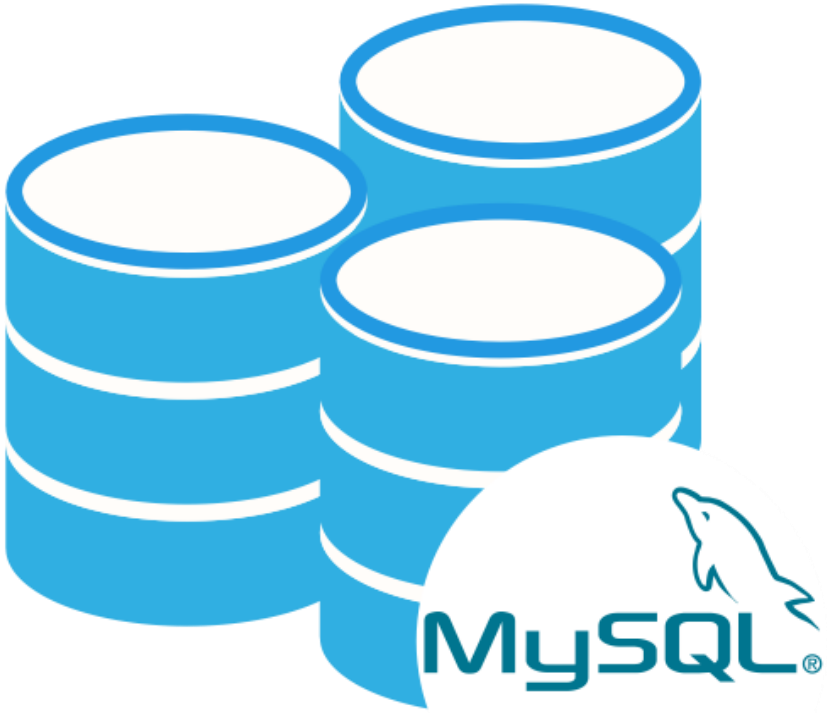


MySQL 기초와 Dynamic query

발표일	2025년 1월 17일
발표자	김수민

목차

1. MySQL 기초

- MySQL 이란?
- DB
- 강제성
- Table 생성
- 기본 문법
- 뷰(view)
- 뷰(view) 명령어

2. Dynamic Query

- Dynamic query(동적 쿼리) 란?
- Dynamic query(동적 쿼리)의 역할
- MySQL 에서의 Dynamic query 구조
- MyBatis란?
- MyBatis 구조
- MyBatis 에서의 XML파일
- MyBatis 에서의 Dynamic query Mapper
- Dynamic query Mapper
  - <if> 조건문
  - <choose>/<when>/<otherwise> 조건 분기
  - <trim> 태그
  - <where> 태그
  - <set> 태그
  - <foreach> 태그

1. MySQL 기초

MySQL 이란?

S Structure

Q Query

L Language

질문을 구조적으로 만드는 언어이다.

1. MySQL 기초

DB

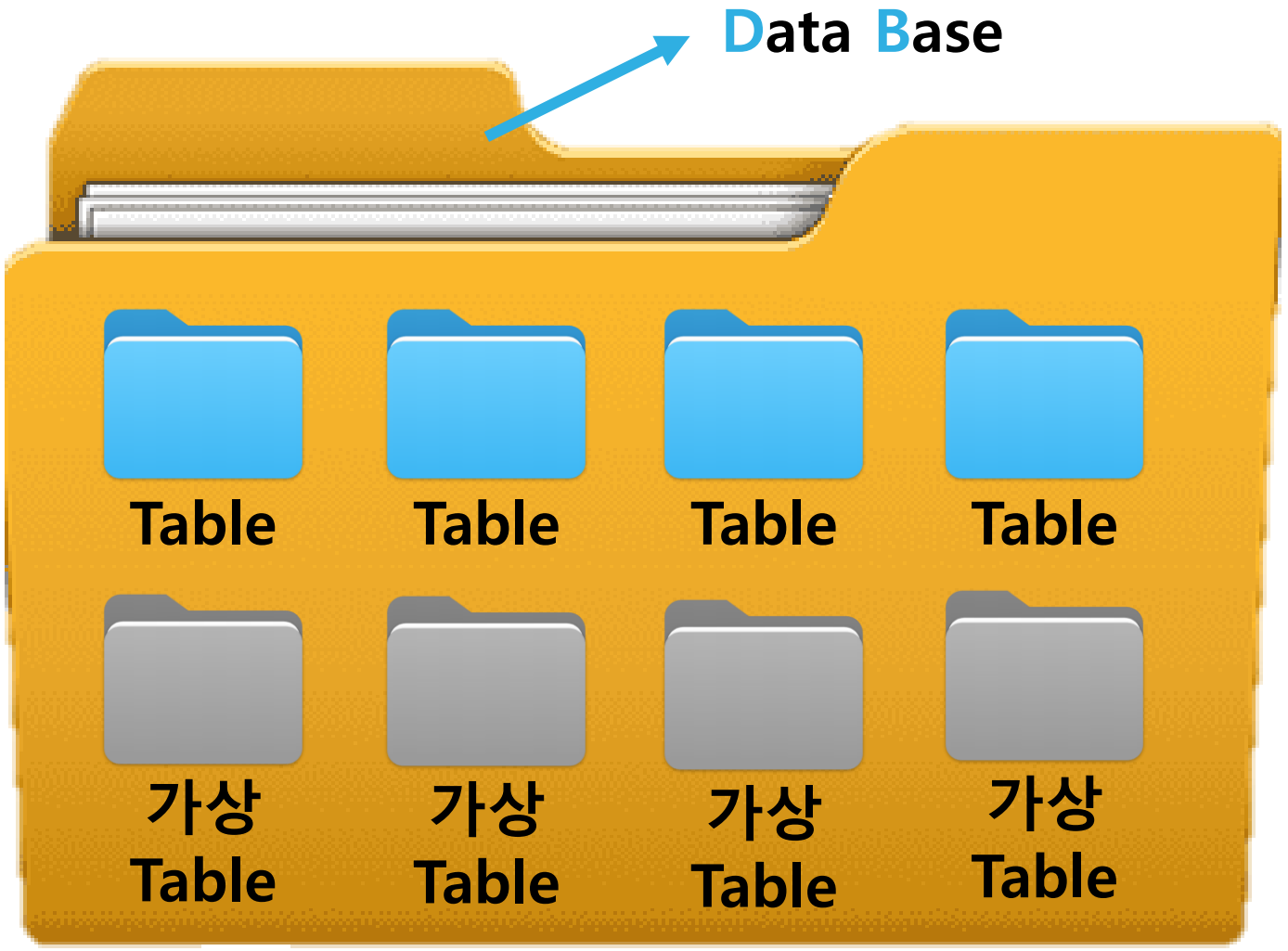
Data Base

자료의 묶음

Table

구조적으로 자료를 구성한 묶음

- **물리 Table**
  : 실제 존재하는 테이블
- **논리 Table**
  = 가상 테이블



1. MySQL 기초

강제성 구조화를 시킨다는 의미

1차적인 강제성
Primary Constraints

- 데이터의 **유형** 강제
  : 잘못된 데이터 유형의 입력 방지
  ex) 정수만, 50자 이하 허용
- **NOT NULL** : 특정 열 값 지정
  ex) name VARCHAR(100) NOT NULL
- **PRIMARY KEY** : 고유 식별자
  ex) id INT PRIMARY KEY

2차적인 강제성
Primary Constraints

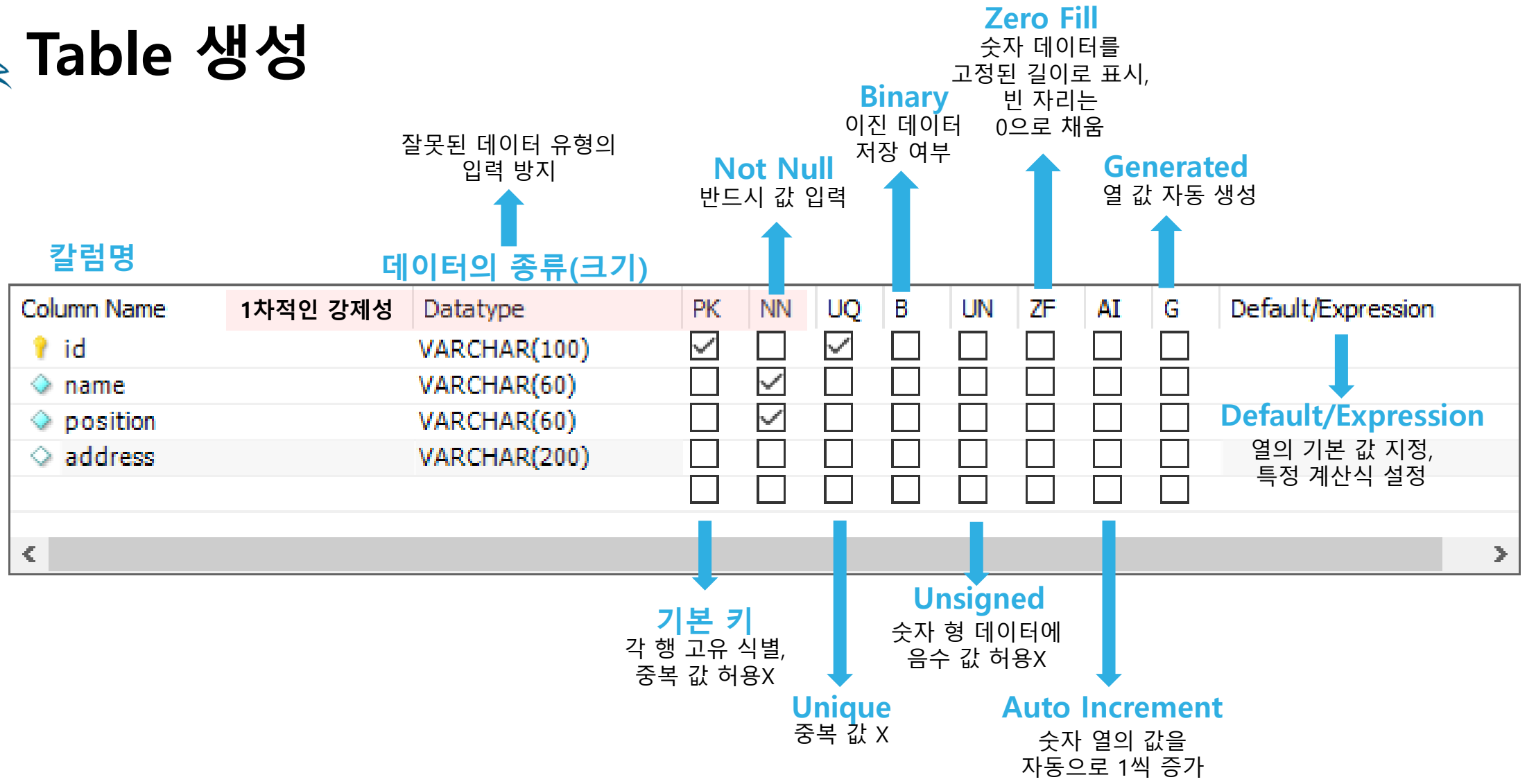
- **FOREIGN KEY** : 외래 키
  ex) customer_id INT PRIMARY KEY
- **CASCADE 규칙**
  : 참조된 데이터를 수정 또는 삭제 시
  연결된 데이터에도 자동 반영
  ex) FOREIGN KEY (customer_id) REFERENCES
  customers(customer_id) ON DELETE CASCADE

3차적인 강제성
Primary Constraints

- **트리거**
  : 특정 이벤트(삽입, 업데이트, 삭제)가
  발생할 때 자동으로 실행되는 규칙
- **스토어드 프로시저**
  : 복잡한 비즈니스 로직을 저장해
  데이터 조작 강제
- **애플리케이션 코드**
  : 데이터베이스에서 강제할 수 없는
  규칙은 애플리케이션에서 구현
  ex) 특정 시간대에만 데이터 삽입 허용

1. MySQL 기초

Table 생성



1. MySQL 기초

기본 문법

DB 객체

Data Base

Table

Table

생성 – Create

수정 - Alter

삭제 – Drop

조회 - Show

Data

" Data는 모든 Table에 저장되어 있다 "

생성 – Insert

수정 - Update

삭제 – Delete

조회 - Select

1. MySQL 기초

뷰 (view)

정의

- 데이터베이스에 존재하는 일종의 가상 테이블
- 행과 열 O , 실제 데이터 X

장점

- 필요한 필드만 보기
- 복잡한 쿼리를 단순화해서 사용
- 사용한 쿼리 재사용

단점

- 한 번 정의된 뷰는 변경 X
- 삽입, 삭제, 갱신 작업에 많은 제한
- 자신만의 인덱스 X

1. MySQL 기초

뷰 (view) 명령어

단일 테이블에서 필요한 필드만 조회

	id	name	email	position	department_id
▶	1	Alice	alice@example.com	Manager	1
	2	Bob	bob@example.com	Developer	2
	3	Charlie	charlie@example.com	Designer	3
	4	Alice	alice@example.com	Manager	1
	5	Bob	bob@example.com	Developer	2
	6	Charlie	charlie@example.com	Designer	3

Table 'employees'

	id	department_name
▶	1	HR
	2	Engineering
	3	Marketing

Table 'departments'

직원 테이블(employees)에서 직원의 id, name, email 을 조회하는 뷰 생성

'뷰를 특정 쿼리의 결과로 정의 ' 하는 역할

```
CREATE VIEW employee_view AS
SELECT id, name, email
FROM employees;
```

➔ 뷰 생성
➔ Id, name, email 을 선택
➔ Employees라는 테이블에서

```
SELECT * FROM employee_view;
```

➔ employee_view 뷰 조회

	id	name	email
▶	1	Alice	alice@example.com
	2	Bob	bob@example.com
	3	Charlie	charlie@example.com
	4	Alice	alice@example.com
	5	Bob	bob@example.com
	6	Charlie	charlie@example.com

뷰 'employee_view'

1. MySQL 기초

id	name	email	position	department_id
1	Alice	alice@example.com	Manager	1
2	Bob	bob@example.com	Developer	2
3	Charlie	charlie@example.com	Designer	3
4	Alice	alice@example.com	Manager	1
5	Bob	bob@example.com	Developer	2
6	Charlie	charlie@example.com	Designer	3

Table 'employees'

id	department_name
1	HR
2	Engineering
3	Marketing

Table 'departments'

뷰 (view) 명령어

여러 테이블에서 필요한 필드만 조회

Employees 테이블과 departments 테이블을 조인하여 직원의 id, name, department_name 을 포함하는 뷰 생성

- ```
  CREATE VIEW employee_department_view AS
  SELECT e.id, e.name, d.department_name
  FROM employees e
  JOIN departments d ON e.department_id = d.id;
  ```
  - ➔ 뷰 생성
  - ➔ employees의 Id, name과 department의 department_name 을 선택
  - ➔ e 를 별칭으로 사용하는 employees라는 테이블에서
  - ➔ D 를 별칭으로 사용하는 departments라는 테이블에서 조인
- `SELECT * FROM employee_department_view;` ➔ employee_view 뷰 조회

조인 조건 : employees 테이블의 department_id 값과 departments 테이블의 id 값이 같을 때 두 테이블의 행 결합

id	name	department_name
1	Alice	HR
2	Bob	Engineering
3	Charlie	Marketing
4	Alice	HR
5	Bob	Engineering
6	Charlie	Marketing

뷰 'employee_department_view'
  ```

1. MySQL 기초

뷰 (view) 명령어

뷰 대체

기존 뷰를 수정하여 'email 필드 제외, position 필드 포함'으로 변경

- ```
  CREATE OR REPLACE VIEW employee_view AS
  SELECT id, name, position
  FROM employees;
  ```
  ➔ 뷰를 대체하여 생성
  ➔ Id, name, position 을 선택
  ➔ Employees라는 테이블에서
- ```
  select * from employee_view;
  ```
  ➔ employee_view 뷰 조회

id	name	email
1	Alice	alice@example.com
2	Bob	bob@example.com
3	Charlie	charlie@example.com
4	Alice	alice@example.com
5	Bob	bob@example.com
6	Charlie	charlie@example.com

id	name	position
1	Alice	Manager
2	Bob	Developer
3	Charlie	Designer
4	Alice	Manager
5	Bob	Developer
6	Charlie	Designer

뷰 'employee_view'

1. MySQL 기초

뷰 (view) 명령어

뷰의 모든 데이터 조회

```
SELECT * FROM employee_view;
```

	id	name	position
▶	1	Alice	Manager
	2	Bob	Developer
	3	Charlie	Designer
	4	Alice	Manager
	5	Bob	Developer
	6	Charlie	Designer

특정 조건을 추가하여 뷰 조회

```
SELECT * FROM employee_department_view
 WHERE department_name = 'Engineering';
```

	id	name	department_name
▶	2	Bob	Engineering
	5	Bob	Engineering

뷰 삭제

- `DROP VIEW employee_view;`

2. Dynamic query

Dynamic query (동적 쿼리) 란?

↔ **정적 쿼리**
: 어떤 조건 또는 상황에도 변경되지 않는 쿼리문

- 실행 시점에서 조건이나 값을 기반으로 SQL 쿼리를 **동적으로 생성하고 실행**하는 기법
- 복잡한 조건 처리, 다양한 입력 값에 따라 쿼리를 유연하게 작성

구분	장점	설명
개발 측면	유연한 구현	복잡한 쿼리의 프로그램 로직 대체로 단순화 가능
	쿼리 재사용	서브 쿼리 및 공통 쿼리의 모듈화 통한 재사용 가능
	개발 편의성	ORM 프레임워크 등을 통해 SQL 개발 용이
성능 측면	조회 속도 보완	NVL 등 성능 저하 쿼리의 회피 및 대체 가능
	실행 계획 최적화	조회 조건 별 실행계획의 최적화 구현 가능
	업무 규칙의 모듈화	업무 규칙의 어플리케이션 제어로 품질 향상 가능

2. Dynamic query

Dynamic query (동적 쿼리) 의 역할

Oracle, MySQL

- 데이터베이스 자체에서 동적 SQL을 생성 및 실행
- SQL 작성과 실행이 DB 내부에서 이루어지므로 DB에 강한 의존성을 가짐
- 복잡한 로직 필요한 경우 PL/SQL(Oracle), 저장 프로시저(MySQL)를 활용해야 함

MyBatis, JPA

- 애플리케이션 코드(JAVA)에서 동적 SQL을 작성
- 파라미터 바인딩과 유지보수성이 뛰어나며, ORM 기반 개발 환경에서 많이 사용
- MyBatis는 SQL 제어가 용이하고, JPA는 객체 중심 개발에 적합

2. Dynamic query

MySQL에서의 Dynamic query 구조

1 고정된 SQL 기본 구조

- ▶ 쿼리의 고정적인 부분 정의 - SELECT, FROM, 기본 WHERE 조건 등등

2 동적으로 변경되는 조건

- ▶ 실행 상황에 필요한 **조건 추가 – WHERE, JOIN, GROUP BY, ORDER BY 등등**
  → 프로그래밍 언어의 조건문(IF, CASE, SWITCH) 또는 SQL 조건 문장 사용

3 파라미터 바인딩

- ▶ 동적으로 생성된 쿼리에 값을 안전하게 삽입하기 위해 파라미터를 바인딩함
- ▶ 이를 통해 SQL 인젝션을 방지하고 성능 최적화 가능

4 쿼리 실행 및 결과 반환

- ▶ 동적으로 완성된 쿼리를 실행하고 결과를 반환

2. Dynamic query

1. 고정된 SQL 기본 구조
2. 동적으로 변경되는 조건
3. 파라미터 바인딩
4. 쿼리 실행 및 결과 반환

MySQL에서의 Dynamic query 구조

	id	name	email	position	department_id
▶	1	Alice	alice@example.com	Manager	1
	2	Bob	bob@example.com	Developer	2
	3	Charlie	charlie@example.com	Designer	3
	4	Alice	alice@example.com	Manager	1
	5	Bob	bob@example.com	Developer	2
	6	Charlie	charlie@example.com	Designer	3

Table 'employees'

```
1  USE company_db;
2
3  -- 필요한 동적 조건 설정
4  SET @department_id = 2;  -- 원하는 부서 ID
5  SET @position = 'Developer'; -- 원하는 직책
6
7  -- 기본 쿼리 설정
8  SET @query = 'SELECT * FROM employees WHERE 1=1';
9
10 -- 조건적으로 쿼리 추가 @department_id 가 Null 이 아닌 경우(값이 있을 경우) , 추가하라(기존 쿼리에 And department_id= 2 조건을), NULL이면 기존 쿼리 유지
11 SET @query = IF(@department_id IS NOT NULL, CONCAT(@query, ' AND department_id = ', @department_id), @query);
12 SET @query = IF(@position IS NOT NULL, CONCAT(@query, ' AND position = "', @position, '"'), @query);
13 @position 가 Null 이 아닌 경우 , 추가하라(기존 쿼리에 And position= Developer 조건을), NULL이면 기존 쿼리 유지
14 -- 쿼리 실행
15 PREPARE stmt FROM @query;
16 EXECUTE stmt;
17 DEALLOCATE PREPARE stmt;
18
19 -- 생성된 쿼리 확인
20 SELECT * FROM employees WHERE 1=1 AND department_id = 2 AND position = "Developer";
```

➔ **필요한 동적 조건 설정**
변수 @department_id=2
변수 @position='Developer'

➔ **기본 쿼리 설정**
employees 테이블에서 모든 데이터를 선택, 추후 동적 조건을 추가하기 위한 항상 참인 1=1 조건 추가

➔ **조건적으로 쿼리 추가**

➔ **쿼리 실행**
→ @query에 저장된 동적 SQL 문자열을 실행 가능한 쿼리로 준비
→ 준비된 쿼리(stmt) 를 실행하여 결과 반환
→ 쿼리를 실행한 후 사용한 자원 해제

➔

	id	name	email	position	department_id
▶	2	Bob	bob@example.com	Developer	2
	5	Bob	bob@example.com	Developer	2

생성된 쿼리 모습

2. Dynamic query

MyBatis 란?

- 자바 오브젝트와 SQL 사이의 자동 매핑 기능을 지원하는 ORM (Object Relational Mapping) 프레임워크
- SQL, 동적 쿼리, 저장 프로시저, 고급 매핑을 지원하는 SQL Mapper

SQL 매핑 프레임워크

- SQL 문장을 직접 작성,
- Java 객체와 데이터베이스 테이블 간의 매핑을 자동으로 처리
- 복잡한 SQL 작성과 실행 지원, 개발자가 SQL의 흐름 제어

SQL과 Java 코드 분리

- SQL 문을 XML 매퍼 파일에 작성 -> Java 코드와 SQL 문이 분리
- 유지보수와 가독성 높음

동적 SQL 지원

- <if>, <choose>, <where>, <foreach> 태그 사용
- 조건에 따라 SQL을 동적으로 생성 가능

파라미터 바인딩 및 결과 매핑

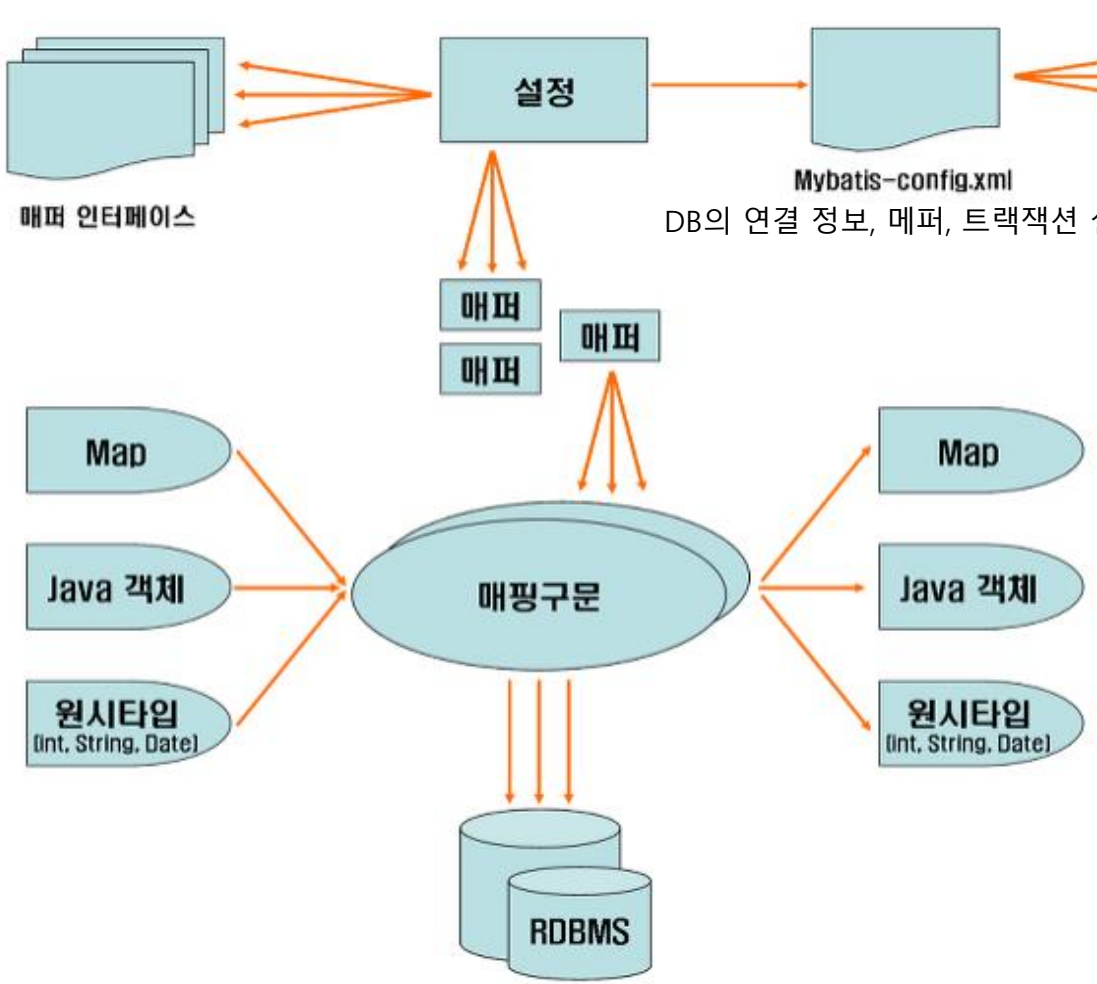
- 입력 값을 SQL 문에 안전하게 바인딩하여 SQL 인젝션 방지
- 실행 결과를 JAVA 객체로 자동 매핑

단순성

- 복잡한 ORM 도구에 비해 간단하고 사용이 쉬움

2. Dynamic query

MyBatis 구조



1. 설정 파일(Mybatis-config.xml)이 MyBatis 환경을 구성
2. 매퍼 인터페이스가 매퍼 XML의 쿼리를 호출
3. 쿼리가 RDBMS에서 실행되어 결과 반환
4. 결과는 매핑구문을 통해 Java 객체, Map, 원시 타입으로 변환
5. 애플리케이션은 변환된 데이터를 활용

2. Dynamic query

MyBatis 에서의 XML파일

- Dynamic query 를 작성하기 위한 파일

MyBatis-config.xml
▷ 설정 파일

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://localhost:3306/mybatis_db" />
                <property name="username" value="root" />
                <property name="password" value="password" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="mapper/UserMapper.xml" />
    </mappers>
</configuration>
```

UserMapper.xml
▷ 메퍼 파일

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.mapper.UserMapper">

    <select id="findUsers" resultType="User">
        SELECT * FROM users
        WHERE 1 = 1
        <if test="name != null">
            AND name = #{name}
        </if>
        <if test="age != null">
            AND age = #{age}
        </if>
    </select>

    <update id="updateUser">
        UPDATE users
        <set>
            <if test="name != null">
                name = #{name},
            </if>
            <if test="age != null">
                age = #{age},
            </if>
        </set>
        WHERE id = #{id}
    </update>

</mapper>
```

2. Dynamic query

MyBatis 에서의 Dynamic query Mapper

▶ 이 매퍼들을 이용해서 동적쿼리를 쉽게 만들 수 있다.

매퍼	설명
<if> 조건문	• 특정 조건이 참일 때 SQL 구문 추가 • 자주 사용되는 가장 기본적인 동적 태그
<choose> / <when>/ <otherwise> 조건 분기	• 여러 조건 중 하나만 만족하는 경우 해당 SQL 실행
<trim> 태그	• SQL의 특정 부분(ex, AND, OR, 콤마 등)을 동적으로 추가하거나 제거 • 불필요한 구문을 제거하는 데 유용
<where> 태그	• 조건이 동적으로 추가될 경우 자동으로 WHERE 생성 • 첫 번째 조건의 앞에 불필요한 AND 또는 OR 을 자동으로 제거
<set> 태그	• UPDATE 문에서 동적 컬럼 설정 처리 • 마지막 쉼표(콤마)를 자동으로 제거
<foreach> 태그	• 반복적인 값을 처리하는 데 사용 • IN 절, 여러 INSERT 구문 생성할 때 유용

2. Dynamic query

Dynamic query Mapper

▶ **<if> 조건문** : 특정 조건이 참일 때 SQL 구문 추가
: 자주 사용되는 가장 기본적인 동적 태그

```xml
<select id="findUsers" resultType="User">
    SELECT * FROM users
    WHERE 1 = 1
    <if test="name != null">
        AND name = #{name}
    </if>
    <if test="age != null">
        AND age = #{age}
    </if>
</select>
```

➔ **Test 속성에 조건식 작성**
조건이 **참일 경우**에만 해당 SQL 실행

파라미터: { name: 'John', age: 25 }
<조건이 모두 참일 경우>

```sql
SELECT * FROM users
WHERE 1 = 1
AND name = 'John'
AND age = 25
```

파라미터: { name: null, age: null }
<조건이 모두 거짓일 경우>

```sql
SELECT * FROM users
WHERE 1 = 1
```

2. Dynamic query

Dynamic query Mapper

<choose>/<when>/<otherwise> 조건 분기

- <when> : 특정 조건이 참일 때 SQL 실행
- <otherwise> : 모든 조건이 거짓일 때 SQL 실행
- 여러 개의 <when> 중 하나만 실행, 조건이 만족하지 않으면 <otherwise>가 실행된다.

```xml
<select id="findUsersByStatus" resultType="User">
    SELECT * FROM users
    WHERE
    <choose>
        <when test="status == 'ACTIVE'">
            status = 'ACTIVE'
        </when>
        <when test="status == 'INACTIVE'">
            status = 'INACTIVE'
        </when>
        <otherwise>
            status = 'UNKNOWN'
        </otherwise>
    </choose>
</select>
```

➔ status=='ACTIVE' 이 참일 경우 실행

➔ status=='ACTIVE' 이 거짓,
➔ status=='INACTIVE' 이 참일 경우 실행

➔ status=='ACTIVE' 이 거짓,
➔ status=='INACTIVE' 이 거짓일 경우 실행

데이터베이스 테이블 `users`

id	name	status
1	John	ACTIVE
2	Alice	INACTIVE
3	Bob	UNKNOWN

파라미터: { status: 'ACTIVE' }

id	name	status
1	John	ACTIVE

파라미터: { status: 'INACTIVE' }

id	name	status
2	Alice	INACTIVE

파라미터: { status: null }

id	name	status
3	Bob	UNKNOWN

2. Dynamic query

Dynamic query Mapper

▶ **<trim> 태그** : SQL의 특정 부분(ex, AND, OR, 콤마 등)을 동적으로 추가하거나 제거
: 불필요한 구문을 제거하는 데 유용

```xml
<select id="findUsers" resultType="User">
    SELECT * FROM users
    <trim prefix="WHERE" prefixOverrides="AND | OR">
        <if test="name != null">
            AND name = #{name}
        </if>
        <if test="age != null">
            AND age = #{age}
        </if>
    </trim>
</select>
```

➔ 조건이 있을 때만 WHERE 를 붙이는 규칙 추가

➔ 불필요한 접두사 (AND, OR)를 제거

➔ 이름 값이 있을 때만

➔ 추가

➔ 나이 값이 있을 때만

➔ 추가

테이블 이름: `users`

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	NULL	ACTIVE
4	Charlie	28	UNKNOWN

파라미터: { name: null, age: null }

<이름 값 X, 나이 값 X>

```sql
SELECT * FROM users;
```

결과 데이터:

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	NULL	ACTIVE
4	Charlie	28	UNKNOWN

파라미터: { name: 'John', age: null }

<이름 값 O, 나이 값 X>

```sql
SELECT * FROM users
WHERE name = 'John';
```

결과 데이터:

id	name	age	status
1	John	25	ACTIVE

파라미터: { name: null, age: 30 }

<이름 값 X, 나이 값 O>

```sql
SELECT * FROM users
WHERE age = 30;
```

결과 데이터:

id	name	age	status
2	Alice	30	INACTIVE

2. Dynamic query

테이블 이름: `users`

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	NULL	ACTIVE
4	Charlie	28	UNKNOWN

Dynamic query Mapper

▶ **<where> 태그**

- 조건이 동적으로 추가될 경우 자동으로 WHERE 생성
- 첫 번째 조건의 앞에 불필요한 AND 또는 OR 을 자동으로 제거

```xml
<select id="findUsers" resultType="User">
    SELECT * FROM users
    <where>
        <if test="name != null">
            AND name = #{name}
        </if>
        <if test="age != null">
            AND age = #{age}
        </if>
        <if test="status != null">
            AND status = #{status}
        </if>
    </where>
</select>
```

➔ 조건이 하나라도 참이면 자동으로 where 키워드 추가

➔ TEST 조건이 참일 경우에만 추가
FALSE일 경우, SQL 쿼리에서 무시

파라미터: { name: 'John', age: null, status: null }

<참, 거짓, 거짓>

```sql
SELECT * FROM users
WHERE name = 'John';
```

결과 데이터:

id	name	age	status
1	John	25	ACTIVE

파라미터: { name: null, age: null, status: null }

<거짓, 거짓, 거짓>

```sql
SELECT * FROM users;
```

결과 데이터:

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	NULL	ACTIVE
4	Charlie	28	UNKNOWN

2. Dynamic query

Dynamic query Mapper

▶ **<set> 태그** : UPDATE 문에서 동적 컬럼 설정 처리
: 마지막 쉼표(콤마)를 자동으로 제거

```xml
<update id="updateUser">
    UPDATE users
    <set>
        <if test="name != null">
            name = #{name},
        </if>
        <if test="age != null">
            age = #{age},
        </if>
        <if test="status != null">
            status = #{status},
        </if>
    </set>
    WHERE id = #{id}
</update>
```

➔ 동적으로 컬럼 추가 & 마지막에 붙는 쉼표(,) 자동 제거

➔ 조건에 따라 컬럼을 업데이트 구문에 포함

테이블 이름: `users`

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	NULL	ACTIVE
4	Charlie	28	UNKNOWN

파라미터: { id: 1, name: 'John Doe', age: null, status: 'ACTIVE' }

<참, 거짓, 참>

```sql
UPDATE users
SET name = 'John Doe',
    status = 'ACTIVE'
WHERE id = 1;
```

결과 데이터:

id	name	age	status
1	John Doe	25	ACTIVE

파라미터 { id: 3, name: null, age: 35, status: 'INACTIVE' }

<거짓, 참, 참>

```sql
UPDATE users
SET age = 35,
    status = 'INACTIVE'
WHERE id = 3;
```

결과 데이터:

id	name	age	status
3	Bob	35	INACTIVE

2. Dynamic query

테이블 이름: `users`

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	35	ACTIVE
4	Charlie	28	UNKNOWN

Dynamic query Mapper

▶ **<foreach> 태그** : 반복적인 값을 처리하는 데 사용 -> 반복 작업 처리
IN 절, 여러 INSERT 구문 생성할 때 유용

< IN 절 >

```xml
<select id="findUsersByIds" resultType="User">
    SELECT * FROM users
    WHERE id IN
    <foreach collection="idList" item="id" open="(" separator="," close=")">
        #{id}
    </foreach>
</select>
```

➔ 구문을 자연스럽게 만들 때 사용 -> (,) 삽입

➔ **idList라는 리스트에 담긴 여러 ID 값을 IN절에 동적으로 삽입**

```sql
SELECT * FROM users WHERE id IN (1, 3);
```

➡

id	name	age	status
1	John	25	ACTIVE
3	Bob	35	ACTIVE

2. Dynamic query

id	name	age	status
1	John	25	ACTIVE
2	Alice	30	INACTIVE
3	Bob	35	ACTIVE
4	Charlie	28	UNKNOWN
5	David	40	ACTIVE
6	Emma	32	INACTIVE
7	Frank	45	ACTIVE

Dynamic query Mapper

▶ **<foreach> 태그** : 반복적인 값을 처리하는 데 사용 -> 반복 작업 처리
IN 절, 여러 INSERT 구문 생성할 때 유용

< 다중 INSERT >

```xml
<insert id="insertUsers">
    INSERT INTO users (name, age)
    VALUES
    <foreach collection="userList" item="user" separator=",">
        (#{user.name}, #{user.age})
    </foreach>
</insert>
```

➔ 각 데이터를 콤마(,)로 구분

➔ **underList라는 리스트에 담긴 여러 사용자의 정보를 한 번에 insert**

```sql
SELECT * FROM users
WHERE id IN (1, 3, 5);
```

➡

id	name	age	status
1	John	25	ACTIVE
3	Bob	35	ACTIVE
5	David	40	ACTIVE

감사합니다