JPA에서는 Model(Bean)에서 데이터 테이블을 만들게 된다.

1. 스프링 JPA

가. 스프링 부트에서 JPA를 통해서 데이터베이스와 연동을 처리해 본다.

나. 스프링 JPA(Java Persistence API)는 자바 관련 기술의 하나이다. 엔티티빈 자바클래스를 통해서 데이터베이스 쿼리문을 생성하고 다루는 기술을 말한다. 별도의 SQL문을 작성하지 않고도 자바클래스를 통해서 쿼리문을 다룬다.

SQL문을 완전히 커버하지는 못한다.

다. 스프링 JPA를 다룰 때 MyBatis가 아닌 하이버네이트라는 프레임워크를 통해서 쿼리문을 다루어 본다.

2. 스프링에서 JPA를 다루기 위해서는 application.properties 파일에 다음과 같은 설정을 해줘야 한다.

```
중략..
..
#table create/update =>create는 기존테이블을 삭제후 다시 생성, update는 변경된 부분
#만 반영
spring.jpa.hibernate.ddl-auto=update
#ddl => DDL 데이터 정의어인 create,alter(테이블 수정문),drop,truncate(전체 레코드 삭
#제문),rename(테이블명,컬럼명 변경문)문 사용시 데이터베이스 고유 기능을
# 사용하겠는가?
spring.jpa.generate-ddl=true
#sql show => 실행되는 SQL문을 보여줄 것인가?
spring.jpa.show-sql=true
#database select => 데이터베이스 선택
spring.jpa.database=oracle
#log
logging.level.org.hibernate=info
#oracle 상세지정
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
```

데이터 정의어

버전을 맞춰서 진행해야함

3. Lombok을 이용하면 setter/getter 등의 메서드를 자동으로 생성하기 때문에 약간의 코드만으로도 클래스를 작성할 수 있다.

net.daum.vo 패키지를 작성하고 Sample2VO.java 클래스를 다음과 같이 작성한다.

```
package net.daum.vo;

import lombok.Data;
import lombok.ToString;

//@Getter //getter()메서드 자동생성
//@Setter //setter()메서드 자동생성
//@ToString //toString()메서드 자동생성
```

Data를 쓰던가 위에것을 쓰던가 선택

```
@Data //getter(),setter(),기본생성자(),canEqual(),equals(),hashCode()메서드까지 한꺼//번에 자동생성
@ToString(exclude= {"val3"}) //exclude속성을 사용하여 val3변수를 제외
public class Sample2VO {

	private String val1;
	private String val2;
	private String val3;
}
```

4. 엔티티빈 클래스를 설계해 본다.

net.daum.vo 패키지를 작성하고 BoardVO.java 클래스를 다음과 같이 작성한다.

```
package net.daum.vo;

import java.sql.Timestamp;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Table;
import lombok.Getter;
import lombok.Setter;
```

```
import lombok.ToString;

@Getter
@Setter
@ToString
@Entity //JPA를 다루는 엔티티빈 클래스
@SequenceGenerator(//@SequenceGenerator는 시퀀스 생성기를 설정하는 애노테이션
                name="bno_seq8_gename", //시퀀스 제너레이터 이름
                sequenceName="bno_seq8", //시퀀스 이름
                initialValue=1, //시작값
                allocationSize=1 //메모리를 통해 할당할 범위 사이즈=>기본값은 50이며,
//1로 설정하는 경우 매번 insert시마다 DB의 시퀀스를 호출해서
                //db시퀀스 번호값을 가져와서 1증가한 값이 할당된다.증가값. 1씩 증가.
                )
@Table(name="tbl_boards3") //tbl_boards3테이블을 생성
public class BoardVO {

        @Id //기본키 컬럼
        //@GeneratedValue(strategy = GenerationType.AUTO)
        //특정키에 맞게 식별키를 자동생성
        @GeneratedValue(
                        strategy=GenerationType.SEQUENCE, //사용할 전략을 시퀀스
로  선택
                        generator="bno_seq8_gename" //시퀀스 생성기에 설정해놓은
//bno_seq8_gename 시퀀스 제너레이터 이름 으로 설정
                        )
        private int bno;//번호
        private String writer;//작성자
        private String title;//제목

        @Column(length=4000) //컬럼 크기를 4000으로 설정
        private String content;//내용

                        hibernate랑 관련있는 에노테이션
        @CreationTimestamp
        private Timestamp regdate;//등록날짜

        @UpdateTimestamp
        //@CreationTiestamp 와 @UpdateTimestamp는 하이버네이트의 특별한 기능으로
//엔티티 빈 생성,수정시점,등록시점 날짜값을 기록
        private Timestamp updatedate;//수정날짜
```

```
}
```

5. JPA 처리를 담당하는 Repository 인터페이스를 설계해 본다.

일반적으로 과거에는 DAO라는 개념을 이용했듯이, JPA를 이용하는 경우에는 Repository라는 용어를 사용한다. net.daum.dao 패키지를 작성하고 BoardRepository.java 인터페이스를를 다음과 같이 작성한다.

```
package net.daum.dao;

import org.springframework.data.repository.CrudRepository;

import net.daum.vo.BoardVO;

public interface BoardRepository extends CrudRepository<BoardVO, Integer> {

        /*JpaRepository 인터페이스의 부모 인터페이스인 PagingAndSortingRepository에서 페이징과 정렬이라는 기능을 제공한다.
          * PagingAndSortingRepository의 부모 인터페이스가 CrudRepository이다.
          * CrudRepository 인터페이스의 부모가 Repository이다.
          */
}
```

6. src/test/java 프로젝트 구조에서 테스트 코드를 작성한다.

BoardRepositoryTests 클래스는 JUnit 테스트용 클래스로 작성한 엔티티 테스트를 실시한다. JPA를 통한 레코드 저장, 검색, 수정,삭제 작업을 해본다.

```
package net.daum;

import org.junit.jupiter.api.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import net.daum.dao.BoardRepository;
import net.daum.vo.BoardVO;

@RunWith(SpringRunner.class)
```

```
@SpringBootTest
public class BoardRepositoryTests {

	@Autowired
	private BoardRepository boardRepo;

	@Test
	public void testInsert() {
		BoardVO b=new BoardVO();

		b.setWriter("홍길동");
		b.setTitle("글제목입니다.");
		b.setContent("글내용입니다.");

		//this.boardRepo.save(b);
	}//저장

	@Test
	public void testRead() {
		//Optional<BoardVO> b=boardRepo.findById(2);
		//System.out.println(b);
	}//읽기

	@Test
	public void testUpdate() {
	  /*  Optional<BoardVO> board = this.boardRepo.findById(2);

	    board.ifPresent(board2 ->{
		System.out.println(board2.getWriter());
		board2.setTitle("수정제목입니다. 수정");
		board2.setContent("수정 내용입니다.");
		System.out.println(board2.getTitle());
		System.out.println(board2.getContent());

		 System.out.println("update title ---------->");
		 this.boardRepo.save(board2);
	    });
	     */
	}//수정
```

```
	@Test
	public void testDelete() {
		System.out.println("엔티티빈 JPA 삭제");
		this.boardRepo.deleteById(1);
	}//삭제
}
```

7. Spring Data JPA에서 메서드의 이름만으로 원하는 질의(Query)를 실행할 수 있는 방법을 제공한다.

이때 쿼리라는 용어는 'select'문에만 해당한다. 이러한 메서드를 쿼리 메서드라고 한다. 이 쿼리 메서드에 의해서 원하는 검색쿼리문이 만들어 진다.

```
package net.daum.dao;

import java.util.Collection;
import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;

import net.daum.vo.BoardVO;

public interface BoardRepository extends CrudRepository<BoardVO, Integer>{

	/* 쿼리 메서드 작성 : 메서드 이름만으로 원하는 질의(쿼리(검색) :query) select 쿼리문을 만들어 내는 메서드를 말한다. 이때 쿼리라는 용어는 'select'문에만
	 * 해당된다. */

	public List<BoardVO> findBoardVOByTitle(String title);//쿼리 메서드중에서
//BoardVO는 엔티티빈 클래스명,Title는 컬럼명=>find+엔티티클래스명+By+컬럼명
	//여기서는 제목 검색

	public Collection<BoardVO> findByWriter(String name);//쿼리 메서드 중에서
//Writer는 빈클래스 속성인 멤버변수명, findBy+속성명(빈클래스변수명)

	//글쓴이에 대한 like % 키워드 % => '%'+검색어+'%'(Containing)
	public Collection<BoardVO> findByWriterContaining(String writer);

	//or 조건 처리 => '%' + 제목 + '%' + Or + '%' + 내용 +'%'
	public Collection<BoardVO> findByTitleContainingOrContentContaining(String
```

반환 타입이 콜렉션 제네릭 타입

findByWriter / findByWriterContaining. .

와일드 카드..% 이름을 정확하게 알지 않아도 해당 키워드를 포함한 모든 걸 검색

와일드 카드 .. _ 해당 자리에 모르는 문자가 하나 있음

```
title,String content);

        //title like %?% And Bno > ?
        public Collection<BoardVO> findByTitleContainingAndBnoGreaterThan(String keyword, int num);

        //bno>? order by bno desc => 게시물 bno가 특정 번호보다 큰 게시물을 bno 기준으로 내림차순 정렬
        public Collection<BoardVO> findByBnoGreaterThanOrderByBnoDesc(int bno);

        @Query("select b from BoardVO b where b.title like %?1% and b.bno> 0 order by b.bno desc") //JPQL(JPA에서 사용하는 Query Language =>
        //Java Persistence Query Language의 약어)이다. JAVA + SQL
        //JPQL은 테이블 대신 엔티빈 클래스를 이용하고,테이블 컬럼대신 엔티티빈 클래스
//의 변수 즉 속성을 이용한다. %?1%에서 ?1은 첫 번째로 전달되는 피라미터 값이라는 의
//미이다.
        public List<BoardVO> findByTitle(String title);

        @Query("select b from BoardVO b where b.content like %:content% and b.bno > 0 order by b.bno desc")
        public List<BoardVO> findByContent(@Param("content") String content);
//:content는 @Param("content")와 연결된다.

        @Query("select b.bno, b.title, b.writer, b.regdate from BoardVO b where b.title like %?1% and b.bno > 0 order by b.bno desc")
        //원하는 컬럼만 추출해서 이용가능 하다. 여러 컬럼을 지정하는 경우 리턴 컬렉션
//제네릭 타입이 엔티티 빈 타입이 아니라 Object[] 타입이라는 것이다.
        public List<Object[]> findByTitle2(String title);
}
```

8. 쿼리 메서드와 JPQL문을 활용한 JUnit 테스트 파일 내용이다.

```
package net.daum;

import java.util.Arrays;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
```

```
import net.daum.dao.BoardRepository;
import net.daum.vo.BoardVO;

@SpringBootTest
class Boot03ApplicationTests {

	@Autowired
	private BoardRepository boardRepo;

	@Test
	void contextLoads() {
	}

	@Test
	public void testInsert20() {

		for(int i=1;i<=20; i++) {
			BoardVO b=new BoardVO();

			b.setWriter("user0"+ (i%10));
			b.setTitle("게시판 제목...:" + i);
			b.setContent("내용..."+i);

			//this.boardRepo.save(b);
		}
	}//저장

	//쿼리 메서드 중에서 제목으로 검색
	@Test
	public void testByTitle() {
		//자바 8
		//this.boardRepo.findBoardVOByTitle("게시판 제목...:17")
		//.forEach(board -> System.out.println(board));

		//자바 8 이전
		 /*List<BoardVO> blist = this.boardRepo.findBoardVOByTitle("게시판
제목...:17");

			if(blist != null && blist.size() > 0) {
				for(int i=0;i<blist.size();i++) {
```

```
                                        System.out.println(blist.get(i));
                                }
                        }else {
                                System.out.println("게시판 목록이 없습니다.");
                        } */
        }//testByTitle()

        //글쓴이로 검색
        @Test
        public void testByWriter() {

        /*Collection<BoardVO> blist = this.boardRepo.findByWriter("user00");

                        blist.forEach(
                                        board->{
                                           System.out.println(board);
                                        }
                                        );
                */
        }//testByWriter()

        //글쓴이에 05가 포함된 게시물 검색 -> '%'+05+'%' like 검색
        @Test
        public void testByWriterContaining() {

                //Collection<BoardVO>                   blist                   =
this.boardRepo.findByWriterContaining("05");
                //blist.forEach(board -> System.out.println(board));

        }//testByWriterContaining()

        //제목 '2'가 포함되거나 내용에 '5'가 포함된 경우
        @Test
        public void testByTitleOrContentContaining() {

                //Collection<BoardVO>                   blist                   =
this.boardRepo.findByTitleContainingOrContentContaining("2", "5");
                //blist.forEach(b -> System.out.println(b));

        }//testByTitleOrContentContaining()
```

```
	//제목에 '5'가 포함되어 있고 게시물 번호가 5보다 큰 자료 검색
	@Test
	public void testByTitleAndBno() {

		//Collection<BoardVO> blist =
boardRepo.findByTitleContainingAndBnoGreaterThan("5",5);
		//blist.forEach(board-> System.out.println(board));
	}//testByTitleAndBno()

	//bno가 10보다 큰 데이터를 내림차순 정렬
	@Test
	public void testBnoOrderBy() {
		//Collection<BoardVO> blist =
this.boardRepo.findByBnoGreaterThanOrderByBnoDesc(10);
		//blist.forEach( b -> System.out.println(b));
	}

	//제목이 들어간 title을 검색
	@Test
	public void testByTitle2() {
		//this.boardRepo.findByTitle("제목")
		//.forEach( b -> System.out.println(b));
	}

	//@Param 내용에 대한 검색 처리
	@Test
	public void testByContent2() {
		//this.boardRepo.findByContent("내용")
		//.forEach(b -> System.out.println(b));
	}

	@Test
	public void testByTitle17() {
		this.boardRepo.findByTitle2("제목")
		.forEach(arr->System.out.println(Arrays.toString(arr)));
	}
}
```

9. JPA 연관관계

연관관계를 처리하기 위해서는 엔티티빈 클래스가 필요함으로 net.daum.vo 패키지에 MemberVO.java 와 Profile.java 엔티티빈 클래스를 설계한다.

```
package net.daum.vo;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter //getter() 메서드 자동 제공
@Setter //setter() 메서드 자동 제공
@ToString //toString() 메서드 자동 제공
@Entity //엔티티빈 클래스
@Table(name="tbl_members6") //@Table 애너테이션을 지정하지 않으면 엔티티빈 클래스명이 테이블명이 된다.name속성에 지정한 tbl_members6이 테이블명이 된다.
@EqualsAndHashCode(of="uid2")
/*
@EqualsAndHashCode
equals()와 hashCode(),canEqual() 메서드를 자동 생성

equals(): 두 객체의 내용이 같은 지 확인
hashcode() : 두 객체가 같은 객체인지 확인

@EqualsAndHashCode(of="uid2"): 연관 관계가 복잡해 질 때, @EqualsAndHashCode에서 서로 다른 연관 관계를 순환 참조하느라 무한 루프가 발생하고,
 결국 stack overflow가 발생할 수 있기 때문에 uid2 값만 주로 사용.
 Stack Overflow는 Stack 영역의 메모리가 지정된 범위를 넘어갈 때 발생한다.
*/
public class MemberVO {

        @Id //각 엔티티 빈을 식별할 수 있도록 해주는 식별키 => 기본키
        private String uid2; //오라클에서는 uid는 컬럼명으로 사용못한다. 내부적으로 사용하고 있음. 오라클에서 UID함수는 세션 사용자를 고유하게 식별하기 위한 정수 ID를 반환
```

예약어

```
        private String upw;
        private String uname;
}
```

```
package net.daum.vo;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.SequenceGenerator;
import jakarta.persistence.Table;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@ToString(exclude="member") //toString()메서드에서 member변수 제외를 하여 호출되지 않게 한다.
@Entity
@SequenceGenerator(//@SequenceGenerator는 시퀀스 생성기를 설정하는 애노테이션
                name="fno_seq_gename", //시퀀스 제너레이터 이름
                sequenceName="fno_seq", //시퀀스 이름
                initialValue=1, //시작값
                allocationSize=1 //메모리를 통해 할당할 범위 사이즈=>기본값은 50이며, 1로 설정하는 경우 매번 insert시마다 DB의 시퀀스를 호출해서 db시퀀스 번호값을 가져와서
                //1증가한 값이 할당된다. 1씩 증가. 증가값
                )
@Table(name="tbl_profile3")
@EqualsAndHashCode(of="fname")
public class Profile {

        @Id  //기본키 컬럼
        //@GeneratedValue(strategy=GenerationType.IDENTITY)
        @GeneratedValue(
                        strategy=GenerationType.SEQUENCE, //사용할 전략을 시퀀스로  선택
```

```
                        generator="fno_seq_gename" //시퀀스 생성기에 설정해놓은 fno_seq_gename 시퀀스 제너레이터 이름 으로 설정
                        )
        private int fno;

        private String fname;

        private boolean current2; //오라클에서 current는 세션에 대한 사용자를 기반으로 해서 세션에 설정된 시간대에 따른 날짜를 얻어다주는 함수를 의미한다. 컬럼명으로 사용못함.

        @ManyToOne //다대일 연관 관계
        private MemberVO member; //외래키로 설정됨
}
```

10. Repository 작성

net.daum.dao 패키지에 두 개의 Repository를 만든다. MemberRepository와 ProfileRepository 인터페이스를 설계한다.

```
package net.daum.dao;

import java.util.List;

import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

import net.daum.vo.MemberVO;

public interface MemberRepository extends CrudRepository<MemberVO, String> {

        @Query(value="select m.uid2,count(fname) from tbl_members6 m left outer join tbl_profile3 p"
                        +" on m.uid2 = p.member_uid2 where m.uid2 = ?1 group by m.uid2"      ,nativeQuery = true)
        //nativeQuery문은 말 그대로 데이터베이스에 종속적인 sql문을 그대로 사용하겠다는 의미이다. 복잡한 쿼리문 작성에 유리하다. 다만, 이 경우 데이터베이스에 독립적이라는
        //장점은 어느 정도 포기해야 한다. @Query에 nativeQuery속성값을 true로 주면 메서드 실행시 value값을 실행한다.=> tbl_members6테이블에는 레코드가 있어나
        //tbl_profile3테이블에는 레코드가 없는 경우 left outer join을 한다. => 단방향
```

```
Fetch Join
        public List<Object[]> getMemberVOWithProfileCount(String uid);//실제 회원아이디와 프로필 사진 개수

        @Query(value="select m.uid2,m.upw, m.uname, p.current2, p.fname from tbl_members6 m left outer join tbl_profile3 p "
                        +" on m.uid2 = p.member_uid2 where m.uid2 = ?1  and p.current2 = 1",nativeQuery = true)
        public List<Object[]> getMemberVOWithProfile(String uid); //회원 아이디 정보와 현재 사용중인 프로필 사진 정보
}
```

```
package net.daum.dao;

import org.springframework.data.repository.CrudRepository;

import net.daum.vo.Profile;

public interface ProfileRepository extends CrudRepository<Profile, Integer> {

}
```

11. JPA를 통해서 연관관계를 활용한 Fetch Join 테스트 코드를 해본다.

```
package net.daum;

import java.util.Arrays;
import java.util.List;
import java.util.stream.IntStream;

import org.junit.jupiter.api.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.annotation.Commit;
import org.springframework.test.context.junit4.SpringRunner;

import lombok.extern.java.Log;
import net.daum.dao.MemberRepository;
```

```
import net.daum.dao.ProfileRepository;
import net.daum.vo.MemberVO;
import net.daum.vo.Profile;

@RunWith(SpringRunner.class)
@SpringBootTest
@Log //Lombok의 로그를 사용할 때 이용하는 애노테이션
@Commit //테스트 결과를 데이터베이스에 commit하는 용도로 사용
public class ProfileTest {

	@Autowired
	private MemberRepository memberRepo;

	@Autowired
	private ProfileRepository profileRepo;

	//코드 중략...

	//user1 아이디정보와 프로필 사진개수 반환 =>Fetch Join
	@Test
	public void testFetchJoin01() {
		//List<Object[]> result =
this.memberRepo.getMemberVOWithProfileCount("user1");

		//result.forEach(arr -> System.out.println(Arrays.toString(arr)));
	}//testFetchJoin01()

	//단방향 Fetch Join2
	@Test
	public void testFetchJoin02() { //user1 회원 아이디 정보와 현재 사용중인 프로
필 사진 정보
		List<Object[]> result =
this.memberRepo.getMemberVOWithProfile("user1");
		result.forEach(arr -> System.out.println(Arrays.toString(arr)));
	}//testFetchJoin02()
}
```