JDBC와 Spring 데이터베이스 연동 지원

JDBC 개념과 역할

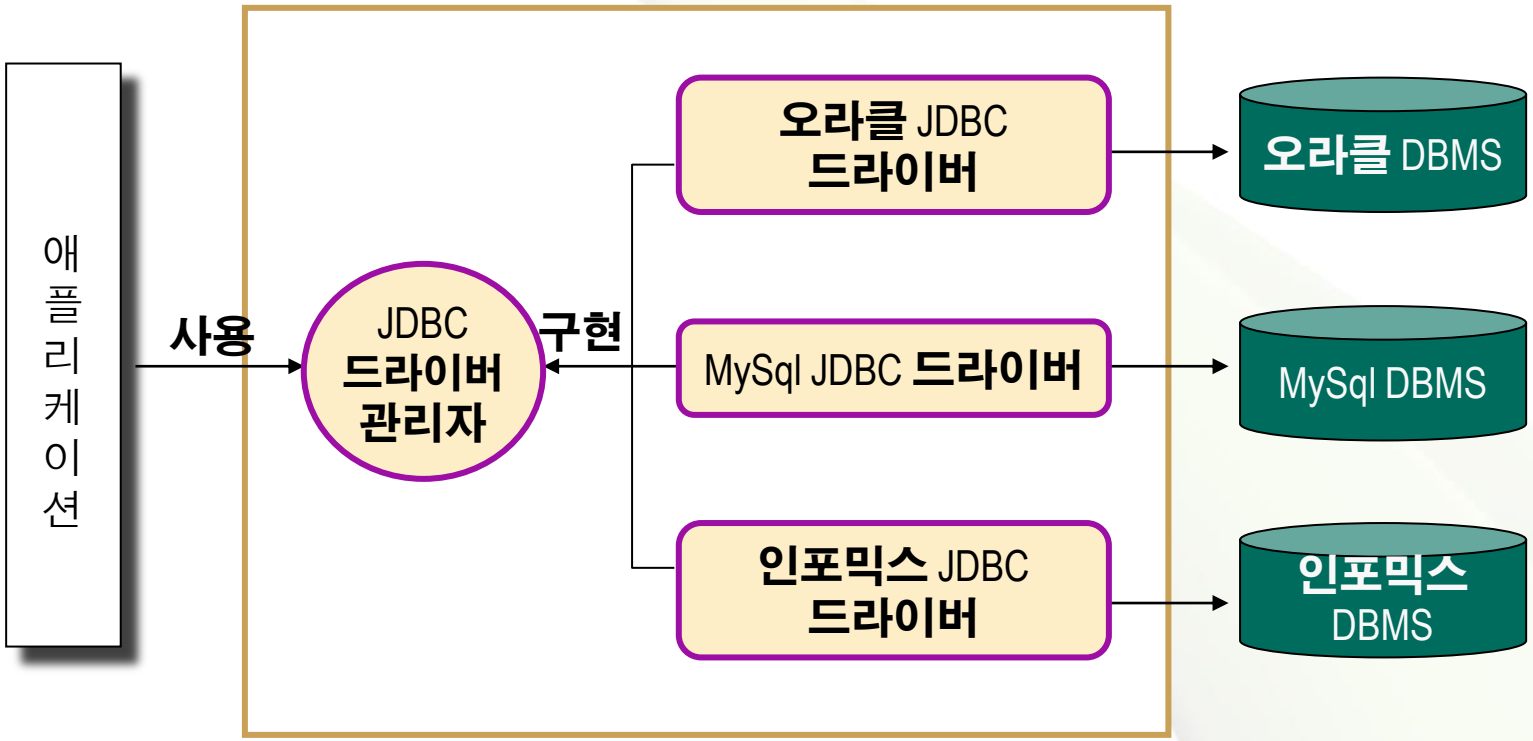
- Java Database Connectivity

- 자바애플리케이션에서 표준화된 데이터베이스 접근 제공.

- 각 데이터베이스 접속에 대한 상세한 정보를 추상화.

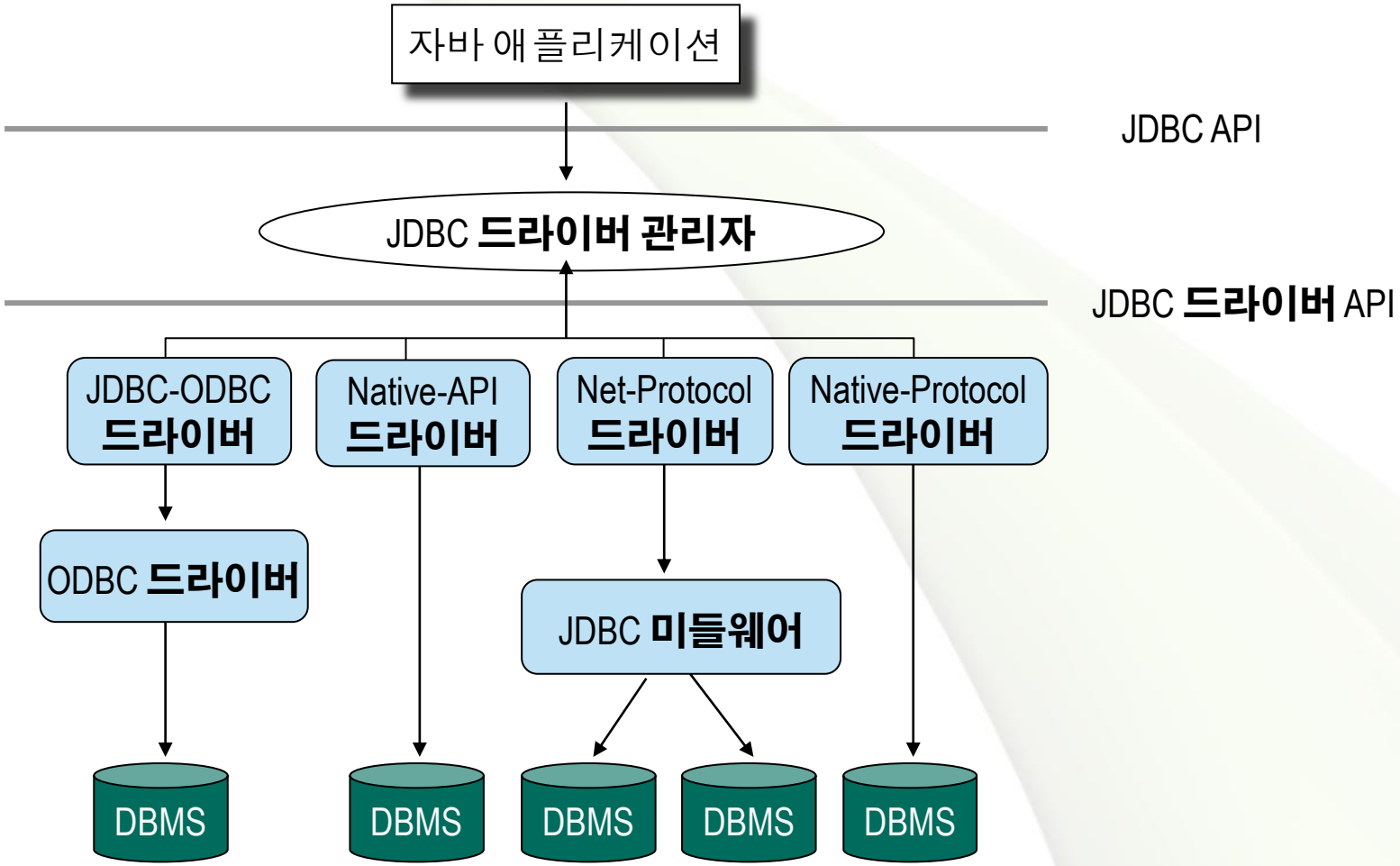
- 이론적으로는 개발된 애플리케이션에서 DB 변경시 JDBC 드라이버 교체만으로 가능

JDBC 구성


애플리케이션
사용
JDBC 드라이버 관리자
구현
오라클 JDBC 드라이버
MySql JDBC 드라이버
인포믹스 JDBC 드라이버
오라클 DBMS
MySql DBMS
인포믹스 DBMS

JDBC 드라이버 유형

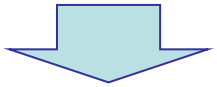
- JDBC 드라이버 구성도



JDBC 드라이버 설치

- JDBC 드라이버 선택
  - JDBC 드라이버는 사용하고자 하는 데이터베이스 벤더 별로 제공 됨

- 설치 디렉터리(다음 중 한 가지를 이용)
  - JDK설치디렉터리₩jre₩lib₩ext₩ 에 복사하는 방법.
  - 톰캣설치디렉터리₩common₩lib 폴더에 복사하는 방법
  - 이클립스 프로젝트의 WebContent₩WEB-INF₩lib 폴더에 복사하는 방법



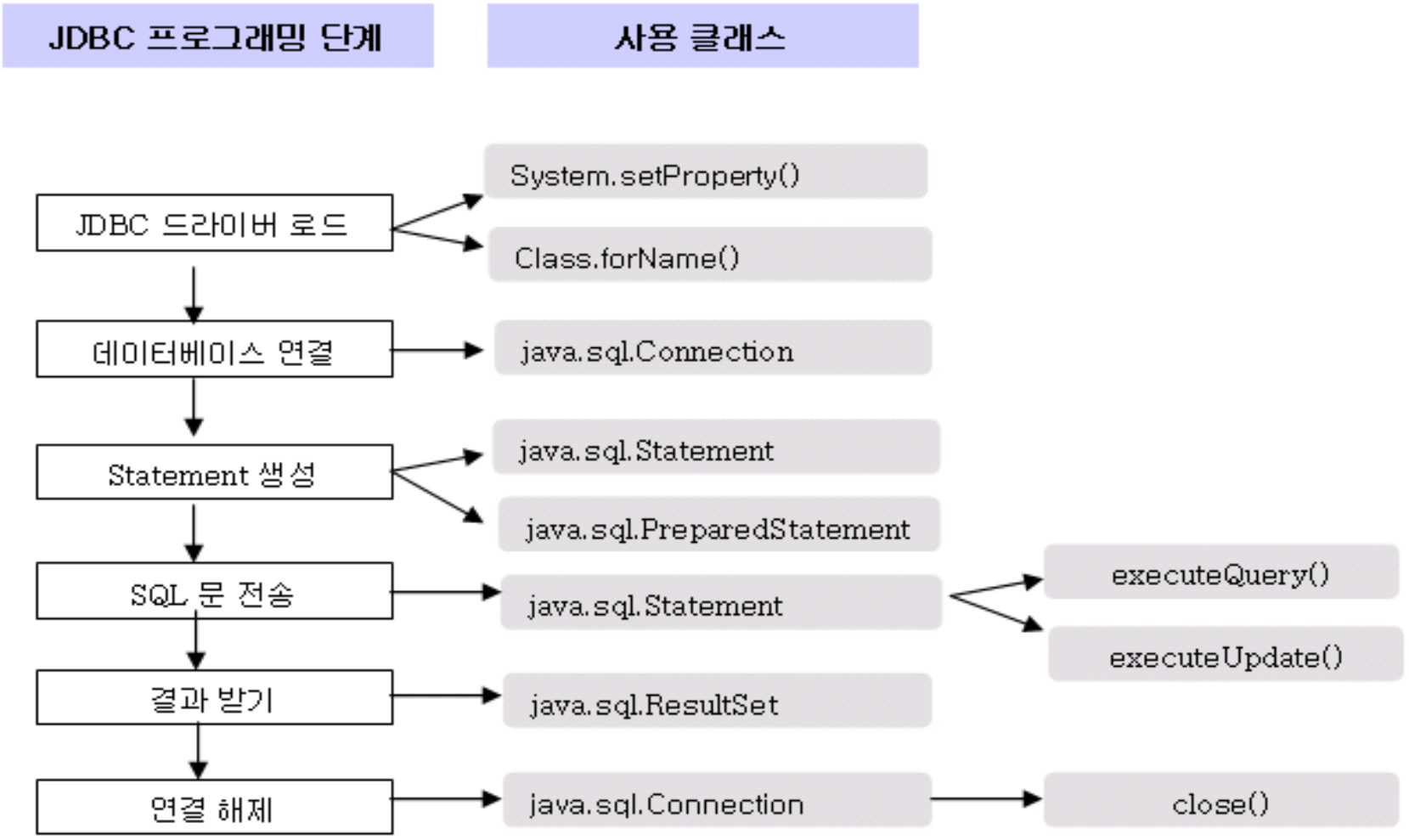
ebContent₩WEB-INF₩lib 폴더에 에 설치

  - Maven을 사용하는 경우 pom.xml 파일에 의존성 패키지 등록

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.33</version>
</dependency>
```

JDBC 프로그래밍 과정

▪JDBC 프로그래밍 단계



JDBC 프로그래밍 단계

- 데이터베이스 드라이버 로드

```
DriverManager.RegisterDriver(new com.mysql.jdbc.Driver());
```

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

- 데이터베이스 연결

```
Connection conn = DriverManger.getConnection("jdbc-url", "id", "pwd");
```

  - JDBC_URL 구성 = jdbc:mysql://ip:port/db-name

JDBC 프로그래밍 단계

- PreparedStatement 생성 및 쿼리 실행
  - PreparedStatement 객체 생성시 SQL 문장을 미리 생성하고 Parameter는 별도의 메서드로 대입하는 방식으로 성능과 관리 면에서 모두 권장 되는 방식

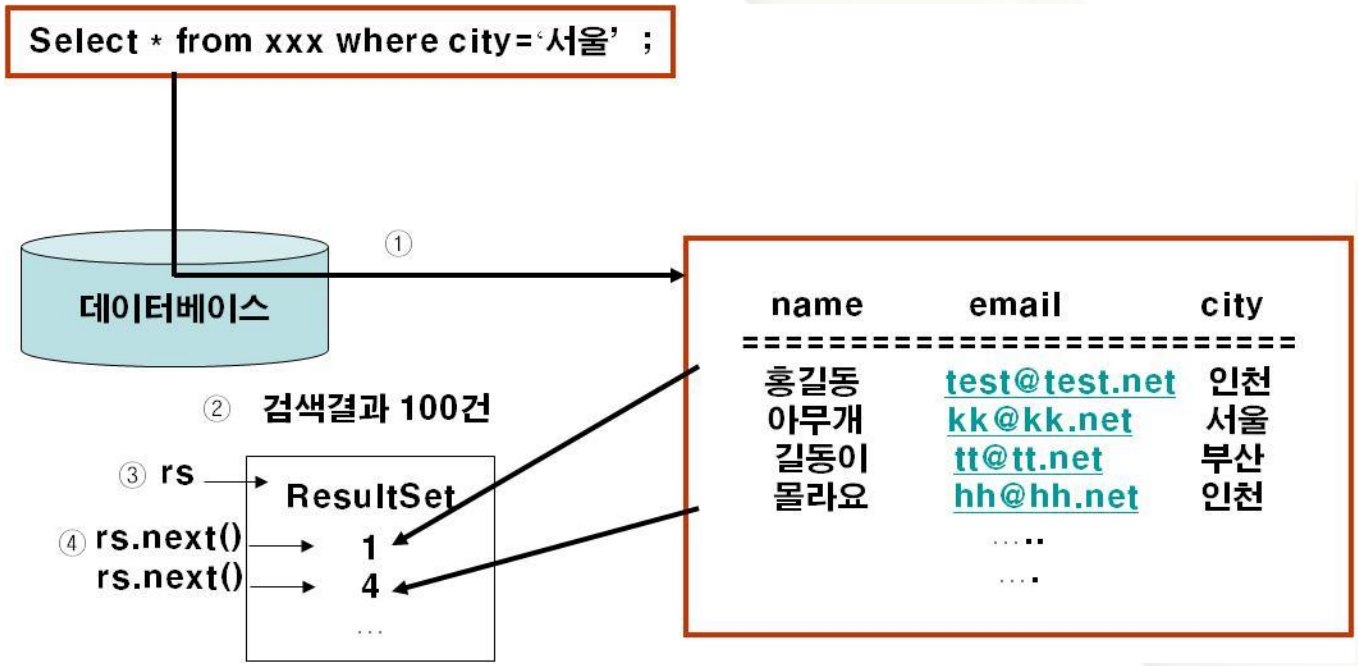
```
PreparedStatement pstmt =
        conn.prepareStatement("insert into test values(?,?) ");
pstmt.setString(1,request.getParameter("username");
pstmt..setString(2, request.getParameter("email");
pstmt.executeUpdate();
```

JDBC 프로그래밍 단계

- 결과 받기

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

  - ResultSet은 커서 개념의 연결 포인터
  - next()메서드를 통해 로우 이동



JDBC 프로그래밍 단계

- 결과 받기

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


while(rs.next()) {
   name = rs.getString(1); // or rs.getString("name");
   age = rs.getInt(2); // or rs.getInt("email");
}


rs.close();
```

JDBC 프로그래밍 단계

- 연결해제
  - Connection 을 close() 하지 않으면 사용하지 않는 연결이 유지되어 DB 자원 낭비.

```
conn.close();
```

저수준 JDBC 코드의 문제

- 구조 코드로 인한 코드량 증가
  - 예외처리 필수
  - 드라이버 등록, 연결 생성, 명령생성, 명령실행, 연결닫기 등의 표준 API 호출
  - 실제 변경되는 내용은 SQL과 전달인자 및 결과 처리 코드

- 데이터 구조 불일치로 인한 효율성 저하
  - 프로그램의 객체와 데이터베이스의 테이블 사이의 호환성 문제
    - 데이터 타입 불일치
    - 관계 불일치
    - 입자성 불일치
    - 상속성 불일치
    - 식별 불일치

Spring 데이터베이스 연동 지원

- 템플릿 클래스를 통한 데이터 접근 지원
  - 동일한 코드의 중복을 제거하고 필요한 최소한의 내용으로 데이터베이스 연동 코드 작성 가능
- 의미 있는 예외 클래스 제공
  - 데이터베이스 연동 과정 중에 발생하는 SQLException을 대체하고 오류의 원인을 예측할 수 있는 다양한 예외 클래스 제공
- 트랜잭션 처리 지원
  - 데이터베이스 연동 기술에 상과 없이 동일한 방식으로 트랙잭션 처리가 가능한 프로그래밍 기법 제공
  - 코드 기반 트랜잭션 및 선언적 트랜잭션 지원

Spring JDBC 설치

- 의존성 패키지 등록 (pom.xml)

```xml
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

Jdbc 지원 라이브러리

```xml
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jdbc</artifactId>
    <version>8.5.27</version>
</dependency>
```

커넥션 풀 라이브러리

```xml
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.45</version>
</dependency>
```

Jdbc 드라이버

Spring DataSource 설정 (연결 설정)

- Spring은 템플릿 클래스 및 ORM 프레임워크 연동 클래스를 사용할 경우 DataSource를 통해 Connection 제공

- 제공 방식
  - 커넥션 풀을 이용한 DataSource 설정
  - JNDI를 이용한 DataSource 설정
  - DriverManager를 이용한 DataSource 설정

커넥션 풀을 이용한 DataSource 설정

- 스프링이 직접 커넥션 풀 구현 클래스를 제공하지는 않지만 Apache Commons DBCP와 같은 커넥션 풀 라이브러리를 이용해서 커넥션 풀 기반의 DataSource 설정 가능
- 스프링 빈 설정 (tomcat connection pool)

```java
@Bean(destroyMethod = "close")
public DataSource dataSource() {
    DataSource ds = new DataSource();
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost/spring5fs?characterEncoding=utf8");
    ds.setUsername("spring5");
    ds.setPassword("spring5");
    ds.setInitialSize(2);
    ds.setMaxActive(10);
    ds.setTestWhileIdle(true);
    ds.setMinEvictableIdleTimeMillis(60000 * 3);
    ds.setTimeBetweenEvictionRunsMillis(10 * 1000);
    return ds;
}
```

교재 185p ~ 190p 참고

```xml
<bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource"
      destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://211.197.18.246:3306/sprinf5fs?characterEncoding=utf8" />
    <property name="username" value="spring5" />
    <property name="password" value="spring5" />
    <property name="initialSize" value="2" />
    <property name="maxActive" value="10" />
    <property name="testWhileIdle" value="true" />
    <property name="minEvictableIdleTimeMillis" value="180000" />
    <property name="timeBetweenEvictionRunsMillis" value="10000" />
</bean>
```

DataSource로 부터 커넥션 구하기

- 사용할 클래스에 필드 선언 후 의존성 주입 처리

```
public class MyClass implements MyInterface {

    @Autowired
    private DataSource dataSource;
    public setDataSource (DataSource dataSource) {
        this.dataSource = dataSource;
    }

    public void testMethod() {
        Connection conn = null;
        try {
            conn = dataSource.getConnection(); //트랜잭션 활용 불가능
            conn = DataSourceUtils.getConnection(dataSource);//트랜잭션 활용 가능
            …
        } finally {
            JdbcUtils.closeConection(conn);// 트랜잭션 사용하지 않는 경우
            DataSourceUtils.releaseConnection(conn, dataSource); //트랜잭션 사용
        }
    }
}
```

스프링 템플릿을 이용한 JDBC 지원

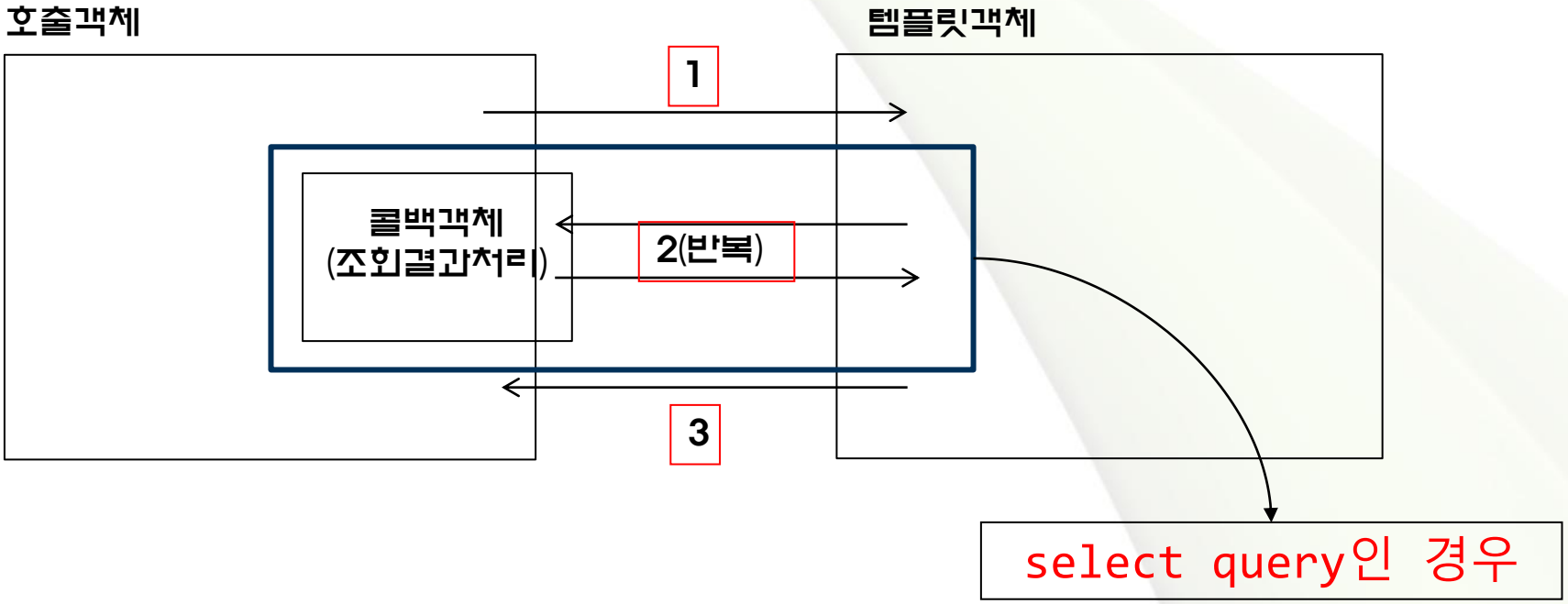
- 연결객체 획득, 예외처리 등 중복코드를 제거하고 효과적인 데이터베이스 연동 코드 작성을 위해 템플릿 지원

- 종류
  - JdbcTemplate
    - SQL 실행을 위한 다양한 메서드 제공
    - 인덱스 기반 전달인자 사용
  - NamedParameterJdbcTemplate
    - 인덱스 기반 전달인자가 아닌 이름 기반의 전달인자 사용 지원
    - 이를 위해 Map이나 SqlParameterSource 등을 사용

템플릿 구조의 작동 원리

- 반복되는 구조를 분리해서 별도의 클래스로 정의하고 변경되는 내용을 전달해서 기능을 처리하는 기법

- SQL, Parameter 매핑 데이터, 조회 결과를 처리할 객체 참조를 전달인자로 제공하면 템플릿의 구조 코드에서 이 전달인자를 사용해서 전체 데이터 연동 코드 수행



스프링 템플릿을 이용한 JDBC 지원

- JdbcTemplate 사용 1 (insert, update, delete, select)

```
public Member selectByEmail(String email) {
    List<Member> results = jdbcTemplate.query(
            "select * from MEMBER where EMAIL = ?",
            new RowMapper<Member>() {
                @Override
                public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                    Member member = new Member(
                            rs.getString("EMAIL"),
                            rs.getString("PASSWORD"),
                            rs.getString("NAME"),
                            rs.getTimestamp("REGDATE").toLocalDateTime());
                    member.setId(rs.getLong("ID"));
                    return member;
                }
            }, email);

    return results.isEmpty() ? null : results.get(0);
}

public void update(Member member) {
    jdbcTemplate.update(
            "update MEMBER set NAME = ?, PASSWORD = ? where EMAIL = ?",
            member.getName(), member.getPassword(), member.getEmail());
}
```

스프링 템플릿을 이용한 JDBC 지원

- JdbcTemplate 사용 2 (자동증가컬럼을 포함하는 insert)

```
public void insert(Member member) {
    KeyHolder keyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(new PreparedStatementCreator() {
        @Override
        public PreparedStatement createPreparedStatement(Connection con)
                throws SQLException {
            // 파라미터로 전달받은 Connection을 이용해서 PreparedStatement 생성
            PreparedStatement pstmt = con.prepareStatement(
                    "insert into MEMBER (EMAIL, PASSWORD, NAME, REGDATE) " +
                    "values (?, ?, ?, ?)",
                    new String[] { "ID" });
            // 인덱스 파라미터 값 설정
            pstmt.setString(1, member.getEmail());
            pstmt.setString(2, member.getPassword());
            pstmt.setString(3, member.getName());
            pstmt.setTimestamp(4,
                    Timestamp.valueOf(member.getRegisterDateTime()));
            // 생성한 PreparedStatement 객체 리턴
            return pstmt;
        }
    }, keyHolder);
    Number keyValue = keyHolder.getKey();
    member.setId(keyValue.longValue());
}
```