JPQL

프로젝트 : B21eJPQL

준비사항

- 의존설정 : JDBC API, Lombok, Oracle Driver, Spring Web, Spring Data JPA
- JSP 사용을 위한 설정을 한다.
- Refresh Gradle Project 를 눌러 적용한다.

JPQL(Java Persistence Query Language)

- JPA에서 사용하는 객체 지향 쿼리 언어
- JPQL은 SQL과 유사하지만, 테이블과 컬럼을 대상으로 하는 대신 엔티티 클래스와 그 필드를 대상으로 함
- 데이터베이스의 테이블이 아닌 자바 객체를 중심으로 데이터를 조회하고 조작할 수 있음

특징

- 객체 지향 쿼리 언어
  - JPQL은 테이블이 아닌 엔티티 객체와 속성을 대상으로 실행
  - SQL ⇒ SELECT * FROM USER
  - JPQL ⇒ SELECT u FROM User u
- 데이터베이스 독립성
  - JPA가 지원하는 데이터베이스에 독립적인 쿼리를 작성할 수 있음
  - 즉, 데이터베이스에 따라 쿼리문이 달라지지 않음
- 경로 표현식
  - 객체의 필드 접근 방식(점(.) 연산자)을 사용하여 관련 엔티티나 컬렉션에 쉽게 접근
  - Ex) SELECT o.customer.name FROM Order o
- 집계 함수 지원
  - COUNT, SUM, AVG, MAX, MIN 과 같은 집계 함수 지원
- 네임드 쿼리
  - 재사용 가능하도록 엔티티 클래스나 orm.xml 파일에 네임드 쿼리로 정의할 수 있음

프로퍼티 : /B21dJPA_Pageable/src/main/resources/application.properties

이전 예제와 동일함.

spring.jpa.properties.hibernate.hbm2ddl.auto 항목만 적절히 설정

엔티티 : /B21eJPQL/src/main/java/com/edu/springboot/jpa/Member.java

```java
package com.edu.springboot.jpa;

import jakarta.persistence.Entity;

@Getter
@AllArgsConstructor
@NoArgsConstructor(access = AccessLevel.PROTECTED)
@Builder
@Entity(name="JPAMEMBER03")
public class Member {
    @Id
    @SequenceGenerator (
        name = "mySequence03",
        sequenceName = "JPAMEMBER03_SEQ",
        initialValue = 1,
        allocationSize = 1
    )
    @GeneratedValue (generator = "mySequence03")
    private Long id;
    private String name;
    private String email;
}
```

리포지토리 : /B21eJPQL/src/main/java/com/edu/springboot/jpa/MemberRepository.java

```java
package com.edu.springboot.jpa;

import java.util.List;

@Repository
public interface MemberRepository extends JpaRepository<Member, Long> {

    @Query("select m from JPAMEMBER03 m where m.name like "
            + " :name1 order by m.id desc")
    List<Member> findMembers(@Param("name1") String name2);

    //정렬을 위한 Sort 사용.
    @Query("select m from JPAMEMBER03 m where m.name like :name1")
    List<Member> findMembers(@Param("name1") String name2, Sort sort);

    //페이징을 위한 Pageable
    @Query("select m from JPAMEMBER03 m where m.name like :name1")
    Page<Member> findMembers(@Param("name1") String name2,
            Pageable pageable);

    //일반적인 SQL문 사용. 테이블 등 대소문자를 구분하지 않는다.
    @Query(value = "select * from jpamember03 where name like :name1 "
            + " order by id desc",
           nativeQuery = true)
    List<Member> findMembersNative(@Param("name1") String name2);
}
```

서비스 : /B21eJPQL/src/main/java/com/edu/springboot/jpa/MemberService.java

```java
package com.edu.springboot.jpa;

import java.util.List;

@Service
public class MemberService
{
    @Autowired
    private MemberRepository memberRepository;

    public List<Member> selectMembers1(String search) {
        List<Member> member = memberRepository
                .findMembers(search);
        return member;
    }

    public List<Member> selectMembers2(String search,
            Sort sort) {
        List<Member> member = memberRepository
                .findMembers(search, sort);
        return member;
    }

    public Page<Member> selectMembers3(String search,
            Pageable pageable) {
        Page<Member> member = memberRepository
                .findMembers(search, pageable);
        return member;
    }

    public List<Member> selectMembers4(String search) {
        List<Member> member = memberRepository
                .findMembersNative(search);
        return member;
    }
}
```

컨트롤러 : /B21eJPQL/src/main/java/com/edu/springboot/MainController.java

```java
package com.edu.springboot;

import java.util.List;

@Controller
public class MainController {

    @Autowired
    MemberService memberService;

    @GetMapping("/")
    public String main() {
        return "main";
    }

    @GetMapping("/selectNameLike1.do")
    public String selectMembers1(
            @RequestParam("name") String pname, Model model) {
        System.out.println("selectNameLike1 : "+ pname);
        //앞뒤로 %가 있으므로 검색어가 포함된 모든 문자열 검색.
        String searchName = "%"+ pname +"%";
        List<Member> result = memberService
                .selectMembers1(searchName);
        model.addAttribute("members", result);

        return "member_list";
    }

    @GetMapping("/selectNameLike2.do")
    public String selectMembers2(@RequestParam("name") String pname,
                                 Model model) {
        System.out.println("selectNameLike2 : "+ pname);
        //뒤에만 %가 있으므로 검색어로 시작되는 레코드 검색.
        String searchName = pname +"%";
        Sort sort = Sort.by(Sort.Order.asc("id"));
        List<Member> result = memberService.selectMembers2(
                searchName, sort);
        model.addAttribute("members", result);

        return "member_list";
    }

}
```

```java
    @GetMapping("/selectNameLike3.do")
    public String selectMembers3(@RequestParam("name") String pname,
                                 @RequestParam("page") String page,
                                 Model model) {
        System.out.println("selectMembers3(검색어) : "+ pname );
        System.out.println("selectMembers3(페이지) : "+ page );

        String name = pname +"%";
        Sort sort = Sort.by(Sort.Order.desc("id"));
        int pageNum = Integer.parseInt(page) - 1;

        Pageable pageable = PageRequest.ofSize(5).withPage(pageNum)
                .withSort(sort);
        Page<Member> result = memberService
                .selectMembers3(name, pageable);
        List<Member> content = result.getContent();
        long totalElements = result.getTotalElements();
        int totalPages = result.getTotalPages();
        int size = result.getSize();
        int pageNumber = result.getNumber() + 1;
        int numberOfElements = result.getNumberOfElements();

        model.addAttribute("members", content);
        model.addAttribute("totalElements", totalElements);
        model.addAttribute("totalPages", totalPages);
        model.addAttribute("size", size);
        model.addAttribute("pageNumber", pageNumber);
        model.addAttribute("numberOfElements", numberOfElements);

        return "member_list";
    }

    @GetMapping("/selectNameLike4.do")
    public String selectMembers4(@RequestParam("name") String pname,
                                 Model model) {
        System.out.println("selectMembers4(검색어) : "+ pname );
        String name = "%"+ pname +"%";
        List<Member> result = memberService.selectMembers4(name);
        model.addAttribute("members", result);

        return "member_list";
    }
}
```

뷰 : /B21eJPQL/src/main/webapp/WEB-INF/views/main.jsp

```
<body>
    <h2>JPQL</h2>
    <ul>
        <li><a href=/selectNameLike1.do?name=test>
                Name Like 조회1</a></li>
        <li><a href=/selectNameLike2.do?name=test>
                Name Like 조회2</a></li>
        <li><a href=/selectNameLike3.do?name=test&page=2>
                Name Like 조회3(페이징 적용)</a></li>
        <li><a href=/selectNameLike4.do?name=test>
                Name Like 조회4 : Native SQL</a></li>
    </ul>
</body>
</html>
```

뷰 : /B21eJPQL/src/main/webapp/WEB-INF/views/member_list.jsp

```
<body>
    <h2>Spring boot 프로젝트</h2>
    <ul>
        <li><a href="/">루트</a></li>
    </ul>

    <h2>JPQL - @Query</h2>

    <c:if test="${not empty totalElements}">
        <h3>Pageable 적용 결과</h3>
        총 레코드 갯수 : ${totalElements} <br>
        전체 페이지 수 : ${totalPages} <br>
        페이지 당 레코드 수 : ${size} <br>
        페이지번호 : ${pageNumber} <br>
        엘리먼트(컨텐츠) 갯수 : ${numberOfElements} <br>
        <hr />
    </c:if>

    <h3>레코드 출력</h3>
    <c:forEach var="member" items="${members}">
        <p>
            아이디 : ${member.id},
            이름 : ${member.name},
            이메일 : ${member.email}
        </p>
    </c:forEach>
</body>
</html>
```