

## 2장. SPRING DATA JPA

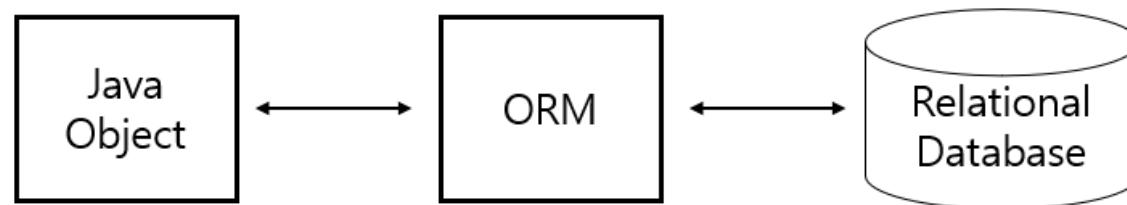
---

- 1. JPA의 특징 / 구조 / 동작 방식 확인
- 2. Spring Data JPA를 이용한 데이터 처리 방법 확인

## 2.1 JPA

---

1. JPA(Java Persistence API)는 자바 ORM 기술에 대한 API 표준
2. ORM이란 'Object Relational Mapping'의 약자로 객체와 관계형 데이터베이스 매핑



[그림 2-1] ORM을 이용한 Java Object와 관계형 데이터베이스 매핑

---

## JPA 장점

---

1. 특정 데이터베이스에 종속되지 않음
  2. 객체지향적으로 설계 가능
  3. 유지보수 용이 및 생산성 향상
-

# JPA 단점

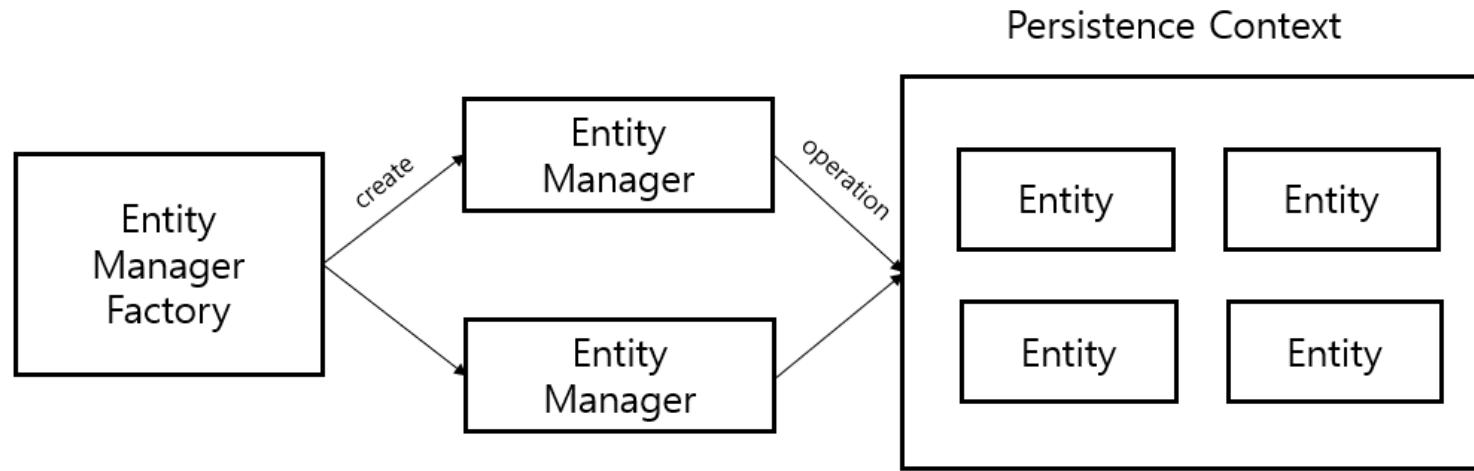
---

1. 복잡한 쿼리 처리

2. 성능 저하 위험

# JPA 동작 방식

---



[그림 2-2] JPA 동작 방식

---

# JPA 동작 방식

---

- 엔티티 : 데이터베이스의 테이블에 대응하는 클래스.  
    @Entity 어노테이션을 붙여서 관리
- 
- \* 엔티티 매니저 팩토리 : 엔티티 매니저 인스턴스를 관리하는 주체

# JPA 동작 방식

---

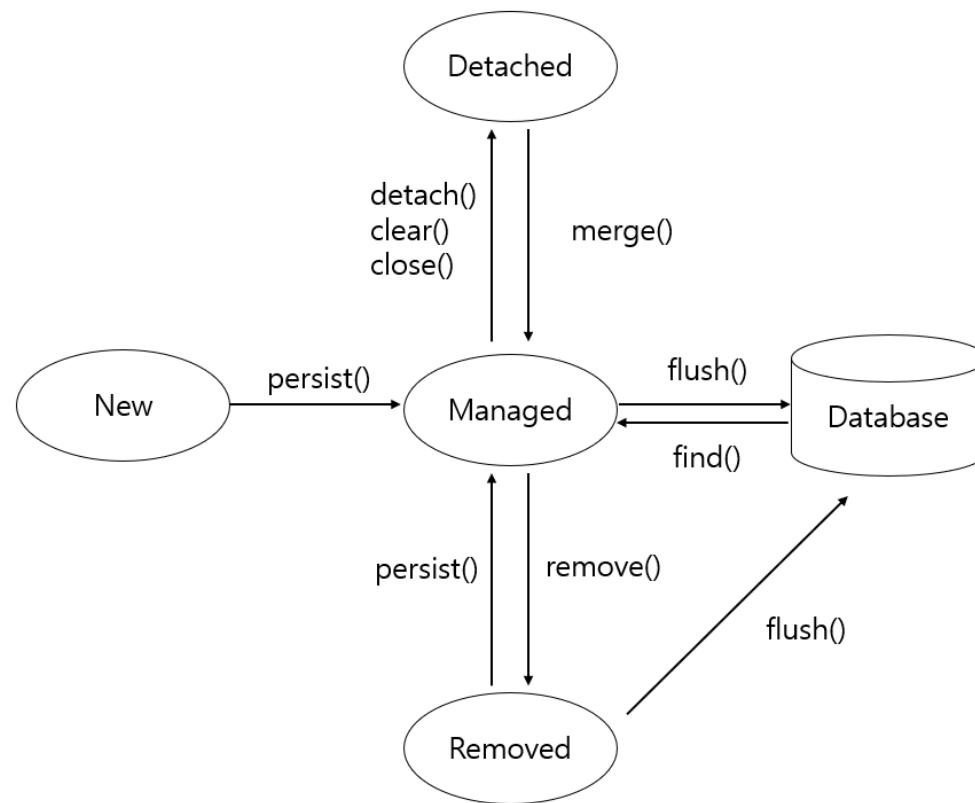
\* 엔티티 매니저 : 영속성 컨텍스트에 접근하여 엔티티에 대한 데이터베이스 작업 제공

[표 1-1] 엔티티 생명주기의 세부 내용

생명주기	내용
비영속(new)	new 키워드를 통해 생성된 상태로 영속성 컨텍스트와 관련이 없는 상태
영속(managed)	<ul style="list-style-type: none"><li>- 엔티티가 영속성 컨텍스트에 저장된 상태로 영속성 컨텍스트에 의해 관리되는 상태</li><li>- 영속 상태에서 데이터베이스에 저장되지 않으며, 트랜잭션 커밋 시점에 데이터베이스에 반영</li></ul>
준영속 상태(detached)	영속성 컨텍스트에 엔티티가 저장되었다가 분리된 상태
삭제 상태(removed)	영속성 컨텍스트와 데이터베이스에서 삭제된 상태

# JPA 동작 방식

---



[그림 2-3] 엔티티 생명주기

---

# JPA 동작 방식

---

```
01 Item item = new Item(); ..... ①
02 item.setItemNm("테스트 상품");
03
04 EntityManager em = entityManagerFactory.createEntityManager(); ..... ②
05
06 EntityTransaction transaction = em.getTransaction(); ..... ③
07 transaction.begin();
08
09 em.persist(item); ..... ④
10
11 transaction.commit(); ..... ⑤
12
13 em.close(); ..... ⑥
14 emf.close(); ..... ⑦
```

상품 엔티티 영속성 컨텍스트 저장 예시 코드

---

# 영속성 컨텍스트



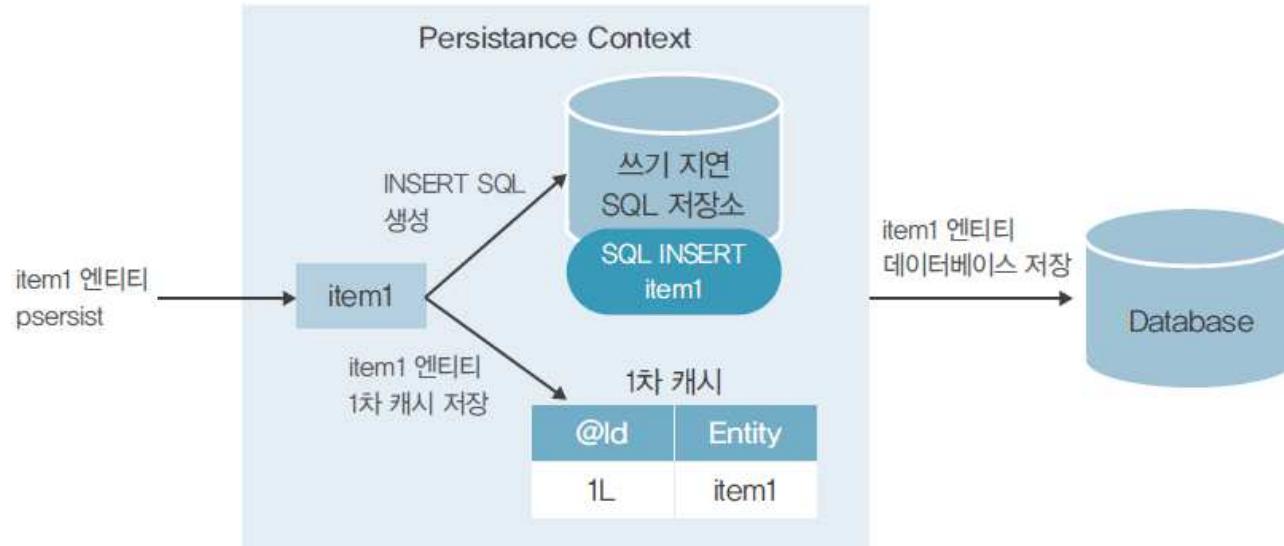
[그림 2-4] 영속성 컨텍스트 1차 캐시 구조

\* 1차 캐시 : 영속성 컨텍스트에 Map<KEY, VALUE>로 저장.

Find() 메소드 호출 시 영속성 컨텍스트의 1차 캐시 조회

\* 동일성 보장 : 하나의 트랜잭션에서 같은 키 값으로 영속성 컨텍스트 조회 시  
같은 엔티티 조회 보장 (1차 캐시)

# 영속성 컨텍스트



[그림 2-5] 영속성 컨텍스트 쓰기 지연 SQL 저장소

- \* 트랜잭션을 지원하는 쓰기 지연 : 쓰기 지연 SQL 저장소에 SQL을 쌓아두고 트랜잭션 커밋 시점에 저장된 SQL 문 flush하여 데이터베이스 반영.
- \* 변경 감지 : 1차 캐시에 데이터베이스에서 처음 불러온 엔티티의 스냅샷 저장 및 커밋 시점에 변경 내용을 반영. 즉, update문을 호출하지 않아도 됨.

## 2.2 쇼핑몰 프로젝트 생성

<https://start.spring.io>

The screenshot shows the Spring Initializr web application interface for generating a Spring Boot project. The page has a header with the Spring logo and the text "spring initializr".

**Project** section:

- Maven Project (radio button selected)
- Gradle Project

**Language** section:

- Java (radio button selected)
- Kotlin
- Groovy

**Dependencies** section:

No dependency selected.

**ADD DEPENDENCIES... CTRL + B** button (highlighted with a blue border).

**Spring Boot** section:

- 2.4.0 (SNAPSHOT)
- 2.4.0 (M2)
- 2.3.4 (SNAPSHOT) (radio button selected)
- 2.3.3
- 2.2.10 (SNAPSHOT)
- 2.2.9
- 2.1.17 (SNAPSHOT)
- 2.1.16

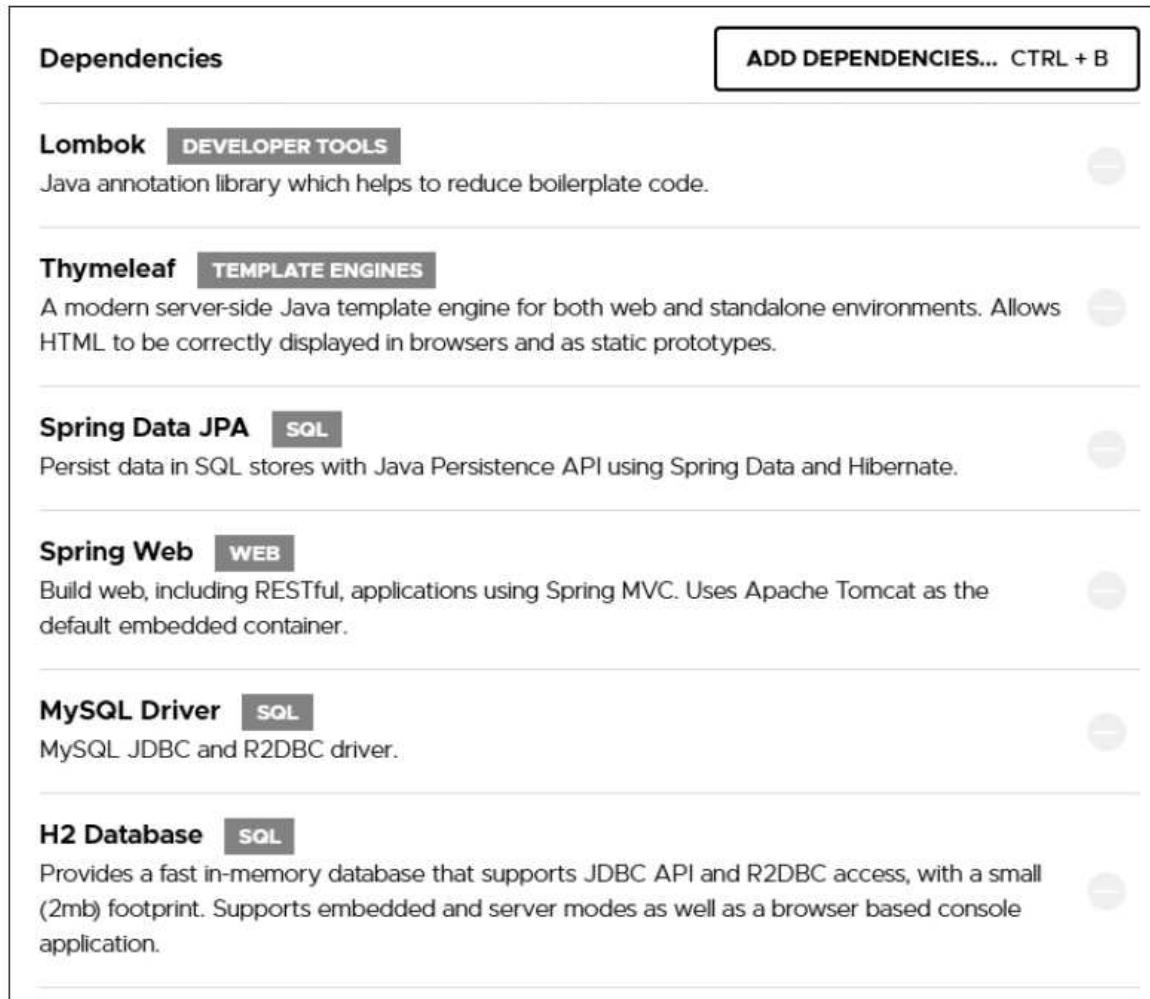
**Project Metadata** section:

- Group: com.shop
- Artifact: shop
- Name: shop
- Description: Shop project for Spring Boot
- Package name: com.shop
- Packaging: Jar (radio button selected)
- Java: 14 (radio button selected)
- 11
- 8

[그림 2-6] 스프링 부트 프로젝트 생성

## 2.2 쇼핑몰 프로젝트 생성

---



[그림 2-7] 스프링 부트 프로젝트 Dependencies 추가

---

# application.properties 설정

---

## 2.2.2 application.properties 설정하기

(src/main/resources/application.properties)

```
01 #애플리케이션 포트 설정
02 server.port=80
03
04 #MySQL 연결 설정
05 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver      1
06 spring.datasource.url=jdbc:mysql://localhost:3306/shop?serverTimezone=UTC 2
07 spring.datasource.username=root                                     3
08 spring.datasource.password=1234                                     4
09
10 #실행되는 쿼리 콘솔 출력
11 spring.jpa.properties.hibernate.show_sql=true
12
```

# application.properties 설정

---

## 2.2.2 application.properties 설정하기

```
13 #콘솔창에 출력되는 쿼리를 가독성이 좋게 포맷팅  
14 spring.jpa.properties.hibernate.format_sql=true  
15  
16 #쿼리에 물음표로 출력되는 바인드 파라미터 출력  
17 logging.level.org.hibernate.type.descriptor.sql=trace  
18  
19 spring.jpa.hibernate.ddl-auto=create ..... ⑤  
20 spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect ..... ⑥
```

## 데이터베이스 초기화 전략 DDL AUTO 옵션

---

### \* 데이터베이스 초기화 전략-DDL AUTO 옵션

- none: 사용하지 않음
- create: 기존 테이블 삭제 후 테이블 생성
- create- drop: 기존 테이블 삭제 후 테이블 생성. 종료 시점에 테이블 삭제
- update: 변경된 스키마 적용
- validate: 엔티티와 테이블 정상 매핑 확인

## 2.3 상품 엔티티 설계

---

- \* 상품의 상태를 나타내는 ItemSellStatus ENUM 생성



[함께 해봐요\_2-1] 상품 클래스 생성하기\_Ver01

(com.shop.constant.ItemSellStatus.java)

```
01 package com.shop.constant;  
02  
03 public enum ItemSellStatus {  
04     SELL, SOLD_OUT  
05 }
```

## 2.3 상품 엔티티 설계

### \* Item 클래스 생성

(com.shop.entity.Item.java)

```
01 package com.shop.entity;
02
03 import com.shop.constant.ItemSellStatus;
04 import lombok.Getter;
05 import lombok.Setter;
06 import lombok.ToString;
07 import java.time.LocalDateTime;
08
09 @Getter
10 @Setter
11 @ToString
12 public class Item {
13
14     private Long id;          //상품 코드
15
16     private String itemNm;    //상품명
17
18     private int price;        //가격
19
20     private int stockNumber;  //재고수량
21
22     private String itemDetail; //상품 상세 설명
23
24     private ItemSellStatus itemSellStatus; //상품 판매 상태
25
26     private LocalDateTime regTime; //등록 시간
27
28     private LocalDateTime updateTime; //수정 시간
29 }
```

## 2.3 상품 엔티티 설계

---

[표 2-1] 엔티티 매핑 관련 어노테이션

어노테이션	설명
@Entity	클래스를 엔티티로 선언
@Table	엔티티와 매핑할 테이블을 지정
@Id	테이블의 기본키에 사용할 속성을 지정
@GeneratedValue	키 값을 생성하는 전략 명시
@Column	필드와 컬럼 매핑
@Lob	BLOB, CLOB 타입 매핑 (용어 설명 참조)
@CreationTimestamp	insert 시 시간 자동 저장
@UpdateTimestamp	update 시 시간 자동 저장
@Enumerated	enum 타입 매핑
@Transient	해당 필드 데이터베이스 매핑 무시
@Temporal	날짜 타입 매핑
@CreateDate	엔티티가 생성되어 저장될 때 시간 자동 저장
@LastModifiedDate	조회한 엔티티의 값을 변경할 때 시간 자동 저장

---

## 2.3 상품 엔티티 설계

[표 2-2] @Column 어노테이션 추가 속성

속성	설명	기본값
name	필드와 매핑할 컬럼의 이름 설정	객체의 필드 이름
unique(DDL)	유니크 제약 조건 설정	
insertable	insert 가능 여부	true
updatable	update 가능 여부	true
length	String 타입의 문자 길이 제약조건 설정	255
nullable(DDL)	null 값의 허용 여부 설정. false 설정 시 DDL 생성 시에 not null 제약조건 추가	
columnDefinition	데이터베이스 컬럼 정보 직접 기술  예 <code>@Column(columnDefinition = "varchar(5) default'10' not null")</code>	
precision, scale(DDL)	BigDecimal 타입에서 사용(BigInteger 가능) precision은 소수점을 포함한 전체 자리수이고, scale은 소수점 자리수. Double과 float 타입에는 적용되지 않음.	

## 2.3 상품 엔티티 설계

---

- \* @GeneratedValue 어노테이션을 통한 기본 키 생성 전략

생성 전략	설명
GenerationType.AUTO (default)	JPA 구현체가 자동으로 생성 전략 결정
GenerationType.IDENTITY	기본키 생성을 데이터베이스에 위임 예 MySql 데이터베이스의 경우 AUTO_INCREMENT를 사용하여 기본키 생성
GenerationType.SEQUENCE	데이터베이스 시퀀스 오브젝트를 이용한 기본키 생성 @SequenceGenerator를 사용하여 시퀀스 등록 필요
GenerationType.TABLE	키 생성용 테이블 사용. @TableGenerator 필요

## 2.3 상품 엔티티 설계



[함께 해봐요\_2-2] 상품 클래스 엔티티 매핑\_Ver02

(com.shop.entity.Item.java)

```
01 package com.shop.entity;                                25 @Column(nullable = false, length = 50) ..... ③
02
03 import com.shop.constant.ItemSellStatus;               26 private String itemNm; //상품명
04 import lombok.Getter;                                 27
05 import lombok.Setter;                               28 @Column(name="price", nullable = false)
06 import lombok.ToString;                            29 private int price; //가격
07 import org.springframework.data.annotation.CreatedDate; 30
08 import org.springframework.data.annotation.LastModifiedDate; 31 @Column(nullable = false)
09
10 import javax.persistence.*;                         32 private int stockNumber; //재고수량
11 import java.time.LocalDateTime;                    33
12
13 @Entity                                         34 @Lob
14 @Table(name="item")                                35 @Column(nullable = false)
15 @Getter                                         36 private String itemDetail; //상품 상세 설명
16 @Setter                                         37
17 @ToString
18 public class Item {
19
20     @Id
21     @Column(name="item_id")
22     @GeneratedValue(strategy = GenerationType.AUTO) ..... ②
23     private Long id;      //상품 코드
```

## 2.4 Repository 설계

---

- Data Access Object의 역할을 하는 Repository 인터페이스 설계
- 첫 번째 제네릭 타입에는 엔티티 타입 클래스, 두 번째 제네릭 타입에는 클래스의 기본 키 타입 세팅



[함께 해봐요\_2-3] 상품 Repository 작성 및 테스트하기

(com.shop.repository.ItemRepository.java)

```
01 package com.shop.repository;  
02  
03 import com.shop.entity.Item;  
04 import org.springframework.data.jpa.repository.JpaRepository;  
05  
06 public interface ItemRepository extends JpaRepository<Item, Long> {  
07  
08 }
```

## 2.4 Repository 설계

---

- JpaRepository는 기본적인 CRUD 및 페이징 처리를 위한 메소드가 정의돼 있음.

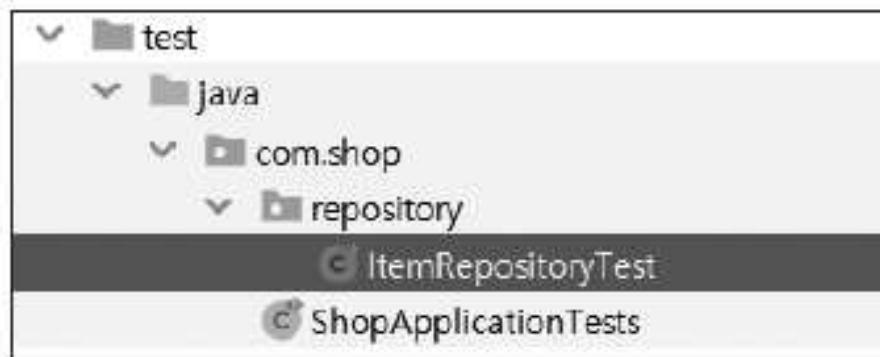
[표 2-3] JpaRepository에서 지원하는 메소드 예시

메소드	기능
<code>&lt;S extends T&gt; save(S entity)</code>	엔티티 저장 및 수정
<code>void delete(T entity)</code>	엔티티 삭제
<code>count()</code>	엔티티 총 개수 반환
<code>Iterable&lt;T&gt; findAll()</code>	모든 엔티티 조회

# 상품 저장 테스트

---

- test.java.com.shop.repository 패키지 아래에 ItemRepositoryTest.java 파일 생성



[그림 2-20] ItemRepositoryTest.java 생성 완료

---

# 상품 저장 테스트

---

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03 import com.shop.constant.ItemSellStatus;  
04 import com.shop.entity.Item;  
05 import org.junit.jupiter.api.DisplayName;  
06 import org.junit.jupiter.api.Test;  
07 import org.springframework.beans.factory.annotation.Autowired;  
08 import org.springframework.boot.test.context.SpringBootTest;  
09 import org.springframework.test.context.TestPropertySource;  
10  
11 import java.time.LocalDateTime;  
12  
13 @SpringBootTest ..... ❶  
14 @TestPropertySource(locations="classpath:application-test.properties") ..... ❷  
15 class ItemRepositoryTest {  
16  
17     @Autowired  
18     ItemRepository itemRepository; ..... ❸
```

# 상품 저장 테스트

```
20     @Test .....  
21     @DisplayName("상품 저장 테스트") .....  
22     public void createItemTest(){ .....  
23         Item item = new Item(); .....  
24         item.setItemNm("테스트 상품"); .....  
25         item.setPrice(10000); .....  
26         item.setItemDetail("테스트 상품 상세 설명"); .....  
27         item.setItemSellStatus(ItemSellStatus.SELL); .....  
28         item.setStockNumber(100); .....  
29         item.setRegTime(LocalDateTime.now()); .....  
30         item.setUpdateTime(LocalDateTime.now()); .....  
31         Item savedItem = itemRepository.save(item); .....  
32         System.out.println(savedItem.toString()); .....  
33     } .....  
34 }
```

4

5

## 2.5 쿼리 메소드

---

- 쿼리 메소드는 스프링 데이터 JPA에서 제공하는 핵심 기능 중 하나로 Repository 인터페이스에 간단한 네이밍 룰을 이용하여 메소드를 작성하면 원하는 쿼리 실행 가능.
- 쿼리 메소드를 이용할 때 가장 많이 사용하는 문법으로 find를 사용.
- 엔티티의 이름은 생략이 가능하며, By 뒤에는 검색할 때 사용할 변수의 이름을 적어준다.

```
find + (엔티티 이름) + By + 변수이름
```

## 2.5 쿼리 메소드

---



[함께 해봐요\_2-4] 쿼리 메소드를 이용한 상품 조회하기

(com.shop.repository.ItemRepository.java)

```
01 package com.shop.repository;  
02  
03 import com.shop.entity.Item;  
04 import org.springframework.data.jpa.repository.JpaRepository;  
05  
06 import java.util.List;  
07  
08 public interface ItemRepository extends JpaRepository<Item, Long> {  
09  
10     List<Item> findByItemNm(String itemNm); ..... ①  
11  
12 }
```

## 2.5 쿼리 메소드

---

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03 .....기존 임포트 생략.....  
04  
05 import java.util.List;  
06  
07 @SpringBootTest  
08 @TestPropertySource(locations="classpath:application-test.properties")  
09 class ItemRepositoryTest {  
10  
11     @Autowired  
12     ItemRepository itemRepository;  
13  
14     .....코드 생략.....  
15 }
```

## 2.5 쿼리 메소드

```
16     public void createItemList(){ ..... ①
17         for(int i=1;i<=10;i++){
18             Item item = new Item();
19             item.setItemNm("테스트 상품" + i);
20             item.setPrice(10000 + i);
21             item.setItemDetail("테스트 상품 상세 설명" + i);
22             item.setItemSellStatus(ItemSellStatus.SELL);
23             item.setStockNumber(100); item.setRegTime(LocalDateTime.now());
24             item.setUpdateTime(LocalDateTime.now());
25             Item savedItem = itemRepository.save(item);
26         }
27     }
28
29     @Test
30     @DisplayName("상품명 조회 테스트")
31     public void findByItemNmTest(){
32         this.createItemList();
33         List<Item> itemList = itemRepository.findByItemNm("테스트 상품1"); ②
34         for(Item item : itemList){
35             System.out.println(item.toString()); ③
36         }
37     }
38
39 }
```

## 2.5 쿼리 메소드

---

[표 2-4] 쿼리 메소드 Sample 및 JPQL snippet

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname findByFirstnames findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
Greater Than	findByAgeGreaterThan	... where x.age > ?1
Greater Than Equal	findByAgeGreaterThanEqual	... where x.age >= ?1

## 2.5 쿼리 메소드

[표 2-4] 쿼리 메소드 Sample 및 JPQL snippet

After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull, Null IsNotNull	findByAge(Is)Null	... where x.age is null
NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc

## 2.5 쿼리 메소드

---

[표 2-4] 쿼리 메소드 Sample 및 JPQL snippet

Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

## 2.5 쿼리 메소드 OR 조건

---



[함께 해봐요\_2-5] OR 조건 처리하기

(com.shop.repository.ItemRepository.java)

```
01 public interface ItemRepository extends JpaRepository<Item, Long> {  
02  
03     ....코드 생략....  
04  
05     List<Item> findByItemNmOrItemDetail(String itemNm, String itemDetail); ❶  
06  
07 }
```

## 2.5 쿼리 메소드 OR 조건

(com.shop.repository.itemRepositoryTest.java)

```
01 @SpringBootTest
02 @TestPropertySource(locations="classpath:application-test.properties")
03 class ItemRepositoryTest {
04
05     .....코드 생략.....
06
07     @Test
08     @DisplayName("상품명, 상품상세설명 or 테스트")
09     public void findByItemNmOrItemDetailTest(){
10         this.createItemList();           ❶
11         List<Item> itemList =
12             itemRepository.findByItemNmOrItemDetail("테스트 상품1", "테스트 상품 상세 설명5"); ❷
13         for(Item item : itemList){
14             System.out.println(item.toString());
15         }
16     }
17 }
```

## 2.5 쿼리 메소드 LessThan 조건

---



[함께 해봐요 2-6] LessThan 조건 처리하기

com.shop.repository.itemRepository.java

```
01 public interface ItemRepository extends JpaRepository<Item, Long> {  
02  
03     ....코드 생략....  
04  
05     List<Item> findByPriceLessThan(Integer price); ①  
06  
07 }
```

## 2.5 쿼리 메소드 LessThan 조건

com.shop.repository.ItemRepositoryTest.java

```
01 @SpringBootTest
02 @TestPropertySource(locations="classpath:application-test.properties")
03 class ItemRepositoryTest {
04
05     .....코드 생략.....
06
07     @Test
08     @DisplayName("가격 LessThan 테스트")
09     public void findByPriceLessThanTest(){
10         this.createItemList();
11         List<Item> itemList = itemRepository.findByPriceLessThan(10005); ❶
12         for(Item item : itemList){
13             System.out.println(item.toString());
14         }
15     }
16 }
```

## 2.5 쿼리 메소드 OrderBy 조건

---



[함께 해봐요 2-7] OrderBy로 정렬 처리하기

com.shop.repository.ItemRepository.java

```
01 public interface ItemRepository extends JpaRepository<Item, Long> {  
02  
03     ....코드 생략....  
04  
05     List<Item> findByPriceLessThanOrderByPriceDesc(Integer price);  
06 }
```

## 2.5 쿼리 메소드 OrderBy 조건

com.shop.repository.ItemRepositoryTest.java

```
01 @SpringBootTest
02 @TestPropertySource(locations="classpath:application-test.properties")
03 class ItemRepositoryTest {
04
05     ....코드 생략....
06
07     @Test
08     @DisplayName("가격 내림차순 조회 테스트")
09     public void findByPriceLessThanOrderByPriceDesc(){
10         this.createItemList();
11         List<Item> itemList =
12             itemRepository.findByPriceLessThanOrderByPriceDesc(10005);
13         for(Item item : itemList){
14             System.out.println(item.toString());
15         }
16     }
17 }
```

## 2.6 SPRING DATA JPA @Query 어노테이션

---

- 조건이 많아질 때 쿼리 메소드를 선언하면 이름이 길어져 오히려 보기 힘든 단점이 있음.
- @Query 어노테이션을 이용하면 SQL과 유사한 JPQL (Java Persistence Query Language)이라는 객체지향 쿼리 언어로 복잡한 쿼리 처리 가능.
- JPQL은 엔티티 객체를 대상으로 쿼리를 수행하는 객체지향 쿼리.
- JPQL은 SQL을 추상화해서 사용하기 때문에 특정 데이터베이스 SQL에 의존하지 않음.

## 2.6 SPRING DATA JPA @Query 어노테이션



[함께 해봐요 2-8] @Query를 이용한 검색 처리하기

com.shop.repository.ItemRepository.java

```
01 package com.shop.repository;  
02  
03 .....기존 임포트 생략.....  
04  
05 import org.springframework.data.jpa.repository.Query;  
06 import org.springframework.data.repository.query.Param;  
07  
08 public interface ItemRepository extends JpaRepository<Item, Long> {  
09  
10     ....코드 생략.....  
11  
12     @Query("select i from Item i where i.itemDetail like  
13         %:itemDetail% order by i.price desc") ①  
14     List<Item> findByItemDetail(@Param("itemDetail") String itemDetail); ②  
15 }
```

## 2.6 SPRING DATA JPA @Query 어노테이션

---

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03 .....기존 임포트 생략.....  
04  
05 @SpringBootTest  
06 @TestPropertySource(locations="classpath:application-test.properties")  
07 class ItemRepositoryTest {  
08  
09     ....코드 생략....  
10  
11     @Test  
12     @DisplayName("@Query를 이용한 상품 조회 테스트")  
13     public void findByItemDetailTest(){  
14         this.createItemList();  
15         List<Item> itemList = itemRepository.findByItemDetail("테스트 상품 상세 설명");  
16         for(Item item : itemList){  
17             System.out.println(item.toString());  
18         }  
19     }  
20  
21 }
```

## 2.6 @Query-nativeQuery 속성 예제

---

- @Query의 nativeQuery 속성을 사용하면 기존 쿼리를 그대로 활용 가능
- 특정 데이터베이스에 종속되는 쿼리문을 사용하기 때문에 데이터베이스에 대해 독립적이라는 장점을 잃어버림.
- 기존에 작성한 통계용 쿼리처럼 복잡한 쿼리를 그대로 사용해야 하는 경우 활용 가능.

## 2.6 @Query-nativeQuery 속성 예제



[함께 해봐요 2-9] @Query-nativeQuery 속성 예제

com.shop.repository.ItemRepository.java

```
01 package com.shop.repository;  
02  
03     ....기존 임포트 생략.....  
04  
05 public interface ItemRepository extends JpaRepository<Item, Long> {  
06  
07     ....코드 생략.....  
08  
09     @Query(value="select * from item i where i.item_detail like  
10         %:itemDetail% order by i.price desc", nativeQuery = true) .....❶  
11     List<Item> findByItemDetailByNative(@Param("itemDetail") String itemDetail);  
12 }
```

## 2.6 @Query-nativeQuery 속성 예제

---

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03 .....기존 임포트 생략.....  
04  
05 @SpringBootTest  
06 @TestPropertySource(locations="classpath:application-test.properties")  
07 class ItemRepositoryTest {  
08  
09     ....코드 생략....  
10  
11     @Test  
12     @DisplayName("nativeQuery 속성을 이용한 상품 조회 테스트")  
13     public void findByItemDetailByNative(){  
14         this.createItemList();  
15         List<Item> itemList =  
16             itemRepository.findByItemDetailByNative("테스트 상품 상세 설명");  
17         for(Item item : itemList){  
18             System.out.println(item.toString());  
19         }  
20     }  
21 }
```

## 2.7 SPRING DATA JPA Querydsl

---

- Querydsl은 JPQL을 코드로 작성할 수 있도록 도와주는 빌더 API
- 문자열 아닌 자바 소스코드로 작성하므로 컴파일 시점에 오류 발견 가능
- 비슷한 쿼리를 재사용할 수 있어 제약 조건 조립 및 가독성 향상
- 고정된 SQL문이 아닌 조건에 맞게 동적으로 쿼리를 생성
- IDE의 도움으로 자동 완성 기능 이용하여 생산성 향상

## 2.7 SPRING DATA JPA Querydsl

---

- Querydsl을 사용하기 위한 의존성 추가

pom.xml

```
01 <dependency>
02   <groupId>com.querydsl</groupId>
03   <artifactId>querydsl-jpa</artifactId>
04   <version>4.3.1</version>
05 </dependency>
06 <dependency>
07   <groupId>com.querydsl</groupId>
08   <artifactId>querydsl-apt</artifactId>
09   <version>4.3.1</version>
10 </dependency>
```

## 2.7 SPRING DATA JPA Querydsl

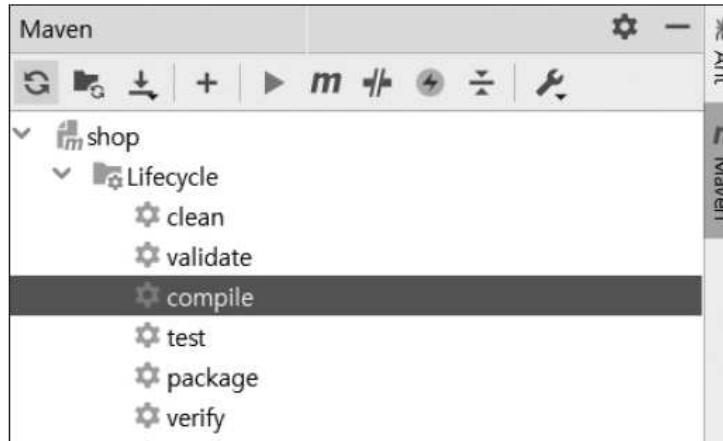
---

pom.xml

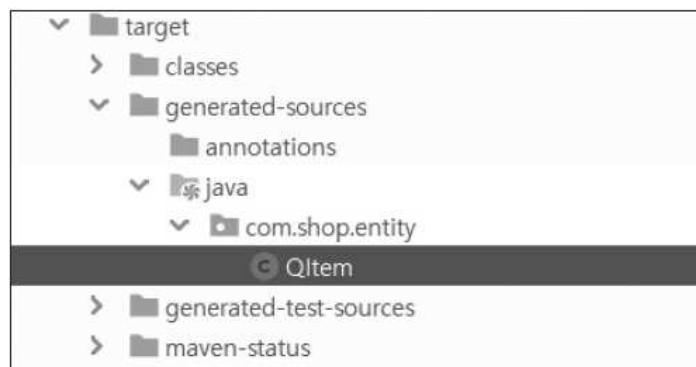
```
01 <plugin>
02   <groupId>com.mysema.maven</groupId>
03   <artifactId>apt-maven-plugin</artifactId>
04   <version>1.1.3</version>
05   <executions>
06     <execution>
07       <goals>
08         <goal>process</goal>
09       </goals>
10       <configuration>
11         <outputDirectory>target/generated-sources/java
12           </outputDirectory>
13         <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor
14           </processor>
15       </configuration>
16     </execution>
17   </executions>
18 </plugin>
```

## 2.7 SPRING DATA JPA Querydsl

---



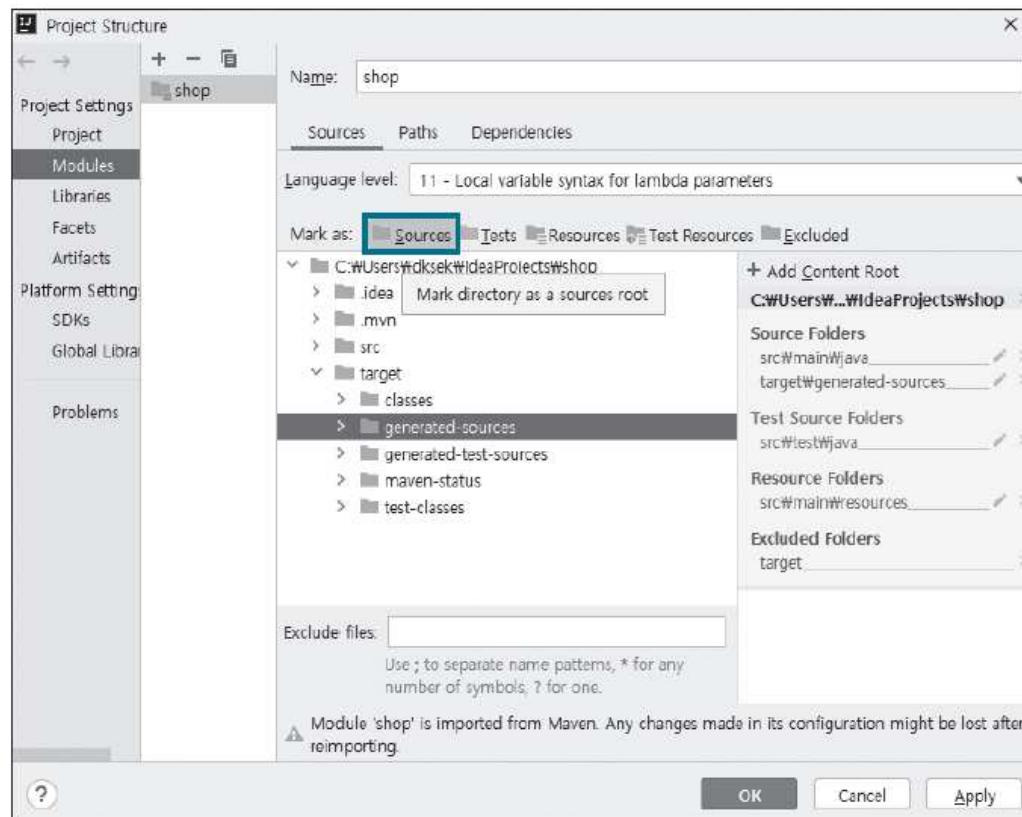
[그림 2-29] 메이븐 컴파일 수행



[그림 2-30] 메이븐 컴파일 수행 결과 QItem 클래스 생성

## 2.7 SPRING DATA JPA Querydsl

- QDomain 임포트가 안 될 경우, target 폴더 아래의 generated-sources 폴더를 클릭하여 <Sources> 버튼을 클릭해 소스코드로 인식할 수 있게 처리



[그림 2-31] generated-sources 폴더 Sources 처리

## 2.7 SPRING DATA JPA QueryDSL



[함께 해봐요 2-10] JPQLQueryFactory를 이용한 상품 조회 예제

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03     ....기존 임포트 생략.....  
04  
05 import com.querydsl.jpa.impl.JPQLQueryFactory;  
06 import com.querydsl.jpa.impl.JPQLQuery;  
07 import com.shop.entity.QItem;  
08 import javax.persistence.PersistenceContext;  
09 import javax.persistence.EntityManager;  
10  
11 @SpringBootTest  
12 @TestPropertySource(locations="classpath:application-test.properties")  
13 class ItemRepositoryTest {  
14  
15     @PersistenceContext  
16     EntityManager em; ..... ①  
17  
18     ....코드 생략.....  
19
```

## 2.7 SPRING DATA JPA Querydsl

---

```
20     @Test
21     @DisplayName("Querydsl 조회 테스트1")
22     public void queryDslTest(){
23         this.createItemList();
24         JPAQueryFactory queryFactory = new JPAQueryFactory(em); ..... ②
25         QItem qItem = QItem.item; ..... ③
26         JPAQuery<Item> query = queryFactory.selectFrom(qItem) ..... ④
27             .where(qItem.itemSellStatus.eq(ItemSellStatus.SELL))
28             .where(qItem.itemDetail.like("%" + "테스트 상품 상세 설명" + "%"))
29             .orderBy(qItem.price.desc());
30
31         List<Item> itemList = query.fetch(); ..... ⑤
32
33         for(Item item : itemList){
34             System.out.println(item.toString());
35         }
36     }
```

## 2.7 SPRING DATA JPA Querydsl

---

[표 2-5] JPQL 데이터 반환 메소드

메소드	기능
List<T> fetch()	조회 결과 리스트 반환
T fetchOne	조회 대상이 1건인 경우 제네릭으로 지정한 타입 반환
T fetchFirst()	조회 대상 중 1건만 반환
Long fetchCount()	조회 대상 개수 반환
QueryResult<T> fetchResults()	조회한 리스트와 전체 개수를 포함한 QueryResults 반환

## 2.7 QuerydslPredicateExecutor 상품 조회 예제



[함께 해봐요 2-11] QuerydslPredicateExecutor를 이용한 상품 조회 예제

com.shop.repository.ItemRepository.java

```
01 package com.shop.repository;  
02  
03     ....기존 임포트 생략.....  
04  
05 import org.springframework.data.querydsl.QuerydslPredicateExecutor;  
06  
07 public interface ItemRepository extends JpaRepository<Item, Long>,  
    QuerydslPredicateExecutor<Item> {  
08  
09     ....코드 생략.....  
10  
11 }
```

1

## 2.7 QuerydslPredicateExecutor 상품 조회 예제

---

[표 2-6] QueryDslPredicateExecutor 인터페이스 정의 메소드

메소드	기능
long count(Predicate)	조건에 맞는 데이터의 총 개수 반환
boolean exists(Predicate)	조건에 맞는 데이터 존재 여부 반환
Iterable findAll(Predicate)	조건에 맞는 모든 데이터 반환
Page<T> findAll(Predicate, Pageable)	조건에 맞는 페이지 데이터 반환
Iterable findAll(Predicate, Sort)	조건에 맞는 정렬된 데이터 반환
T findOne(Predicate)	조건에 맞는 데이터 1개 반환

## 2.7 QuerydslPredicateExecutor 상품 조회 예제

---

com.shop.repository.ItemRepositoryTest.java

```
01 package com.shop.repository;  
02  
03     ....기존 임포트 생략.....  
04  
05 import com.querydsl.core.BooleanBuilder;  
06 import org.springframework.data.domain.Page;  
07 import org.springframework.data.domain.PageRequest;  
08 import org.springframework.data.domain.Pageable;  
09 import org.thymeleaf.util.StringUtils;  
10  
11 @SpringBootTest  
12 @TestPropertySource(locations="classpath:application-test.properties")  
13 class ItemRepositoryTest {  
14  
15     ....코드 생략....
```

## 2.7 QuerydslPredicateExecutor 상품 조회 예제

---

```
17     public void createItemList2(){  
18         for(int i=1;i<=5;i++){  
19             Item item = new Item();  
20             item.setItemNm("테스트 상품" + i);  
21             item.setPrice(10000 + i);  
22             item.setItemDetail("테스트 상품 상세 설명" + i);  
23             item.setItemSellStatus(ItemSellStatus.SELL);  
24             item.setStockNumber(100);  
25             item.setRegTime(LocalDateTime.now());  
26             item.setUpdateTime(LocalDateTime.now());  
27             itemRepository.save(item);  
28     }
```

## 2.7 QuerydslPredicateExecutor 상품 조회 예제

---

```
17     public void createItemList2(){ ..... ①
18         for(int i=1;i<=5;i++){
19             Item item = new Item();
20             item.setItemNm("테스트 상품" + i);
21             item.setPrice(10000 + i);
22             item.setItemDetail("테스트 상품 상세 설명" + i);
23             item.setItemSellStatus(ItemSellStatus.SELL);
24             item.setStockNumber(100);
25             item.setRegTime(LocalDateTime.now());
26             item.setUpdateTime(LocalDateTime.now());
27             itemRepository.save(item);
28         }
29
30         for(int i=6;i<=10;i++){
31             Item item = new Item();
32             item.setItemNm("테스트 상품" + i);
33             item.setPrice(10000 + i);
34             item.setItemDetail("테스트 상품 상세 설명" + i);
35             item.setItemSellStatus(ItemSellStatus.SOLD_OUT);
36             item.setStockNumber(0);
37             item.setRegTime(LocalDateTime.now());
38             item.setUpdateTime(LocalDateTime.now());
39             itemRepository.save(item);
40         }
41     }
```

## 2.7 QuerydslPredicateExecutor 상품 조회 예제

```
43     @Test
44     @DisplayName("상품 Querydsl 조회 테스트 2")
45     public void queryDslTest2(){
46
47         this.createItemList2();
48
49         BooleanBuilder booleanBuilder = new BooleanBuilder(); .....❷
50         QItem item = QItem.item;
51         String itemDetail = "테스트 상품 상세 설명";
52         int price = 10003;
53         String itemSellStat = "SELL";
54
55         booleanBuilder.and(item.itemDetail.like("%" + itemDetail + "%")); ❸
56         booleanBuilder.and(item.price.gt(price));
57
58         if(StringUtils.equals(itemSellStat, ItemSellStatus.SELL)){
59             booleanBuilder.and(item.itemSellStatus.eq(ItemSellStatus.SELL));
60         }
61
62         Pageable pageable = PageRequest.of(0, 5); .....❹
63         Page<Item> itemPagingResult =
64         itemRepository.findAll(booleanBuilder, pageable); .....❺
65         System.out.println("total elements : " +
66                         itemPagingResult.getTotalElements ());
67
68         List<Item> resultItemList = itemPagingResult.getContent();
69         for(Item resultItem: resultItemList){
70             System.out.println(resultItem.toString());
71         }
72     }
73 }
```