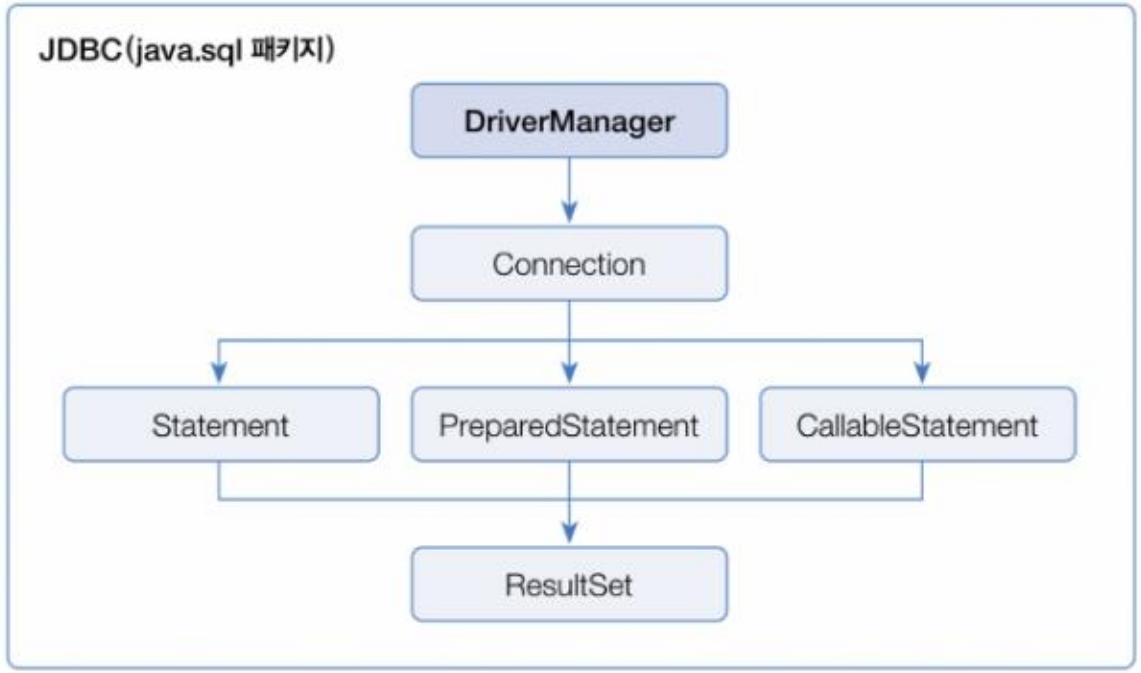
Java

데이터베이스 입출력

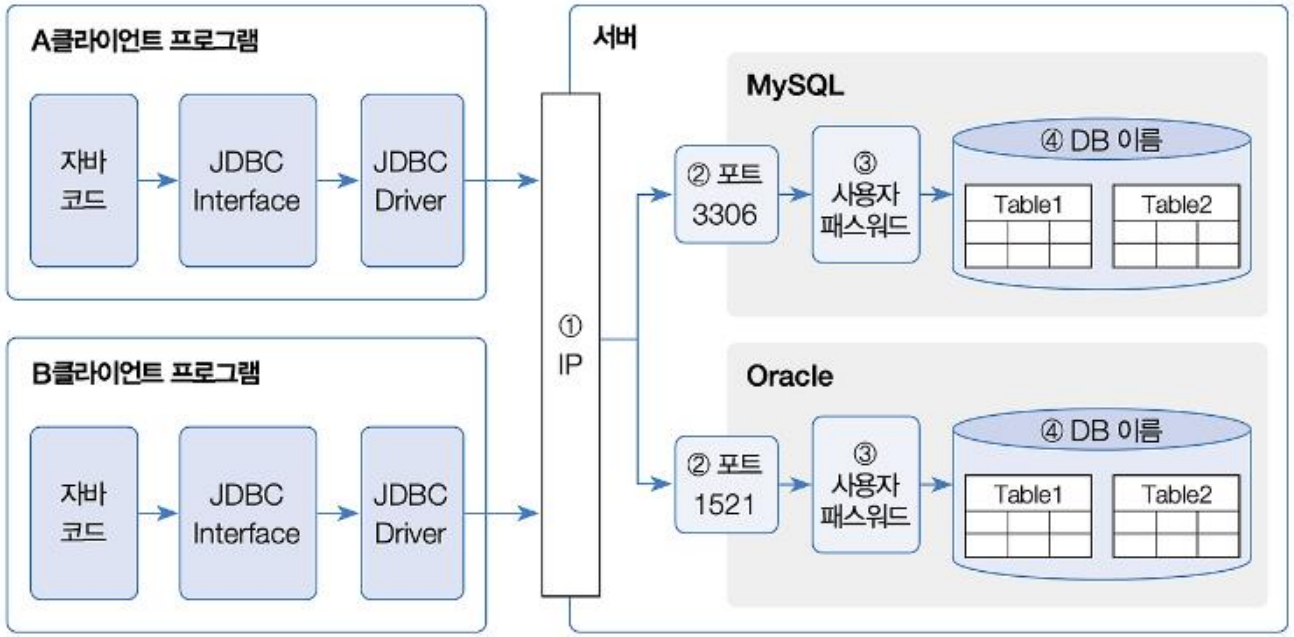
- Java는 DB와 연결해서 데이터 입출력 작업을 할 수 있도록 JDBC(Java Database Connectivity) 라이브러리를 제공한다.



데이터베이스 입출력

- JDBC Driver : JDBC 인터페이스를 구현한 것으로 DBMS마다 별도로 라이브러리를 다운로드 받아 사용해야 한다.

- DB 연결을 위해서는 IP 주소, DBMS 포트번호, 사용자 이름 및 비밀번호, 접속하려는 DB 이름이 필요하다.



데이터베이스 입출력

- DB를 연결하기 위해서는 JDBC Driver를 먼저 메모리로 로딩하는 작업이 필요하다.
- JDBC Driver 클래스의 static 블록이 실행되면, DriverManager에 JDBC Driver 객체를 등록한다.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

데이터베이스 입출력

- DriverManager 클래스
  - JDBC Driver를 등록 및 관리하고, DB와 연결을 생성하는 클래스
  - 정적메소드 getConnection( )을 통해 연결하고, Connection 구현 객체를 반환한다.

```
String url = "jdbc:mysql://localhost:3306/kostagram";
String user = "root";
String password = "1234";

Connection conn = DriverManager.getConnection(url, user, password);
```

데이터베이스 입출력

- Connection 인터페이스
  - 데이터베이스 연결을 나타내는 인터페이스
  - SQL에 요청할 객체를 생성할 때 사용된다.
  - createStatement() 메서드로 Statement 구현 객체를 반환한다.
  - prepareStatement() 메서드로 PreparedStatement 구현 객체를 반환한다.

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

```
String sql = "SELECT * FROM users where id = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
```

데이터베이스 입출력

- Statement 인터페이스
  - executeQuery( ) 메소드를 통해 SQL문 실행 요청을 하고, 결과를 ResultSet 구현 객체로 반환한다.
  - 주로 정적 SQL문을 실행할 때 사용된다.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```

- PreparedStatement 인터페이스
  - setXXX( ) 메소드로 SQL문에 '?' 표시된 파라미터 값을 설정한다.
  - executeQuery( ) 메소드를 통해 SQL문 실행 요청을 하고, 결과를 ResultSet 구현 객체로 반환한다.

```
pstmt.setInt(1, 1);
ResultSet rs = pstmt.executeQuery();
```

데이터베이스 입출력

- ResultSet 인터페이스
  - SQL문의 실행 결과를 나타내는 인터페이스이다.
  - next( ) 메소드로 결과 집합의 다음 레코드로 이동한다.
  - getXXX() 메소드로 결과 집합의 값을 가져온다.

```
while (rs.next()) {
System.out.println(
    rs.getInt("id") + "\t" + rs.getString("email") + "\t" +
    rs.getString("nickname") + "\t" + rs.getString("password")
);
}
```

DB 연동

- Java와 DBMS를 연동하면, 값을 저장하고, 데이터를 가져오고, 수정 또는 삭제를 하는 작업을 Java에서 할 수 있게 된다.
- Java에 작성된 특정 코드가 실행되면, 지정된 DBMS으로 쿼리를 전송하고 결과를 얻어 오게끔 하는 것이다.

- 쉽게 말해, Java를 통해 데이터베이스에 접근할 수 있는 별도의 프로그램을 하나 만드는 것이라 볼 수 있다.

DB 연결

- 클라이언트 프로그램에서 DB와 연결하려면 몇 가지 준비가 필요하다.

    1. 해당 DBMS의 JDBC Driver
    2. DBMS가 설치된 컴퓨터의 IP 주소
    3. DBMS가 허용하는 포트(Port) 번호
    4. DB 사용자 계정 및 비밀번호
    5. 사용하고자 하는 DB 이름

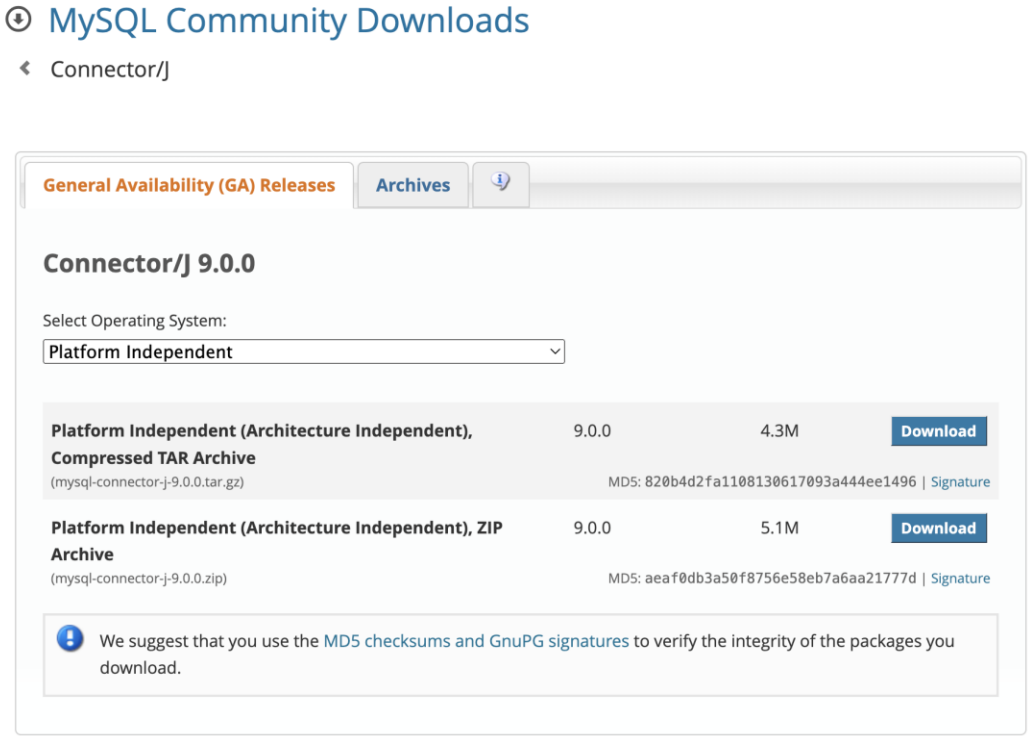
- IP 주소는 컴퓨터를 찾아가기 위해, PORT 번호는 DBMS로 연결하기 위해 필요하다.
  또한 여러 개의 DB 중 사용할 DB의 이름과 사용자 인증을 위한 계정과 비밀번호도 필요하다.

DB 연동 – MySQL Connector/J 다운로드

- DBMS마다 제공하는 JDBC Driver는 상이하며, MySQL의 경우에는 아래의 링크에서 다운로드 받을 수 있다.

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

- 다운로드 받은 .jar 파일을 Build Path에 추가하여 사용하면 된다.



DB 연결 - 1.JDBC 드라이버 로딩

- 클라이언트 프로그램을 DB와 연결하는 가장 첫번째 작업은 JDBC Driver를 메모리로 로딩하는 것이다.
- Class.forName( ) 메소드는 문자열로 주어진 JDBC Driver 클래스를 Build Path에서 찾고, 메모리에 로딩한다.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- 이 과정에서 JDBC Driver 클래스의 static 블록이 실행되면서 DriverManager에 JDBC Driver 객체를 등록하게 된다.
- 만약 Build Path에서 JDBC Driver 클래스를 찾지 못하면 ClassNotFoundException을 발생시킨다.

DB 연결 - 2. Database 연결

- DriverManager 클래스에 JDBC Driver가 등록되면, getConnection( )라는 정적 메소드로 DB와 연결할 수 있다.

```
String url = "jdbc:mysql://localhost:3306/board";
String user = "root";
String password = "1234";

Connection conn = DriverManager.getConnection(url, user, password);
```

DB 연결 - 3. Database 연결 끊기

- 성공했던 DB 연결을 끊기 위해서는 Connection 객체의 close( ) 메소드를 호출한다.

```
if (conn != null) {
    try {
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

DB 연결 - 데이터 삽입

```
public class DatabaseConnection {
    public static Connection conn = null; // DB 연결 객체
    public static void main(String[] args) {
        try {
            // 1. JDBC 드라이버 로딩
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. 데이터베이스 연결
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/kostagram", "root", "1234");
        } catch (Exception e) {
            e.printStackTrace();
        }
        // 3. 연결 종료
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

DB 연결 - 데이터 삽입

- users 테이블에 새로운 사용자 정보(id, email, name, password)를 저장해보자.
- 값을 ?(물음표)로 대체한 매개변수화된 INSERT문은 아래와 같다.

```
INSERT INTO users (name, email, password)
VALUES (?, ?, ?)
```

- 위의 INSERT문을 String 타입 변수로 대입한다.

```
String sql = new StringBuilder()
    .append("INSERT INTO users (name, email, password) ")
    .append("VALUES (?, ?, ?)")
    .toString();


String sql = "INSERT INTO users (name, email, password) " +
        "VALUES (?, ?, ?)";
```

DB 연결 - 데이터 삽입

- 해당 문자열을 실행하기 위해서는 PreparedStatement가 필요하다.
- Connection의 prepareStatement( ) 메소드로부터 PreparedStatement 구현 객체를 얻을 수 있다.

```
PreparedStatement pstmt = conn.prepareStatement(sql);
```

- 그리고 "? 에 들어갈 값을 지정해주어야 하는데, "?"는 순서에 따라 1번부터 번호가 부여된다.
- 따라서 값 | 타입에 따라 Setter 메소드를 선택한 후, 순서와 값을 매개변수로 전달한다.

```
pstmt.setString(1, "최인규");
pstmt.setString(2, "cik@gmail.com");
pstmt.setString(3, "1234");
```

DB 연결 - 데이터 삽입

- 값을 지정한 후, PreparedStatement의 executeUpdate( ) 메소드를 호출하면,
  SQL문이 실행되면서 users 테이블에 1개의 행이 저장된다.

- executeUpdate( ) 메소드는 저장된 행의 개수를 int 값으로 반환한다.
  즉, 정상적으로 실행된 경우에는 1을 반환한다.

```
int resultRow = pstmt.executeUpdate();
```

- 만약 PreparedStatement를 더 이상 사용하지 않는다면, close( ) 메소드를 호출해 메모리를 해제해준다.

```
pstmt.close();
```

[실습] 게시판 데이터 삽입하기

```
String sql = "INSERT INTO posts (user_id, content, image) VALUES (?, ?, ?)";


PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, 1);
pstmt.setString(2, "내용");
pstmt.setString(3, "image.png");
int resultRow = pstmt.executeUpdate();
```

DB 연결 - 데이터 수정

- boards 테이블에 작성된 게시물 내용(id, email, name, password)을 수정해보자.
- 값을 ?(물음표)로 대체한 매개변수화된 UPDATE문은 아래와 같다.

```
UPDATE posts SET content=? WHERE id = 1;
```

- 위의 UPDATE문을 String 타입 변수로 대입한다.

```
String sql = "UPDATE posts SET content = ? WHERE id = 1";

PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "변경된 내용");


int resultRow = pstmt.executeUpdate();
```

DB 연결 - 데이터 삭제

```
String sql = "DELETE FROM posts WHERE id = ?";


PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, 1);
int resultRow = pstmt.executeUpdate();
```

DB 연결 - 데이터 읽기

- PreparedStatement를 생성할 때,
  SQL문이 INSERT, UPDATE, DELETE인 경우에는 executeUpdate( ) 메소드를 호출하지만,
  데이터를 가져오는 SELECT 문일 경우에는 executeQuery( ) 메소드를 호출해야 한다.

- executeQuery( ) 메소드는 가져온 데이터를 ResultSet에 저장하고 반환한다.

```
String sql = "SELECT name, email, password FROM users";


PreparedStatement pstmt = conn.prepareStatement(sql);
ResultSet rs = pstmt.executeQuery();
```

DB 연결 - 데이터 읽기

- ResultSet은 SELECT 문에 기술된 컬럼으로 구성되어 있는 행의 집합이다.

- ResultSet에서 데이터는 아래와 같이 컬럼 이름 또는 컬럼 순번을 통해 가져올 수 있다.

```
while(rs.next()) {
    String userName = rs.getString("name");
    String userEmail = rs.getString("email");
    String userPw = rs.getString("password");
}

rs.close();
pstmt.close();
```

```
while(rs.next()) {
    String userName = rs.getString(1);
    String userEmail = rs.getString(2);
    String userPw = rs.getString(3);
}

rs.close();
pstmt.close();
```

- 만약 가져오려는 데이터가 1개의 행으로 구성된 경우에는 while 대신에 if를 사용할 수 있다.
- 만약 SQL 컬럼명에 별명이 있다면, 별명이 컬럼 이름이 된다.

DB 연결 - 실습

- User, Post, KostagramExample 클래스 생성

- KostagramExample
  - 필드 : Scanner, Connection
  - 기본 생성자 : JDBC Driver 등록 및 DB 연결
  - static void main(String[] args) : 객체 생성 및 사용자 메뉴 호출

  - void mainMenu( ) : 사용자 메뉴
    1. 회원 관리
       1) 회원가입
       2) 특정 유저 팔로우
       3) 특정 유저 언팔로우
       4) 팔로우 리스트
       5) 회원 탈퇴
    2. 게시물 관리
       1) 전체 게시물
       2) 좋아요
       3) 좋아요 취소
    3. 프로그램 종료
  - void exit( ) : 프로그램 종료

  - void join( ) : 회원가입
  - void secession( ) : 회원 탈퇴
  - void follow( ) : 특정 유저 팔로우
  - void followList( ) : 선택한 유저 팔로우 리스트 출력

  - void list( ) : 전체 게시물 조회
  - void create( ) : 게시물 추가
  - void read( ) : 게시물 조회
  - void update( ) : 게시물 수정
  - void delete( ) : 게시물 삭제
  - void like( ) : 게시물 좋아요
  - void unlike( ) : 게시물 좋아요 취소