Java에서 데이터베이스와 상호 작용 (1)

JDBC를 살펴보면서…

2025.10.29.

System Software Lab.

오늘 할 이야기

0 1 1997년 이전의 Java에서 데이터베이스와 상호 작용했던 방법

0 2 1997년 이후 Java에서 데이터베이스와 상호 작용했던 방법, JDBC

0 3 JDBC를 사용해 Java와 데이터베이스 간 상호 작용

아주 옛날, Java에서 데이터베이스와 상호 작용

- 데이터베이스에서 C/C++ 클라이언트 라이브러리를 제공하는 경우,
  - JNI*(Java Native Interface)*를 사용하여 C/C++ 애플리케이션과 상호 작용할 수 있는 native 메서드를 선언
  - Java 컴파일러가 생성한 C/C++ 헤더 파일을 불러와 C/C++ 소스 코드 구현
    - 소스 코드 내에는 데이터베이스 라이브러리의 함수를 호출

특징

- 데이터베이스 종속적이므로, 운영 상 데이터베이스 변경이 필요한 경우 새로운 C/C++ 소스 코드를 구현해야 함
- C/C++로 컴파일된 프로그램은 플랫폼 종속적이므로 Java의 WORA*(Write Once, Run Anywhere)*원칙을 깸
- 두 언어를 번갈아 디버깅을 수행해야 하므로 유지보수가 까다로움

아주 옛날, Java에서 데이터베이스와 상호 작용

- Java 애플리케이션과 데이터베이스 간 게이트웨이 애플리케이션을 둠
  - Java 애플리케이션이 게이트웨이 애플리케이션에 요청을 보내
  - 게이트웨이 애플리케이션이 데이터베이스에 요청을 보내고 결과를 가져와
  - Java 애플리케이션에 전송

특징

- Java 애플리케이션은 플랫폼(OS, 데이터베이스 등) 종속적이지 않음
- 프로젝트별 프로토콜, 응답 포맷, 예외 처리 방식이 달라 재사용이 어려움

아주 옛날, Java에서 데이터베이스와 상호 작용

표준의 필요성

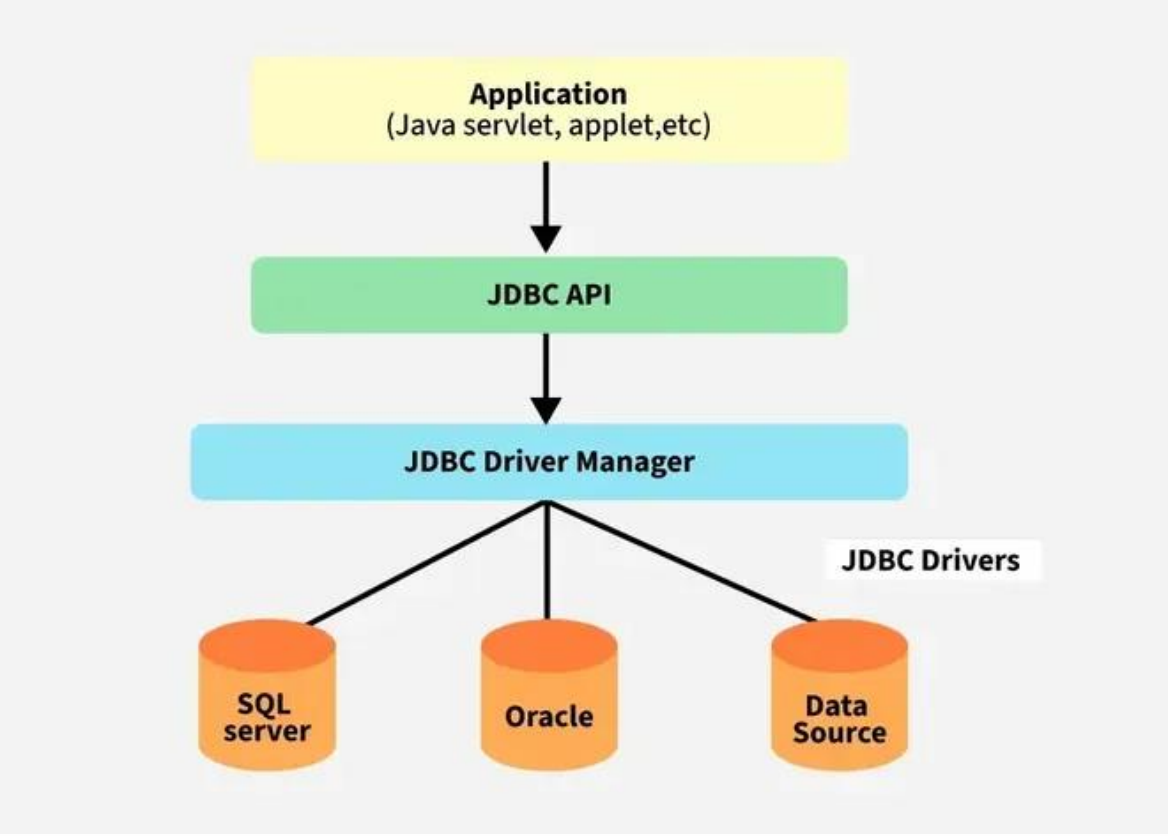
- 데이터베이스별로 제공하는 클라이언트 라이브러리 소스 코드가 다르므로

  요구사항 변화에 따른 데이터베이스 변경 시 애플리케이션을 갈아엎는 수준을 요구
- 트랜잭션, 데이터베이스 커넥션 관리 등을 직접 구현해야 함
- C/C++에서 발생하는 메모리 접근 오류, 메모리 관리 등 Java에서 해결할 수 있는 수준이 아닌 오류가 발생하고,

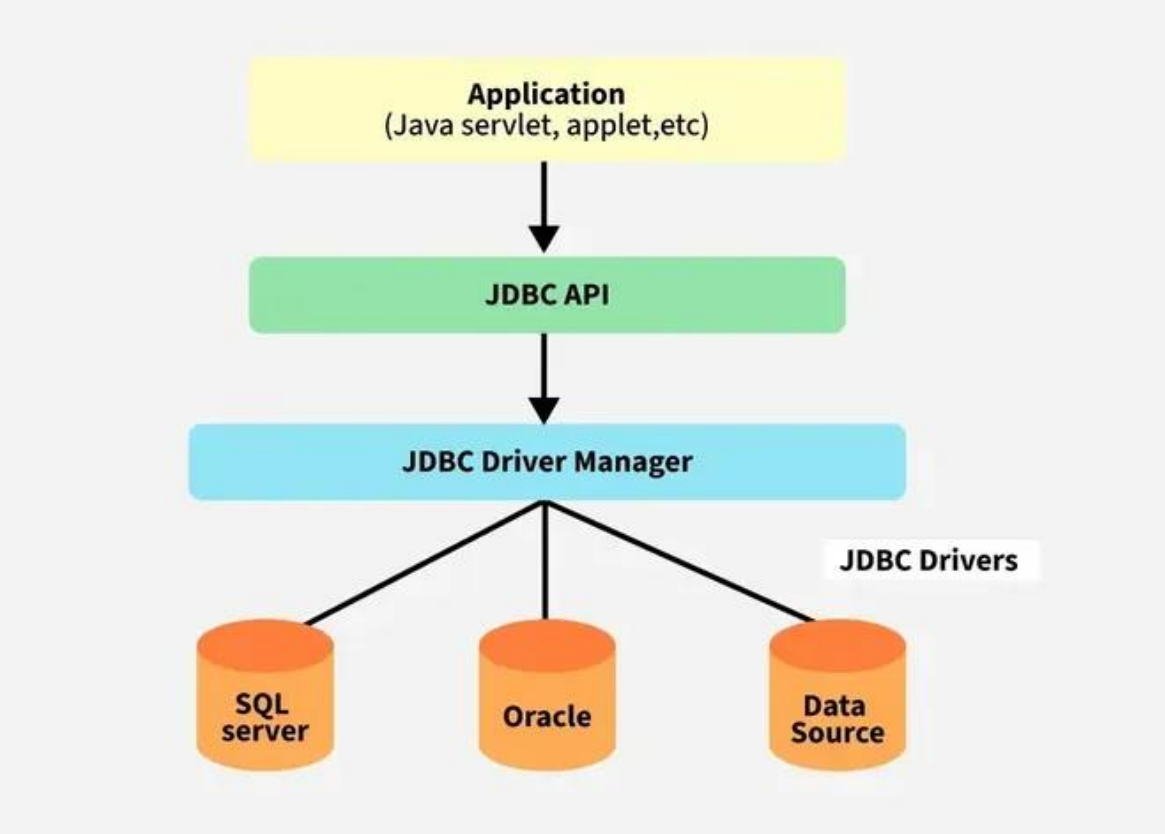
  두 언어를 번갈아 디버깅하기 까다로움

JDBC *(Java DataBase Connectivity)*

- 1997년 JDK 1.1과 함께 출시
- Java 기반 애플리케이션의 데이터를 데이터베이스에 저장, 갱신하거나
  데이터베이스에 저장된 데이터를 Java 애플리케이션에서 사용할 수 있도록 하는 Java API



JDBC *(Java DataBase Connectivity)*



JDBC API

- Java 애플리케이션이 SQL 쿼리를 실행하고 데이터베이스에서 결과를 얻을 수 있도록 정의한 인터페이스

java.sql.Connection

데이터베이스 연결을 위한 표준 인터페이스

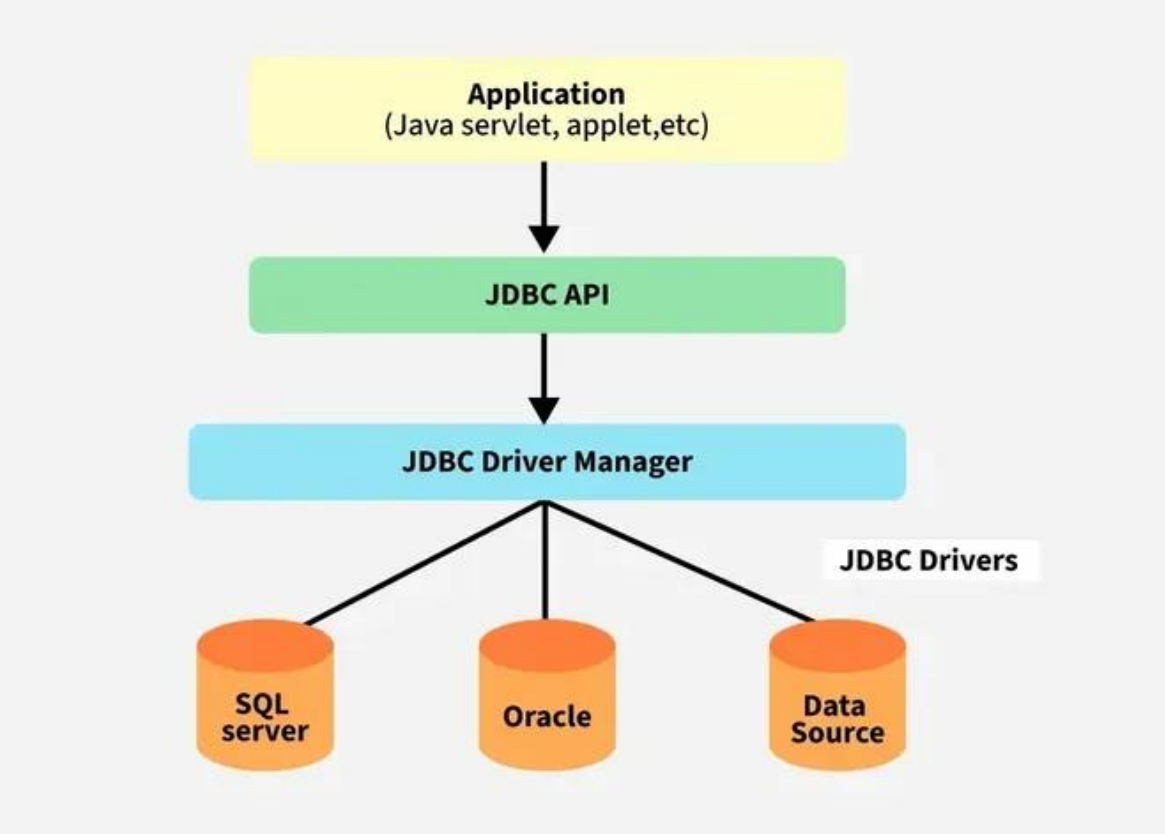
java.sql.Statement

SQL 쿼리를 담기 위한 표준 인터페이스

java.sql.ResultSet

SQL 요청에 대한 응답을 담기 위한 표준 인터페이스

JDBC *(Java DataBase Connectivity)*



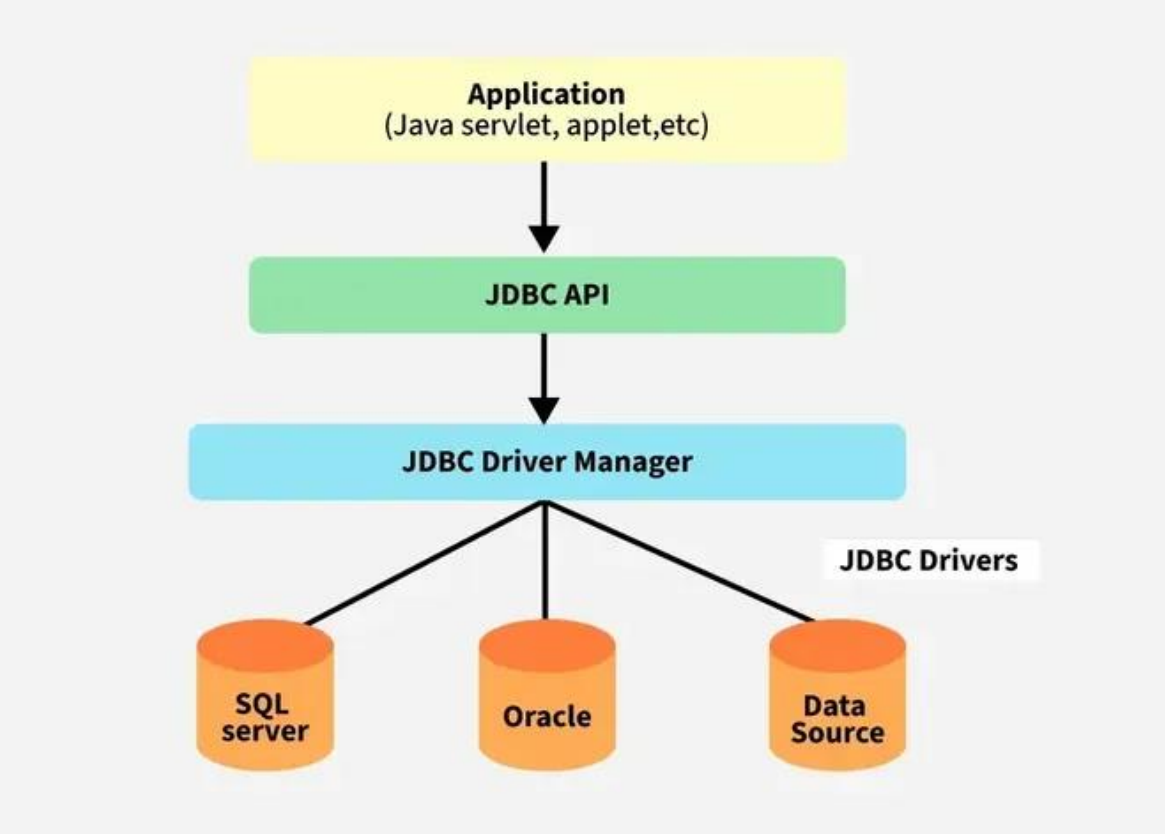
JDBC Driver Manager

- 라이브러리로 등록된 DB 드라이버를 관리하고 데이터베이스 커넥션을 획득하는 객체

```
/**
 * This method returns a driver that can connect to the specified
 * JDBC URL string.  This will be selected from among drivers loaded
 * at initialization time and those drivers manually loaded by the
 * same class loader as the caller.
 *
 * @param url The JDBC URL string to find a driver for.
 *
 * @return A <code>Driver</code> that can connect to the specified
 * URL.
 *
 * @exception SQLException If an error occurs, or no suitable driver can be found.
 */
public static Driver getDriver(String url) throws SQLException
{
  // FIXME: Limit driver search to the appropriate subset of loaded drivers.
  Enumeration e = drivers.elements();
  while(e.hasMoreElements())
    {
  Driver d = (Driver)e.nextElement();
  if (d.acceptsURL(url))
        return d;
    }

  throw new SQLException("No driver found for " + url);
}
```

java.sql.DriverManager

JDBC *(Java DataBase Connectivity)*



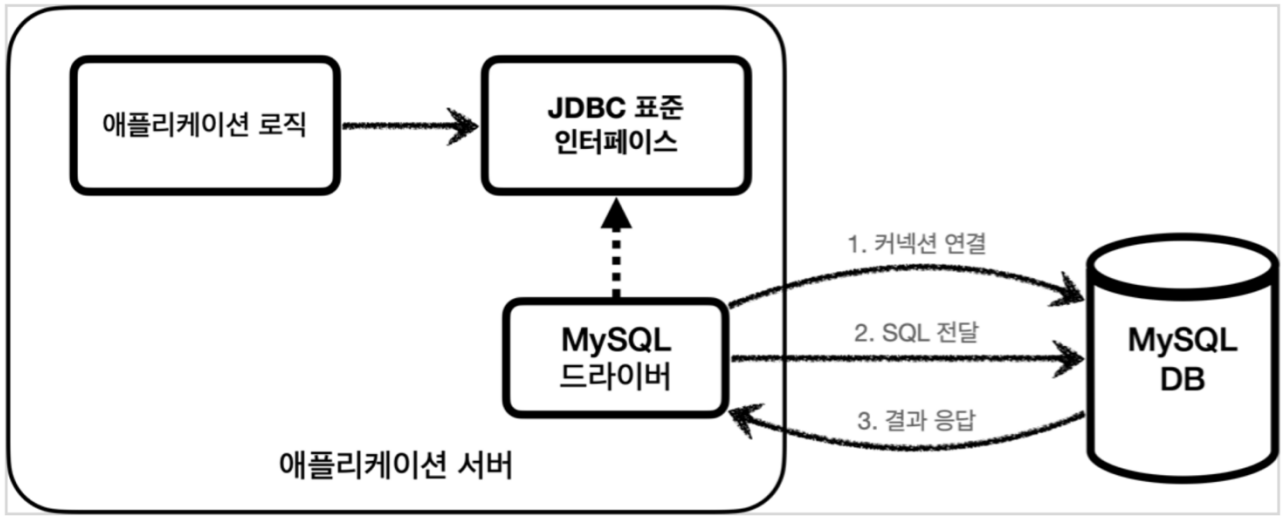
JDBC Driver

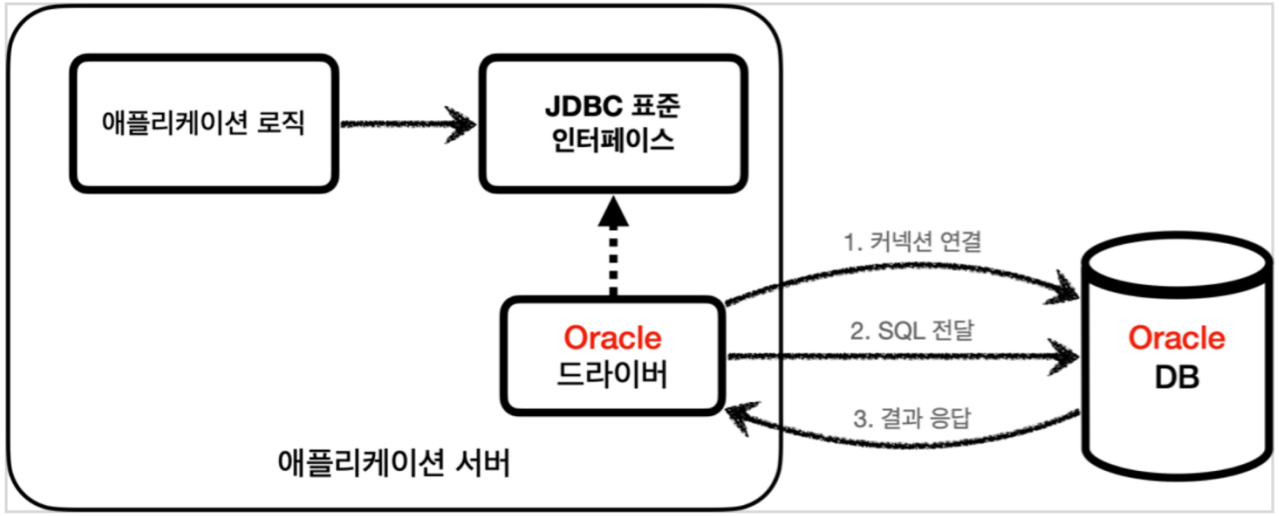
- 데이터베이스에서 제공하는 데이터베이스 연결을 위한 Java 라이브러리

```
/**
 * Check if the driver understands this URL.
 * This method should not be called by an application.
 *
 * @param url the database URL
 * @return if the driver understands the URL
 * @throws SQLException if URL is {@code null}
 */
@Override
public boolean acceptsURL(String url) throws SQLException {
    if (url == null) {
        throw DbException.getJdbcSQLException(ErrorCode.URL_FORMAT_ERROR_2,
    } else if (url.startsWith(Constants.START_URL)) {
        return true;
    } else if (url.equals(DEFAULT_URL)) {
        return DEFAULT_CONNECTION.get() != null;
    } else {
        return false;
    }
}
```

org.h2.Driver

JDBC *(Java DataBase Connectivity)*



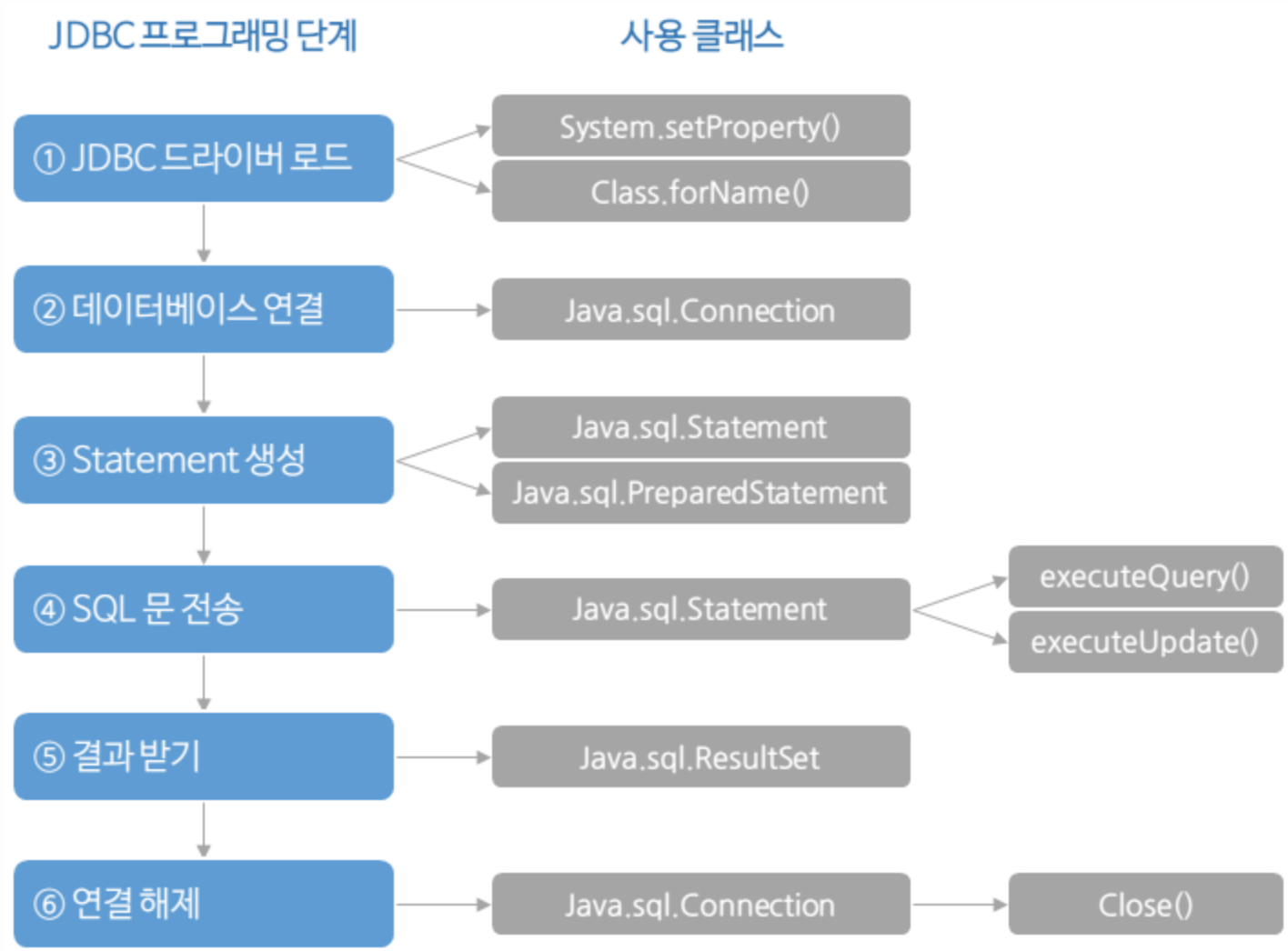


JDBC *(Java DataBase Connectivity)*

- 애플리케이션은 데이터베이스에서 제공하는 라이브러리가 아닌,
  JDBC 인터페이스만 의존하므로 요구사항 변경에 따른 데이터베이스 변경 시
  JDBC 구현 라이브러리인 드라이버와 데이터베이스만 교체하면 됨
- 데이터베이스별 커넥션 연결, SQL 전송, 결과 응답 방법을 새로 학습하지 않고
  JDBC 표준 인터페이스 사용 방법만 학습하면 되므로 학습 곡선*(Learning curve)* 낮음

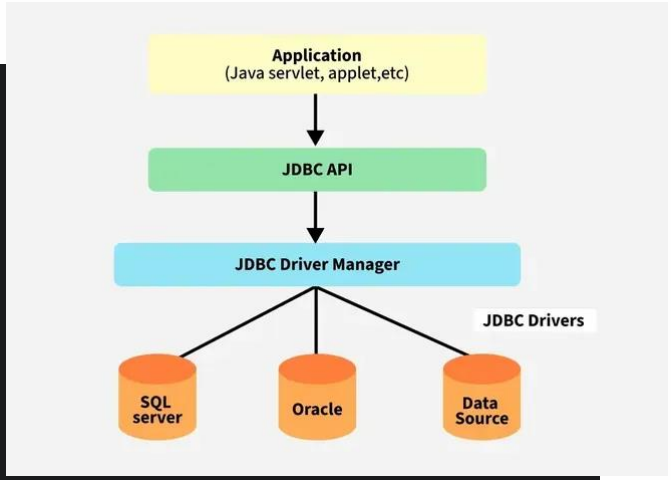
- 다만, 데이터베이스별로 SQL과 데이터 타입 등이 다를 수 있으므로
  애플리케이션에서 사용하는 SQL은 데이터베이스에 맞게 변경 필요
- 이를 해결하기 위해 나온 것이 MyBatis, Hibernate와 JPA*(Java Persistence API)* 등이며 다음에 발표할 예정

JDBC 사용 (0) : JDBC 실행 과정



JDBC 사용 (1) : 애플리케이션과 데이터베이스 간 연결 생성

```java
@SpringBootApplication
public class DemoApplication {
    private static final String DATASOURCE_DRIVER = "org.h2.Driver";  no usages
    private static final String DATASOURCE_URL = "jdbc:h2:./datasource/ssl";  1 usage
    private static final String DATASOURCE_USERNAME = "ssl251029";  1 usage
    private static final String DATASOURCE_PASSWORD = "kumohD449!";  1 usage

    public static void main(String[] args) throws SQLException {
        SpringApplication.run(DemoApplication.class, args);

        // 데이터베이스 연결 (try-with-resource는 자원을 자동으로 반납)
        try (Connection conn = DriverManager.getConnection(DATASOURCE_URL, DATASOURCE_USERNAME, DATASOURCE_PASSWORD)) {
            // 데이터베이스 연결 확인
            System.out.println(conn.isValid( timeout: 1) ? "데이터베이스 연결 성공" : "데이터베이스 연결 실패");
        } catch (Exception e) {
            System.out.println("데이터베이스 연결 실패 또는 잘못된 오류 발생: " + e);
        }
    }
}
```




```
데이터베이스 연결 성공
```

JDBC 사용 (1) : 애플리케이션과 데이터베이스 간 연결 생성

```java
@SpringBootApplication
public class DemoApplication {
    private static final String DATASOURCE_DRIVER = "org.h2.Driver";  no usages
    private static final String DATASOURCE_URL = "jdbc:h2:./datasource/ssl";  1 usage
    private static final String DATASOURCE_USERNAME = "ssl251029";  1 usage
    private static final String DATASOURCE_PASSWORD = "kumohD449!!";  usage

    public static void main(String[] args) throws SQLException {
        SpringApplication.run(DemoApplication.class, args);

        // 데이터베이스 연결 (try-with-resource는 자원을 자동으로 반납)
        try (Connection conn = DriverManager.getConnection(DATASOURCE_URL, DATASOURCE_USERNAME, DATASOURCE_PASSWORD)) {
            // 데이터베이스 연결 확인
            System.out.println(conn.isValid( timeout: 1) ? "데이터베이스 연결 성공" : "데이터베이스 연결 실패");
        } catch (Exception e) {
            System.out.println("데이터베이스 연결 실패 또는 잘못된 오류 발생: " + e);
        }
    }
}
```

kumohD449!에서 kumohD449!!로 변경

```
데이터베이스 연결 실패 또는 잘못된 오류 발생: org.h2.jdbc.JdbcSQLInvalidAuthorizationSpecException: Wrong user name or password [28000-232]
```

JDBC 사용 (2) : 애플리케이션과 데이터베이스 간 쿼리 전송

```
public static void main(String[] args) throws SQLException {
    SpringApplication.run(DemoApplication.class, args);

    // 데이터베이스 연결 (try-with-resource는 자원을 자동으로 반납)
    try (Connection conn = DriverManager.getConnection(DATASOURCE_URL, DATASOURCE_USERNAME, DATASOURCE_PASSWORD)) {
        // 데이터베이스 연결 확인
        System.out.println(conn.isValid( timeout: 1) ? "데이터베이스 연결 성공" : "데이터베이스 연결 실패");

        String[] departmentList = {"건축공학전공","건축학전공","토목공학전공","환경공학전공","기계공학전공","스마트모빌리티전공
        String insertMemberQuery = "insert into member (id, name, department) values (?, ?, ?);";
        PreparedStatement pstmt = conn.prepareStatement(insertMemberQuery);
        Random random = new Random();
        for (int i = 1; i ≤ 100; ++i) {
            pstmt.setLong( parameterIndex: 1, i);
            pstmt.setString( parameterIndex: 2, x: "사용자 " + i);
            pstmt.setString( parameterIndex: 3, departmentList[random.nextInt(departmentList.length)]);
            pstmt.execute();
        }

        // 데이터베이스에서 member 테이블 조회
```

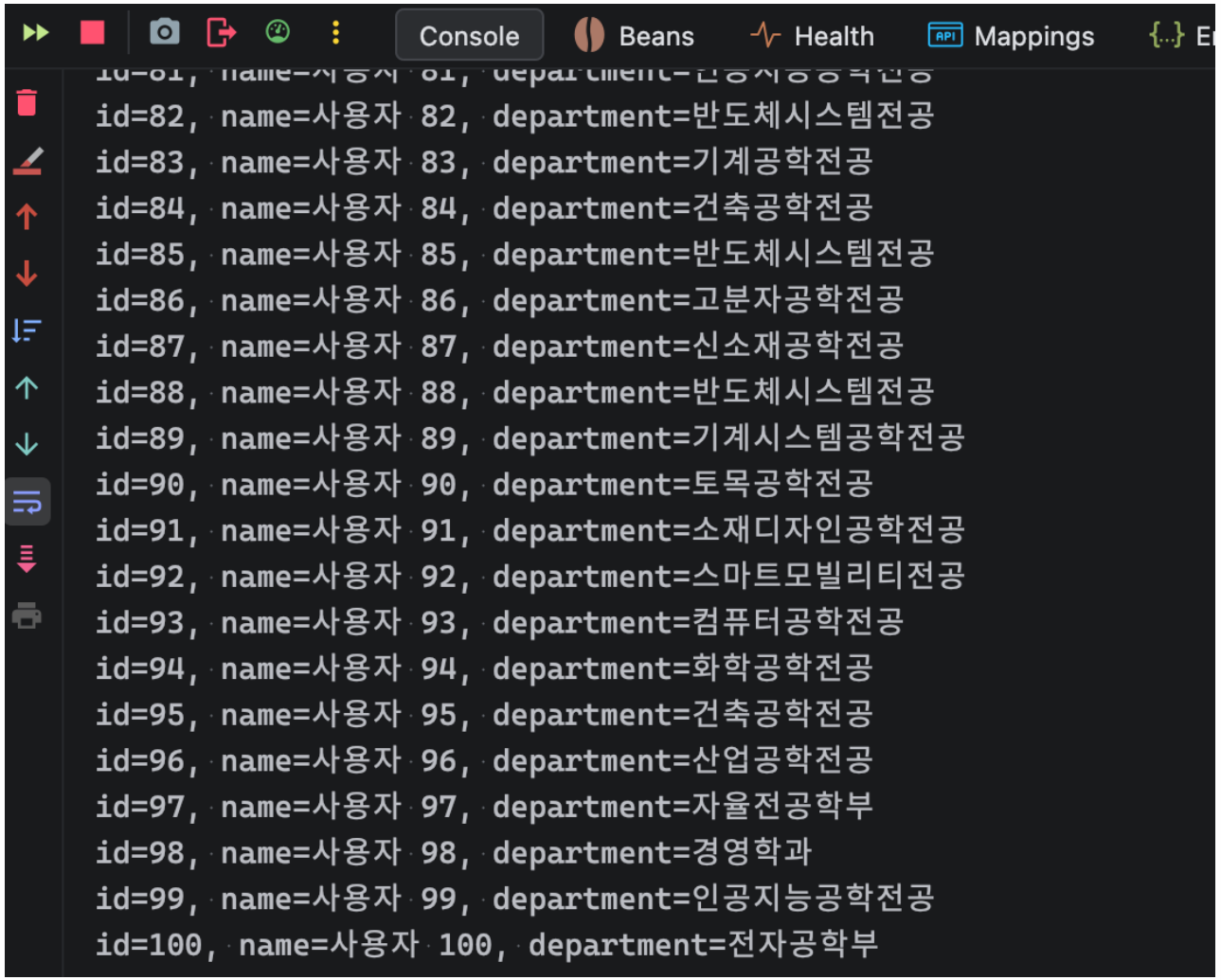
JDBC 사용 (3) : 애플리케이션에서 데이터베이스 쿼리 결과 이용

```java
public static void main(String[] args) throws SQLException {
    SpringApplication.run(DemoApplication.class, args);

    // 데이터베이스 연결 (try-with-resource는 자원을 자동으로 반납)
    try (Connection conn = DriverManager.getConnection(DATASOURCE_URL, DATASOURCE_USERNAME, DATASOURCE_PASSWORD)) {
        // 데이터베이스 연결 확인
        System.out.println(conn.isValid( timeout: 1) ? "데이터베이스 연결 성공" : "데이터베이스 연결 실패");

        // 데이터베이스에서 member 테이블 조회
        String selectMemberQuery = "select * from member;";
        Statement stmt = conn.createStatement();
        ResultSet selectMemberQueryResultSet = stmt.executeQuery(selectMemberQuery);
        // member 테이블의 레코드 하나씩 객체로 변환
        while (selectMemberQueryResultSet.next()) {
            Long id = selectMemberQueryResultSet.getLong( columnLabel: "id");
            String name = selectMemberQueryResultSet.getString( columnLabel: "name");
            String department = selectMemberQueryResultSet.getString( columnLabel: "department");

            User user = new User(id, name, department);
            System.out.println(user);
        }
    } catch (Exception e) {
        System.out.println("데이터베이스 연결 실패 또는 잘못된 오류 발생: " + e);
    }
}
```

JDBC 사용 (3) : 애플리케이션에서 데이터베이스 쿼리 결과 이용

78	78	사용자 78	건축공학전공
79	79	사용자 79	자율전공학부
80	80	사용자 80	소프트웨어전공
81	81	사용자 81	인공지능공학전공
82	82	사용자 82	반도체시스템전공
83	83	사용자 83	기계공학전공
84	84	사용자 84	건축공학전공
85	85	사용자 85	반도체시스템전공
86	86	사용자 86	고분자공학전공
87	87	사용자 87	신소재공학전공
88	88	사용자 88	반도체시스템전공
89	89	사용자 89	기계시스템공학전공
90	90	사용자 90	토목공학전공
91	91	사용자 91	소재디자인공학전공
92	92	사용자 92	스마트모빌리티전공
93	93	사용자 93	컴퓨터공학전공
94	94	사용자 94	화학공학전공
95	95	사용자 95	건축공학전공
96	96	사용자 96	산업공학전공
97	97	사용자 97	자율전공학부
98	98	사용자 98	경영학과
99	99	사용자 99	인공지능공학전공
100	100	사용자 100	전자공학부

데이터베이스에 저장된 레코드



애플리케이션의 출력 화면

정리

- 데이터베이스별로 제공하는 클라이언트 라이브러리 소스 코드가 다르므로 요구사항 변화에 따른 데이터베이스 변경 시 애플리케이션을 갈아엎는 수준을 요구
- 이에 따라 Java에서 데이터베이스에 접근하는 표준화된 방법이 필요하게 되어 1997년 JDBC가 탄생
- 애플리케이션 개발자는 JDBC API(인터페이스)에 정의된 메서드를 호출하여 데이터베이스 변경에도 커넥션을 맺고, 쿼리 전송 등에 대한 부분을 신경 쓰지 않고 개발 가능
    - 데이터베이스에서 JDBC API(인터페이스)를 구현한 구현체를 제공해야 함
- 애플리케이션의 JDBC Driver Manager는 정의된 데이터베이스 URL에 맞는 데이터베이스 라이브러리(구현체)를 불러옴

- JDBC는 데이터베이스와 커넥션을 맺고, 쿼리를 전송하는 부분에 대한 표준
- 쿼리 전송 시 SQL은 표준인 ANSI SQL이 있다 하더라도 표준에 없는 특수 기능을 활용하면 데이터베이스 변경 시 애플리케이션에 정의한 SQL을 수정해야 함
    - 이를 해결하기 위해 등장한 것이 MyBatis, Hibernate(구현체), JPA(인터페이스) 등
    - Hibernate와 JPA는 다음 발표에서 다룰 예정

들어 주셔서 감사합니다

E-mail	park@duck.com
LinkedIn	yeonjong-park
Instagram	yeonjong.park
GitHub	patulus

2025.10.29.

System Software Lab.