Presentation Index

1 TECH SKILL

사용 언어 및 기타 기술

01

Java #1

자바 언어의 기본적인 문법을 바탕으로
객체 지향 프로그래밍을 학습하였습니다.

02

Spring #2

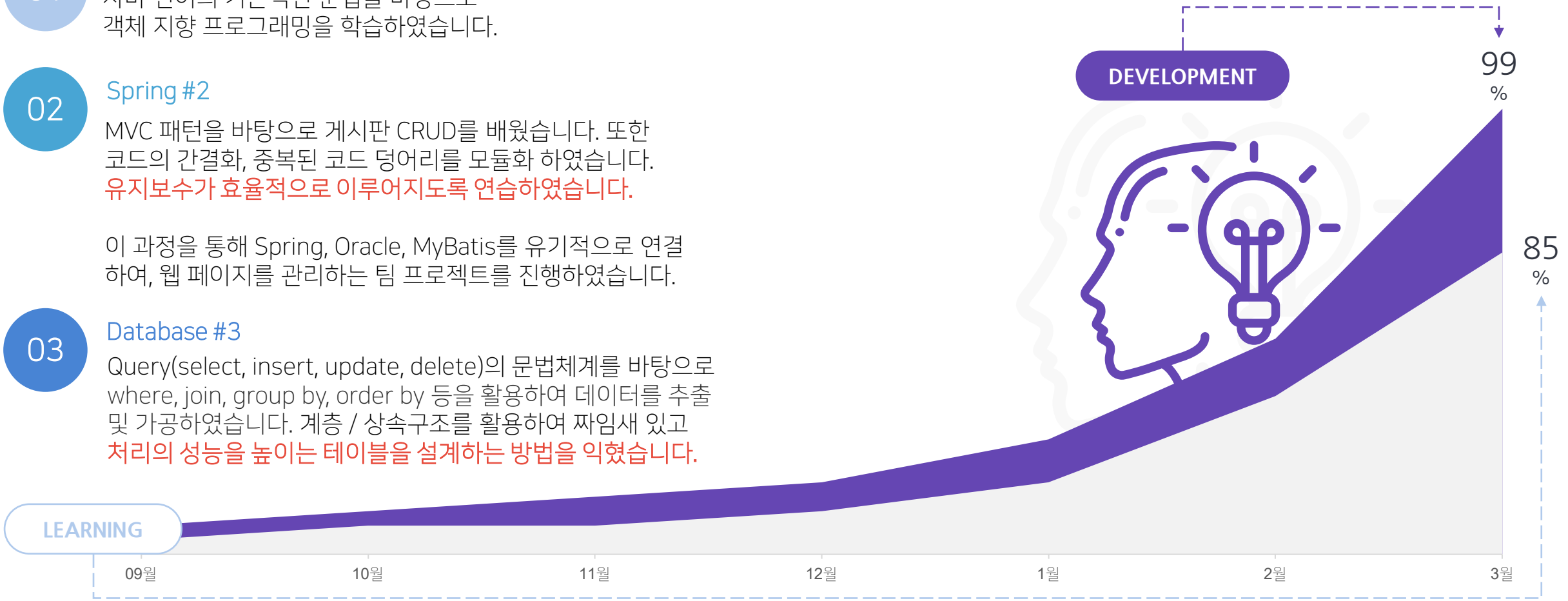
MVC 패턴을 바탕으로 게시판 CRUD를 배웠습니다. 또한
코드의 간결화, 중복된 코드 덩어리를 모듈화 하였습니다.
유지보수가 효율적으로 이루어지도록 연습하였습니다.

이 과정을 통해 Spring, Oracle, MyBatis를 유기적으로 연결
하여, 웹 페이지를 관리하는 팀 프로젝트를 진행하였습니다.

03

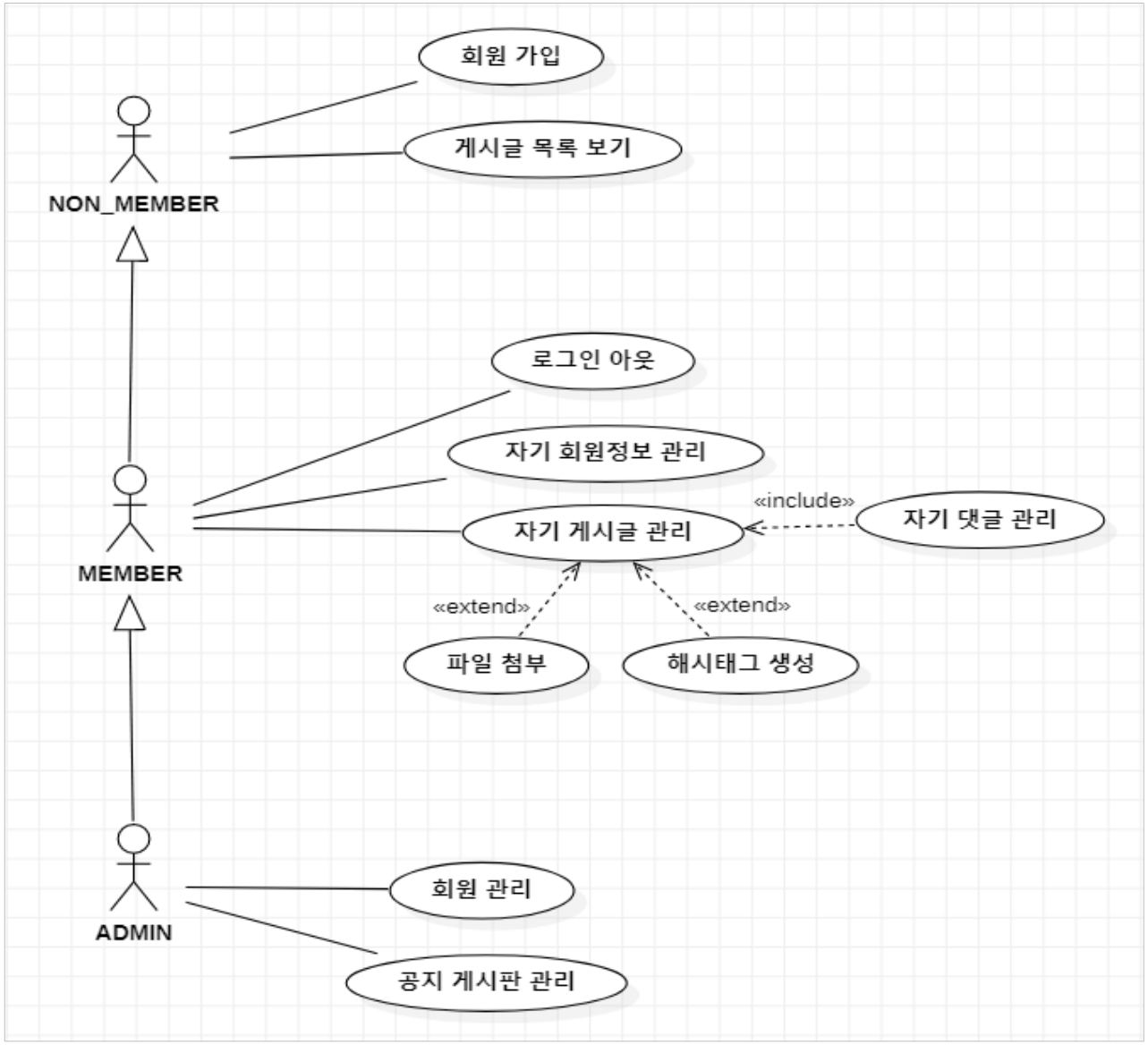
Database #3

Query(select, insert, update, delete)의 문법체계를 바탕으로
where, join, group by, order by 등을 활용하여 데이터를 추출
및 가공하였습니다. 계층 / 상속구조를 활용하여 짜임새 있고
처리의 성능을 높이는 테이블을 설계하는 방법을 익혔습니다.



2 PROJECT

▪ 설 계 [2-1] UML 'UseCase Diagram'



Actor 사이의 일반화 관계

Admin 액터는 다른 액터(Member, Non_Member)의
구체화된 use case를 포함하여 동작되도록 설계

<<include>> 포함 관계

게시글이 반드시 있어야 댓글을 관리 할 수 있게 함

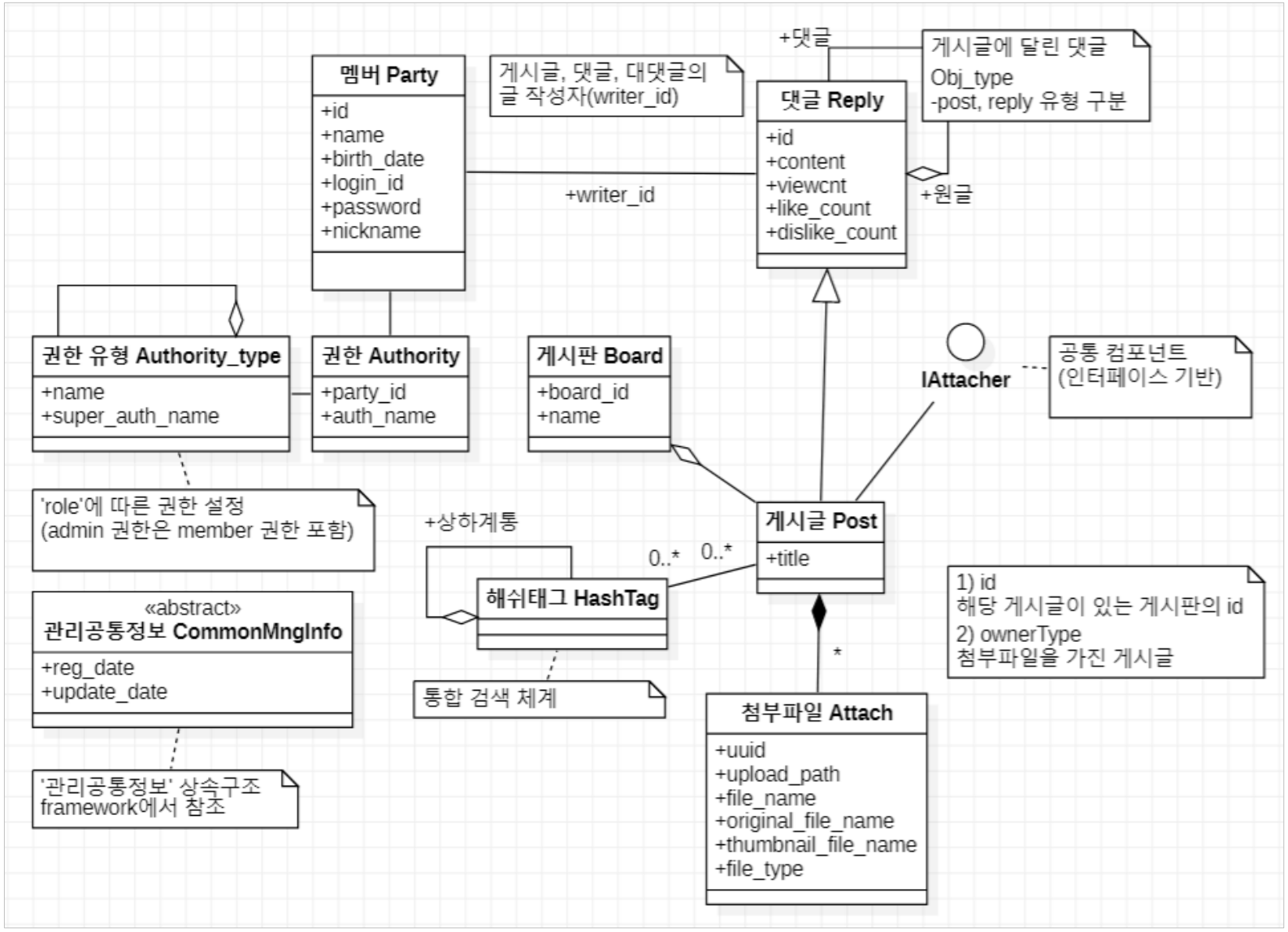
<<extends>> 확장 관계

사용자가 게시글에 파일첨부 혹은 해시태그 생성을
하는 것은 선택적으로 수행 할 수 있음을 표현

사용자의 행위를 동사로 정의, 가독성을 높임

2 PROJECT

▪ 설 계 [2-2] UML 'Class Diagram'



처리 성능 향상, 효율적인 데이터관리 #1

해시태그의 키워드를 인덱스로 활용해 **통합 검색 체계를 설계**하였습니다. 데이터의 중복이 낮은 컬럼을 저장시켜 **정렬 속도를 향상하였고, 시스템의 부하를 줄였습니다.**

안정적인 유지 보수 #2

해시태그는 각종 대상 테이블을 연결하면 사용 가능 합니다.
중복으로 사용하는 코드 및 함수는 모듈화 및 상속 시킵니다.

유사한 객체는 구성으로 설계 #3

Composite Pattern을 바탕으로 원글(게시글)과 댓글은 1:N 관계로 나타내며, Obj_Type(유형)으로 구분하였습니다.

공통 컴포넌트 구현 (Interface 기반) #4

첨부파일의 데이터 값이 중복되거나 다를 때 **기준을 표준화하여 정보를 선별**합니다. 다양한 파일을 통합적으로 관리합니다.

▪ 설 계 [2-3] 개발 기능

<table>
<tr><th></th><th>Create</th><th>Read</th><th>Update</th><th>Delete</th><th>Paging</th><th>Etc</th></tr>
<tr><td>Member</td><td>가입</td><td>내 정보 보기</td><td>내 정보 변경
+비밀번호 변경</td><td>탈퇴</td><td></td><td>임시 비밀번호 발급을
위한 email 전송</td></tr>
<tr><td rowspan="2">Admin</td><td colspan="4">Member와 동일한 기능 가능</td><td rowspan="3">Table을
목록 형식
으로 이용
화면 처리</td><td>공지사항 게시판 관리
(CRUD)</td></tr>
<tr><td></td><td>모든 회원 정보 열람</td><td></td><td>Member
강제탈퇴</td><td></td></tr>
<tr><td>Post
Reply
Re-Reply</td><td>작성</td><td>보기</td><td>수정</td><td>삭제</td><td>Post에 대한
조회수 증가,
좋아요, 싫어요 기능</td></tr>
<tr><td>Hash Tag</td><td>생성</td><td>추출된 해시태그 Post에서 보기</td><td rowspan="2">이미 생성된 데이터를
전체 삭제 후
다시 생성하는 방식</td><td rowspan="2">삭제</td><td rowspan="2"></td><td>선택적으로 추가 가능
생성된 해시태그로
Post 검색 가능</td></tr>
<tr><td>Attach</td><td>첨부
이미지,
비디오
썸네일
생성</td><td>첨부된 파일 목록 Post에서 보기

썸네일 클릭 시,
이미지(원본 이미지 보여줌)
비디오(원본 동영상 자동재생)

오디오 클릭 시 자동재생</td><td>선택적으로 추가 가능

오디오, 기타 형식
(다운로드 가능)

Exe, zip, alz, sh
파일형식 첨부 불가</td></tr>
</table>

3 PROJECT

▪ 구 현 [3-1] 처리 성능의 향상

HashTagMapper.xml

```
<select id="selectNewID" resultType="long">
    select max(newid)
      from (
           select 1 as newid from dual
           union
           select newid
             from (
                  select (id + 1) as newid
                    from T_Hash_Tag
                   where rownum = 1
                   order by id desc
                   )
           )
</select>
<insert id="createHashTag">
   <foreach item="obj" collection="list"
            open="INSERT ALL "
            close=" SELECT * FROM DUAL">
        INTO T_Hash_Tag(id, word)
        VALUES (#{obj.id}, #{obj.word})
   </foreach>
</insert>
```

단점

반복 호출

Seq.nextval

해시태그를 10개 생성하는 데 id 값에 sequence를 활용해 nextval을 사용하면 insert 쿼리문이 10번 실행되어 반복 호출 하므로 이는 성능을 저하시킵니다.

대안

호출 최소화

Foreach, Insert

selectNewId는 새로 생성 시킬 id를 1개의 배열에서 10개의 값을 목록으로 만들어 일괄 Insert하였습니다. 이는 공통적으로 필요한 것은 Foreach로 반복 처리하여 DB를 2번만 호출하게 됩니다.

대량의 Insert 처리시, DB호출을 줄이는데 효과적

3 PROJECT

▪ 구 현 [3-1] 처리 성능의 향상

ReplyMapper.xml

```
<select id="findReplyById" resultMap="replyResultMap">
<![CDATA[
    select p.board_id, p.id, p.content, p.writer_id, p.title, p.like_count,
           p.dislike_count, p.obj_type,
           p.reg_date, p.update_date,
           c.count_of_reply,
           w.id w_id, w.nickname w_nickname,
           ht.id ht_id, ht.word ht_word, ht.descript ht_descript
      from (
            select a.id, count(b.id) count_of_reply
              from T_Reply a
                   left outer join T_Reply b on a.id = b.original_id
             where a.id = #{id}
             group by a.id
           ) c, T_Reply p
           left outer join T_Party w on p.writer_id = w.id
           left outer join M_HT_Reply mht on p.id = mht.reply_id
           left outer join T_Hash_Tag ht on mht.hash_tag_id = ht.id
     where c.id = p.id
]]>
</select>
```

적절한 where조건절 사용

테이블의 모든 컬럼에 접근하는 방법 대신 조건을 주어 필요한 최종 레코드만 빠르게 반환되도록 한정합니다.

Left outer join 중복제거

여러 테이블 간의 관계성에 따라 공통적으로 가진 필드값(id)을 접점으로 결합합니다. 미리 조건을 준 테이블과 JOIN을 하여 성능을 최적화 하였습니다.

구조화 된 Subquery

하나의 쿼리 안 목표로 하는 데이터를 추출합니다.
서브 쿼리는 독립적으로 실행된 후 메인 쿼리는 그 결과를 이용하여 실행되도록 하였습니다.

모든 데이터에 Full Scan해서 접근하는 것보다 시간을 단축

3 PROJECT

▪ 구 현 [3-2] 안정적인 유지 보수

changePW.jsp

```
        $("#pw1").on("focusout", function isSame() {
            var pw1 = $("#pw1").val();
            // 정규 표현식: 숫자, 특문 각 1회 이상, 영문은 2개 이상 사용, 글자수 8~15제한
            var pwPattern = /(?=.*\d{1,50})(?=.*[~`!@#$%\^&*()-+=]{1,50})(?=.*[a-zA-Z]{2,50}).{8,15}$/;

            if (isEmpty(pw1))
                return;
            else if (pw1.length < 8 || pw1.length > 15) {
                window.alert('The password can only be used with 8~15 characters.');
                document.getElementById('pw1').value='';
                $("#pw1").focus();
            } else if(pwPattern.test(document.getElementById('pw1').value) != true){
                alert("Password doesn't fit the format");
                document.getElementById('pw1').value='';
                $("#pw1").focus();
            }
        });

        var frmChangePW = $("#frmChangePW");
        $("#btnChangePW").on("click", function(e) {
            e.preventDefault();
            frmChangePW.submit();
        });
    })

// 넘어온 값이 빈값인지 체크합니다.
// !value 하면 생기는 논리오류제거는 명시적으로 value == 사용합니다.
// [], {} 도 빈값으로 처리합니다.
var isEmpty = function(value){
  if( value == "" || value == null || value == undefined ||
          ( value != null && typeof value == "object" && !Object.keys(value).length ) ){
    return true
  }else{
    return false
  }
};
```

HashTagVO.java

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    HashTagVO other = (HashTagVO) obj;
    if (word == null) {
        if (other.word != null)
            return false;
    } else if (!word.equals(other.word))
        return false;
    return true;
}
```

꼼꼼한 문자열 객체 관리 #1

정규식 사용으로 형식을 패턴화하여 재사용성을 높였습니다.
String은 비교할 문자열을 먼저 넣고, Equals로 제어합니다.

신중한 예외 처리 #2

값의 모호함을 감소시키고, 예외는 사전에 방지합니다.
NullPointException 예방을 위한 Null 여부를 비교합니다.

간결한 코드 작성과 오류 감소로 품질 향상에 노력

3 PROJECT

▪ 구 현 [3-2] 안정적인 유지 보수

postCommon.jsp

```
<script type="text/javascript">
    //create read update
    function setOprationMode(oprationMode) {
        if ("read" === oprationMode) {
            $("#title").attr("readonly", true);
            $("#txacontent").attr("readonly", true);
            $("#hashTag").attr("readonly", true);
            //btn: extractHashTag
            $("#btnExtractHashTag").hide();
            $("#clickTxt").hide();
            //btn: file select, file upload
            $("#uploadFileLbl").hide();
            $("#uploadFile").hide();
            $("#btnUpload").hide();
            $("#clickAttachTxt").hide();
        } else if ("create" === oprationMode) {
            $("#divIdentifier").hide();
            $("#btnLike").hide();
            $("#btnDislike").hide();
        } else if ("update" === oprationMode) {
            $("#btnLike").hide();
            $("#btnDislike").hide();
            }
    }
</script>
```

1. postDetail.jsp = read
2. registerPost.jsp = create
3. modifyPost.jsp = update

Quick Search

Case SENSITIVE Pattern (? = any character, * = any string)

setOprationMode

Matching items:

L...	Text	Path
39	setOprationMode("create");	registerPost.jsp - PG0222/
49	setOprationMode("remove");	removeParty.jsp - PG0222/
54	setOprationMode("read");	partyDetail.jsp - PG0222/sr
53	setOprationMode("update");	modifyParty.jsp - PG0222/
47	setOprationMode("change");	changePW.jsp - PG0222/sr
62	function setOprationMode(oprationMode) {	partyCommon.jsp - PG022
04	setOprationMode("read");	postDetail.jsp - PG0222/sr
42	setOprationMode("update");	modifyPost.jsp - PG0222/s
59	function setOprationMode(oprationMode) {	postCommon.jsp - PG022
14	function setOprationMode(oprationMode) {	attachUploadCommon.jsp

각자의 역할에 따른 활성화 혹은 비활성화

화면에서 나타낼 값을 id선택자로 호출, readonly, show, hide로 제어합니다.
각 JSP는 해당 역할(CRUD)을 구분하여, 수정이 필요하면 쉽게 변경 가능합니다.

CommonMngInfoVO.java

```
package www.dream.com.framework.model;

import java.util.Date;

/**
 * 모든 테이블에 있는 관리 정보를 각 VO에 중복 개발하지 않고,
 * framework패키지 내부에 상속 구조를 활용하여 공통화합니다.
 */
public abstract class CommonMngInfoVO {
    @Getter @Setter
    @Caption(whenUse=WhenUse.all, caption="RegistrationDate")
    protected Date regDate;
    @Getter @Setter
    protected Date updateDate;
}
```

PartyVO.java

```
package www.dream.com.party.model;

import java.io.Serializable;

@Data
@NoArgsConstructor
public class PartyVO extends CommonMngInfoVO implements Serializable
```

계층적 상속(Hierarchical Inheritance)구조로 설계

여러 테이블에서 반복적으로 사용될 수 있는 정보는 분할하였습니다.
이는 하나의 클래스로 간결하게 작성하여 상속구조로 사용합니다.

공통으로 사용하는 코드들을 통합적으로 관리하여 개발시간을 단축

3 PROJECT

▪ 구 현 [3-2] 안정적인 유지 보수

```java
package www.dream.com.framework.model;
@Getter
public class Criteria {
    /** 표시할 '페이지의 전체 개수' */
    private static final long PAGING_AMOUNT = 10;
    /** 한 페이지에서 보여줄 '게시글의 개수' */
    public static final int DEFAULT_AMOUNT = 10;

    public Criteria(long pageNum, long totalDataCount) {
        this.totalDataCount = totalDataCount;
        setPageNum(pageNum);
    }

    public long getOffset() {
        return (pageNum - 1) * DEFAULT_AMOUNT;
    }
    public long getLimit() {
        return pageNum * DEFAULT_AMOUNT;
    }
    public void setPageNum(long pageNum) {
        this.pageNum = pageNum;
        //아래는 구글 스타일을 활용합니다.
        endPage = pageNum + PAGING_AMOUNT / 2;
        endPage = endPage < PAGING_AMOUNT ? PAGING_AMOUNT : endPage;
        startPage = endPage - PAGING_AMOUNT + 1;
        int realEnd = (int) Math.ceil((float) totalDataCount / DEFAULT_AMOUNT);
        if (realEnd < endPage)
            endPage = realEnd;
        hasPrev = startPage > 1;
        hasNext = endPage < realEnd;
    }
    public void setTotalDataCount(long totalDataCount) {
        this.totalDataCount = totalDataCount;
        int realEnd = (int) Math.ceil((float) totalDataCount / DEFAULT_AMOUNT);
        if (realEnd < endPage)
            endPage = realEnd;
        hasNext = endPage < realEnd;
    }
}
```

Criteria.java --▸ Reply, Post, Party 3개의 객체 --▸ 9개 이상의 함수 사용

1. ' 개수 산출과 페이지 ' 를 함께 처리 (3개의 함수)
2. ' 목록' 처리 (3개의 함수), 각 목록에 대한 페이지 처리 (3개의 함수)

기준

1. 변하지 않는 값의 상수화 (PAGING_AMOUNT, DEFAULT_AMOUT)
2. 규칙 정의 및 수식화 (offset, limit, pageNum, endPage, startPage 등)

여러 곳에서 용도에 따라 함수를 사용할 수 있도록 일정한 체계를 사용

3 PROJECT

▪ 구 현 [3-3] 효율적인 데이터 관리

HashTagService

```
@Service
/*
 * 컨텍스트에서 빈을 만드는 기본은 Singleton(단 한개 - Service, Control Bean은 처리 객체다.)
 * 다중 처리 등으로 여러 객체을 만들 필요가 있다면 @Scope("prototype")을 적용해 놓고
 */
@Scope("prototype")
public class HashTagService {
    private static Komoran komoran = new Komoran(DEFAULT_MODEL.FULL);

    @Autowired
    private HashTagMapper hashTagMapper;

    // 데이터의 추가나 삭제시 트리가 한쪽으로 치우쳐지지 않도록 균형을 맞췄습니다.
    // 여기서 사용하는 이유는 해시태그 생성시 한쪽(품사)으로 쏠리지 않게 하기 위해 사용합니다.
    private static Set<String> setTargetLexiconType = new TreeSet<>();
    static {
        //코모란 형태소 품사정보 분류에 따라 필요한 것만 골라 정의
        setTargetLexiconType.add("SL");  // 외국어
        setTargetLexiconType.add("NA");  // 형용사
        setTargetLexiconType.add("NNG"); // 일반명사
        setTargetLexiconType.add("NNP"); // 고유명사
    }

    public Set<String> extractHashTag(String... varText) {
        Set<String> ret = new TreeSet<>();

        for (String text : varText) {
            KomoranResult komoranResult = komoran.analyze(text);
            List<Pair<String, String>> sentence = komoranResult.getList();
            for (Pair<String, String> token : sentence) {
                if (setTargetLexiconType.contains(token.getSecond()))
                    ret.add(token.getFirst());
            }
        }
        return ret;
    }
```

인덱스를 활용한 정렬 #1

해시태그의 키워드를 단어집(Lexicon)으로 만들고 이를 인덱스로 정렬합니다

높은 검색 정확도 #2

상위 계통에서 하위 계통에 있는 것까지 모두 검색 가능하도록 합니다.

```
SELECT *
  FROM T_Hash_Tag
 START WITH word = 'IT'
CONNECT BY PRIOR  id = super_id;
```

	ID	SUPER_ID	WORD
1	39	(null)	IT
2	40	39	ProgramingLanguage
3	41	40	java
4	42	41	thread
5	43	41	lambda

균형 잡힌 단어 체계 #3

이 단어집은 데이터의 균형을 이루는 TreeSet 구조로 설계 되었으며,

단어집의 체계는 KOMORAN 한국어 형태소에서 분석한 품사를 활용합니다.

해시태그를 이용한 검색 성능 향상

▪ 구 현 [3-3] 효율적인 데이터 관리

```
protected static List<HashTagVO> identifyOldAndNew(PostVO post, HashTagService hashTagService) {
    //교집합 되는 것들, 새것들
    DreamPair<List<HashTagVO>, List<HashTagVO>> pair = hashTagService.split(arrHashTag);
    /* 신규 단어 등록 */
    hashTagService.createHashTag(pair.getSecond());
    //전체 단어와 게시글 사이의 연결 고리를 만들어 줍니다.
    pair.getFirst().addAll(pair.getSecond());
    return pair.getFirst();
}
```

```
/**
 * @return first : 있는 것. 즉 교집합, second : 새로운 단어
 */
public DreamPair<List<HashTagVO>, List<HashTagVO>> split(String[] arrHashTag) {
    //기존 테이블에서 관리 중인 단어 객체를 찾음
    List<HashTagVO> listExisting = hashTagMapper.findExisting(arrHashTag);
    //HashTagVO.equals 기능을 word를 바탕으로 만들어서 일괄 삭제함.
    //따라서 새로이 나타난 단어들을 찾음
    listFullTag.removeAll(listExisting);

    return new DreamPair<>(listExisting, listFullTag);
}
```

```
public ResponseEntity<DreamPair<Criteria, List<ReplyVO>>> listReplyWithPaging
    DreamPair<Criteria, List<ReplyVO>> dreamPair = new DreamPair<>(criteria, listReply);
public ResponseEntity<DreamPair<Long, Integer>> registerReply
    long cnt = replyService.registerReply(reply);
    return new ResponseEntity<>(new DreamPair(cnt, Criteria.DEFAULT_AMOUNT), HttpStatus.OK);
```

```
package www.dream.com.framework.dataType;

import java.io.Serializable;

public class DreamPair<F, S> implements Serializable {
    private F first;
    private S second;
    public DreamPair(F first, S second) {
        this.first = first;
        this.second = second;
    }
    public F getFirst() {
        return first;
    }
    public S getSecond() {
        return second;
    }
}
```

DreamPair.java

교집합을 제외, 새로 생성되는 것을 추가 #1

다른 자료형의 두 값을 비교 및 일괄 처리 #2

데이터의 중복을 최소화하고, 빠른 처리 속도로 효율적

▪ 구 현 [3-3] 효율적인 데이터 관리

```java
public class ClassUtil {
    public static List<Field> getListField(Class<?> targetClass, Class<Caption> annotation) {
        List<Field> listField = new ArrayList<>();
        //부수효과(Side Effect) 활용. 인자에 내용이 호출로 바뀝니다.
        getAllField(targetClass, listField);
        //ConcurrentModificationException 발생을 거부합니다.
        Iterator<Field> iter = listField.iterator();
        while (iter.hasNext()) {
            Field field = iter.next();
            Caption annoCaption = field.getAnnotation(annotation);
            if (annoCaption == null)
                iter.remove();
        }
        return listField;
    }
```

www.dream.com.framework.display 패키지 [ClassUtil.java / TableDisplayer.java 클래스]

```java
public class TableDisplayer {
    public static String displayHeader(Class<?> targetClass) {
        StringBuilder sb = new StringBuilder("<tr>");
        List<Field> listField = ClassUtil.getListField(targetClass, Caption.class);
        for (Field field : listField) {
            Caption annoCaption = field.getAnnotation(Caption.class);
            if (WhenUse.isTableTarget(annoCaption.whenUse())) {
                sb.append("<th>").append(annoCaption.caption()).append("</th>");
            }
        }
        sb.append("</tr>");
        return sb.toString();
    }
}
```

01 값의 변경 시 오류 방지 #1

리스트를 순회하면서 특정 값을 수정 할 때, 반복문 안에서 크기, 인덱스가 변해 ConcurrentModificationException 예외가 생기는 오류를 Iterator를 사용하여 방지하였습니다.

02 메모리 공간을 절약 #2

StringBuilder를 사용, 가변길이로 생성될 수 있는 테이블 가로줄(tr)을 새로운 문자열 객체로 작성하지 않고, 기존의 데이터에 더하는 방식을 사용하여 시스템 부하를 줄였습니다.

한 타입의 변수를 사용함으로써 여러 타입의 객체를 참조하여 다형성 구현