



Database & SQL

김기정 (bangry313@gmail.com)

목차 (Table of Contents)

1. Database 소개
2. SQL 개요
3. SQL 기본 구문
4. Oracle 함수
5. JOIN
6. SUBQUERY

DATABASE



```
SELECT empCode, empName, empSalary  
FROM Employee  
WHERE empName in
```

```
(SELECT DISTINCT empName  
FROM population  
WHERE Country = "TH")  
AND empSalary <=  
(SELECT AVG(salary)  
FROM Salary  
WHERE gender = "M")
```



1. Database 소개

- 1.1 Database 개요
- 1.2 Database Management System
- 1.3 Oracle 11g XE 설치

1.1 Database 개요 (1/2)

✓ Database 정의

- 여러 사람들이 공유하기 위하여 통합, 관리하는 데이터의 집합(창고)이다
- 여러 응용 프로그램들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 묶음이다

✓ Database 특징

- 데이터베이스는 여러 응용프로그램들에 의해 공유된다
- 데이터베이스의 데이터들은 서로 연관성을 가지며, 중복된 데이터 없이 안전하고, 영속적으로 관리된다
- 쉽고 빠르게 저장, 검색, 수정, 삭제(Create, Read, Update, Delete) 등이 가능하다
- 예) 쇼핑몰 데이터베이스, 학사 정보 데이터베이스, 도서 정보 데이터베이스, 인사 정보 데이터베이스, 판매정보 데이터베이스 등

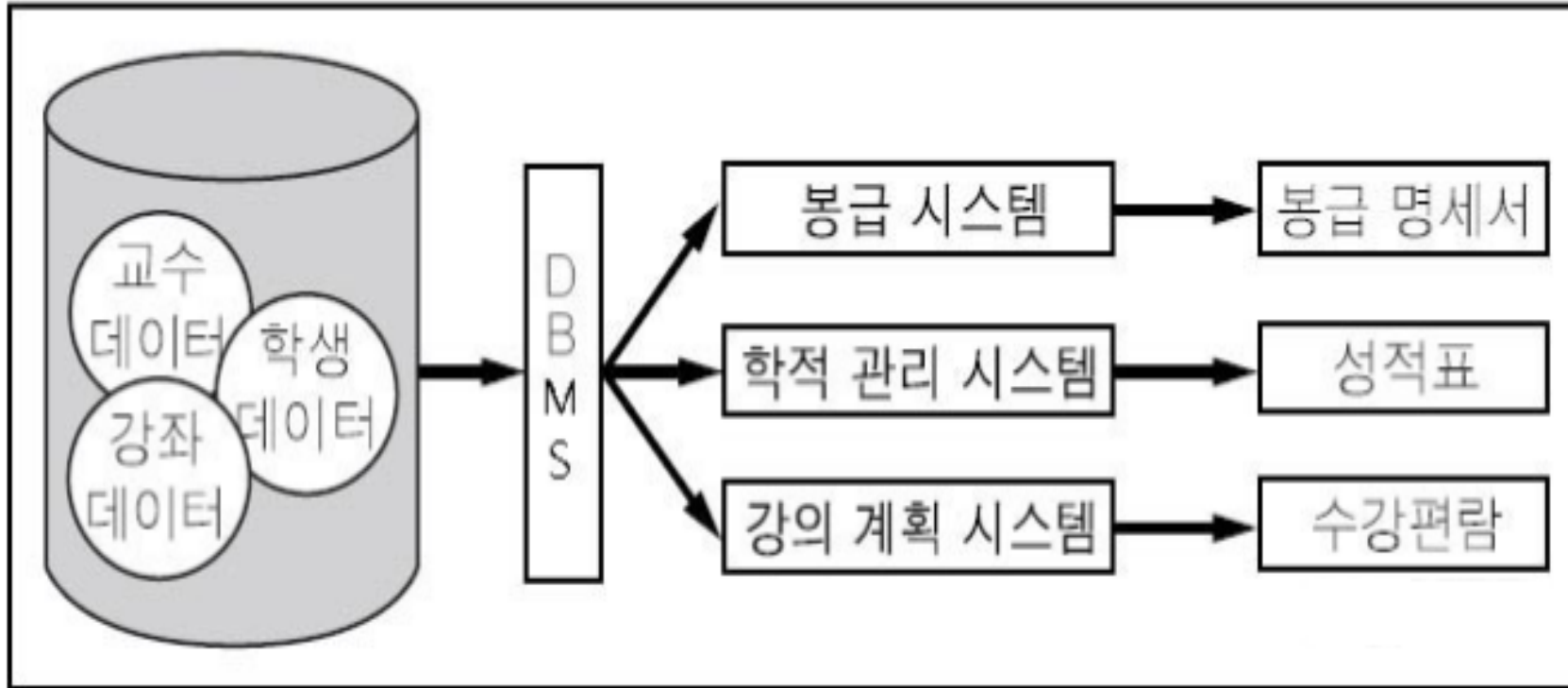
✓ Database 등장 배경

- 기존 파일 시스템의 한계
 - 파일 시스템은 중복된 데이터 저장이 빈번히 발생할 수 있으며, 대용량의 데이터를 저장하고 관리하기에는 한계가 있다
 - 여러 응용 프로그램들이 데이터를 공유하기 쉽지 않다 (일반적으로 하나의 데이터 파일은 하나의 응용 프로그램에 의해 사용)
 - 보안에 취약할 수 있다

✓ “파일시스템”의 한계를 극복하고, 대용량의 데이터를 체계적으로 저장, 관리 하기 위한 개념이 Database 이다

1.1 Database 개요 (2/2)

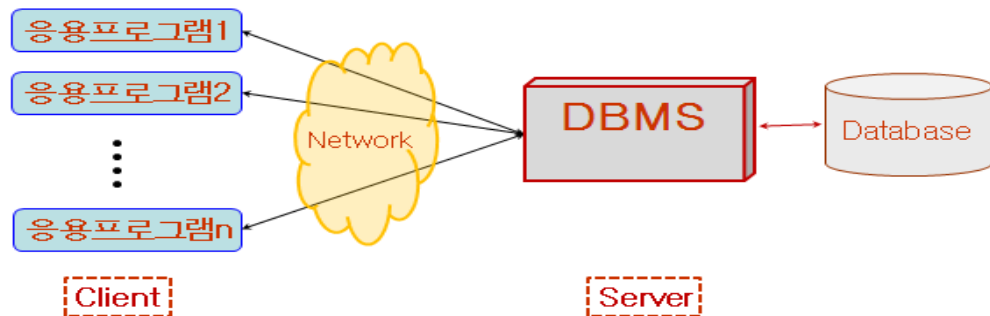
✓ Database 기반 정보 시스템 구축 사례



1.2 Database Management System (1/2)

✓ DBMS 정의

- Database를 통합 관리하는 소프트웨어를 데이터베이스 관리시스템(DBMS)이라 한다
- Client와 Database 의 중재자로서 Database를 보다 안전하고 효율적으로 관리하며, 클라이언트의 데이터 요청(저장, 검색, 수정, 삭제)에 대해 다양한 서비스를 제공하는 **TCP/IP 기반 Server** 이다



✓ DBMS 특징

- 모든 클라이언트들이 Database를 공유할 수 있게끔 관리해 준다.
- 사용자가 새로운 데이터베이스를 생성하고, 데이터베이스의 구조를 명시할 수 있게 하고, 사용자가 데이터를 효율적으로 질의 (Query)하고 수정할 수 있도록 한다
- 시스템의 고장이나 권한이 없는 사용자로부터 데이터를 안전하게 보호하며, 동시 사용자가 데이터베이스를 접근하는 것을 제어한다
- DBA(Database Administrator)는 Database의 전체적인 운영과 관리 업무 담당한다

✓ DBMS 종류

- Oracle, MS-SQL, Informix, Sybase, DB2, MySQL, MariaDB 등

1.2 Database Management System (2/2)

✓ DBMS 변천사

- 1960년대 : File System(SAM)
- 1970년대 : Network-DBMS, Hierarchical-DBMS
- 1980년대 : 관계형-DBMS
- 1990년대 : 객체지향형-DBMS
- 2000년대 : 객체관계형-DBMS

✓ 관계형 DBMS(RDBMS)

- 대표적인 DBMS로 데이터 저장 단위로 2차원 테이블(표)을 이용하며, 여러 테이블 간의 관계를 통해 데이터를 조직화 한다
- 데이터 무결성, 트랜잭션 처리 등 기본적 기능이 우수하다
- SQL(Structured Query Language)을 데이터베이스 통신 언어로 사용한다

1.3 Oracle 11g XE (1/6) – 소개

✓ 가장 대표적인 RDBMS 제품 중의 하나로 Oracle이란 이름은 회사 이름인 동시에 DBMS 제품 이름이다

✓ Oracle 종류

- Standard Edition(SE)
 - CPU 4개 이하 사용 가능
- Enterprise Edition(EE)
 - 가장 널리 사용됨
 - CPU 4개 이상, 모든 기능 사용 가능
- Express Edition(XE)
 - 데이터베이스를 처음 접하는 사용자들을 위해 쉬운 설치와 편리하게 사용할 수 있는 환경을 제공
 - CPU 1개, 1GB 메모리, 최대저장용량 4GB로 제한

1.3 Oracle 11g XE (2/6) – 설치

✓ 다운로드

- <https://www.oracle.com/database/technologies/xe-prior-release-downloads.html>

✓ 설치

Oracle Database 11g Express Edition - Install Wizard

Choose Destination Location

Select folder where setup will install files.

ORACLE
DATABASE
EXPRESS EDITION

Setup will install Oracle Database 11g Express Edition in the following folder.

To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

☒ Oracle Database 11g Express Edition

631124 K

Destination Folder

C:\oraclexe\

Browse...

Space Required on C: 631124 K

Space Available on C: 77995156 K

InstallShield

BackNextCancel

Oracle Database 11g Express Edition - Install Wizard

Specify Database Passwords

ORACLE
DATABASE
EXPRESS EDITION

Enter and confirm passwords for the database. This password will be used for both the SYS and the SYSTEM database accounts.

Enter Password

Confirm Password

InstallShield

BackNextCancel

1.3 Oracle 11g XE (3/6) – SQL*Plus를 이용한 접속

✓ SQL*Plus 실행

- SQL*Plus는 Oracle XE 설치 시 자동 설치되는 내장 Oracle Client(CLI : Command Line Interface) 이다
- [시작] – [프로그램] – [Oracle Database 11g Express Edition] – [Run SQL Command Line]

✓ Admin계정(sys 또는 system) 으로 Oracle XE 로그인



```
SQL*Plus: Release 11.2.0.2.0 Production on 목 7월 14 11:42:38 2022  
Copyright (c) 1982, 2014, Oracle. All rights reserved.  
  
SQL> connect  
Enter user-name: system  
Enter password: _
```

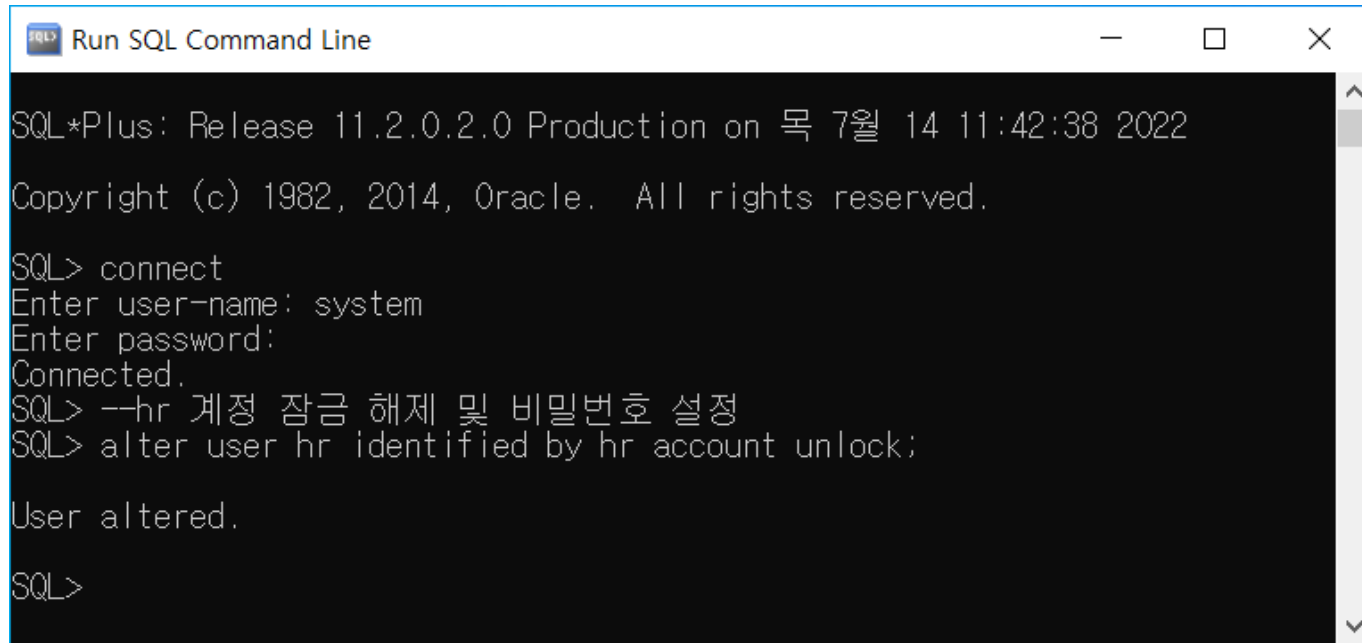
1.3 Oracle 11g XE (4/6) – SQL*Plus를 이용한 접속

✓ Oracle 사용자 계정

- **SYS와 SYSTEM**은 일반 사용자 계정을 만들거나 데이터베이스를 관리할 수 있는 막강한 DBA 권한을 가진다
 - sys : 오라클 **super 사용자** 계정이며 데이터베이스에서 발생하는 모든 문제를 처리할 수 있는 권한을 가진다
 - system : 데이터베이스를 유지보수 관리할 때 사용하는 계정으로 sys와 차이점은 데이터베이스를 생성할 수 있는 권한이 없다
- **SCOTT, HR** 등은 일반 사용자 SAMPLE 계정으로 오라클의 기본적인 SQL문을 실행할 수 있는 권한을 가진다
 - scott 은 오라클 엔진을 개발한 사람이다

✓ 샘플 계정으로 생성되어 있는 hr 계정 비밀번호 설정 및 잠금 해제

- hr 계정은 디폴트로 잠겨 있음



```
Run SQL Command Line

SQL*Plus: Release 11.2.0.2.0 Production on 목 7월 14 11:42:38 2022

Copyright (c) 1982, 2014, Oracle. All rights reserved.

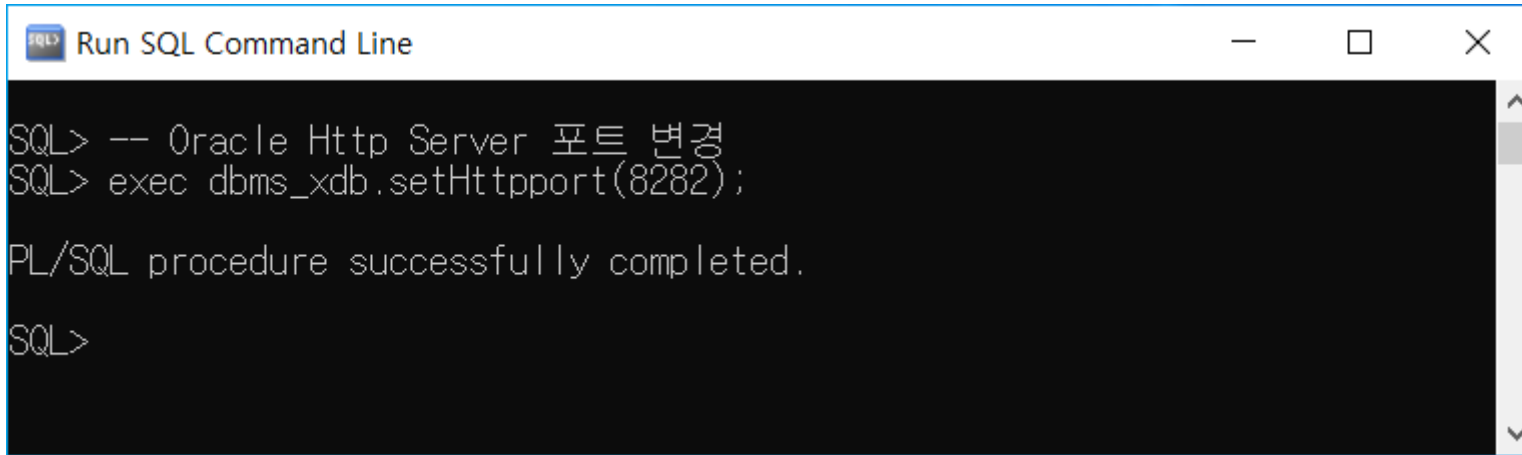
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> --hr 계정 잠금 해제 및 비밀번호 설정
SQL> alter user hr identified by hr account unlock;

User altered.

SQL>
```

1.3 Oracle 11g XE (5/6) – SQL*Plus를 이용한 접속

- ✓ HTTP Server 포트 충돌 시 프로시저를 이용한 포트 변경
- ✓ 디폴트 포트 : 8080



```
SQL> -- Oracle Http Server 포트 변경
SQL> exec dbms_xdb.setHttpport(8282);

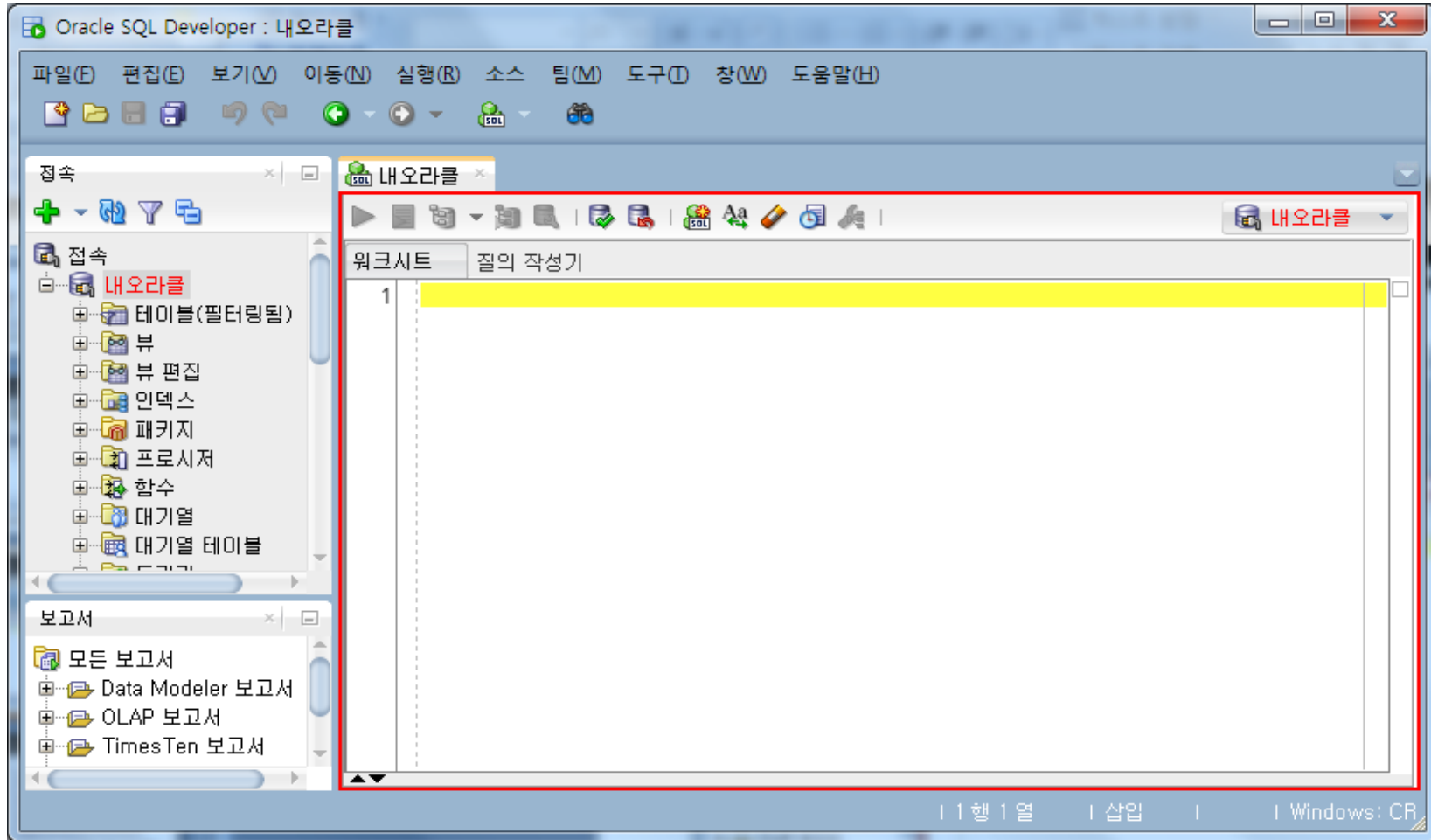
PL/SQL procedure successfully completed.

SQL>
```

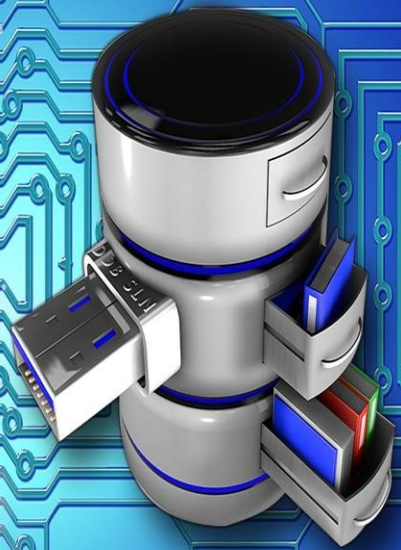

1.3 Oracle 11g XE (6/6) – 3rd Party GUI Client Tool을 이용한 접속

✓ SQL Developer 활용(Oracle Freeware)

- 다운로드 : <https://www.oracle.com/database/sqldeveloper/>



DATABASE



```
SELECT empCode, empName, empSalary
FROM Employee
WHERE empName in
    (SELECT DISTINCT empName
     FROM population
     WHERE Country = "TH")
    AND empSalary <=
    (SELECT AVG(salary)
     FROM Salary
     WHERE gender = "M")
```

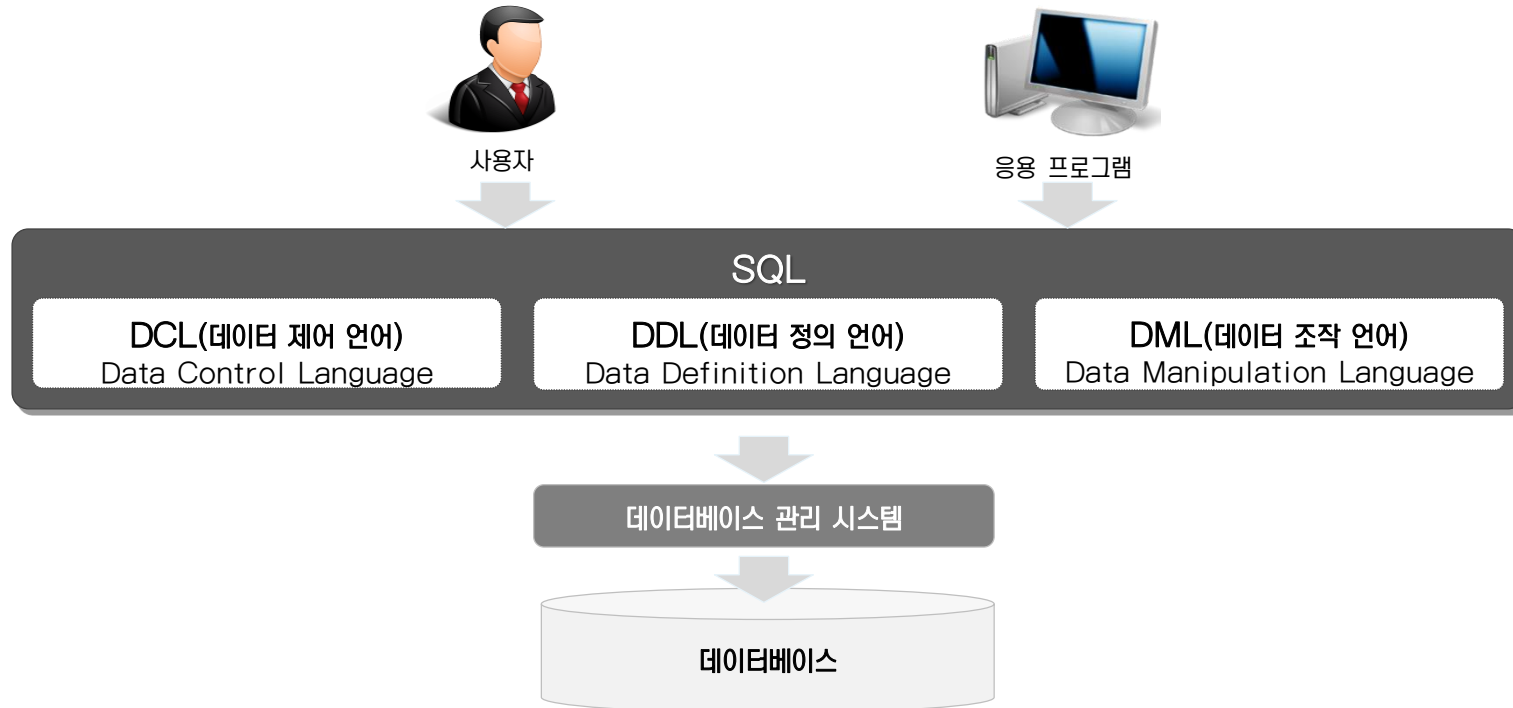


2. SQL 개요

- 2.1 SQL 개요
- 2.2 SQL 종류
- 2.3 SQL 작성 규칙
- 2.4 SQL*Plus 로그인 & SQL 실행

2.1 SQL 개요 (1/4)

- ✓ SQL는 Structured Query Language (**구조적 질의 언어**) 의 약어로서, 프로그래밍 언어의 한 종류이다
 - 무슨 데이터를 저장, 추출하고 변경할 것인가를 서술하는 비절차적 언어이다
- ✓ 관계형 데이터베이스(RDB)에 데이터를 저장(Create)하거나 조회(Read), 수정(Update), 삭제>Delete)할 수 있도록 고안된 구조적 표준 질의 언어이다
- ✓ SQL을 이용하여 사용자 또는 응용프로그램이 데이터베이스 관리 시스템에 접근할 수 있다
- ✓ SQL는 ISO(International Standardization Organization :국제표준화기구) 국제 표준 질의 언어로 채택하고 있다
 - SQL 표준에 맞춰 제작된 대부분의 DBMS에는 데이터타입과 내장 함수의 차이만 있을 뿐 SQL 기본 명령어는 동일하다



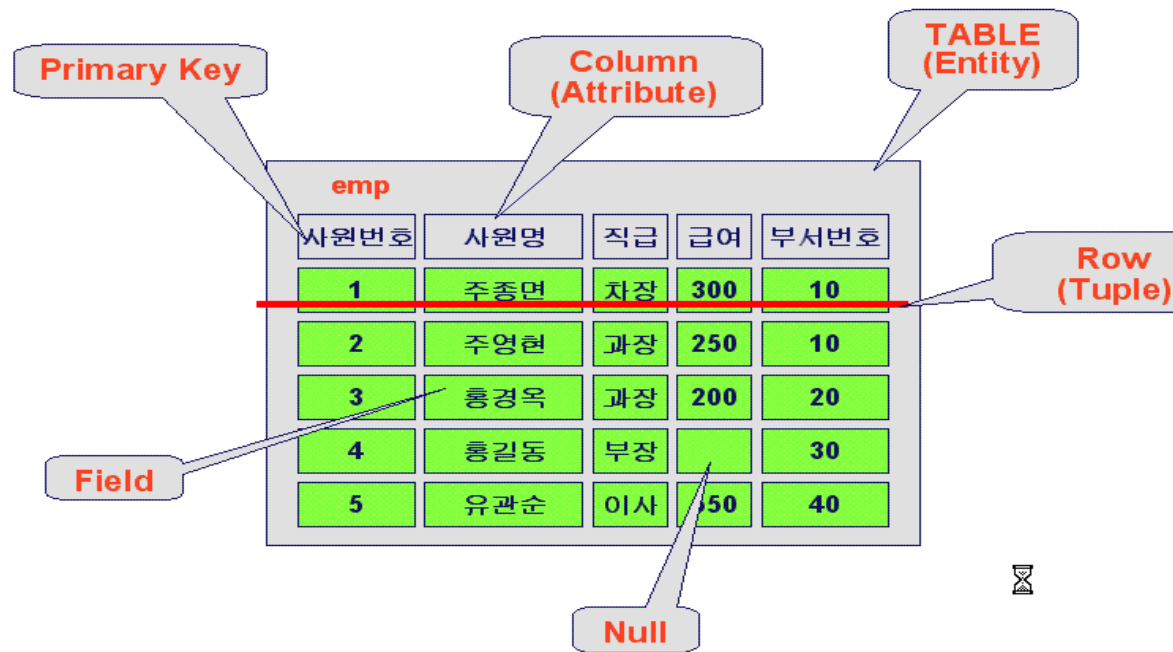
2.1 SQL 개요 (2/4) – 특징

- ✓ SQL은 관계형 데이터베이스의 기본 데이터 저장 단위인 **테이블**과 **테이블 간의 관계**를 이용한다
- ✓ RDBMS는 관계형 데이터베이스로 물리적 파일을 사용한다
 - 사용자나 애플리케이션이 파일을 직접 접근하는 것이 아니라, DBMS와의 SQL을 통신을 통해 데이터를 저장, 조회, 수정, 삭제한다
- ✓ 많은 프로그래밍 언어(C, Visual Basic, Delphi, C++, C#, Java 등)에서 SQL문을 내장(Embed)하여 데이터베이스 연동 애플리케이션을 개발할 수 있다
- ✓ SQL은 RDBMS(Oracle, MySQL, MariaDB, Sybase, Informix, DB2 등)에서 표준 질의 언어로 채택하고 있다

2.1 SQL 개요 (3/4) – 관계형 데이터베이스(RDB) 기본 용어 (1/2)

✓ 테이블 (Table)

- 관계형 데이터베이스의 기본 데이터 저장 단위로 2차원 구조의 테이블을 사용한다
 - 시스템에서 영속적으로 저장되길 원하는 현실세계의 엔티티(Entity)를 표현한다
 - 예) 현실세계의 엔티티(직원, 부서, 제품, 주문, 재고 등)를 테이블로 표현



- 테이블(Table)은 여러 개의 행(Row)과 열(Column)의 2차원 구조로 행은 파일 시스템의 '레코드'에 비유 된다
- 컬럼(Column)는 속성(사원번호, 사원명, 직급, 급여 등)을 나타낸다
- Java 언어의 Object vs Entity, Class vs Table, Instance vs Row, Attribute vs Column 으로 비유 될 수 있다

2.1 SQL 개요 (4/4) – 관계형 데이터베이스(RDB) 기본 용어 (2/2)

✓ 관계 (Relation)

- 데이터베이스는 여러 테이블 간의 관계로 구성된다
 - 예) 고객 테이블과 주문 테이블 간의 관계, 주문 테이블과 제품 테이블 간의 관계 등
- 기본키(Primary Key)와 외래키(Foreign Key)를 이용하여 테이블 간의 관계를 표현한다



2.2 SQL 종류

✓ SQL 구문은 DQL, DML, DDL, TCL, DCL로 구분한다

구분	형식	비고
DQL (Data Query Language)	SELECT column-1, column-2, FROM table명 WHERE 조건절 ;	검색시 사용
DML (Data Manipulation Language)	UPDATE table명; INSERT INTO table명; DELETE table명;	변경시 사용
DDL (Data Definition Language)	CREATE TABLE table명; DROP TABLE table명; ALTER TABLE table명;	Object의 생성과변경 시
TCL (Transaction Control Language)	COMMIT; ROLLBACK; SAVEPOINT;	Transaction 종료 및 취소
DCL (Data Control Language)	GRANT; REVOKE;	권한 부여 및 취소

2.3 SQL 작성 규칙

- ✓ SQL은 대소문자를 구별하지 않지만 키워드(명령어)는 대문자, 식별자(테이블명, 컬럼명 등)는 소문자 사용을 권장한다
- ✓ 하나의 SQL 명령문을 여러 줄에 걸쳐 입력할 수 있으며, 명령문 종료 시에는 세미콜론(;)을 사용한다
- ✓ 각 절은 되도록 다른 행에 작성하여 가독성이 좋고, 수정이 용이하도록 작성한다
- ✓ 문자 데이터는 작은 따옴표(' ')로 묶어 사용한다
- ✓ 단일 행 주석은 '--', 다중 행 주석은 '/* ~ */' 을 사용한다

```
-- Single-line Comment
/* Multi-line Comment */
SELECT  department_id,
        last_name,
        first_name
FROM    employees
WHERE   department_id = 100;
```


2.4 SQL*Plus (1/7) – 로그인 & SQL 실행 (1/2)



```
SQL> Run SQL Command Line

SQL*Plus: Release 11.2.0.2.0 Production on 금 7월 15 10:58:16 2022

Copyright (c) 1982, 2014, Oracle. All rights reserved.

SQL> conn hr/hr
Connected.
SQL> -- 사용자 테이블 목록 조회
SQL> SELECT table_name
       2 FROM user_tables;

TABLE_NAME
-----
REGIONS
LOCATIONS
DEPARTMENTS
JOBS
EMPLOYEES
JOB_HISTORY
COUNTRIES
```

2.4 SQL*Plus (2/7) – 로그인 & SQL 실행 (2/2)



```
SQL> -- employees 테이블로부터 사원정보 조회
SQL> SELECT employee_id, first_name, salary
2  FROM employees;
```

EMPLOYEE_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	Lex	17000
103	Alexander	9000
104	Bruce	6000
105	David	4800
106	Valli	4800
107	Diana	4200
108	Nancy	12008
109	Daniel	9000
110	John	8200

EMPLOYEE_ID	FIRST_NAME	SALARY
111	Ismael	7700
112	Jose Manuel	7800

2.4 SQL*Plus (3/7) – 명령어 (1/5)

구 분	명령어	내 용
파일 명령어	EDIT {파일명} SAVE {파일명} START {파일명} = @ GET {파일명} SPOOL {파일명} SPOOL OFF HOST EXIT CONNECT {사용자명/암호}	파일 E는 버퍼의 내용을 편집기로 불러온다 버퍼의 내용을 파일에 저장한다. 저장된 SQL 스크립트를 실행한다. 파일의 내용을 버퍼로 읽어온다. 조화결과를 화면에 저장한다. 운영체제(Shell)로 빠져나간다. 운영체제(OS Prompt)로 빠져나간다. 다른 사용자로 접속할 때 사용한다.
편집 명령어	A {문자 스트링} C L I DEL n N(숫자) CLEAR BUFFER	현재 버퍼의 끝에 새로운 문자 스트링을 추가 현재 행의 문자열을 치환한다. 버퍼의 전체 List를 출력한다 버퍼에 새로운 행을 추가한다. 현재 행을 삭제한다. 현재 행을 출력한다. 버퍼의 전체 내용을 삭제한다.
실행 명령어	START {파일명} @ {파일명} RUN {파일명} /	SQL 스크립트를 실행 할 때 SQL 스크립트를 실행 할 때 버퍼의 내용을 실행할 때 버퍼의 내용을 실행할 때

2.4 SQL*Plus (4/7) – 명령어 (2/5)

구 분	명령어	내 용
환경 명령어	SET ECHO {off on} FEED[BACK] {n off on} HEAD[ING] LINE[SIZE] {80 n} PAGE[SIZE] {24 n} PAU[SE] {on off} SQLPREFIX {# c} NULL {text} SPACE {1 n} UNDERLINE {기호 on off} WRAP {on off}	SQL 스크립트를 실행 할 때 명령어의 출력여부 조화결과 메시지 출력여부 컬럼의 Head 출력여부 출력될 한 라인의 길이 출력 페이지 당 라인 수 화면 이동제어(한 페이지씩 보고 싶을 때) SQL 명령어 사이에 SQL*Plus 명령어를 사용할 때 NULL값을 대체할 text 정보를 설정할 때 출력된 컬럼 간의 여유공간을 설정할 때 컬럼의 heading 밑에 사용될 UnderLine을 설정 컬럼들이 지정된 LineSize를 초과할 때 출력여부
형식 명령어	COLUMN TTITLE BTITLE BREAK	컬럼의 FORMAT을 변경할 때 보고서의 제목을 설정할 때 보고서의 꼬리말을 설정할 때 컬럼 또는 행의 값이 바뀔 때 마다 새로운 보고서 format을 설정할 때

2.4 SQL*Plus (5/7) – 명령어 (3/5)



The screenshot shows a SQL*Plus window titled "SQL 명령줄 실행". It displays the execution of an SQL query and the saving of the command into a file.

```
SQL> SELECT first_name, salary
2  FROM employees
3  WHERE last_name LIKE 'K%';
```


FIRST_NAME	SALARY
Payam	7900
Alexander	3100
Janette	10000
Steven	24000
Neena	17000
Sundita	6100

6 개의 행이 선택되었습니다.

```
SQL> SAVE C:\WExample.sql
file C:\WExample.sql(이)가 생성되었습니다
SQL> GET C:\WExample.sql
1  SELECT first_name, salary
2  FROM employees
3* WHERE last_name LIKE 'K%'
SQL> run
1  SELECT first_name, salary
2  FROM employees
3* WHERE last_name LIKE 'K%'
```

FIRST_NAME	SALARY
Payam	7900
Alexander	3100
Janette	10000
Steven	24000
Neena	17000
Sundita	6100

2.4 SQL*Plus (6/7) – 명령어 (4/5)



The screenshot shows a Windows-style window titled "SQL 명령줄 실행" (SQL Command Line Execution). The window has a blue background and a yellow text color. It displays two SQL queries and their results. The first query is "start C:\Example.sql" and the second is "@ C:\Example.sql". Both queries result in a table of employee names and salaries. The table has two columns: "FIRST_NAME" and "SALARY". The data rows are: Payam (7900), Alexander (3100), Janette (10000), Steven (24000), Neena (17000), and Sundita (6100). Below each table, a message states "6 개의 행이 선택되었습니다." (6 rows selected).

```
SQL> start C:\Example.sql

FIRST_NAME                SALARY
-----
Payam                      7900
Alexander                  3100
Janette                    10000
Steven                     24000
Neena                      17000
Sundita                     6100

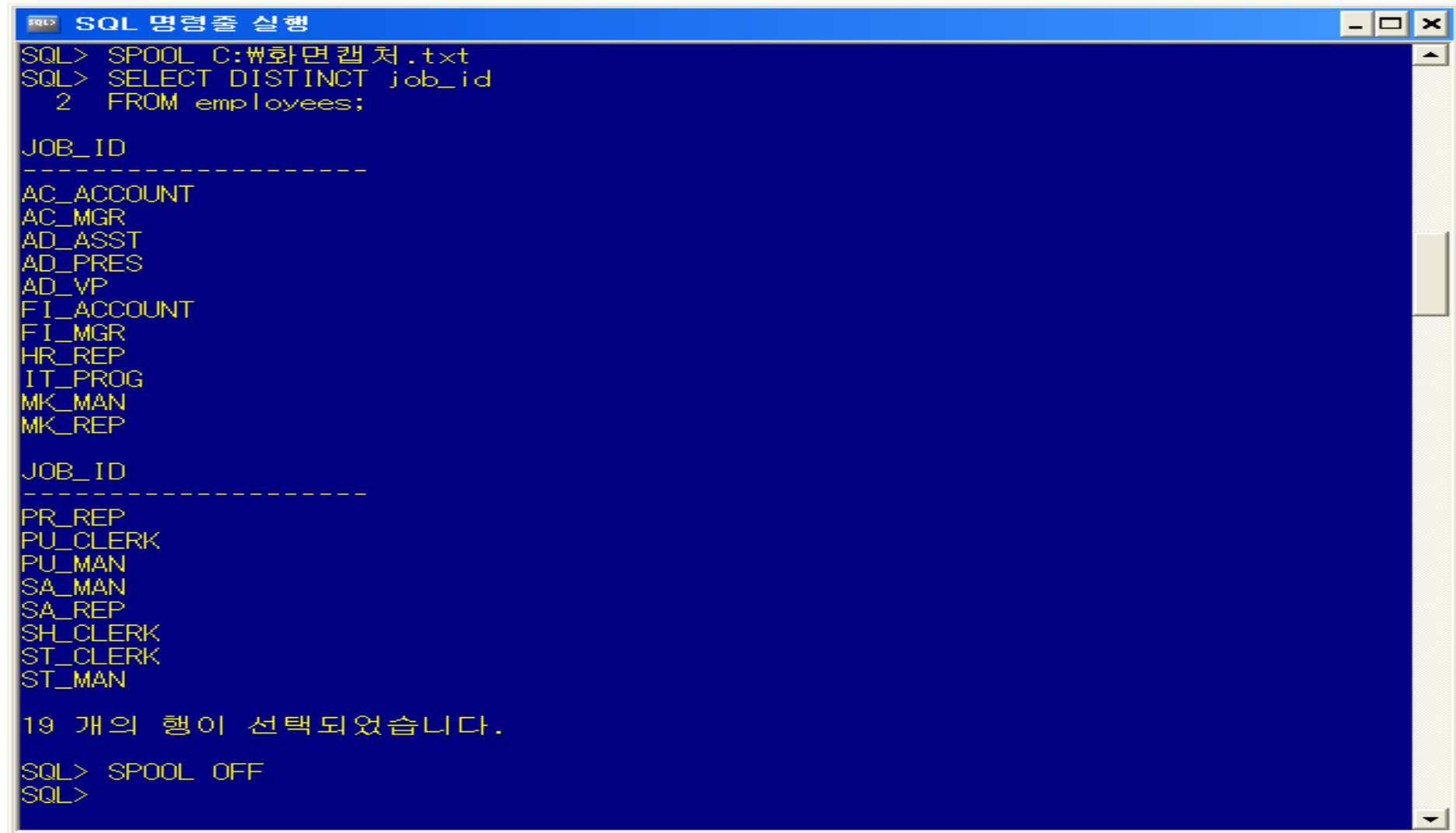
6 개의 행이 선택되었습니다.

SQL> @ C:\Example.sql

FIRST_NAME                SALARY
-----
Payam                      7900
Alexander                  3100
Janette                    10000
Steven                     24000
Neena                      17000
Sundita                     6100

6 개의 행이 선택되었습니다.
```

2.4 SQL*Plus (7/7) – 명령어 (5/5)



The screenshot shows a SQL*Plus window titled "SQL 명령줄 실행". The user has entered the command `SPOOL C:\₩화면캡처.txt` to save the output to a file. Then, they executed `SELECT DISTINCT job_id FROM employees;`. The output lists 19 distinct job IDs: AC_ACCOUNT, AC_MGR, AD_ASST, AD PRES, AD_VP, FI_ACCOUNT, FI_MGR, HR_REP, IT_PROG, MK_MAN, MK_REP, PR_REP, PU_CLERK, PU_MAN, SA_MAN, SA_REP, SH_CLERK, ST_CLERK, and ST_MAN. A message at the bottom states "19 개의 행이 선택되었습니다." (19 rows selected). Finally, the user entered `SPOOL OFF` to stop saving output to the file.

```
SQL> SPOOL C:\₩화면캡처.txt
SQL> SELECT DISTINCT job_id
      2  FROM employees;

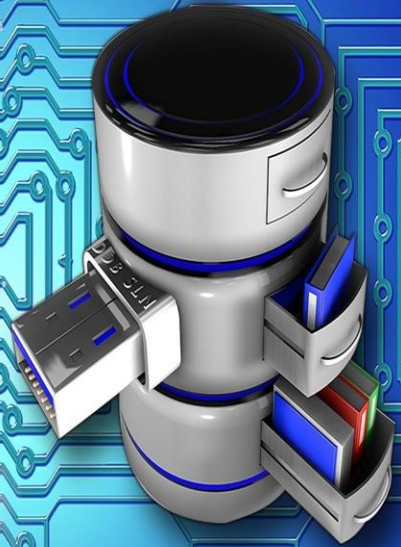
JOB_ID
-----
AC_ACCOUNT
AC_MGR
AD_ASST
AD PRES
AD_VP
FI_ACCOUNT
FI_MGR
HR_REP
IT_PROG
MK_MAN
MK_REP

JOB_ID
-----
PR_REP
PU_CLERK
PU_MAN
SA_MAN
SA_REP
SH_CLERK
ST_CLERK
ST_MAN

19 개의 행이 선택되었습니다.

SQL> SPOOL OFF
SQL>
```

DATABASE



```
SELECT empCode, empName, empSalary
FROM Employee
WHERE empName in
    (SELECT DISTINCT empName
     FROM population
     WHERE Country = "TH")
    AND empSalary <=
    (SELECT AVG(salary)
     FROM Salary
     WHERE gender = "M")
```



3. SQL 기본 구문

- 3.1 Table
- 3.2 Data Type
- 3.3 DQL(Data Query Language)
- 3.4 DDL(Data Definition Language)
- 3.5 DML(Data Manipulation Language)
- 3.6 TCL(Transaction Control Language)
- 3.7 DCL(Data Control Language)

3.1 Table

✓ 테이블은 관계형 데이터베이스의 기본적인 데이터 저장 단위이다

- 행과 열로 구성된 2차원 행렬 구조

✓ 각 테이블은 기본키와 외래키를 가질 수 있다

- 기본키(Primary Key)는 테이블의 각 행을 고유하게 식별하는 열 또는 열의 조합을 의미하며, Null이 될 수 없다
- 외래키(Foreign Key)는 테이블 간의 논리적 관계를 유지하기 위해 참조하는 테이블의 기본 키를 바라본다

테이블 생성시 고려 사항

- 테이블 생성 권한을 확인한다
- 동일 계정 내에서 테이블 이름은 중복될 수 없다
- 테이블 이름에는 예약어를 사용할 수 없다
- 테이블 이름과 컬럼은 공백을 포함할 수 없다
- 테이블 이름과 컬럼은 알파벳으로 시작해야 하며, A~Z, a~z, 0~9, _ , \$, #으로 구성한다
- 한 테이블 안에서 컬럼 이름은 중복될 수 없다

3.2 Data Type (1/2)

- ✓ 테이블의 각 컬럼은 숫자, 문자, 날짜 등과 같은 데이터 타입을 가진다
- ✓ 컬럼의 데이터 타입은 저장할 데이터의 성격과 저장 범위를 고려하여 결정한다
- ✓ 데이터 타입에 따라 데이터를 다루는 로직을 간소화할 수 있다
- ✓ 데이터 타입의 종류나 유효 범위는 DBMS에 따라 차이가 있을 수 있다

숫자	문자	날짜	대용량 데이터
NUMBER INT INTEGER SMALLINT BIGINT FLOAT	CHAR VARCHAR VARCHAR2	DATE DATETIME TIMESTAMP	CLOB BLOB TEXT

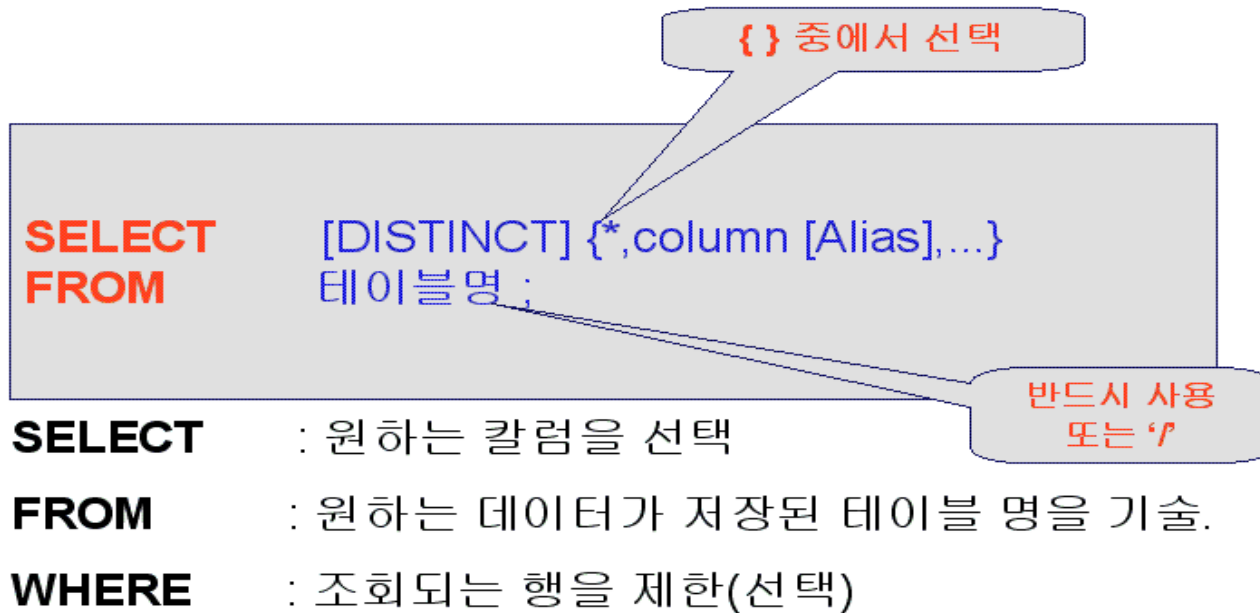
3.2 Data Type (2/2)

✓ Oracle Data Type

데이터 타입	설 명
CHAR(size)	size 바이트 크기의 고정 길이 문자 데이터 타입 최소크기: 1바이트, 최대크기: 2,000바이트
VARCHAR2(size)	size 바이트 크기의 가변 길이 문자 데이터 타입 최소크기: 1바이트, 최대크기: 4,000바이트
NUMBER(precision, scale)	숫자를 저장하기 위한 데이터 타입 소수점 포함 전체 precision자리 중 소수점 이하 s자리(p:1~38, s:-84~127) s를 생략한 채 p만 지정하면 소수점 이하는 반올림되어 정수가 저장되며, p, s 모두 생략하면 입력한 값 만큼만 공간이 할당
DATE	날짜 및 시간 데이터를 저장하기 위한 데이터 타입(7Byte)
LONG	가변 길이 문자 데이터 타입(1~2GB)
CLOB	대용량의 텍스트 데이터 저장을 위한 데이터 타입(1~4GB)
BLOB	대용량의 이진 데이터 저장을 위한 데이터 타입(1~4GB)

3.3 DQL(Data Query Language) (1/18) – SELECT 문

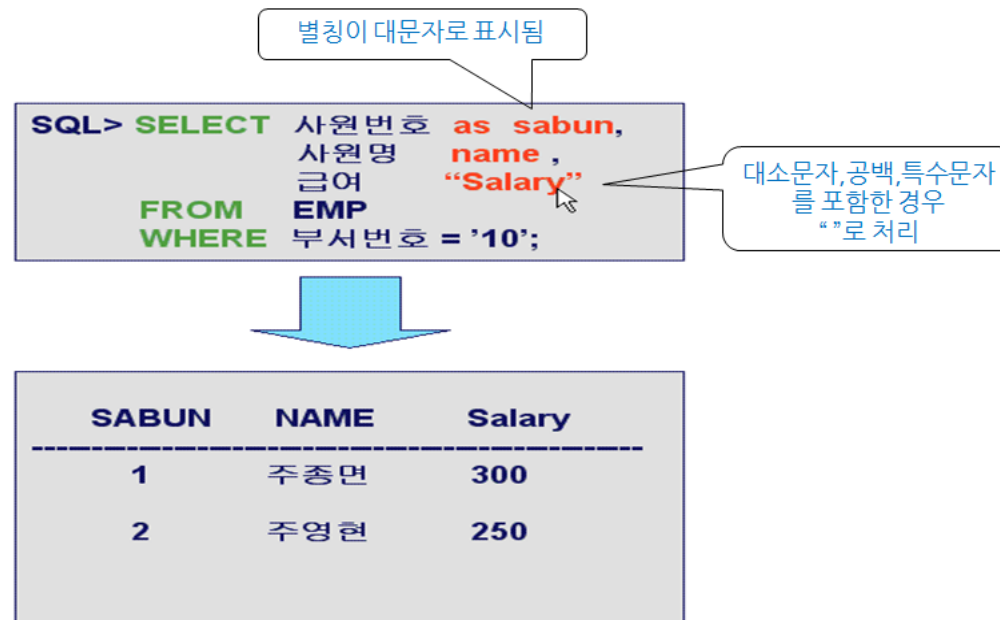
✓ 테이블에 저장되어 있는 데이터를 조회하는데 사용하는 구문이다



```
SELECT *  
FROM employees;
```

3.3 DQL(Data Query Language) (2/18) – Alias(별칭)

- ✓ 필요에 따라 검색 컬럼명에 대해 별칭(ALIAS)을 부여할 수 있다
- ✓ 컬럼 이름 다음에 공백을 두고 AS를 이용하여 별칭을 부여한다(AS는 생략 가능)
- ✓ 별칭에 특수문자나 공백이 있을 때는 " "로 묶어야 한다
- ✓ ORDER BY 절이나 Inline View에서 유용하게 사용할 수 있다



```
SELECT employee_id, salary, salary * 12 "연 봉"  
FROM employees;
```

3.3 DQL(Data Query Language) (3/18) – DISTINCT

✓ 중복 DATA 제거

- 유일한 값을 추출하기 위해 내부적인 정렬을 수행한다
- DISTINCT 바로 다음에 사용된 열에만 적용되는 것이 아니라 조회하는 모든 열에 대해 적용된다
- 특정 컬럼만 묶어서 표현할 때는 GROUP BY를 권장한다

EMP

사원번호	사원명	직급	급여	부서번호
1	주종면	차장	300	10
2	주영현	과장	250	10
3	홍경옥	과장	200	20
4	홍길동	부장	350	
5	유관순	이사	550	40

SQL> **SELECT DISTINCT** 직급
FROM EMP;

직급

과장
부장
이사
차장

SQL> **SELECT ALL** 직급
FROM EMP;

```
SELECT DISTINCT job_id
FROM employees;
```

3.3 DQL(Data Query Language) (4/18) – WHERE 절 (1/8)

- ✓ 검색 조건을 제시하여 테이블에서 조건을 만족하는 행 만을 검색 할 때 사용한다 (행 필터링)
 - 예) 사원 테이블에서 사원번호가 1004인 사원정보 검색

```
SELECT [DISTINCT] {*,column [Alias],...}  
FROM 테이블명  
WHERE [칼럼명1] [연산자] [값1]  
      [AND | OR]  
      [칼럼명2] [연산자] [값2]  
      ....;
```

★ 연산자의 종류

산술 연산자	(), * , / , + , - 순으로 우선순위가 결정됨	
비교 연산자	=, <>, !=, <, > 등	
논리 연산자	NOT, AND, OR	
SQL 연산자	IN, BEWTEEN, IS NULL, LIKE, EXISTS 등	
결합 연산자	(스트링 값을 결합할 때 사용)	
집합 연산자	UNION ALL, UNION, INTERSECT, MINUS	

3.3 DQL(Data Query Language) (5/18) – WHERE 절 (2/8)

✓ 산술 연산자

- 연산자는 where 절에서 조건 검색 시 주로 사용하지만 컬럼에도 사용할 수 있다

종 류	예 제
+	SELECT empno, ename, sal + comm FROM emp;
-	SELECT empno, ename, sal - 1500 FROM emp;
*	SELECT empno, ename, sal * 1.1 FROM emp;
/	SELECT empno, ename, sal / 2 FROM emp;

- 우선순위 (* → / → + → -)
 - 곱하기, 나누기는 더하기, 빼기 보다 우선순위가 빠릅니다.
 - 괄호를 사용하면 우선순위가 임의로 바뀝니다.
 - 같은 우선 순위인 경우에는 좌측에서 우측 순으로 빠릅니다

3.3 DQL(Data Query Language) (6/18) – WHERE 절 (3/8)

✓ 비교 연산자

종 류	예 제
=	SELECT * FROM emp WHERE sal = 500;
!=, ^=, <>	SELECT * FROM emp WHERE sal != 500;
>, <	SELECT * FROM emp WHERE sal > 1500; SELECT * FROM emp WHERE sal < 1500;
>=, <=	SELECT * FROM emp WHERE sal >= 1500; SELECT * FROM emp WHERE sal <= 1500;

- WHERE 절에서 사용하는 데이터 타입은 문자, 숫자, 날짜 등 다양한 데이터 타입이 사용할 수 있다
- 문자와 날짜의 경우 반드시 작은따옴표(' ')로 묶어 표현해야 한다

3.3 DQL(Data Query Language) (7/18) – WHERE 절 (4/8)

✓ 비교 연산자

```
SELECT employee_id, last_name, salary
FROM employees
WHERE employee_id = 100;
--WHERE salary >= 5000;
```

```
SELECT employee_id, last_name, hire_date
FROM employees
WHERE hire_date < '2002/01/01';
```

```
SELECT employee_id, last_name, hire_date
FROM employees
WHERE hire_date >= '2002/01/01'
AND hire_date <= '2002/12/31';
```

3.3 DQL(Data Query Language) (8/18) – WHERE 절 (5/8)

✓ 논리 연산자

- 조건을 여러 개 조합해서 결과를 얻어야 할 경우에 조건을 연결해 주는 역할

종 류	예 제
AND	두개의 조건 모두를 만족해야 참인 경우 SELECT * FROM emp WHERE deptno = 10 AND sal < 1500;
OR	두개의 조건 중 하나를 만족하면 참인 경우 SELECT * FROM emp WHERE deptno = 10 OR sal < 1500;
NOT	조건을 만족하지 못하면 참인 경우 SELECT * FROM emp NOT(salary = 10000)

3.3 DQL(Data Query Language) (9/18) – WHERE 절 (6/8)

✓ SQL 연산자

- 조건을 여러 개 조합해서 결과를 얻어야 할 경우에 조건을 연결해 주는 역할

종 류	예 제
IN	여러 개의 조건 값 중 하나 만 만족하면 참인 경우 SELECT * FROM emp WHERE job IN ('CLERK', 'ANALYST');
LIKE	조건 값을 명확히 알지 못하는 경우 SELECT * FROM emp WHERE ename LIKE 'F%'; SELECT * FROM emp WHERE ename LIKE 'F_%' ESCAPE '\';
BETWEEN A AND B	범위를 가진 조건 값으로 검색해야 하는 경우 SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000;
IS NULL	NULL 값을 가진 컬럼을 검색하는 경우 SELECT * FROM emp WHERE comm IS NULL;

3.3 DQL(Data Query Language) (10/18) – WHERE 절 (7/8)

✓ SQL 연산자

```
SELECT employee_id, last_name, job_id
FROM employees
WHERE job_id in('SA_MAN', 'IT_PROG');
--WHERE job_id = 'SA_MAN' OR job_id = 'IT_PROG';
```

```
SELECT employee_id, last_name, hire_date
FROM employees
WHERE hire_date BETWEEN '1995/01/01' AND '1995/12/31'
--WHERE hire_date >= '1995/01/01' AND hire_date <= '1995/12/31';
--WHERE hire_date >= '2000/01/01';
```

3.3 DQL(Data Query Language) (11/18) – WHERE 절 (8/8)

✓ SQL 연산자

```
SELECT employee_id, last_name, job_id
FROM employees
WHERE last_name LIKE 'K%';    -- % : 0개이상의 문자
--WHERE last_name LIKE '%K%';
--WHERE last_name LIKE '%K';
--WHERE last_name LIKE 'K__'; -- _ : 임의의 한 문자
--WHERE last_name LIKE '_e%';
--WHERE job_id LIKE 'IT\_%' ESCAPE '\;
```

```
SELECT employee_id, first_name, last_name, job_id
FROM employees
WHERE commission_pct IS NOT NULL;
```

```
-- 결합연산자(사원의 FULL NAME을 1개의 문자열로 연결하여 출력)
SELECT first_name || ' ' || last_name "Full Name"
FROM employees
```

3.3 DQL(Data Query Language) (12/18) – ORDER BY 절

- ✓ 특정 컬럼(별칭)을 기준으로 오름 또는 내림차순 정렬하여 검색한다
- ✓ ORDER BY절은 SELECT 문에서 가장 마지막에 실행된다

SELECT
FROM
ORDER BY

[DISTINCT] {*,column [Alias],...}
테이블명
[칼럼명1], [칼럼명2], [ASC/DESC];

EMP				
사원번호	사원명	직급	급여	부서번호
5	유관순	이사	550	40
2	주영현	과장	250	10
1	주종면	차장	300	10
4	홍길동	부장	350	
3	홍경옥	과장	200	20

SQL> **SELECT** 사원번호, 사원명
FROM EMP
ORDER BY 사원명 **ASC**;

사원번호	사원명

1	주종면
2	주영현
3	홍경옥
4	홍길동
5	유관순

3.3 DQL(Data Query Language) (13/18) – GROUP BY 절

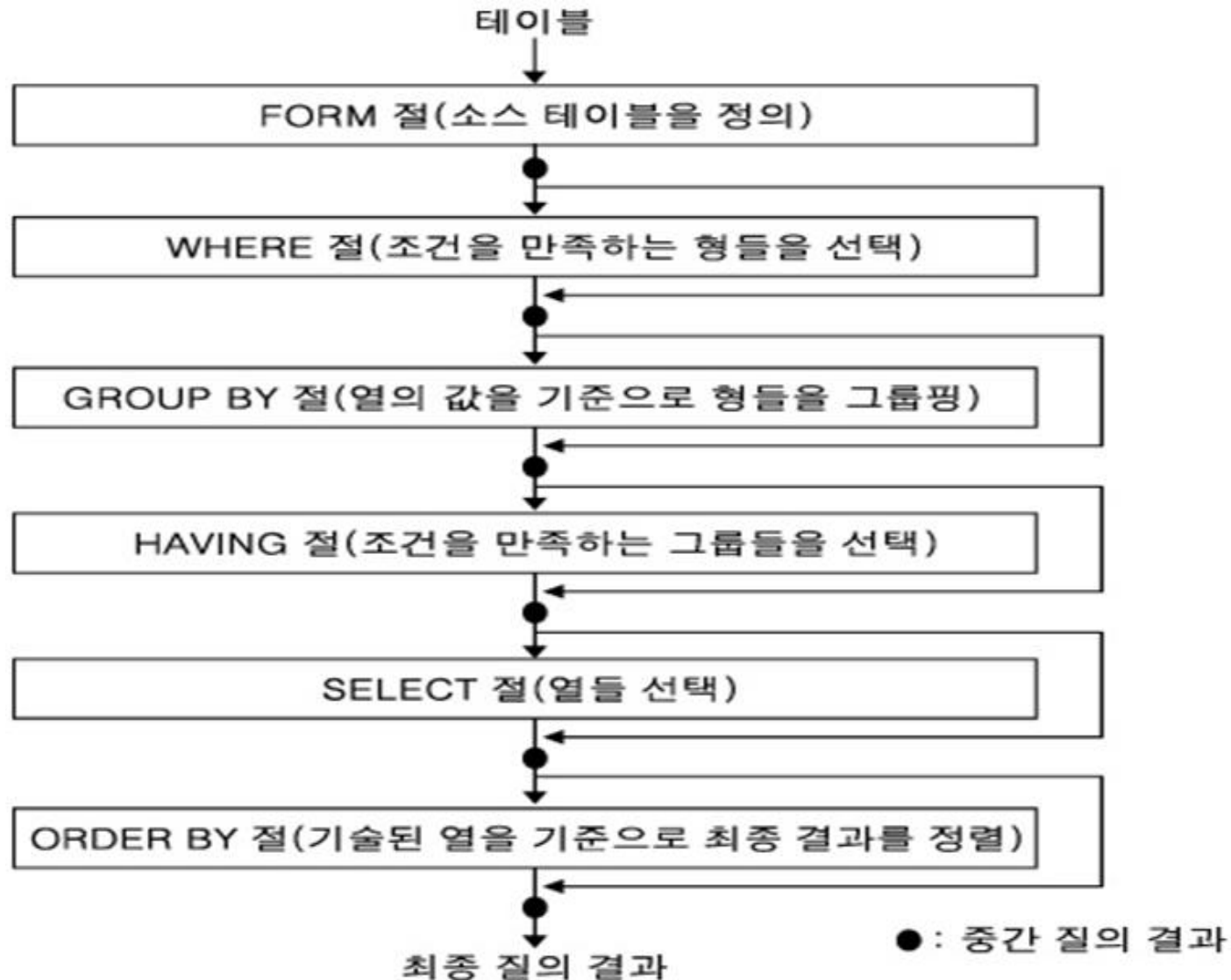
✓ 특정 컬럼을 기준으로 그룹화 하기 위해 사용한다

✓ HAVING 절

- 조건을 만족하는 그룹을 선택한다 (그룹 필터링)

```
-- 부서별 사원수 검색하기
SELECT department_id, count(department_id)
FROM employees
GROUP BY department_id
--HAVING department_id IS NOT NULL
ORDER BY department_id;
```

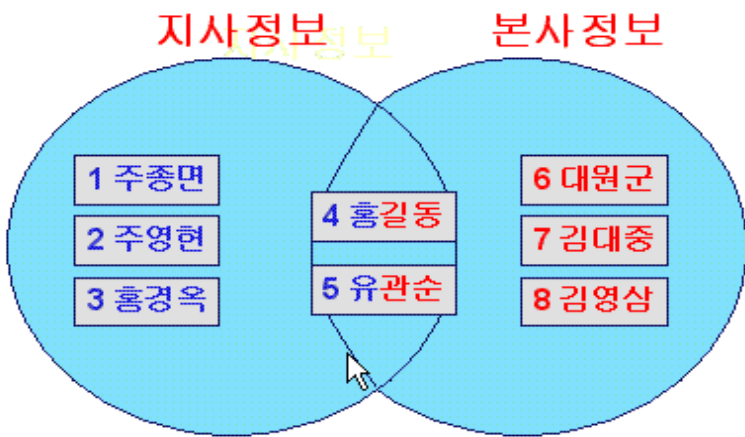
3.3 DQL(Data Query Language) (14/18) – SELECT 문 실행 순서



3.3 DQL(Data Query Language) (15/18) – UNION 집합 연산자

- 첫번째 Query와 두번째 Query의 중복된 정보는 한번 만 보여줌
(열의 개수와 타입이 동일해야 합니다)

```
SELECT * FROM jisa UNION SELECT * FROM bonsa;
```



사원번호	사원명
1	주종면
2	주영현
3	홍경옥
4	홍길동
5	유관순
6	대원군
7	김대중
8	김영삼

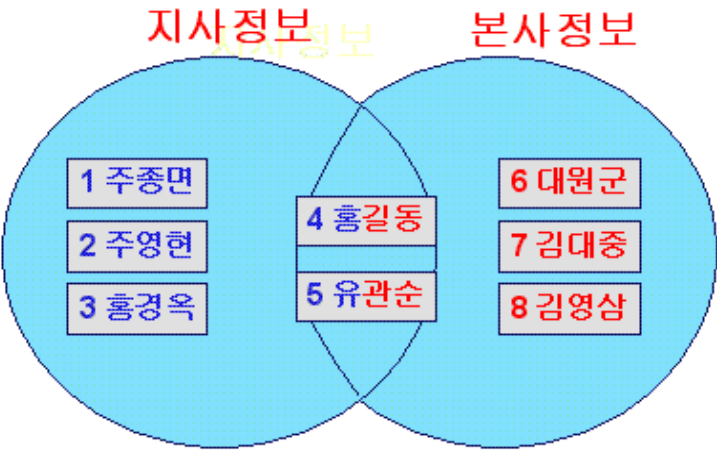
```
-- UNION 실습을 위한 테이블 복사  
CREATE TABLE emp AS  
SELECT *  
FROM employees
```

```
SELECT first_name  
FROM employees  
UNION  
SELECT first_name  
FROM emp
```


3.3 DQL(Data Query Language) (16/18) – UNION ALL 집합 연산자

첫번째 Query와 두번째 Query에 있는 모든 데이터를 보여줌

```
SELECT * FROM jisa UNION ALL SELECT * FROM bonsa;
```



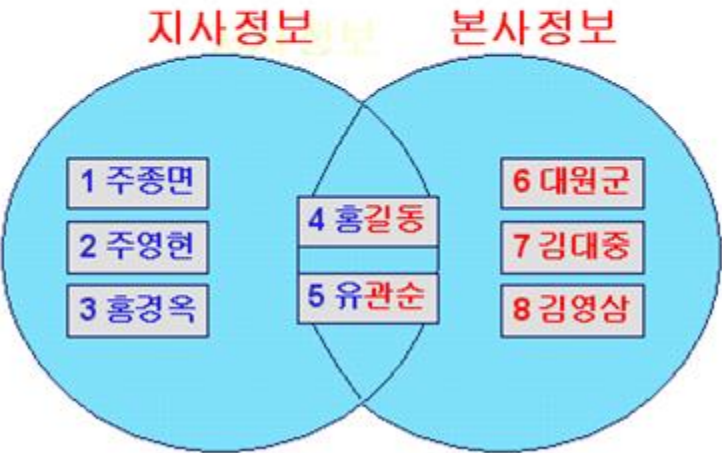
사원번호	사원명
1	주종면
2	주영현
3	홍경옥
4	홍길동
5	유관순
4	홍길동
5	유관순
6	대원군
7	김대중
8	김영삼

```
SELECT first_name FROM employees
UNION ALL
SELECT first_name FROM emp
```

3.3 DQL(Data Query Language) (17/18) – MINUS 집합 연산자

. 두번째 Query에는 없고 첫번째 Query에 만 있는 데이터를 보여줌

```
SELECT * FROM jisa MINUS SELECT * FROM bonsa;
```



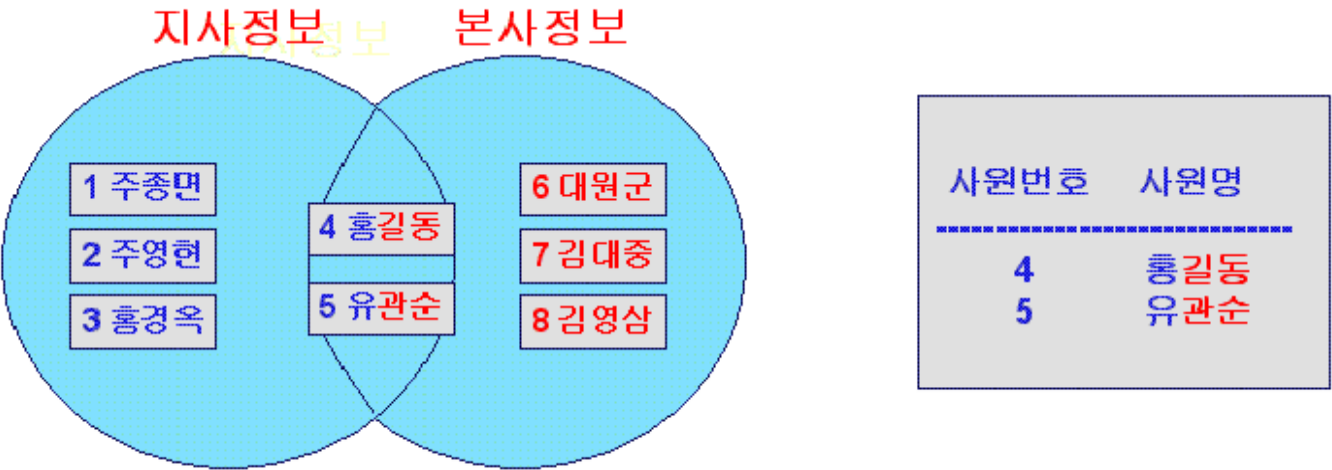
사원번호	사원명
1	주종면
2	주영현
3	홍경옥

```
SELECT first_name FROM employees
MINUS
SELECT first_name FROM emp
```

3.3 DQL(Data Query Language) (18/18) – INTERSECT 집합 연산자

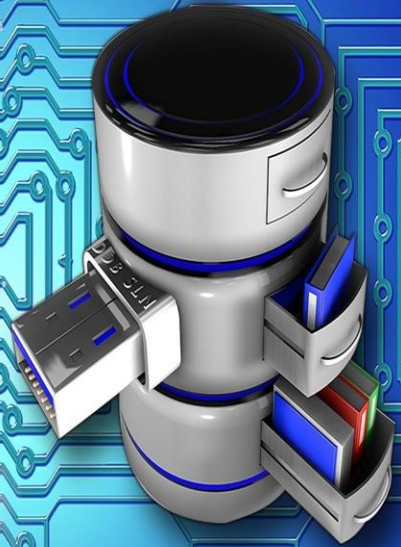
· 두번째 Query와 첫번째 Query에서 중복된 데이터를 보여줌

```
SELECT * FROM jisa INTERSECT SELECT * FROM bonsa;
```



```
-- 사원이 1명이상 존재하는 부서번호 검색
SELECT department_id FROM employees
INTERSECT
SELECT department_id FROM departments
```

DATABASE



```
SELECT empCode, empName, empSalary
FROM Employee
WHERE empName in
  (SELECT DISTINCT empName
   FROM population
   WHERE Country = "TH")
  AND empSalary <=
  (SELECT AVG(salary)
   FROM Salary
   WHERE gender = "M")
```



4. Oracle 함수

- 4.1 오라클 함수
- 4.2 단일행 함수
- 4.3 다중행(그룹) 함수

4.1 Oracle 함수

- ✓ Oracle은 다양한 데이터 처리를 위해 다양한 함수를 제공한다
- ✓ DBMS 제품마다 SQL 표준 함수와 함께 제품 고유의 내장함수를 제공한다
- ✓ 함수를 사용함으로써 SQL 코드량을 줄이고, 간결하게 작성할 수 있다



4.2 단일행 함수 (1/17) – 문자 함수 (1/4)

✓ CONCAT(char1, char2)

- 주어진 char1, char2를 결합한 문자열을 반환('||' 연산자와 동일)

✓ INITCAP(char)

- 주어진 char의 첫 번째 문자를 대문자로 변환하여 반환

✓ LOWER(char)

- 주어진 char를 소문자로 변환하여 반환

✓ UPPER(char)

- 주어진 char를 대문자로 변환하여 반환

✓ LPAD(char1, n [,char2])

- 주어진 char1에 대해 n자리 확보 후, 오른쪽으로 정렬하여 왼쪽에 생긴 빈 공백에 char2를 채워 반환

✓ RPAD(char1, n [,char2])

- 주어진 char2에 대해 n자리 확보 후 왼쪽으로 정렬 후 오른쪽에 생긴 빈 공백에 char2를 채워 반환

✓ SUBSTR(char, pos, length)

- 주어진 char에서 pos번째 자리부터 length개의 문자열을 추출하여 반환

✓ LENGTH(char)

- 주어진 char의 길이를 반환

4.2 단일행 함수 (2/17) – 문자 함수 (2/4)

✓ REPLACE(char, char1, char2)

- 주어진 char의 char1 문자를 char2 문자로 변환하여 반환

✓ INSTR(char, char1, pos, index)

- 주어진 char에서 char1 문자가 pos 시작위치에서 index번째 출현하는 위치 반환

✓ LTRIM(char, char1)

- 주어진 char의 왼쪽에서 공백이나 char1 제거 후 반환

✓ RTRIM(char, char1)

- 주어진 char의 오른쪽에서 공백이나 char1 제거 후 반환

✓ TRIM(char)

- 주어진 문자열의 왼쪽과 오른쪽으로부터 공백 제거 후 반환

4.2 단일행 함수 (3/17) – 문자 함수 (3/4)

```
SELECT CONCAT('Oracle', 'Java Developer')  
FROM dual;
```

```
SELECT INITCAP('kim ki jung')  
FROM dual;
```

```
SELECT first_name, last_name  
FROM employees  
--WHERE first_name = 'james';  
WHERE LOWER(first_name) = 'james';
```

```
SELECT UPPER('bangry')  
FROM dual;
```

```
SELECT LPAD('DataBase', 10, '*')  
FROM dual;
```

```
SELECT SUBSTR('Java Developer', 6, 9)  
FROM dual;
```

```
SELECT SUBSTR('서울시가산동', 4)  
FROM dual;
```

4.2 단일행 함수 (4/17) – 문자 함수 (4/4)

```
SELECT first_name, LENGTH(first_name)
FROM employees;
```

```
SELECT REPLACE('기정바보', '바보', '최고')
FROM dual;
```

```
SELECT REPLACE('서울 시', ' ', '')
FROM dual;
```

```
SELECT INSTR('DataBase', 'B')
--SELECT INSTR('DataBase', 'a', 1, 2)
FROM dual;
```

```
--SELECT LTRIM(' JavaDeveloper')
--SELECT LTRIM(' JavaDeveloper', ' ')
SELECT LTRIM('JavaDeveloper', 'Java')
FROM dual;
```

```
--SELECT RTRIM('JavaDeveloper ')
--SELECT RTRIM('JavaDeveloper ', ' ')
SELECT RTRIM('JavaDeveloper', 'Developer')
FROM dual;
```

```
SELECT TRIM(' Java Developer ')
FROM dual;
```

4.2 단일행 함수 (5/17) – 숫자 함수 (1/2)

- ✓ ROUND(number, i)
 - 주어진 number를 소수점 i+1번째 자리에서 반올림하여 반환 (i를 생략하면 기본값으로 0이 사용되어 반환)
- ✓ TRUNC(number, i)
 - 주어진 number를 소수점 i+1번째 자리에서 버린 후 반환 (i를 생략하면 기본값으로 0이 사용되어 반환)
- ✓ MOD(number1, number)
 - 주어진 number1를 number2로 나눈 나머지를 반환
- ✓ CEIL(number)
 - 주어진 number 보다 큰 정수 중에 가장 작은 정수를 반환
- ✓ FLOOR(number)
 - 주어진 number보다 작은 정수 중에 가장 큰 정수를 반환
- ✓ ABS(number)
 - 주어진 number의 절대값 반환
- ✓ POWER(number1, n)
 - 주어진 number1을 n승 거듭제곱한 값을 반환
- ✓ SQRT(number)
 - 주어진 number의 루트 값을 반환
- ✓ SIN(number)
 - 주어진 number의 SIN 값을 반환
- ✓ COS(number)
 - 주어진 number의 COS 값을 반환
- ✓ TAN(number)
 - 주어진 number의 TAN 값을 반환

4.2 단일행 함수 [6/17] – 숫자 함수 [2/2]

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

```
SELECT TRUNC(45.923), TRUNC(45.923, 0), TRUNC(45.923, 2), TRUNC(45.923, -1)
FROM dual;
```

```
SELECT CEIL(123.123)
FROM dual;
```

```
SELECT FLOOR(123.123)
FROM dual;
```

```
SELECT ABS(-500)
FROM dual;
```

```
SELECT POWER(5, 2), SQRT(5), SIN(30), COS(30), TAN(30)
FROM dual;
```

-- 최소값 반환

```
SELECT LEAST(10, 20, 30, 40)
FROM dual;
```

-- 최대값 반환

```
SELECT GREATEST(10, 20, 30, 40)
FROM dual;
```

4.2 단일행 함수 (7/17) – 날짜 함수 (1/2)

- ✓ SYSDATE
 - 현재 시스템 날짜 반환
- ✓ SYSTIMESTAMP
 - 현재 시스템 타임스탬프 반환
- ✓ ADD_MONTHS(date, int)
 - date에 int 수 만큼 월을 더한 날짜 반환
- ✓ MONTHS_BETWEEN(date1, date2)
 - date1을 기준으로 date2 날짜 사이의 개월 수 반환
- ✓ LAST_DAY(date)
 - 주어진 date 월의 마지막 날짜 반환
- ✓ NEXT_DAY(date, char)
 - 주어진 date 기준으로 char에 명시한 다음 요일의 날짜 반환
 - char: 1(일요일), 2(월요일), 3(화요일), 4(수요일), 5(목요일), 6(금요일), 7(토요일)
- ✓ ROUND(date, format)
 - format에 따라 반올림 날짜 반환
 - format(YEAR: 7월 1일부터 반올림, MONTH: 매월 16일부터 반올림)
- ✓ TRUNC(date, format)
 - format에 따라 잘라낸 날짜 반환

4.2 단일행 함수 (8/17) – 날짜 함수 (2/2)

```
SELECT SYSDATE  
FROM dual;
```

-- DATE 타입에 연산 가능

```
SELECT SYSDATE - 1 "어제" , SYSDATE "오늘", SYSDATE + 1 "내일"  
FROM dual;
```

-- 사원별 근무 일수 검색

```
SELECT first_name, hire_date, SYSDATE, CEIL(SYSDATE - hire_date) "근무일수"  
FROM employees;
```

-- 사원별 근무 개월수 검색

```
SELECT first_name, TRUNC(MONTHS_BETWEEN(SYSDATE, hire_date)) "근무개월수"  
FROM employees;
```

-- 특정개월수를 더한 날짜 반환

```
SELECT SYSDATE, ADD_MONTHS(SYSDATE, 2) "오늘부터 2개월 후"  
FROM dual;
```

-- 이번주 토요일 날짜

```
SELECT SYSDATE "오늘", NEXT_DAY(SYSDATE, 7) "이번주 토요일"  
FROM dual;
```

-- 이번달 마지막 날짜

```
SELECT SYSDATE, LAST_DAY(SYSDATE) "이번달 마지막날"  
FROM dual;
```

4.2 단일행 함수 (9/17) – 형 변환 함수 (1/6)

- ✓ TO_CHAR(number|date,[format])
 - 주어진 number나 date를 format 문자에 맞는 문자로 변환하여 반환
- ✓ 숫자 관련 format 문자

포맷문자	설 명
9	출력할 자릿수를 지정하고, 값이 지정한 자릿수보다 작으면 공백으로 표시
0	출력할 자릿수를 지정하고, 값이 지정한 자릿수보다 작으면 앞을 0으로 표시
\$	달러 기호를 숫자 앞에 표시
,(콤마)	명시한 위치에 콤마(,) 표시
.(소수점)	명시한 위치에 소수점(.) 표시
S	숫자 앞에 부호(+, -) 표시
MI	오른쪽에 음수 “-” 기호로 표시
PR	음수값을 “<>” 기호로 표시
EEEE	지수(과학적 표기)로 표시
L	지역 통화로 표시
RN(rm)	로마자 대(소)문자로 표시
X(x)	16진수 알파벳 대(소)문자로 표시

4.2 단일행 함수 (10/17) – 형 변환 함수 (2/6)

✓ 날짜 관련 format 문자

포맷문자	설 명
YYYY	4자리 년도(예 : 2002)
YY	2자리 년도(예 : 02)
Q	분기(1, 2, 3, 4)
MM	2자리 숫자 월 (예 : 10)
MON	알파벳 월 이름 3자리(예 : JAN , FEB , MAR 등)
MONTH	알파벳 월 이름(예 : JANUARY,FEBUARY,MARCH 등)
DD	날짜(예 : 14)
D	주의 일수(예 : 4)
DY	알파벳 요일 이름 3자리(예 : SUN , MON , TUE 등)
DAY	알파벳 요일 이름(예 : SUNDAY , MONDAY 등)
HH	12시간
HH24	24시간
MI	분
SS	초
AMPM	오전/오후

4.2 단일행 함수 (11/17) – 형 변환 함수 (3/6)

```
SELECT TO_CHAR(12345), TO_CHAR(12345.67)
FROM dual;
```

```
SELECT TO_CHAR(12345, '999,999'), TO_CHAR(12345.677, '999,999.99')
FROM dual;
```

```
SELECT TO_CHAR(12345, '000,000'), TO_CHAR(12345.677, '000,000.00')
FROM dual;
```

```
SELECT TO_CHAR(150, '$9999'), TO_CHAR(150, '$0000')
FROM dual;
```

```
SELECT TO_CHAR(150, 'S9999'), TO_CHAR(150, 'S0000')
FROM dual;
```

```
SELECT TO_CHAR(150, '9999MI'), TO_CHAR(-150, '9999MI')
FROM dual;
```

```
SELECT TO_CHAR(150, '9999EEEE'), TO_CHAR(150, '99999B')
FROM dual;
```

4.2 단일행 함수 (12/17) – 형 변환 함수 (4/6)

```
SELECT TO_CHAR(150, 'RN'), TO_CHAR(150, 'rn')  
FROM dual;
```

```
SELECT TO_CHAR(10, 'X'), TO_CHAR(10, 'x'), TO_CHAR(15, 'X')  
FROM dual;
```

```
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD AM HH:MI:SS DAY')  
FROM dual;
```

```
SELECT first_name, hire_date, TO_CHAR(hire_date, 'YYYY-MM-DD HH24:MI')  
FROM employees;
```

-- 입사연도가 2002년도인 사원들

```
SELECT first_name, hire_date  
FROM employees  
WHERE TO_CHAR(hire_date, 'YYYY') = '2002';
```

4.2 단일행 함수 (13/17) – 형 변환 함수 (5/6)

✓ TO_NUMBER(char|date, [format])

- 주어진 char를 format에 맞는 숫자로 반환

```
SELECT TO_NUMBER('12345') + 1  
FROM dual;
```

```
SELECT TO_NUMBER('12,345', '00,000') + 1  
FROM dual;
```

```
SELECT TO_NUMBER('1000') + TO_NUMBER('2,000', '0,000') + 1  
FROM dual;
```

4.2 단일행 함수 (14/17) – 형 변환 함수 (6/6)

✓ TO_DATE(char|number, [format])

- 주어진 char나 number를 format에 맞는 날짜로 반환

```
SELECT TO_DATE('2021/12/31 18:45:23', 'YYYY/MM/DD HH24:MI:SS')  
FROM dual;
```

```
SELECT first_name, hire_date  
FROM employees  
WHERE hire_date = TO_DATE('2003-06-17', 'YYYY-MM-DD');
```

```
SELECT first_name, hire_date  
FROM employees  
WHERE hire_date = TO_DATE(20030617, 'YYYY-MM-DD');
```


4.2 단일행 함수 (15/17) – 일반 함수

✓ NVL(char|number|date, expression)

- 주어진 char, number, date가 NULL 이면 expression 값을 반환한다

✓ NVL2(char|number|date, expression1, expression2)

- 주어진 char, number, date가 NULL 이면 expression2 값을, NULL 이 아니면 expression1 값을 반환한다

```
SELECT NVL(NULL, 10) FROM dual;
```

```
SELECT 10*NULL, 10*NVL(NULL, 1)  
FROM dual;
```

```
SELECT first_name, salary, NVL(commission_pct, 0)  
FROM employees;
```

```
SELECT first_name, salary, commission_pct, ( salary + ( salary * commission_pct ) ) * 12 "연봉"  
FROM employees;
```

```
SELECT first_name, salary, commission_pct, ( salary + ( salary * NVL(commission_pct, 0) ) ) * 12 "연봉"  
FROM employees;
```

```
SELECT first_name, salary, NVL2(commission_pct, commission_pct, 0)  
FROM employees;
```

```
SELECT first_name, salary, NVL2(TO_CHAR(commission_pct), TO_CHAR(commission_pct), '신입사원') "인센티브"  
FROM employees;
```

4.2 단일행 함수 (16/17) – 조건 함수 (1/2)

✓ DECODE(기준값, 조건1, 결과1 [, 조건2, 결과2, ...] [, 기본값])

- 프로그래밍 언어에서 사용하는 if ~ else ~ 문과 같은 기능 수행한다
- 필요에 따라 중첩하여 사용할 수 있다

```
SELECT first_name, job_id,  
       DECODE(job_id, 'IT_PROG', '개발자',  
               'AC_MRG', '관리자',  
               'FI_ACCOUNT', '회계사',  
               '직원')  
FROM employees;
```

```
-- 회사 직종별 급여 인상  
SELECT first_name, job_id, salary,  
       DECODE(job_id, 'IT_PROG', salary * 1.5,  
               'AC_MRG', salary * 1.3,  
               'AC_ASST', salary * 1.1,  
               salary) "인상급여"  
FROM employees;
```

4.2 단일행 함수 (17/17) – 조건 함수 (2/2)

✓ CASE ~ WHEN 표현식

- DECODE() 함수와 유사하게 사용할 수 있는 표현식이다

```
CASE 기준값 WHEN 비교값1 THEN 결과1
           WHEN 비교값2 THEN 결과2
           ELSE   결과3
END;
```

```
SELECT first_name,
       department_id,
       CASE
         WHEN department_id = 10 THEN '영업부'
         WHEN department_id = 20 THEN '총무부'
         WHEN department_id = 30 THEN '인사부'
         ELSE '인사발령'
       END "부서명"
FROM   employees
ORDER BY department_id ASC;
```

4.3 다중행(그룹) 함수 (1/4)

✓ 여러 개의 행 당 하나의 결과값을 반환하는 함수

SELECT	그룹 함수([DISTINCT]/all), {column [Alias] ,...}
FROM	테이블명
[WHERE]	Query 조건(들)
[GROUP BY]	컬럼1, 컬럼2, ...n
[HAVING]	Group-Conditiona
[ORDER BY]	컬럼1, 컬럼2, [ASC/DESC] ;

- GROUP BY** : 결과값을 지정한 컬럼을 기준으로 그룹화
- HAVING** : GROUP BY에 의한 결과에 대한 조건 절
- ORDER BY** : 결과값을 분류(Sorting)할 때

COUNT()	조건을 만족하는 모든 행의 개수를 반환
SUM()	조건을 만족하는 모든 행의 총합계를 반환
AVG()	조건을 만족하는 모든 행의 평균을 반환
MIN()	조건을 만족하는 모든 행의 최소값을 반환
MAX()	조건을 만족하는 모든 행의 최대값을 반환

4.3 다중행(그룹) 함수 (2/4)

-- 전체 사원수(NULL은 포함 안됨)

```
SELECT COUNT(employee_id) FROM employees;
```

-- 전체 사원수(NULL 포함)

```
SELECT COUNT(*) FROM employees;
```

-- 커미션 받는 사원의 수

```
SELECT COUNT(commission_pct)
FROM employees;
```

```
SELECT COUNT(*) "전체사원수", COUNT(commission_pct) "커미션사원수"
FROM employees;
```

-- 급여 총액(NULL은 무시)

```
SELECT SUM(salary)
FROM employees;
```

-- 급여 평균(NULL은 무시)

```
SELECT AVG(salary)
FROM employees;
```

-- 커미션 평균

```
SELECT AVG(commission_pct), AVG(NVL(commission_pct, 0))
FROM employees;
```

4.3 다중행(그룹) 함수 (3/4)

-- 최대값, 최소값

```
SELECT MAX(salary), MIN(salary), MAX(commission_pct), MIN(commission_pct)
FROM employees;
```

```
SELECT MAX(hire_date), MIN(hire_date), MAX(hire_date) - MIN(hire_date) "짬밥차"
FROM employees;
```

-- GROUP BY 절(특정 컬럼을 기준으로 그룹핑)

```
SELECT department_id
FROM employees
GROUP BY department_id;
```

-- 부서별 급여총액, 평균

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

-- HAVING 절(그룹에 대한 조건)

```
SELECT department_id, SUM(salary), AVG(salary)
FROM employees
GROUP BY department_id
HAVING department_id = 10;
```

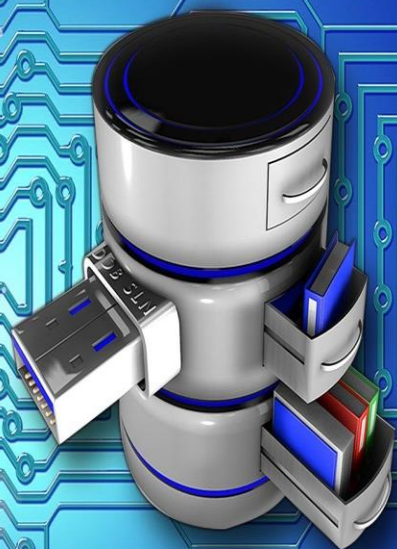
4.3 다중행(그룹) 함수 (4/4)

```
SELECT department_id, SUM(salary), AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) >= 3000;
```

```
SELECT department_id, MAX(salary), MIN(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary) > 20000;
```

```
SELECT hire_date, COUNT(*)
FROM employees
GROUP BY hire_date
ORDER BY hire_date;
--ORDER BY COUNT(*);
```


DATABASE



```
SELECT empCode, empName, empSalary
FROM Employee
WHERE empName in
  (SELECT DISTINCT empName
   FROM population
   WHERE Country = "TH")
  AND empSalary <=
  (SELECT AVG(salary)
   FROM Salary
   WHERE gender = "M")
```



5. JOIN(조인)

5.1 JOIN 개요

5.2 JOIN 종류

5.1 JOIN 개요

- ✓ 하나의 테이블에 원하는 데이터가 모두 저장되어 있지 않은 경우, 두개 이상의 테이블을 묶어 조건을 만족하는 테이블 데이터를 조회하는 것을 JOIN 이라 한다
- ✓ 두 테이블의 조인을 위해서는 기본키(PRIMARY KEY, PK)와 외래키(FOREIGN KEY, FK) 관계를 가진 즉, 공통된 컬럼을 가지고 있는 테이블들을 행과 행을 결합하여 데이터를 조회할 수 있다
 - 예) 사원테이블의 사원이름과 부서테이블의 부서명을 함께 조회할 JOIN을 사용하여야 한다

PK

EMP				
사원번호	사원명	직급	급여	부서번호
1	주종면	차장	300	10
2	주영현	과장	250	10
3	홍경옥	과장	200	20

FK



PK

DEPT		
부서번호	부서명	지역코드
10	전산과	1
20	경영지원과	1

5.2 JOIN 종류 (1/11)

✓ CROSS JOIN(상호 조인)

- 한쪽 테이블의 모든 행과 다른 쪽 테이블의 모든 행을 조인하는 기능이다

✓ INNER JOIN(내부 조인)

- 부모와 자식의 관계를 가진 2개 이상의 테이블에서 공통되는 컬럼(PK & FK) 값을 비교하여 조건을 만족하는 행과 행을 조인한다
- 두 테이블에서 공통적으로 존재하는 컬럼 값을 기준으로 행과 행을 조인한다

✓ OUTER JOIN(외부 조인)

- 두 테이블을 조인할 때, 1개의 테이블에만 행 데이터가 존재하여도 조회할 수 있다

✓ SELF JOIN(자체 조인)

- 자신이 자신과 조인한다는 의미로, 1개의 테이블을 사용한다

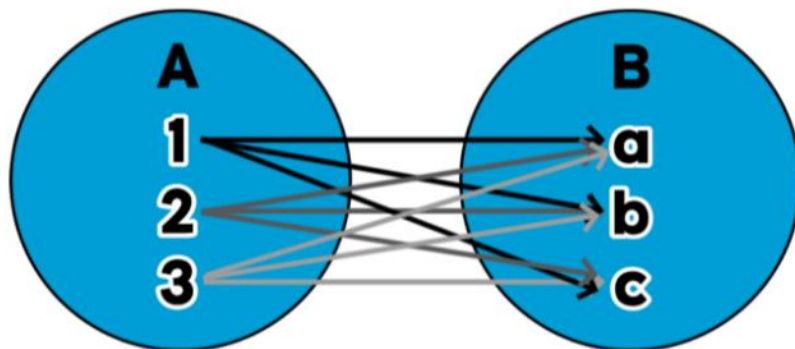
5.2 JOIN 종류 (2/11) – CROSS JOIN (1/2)

✓ 한쪽 테이블의 모든 행과 다른 쪽 테이블의 모든 행을 조인하는 기능이다

- 조인 조건 없이 한 개 이상의 테이블에 대한 조인을 말한다
- 카티션 곱(CARTESIAN PRODUCT)라고도 한다
- 모든 행들의 조합이 검색되므로 매우 많은 결과를 출력할 수 있으며, 그만큼 과부하의 위험을 초래한다(이론상의 조인)

CROSS JOIN

*CARTESIAN PRODUCT



CROSS JOIN 결과: 전체 행 개수 = 9
3(A 테이블의 행 개수) X 3(B 테이블의 행 개수)

5.2 JOIN 종류 (3/11) – CROSS JOIN (2/2)

-- 하나의 사원은 27개 모든 부서에 소속될 수 있다 (107*27=2889)

```
SELECT first_name, department_name  
FROM employees, departments;
```

-- 조인 시 컬럼명의 모호성을 해결하기 위해 테이블명이나 별칭 사용을 권장

```
SELECT employees.first_name, departments.department_name  
FROM employees, departments;
```

```
SELECT e.first_name, d.department_name  
FROM employees e, departments d;
```

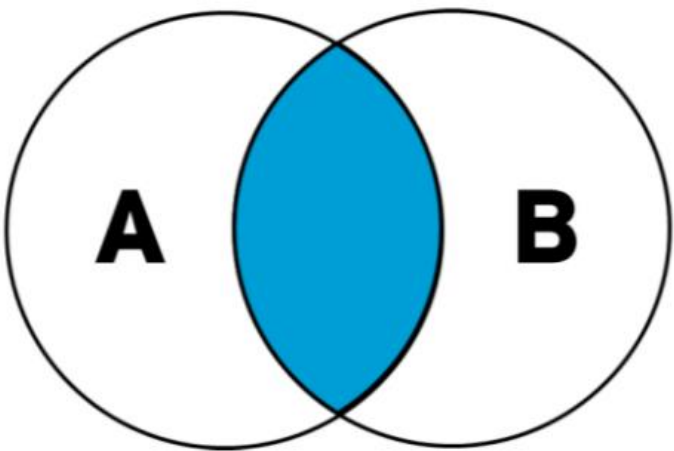
-- DBMS간의 호환을 위해 ANSI(미국규격협회)에서 제시한 표준 CROSS JOIN 구문 (오라클 9i부터 지원)

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

5.2 JOIN 종류 (4/11) – INNER JOIN (1/4)

- ✓ INNER JOIN(내부 조인)은 부모와 자식의 관계를 가진 2개 이상의 테이블에서 공통되는 컬럼(PK & FK) 값을 비교하여 조건을 만족하는 행과 행을 조인한다
- 두 테이블에서 공통적으로 존재하는 컬럼 값을 기준으로 행과 행을 조인한다
 - 테이블 간의 교집합만을 조인한다
 - 가장 자연스런 조인으로 Natural Join이라고도 한다

INNER JOIN



employees

FIRST_NAME	LAST_NAME	EMAIL	DEPARTMENT_ID
스티븐	킹	SKING	90
리나	Kochhar	NKOCHHAR	90
렉스	De Haan	LDEHAAN	90

departments

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	경영지원(Administration)	200	1700
20	마케팅	201	1800
30	구매부	114	1700
40	인사부	203	2400
50	물류(Shipping)	121	1500
60	IT	103	1400
70	홍보부(Public Relations)	204	2700
80	판매(Sales)	145	2500
90	경영(Executive)	100	1700
100	회계(Finance)	108	1700
110	경리(Accounting)	205	1700
120	재무(Treasury)		1700

5.2 JOIN 종류 (5/11) – INNER JOIN (2/4)

✓ EQUI JOIN

- 조인하고자 하는 두 테이블에서 공통적으로 존재하는 컬럼(department_id)의 값이 일치하는 행과 행을 조인한다

✓ NON-EQUI JOIN

- 조인 조건에서 '=' 연산자를 이용한 동등비교가 아닌 다른 비교연산자를 사용하여 특정범위로 행과 행을 연결하여 조인한다
- 예) 사원 급여에 따른 급여등급 출력

```
-- 오라클 EQUI JOIN 구문
SELECT e.employee_id,
       e.last_name,
       d.department_name
FROM   employees e,
       departments d
WHERE  e.department_id = d.department_id
       AND e.salary >= 3000;
```

```
-- ANSI 표준 EQUI JOIN 구문
SELECT e.employee_id,
       e.last_name,
       d.department_name
FROM   employees e
/*INNER*/ JOIN departments d
--on e.department_id = d.department_id
using(department_id)
WHERE  e.salary >= 3000;
```


5.2 JOIN 종류 (6/11) – INNER JOIN (3/4)

✓ EQUI JOIN

- 조인하고자 하는 두 테이블에서 공통적으로 존재하는 컬럼(department_id)의 값이 일치하는 행과 행을 조인한다

-- 3개이상 테이블 조인(딕셔너리(테이블)로부터 테이블 종류 조회)

```
SELECT *  
FROM user_tables;  
  
SELECT e.employee_id,  
       e.last_name,  
       d.department_name,  
       l.city,  
       l.state_province,  
       c.country_name  
FROM   employees e  
       JOIN departments d  
         ON e.department_id = d.department_id  
       JOIN locations l  
         ON d.location_id = l.location_id  
       JOIN countries c  
         ON l.country_id = c.country_id;
```

5.2 JOIN 종류 (7/11) – INNER JOIN (4/4)

✓ NON-EQUI JOIN

- 조인 조건에서 '=' 연산자를 이용한 동등비교가 아닌 다른 비교연산자를 사용하여 특정범위로 행과 행을 연결하여 조인한다
- 예) 사원 급여에 따른 급여등급 출력

```
-- 오라클 NON-EQUI JOIN
SELECT e.ename,
       e.sal,
       s.grade
FROM   emp e,
       salgrade s
WHERE  e.sal BETWEEN s.losal AND s.hisal;

SELECT e.employee_id,
       e.last_name,
       e.salary,
       j.job_title
FROM   employees e,
       jobs j
WHERE  e.salary BETWEEN j.min_salary AND
j.max_salary
ORDER BY e.employee_id;
```

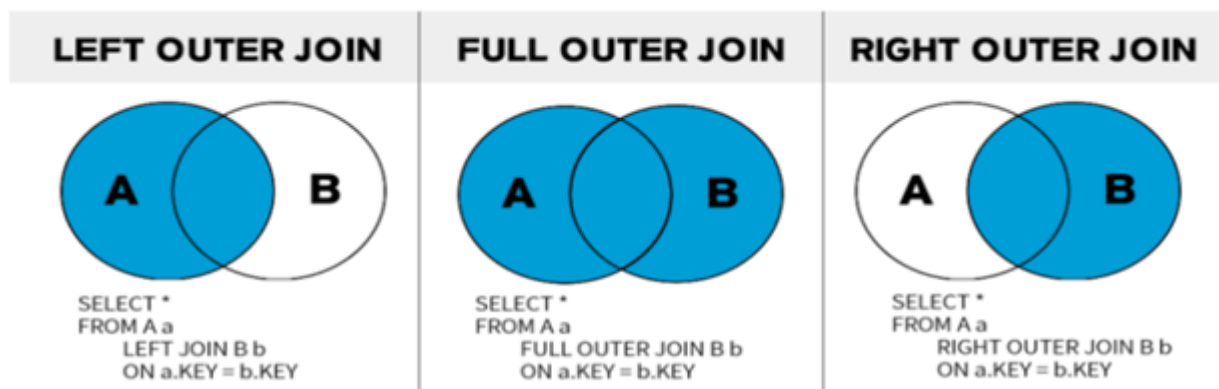
```
-- ANSI 표준 NON-EQUI JOIN
SELECT e.ename,
       e.sal,
       s.grade
FROM   emp e
       JOIN salgrade s
       ON e.sal BETWEEN s.losal AND s.hisal;
-- desc jobs;
SELECT e.employee_id,
       e.last_name,
       e.salary,
       j.job_title
FROM   employees e
       JOIN jobs j
       ON e.salary BETWEEN j.min_salary
AND j.max_salary
ORDER BY e.employee_id;
```

5.2 JOIN 종류 (8/11) – OUTER JOIN (1/3)

✓ OUTER JOIN(외부 조인)

- 두 테이블을 조인할 때, 1개의 테이블에만 행 데이터가 존재하여도 조회할 수 있다
- INNER JOIN 시 조인 조건을 만족하지 않으면 행은 검색되지 않는다
- 조인 조건을 만족하지 않는 행들도 검색에 포함하고자 할 경우 사용한다

OUTER JOIN



5.2 JOIN 종류 (9/11) – OUTER JOIN (2/3)

✓ Oracle OUTER JOIN

```
-- 오라클 OUTER JOIN에서는 조인 시킬 값이 없는 쪽에 (+) 기호를 추가한다
-- employees 테이블에서 부서번호가 NULL 인 Kimberly는 EQUI Join만으로는 검색되지 않음
SELECT e.first_name, d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;

-- 부서번호가 없는 사원도 조회
-- 오라클 OUTER JOIN
SELECT e.employee_id,
       e.first_name,
       e.last_name,
       d.department_name
FROM   employees e,
       departments d
WHERE  e.department_id = d.department_id(+);
```

5.2 JOIN 종류 (10/11) – OUTER JOIN (3/3)

✓ ANSI 표준 OUTER JOIN

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

```
-- LEFT OUTER JOIN
SELECT e.first_name,
       d.department_name
FROM   employees e
       LEFT OUTER JOIN departments d
         ON e.department_id = d.department_id;
```

```
-- RIGHT OUTER JOIN
SELECT e.first_name,
       d.department_name
FROM   employees e
       RIGHT OUTER JOIN departments d
         ON e.department_id = d.department_id;
```

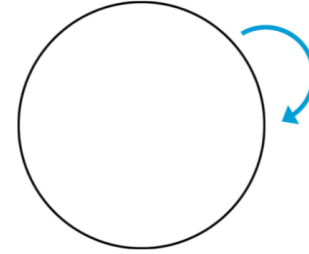
```
-- FULL OUTER JOIN
SELECT e.first_name,
       d.department_name
FROM   employees e
       FULL OUTER JOIN departments d
         ON e.department_id = d.department_id;
```

5.2 JOIN 종류 (11/11) – SELF JOIN

✓ SELF JOIN(자체 조인)

- 자신이 자신과 조인한다는 의미로, 1개의 테이블을 사용한다
- 같은 테이블에 별칭을 부여하여 또 다른 테이블처럼 조인한다
 - 예) 사원 별 상사(매니저) 이름 조회

SELF JOIN



```
-- 오라클 SELF JOIN
-- 사원별 매니저 검색
SELECT e.first_name ,m.first_name
FROM   EMPLOYEES e, EMPLOYEES m
WHERE  e.manager_id = m.employee_id;

-- 상사가 없는 사원도 검색 시 OUTER JOIN 필요
SELECT e.first_name , m.first_name
FROM   EMPLOYEES e, EMPLOYEES m
WHERE  e.manager_id = m.employee_id(+);
```

```
-- ANSI 표준 SELF JOIN
SELECT e.first_name , m.first_name
FROM EMPLOYEES e
     LEFT OUTER JOIN EMPLOYEES m
       ON e.manager_id = m.employee_id;
```

DATABASE



```
SELECT empCode, empName, empSalary
FROM Employee
WHERE empName in
    (SELECT DISTINCT empName
     FROM population
     WHERE Country = "TH")
    AND empSalary >=
    (SELECT AVG(salary)
     FROM Salary
     WHERE gender = "M")
```

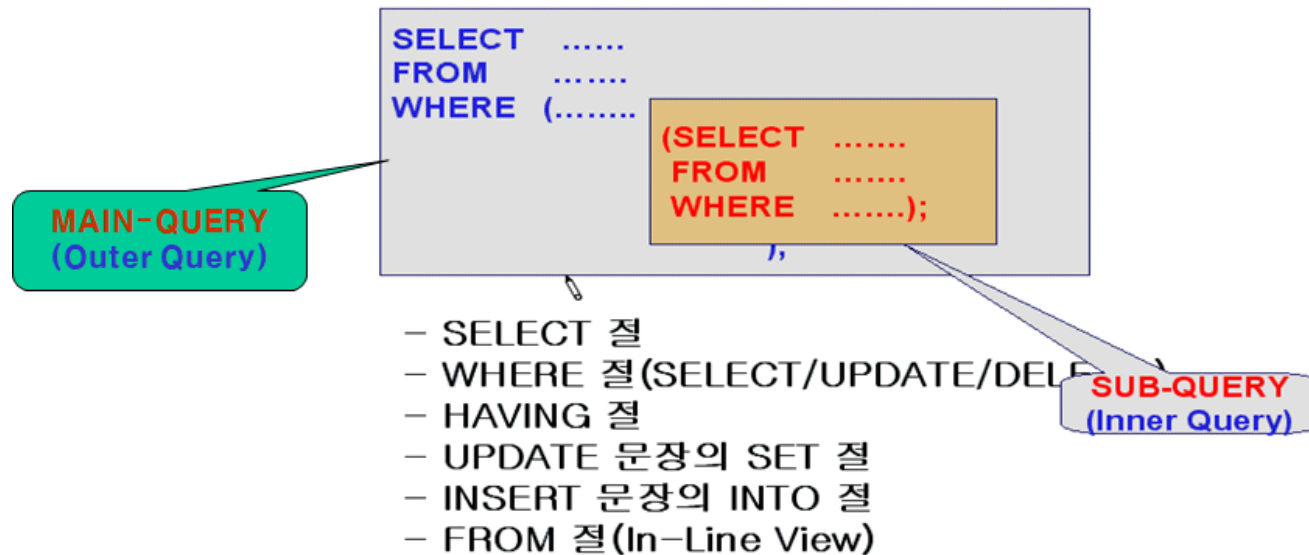


6. SUBQUERY(서브쿼리)

- 6.1 SUBQUERY 개요
- 6.2 SUBQUERY 종류
- 6.3 일반 SUBQUERY
 - 단일행 서브쿼리(SingleRow Subquery)
 - 다중행 서브쿼리(MultiRow Subquery)
 - 다중컬럼 서브쿼리(MultiColumn Subquery)
- 6.4 인라인 뷰(Inline View)
- 6.5 스칼라 서브쿼리(Scalar Subquery)
- 6.6 상호 연관 서브쿼리(Correlative Subquery)
- 6.7 ROWID & ROWNUM

6.1 SUBQUERY 개요

- ✓ SQL 내부에서 보조적으로 사용되는 또 다른 SELECT문을 서브 쿼리라 한다
- ✓ 최종 결과를 조회하는 쿼리를 메인 쿼리라고 하며, 중간 단계 혹은 보조 역할의 내부 SELECT문을 서브 쿼리라 한다
- ✓ 서브 쿼리에서의 실행 결과를 메인 쿼리에 전달하여 새로운 결과를 검색 하기 위해 사용한다
- ✓ 예) 사원 중에 이름이 Seo 인 사원과 동일한 급여(salary)를 받는 사원을 조회
 - 서브 쿼리에서 사원테이블의 이름이 Seo 인 사원의 급여를 조회한 후 메인 쿼리에서 동일한 급여를 받는 사원 목록 조회
- ✓ 특징
 - 서브 쿼리는 디스크에서 한번 읽어온 데이터를 메모리 변수처럼 재사용할 수 있도록 도와준다(물리적인 I/O를 줄여준다)
 - 서브 쿼리는 메인 쿼리가 실행되기 이전에 한 번만 실행되며, 서브 쿼리는 여러 번 사용할 수 있다
 - 서브 쿼리는 SELECT, FROM, WHERE 절 모두 사용 가능하며, INSERT, UPDATE, MERGE, DELETE 문에서도 사용 가능하다



6.2 SUBQUERY 종류

✓ 일반 서브쿼리(Subquery)

- 하나의 변수처럼 사용한다. 쿼리 결과에 따라 다음과 같이 분류한다
- **단일행 서브쿼리(SingleRow Subquery)**
 - 쿼리 결과가 단일행만을 반환하는 서브쿼리이다
- **다중행 서브쿼리(MultiRow Subquery)**
 - 쿼리 결과가 다중행을 반환하는 서브쿼리이다
- **다중컬럼 서브쿼리(MultiColumn Subquery)**
 - 쿼리 결과가 다중컬럼을 반환하는 서브쿼리이다

✓ 인라인 뷰(Inline View)

- 뷰 형태로써 테이블을 반환하는 서브쿼리이다(테이블 대체)

✓ 스칼라 서브쿼리(Scalar Subquery)

- 하나의 컬럼처럼 사용하는 서브쿼리이다

✓ 상호 연관 서브쿼리(Correlative Subquery)

- 메인 쿼리의 값을 사용하는 서브쿼리를 상호 연관 서브쿼리라 한다
- 일반 서브쿼리, 인라인 뷰, 스칼라 서브쿼리도 메인 쿼리의 값을 사용하면 상호 연관 서브쿼리이다

6.3 일반 서브쿼리 (1/4) – 단일행 서브쿼리

- ✓ 쿼리 결과가 단일행만을 반환하는 서브쿼리이다
- ✓ 단일행을 반환하기 때문에 단일행 비교연산자(=, >, <, <>)만 사용해야 한다

-- 서브쿼리를 사용하지 않을 경우

```
SELECT salary
FROM employees
WHERE LOWER(last_name) = 'seo';
```

```
SELECT *
FROM employees
WHERE salary = 2700;
```

-- 단일행 서브쿼리

```
SELECT *
FROM employees
WHERE salary = (SELECT salary
                  FROM employees
                  WHERE LOWER(last_name) = 'seo');
```

-- 전체사원 평균급여보다 더 많은 급여를 받는 사원 목록 조회

```
SELECT *
FROM employees
WHERE salary > (SELECT AVG(salary)
                 FROM employees);
```

6.3 일반 서브쿼리 (2/4) – 다중행 서브쿼리 (1/2)

- ✓ 쿼리 결과가 다중행을 반환하는 서브쿼리이다
- ✓ 다중행을 반환하기 때문에 다중행 비교연산자(IN, ANY, ALL, EXISTS)만 사용해야 한다

```
-- IN 연산자
-- 30번 부서에 소속된 직원들과 동일한 업무를 가지는 전체 직원목록 조회
SELECT last_name,
       job_id,
       department_id
FROM   employees
WHERE  job_id IN (SELECT job_id
                  FROM   employees
                  WHERE  department_id = 30);

-- ANY 연산자
-- 30번 부서의 최소급여자 보다 더 많은 급여를 받는 전체 직원목록 조회
SELECT *
FROM   employees
WHERE  salary > ANY (SELECT salary
                    FROM   employees
                    WHERE  department_id = 30);
```

6.3 일반 서브쿼리 (3/4) – 다중행 서브쿼리 (2/2)

- ✓ 쿼리 결과가 다중행을 반환하는 서브쿼리이다
- ✓ 다중행을 반환하기 때문에 다중행 비교연산자(IN, ANY, ALL, EXISTS)만 사용해야 한다

```
-- ALL 연산자
-- 30번 부서의 최대급여자 보다 더 많은 급여를 받는 전체 사원목록 조회
SELECT *
FROM employees
WHERE salary > ALL (SELECT salary
                    FROM employees
                    WHERE department_id = 30);

-- EXISTS 연산자
-- 서브쿼리의 결과 유무에 따른 조회
SELECT *
FROM employees
WHERE EXISTS(SELECT *
            FROM departments
            WHERE department_id = 30)
AND department_id = 30;
```

6.3 일반 서브쿼리 (4/4) – 다중컬럼 서브쿼리

✓ 쿼리 결과가 다중컬럼을 반환하는 서브쿼리이다

```
-- 부서별 최소급여를 받는 사원 정보
SELECT *
FROM employees
WHERE (department_id, salary) IN(SELECT department_id, MIN(salary)
                                FROM employees
                                GROUP BY department_id)
ORDER BY department_id;
```

6.4 인라인 뷰 (Inline View)

✓ 서브쿼리가 하나의 테이블처럼 사용되는 경우이다

- 굉장히 많이 사용되는 서브쿼리이다

```
-- 부서별 최대 급여자 정보 확인
SELECT e.employee_id, e.first_name, e.department_id, e.salary
FROM   employees e
       JOIN (SELECT department_id, MAX(salary) maxsal
             FROM   employees
             GROUP BY department_id) i
       ON e.department_id = i.department_id
WHERE  e.salary = i.maxsal;
```

6.5 스칼라 서브쿼리 (Scalar Subquery)

✓ 하나의 컬럼처럼 사용하는 서브쿼리로 SELECT 절에서 서브쿼리를 사용한다

```
-- 사원 번호, 사원 이름, 부서명 조회
SELECT employee_id,
       first_name,
       (SELECT department_name
        FROM departments d
        WHERE d.department_id = e.department_id) "department_name"
FROM employees e;
```

- 컬럼 역할을 하기 때문에 스칼라 서브쿼리는 반드시 한 개의 행만 반환되어야 한다
- 스칼라 서브쿼리는 행마다 서브쿼리가 실행되기 때문에 성능이 좋지 않다. 웬만하면 조인으로 표현 가능하니 조인 사용을 권장한다

6.6 상호 연관 서브쿼리 (Correlative Subquery)

✓ 메인 쿼리의 값을 사용하는 서브쿼리를 상호 연관 서브쿼리라 한다

```
-- 부서별 평균급여보다 많은 급여를 받는 사원 정보 조회
SELECT department_id, employee_id, last_name, salary, job_id
FROM employees e1
WHERE salary > (SELECT AVG(salary)
                FROM employees e2
                WHERE e2.department_id = e1.department_id)
ORDER BY e1.department_id;
```

6.7 가상 컬럼 (PSEUDO COLUMN) (1/6) – ROWID (1/2)

- ✓ ROWID는 Oracle 데이터베이스 내에서 데이터를 구분하기 위한 유일한 값이다
 - 테이블에 데이터를 입력하면 자동으로 생성되는 컬럼 값이다
- ✓ Oracle에서 INDEX를 생성하기 위해 내부적으로 사용하는 가상 컬럼으로 사용자가 임의로 변경하거나 삭제할 수 없다
 - 테이블의 가장 빠른 ACCESS 방법을 제공한다
- ✓ 테이블의 한 컬럼처럼 참조만 가능하며 데이터베이스에 저장되지는 않는다
- ✓ SELECT rowid, employee_id FROM employees 와 같은 SELECT문으로 확인할 수 있다
- ✓ ROWID를 통해서 데이터가 어떤 데이터 파일, 어느 블록에 저장되어 있는지 확인할 수 있다

✓ ROWID 구조

- AAAArs AAD AAAAUa AAA - 18문자로 구성
 - AAAArs : Data Object Id(객체 고유번호)
 - AAD : File Id(상대적 파일 번호)
 - AAAAUa : Block Id(데이터 블록 번호)
 - AAA : Row Slock 번호(블록 내 행 번호)

```
-- rowed, 사원 번호, 사원 이름 조회
SELECT ROWID, employee_id, first_name
FROM employees;
```

6.7 가상 컬럼 (PSEUDO COLUMN) (2/6) – ROWID (2/2)

✓ ROWID 활용

- 사원 테이블에서 부서번호 중복 제거 후 사원번호, 부서번호 조회하기

```
-- 부서번호 중복 제거 후 조회(에러)
SELECT employee_id, DISTINCT department_id
FROM employees;

-- 서브쿼리와 rowid 활용
SELECT employee_id, department_id
FROM employees
WHERE rowid IN (SELECT MIN(ROWID)
                FROM employees
                GROUP BY department_id);
```

6.7 가상 컬럼 (PSEUDO COLUMN) (3/6) – ROWNUM (1/4)

- ✓ ROWNUM은 SELECT문이 실행되는 과정에서 행이 인출(Fetch)된 후에 순차적으로 부여되는 논리적 일련번호(1, 2, 3..)를 위한 가상 컬럼이다
- ✓ ROWNUM은 실행 시마다 동적 생성되기 때문에 테이블의 같은 행이라도 SELECT 문에 따라 다른 값을 가질 수 있다
- ✓ 활용
 - 조회되는 행 수를 제한할 때 사용한다
 - 웹 페이지에서 페이지 단위 출력을 위해서 ROWNUM과 Inline View를 사용해야 한다

```
-- ROWNUM
SELECT ROWNUM, employee_id, first_name
FROM   employees;

-- 테이블의 같은 행이라도 서로 다른 ROWNUM을 가질 수 있다
SELECT ROWNUM, employee_id
FROM   employees;

SELECT ROWNUM, employee_id
FROM   employees
ORDER BY employee_id DESC;
```

6.7 가상 컬럼 (PSEUDO COLUMN) (4/6) – ROWNUM (2/4)

✓ 활용

- 조회되는 행 수를 제한할 때 사용한다
- 웹 페이지에서 페이지 단위 출력을 위해서 ROWNUM과 Inline View를 사용해야 한다

-- 사원 3명 조회

```
SELECT ROWNUM, employee_id, first_name  
FROM employees  
WHERE ROWNUM <= 3;
```

-- 주의

-- 첫번째 행의 rownum이 1이므로

-- 1 > 1은 false가 되어 rownum은 더이상 증가하지 않으며, 하나의 행도 반환되지 않음

```
SELECT ROWNUM, employee_id, first_name  
FROM employees  
WHERE ROWNUM > 1;
```

6.7 가상 컬럼 (PSEUDO COLUMN) 5/6) – ROWNUM (3/4)

```
SELECT ROWNUM, employee_id, first_name
FROM   employees
WHERE  ROWNUM <= 10;
```

/* 특정 컬럼을 기준으로 정렬하여 상위 5개(범위)를 조회하고자 한다면 */

-- 예) 전체 사원의 급여순으로 5명 가져오기

-- 전체 급여 순위가 아닌 처음 5명안에서의 급여순위가 됨

```
SELECT first_name, salary
FROM   employees
WHERE  ROWNUM <= 5
ORDER BY salary DESC;
```

-- FROM절에서 서브쿼리(Inline View)를 사용해야 한다

```
SELECT *
FROM   (SELECT *
        FROM   employees
        ORDER BY salary DESC)
WHERE  ROWNUM <= 5;
```

6.7 가상 컬럼 (PSEUDO COLUMN) 6/6) – ROWNUM (4/4)

```
-- 급여순으로 페이징 처리
SELECT page,
       employee_id,
       first_name,
       salary
FROM   (SELECT CEIL(ROWNUM / 10) page,
              employee_id,
              first_name,
              salary
        FROM   (SELECT ROWNUM,
                      employee_id,
                      first_name,
                      salary
                 FROM   employees
                 ORDER BY salary DESC))
WHERE  page = 2;
```

End of Document

✓ Q&A



감사합니다...