

4. 엔티티 매핑 - 목차

1. 객체와 테이블 매핑
2. 데이터베이스 스키마 자동 생성
3. 필드와 컬럼 매핑
4. 기본 키 매핑
5. 실전 예제 - 1. 요구사항 분석과 기본 매핑

4. 엔티티 매핑 - 소개

1. 객체와 테이블 매핑 : `@Entity`, `@Table`
2. 데이터베이스 스키마 자동 생성
3. 필드와 컬럼 매핑 : `@Column`
4. 기본 키 매핑 : `@Id`
5. 연관 관계 매핑 : `@ManyToOne`, `@JoinColumn`

4-1. 객체와 테이블 매핑

@Entity

- @Entity가 붙은 클래스는 JPA가 관리, 엔티티라 한다.
- JPA를 사용해서 테이블과 매핑할 클래스는 @Entity 필수

주의)

- 기본 생성자 필수 (파라미터가 없는 public 또는 protected 생성자)
- final 클래스, enum, interface, inner 클래스 사용 X
- 저장할 필드에 final 사용 X

4-1. 객체와 테이블 매핑

@Entity

속성) name

- JPA에서 사용할 엔티티 이름을 지정한다.
- 기본값: 클래스 이름을 그대로 사용(예: Member)
- 같은 클래스 이름이 없으면 가급적 기본값을 사용한다.

```
package hellojpa;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Member {
    @Id
    private Long id;
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

4-1. 객체와 테이블 매핑

@Table

- @Table은 엔티티와 매핑할 테이블 지정

속성	기능	기본값
name	매핑할 테이블 이름	엔티티 이름을 사용
catalog	데이터베이스 catalog 매핑	
schema	데이터베이스 schema 매핑	
uniqueConstraints (DDL)	DDL 생성 시에 유니크 제약 조건 생성	

```
@Table(name = "MBR")
public class Member {
    @Id
    private Long id;
```

```
Hibernate:
    select
        member0_.id as id1_0_0_,
        member0_.name as name2_0_0_
    from
        MBR member0_
    where
        member0_.id=?
```

4-2. 데이터베이스 스키마 자동 생성

- DDL을 애플리케이션 실행 시점에 자동 생성
- 테이블 중심 -> 객체 중심
- 데이터베이스方言을 활용해서 데이터베이스에 맞는 적절한 DDL 생성
- 이렇게 생성된 DDL은 개발 장비에서만 사용
- 생성된 DDL은 운영서버에서는 사용하지 않거나, 적절히 다듬은 후 사용

4-2. 데이터베이스 스키마 자동 생성

hibernate.hbm2ddl.auto

```
java × Member.java × persistence.xml ×
<property name="javax.persistence.jdbc.password" value=""/>
<property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test"/>
<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>

<!-- 옵션 -->
<property name="hibernate.show_sql" value="true"/>
<property name="hibernate.format_sql" value="true"/>
<property name="hibernate.use_sql_comments" value="true"/>
<!--<property name="hibernate.hbm2ddl.auto" value="create" />-->
</properties>
</persistence-unit>
```

옵션	설명
create	기존테이블 삭제 후 다시 생성 (DROP + CREATE)
create-drop	create와 같으나 종료시점에 테이블 DROP
update	변경분만 반영(운영DB에는 사용하면 안됨)
validate	엔티티와 테이블이 정상 매핑되었는지만 확인
none	사용하지 않음

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "MBR")
public class Member {
    @Id
    private Long id;
    private String name;
    private int age;
```

4-2. 데이터베이스 스키마 자동 생성

주의)

- 운영 장비에는 절대 create, create-drop, update 사용하면 안된다.
- 개발 초기 단계는 create 또는 update
- 테스트 서버는 update 또는 validate
- 스테이징과 운영 서버는 validate 또는 none

4-2. 데이터베이스 스키마 자동 생성

DDL 생성 기능)

- 제약조건 추가: 회원 이름은 필수, 10자 초과X
- @Column(nullable = false, length = 10)
- 유니크 제약조건 추가
- @Table(uniqueConstraints = {@UniqueConstraint(name = "NAME_AGE_UNIQUE", columnNames = {"NAME", "AGE"})})
- DDL 생성 기능은 DDL을 자동 생성할 때만 사용되고 JPA의 실행 로직에는 영향을 주지 않는다.

```
alter table Member
  add constraint UK_ektea7vp6e3low620iewuxhlq unique (name)
01, 2019 11:25:50 PM org.hibernate.tool.schema.internal.Schema
FO: HHH0000476: Executing import script 'org.hibernate.tool.sche
```

4-3. 필드와 컬럼 매핑

매핑 어노테이션

어노테이션	설명
@Column	컬럼 매핑
@Temporal	날짜 타입 매핑
@Enumerated	enum 타입 매핑
@Lob	BLOB, CLOB 매핑
@Transient	특정 필드를 컬럼에 매핑하지 않음(매핑 무시)

4-3. 필드와 컬럼 매핑

@Column

속성	설명	기본값
name	필드와 매핑할 테이블의 컬럼 이름	객체의 필드 이름
insertable, updatable	등록, 변경 가능 여부	TRUE
nullable(DDL)	null 값의 허용 여부를 설정한다. false로 설정하면 DDL 생성 시에 not null 제약조건이 붙는다.	
unique(DDL)	@Table의 uniqueConstraints와 같지만 한 컬럼에 간단히 유니크 제약조건을 걸 때 사용한다.	
columnDefinition(DDL)	데이터베이스 컬럼 정보를 직접 줄 수 있다. ex) varchar(100) default 'EMPTY'	필드의 자바 타입과 방언 정보를 사용해
length(DDL)	문자 길이 제약조건, String 타입에만 사용한다.	255
precision, scale(DDL)	BigDecimal 타입에서 사용한다(BigInteger도 사용할 수 있다). precision은 소수점을 포함한 전체 자릿수를, scale은 소수의 자릿수다. 참고로 double, float 타입에는 적용되지 않는다. 아주 큰 숫자나 정밀한 소수를 다루어야 할 때만 사용한다.	precision=19, scale=2

4-3. 필드와 컬럼 매핑

@Temporal

날짜 타입(`java.util.Date`, `java.util.Calendar`)을 매핑할 때 사용

참고: `LocalDate`, `LocalDateTime`을 사용할 때는 생략 가능(최신 하이버네이트 지원)

속성	설명	기본값
value	<ul style="list-style-type: none">• TemporalType.DATE: 날짜, 데이터베이스 date 타입과 매핑(예: 2013-10-11)• TemporalType.TIME: 시간, 데이터베이스 time 타입과 매핑(예: 11:11:11)• TemporalType.TIMESTAMP: 날짜와 시간, 데이터베이스 timestamp 타입과 매핑(예: 2013-10-11 11:11:11)	

4-3. 필드와 컬럼 매핑

@Enumerated

자바 enum 타입을 매핑할 때 사용

주의! ORDINAL 사용X

속성	설명	기본값
value	<ul style="list-style-type: none">EnumType.ORDINAL: enum 순서를 데이터베이스에 저장EnumType.STRING: enum 이름을 데이터베이스에 저장	EnumType.ORDINAL

4-3. 필드와 컬럼 매핑

@Lob

데이터베이스 BLOB, CLOB 타입과 매핑

- @Lob에는 지정할 수 있는 속성이 없다.
- 매핑하는 필드 타입이 문자면 CLOB 매핑, 나머지는 BLOB 매핑
- CLOB: String, char[], java.sql.CLOB
- BLOB: byte[], java.sql.BLOB

4-3. 필드와 컬럼 매핑

@Transient

- 필드 매핑X
- 데이터베이스에 저장X, 조회X
- 주로 메모리상에서만 임시로 어떤 값을 보관하고 싶을 때 사용
- @Transient

```
private Integer temp;
```

4-4. 기본 키 매핑

- @Id
- @GeneratedValue

매핑 방법)

- 직접 할당: @Id만 사용
- 자동 생성(@GeneratedValue)
 - IDENTITY: 데이터베이스에 위임, MYSQL
 - SEQUENCE: 데이터베이스 시퀀스 오브젝트 사용, ORACLE
@SequenceGenerator 필요
 - TABLE: 키 생성용 테이블 사용, 모든 DB에서 사용
@TableGenerator 필요
 - AUTO: 방언에 따라 자동 지정, 기본값

4-4. 기본 키 매핑

IDENTITY 전략 - 특징

- 기본 키 생성을 데이터베이스에 위임
- 주로 MySQL, PostgreSQL, SQL Server, DB2에서 사용
(예: MySQL의 AUTO_INCREMENT)
- AUTO_INCREMENT는 데이터베이스에 INSERT SQL을 실행한 이후에 ID 값을 알 수 있음
- IDENTITY 전략은 em.persist() 시점에 즉시 INSERT SQL 실행하고 DB에서 식별자를 조회

TABLE 전략

- 키 생성 전용 테이블을 하나 만들어서 데이터베이스 시퀀스를 흉내내는 전략
- 장점: 모든 데이터베이스에 적용 가능
- 단점: 성능

SEQUENCE 전략 - 특징

- 데이터베이스 시퀀스는 유일한 값을 순서대로 생성하는 특별한 데이터베이스 오브젝트(예: 오라클 시퀀스)
- 오라클, PostgreSQL, DB2, H2 데이터베이스에서 사용

Auto

- 셋 중 하나 선택

4-4. 기본 키 매핑

권장하는 식별자 전략

- 기본 키 제약 조건: null 아님, 유일, **변하면 안된다**.
- 미래까지 이 조건을 만족하는 자연키는 찾기 어렵다. 대리키(대체키)를 사용하자.
- 예를 들어 주민등록번호도 기본 키로 적절하지 않다.
- 권장: Long형 + 대체키 + 키 생성전략 사용