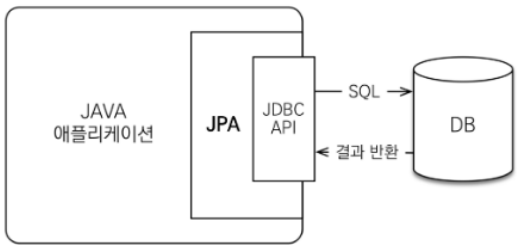
원상_JPA

Created	@2022년 9월 7일 오전 10:59
Tags	
작성자	
분류	JPA

JPA

JPA 동작 과정



- 애플리케이션과 JDBC 사이에서 동작
  - 개발자가 JPA 호출 → JPA내부에서 JDBC API를 사용해 SQL호출, DB와 통신

JPA의 생산성

- JPA를 사용하는 것은 마치 Java Collection에 데이터를 넣었다 빼는 것처럼 사용할 수 있게 만든 것

유지보수

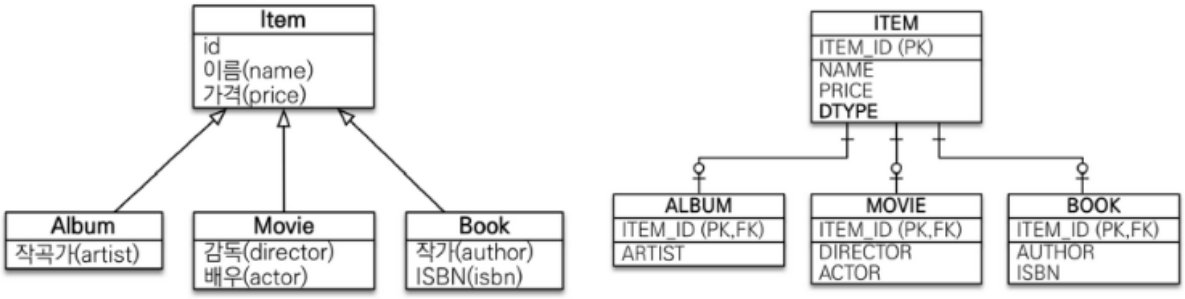
- 필드만 추가하면 SQL은 JPA가 처리하기 때문에 손댈 것이 없다.

JPA 의 간단한 CRUD

- 저장: jpa.persist(dto)
  - 개발자가 JPA에 객체를 넘김 → JPA는 객체 엔티티 분석 → Insert SQL 생성 → JDBC API를 이용해 DB에 SQL실행
- 조회: Dto dto = jpa.find(dto.pk)
  - 개발자가 JPA에 객체의 pk값을 넘김 → JPA는 엔티티의 매핑 정보를 바탕으로 SELECT SQL 생성 → JDBC API를 사용해 DB에 SQL 실행 → DB로부터 결과를 받아 ResultSet을 객체에 모두 매핑
- 수정: dto.setAttr(”변경 값”)

    - 객체를 변경하면 알아서 DB에 UPDATE 쿼리가 실행 되어 업데이트 됨
- 삭제: jpa.remove(dto)

JPA 상속



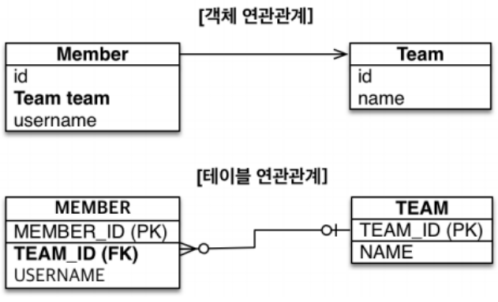
- Album 클래스를 저장하는 경우

```
jpa.persist(album) // 앨범 객체 저장
/**
INSERT INTO ITEM(ID, NAME, PRICE) ...
INSERT INTO ALBUM(ARTIST) ...
와 같이 위 자바 코드를 SQL 쿼리로 변환해서 실행
*/
```

- Album 클래스 조회시

```
Album album = jpa.find(Album.class, albumId); // 앨범 객체 조회
/**
SELECT I.* A.*
  FROM I
  JOIN ALBUM A ON I.ITEM_ID = A.ITEM_ID
와 같이 자바 코드를 SQL쿼리로 변환해서 실행
*/
```

JPA의 연관관계



- Member 클래스가 Team 타입의 team 필드 변수를 가지고 있는 경우

```
class Member {
 String id;
 Team team;
```

```
 String username;
}

class Team {
 Long id;
 String name;
}
```

- Member 객체 저장

```
Member member = new Member();
member.setId("100");
member.setUsername("dbjh");

Team team = new Team();
team.setName("dev_team");

member.setTeam(team);
jpa.persist(member);
/**
INSERT INTO MEMBER(ID, TEAM_ID, USERNAME) ...
INSERT INTO TEAM(ID, NAME) ...
*/
```

- Member 객체 조회

```
Member member = jpa.find(Member.class, memberId);
Team team = member.getTeam();

/**
SELECT M.*, T.*
 FROM MEMBER M
 JOIN TEAM T ON M.TEAM_ID = T.TEAM_ID
*/
```

JPA의 성능 최적화 기능

- 1차 캐시와 동일성 보장 (캐싱 기능)
  - 한 트랜잭션 안에서는 조회할 때 같은 엔티티를 반환. 1차 캐시에 저장해 둠
- 트랜잭션을 지원하느 쓰기 지연 - 버퍼링 기능
  - insert

```
/** 1. 트랜잭션을 커밋할 때까지 INSERT SQL을 모음 */
transaction.begin();  // [트랜잭션] 시작
em.persist(memberA);
em.persist(memberB);
em.persist(memberC);
// -- 여기까지 INSERT SQL을 데이터베이스에 보내지 않는다.
// 커밋하는 순간 데이터베이스에 INSERT SQL을 모아서 보낸다. --
/** 2. JDBC BATCH SQL 기능을 사용해서 한번에 SQL 전송 */
transaction.commit(); // [트랜잭션] 커밋
```

- 트랜잭션을 commit할 때까지 INSERT SQL을 메모리에 쌓음 → DB에 INSERT쿼리를 날리기 위한 네트워크를 여러번 타게 됨.
- JDBC Batch SQL 기능을 사용해서 한 번에 SQL을 전송

IntelliJ에서 스프링 실행

- 프로젝트 생성: File → New → project → Spring Initializr

application.properties

```
spring.jpa.show-sql=true // 실행한 SQL쿼리를 콘솔에 찍어줌
spring.jpa.properties.hibernate.format_sql=true // SQL 쿼리를 정렬하여 보여줌

spring.jpa.hibernate.ddl-auto=update // 자동 테이블 생성 설정

spring.jpa.database=mysql // jpa에서 mysql 사용 설정
```

DTO

```
@Entity(name = "user") // 엔티티 설정, 테이블 생성 (name: 테이블 이름)
@Data  // lombok Data 설정
@AllArgsConstructor // 생성자 설정 (없으면 Builder 안됨)
@NoArgsConstructor  // 생성자 설정 (없으면 DTO 에러)
@Builder // 빌더 패턴
public class User {
    @Id  // Primary Key 설정
    @GeneratedValue(strategy = GenerationType.IDENTITY) // 값 생성 annotation, IDENTITY 는 auto increment
    private Long id;

    @Column(length = 100, nullable = false) // 컬럼 설정 (없으면 String의 경우는 varchar(255))
    private String username;
    @Column(length = 100, nullable = false) // 컬럼 설정 (없으면 String의 경우는 varchar(255))
    private String password;
}
```

Repository

```
import com.example.springtest.model.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
} // JpaRepository<(DTO 클래스), (PK 타입)> 상속하면 jpa 메소드 사용 가능
```

Controller

```
@Autowired
private UserRepository userRepository; // repository 연결
```

```
@PostMapping("/user")
public User create(@RequestBody User user) {
    return userRepository.save(user); // DB 에 user 데이터 저장
}

@GetMapping("/user/{id}")
public Optional<User> read(@PathVariable long id) {
    Optional<User> userOptional = userRepository.findById(id); // DB에서 id를 pk로 하는 user 데이터 조회
    userOptional.ifPresent(System.out::println);
    return userOptional;
}
```

ps

- 스프링에서 흔히 사용하는 것으로 알고 있는 JPA는 JPA를 이용하는 spring-data-jpa 프레임워크이지 JPA는 아니다.