MySQL, JDBC

524730-1
2024년 봄학기
5/8/2024
박경신

데이터베이스의 개념

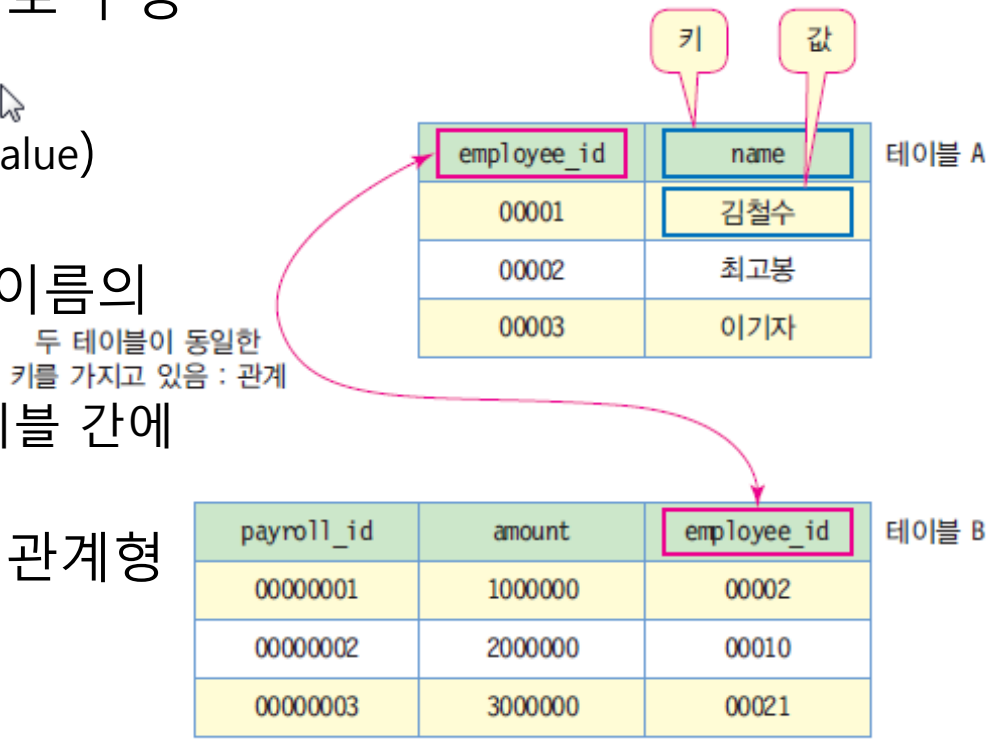
- 데이터베이스(Database)
  - 여러 응용 시스템들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 집합
  - 데이터의 저장, 검색, 갱신을 효율적으로 수행할 수 있도록 데이터를 고도로 조직화하여 저장
- DBMS
  - 데이터베이스 관리 시스템(DataBase Management System)
    - 오라클(Oracle), 마이크로소프트의 SQL Server, MySQL, IBM의 DB2 등

데이터베이스의 개념

- 데이터베이스 종류
  - 관계형 데이터베이스
    - 키(key)와 값(value)들의 관계를 테이블로 표현한 데이터베이스 모델
    - 키는 테이블의 열(column)이 되며 테이블의 행(row)은 하나의 레코드(record)를 표현
    - 현재 사용되는 대부분의 데이터베이스는 관계형 데이터베이스
  - 객체 지향 데이터베이스
    - 객체 지향 프로그래밍에 쓰이는 것으로, 정보를 객체의 형태로 표현하는 데이터베이스 모델
    - 오브젝트 데이터베이스(object database)라고도 부름

관계형 데이터베이스 구조

- 관계형 데이터 베이스
  - 데이터들이 다수의 테이블로 구성
  - 각 행은 하나의 레코드
  - 각 테이블은 키(key)와 값(value)들의 관계로 표현
  - 여러 테이블 간에 공통된 이름의 열 포함
    - 이런 경우 서로 다른 테이블 간에 관계(relation)가 성립
  - 대부분의 데이터베이스는 관계형 데이터베이스
    - JDBC API



객체지향 데이터베이스

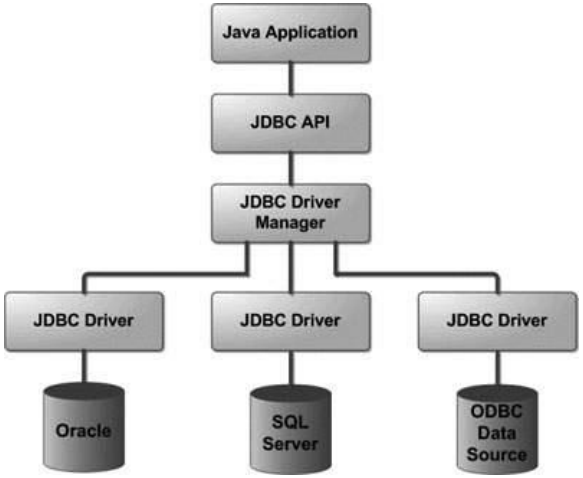
- 객체지향 데이터 베이스
  - 객체 지향 프로그래밍에 사용
  - 정보를 객체의 형태로 표현
  - 오브젝트 데이터베이스(object database)라고도 부름
  - 객체 모델이 그대로 데이터베이스에도 적용되므로
    응용프로그램의 객체 모델과 데이터베이스의 모델이 부합됨

SQL과 JDBC

- SQL(Structured Query Language)
  - 관계형 데이터베이스 관리 시스템에서 사용
  - 데이터베이스 스키마 생성, 자료의 검색, 관리, 수정, 그리고 데이터베이스 객체 접근 관리를 위해 고안된 언어
  - 데이터베이스로부터 정보를 추출하거나 갱신하기 위한 표준 대화식 프로그래밍 언어
    - 다수의 데이터베이스 관련 프로그램들이 SQL을 표준으로 채택
- JDBC(Java DataBase Connectivity)
  - 관계형 데이터베이스에 저장된 데이터를 접근 및 조작할 수 있게 하는 API
  - 다양한 DBMS에 대해 일관된 API로 데이터베이스 연결, 검색, 수정, 관리 등을 할 수 있게 함

JDBC 구조

- JDBC 드라이버 매니저
  - 자바 API에서 지원하며 DBMS를 접근할 수 있는 JDBC 드라이버 로드
- JDBC 드라이버
  - DBMS마다 고유한 JDBC 드라이버 제공, JDBC 드라이버와 DBMS는 전용 프로토콜로 데이터베이스 처리
- DBMS
  - 데이터베이스 관리 시스템. 데이터베이스 생성·삭제, 데이터 생성·검색·삭제 등 전담 소프트웨어 시스템



MySQL

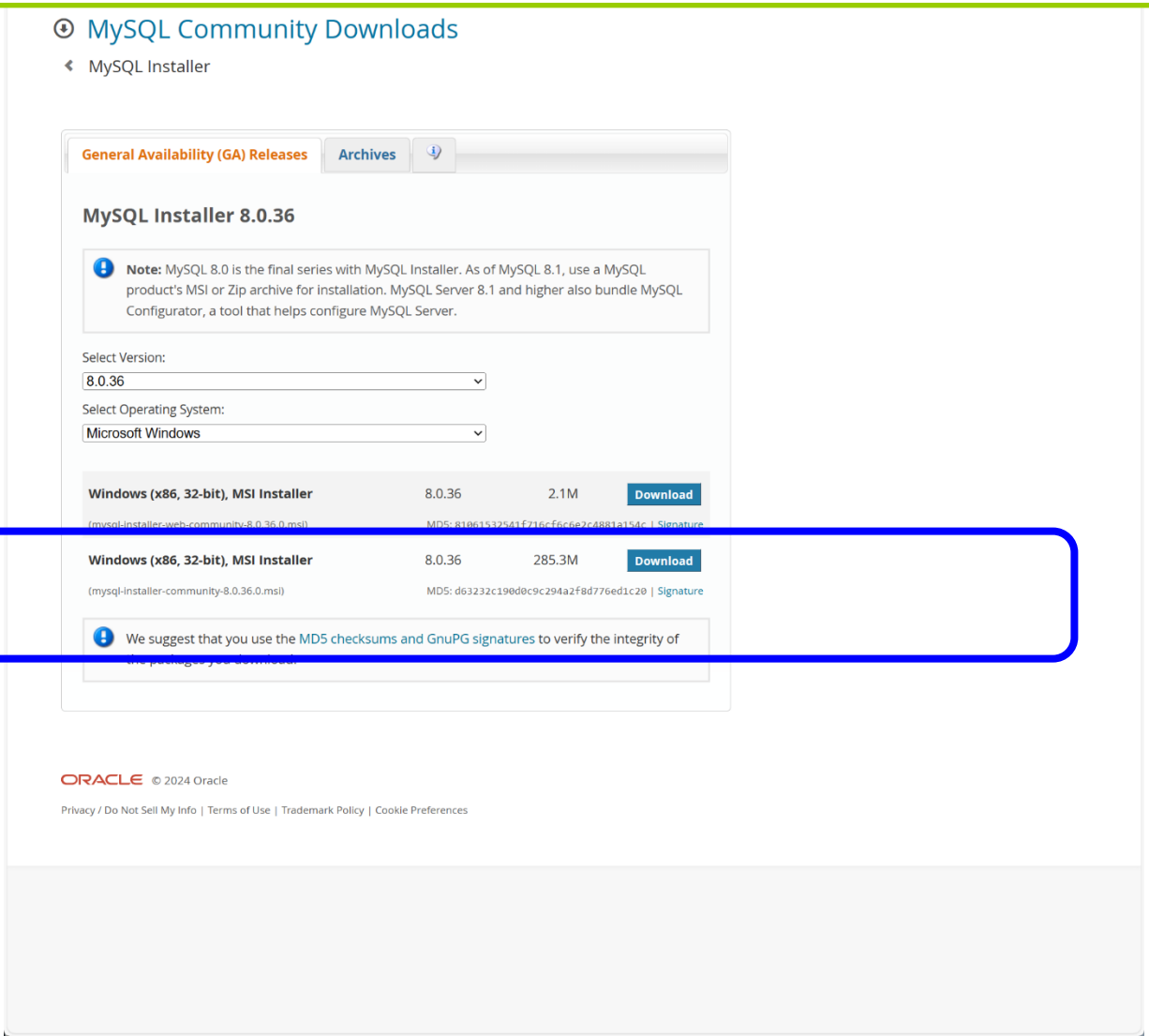
- MySQL
  - 전 세계적으로 가장 널리 사용되고 있는 오픈 소스 관계형 데이터베이스 관리 시스템
  - 관계형 데이터베이스 관리 시스템의 표준화된 사용자 및 프로그래밍 인터페이스인 질의언어 SQL(Structured Query Language)을 사용
  - 매우 빠르고 유연하고 사용하기 쉽기 때문에 많은 기업에서 다양한 웹 기반 애플리케이션을 개발하는 데 사용

MySQL 서버 설치

https://www.mysql.com/downloads/

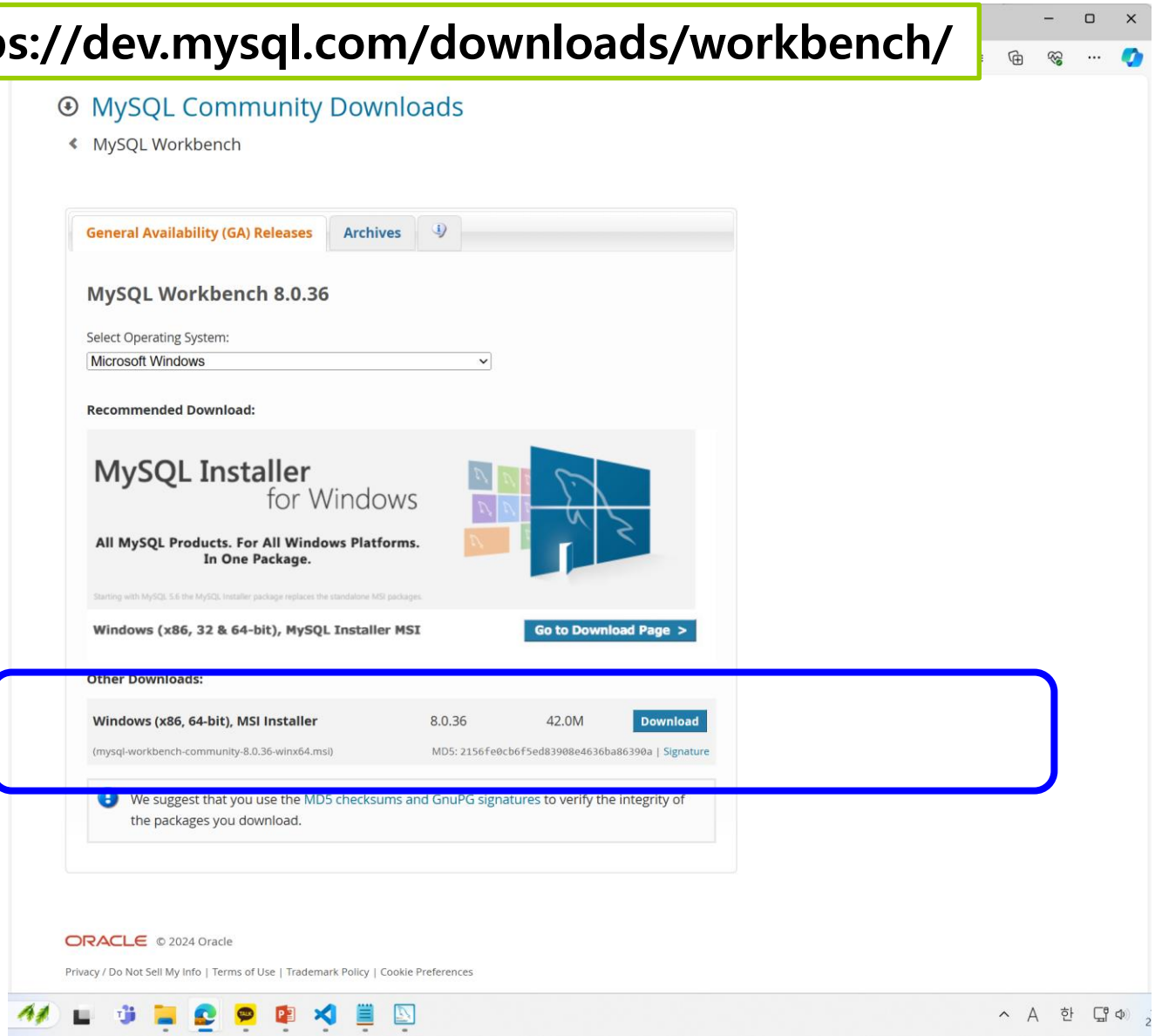
MySQL 서버 설치파일 다운로드

https://dev.mysql.com/downloads/windows/installer/8.0.html

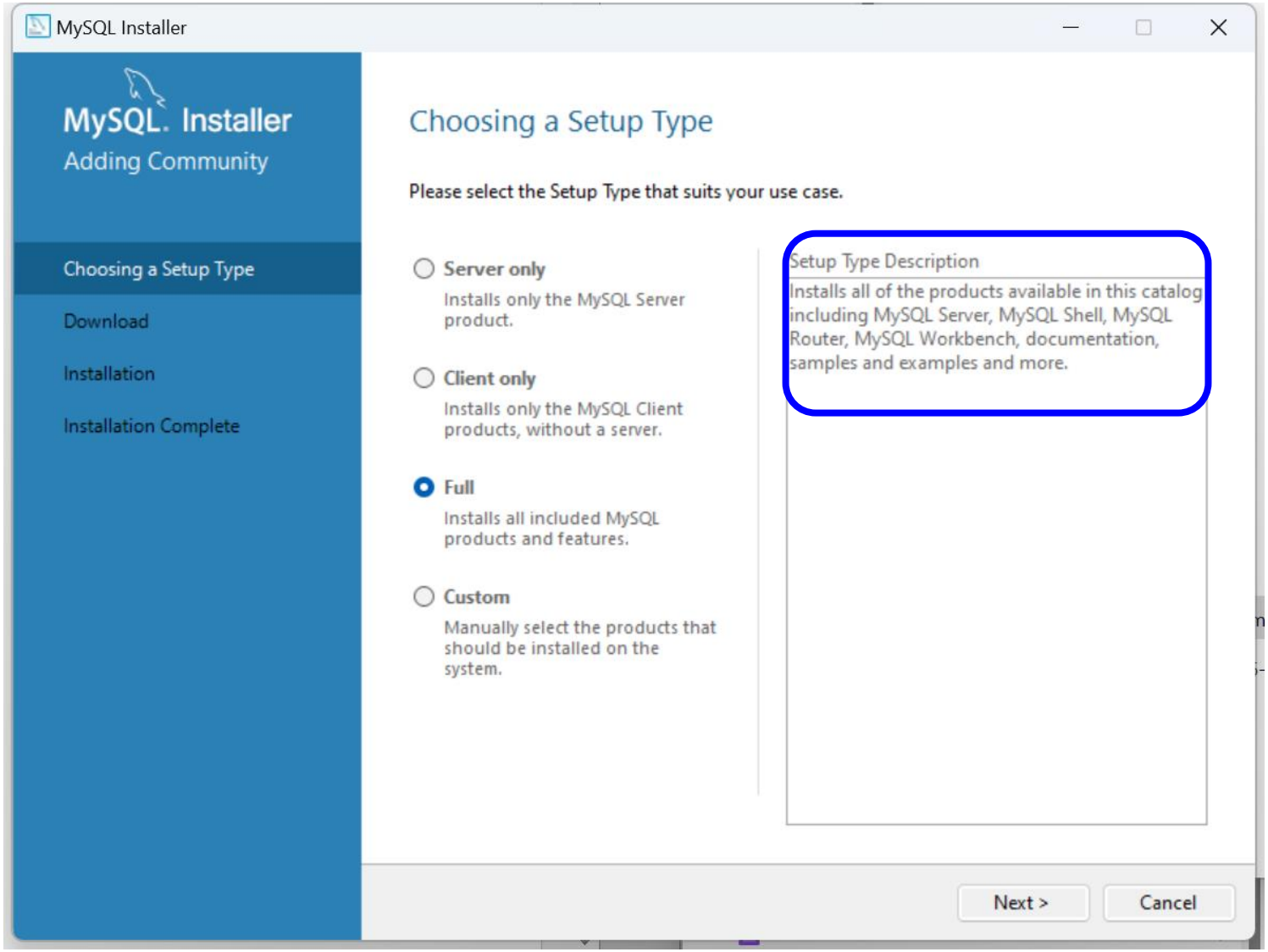


MySQL 서버 설치파일 다운로드

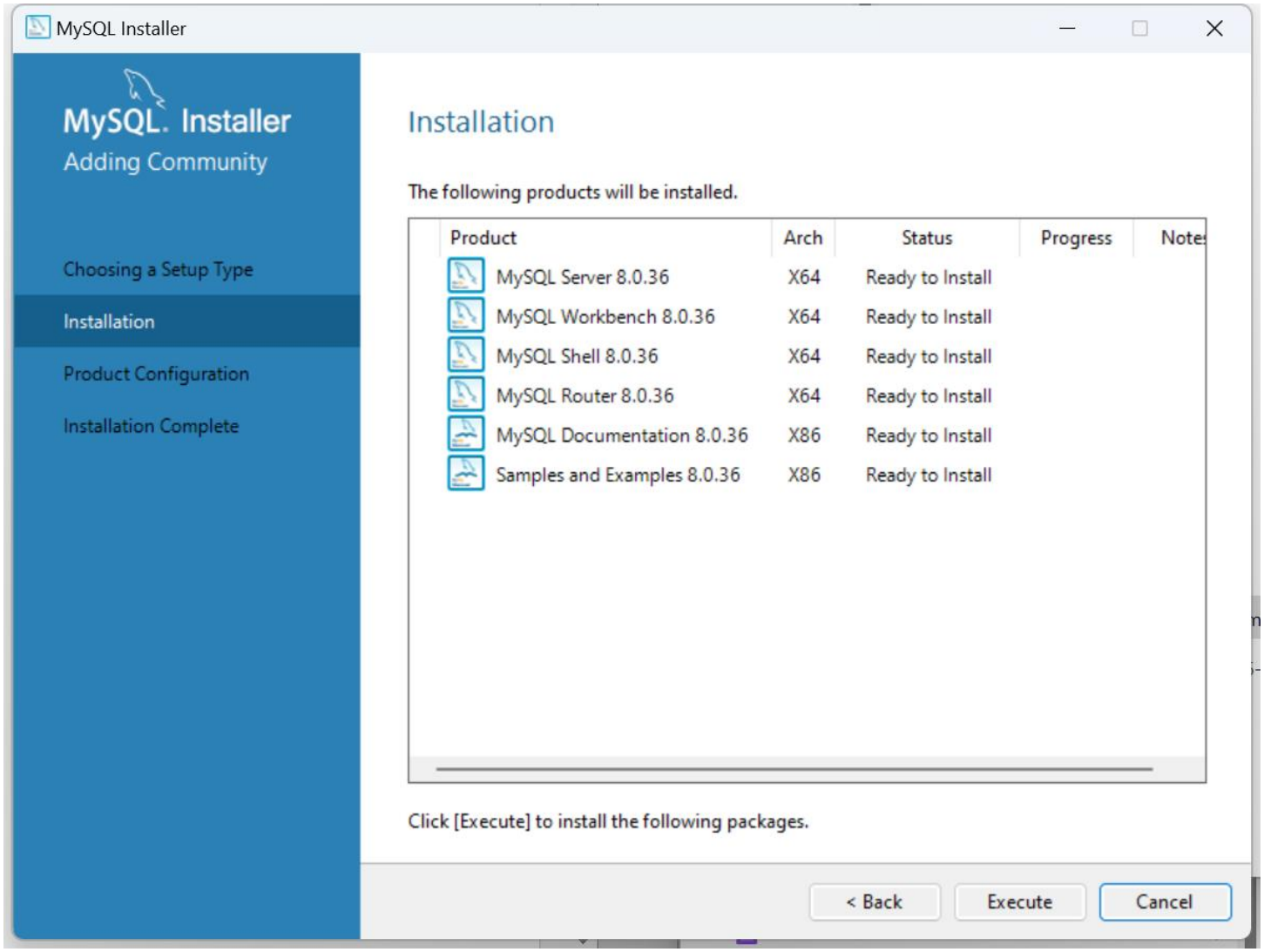
https://dev.mysql.com/downloads/workbench/



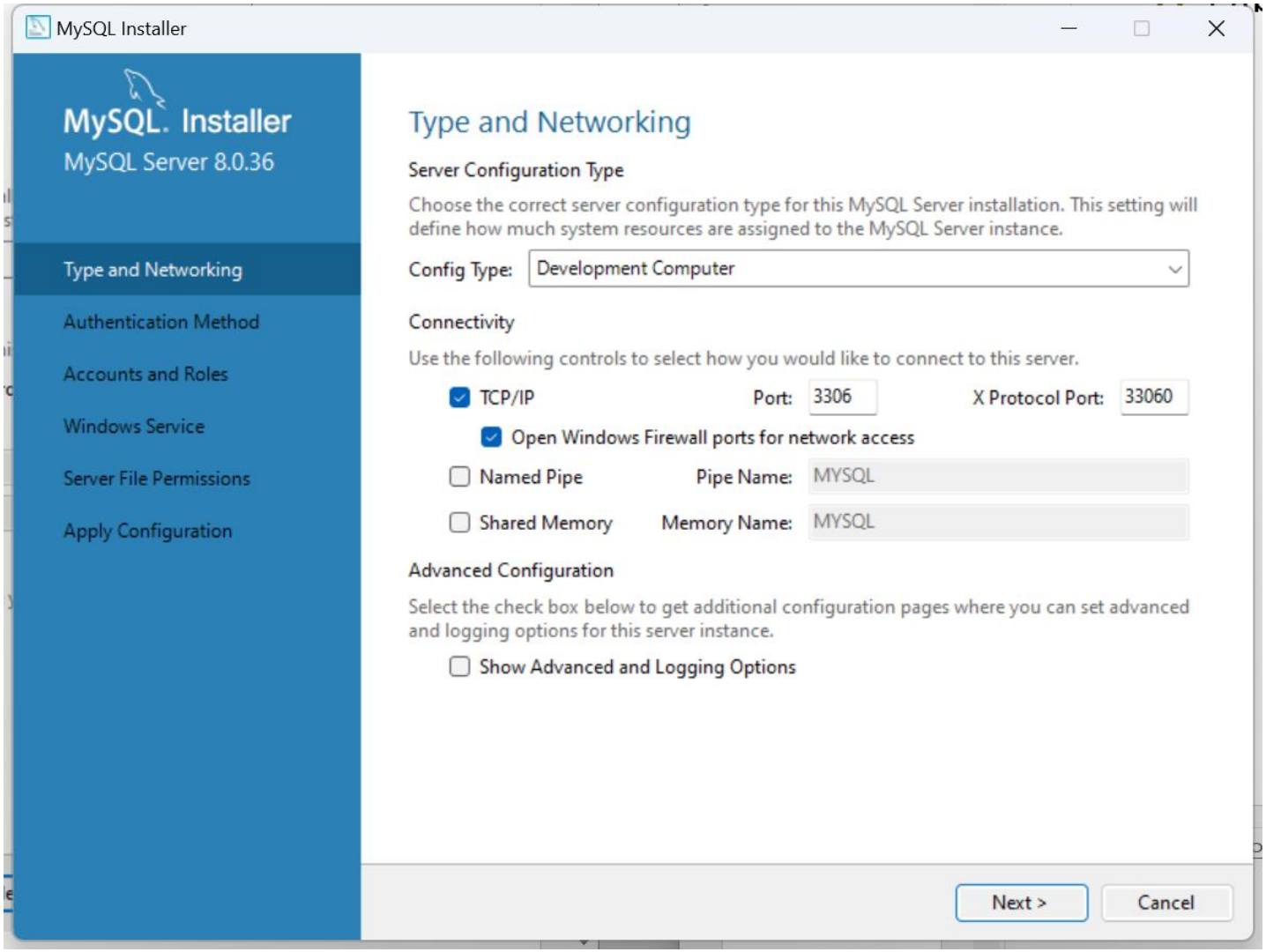
MySQL 서버 Default 타입으로 설치



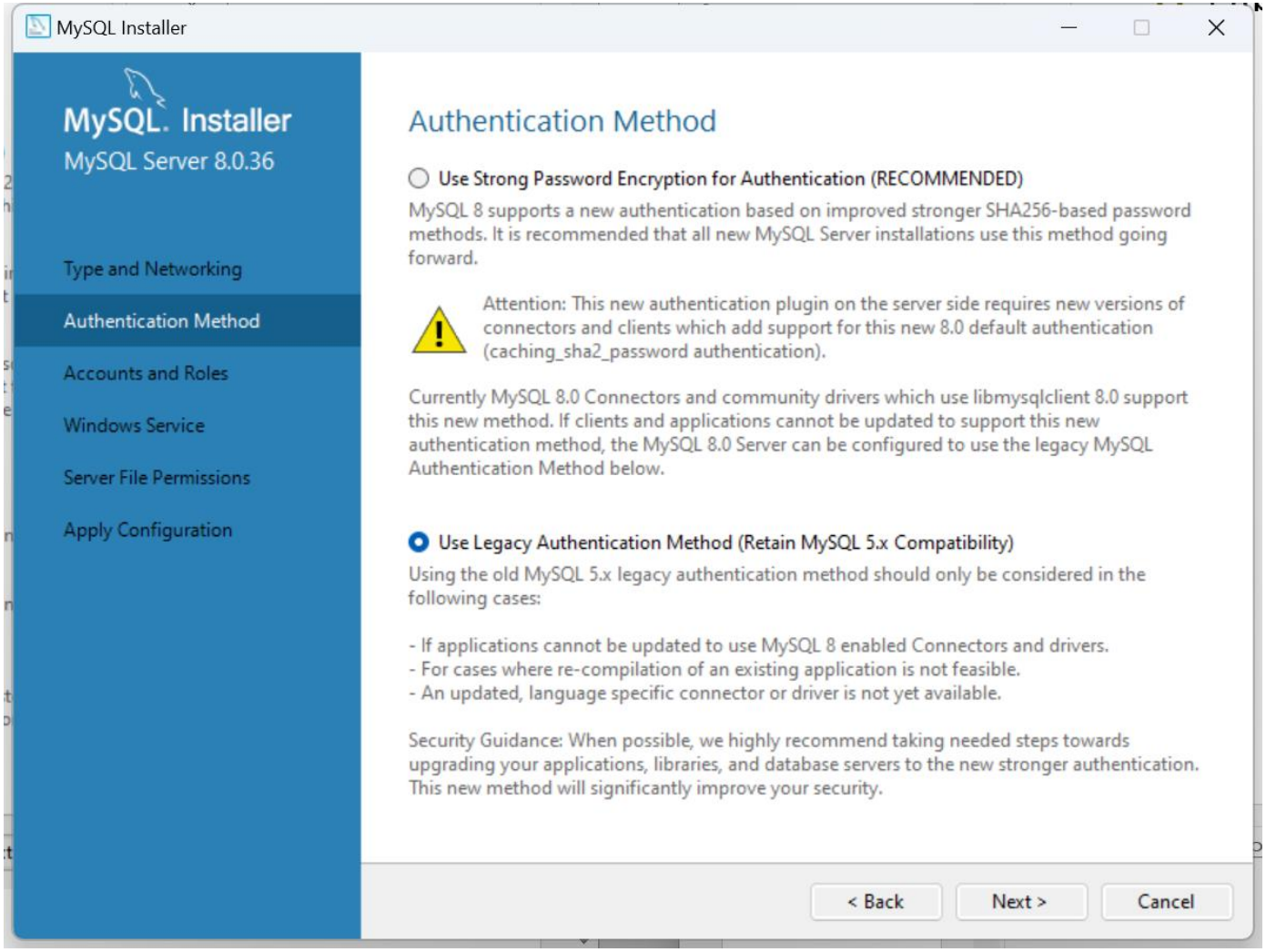
MySQL 서버 설치 및 설정



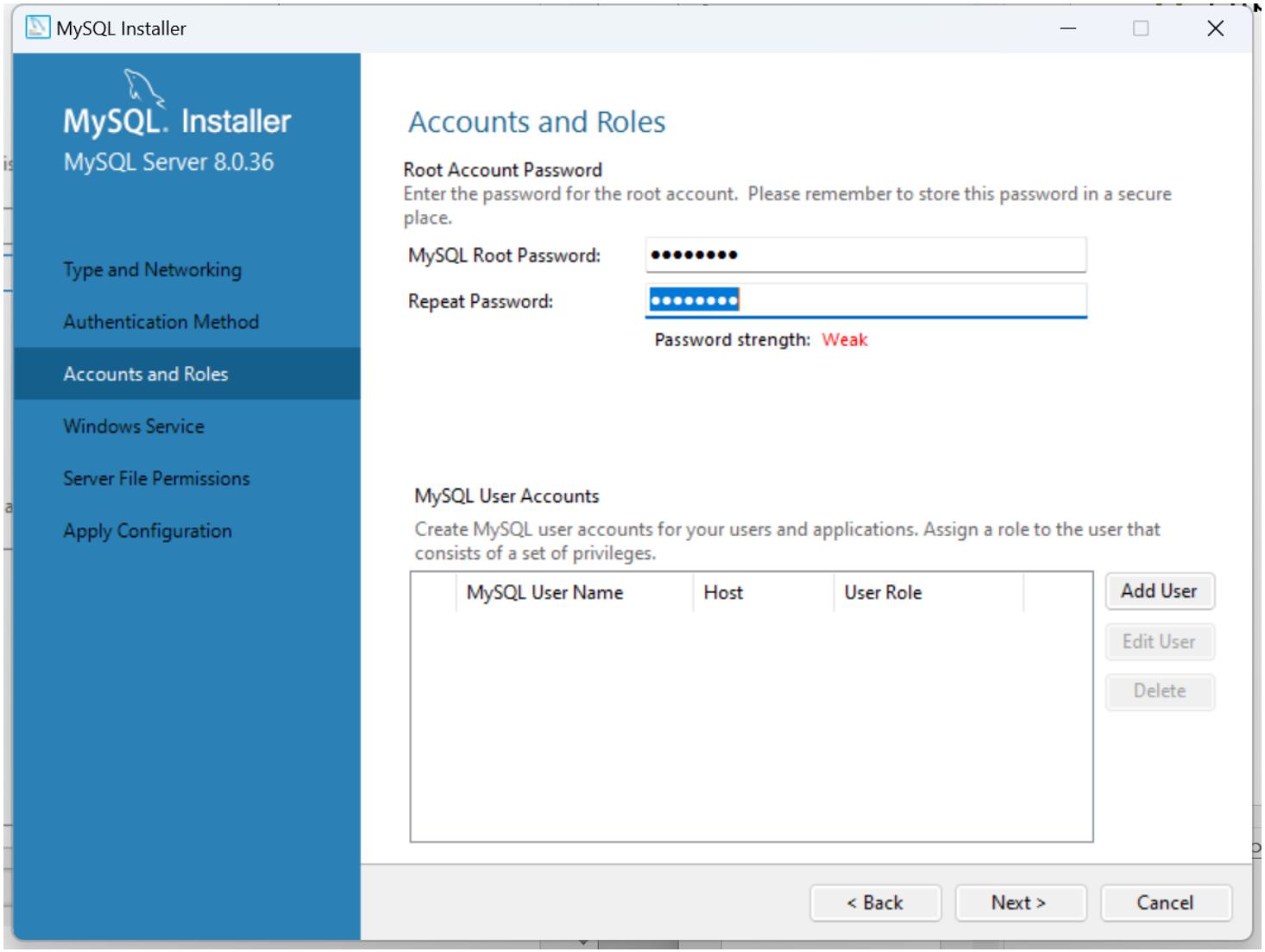
MySQL 서버 설치 및 설정



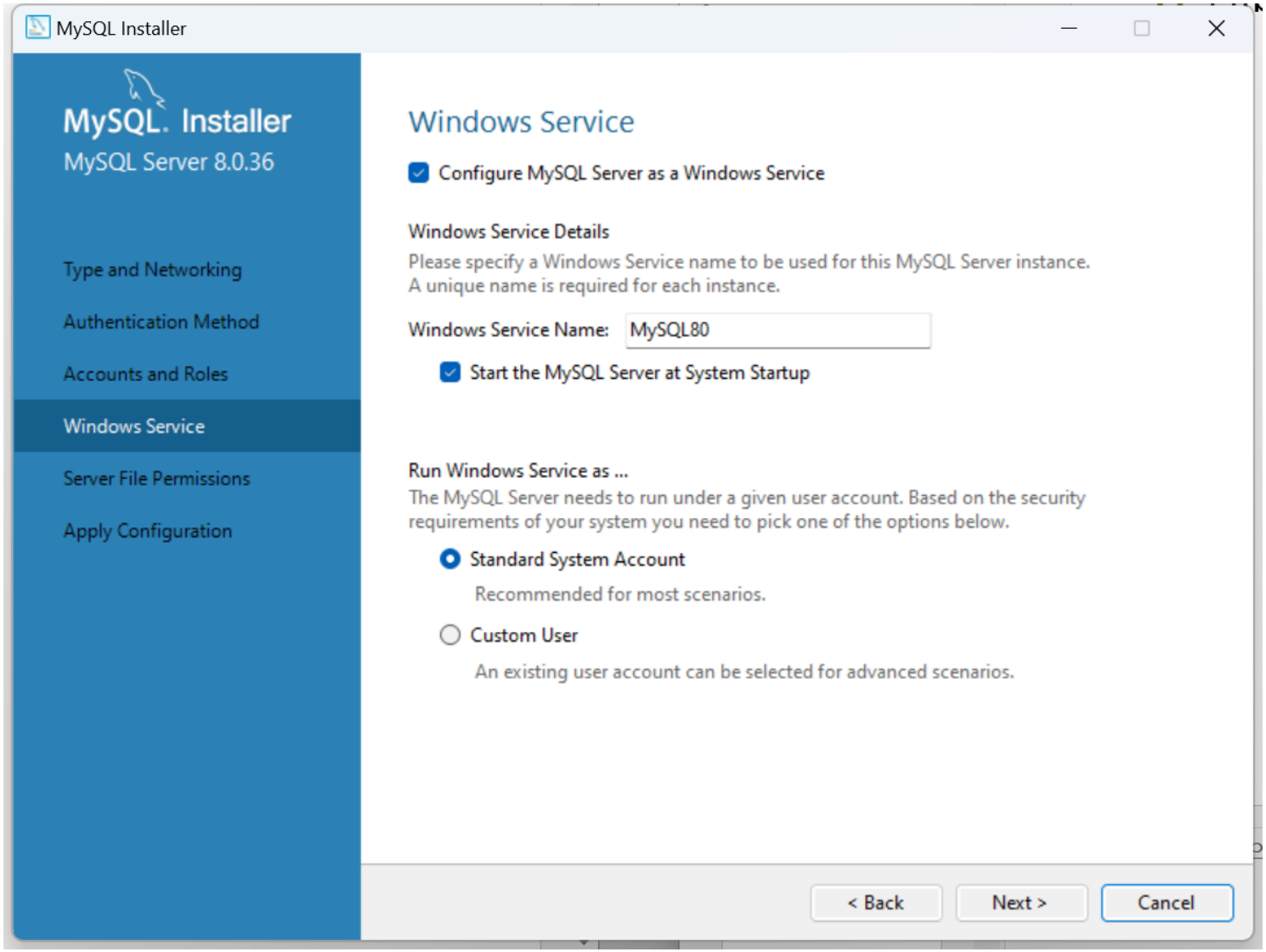
MySQL 서버 설치 및 설정



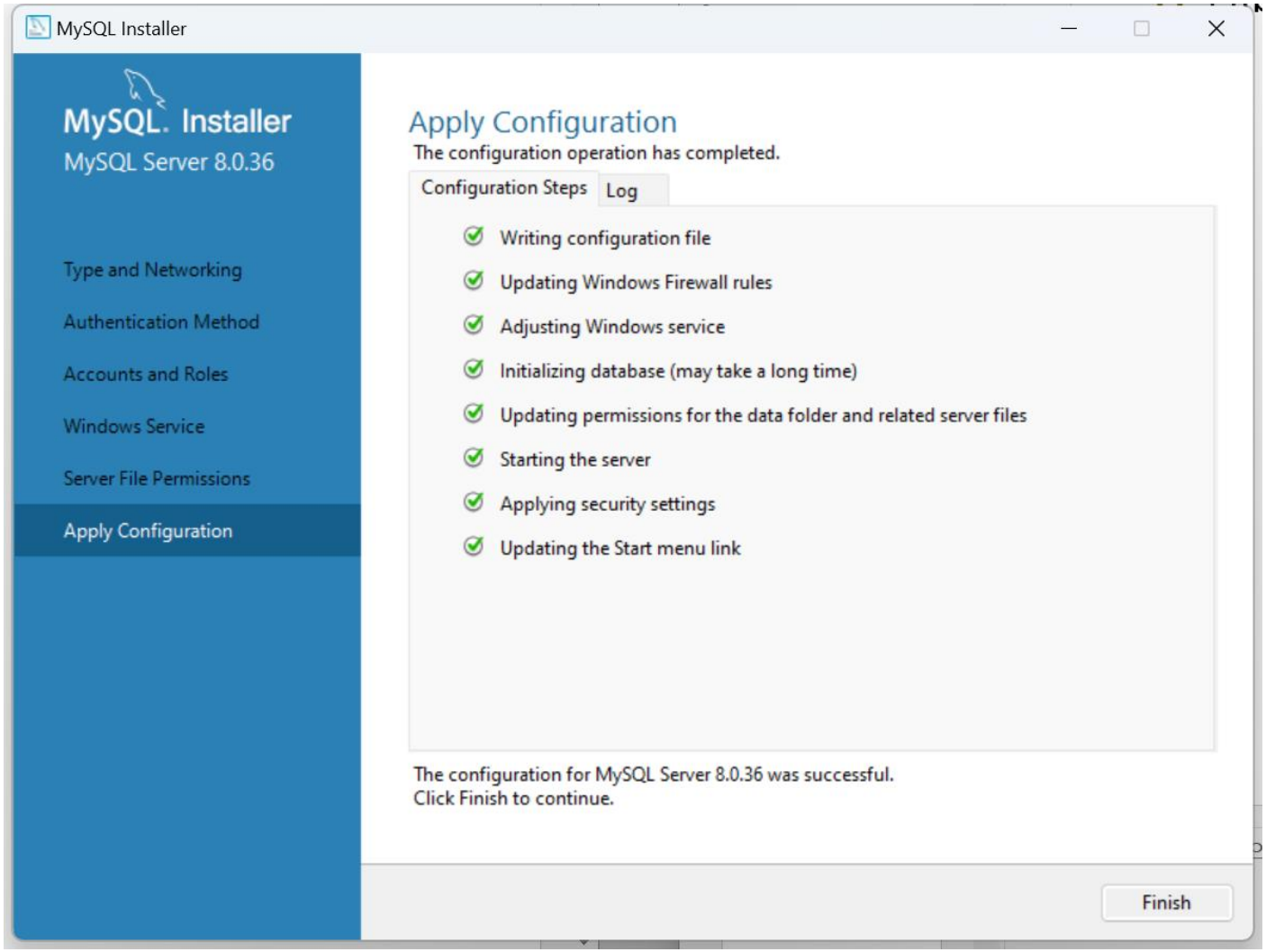
MySQL 서버 설치 및 설정



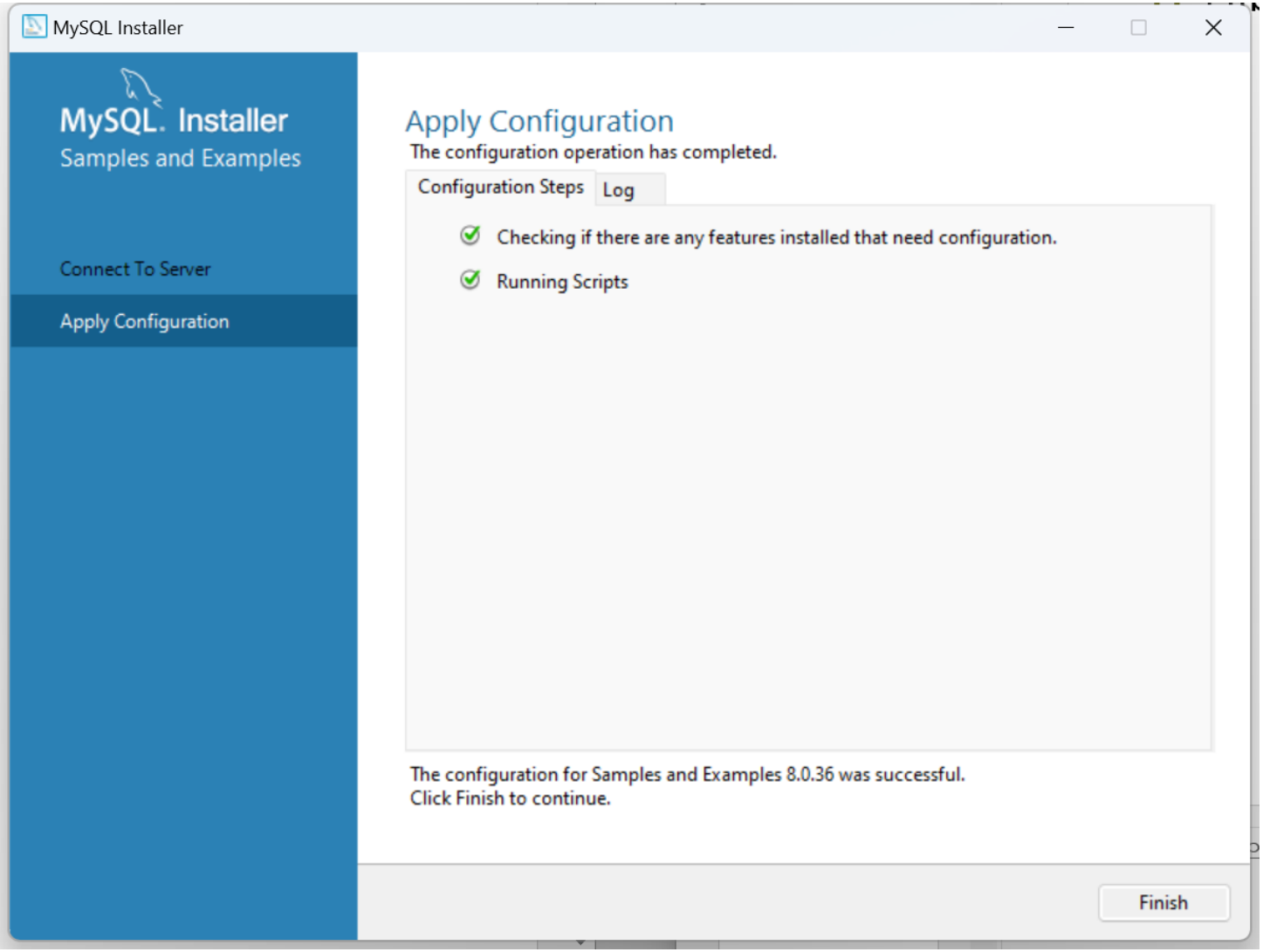
MySQL 서버 설치 및 설정



MySQL 서버 설치 및 설정

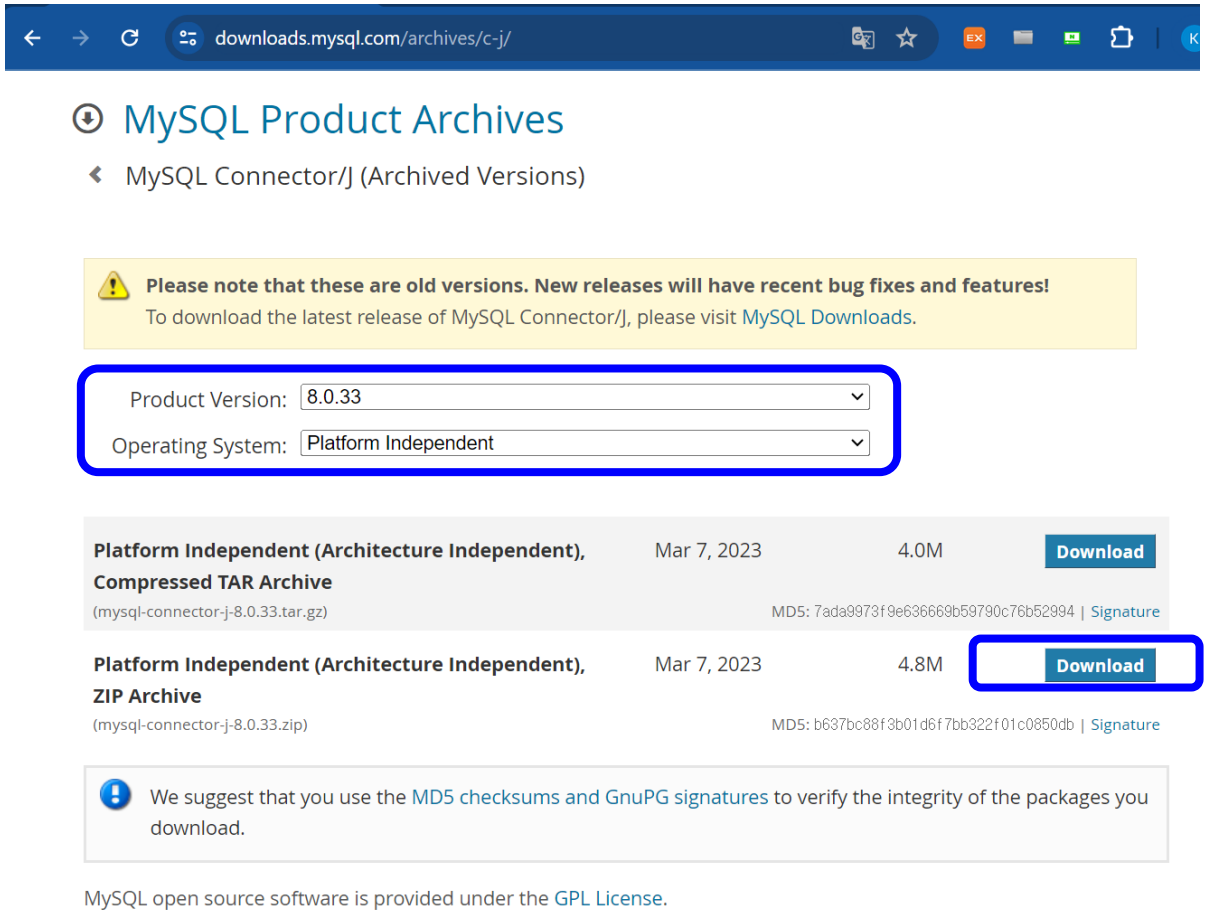


MySQL 서버 설치 및 설정



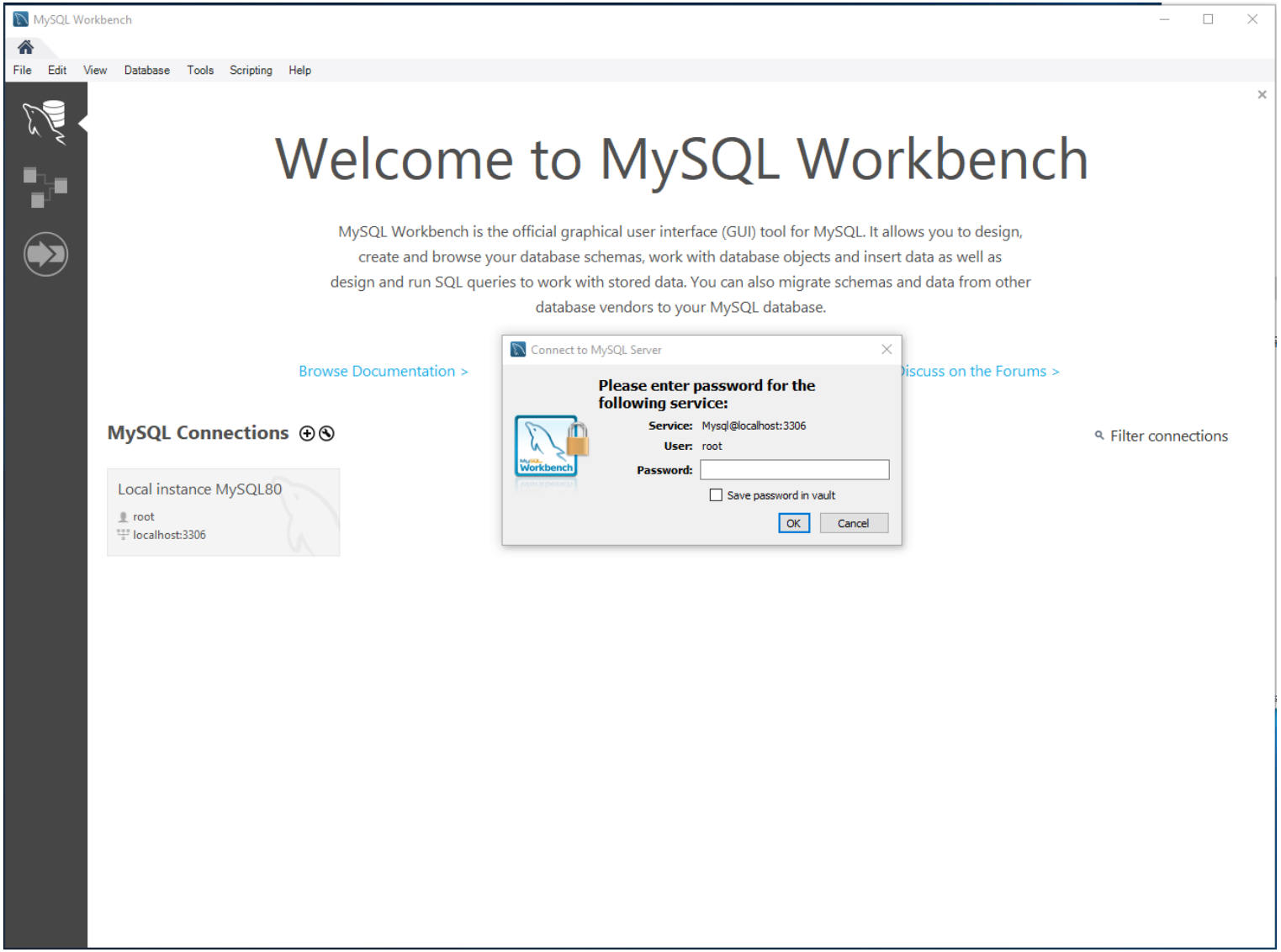
JDBC 드라이버 설치

- Connector/J 8.0.33 JDBC 드라이버 설치



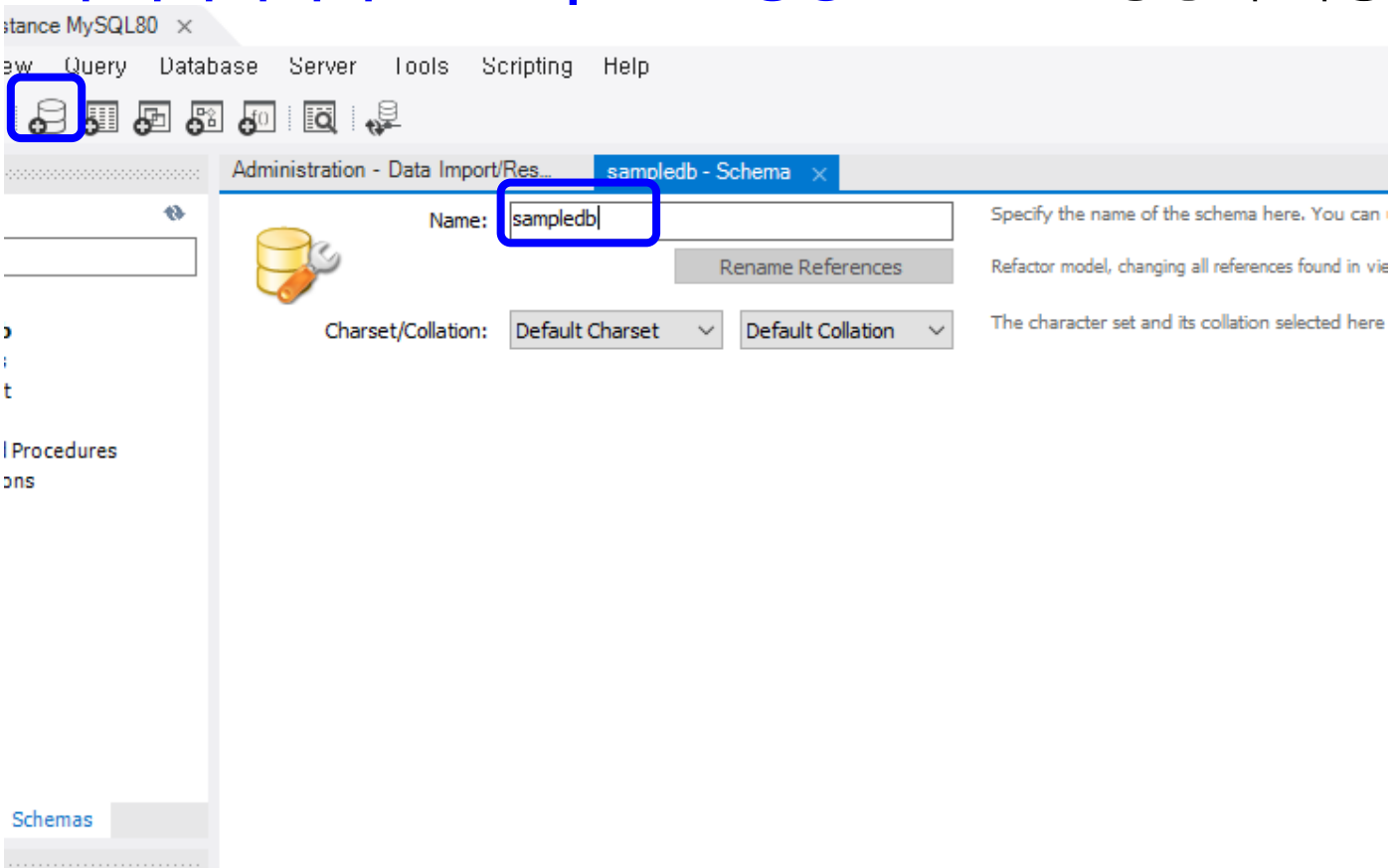
https://dev.mysql.com/downloads/connector/j/

MySQL Workbench 실행



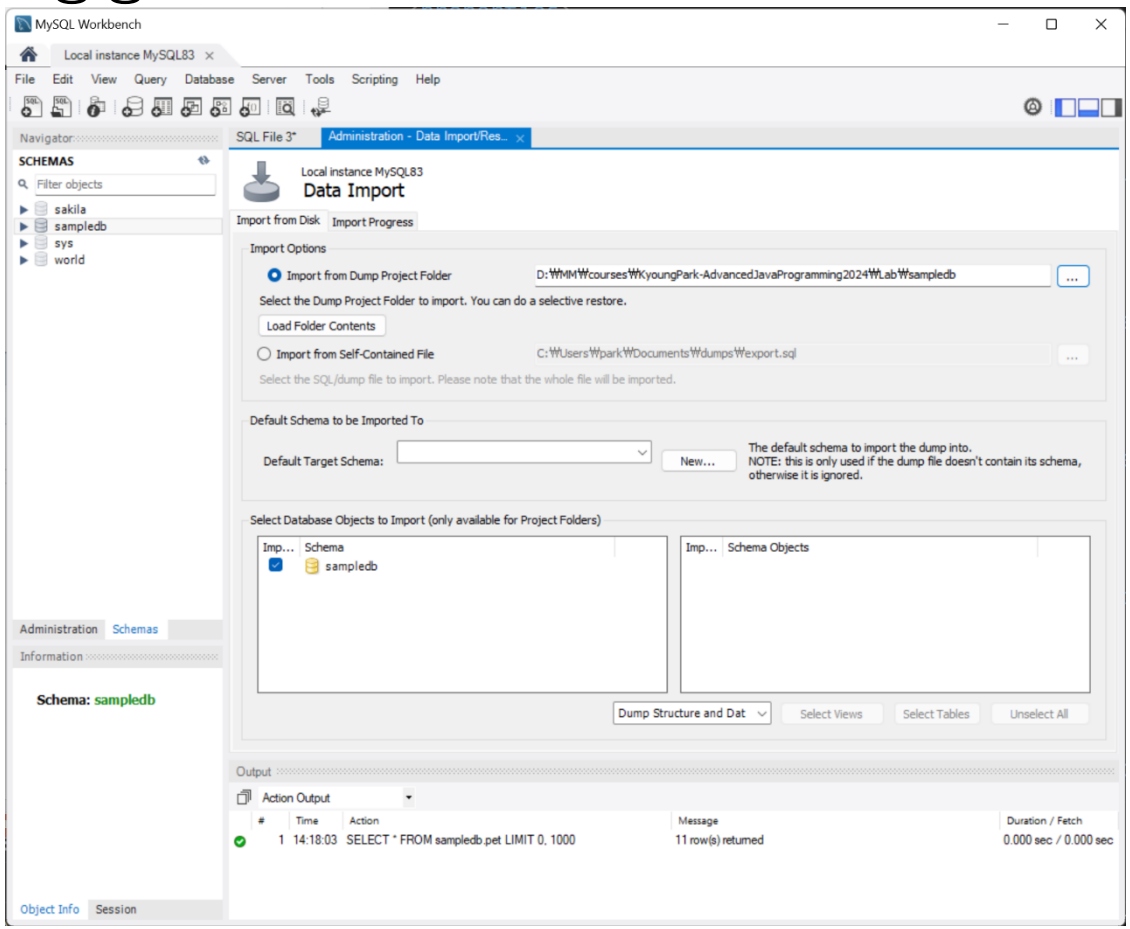
MySQL Workbench 실행

- MySQL Workbench를 사용하여 새 DB 생성
  - MySQL Command Line Client 또는 **MySQL Workbench**를 실행한 후 root 계정의 비밀번호를 입력하여 MySQL에 접속
  - **새 데이터베이스 'sampledb' 생성** - CREATE 명령어 사용



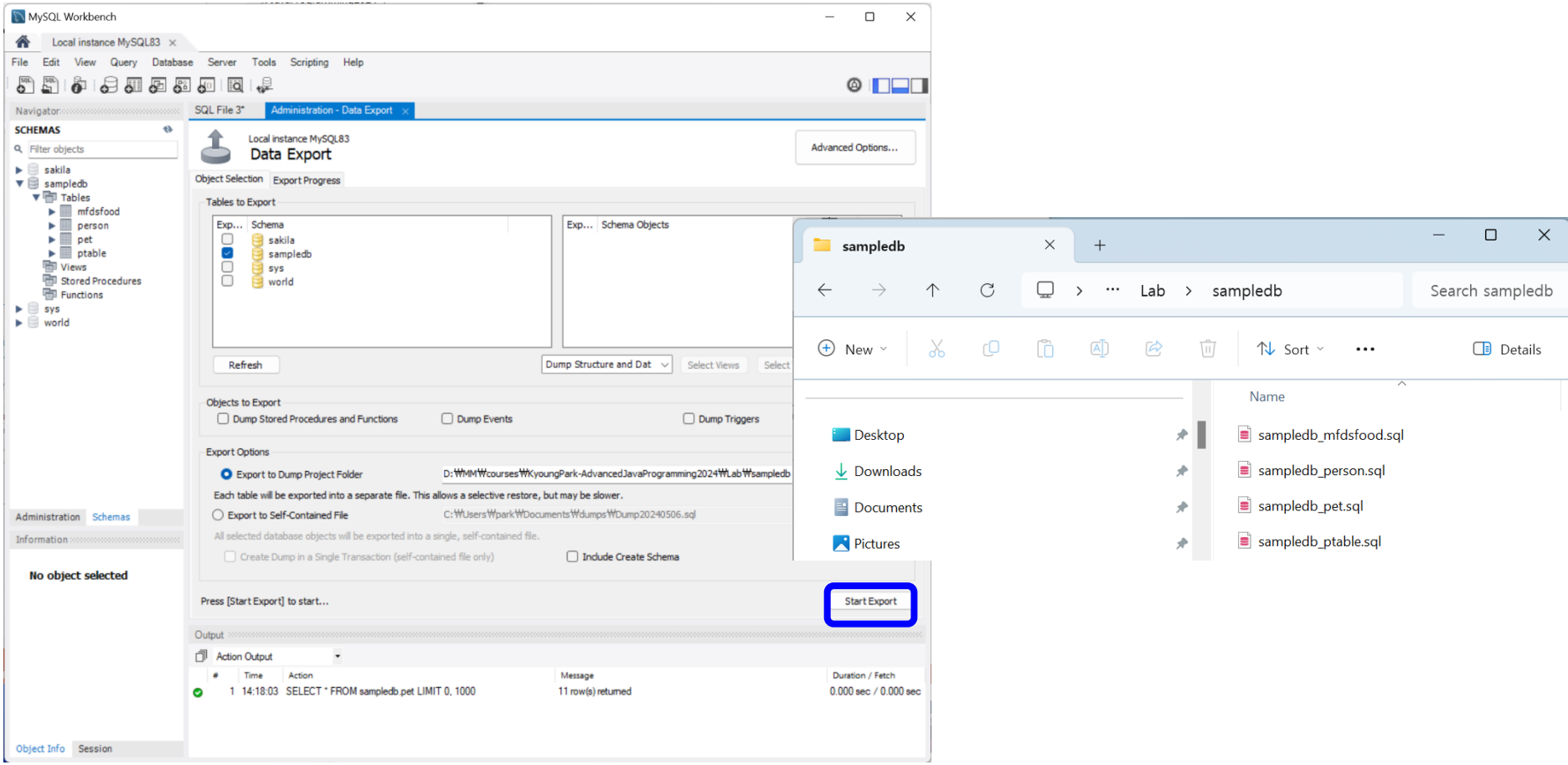
MySQL Workbench 실행

- MySQL 데이터베이스 import
  - MySQL Workbench Server->Data Import ‘sampledb_pet.sql’ 등등



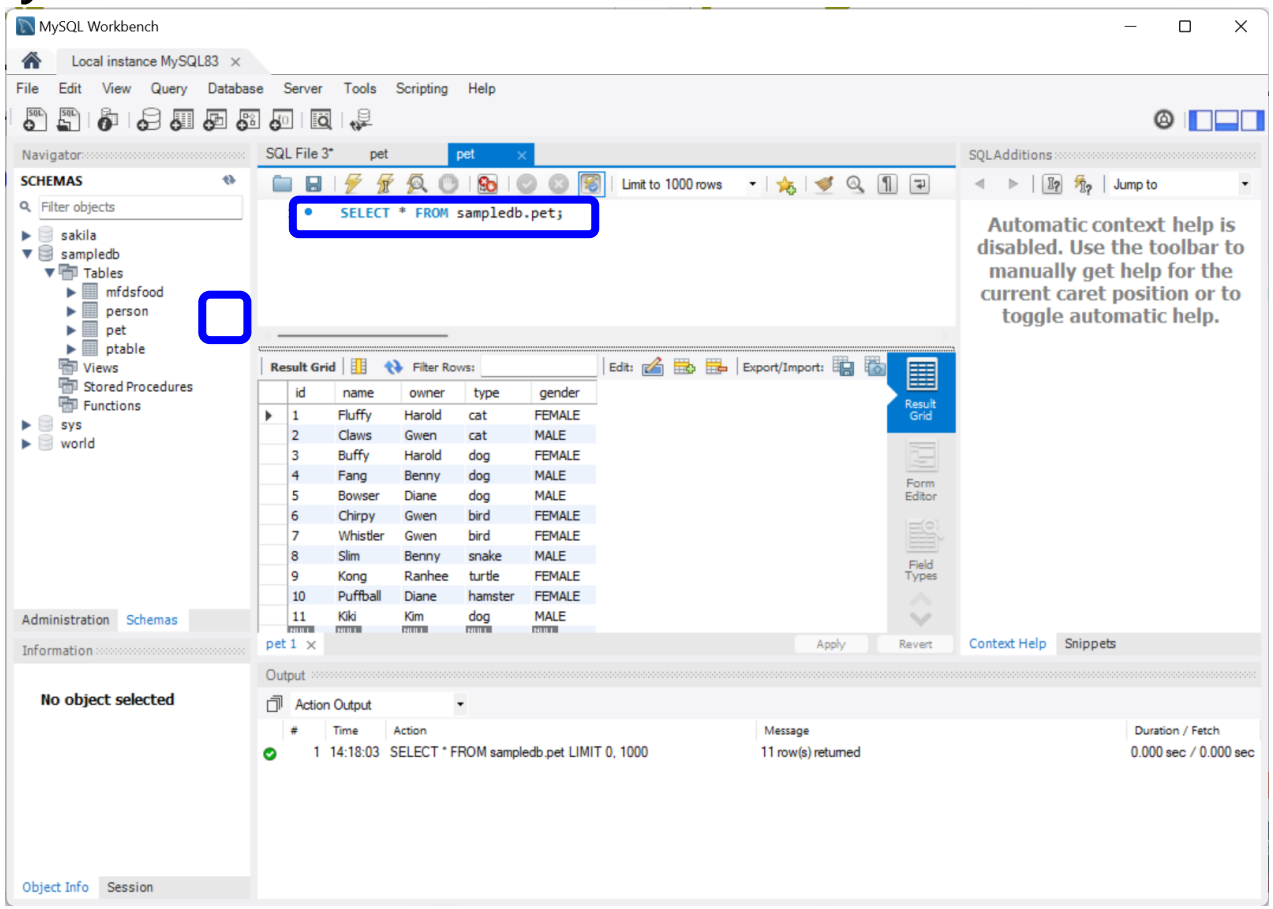
MySQL Workbench 실행

- MySQL 데이터베이스 export
  - MySQL Workbench Server->Data Export



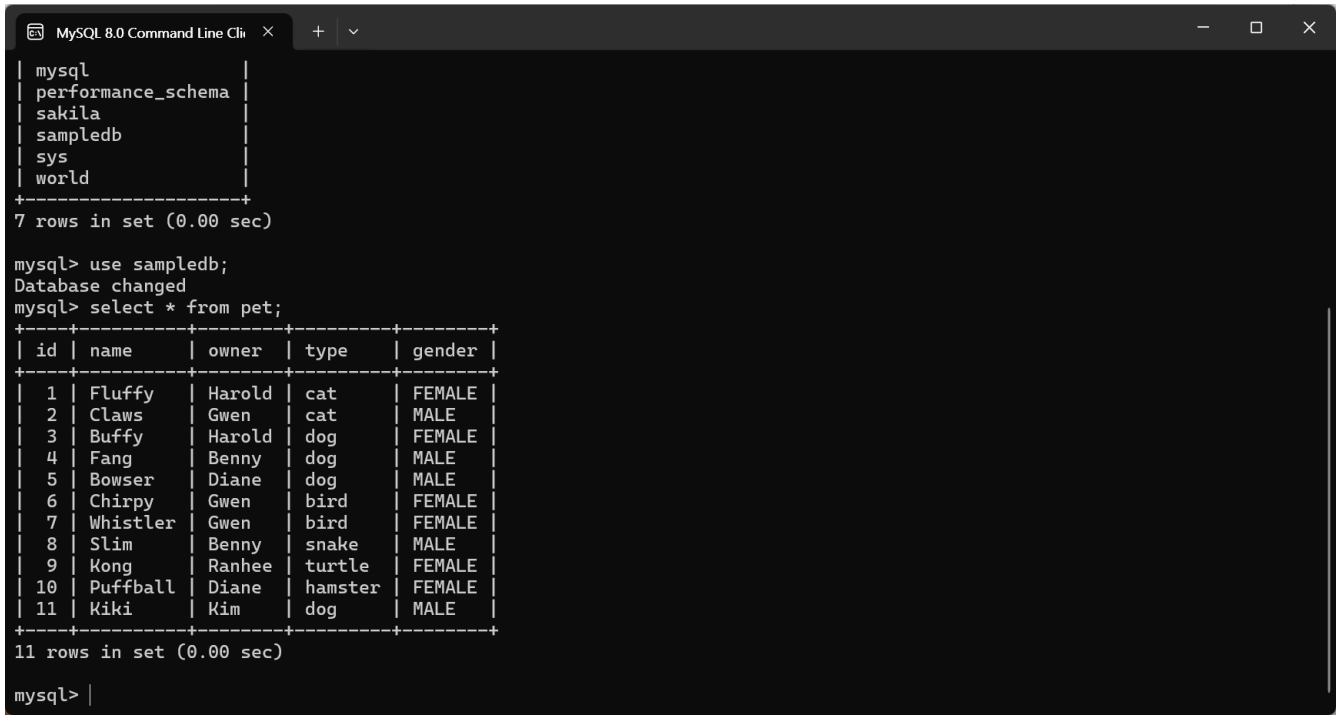
MySQL Workbench에서 DB 확인

- MySQL Workbench를 사용하여 DB 확인



데이터 베이스 접속 및 사용

- DB 사용
  - MySQL 8.0 Command Line Client 실행하여, 생성된 DB에 접속



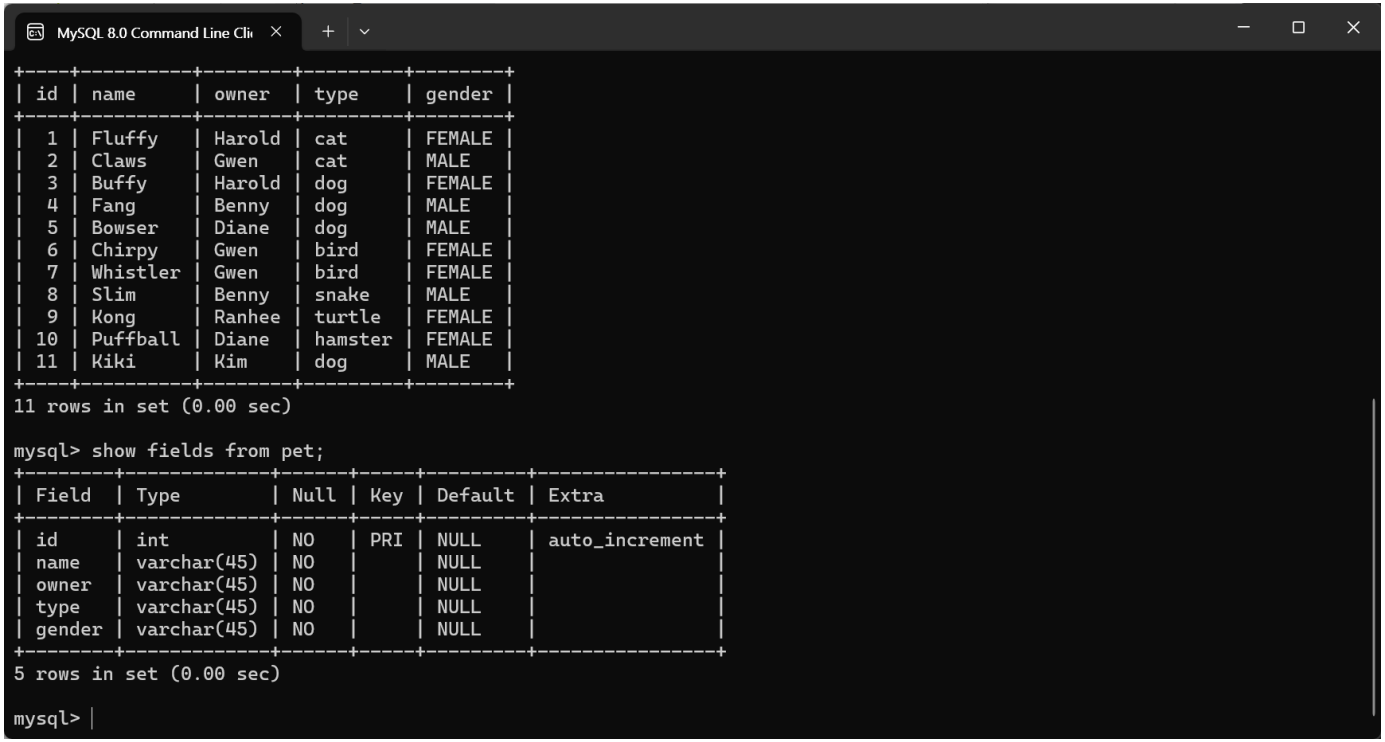
  - -u 옵션은 root 계정으로 명령 수행을 의미
  - use는 DB 사용 명령
  - sampledb는 사용할 DB 이름

테이블 생성

- 다음과 같은 구조의 pet 테이블 생성

id	name	owner	type	gender
INT	varchar(45)	varchar(45)	varchar(45)	varchar(45)

  - id 는 INT primary key not null auto_increment
  - name, owner, type, gender은 varchar 타입으로 45자



테이블 생성

- create 문으로 테이블 생성

```
mysql>CREATE TABLE pet (id INT NOT NULL AUTO_INCREMENT, name
VARCHAR(45) NOT NULL, owner VARCHAR(45) NOT NULL, type VA
RCHAR(45) NOT NULL, gender VARCHAR(45) NOT NULL, PRIMARY
KEY(id));
Query OK, 0 rows affected (0.02 sec)
```

  - 열 이름 데이터 타입(데이터 크기)을 콤마로 분리하여 나열
  - not null은 행의 데이터에 null값이 올 수 없음을 의미
  - primary key는 키로 사용될 행 지정

- drop명령은 기존의 테이블 삭제

```
mysql> DROP TABLE pet;
Query OK, 0 rows affected (0.01 sec)
```

테이블 생성

- desc명령은 생성된 테이블의 구조 표시

```
mysql> DESCRIBE pet;
+--------+------------+------+-----+---------+----------------+
| Field  | Type       | Null | Key | Default | Extra          |
+--------+------------+------+-----+---------+----------------+
| id     | int        | NO   | PRI | NULL    | auto_increment |
| name   | varchar(45) | NO  |     | NULL    |                |
| owner  | varchar(45) | NO  |     | NULL    |                |
| type   | varchar(45) | NO  |     | NULL    |                |
| gender | varchar(45) | NO  |     | NULL    |                |
+--------+------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

데이터 추가

- insert 문으로 테이블의 데이터 추가

```
mysql> INSERT INTO pet
    -> VALUES (12,'Happy', 'Park', 'cat', 'FEMALE');
Query OK, 1 row affected (0.10 sec)
mysql> INSERT INTO pet
    -> VALUES (13,'Baby', 'Park', 'dog', 'MALE');
Query OK, 1 row affected (0.10 sec)
```

  - insert into 다음에 테이블 이름 지정
  - 테이블 이름 다음 괄호 안에 열 이름을 콤마로 구분하여 나열
  - values 다음 괄호 안에 열의 값들을 콤마로 구분하여 나열
  - 문자 타입의 데이터는 단일 인용 부호로 묶어서 표시함에 유의

데이터 추가

- load문으로 파일로부터 테이블의 데이터 추가

```
mysql>LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/pet.csv' INTO TABLE pet FIELDS TERMINATED BY ',' LINES
TERMINATED BY '₩r₩n';
Query OK, 8 rows affected (0.10 sec)
mysql> SELECT * from pet;
+----+----------+--------+---------+--------+
| id | name     | owner  | type    | gender |
+----+----------+--------+---------+--------+
|  1 | Fluffy   | Harold | cat     | FEMALE |
|  2 | Claws    | Gwen   | cat     | MALE   |
|  3 | Buffy    | Harold | dog     | FEMALE |
|  4 | Fang     | Benny  | dog     | MALE   |
|  5 | Bowser   | Diane  | dog     | MALE   |
|  6 | Chirpy   | Gwen   | bird    | FEMALE |
|  7 | Whistler | Gwen   | bird    | FEMALE |
|  8 | Slim     | Benny  | snake   | MALE   |
+----+----------+--------+---------+--------+
```

데이터 검색

- select문으로 테이블 내의 데이터 검색

```
mysql> SELECT * from pet where owner='Gwen';
+----+---------+-------+------+--------+
| id | name    | owner | type | gender |
+----+---------+-------+------+--------+
|  2 | Claws   | Gwen  | cat  | MALE   |
|  6 | Chirpy  | Gwen  | bird | FEMALE |
|  7 | Whistler | Gwen | bird | FEMALE |
+----+---------+-------+------+--------+
3 rows in set (0.00 sec)
```

  - select 다음에는 데이터를 추출할 열 이름을 콤마로 분리하여 나열
  - 모든 열에 대해 데이터를 추출할 때는 *를 열 이름 대신 사용
  - from 다음에 테이블 이름을 지정
  - where 다음에 검색 조건 지정. where는 생략 가능

데이터 검색

- select문으로 테이블 내의 데이터 검색

```
mysql> SELECT name, type, gender from pet where owner='Gwen';
+---------+------+--------+
| name    | type | gender |
+---------+------+--------+
| Claws   | cat  | MALE   |
| Chirpy  | bird | FEMALE |
| Whistler | bird | FEMALE |
+---------+------+--------+
3 rows in set (0.00 sec)
mysql> select name, type, gender from pet where type='dog' and gender='MALE';
+-------+------+--------+
| name  | type | gender |
+-------+------+--------+
| Fang  | dog  | MALE   |
| Bowser | dog  | MALE   |
| Kiki  | dog  | MALE   |
+-------+------+--------+
3 rows in set (0.00 sec)
```

데이터 수정

- update문을 이용하여 데이터 수정

```
mysql> UPDATE pet set owner='Benny' where name='Puffball';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * from pet where owner='Benny';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | gender | birth      | death |
+----------+-------+---------+------+------------+-------+
| Fang     | Benny | dog     | m    | 1990-08-27 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
| Puffball | Benny | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
3 rows in set (0.00 sec)
```

  - update 다음에는 테이블 이름 지정
  - set 다음에 수정할 열의 이름과 값을 콤마로 분리하여 나열
  - where 다음에는 검색 조건을 지정. where는 생략 가능

데이터 삭제

- delete문을 이용하여 데이터 삭제

```
mysql> DELETE from pet where name='Puffball';
Query OK, 1 row affected (0.06 sec)
mysql> SELECT * from pet where owner='Benny';
+------+-------+---------+------+------------+-------+
| name | owner | species | gender  | birth       | death |
+------+-------+---------+------+------------+-------+
| Fang | Benny | dog     | m    | 1990-08-27 | NULL  |
| Slim | Benny | snake   | m    | 1996-04-29 | NULL  |
+------+-------+---------+------+------------+-------+
2 rows in set (0.00 sec)
```

  - delete from 다음에는 테이블 이름 지정
  - where 다음에는 검색 조건을 지정. 위의 예에서는 name 값이'Puffball'인 레코드 삭제
  - where는 생략 가능

Spring Boot에서 JDBC 프로그래밍

- pom.xml에 JDBC와 MySQL dependency 추가

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
</dependency>
```

Spring Boot에서 JDBC 프로그래밍

- application.properties에서 MySQL JDBC 드라이버 설정

```
# 스프링 데이터소스 드라이버 클래스 이름 입력
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# JDBC MySQL DB 접속 URL 입력
spring.datasource.url=jdbc:mysql://localhost:3306/sampledb
# JDBC MySQL DB 사용자 아이디와 비밀번호 입력
spring.datasource.username=root
spring.datasource.password=ace12345
```

Spring Boot에서 JDBC 프로그래밍

```
@RequiredArgsConstructor
@Repository
public class PetJdbcRepository implements PetRepository {
    private final DataSource dataSource; // DB Connection Manager Interface
    @Override
    public int insert(Pet pet) {
        String query = "insert into pet (name,owner,type,gender) values(?,?,?,?)";
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
                conn = getConnection();
                pstmt = conn.prepareStatement(query);
                // sql string의 '?,?,?,?'에 name, owner,type,gender 값을 바인딩
                pstmt.setString(1, pet.getName());
                pstmt.setString(2, pet.getOwner());
                pstmt.setString(3, pet.getType());
                pstmt.setString(4, pet.getGender().toString());
```

Spring Boot에서 JDBC 프로그래밍

```
            // executeUpdate
            return pstmt.executeUpdate();
        }
        catch (Exception e) {
            throw new IllegalStateException(e);
        } finally {
            close(conn, pstmt, rs);
        }
    }
    private Connection getConnection() {
        return DataSourceUtils.getConnection(dataSource);
    }
    private void close(Connection conn, PreparedStatement pstmt, ResultSet rs) {
        try {
            if (conn != null) {
                DataSourceUtils.releaseConnection(conn, dataSource);
            }
        }... 중간 생략
    }
```

Spring Boot에서 JDBC 프로그래밍

```
@Override
public List<Pet> findAll() {
    String query = "select * from pet";
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
            conn = getConnection();
            pstmt = conn.prepareStatement(query);

            // executeQuery
            rs = pstmt.executeQuery();
```

Spring Boot에서 JDBC 프로그래밍

```
            // get List<Pet>
            List<Pet> petList = new ArrayList<>();
            while (rs.next()) {
                Pet pet = new Pet();
                pet.setId(rs.getLong("id"));
                pet.setName(rs.getString("name"));
                pet.setOwner(rs.getString("owner"));
                pet.setType(rs.getString("type"));
                pet.setGender(Gender.valueOf(rs.getString("gender")));
                petList.add(pet);
            }
            return petList;
        }
        catch (Exception e) {
            throw new IllegalStateException(e);
        } finally {
            close(conn, pstmt, rs);
        }
    }
```

데이터베이스 사용

- Stetement 클래스
  - SQL문을 실행하기 위해서는 Statement 클래스를 이용
  - 주요 메소드

메소드	설명
ResultSet executeQuery(String sql)	주어진 SQL문을 실행하고 결과는 ResultSet 객체에 반환
int executeUpdate(String sql)	INSERT, UPDATE, 또는 DELETE과 같은 SQL문을 실행하고, SQL문 실행으로 영향을 받은 행의 개수나 0을 반환
void close()	Statement 객체의 데이터베이스와 JDBC 리소스를 즉시 반환

  - 데이터 검색을 위해 executeQuery() 메소드 사용
  - 추가, 수정, 삭제와 같은 데이터 변경은 executeUpdate() 메소드 사용

데이터베이스 사용

- ResultSet 클래스
  - SQL문 실행 결과를 얻어오기 위해서는 ResultSet 클래스를 이용
  - 현재 데이터의 행(레코드 위치)을 가리키는 커서(cursor)를 관리
  - 커서의 초기 값은 첫 번째 행 이전을 가리킴

메소드	설명
boolean first()	커서를 첫 번째 행으로 이동
boolean last()	커서를 마지막 행으로 이동
boolean next()	커서를 다음 행으로 이동
boolean previous()	커서를 이전 행으로 이동
boolean absolute(int row)	커서를 지정된 행으로 이동
boolean isFirst()	첫 번째 행이면 true 반환
boolean isLast()	마지막 행이면 true 반환
void close()	ResultSet 객체의 데이터베이스와 JDBC 리소스를 즉시 반환

데이터베이스 사용

- 테이블의 모든 데이터 검색

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from student");
```

  - Statement의 executeQuery()는 SQL문의 실행하여 실행 결과를 넘겨줌
  - 위의 SQL문의 student 테이블에서 모든 행의 모든 열을 읽어 결과를 rs에 저장

- 특정 열만 검색

```
ResultSet rs = stmt.executeQuery("select name, id from student");
```

  - 특정 열만 읽을 경우는 select문을 이용하여 특정 열의 이름 지정

- 조건 검색

```
rs = stmt.executeQuery("select name, id, dept from student where
id='0494013'");
```

  - select문에서 where절을 이용하여 조건에 맞는 데이터 검색

데이터베이스 사용

- 검색된 데이터의 사용

```
while (rs.next()) {
    System.out.println(rs.getLong("id"));
    System.out.println(rs.getString("name"));
    System.out.println(rs.getString("owner"));
    System.out.println(rs.getString("type"));
    System.out.println(rs.getString("gender"));
}
rs.close();
```

  - Statement객체의 executeQuery() 메소드
    - ResultSet 객체 반환
  - ResultSet 인터페이스
    - DB에서 읽어온 데이터를 추출 및 조작할 수 있는 방법 제공
  - next() 메소드
    - 다음 행으로 이동

데이터베이스 사용

- ResultSet의 getXXX() 메소드
  - 해당 데이터 타입으로 열 값을 읽어옴
  - 인자로 열의 이름이나 인덱스를 줄 수 있음
  - DB 데이터 타입에 해당하는 자바 데이터 타입으로 데이터를 읽어야 함.
  - 모든 데이터 타입에 대해 getString() 메소드로 읽을 수 있으나 사용할 때는 해당 데이터 타입으로 변환해서 사용
- ResultSet에서 모든 데이터를 읽은후 close()를 호출하여 자원 해제

메소드	설명
Xxx getXxx(String columnLable)	ResultSet 객체의 데이터베이스와 JDBC 리소스를 즉시 반환 Xxx는 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 이름에 해당하는 데이터를 반환. 예를 들어, int형 데이터를 읽는 메소드는 getInt()이다.
Xxx getXxx(int columnIndex)	Xxx는 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 인덱스에 해당하는 데이터를 반환. 예를 들어, int형 데이터를 읽는 메소드는 getInt()이다.

한글 처리 문제

- 문자열 코드 문제
  - MySQL의 문자 집합은 ISO-8859-1
    - Unicode를 사용하는 자바에서 한글이 깨져 출력
    - ISO-8859-1를 Unicode로 변환하여 출력해야 함

```
String ISO_8859_1String = rs.getString("name");
byte [] UnicodeBytes = ISO_8859_1String.getBytes("ISO-8859-1");
String UnicodeString = new String(UnicodeBytes);
System.out.print(UnicodeString);
```

유니코드로 변환

    - 반대로 MySQL에서 사용될 한글이 포함된 문자열은 Unicode에서 ISO-8859-1로 변환해야 MySQL에서 정상적으로 처리됨

```
byte [] UnicodeBytes = "홍길동".getBytes();
String IOS_8859_1String = new String(UnicodeBytes, "ISO-8859-1");
stmt.executeQuery("select name, id, dept from student where name='"+
IOS_8859_1String +"'");
```

유니코드를 ISO-8859-1로 변환

데이터의 변경

- 레코드 추가

```
stmt.executeUpdate("INSERT INTO pet VALUES (1,'Puffball', 'Diane',
'hamster', 'FEMALE');₩r₩n");
```

  - DB에 변경을 가하는 조작은 executeUpdate() 메소드 사용
  - SQL문 수행으로 영향을 받은 행의 개수 반환

- 데이터 수정

```
stmt.executeUpdate("UPDATE pet set owner ='Benny' where
name='Puffball'");
```

  - where문의 MySQL에서 처리되므로 문자열을 Unicode에서 ISO-8859-1로 변환에 주의

- 데이터 삭제

```
stmt.executeUpdate("DELETE from pet where name='Puffball'");
```