2020-12-14

SQL 중심적인 개발의 문제점

객체지향언어 관계형 DB -> SQL 중심적인 개발 (무한 반복 ,지루한 코드 CRUD)

필드 추가나 수정시 모든 SQL문을 수정을 해야한다.

SQL에 의존적인 개발을 피하기 어렵다 .

패러다임의 불일치

**\*객체를 자바 컬렉션에 저장 하듯이 DB에 저장할 수는 없을까?**

JPA –Java Persistence API

ORM?

Object-relational mapping(객체 관계 매핑)

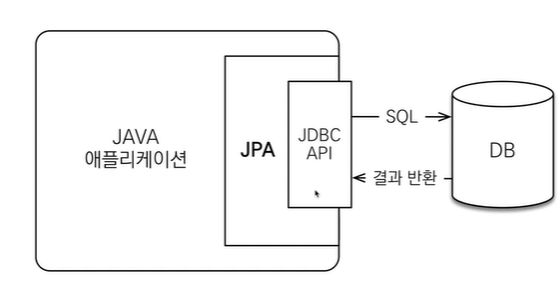
객체는 객체대로 설계

관계형 데이터베이스는 관계형 데이터 베이스대로 설계

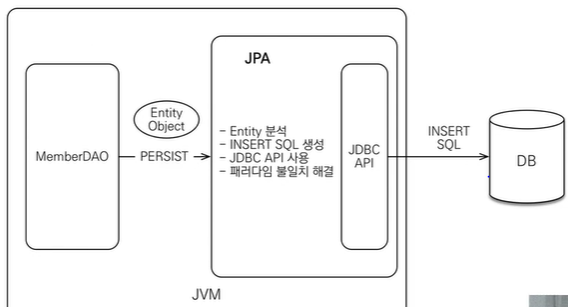
ORM 프레임워크가 중간에서 매핑

대중적인 언어에는 대부분 ORM 기술이 존재

JPA는 애플리케이션과 JDBC 사이에서 동작



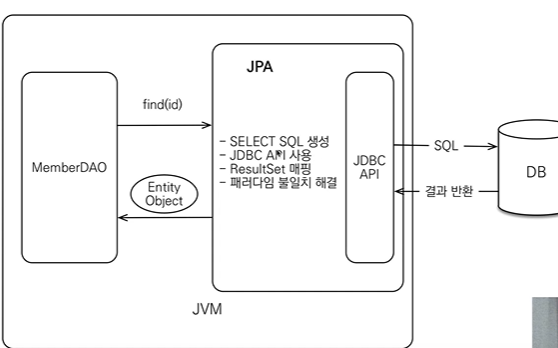
데이터 생성



\*패러다임 불일치 해결이 중요하다 .!

SQL을 JPA가 만들어주는것이다 .

데이터조회

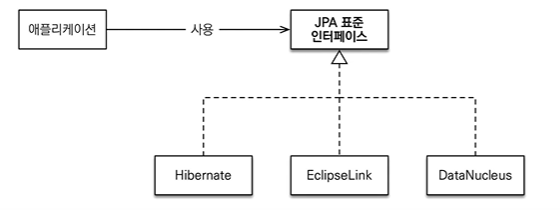


JPA 소개

EJB – 엔티티빈 (자바 표준) -> 하이버네이트(오픈 소스) -> JPA(자바 표준)

성능도 좋지않다 등등 -> 오픈소스를 통해서 많이 발전함 -> 하이버네이트정리해서 만듬

JPA는 인터페이스의 모음



JPA를 왜 사용해야 하는가 ?

SQL 중심적인 개발에서 객체 중심으로 개발-> 객체 상속관계 ,

생산성, 유지보수 -> CRUD가 만들어져있다 .,필드 추가시 모든 SQL이 알아서 수정된다.

패러다임의 불일치 해결

성능

데이터 접근 추상화와 벤더 독립성 //자유로운 객체 그래프탐색 가능

표준

JPA의 성능 최적화 기능

1차캐시와 동일설 보장 -> 약간의 조회 성능 향상

트랜잭션을 지원하는 쓰기 지연 em.persist // // transaction .commit

지연 로딩 : 객체가 실제 사용될 때 로딩 != ( 즉시 로딩 JOIN SQL로 한번에 연관된 객체까지 미리조회)

ORM은 객체와 RDB 두 기둥위에 있는 기술

2020-12-17

데이터 베이스 방언

JPA는 특정 데이터베이스에 종속 되어있지 않는다.

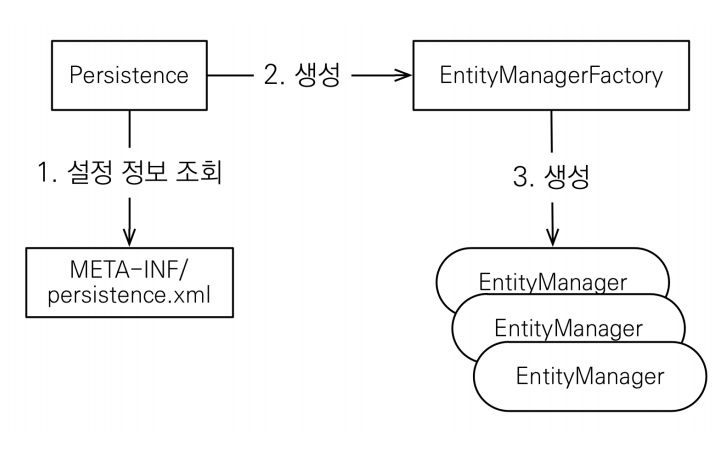
각각의 데이터베이스가 제공하는 sql 문법과 함수는 조금씩 다름

가변문자 :MySQL은 VARCHAR ,Oracle 은 VARCHAR2

문자열을 자르는 함수 :SQL 표준은 SUBSTRING(), Oracle은 SUBSTR

페이징 : MySQL은 LIMIT oracle은 ROWNUM

JPA 구동방식



엔티티 매니저 팩토리는 하나만 생성해서 애플리 케이션 전체에서 공유

엔티티 매니저는 쓰레드간에 공유 X (사용하고 버려야 한다.)

JPA의 모든 데이터 변경은 트랜잭션 안에서 실행

JPQL

JPA를 사용하면 에티티 객체를 중심으로 개발

문제는 검색 쿼리

검색을 할 때도 테이블이 아니 엔티티 객체를 대상으로 검색

모든 DB데이터를 객체로 변환해서 검색하는 것은 불가능

애플리케이션이 필요한 데이터만 DB에서 불러오려면 결국 검색 조건이 포함된 SQL 이 필요

Jpa는 SQL을 추상화한 JPQL이라는 객체 지향 쿼리 언어 제공

SQL과 문법 유사, SELECT,FROM,WHERE,GROUP BY ,HAVING ,JOIN 지원

차이점

JPQL은 엔티티 객체를 대상으로 쿼리

SQL은 데이터 베이스 테이블을 대상으로 쿼리

영속성 관리

JAP 에서 가장 중요한 2가지

객체와 관계형 데이터베이스 매핑하기

영속성 컨텍스트

엔티티를 영구 저장하는 환경이라는 뜻.

EntitiyManager.persist(entity);

엔티티매니저,영속성 컨텍스트

영속성 컨텍스트는 논리적인 개념

눈에 보이지 않는다.

엔티티 매니저를 통해서 영속성 컨텍스트에 접근

엔티티의 생명주기

비영속

영속성 컨텍스트와 전혀 관계가 없는 새로운 상태 (객체만 생성한 상태 )

영속

영속성 컨텍스트에 관리되는 상태(em.persist())

준영속

영속성 컨텍스트에 저장되었다가 분리된 상태

영속-> 준영속

영속 상태의 엔티티가 영속성 컨텍스트에서 분리(detached)

영속성 컨텍스트가 제공하는 기능을 사용 못함

Em.detach(entity)-특정 엔티티만 준영속 상태로 전환

em.clear() –영속성 컨텍스트를 완전히 초기화

em.close()-영속성 컨텍스트를 종료

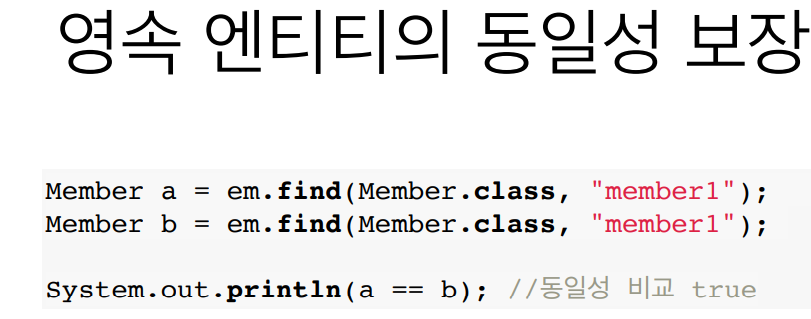
삭제

삭제된 상태

영속성 컨텍스트의 이점

1차 캐시: 똑같은 것을 두번 조회 할때는 캐시 에서 가져온다.

동일성(identity) 보장:



트랜잭션을 지원하는 쓰기 지연(transactional write- behind) :commit 하는 순간에 SQL문을 보낸다.

변경 감지(Dirty Checking): 컬렉션을 다루둣이 하기위해서 update()같은 것은 필요하지않는 다.

플러시 –영속성 컨텍스트의 변경내용을 데이터 베이스에 반영,

영속성 컨텍스트를 비우지 않음,영속성 컨텍스트의 변경 내용을 데이터

베이스에 동기화

트랜잭션이라는 작업 단위가 중요-> 커밋 직전에만 동기화 하면 됨

플러시 발생

변경감지,수정된 엔티티 쓰기 지연 SQL 저장소에 등록,

쓰기지연 SQL 저장소의 쿼리를 데이터 베이스에 전송

영속성 컨텍스트를 플러시 하는 방법

em.flush()-직접 호출 (1차캐시는 지워 지지 않는다 )

트랜잭션 커밋 –플러시 자동 호출

JPQL 쿼리 실행 –플러시 자동 호출

지연 로딩 (Lazy Loading)

2020-12-19

엔티티 매핑

객체와 테이블 매핑 :@Entitiy,@Table

@Entity

어노테이션이 붙은 클래스는 JPA가 관리, JPA를 사용해서 테이블과 매핑할 클래스는 필수

기본 생성자 필수 (파라미터가 없는 public 또는 protected 생성자)

Final 클래스,enum,interface,inner 클래스 사용 X

저장할 필드에 final 사용 X

@Table 은 엔티티와 매핑할 테이블 지정

Name ,catalog, schema,uniqueConstraints(DDL 생성 시에 유니크 제약 조건 생성 )

\*데이터 베이스 스키마 자동생성

DDL을 애플리케이션 실행 시점에 자동 생성

테이블 중심 -> 객체 중심

데이터베이스 방언을 활용해서 데이터 베이스에 맞는 적절한 DDL 생성

이렇게 생성된 DDL은 개발 장비에서만 사용

생성된 DDL은 운영서버에서는 사용하지않거나 적절히 다듬은후 사용

DDL 생성 기능 – 제약조건 추가 ( @Column(nullable = false,length=10))

유니크 제약조건 추가

DDL 생성기능은 DDL을 자동 생성할 때만 사용되고 JPA의 실행 로직에는 영향을 주지 않는다.

필드와 컬럼 매핑 :@Column

기본 키 매핑 :@Id

연관관계 매핑: @ManyToOne,@JoinColumn