Section 1~3

JPA

목차

JPA 소개

SQL 중심적인 개발의 문제점과 JPA 소개

SQL작업 시 CRUD의 작업이 반복됨
객체지향 프로그래밍과 SQL은 차이점이 많음
1. 상속 : 슈퍼타입 서브타입으로 작성하여도 계속 2번 이상 insert 해야함, 조회 시에도 각각의 join쿼리를 작성해야함
2. 연관관계 : 객체는 참조 사용, 테이블은 외래 키 사용, 따라서 객체를 테이블에 맞추어 모델링하는 문제가 생김
3. 처음 실행하는 SQL에 따라 탐색 범위가 결정되는 문제 → 엔티티 신뢰 문제가 생김
4. 같은 id로 조회를 해와도 비교할 때 다르다고 인식하는 문제가 생김(자바에서는 다른 객체이기 때문)

객체답게 모델링 할수록 매핑 작업이 늘어나는데 이를 JPA가 해결 가능하다.

JPA는 자바 진영의 ORM(객체 관계 매핑) 기술 표준
JAVA<-> JPA <-> JDBC 동작

장점
1. 객체 중심 개발 가능
2. 생산성, 유지보수
3. 성능 : 1차 캐시와 동일성 보장, 트랜잭션을 지원하는 쓰기 지연, 지연 로딩(사용될 때 조회)과 즉시 로딩(미리 조회)
4. 데이터 접근 추상화와 벤더의 독립성
5. 표준

JPA 시작하기 1

프로젝트 생성, 애플리케이션 개발

H2 데이터 베이스 사용 : chmod u+x h2.sh 후 sudo ./h2.sh 로 실행
사용자 : test 비밀번호 : testpw

Maven 프로젝트 시작 → https://mvnrepository.com에서 관련 dependency 추가
Persistence 작성
Member table 생성
만약 테이블 명과 클래스 명이 다르면 따로 매핑을 해주면 된다 ex) @Table, @Column

```java
@Entity
public class Member {
    2 usages
    @Id
    private Long id;
    2 usages
    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; }
}
```

JPA 시작하기 2

INSERT 실습

JPA insert 해보기

```
EntityManagerFactory emf =  Persistence.createEntityManagerFactory( persistenceUnitName: "h2");
EntityManager em= emf.createEntityManager();

EntityTransaction tx = em.getTransaction();
tx.begin();
Member member = new Member();
member.setId(1L);
member.setName("hello");
em.persist(member);

tx.commit();

em.close();
emf.close();
```

결과 : 자동으로 쿼리를 작성 해준다

```
Hibernate:
    /* insert jpa.Member
        */ insert
        into
            Member
            (name, id)
        values
            (?, ?)
```

```
SELECT * FROM MEMBER
```

SELECT * FROM MEMBER;

ID	NAME
1	hello

JPA 시작하기 3

SELECT, UPDATE

JPA Select & Update 해보기

```
EntityManagerFactory emf =  Persistence.createEntityManagerFactory( persistenceUnitName: "h2");
EntityManager em= emf.createEntityManager();

EntityTransaction tx = em.getTransaction();
tx.begin();
try {
    Member member = em.find(Member.class, o: 1L);
    member.setName("changeName");
    em.persist(member);
    tx.commit();
}catch (Exception e){
    tx.rollback();
}finally {
    em.close();
}
emf.close();
```

결과 : 따로 업데이트 하지 않아도 변경하면 자동으로 Update 된다

```
Hibernate:
    select
        member0_.id as id1_0_0_,
        member0_.name as name2_0_0_
    from
        Member member0_
    where
        member0_.id=?
Hibernate:
    /* update
        DAO.Member */ update
            Member
        set
            name=?
        where
            id=?
```

SELECT * FROM MEMBER;

ID	NAME
1	changeName

JPA 시작하기 4

주의점과 JPQL

주의점
엔티티 매니저 팩토리는 하나만 생성해 애플리케이션 전체에 공유
엔티티 매니저는 쓰레드간 공유X

복잡한 쿼리는 어떻게 처리할까? → JPQL

JPQL은 엔티티 객체를 대상으로 쿼리를 작성한다
SELECT, FROM, WHERE, GROUP BY, HAVING, JOIN 지원

JPA 영속성 관리

영속성 컨텍스트

영속성 컨텍스트
- 엔티티를 영구 저장하는 환경이라는 뜻
- 영속성 컨텍스트는 논리적인 개념, 엔티티 매니저를 통해 접근한다

엔티티 생명주기
1. 비영속 2. 영속 3. 준영속 4. 삭제

영속성 컨테스트의 이점
- 1차 캐시 ➔ 조회 시 1차 캐시에서 조회, 없으면 DB조회 후 1차 캐시에 저장
1차 캐시는 한 트랜잭션에서 존재, 트랜잭션이 끝나면 사라진다
- 동일성 보장
- 트랜잭션을 지원하는 쓰기 지연 ➔ 커밋 하는 순간 트랜잭션 실행(모아서 진행, 쓰기 지연 SQL 저장소에 저장)
- 변경 감지 ➔ 자바 컬렉션처럼 변경을 감지하고 UPDATE 실행
스냅샵과 비교해서 다르면 update 실행
- 지연로딩

JPA 영속성 관리

플러시

플러시
-변경 감지
- 스냅샵과 비교해 다르면 쓰기 지연 SQL 저장소에 등록
- 쓰기 지연 SQL 저장소의 쿼리를 데이터베이스에 전송
- 1차 캐시는 지워지지 않는다

플러시 하는 법
1. em.flush() 2. 트랜잭션 커밋 3. JPQL 쿼리 실행

플러시 모드 옵션
1. FlushModeType.AUTO : 커밋이나 쿼리를 실행할 때 플러시 (Default)
2. FlushModeType.COMMIT : 커밋할 때만 플러시

JPA 영속성 관리

준영속

준영속
-영속 상태의 엔티티가 영속성 컨텍스트에서 분리(detached), 영속성 컨텍스트가 제공하는 기능 사용 못함

만드는 방법
- em.detach(entitiy) : 엔티티만 준영속 상태로 전환
- em.clear() : 영속성 컨텍스트를 완전히 초기화
- em.close() : 영속성 컨텍스트를 종료