Spring Data JPA

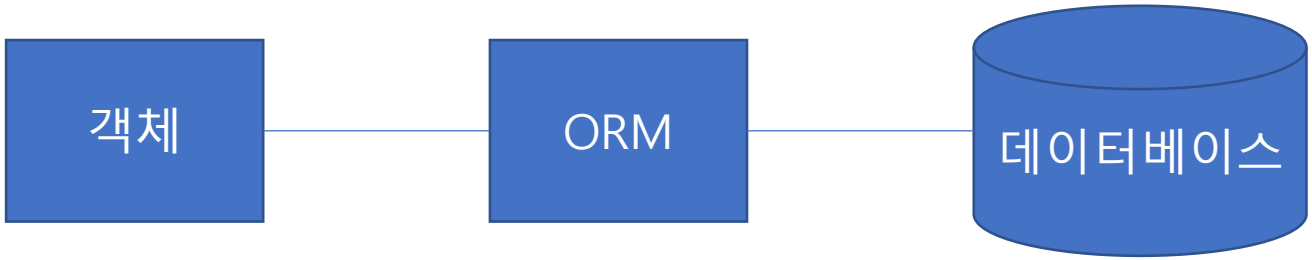
Spring Boot

ORM

JPA

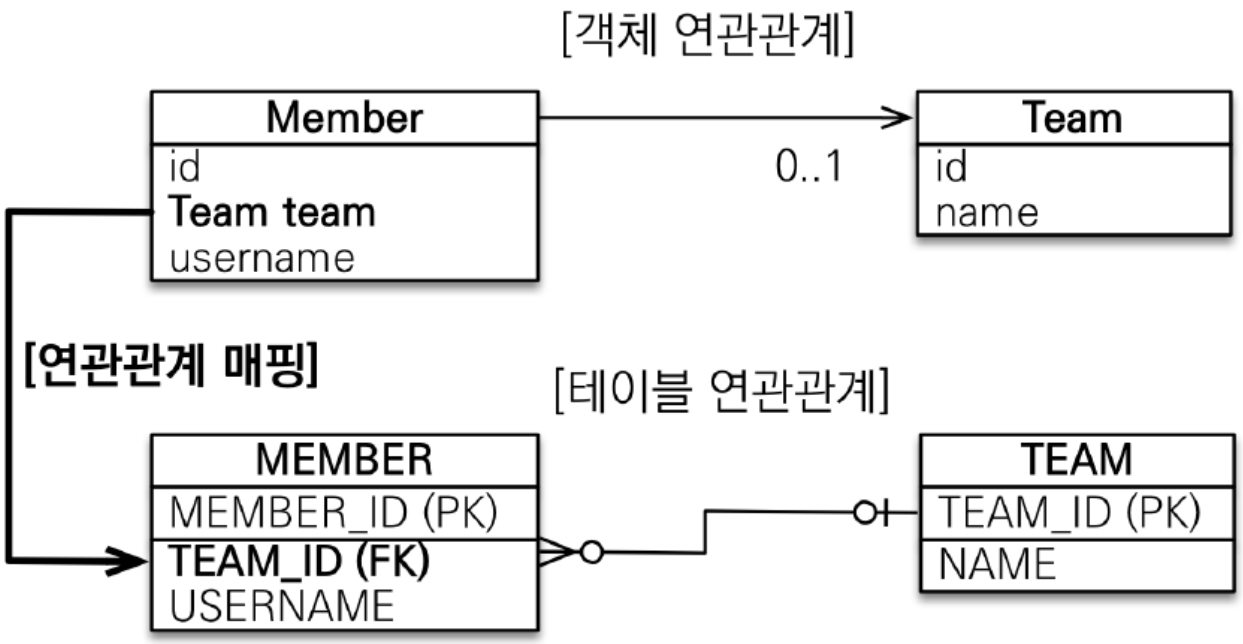
ORM

- Object Relational Mapping(객체 관계 매핑)
- 객체는 객체대로 설계
- 관계형 DB는 DB대로 설계
- ORM 프레임워크가 중간에서 매핑
- 대중적인 언어에는 대부분 ORM 기술이 존재



https://www.tutorialspoint.com/jpa/jpa_orm_components.htm

객체 지향 모델링(ORM 매핑)



ORM 장점

DB Query를 객체지향적으로 조작

- 개발 비용 감소
- 높은 가독성

재사용성 및 유지보수성

- 재사용성 높음
- 유지보수성 높음

DB에 비종속적

- ORM을 통한 자동 생성 SQL문
- DB 교체에 대한 적은 리스크

ORM 단점

ORM의 한계

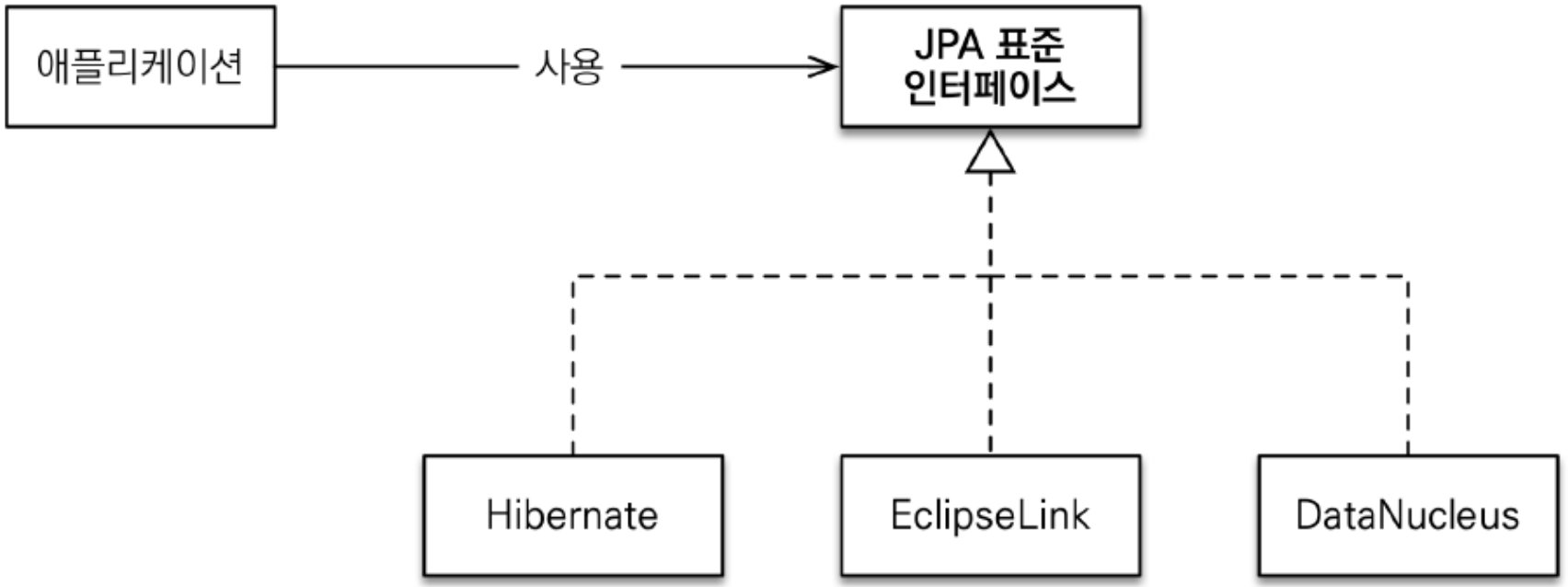
- 복잡한 Query는 코드 구현이 어려움
- 잘못된 ORM 구성은 성능 저하 문제 발생

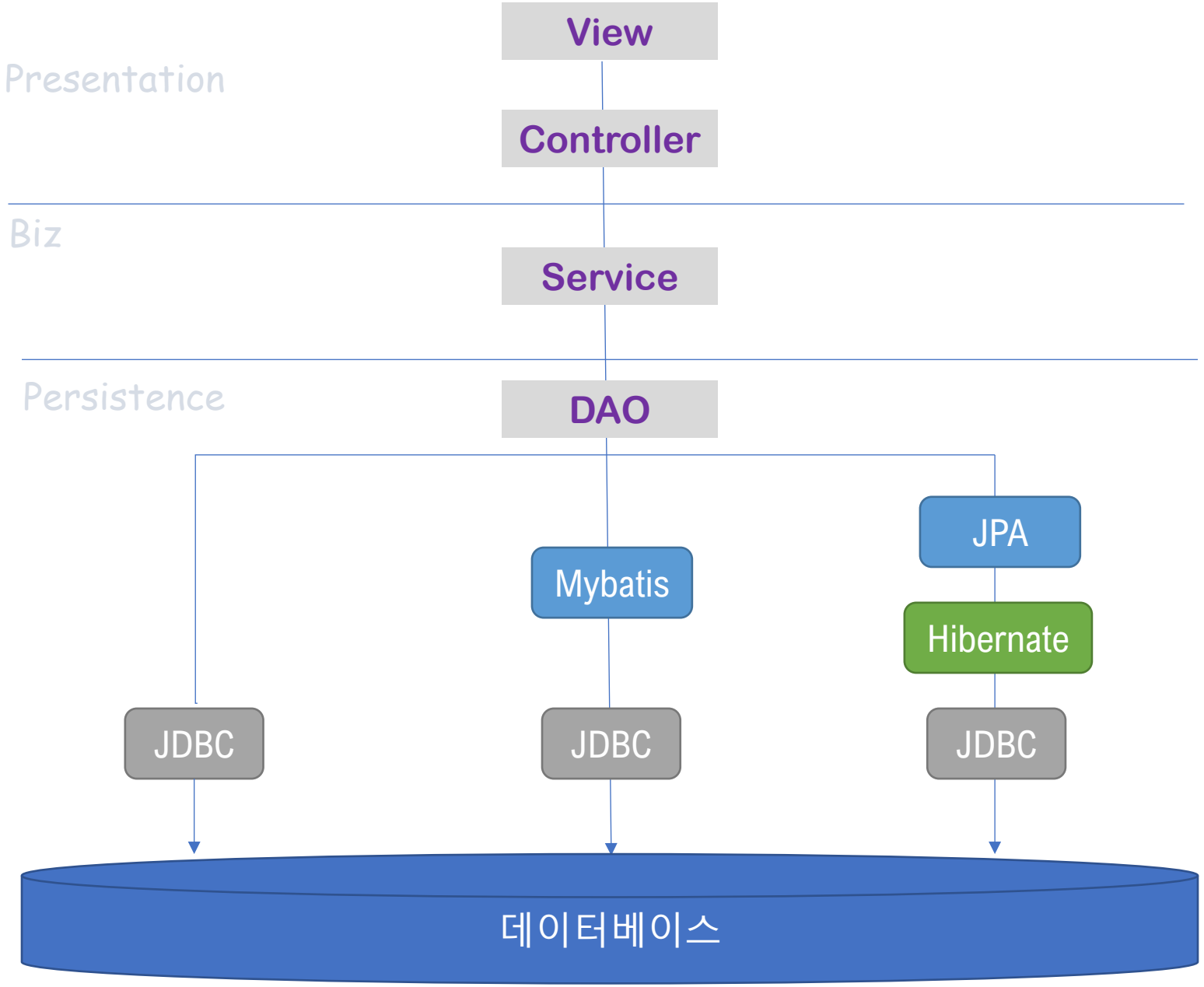
Application과 DB의 관점 불일치

- 세분성(Granularity)
- 상속성(Inheritance)
- 식별성(Identity)
- 연관성(Associations)
- 탐색(Navigation)

JPA

- JPA는 표준 명세(인터페이스)
- JPA를 구현한 3가지 구현체
- hibernate, EclipseLink, DataNucleus



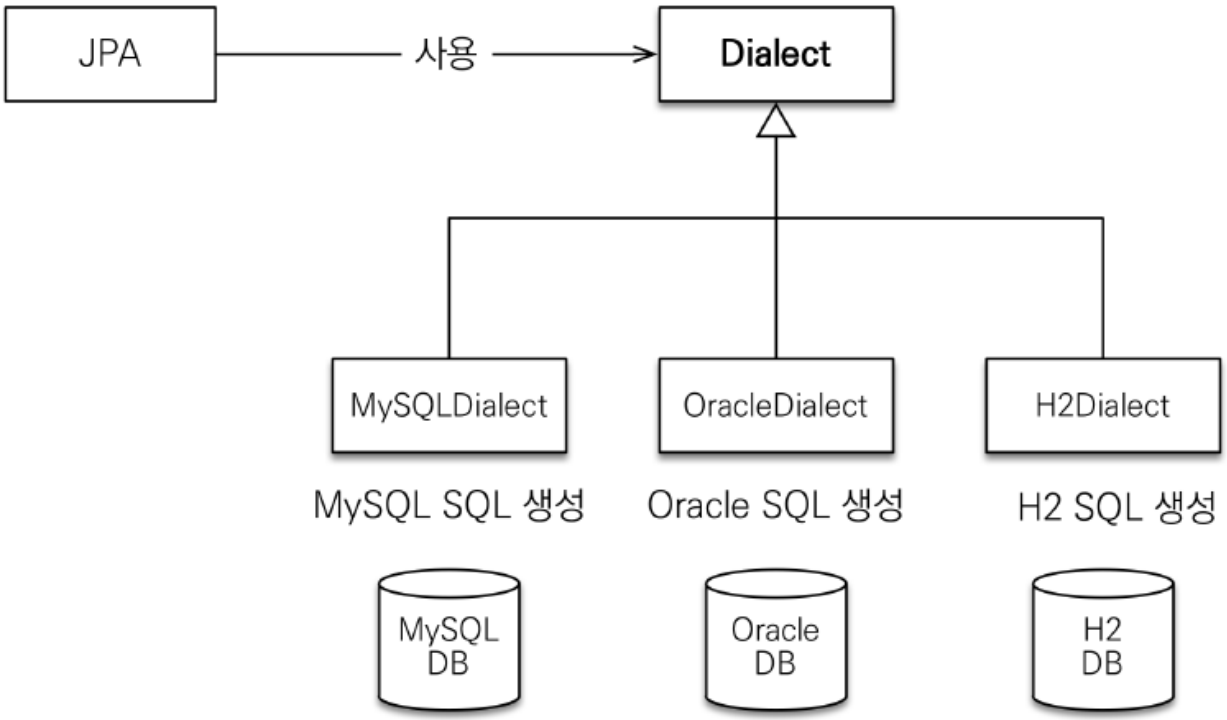

Presentation
View
Controller
Biz
Service
Persistence
DAO
JPA
Mybatis
Hibernate
JDBC
JDBC
JDBC
데이터베이스

application.properties

```
#H2
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:tcp://localhost/~/test
spring.datasource.username=sa
spring.datasource.password=

#jpa
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

DB Dialect


JPA
사용
Dialect
MySQLDialect
OracleDialect
H2Dialect
MySQL SQL 생성
Oracle SQL 생성
H2 SQL 생성
MySQL DB
Oracle DB
H2 DB

Dialect 클래스 설정

spring.jpa.database-platform =

데이터베이스	Dialect 클래스
DB2	org.hibernate.dialect.DB2Dialect
PostgreSQL	org.hibernate.dialect.PostgreDialect
Oracle	org.hibernate.dialect.OracleDialect
Oracle9/10g	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDialect
H2	org.hibernate.dialect.H2Dialect
MySQL	org.hibernate.dialect.MySQL5InnoDBDialect

JPA 구현체

속성	의미
spring.jpa.hibernate.ddl-auto	테이블 생성(create)이나 변경(alter), 삭제(drop) 같은 DDL 구분을 자동으로 실행할지 지정 (값 = create, update, create-drop, validate, none)
	• create : 기존 테이블 삭제 후 다시 생성 • create-drop : create와 같으나 종료시점에 테이블 drop • update : 변경된 부분만 반영 • validate : 엔티티와 테이블이 정상적으로 매핑되었는지만 확인 • none : 사용하지 않음
spring.jpa.show-sql	하이버네이트가 생성한 SQL을 콘솔에 출력 (값 = true)
spring.jpa.properties.hibernate.format_sql	하이버네이트가 생성한 SQL을 출력할때 보기 좋은 포맷으로 출력(값=true)
spring.jpa.database	사용하는 DB를 지정(값 = H2)

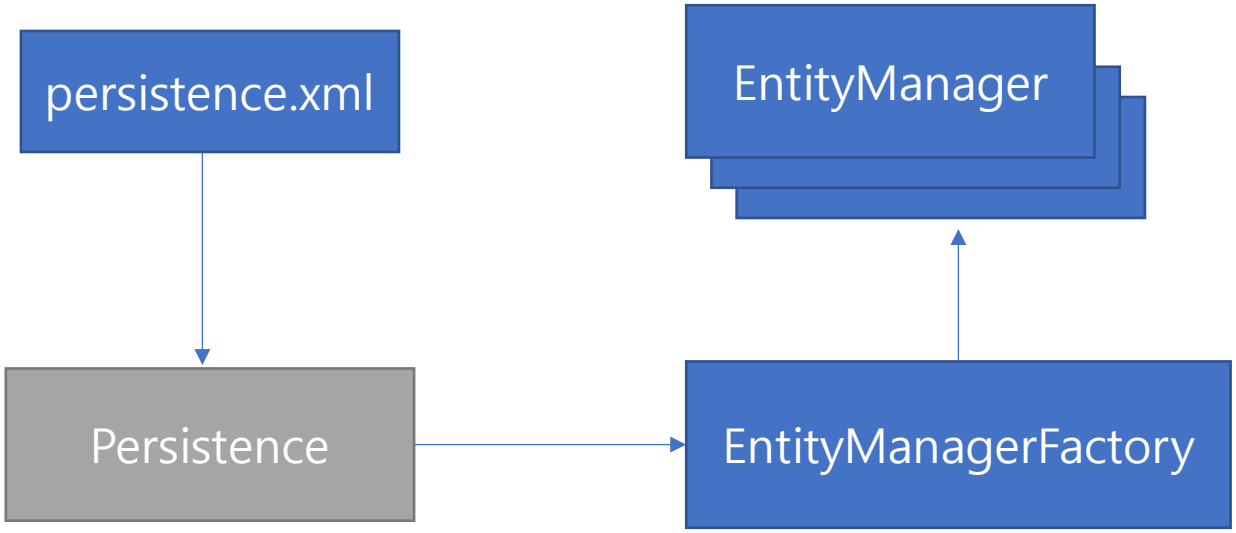
DB Schema 자동 생성시 주의 사항

- 운영 장비에는 create, create-drop, update 사용 안됨
- 개발 초기 단계는 create 또는 update
- 테스트 서버는 update 또는 validate
- 스테이징과 운영 서버는 validate 또는 none

Persistent Context

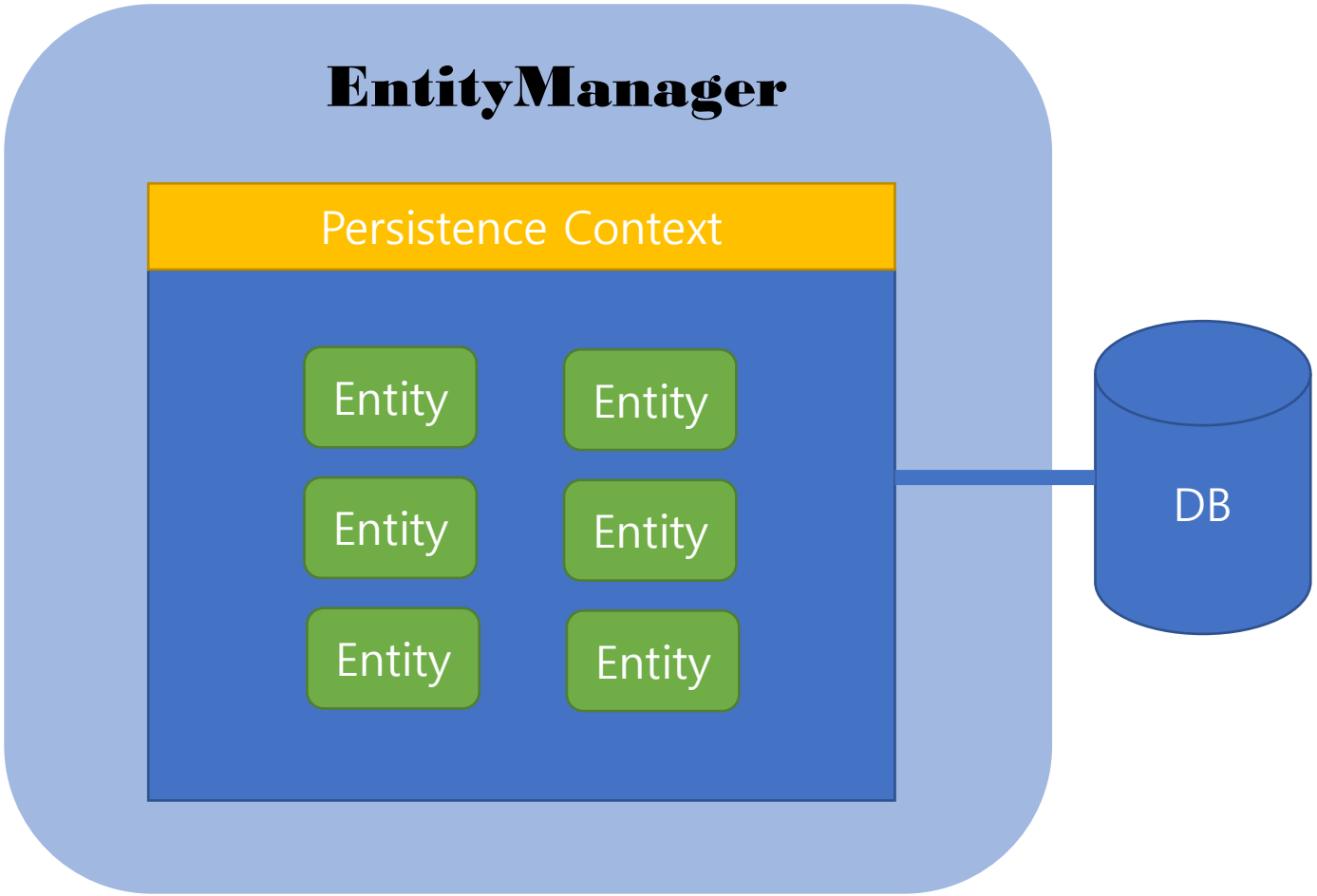
JPA

EntityManager

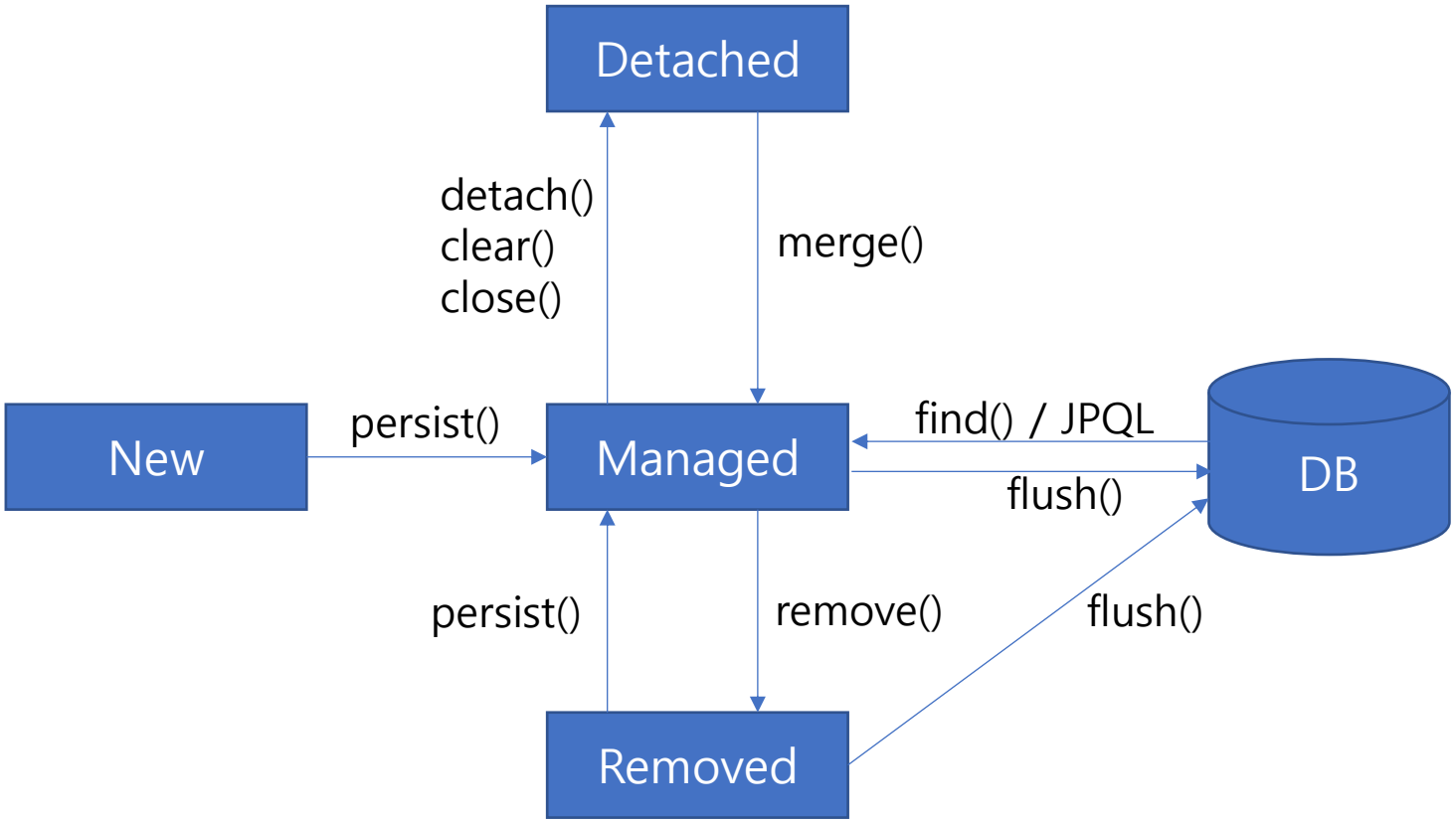


```
//엔티티 매니저 팩토리 생성
EntityManagerFactory emf =
                Persistence.createEntityManagerFactory("boot");

EntityManager em = emf.createEntityManager(); //엔티티 매니저 생성
EntityTransaction tx = em.getTransaction();   //엔티티 트랜잭션 생성

try{
        //엔티티 생성
        Member member = new Member();
        member.setId("user1");
        member.setName("Amy");

        tx.begin(); //트랜잭션 시작

        em.persistent(member); //회원 등록 요청

        tx.commit(); //트랜잭션 종료(COMMIT)

}catch(Exception e){
        e.printStackTrace();

        tx.rollback(); //트랜잭션 종료(ROLLBACK)
}
```

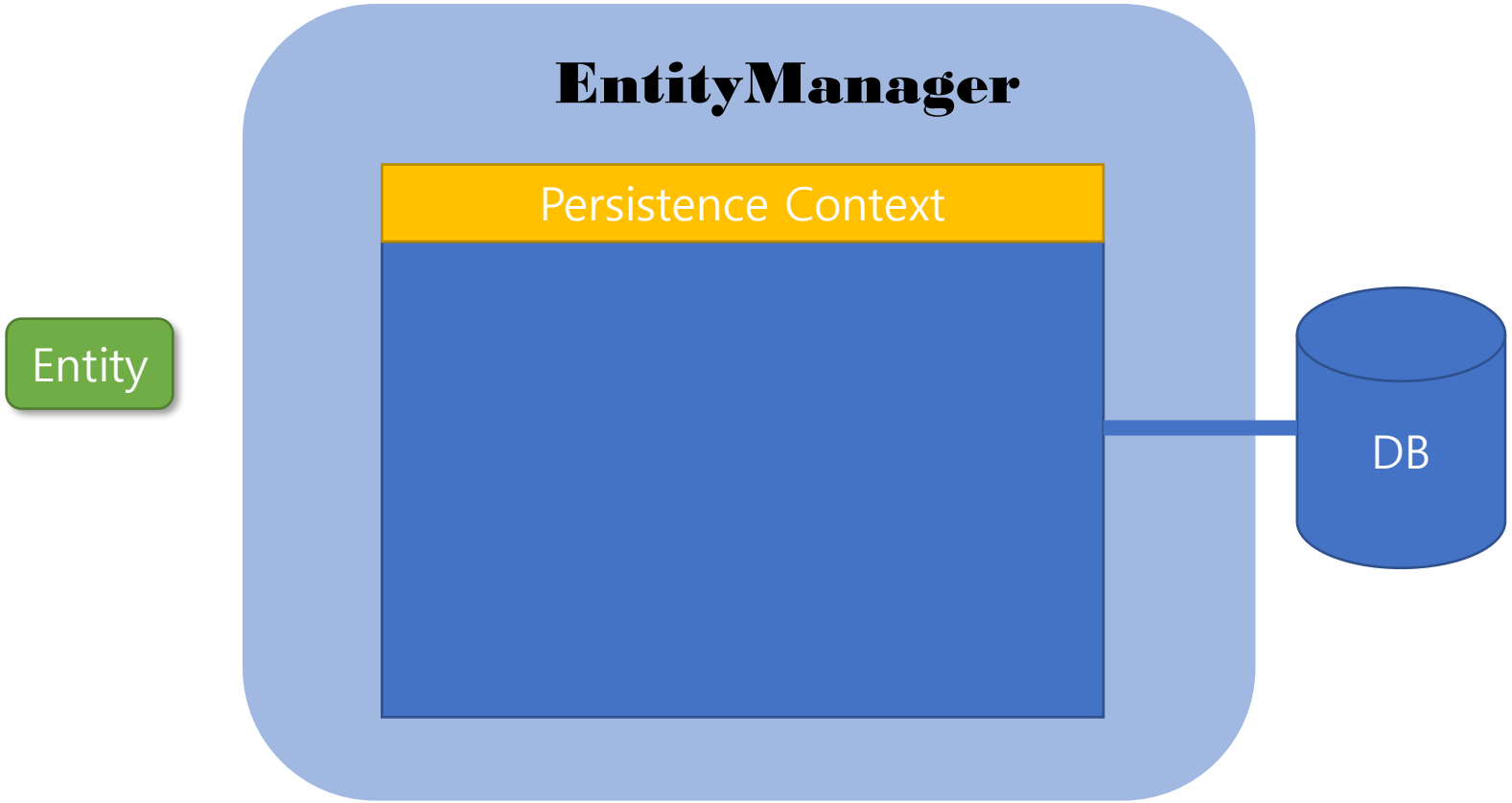
EntityManager



엔티티 상태와 상태 전이 메소드

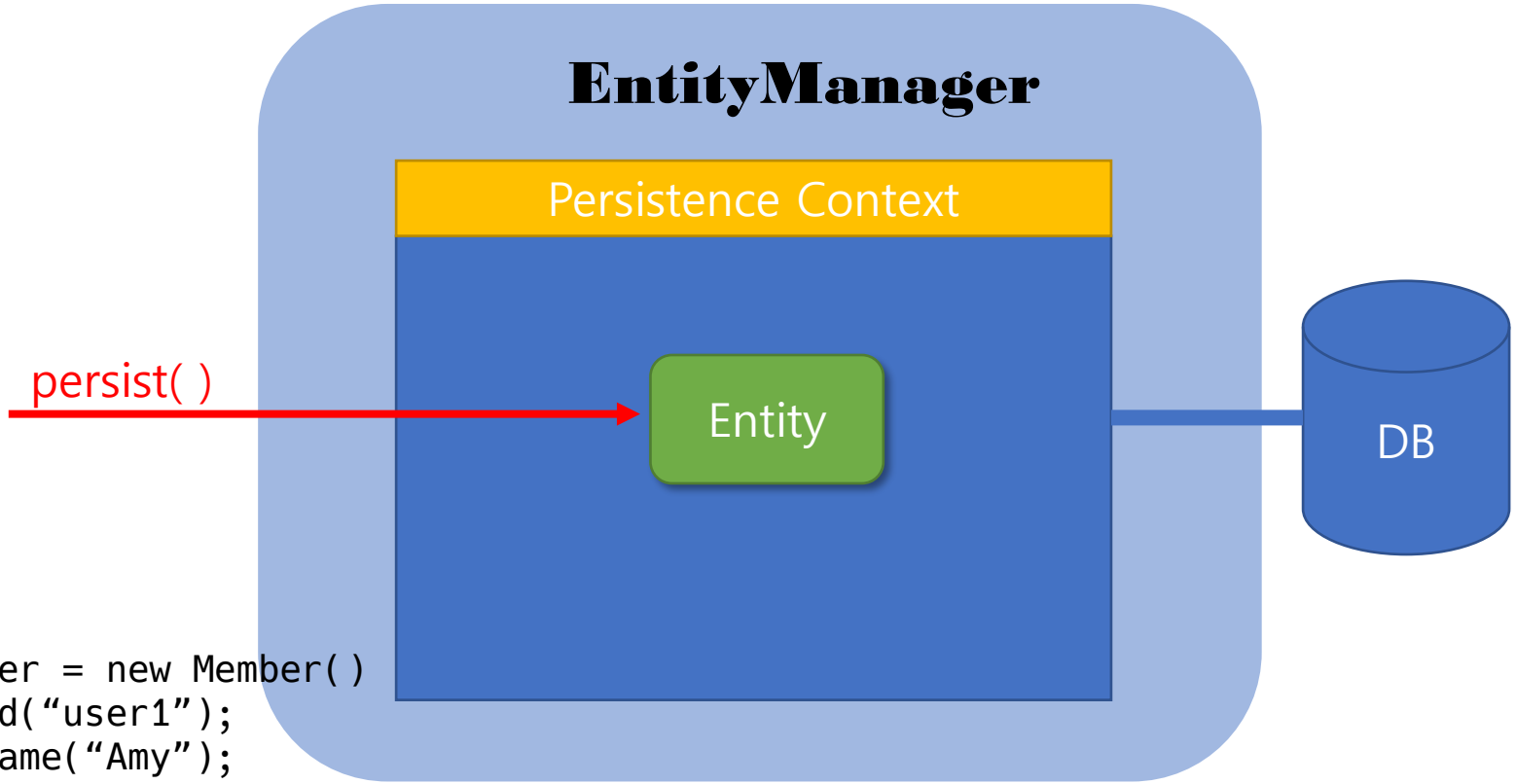


비영속 상태(New)



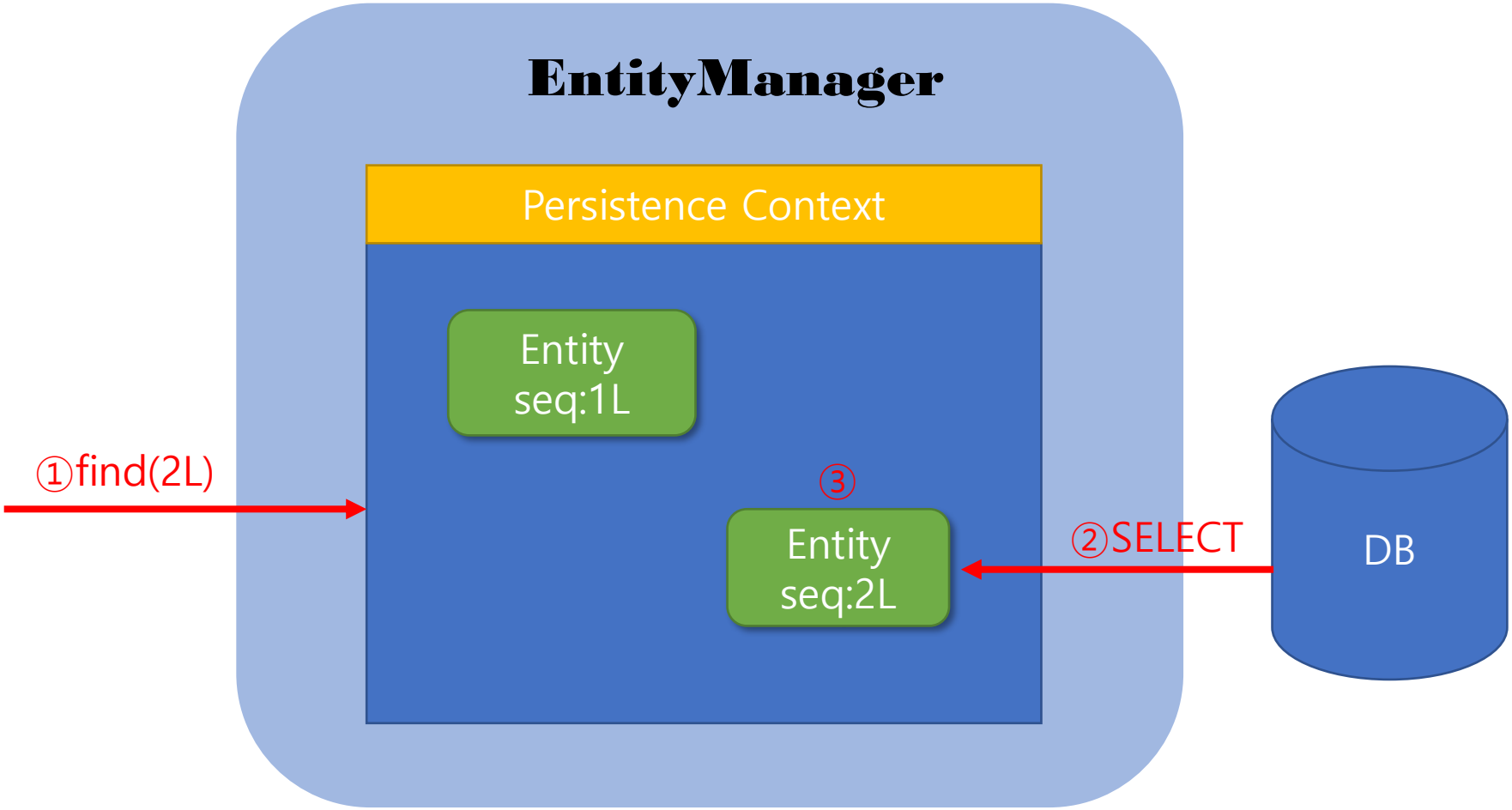

```
//엔티티 생성(비영속)
Member member = new Member()
member.setId("user1");
member.setName("Amy");
```

영속 상태(Managed)

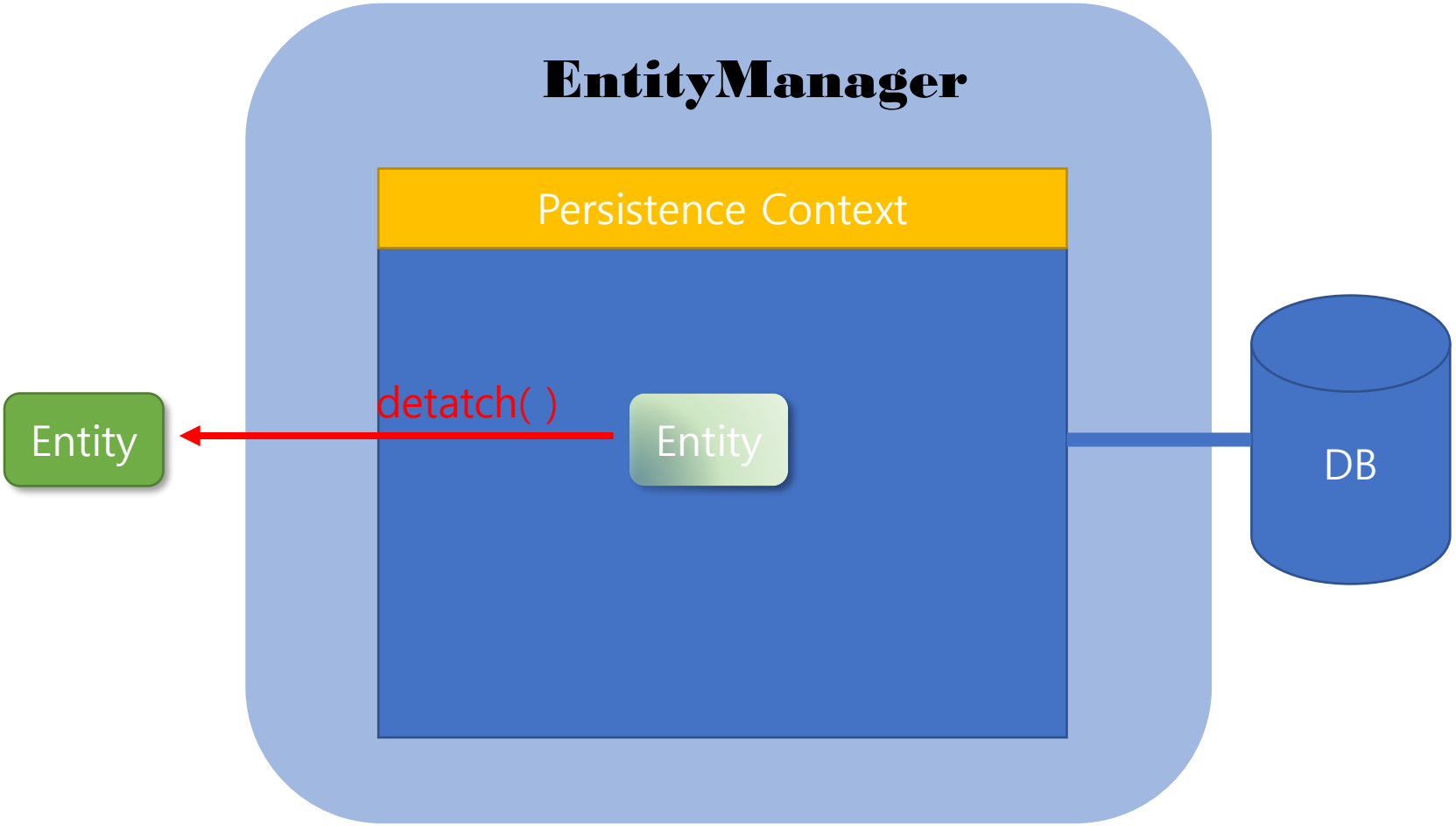



```
//엔티티 생성
Member member = new Member()
member.setId("user1");
member.setName("Amy");

//트랜잭션 시작
tx.begin();
//회원 등록 요청
em.persist(member)
```

영속 상태(Managed)

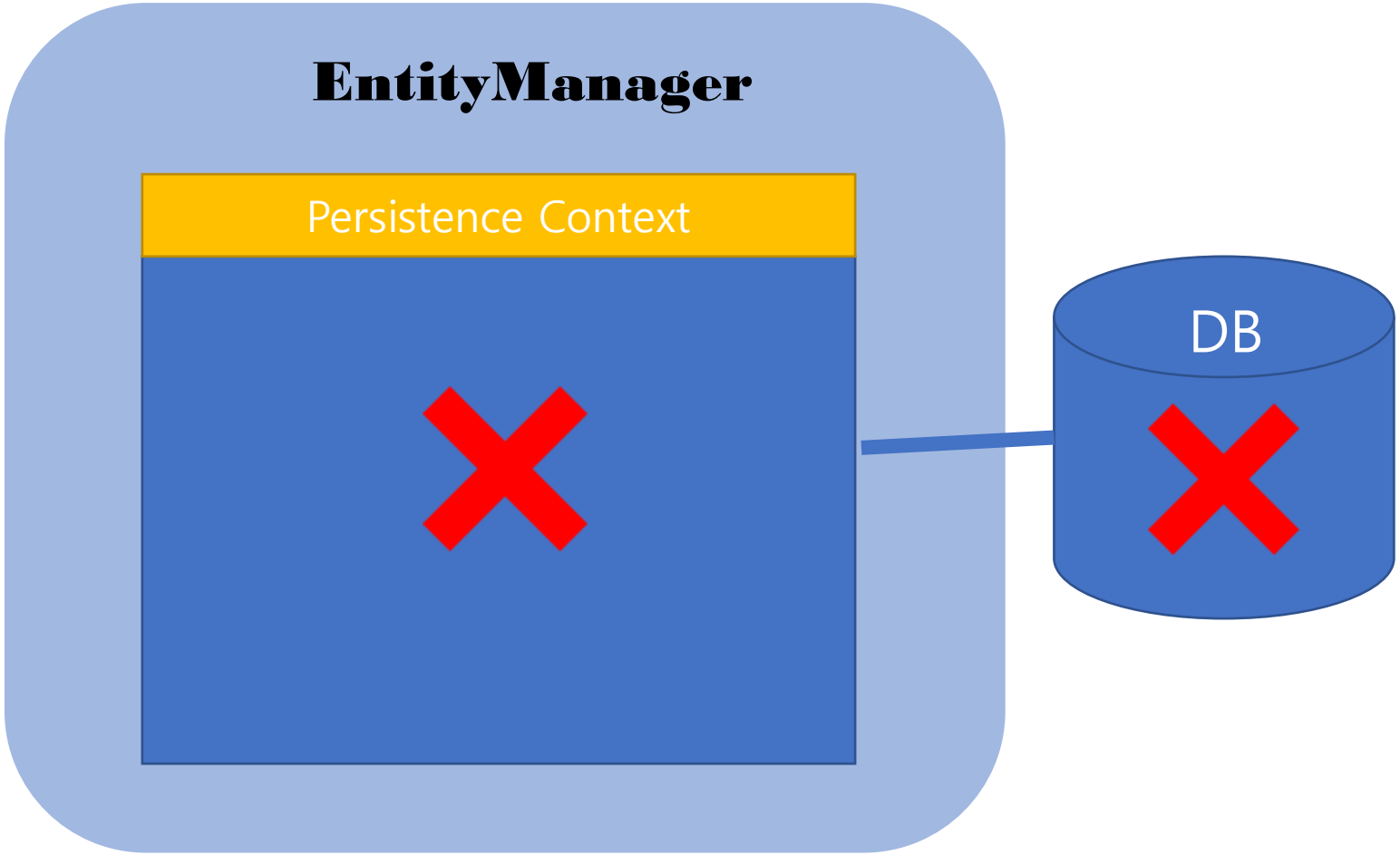


준영속 상태(Detatched)



detatch(entity) : 특정 엔티티만 준영속 상태로 전환한다.
clear() : 영속성컨텍스트를 초기화한다. 영속성컨텍스트가 관리하던 엔티티들 모두 삭제된다.
close() : 영속성컨텍스트를 종료한다. 영속성컨텍스는 종료되기 직전에 자신이 관리하던 엔티티들을 모두 삭제한다.

삭제(Removed)

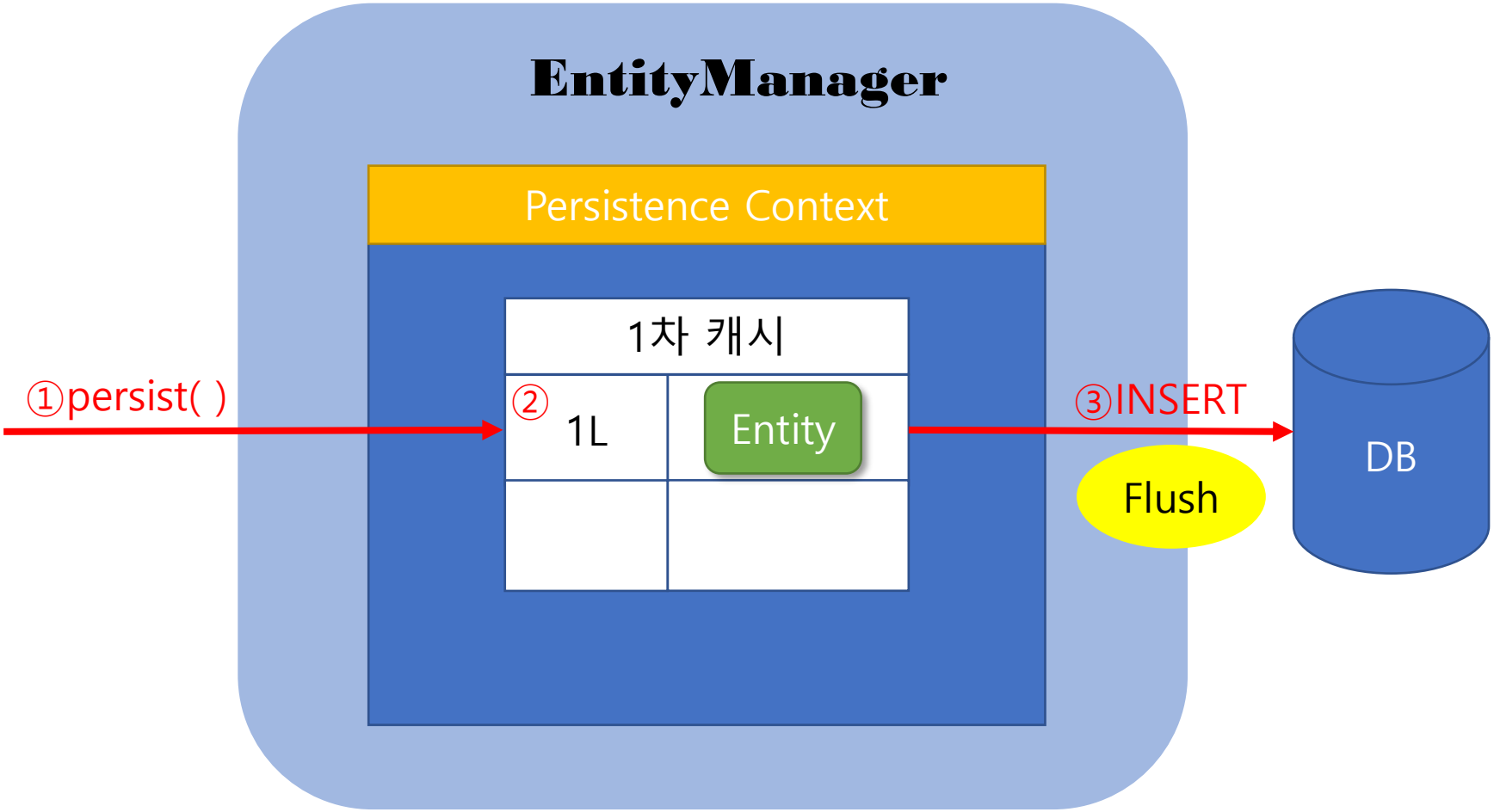



```
//객체 삭제
em.remove(member)
```

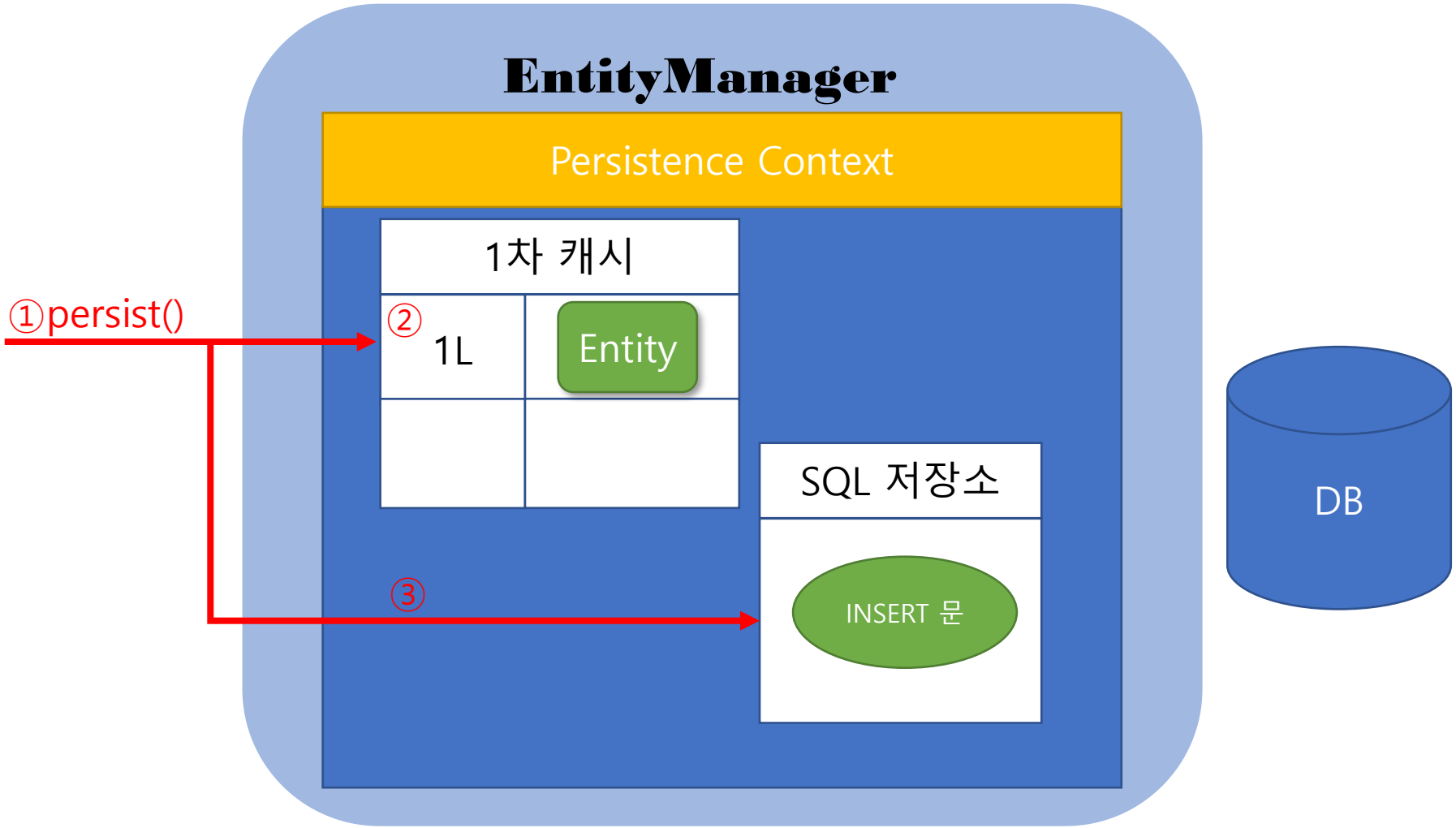
영속성 컨텍스트의 장점

- 1차 캐시
- 동일성(identity) 보장
- 트랜잭션을 지원하는 쓰기 지연(transactional write-behind)
- 변경 감지(Dirty Checking)
- 지연 로딩(Lazy Loading)

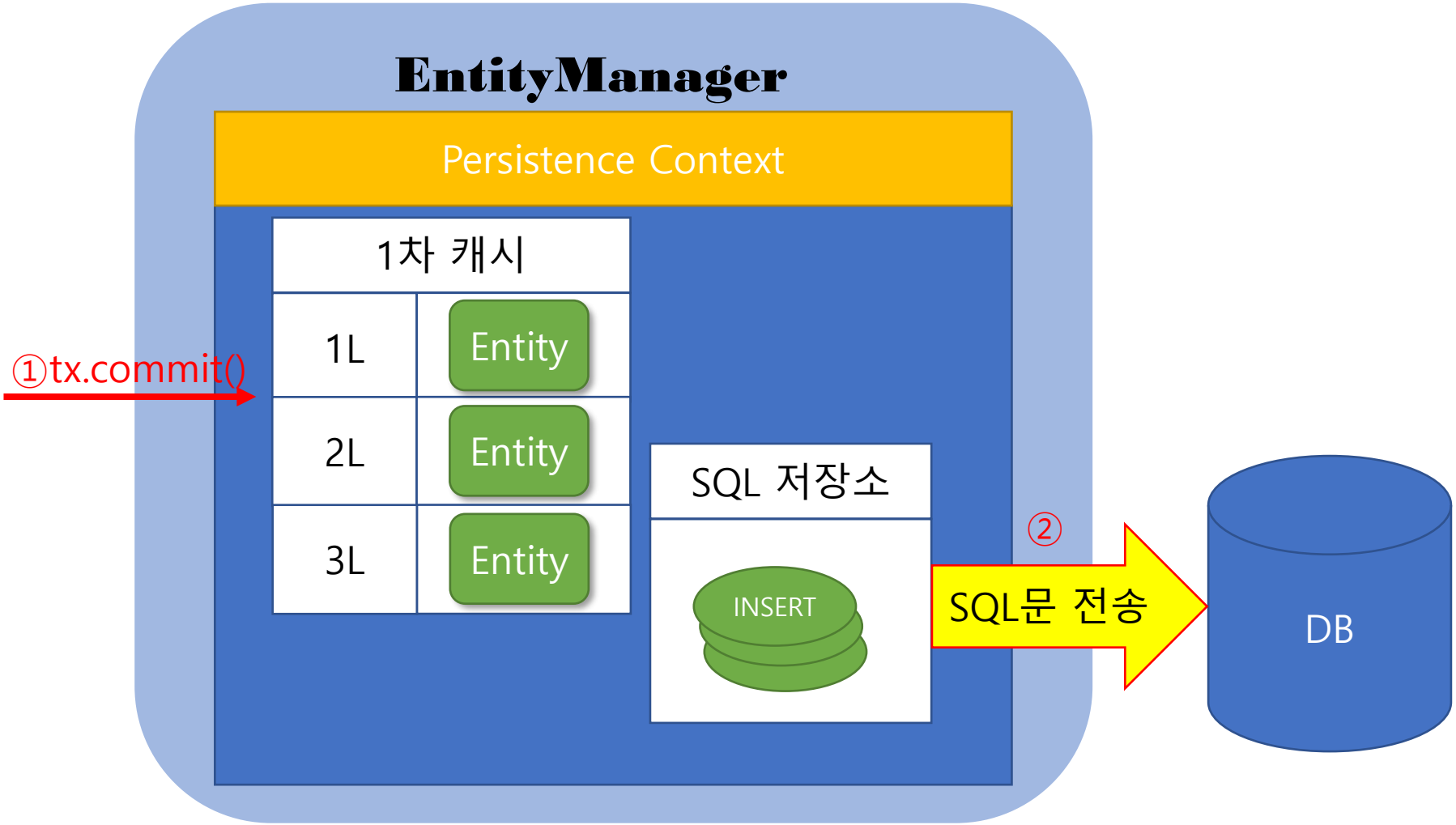
영속성 컨텍스트와 1차 캐시



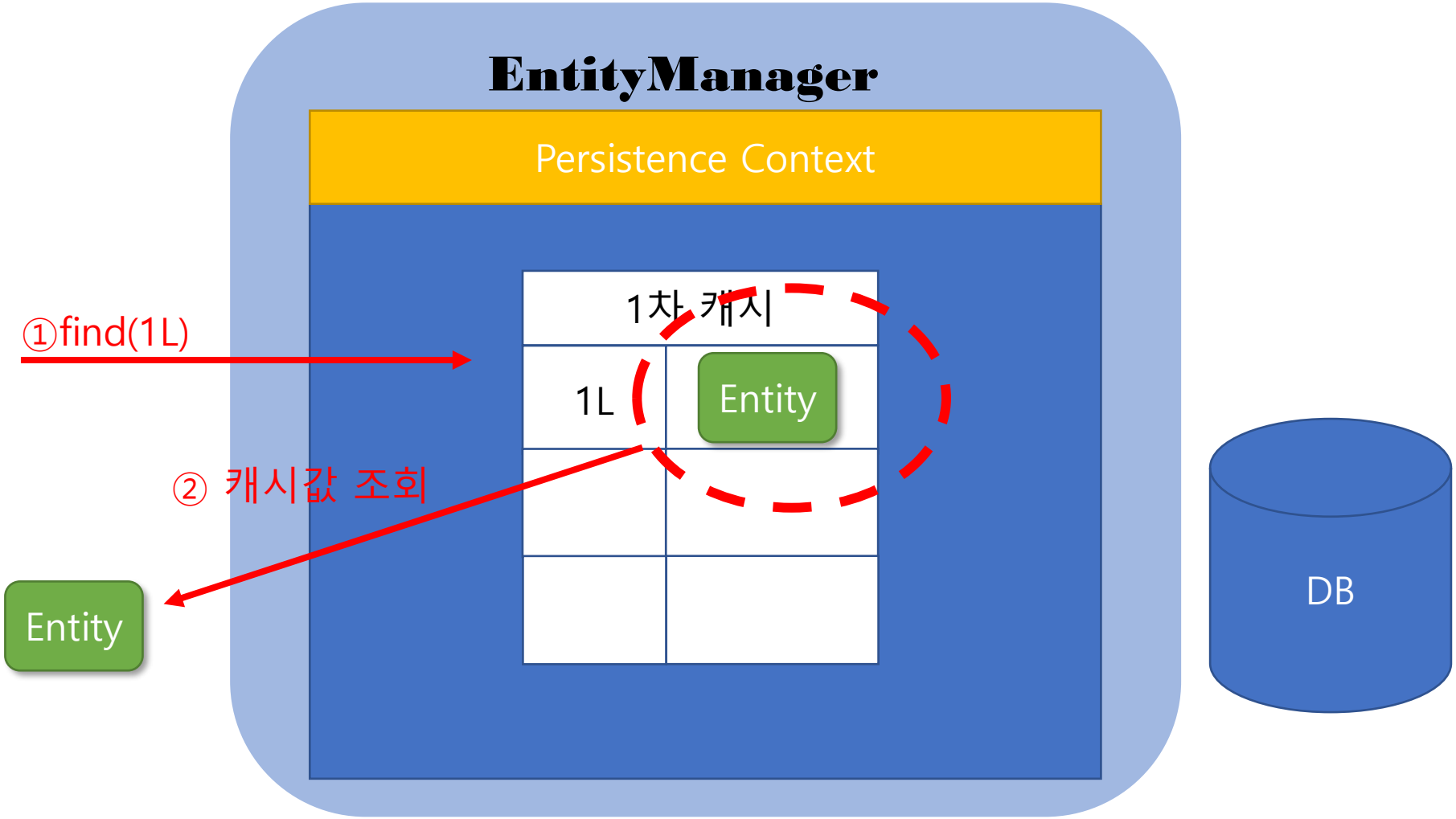
1차 캐시와 SQL 저장소



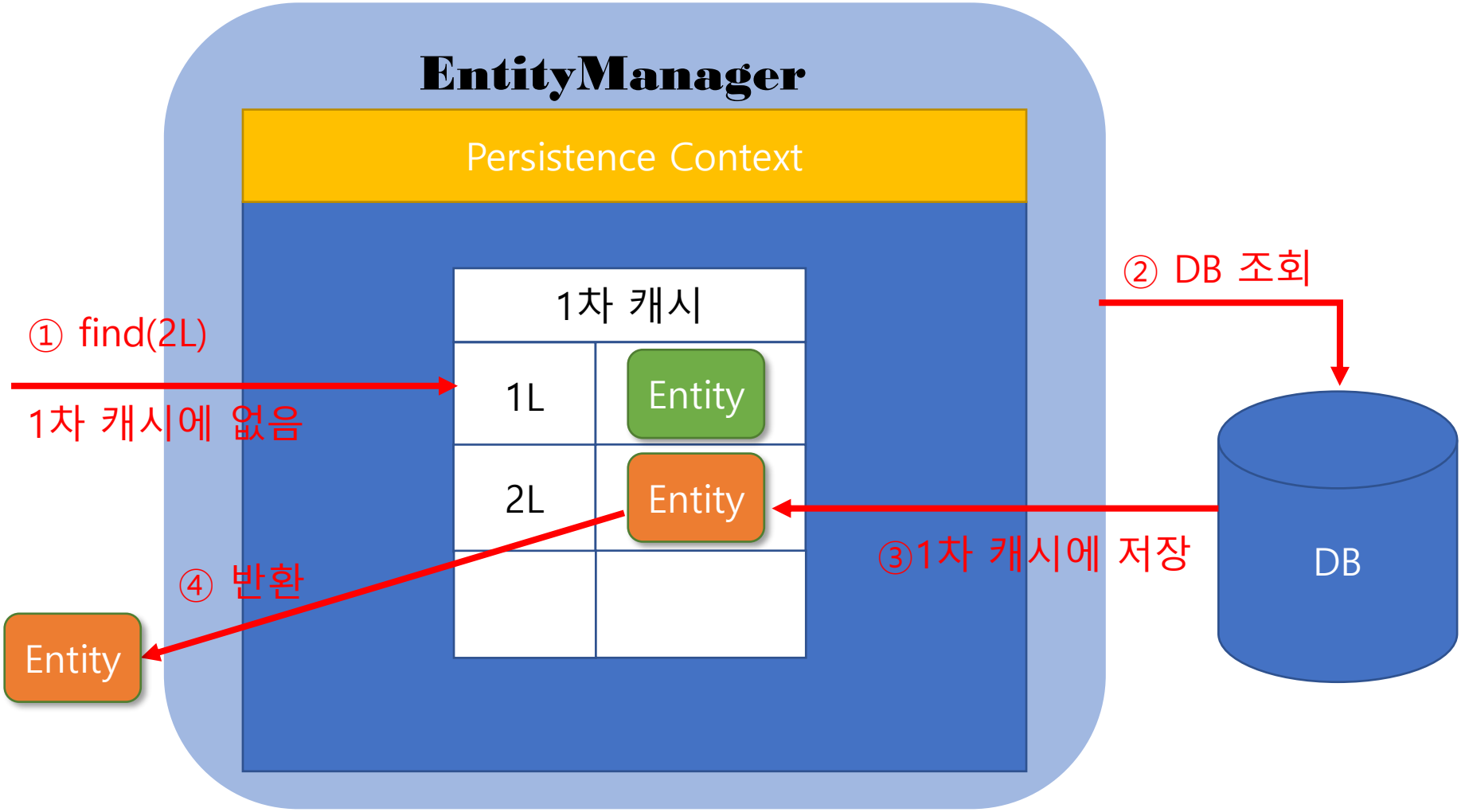
1차 캐시와 SQL 저장소



1차 캐시에서 조회


EntityManager
Persistence Context
1차 캐시
1L
Entity
①find(1L)
② 캐시값 조회
Entity
DB

데이터베이스에서 조회


EntityManager
Persistence Context
1차 캐시
1L
Entity
2L
Entity
① find(2L)
1차 캐시에 없음
② DB 조회
③1차 캐시에 저장
④ 반환
Entity
DB

영속 엔티티의 동일성 보장

```
Member m1 = em.find(Member.class, 1L);
Member m2 = em.find(Member.class, 1L);

System.out.println(a == b) // 동일성 비교 true
```

트랜잭션을 지원하는 쓰기 지연

```
transaction.begin()

em.persist(memberA);
em.persist(memberB);
//여기까지 INSERT SQL문을 DB에 보내지 않는다.

transactin.commit();
//커밋하는 순간 DB에 INSERT SQL을 보낸다.
```

엔티티 수정

```
transaction.begin()

//엔티티 조회
Member member = em.find(1L)

//엔티티 수정
member.setName("홍길동");
member.setAge(23)

//DB에 UPDATE SQL을 보낸다.
transaction.commit();
```

엔티티 수정

- 수정할 엔티티는 반드시 영속성 컨텍스트에 존재해야 함 그렇지않으면 예외 발생
- 수정전 검색해서 영속성컨텍스트에 등록
- 영속성 컨텍스트에 엔티티 등록시 복사본을 스냅샷(Snapshot)에 저장
- 트랜잭션 종료시 1차 캐시의 엔티티와 스냅샷의 엔티티를 비교하여 UPDATE 구문 생성
- JPA 기본 전략은 UPDATE 시 모든 필드 수정

엔티티 삭제

- 삭제할 엔티티는 반드시 영속성 컨텍스트에 존재해야 함 그렇지않으면 예외 발생
- 삭제전 검색해서 영속성컨텍스트에 등록
- 엔티티 삭제시 작업 순서
  - 영속성 컨텍스트에서 제거
  - DELETE 문이 SQL 저장소에 등록
  - 트랜잭션 종료시 DELETE문이 DB에 전송

ENTITY

JPA

BoardEntity 클래스

```
package com.edu.entity;

import java.time.LocalDateTime;

import org.hibernate.annotations.DynamicUpdate;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.jpa.domain
                                .support.AuditingEntityListener;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "board")
@EntityListeners(AuditingEntityListener.class)
@DynamicUpdate
public class BoardEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long seq;

    private String title;
    private String writer;
    private String content;

    @Column(name = "regdate",
            columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
    @CreatedDate
    private LocalDateTime regDate;

    @Column(columnDefinition = "INT DEFAULT 0")
    private int cnt;
}
```

@EntityListeners(AuditingEntityListener.class)

@EntityListeners

- JPA 엔티티 클래스에 대한 이벤트 리스너를 지정하는 어노테이션이다. 이를 통해 엔티티의 생명주기 이벤트(예: 생성, 수정, 삭제)에 반응하는 메서드를 포함하는 리스너 클래스를 등록할 수 있다.

감사(Auditing) 기능

- AuditingEntityListener는 스프링 데이터 JPA의 감사 기능을 활성화한다. 이를 통해 엔티티가 생성되거나 수정될 때 자동으로 타임스탬프와 사용자 정보를 기록할 수 있다.

엔티티 클래스

```java
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.Id;
import java.time.LocalDateTime;

@Entity
@EntityListeners(AuditingEntityListener.class)
public class MyEntity {

    @Id
    private Long id;

    @CreatedDate
    private LocalDateTime createdDate;

    @LastModifiedDate
    private LocalDateTime lastModifiedDate;

}
```

감사 기능 활성화

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
@EnableJpaAuditing
public class MyApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

ENTITY 클래스 어노테이션

어노테이션	설명
@Entity	해당 클래스가 엔티티임을 명시한다. 클래스 자체는 테이블과 일대일로 매칭되며, 인스턴스는 하나의 레코드를 의미
@Table	클래스와 테이블의 이름이 다를때 사용@Table(name=값)
@DynamicUpdate	실제 값이 변경된 컬럼으로만 UPDATE 구문을 생성
@SequenceGenerator	

속성	설명	기본값
name	식별자 생성기 이름	필수
sequenceName	DB에 등록되어 있는 시퀀스 이름	hibernate_sequence
initialValue	시퀀스 DDL생성시 시작 값	1
allocationSize	시퀀스 증가값	50
catalog, schema	DB catalog, schema이름	

https://www.datanucleus.org/products/accessplatform/jpa/annotations.html

ENTITY 필드 어노테이션

어노테이션	설명
@Id	PK에 해당하는 필드에 지정. 반드시 지정해야 함
@GeneratedValue	@Id 필드의 값 지정 방식(strategy=GenerationType.값)

값	설명	DBMS
AUTO	DB Dialect에 따라 자동 지정 (기본값)	
IDENTITY	DB에서 값 생성 (auto increment 방식)	MYSQL
SEQUENCE	DB의 Sequence을 사용해 값 생성, @SequenceGenerator 필요(시퀀스 생성)	ORACLE
TABLE	키 생성용 테이블 사용, @TableGenerator 필요	모든 DBMS

ENTITY 필드 어노테이션

<table>
<tr><th>어노테이션</th><th>설명</th></tr>
<tr><td>@Column</td><td>엔티티 필드와 테이블 컬럼을 매핑할 때 사용.
<table>
<tr><th>속성</th><th>Type</th><th>Description</th><th>Default</th></tr>
<tr><td>name</td><td>String</td><td>컬럼이름</td><td>필드이름</td></tr>
<tr><td>unique</td><td>boolean</td><td>유니크 여부</td><td>false</td></tr>
<tr><td>nullable</td><td>boolean</td><td>null 허용 여부</td><td>false</td></tr>
<tr><td>insertable</td><td>boolean</td><td>등록 가능 여부</td><td>true</td></tr>
<tr><td>updateable</td><td>boolean</td><td>수정 가능 여부</td><td>true</td></tr>
<tr><td>columnDefinition</td><td colspan="3">컬럼에 대한 DDL문을 직접 기술</td></tr>
<tr><td>length</td><td>int</td><td>컬럼 사이즈</td><td>255</td></tr>
<tr><td>precision</td><td>int</td><td>소수점 포함 전체 자릿수</td><td>19</td></tr>
<tr><td>scale</td><td>int</td><td>소수점 이하 자리수</td><td>2</td></tr>
</table>
</td></tr>
</table>

ENTITY 필드 어노테이션

어노테이션	설명
@Enumerated	자바의 enum타입을 매핑한다
@Temporal	날짜 타입(java.util.Date, java.util.Calendar) 매핑. 자바8에서 지원하는 LocalDate, LocalDateTime을 사용할때는 생략 가능 (하이버네이트 지원) TemporalType.DATE, TemporalType.TIME, Temporal.TIMESTAMP
@Transient	DB와 상관없이 개발자가 필요에 의해 메모리에서만 사용 DB에 저장, 조회 되지 않음

REPOSITORY

Spring Data JPA

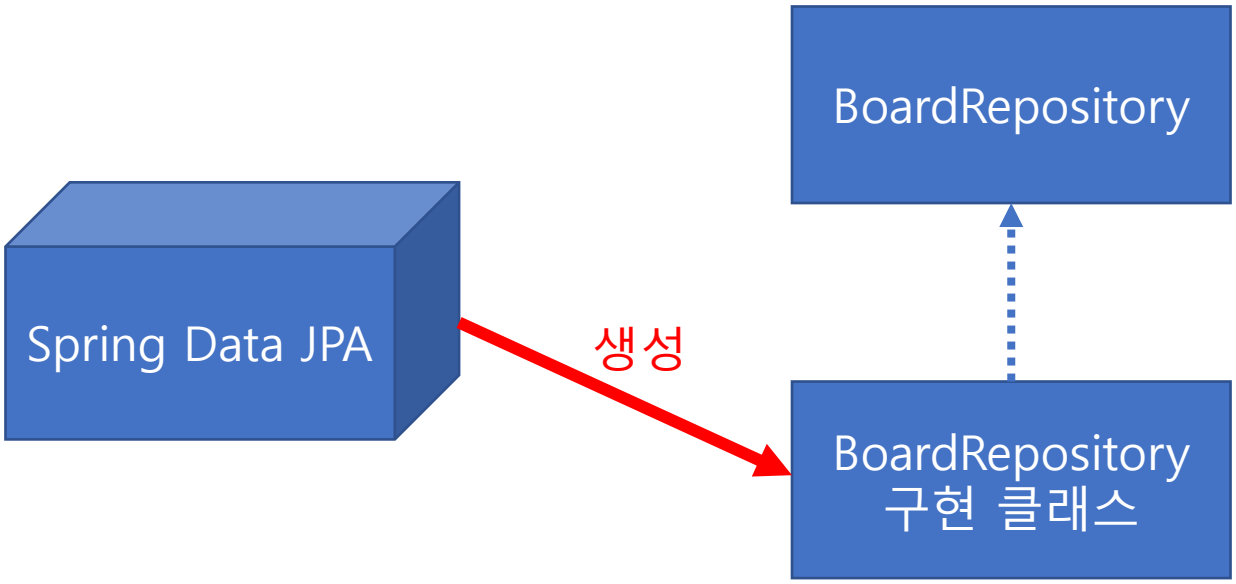
Spring Data JPA

- Spring Boot에서 JPA를 쉽게 사용할 수 있도록 지원하는 모듈
  - ✓JPA 를 사용하는데 필요한 라이브러리나 XML 설정 지원
  - ✓JPA 객체(EntityManagerFactory, EntityManger, EntityTransaction) 지원
- JPA 개념이나 동작원리를 모르고도 쉽게 JPA를 사용할 수 있음
  - ✓But JPA 동작원리를 모르면 개발시 발생되는 문제들을 해결할 수 없음
- 개발시 객체 구현없이 인터페이스 정의만 함
  - ✓정의된 인터페이스 구현 객체는 자동으로 생성됨

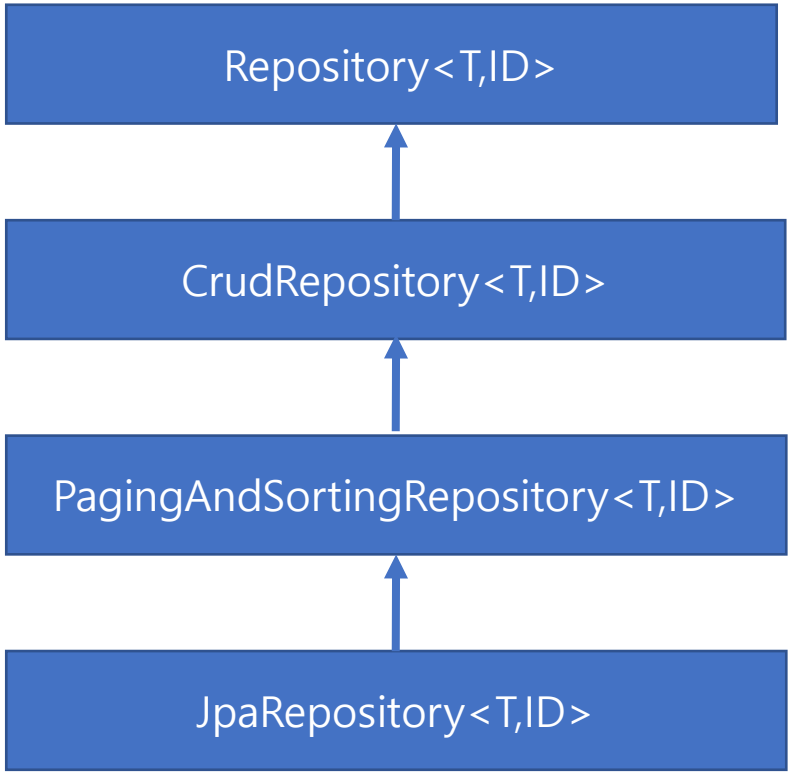
Spring Data JPA

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
public interface BoardRepository extends JpaRepository<BoardEntity, Long> {


}
```



JpaRepository 상속 구조




```
org.springframework.data.repository.Repository
org.springframework.data.repository.CrudRepository
org.springframework.data.repository.PagingAndSortingRepository
org.springframework.data.jpa.repository.JpaRepository
```

T : 엔티티의 클래스 타입
ID : 식별자타입(@ID로 매핑한 식별자 변수의 타입

JpaRepository 상속 구조

Repository<T, ID>

- 모든 리포지토리 인터페이스의 기본 부모 인터페이스
- 모든 리포지토리에 공통된 기본 기능을 정의
- 메서드

  CRUD(Create, Read, Update, Delete) 메서드 없이 마커 인터페이스 역할

CrudRepository<T, ID>

- 기본 CRUD 기능을 제공하는 리포지토리 인터페이스
- 엔티티에 대한 생성, 읽기, 업데이트, 삭제 기능을 제공
- 메서드
  - ❖save(S entity)
  - ❖findById(ID id)
  - ❖existsById(ID id)
  - ❖findAll()
  - ❖count()
  - ❖deleteById(ID id)
  - ❖delete(S entity)
  - ❖deleteAll(Iterable<? extends S> entities)
  - ❖deleteAll()

PagingAndSortingRepository<T, ID>

- 페이징 및 정렬 기능을 추가로 제공하는 리포지토리 인터페이스
- CRUD 기능 외에도 페이징 및 정렬 기능을 제공
- 메서드
  - ❖Iterable<T> findAll(Sort sort)
  - ❖Page<T> findAll(Pageable pageable)

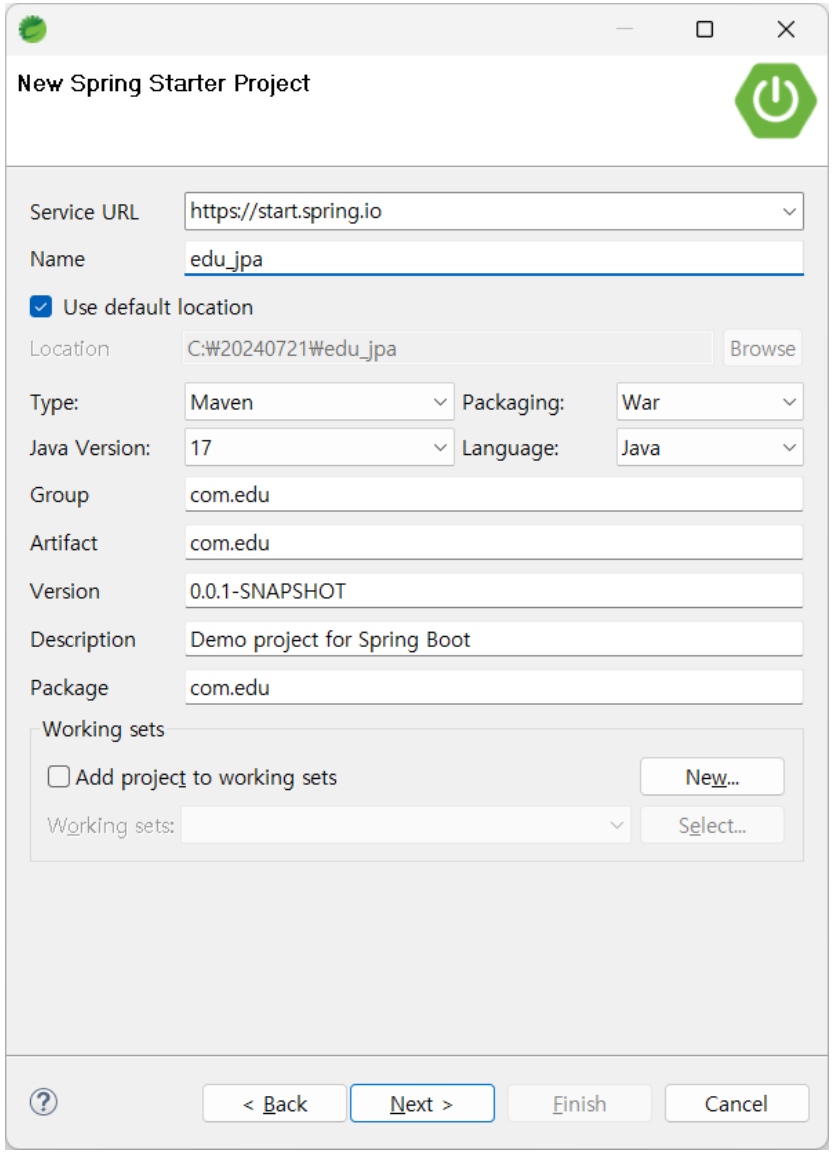
JpaRepository<T, ID>

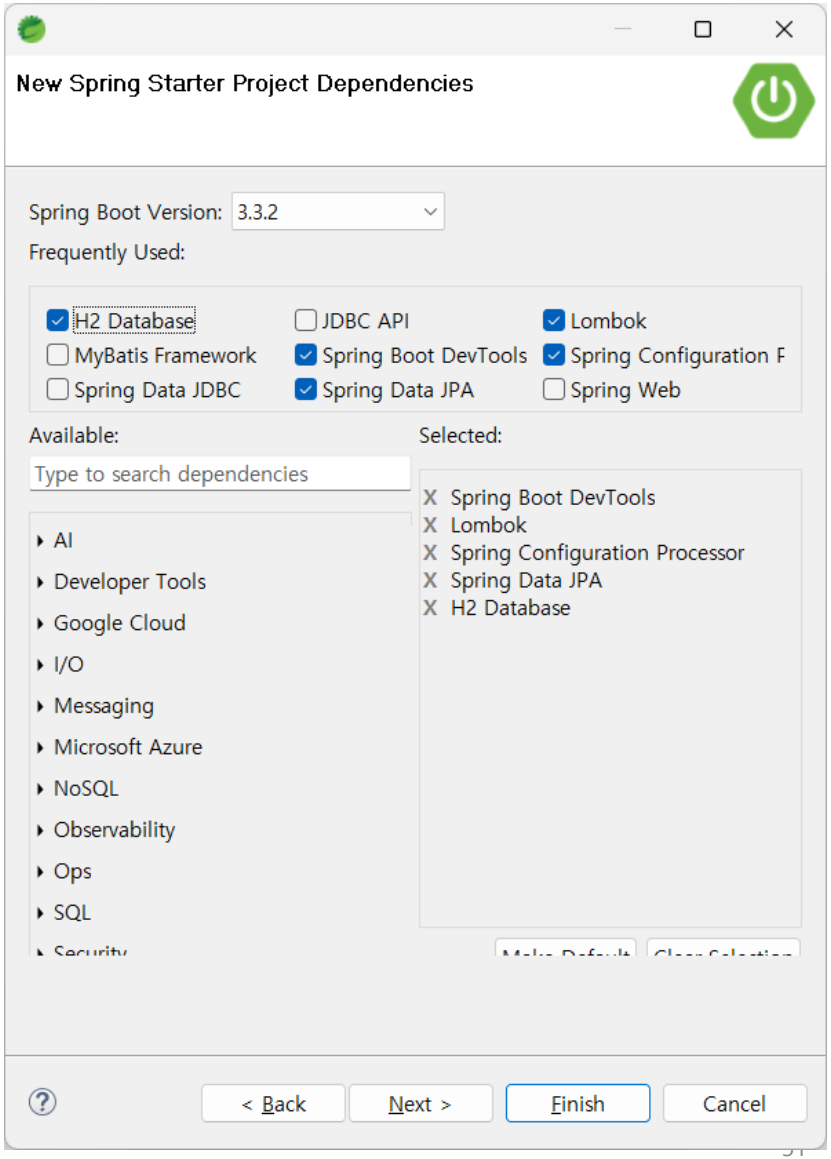
- JPA를 위한 확장 리포지토리 인터페이스
- CRUD, 페이징, 정렬 기능 외에도 JPA 관련 추가 기능을 제공
- 메서드
  - ❖List<T> findAll()
  - ❖List<T> findAll(Sort sort)
  - ❖List<T> findAllById(Iterable<ID> ids)
  - ❖void flush()
  - ❖S saveAndFlush(S entity)
  - ❖void deleteInBatch(Iterable<T> entities)
  - ❖void deleteAllInBatch()
  - ❖T getOne(ID id)

실습

Spring Data JPA

New → Spring Starter Project





application.properties

```
spring.application.name=edu_jpa

#H2
spring.datasource.url=jdbc:h2:tcp://localhost/~/test
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
# H2 콘솔 활성화 설정, 이 속성을 true로 설정하면 H2 데이터베이스의 웹 콘솔을 활성화합니다.
spring.h2.console.enabled=true

# H2 콘솔 경로 설정, 이 속성은 H2 콘솔의 웹 접근 경로를 설정합니다.
# 기본값은 '/h2-console'이며, 설정된 경로를 통해 브라우저에서 H2 콘솔에 접근할 수 있습니다.
spring.h2.console.path=/h2-console

# Hibernate SQL 쿼리 로깅 활성화
spring.jpa.show-sql=true

# SQL 쿼리를 포맷팅하여 가독성을 높임
spring.jpa.properties.hibernate.format_sql=true
```

com.edu.entity.BoardEntity

```
package com.edu.entity;

import java.time.LocalDateTime;

import org.hibernate.annotations.DynamicUpdate;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EntityListeners;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "board")
@EntityListeners(AuditingEntityListener.class)
@DynamicUpdate
```

```java
public class BoardEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long seq;

    private String title;
    private String writer;
    private String content;

    @Column(name = "regdate",
       columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
    @CreatedDate
    private LocalDateTime regDate;

    @Column(columnDefinition = "INT DEFAULT 0")
    private int cnt;
}
```

EduJpaApplication

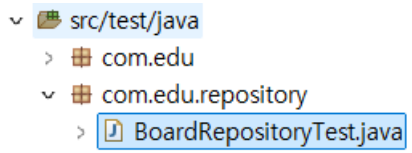
```java
package com.edu;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
@EnableJpaAuditing
public class EduJpaApplication {

    public static void main(String[] args) {
        SpringApplication.run(EduJpaApplication.class, args);
    }

}
```

com.edu.repository.BoardRepository

```
package com.edu.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.edu.entity.BoardEntity;

public interface BoardRepository extends
                            JpaRepository<BoardEntity, Long> {

}
```

src/test/java/com.edu.repository.BoardRepositoryTest

```
package com.edu.repository;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;

import com.edu.entity.BoardEntity;

@ExtendWith(SpringExtension.class)
@SpringBootTest
public class BoardRepositoryTest {
```



```java
@Autowired
private BoardRepository boardRepository;

@Test
@DisplayName("게시글 등록이 정상 동작한다")
public void insertBoardTest() {
    BoardEntity board = BoardEntity.builder()
                        .title("JPA")
                        .writer("이순신")
                        .content("자바 표준 ORM 스펙입니다")
                        .build();

    BoardEntity saveBoard = boardRepository.save(board);

    assertThat(saveBoard.getTitle()).isEqualTo(board.getTitle());
}
```

```java
@Test
@DisplayName("게시글 추출이 정상 동작한다.")
public void getBoardTest() {
    Optional<BoardEntity> board = boardRepository.findById(1L);
    assertTrue(board.isPresent());
}

@Test
@DisplayName("게시글 수정이 정상 동작한다")
public void updateBoardTest() throws Exception {
    BoardEntity board = boardRepository.findById(1L)
                        .orElseThrow(Exception::new);

    board.setTitle("AWS2");
    board.setContent("아마존 클라우드 서비스");

    BoardEntity savedBoard = boardRepository.save(board);

    assertThat(savedBoard.getTitle()).isEqualTo("AWS");
}
```

```
    @Test
    @DisplayName("게시글 전체 조회가 정상 동작한다")
    public void getBoardListTest() {
        List<BoardEntity> boardList = boardRepository.findAll();
        boardList.forEach(System.out::println);
    }

    @Test
    @DisplayName("게시글 삭제가 정상 동작한다")
    public void deleteBoardTest() {
        boardRepository.deleteById(1L);
        boolean exists = boardRepository.existsById(1L);
        assertFalse(exists);
    }
}
```

쿼리문 실행 방법

- JpaRepository 메소드 실행
- Query Method 선언후 실행
- @Query에 JPQL문 작성후 실행
- Querydsl 작성후 실행

Query Method

Spring Data JPA

Query Methods

이름 생성 규칙 : 키워드+엔티티이름+By+변수이름

- 엔티티이름 생략시 Repository 인터페이스에 선언된 엔티티 타입으로 자동 지정

find, read, get, query, search, stream ~ By

- 조회 처리
- 리턴 타입 : Collection 하위객체, Stream 하위객체

exists~By

- 특정 데이터가 존재하는지 확인
- 리턴 타입 : Boolean

Query Methods

count~By

- 조회 쿼리를 수행한 후 쿼리 결과로 나온 레코드수 리턴
- 리턴 타입 : long

delete~By, remove~By

- 삭제 쿼리 수행
- 리턴 타입 : void 또는 삭제한 횟수

~First<숫자>~, ~Top<숫자>~

- 쿼리를 통해 조회된 결과값의 개수를 제한. 두개 동일
- 리터 타입 : Collection 하위객체

https://docs.spring.io/spring-data/jpa/reference/repositories/query-keywords-reference.html

Supported keywords inside method names

Keyword	Sample	JPQL snippet
Distinct	findDistinctByLastnameAndFirstname	select distinct … where x.lastname = ?1 and x.firstname = ?2
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	… where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	… where x.age not null

Supported keywords inside method names

Keyword	Sample	JPQL snippet
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstname) = UPPER(?1)

com.edu.repository.BoardRepository

```
package com.edu.repository;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import com.edu.entity.BoardEntity;

public interface BoardRepository extends JpaRepository<BoardEntity, Long> {


    List<BoardEntity> findByWriter(String writer);
    List<BoardEntity> findByTitleContaining(String title);
    List<BoardEntity> findByTitleOrContent(String title, String content);
    List<BoardEntity> findByCntGreaterThan(int cnt);
    List<BoardEntity> findByCntBetween(int start, int end);
    List<BoardEntity> findByWriterIn(List<String> writers);


    boolean existsByWriter(String writer);
    long countByWriter(String writer);
    void deleteByWriter(String writer);
    long removeByWriter(String writer);
}
```

src/test/java/com.edu.repository.BoardRepositoryTest

```
@Test
public void createBoardList() {
    Random random = new Random();
    String[] names= {"홍길동","이순신","유관순"};
    for (int i = 1; i <= 10; i++) {
        BoardEntity board = new BoardEntity();
        board.setTitle("테스트 제목 " + i);
        board.setWriter(names[random.nextInt(3)]);
        board.setContent("테스트 내용 " + i);
        board.setCnt(random.nextInt(20));
        boardRepository.save(board);
    }
}

@Test
@DisplayName("작성자(writer) 조건으로 조회가 정상 동작한다.")
public void findByWriterTest() {
    this.createBoardList();
    List<BoardEntity> boardList = boardRepository.findByWriter("홍길동");
    boardList.forEach(System.out::println);
}

@Test
@DisplayName("제목(title) 일부 조건으로 조회가 정상 동작한다.")
public void findByTitleContainingTest() {
    List<BoardEntity> boardList = boardRepository.findByTitleContaining("테스트");
    boardList.forEach(System.out::println);
}
```

```java
@Test
@DisplayName("제목(title) or 내용(content) 조건으로 조회가 정상 동작한다.")
    public void findByTitleOrContentContainingTest() {
    List<BoardEntity> boardList =
            boardRepository.findByTitleOrContent("테스트 제목 1", "테스트 내용 2");
    boardList.forEach(System.out::println);
}

@Test
@DisplayName("조회수(cnt) 크기 비교 조건으로 조회가 정상 동작한다.")
public void findByCntGreaterThanTest() {
    List<BoardEntity> boardList = boardRepository.findByCntGreaterThan(5);
    boardList.forEach(System.out::println);
}

@Test
@DisplayName("조회수(cnt) 범위 지정 조건으로 조회가 정상 동작한다.")
public void findByCntBetweenTest() {
    List<BoardEntity> boardList = boardRepository.findByCntBetween(5,10);
    boardList.forEach(System.out::println);
}

@Test
@DisplayName("작성자(writer) In 조건으로 조회가 정상 동작한다.")
public void findByWriterInTest() {
    List<String> writers = Arrays.asList("이순신","유관순");
    List<BoardEntity> boardList = boardRepository.findByWriterIn(writers);
    boardList.forEach(System.out::println);
}
```

```java
@Test
@DisplayName("exists~By : 특정 데이터가 존재하는지 확인한다")
public void existsByWriterTest() {
        assertTrue(boardRepository.existsByWriter("홍길동"));
}

@Test
@DisplayName("조회 쿼리를 수행한 후 쿼리 결과로 나온 레코드수를 리턴한다")
public void countByWriterTest() {
        assertThat(boardRepository.countByWriter("홍길동")).isEqualTo(1);
}

@Test
@Transactional
@DisplayName("삭제쿼리 수행. 리턴값이 없거나 삭제한 횟수 리턴한다")
public void deleteByWriterTest() {
    boardRepository.deleteByWriter("이순신");
    assertThat(boardRepository.existsByWriter("이순신")).isFalse();
}

@Test
@Transactional
@Rollback(false)
@DisplayName("삭제쿼리 수행. 리턴값이 없거나 삭제한 횟수 리턴한다")
public void deleteByWriterTest2() {
    boardRepository.deleteByWriter("이순신");
    assertThat(boardRepository.existsByWriter("이순신")).isFalse();
}
```

데이터 정렬

- 메소드 이름에 키워드로 지정
- OrderBy + 변수 + Asc or Desc

```
List<BoardEntity>  findByWriterOrderByCntDesc(String writer);
```

```
@Test
@DisplayName("조회수(cnt) 내림차순 조건으로 조회가 정상 동작한다.")
public void findByWriterOrderByCntDescTest() {
    List<BoardEntity> boardList =
                            boardRepository.findByWriterOrderByCntDesc("홍길동");
    boardList.forEach(System.out::println);
}
```

데이터 정렬

- 매개 변수로 지정
- org.springframework.data.domain.Sort

```
List<BoardEntity> findByWriter(String writer, Sort sort);
```

```
@Test
@DisplayName("Sort 매개변수 정렬 조건이 정상 동작한다.")
public void findWriter() {
    List<BoardEntity> boardList =
    boardRepository.findByWriter("홍길동", Sort.by(Sort.Direction.DESC, "cnt"));
    boardList.forEach(System.out::println);
}

@Test
@DisplayName("Sort 매개변수 2차 정렬 조건이 정상 동작한다.")
public void findAllTest() {
    List<BoardEntity> boardList =
    boardRepository.findAll(Sort.by(Sort.Order.desc("cnt"), Sort.Order.asc("seq")));
    boardList.forEach(System.out::println);
}
```

페이징 처리

org.springframework.data.domain

Interface Pageable

All Known Implementing Classes:

`AbstractPageRequest`, `PageRequest`, `QPageRequest`

org.springframework.data.domain

Class PageRequest

java.lang.Object
    org.springframework.data.domain.AbstractPageRequest
        org.springframework.data.domain.PageRequest

All Implemented Interfaces:

`Serializable`, `Pageable`

`static PageRequest`	**`of`**`(int page, int size)` Creates a new unsorted **`PageRequest`**.
`static PageRequest`	**`of`**`(int page, int size, Sort.Direction direction, String... properties)` Creates a new **`PageRequest`** with sort direction and properties applied.
`static PageRequest`	**`of`**`(int page, int size, Sort sort)` Creates a new **`PageRequest`** with sort parameters applied.

페이징 처리

org.springframework.data.domain

Interface Page<T>

Type Parameters:

`T` -

All Superinterfaces:

`Iterable<T>, Slice<T>, Streamable<T>, Supplier<Stream<T>>`

Modifier and Type	Method and Description
`static <T> Page<T>`	**`empty()`** Creates a new empty **`Page`**.
`static <T> Page<T>`	**`empty(Pageable pageable)`** Creates a new empty **`Page`** for the given **`Pageable`**.
`long`	**`getTotalElements()`** Returns the total amount of elements.
`int`	**`getTotalPages()`** Returns the number of total pages.
`<U> Page<U>`	**`map(Function<? super T,? extends U> converter)`** Returns a new **`Page`** with the content of the current one mapped by the given **`Function`**.

페이징 처리

```
Page<BoardEntity>  findAll(Pageable pageable);
```

```
@Test
@DisplayName("Paging 처리 조건이 정상 동작한다.")
public void findAllPagingTest() {
    //Page<BoardEntity> pages = boardRepository.findAll(PageRequest.of(0, 5));
    Page<BoardEntity> pages =
       boardRepository.findAll(PageRequest.of(0, 5, Sort.Direction.DESC, "cnt"));
    pages.forEach(System.out::println);

    System.out.println(pages);
}
```

@Query

Spring Data JPA

@Query

어노테이션에 JPQL 쿼리문 작성

- SQL문과 거의 유사한 문법 사용
- 테이블이 아니라 엔티티 객체를 대상으로 함

복잡한 쿼리문 작성

- 기본 API와 쿼리 메소드에서 처리되지 않는 쿼리문 작성

성능향상을 위한 쿼리문 작성

- N+1과 같은 현상을 방지

Object[] 리턴

- 쿼리 후 엔티티 객체가 아닌 Object[ ] 타입을 리턴 받음
- Object[ ]에는 추출된 컬럼들 포함

Native SQL 처리

- nativeQuery=true 속성 지정시 DB 고유의 SQL 사용

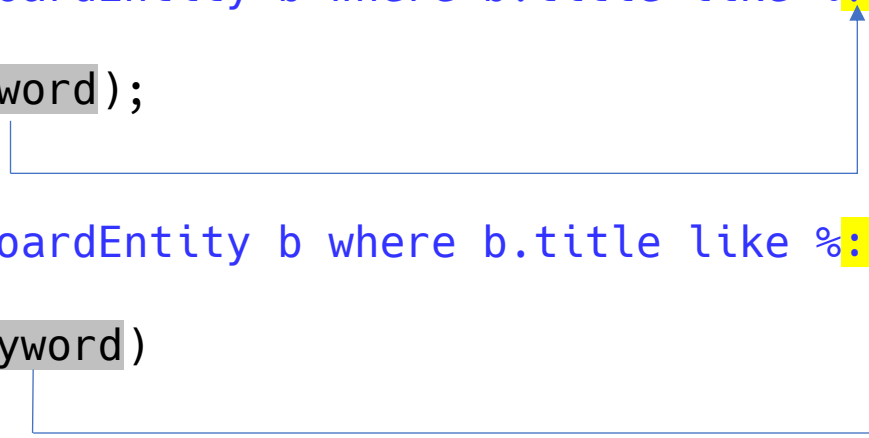
JPQL 구문

- 엔티티와 속성은 대소문자 구분함
  - Member, age
- JPQL 키워드는 대소문자 구분 안함
  - SELECT, FROM, where
- 테이블 이름이 아닌 엔티티 이름 사용
  - Member
- 엔티티 별칭은 필수(m)

```
select m from Member m where m.age > 18
```

파라미터 바인딩- 이름 기준, 위치 기준

```
@Query("select b from BoardEntity b where b.title like %?1%")
getBoardList(String keyword);

@Query("select b from BoardEntity b where b.title like %:keyword%")
getBoardList3(String keyword)
```



JPQL 기본 함수 및 연산자

CONCAT
SUBSTRING
TRIM
LOWER, UPPER
LENGTH
LOCATE
ABS, SQRT, MOD
SIZE, INDEX(JPA 용도)

- 서브 쿼리 지원
- EXISTS, IN
- BETWEEN, LIKE, IS NULL

조인

내부 조인

SELECT m FROM Member m [INNER] JOIN m.team t

외부 조인

SELECT m FROM Member m LEFT [OUTER] JOIN m.team t

세타 조인

select count(m) from Member m, Team t where m.username = t.name

```
@Query("select b
        from BoardEntity b
        where b.title like %:keyword%
        order by b.cnt desc")
List<BoardEntity> boardQuery1(String keyword);

@Query(value = "select seq, title, writer from board
                where title like '%'||?1||'%'
                order by seq desc",
       nativeQuery = true)
List<Object[]> boardQuery2(String keyword);
```

```
@Test
public void boardQuery1Test() {
    List<BoardEntity> boardList = boardRepository.boardQuery1("테스트");
    boardList.forEach(System.out::println);
}

@Test
public void boardQuery2Test() {
    List<Object[]> boardList = boardRepository.boardQuery2("테스트");
    boardList.forEach((row) -> {
        System.out.println(Arrays.toString(row));
    });
}
```

연관 관계 매핑

Spring Data JPA

다대일(N:1) 매핑

```java
@Getter
@Setter
@ToString
@Entity
public class Member {

	@Id
	@Column(name = "MEMBER_ID")
	private String id;
	private String password;
	private String name;
	private String role;

}
```

```java
@Setter
@Getter
@ToString
@Entity
@Table(name = "board2")
@EntityListeners(AuditingEntityListener.class)
public class Board {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long seq;
	private String title;
	private String content;

	@ManyToOne
	@JoinColumn(name = "MEMBER_ID")
	private Member member;
}
```

다대일(N:1) 매핑

```
public interface BoardRepository2 extends JpaRepository<Board, Long> {
}
```

```
public interface MemberRepository extends JpaRepository<Member, String> {
}
```

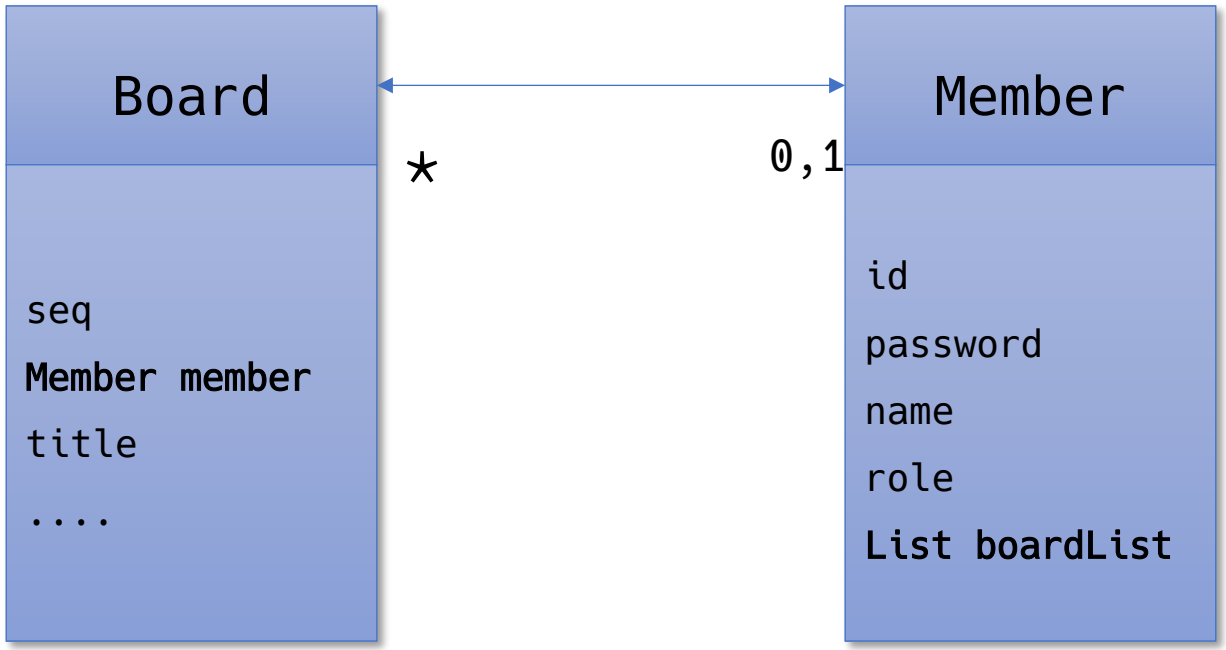
다대일(N:1) 매핑

```java
@ExtendWith(SpringExtension.class)
@SpringBootTest
public class RelationMappingTest {
	@Autowired
	private BoardRepository2 boardRepo;

	@Autowired
	private MemberRepository memberRepo;

	@Test
	@DisplayName("게시글 등록 테스트")
	public void testManyToOneInsert() {
		Member user1 = new Member();
		user1.setId("user1");
		user1.setPassword("1111");
		user1.setName("홍길동");
		user1.setRole("member");
		memberRepo.save(user1);

		Member user2 = new Member();
		user2.setId("user2");
		user2.setPassword("2222");
		user2.setName("이순신");
		user2.setRole("admin");
		memberRepo.save(user2);

		for (int i = 1; i <= 3; i++) {
			Board board = new Board();
			board.setMember(user1);
			board.setTitle("홍길동 제목 " + i);
			board.setContent("홍길동 내용 " + i);
			boardRepo.save(board);
		}

		for (int i = 1; i <= 3; i++) {
			Board board = new Board();
			board.setMember(user2);
			board.setTitle("이순신 제목" + i);
			board.setContent("이순신 내용" + i);
			boardRepo.save(board);
		}

}
```

```java
@Test
@DisplayName("게시글 상세 조회")
public void testManyToOneSelect() {
    Board board = boardRepo.findById(1L).get();
    System.out.println(board);
}
```

```
Hibernate:
    select
        b1_0.seq,
        b1_0.content,
        m1_0.member_id,
        m1_0.name,
        m1_0.password,
        m1_0.role,
        b1_0.title
    from
        board2 b1_0
    left join
        member m1_0
            on m1_0.member_id=b1_0.member_id
    where
        b1_0.seq=?
Board(seq=1, title=홍길동 제목 1, content=홍길동 내용 1, member=Member(id=user1,
password=1111, name=홍길동, role=member))
```

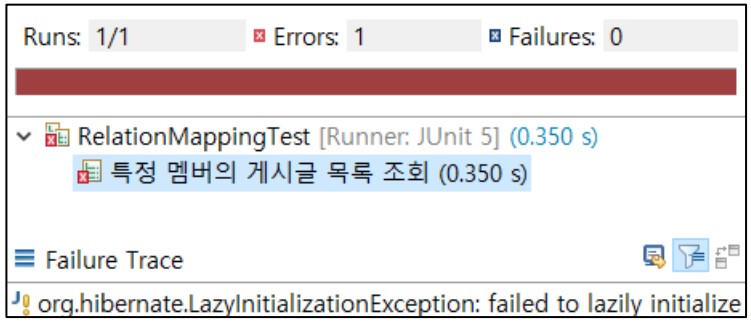
양방향 매핑


Board
seq
Member member
title
....
*
0,1
Member
id
password
name
role
List boardList

```java
package com.edu.entity;
~ 생략 ~

@Getter
@Setter
@ToString
@Entity
public class Member {
    @Id
    @Column(name = "MEMBER_ID")
    private String id;
    private String password;
    private String name;
    private String role;

    @OneToMany(mappedBy = "member")
    private List<Board> boardList = new ArrayList<>();
}
```

```java
@Test
@DisplayName("특정 멤버의 게시글 목록 조회")
public void testTwoWayMapping() {
    Member member = memberRepo.findById("user1").get();
    List<Board> boardList = member.getBoardList();
    boardList.forEach(System.out::println);
}
```

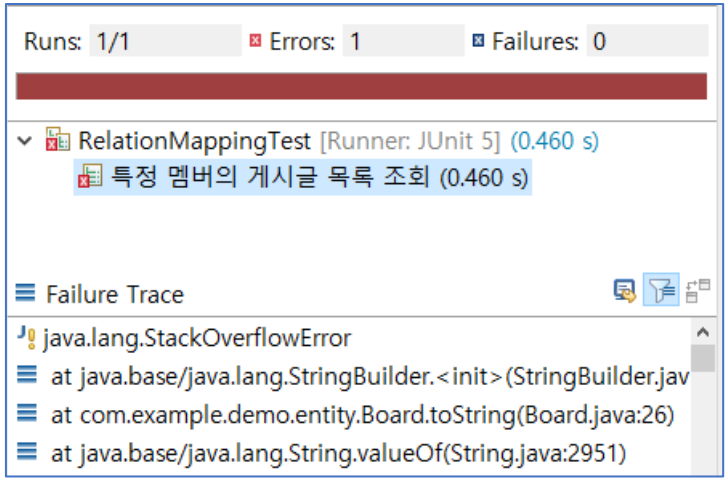
```
Hibernate:
    select
        m1_0.member_id,
        m1_0.name,
        m1_0.password,
        m1_0.role
    from
        member m1_0
    where
        m1_0.member_id=?
```



```java
package com.edu.entity;
~ 생략 ~

@Getter
@Setter
@ToString
@Entity
public class Member {
    @Id
    @Column(name = "MEMBER_ID")
    private String id;
    private String password;
    private String name;
    private String role;

    @OneToMany(mappedBy = "member", fetch = FetchType.EAGER)
    private List<Board> boardList = new ArrayList<>();
}
```

```java
@Test
@DisplayName("특정 멤버의 게시글 목록 조회")
public void testTwoWayMapping() {
    Member member = memberRepo.findById("user1").get();
    List<Board> boardList = member.getBoardList();
    boardList.forEach(System.out::println);
}
```

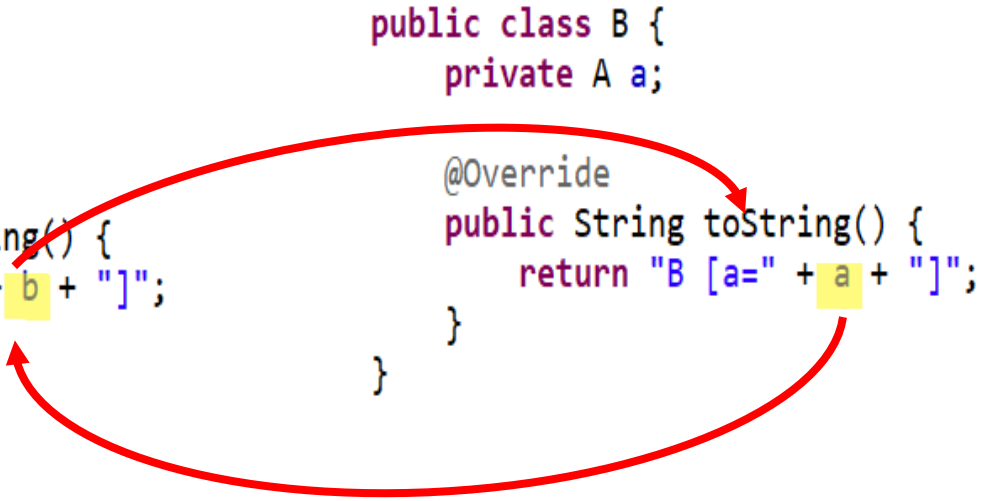
```
Hibernate:
    select
        m1_0.member_id,
        m1_0.name,
        m1_0.password,
        m1_0.role,
        bl1_0.member_id,
        bl1_0.seq,
        bl1_0.content,
        bl1_0.title
    from
        member m1_0
    left join
        board2 bl1_0
            on m1_0.member_id=bl1_0.member_id
    where
        m1_0.member_id=?
```



양방향 연관 관계시 주의점

```java
public class A {
    private B b;

    @Override
    public String toString() {
        return "A [b=" + b + "]";
    }
}
```

```java
public class B {
    private A a;

    @Override
    public String toString() {
        return "B [a=" + a + "]";
    }
}
```



```java
@Getter  @Setter
@ToString(exclude="boardList")
@Entity
public class Member {
    @Id
    @Column(name = "MEMBER_ID")
    private String id;
    private String password;
    private String name;
    private String role;

    @OneToMany(mappedBy = "member", fetch=FetchType.EAGER)
    private List<Board> boardList = new ArrayList<>();
}
```

```java
@Getter @Setter
@ToString(exclude="member")
@Entity
@Table(name = "board2")
@EntityListeners(AuditingEntityListener.class)
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long seq;
    private String title;
    private String content;

    @ManyToOne
    @JoinColumn(name = "MEMBER_ID", nullable = false)
    private Member member;
}
```

```
Hibernate:
    select
        m1_0.member_id,
        m1_0.name,
        m1_0.password,
        m1_0.role,
        bl1_0.member_id,
        bl1_0.seq,
        bl1_0.content,
        bl1_0.title
    from
        member m1_0
    left join
        board2 bl1_0
            on m1_0.member_id=bl1_0.member_id
    where
        m1_0.member_id=?
```

```
Board(seq=1, title=홍길동 제목 1, content=홍길동 내용 1)
Board(seq=2, title=홍길동 제목 2, content=홍길동 내용 2)
Board(seq=3, title=홍길동 제목 3, content=홍길동 내용 3)
```

영속성 전이

```
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long seq;
    private String title;
    private String content;

    @ManyToOne
    @JoinColumn(name = "MEMBER_ID")
    private Member member;

    public void setMember(Member member) {
        this.member = member;
        member.getBoardList().add(this);
    }
}
```

```java
@Test
@DisplayName("영속성 테스트")
public void testCascade() {
    Member user3 = new Member();
    user3.setId("user3");
    user3.setPassword("3333");
    user3.setName("유관순");
    user3.setRole("member");
    //memberRepo.save(user1);

    for (int i = 1; i <= 3; i++) {
        Board board = new Board();
        board.setMember(user3);
        board.setTitle("유관순 제목 " + i);
        board.setContent("유관순 내용 " + i);
        //boardRepo.save(board);
    }

    memberRepo.save(user3);
}
```

- member 테이블에 user3 유관순 데이터 삽입
- board2 테이블에 등록되는 데이터는 없음

영속성 전이를 이용하여 등록

```
@Getter
@Setter
@ToString(exclude = "boardList")
@Entity
public class Member {

    @Id
    @Column(name = "MEMBER_ID")
    private String id;
    private String password;
    private String name;
    private String role;

    @OneToMany(mappedBy = "member", fetch = FetchType.EAGER,
                                    cascade = CascadeType.ALL)
    private List<Board>  boardList = new ArrayList<>();

}
```

```java
@Test
@DisplayName("영속성 테스트")
public void testCascade2() {
    Member user4 = new Member();
    user4.setId("user4");
    user4.setPassword("4444");
    user4.setName("강감찬");
    user4.setRole("member");
    //memberRepo.save(user1);

    for (int i = 1; i <= 3; i++) {
        Board board = new Board();
        board.setMember(user4);
        board.setTitle("강감찬 제목 " + i);
        board.setContent("강감찬 내용 " + i);
        //boardRepo.save(board);
    }

    memberRepo.save(user1);
}
```

- member 테이블에 user4 강감찬 데이터 등록
- board2 테이블에 강감찬 작성글 등록

영속성 전이를 이용하여 삭제

```
@Test
@DisplayName("특정 멤버 삭제")
public void testCascadeDelete() {
    Member member = memberRepo.findById("user4").get();
    memberRepo.delete(member);
}
```

- member 테이블에 user4 강감찬 데이터 삭제됨
- board2 테이블에 강감찬 작성글 삭제됨

OneToOne 단방향 매핑

```java
@Getter @Setter @ToString
@Entity
public class Profile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String tel;
    private String address;

    @OneToOne
    @JoinColumn(name = "member_id", nullable = false)
    private Member member;

}
```

```java
public interface ProfileRepository extends JpaRepository<Profile, Long> {

}
```

OneToOne 단방향 매핑

```
@Autowired
ProfileRepository profileRepo;

@Test
public void testOneToOne() {
    Member member = memberRepo.findById("user1").get();

    Profile profile = new Profile();
    profile.setMember(member);
    profile.setTel("010-123-4567");
    profile.setAddress("서울 강남구 논현동");

    Profile savedProfile=profileRepo.save(profile);
    System.out.println(savedProfile);
}
```

```
Profile(id=1, tel=010-123-4567, address=서울 강남구 논현동,
                member=Member(id=user1, password=1111, name=홍길동, role=member))
```

OneToOne 양방향 매핑

```
@Getter @Setter
@ToString(exclude="boardList")
@Entity
public class Member {

    @Id
    @Column(name = "MEMBER_ID")
    private String id;
    private String password;
    private String name;
    private String role;

    @OneToMany(mappedBy = "member", fetch = FetchType.EAGER,
                                    cascade = CascadeType.ALL)
    private List<Board> boardList = new ArrayList<>();

    @OneToOne(mappedBy = "member")
    private Profile profile;

}
```

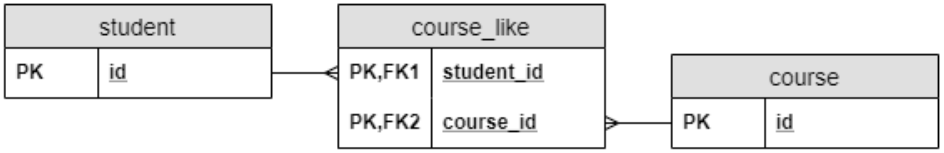
OneToOne 양방향 매핑

```java
@Getter
@Setter
@ToString(exclude = "member")
@Entity
public class Profile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    ~ 생략 ~
}
```

```java
@Test
public void testOneToOne2() {
    Member member = memberRepo.findById("user1").get();
    System.out.println(member);
}
```

```
Member(id=user1, password=1111, name=홍길동, role=member,
       profile=Profile(id=1, tel=010-123-4567, address=서울 강남구 논현동))
```

ManyToMany 연관관계 매핑




```java
@Entity
class Student {

    @Id
    Long id;

    @ManyToMany
    Set<Course> likedCourses;

    // additional properties
    // standard constructors, getters, and setters
}

@Entity
class Course {

    @Id
    Long id;

    @ManyToMany
    Set<Student> likes;

    // additional properties
    // standard constructors, getters, and setters
}
```

```java
@ManyToMany
@JoinTable(
  name = "course_like",
  joinColumns = @JoinColumn(name = "student_id"),
  inverseJoinColumns = @JoinColumn(name = "course_id"))
Set<Course> likedCourses;
```

ManyToMany 연관관계 매핑

```java
@Embeddable
class CourseRatingKey implements Serializable {

    @Column(name = "student_id")
    Long studentId;

    @Column(name = "course_id")
    Long courseId;

    // standard constructors, getters, and setters
    // hashcode and equals implementation
}
```

```java
@Entity
class CourseRating {

    @EmbeddedId
    CourseRatingKey id;

    @ManyToOne
    @MapsId("studentId")
    @JoinColumn(name = "student_id")
    Student student;

    @ManyToOne
    @MapsId("courseId")
    @JoinColumn(name = "course_id")
    Course course;

    int rating;

    // standard constructors, getters, and setters
}
```

ManyToMany 연관관계 매핑

```java
@Data
@Entity
@Table(name = "S_ORD")
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date orderDate;

    @ManyToMany(fetch = FetchType.EAGER)
    private List<Product> productList = new ArrayList<Product>();
}
```

```java
@Data
@Entity
@Table(name = "S_PRODUCT")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;

    public Product(String name) {
        this.name = name;
    }
}
```

```java
@Autowired
private ProductRepository productRepo;

@Autowired
private OrderRepository orderRepo;

@Test
public void testManyToMany() {
    // 상품 등록
    Product product1 = new Product("Galaxy S23");
    Product product2 = new Product("iPhone 14Pro");

    productRepo.save(product1);
    productRepo.save(product2);

    // 주문 등록
    Order order = new Order();
    order.setOrderDate(new Date());
    order.getProductList().add(product1);
    order.getProductList().add(product2);

    Order savedOrder = orderRepo.save(order);
    System.out.println(savedOrder);
}
```

```
Order(id=1, orderDate=Sat Feb 25 12:40:38 KST 2023,
      productList=[Product(id=1, name=Galaxy S23), Product(id=2, name=iPhone 14Pro)])
```