서 학생이 수강하는 과목이 하나도 존재하지 않을 수 있다면 하나의 학생은(1) 수강하지 않아도(0)되므로 1:0관계라 한다.

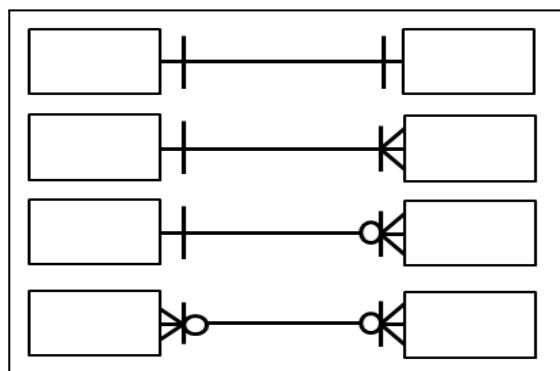
테이블의 하나의 데이터가 다른 테이블 하나의 데이터와 관계를 가지면 세로 작대기 모양을 사용하여 표현하고 1대1 관계라고 이야기 한다. 이 경우 해당되는 테이블의 하나의 데이터는 반드시 반대쪽 테이블의 하나의 데이터와 조인하게 된다.

예를 들어 보면 학생과 수강 테이블에서 학생이 수강하는 과목이 하나만 존재 할 수 있다면 하나의 학생은(1) 한과목만 수강(1)할 수 있으므로 1:1관계라 한다.

테이블의 하나의 데이터가 다른 테이블 여러 개의 데이터와 관계를 가진다면 새다리 모양처럼 된 세개의 막대기 모양을 사용하여 표현하고 1 대 다, 1 대 N 관계라고 이야기 한다. 이 경우 해당되는 테이블의 하나의 데이터는 조인하게 되면 반대쪽 테이블의 N개와 N번 합쳐져야 하므로 총 N개의 데이터가 만들어진다.

두 테이블에서 발생할 수 있는 1대0, 1대1, 1대N 등과 같은 관계를 세다리로 표현 할 수 있다.

예를 들어 보면 학생과 수강 테이블에서 학생이 수강하는 과목이 여러개 존재 할 수 있다면 하나의 학생은(1) 여러 과목을 수강(n)할 수 있으므로 1:N관계라 한다.



왼쪽 이미지에서 왼쪽 테이블을 A, 오른쪽 테이블을 B라 하고 A를 기준으로 설명하면 A 테이블의 하나의 데이터가 B 테이블 데이터와 관계가 어떻게 되는지 B쪽 테이블 옆의 새다리로 표현해 준다. 그 다음 반대로 기준 테이블이 B 테이블이라면 B 테이블의 하나의 데이터가 A 테이블 데이터와 관계가 어떻게 되는지 확인하여 A 테이블 옆의 새다리로 표현해주면 두 테이블 간의 관계 표현이 끝난다.

상위 4개의 그림을 보고 두 테이블 관계를 이야기해 보면 다음과 같다.

왼쪽 테이블을 A, 오른쪽 테이블을 B라 할 때

1번째 그림을 글로 읽어 보면 다음과 같다. A테이블의 하나의 데이터는 B테이블의 하나의 데이터와 관계가 있다. B테이블의 하나의 데이터는 A테이블의 하나의 데이터와 관계가 있다.

2번째 그림을 글로 읽어 보면 다음과 같다. A테이블의 하나의 데이터는 B테이블의 하나의 데이터 혹은 여러개의 데이터가 관계가 있다. B테이블의 하나의 데이터는 A테이블의 하나의 데이터와 관계가 있다.

3번째 그림을 글로 읽어 보면 다음과 같다. A테이블의 하나의 데이터는 B테이블의 데이터와 관계가 없거나, 하나의 데이터 혹은 여러개의 데이터와 관계가 있다. B테이블의 하나의 데이터는 A테이블의 하나의 데이터와 관계가 있다.

4번째 그림을 글로 읽어 보면 다음과 같다. A테이블의 하나의 데이터는 B테이블의 데이터와 관계가 없거나, 하나의 데이터 혹은 여러개의 데이터와 관계가 있다. B테이블의 하나의 데이터는 A테이블의 데이터와 관계가 없거나, 하나의 데이터 혹은 여러개의 데이터와 관계가 있다.

아래의 그림은 테이블에 A,B테이블과 관계를 가지는 컬럼의 값들만 넣은 것이다. 보통 조인할 때 사용하는 컬럼의 값을 표시한 것이다. 어떤 관계를 가지는지 왼쪽부터 A,B테이블의 관계를 새다리 표기법으로 그려보자.

A	B	A	B	A	B	A	B
1	1	1	1	1	1	1	1
2	4	2	2	2	1	2	1
3	2	2	2	3	2	3	2
4	5	2	2	4	2	1	7
5	3	2	2	5	2	5	8
6	6	2	6	6	3	6	9

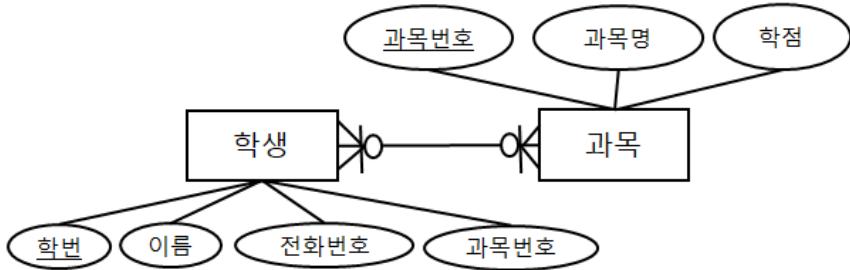
정답은 이전 새다리 그림 순서와 동일 하다.

문제가 되는 테이블을 찾아 보자.

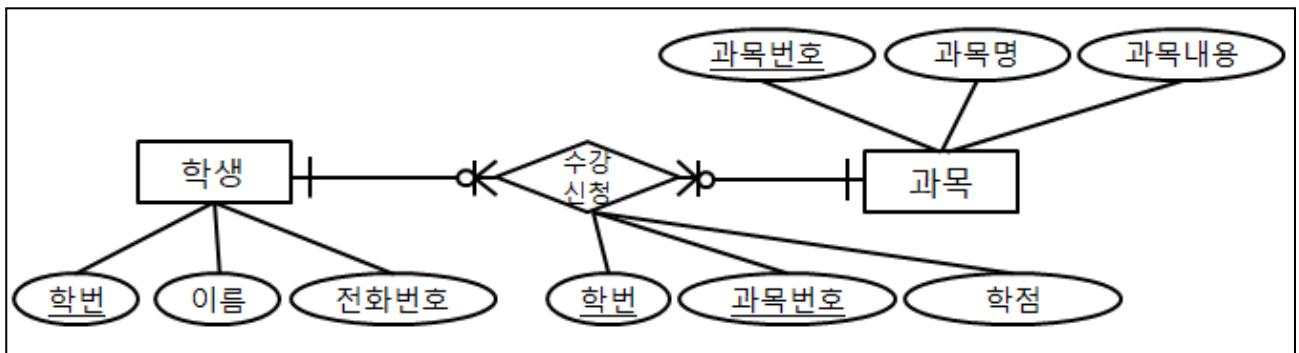
상위 4개의 그림 중 1번째 그림의 1:1 관계는 사실상 하나의 테이블을 둘로 쪼개야만 나올 수 있는 형태여서 일반적인 테이블 간의 관계에서는 잘 나오지 않는다. 1:1 관계는 잘 생각해 보면 원래 하나의 테이블이다. 그리고 4번째 다대다 N:N관계는 일반적인 테이블

관계에서 나올 수 없는 형태이다. 본인이 의도한 것이 아니라면 테이블 설계가 잘못되어 나타난 경우여서 N:1관계를 가지는 3개의 테이블로 변경하여야 한다. 아래에 설계한 학생, 수강신청, 과목을 잘못 설계하여 학생, 과목 테이블로 만들었다면 N:N관계가 나타나는 것을 확인 할 수 있고 이를 해결하기 위해서는 학생, 수강신청, 과목의 3개의 테이블로 분리해서 테이블간의 관계를 1:n 관계로 만들어야 한다.

다음은 잘못 설계된 테이블을 N:N를 정상적인 ERD와 비교해보자.



학생의 학점을 어떻게 처리할지 고민하다가 과목에 학번과 학점을 추가하게 되면 과목번호와 과목명이 중복되게 된다. 학점을 학생테이블에 넣게 되면 학생테이블에 과목번호를 넣어 처리해야하기 때문에 올바르게 ERD를 그리면 다음과 같다.



학생과 과목은 개체이고 두 개체 사이에 수강 신청이라는 관계가 있다.

학생객체는 학번, 이름, 전화번호의 속성을 가지고 학번은 기본키값에 해당 한다.  
과목객체는 과목번호, 과목명, 과목내용의 속성을 가지고 과목번호가 기본키값에 해당한다.  
수강신청은 학생과 과목사이의 관계에 의해서 발생한 관계이며 학번, 과목번호, 수업시간과 같은 속성을 가지고 학번, 과목 번호를 기본키로 가지고 있어 복합키를 가지며 학생 테이블과 관련된 외래키 학번 컬럼과 과목테이블과 관련된 외래키 과목번호가 있다.

학생과 수강신청의 관계를 새발 표현법으로 이야기해 보자면 다음과 같다.

하나의 학생은 수강신청이 없어도 된다. 이 문제에 대해서 실제 DB를 만들때 허용하고 싶다면 상위 이미지처럼 학생과 수강신청을 연결하는 선에 수강신청쪽에 동그라미를 추가하였다.

하나의 학생은 하나의 수강신청을 할 수 있다. 이 문제에 대해서 실제 DB를 만들때

허용하고 싶다면 상위 이미지처럼 학생과 수강신청을 연결하는 선에 수강신청쪽에 세로선을 추가하면 된다.

하나의 학생은 여러개의 수강신청을 할 수 있다. 이 문제에 대해서 실제 DB를 만들때 허용하고 싶다면 상위 이미지처럼 학생과 수강신청을 연결하는 선에 수강신청쪽에 새발모양의 선을 추가하면 된다.

하나의 수강 신청은 하나의 학생 정보를 가지고 있다. 이 문제에 대해서 실제 DB를 만들때 허용하고 싶다면 상위 이미지처럼 학생과 수강신청을 연결하는 선에 학생쪽에 세로선을 추가하면 된다.

지금까지는 상위 그림을 이용해서 얻어 낼 수 있는 정보들이다. 다음 문제들이 잘못된 이유를 생각해 보자.

하나의 수강 신청은 여러명의 학생 정보를 가지고 있다. 이건 잘못된 문제인데 하나의 수강신청 데이터에는 하나의 수강자만 기술되는 것이 맞다.

하나의 수강 신청은 학생 정보가 없어도 된다. 이건 잘못된 문제인데 수강신청하지 않은 학생의 정보를 수강신청 테이블에 넣을 이유가 없다.

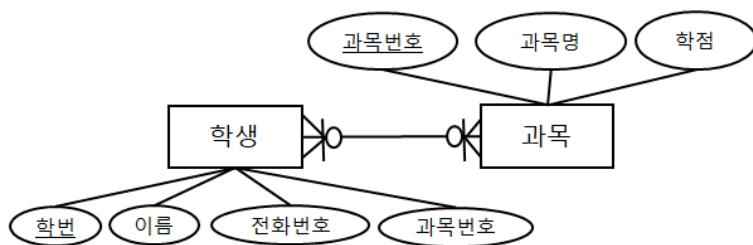
하나의 수강신청은 여러명의 학생정보를 가진다. 이건 잘못된 문제인데 하나의 수강신청은 한 학생의 수강정보를 가지고 있지 여러 명의 학생정보를 가지고 있지 않다.

설계 과정을 확인해보자.

N:N관계로 잘못 설계한 테이블에 데이터를 넣어 보자.

학생 : 학번, 이름, 전화번호, 과목번호

과목 : 과목번호, 과목명, 학점



ERD를 확인해보고 학생, 과목 테이블로 n:n관계로 테이블을 구성해서 만들었을 때의 문제점이 무엇인지 설명해 보자.

```
CREATE TABLE student(
    student_no number,
    name nvarchar2(30),
    phone nvarchar2(30),
    course_no number
```

```

);
CREATE TABLE course (
    course_no number,      name nvarchar2(30),      content nvarchar2(30)
);

```

다음 정보를 넣어보자.

1번학생 1번과목 A, 1번학생 2번과목 B, 1번학생 3번과목 C, 2번학생 1번과목 A  
, 3번학생 신청과목 없음

학생 넣기

```

insert into Student values(1,'홍길동', '01012341234', 1);
insert into Student values(1,'홍길동', '01012341234', 2);
insert into Student values(1,'홍길동', '01012341234', 3);
insert into Student values(2,'홍길준', '01012341234', 4);
insert into Student values(3,'홍길동', '01012341234', null);

```

과목 넣기

```

insert into course values(1,'수학','더하기', 'A');
insert into course values(2,'영어','hello', 'B');
insert into course values(3,'국어','안녕', 'C');
insert into course values(4,'수학','더하기', 'A');

```

과목 테이블에 중복데이터가 쌓인다. 5명이 수학을 들으면 과목 테이블에 중복된 과목 데이터가 5개 생긴다. 잘못된 결과이다.

다양한 다대다 관계로 문제를 해결하려 노력해 보면 역시 다양한 문제가 발생한다는 것을 알 수 있다.

만약 다대다 관계로 문제없이 해결할 수 있는 방법이 있으면 설계해서 데이터를 넣어 문제가 없는지 확인해 보자.

2. 학생, 수강신청, 과목 객체를 가지고 DB를 만든다면 테이블이 3개가 될것이고 정상적인 테이블이 될 것이다. 해당 그림을 이용해서 테이블과 제약조건을 만들고 데이터를 임의로 만들어 넣어 보자.

3개의 테이블중 참조 테이블은 어떤 테이블인가? 참조 상태에 따라 생성, 삭제 순서가 다를 수 있으니 신경써서 만들어 보자.

학생, 과목 테이블이 수강신청에서 참조하는 참조 테이블이다.

각테이블의 pk이와 fk를 확인해 보자.

테이블을 만들고 데이터를 넣는 sql를 작성해 보자.

```
create table student(--학생
    student_id number primary key,
    name varchar(20),
    phone varchar(12)
);
create table course(--교과목
    course_id number primary key,
    title varchar(20),
    content varchar(25)
);
create table register(--수강신청
    student_id number references student(student_id),
    course_id number references course(course_id),
    grade varchar(4),
    primary key(student_id ,course_id )
);
INSERT INTO student  VALUES (1,'홍길동', '01099999999');
INSERT INTO student  VALUES (2,'홍길남', '01088888888');
INSERT INTO student  VALUES (3,'홍길순', '01077777777');

insert into course values(1,'수학','더하기');
insert into course values(2,'영어','hello');
insert into course values(3,'국어','안녕');

insert into register values(1,1,'A');
insert into register values(1,2,'B');
insert into register values(1,3,'C');
insert into register values(2,1,'C');

commit;

select * from student; select * from course; select * from attend;

drop table register; drop table student; drop table course;
```

정리

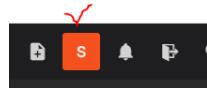
1:1관계와 N:M관계는 잘 생각해 보면 있을 수 없는 상태이다. 1:1관계는 하나의 테이블을  
둘로 나눈 경우여서 굳이 2개의 테이블로 분리할 이유가 없고, N:M관계는 중복된 부분을  
없애야 하기 때문에 3개의 테이블로 분리한 다음 1:N관계로 변경해야 한다.

추가적으로 올바른 테이블간의 관계를 만들기 위해서 정규형을 사용한다. 정규형과 역정규형에 대해서 검색해서 공부해 보자.

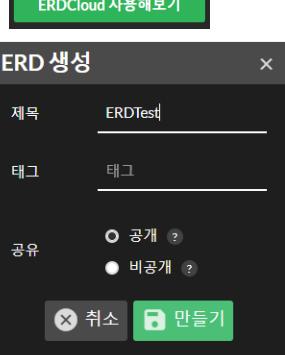
문제1) 현실세계에서 발생할 수 있는 학생,교과목,수강신청 테이블을 찾아 erd,sql를 만들어 보자.

## > 02.ERDCloud

<https://www.erdccloud.com/> 사이트를 이용해서 erd 를 만들어 보자. 로그인이 필요하면 해당 사이트에 로그인을 하자.

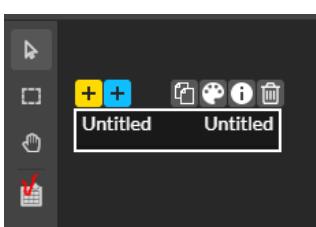


이전에 만든 ERD를 다시 부르고 싶다면 왼쪽 상단 로그인 표시 부분을 선택하면 화면 아래 부분에 이전에 만든 ERD들이 보인다. 다시 부르고 싶은 ERD를 선택하면 된다.



새로 ERD를 만들고 싶다면 화면 중간에 왼쪽 이미지 같은 버튼 부분을 찾아 클릭해서 들어 가면 된다.

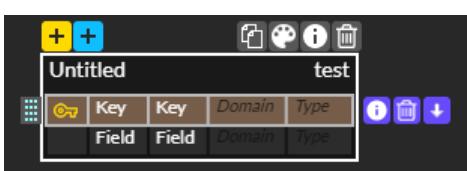
새로 ERD를 만들면 왼쪽 이미지 같은 창이 뜨고 제목 부분에 새로 만들 ERD이름을 입력하고 만들기 버튼을 누르면 새로운 ERD를 만들 수 있는 화면이 생성된다.



왼쪽 이미지 체크를 클릭한 다음 오른쪽 검은화면을 클릭하면 테이블이 하나 만들어 진다.

테이블을 선택하고 노란색+를 누르면 기본키 컬럼이 하나 추가되고 파란색 +를 누르면 일반 컬럼이 하나 추가된다.

i 부분을 누르면 테이블관련 정보를 입력할 수 있다. 논리 이름은 테이블을 쉽게 이해하기 위해서 사용되는 별명이고, 물리이름은 sql를 제작시 사용할 테이블 명이된다. 원하는 값을 입력하고 저장을 누르면 된다. 삭제하고 싶으면 휴지통 모양을 클릭하면 된다.



노란색 + 한번, 파란색 + 한번을 누르면 왼쪽 이미지처럼 기본키 컬럼, 일반 컬럼이 하나씩 추가 된다. 각 컬럼의 내용은 컬럼 오른쪽 아이콘 중 i 를 누르면 변경할 수 있다.

화살표를 누르면 기본키 컬럼은 일반 칼럼, 일반 컬럼은 기본키 컬럼으로 변경된다.



왼쪽 이미지 처럼 두 테이블 간의 관계를 설정하고 싶다면, 테이블을 2개 만든 다음 왼쪽에 있는 메뉴에서 적절한 새발 표기법 이미지를 선택한 다음에 pk테이블을 먼저 선택한 다음 참조 테이블 fk테이블을 선택한다. 두번째 테이블을 선택할 경우 식별관계로 할것인지 비식별 관계로 할 것인지 물어보는 화면이 나오는데 식별관계와 비식별 관계는 다음과 같다.



식별관계 : 부모 테이블(=참조되는 테이블)의 기본키를 자식 테이블(=참조하는 테이블)의 기본키로 이용하는 방법을 말한다.

비식별관계:부모 테이블(=참조되는 테이블)의 기본키를 자식 테이블(=참조하는 테이블)의 외래키로 이용하는 방법을 말한다.

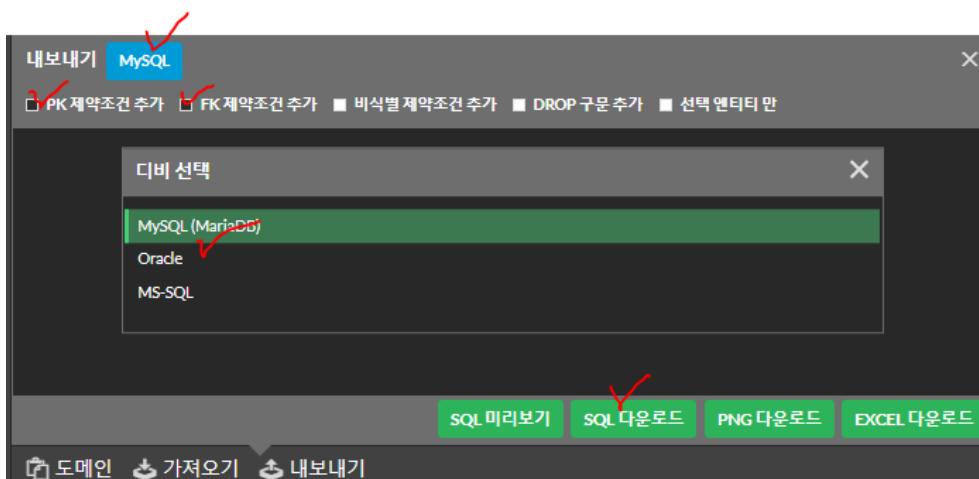
```
CREATE TABLE DEPT(
    DEPTNO NUMBER(2),
    DNAME VARCHAR2(14),
    LOC VARCHAR2(13)
);

CREATE TABLE BONUS(
    ENAME VARCHAR2(10),
    JOB VARCHAR2(9),
    SAL NUMBER,
    COMM NUMBER
);

CREATE TABLE SALGRADE(
    GRADE NUMBER,
    LOSAL NUMBER,
    HISAL NUMBER
);
```

메뉴를 이용해서 직접 ERD를 만드는 것은 쉬운 작업이 아니다. sql로 만든 다음 왼쪽 이미지 처럼 프로그램 왼쪽 하단에 가져오기 버튼을 클릭한다음 만든 sql를 붙여 넣은 다음 가져오기 버튼을 클릭하면 자동으로 그려준다.

만든 ERD를 sql로 다운로드 하고 싶다면 아래 이미지 처럼 오라클을 선택한 다음 원하는 항목을 클릭한 다음 sql다운로드를 선택 하면된다.



직접작성하는데는 어려움이 따르는 가져오기로 작성한 sql를 erd로 그리기를 추천 한다.

---

## > 03. HR 데이터베이스 살펴보기

---

hr테이블관련 DB를 생성해 보자. 책 맨앞부분에 HR테이블관련 정보들이 있다 다시확인하고 다음 쿼리를 작성해보자.

### ▣ 소스코드- employees

1. employees 테이블의 모든 데이터를 출력하시오.

```
select * from employees;
```

2. employees 테이블의 중복되지 않은 Job\_Id컬럼의 값들을 출력하시오[1명 이상의 직원이 있는 Job\_Id 출력]

```
select Job_Id from employees group by Job_Id;
```

3. employees 테이블의 중복되지 않은 dept\_id컬럼의 값을 출력하시오 단 부제목(컬럼)은 "소속부서"로 출력한다.

```
select Job_Id as 소속부서 from employees group by Job_Id;
```

4. employees 테이블에서 이름(First\_Name), 이름(Last\_Name), 급여(Salary)를 출력하시오.

```
select First_Name, Last_Name, Salary from employees;
```

5. employees 테이블에서 이름(First\_Name), 이름(Last\_Name), 급여(Salary)를 급여 오름차 순으로 출력하시오.

```
select First_Name, Last_Name, Salary from employees order by Salary asc;
```

6. employees 테이블에서 이름(First\_Name), 이름(Last\_Name), 급여(Salary), 총급여(Salary\*commission)를

출력하되, First\_Name을 내림차순, 총급여(Salary\*commission)으로 출력하시오.

```
select First_Name, Last_Name, Salary, Salary+(Salary * nvl(commission_pct,0))as total  
from employees order by First_Name desc, total asc ;
```

7. 125\*32+12500의 수식을 계산하시오.

```
select 125*32+12500 from dual;
```

8. employees테이블로 부터 'AD\_PRES'인 Job\_Id 직원이름을 출력하시오.

```
select First_Name, Last_Name from employees where Job_Id = 'AD_PRES';
```

9. employees테이블로 부터 급여가 10000 이상인 직원의 Salary를 출력하되, 보너스를 내림차순으로 출력하시오.

```
select Salary, commission_pct from employees where Salary >=10000 order by commission_pct desc ;
```

10. employees테이블로 테이블에서 Job\_Id가 'IT\_PROG'인 직원의 직원번호,첫이름,이메일을 출력하시오.

```
select employee_Id, First_Name, Email from employees;
```

11. employees 테이블로부터 Job\_Id가 employee\_Id, employee\_Id, Email을 자연어로 출력하되, employee\_ID 순으로 출력하시오.

```
select employee_Id, First_Name, Email from employees order by employee_Id asc;
```

12. employees 테이블로 부터 'D'씨 성의 직원명단을 출력하시오.

```
select * from employees where First_Name like 'D%';
```

13. employees 테이블로 부터 First\_Name 컬럼의 중간 글자가 'n'자인 직원 검색하여 출력하시오.

```
select * from employees where First_Name like '%n%';
```

14. employees 테이블에서 학과 코드(Job\_Id)가 'IT\_PROG', 'FI\_ACCOUNT' Job\_Id에 근무중인 직원의 명단을 Employee\_Id 순으로 출력하시오(in 사용)

```
select * from employees where Job_Id in('IT_PROG','FI_ACCOUNT') order by Employee_Id;
```

15. 14번을 관계연산자와 논리 연산자로 검색하시오.

```
select * from employees where Job_Id = 'IT_PROG' or Job_Id = 'FI_ACCOUNT' order by Employee_Id;
```

16. employees 테이블에서 Salary가 8000부터 10000까지의 범위내 급여를 오름차순으로 출력하시오.(between 사용)

```
select * from employees where Salary between 8000 and 10000 order by Salary;
```

17. 16번을 관계연산자와 논리연산자로 처리하시오.

```
select * from employees where Salary >= 8000 and Salary <= 10000 order by Salary;
```

18. employees 테이블로 부터 Commission\_pct가 널인 행을 검색하여 출력하시오.

```
select * from employees where Commission_pct is null;
```

1. employees 테이블의 'Job\_Id', 'Salary' 행들을 출력하고, 'Salary' 의 평균보다 높은 점수를 출력하시오.(서브쿼리)

```
select Job_Id, Salary from employees where Salary >= (select avg(Salary) from employees ) ;
```

2. employees 테이블의 Salary 최고급여를 받은 직원의 직원번호, 첫이름, 마지막이름, 급여를 출력하시오.

```
select employee_Id, First_Name, Last_Name, Salary from employees where Job_ID = (select Job_ID from Jobs where Max_salary = 40000);
```

3. in연산자를 사용해서 employees 테이블의 Job\_id별  
최고급여를 받은 직원들의 Job\_Id, 급여 출력하시오.

```
select Job_Id, max(salary) from employees where salary in( select max(Salary) from employees group by Job_Id) group by Job_Id;
```

4. any나 some연산자를 사용해서 employees 테이블의  
'IT\_PROG' 'AD\_VP' Job\_id에서 급여가 평균보다 높은 직원에 대하여 직원번호, 첫이름을  
출력하시오.

```
select employee_Id, First_Name, Job_id from employees where Salary > (select avg(Salary) from employees) and Job_id = 'IT_PROG' or Job_id = 'AD_VP';
```

새로운 직원이 추가되었다. employees테이블에 추가하시오, 첫이름은 'aaa'...이다

```
insert into employees values  
(60,'aaa','aaa','aaa','aaa',sysdate,'SH_CLERK',1,0.1,103,60);
```

새로운 직원이 추가되었다. employees테이블에 추가하시오, 첫이름은 'bbb'...이다

```
insert into employees values  
(61,'bbb','bbb','bbb','bbb',sysdate,'SH_CLERK',1,0.1,103,60);
```

직원번호가 '60'인 직원의 Hire\_date 는 2010년 06월 28일이다. 이정보를 employees  
테이블에 입력하시오.

```
update employees set Hire_date = '2010/06/28' where Employee_Id = 60;
```

직원코드가 '62', 직원명이 'ccc' Job\_id가 'SH\_CLERK'...일때 이정보를  
employees테이블에 입력하시오.

```
insert into employees values  
(62,'ccc','ccc','ccc','ccc',sysdate,'SH_CLERK',1,0.1,103,60);
```

employees 테이블의 job\_id 가 'AD\_VP'인 직원들에 대한 재직월 수를 계산하여 출력하시오.

```
select First_Name, trunc((sysdate-hire_date)/365,0)as years from employees  
where job_id = 'AD_VP';
```

employees 테이블의 Salary 10000 이상자에 대하여 일자를 'YYYY/MM/DD'형식의 문자형으로 변환 출력 하시오.

```
select First_Name,to_date(Hire_Date,'YYYY/MM/DD') from employees where Salary  
>=10000;
```

▣ 소스코드- job\_history

-- 1. job\_history 테이블의 모든 데이터를 출력하시오.

```
select * from job_history;
```

-- 2. job\_history 테이블의 employee\_id를 출력하시오.

```
select employee_id from job_history;
```

-- 3. job\_history 테이블의 job\_id를 출력하시오.

```
select job_id from job_history;
```

-- 4. job\_history 테이블에서 job\_id를 기준으로 내림차순으로 출력하시오.

```
select * from job_history order by job_id desc;
```

-- 5. job\_history 테이블에서 department\_id을 기준으로 알파벳순으로 출력하시오.

```
select * from job_history order by department_id;
```

-- 6. job\_history 테이블에 있는 모든 행을 삭제한 후 rollback하시오.

```
delete from job_history;
```

```
Rollback;
```

-- 7. job\_history 테이블에서 employee\_id가 200인 id를 출력하시오.

```
select employee_id from job_history;
```

```
select * from job_history where employee_id = 200;
```

-- 8. job\_history 테이블에 department\_id의 수를 출력하시오.

```
select count(department_id) from job_history;
```

-- 9. job\_history 테이블의 department\_id를 'ID'로 출력하시오.

```

select department_id as ID from job_history;

-- 10. job_history 테이블의 job_id에서 알파벳 M자가 들어간 id를 출력하시오
select * from job_history where job_id like '%M%';

-- 11. employee_id가 150이하인 정보를 출력하시오.
select employee_id from job_history where employee_id <= 150;

-- 12. start_date를 오름차순, end_date를 내림차순으로 출력하세요
select * from job_history order by start_date asc, end_date desc;

-- 13. job_id에서 'CLERK'가 안들어간 사람의 department_id를 출력하시오.
select department_id from job_history where job_id!= '%CLERK%';

-- 14. job_history에서 중복되지 않는 department_id를 출력하시오.
-- 단, 부제목은 "ID"로 출력한다.
select distinct department_id as ID from job_history;
select * from job_history;

```

#### □소스코드-jobs

--Jobs 테이블 시나리오--

1.최대 급여가 8000 이상인 직업의 id와 title을 출력하시오.

```
select job_id, job_title from jobs where max_salary >=8000;
```

2.최소 급여가 3000 이하인 직업의 id와 title을 출력하시오.

```
select job_id, job_title from jobs where min_salary <=3000;
```

3.job\_id가 ST\_CLERK 인 직업의 최소, 최대 급여를 출력하시오.

```
select min_salary, max_salary from jobs where job_id like 'ST_CLERK';
```

4.Purchasing 부서 직군의 가장 급여를 적게받는 사람의 평균급여를 구하시오.

```
select avg(min_salary) from jobs where job_id like 'PU%';
```

5.Administration 부서 직무를 수행하는 모든 job의 명칭을 출력하시오.

```
select job_title from jobs where job_id like 'AD%';
```

6.각부서의 매니저 역할을 담당하는 직업 이름을 출력하시오.

```
select job_title from jobs where job_id like '%MAN%';
```

7.각 직업군의 최대급여가 큰 순서로 직업이름과 급여를 출력하시오.

```
select job_title, max_salary from jobs order by max_salary desc;
```

8. 최대 받을 수 있는 급여가 30000 이상인 직군의 job\_id, job\_title, 이름과 급여를 출력하시오.

```
select j.job_id, j.job_title, e.first_name, e.salary from employees e, jobs j
where (e.job_id = j.job_id) and
j.max_salary>=30000;
```

9. Shelli가 자신의 직무를 유지하면서 받을 수 있는 최대 급여는 얼마인지 출력하시오.

```
select j.job_id, j.job_title, e.first_name, j.max_salary from employees e, jobs j
where e.job_id = j.job_id and e.first_name like 'Shelli';
```

10. Jobs 테이블에 IT\_DBA, database administrator, 최저급여 10000, 최대급여 30000인 행을 추가하시오.

```
insert into jobs values ('IT_DBA', 'Database Administrator', '10000', '30000');
```

11. IT\_DBA의 최저 급여를 15000으로 수정하시오.

```
update jobs set min_salary = '15000' where job_id like 'IT_DBA';
```

12. jobs테이블에서 규정하는 직군의 최대 급여를 받고있는 사원의 job\_id, job\_title, 이름, 급여를 출력하시오.

```
select e.job_id, j.job_title, e.first_name, e.salary from employees e, jobs j
where e.job_id = j.job_id and e.salary = j.max_salary;
```

13. 프로그래머의 최대 급여보다 더 높은 금액을 받는 직원들의 job\_id, job\_title, 이름, 급여를 급여가 높은순으로 출력하시오.

```
select j.job_id, j.job_title, e.first_name, e.salary from jobs j, employees e
where j.job_id = e.job_id and e.salary > all (select j.max_salary from jobs j
where j.job_title like 'Programmer')
order by e.salary desc;
```

14. job\_id 가 AD\_PRES 인 직원보다 hiredate가 더 오래된 직원들의 employee\_id, job\_id, 이름, hiredate를 출력하시오.

```
select employee_id, job_id, first_name, hire_date from employees where
substr(hire_date,1,5) <= (select substr(hire_date,1,5) from employees where
job_id like 'AD_PRES');
```

□소스코드(locations)

LOCATIONS 시나리오

--.locations 테이블의 모든 데이터 출력

```

--select * from locations;

--LOCATION_ID 가 1000인 street_address 출력하기
select street_address from LOCATIONS where LOCATION_ID=1000;

--locations 테이블에서 location_id street_address 를 출력한다.
-- select location_id, street_address from locations;

--locations 테이블의 중복되지 않는 country_id값들을 출력하시오
    단 부제목은 "지역"로 출력한다.
--select distinct country_id as 지역 from locations;

--locations 테이블에서 location_id, city 를 오름차 순으로 출력
--select location_id, city from locations order by location_id;

--CITY가 London인 LOCATION_ID, street_address, city 출력 하시오
select LOCATION_ID,street_address,city from LOCATIONS where CITY='London';

--locations 테이블로부터 country_id가 it인 street_address를 출력
select street_address from locations where country_id='IT';

--locations 테이블로부터 location_id가 2000이상 이상인
    데이터를 출력하되 오름차순으로 출력
--select * from locations where location_id>2000 order by location_id;

--locations 테이블로부터 city가 Bombay인 데이터의 location_id와 city를 출력
select location_id, city from locations where city='Bombay';

--country_id가 US 이고, city=Southlake 인 모든 데이터 출력
select * from locations where country_id='US' and city='Southlake';

--city에 R이 들어가는 location_id, city 출력하시오.
select location_id,city from locations where city like '%R%';

--locations 테이블로 부터 city가 'S'로 시작하는 모든 데이터를 출력
select * from locations where city like 'S%';

--city에 첫 글자가 B로 시작하는 street_address, postal_code, city 출력하기
select street_address,postal_code,city from locations where city like 'B%';

```

```
--locations 테이블로 부터 street_address컬럼의 중간 글자가  
'ch'가 들어가는 데이터를 출력하시오  
select * from locations where street_address like '%ch%';
```

```
--locations 테이블에서 location_id 1500 부터 2000까지의  
범위 내 데이터를 내림차 출력하시오  
select * from locations where location_id between 1500 and 2000 order by  
location_id;
```

```
--location_id가 평균(location_id) 보다 높은 location_id 출력하기  
select location_id from locations where location_id >  
(select avg(location_id) from locations);
```

```
--locations 테이블의 location_id가 최대값인 모든 데이터 출력하기  
select * from locations where location_id =  
(select max(location_id) from locations );
```

```
--location_id가 평균location_id보다 이상인 최대값과 최소값 을 출력하기  
select MAX(location_id),min(location_id) from locations where location_id >  
(select avg(location_id) from locations);  
--all를 이용하여 locations 테이블의 location_id가 제일 높은 컬럼의  
모든 데이터를 출력하기  
select * from locations where location_id=all  
(select max(location_id) from locations);
```

```
-- locations, countries 를 조인하여 합친 모든 데이터를 출력하시오  
select * from locations a,countries b where a.country_id=b.country_id;
```

```
-- locations, countries 를 조인하여 둘다 중복되지 않는 country_id, country_name  
출력하시오  
select distinct a.country_ID,b.country_name from locations a,countries b;  
  
--  
--locations, countries 를 조인하여 region_id=4인 중복되지않는 city를 출력하시오  
select distinct a.city from locations a,countries b where b.region_id=4;
```

```
--locations 테이블을 이용하여 country_id그룹별 평균location_id값을 구하시오  
select avg(location_id),country_id from locations group by country_id;  
□소스코드-countries
```

## 1.SELECT

- countries 테이블의 모든 데이터를 출력하시오.  
- select \* from countries;

2. countries 테이블의 중복되지 않은 country\_id컬럼의 값을 출력하시오
  - select distinct country\_id from countries;
3. countries 테이블의 중복되지 않은 country\_name컬럼의 값을 출력하시오. 단부제목은 "국가명"으로 출력한다.
  - select distinct country\_name as 국가명 from countries;
4. countries 테이블에서 country\_id, country\_name을 출력하시오
  - select country\_id, country\_name from countries;
5. countries 테이블에서 country\_id, country\_name을 country\_name오름차순으로 출력하시오.
  - select country\_id, country\_name from countries order by country\_name asc;
6. countries 테이블에서 country\_id, country\_name, region\_id를 출력하되 country\_name을 오름차순, region\_id를 내림차순으로 정렬하여 출력하시오.
  - select country\_id, country\_name, region\_id from countries order by country\_name asc, region\_id desc;
7. 125\*32+12500의 수식을 계산하시오.
  - select 125\*32+12500 from dual;
8. countries 테이블에서 country\_name이 'Canada'인 나라의 country\_id를 출력하시오.
  - select country\_id from countries where country\_name='Canada';
9. countries 테이블로부터 region\_id가 1보다 큰 country\_id를 출력하되, country\_name 을 내림차순으로 출력하시오.
  - select country\_id from countries where region\_id>1 order by country\_name desc;
10. countries 테이블에서 region\_id가 '2'인 국가의 country\_id와 country\_name을 출력하시오.
  - select country\_id, country\_name from countries where region\_id=2;
11. countries 테이블로부터 country\_name에 'U'가 들어간 국가명단을 출력하시오.
  - select \* from countries where country\_name like '%U%';
12. countries 테이블로부터 country\_name 중간글자에 'e'가 들어간 국가명단을 출력하시오.
  - select \* from countries where country\_name like '%e%';
13. countries 테이블로부터 region\_id가 NULL인 자료를 출력하시오.
  - select \* from countries where region\_id is null;

## 2.databaseupdate

1. 새로운 국가가 추가되었다. 테이블에 추가하시오. 국가코드는 'K0'이고, 국가명은 'Korea' region\_id는 1이다.
  - insert into countries values ('K0','Korea',1);
2. 새로운 국가가 추가되었다. 테이블에 추가하시오. 국가코드는 'JP'이고, 국가명은 'Japan' region\_id는 2이다.
  - insert into countries values ('JP','Japan',2);

3. 국가코드 'KO'의 region\_id를 3으로 수정하시오.

```
- update countries set region_id=3 where country_id='KO';
```

### 3.function

1. countries테이블의 country\_id와 country\_name칼럼의 문자열 길이와 바이트 단위로 계산하여 출력하시오.

```
- select length(country_id),length(country_name) from countries;
```

2. countries테이블의 region\_id에 '00-'을 추가하여 출력하시오.

```
- select lpad(region_id, 5, '00-') from countries;
```

3. countries테이블의 country\_id, country\_name칼럼을 결합하여 출력하시오.

```
- select country_id, country_name, concat(country_id, country_name)  
from countries;
```

4. countries테이블에서 country\_name이 NULL인 행을 출력하시오. 단, NULL일 경우 0으로 변환하여 출력

```
- select country_id,nvl(country_name,0),region_id from countries  
where country_name is null;
```

### 4.groupfunction

1. countries테이블을 이용하여 국가 수를 출력하시오.

```
- select distinct count(country_name) from countries;
```

2. countries테이블의 행의 수를 반환하시오.

```
- select count(*) from countries;
```

#### ▣ 소스코드-regions

--regions 테이블의 모든 데이터 출력

--region\_ID 가 1인 region\_name 출력

--region name에 A가 포함된 데이터 출력

--region\_ID가 2이상인 데이터 출력

--region\_ID가 2이상이고, region\_name에 m이 들어가는 데이터 출력

--in을 이용하여 region\_id가 3,4인 데이터 출력

-- countries 테이블과 조인하여 데이터를 출력하시오

#### ▣ 소스코드-departments

테이블의 모든 데이터를 출력하시오.

테이블의 중복되지 않은 location\_id컬럼의 값들을 출력하시오

departments\_id, departments\_name을 출력하시오

departments\_id, departments\_name을 departments\_name오름차순으로 출력하시오.

country\_name이 'sales'인 부서의 departments\_id를 출력하시오.

departments\_id가 100보다 큰 departments\_id를 출력하되, departments\_name 을 내림차순으로 출력하시오.

departments\_name에 'e'가 들어간 데이터를 출력하시오.

`manager_id`가 `NULL`인 자료를 출력하시오.

`departments_name` 에 '`con`'이 들어가는 데이터의 `location_id`를 `1000`으로 수정하시오.

`departments_id`, `departments_name`칼럼을 결합하여 출력하시오.

`departments_id`를 이용하여 부서의 수를 출력하시오.

`employees` 테이블을 조인하여 `manager`의 `first_name`을 출력하시오

`locations`테이블을 조인하여 우편번호가 `98199`인 부서정보를 출력하시오

## > 04. Northwind 데이터베이스 살펴보기

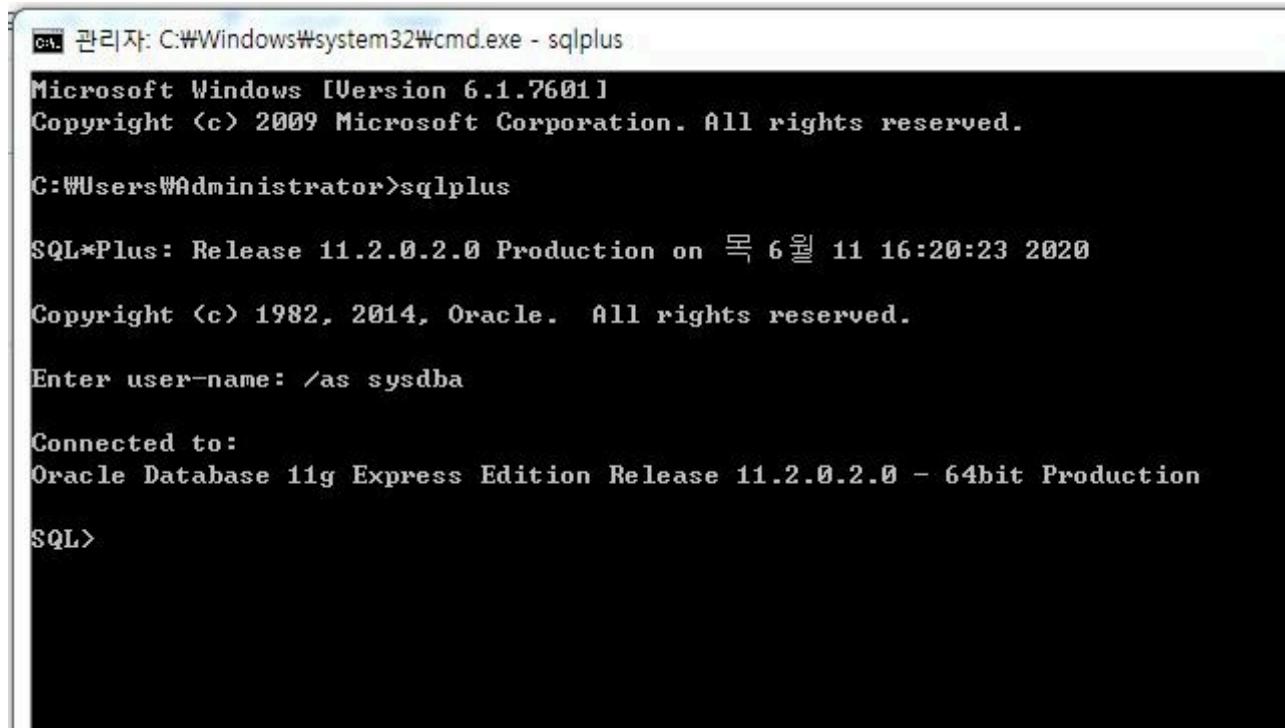
northwind oracle database download로 검색해서 다음 사이트를 찾았다.

<https://binaryworld.net/blogs/northwind-database-creation-script-for-sql-server/>

사이트에서 다음 파일들을 다운로드 받아 northwind를 설치해 보자.

Oracle\_Northwind\_1\_CreateObjects.sql, Oracle\_Northwind\_DropTables.sql,  
Oracle\_Northwind\_2\_InsertData.sql

### 1. 계정 생성



```
관리자: C:\Windows\system32\cmd.exe - sqlplus
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>sqlplus

SQL*Plus: Release 11.2.0.2.0 Production on 목 6월 11 16:20:23 2020

Copyright <c> 1982, 2014, Oracle. All rights reserved.

Enter user-name: /as sysdba

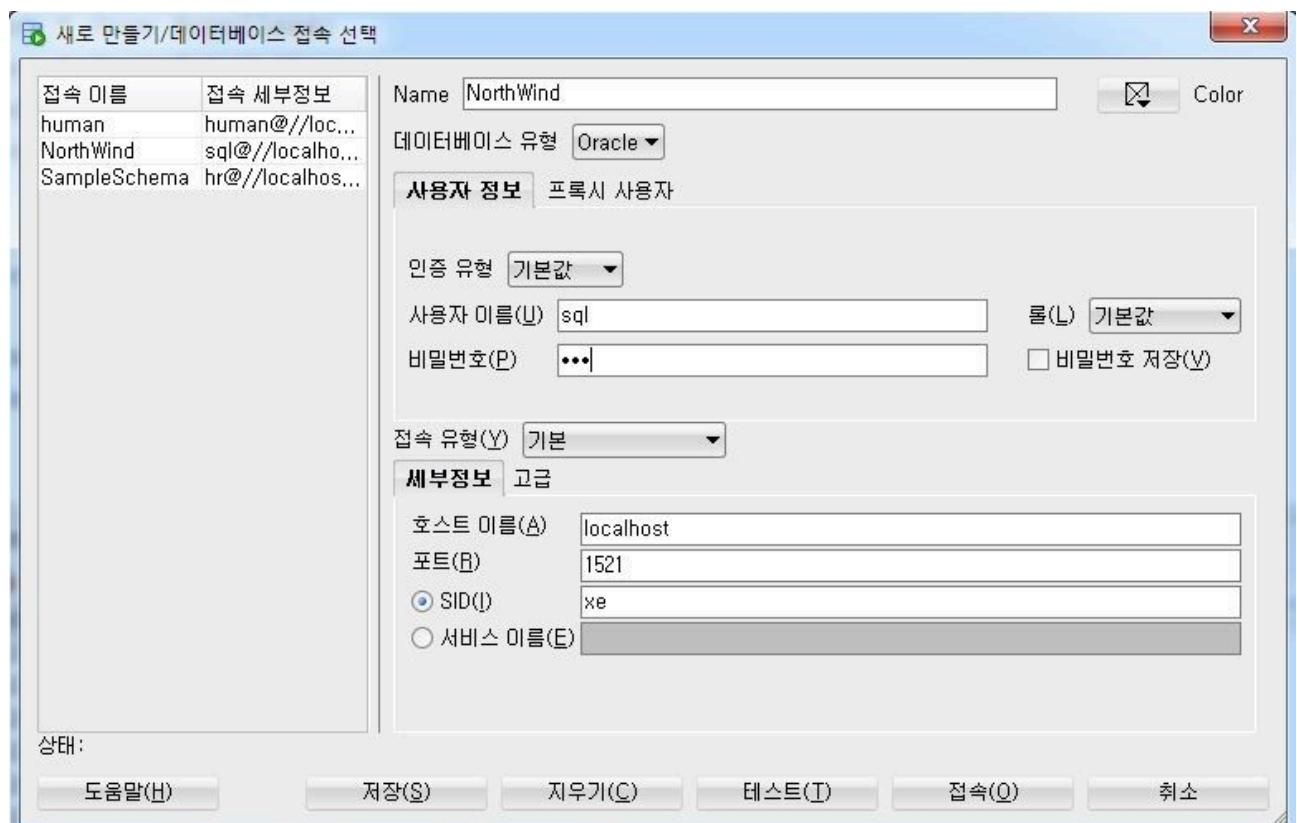
Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

SQL>
```

sql> 에

```
create user sql identified by sql
grant connect,resource,dba to sql
```

### 2. sqldeveloper 접속 만들기



### 3.첨부된 sql 파일 받아 c드라이브에 넣기

Program Files	2020-06-11 오후...	파일 폴더
Program Files (x86)	2020-04-16 오후...	파일 폴더
ProgramData	2020-04-16 오후...	파일 폴더
Python	2020-05-28 오전...	파일 폴더
SQL	2020-06-11 오후...	파일 폴더
sqldeveloper	2020-06-11 오후...	파일 폴더
sts-bundle	2020-06-11 오후...	파일 폴더
TEMP	2020-06-11 오후...	파일 폴더
Windows	2020-05-22 오전...	파일 폴더
사용자	2020-01-28 오전...	파일 폴더
AMTAG.BIN	2019-11-28 오후...	BIN 파일 1KB
msdia80.dll	2006-12-01 오후...	응용 프로그램 확장 884KB

이름	수정한 날짜	유형	크기
ec.sql	2020-04-20 오후...	SQL-Script	3KB
ec_data.sql	2020-04-20 오후...	SQL-Script	7KB
excercise.sql	2020-05-21 오후...	SQL-Script	37KB
haksa_data.sql	2020-05-19 오후...	SQL-Script	8KB
human.sql	2020-05-25 오후...	SQL-Script	22KB
NorthWind.sql	2020-06-11 오후...	SQL-Script	1KB
Oracle_Northwind_1_CreateObjects.sql	2020-05-25 오후...	SQL-Script	6KB
Oracle_Northwind_2_InsertData.sql	2012-07-17 오전...	SQL-Script	692KB
Oracle_Northwind_DropTables.sql	2012-07-17 오전...	SQL-Script	1KB
SampleSchema.sql	2020-06-08 오후...	SQL-Script	1KB
scott.sql	2020-04-17 오후...	SQL-Script	4KB

#### 4. DB import

NorthWind.sql

SQL 워크시트(W) 내역

워크시트 질의 작성기

```
--select * from Employees;
--select * from Customers;
--select * from Shippers;
--select * from Products;
--select * from Suppliers;
--select * from Categories;
--select * from Territories;
--select * from Region;
--select * from tab;

@c:\SQL\Oracle_Northwind_1_CreateObjects.sql;
@c:\SQL\Oracle_Northwind_2_InsertData.sql;
```

컨트롤 + enter

다음은 Northwind 쇼핑몰 관련 ERD와 테이블들이다.

Table Customers		고객정보를 저장하는 테이블			
NO	Column Name	Data Type	Key	Null Option	Comments
1	CustomerID	varchar	PK	N	회사명(CompanyName)의 첫 5글자
2	CompanyName	varchar		N	회사명
3	ContactName	varchar		Y	고객명
4	ContactTitle	varchar		Y	고객직위
5	Address	varchar		Y	고객주소
6	City	varchar		Y	도시명
7	Region	varchar		Y	지역명
8	PostalCode	varchar		Y	우편번호
9	Country	varchar		Y	나라
10	Phone	varchar		Y	고객 전화번호
11	Fax	varchar		Y	고객 팩스번호

Table Employees		직원정보 테이블 / Order 테이블에서 직원ID를 통해 판매기록을 알 수 있음			
NO	Column Name	Data Type	Key	Null Option	Comments
1	EmployeeID	int		N	직원 ID (시퀀스)
2	LastName	varchar		N	성
3	FirstName	varchar		N	이름
4	Title	varchar		Y	직위
5	TitleOfCourtesy	varchar		Y	영어호칭(Mr, Ms, Mrs ....)
6	BirthDate	date		Y	생일
7	HireDate	date		Y	고용일
8	Address	varchar		Y	주소
9	City	varchar		Y	도시명
10	Region	varchar		Y	지역명
11	PostalCode	varchar		Y	우편번호
12	Country	varchar		Y	나라
13	HomePhone	varchar		Y	집전화
14	Extension	varchar		Y	내선번호
15	Photo	blob		Y	사진
16	Notes	varchar		N	직원 정보 기록
17	ReportsTo	int		Y	매니저 ID (self join으로 매니저 찾기)
18	PhotoPath	varchar		Y	
19	Salary	int		Y	급여

Table Region		Territories 테이블에서 FK로 사용 (종류 동부, 서부, 남부, 북부 4가지)				
NO	Column Name	Data Type	Key	Null Option	Comments	
1	RegionID	int	PK	N	지역ID (시퀀스)	
2	RegionDescription	varchar		N	지역정보	
Table Territories						
NO	Column Name	Data Type	Key	Null Option	Comments	
1	TerritoryID	varchar	PK	N	ZIP code(=postal code) 우편번호	
2	TerritoryDescription	varchar		N	지역이름	
3	RegionID	int	FK	N	지역ID	
Table EmployeeTerritories	territories 테이블과 함께 직원 별 담당지역정보를 알 수 있음. 지역 4개 미포함					
NO	Column Name	Data Type	Key	Null Option	Comments	
1	EmployeeID	int	PK,FK	N	직원 ID	
2	TerritoryID	varchar	PK,FK	N	ZIP code(=postal code) 우편번호	

Table Categories		Products 테이블과 함께 해당 제품의 카테고리 확인 가능				
NO	Column Name	Data Type	Key	Null Option	Comments	
1	CategoryID	int	PK	N	카테고리 ID(시퀀스)	
2	CategoryName	varchar		N	카테고리명	
3	Description	text		Y	카테고리 정보	
4	Picture	blob		Y	사진	

Table Suppliers		공급업체 정보 테이블				
NO	Column Name	Data Type	Key	Null Option	Comments	
1	SupplierID	int	PK	N	공급업체 ID(시퀀스)	
2	CompanyName	varchar		N	회사명	
3	ContactName	varchar		Y	이름	
4	ContactTitle	varchar		Y	직위	
5	Address	varchar		Y	주소	
6	City	varchar		Y	도시명	
7	Region	varchar		Y	지역	
8	PostalCode	varchar		Y	우편번호	
9	Country	varchar		Y	나라	
10	Phone	varchar		Y	연락처	
11	Fax	varchar		Y	팩스번호	
12	HomePage	text		Y	홈페이지주소	

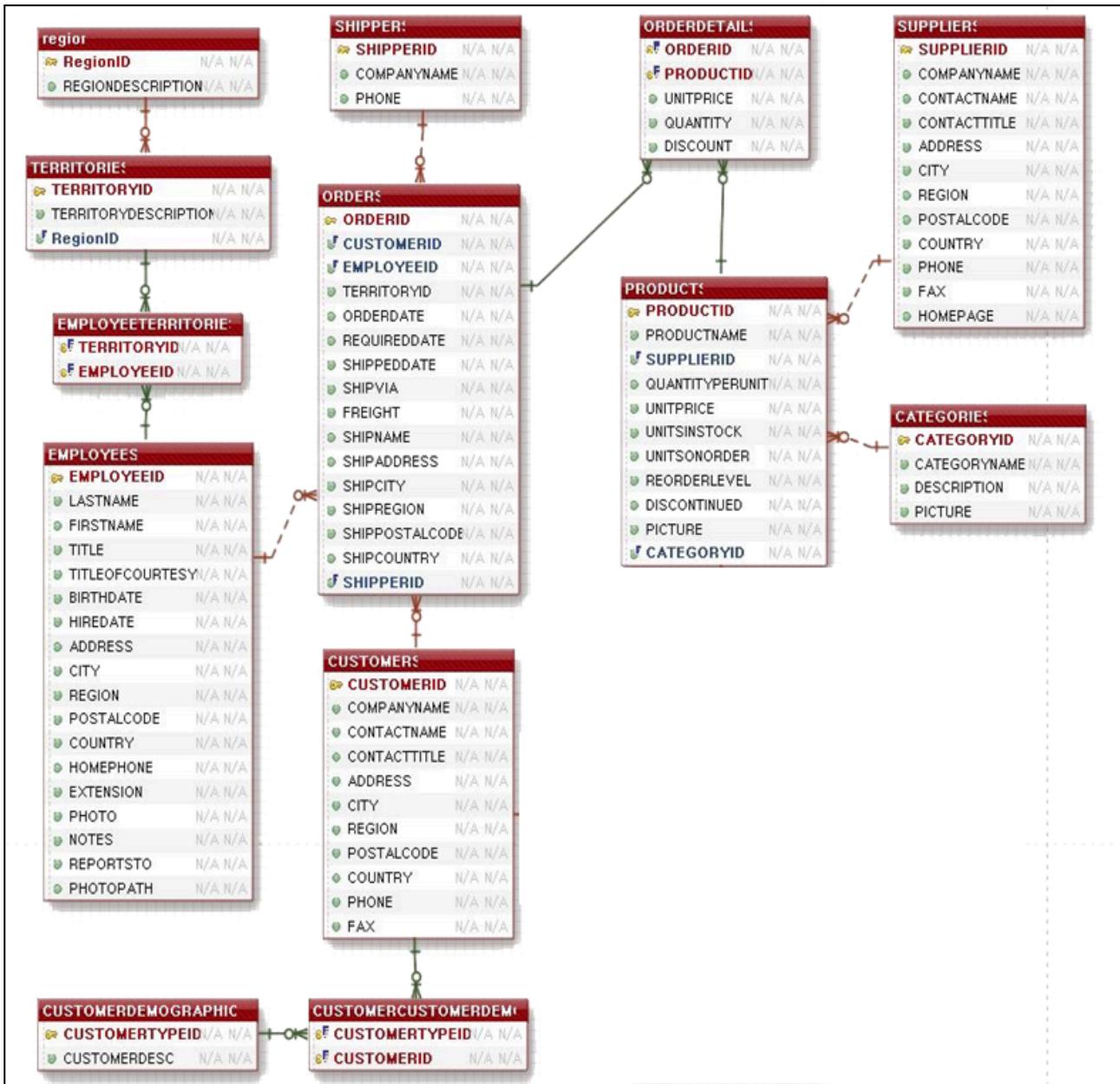
Table Products		제품정보 테이블				
NO	Column Name	Data Type	Key	Null Option		Comments
1	ProductID	int	PK	N	제품ID(시퀀스)	
2	ProductName	varchar		N	제품명	
3	SupplierID	int	FK	N	공급업체 ID	
4	CategoryID	int	FK	N	카테고리 ID	
5	QuantityPerUnit	varchar		Y	단위당 수량	
6	UnitPrice	decimal		N Default 0	단가	
7	UnitsInStock	int		N Default 0	재고	
8	UnitsOnOrder	int		N Default 0	주문단위	
9	ReorderLevel	int		N Default 0		
10	Discontinued	boolean		N Default False	할인여부	

Table Shippers		화물발송 (송하인)관련 테이블				
NO	Column Name	Data Type	Key	Null Option		Comments
1	ShipperID	int	PK	N	화물발송자 ID(시퀀스)	
2	CompanyName	varchar		N	회사명	
3	Phone	varchar		Y	연락처	

Table Orders		주문정보(고객 ID, 배송일자, 배송지정보) 테이블				
NO	Column Name	Data Type	Key	Null Option		Comments
1	OrderID	int	PK	N	주문 ID (시퀀스)	
2	CustomerID	varchar	FK	Y	고객 ID (customers 테이블)	
3	EmployeeID	int	FK	N	담당직원 ID	
4	OrderDate	date		Y	주문일자	
5	RequiredDate	date		Y	요청일자	
6	ShippedDate	date		Y	배송일자	
7	ShipVia	int	FK	Y	발송회사ID(shippers 의 ShipperID)	
8	Freight	decimal		N Default 0	운임요금(?)	
9	ShipName	varchar		Y	배송받는 회사명(Customer 회사명)	
10	ShipAddress	varchar		Y	배송지 주소	
11	ShipCity	varchar		Y	배송지 도시명	
12	ShipRegion	varchar		Y	배송지 지역명	
13	ShipPostalCode	varchar		Y	배송지 우편번호	
14	ShipCountry	varchar		Y	배송지 나라	

Table	Order Details	주문정보(단가, 주문량, 할인여부) 테이블				
NO	Column Name	Data Type	Key	Null Option	Comments	
1	OrderID	int	PK,FK	N	주문 ID	
2	ProductID	int	PK,FK	N	제품 ID	
3	UnitPrice	decimal		N Default 999999.99	단가	
4	Quantity	int		N Default 1	주문량	
5	Discount	double		N Default 0	할인여부	

Table	CustomerDemographics				
NO	Column Name	Data Type	Key	Null Option	Comments
1	CustomerTypeID	varchar	PK	N	
2	CustomerDesc	text		Y	
Table	CustomerCustomerDemo				
NO	Column Name	Data Type	Key	Null Option	Comments
1	CustomerID	varchar	PK	N	
2	CustomerTypeID	varchar	PK	N	



## □ 시나리오 -region

- Region테이블의 모든 데이터를 출력하시오.  
A. `select * from Region;`
- Region테이블의 RegionID를 출력하시오.  
A. `select RegionID from Region;`
- Region테이블의 RegionDescription을 출력하시오.  
A. `select RegionDescription from Region;`
- Region테이블에서 RegionID를 기준으로 내림차순으로 출력하시오.  
A. `select * from Region order by RegionID desc;`
- Region테이블에서 RegionDescription을 기준으로 알파벳순으로 출력하시오.  
A. `select * from Region order by RegionDescription;`
- Region테이블에 있는 모든 행을 삭제한 후 rollback하시오.  
A. `delete from Region; Rollback;`

7. RegionID가 2인 RegionDescription을 출력하시오.  
A. `select * from Region where RegionID=2;`
8. RegionDescription이 'Southern'인 지역을 출력하시오.  
A. `select * from Region where RegionDescription='Southern';`
9. Region 테이블의 지역 수를 출력하시오.  
A. `select count(RegionID) from Region;`
10. Region테이블에서 RegionID 번호가 가장 높은 데이터와 가장 낮은 데이터를 출력하시오.  
A. `select max(RegionID), min(RegionID) from Region;`
11. Region테이블에서 RegionID 평균 값과 모두 더한 값을 출력하시오.  
A. `select avg(RegionID), sum(RegionID) from Region;`
12. Region테이블의 RegionID를 'ID'로 출력하시오.  
A. `select RegionID as ID from Region;`
13. Region테이블의 RegionDescription에서 'E'가 들어간 정보를 출력하시오.  
A. `select * from Region where RegionDescription like '%E%';`

□ 시나리오 -territories

1. Territories 의 모든 데이터를 출력하시오.
2. Territories 의 중복되지 않은 TerritoryDescription 출력
3. Territories 테이블에서 중복되지 않은 TerritoryDescription 출력 컬럼명은 영토기술
4. Territories 테이블에서 regionID를 오름차순으로 출력하시오.
5. Territories 테이블에서 (regionID + 3)을 하고 오름차순으로 출력하시오.
6. Territories 테이블에서 TerritoryID를 출력하시오.
7. Territories 테이블에서 regionID가 2이하를 출력하시오
8. Territories 테이블에서 regionID가 1인 곳을 출력하시오.
9. Territories 테이블에서 TerritoryDescription에 s가 들어가는 정보를 출력하시오.
10. Territories 테이블에서 TerritoryID를 오름차순으로 출력하시오.
11. Territories 테이블에서 TerritoryID가 80000~90000사이인 정보를 출력하시오.
12. Territories 테이블과 employees 테이블을 이용하여 소속된 city를 출력하시오.
13. Territories 테이블과 employees 테이블을 이용하여 New York에 살지 않는 사람들을 출력하시오.
14. Territories 테이블과 동일한 T\_Territories 빈 테이블을 생성하시오.
15. Territories 테이블에 TerritoryID= 98501 TerritoryDescription='Seoul' regionID=7인 정보 추가.
16. Territories 테이블에 TerritoryID= 98801 TerritoryDescription='Busan' regionID=7인 정보 추가.
17. Territories 테이블과 employees 테이블을 이용하여 hireDate를 출력하시오.
18. Territories 테이블에 TerritoryID = 98802 TerritoryDescription='Busan' regionID=7인 정보 추가.
19. Territories 테이블에 TerritoryID = 98803 TerritoryDescription='Busan' regionID=7인 정보 추가.

20. Territories 테이블에 RegionID를 '지역번호'로 바꾸시오
21. Territories 테이블의 TerritoryDescription를 '지역명'으로 바꾸시오.
22. Territories 테이블에 TerritoryID가 '98801' 인 ID의 TerritoryDescription을 "New York"으로 변경하시오.
23. Territories 테이블의 모든 행을 삭제하시오. rollback 하시오.
24. Territories 테이블에 RegionID가 2인 곳을 삭제하시오.
25. Territories 테이블에서 TerritoryDescription가 'New York' 인 곳을 삭제하시오
  
26. Territories 테이블의 TerritoryID의 문자열 길이와 바이트 단위로 계산하여 출력하시오.
27. Territories 테이블의 RegionID에 '01-'를 추가하여 출력하시오.
28. Territories 테이블의 RegionID 와 TerritoryDescription을 결합하여 출력하시오.
29. Territories 테이블에 TerritoryDescription의 'New York' 이 'blueMoon'으로 변경되었다. 변경하여 출력하오.
30. Territories 테이블의 RegionID 가 north(1~2) south(2~4)로 반환하여 출력하시오.
31. Territories 테이블의 RegionID 가 north(1~2) south(2~4)로 반환하여 출력하시오. (함수를 만들어서)
32. Territories 테이블을 이용해서 TerritoryID가 몇 개인지 출력
33. Territories 테이블을 이용해서 TerritoryDescription 한 지역 인 곳 출력
34. Territories 테이블과 employees 테이블을 조인하시오.
35. Territories 테이블의 RegionID 와 employees 테이블의 Address를 이용하여 RegionID가 2인 Address를 출력하시오.
36. Territories 테이블과 employees 테이블을 이용하여 RegionID가 1인 사람들의 HomePhone 출력.
37. Territories 테이블과 employees 테이블을 이용하여 TerritoryDescription이 'Seattle' 인 사람의 hireDate 출력.
38. Territories 테이블과 employess 테이블을 이용하여 TerritoryDescription이 'New York' 인 사람의 BirthDate 출력
39. Territories 테이블과 employess 테이블을 이용하여 RegionID가 2인 사람들의 LastName 출력
40. insert into Territories values('England','AKMU',100);
41. Territories 테이블과 employess 테이블을 이용하여 FirstName, LastName, RegionID 출력
42. 12.Territories 테이블과 T\_Terriories 테이블을 합하여 RegionID 순으로 출력하시오.

□ 시나리오 -employeeterritories

1. EmployeeTerritories 테이블의 모든 데이터 출력.
2. EmployeeTerritories 테이블의 중복되지 않은 TerritoryID컬럼의 값들을 출력.
3. EmployeeTerritories 테이블의 중복되지 않은 TerritoryID컬럼의 값들을 부제목(컬럼) '소속지역'으로 출력.

4. EmployeeID 를 출력
5. EmployeeID, TerritoryID 를 EmployeeID 오름차순으로 출력
6. EmployeeID는 내림차순, TerritoryID 를 오름차순으로 출력
7. EmployeeID가 '5' 인 TerritoryID를 출력.
8. TerritoryID가 30000 이상인 TerritoryID를 내림차순으로 모두 출력
9. EmployeeID가 '7' 인 TerritoryID를 출력.
10. EmployeeID 순으로 모든 정보를 출력.
11. TerritoryID에서 9로 시작하는 값을 TerritoryID만 출력.
12. TerritoryID에 '02' 데이터가 들어간 TerritoryID데이터를 출력
13. EmployeeID의 2와 9값을 EmployeeID순으로 출력
14. 14번을 관계연산자와 논리 연산자로 검색하시오.
15. TerritoryID의 35000부터 60000까지의 범위를 EmployeeID 순으로 출력
16. 16번을 관계연산자와 논리연산자로 처리하시오.
17. EmployeeID의 평균 값보다 높은 EmployeeID와 TerritoryID를 출력.(서브쿼리)
18. TerritoryID 최대값의 EmployeeID, TerritoryID 출력.(서브쿼리)
19. EmployeeID 값별로 TerritoryID의 최대값을 EmployeeID순으로 출력.(in연산자)
20. EmployeeID 값 7의 TerritoryID 최소값보다 높은 값 출력.(any나 some)
21. EmployeeID 값 1의 TerritoryID 최대값보다 낮은 값 출력.(all 사용)
22. EmployeeTerritories 테이블에서 EmployeeID 값이 1인 것을 Employee02 테이블을 생성하고 저장
23. EmployeeTerritories 테이블과 동일한 T\_EmployeeTerritories 빈 테이블 생성.
24. EmployeeID에 10, TerritoryID는 '25360'를 추가.
25. EmployeeID에 11, TerritoryID는 '99502' 추가.
26. EmployeeID에 11, TerritoryID는 '01234' 추가.
27. EmployeeID가 12일때 TerritoryID는 '78978' 추가.
28. EmployeeID가 5일때 TerritoryID는 '55555' 추가.
29. EmployeeID는 1, TerritoryID는 '11111' 추가.
30. EmployeeID는 2, TerritoryID는 '22222' 추가.
31. EmployeeID는 3, TerritoryID는 '33333' 추가.
32. EmployeeID는 4, TerritoryID는 '44444' 추가.
33. EmployeeID는 6, TerritoryID는 '66666' 추가.
34. EmployeeID는 7, TerritoryID는 '77777' 추가.
35. TerritoryID 컬럼명을 TerritoryNO 로 수정.
36. TerritoryID의 '19713'을 "일구칠일삼"으로 변경.
37. EmployeeID 6의 TerritoryID 값'65656'을 '66666'으로 변경.
38. EmployeeTerritories 테이블의 EmployeeID가 EmployeeID 평균값보다 높으면EmployeeID의 값에 1을 뺀 값을 변경
39. EmployeeTerritories 테이블의 모든 행을 삭제.
40. EmployeeID는 4이고 TerritoryID는 '44444' 인 행을 삭제.
41. EmployeeID가 9인 행을 삭제.
42. EmployeeID의 문자열 길이와 TerritoryID 바이트 단위로 계산하여 출력

43. TerritoryID의 앞 2자리만 추출하여 출력하고, TerritoryID에 1이 몇번째에 시작하는지 출력 (instr, substr 사용)
44. TerritoryID에서 2이나 9를 제거.
45. TerritoryID에서 5 제거.
46. TerritoryID 앞에 '053-'를 추가하여 출력
47. EmployeeID, TerritoryID를 결합하여 출력
48. EmployeeID의 값1을 101로 변경하여 출력
49. TerritoryID에서 '2'를 '구'로 '9'를 '이'로 변경하여 출력.(translate 사용)
50. TerritoryID에서 10을 ()로 변경하여 출력(replace 사용)
51. EmployeeID의 평균값을 Round, trunc, ceil, floor 함수를 이용하여 출력
52. TerritoryID를 3으로 나눈 나머지를 출력
53. EmployeeID의 5이상인 값을 'L999.00' 형식으로 출력
54. TerritoryID를 문자로 변환하여 출력
55. EmployeeID의 50%값을 더해 출력
56. EmployeeID가 1은 '일', 2는 '이'~ 9는 '나'까지 출력
57. TerritoryID 10000~30000 은 ' ID : 10000~30000', TerritoryID 30001~60000 은 ' ID : 30001~60000', TerritoryID 60001~90000 은 ' ID : 60001~90000', '이외의 나머지 ID'로 EmployeeID\_char열에 출력(TerritoryID도 함께 출력)

□ 시나리오 -employees

<employees 와 겹치는것 : employee Id (orders 테이블), employee id(employterritories 테이블)>

1. employees 테이블의 모든 데이터를 출력하시오.

```
select * from employees;
```

2.(서브쿼리) employees 테이블에서 'Janet'과 first name 이 같은 사람들의 employeeId,firstname 를 출력하시오.

```
select employeeId,firstname from employees where firstname=(select firstname from employees where firstname='Janet');
```

3. employees 테이블에서 Nancy와 도시가 같은 사람의 firstname, 생일 출력하되, first name순으로 내림차순으로 정렬하시오.

```
select firstname,birthdate from employees where city=(select city from employees where firstname='Nancy') order by firstname desc;
```

4. employees 테이블에서 Country 중 'a'가 들어가는 사람의 first name, last name, country를 출력하시오.

```
select firstname, lastname from employees where country like '%a%';
```

5. employees 테이블에서 Dodsworth 과 같은 도시에 사는 사람의 firstname 과 도시이름을 출력하시오.

```
select fistname,city from employees where city=(select city from employees  
where firstname='Dodsworth '');
```

6. employees 테이블에서 employee id가 1~5사이인 사람의 full name과 입사일을 출력하시오.

```
select firstname, lastname,hiredate from employees where employeeID between  
1and 5;
```

7. employees 테이블에서 미국 Seattle WA구역에 사는 사람 중 입사일이 가장 오래된 사람보다 더 입사일이 오래된 사람의 firstname, 입사일을 출력하시오.

```
select firstname, hiredate from employees where hiredate< all(select hiredate  
from employees where country='USA' and and region='WA' and city=Seattle');
```

8. employees 테이블에서 생일이 가장 느린 사람보다 빠른 생일을 가진 사람들의 employeesId와 생일과 전화번호를 출력하되 출신도시가 London인 사람은 출력하지 않습니다.

```
select birthdate,homephone from employees where birthdate>any(select birthdate  
from employees where city='London');
```

9. firstname이 Steven인 사람이 존재하는지, firstName과 현재나이를 출력하시오(올해는 2020년이고, 칼럼명은 올해나이로 설정한다.)

```
select firstName, months_between(trunc(sysdate,'year'),trunc(select birthdate  
from employees where firstname='Steven','year'))/12+1 현재나이 from employees;
```

10.(동등join) employee Id가 7번인 사람의 이름과 orderdate를 출력하시오.

```
select firstname,lastname,orderdate from employees e, orders o where  
e.employee ID=o.employee ID and employee ID=7;
```

11.(셀프join) 각사원의 관리자 이름을 출력하시오.

```
select a.employeeID,a.firstname, a.reportsto,b.firstname 관리자 from employees  
a, employees b where a.reportsto=b.employeesID;
```

12.(서브쿼리) Steven과 같은 날에 입사한 사원을 출력하시오.

```
select * from employees where hiredate=(select hiredate from employees where  
firstname='Steven');
```

13. 사원중에 lastname 에 'a'가 나오는 위치를 조회하시오.

```
select lastname, instr(lastname,'a',1)from employees;
```

14. 사원중에 `firstname`의 뒤에서 두번째부터 세글자를 조회하고 사원`firstname`과 결과를 출력하시오.

```
select firstname, substr(firstname,-2,3)from employees;
```

15. London에 사는 사원의 입사날짜 중 입사년도만 조회하시오.

```
select hiredate,substr(hiredate,1,2)from employees where city='London';
```

16. 1994년도에 입사한 사원의 이름, `employeeid`, 입사일을 출력하시오.

```
select firstname, lastname,employeeid, hiredate where hiredate like '1994%';
```

17. 각사원의 입사일로부터 6개월이 지난 날짜,`employeeid`를 조회하시오.

```
select add_months(hiredate,6)입사6개월 후,employeeid from employees;
```

18. 나라별로 가장 입사일이 오래된 사원의 정보(사원번호, `firstname`,`입사일`)를 출력하시오.

```
select employeeID,firstname,hiredate from employees where hiredate in(select max(hiredate) from employees group by country) order by country;
```

19. `employees` 테이블의 중복되지 않은 `title` 컬럼의 값들을 출력하시오. 단 컬럼명은 "부서"로 출력한다.

```
select distinct title 부서 from employees;
```

20. `employees` 테이블에서 `firstname`은 내림차순, `hiredate`를 오름차순으로 출력하시오.

```
select firstname, hiredate from employees order by firstname desc , hiredate ;
```

21. `employees` 테이블에서 보고자가 없는 사람을 출력하시오.

```
select reportsto , firstname from employees where reportsto is null;
```

22. `employees` 테이블에서 전사원의 평균나이(예:n세)를 출력하되, 컬럼명은 '평균 나이'로 하시오.

```
select concat(avg(months_between(trunc(sysdate,'year'),trunc(birthdate,'year'))/12+1),  
'세')as "평균 나이" from employees;
```

23.(동등join)각사원의 `id`와 `TerritoryId`를 출력하시오.

```
select employeeID ,TerritoryId from employees e,employeeterritories t where  
e.employeeId=t.employeeId;
```

24.(서브쿼리)`employees` 테이블과 `orders`테이블을 이용하여 배송국가가 독일인 사람의 `employeeid`와 배송국가를 출력하시오.

```
select employeeID, ShipCountry from employees e, orders o where  
e.employeeId=o.employeeId and ShipCountry='GERMANY';
```

25.(서브쿼리)employees 테이블과 orders테이블을 이용하여 직원들의 OrderId,employeeId를 출력하시오.

```
select employeeID,OrderId from employees e, orders o where e.employeeId=o.employeeId;
```

26.(update) 새로운 직원이 입사했다.

id는 10번, full name은 Jeremy Scott, 직업은 Sales Manager, 남자이며 생일은 1980년 1월 20일, 입사일은 2000년 5월 1일이다.

주소는 203 trole Rd.이며 출신국가는 네덜란드, 헤이그 출신이다. 지역은 woo 지역이며 우편번호는 10203이며 전화번호는 (102) 332-1323이다. 내선번호는 333이며 사진은 www.gif이다. 메모사항은 없으며, 보고자는 2번이며 photopath는 http://www.naver.com이다. 정보를 입력하시오.

```
insert into employees values (10,'Jeremy,Scott', 'Sales Manager', 'Mr','1980-01-20', '2000-05-01','203 trole Rd','Hague','woo','10203','Netherlands','(102) 332-1323',333,'www.gif',null,2,'http://www.naver.com',);
```

27.(function) Lower()함수를 이용하여 'HELLO'를 소문자로 출력하시오.

```
select lower('hello') from dual;
```

28.(function) INITCAP()함수를 사용하여 'this is a book'를 변환하시오.

```
select initcap('this is a book') from dual;
```

29.(function) employees 테이블에서 Steven 의 입사날짜에서 3개월후의 날짜를 출력하시오.

```
select add_months(hiredate,3) from employees where firstname='Steven';
```

30.(function) 사원들의 생일을 반올림과 내림하여 사원id, firstname과 함께 출력하시오.

```
select employeeId,firstname,round(birthdate,'Mon') from employees;
```

```
select employeeid,firstname,trunc(birthdate,'MON') from employees;
```

31.(function) employees테이블에서 reportsto가 null인 행을 출력하되, 0으로 변환하여 출력하시오.

```
select employeeid,lastname,nvl(reportsto,0)from employees;
```

□ 시나리오 -orders

1. 자신의 테이블의 모든 데이터를 출력하시오.

A. select \* from Orders;

2. 자신의 테이블의 중복되지 않은 EmployeeID컬럼의 값들을 출력하시오

A. select DISTINCT EmployeeID from Orders;

3. orders 테이블의 중복되지 않은 ShipCountry 컬럼의 값들을 출력하시오.

A. select distinct ShipCountry from orders;

4. Orders 테이블에서 주문자와 주문번호, 배송 받을 나라를 출력하시오  
A. `select OrderID, CustomerID, ShipCountry from Orders;`
5. Orders 테이블에서 OrderID 와 CustomerID, ShipCountry를 출력하시오 (CustomerID 오름차순)  
A. `select OrderID, CustomerID, ShipCountry from Orders order by CustomerID asc;`
6. Orders 테이블에서 OrderID 와 CustomerID, ShipCountry 를 출력하시오( CustomerID 내림차순)  
A. `select OrderID, CustomerID, ShipCountry from Orders order by CustomerID Desc;`
7. Orders 테이블에서 ShipCountry 가 ‘Brazil’ 인 고객을 출력하시오  
A. `select * from Orders where ShipCountry='Brazil';`
8. Orders 테이블에서 판매 담당자 ID가 2인 주문번호와 고객 ID, 판매 담당자ID를 출력하시오  
A. `select OrderID, CustomerID, EmployeeID from Orders where EmployeeID = 2;`
9. Orders 테이블에서 고객ID가 V로 시작하는 고객의 정보를 출력하시오.  
A. `select * from Orders where CustomerID like 'V%'`
10. Orders 테이블에서 고객ID에 v가 들어간 사람의 정보를 출력하시오  
A. `select * from Orders where CustomerID like '%V%'`
11. Orders 테이블에 판매담당자ID가 3,4,5 인 고객을 출력하시오 ( in 사용 )  
A. `select * from Orders where EmployeeID in(3,4,5);`
12. Orders 테이블에서 판매담당자ID가 3이상인 주문자 정보를 출력하시오. ( 논리연산자 사용 )  
A. `select * from Orders where EmployeeID>=3;`
13. Orders 테이블에서 OrderID 가 10240 ~ 10250 인 데이터를 출력하시오. (between 사용)  
A. `select * from Orders where OrderID between 10240 and 10250;`
14. 13번을 관계연산자와 논리연산자로 처리하시오.  
A. `select * from Orders where OrderID >=10240 and OrderID<=10250;`
15. Orders 테이블의 OrderID, CustomerID, ShippedDate, Freight를 출력하고, Freight의 평균보다 높은 값을 가진 데이터를 출력하시오  
A. `select OrderID, CustomerID, ShippedDate, Freight from Orders where Freight>(select avg(Freight) from Orders);`
16. orders 테이블의 Freight에서 최고 수치를 받은 데이터를 출력하시오  
A. `select * from orders where Freight=any(select max(Freight) from orders);`
17. any나 some 연산자를 사용해서, orders 테이블의 Freight에서 평균 수치보다 높은 수치의 데이터를 출력하시오.  
A. `select * from orders where Freight>any(select avg(Freight) from orders);`
18. Orders 테이블에서 EmployeeID 가 1인 고객의 데이터를 Emp01 테이블로 생성하고 저장하시오  
A. `create table emp01 as select * from Orders where EmployeeID =1;`
19. Orders 테이블과 동일한 Orderder 빈 테이블을 생성하시오.

- A. create table Orderder as select \* from Orders where 0=1;
20. 새로운 고객이 들어왔다. Orders 테이블에 추가하시오. OrderID = 10268, CustomerID = 'dongy' EmployeeID = 3이며 주문일자는 1996년 7월 23일이고 요청날짜는 1996년 8월 2일이다. 배송일자는 1996년 8월 12일, Shipvia 1, Freight 55.28, ShipName 'Blondel p?e et fils', ShipAddress '24, place Kl?er', ShipCity 'Strasbourg', ShipRegion null, ShipPostalCode '67000', ShipCountry 'Brazil'
- A. INSERT INTO Orders Values ('10268','dongy',3,'1996-07-23 00:00:00.000','1996-08-02 00:00:00.000','1996-08-12 00:00:00.000',1,55.28,'Blondel p?e et fils','24, place Kl?er','Strasbourg',NULL,'67000','Brazil');
21. Orders 테이블에 OrderID가 10265인 사람이 배송을 'Brazil'에서 받고 싶어 합니다. 이 정보를 수정하세요.
- A. update Orders set ShipCountry ='Brazil' where OrderID =10265;
22. Orders 테이블에 새로운 주문정보를 입력하세요.
23. OrderID = 10269, CustomerID = 'PARK', EMPLOYEEID= 8, 주문날짜는 '1996-08-10', 요청날짜 '1996-08-20', 배송날짜 '1996-08-14', 배송받을 나라 'Korea', 사는 도시는 'Seoul' 이 정보를 입력하시오
- A. INSERT INTO Orders Values ('10269','park',8,'1996-08-10','1996-08-20','1996-08-14',1,55.28,'Blondel p?e et fils','24, place Kl?er','서울',NULL,'67000','Korea');
24. 7월 25일에 주문한 정보를 입력하였다. 주문날짜를 수정하시오
- A. update Orders set OrderDate ='1996-07-27 00:00:00.000' where OrderID =10265;
25. Orders 테이블에 CustomerID의 공간이 부족합니다. varchar2(5)에서 varchar2(10)으로 수정하시오
- A. ALTER TABLE Orders add(CustomerID varchar2(10));
26. Orders 테이블의 OrderID가 10265인 데이터의 배송받을 나라를 'Korea'로 수정하시오
- A. update Orders set ShipCountry ='Korea' where OrderID =10265;
27. Orders 테이블에 있는 모든 행을 삭제하시오 (삭제후 rollback 하시오.)
- A. delete from Employee;
- B. rollback
28. Orders 테이블에 OrderID 가 10265인 데이터를 삭제하시오
- A. delete from Orders where OrderID = 10265;
29. Orders 테이블에 EMPLOYEEID 가 3인 데이터를 모두 삭제하세요.
- A. delete from Orders where EMPLOYEEID = 3;
30. Orders 테이블에서 CustomerID의 데이터를 소문자로 출력
- A. select lower (CustomerID) from Orders;
31. Orders 테이블에서 CustomerID의 데이터를 대문자로 출력
- A. select upper (CustomerID) from Orders;
32. Orders 테이블에 OrderID, CustomerID, EmployeeID, OrderDate의 문자열 길이와 바이트 단위로 계산하여 출력하시오.

- A. select length ('10248'),length('VINET'), lengthb('5'),lengthb('1996-07-04 00:00:00.000') from Orders;
33. instr를 이용하여 A가 2개인 고객의 이름을 출력하시오 substr를 이용하여 E로 끝나는 고객을 출력하시오
- A. select \* from Orders where instr(CustomerID,'A',1,2)>1;
- B. select \* from Orders where substr(CustomerID,-1,1)='E';
34. Orders 테이블에 EmployeeID 와 ShipVia 에 100을 더하는 작업을 하시오
- A. select EmployeeID+100,ShipVia+100 from Orders;
35. Orders 테이블에 ShipVia 가 1이면 남학생 ShipVia 가 2면 여학생 ShipVia = 3 이면 교사용을 출력하세요
- A. select OrderID, CustomerID,ShipVia, case when ShipVia=1 then '남학생' when ShipVia=2 then '여학생' when ShipVia=3 then '교사' end as School from Orders;
36. case 문을 활용하여 Order 테이블의 판매담당자의 번호를 1 = A. 2 = B ... 9 = I 로 변경하시오
- A. select OrderID, CustomerID,EmployeeID, case  
 i. when EmployeeID=1 then 'A'  
 ii. when EmployeeID=2 then 'B'  
 iii. when EmployeeID=3 then 'C'  
 iv. when EmployeeID=4 then 'D'  
 v. when EmployeeID=5 then 'E'  
 vi. when EmployeeID=6 then 'F'  
 vii. when EmployeeID=7 then 'G'  
 viii. when EmployeeID=8 then 'H'  
 ix. when EmployeeID=9 then 'i'  
 end  
 as NumberString from Orders;
37. Orders 테이블에 고객 회원의 수를 출력하시오
- A. select count(OrderID) from Orders;
38. Orders 테이블에 OrderID 번호가 가장 높은 사람과 가장 낮은 사람을 출력하시오
- A. select max(OrderID), min(OrderID) from Orders;
39. Orders 테이블에서 OrderID 의 평균 값과 더한 값을 출력하시오
- A. select avg(OrderID), sum(OrderID) from Orders;
40. Orders 테이블과 Employees 테이블을 크로스 조인하시오. ( employeeid = employeeid )
- A. select \* from employee e, orders o where e.employeeid = o.employeeid;
41. Orders 테이블과 Employee 테이블을 조인하여 OrderID, CustomerID, EmployeeID, EmployeeName, OrderDate를 출력하시오
42. select o.OrderID, o.CustomerID, o.EMPLOYEEID, e.EmployeeNAME, o.OrderDate  
 from Orders o, Employee e where o.EmployeeID = e.EmployeeID;
43. Orders 테이블과 Employee 테이블을 조인하여 OrderID 가 10248인 사람의 OrderID, CustomerID, EMPLOYEEID, EmployeeNAME, OrderDate를 출력하시오

A. select o.OrderID, o.CustomerID, o.EMPLOYEEID, e.EmployeeNAME, o.OrderDate  
from Orders o, Employee e where o.OrderID = 10248 and o.EmployeeID =  
e.EmployeeID;

44. Orders 테이블과 Employee 테이블을 조인하여 OrderID 가 10248인 사람의  
OrderID, CustomerID, EMPLOYEEID, EmployeeNAME, OrderDate를 출력하시오

A. select o.OrderID, o.CustomerID, o.EMPLOYEEID, e.EmployeeNAME, o.OrderDate  
from Orders o, Employee e where o.OrderID = 10248 and o.EmployeeID =  
e.EmployeeID;

□ 시나리오 -shippers

1. shippers 테이블의 모든 데이터를 출력하시오

2. shippers 테이블의 중복되지 않은 shipperID컬럼의 값들을 출력하시오[회사명 출력]

3. shippers 테이블의 중복되지 않은 shipperID컬럼의 값을 출력하시오. 단  
부제목(컬럼)은 "회사코드"로 출력한다.

4. orders 테이블에서 shipName, orderDate, requireDate를 출력하시오.

5. orders 테이블에서 shipName, orderDate, requireDate를 주문 날짜 오름차순으로  
출력하시오.

6. orders 테이블에서 shipName, shipDate, requireDate, shippedDate를 출력하되,  
shippedDate를 내림차순, requireDate 를 오름차순으로 출력하시오.

7. shippers 테이블로부터 '000-0000-0000' 전화번호의 회사명을 출력하시오

8. orders 테이블로부터 orderDate가 yyyy/mm/dd 이전인 회사명을 출력하되, orderDate를  
내림차순으로 출력하시오.

9. orders 테이블에서 orderDate가 'yyyy/mm/dd'인 shipName, shipAddress, shipCity를  
출력하시오.

10. shippers 테이블로부터 회사명, 전화번호를 자연어로 출력하되, 회사코드 순으로  
출력하시오.

11. orders 테이블로 부터 '000' customerID를 가진 shipName을 출력하시오.

12. orders 테이블로 부터 customerID컬럼의 중간글자가 'o'인 고객을 검색하여  
출력하시오.

13. orders 테이블에서 employeeID가 '00', orderDate가 'yyyy-mm-dd'에 속하는 고객의  
명단을 requiredDate순으로 출력하시오(in 사용)

15. 14번을 관계연산자와 논리 연산자로 검색하시오.

16. orders 테이블에서 orderDate가 yyyy-mm-dd 부터 yyyy-mm-dd까지의 기간 내 날짜를  
날짜순으로 출력하시오.(between 사용)

17. 16번을 관계연산자와 논리연산자로 처리하시오.

18. order 테이블로부터 shipRegion이 널인 행을 검색하여 출력하시오.

□ 시나리오 -customers

1. customers 테이블의 모든 데이터를 출력하시오.

A. select \* from cumtomers

2. customers 테이블의 중복되지 않은 CompanyName컬럼의 값들을 출력하시오

A. select unique companyname from customers

3. customers 테이블의 중복되지 않은 CompanyName컬럼의 값을 출력하시오. 단, 부제목(컬럼)은 "소속회사"로 출력한다.

A. select unique companyname as 소속회사 from customers;

4. customers 테이블에서 CustomerID, CompanyName, ContactName, ContactTitle을 출력하시오.

A. select CustomerID, CompanyName, ContactName, ContactTitle from customers;  
5. customers 테이블에서 CustomerID, CompanyName, ContactName, ContactTitle을 CustomerID 오름차 순으로 출력하시오.

A. select CustomerID, CompanyName, ContactName, ContactTitle from customers  
order by customerid;

6. customers 테이블에서 CustomerID, CompanyName, ContactName, ContactTitle을 출력하되, CustomerID를 내림차순, CompanyName는 오름차순으로 출력하시오.

A. select CustomerID, CompanyName, ContactName, ContactTitle from customers  
order by customerid asc, companyname desc;

7. customers 테이블로부터 City가 'London'인 고객명을 출력하시오.

A. select CustomerID, city from customers where city = 'London';

8. customers 테이블로부터 Country가 'Mexico'인 고객명을 출력하시오.

A. select CustomerID, country from customers where country = 'Mexico';

9. customers 테이블로부터 City가 'London'인 CompanyName, ContactName,  
ContactTitle을 출력하시오.

A. select CompanyName, ContactName, ContactTitle, city from customers where  
city = 'London';

10. customers 테이블에서 CustomerID, CompanyName, ContactName, ContactTitle을  
자연어로 출력하되, CustomerID순으로 출력하시오.

A. select CompanyName || ' 자연어처리 '|| ContactName || ' 뭔가좀이상한데 '||  
ContactTitle, city from customers where city = 'London' order by customerid

11. customers 테이블에서 CustomerID가 'B'로 시작하는 고객명단을 출력하시오.

A. select customerid from customers where customerid like 'B%'

12. customers 테이블에서 CustomerID가 중간 글자가'RA'인 CustomerID를 출력하시오.

A. select customerid from customers where customerid like '%RA%'

13. customers 테이블로부터 Region이 널인 행을 검색하여 출력하시오.

A. select customerid, region from customers where region is null

14. customers 테이블을 이용하여 지역별 고객인원을 출력하시오.

A. select count(customerid), region from customers group by region;

15. 2. customers 테이블을 이용하여 지역별 고객인원 1명인 지역을 출력하시오.

A. select count(customerid), region from customers group by region having  
count(customerid) = 1;

16. 3. customers 테이블의 행의 수와 region이 null이 아닌 행의 수를 반환한다.

A. select count(customerid) from customers where customerid in (select  
customerid from customers where region is not null);

17. 6. customers 테이블로부터 도시 별 인원수를 계산하여 도시별, 지역별로  
출력하시오.

A. select count(customerid), city from customers group by city;

□ 시나리오 -customercustomerdemo

1. Customers 테이블의 모든 데이터를 출력하시오

A. select \* from Customers;

2. CustomerCustomerDemo 테이블의 모든 데이터를 출력하시오

A. select \* from CustomerCustomerDemo;

3. Customers 테이블의 중복되지 않은 CustomerID컬럼의 값들을 출력하시오 [1명 이상의 과장이 존재하는 ContactTitle명 출력]

A. select distinct ContactTitle from Customers;

4. Customer 테이블의 중복되지 않은 CompanyName 컬럼의 값들을 출력하시오 [단 컬럼명은 '회사명'으로 출력한다]

A. select distinct CompanyName "회사명" from Customers;

5. Customers 테이블에 고객id(CustomerID), 담당자명(ContactName), 연락처(Phone)를 출력하시오

A. select CustomerID,ContactName,Phone from Customers;

6. Customers 테이블에 고객id(CustomerID), 담당자명(ContactName), 연락처(Phone)를 고객id 오름차 순으로 출력하시오

A. select CustomerID,ContactName,Phone from Customers order by CustomerID desc;

7. Customers테이블로 부터 'Mexico' 회사의 나라를 출력하시오

A. select \* from Customers where Country='Mexico';

8. Customers 테이블로 부터 'A'로 시작하는 ContactName 명단을 출력하시오

A. select \* from Customers where ContactName like 'A%'

9. Customer 테이블로부터 회사명(CompanyName) 컬럼의 중간 글자가 "a"인 사명을 검색하여 출력하시오

A. select \* from Customers where CompanyName like '\_a%';

10. Customer 테이블에서 회사명(CompanyName)이 ' Vaffeljernet ',' Ricardo Adocicados '인 담당자의 명단을 출력하시오(in사용)

A. select \* from Customers where CompanyName in (' Vaffeljernet ',' Ricardo Adocicados');

11. 10번을 관계연산자와 논리연산자로 출력하시오

A. select \* from Customers where CompanyName='Vaffeljernet' or CompanyName='Ricardo Adocicados';

12. Customer 테이블로부터 city 가 널인 행을 검색하여 출력하시오

A. select \* from Customers where city is null;

□ 시나리오 -customerdemographics

1. CustomerDemographics 테이블에 insert를 사용하여 데이터를 추가해보세요.

2. CustomerDemographics 테이의 데이터를 전체출력 하세요.

3. CustomerDemographics 테이블의 중복되지 않은 CustomerTypeID컬럼의 값들을 출력하시오.

4. CustomerDemographics 테이블의 CustomerTypeID컬럼 제목을 'Id'로 출력하세요.
5. 500\*5+120의 수식을 계산 하시오(Dual 테이블 사용).
6. CustomerDemographics 테이블에 CustomerTypeID가 a인 데이터를 출력해보세요.
7. CustomerDemographics 테이블의 CustomerTypeID가 'aaa'이거나 'bbb'인 CustomerDesc를 출력하세요.(in사용)
8. 7번 문제를 관계 연산자와 논리 연산자로 검색하세요.
9. CustomerDemographics 테이블의 CustomerTypeID컬럼의 중간 글자가 'b'가 들어간 데이터를 출력하세요.
10. CustomerDemographics 테이블의 CustomerDesc컬럼에 null이 들어간 데이터만 출력하세요.
11. CustomerDemographics 테이블의 CustomerDesc컬럼에 asd이면서 CustomerDesc가 asd인 데이터를 출력하세요.
12. CustomerDemographics 테이블의 CustomerDesc컬럼의 문자열 길이와 바이트 단위로 계산하여 출력하세요.
13. CustomerDemographics 테이블의 CustomerDesc컬럼의 문자열 길이와 바이트 단위로 계산하여 출력하세요.
14. 100을 3으로 나눈 나머지를 계산하여 출력하세요.
15. YYYY/MM/DD HH24:HH:SS 형태로 날짜를 출력하세요.
16. Lower()함수를 이용하여 CustomerDesc컬럼의 HELLO를 소문자로 출력하세요.
17. Upper()함수를 이용하여 CustomerTypeID컬럼의 hello데이터를 대문자로 출력하세요.
18. CustomerCustomerDemo테이블과 CustomerDemographics테이블을 크로스조인을 해보세요.
19. CustomerDemographics테이블에 CustomerDemographics을 이용하여 CustomerID를 구하세요.
20. CustomerDemographics컬럼이 a로 시작하는 데이터를 모두 조회해보세요.
21. CustomerDemographics컬럼에 b가 들어가는 데이터를 모두 출력하세요.

#### □ 시나리오 -order\_details

1. Order\_Details 테이블에서 모든 데이터를 출력하시오.
- A. 

```
select * from Order_Details;
```
2. Order\_Details 테이블에서 모든 OrderID를 출력하시오.(중복x)
- A. 

```
select distinct orderID from order_details order by orderID;
```
3. Order\_Details 테이블에서 모든 ProductID를 출력하시오. 부제목은 '제품명'(중복x)
- A. 

```
select distinct productID 제품명 from order_details order by productID;
```
4. Order\_Details 테이블에서 OrderID와 가격을 출력하되 Order\_ID는 오름차순, UnitPrice는 내림차순으로 출력하시오.
- A. 

```
select orderID, productID from order_details order by orderid asc, productid desc;
```
5. Order\_Details 테이블에서 orderID, productId, unitPrice, quantity, 총 주문금액을 출력하시오.
- A. 

```
select orderID, productId, unitPrice, quantity, quantity*unitPrice
      주문금액 from order_details;
```

6. Order\_Details 테이블에서 UnitPrice 반올림이 19 이상인 데이터들만 출력하시오.
- A. `select * from order_details where round(unitprice)>=19;`
7. Order\_Details 테이블에서 할인율(Discount)이 '0'이 아닌 모든 데이터를 출력하시오.
- A. `select * from order_details where discount !=0;`
8. Order\_Details 테이블에서 UnitPrice가 25.0 이상인 모든데이터를 출력하시오
- A. `select * from order_details where unitprice>=25.0;`
9. Order\_Details 테이블에서 Quantity(수량)이/가 20 이상 30 이하 이면서 Discount가 0.15인 데이터들만 출력하시오.
- A. `select * from order_details where quantity between 20 and 30 and discount=0.15;`
10. Order\_Details 테이블에서 OrderID가 '11077'인 사람이 주문한 금액의 평균을 구하시오.
- A. `select orderid, avg(unitprice) from order_details where orderid=11077 group by orderid;`
11. Order\_Details 테이블에서 각 OrderID 마다 Unitprice합계와 평균을 출력하되 OrderID 오름차순으로 출력하시오.
- A. `select orderID, sum(unitprice), avg(unitprice) from order_details group by orderID order by orderID asc;`
12. Order\_Details 테이블에서 OrderID별 최고unitPrice,최저unitPrice를 출력하시오.
- A. `select orderid, max(unitPrice), min(unitPrice) from order_details group by orderID order by orderid;`
13. Order\_Details 테이블에서 OrderID가 '10279'인 사람과 같은 주문상품(ProductID)을 주문한 OrderID, ProductID를 출력하시오.
- A. `select orderId,productId from order_details where productid=(select productId from order_details where orderid=10279);`
14. Order\_Details 테이블에서 unitPrice 전체평균보다 높은 OrderID, unitPrice 를 출력하세요
- A. `select orderId,unitprice from order_details where unitprice>(select avg(unitprice) from order_details);`
15. Order\_Details 테이블에서 OrderID '10654'가 가지는 dicount와 같은 discount를 가지는 OrderID와 discount를 출력하시오.
- A. `select distinct Orderid,discount from order_details where discount=(select distinct discount from order_details where orderid=10654);`
16. Order\_Details 테이블에서 OrderID '10261'이 가지는 quantity와discount가 같은 OrderID, quantity, discount 데이터를 출력하시오.
- A. `select distinct orderid, quantity, discount from order_details where quantity=(select distinct quantity from order_details where orderid=10261) and discount=(select distinct discount from order_details where orderid=10261) order by orderID;`
17. order\_details과 동일한 T\_Order\_Details 빈 테이블 생성하시오.
- A. `create table T_Order_Details as select * from Order_Details where 1=0;`

18. Order\_Details 테이블과 Products테이블을 조인하시오.
- A. select \* from order\_details o, Products p where o.productId=p.productId;
19. Order\_Details 테이블과 Orders테이블을 조인하시오.
- A. select \* from order\_details o, orders os where o.orderID=os.orderid;
20. Order\_Details 테이블과 Products테이블을 참조해 OrderID마다 주문한 ProductID,ProductName을 출력하시오.
- A. select o.orderId, o.productId, p.productName from order\_details o, products p where o.productid=p.productid;
21. Order\_Details 테이블과 Orders테이블을 참조해 OrderID 마다 가지는 CustomerID를 출력하되 OrderID 오름차순으로 출력하시오.(중복x)
- A. select distinct o.orderid, customerID from order\_details o, orders os where o.orderID=os.orderid order by orderId;
22. Order\_Details 테이블과 Orders테이블을 참조해 OrderID 마다 가지는 CustomerID를 출력하되 CustomerID가 'W'로 시작하는 데이터만 출력하시오.
- A. select distinct o.orderid, customerID from order\_details o,orders os where o.orderID=os.orderid and customerID like 'W%';

□ 시나리오 -products

1. products 테이블의 모든 데이터를 출력하시오.
2. Products 테이블에서 단가(UnitPrice)와 재고(UnitsInStock) 열을 곱하여 '재고금액' 열을 생성하는 SELECT 문을 출력.
3. products 테이블의 중복되지 않은 공급자id(SupplierID)컬럼의 값들을 오름차순으로 출력하시오.
4. products 테이블의 상품코드(productid),상품이름(productname),단가(unitprice)를 출력하시오.
5. products 테이블의 상품이름(productname)과 단가(unitprice)를 단가가 낮은순으로 출력하라.
6. products 테이블의 1개 이상 주문된 상품이름(productname)과, 주문수량(unitsonorder)을 출력하라.
7. products 테이블의 주문수량(unitonorder)과 상품단가(unitprice)를 이용해 주문된 상품이름(productname)과 주문수량(unitsonorder),총 주문가격 을 출력하라. (단 주문수량이 0개인곳은 출력하지 않는다).
8. products 테이블의 재주문레벨(ReorderLevel)이 제일 높은 순으로 상품이름(productname)과 재주문레벨(ReorderLevel)출력하라.(단 재주문횟수가 0개인곳은 출력하지 않는다.)
9. products 테이블의 상품 총 주문금액이 1000 이상인 상품코드(productid)와 상품이름(productname), 재주문레벨(ReorderLevel)을 출력하라.
10. products 테이블의 상품중에서 생산중단(Discontinued=1)된 상품코드와 상품이름을 출력하라.
11. products 테이블에서 공급자id(SupplierID)가 5인 상품(productname)과 상품단가(unitprice), 공급자id(SupplierID)를 출력하라.

12. products 테이블에서 카테고리id(CategoryID) 별로 상품단가(unitprice)가 제일 높은금액과 낮은금액을 출력하라
13. products 테이블에서 공급자id 의 갯수는 총 몇개인지 출력하라.
14. 14,products 테이블에서 상품이름이 'Tofu' 인 공급자ID(SupplierID)와 상품단가(unitprice), 재주문레벨(reorderlevel)을 출력하라
15. products 테이블에서 카테고리별(categoryid)로 재주문횟수(reorderedlevel)가 가장 높은것과 가장 낮은것을 출력하라
16. products 테이블의 공급자id(SupplierID)가 14인 상품이름(productname)과 공급자(suppliers 테이블)의 주소(address)를 출력하라.(조인)
17. products 테이블의 카테고리id(categoryid)가 3인 상품이름(productname)과 카테고리이름(categoryname)을 출력하라.(categories 테이블 이용)(조인)
18. products 테이블의 공급자id(supplierid)가 5인 공급자(suppliers 테이블)의 회사이름(companyname), 상품이름(productname), 상품단가(unitprice)를 출력하라.(조인)
19. products 테이블의 카테고리id(categoryId)가 1인 상품이름(productname)과 상품코드(productid)를 출력하라.
20. products 테이블의 상품명이 pavlova와 같은 공급자id(supplierid)인 상품이름(productname)과 상품단가(unitprice)를 출력시오.(서브쿼리)

□ 시나리오 -suppliers

1. supplier 테이블의 모든 데이터를 출력하시오.  
A. `select * from supplier;`
2. supplier 테이블의 중복되지 않는 나라의 값을 출력하시오.  
A. `select distinct country from Suppliers;`
3. supplier 테이블의 중복되지 않는 나라의 값을 출력하되 컬럼을 바꾸시오.  
A. `select distinct country as 나라 from Suppliers;`
4. supplier 테이블의 부분출력 (회사이름, 관계자이름, 나라)  
A. `select companyname, contactname, country from Suppliers;`
5. supplier 테이블의 회사이름으로 정렬 (굳이 할 필요없음)  
A. `select * from Suppliers order by companyname;`
6. supplier 테이블의 국가가 USA인 회사이름을 출력하시오.  
A. `select companyname from Suppliers where country = 'USA' ;`
7. supplier 테이블의 국가가 USA인 도시이름을 출력하시오.  
A. `select city from Suppliers where country = 'USA' ;`
8. supplier 테이블의 나라가 'USA'인 (회사이름, 관계자이름, 나라) 출력  
A. `select companyname, contactname, country from Suppliers where country = 'USA' ;`
9. supplier 테이블의 (회사이름, 관계자이름, 나라) 출력하되 나라순으로  
A. `select companyname, contactname, country from Suppliers order by country;`
10. supplier 테이블의 M으로 시작하는 사람의 이름을 출력  
A. `select contactname from Suppliers where contactname like 'M%' ;`
11. supplier 테이블의 이름 중간에 r 들어가는 사람의 이름을 출력  
A. `select contactname from Suppliers where contactname like '%r%' ;`

12. in 연산자 이용 supplier테이블에서 국가가 usa,germany인 도시를 출력  
A. select city from Suppliers where country in ('USA', 'Germany') ;
13. supplier테이블에서 미국이라는 나라가 존재하면 일본을 나라로 가지고 있는 모든것 출력  
A. 1. select \* from Suppliers where country= 'Japan' and exists(select \* from Suppliers here country = 'USA');
14. supplier테이블에서 미국에 소재지를 둔 회사의 모든 product테이블의 정보를 출력하시오.  
A. select \* from products where supplierid in (select supplierid from Suppliers where country = 'USA');
15. Suppliers 테이블의 회사이름을 소문자로 출력  
A. select lower(companyname) from Suppliers;
16. Suppliers 테이블의 회사이름을 대문자로 출력  
A. select upper(companyname) from Suppliers;
17. Suppliers 테이블의 회사이름을 첫글자만 대문자로 출력  
A. select initcap(companyname) from Suppliers;
18. Suppliers 테이블의 회사이름, 도시이름을 문자열길이와 바이트 단위 계산하여 출력  
A. select length(companyname), length(city), lengthb(companyname), lengthb(city) from Suppliers;
19. Suppliers 테이블의 전화번호 앞에 063 추가  
A. select '(063)' || phone from Suppliers;
20. Suppliers 테이블의 회사이름, 매니저이름, 합치기  
A. select concat(companyname, contactname) from Suppliers;
21. Suppliers 테이블의 회사이름, 매니저이름, 나라 합치기  
A. select companyname || contactname || country from Suppliers;
22. Suppliers 테이블의 나라이름이 USA인 나라를 uussaa로 바꾸기  
A. select replace(country, 'USA', 'uussaa') from Suppliers;
23. Suppliers 테이블의 지역이 널인 모든 데이터 찾기  
A. select \* from Suppliers where region is null;
24. Suppliers 테이블의 지역이 널인 모든 데이터는 0으로 치환  
A. select \* from Suppliers where nvl(region,0);
25. 현재 거래하고 있는 회사 수  
A. select count(distinct companyname) from suppliers;
26. 현재 거래하고 있는 나라의 수  
A. select count(distinct country) from suppliers;
27. 팩스번호가 널이 아닌 회사의 수  
A. select count(distinct companyname) from suppliers where fax is not null;
28. 현재 거래하고 있는 회사 수  
A. select count(distinct companyname) from suppliers;
29. 현재 거래하고 있는 회사 수와 거래하고 있는 나라의 수  
A. select count(distinct companyname), count(distinct country) from suppliers;

30. Suppliers 테이블과 products 테이블의 크로스조인  
A. `select * from Suppliers, products;`
31. Suppliers 테이블과 products 테이블을 이용하여 나라가 일본인 물건을 모든 것 출력  
A. `select * from products where supplierid in (select supplierid from Suppliers where country = 'Japan');`
32. 회사명 Exotic Liquids인 회사의 모든 제품 정보 출력  
A. `select * from products where supplierid in (select supplierid from Suppliers where companyname = 'Exotic Liquids');`
33. 회사명 Exotic Liquids인 회사의 제품이름, 넘버링, 가격 출력  
A. `select productname, supplierid, unitprice from products where supplierid in (select supplierid from Suppliers where companyname = 'Exotic Liquids');`
34. 제품명, 넘버링, 가격, 회사이름, 회사국적나라 출력  
A. `select p.productname, p.supplierid, p.unitprice, s.companyname, s.country from products p, Suppliers s;`

□ 시나리오 -categories

1. categories 테이블의 모든 데이터를 출력하시오.
2. categories 테이블에서 카테고리명(CategoryName), 카테고리설명(Description)을 출력하시오.
3. categories 테이블로부터 카테고리 ID가 3인 카테고리명을 출력하시오.
4. categories 테이블로부터 카테고리 설명에 'Desserts'가 들어간 데이터를 출력하시오.
5. categories 테이블로부터 카테고리명에 'Pro'가 들어간 데이터를 출력하시오.
6. categories 테이블에서 카테고리명이 'Seafood', 'Grains/Cereals' 인 데이터를 카테고리 ID순으로 출력하시오.(in 사용)
7. 윗 번호를 관계연산자와 논리 연산자로 검색하시오.
8. categories 테이블과 동일한 t\_categories 빈 테이블을 생성하시오.
9. 새로운 카테고리가 추가되었다. categories 테이블에 추가하시오. 카테고리 ID는 9, 카테고리명 'Coffee', 카테고리 설명 'All kinds of coffee' 사진 'X'이다.
10. categories 테이블에서 카테고리명이 'Dairy Products'인 데이터의 카테고리 설명을 변경하고자 한다. 해당 데이터의 CategoryDescription를 'Cheeses, milks'로 변경하시오.
11. categories 테이블에서 카테고리 ID가 9인 데이터를 삭제하시오.
12. categories 테이블에서 카테고리 명이 'Products'인 행을 삭제하시오.(삭제 후 rollback)
13. categories 테이블과 products 테이블을 크로스 조인하시오.

## > 05. 게시판 데이터베이스 설계하기

이미지를 참고하여 여러 사람이 글쓴 내용을 저장하는 게시판 테이블을 설계해 보자.

```
drop table board;
CREATE TABLE board (
    bId NUMBER PRIMARY KEY,
    bName NVARCHAR2(20) NOT NULL,
    bTitle NVARCHAR2(255) NOT NULL,
    bContent NVARCHAR2(2000) NOT NULL,
    bEtc NVARCHAR2(2000) NOT NULL,
    bWriteTime DATE DEFAULT sysdate,
    bUpdateTime DATE DEFAULT null,
    bHit NUMBER DEFAULT 0,
    bGroup NUMBER,
    bStep NUMBER,
    bIndent NUMBER,
    bDelete NCHAR(1) DEFAULT 'N'
);
```

여러 게시판을 하나의 테이블에 넣고 싶다면 category 컬럼을 추가 하면된다. 이해가 안되면 그냥 넘어가자.

	제목	작성자	작성일	조회
13391	R언어 시험문제.	foxman12	2021.05.04.	76
13390	빅데이터 Ⓢ	foxman12	2021.05.04.	80
13383	사전사후 인공지능 빅데이터 doc파일들 Ⓢ	foxman12	2021.05.03.	31
13360	출석 날짜	foxman12	2021.04.28.	58

상위 이미지 맨 왼쪽에 13391 13390 등의 데이터는 board 테이블의 모든 데이터를 식별하는 primary key 이고 컬럼명을 id로 할 예정이며 지속적으로 글을 작성할 때마다 글번호는 하나씩 증가해서 테이블의 모든 데이터를 식별하는데 사용하기 때문에 boardCount라는 시퀀스를 만들어서 사용하면 편리 할 것이다. 제목은 게시판에 쓴 글의 제목이고 컬럼명을 title로 할 예정이고, 작성자 writer, 작성일 wDate, 조회 hit로 할 예정이다. 그리고 이미지에는 없지만 글의 내용을 저장하는 content 컬럼이 필요할 것이다. 작성일은 입력할 때 데이터베이스에 insert하는 시점이 글쓴 시간이므로 기본값을 현재시간을 나타내는 sysdate를 사용하고 hit 같은 경우는 insert하는 시점에 조회수가 0이므로 기본값을 0으로 할 예정이다. etc는 예상하지 못한 컬럼이 있을 때 임시로 사용하기 위해서

만들어 놓은 컬럼이다. 추가로 할 수 있는 delete 컬럼은 해당글을 지울때 사용하는 컬럼이다 y이면 지워진다.

다음 이미지 처럼 답변형 게시판을 만들려면 group, indent, step 같은 컬럼을 추가해야 가능하다.

1	4259	게시판 코드 [1]
2		↳ 게시판 코드
3	4253	↳ 게시판
4		↳ 게시판
5		↳ 게시판2
6		↳ 게시판1
7	4252	평가자 체크리스트 ALL
8	4251	체크리스트 평가기준
9	4250	쇼핑몰 관련 예제

group은 관련있는 글을 묶어놓은 컬럼이다. 관련있는 글이란? 처음쓴 글과 관련된 댓글들에 해당한다.  
관련있는 글들은 group 컬럼에 같은 숫자를 넣어서 구분한다.  
맨 처음 쓴 글의 id가 4259번 이라면 해당글의 group번호는 4259이다.  
맨 처음 쓴글의 group번호는 해당

글의 id를 넣는다. 맨 처음 쓴글이 아니더라도 다른 글의 답글들은 group번호에 본인의 id를 넣는 것이 아니고 답글의 부모 group 번호를 넣는다. 이렇게 하면 처음 쓴 글과 관련있는 답글들은 모두 같은 group번호를 가진다.

이미지에서 group번호가 같은 것끼리 묶어보면 9번 제일 먼저 생성된 그룹이고 4250이 들어가 있다. 8 4251, 7 4252, 도 각각 그룹이고 3,4,5,6번은 4253글 그룹이고 1,2번은 4259번 글그룹이다.

group데이터를 넣는 방법은 답글이 아닌 첫 글은 해당글의 id를 넣으면 되고, group의 첫글이 아닌 답글을 쓸때는 부모의 group과 같은 번호를 넣어주면 된다.

그룹(column)을 만든 이유는 id(column)으로 내림차순으로 정렬하면 답글들은 나중에 입력되는 특성상 연속적으로 정렬되지 않아 보기 어렵다. 하지만, 그룹(column)이 같은 순으로 정렬하면 그룹으로 묶여 출력됩니다.

그룹(column)이 있어도 그룹 번호가 같은 글 사이에서는 정렬을 보장받지 못하기 때문에, 이를 보완하기 위해 'step'이라는 컬럼을 추가하여 화면에 보여주는 순서대로 0,1,... 과 같은 데이터를 0부터 1씩 증가한 숫자를 넣어 같은 group 안에 step 컬럼을 이용하여 정렬된 결과를 얻을 수 있고 sql로는 다음과 같다.

```
select * from board order by group desc, step asc  
상위 이미지에서 현재 각 step값들을 확인해 보자.
```

1,3,7,8,9 번 데이터는 각 그룹에서 맨상단에 위치해 있어서 step값은 0이다. 2,4 번은 각 그룹의 두 번째에 위치하고 있어서 step값은 1이다. 5번은 step값이 2가 되고, 6번은 step값이 6번이다.

step에 값을 넣는 방법은 group에 첫번째 데이터를 넣을때 step값은 항상 0이고 답글을 입력 할때는 부모글 바로 아래 답글이 있어야 하므로 부모보다 하나큰 step값을 넣어야 한다. step이 0인 글에 답글을 단다면 step이 1이여야 한다. step이 3인 글에 답글을 단다면 step이 4여야 한다.

여기서 step이 0인 글에 답글을 달때 같은 group에 step 1인 데이터가 이미 있다면 같은 데이터가 여러개 들어가 중복된 step이 1인 값을 가지게 된다. 이런 문제를 해결방법은 step이 1인 데이터를 넣기 전에 같은 그룹에 이미 step 1인 값이 있는지 확인해서 같은 그룹에 step 1보다 같거나 큰 모든 데이터를 하나 증가한 다음에 step이 1인 값을 넣으면 된다.

step이 3인 글에 답글을 달때 같은 group에 step 4인 데이터가 이미 있다면 같은 데이터가 여러개 들어가 중복된 step이 4인 값을 가지게 된다. 이런 문제를 해결방법은 step이 4인 데이터를 넣기 전에 같은 그룹에 이미 step 4인 값이 있는지 확인해서 같은 그룹에 step 4보다 같거나 큰 모든 데이터를 하나 증가한 다음에 step이 1인 값을 넣으면 된다.

같은 group에서 step 값이 같은 데이터가 여러개 만들어 진다면 정렬이 불가능 하다.

indent의 경우에는 상위 이미지에서 새로운 부분을 보면 그룹에 맨처음 작성된 1,3,7,8,9 번 데이터의 경우 왼쪽에 딱 붙어 있지만 나머지 글들인 답글들은 적절간격으로 뛰어져 있는 것을 확인 할 수 있다. indent는 왼쪽에서 얼마나 떨어져 있느냐를 숫자로 저장한 컬럼 이여서 1,3,7,8,9번 데이터는 0이되고 2,4,6 번은 1이되고 5번은 2가 된다.

indent에 값을 넣는 방법은 잘 보면 부모보다 하나더 들여 쓰고 있는것을 확인할 수 있다. 따라서, 첫글은 0를 넣고 답글은 부모보다 하나더 큰 수를 넣으면 된다.

상위 답글형 게시판 이미지와 동일한 데이터를 넣어보는 시나리오를 작성해보자.

1. 상위에서 제일 먼저 입력되는 데이터는 9번 데이터이다. 시퀀스를 이용해서 id값을 넣어야 하지만, 상위 데이터와 동일한 아이디로 만들기 위해서 인위적으로 4250를 입력해서 넣어 보자.
2. 다음에 들어갈 순서는 8,7,3,1순서가 될것이다.
3. 다음에 들어갈 순서는 2,6번이 될것이다.
4. 다음에 들어갈 순서는 4번이 될것이다. 4번 넣기 전에 부모글 보다 step이 큰 6번의 step 값을 증가시키는 작업을 한다음에 4번을 입력해야 할것이다.
5. 다음에 들어갈 순서는 5번이 될 것이다. 5번 넣기 전에 6번의 step 값을 증가시키는 작업을 한다음에 5번을 입력해야 할것이다.

최종 sql를 작성해서 select문을 이용해서 똑같이 출력되는지 확인해 보자.

## > 06. 정규형

정규화란? 테이블에서 발생할수 있는 삽입 수정 삭제 이상이 발생하지 않도록 테이블을 쪼개 올바른 테이블들로 만드는 데이터베이스 설계 방법 중 하나이다.

1NF(제1정규형): 중복된 데이터를 제거하고 각 컬럼에는 원자값만 존재하도록 만듭니다.

2NF(제2정규형): 부분적 종속을 제거하여 모든 컬럼이 기본키에 대해 완전 함수 종속이 되도록 합니다.

3NF(제3정규형): 이행적 종속을 제거하여 모든 컬럼이 기본키에 대해 이행적 종속이 되지 않도록 합니다.

BCNF(보이스-코드-정규형): 모든 결정자가 후보키가 되도록 하여 다른 종속성을 제거합니다.

4NF(제4정규형): 다치 종속을 제거하여 여러 개의 다치 종속이 발생하지 않도록 합니다.

### - 제1정규형

GNo	GPrice
1	30000,35000
2	25000
3	20000

GNo	GPrice
1	30000
1	35000
2	25000
3	20000

제1정규형 속성의 도메인이 원자값이다. 테이블 하나의 컬럼의 값으로 하나의 데이터만 들어 간다는 의미이다. 속성 컬럼 도메인 컬럼의 값의 범위, 원자값 나누어 질수 없는 하나의 값

하나의 데이터 row는 하나의 컬럼의 값으로 한 개의 데이터만 가져야 하는데 왼쪽처럼 하나의 포도등급 1에 30000이라는 값과 35000이라는 값을 함께 가지고 있다. 제1정규형에 위배된 것이어서 아래와 같이 변경하여야 한다.

### - 제2정규형

함수 종속(Functional Dependency)은 관계형 데이터베이스에서 한 속성(attribute)의 값이 다른 속성에 종속되는 관계를 나타냅니다. 특정 관계에서 A와 B가 있을 때, A의 값이 주어지면 B의 값이 항상 유일하게 결정된다면, "A 함수적으로 B에 종속한다"라고 말할 수 있습니다.

제2정규형 기본키가 아닌 모든 속성은 부분 함수 종속을 제거 하여 기본키에 완전함수 종속이어야 한다. 기본키가 다른 컬럼의 값을 결정 한다는 의미로 테이블 안에서 pk 컬럼에 하나의 값은 다른 컬럼의 값을 결정한다는 의미이다.

Cno	GNo	count	gprice
1	1	5	30000
1	2	2	25000
1	4	4	15000
2	3	3	20000
3	2	2	25000
4	3	3	20000

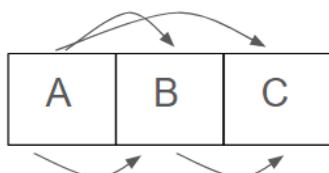
왼쪽 테이블은 고객이 특정 포도 등급 박스를 얼마나 구매하는지 기록하는 테이블이다. cno는 고객번호, gno는 포도등급, count는 판매한 박스수, gprice는 등급별 가격이다.

기본키가 cno, gno이고 count를 결정하지만 gprice는 gno가 결정하여 기본키가 아닌 gno가 결정자이므로 pk가 아닌 부분이 결정을 하고 있어서 완전 함수종속이라 할 수 없다. cno, gno, count로 이루어진 테이블과 gno, gprice로 이루어진 테이블로 분리되어야 한다.

다음 문제를 풀어보자. 성적, 학번, 과목번호, 지도교수, 학과 컬럼을 가지는 테이블에 대해서 2nf를 구성해 보자.

### - 제3정규형

제3정규형은 제2정규형에 속하고, 이행적 함수종속이 되지 않아야 한다. 이행적 함수 종속이란? 숫자에서 A<B<C 이면 A<C가 성립하는데 이런 경우를 이행적 함수 종속이라고 한다.



한 테이블에 A, B, C 컬럼이 있을 때 A가 B를 결정하고, A가 C를 결정하면 제 2 정규형에 해당해서 완전 함수 종속이 되어 정상적인 테이블이 된다. 하지만, A가 B를 결정하고 B가 C를 결정하면 이행적 함수 종속을 포함하고 있는 테이블이 되고 이행적 함수 종속을 없애기 위해서 A, B 컬럼을 포함한 테이블과 B, C를 포함한 2개의 테이블로 분리해야 한다.

Bno	GNo	gprice
1	1	30000
2	2	25000
3	4	15000
4	3	20000
5	2	25000
6	3	20000

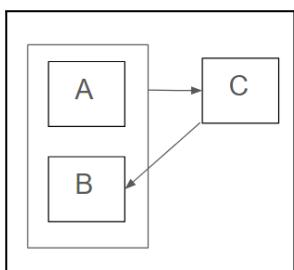
Bno	GNo
1	1
2	2
3	4
4	3
5	2
6	3

GNo	gprice
1	30000
2	25000
4	15000
3	20000

왼쪽 테이블에서 bno는 포도 박스의 박스 번호를 의미하고 gno는 포도 등급을 의미하고 gprice는 등급별 가격을 의미 한다. 결정자이자 pk인 bno 컬럼은 gno와 gprice를 결정한다. 테이블의 모든 컬럼이 키에 위해서 결정되고 있어서 정상적인 테이블처럼 보이지만 자세히 보면 gno도 gprice를 결정 하여 이행적 함수 종속이 존재 한다. bno가 gno를 결정하고 gno가 gprice를 결정 한다. 따라서, bno와 gno로 구성된 테이블과 gno와 gprice로 구성된 테이블로 분리하면 분리된 테이블에 중복된 데이터를 제거할 수 있다. 완전 함수종속이라 할 지라도 이행적 함수 종속이 존재하면 중복값이 생긴다.

다음 문제를 풀어보자. 학번, 과목번호, 교수 컬럼을 통해서 3nf를 구성해 보자.

### - 보이스/코드 정규형



- A, B 컬럼으로 이루어진 기본키가 C 컬럼을 결정해서 모든 컬럼이 기본키에 의해서 완전 함수수 종속이다. 문제가 없어 보이지만 C컬럼은 기본키의 일부분인 B컬럼을 결정한다. 후보키가 아닌 C컬럼이 B컬럼을 결정한다. 이럴 경우 A,C 컬럼과 B,C컬럼으로 테이블을 분리하여 C컬럼의 중복 데이터를 없애고 후보키로 만들어 줘야 한다.

완전 함수 종속에서 모든 결정자는 반드시 후보키여야 한다. 완전 함수 종속인 테이블이라도 특정 컬럼이 결정자인데 후보키가 아니면 보이스/코드 정규형에 위배 된다는 이야기이다.

Cno	Gkind	Gplantation
1	왕포도	강원농장1
1	건포도	강원농장2
1	청포도	제주농장1
2	청포도	제주농장1
2	건포도	제주농장5
3	왕포도	강원농장1

Cno	Gplantation
1	강원농장1
1	강원농장2
1	제주농장1
2	제주농장1
2	제주농장5
3	강원농장1

Gkind	Gplantation
왕포도	강원농장1
건포도	강원농장2
건포도	제주농장5
청포도	제주농장1

상위 맨 왼쪽 테이블은 고객이 주문한 포도의 원산지를 표시한 테이블이다.

cno는 고객번호이고 gkind는 포도종류이고 gplantation은 포도가 재배된 농장이다.

테이블 설계:

테이블1: (cno, gkind, gplantation) 한 고객이 구매한 여러 포도종류와 포도농장이 기록됨

테이블2: (cno, gplantation) 한 고객이 구매한 농장들이 기록됨

테이블3: (gkind, gplantation) 한 농장에서는 한 종류의 포도만 재배됨.

테이블별 기본키

테이블1: (cno, gkind)로 복합키, 모든 컬럼이 기본키에 종속됨.

테이블2: (cno, gplantation)로 복합키, 모든 컬럼이 기본키에 종속됨.

테이블3: (gkind, gplantation)로 복합키, 모든 컬럼이 기본키에 종속됨.

테이블1에서 gplantation은 후보키가 아니므로 BCNF를 만족하지 않음.

테이블1을 분해하여 테이블2와 테이블3으로 나누어서 BCNF를 만족하도록 함.

- 다음 문제를 생각해 보자. 학번, 과목, 강사로 이루어진 컬럼들에서 bcnf를 구성해 보자.

다음 문제를 확인하여 정규화를 통하여 ERD와 sql을 작성해 보자.

다음 페이지에 있는 이미지들과 같은 결과를 얻을 수 있는 데이터 베이스를 만들어 보자. 아래에 있는 이미지는 특정과정에 대한 자기진단을 하는 시험지인데 사전, 사후, 체크리스트 등 다양한 형태로 제공되며 여러 학교와 여러 과정과 여러 학생들의 시험결과를 시험일정에 맞춰서 시험을 보고 있다. 능력단위 요소는 능력단위명에 따라 현재는 2개 존재하지만 3~4개등 다양하게 존재한다. 능력단위 요소별 진단 문항은 다양 하다.

아래와 같은 결과물을 얻을 수 있는 시험 결과 관리 프로그램을 만들어 사용할 수 있도록 데이터 베이스를 만들어 보자.

1. 시나리오 작성
2. ERD 그림으로 그려서 작성
3. 테이블 sql작성
4. 가상데이터 입력sql작성
5. 설계한 데이터베이스에서 발생할 수 있는 시나리오에 대한 sql를 작성해 보자.

## 자기진단 평가(사후평가)

### (사후) 학습자 자가 진단 평가

교육기관	휴먼교육센터	교육기간	2021년 03월 02일 ~ 2021년 03월 09일		
평가일시	2021-03-09 화요일	과정명	Python(파이썬), Java(자바)기반 AI 활용 응용소프트웨어 개발자 양성과정	학생명	
교과목	애플리케이션 설계			평가자	박 수 민 (인)
능력단위명	애플리케이션 설계 (2001020221_16v4)				
능력단위요소	공통 모듈 설계하기, 타 시스템 연동설계하기				

진단영역	진 단 문 항	매우 미흡	미흡	보통	우수	매우 우수
공통 모듈 설계하기	1.1 지시용성 확보와 중복개발을 회피하기 위하여, 관제 시스템 차원과 단위 시스템 차원의 공동부분을 식별하여 이에 대한 상세 명세를 작성할 수 있다.	①	②	③	④	⑤
	1.2 개발할 응용소프트웨어의 전반적인 기능과 구조를 이해하기 쉬운 크기로 공동 모듈을 설계할 수 있다.	①	②	③	④	⑤
	1.3 소프트웨어 측정지표 중 모듈간의 결합도는 줄이고 개별 모듈들의 내부 융집도는 높이기 위한 공동 모듈을 설계할 수 있다.	①	②	③	④	⑤
	1.4 전반적인 처리 논리 구조에 예기치 못한 영향을 끼치지 않도록 공동 모듈 인터페이스의 인덱스 번호나 기능 코드를 설계할 수 있다.	①	②	③	④	⑤
타 시스템 연동설계하기	2.1 소프트웨어 아키텍처에서 경의한 타 시스템 연동 리스트 및 연동 방안을 참조하여, 타 시스템 연동 상세 설계 가이드라인을 작성할 수 있다.	①	②	③	④	⑤
	2.2 소프트웨어 아키텍처의 경의를 반영한 연동 상세 설계 가이드라인에 따라, 타 시스템 연동 상세 설계할 수 있다.	①	②	③	④	⑤
	2.3 소프트웨어 아키텍처에 따라 선정된 개발 및 운영 환경에 사용될 기술 영역별 미들웨어/솔루션에 대하여 명세를 작성할 수 있다.	①	②	③	④	⑤
	2.4 소프트웨어 아키텍처에 따른 시스템간의 연동 시, 발생할 수 있는 오류를 예측하고 이의 대응 방안에 대해 제시할 수 있다.	①	②	③	④	⑤

#### 【진단결과】

진단영역	문항 수	점수	점수÷문항 수
공통 모듈 설계하기	4		
타 시스템 연동설계하기	4		
합계	8		

▣ 자신의 점수를 문항 수로 나눈 값이 '3점' 이하에 해당하는 영역은 업무를 성공적으로 수행하는데 요구하는 능력이 부족한 것으로 교육훈련이나 개인학습을 통한 개발이 필요함

▣ 평가항목(수행준거) 앞에 "나는 ~~ 할 수 있다" 만 붙여서 학습자 스스로 자가 진단 평가서로 활용할

## 인공지능 플랫폼 기능 구현 문제해결시나리오

교육기관	휴먼교육센터	교육기간	2021년 04월 27일 ~ 2021년 05월 06일						
평가일시	2021-05-06 목요일	과정명	Python(파이썬),Java(자바)기반 AI활용 응용소프트웨어 개발자 양성과정		학생명				
교과목	인공지능 플랫폼 기능 구현			평가자	박 수 민 (인)				
능력단위명	인공지능 플랫폼 기능 구현 (2001070105_18v1)								
성취기준	매우 우수(5) (90~100점)	우수(4) (80~89점)	보통(3) (70~79점)	미흡(2) (60~69점)	매우 미흡(1) (60점 미만)				
	PASS			FAIL					
	FAIL의 경우 보충학습 후 재평가 실시 (최대 3차 평가까지 진행)								
평가문항 (수행내용)	적절한 분석 데이터를 검색하여 KoNLP와 텍스트 마이닝을 통해서 처리하여 워드 크라우드 형태로 처리할 수 있는 주제를 찾고 제거 할 텍스트를 선정 등 프로그램 제작에 필요한 사항들을 계획 시나리오를 ppt로 작성하시오								
능력단위요소	<b>평가내용</b>				매우 미흡	미흡	보통	우수	매우 우수
인공지능 학습 기능 구현하기	1.1 분석 주제에 따라 학습을 위한 데이터 생성 기능을 구현할 수 있다.				①	②	③	④	⑤
	1.2 학습이 최적화될 수 있도록 데이터를 학습데이터와 검증데이터로 용도에 맞게 구분하는 기능을 구현할 수 있다.				①	②	③	④	⑤
	1.3 학습 주제에 따라 적합한 학습 알고리즘과 초매개 변수(Hyper Parameter)를 설정하는 기능을 구현할 수 있다.				①	②	③	④	⑤
	1.4 학습데이터를 학습 알고리즘에 적용하여 학습모델을 생성하는 기능을 구현할 수 있다.				①	②	③	④	⑤
	1.5 검증데이터를 바탕으로 학습모델을 평가하는 기능을 구현할 수 있다.				①	②	③	④	⑤
인공지능 초록 기능	2.1 플랫폼 기능 설계 따라 학습된 모델을 서비스 운영 환경에 적용하는 기능을 구현할 수 있다.				①	②	③	④	⑤
	2.2 서비스 운영 환경에 적용된 학습모델을 바탕으로 추론 결과를 제공하는 기능을 구현할 수 있다.				①	②	③	④	⑤

인공지능 추론 기능 구현하기	2.1 플랫폼 기능 설계 따라 학습된 모델을 서비스 운영 환경에 적용하는 기능을 구현할 수 있다.	①	②	③	④	⑤	
	2.2 서비스 운영 환경에 적용된 학습모델을 바탕으로 추론 결과를 제공하는 기능을 구현할 수 있다.	①	②	③	④	⑤	
	2.3 사용자를 쉽게 이해시키기 위해 추론된 결과를 변환하는 기능을 구현할 수 있다.	①	②	③	④	⑤	
	2.4 추론된 결과에 대해 사용자의 환류를 반영하여 재학습하는 기능을 구현할 수 있다.	①	②	③	④	⑤	
인공지능 인지 기능 구현하기	3.1 학습데이터와 적용 알고리즘을 연결할 수 있는 분석환경을 통해 인지 모델을 구현할 수 있다.	①	②	③	④	⑤	
	3.2 인지 모델과 분석결과를 활용할 수 있는 연계모듈을 통합하여 인지 기능을 구현할 수 있다.	①	②	③	④	⑤	
	3.3 향후 개선을 위하여 모델 결과의 성공 여부를 검증하여 인지 기능을 보완할 수 있다.	①	②	③	④	⑤	
평가기준	<p style="text-align: center;">모든 능력단위 평가기준은 다음과 같음</p> <p>* 배점 기준 ( 전체 내용이 없으면 매우미흡, 위치를 맞춘 개수가 1개면 미흡, 2개면 보통, 3개면 우수, 4개면 매우우수, 총배점 50점)</p>						
과제물 제출 및 보관	제출물		보고서 파일(한글, PPT 등 문서파일)				
	평가자료 보관방법		1인당 파일 1개 보관, 출력 보관				
평가항목	<p>자료수집, 시안방향구분, 컨셉 적용 적합여부, 시안 작업 방법 계획, 개발 시안 예상, 아이디어 도출 어느 곳에서든 접근하여 설명한 컨트롤을 문제없이 사용할 수 있다.</p> <p>작성한 사용 방법을 이용해서 작성한 결과물이 문제없이 돌아가는지 확인한 후 능력단위 평가 내용을 기준으로 평가 항목에 따라 평가한다.</p>						
성취수준		점수환산	환산식: (총 득점 / 만점기준) * 50			총점 :	