

3장 영속성 관리

3장 영속성 관리

[3.1 엔티티 매니저 팩토리와 엔티티 매니저](#)

[3.2 영속성 컨텍스트란?](#)

[3.3 엔티티의 생명주기](#)

[3.4 영속성 컨텍스트의 특징](#)

[3.5 플러시](#)

[3.6 준영속](#)

[3.7 정리](#)

3장 영속성 관리

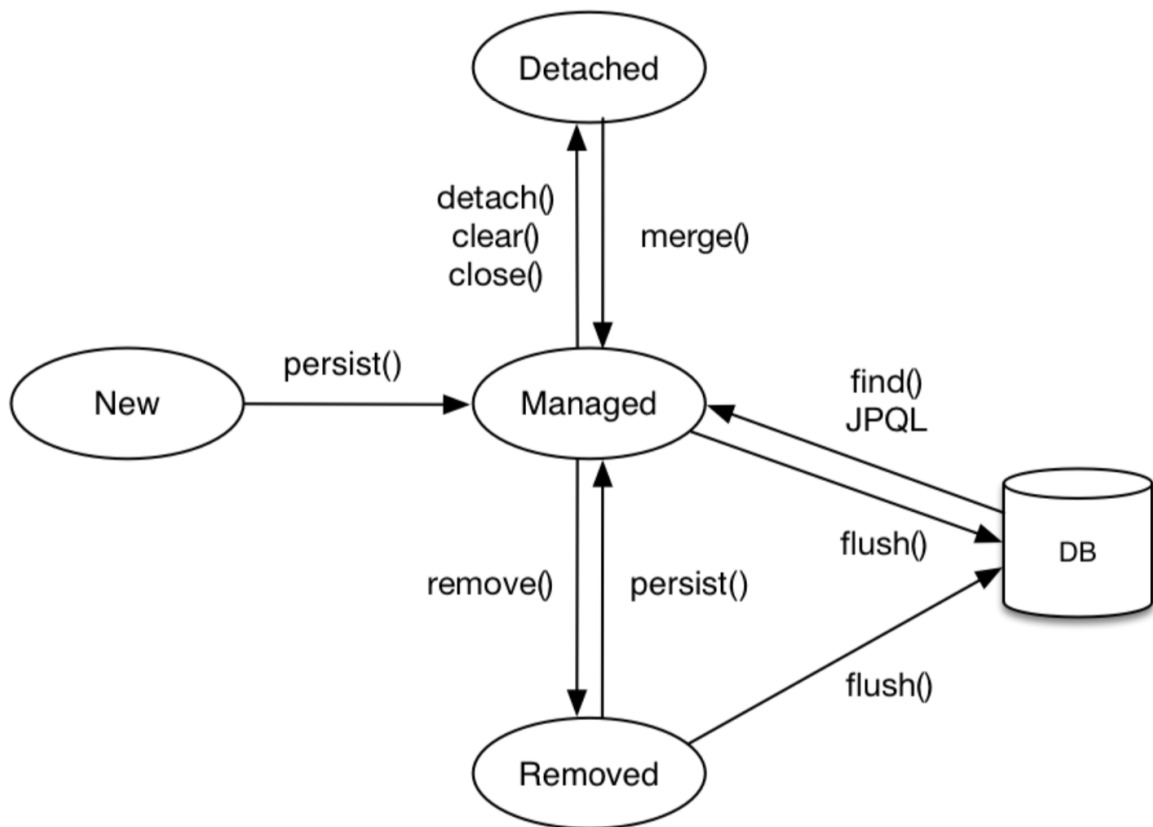
3.1 엔티티 매니저 팩토리와 엔티티 매니저

- 엔티티 매니저 팩토리
 - 엔티티 매니저를 만드는 공장, 비용 多, 한개만 만들어 애플리케이션 전체 공유하여 사용
 - 다른 스레드 간 공유 가능
- 엔티티 매니저
 - 엔티티와 관련된 모든 일 처리 (엔티티를 저장하는 가상의 데이터베이스 정도로 생각)
 - 여러 스레드가 동시 접근 시 동시성 문제 발생, 절대 스레드간 공유 하면 안 됨
 - 데이터베이스 연결이 꼭 필요한 시점 까지 DB 커넥션을 얻지 않음
 - 보통 트랜잭션 시작 시 커넥션 획득
 - J2SE (Standard Edition) 환경
 - EntityManagerFactory 생성 시 커넥션 풀도 함께 만든다.
 - J2EE(Enterprise Edition) 환경
 - 보통 SpringFramework 에서 제공 하는 데이터 소스를 사용

3.2 영속성 컨텍스트란?

- 영속성 컨텍스트
 - 엔티티를 영구 저장 하는 환경
 - ex) persist() 메소드의 경우 엔티티 매니저를 사용해 엔티티를 영속성 컨텍스트에 저장 함
 - Spring 환경에서 같은 트랜잭션일 경우 여러 엔티티 매니저가 같은 영속성 컨텍스트에 접근 할 수 있다.
 - 3.4장 특징 에서 좀 더 자세히 작성

3.3 엔티티의 생명주기



[그림1. 생명 주기]

- 비영속(new/transient) : 영속성 컨텍스트와 전혀 관계가 없는 상태
 - 영속성 컨텍스트나 데이터베이스와 전혀 관련 없는 상태
 - 객체만 생성한 상태

```

Member member = new Member();
member.setid("member1");
member.setUsername("회원1");

```

- 영속(managed) : 영속성 컨텍스트에 저장된 상태
 - em.persist(member); //객체를 저장한 상태
- 준영속(detached) : 영속성 컨텍스트에 저장되었다가 분리된 상태
 - em.detach(member); //회원 엔티티를 영속성 컨텍스트에서 분리
 - em.close() , em.clear() 작업 역시 관리중인 엔티티를 컨텍스트에서 분리
- 삭제(removed) : 삭제된 상태
 - em.remove(member); //삭제

3.4 영속성 컨텍스트의 특징

- 식별자 값 (@Id)이 반드시 있어야 한다.
- 트랜잭션을 커밋하는 순간 영속성 컨텍스트에 있는 엔티티를 데이터베이스에 반영 → 플러시
- 장점
 - 1차 캐시
 - 최초 엔티티를 영속성 컨테이너에 반영 했을 경우엔 데이터베이스에 바로 저장되지 않고 1차 캐시에 엔티티를 저장 한다.
 - em.find() 호출 시 1차 캐시에서 식별자 값 (@Id)으로 엔티티를 찾는다. 캐시에 데이터 존재 시 데이터베이스를 조회하지 않으며, 캐시에 데이터가 존재하지 않을 경우 데이터베이스를 조회하여 데이터를 찾은 뒤 1차 캐시에 데이터를 넣어 준다. (캐시를 먼저 조회하므로 성능상 이점을 누릴 수 있음)
 - 동일성 보장

```
Member a = em.find(Member.class, "member1");
Member b = em.find(Member.class, "member1");

a == b (성립)
```

- 같은 캐시에 있는 엔티티 인스턴스를 반환하므로 두 객체 a,b 의 값은 같다. 즉 동일성을 보장 한다.
 - 동일성 : 참조값을 비교하는 == 비교의 값이 같다.
 - 동등성 : 실제 인스턴스는 달라도 인스턴스가 가지고 있는 값이 같은 경우

```
Member a = new Member();
a.setId("member1");
Member b = new Member();
b.setId("member1");

a == b (성립 하지 않음, 객체 인스턴스가 다르다.)

자바에서 동등성 비교는 equals() 를 오버라이딩 하여 구현 해줘야 한다.
하지만 두 객체간 정확한 비교를 위해서는 hashCode() 역시 재정의 해줘야 하는 사실
-> hashCode() 비교 후 같은 해시값일 경우 equals()로 값 비교
```

- 트랜잭션 지원하는 쓰기 지연
 - 트랜잭션 커밋 전 까지 행위에 대한 쿼리를 영속 컨텍스트에 저장하고 있다가 모아둔 쿼리를 한번에 데이터베이스에 보내는 행위
- 변경 감지 (dirty checking)
 - 엔티티 수정 시
 - JPA는 엔티티를 영속성 컨텍스트에 보관 시 스냅샷을 보관한다. (최초 상태를 복사 후 저장)
 - 플러시 시점에서 스냅샷과 엔티티를 비교해 변경 된 엔티티를 찾는다.
 - 변경된 엔티티가 있으면 수정쿼리를 생성 해 쓰기지연 SQL 저장소에 보낸다.
 - 변경 감지는 영속성 컨텍스트가 관리 하는 영속 상태의 엔티티에만 적용 된다.
 - JPA 기본전략은 업데이트 시 엔티티의 모든 필드를 업데이트 한다.

- 업데이트 시 모든 필드를 업데이트 하는 이유
 - 수정쿼리가 항상 같아 수정 쿼리를 미리 생성 해 두고 재사용 할 수 있다.
 - 데이터베이스에 동일한 쿼리를 보내면 데이터베이스는 이전에 한 번 파싱 된 쿼리를 재사용 할 수 있다.
 - 필드가 너무 많을 시 @DynamicUpdate 어노테이션을 사용 시 수정된 데이터만 사용해 동적으로 업데이트 쿼리를 생성 한다. @DynamicInsert 도 있다.
- 지연 로딩(lazy loading)
 - 실제 객체 대신 프록시 객체를 로딩해두고 해당 객체를 실제 사용 할 때 영속성 컨텍스트를 이용

3.5 플러시

- 영속성 컨텍스트의 변경 내용을 데이터베이스에 반영한다.
- 플러시 하는 방법
 - em.flush() 를 직접 호출
 - 강제 플러시, 거의 사용 안함
 - → 강제 플러시를 이용하는 경우? : 테스트 시 한번에 많은 데이터를 넣을 경우 데이터베이스에 중간 저장하는 목적으로 강제 플러시를 사용 할 수 있다.
 - 트랜잭션 커밋시 플러시 자동 호출
 - DB 반영을 위해 트랜잭션 커밋전 플러시 자동 호출
 - JPQL 쿼리 실행 시 플러시 자동 호출
 - ex) 쿼리 실행 전 미리 플러시 하여 쿼리 결과로 조회 되도록 함
- 영속성 컨텍스트에 보관된 엔티티를 지우는것이 아님, 영속성 컨텍스트의 변경내용을 데이터베이스에 동기화 하는 것 → 다른 인스턴스로 계속 컨텍스트에 persist (지속) 시 EntityManager 내부에 있는 map 의 사이즈가 계속 커지게 된다.

3.6 준영속

- 영속상태에서 분리된 상태
- 영속 상태의 엔티티를 준영속 상태로 만드는 방법
 - em.detach(entity) : 특정 엔티티만 준영속 상태로 전환
 - 1차캐시 부터 쓰기 지연 SQL 저장소까지 관리중이던 엔티티 정보 모두 삭제
 - em.clear() : 영속성 컨텍스트 초기화
 - em.close() : 영속성 컨텍스트 종료
- 준영속 상태의 특징
 - 거의 비영속 상태에 가깝다.
 - 반드시 식별자 값을 가지고 있다.
 - 지연로딩을 할 수 없다.

- 준영속 상태의 엔티티를 다시 영속 상태로 만드는 방법
 - merge()
 - 준영속, 비영속을 신경쓰지 않고 병합 한다.
 - 식별자 값으로 엔티티 조회 가능 시 불러서 병합, 조회 할 수 없을 시 새로 생성하여 병합
 - save or update 기능 수행
- 예제 실행 시 EntityManager에 최초 merge 했을 때 정상동작 한 이슈에 관한 내용 정리 (TC추가 완료)
 - persist 대신 바로 merge 했을 경우, 관리되는 다른 인스턴스를 리턴 하므로 (원래는 persist에 넣었던 객체로 리턴) 이후 엔티티 값 변화 시 update 되지 않음

```
@Test
@DisplayName("바로 merge 했을 경우, 이후 엔티티 값 변화 시 update 되지 않음, 관리되는 다른 인스턴스를 리턴 하므로")
void init_merge_test() {
    EntityTransaction transaction = em.getTransaction();
    transaction.begin();
    em.merge(member);
    member.setUsername("최초 머지");
    transaction.commit();
}

//결과 : update 쿼리 생성하지 않음
```

- merge 한 엔티티를 다른 인스턴스에 넣을 경우 (참조), 컨텍스트에서 관리 됨

```
@Test
@DisplayName("merge 한 엔티티를 다른 인스턴스에 넣을 경우 (참조), 컨텍스트에서 관리 됨")
void merge_anther_instance() {
    EntityTransaction transaction = em.getTransaction();
    transaction.begin();
    Member another = em.merge(member);
    another.setUsername("다른 인스턴스에 넣은 후 setter 로 값 변경");
    transaction.commit();
}

//결과 : update 쿼리 생성 됨, merge 한 결과를 다룰수 있는 인스턴스에서 참조 하므로
```

- 참고 : <https://stackoverflow.com/questions/1069992/jpa-entitymanager-why-use-persist-over-merge>

3.7 정리

- 엔티티 매니저는 엔티티 매니저 팩토리 에서 생성 됨
- 영속성 컨텍스트는 애플리케이션과 데이터베이스 사이 객체를 보관하는 가상의 데이터베이스 같은 역할을 함
- 영속성 컨텍스트에 저장한 엔티티는 플러시 시점에 데이터베이스에 반영 된다.
- 엔티티가 준영속 상태가 되면 더이상 영속성 컨텍스트의 관리를 받지 못함 (1차캐시, 쓰기 지연,
- 변경감지, 동일성보장, 지연로딩 같은 기능 사용 못함)