3. JDBC 프로그래밍

JDBC(Java Database Connectivity)란 자바 환경에서 데이터베이스를 연결할 수 있는 기능을 의미합니다. 데이터베이스를 사용하려면 JVM 환경과 데이터베이스 환경사이에 표준이 있어야 하는데 이러한 표준이 바로 JDBC입니다. 이 장에서는 JDBC를 이용해 데이터베이스에 접속하여 데이터를 입력/수정/삭제/조회 하는 방법에 대해 설명합니다.

주요 내용입니다.

- 데이터베이스 설정
- SQL 기본 문법
- JDBC 드라이버
- JDBC 프로그래밍 기본
- Statement
- PreparedStatement와 CallableStatement
- ResultSetMetaData와 DatabaseMetaData

3.1. 데이터베이스 개요

이 장에서는 데이터베이스를 다루는데 필요한 기본적인 용어와 기능 등에 대해 설명하고, 일반적으로 쉽게 사용할 수 있는 실제 데이터베이스 엔진을 선정하여 설치에서 사용 방법에 이르기까지 일련의 조작 과정을 설명하기로 하겠습니다.

3.1.1. 데이터베이스 용어

- 테이블 : 테이블은 RDBMS의 기본적인 저장구조로, 한 개 이상의 행(Row)과 한개 이상의 열(Column)로 구성됩니다.
- Column : 한 Column은 테이블에서 단일 종류의 데이터로 구성되는데 예를 들면 성적 테이블에서 이름, 국어, 영어 등이 Column입니다. 이것은 특정 데이터 타입 및 크기를 갖고 있습니다.
- Row : Row는 Column 값의 조합입니다. 성적 테이블에서 한 학생에 대한 성적내용이 한 Row가 됩니다. 예를 들면, "홍길동, 100, 95" 가 하나의 Row입니다.
- 필드 : Row와 Column의 교차점을 필드라고 합니다. 데이터가 없는 필드는 널(Null)값을 갖습니다. 예를 들면, 국어 Column과 홍길동 학생의 Row의 필드 값은 "100"입니다.
- 기본키(Primary Key) : 기본키는 한 테이블의 레코드를 유일하게 식별해주는 Column 또는 Column의 조합입니다. 예를 들면, 앞의 성적테이블에서 각 학생의 이름을 기본키로 사용할 수 있으며, 더 정확하게 하려면 학번 Column을 추가 시켜 기본키로 사용할 수 있습니다. 기본키는 널 값을 가질 수 없습니다.
- 외래키 : 두개 이상의 테이블을 사용하는 데이터베이스에서 두 테이블에 있는 데이터를 조작하려고 할 때 두 테이블을 연관시켜주는 Column 또는 Column의 조합을 의미합니다.

3.2. Oracle

3.2.1. Oracle Database 11g Express Edition

3.2.1.1. 다운로드

오라클 데이터베이스 설치를 위해 https://www.oracle.com 으로 연결합니다. Downloads 메뉴의 하위 메뉴에서 Oracle Database 11g Express Edition(이후 오라클 XE)을 선택합니다.

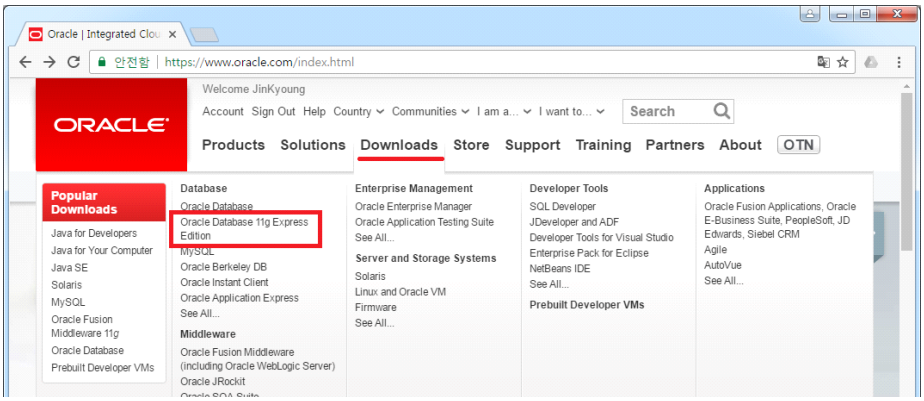


그림 39. 오라클사 홈페이지

라이선스 동의를 선택한 다음 자신의 운영체제에 맞는 제품을 다운로드 합니다.

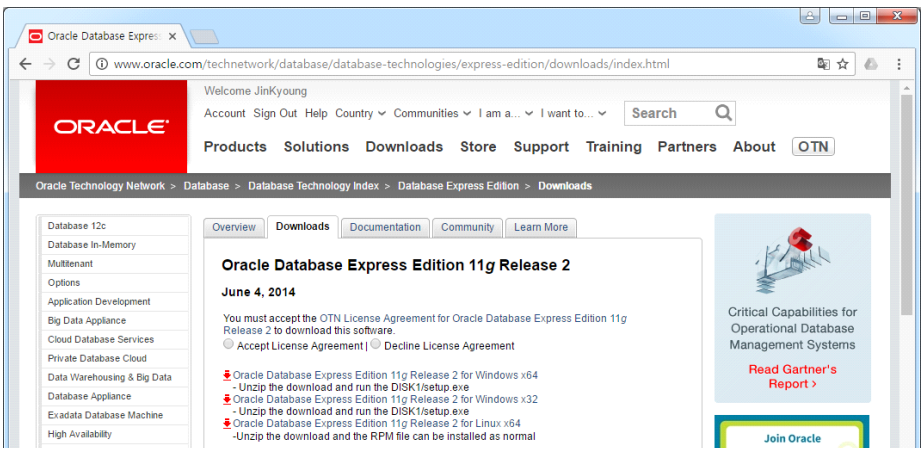


그림 40. Oracle Database Express Edition 12g Release 2 다운로드

Oracle Database 11g Express Edition을 다운로드 받으려면 계정이 있어야 합니다. 계정은 무료로 만들 수 있습니다.

3.2.1.2. 설치

오라클 데이터베이스 11g 익스프레스 에디션은 데스크탑 윈도우 환경에서 SQL을 배우기에 충분합니다. 윈도우용 설치 파일은 압축파일(zip 파일)로 제공됩니다. 압축파일을 풀면 DISK1 폴더를 볼 수 있습니다.

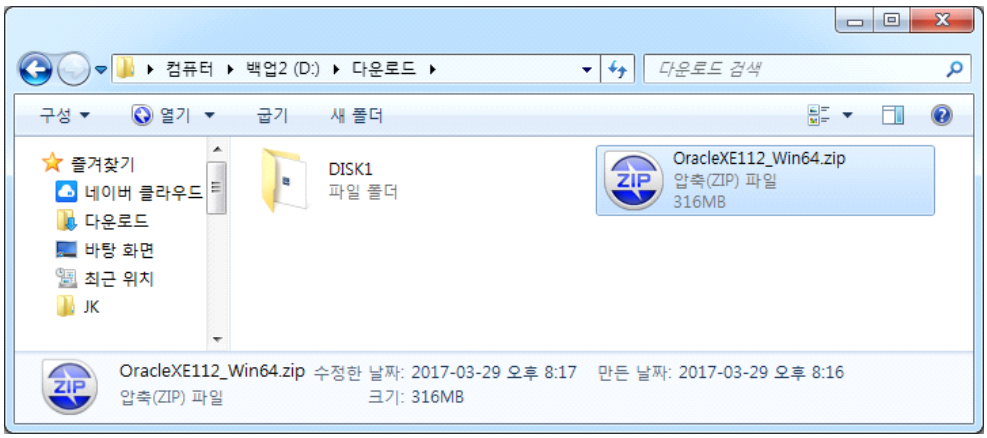


그림 41. 다운로드 파일과 압축 풀기 한 폴더

DISK1 폴더에서 setup.exe 파일을 더블클릭하면 오라클 데이터베이스 11g 익스프레스 에디션을 설치 시작합니다.

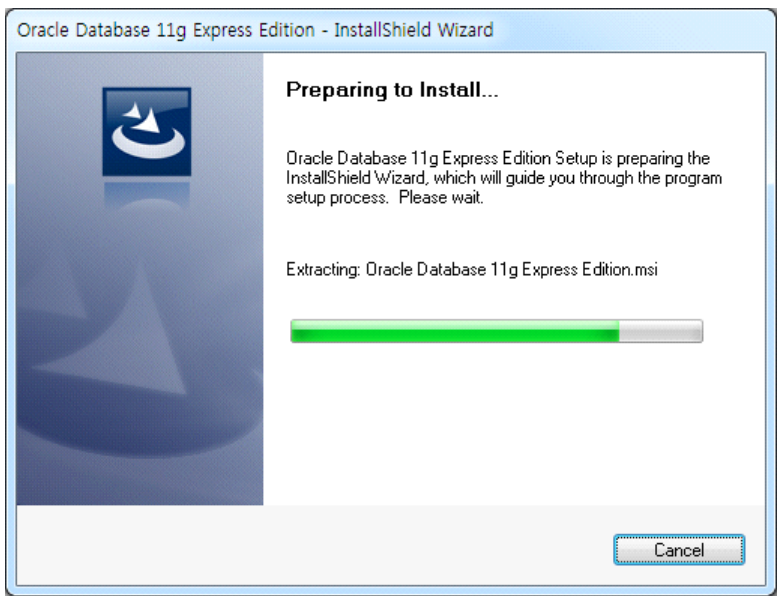


그림 42. 설치 시작

설치 마법사가 시작되면 [Next] 버튼을 클릭합니다.

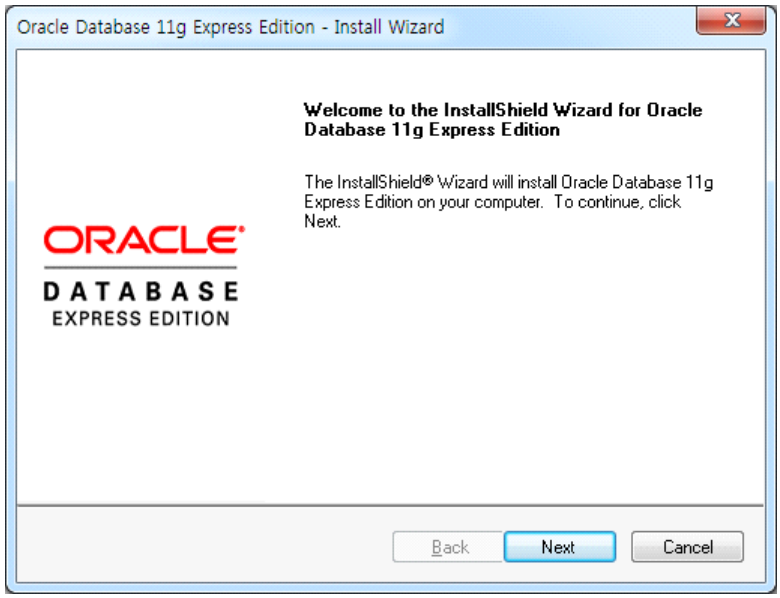


그림 43. 설치 마법사

라이선스 동의를 선택하고 [Next] 버튼을 클릭합니다.

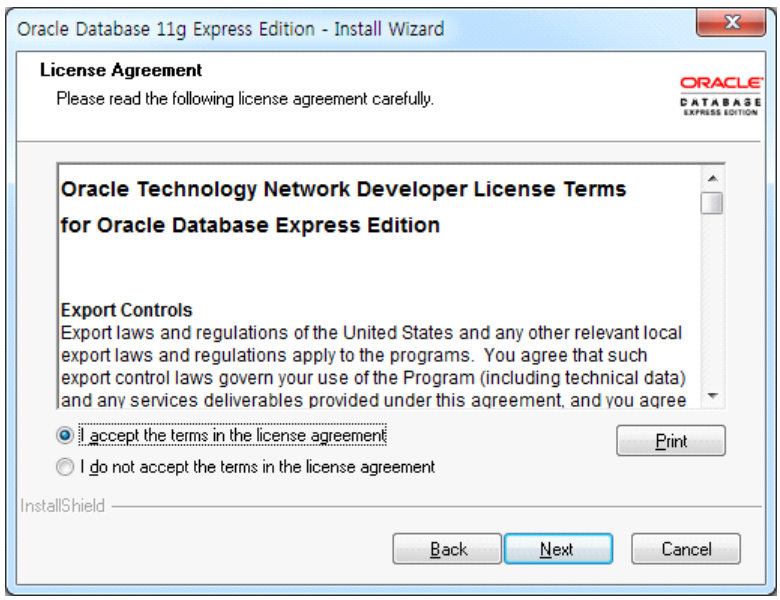


그림 44. 라이선스 동의

데이터베이스 설치 폴더를 선택할 수 있습니다. 여기에서는 기본 값을 설치 폴더로 사용합니다. [Next] 버튼을 클릭합니다.

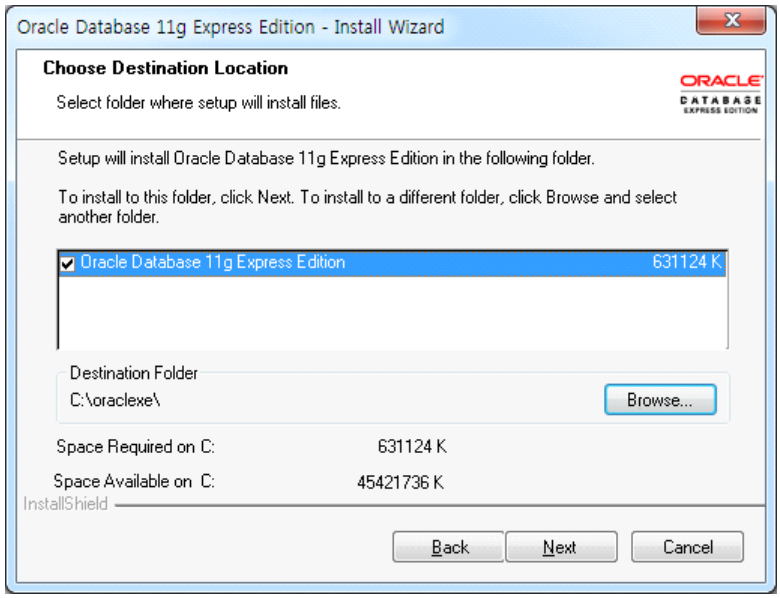


그림 45. 설치 폴더 선택

관리자(SYS, SYSTEM) 비밀번호를 입력하고 [Next]버튼을 클릭합니다.

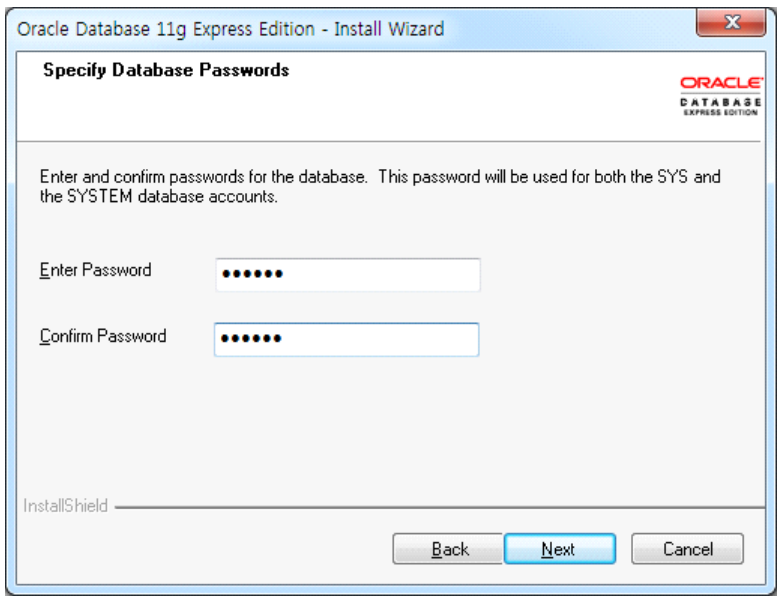


그림 46. 관리자(SYS, SYSTEM) 비밀번호 입력

설치 요약 내용을 확인한 다음 이상이 없다면 [Install] 버튼을 클릭합니다.

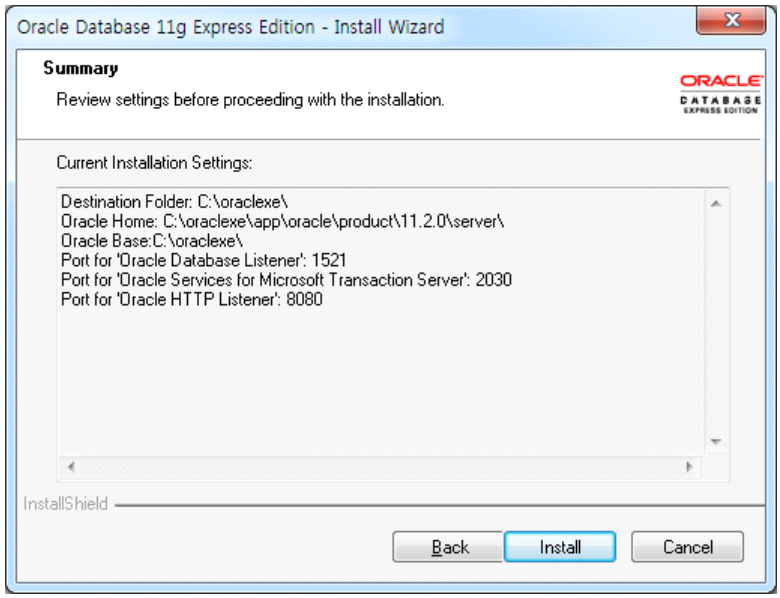


그림 47. 설치 요약

설치가 완료될 때 까지 잠시 기다립니다.

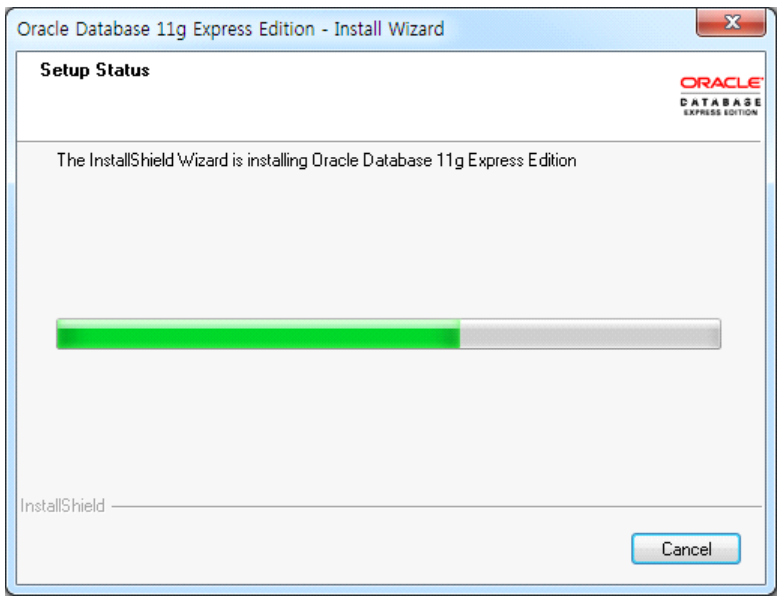


그림 48. 설치 중

설치가 완료되면 [Finish] 버튼을 클릭합니다.

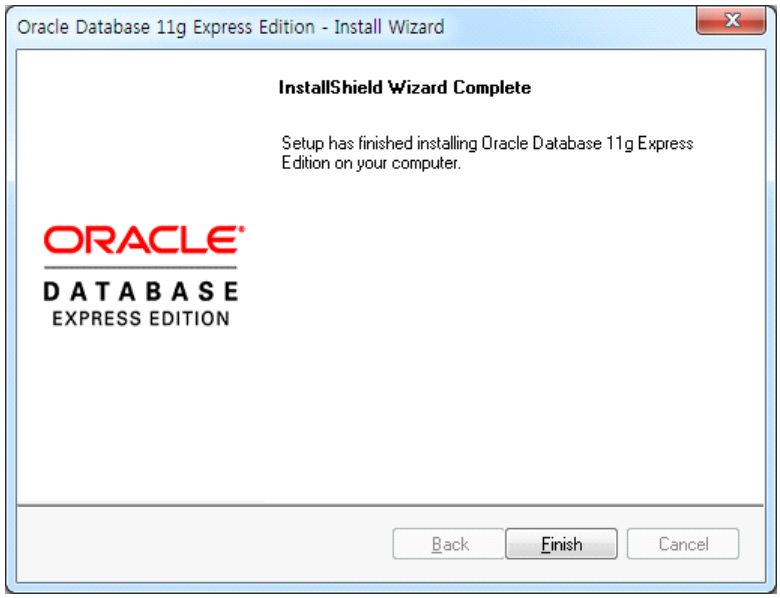


그림 49. 설치 완료

3.2.1.3. 설치 및 실행 확인

설치가 완료되면 C:₩oraclexe₩app₩oracle 폴더에 제품이 설치된 것을 확인할 수 있습니다.

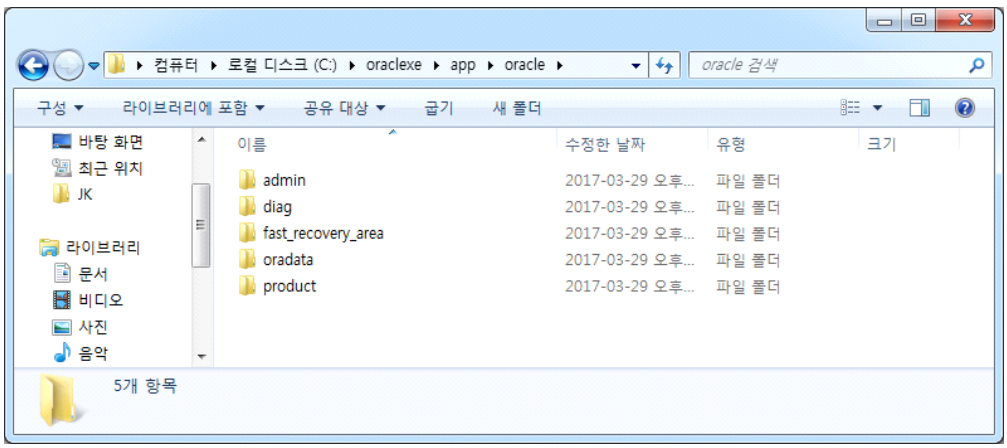
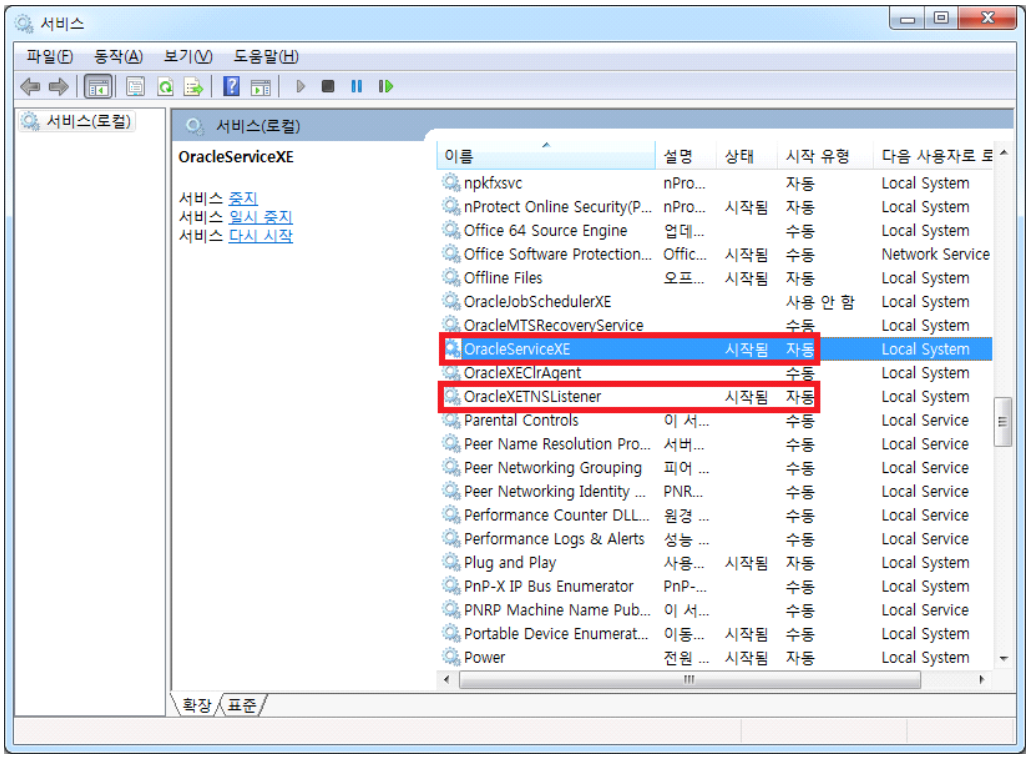


그림 50. 오라클XE 설치 폴더

오라클 데이터베이스가 정상 실행되고 있다면 [제어판] >> [서비스]에서 오라클 서비스(OracleServiceXE)와 리스너(OracleXETNSListener) 항목이 시작됨 상태여야 합니다.



3.2.2. HR 스키마 활성화

오라클XE에서는 HR 스키마를 이용할 수 있습니다. HR 스키마는 employees, departments 등의 테이블을 가지고 있습니다. 이 책에서는 HR 스키마의 employees 테이블과 departments 테이블 등을 실습에 사용합니다.

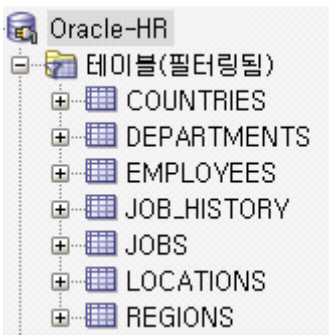


그림 52. 실습 테이블

오라클XE를 설치 한 다음 바로 HR 스키마를 사용할 수 없습니다. 오라클XE 설치 시 HR 스키마가 자동으로 구성되지만 스키마는 잠금 상태로 돼 있습니다. HR 스키마를 사용하기 위해선 스키마의 Lock을 풀고 비밀번호를 설정해야 합니다. HR 스키마의 잠금을 풀고 비밀번호를 설정하려면 [Run SQL Command Line]을 실행한 후 다음과 같이 입력합니다.

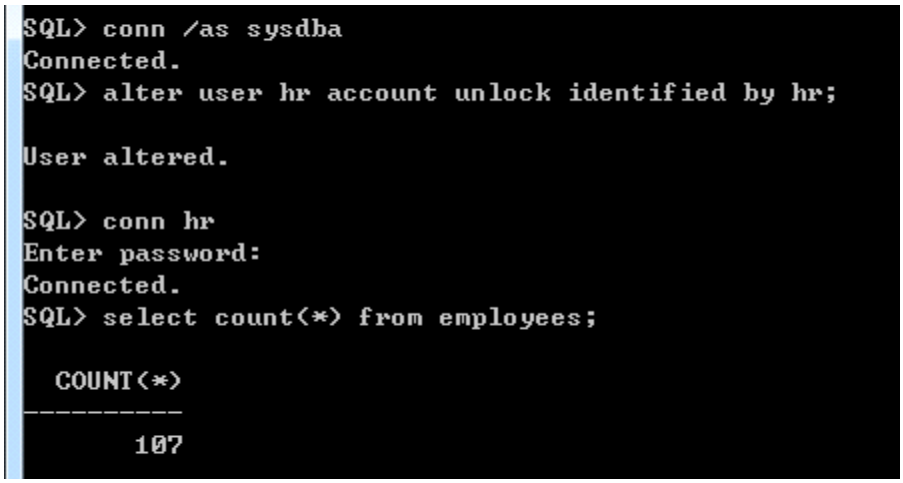


그림 53. HR 스키마 활성화

오라클에서 스키마는 객체들의 모음을 일컫는 말입니다. 오라클에서 스키마는 계정 이름과 동일합니다.

```
conn /as sysdba
```

이 명령은 로컬에서 sysdba 권한으로 데이터베이스에 접속합니다. 만일 이 명령으로 연결이 되지 않는다면 conn sys /as sysdba 명령을 입력한 다음 오라클XE 설치 시 입력했던 비밀번호를 입력합니다.

```
alter user hr account unlock identified by hr;
```

이 명령은 hr 계정의 락을 풀고, 비밀번호를 hr로 설정합니다. 명령어에서 앞의 hr은 계정의 이름이고, 뒤의 hr을 비밀번호를 의미합니다. 그러므로 여러분은 hr 계정의 비밀번호를 다른 비밀번호 바꿀 수 있을 것입니다.

```
conn hr
```

계정의 락을 풀고 비밀번호를 설정한 후 hr 계정으로 접속하세요. 비밀번호는 이 책대로라면 hr입니다.

```
select count(*) from employees;
```

employees 테이블의 레코드의 수를 확인해 봅니다.

3.2.3. SQL Developer

SQL Command Line을 이용하여 SQL문을 배우고 연습할 수 있습니다. 그러나 비슷하거나 동일한 쿼리문을 반복적으로 실행시켜야 한다면 SQL Command Line은 불편합니다. SQL Developer는 SQL을 좀 더 쉬운 환경에서 입력할 수 있도록 도와주는 도구입니다. SQL Developer는 오라클사 홈페이지(https://www.oracle.com)에서 다운로드 할 수 있습니다.

3.2.3.1. SQL Developer 다운로드

홈페이지에서 [Downloads] 하위 메뉴에서 SQL Developer 링크를 클릭합니다.

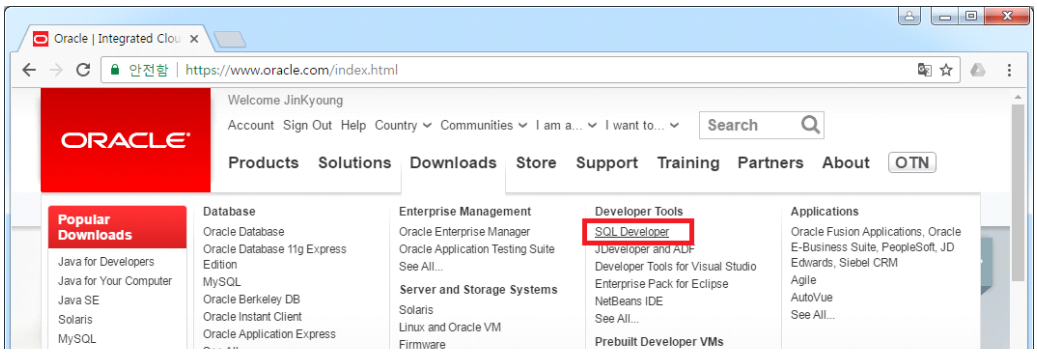


그림 54. 오라클사 홈페이지

라이선스에 동의한 후 SQL Developer를 다운로드 받습니다.

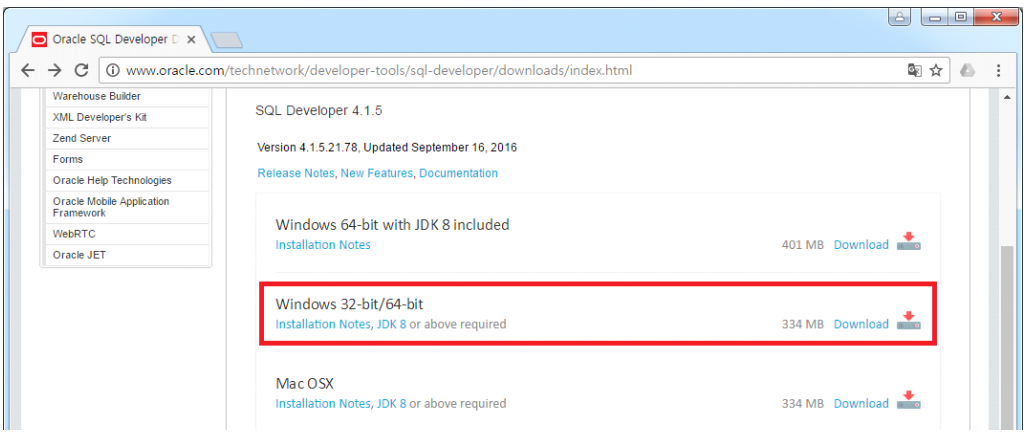


그림 55. Windows 32-bit/64-bit 다운로드

여러분의 컴퓨터가 64비트 컴퓨팅 환경이라면 [Windows 64-bit with JDK 8 included] 제품을 다운로드 할 수 있습니다. 만일 여러분의 컴퓨터가 32비트이거나 JDK 8이상을 이

미 설치했다면 [Windows 32-bit/64bit] 제품을 다운로드 하면 됩니다. 필자는 JDK를 포함하지 않은 SQL Developer [Windows 32-bit/64bit] 제품을 다운로드 했습니다. [Windows 32-bit/64bit]제품을 다운로드해서 사용하려면 JD 8 이상 설치해야 합니다.[3)]

3.2.3.2. SQL Developer 설치

다운로드 받은 압축파일을 적당한 곳에 압축만 풀어 놓으면 실행시킬 수 있습니다. 필자의 경우 C:₩Projects₩ 폴더에 압축을 풀어놓았습니다.

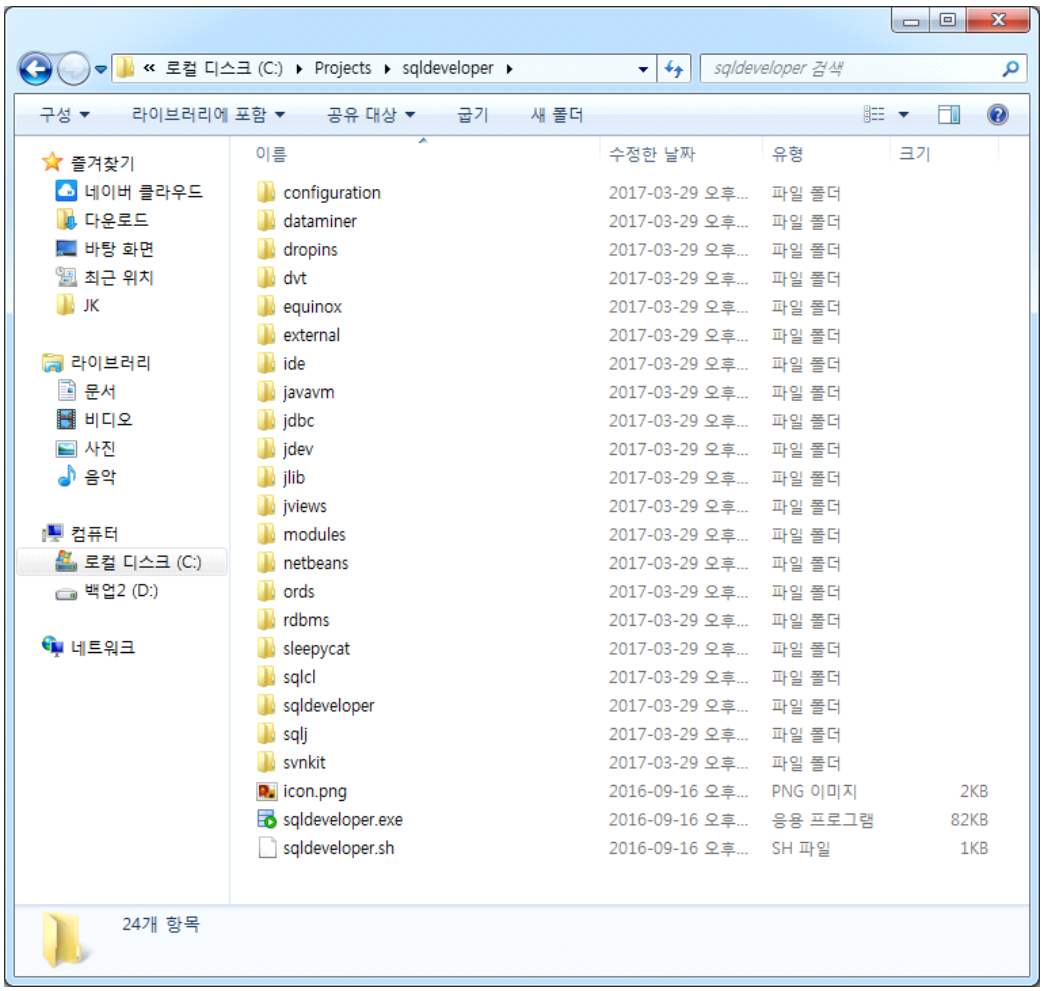


그림 56. 압축 풀어 놓은 SQL Developer

3) http://java.sun.com >> Java SE >> Java Platform (JDK) 8u121 >> 라이선스 동의 후 JDK 다운로드
http://www.oracle.com/technetwork/java/javase/downloads/index.html

3.2.3.3. SQL Developer 실행

SQL Developer 설치 폴더에서 sqldeveloper.exe 파일을 더블클릭하면 SQL Developer를 실행시킬 수 있습니다. JDK를 포함하지 않은 SQL Developer를 처음 실행하면 다음 그림처럼 JDK 홈디렉토리를 입력하는 창이 나타납니다. [Browse...] 버튼을 클릭하여 JDK 홈디렉토리 경로를 선택하여 입력한 후 [OK]버튼을 클릭합니다.

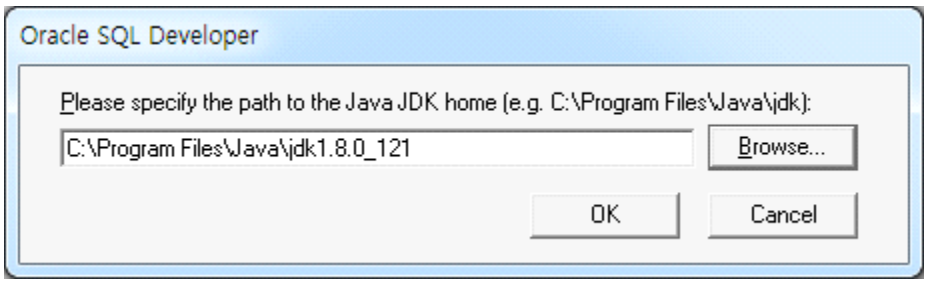


그림 57. JDK 홈디렉토리 설정

이전 SQL Developer 설치에서 설정했던 환경설정을 임포트 할지 묻는다면 [아니오(N)]을 클릭하면 됩니다.

SQL Developer가 시작되면 다음 그림과 같은 Oracle SQL Developer 시작페이지를 볼 수 있습니다.

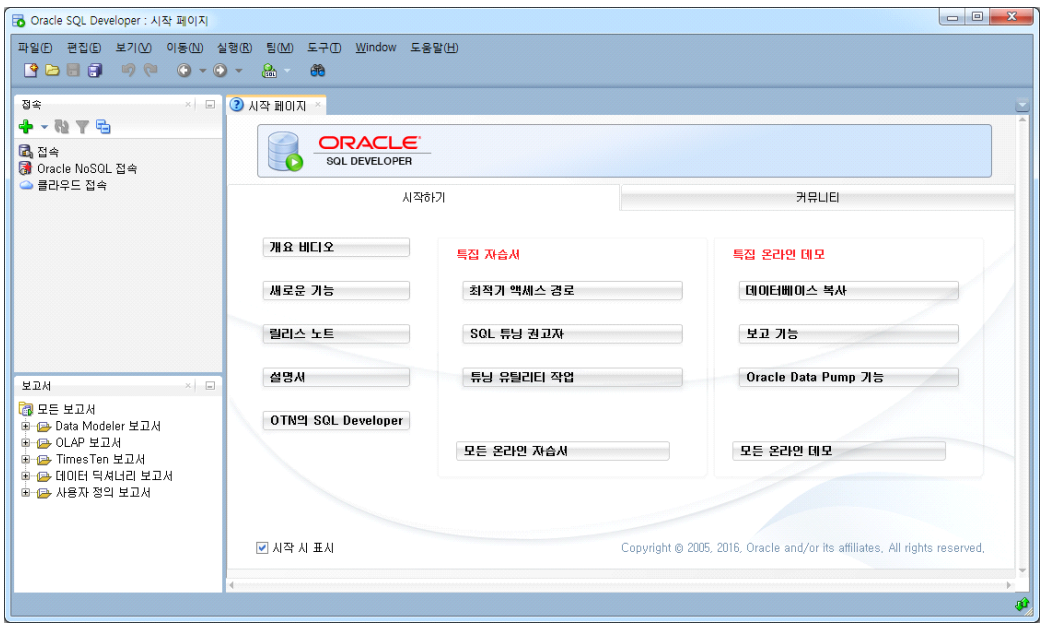


그림 58. SQL Developer 시작 페이지

3.2.3.4. SQL Developer 연결 설정

SQL Developer에서 오라클XE에 연결하기 위해 새 접속을 설정해야 합니다. 아래 그림에서 초록색 [+] 버튼을 클릭하면 새 접속을 설정할 수 있습니다.

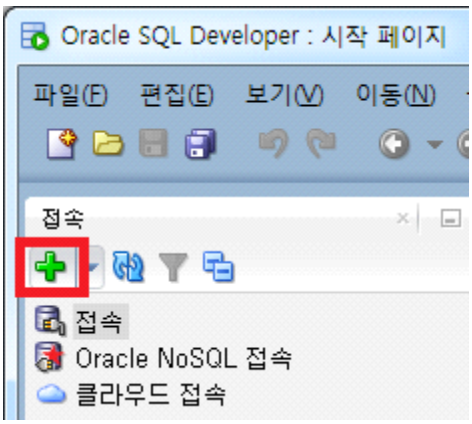


그림 59. 새 접속

새로 만들기/데이터베이스 접속 선택 창이 뜨면 [접속 이름], [사용자 이름], [비밀번호]를 입력합니다. [비밀번호 저장] 체크박스에 체크표시 해 놓으면 다음 접속 시 비밀번호를 묻지 않습니다. 아래 버튼들 중에서 [테스트] 버튼을 클릭했을 때 왼쪽 아래쪽에 상태: 성공이라고 보이면 연결정보를 바르게 입력한 것입니다. 상태: 실패가 보이면 연결 정보를 확인하세요. [접속] 버튼을 클릭하면 해당 데이터베이스에 연결합니다.

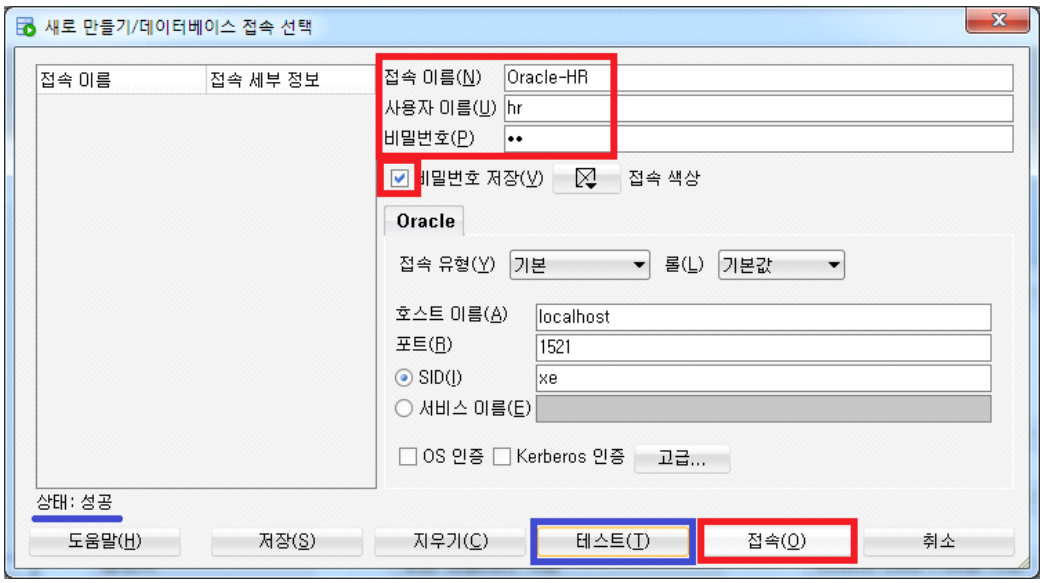


그림 60. 연결 정보 입력

3.2.3.5. 쿼리 실행

SQL Developer의 SQL 파일 탭의 워크시트에 쿼리문[4]을 입력 하고 Ctrl+Enter를 누르면 쿼리문이 실행되고 결과는 화면 아래 [질의 결과] 탭에 조회됩니다.

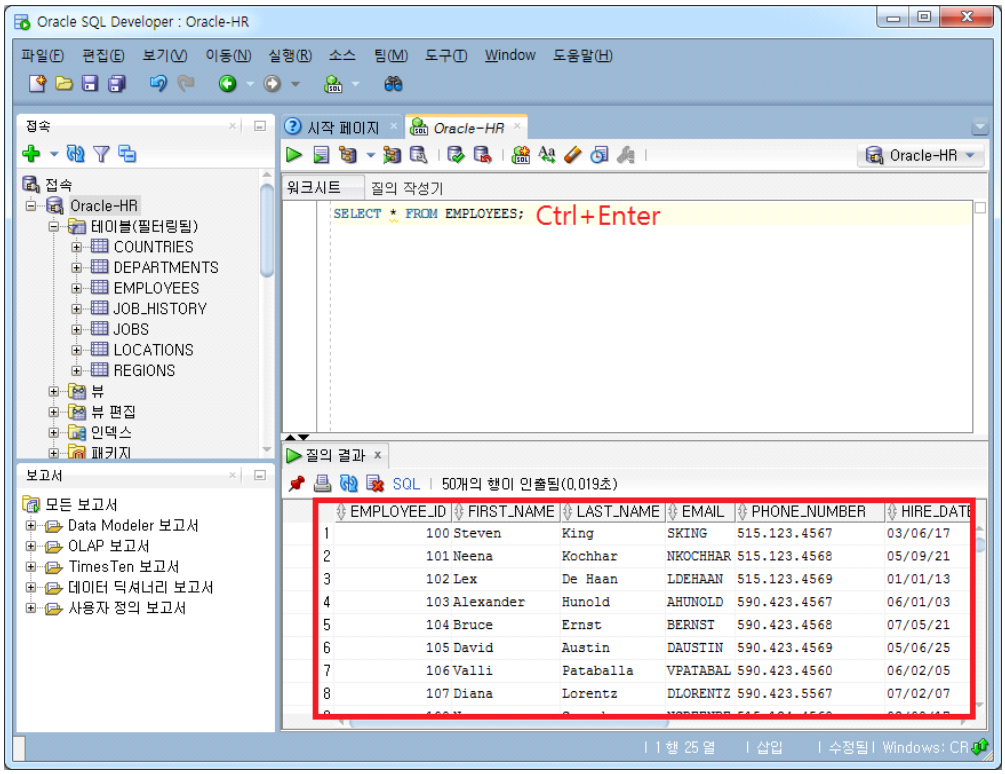


그림 61. SQL 문장 실행

4) SQL은 Structured Query Language입니다. 보통 줄여서 쿼리(Query) 문이라고도 부릅니다.

3.3. MariaDB

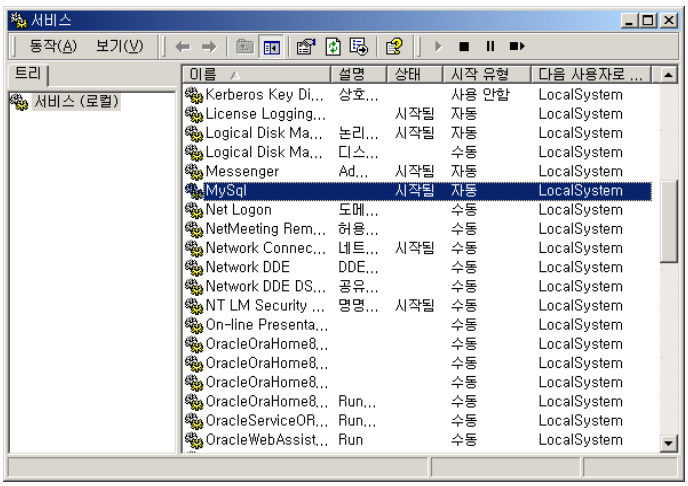
3.3.1. MariaDB

데이터베이스를 사용하기 위해서는 먼저 데이터베이스 엔진이 설치되어야 합니다. 본 교재에서는 MariaDB과 오라클을 기준으로 설명합니다. MariaDB 설치에 관해서는 언급하겠지만, 오라클의 설치에 관해서는 다루지 않겠습니다. 오라클은 설치 후 리스너(제어판 -> 서비스에서 확인) 정상적으로 동작하는지 확인해야 합니다.

3.3.1.1. MariaDB 설치

Ⅰ. https://downloads.mariadb.org/에서 최신 버전의 MariaDB을 다운로드 해 압축을 푼 다음 설치합니다. 윈도우용이라면 설치하기 어렵지 않을 것입니다.

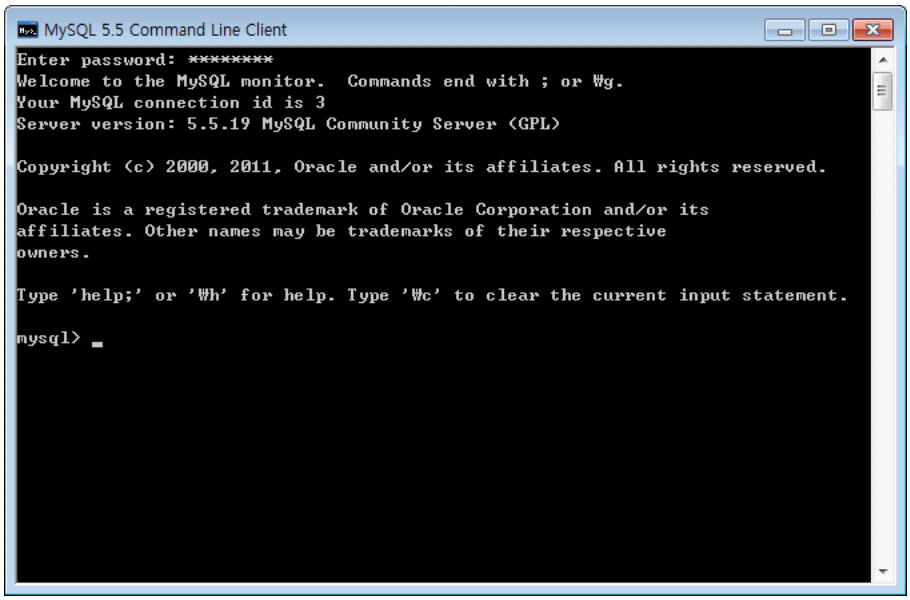
Ⅱ. 플랫폼에 설치했다면 제어판 -> 관리도구 -> 서비스에서 다음 그림과 같이 MariaDB 구성요소가 추가된 것을 볼 수 있습니다.



Ⅲ. 이제 MariaDB 데이터베이스로 접속할 수 있는 상태가 된 것입니다. MariaDB 데몬이 정상적으로 가동되었으면 시작메뉴에서 [MariaDB 5.5 Command Line Client]를 클릭하여 MariaDB 데이터베이스에 접속합니다. 입력하는 비밀번호는 처음 설치시 설정했던 비밀번호를 입력합니다. 아니면 직접 콘솔화면에서 mysql을 입력해도 됩니다.

```
C:\Program Files\MariaDB\MariaDB Server 5.5\bin>mysql
-u root -p
Enter password: ********
```

다음 그림과 같이 mysql> 프롬프트가 나타나면 접속된 것입니다.



3.3.1.2. 데이터베이스 설정

Ⅰ. MariaDB이 정상적으로 실행되었으면 테이블이 저장될 데이터베이스를 생성합니다. 데이터베이스는 다음과 같은 SQL문장으로 생성할 수 있습니다. 이 책에서는 데이터베이스 이름을 mydb로 설정했습니다.

```
mysql> create database mydb;
```

Ⅱ. 다음은 사용자를 추가시키기 위해 데이터베이스로 이동합니다. mysql이라는 데이터베이스에 사용자와 권한을 설정합니다. 다음처럼 use라는 명령을 이용하여 mysql 데이터베이스로 이동합니다.

```
mysql> use mysql
```

Ⅲ. 다음은 mysql이라는 데이터베이스 안에서 사용자를 추가합니다. 사용자의 아이디는 user, 비밀번호는 user123으로 설정합니다.

```
mysql> create user 'user'@'%' identified by 'user123';
```

Ⅳ. 사용자 user에게 권한을 부여합니다.

```
mysql> grant all privileges on *.* to 'user'@'%';
```

V. 이상의 설정 내용을 MariaDB이 새로 읽어들여 적용시키게 합니다.

```
mysql> flush privileges;
```

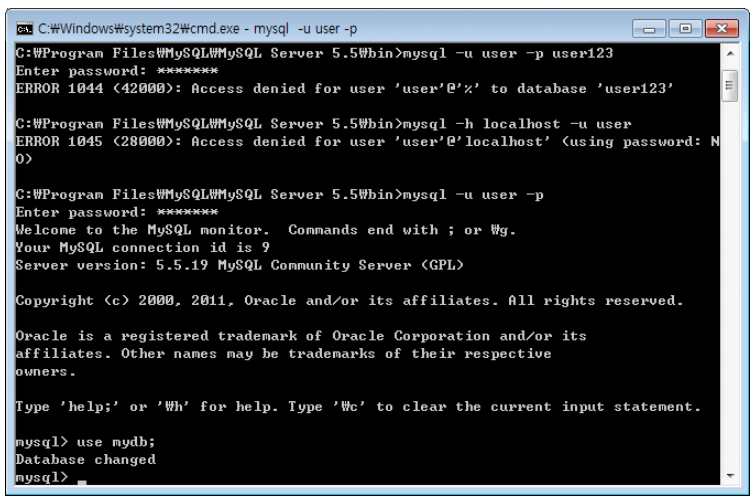
VI. 사용자가 추가되었는지 확인하기 위해 quit명령으로 MariaDB 클라이언트를 종료한 다음 새로 추가한 사용자 아이디와 패스워드로 다시 접속해 보세요.

```
mysql> quit
C:\Program Files\MariaDB\MariaDB Server 5.5\bin>mysql
-u user -p
Enter password: *******
```

VII. mydb 데이터베이스를 사용하기 위해 use명령을 이용하여 mydb라는 데이터베이스로 이동합니다.

```
mysql> use mydb;
```

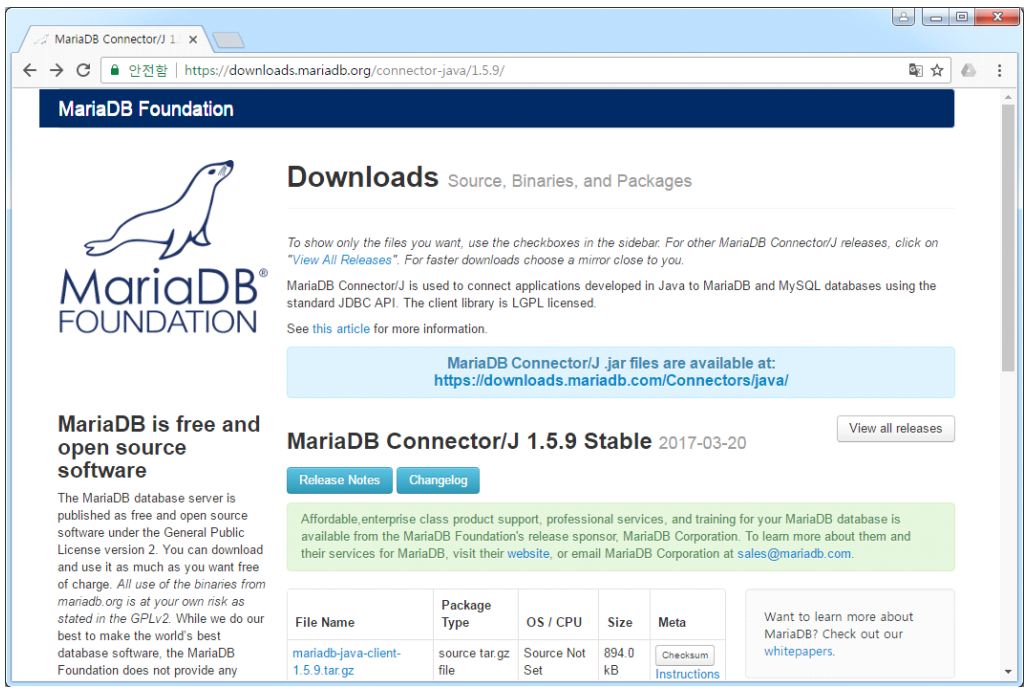
다음 그림과 같이 데이터베이스가 변경되었습니다. 이제 MariaDB 데이터베이스도 정상적으로 설치되었고, 사용자 아이디와 비밀번호도 만들어졌습니다.



3.3.1.3. MariaDB JDBC 드라이버 다운로드

https://downloads.mariadb.org/connector-java/1.5.9/ 에서 MariaDB Connector/J를 다운로드 받아야 합니다. Connector/J는 자바 응용프로그램에서 MariaDB 접속하여 데이

터베이스를 사용할 수 있도록 하는 JDBC 드라이버입니다. 윈도우 시스템이라면 zip 파일을 다운로드 받아 적당한 장소(C:₩java₩lib₩ 폴더)에 압축을 풀어 놓으세요. 개발환경이 이클립스나 NetBeans 등 IDE 툴에서 개발하고 있다면 드라이버 파일을 프로젝트 빌드패스(Build Path)에 포함[5]시켜줘야 합니다. MariaDB JDBC 라이브러리의 경우 파일명이 mariadb-java-client-1.x.y.tar.gz로 되어 있습니다.



5) 이클립스의 경우 프로젝트에서 마우스 오른쪽버튼 -> Build Path -> Add External Archives 선택한 다음 JDBC 라이브러리 jar 파일을 선택합니다.

3.4. SQL

3.4.1. SQL 문법의 개요

JDBC에 들어가기 전에 기본적으로 사용되는 SQL에 관한 기본적인 내용을 살펴보기로 하겠습니다. 사용하는 데이터베이스에 따라 약간의 차이가 있을 수 있으며 설명은 오라클과 MariaDB을 기준으로 예를 들겠습니다.

3.4.1.1. CREATE

테이블을 생성하는데 사용되며 사용방법은 다음과 같습니다.

```
CREATE TABLE table_name
   ( column_name datatype(size) [DEFAULT data] ,
... );
```

CREATE문의 경우에는 오라클과 MariaDB의 데이터 타입이 다르므로 질의문이 다를 수도 있습니다.

질의문 사용 예 - 오라클

```
1:  CREATE TABLE customer (
2:     id        NUMBER(4),
3:     name      CHAR(20),
4:     addr      CHAR(50)
5:  );
```

질의문 사용 예 - MariaDB

```
1:  CREATE TABLE customer (
2:     id        INT(4),
3:     name      CHAR(20),
4:     addr      CHAR(50)
5:  );
```

3.4.1.2. INSERT

테이블에 레코드를 삽입합니다.

```
INSERT INTO table_name ( column1, column2, column3... )
   VALUES ( 값1, 값2, 값3... );
```

질의문 사용 예

```
1:  INSERT INTO customer ( id, name, addr )
2:     VALUES ( 1, '홍길동', '서울' );
```

또는

```
1:  INSERT INTO customer VALUES( 1, '홍길동', '서울' );
```

데이터가 문자열 타입일 경우 name과 addr의 데이터처럼 단일 따옴표(')로 문자열을 감싸주어야 합니다.

3.4.1.3. SELECT

테이블에서 조건을 만족하는 레코드를 찾을 때 사용합니다.

```
SELECT [DISTINCT] column_name [, column_name, ...]
   FROM table_name
   [WHERE 조건식]
   [ORDER BY ASC|DESC];
```

Query문 사용 예

```
1:  SELECT * FROM customer WHERE id=1;
```

3.4.1.4. UPDATE

레코드 값을 갱신할 때 사용합니다.

```
UPDATE table_name
   SET column1 = 값1 [, column2 = 값2 ,...]
   [WHERE 조건식] ;
```

Query문 사용 예

```
1:  UPDATE  customer
2:     SET addr='NewYork'
3:     WHERE id=1 AND name='홍길동';
```

3.4.1.5. DELETE

레코드를 삭제할 때 사용합니다.

```
DELETE FROM table_name [WHERE 조건식];
```

Query문 사용 예

```
1:  DELETE FROM customer WHERE id=1;
```

3.4.1.6. DROP

테이블을 삭제할 때 사용합니다.

```
DROP TABLE table_name;
```

Query문 사용 예

```
1:  DROP TABLE customer;
```

3.4.1.7. COMMIT / ROLLBACK

COMMIT과 ROLLBACK은 트랜잭션처리에 사용하는데 이전에 지시한 명령을 확인(디스크에 저장)시키기 위해 COMMIT을 이용하고, 이전에 지시했던 명령을 취소시키는데 ROLLBACK을 사용합니다. 참고로 MariaDB에서는 트랜잭션처리를 할 수 없으므로 필요가 없지만 오라클에서는 반드시 COMMIT을 해 줘야 데이터가 변경됩니다. 사용 방법은 다음과 같습니다.

```
COMMIT;
```

```
ROLLBACK;
```

지금까지 데이터베이스에서 사용될 몇 가지 질의문에 대하여 알아보았습니다. 위에 언급되

지 않은 질의문에 대해서는 데이터베이스 관련 서적을 참고하길 바랍니다.

3.4.2. 데이터 타입(Oracle과 MariaDB)

데이터베이스 버전에 따라 사용하는 데이터타입이 다르기 때문에 데이터 타입에 대한 정보는 별도로 제공받아야 합니다. 다음은 오라클의 데이터 타입과 MariaDB의 데이터 타입을 요약해 놓은 것입니다.

3.4.3. Oracle 기본데이터 타입

CHAR(size) : 길이가 size인 고정길이 문자 값으로 최소길이는 1, 최대길이는 255입니다.
VARCHAR2(size) : CHAR의 확장형으로 최대길이는 4000입니다.
NUMBER : 38자리까지 유효한 부동소수점 숫자를 표현합니다.
NUMBER(p,s) : p는 전체자리수이고, 소수점 이하자리수가 s인 NUMBER를 표현합니다.
DATA : B.C.4712년 1월에서 A.D.4712년 12월 1일 사이의 일자와 시간을 표현합니다.
LONG : 2G바이트까지의 가변길이 문자 값으로, 테이블 당 한 개의 LONG열만 표현합니다.
RAW, LONGRAW : 각각 VARCHAR2, LONG과 같지만 이진 데이터를 저장하는데 표현합니다.

3.4.4. MariaDB 기본 데이터 타입

CHAR(size) : 최대 1~255사이의 길이를 갖는 문자 값을 표현합니다. 길이가 size 만큼 고정됩니다.
VARCHAR(size) : 최대 1~255사이의 길이를 갖는 문자 값을 표현합니다. 가변길이로 저장됩니다
TEXT : 최대 2^{16}(65535)의 길이를 갖는 문자열을 표현합니다. 대소문자를 구분하지 않습니다.
BLOB : 최대 2^{16}(65535)의 문자를 표현합니다. 대소문자를 구분합니다.
LONGTEXT : 최대 2^{32}(4G)의 길이를 갖는 문자열을 표현합니다.
LONGBLOB : 최대 입력이 2^{32}(4G)바이트 이진 데이터를 표현합니다.
INT(size) : 길이가 size인 정수를 표현합니다. 표현범위는 $-2^{31} \sim 2^{31}-1$까지 표현이 가능합니다.
FLOAT : 24자리까지 유효한 부동소수점을 표현합니다.
FLOAT(p,s) : 전체 자리수가 p이고, 소수점 이하 자리수가 s인 부동소수점을 표현합니다.

DOUBLE : 부동소수점을 표현합니다.
DATE : 1000-01-01부터 9999-12-31 까지 사이의 날짜를 표현합니다.

3.4.5. JDBC 환경설정

JDBC 환경에 앞서 JDK 환경설정이 이루어져야 합니다. 우선 JDK를 설치했으면, 어떤 데이터베이스엔진(DBMS)을 사용할 것인지를 결정해야 합니다. 현재 많이 사용되는 데이터베이스 엔진은 Oracle, MS-SQL, Sybase, Informix 등이 있고 무료로 사용할 수 있는 제품은 mSQL, MariaDB 등이 있습니다. 상용으로 쓰이는 엔진이라도 평가판은 무료로 이용할 수 있습니다. 각 데이터베이스 회사의 홈페이지를 이용하면 쉽게 다운로드 받을 수 있습니다.
사용할 데이터베이스를 결정했으면 해당 소프트웨어를 다운로드하여 설치합니다. 이때 해당 JDBC 드라이버 클래스도 함께 다운로드 해야 합니다. 주로 .jar이나 .zip 형태로 클래스가 묶여 있습니다. 오라클은 8i와 9i버전의 경우 classes12.zip파일로 되어 있으며, 10g, 11g 에서는 JDBC API 버전에 따라 ojdbc14.jar, ojdbc15.jar 등의 파일로 되어 있습니다. MariaDB은 버전에 따라 조금씩 다르지만 mm.mysql-X.X.X-bin.jar 파일로 되어있습니다.
데이터베이스를 설치한 후에 해당 JDBC 드라이버 파일이 있는 디렉터리를 classpath에 설정하거나, JDK 1.2 이상의 환경에서는 JDK가 설치되어 있는 디렉터리 아래의 jre/lib/ext에 복사해도 됩니다. 하지만 컴파일시 클래스파일을 찾지 못하면 따로 classpath 환경변수를 설정해주어야 합니다. 클래스패스 환경변수 설정시 현재 폴더(.)와 JDK가 설치된 폴더도 같이 설정하는 것을 잊지 마세요.

이클립스를 사용한다면 프로젝트 Properties 창에서 Java Build Path -> Libraries 탭 -> Add External JARs...를 선택한 다음 드라이버 파일을 추가해주면 됩니다.

데이터베이스와 JDBC 드라이버 설치를 마치면 교재에서 예제로 사용할 테이블을 만들어야 합니다.
오라클에서는 SQL*Plus를 이용하여 다음과 같은 SQL을 실행시켜 테이블을 만듭니다.

/* customer table sql - Oracle */

```
1:  CREATE TABLE customer (
2:     id     NUMBER(4) PRIMARY KEY,
3:     name   VARCHAR2(20),
4:     addr   VARCHAR2(50)
5:  );
```

MariaDB은 MariaDB을 실행시키고 use명령으로 앞에서 만들었던 mydb 데이터베이스로 이동한 후 다음과 같은 질의문을 실행시켜 테이블을 만듭니다.

/* customer table sql - MariaDB */

```
1:  CREATE TABLE customer (
2:     id     INT(4) PRIMARY KEY,
3:     name      CHAR(20),
4:     addr      CHAR(50)
5:  );
```

테이블이 생성되었으면 다음의 질의문으로 테이블에 데이터를 입력합니다.

/* data insert */

```
1:  INSERT INTO customer (id, name, addr) VALUES(1, '홍길동', '서울시');
2:  INSERT INTO customer (id, name, addr) VALUES(2, '허현준', '광주시');
3:  INSERT INTO customer (id, name, addr) VALUES(3, '허현수', '대전시');
4:  INSERT INTO customer (id, name, addr) VALUES(4, '허현정', '부산시');
```

자료 입력이 끝났으면 오라클의 경우에는 commit;를 입력하여 디스크에 저장합니다.

```
1:  COMMIT;
```

MariaDB에서는 위의 COMMIT 질의문이 필요 없습니다.

다음 프로그램은 앞에서 만든 데이터베이스에 연결하여 원하는 결과를 얻는 예를 보인 것입니다. 수정할 곳이 있으면 자신의 환경에 맞게 수정한 후 컴파일 합니다.

exam/java/chapter10/SimpleJDBCExample.java

```
1:  //수정할 사항 (URL)
2:  //- "127.0.0.1" 은 데이터베이스가 설치된 Server의 주소
3:  //- "orcl"은 Oracle DB의 SID 즉 시스템 id
4:  //- "mydb"는 MariaDB에서 사용자가 만든 database
5:  //- "user","user123"는 DB 사용자 id와 password
6:
7:  package exam.java.chapter10;
8:
9:  import java.sql.*;
10:
11: public class SimpleJDBCExample {
12:
13:    public static void main(String[] args) {
14:
15:       try {
16:          Class.forName("oracle.jdbc.driver.OracleDriver");
17: //       Class.forName("org.gjt.mm.mysql.Driver"); //MariaDB
18:          System.out.println("드라이버가 로드됐습니다.");
19:       } catch (ClassNotFoundException e) {
20:          System.out.println("드라이버 클래스를 찾을 수 없습니다.");
21:          System.out.println("클래스 이름을 정확히 입력했는지 확인하세요.");
22:          System.out.println("프로젝트 빌드 패스에 드라이버가 설정됐는지 확인하세요.");
23:       }
24:
```

```
25:        Connection conn = null;
26:        try {
27:           conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@127.0.0.1:1521:xe", "hr", "hr" );         //Oracle
28: //        conn = DriverManager.getConnection(
    "jdbc:mysql://127.0.0.1:3306/mydb", "user", "user123" ); //MariaDB
29:           System.out.println("커넥션이 연결됐습니다.");
30:
31:           Statement stmt = conn.createStatement();
32:           String sql = "SELECT id, name, addr FROM customer";
33:
34:           ResultSet rset = stmt.executeQuery(sql);
35:
36:           System.out.println("id\tname\taddr");
37:           while(rset.next()) {
38:              System.out.print(rset.getInt(1));
39:              System.out.print("\t" + rset.getString(2));
40:              System.out.println("\t" + rset.getString(3));
41:           }
42:        } catch (SQLException e) {
43:           System.out.println(e.getMessage());
44:        } finally {
45:           if(conn != null) try {conn.close();} catch (SQLException e) {}
46:        }
47:     }
48: }
```

실행 결과

id	name	addr
1	홍길동	서울시
2	허현준	광주시
3	허현수	대전시
4	허현정	부산시

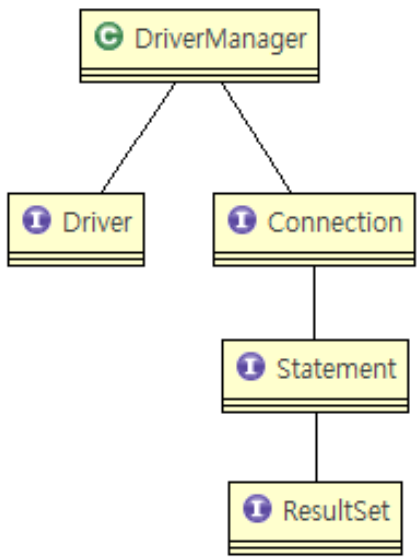
앞의 프로그램을 실행시켰을 때에 예외가 발생하지 않고 정상적으로 수행되면 JDBC 개발 환경 및 실행환경을 잘 구축한 것입니다. 예외가 발생한다면 에러 메시지를 보고 해결하기 바랍니다. 예상되는 예외상황은 드라이버가 없을 경우, URL에 오타가 발생한 경우, 아이디/비밀번호가 잘못된 경우, 테이블이름 또는 열 이름이 잘못된 경우 등입니다. 프로그램에 대한 자세한 설명은 뒤에서 하겠습니다.

3.5. JDBC API

JDBC 1.0 스펙에서는 java.sql 패키지를 제공하였고, JDBC2.0에서는 몇몇 기능이 추가된 java.sql 패키지와 javax.sql 패키지를 제공하고 합니다. java.sql 패키지는 J2SE에 포함되어 있으며 CoreAPI라하고, javax.sql 패키지는 J2EE에 포함되어 있으며 Standard extension API라고 합니다. JDK 1.4버전에서는 JDBC 스펙 3.0까지 지원하고 있는데 JDK 1.4 버전 이후의 Java SE API에는 java.sql 패키지와 javax.sql 패키지를 모두 포함하고 있습니다.

3.5.1. JDBC API

자바에서 인터페이스는 서로 주고받는 표준이라고 표현할 수 있습니다. 그래서 서로 같은 인터페이스를 사용하면 상대방이 누구이건 내가 알고 있는 방법으로 통신을 할 수 있습니다. JDBC는 이러한 인터페이스의 사상을 잘 구현하고 있는 예입니다. 데이터베이스엔진 개발자는 java.sql 패키지에 있는 드라이버 인터페이스를 구현하여 자바와 자신의 DBMS를 연결할 수 있는 드라이버를 만들고, 자바 프로그램 개발자는 java.sql 패키지에 있는 인터페이스와 클래스를 가지고 데이터베이스를 사용하는 프로그램을 개발하게 됩니다.
JDBC 프로그램에 쉽게 적응하려면 java.sql 패키지에 있는 클래스, 인터페이스 및 관련 메서드를 잘 이해해야 합니다. 다음 그림은 JDBC 클래스 및 인터페이스들의 의존관계를 나타낸 것입니다.



- DriverManager : JDBC 드라이버를 관리하는 클래스입니다. 드라이버를 등록하고, Connection 객체를 반환 받을 때 사용합니다.
- Driver : 데이터베이스를 만든 회사(Oracle, Informix, Sybase, 등)에서 데이터베이스를 연결할 수 있는 JDBC 드라이버 클래스를 만들기 위한 인터페이스입니다.
- Connection : 데이터베이스와 연결을 가지고 있는 인터페이스입니다. 프로그래머는 데이터베이스에 연결하기위해 DriverManger 클래스의 getConnection() 메서드를 사용하여 커넥션 개체를 반환 받아 데이터베이스와 연결합니다.
- Statement : SQL문을 실행하기 위해 Connection으로부터 생성하는 객체입니다. SQL문을 저장하기 위한 인터페이스입니다.
- ResultSet : Statement를 통해 SQL SELECT문을 실행한 결과 데이터를 갖게 하는 인터페이스입니다. 한 개의 Statement는 한 개의 ResultSet 공간만 할당받습니다.

3.5.2. JDBC 드라이버 타입

JDBC 드라이버 타입은 다음과 같이 4가지가 있습니다.

3.5.2.1. 타입 I 드라이버(JDBC-ODBC Bridge Driver)

ODBC 드라이버를 연결합니다. 이 타입의 드라이버는 마이크로소프트의 장점을 살리기 위해 제공되는 드라이버입니다. 마이크로소프트의 표준인 ODBC 드라이버를 연결하고 이를 통하여 데이터베이스를 접근하는 방식이기 때문에 JDBC-ODBC 브릿지(Bridge) 드라이버라고도 합니다. 타입 I 드라이버를 사용하려면 클라이언트 컴퓨터에 ODBC 드라이버와 JDBC 드라이버가 설치되어 있어야 합니다. 이 때문에 동적으로 드라이버가 다운로드 되는 환경에서는 사용하기 어렵고 고정된 드라이버를 사용하는 환경에 사용할 수 있습니다. JDBC 1.2부터 이 드라이버는 기본으로 장착되어 있으므로 따로 설치할 필요가 없습니다. JVM에 기본적으로 제공하는 JDBC-ODBC 드라이버는 sun.jdbc.odbc.JdbcOdbcDriver 입니다. 이 드라이버의 장점은 거의 모든 데이터베이스에 사용이 가능하다는 것입니다. 단점으로는 ODBC를 사용하므로 100% 자바가 아닐 수 있습니다. 이는 프로그램이 데이터베이스에 의존적이고 또한 플랫폼 독립적이지 않다는 것을 의미합니다.

3.5.2.2. 타입 II 드라이버(Native API Partly Driver)

타입 II 드라이버는 일부가 자바로 되어있는 JDBC 드라이버로 데이터베이스 회사에서 제공해 주는 드라이버입니다. 따라서 플랫폼 의존적입니다. 전체가 자바로 구현되지는 않았지만 수행능력이 뛰어나 속도가 빠르다는 장점을 가지고 있습니다. 타입 III 드라이버와 마

찬가지로 클라이언트 컴퓨터에 설치하는 드라이버가 필요합니다. 이러한 형식은 오라클, 인포믹스(Informix), DB2 등 DBMS 회사가 제공하는 API를 다시 호출하게 되는데, 타입 Ⅰ 드라이버와 마찬가지로 클라이언트 컴퓨터에 정적으로 설치하여 데이터베이스와 연동하는 환경에 적합합니다. 오라클의 oci타입이 이에 해당합니다.

3.5.2.3. 타입 Ⅲ 드라이버(JDBC-NET Pure Java Driver)

Type Ⅲ 드라이버는 프로토콜이나 플랫폼에 무관한 완전한 자바 기술 및 코드로 이루어진 JDBC 드라이버를 의미합니다. 이 타입은 JDBC API 표준에 따라 만들어졌기 때문에 DBMS의 종류에 상관없이 사용할 수 있습니다. 4가지 타입 중에서 가장 융통성이 뛰어나기 때문에 동적으로 내려 받는 애플릿과 같은 환경에 적합하며, 클라이언트 컴퓨터가 정적으로 이루어졌다고 하더라도 시스템 유지보수가 쉬워집니다. 단점이라면 유료입니다. 특별한 요구사항이 필요한 경우 즉, 특정 DBMS에만 제공되는 기능을 사용할 때 또는 DBMS 자체가 타입 Ⅲ 드라이버를 지원하지 않는 경우를 제외하고는 타입 Ⅲ 드라이버를 사용하는 것이 가장 이상적입니다.

3.5.2.4. 타입 Ⅳ 드라이버(Native Protocol Pure Java)

Type Ⅳ 드라이버는 데이터베이스 회사의 프로토콜을 사용하며, 전체가 자바 기술 및 코드로 이루어진 JDBC 드라이버를 의미합니다. Type Ⅲ 드라이버와는 다른 점은 JDBC API 표준에 따르지 않고 DBMS 회사가 표준에 기능을 추가하거나 삭제하여 독자적인 형태로 만든 타입입니다. 따라서 특정 DBMS에 의존적인 반면에 해당 DBMS만이 제공하는 기능을 사용할 수 있는 장점이 있습니다. 클라이언트가 동적인 상황에서 특정 DBMS만 제공하는 기능을 사용할 때 적합합니다. 거의 대부분의 데이터베이스 사이트에서 무료로 다운로드 받아 사용할 수 있습니다. 플랫폼 독립적인 프로그래밍이 가능하지만 속도가 느리다는 단점도 있습니다, 본 교재에서는 바로 타입 Ⅳ 드라이버를 사용합니다. 참고로 오라클은 thin타입에 해당합니다. 대부분의 데이터베이스 벤더들은 타입 Ⅳ드라이버를 제공하고 있습니다.

3.6. JDBC 프로그램 구조

JDBC 프로그램을 이해하려면 데이터베이스를 사용하는데 필요한 몇 가지 절차를 확실하게 이해해야 합니다.

JDBC 프로그램 작성과정

1. 드라이버를 로딩합니다.
2. 데이터베이스와 연결하여 Connection 객체를 반환받습니다.
3. Connection으로부터 SQL을 실행하기 위한 Statement 객체를 반환받습니다.
4. Statement 객체의 메서드를 이용하여 SQL을 실행합니다.
5. SQL 실행결과인 ResultSet 객체를 반환받습니다.(SELECT문 경우)
6. ResultSet 객체의 결과를 반복문을 이용하여 처리합니다.

3.6.1. 드라이버 로딩(Driver Loading)

JDBC드라이버를 로딩하는 방법은 크게 4가지가 있습니다. 이 책에서는 세 번째 방법을 사용하겠지만 다른 방법도 참고로 알고 있으면 좋을 것입니다.

[방법 1]
JDBC 드라이버를 설치한 후 프로그램에서 해당 드라이버를 로딩하려면 다음과 같이 java.sql 패키지에 있는 DriverManager의 registerDriver() 메서드를 이용합니다.

Driver Manager에 등록 - Oracle

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

Driver Manager에 등록 - MariaDB

```
DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());
```

new oracle.jdbc.driver.OracleDriver()는 해당 드라이버의 인스턴스를 생성합니다. 명칭이 길어서 불편하지만 오라클 사에서 만든 클래스 이름이기 때문에 그대로 사용할 수밖에 없습니다. 만약 다른 데이터베이스를 사용한다면 해당 데이터베이스의 JDBC 메뉴얼을 통하여 어떤 이름으로 드라이버의 인스턴스를 만들어야 하는지 확인해야 합니다.

[방법 2]
DriverManager클래스를 사용하지 않고 다음과 같이 해당 드라이버의 인스턴스만 생성해도 자동으로 로딩됩니다.

해당 드라이버의 인스턴스만 생성 - Oracle

```
new oracle.jdbc.driver.OracleDriver()
```

해당 드라이버의 인스턴스만 생성 - MariaDB

```
new org.gjt.mm.mysql.Driver();
```

[방법 3]
System 패키지의 Class 클래스의 API를 이용하면 문자열을 이용하여 드라이버를 로딩할 수 있습니다.

Class.forName() 사용 - Oracle

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Class.forName() 사용 - MariaDB

```
Class.forName("org.gjt.mm.mysql.Driver");
```

Class.forName() 메서드는 ClassNotFoundException을 발생하므로 해당하는 예외처리를 해 주어야 합니다.

DriverManager에 등록하거나 Class.forName()을 이용해도 모두 비슷하지만 new를 이용할 때에는 먼저 클래스 파일을 컴파일할 때 체크해야하고, Class.forName()을 이용할 때는 실행시 체크해야 합니다. 또한 new를 이용하여 객체를 직접 생성할 경우에는 해당하는 드라이버의 static 블록에서도 객체를 만들기 때문에 사실상 객체가 두 개씩 만들어진다. 따라서 Class.forName()을 사용하는 것이 더 편리하다.

[방법 4]
다음과 같이 프롬프트 상에서 자바를 실행하면서 JVM의 jdbc.drivers 프로퍼티를 설정하는 방법도 있습니다.

System Property 사용 - Oracle

```
> java -D jdbc.drivers=oracle.jdbc.driver.OracleDriver SimpleJDBC
```

System Property 사용 - MariaDB

```
> java -D jdbc.drivers=org.gjt.mm.mysql.Driver SimpleJDBC
```

3.6.2. Connection 객체 생성

Connection 객체를 얻는 방법은 DriverManager 클래스의 getConnection() 메서드를 이용합니다. 하지만 시스템에 로딩된 어떤 드라이버의 Connection을 가져올 것인지는 URL을 이용하여 결정해야 합니다. (JVM은 여러 가지 드라이버가 로딩될 수 있기 때문.) 이

외에 데이터베이스 사용자 아이디와 암호를 입력해야 합니다.

Connection 객체 생성

```
Connection con = DriverManager.getConnection( URL, ID, PASSWD );
```

URL - Oracle : jdbc:oracle:thin:@127.0.0.1:1521:*database_SID*
MariaDB : jdbc:mysql://127.0.0.1:3306/*database_name*
ID : 아이디
PASSWD : 비밀번호

URL(Unified Resource Locator)은 DBMS가 설치된 컴퓨터가 어디인가를 알려주는 역할을 하는데, JDBC 드라이버를 만든 회사측에서 제공하는 고유의 String을 알아야 하기 때문에 해당 드라이버의 JDBC 메뉴얼을 참고해야 합니다.

3.6.3. Statement 객체생성

마지막으로 SQL을 수행하려면 Statement 객체를 Connection 객체로부터 얻어야 합니다.

Statement 객체 생성

```
Statement stmt = conn.createStatement();
```

3.6.4. Statement 객체의 메서드를 이용한 SQL 실행

데이터베이스를 이용하려면 SQL문을 문자열로 만들어 실행해야 합니다.

SELECT문

```
String sql = "SELECT id, name, addr FROM customer";
ResultSet rset = stmt.executeQuery(sql);
```

SELECT 문은 Statement객체의 executeQuery() 메서드를 사용합니다. 반환 값은 ResultSet 객체입니다.

DML(INSERT, UPDATE, DELETE) 및 DDL(CREATE TABLE, DROP TABLE, ALTER TABLE, TRUNCATE TABLE)

```
String sql = "INSERT INTO customer ( id, name, addr ) VALUES ( 5, 'JAMES', 'NEW YORK'
)";
int  count = stmt.executeUpdate(sql);
```

Statement 객체의 executeUpdate() 메서드를 사용합니다. 반환값은 처리된 행(row)의 개수입니다.

3.6.5. 질의 결과를 얻기위한 SELECT문

SELECT문 이외의 질의문을 사용하려면 executeUpdate()메서드를 이용하는데 이때의 결과는 질의문이 적용된 행의 수를 반환합니다. 하지만 SELECT 문일 경우 Statment.executeQuery() 메서드의 반환값이 ResulSet이므로 다음과 같이 ResultSet 객체를 반환합니다.

SELECT문

```
String sql = "SELECT id, name, addr FROM customer";
ResultSet rset = stmt.executeQuery(sql);
```

3.6.6. 데이터 추출

executeQuery() 메서드로부터 ResultSet 객체를 얻은 다음 테이블에서 원하는 결과를 리턴받습니다. ResultSet은 2차원 테이블에서 결과를 유도해 낼 수 있습니다. 첫 번째 레코드에 접근하려면 next() 메서드를 호출하는데 레코드가 있으면 true를 반환하고, 없으면 false를 반환합니다. 그 다음 레코드도 마찬가지로 next()를 이용하여 다음 레코드로 이동합니다.
원하는 데이터를 얻기 위해서는 getString(), getInt(), getDate(), getObject() ... 등의 메서드를 데이터 타입에 맞도록 사용할 수 있습니다. SELECT된 한 행에서 데이터의 순서는 0부터 시작하는 것이 아니고 1부터 시작합니다. getXxx() 메서드의 인자로 결과 열의 순서가 아닌 열 이름을 줄 수 있습니다.

Looping 처리

```
while(rset.next()) {
    System.out.println(rset.getInt(1));
    System.out.println(rset.getString(2));
    System.out.println(rset.getString("addr"));
}
```

3.7. 실전 JDBC API

다음은 몇 가지 유용하게 사용되는 API를 설명하기로 하겠습니다. API문서를 참고하면 데이터베이스 관련 작업을 더 쉽게 할 수 있습니다. 이후의 코드는 오라클과 MariaDB 데이터베이스를 예로 들었습니다. 만약 다른 데이터베이스를 사용하려면 해당하는 드라이버로 바꾸어 실행해야 합니다.

3.7.1. PreparedStatement

같은 SQL을 반복 사용하는 경우 문자열로 만드는 것은 번거로운 작업입니다. 따라서 이런 경우 PreparedStatement 인터페이스를 사용할 수 있습니다.
PreparedStatement는 SQL 문자열을 사용하여 미리 준비해 놓는 것을 의미합니다. 쿼리문 실행 도중에 동적으로 변경되어야 할 값은 "?"를 사용하여 표현하고 SQL을 실행하기 전에 "?" 값을 setXxx()함수를 이용하여 치환하는 방식입니다. setXxx()함수는 setInt(), setLong(),setString().. 등 데이터 타입에 따라 다양한 형태를 취하며 첫 번째 인자는 ?의 순서를 지정합니다. 그리고 두 번째 인자는 ? 위치에 입력될 값을 설정합니다.

다음 코드를 분석하면 PreparedStatement 의 쓰임세를 알 수 있습니다.

exam/java/chapter10/jdbc/PreparedExample.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.sql.*;
4:
5:  public class  PreparedExample {
6:     public static void main(String[] args) {
7:        try {
8:           Class.forName("oracle.jdbc.driver.OracleDriver");
9:           System.out.println("드라이버가 로드됐습니다.");
10:       } catch (ClassNotFoundException e) {
11:          System.out.println("드라이버 클래스를 찾을 수 없습니다.");
12:          System.out.println("클래스 이름을 정확히 입력했는지 확인하세요.");
13:          System.out.println("프로젝트 빌드 패스에 드라이버가 설정됐는지 확인하세요.");
14:       }
15:
16:       Connection conn = null;
17:       try {
18:          conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@127.0.0.1:1521:xe", "hr", "hr" );         //Oracle
19:          System.out.println("커넥션이 연결됐습니다.");
20:
21:          Statement stmt = conn.createStatement();
22:
```

```
23:           int[] id = { 100, 200, 300 };
24:           String[] name = { "AAA", "BBB", "CCC" };
25:           String[] addr = { "Seoul", "Pusan", "Gwangju" };
26:
27:           String sql = "INSERT INTO customer (id, name, addr) VALUES ( ?, ?, ? )";
28:
29:           PreparedStatement pstmt = conn.prepareStatement(sql);
30:
31:           for(int i=0; i < id.length; i++) {
32:              pstmt.setInt(1, id[i]);
33:              pstmt.setString(2, name[i]);
34:              pstmt.setString(3, addr[i]);
35:              pstmt.executeUpdate();
36:           }
37:
38:           ResultSet rset = stmt.executeQuery("SELECT id, name, addr FROM
    customer");
39:
40:           while(rset.next()) {
41:              System.out.println("DB로부터 데이터를 가져옵니다.");
42:              System.out.println("id: " + rset.getInt(1));
43:              System.out.println("name: " + rset.getString(2));
44:              System.out.println("addr: " + rset.getString(3));
45:           }
46:        } catch (SQLException e) {
47:           System.out.println(e.getMessage());
48:        } finally {
49:           if(conn != null) try {conn.close();} catch (SQLException e) {}
50:        }
51:     }
52: }
```

3.7.2. CallableStatement

대부분의 상용 DBMS는 내장함수(Stored Procedure)를 제공합니다. 내장함수는 빈번하게 사용되는 함수 등을 미리 프로그래밍하여 내장시켜 사용자가 함수명만 호출하여 사용하게 만든 방식입니다. 데이터베이스의 내장함소를 호출하기 위해서는 CallableStatement를 사용합니다. CallableStatement는 PreparedStatement를 상속받은 인터페이스이며 인자값 설정은 PreparedStatement로부터 상속받은 setXxx() 메서드를 사용하고, 결과 값을 반환 받으려면 getXxx() 메서드를 이용합니다.

내장함수의 호출방법은 다음과 같이 표현하며, 변수는 PreparedStatment와 같이 "?"로 표시합니다. 아래의 CallableStatement 예제는 오라클에서 사용할 수 있습니다.

exam/java/chapter10/jdbc/CallableExample.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.sql.*;
4:
5:  public class  CallableExample {
6:
7:     public static void main(String[] args) {
8:        try {
9:           Class.forName("oracle.jdbc.driver.OracleDriver");
10:          System.out.println("드라이버가 로드됐습니다.");
11:       } catch (ClassNotFoundException e) {
12:          System.out.println("드라이버 클래스를 찾을 수 없습니다.");
13:          System.out.println("클래스 이름을 정확히 입력했는지 확인하세요.");
14:          System.out.println("프로젝트 빌드 패스에 드라이버가 설정됐는지 확인하세요.");
15:       }
16:
17:       Connection conn = null;
18:       try {
19:          conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@127.0.0.1:1521:xe", "hr", "hr" );         //Oracle
20:          System.out.println("커넥션이 연결됐습니다.");
21:
22:          String call = "{ ? = call maxplus( ? , ? ) }";
23:
24:          CallableStatement cstmt = conn.prepareCall(call);
25:          cstmt.setInt(2,500);
26:          cstmt.setInt(3,20);       //(최대값 + 1500) / 20 =
27:          cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
28:
29:          cstmt.execute();
30:          System.out.println("result:" + cstmt.getInt(1));
31:       } catch (SQLException e) {
32:          System.out.println(e.getMessage());
33:       } finally {
34:          if(conn != null) try {conn.close();} catch (SQLException e) {}
35:       }
36:    }
37: }
```

앞의 예제를 실행시키려면 먼저 오라클의 SQL*Plus에서 다음처럼 테이블을 만들고 데이터를 입력해야 합니다.

Oracle의 Query문

```
1:  CREATE TABLE shares
    ( ssn     CHAR(15)  NOT NULL,
      symbol  CHAR(8)   NOT NULL,
      quantity   NUMBER(4)    NOT NULL );
2:  INSERT INTO shares(ssn, symbol, quantity) VALUES ('111-120', 'SUNW', 100);
```

```
3:  INSERT INTO shares(ssn, symbol, quantity) VALUES ('111-120', 'DUKE', 200);
4:  COMMIT;
```

앞에서처럼 테이블이 만들어지면 다음 내용을 입력하여 maxplus함수를 선언합니다. 마지막 라인의 / 앞에는 공백이 있으면 안 됩니다.

maxplus 함수 작성

```
1:  create or replace function maxplus( v_value in number, v_factor in number )
2:     return number is max_quantity number(4);
3:     begin
4:        select max(quantity) into max_quantity from shares;
5:        max_quantity := (max_quantity + v_value) / v_factor;
6:     return(max_quantity);
7:  end maxplus;
8:  /
```

실행 결과

```
result:3
```

앞에서 quantity 필드의 최대값은 200이고 자바코드에서 데이터베이스의 maxplus함수 인자 값으로 각각 500과 20이 입력되었으므로 (200 + 500) / 20 = 35가 되는 것입니다.

3.7.3. ResultSetMetaData

ResultSetMetaData는 ResultSet에 대한 정보 즉, 컬럼명(필드명), 개수, 타입 등 메타정보를 알 수 있는 인터페이스로서 ResultSet의 getMetaData() 메서드를 통해 인스턴스를 얻을 수 있습니다. 이러한 정보를 이용하면 JDBC 프로그램을 더욱 융통성 있게 만들 수 있습니다. 다음은 테이블 내의 각 열의 이름을 Header로 출력하는 예제입니다.

exam/java/chapter10/jdbc/ResultSetMetaDataExample.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.sql.*;
4:
5:  public class  ResultSetMetaDataExample {
6:
7:     public static void main(String[] args) throws Exception {
8:
9:  //     Class.forName("oracle.jdbc.driver.OracleDriver");
10:       Class.forName("org.gjt.mm.mysql.Driver"); //MariaDB
11:
```

```
12: //     Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@127.0.0.1:1521:orcl", "scott", "tiger" );
13:        Connection conn = DriverManager.getConnection(
    "jdbc:mysql://127.0.0.1:3306/mydb", "user", "user123" );
14:
15:        Statement stmt = conn.createStatement();
16:
17:        String sql = "SELECT * from customer";
18:        ResultSet rset = stmt.executeQuery(sql);
19:
20:        ResultSetMetaData rm = rset.getMetaData();
21:
22:        int colCount = rm.getColumnCount();
23:
24:        // Header print, index가 1부터 시작함
25:        for(int i=1; i <= colCount; i++) {
26:           System.out.print(rm.getColumnName(i) + "\t\t");
27:        }
28:        System.out.println();
29:
30:        // Data print
31:        while(rset.next()) {
32:           System.out.print(rset.getInt(1) + "\t\t");
33:           System.out.print(rset.getString(2) + "\t\t");
34:           System.out.print(rset.getString(3));
35:           System.out.println();
36:        }
37:        conn.close();
38:     }
39: }
```

실행 결과

id	name	addr
1	홍길동	서울시
2	허현준	광주시
3	허현수	대전시
4	허현정	부산시
100	AAA	Seoul
200	BBB	Pusan
300	CCC	Gwangju

3.7.4. DatabaseMetaData

DatabaseMetaData는 사용 중인 DBMS에 관련된 여러 가지 정보를 얻을 수 있는 인터페

이스입니다. 시스템 카탈로그, 키 관련사항, 테이블 관련사항, 트랜잭션 관련사항 등 다양한 정보를 제공합니다. 하지만 모든 DBMS가 다 지원하는 것은 아니고 일부 함수는 DBMS에 따라 지원하기도 하고, 지원하지 않을 수도 있습니다.
DatabaseMetaData는 Connection 객체의 getMetaData() 함수를 이용하여 인스턴스를 얻을 수 있습니다.

다음 프로그램은 JDBC 드라이버의 이름과 버전을 출력하는 예제입니다.

exam/java/chapter10/jdbc/DatabaseMetaDataExample.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.sql.*;
4:
5:  public class  DatabaseMetaDataExample {
6:
7:     public static void main(String[] args) throws Exception {
8:
9:  //     Class.forName("oracle.jdbc.driver.OracleDriver");
10:       Class.forName("org.gjt.mm.mysql.Driver"); //MariaDB
11:
12: //     Connection conn =
          DriverManager.getConnection( "jdbc:oracle:thin:@127.0.0.1:1521:orcl",
          "scott", "tiger" );
13:       Connection conn =
          DriverManager.getConnection( "jdbc:mysql://127.0.0.1:3306/mydb",
          "scott", "tiger" );
14:
15:       DatabaseMetaData dbmd = conn.getMetaData();
16:
17:       System.out.println( dbmd.getDriverName() );
18:       System.out.println( dbmd.getDriverVersion() );
19:
20:       conn.close();
21:    }
22: }
```

실행 결과

```
MariaDB-AB JDBC Driver
mysql-connector-java-5.1.18 ( Revision: tonci.grgin@oracle.com-... )
```

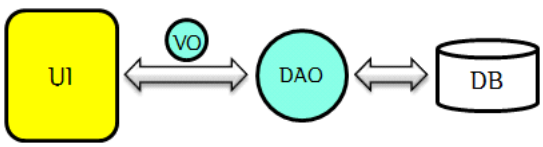
이 책에 언급되지 않은 JDBC API 들이 더 있습니다. 더 자세한 내용을 알고 싶으시면 API 문서를 참고하시기 바랍니다.

3.8. 리소스 관리와 데이터 객체

JDBC 프로그래밍에서 간과하기 쉬운 것이 있습니다. 앞에 언급된 JDBC 샘플 코드들은 모두 예외 처리를 main() 메서드에서 throws 구문을 사용했습니다. 실제로 JDBC 프로그램에서 이와 같은 예외 처리 방법은 권장하지 않습니다. JDBC 프로그래밍에서 가장 중요하게 생각해야 할 것이 바로 Connection 의 관리입니다. main() 메서드에서 throws 구문을 사용해 예외를 처리하게 된다면 도중에 예외가 발생하였을 때 리소스를 반납하는 코드(conn.close())가 실행이 되지 않을 수 있습니다. 그러므로 다음 코드에서처럼 JDBC 프로그래밍시 적절한 예외처리와 finally 블록에서 리소스의 반납이 이루어져야 합니다.

```
    } finally {
        if(con!=null) {
           try {
              con.close();
           } catch (SQLException e1) {
              //nothing
           }
        }
    }
```

리소스 관리뿐만 아니라 소스코드의 재사용성에 대해서도 생각해 봐야 할 것입니다. 아래의 그림을 보세요.



위 그림에서 UI(User Interface)는 사용자 인터페이스를 담당하는 객체입니다. main() 메서드가 있는 클래스라고 생각해도 좋습니다. 그리고 DAO(Data Access Object)는 데이터베이스와 직접 연결되어 쿼리를 수행하는 객체입니다. DAO는 커넥션을 얻어오고, 데이터베이스에 쿼리를 전송하는 코드를 작성합니다. VO(Value Object)는 UI 클래스에서 DAO 클래스의 메서드를 호출하고, 리턴 받을 때 사용하는 객체입니다. VO 클래스는 데이터베이스의 테이블을 추상화한 클래스입니다.

다음 코드들은 앞에서 예들 들었던 SimpleJDBCExample 클래스를 VO, DAO, UI 클래스로 나누어 작성했습니다. 기존의 코드에 비해 라인 수가 많아졌습니다. 그러나 앞에서 설명한 여러 가지 요구사항을 만족합니다.

다음 클래스는 UI와 DAO 사이에 값을 주고받기 위해서 데이터베이스 테이블을 추상화할 클래스입니다.

exam/java/chapter10/jdbc/CustomerVO.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  public class CustomerVO {
4:     int id;
5:     String name;
6:     String addr;
7:
8:     public CustomerVO() {
9:        super();
10:    }
11:    public CustomerVO(int id, String name, String addr) {
12:       super();
13:       this.id = id;
14:       this.name = name;
15:       this.addr = addr;
16:    }
17:
18:    public int getId() {
19:       return id;
20:    }
21:
22: //id는 primary key 속성이므로 변경이 안되도록 합니다.
23: // public void setId(int id) {
24: //    this.id = id;
25: // }
26:
27:    public String getName() {
28:       return name;
29:    }
30:
31:    public void setName(String name) {
32:       this.name = name;
33:    }
34:
35:    public String getAddr() {
36:       return addr;
37:    }
38:
39:    public void setAddr(String addr) {
40:       this.addr = addr;
41:    }
42:
43: }
```

다음 클래스는 데이터베이스와 연결하고, 쿼리를 실행하는 DAO클래스입니다.

exam/java/chapter10/jdbc/CustomerDAO.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.sql.Connection;
4:  import java.sql.DriverManager;
5:  import java.sql.PreparedStatement;
6:  import java.sql.ResultSet;
7:  import java.sql.SQLException;
8:  import java.util.ArrayList;
9:
10: public class CustomerDAO {
11:    public Connection getConnection() {
12:       Connection con=null;
13:       try {
14: //        con = DriverManager.getConnection(
    "jdbc:oracle:thin:@127.0.0.1:1521:orcl", "scott", "tiger" ); //Oracle
15:          con = DriverManager.getConnection(
    "jdbc:mysql://127.0.0.1:3306/mydb", "user", "user123" ); //MariaDB
16:          System.out.println(con);
17:       } catch (SQLException e1) {
18:          e1.printStackTrace();
19:       }
20:       return con;
21:    }
22:
23:    public ArrayList<CustomerVO> getAllCustomers() {
24:       Connection con=null;
25:       ArrayList<CustomerVO> listData = new ArrayList<>();
26:       try {
27:          con = getConnection();
28:
29:          String sql = "SELECT id, name, addr FROM customer";
30:          PreparedStatement pstmt = con.prepareStatement(sql);
31:
32:          ResultSet rs = pstmt.executeQuery();
33:          while(rs.next()) {
34:             CustomerVO vo = new CustomerVO(rs.getInt("id"),
    rs.getString("name"), rs.getString("addr"));
35:             listData.add(vo);
36:          }
37:       } catch (SQLException e1) {
38:          e1.printStackTrace();
39:       } finally {
40:          if(con!=null) try {con.close();} catch (SQLException e1) {  }
41:       }
42:       return listData;
43:    }
44: }
```

다음 코드는 UI클래스입니다. static initializer를 이용해 드라이버를 로딩했습니다.

exam/java/chapter10/jdbc/SimpleMVCExample.java

```
1:  package exam.java.chapter10.jdbc;
2:
3:  import java.util.ArrayList;
4:
5:  public class SimpleMVCExample {
6:
7:     static {
8:        try {
9:  //        Class.forName("oracle.jdbc.driver.OracleDriver"); //Oracle
10:          Class.forName("org.gjt.mm.mysql.Driver"); //MariaDB
11:          System.out.println("드라이버 로딩 성공");
12:       } catch (ClassNotFoundException e) {
13:          System.out.println("드라이버 로딩 실패");
14:       }
15:    }
16:
17:    public static void main(String[] args) throws Exception{
18:
19:       CustomerDAO dao = new CustomerDAO();
20:       ArrayList<CustomerVO> listData = dao.getAllCustomers();
21:       System.out.println("id\tname\taddr");
22:       for(CustomerVO vo : listData) {
23:          System.out.print(vo.getId());
24:          System.out.print("\t" + vo.getName());
25:          System.out.println("\t" + vo.getAddr());
26:       }
27:
28:    }//end main
29: }//end class
```

3.9. 요점 정리

1. JDBC Programming

드라이버 로딩

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

커넥션 생성

```
DriverManager.getConnection(url, id, pw);
```

쿼리 전송문 생성

```
String sql = "select * from employees";
PreparedStatement pstmt = con.prepareStatement(sql);
```

쿼리 전송

```
select 문일 경우 : ResultSet rs = pstmt.executeQuery();
DML일 경우 : int count = pstmt.executeUpdate();
```

결과값 사용

```
rs.getXxx("컬럼이름");
```

커넥션 닫기

```
finally블록 안에 if(con!=null) try {con.close(); } catch (SQLException e) { }
```

아래 그림과 관련된 예제를 잘 알아두세요.