9. MyBatis: a Data Mapper Framework

Content

Introduction

- Data Mappers
  - 객체와 데이터베이스 사이에 데이터를 이동시킴(moves data between objects and a database)
  - 객체와 데이터베이스가 서로 독립적이고 mapper에 대해서도 독립적 (while keeping them independent of each other and the mapper itself)

- 특징
  - SQL에 대한 고수준 mapping을 지원하는 영속성 프레임워크(persistence framework)
  - JDBC API를 이용하는 코드들을 생략 가능(구현 불필요)
  - SQL 질의에 대한 파라미터 설정 및 결과 처리 수행
  - XML과 annotation을 이용한 mapping 설정 방법 지원
  - Java의 기본 타입 값(primitive-type value), Map, Java POJO 객체 등을 데이터베이스의 레코드와 mapping

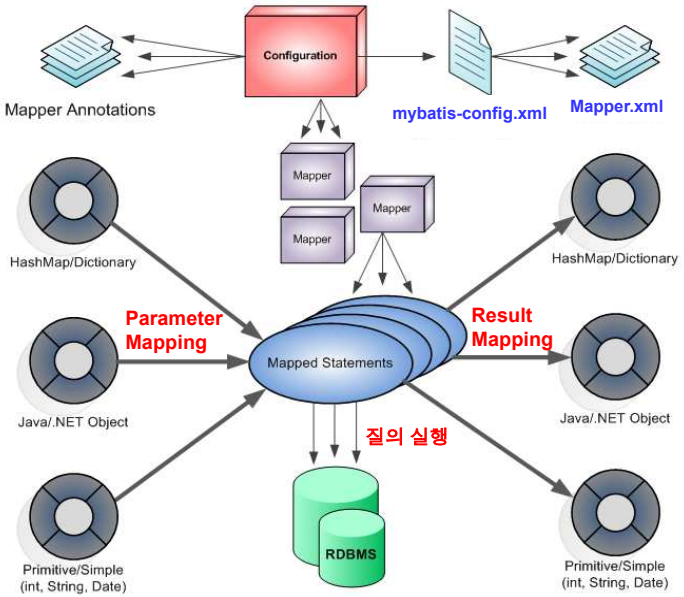
Introduction

- Installation
  - Github repository에서 직접 다운로드
    - https://github.com/mybatis/mybatis-3/releases
    - mybatis-3.5.x.zip: mybatis-3.5.x.jar 및 의존 라이브러리들을 포함
    - 프로젝트 내에 저장하고 classpath에 포함시켜 사용
  - Maven 이용
    - pom.xml

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.7</version>
</dependency>
```

  - 주의: 사용할 DBMS의 JDBC driver는 별도로 설치 필요

MyBatis Workflow



MyBatis Workflow

1. Java 객체나 값을 SQL statemen의 parameter로 전달
   - Parameter는 JavaBeans, Map, 또는 primitive(simple) data 이용
   - Parameter는 SQL 문이나 stored procedure 호출에서 runtime value로 사용됨
     - ✓ Update 문의 입력 값, Select 문의 where 조건 등

2. Mapped SQL statement 생성 및 실행
   - SQL statement에 대해 JDBC `PreparedStatement` 생성
   - Parameter 값들을 `PreparedStatement` 내에 setting
   - SQL 문이나 stored procedure를 데이터베이스에서 실행

3. 질의 실행 결과(`ResultSet`)로부터 객체 생성 및 반환
   - Select 질의: 하나 이상의 객체들을 생성 및 반환
   - Insert/Update/Delete 문: 처리된 행(레코드)의 개수 반환

MyBatis 설정

- MyBatis 기본 설정(configuration)
  - Data source(DB 접속 정보), mapper XML file, mapper interface 등에 관한 정보 포함

- Mapper 설정
  - SQL 문들을 포함하는 mapper XML 또는 mapper interface 정의
    - SQL statement 정의
    - Parameter mapping, Result mapping 정의
      - ✓ SQL 질의에 파라미터 값을 설정하고, 질의 실행 결과를 Java 객체나 Map 등으로 저장하는 규칙 정의

MyBatis 설정

- 기본 설정 예 (*mybatis-config.xml*)

```
<?xml version="1.0" encoding="UTF-8"?> …
<!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN" …>
<configuration>
    <typeAliases>                                   <!-- type에 대한 별칭 정의 -->
        <typeAlias type="model.Comment" alias="Comment"/>
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />      <!-- transaction 처리 방식 -->
            <dataSource type="POOLED">              <!-- data source (DBCP) 설정 -->
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://localhost/exampleDB" />
                <property name="username" value="dbp" />
                <property name="password" value="dbp" />
            </dataSource>
        </environment>
    </environments>
    <mappers>                       <!-- mapper XML file 또는 mapper interface 설정 -->
        <mapper resource="repository/mybatis/mapper/CommentMapper.xml" />
        <mapper class="repository.mybatis.mapper.CommentMapper2" />
    </mappers>
</configuration
```

MyBatis 설정

- Mapper XML 예

```
<mapper namespace="repository.mybatis.mapper.CommentMapper">

    <!-- SQL statement, parameter mapping, result mapping 정의 -->
    <select id="selectCommentByPrimaryKey" parameterType="long"
        resultType="Comment">
        SELECT  comment_no AS commentNo, user_id AS userId,
                comment_content AS commentContent, reg_date AS regDate
        FROM  Comments
        WHERE  comment_no = #{commentNo}
    </select>                                   ? 대신
    <insert  id="insertProduct" parameterType="model.Product">
        INSERT INTO Product (prd_id, prd_descr)
        VALUES (#{id},  #{description})
    </insert> <-- 0이나 1을 리턴
    …
</mapper>
```


SqlSessionFactory

- SqlSessionFactory
  - 데이터베이스 연동을 위한 SqlSession 객체를 생성하는 factory 객체
  - SqlSessionFactoryBuilder 객체로부터 생성됨
    - build(*InputStream is*) 메소드 이용
      - ✓ *InputStream*: MyBatis 기본설정 파일에 대한 스트림
      - ✓ MyBatis에 대한 초기화 수행(예: DataSource 생성), SqlSessionFactory 생성
  - SqlSessionFactory의 Methods

```
Configuration getConfiguration()
SqlSession openSession()                    // SqlSession 객체 생성, 반환
SqlSession openSession(boolean autoCommit)
SqlSession openSession(Connection connection)
SqlSession openSession(ExecutorType execType)
```


SqlSessionFactory

- SqlSessionFactory 객체 생성 및 사용 예

```
import java.io.InputStream;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.*;

String resource = "mybatis-config.xml";          // MyBatis 기본설정 파일
InputStream inputStream;
try {
    inputStream = Resources.getResourceAsStream(resource);
} catch (IOException e) {
    throw new IllegalArgumentException(e);
}
SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
SqlSessionFactory sqlSessionFactory = builder.build(inputStream);

SqlSession sqlSession = sqlSessionFactory.openSession();
…
```


SqlSession

- SqlSession
  - Database connection 생성(from DataSource), SQL 질의 실행, transaction 제어(commit/rollback) 등을 실행

```
<T> T selectOne(String statement)           // 질의 실행 후 하나의 결과 객체 반환
<T> T selectOne(String statement, Object parameter)   // 파라미터 객체/값 이용
<E> List<E> selectList(String statement)    // 여러 개의 결과 객체들을 반환
<E> List<E> selectList(String statement, Object parameter)
<K,V> Map<K,V> selectMap(String statement, String mapKey)
<K,V> Map<K,V> selectMap(String statement, Object parameter, String mapKey)
                                            // 질의 결과를 포함하는 Map 반환
int insert(String statement)                // 데이터 삽입
int insert(String statement, Object parameter)
int update(String statement)                // 데이터 수정
int update(String statement, Object parameter)
int delete(String statement)                // 데이터 삭제
int delete(String statement, Object parameter)

<T> T getMapper(Class<T> type)              // T 타입의 Mapper 객체 반환
```

Example: Managing Comment

- Object & Database
  - Java object (예: model.Comment)

```
public class Comment {
    private Long commentNo;
    private String userId;
    private String commentContent;
    private Date regDate;

    public Long getCommentNo() { return commentNo; }
    public void setCommentNo(Long commentNo) { this.commentNo = commentNo; }
    // … 각 property에 대한 getter/setter 정의…
}
```

  - Database table (in Oracle)

```
CREATE TABLE Comments (
   comment_no          NUMBER          NOT NULL PRIMARY KEY,
   user_id             VARCHAR2(32)    NOT NULL,
   comment_content     VARCHAR2(256)   NOT NULL,
   reg_date            DATE            NOT NULL );
```


Data Access Object 구현: JDBC 기반

- Pure JDBC DAO class

```
public class CommentJdbcRepository {
    …
    private Connection getConnection() {          // Database Connection 생성
        try {
            return DriverManager.getConnection(
                    "jdbc:mysql://localhost/exampleDB", "dbp", "dbp");
        } catch (Exception e) {
            throw new IllegalStateException(e);
        }

        /* 또는 Connection pool을 제공하는 DataSource 객체로부터 Connection 획득
         ds = new BasicDataSource();
         ds.setDriverClassName("com.mysql.jdbc.Driver");    // JDBC driver 설정
         ds.setUrl("jdbc:mysql://localhost/exampleDB");     // connection URL 설정
         ds.setUsername("dbp");                             // 사용자 인증 정보 설정
         ds.setPassword("dbp");
         return ds.getConnection();  */
    }
```


DAO 구현: JDBC 기반

- Pure JDBC DAO class (계속)

```
public Comment findCommentByCommentNo(long commentNo) {
    Connection conn = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;
    String sql = "SELECT comment_no, user_id, comment_content, reg_date " +
                 "FROM Comments  WHERE comment_no = ?");    // SQL 질의문
    try {
        conn = this.getConnection();                    // Connection 생성
        stmt = conn.prepareStatement(sql);              // PreparedStatement 생성
        stmt.setLong(1, commentNo);                     // 질의 parameter 설정
        rs = stmt.executeQuery();                       // 질의 실행
        if (rs.next()) {
            Comment comment = new Comment();            // Java 객체를 생성하여
            comment.setCommentNo(rs.getLong("comment_no"));  // 질의 결과 저장
            comment.setUserId(rs.getString("user_id"));
            comment.setContent(rs.getString("comment_content"));
            comment.setRegDate(rs.getDate("reg_date"));
            return comment;
        }
```


DAO 구현: JDBC 기반

- Pure JDBC DAO class (계속)

```
        } catch (SQLException e) {                          // 예외 처리
            e.printStackTrace();
        } finally {                                         // 자원 해제
            try { rs.close(); }
                catch (SQLException e) { e.printStackTrace(); }
            try { stmt.close(); }
                catch (SQLException e) { e.printStackTrace(); }
            try { conn.close(); }
                catch (SQLException e) { e.printStackTrace(); }
        }
        return null;
    }
```

p.15, 16 → 3줄로 바꿈

DAO 구현: JDBC 기반

- Pure JDBC DAO class (계속)

```
public int insertComment(Comment comment) {
    Connection conn = null;
    PreparedStatement stmt = null;
    String sql = "INSERT INTO comments (comment_no, user_id, "
               + "comment_content, reg_date) VALUES (?, ?, ?, ?)";
    try {
        conn = this.getConnection();
        stmt = conn.prepareStatement(sql);
        stmt.setLong(1, comment.getCommentNo());
        stmt.setString(2, comment.getUserId());
        stmt.setString(3, comment.getCommentContent());
        stmt.setDate(4, new java.sql.Date(comment.getRegDate().getTime()));
        return stmt.executeUpdate();
    } catch (Exception e) {                                  // 예외 처리
        e.printStackTrace();
    } finally {                                              // 자원 해제
        try { rs.close(); } catch (SQLException e) { e.printStackTrace(); }
        …
    }   …
```


DAO 구현: MyBatis 기반

- Mapper XML file (*CommentMapper.xml*)
  - Mapped SQL Statement 정의

```
<mapper namespace="repository.mybatis.mapper.CommentMapper">
    <cache />

    <!-- SQL statement, parameter mapping, result mapping 설정 -->
    <select id="selectCommentByPrimaryKey" parameterType="long"
            resultType="Comment">→ Comment 타입의 객체를 생성
        SELECT  comment_no AS commentNo,
                user_id AS userId,
                comment_content AS commentContent,
                reg_date AS regDate
        FROM    Comments
        WHERE   comment_no = #{commentNo}
    </select>
    …
</mapper>
```


DAO 구현: MyBatis 기반

- MyBatis-based DAO class

```
public class CommentSessionRepository {
    private String namespace = "repository.mybatis.mapper.CommentMapper";
    private SqlSessionFactory sqlSessionFactory = createSqlSessionFactory();
    private SqlSessionFactory createSqlSessionFactory() {
        …                                    // sqlSessionFactory 생성 (p.11 참조)
        return new SqlSessionFactoryBuilder().build(inputStream);
    }

    public Comment findCommentByCommentNo(long commentNo) {
        SqlSession sqlSession = sqlSessionFactory.openSession();
                                             // SqlSession 객체 생성
        try {
            // mapped statement에 parameter setting, 질의 실행, 결과 객체 생성, 반환
            return (Comment) sqlSession.selectOne(
                namespace + ".selectCommentByPrimaryKey", commentNo);
        } finally { sqlSession.close(); }    // 자원 해제
    } …
}
```

T를 타입 캐스팅해서 Comment 객체로

DAO 구현: MyBatis 기반

- Insert/Update/Delete: Mapper XML file

```
<mapper namespace="repository.mybatis.mapper.CommentMapper">
    <insert id="insertComment" parameterType="Comment">
        INSERT INTO Comments (comment_no, user_id, comment_content, reg_date)
        VALUES (#{commentNo}, #{userId}, #{commentContent}, #{regDate})
    </insert>

    <update id="updateComment" parameterType="Comment">
        UPDATE Comments
        SET  comment_content = #{commentContent}
        WHERE comment_no = #{commentNo};
    </update>

    <delete id="deleteComment" parameterType="long">
        DELETE FROM Comments
        WHERE comment_no = #{commentNo};
    </delete>
</mapper>
```

DAO 구현: MyBatis 기반

- Insert/Update/Delete: MyBatis-based DAO class

```
private String namespace = "repository.mybatis.mapper.CommentMapper";

public int insertComment(Comment comment) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        int result = sqlSession.insert(namespace + ".insertComment", comment);
        if (result > 0) { sqlSession.commit(); }
        return result;
    } finally { sqlSession.close(); }
}
```


DAO 구현: MyBatis 기반

- Insert/Update/Delete: MyBatis-based DAO class

```
public int updateComment(Comment comment) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        int result = sqlSession.update(namespace + ".updateComment",
                                       comment);
        if (result > 0) { sqlSession.commit(); }
        return result;
    } finally { sqlSession.close(); }
}
public int deleteComment(Long commentNo) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        int result = sqlSession.delete(namespace + ".deleteComment",
                                       commentNo);
        if (result > 0) { sqlSession.commit(); }
        return result;
    } finally { sqlSession.close(); }
}
```


트랜잭션 처리

- 트랜잭션 속성 설정
  - Auto-commit mode, isolation level 등 설정
  - SqlSessionFactory를 통해 SqlSession 객체 생성 시 설정 가능
    - SqlSessionFactory#**openSession**(**boolean autoCommit**)
    - SqlSessionFactory#**openSession**(**TransactionIsolationLevel level**)

- 트랜잭션 제어
  - SqlSessionFactory#**openSession**() 호출 시 새로운 트랜잭션이 시작됨
  - SqlSession#**commit**() 또는 SqlSession#**rollback**()를 통해 트랜잭션 종료
  - Spring framework 등에서 MyBatis를 사용할 경우, 일반적으로 트랜잭션을 직접 제어하지 않고 framework에게 위임함
    - Framework에 포함된 Transaction Manager 이용

트랜잭션 처리

- 트랜잭션 처리 예

```
public void replaceComment(Comment newComm, Long oldCommNo) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        // start transaction
        int result1 = sqlSession.insert(namespace + ".insertComment", newComm);
        int result2 = sqlSession.delete(namespace + ".deleteComment",
                                        oldCommNo);
        if (result1 <= 0 || result2 <= 0) {      // 위 insert 또는 update 작업 실패
            sqlSession.rollback();               // rollback transaction
        }
        sqlSession.commit();                     // commit transaction
    } finally { sqlSession.close(); }
}
```

삽입이나 삭제가 함께 성공하거나 함께 실패한다

Using Mapper Interfaces

- Interface 및 Annotation 사용
  - Mapper XML 대신 Mapper interface 정의
    - 질의 실행을 위한 메소드를 정의하고 annotation을 통해 SQL 문 설정
    - 질의 parameter 및 result mapping은 메소드의 parameter 및 return type으로 나타냄
  - DAO에서는 mapper interface의 메소드를 호출
    - SqlSession#getMapper()를 통해 mapper 객체를 구하고, mapper interface에 정의된 메소드를 호출함
      - ✓ 주의: SqlSession의 selectOne(), insert(), update(), delete() 등의 메소드는 사용하지 않음
  - 비교

Mapper XML	Mapper interface
<mapper>의 namespace	Mapper interface의 package 및 interface명
<select/insert/update/delete>의 id	Method 명
<select/…>의 parameterType	Method의 parameter type
<select/…>의 resultType	Method의 return type

Using Mapper Interfaces

- 예
  - *CommentMapper* interface

```
package repository.mybatis.mapper;
import model.Comment;
public interface CommentMapper  {
    @Select( {"SELECT comment_no, user_id, comment_content, reg_date "
                "FROM Comments  WHERE comment_no = #{commentNo}" } )
    Comment selectCommentByPrimaryKey(Long commentNo);
}
```

  - *CommentMapper* interface를 이용한 DAO 구현

```
import repository.mybatis.mapper.CommentMapper;
public class CommentMapperRepository {
    public Comment findCommentByCommentNo(Long commentNo) {
        …
        return sqlSession.getMapper(CommentMapper.class).
                selectCommentByPrimaryKey(commentNo);
            // interface에 정의된 메소드를 직접 호출
            // → Comment type 객체가 반환되므로 명시적 type casting 필요 없음
    } …
```


Using Mapper Interfaces

  - Mapper XML 방식과 비교 (p.18~19)

```
<mapper namespace="repository.mybatis.mapper.CommentMapper">
    <select id="selectCommentByPrimaryKey" parameterType="long"
        resultType="Comment">
        SELECT comment_no, user_id, comment_content, reg_date
        FROM  Comments
        WHERE  comment_no = #{commentNo}
    </select>  …
</mapper>
```

```
public class CommentSessionRepository {
  private String namespace="repository.mybatis.mapper.CommentMapper";
  public Comment findCommentByCommentNo(Long commentNo) {
        …
        return (Comment)sqlSession.selectOne(
                namespace + ".selectCommentByPrimaryKey", commentNo);
  } …
}
```


Mapper XML 및 Interface 활용

- Mapper XML과 Mapper Interface를 함께 사용
  - SQL statement는 Mapper XML file에서만 정의
    - Mapper interface에서는 생략
  - 주의: 서로 match될 수 있도록 Mapper XML과 Mapper interface의 대응되는 이름들을 서로 일치시켜야 함

Mapper XML	Mapper interface
<mapper>의 namespace	Mapper interface의 package 및 interface명
<select/insert/update/delete>의 id	Method 명
<select/…>의 parameterType	Method의 parameter type
<select/…>의 resultType	Method의 return type

  - DAO에서는 Mapper interface에 정의된 메소드를 호출
  - Mapper XML과 Mapper Interface 두 방식의 장점을 모두 활용

Mapper XML 및 Interface 활용

- 예
  - Mapper XML

```
<mapper namespace="repository.mybatis.mapper.CommentMapper">
    <select id="selectCommentByPrimaryKey" parameterType="long"
        resultType="Comment">
        SELECT comment_no, user_id, comment_content, reg_date
        FROM  Comments
        WHERE  comment_no = #{commentNo}          // XML에서 SQL 문 정의
    </select>  …
</mapper>
```

  - Mapper interface

```
package repository.mybatis.mapper;
import model.Comment;
public interface CommentMapper  {      // annotation을 통한 SQL문 정의는 생략
    Comment selectCommentByPrimaryKey(Long commentNo);
}
```


Mapper XML 및 Interface 활용

- Mapper 정의 방식의 비교

정의 방식	장점	단점
Mapper XML	- mapper의 모든 기능 사용 가능 - iBATIS와 호환	- SQL statement의 id를 문자열로 정의 및 참조해야 하므로 runtime error 발생 가능성이 있음 - 범용 API를 사용하므로 type 변환 필요
Mapper interface	- Interface 및 메소드 호출을 사용하므로 runtime error 가능성 적음 - Type 변환 필요 없음	- Annotation 특성상 동적 SQL 작성, 1:N 관계 mapping 등이 어렵거나 불가능함 - Result mapping에 제약이 있음
Mapper XML과 Mapper interface 병용	위 장점들을 모두 활용 가능	Mapper XML과 interface를 모두 정의해야 하므로 code 작성량 증가

Mapper XML file

- Elements
  - cache – Configuration of the cache for a given namespace
  - cache-ref – Reference to a cache configuration from another namespace
  - sql – A reusable chunk of SQL that can be referenced by other statements
  - select – A mapped SELECT statement
  - insert – A mapped INSERT statement
  - update – A mapped UPDATE statement
  - delete – A mapped DELETE statement
  - resultMap – The most complicated and powerful element that describes how to load your objects from the database result sets

Mapped Statements

- <select>

```
<select
    id="selectUser"                          ← 이 SQL문을 참조하기 위한 식별자
    parameterType="int"                      ← SQL문에 필요한 parameter의 타입(optional)
    resultType="hashmap"                     ← SQL문 실행 후 반환될 결과의 타입
    resultMap="userResultMap"                ← 실행 결과를 정의하는 <resultMap>의 id 값
    flushCache="false"                       ← SQL문 실행 후 cache를 flush할지 여부
    useCache="true"                          ← 실행 결과를 2nd level cache에 저장할지 여부
    timeout="10000"                          ← DB에서 결과가 반환될 때까지 대기하는 시간(초)
    fetchSize="256"                          ← 한 번에 가져올 결과 행의 개수
    statementType="PREPARED"                 ← 사용할 Statement 객체의 종류
    resultSetType="FORWARD_ONLY">            ← 사용할 result set type
    SELECT * FROM USERS                      ← SQL문
    WHERE ID = #{id}
</select>
```

둘 중 하나만 사용

Mapped Statements

◆ <select>의 주요 속성

id	A unique identifier in this namespace that can be used to reference this statement.
parameterType	The fully qualified class name or alias for the parameter that will be passed into this statement. **This attribute is optional** because MyBatis can calculate the TypeHandler to use out of the actual parameter passed to the statement. Default is unset.
resultType	The fully qualified class name or alias for the expected type that will be returned from this statement. Note that in the case of collections, this should be the type that the collection contains, not the type of the collection itself. Use **resultType OR resultMap, not both.**
resultMap	A named reference to an **external resultMap**. Result maps are the most powerful feature of MyBatis, and with a good understanding of them, many difficult mapping cases can be solved. Use resultMap OR resultType, not both.
statementType	Any one of **STATEMENT, PREPARED or CALLABLE**. This causes MyBatis to use **Statement**, **PreparedStatement** or **CallableStatement** respectively. **Default: PREPARED.**

Mapped Statements

◆ SQL Statement 정의

- Database와 JDBC driver에 대해 유효한 모든 SQL 문 사용 가능
- 주의사항
  - 하나의 설정 파일에서 XML과 SQL을 동시에 사용하므로 **<**, **>**와 같은 특수 문자 사용에 주의
  - XML CDATA Section 이용
- 예:

```
<select  id="getUsersByAge" parameterType="int" resultType="User">
    <![CDATA[                ← CDATA Section 내부는 parsing 되지 않음
        SELECT *
        FROM USERS
        WHERE AGE > #{value}
    ]]>                      CDATA안 쓸거면 &lt 사용
</select>
```


Mapped Statements

◆ <insert>, <update>, <delete>

```
<insert (or update or delete)>
    id="insertCommunity" parameterType="Community"
    flushCache="true"              ← DML문의 경우 실행 후 cache를 flush함(default)
    statementType="PREPARED"
    timeout="20"
    keyProperty="" keyColumn="" useGeneratedKeys=""> (insert/update only)
     autoGenerate값
    INSERT INTO Community (cId, cName, descr, startDate)        ← insert 문
    VALUES (#{id}, #{name}, #{description}, SYSDATE)

    <!-- UPDATE Community                                       ← update 문
         SET cName = #{name}, descr = #{description},
        WHERE cId = #{id}  -->

    <!-- DELETE FROM Community WHERE cId = #{id} -->            ← delete 문
</insert (or update or delete)>
```


Mapped Statements

◆ <insert>의 Auto-Generated Key 관련 속성

useGeneratedKeys	This tells MyBatis to use the JDBC *getGeneratedKeys* method to retrieve keys generated internally by the database (e.g. auto increment fields in RDBMS like MySQL or SQL Server). Default: false
keyProperty	Identifies a property into which MyBatis will set the key value returned by *getGeneratedKeys*, or by a *selectKey* child element of the *insert* statement. Default: unset. Can be a comma separated list of property names if multiple generated columns are expected.
keyColumn	Sets the name of the column in the table with a generated key. This is only required in certain databases (like PostgreSQL) when the key column is not the first column in the table. Can be a comma separated list of columns names if multiple generated columns are expected.

Mapped Statements

◆ 참고: JDBC getGeneratedKeys() 사용 예

```
String insertQuery = "INSERT INTO Community (…) VALUES (?, ?, ?, …);
                                          // PK 값은 자동으로 생성된다고 가정
String key[]={"cId"};              // PK column들의 배열
PreparedStatement pstmt = conn.prepareStatement(insertQuery, key);
// query parameter 설정 …
pstmt.executeUpdate();                    //  insert 실행

ResultSet rs = pstmt.getGeneratedKeys();
If (rs.next()) {
   int commId = rs.getInt(1);             //  생성된 PK 값 구함
   community.setId(commId);               //  필요 시 객체에 저장해서 반환
   return commId;   // 또는 return community;
}
```


Mapped Statements

◆ Auto-Generated Key 사용 예

– MySQL auto_increment / SQL Server IDENTITY column 사용 시

```
<insert id="insertCommunity" parameterType="Community"
        useGeneratedKeys="true" keyProperty="id">
   INSERT INTO Community (cName, descr, startDate)    <!-- cId 컬럼은 생략 -->
   VALUES (#{name}, #{description}, SYSDATE)          <!-- #{id}는 생략 -->
</insert>
```

– Oracle Sequence 사용 시

```
<insert id="insertCommunity" parameterType="Community">
   <selectKey keyProperty="id" resultType="int" order="BEFORE">
      SELECT commId_seq.nextval AS id FROM DUAL
   </selectKey>
   INSERT INTO Community (cId, cName, descr, startDate)
   VALUES (#{id}, #{name}, #{description}, SYSDATE)
</insert>
```

← 아래의 insert 문 보다 먼저 실행됨

– 생성된 키는 테이블의 PK 컬럼(cId)과 parameter 객체(Community)의 해당 property(id)에 저장됨

Mapped Statements

◆ Reusing SQL Fragments

– <sql> 및 <include>를 이용하여 SQL 문의 일부를 재사용 가능

– 예

```
<sql  id="selectProd_fragment">
   FROM Product
   WHERE prd_cat_id = #{value}
</sql>

<select  id="selectProdCount" resultType="int">
   SELECT COUNT(*) AS total
   <include refid="selectProd_fragment"/>
</select>
```

⇨

```
SELECT COUNT(*) AS total
FROM Product
WHERE prd_cat_id = #{value}
```

```
<select  id="selectProducts" resultType="Product">
   SELECT prd_id, prd_name
   <include refid="selectProd_fragment"/>
</select>
```

⇨

```
SELECT prd_id, prd_name
FROM Product
WHERE prd_cat_id = #{value}
```


Parameter Mapping

◆ parameterType 속성

```
<select id="selectUsers" parameterType="int" resultType="User">
    select ID, USERNAME, PASSWORD
    from USERS
    where ID = #{id}
</select>
<insert  id="insertProduct" parameterType="Product">
    insert into PRODUCT (PRD_ID, PRD_DESCR)
    values (#{id},  #{description})
</insert>
<insert  id="insertProduct" parameterType="Product">
    insert into PRODUCT (PRD_ID, PRD_DESCR)
    values (#{id, javaType=int, jdbcType=NUMERIC, numericScale=2},
            #{description, javaType=String, jdbcType=VARCHAR})
</insert>
```

value값이 여러 개일 때

– 주의: Null 값이 전달될 수 있는 nullable 컬럼에 대해서는 반드시 jdbcType 속성을 명시해야 함 (PreparedStatement#setNull(..., int sqlType) 에서 필요)

Parameter Mapping

- Map-type Parameters
  - JavaBeans 객체를 사용하지 않고 여러 개의 parameter 값들을 전달해야 할 때 parameter 객체로 java.util.Map 객체 사용 가능
  - 예

```
<select  id="getProduct" parameterType="java.util.Map" … >  또는 "hashmap"사용
        select * from PRODUCT
        where PRD_CAT_ID = #{catId} and PRD_CODE = #{code}
</select>
```

  - 위 SQL statement 호출 시 "catId"와 "code"를 key 값으로 갖는 entry들을 포함한 Map 객체를 전달해야 함

```
Map<String, String> param = new HashMap<String, String>(2);
param.put("catId", "FISH");
param.put("code", "FI-SW-01");
Product prod = (Product)sqlSession.selectOne("getProduct", param);
          // 또는 sqlSession.getMapper(productMapper.class).getProduct(param);
```


Parameter Mapping

- 여러 개의 parameter 전달
  - 각 parameter에 대한 참조는 "param" 키워드와 위치를 나타내는 숫자를 이용
  - 예: #{param1}, #{param2}

```
// in a DAO class
public Account getAccount(String username, String password)
            throws DataAccessException {
    return accountMapper.getAccountByUsernameAndPassword(username, password);
}
```

```
<!-- in a Mapper XML for AccountMapper -->
<select id="getAccountByUsernameAndPassword" resultType="Account">
   SELECT USERNAME, …
   FROM    ACCOUNT, PROFILE, SIGNON, BANNERDATA
   WHERE  ACCOUNT.USERID = #{param1}
       AND SIGNON.PASSWORD = #{param2} …
</select>
```


Parameter Mapping

- 여러 개의 parameter 전달 시 이름 지정 방법
  - 메소드 호출 시 @Param("*paramName*") annotation 사용

```
import org.apache.ibatis.annotations.Param;
// a mapper interface
public interface AccountMapper {
    Account getAccountByUsernameAndPassword(
                    @Param("uname")  String username,
                    @Param("passwd")  String password);
    …
}
```

```
<!-- in a Mapper XML for AccountMapper -->
<select id="getAccountByUsernameAndPassword" resultType="Account">
   SELECT USERNAME, …
   FROM    ACCOUNT, PROFILE, SIGNON, BANNERDATA
   WHERE  ACCOUNT.USERID = #{uname}
       AND SIGNON.PASSWORD = #{passwd} …
</select>
```


Result Mapping

- resultType 속성
  - JavaBeans 객체의 property 이름들이 질의 결과로 반환되는 column명이나 alias명들과 일치할 경우 사용 가능

```
<select id="selectComment1" parameterType="long" resultType="Comment">
    SELECT  commentNo, userId, commentContent, regDate
    FROM    Comments2
    WHERE  commentNo = #{cNo}
</select>

<select id="selectComment2" parameterType="long" resultType="Comment">
    SELECT  comment_no AS commentNo,  user_id AS userId,
            comment_content AS commentContent,  reg_date AS regDate
    FROM    Comments
    WHERE  commentNo = #{cNo}
</select>
```

Comment 객체의 property명과 일치

Result Mapping

◆ <resultMap>

– 테이블의 column명과 JavaBeans 객체의 property 이름이 일치하지 않을 경우 별도의 result mapping 정의 필요

```
<resultMap id="baseResultMap" type="Comment">
    <id column="comment_no" jdbcType="NUMERIC" property="commentNo" />
    <result column="user_id" jdbcType="VARCHAR" property="userId" />
    <result column="comment_content" jdbcType="VARCHAR"
        property="commentContent" />
    <result column="reg_date" jdbcType="TIMESTAMP" property="regDate" />
</resultMap>

<select  id="selectComment" parameterType="long" resultMap="baseResultMap">
    SELECT  comment_no, user_id, comment_content, reg_date
    FROM    Comments
    WHERE  commentNo = #{commentNo}
</select>
```

column타입

Result Mapping

◆ <resultMap>의 하위 elements

– **id** – an ID result; flagging results as ID will help improve overall performance

– **result** – a normal result injected into a field or JavaBean property

– constructor - used for injecting results into the constructor of a class upon instantiation

– **association** – a complex (user-defined) type association; many results will roll up into this type (N : 1 관계)

– **collection** – a collection of complex types (1 : N 관계)

– discriminator – uses a result value to determine which resultMap to use

Result Mapping

◆ <id> & <result>의 속성

Attribute	Description
property	**The field or property to map the column result to**. If a matching JavaBeans property exists for the given name, then that will be used. Otherwise, MyBatis will look for a field of the given name. In both cases you can use complex property navigation using the usual dot notation. For example, you can map to something simple like: username, or to something more complicated like: address.street.number.
column	**The column name from the database, or the aliased column label**. This is the same string that would normally be passed to resultSet.getString(columnName).
javaType	**A fully qualified Java class name, or a type alias**. MyBatis can usually figure out the type if you're mapping to a JavaBean. However, if you are mapping to a HashMap, then you should specify the javaType explicitly to ensure the desired behaviour.
jdbcType	**The JDBC Type from the list of supported types**. The JDBC type is only required for nullable columns upon insert, update or delete. This is a JDBC requirement, not a MyBatis one. So even if you were coding JDBC directly, you'd need to specify this type – but only for nullable values.

Result Mapping

◆ Supported JDBC Types

BIT	FLOAT	CHAR	TIMESTAMP	OTHER	UNDEFINED
TINYINT	REAL	VARCHAR	BINARY	BLOG	NVARCHAR
SMALLINT	DOUBLE	LONGVARCHAR	VARBINARY	CLOB	NCHAR
INTEGER	NUMERIC	DATE	LONGVARBINARY	BOOLEAN	NCLOB
BIGINT	DECIMAL	TIME	NULL	CURSOR	ARRAY

Result Mapping

- 생성자를 통한 객체 생성 (<constructor>)
  - 생성자만 존재하고 setter method가 없는 immutable object에 대해 적용

```
public class ImmutableComment implements Serializable {
    private final Long commentNo;
    private final String userId;
    private final Date regDate;
    private final String commentContent;

    public ImmutableComment(Long commentNo, String userId, Date regDate,
            String commentContent) {
        this.commentNo = commentNo;
        this.userId = userId;
        this.regDate = regDate;
        this.commentContent = commentContent;
    }

    public Long getCommentNo() { return commentNo; } …    // no setter methods
}
```


Result Mapping

- 생성자를 통한 객체 생성 (<constructor>)

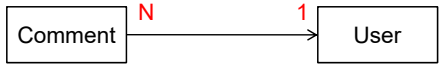
ImmutableComment

```
<resultMap id="constructorResultMap" type="Comment">
    <constructor>
        <idArg column="comment_no" javaType="long" />
        <arg column="user_id" javaType="string" />
        <arg column="reg_date" javaType="date" />
        <arg column="comment_content" javaType="string" />
    </constructor>
</resultMap>
```

  - <idArg> – ID argument; flagging results as ID will help improve overall performance
  - <arg> – a normal result injected into the constructor
  - element들의 순서는 생성자의 argument들의 순서와 일치해야 함

Result Mapping

- Mapping Complex Types Properties
  - Database 내의 두 table의 관계가 1:1 또는 N:1 관계인 경우
  - "many" side class가 "one" side class type의 property를 소유(객체 참조)
  - 예: Comment → User (comment 작성자)




```
// "many" side class
public class Comment {
    private Long commentNo;
    private String userId;
    private String commentContent;
    private Date regDate;
    private User  user;
    …
}
```

참조

```
// "one" side class
public class User {
    private String userId;
    private String userName;
    …
}
```


Result Mapping

- 예: Database Schema

COMMENTS table

```
COMMENT_NO      NUMBER NOT NULL PRIMARY KEY,
COMMENT_CONTENT VARCHAR2(256)  NOT NULL,
REG_DATE        DATE NOT NULL,
…USER_ID        VARCHAR2(32) NOT NULL
                FOREIGN KEY USERS (USER_ID),
```

USERS table

```
USER_ID         VARCHAR2(32) NOT NULL PRIMARY KEY,
USER_NAME       VARCHAR2(32) NOT NULL,
…
```

Result Mapping

– Join 질의 및 <association>을 이용한 mapping 설정

```
<resultMap id="associationResultMap" type="Comment">
    <id column="comment_no" jdbcType="NUMERIC" property="commentNo" />
    <result column="user_id" jdbcType="VARCHAR" property="userId" />
    <result column="comment_content" jdbcType="VARCHAR" property="commentContent" />
    <result column="reg_date" jdbcType="TIMESTAMP" property="regDate" />
    <association column="user_id" property="user" javaType="User" >
        <id column="user_id" property="userId" />
        <result column="user_name" property="userName" />
    </association>
</resultMap>

<select id="selectCommentByPrimaryKeyAssociation" parameterType="long"
        resultMap="associationResultMap">
    SELECT c.comment_no, c.user_id, c.comment_content, c.reg_date, u.user_name
    FROM   Comments c, Users u
    WHERE  c.user_id = u.user_id  AND c.comment_no = #{commentNo}
</select>
```

Comment 객체의 user 필드

foreign key column

Result Mapping

– 참조 객체에 대한 member 이름과 property명을 이용한 설정 방법

```
<select id="selectCommentByPrimaryKeyAssociation" parameterType="long"
        resultType="Comment">
    SELECT   c.comment_no AS commentNo,
             c.user_id AS userId,
             c.comment_content AS commentContent,
             c.reg_date AS regDate,
             u.user_id AS "user.userId",
             u.user_name AS "user.userName"
    FROM     Comments c, Users u
    WHERE    c.user_id = u.user_id  AND c.comment_no = #{commentNo}
</select>
```

user: Comment 클래스의 user 필드

```
public class Comment {
    private Long commentNo;
    private String userId;
    private String commentContent;
    private Date regDate;
    private User  user;
    … }
```

참조

```
public class User {
    private String userId;
    private String userName;
    …

}
```


Result Mapping

– DAO 구현

```
String namespace="repository.mybatis.mapper.CommentMapper";
public Comment findCommentAndUserByCommentNo(long commentNo) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        return (Comment) sqlSession.selectOne(
                namespace + ".selectCommentByPrimaryKeyAssociation",
                commentNo);
    } finally { sqlSession.close(); }
}
```

– Mapper interface를 이용한 DAO 구현

```
SqlSession sqlSession = sqlSessionFactory.openSession();
try {
    return sqlSession.getMapper(CommentMapper.class).
            selectCommentByPrimaryKeyAssociation(commentNo);
} finally { sqlSession.close(); }
```

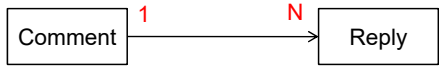

Result Mapping

– 참고: JDBC API를 이용한 구현

```
String sql = "SELECT c.comment_no, c.user_id, c.comment_content,
                     c.reg_date, u.user_name
              FROM   Comments c, Users u
              WHERE  c.user_id = u.user_id  AND c.comment_no = ?";
conn = this.getConnection();                    // Connection 생성
stmt = conn.prepareStatement(sql);              // PreparedStatement 생성
stmt.setLong(1, commentNo);                     // 질의 parameter 설정
rs = stmt.executeQuery();                       // 질의 실행
if (rs.next()) {
    Comment comment = new Comment();            // Comment 객체 생성
    comment.setCommentNo(rs.getLong("comment_no")); // Comment 정보 저장
    comment.setUserId(rs.getString("user_id"));
    …
    User user = new User();                     // User 객체 생성
    user.setUserName(rs.getString("user_name")); // user_name 저장
    comment.user = user;                        // Comment 객체에서 User 객체 참조
    return comment;                             // Comment 객체 반환
}
```

Result Mapping

- Mapping Complex Collection Properties
  - Database 내의 두 table의 관계가 1:N 또는 M:N 관계인 경우
  - "one" side class가 "many" side class 객체들을 포함하는 Collection type property를 소유(객체 참조)
  - 예: Comment → Reply



참조되는 쪽이 Many

```
// "one" side class
public class Comment {
        private Long commentNo;
        private String userId;
        private String commentContent;
        private Date regDate;
        private User  user;
        private List<Reply> replies;
                // Reply 객체들의 list 참조
}
```

```
// "many" side class
public class Reply {
        private Long replyId;
        private Long commentNo;
        private String userId;
        private String replyContent;
        private Date regDate;
}
```


Result Mapping

- 예: Database Schema

COMMENTS table

```
COMMENT_NO      NUMBER NOT NULL PRIMARY KEY,
USER_ID         VARCHAR2(32) NOT NULL,
COMMENT_CONTENT VARCHAR2(256)  NOT NULL,
REG_DATE        DATE NOT NULL,
…
```

REPLY table

```
REPLY_ID        NUMBER NOT NULL PRIMARY KEY,
COMMENT_NO      NUMBER NOT NULL
                FOREIGN KEY  COMMENTS (COMMENT_NO),
USER_ID         VARCHAR2(32) NOT NULL,
REPLY_CONTENT   VARCHAR2(256) NOT NULL,
REG_DATE        DATE NOT NULL,
…
```


Result Mapping

- Join 질의 및 <collection>을 이용한 mapping 설정

```
<resultMap id="collectionResultMap" type="Comment">
    <id column="comment_no" jdbcType="BIGINT" property="commentNo" />
    <result column="user_id" jdbcType="VARCHAR" property="userId" />
    <result column="comment_content" jdbcType="VARCHAR" property="commentContent" />
    <result column="reg_date" jdbcType="TIMESTAMP" property="regDate" />
    <collection property="replies" ofType="Reply">   ← replies: Comment 객체의 replies 필드
        <id  column="reply_id" property="replyId" />
        <result  column="reply_user_id" property="userId" />
        <result  column="reply_content" property="replyContent" />
        <result  column="reply_date" property="regDate" />
    </collection>
</resultMap>
<select id="selectCommentByPrimaryKeyCollection" parameterType="long"
        resultMap="collectionResultMap">
    SELECT   c.comment_no,  c.user_id,  c.comment_content,  c.reg_date,
             r.reply_id, r.user_id AS reply_user_id, r.reply_content, r.reg_date AS reply_date
    FROM     Comments c, Reply r
    WHERE    c.comment_no = r.comment_no AND c.comment_no = #{commentNo}
</select>
```


Result Mapping

- DAO 구현

```
String namespace="repository.mybatis.mapper.CommentMapper";
public Comment findCommentAndRepliesByCommentNo(long commentNo) {
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        return (Comment) sqlSession.selectOne(
                namespace + ".selectCommentByPrimaryKeyCollection",
                commentNo);
    } finally { sqlSession.close(); }
}
```

- Mapper interface를 이용한 DAO 구현

```
    SqlSession sqlSession = sqlSessionFactory.openSession();
    try {
        return sqlSession.getMapper(CommentMapper.class).
                selectCommentByPrimaryKeyCollection(commentNo);
    } finally { sqlSession.close(); }
```

– 참고: JDBC API를 이용한 구현

```
String sql = "SELECT  c.comment_no,  c.user_id,  c.comment_content,  c.reg_date,
      r.reply_id,  r.user_id AS reply_user_id,  r.reply_content, r.reg_date AS reply_date
            FROM  Comments c, Reply r
            WHERE  c.comment_no = r.comment_no AND c.comment_no =?";
…
rs = stmt.executeQuery();                          // 질의 실행
Comment comment = null;                            // Comment 참조 변수 선언
List<Reply> replyList = null;                      // List<Reply> 참조 변수 선언
while (rs.next()) {
    if (comment == null) {
        comment = new Comment();                   // Comment 객체 생성
        comment.setCommentNo(rs.getLong("comment_no"));
        …                                          // Comment 정보 저장
        replyList = new ArrayList<Reply>();        // List<Reply> 객체 생성
    }
    Reply reply = new Reply();                     // Reply 객체 생성
    reply.setReplyId(rs.getString("reply_id"));
    …                                              // Reply 정보 저장
    replyList.add(reply);
}
if (comment != null) {
    comment.replies = replyList;          // Comment 객체에서 List<Reply> 객체 참조
}                  setReplies(replyList);
return comment;                           // Comment 객체 반환
```


Dynamic Mapped Statements

- Dynamic SQL
  - Run-time에 조건에 따라 SQL Statement를 생성하기 위한 방법
  - Elements
    - if
    - choose (when, otherwise)
    - trim (where, set)
    - foreach

Dynamic Mapped Statements

– if

```
<select id="selectCommentByConditionIf" parameterType="hashmap"
    resultType="Comment">
    SELECT
        comment_no,
        user_id,
        comment_content,
        reg_date
    FROM  Comment
    <where>                                    ← WHERE 절 생성
        <if test="commentNo != null">
            comment_no = #{commentNo}
        </if>
        <if test="user != null and user.userId != null">
            AND user_id = #{user.userId}
        </if>
    </where>
</select>
```


Dynamic Mapped Statements

– choose, when, otherwise

```
<select id="selectCommentByConditionChoose" parameterType="hashmap"
    resultType="Comment">
    SELECT comment_no, user_id, comment_content, reg_date
    FROM  Comment
    <choose>
        <when test="commentNo != null">
            WHERE comment_no = #{commentNo}
        </when>
        <when test="user != null and user.userId != null">
            WHERE user_id = #{user.userId}
        </when>
        <otherwise>
            WHERE comment_no = 1 AND user_id = 'user1'
        </otherwise>
    </choose>
</select>
```

Dynamic Mapped Statements

- trim, foreach

```
<select id="selectCommentByConditionForeach" parameterType="hashmap"
        resultType="Comment">
        SELECT  comment_no,
                user_id,
                comment_content,
                reg_date
        FROM  Comment
        <where>
                <if test="commentNos != null">
                        <foreach collection="commentNos" item="commNo" index="index"
                                open="comment_no IN (" close=")" separator=",">
                                #{commNo}
                        </foreach>
                </if>
        </where>
</select>
```

List, Set, Map, Array 등 가능

← commentNos의 값들의 리스트 생성(예: (10, 20, 30))

References

- MyBatis Project
  - Blog: http://blog.mybatis.org/
    - Products: http://blog.mybatis.org/p/products.html
  - Github: https://github.com/mybatis/mybatis-3
- Reference Documentation
  - http://www.mybatis.org/mybatis-3/
    - 한글 버전: http://www.mybatis.org/mybatis-3/ko/index.html
- Book
  - 이동국, 마이바티스 프로그래밍 JDBC를 대체하는 쉽고 빠른 자바 데이터베이스 프레임워크, 에이콘출판, 2013.