Spring Data JPA

1. JPA 기초와 핵심 원리 이해

1-1. ORM과 JPA 소개

1) ORM(Object Relational Mapping)

JDBC API를 사용할 때는 애플리케이션의 비즈니스 로직보다 SQL과 JDBC관련 코드 작업이 더 많은 시간을 보내게 했다. 하지만 MyBatis 또는 Hibernate와 같은 ORM프레임워크를 사용하면서 데이터베이스의 자원들을 엔티티 객체로 얻어내기까지 자동화 되어 보다 효율적인 개발 환경을 제공해 주었지만 여전히 등록,조회,수정,삭제(CRUD) 작업에 필요한 SQL문장들의 반복적인 작업은 계속 이어졌으며 이 부분에 있어서는 비생산적이었다.

2) JPA(Java Persistence API)

JPA는 정보를 데이터베이스에 저장하고 관리할 때, 개발자가 직접 SQL을 작성하는 것이 아니라 JPA가 제공하는 API를 사용하면 된다. 즉, JPA가 개발자 대신에 적절한 SQL을 생성해서 데이터베이스에 전달하게 되는 자바환경의 ORM표준 라이브러리다.

[예제1] RDBMS는 오라클을 사용할 것이며, employees테이블의 자원들을 모두 검색하는 예제를 만들어 보자! (물론 오라클 드라이버(ojdbc14.jar)파일은 라이브러리로 등록이 되어야 함)

먼저 스프링부트 프로젝트(Ctrl+Shift+p)를 다음과 같은 dependencies를 지정하자!

```
Spring Boot DevTools
Spring Web
Lombok
Spring Data JPA
MySQL Driver
```

그리고 설정파일인 application.properties파일의 확장자를 yml로 변경하고 다음과 같이 설정하자!

```
server:
    port: 8080

spring:
```

```
    datasource:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://localhost:3306/test?serverTimezone=UTC
        username: test_admin
        password: 1111

    jpa:
        show_sql: true
        generate-ddl: true
        properties:
            hibernate:
                format_sql: true
```

다음은 Entity객체 생성

store/ProductJPO.java

```
package com.sist.ex1_jpa.store;

import java.time.LocalDate;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity(name = "product_t")
@Getter
@Setter
@NoArgsConstructor
@Builder
@AllArgsConstructor
```

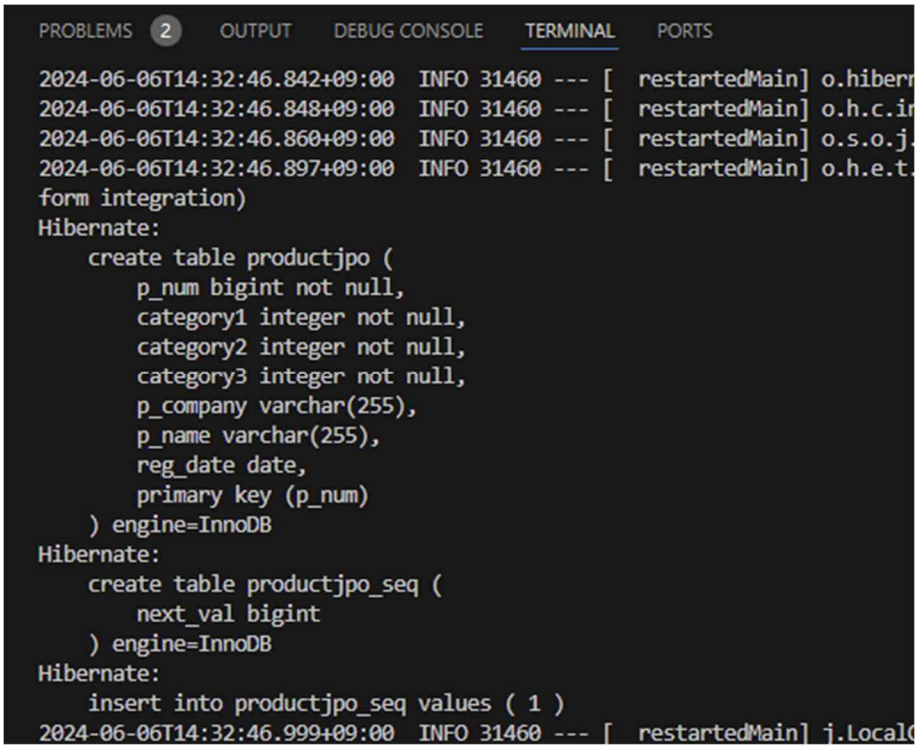
```
public class ProductJPO {
    @Id
    @GeneratedValue
    private long pNum;
    private String pName;
    private String pCompany;
    private LocalDate regDate;
    private int category1;
    private int category2;
    private int category3;
}
```

다음은 Repository객체를 정의하자!

repository/ProductRepository.java

```
package com.sist.ex1_jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.sist.ex1_jpa.store.ProductJPO;

@Repository
public interface ProductRepository extends JpaRepository<ProductJPO, Long>{

}
```

여기까지 한 후 서버를 구동시키면 Log상에 다음과 같은 문장을 확인할 수 있다.



생성되는 테이블의 이름을 변경하려면 엔티티객체 @Entity(name = "product_t") 수정하면 됨!

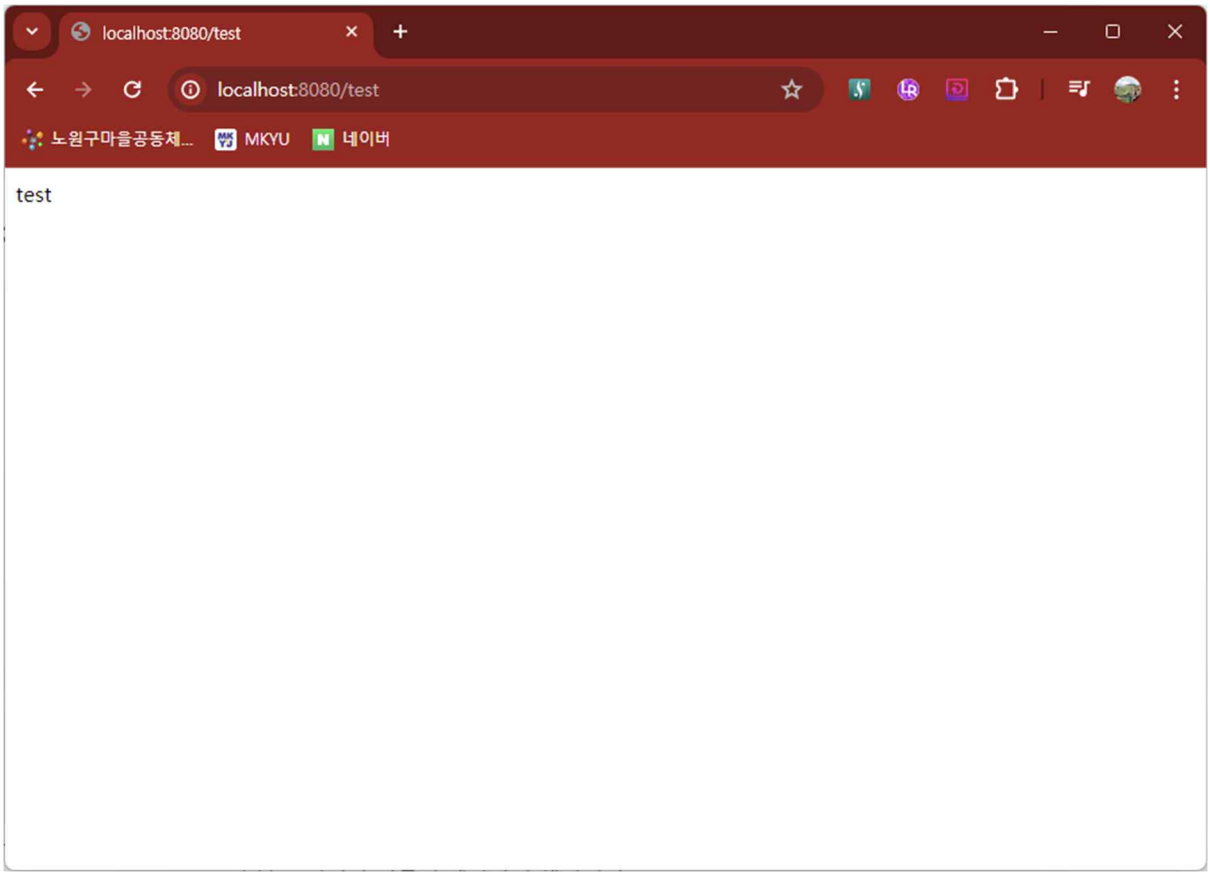
테스트를 하기 위해 컨트롤러를 만들어서 저장을 해보자!

controller/ProductController.java

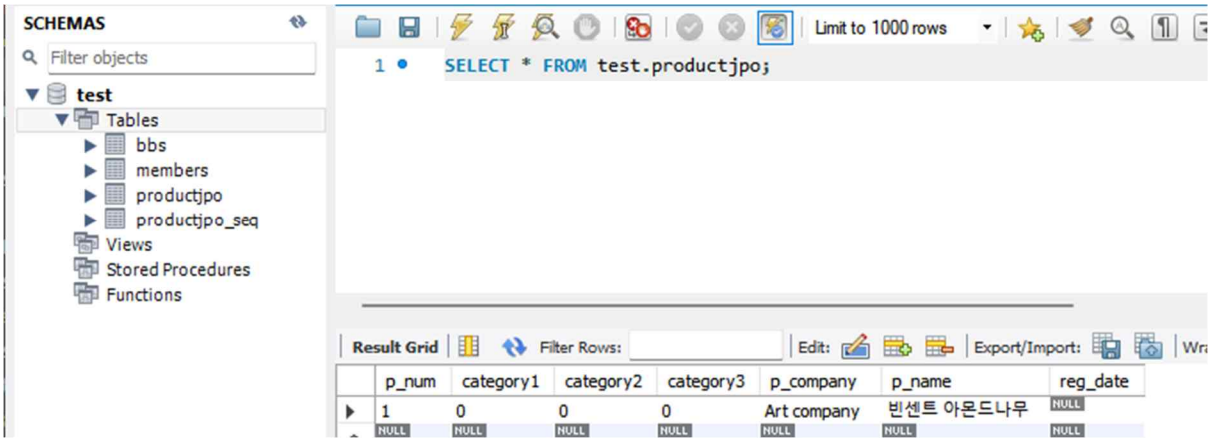
```
package com.sist.ex1_jpa.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RestController;

import com.sist.ex1_jpa.repository.ProductRepository;
import com.sist.ex1_jpa.store.ProductJPO;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@RestController
public class ProductController {

    @Autowired
```

```
    private ProductRepository productRepository;

    @GetMapping("test")
    public String test() {
        ProductJPO p1 = ProductJPO.builder()
            .pNum(100L)
            .pName("빈센트 아몬드나무")
            .pCompany("Art company").build();

        System.out.println(p1);
        productRepository.save(p1);
        return "test";
    }

}
```

브라우저를 하나 열어서 주소 창에 다음과 같이 입력한 후 MySQL Workbench를 열어 확인해 부자!

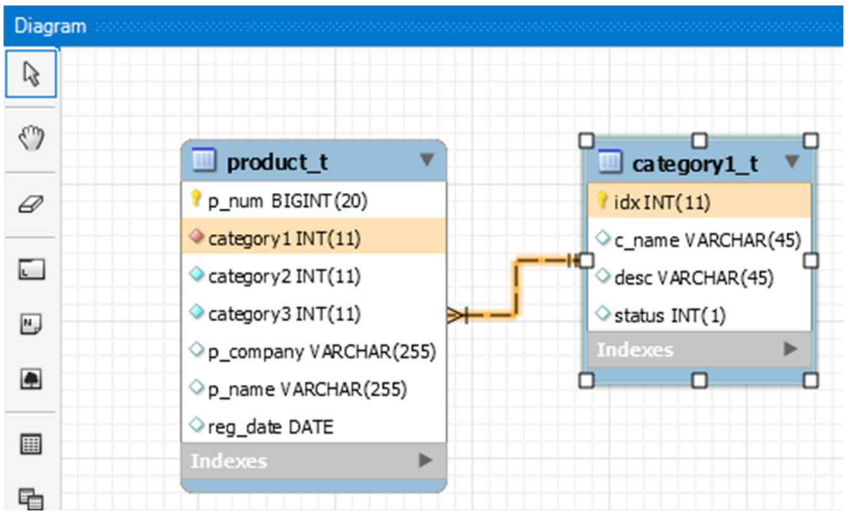


MySQL Workbench에서 조회 화면 결과

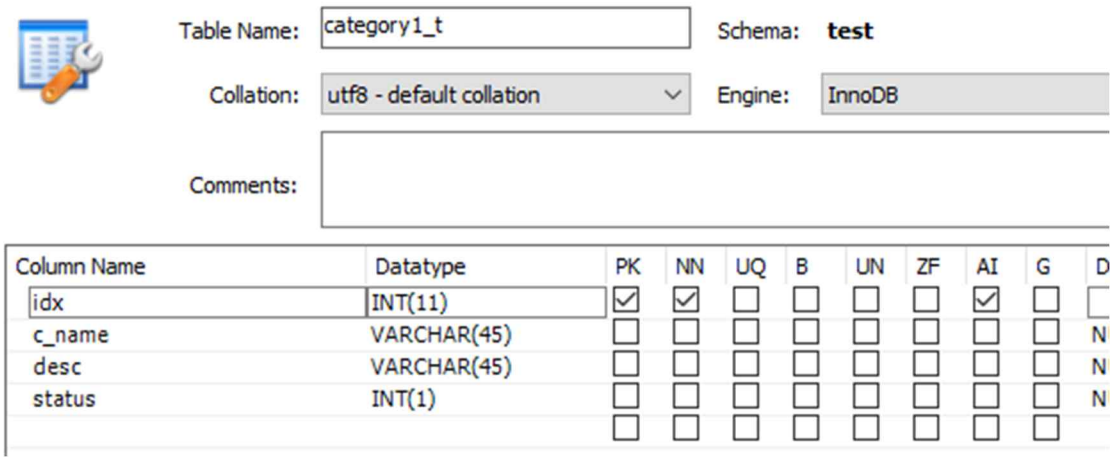


그리고 **save()메서드는 기본키(pNum)의 값이 이미 저장된 값일 경우엔 insert의 역할을 수행하지 않고 update로 해석되어 실행**한다.

이제는 MyBatis에서 사용해 봤던 resultMap형태와 같은 복합적인 결과를 얻기 위해 JPA에서는 어떻게 해야하는지?를 알아보도록 하자! 다음과 같은 테이블 구조를 만들어 보자!



Product_t라는 테이블은 있으니 우린 category1_t라는 테이블만 다음과 같이 구성해 보자!



이제는 위의 테이블 자원과 연동되는 Entity를 만들어 보자!

store/Category1JPO.java

```
package com.sist.ex1_jpa.store;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity(name = "category1_t")
@Getter
@Setter
@NoArgsConstructor
@Builder
@AllArgsConstructor
public class Category1JPO {

    @Id
    @GeneratedValue
    private int idx;
    private String cName;
    private String desc;
    private int status;
}
```

그리고 ProductJPO와 Category1JPO객체간 연관관계의 기준을 다음과 같이 정의하자!

store/ProductJPO.java

```
package com.sist.ex1_jpa.store;
...
```

```
@Entity(name = "product_t")
@Getter
@Setter
@NoArgsConstructor
@Builder
@AllArgsConstructor
public class ProductJPO {
    @Id
    @GeneratedValue
    private long pNum;
    private String pName;
    private String pCompany;
    private LocalDate regDate;

    @Column(name = "category1")
    private int category1;

    private int category2;
    private int category3;

    @ManyToOne
    @JoinColumn(name = "category1", insertable = false, updatable = false)
    private Category1JPO cvo1;
}
```

여기서 중요한 부분은

```
@JoinColumn(name = "category1", insertable = false, updatable = false)
```

해당 컬럼을 Hibernate가 이러한 필드를 삽입하거나 업데이트하지 않도록 지정하는 것이다.

다음은 Controller에 결과를 확인할 수 있는 메서드와 mapping을 정의해 보자!

controller/ProductController.java

```
...
@RestController
public class ProductController {
    ...
  ...
   @GetMapping("list")
```

```
  public String getList() {

        List<ProductJPO> list =
productRepository.findAll(Sort.by(Sort.Direction.DESC,"pNum"));
        StringBuffer sb = new StringBuffer();

        for(ProductJPO p : list){
            sb.append(p.getPNum());
            sb.append("/");
            sb.append(p.getPName());
            sb.append("/");
            sb.append(p.getCvo1().getCName());
            sb.append("/");
            sb.append(p.getCvo1().getDesc());
            sb.append("<br/>");
        }
        return sb.toString();
  }
}
```

결과를 확인하기 위해 브라우저에서 다음과 같이 입력하고 요청해 보자!