

3. 영속성 컨텍스트

jpa 내부 구조

영속성 컨텍스트 : 엔티티를 영구 저장하는 컨텍스트(환경, 문맥)

EntityManager.persist(entity); : 영속성 컨텍스트를 통해서 엔티티를 영속화 시킨다.
entity를 db가 아니라 영속성 컨텍스트 라는 곳에 저장한다.

엔티티 매니저? 영속성 컨텍스트?

엔티티 매니저를 생성을 하면 1대1로 영속성 컨텍스트가 생김. 눈에 보이지 않는 공간이 생긴다고 이해하면 됨. 이건 스프링 프레임워크 해야 이해됨

엔티티의 생명주기?

```
` java
```

```
Member member = new Member();  
member.setId(100L);  
member.setName("HelloJPA");
```

```
System.out.println("BEFORE");  
em.persist(member);  
System.out.println("AFTER");
```

```
tx.commit();
```

```
`
```

BEFORE

AFTER

Hibernate:

```
/* insert hellojpa.Member
*/ insert
into
    Member
    (name, id)
values
    (?, ?)
```

그냥 한번 봐보자.

- 비영속 : 멤버 객체를 생성만 하고 엔티티 매니저로 어디에도 넣지 않은 상태.
 - 영속 : 객체를 생성하고 엔티티 매니저를 얻어와서 persist로 객체를 저장한 상태. persist로 객체를 저장했다고 설명했으나 사실은 그게 아니었음..
 - commit 할 때 영속성 컨텍스트에 있던 애가 디비에 넘어가게 됨.
 - 준영속 : 영속성 컨텍스트에 있다가 detach로 분리된 상태
 - 삭제 : remove로 삭제된 상태
- => 애플리케이션과 디비 사이에 중간 계층이 있음!! → 얻을 수 있는 이점이 많다... 버퍼링, 캐싱, 등등의 이점을 누릴 수 있다.

엔티티 조회, 1차 캐시

멤버 객체 생성, 값 세팅 / 비영속

persist로 엔티티 영속 / 영속

영속 컨텍스트 안에는 1차 캐시가 있는데 이걸 영속 컨텍스트로 생각해도 됨.

@Id가 pk. pk에 대한 값이 member 객체 자체.

1차 캐시에서 조회할 때 find할 때 일단 1차 캐시에서 바로 조회함. → 빠르다는 이점이 있다.

만약 member2 조회하면?

1차캐시에 일단 없음 → 영속성 컨텍스트에 없네? → db 조회 → member2 1차 캐시에 저장 → 저장한 1차 캐시에서 반환.

그러나 이점은 크게 없다. 왜냐면 데이터베이스 트랜잭션이랑 엔티티 매니저는 같이 가는거기 때문에 .. 고객의 요청이 하나 들어와서 비즈니스가 끝나버리면 영속 컨텍스트도 날아감.. 여러명의 고객이 사용하는 캐시는 아님. 데이터베이스 한 트랜잭션 안에서만 성능 업글 가능

BEFORE

AFTER

```
findMember.id = 101
```

```
findMember.id = HelloJPA
```

Hibernate:

```
    /* insert hellojpa.Member
```

```
    */ insert
```

```
into
```

```
    Member
```

```
    (name, id)
```

```
values
```

```
    (?, ?)
```

101을 persist했는데 쿼리가 날아가기도 전에 찾아버린다.. 왜냐하면 1차캐시에 저장이 되기 때문에. 디비

가 아니라 1차캐시에서 먼저 조회하므로 // persist, find 확인

```
6월 08, 2023 12:54:07 오전 org.hibernate
INFO: HHH000400: Using dialect: org.
Hibernate:
    select
        member0_.id as id1_0_0_,
        member0_.name as name2_0_0_
    from
        Member member0_
    where
        member0_.id=?
6월 08, 2023 12:54:07 오전 org.hibernate
INFO: HHH10001008: Cleaning up connection
```

한번 조회했으므로 디비에 select는 딱 한번. 처음에 디비에서 조회할 때 1차캐시에 들어오기 때문에

영속 엔티티의 동일성 보장

같은 트랜잭션 안에서 == 비교하면 true가 뜬다.

엔티티 등록 트랜잭션을 지원하는 쓰기 지연

트랜잭션 실행하고 persist로 a, b 넣어놓는다.

근데 여기까지 commit 안하고 commit 하는 순간 db에 집어넣음.

memberA를 persist를 통해서 넣으면 일단 1차캐시에 들어감.

그리고 jpa가 그와 동시에 엔티티를 분석해서 insert query를 생성. 그래서 쓰기지연 저장소에 저장

그다음 memberB를 persist를 통해서 넣음.

애도 마찬가지로 1차캐시에 들어가고 쓰기지연 저장소에 또 insert query가 들어감
commit()이 오면 쓰기지연 저장소에서 sql들이 플러시 되고 db에 데이터가 커밋됨.

member 객체에 에러가 뜨는 이유 : 인텔리제이가 자동으로 잡아주는거임. 밑에가 이유

jpa는 기본적으로 내부적으로 reflection 같은걸 쓰기 때문에 동적으로 객체를 생성해놔야한다. 그래서 기본 생성자가 하나씩 있어야함.

```
=====
```

```
Hibernate:
```

```
    /* insert hellojpa.Member
      */ insert
      into
          Member
          (name, id)
      values
          (?, ?)
```

```
Hibernate:
```

```
    /* insert hellojpa.Member
      */ insert
```

=====으로 선을 긋고 나서 select 문 flush해버림. 그림과 똑같이 동작하는 것을 볼 수 있다.

이게 버퍼링 한 다음에 한번에 분출할 수 있다는 것. xml 파일에 추가한 batch_size가 이 버퍼링하는 배치의 크기를 정해준다.

엔티티 수정. 변경 감지(dirty checking)

```
Member member = em.find(Member.class, 150L);  
member.setName("ZZZZZ");
```

```
// em.persist(member);
```

자바컬렉션에서 값을 수정하듯이 변경해주면 된다.

자바컬렉션에서도 값만 바꾸지 수정하는 커밋? 같은걸 해주지 않기 때문에...

SELECT * FROM MEMBER;

ID	NAME
1	HelloJPA
2	HelloB
100	HelloJPA
101	HelloJPA
150	A
160	B

(6 행, 3 ms)

편집

SELECT * FROM MEMBER;

ID	NAME
1	HelloJPA
2	HelloB
100	HelloJPA
101	HelloJPA
150	ZZZZZ
160	B

(6 행, 8 ms)

편집

정상적으로 바뀌는 걸 볼 수 있다!!!

****변경 감지** : 영속성 컨텍스트!!!!**

jpa는 db 트랜잭션을 커밋하는 순간에 flush라는 것을 한다.

commit()을 하면 엔티티와 스냅샷을 비교한다.

1차캐시 안에는 스냅샷이라는 것도 있음. 그니 최초로 영속성 컨텍스트에 들어온 그 상황을 스냅샷이라는 것에 찍어놓음.

엔티티와 스냅샷을 일일이 비교해보고 바뀌게 있으면 업데이트 쿼리를 쓰기지연 저장소에 넣어놓고 flush 할 때 sql 던지고 커밋함.

jpa는 그냥 값을 바꾸면 트랜잭션이 커밋되는 시점에 변경을 반영하는구나 라고 생각하고 코드를 작성하는 것이 낫다.

플러시

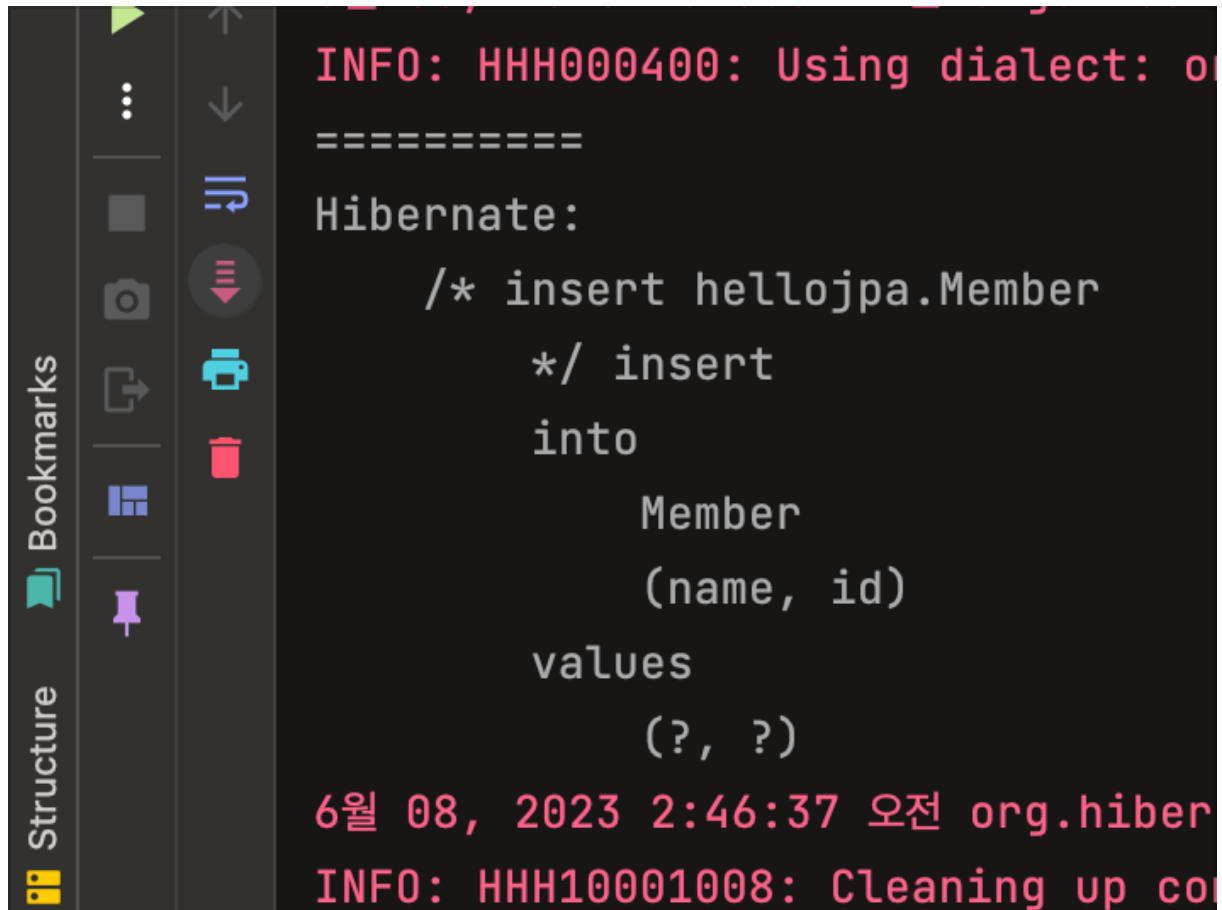
쌓아놔던 sql을 DB에 날려줌.

=> 플러시가 발생한다고 해서 데이터베이스 트랜잭션이 커밋되는게 아니고 .commit()하는 경우에만 커밋된다.

```
6월 08, 2023 1:42:42 오전 org.hibernate
INFO: HHH000400: Using dialect: org
Hibernate:
    /* insert hellojpa.Member
    */ insert
    into
        Member
        (name, id)
    values
        (?, ?)

=====
6월 08, 2023 1:42:42 오전 org.hibernate
```

flush를 쓰니까 sql이 먼저 날라감.



```
INFO: HHH000400: Using dialect: o
=====
Hibernate:
    /* insert hellojpa.Member
    */ insert
    into
        Member
        (name, id)
    values
        (?, ?)
6월 08, 2023 2:46:37 오전 org.hiber
INFO: HHH10001008: Cleaning up co
```

flush를 안하니깐 =====이 먼저 나옴.

em.persist 했을 땐 영속성 컨텍스트에 member가 담기고 쿼리가 저장소에 담김.

근데 flush 하니깐 쿼리가 저장소에서 디비로 날아감.

flush해도 1차 캐시는 안지워짐. 단지 쓰가지면 저장소에 있는 등록수정삭제 쿼리들이 바로 반영이됨.

```
em.persist(memberA);
em.persist(memberB);
em.persist(memberC);

//중간에 JPQL 실행
query = em.createQuery("select m from Member m", Member.class);
List<Member> members= query.getResultList();
```

createQuery를 할 때 그냥 자동적으로 flush를 해줌.

=> 영속성 컨텍스트를 비우는게 아니라 변경내용을 데이터베이스에 동기화!!

어쨌든 트랜잭션 커밋 직전에만 동기화를 해주면 된다!!

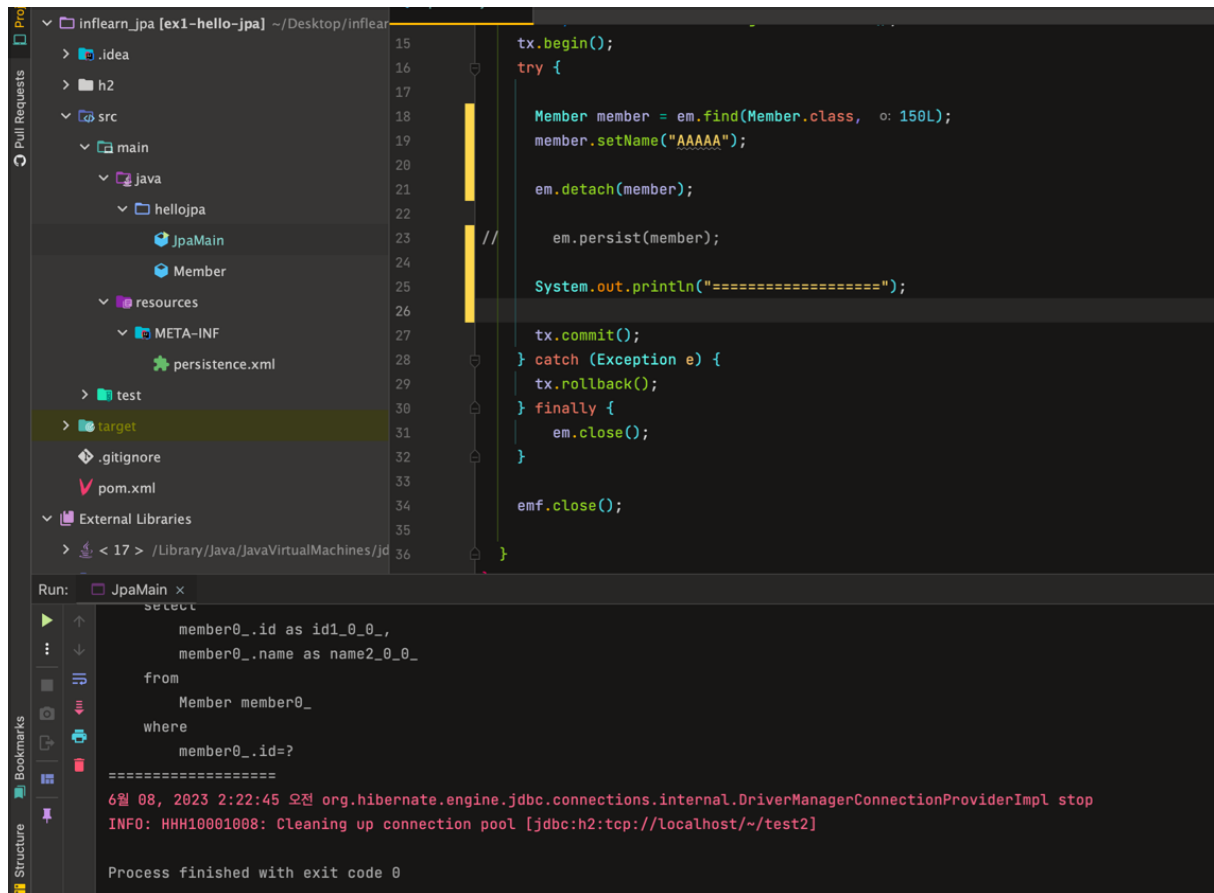
준영속 상태

em.persist로만 영속 상태가 되는게 아님.

em.find로 조회했을 때 1차캐시에 없을 때 db에서 조회해서 1차 캐시에 올라갔다면 영속성 컨텍스트에 올

라가는것이므로 영속 상태가 됨.

em.detach(member)와 같은 형식으로 진행할 수 있는데, 이렇게 되면 1차 캐시에서 빠지고 영속성 컨텍스트가 관리하는 것이 아니게 된다.



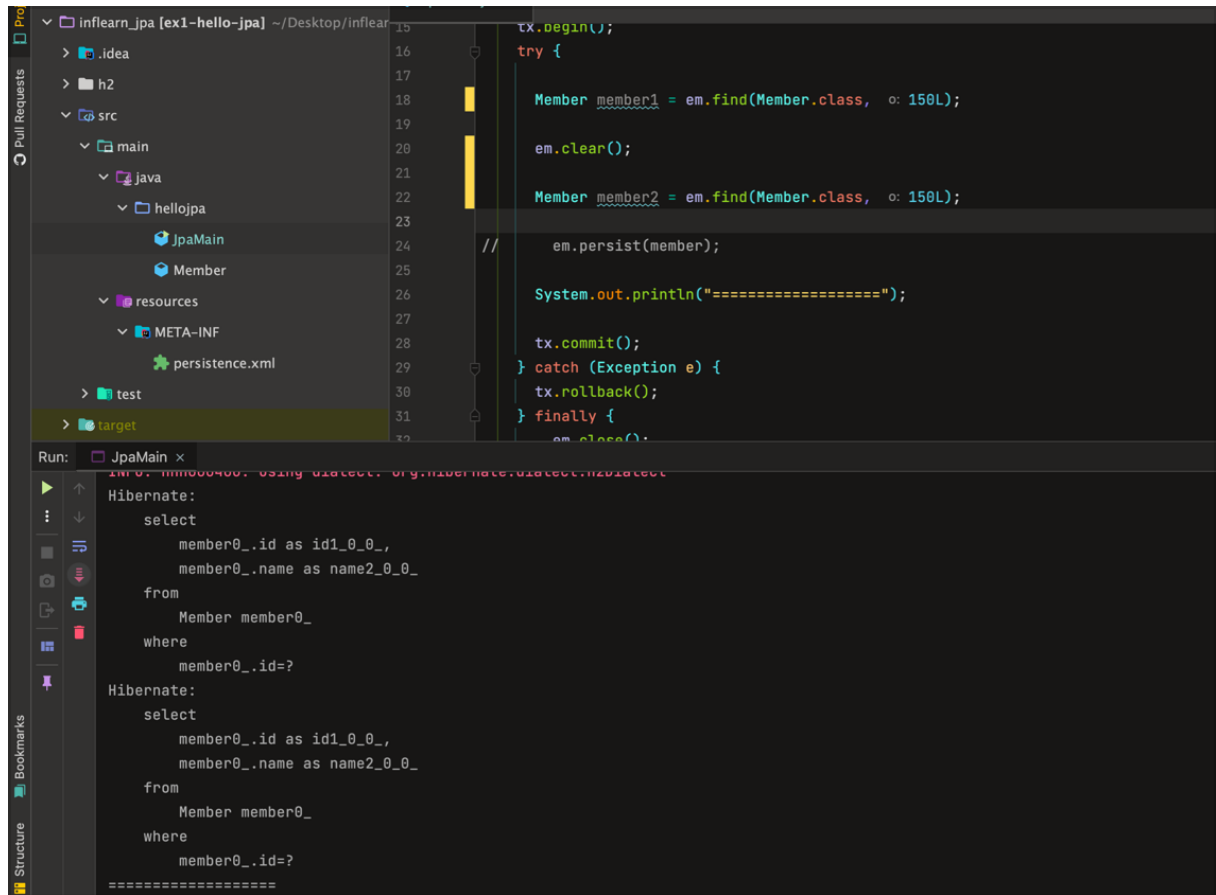
The screenshot shows an IDE with a project structure on the left and a code editor in the center. The project structure includes a package 'hellojpa' with a 'JpaMain' class and a 'Member' entity. The code editor shows a Java class with a transaction and a detach operation. The console output at the bottom shows a SQL query and a log message.

```
15 tx.begin();
16 try {
17
18     Member member = em.find(Member.class, 150L);
19     member.setName("AAAAA");
20
21     em.detach(member);
22
23     //
24     em.persist(member);
25
26     System.out.println("=====");
27
28     tx.commit();
29 } catch (Exception e) {
30     tx.rollback();
31 } finally {
32     em.close();
33 }
34 emf.close();
35
36 }
```

Run: JpaMain x

```
select
  member0_.id as id1_0_0_,
  member0_.name as name2_0_0_
from
  Member member0_
where
  member0_.id=?
=====
6월 08, 2023 2:22:45 오전 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:h2:tcp://localhost/~/.test2]
Process finished with exit code 0
```

실행해보면 선택만 나가고 update는 나가지 않는다. 왜냐하면 detach를 했기 때문에.



clear로 1차 캐시를 지우는 것을 보면 더 잘 이해된다. 1차캐시에서 없어졌기 때문에 캐싱한 데이터를 캐시에서 찾지 못함.

#JPA