# 객체지향 개념 기반의 SELECT 쿼리의 정의 및 구현

신형기[O] 김장원[1] 백두권[1]
[O]고려대학교 소프트웨어공학과
[1]고려대학교 컴퓨터전파통신공학과
[O]shk101@msn.com, [1]jwkim@software.korea.ac.kr, [1]baikdk@korea.ac.kr

# Implementation and Definition of select query based on object-oriented concept

Hyung-Ki Shin[O] Jangwon Kim[1] Doo-Kwon Baik[1]
[O]Dept, of Software Engineering, Korea University
[1]Dept. of Computer Science & Radio Communications Engineering, Korea University

**요 약**

프로그래밍 코드 안에서 작성하는 기존의 문자열 조합 방식으로 SELECT 쿼리를 작성할 때 일정한 규칙이 없이 다양한 형태로 쿼리가 작성되고 있으며, 이로 인해 쿼리 작성과 관리에 불편함을 주고 있다. 이런 불편함을 해소하고 일관된 규칙을 적용할 수 있는 쿼리 작성 방법을 객체지향 개념을 이용해서 SELECT 쿼리 객체를 정의하고 사용하는 방식으로 제시한다. 이를 위하여 기존 Dynamic Query Class[1]에 SELECT 쿼리 객체를 정의하여 추가하여 전체 상속 관계를 완성하고, SELECT 쿼리에서 둘 이상의 테이블 조인 관계를 객체를 사용해서 구현할 수 있음을 제시한다.

## 1. 서 론

어플리케이션에서 SQL 쿼리를 사용하는 방식은 크게 클라이언트에서 쿼리를 작성하는 방법과 서버에서 프로시저 등으로 작성하는 방법이 있다[2,3,4]. 클라이언트에서 작성하는 방법에는 쿼리들을 외부 파일(XML 파일 등)에 저장하여 프로그래밍 코드에서 로드하여 사용하는 방법과, 프로그래밍 코드에서 직접 SQL 쿼리들을 문자열로 조합하여 작성하는 방법으로 나눌 수 있다.

본 논문에서는 프로그래밍 코드에서 직접 SELECT 쿼리를 문자열로 조합해서 사용할 때 발생하는 문제점과 불편사항들을 해소하고자 쿼리를 캡슐화해서 객체로 정의해서 사용하는 방법을 제안한다. 예를 들어서 "SELECT ID, Name, Age, Zip FROM Members WHERE Name = '신형기'" 쿼리를 작성하고자 한다면 다음 표 1과 같이 문자열을 조합하여 사용한다.

표 1. 쿼리의 문자열 조합 예

```
string str = null;
str = "SELECT ";
str += "ID,";
str += "Name,";
str += "Age, Zip ";
str += "WHERE Name = '신형기'";
```

이러한 작성 방법은 작성자에 따라 형태가 매우 다양하게 생성되고 있으며 일정한 규칙이나 체계가 없고, 필드 타입에 따른 문자 처리 조합에도 신경을 써야 한다. 결국은 코드의 작성과 관리에 불편함을 주고 일관성이 없는 문자열 조합의 쿼리를 생성시킨다. 따라서 프로그래밍 코드에서 SELECT 쿼리를 작성할 때 일관성 있는 규칙을 적용하기 위해서, 객체지향 개념[5,6]을 이용해서 SELECT 쿼리 객체를 정의하고 사용하는 방식으로 제시한다. 이를 위해 기존 Dynamic Query Class의 상속 관계에 SELECT 쿼리 객체를 정의하여 추가하여 전체 상속 관계를 완성한다[7]. 그리고 부모 객체를 상속받은 SELECT 객체가 어떻게 단일 테이블의 SELCET 쿼리와 2개 이상의 테이블 조인의 SELECT 쿼리를 구현하는지를 보임으로써 기존의 문자열 조합 방식보다 체계적으로 작성할 수 있음을 제시한다.

## 2. 관련 연구

### 2.1 iBatis와 Hibernate 그리고 Dynamin Query Class

클라이언트에서 쿼리를 생성하는 방법은 쿼리들을 외부 파일(XML 파일 등)에 저장하여 프로그래밍

코드에서 로드하여 사용하는 방법과 프로그래밍 코드에서 직접 SQL 쿼리들을 문자열로 조합하여 작성하는 방법으로 나눌 수 있다. iBatis[8,10]와 Hibernate[9,10]는 외부 XML 파일에 쿼리나 매핑된 컬럼 정보들을 저장하여 코드에서 로드하여 사용하는 방법이고, Dynamic Query Class는 정의된 객체를 사용해서 프로그래밍 코드에서 쿼리들을 실시간으로 생성해서 사용하는 방법이다. 다음 표 2는 간단한 비교이다.

표 2. 각 프레임웍 비교

| 프레임웍 | 비 교 |
|---|---|
| iBatis | 사용할 쿼리들을 파라미터화하여 XML 파일에 정의하고 그에 대한 클래스를 정의하여 코드에서 로드함 |
| Hibernate | 테이블의 row에 대한 정보를 XML 파일에 정의하고 그에 대한 클래스를 정의하여 코드에서 로드함 |
| Dynamic Query Class | 미리 정의된 객체를 사용하여 코드에서 실시간으로 쿼리(Insert, Update, Delete)를 자동 생성함 |

iBatis와 Hibernate는 사용할 모든 쿼리에 대해서 XML파일과 클래스에 대해서 일일이 정의를 해줘야 하지만, Dynamic Query Class는 dqInsert(insert), dqUpdate(update), dqDelete(Delete) 3개의 객체로 모든 Transaction 쿼리를 코드에서 실시간으로 생성하여 실행한다.

본 논문에서는 Dynamic Query Class에서 정의되어 있지 않는 SELECT 쿼리 구문을 객체화하여, Dynamic Query Class의 상속 관계에 추가하여 전체 상속관계를 완성하고 SELECT 쿼리의 구현을 보인다. 다음은 먼저 기존 Dynamic Query Class에 대한 간단한 설명이다.

## 2. 2 Dynamic Query Class 보기

Transaction 쿼리인 INSERT, UPDATE, DELETE 구문을 그림 1처럼 각각 캡슐화하여 하나의 객체(클래스)로써 정의되었다.

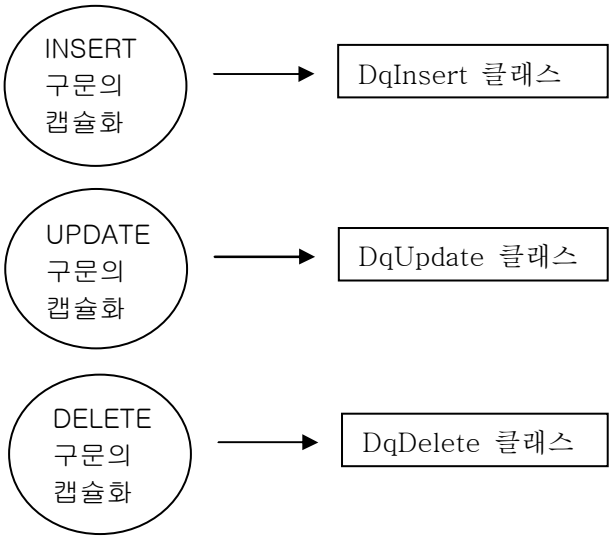


그림 1. Transaction 구문의 객체화

### 2.2.1 Dynamic Query Class 의 상속 관계

UPDATE 쿼리와 DELETE 쿼리는 공통적으로 WHERE 구문이 필요하기 때문에 먼저 WHERE 구문을 처리하기 위한 객체가 그림 2처럼 정의되어 있다.

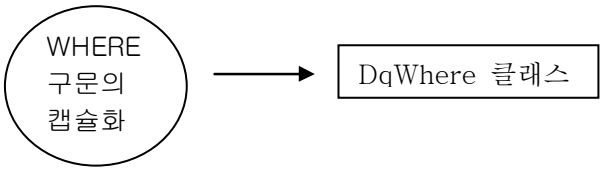


그림 2. WHERE 구문의 객체화

DqWhere 클래스는 Where 구문의 처리를 위한 함수를 가지고 있는데, 표 3은 주요 함수와 구현되는 예를 나타낸다.

표 3. DqWhere 객체의 주요함수와 Where구의 사용 예

| 주요 함수 | Wherer구의 사용 예 |
|---|---|
| AddWhrStr<br>필드타입이 문자열 | Where Name = '신형기' →<br>AddWhrStr(null,"Name","=","신형기") |
| AddWhrInt<br>필드타입이 숫자형 | Where Age >= 20 →<br>AddWhrInt(null,"Age",">=","20") |
| AddWhrInStr<br>필드타입이 문자열 | Where A in ('a1','a2','a3') →<br>AddWhrInStr(null,"A","a1","a2","a3") |
| AddWhrInInt<br>필드타입이 숫자형 | Where B in (10,20,30,45) →<br>AddWhrInInt(null,"B",10,20,30,45) |
| AddWhrBetweenInt<br>필드타입이 숫자형 | Where C BETWEEN 10 AND 30 →<br>AddWhrBetweenInt(null,"C","10","30") |

이 DqWhere 객체를 UPDATE 객체(DqUpdate)와 DELETE 객체(DqDelete)가 상속받음으로써 쿼리를 생성할 때 DqWhere 객체를 이용하여 WHERE 구를 만들어 낸다. 상속관계는 그림 3과 같다.

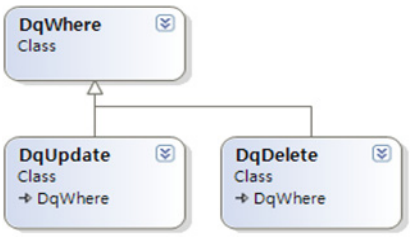


그림 3. DqWhere 객체를 상속받는 두 쿼리 객체

모든 Transaction 처리 쿼리가 만들어지면 DB 서버에 접속해서 최종 쿼리를 실행하는데, 이 기능을 하는 객체도 그림 4와 같이 따로 정의되어 있다.

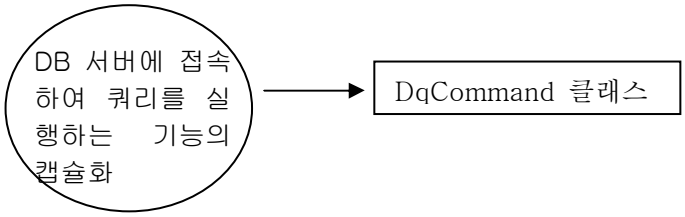


그림 4. DB서버 접속 객체의 정의

이로써 모든 쿼리 객체가 DqCommand 객체를 상속받게 함으로써 DB 서버에 접속해서 자기의 쿼리를 실행하게 된다. 그림 5는 전체 Transaction 쿼리 객체의 상속 관계이다.

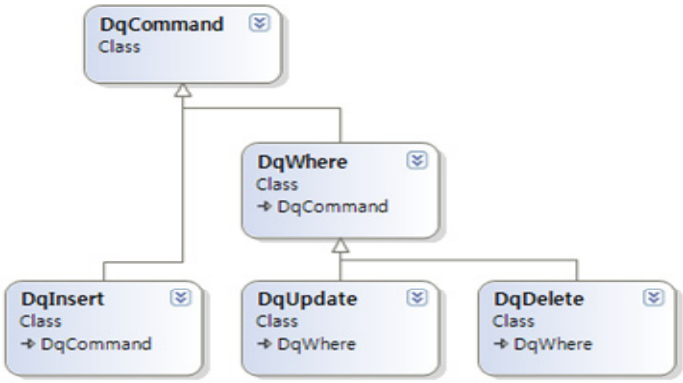


그림 5. Dynamic Query Class의 상속 관계

## 3. SELECT 쿼리의 객체 정의와 상속관계

SELECT 구문을 처리하기 위해서 이를 캡슐화하여 SELECT 구문을 위한 객체(클래스)를 정의한다.

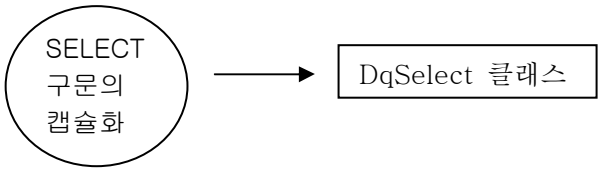


그림 6. SELECT 구문의 객체화

SELECT 쿼리는 UPDATE 쿼리와 DELETE 쿼리처럼 WHERE 구문이 필요하므로 Dynamic Query Class 상속 관계에서 WHERE 구문을 위한 객체(DqWhere)를 그림 7처럼 상속받게 한다.

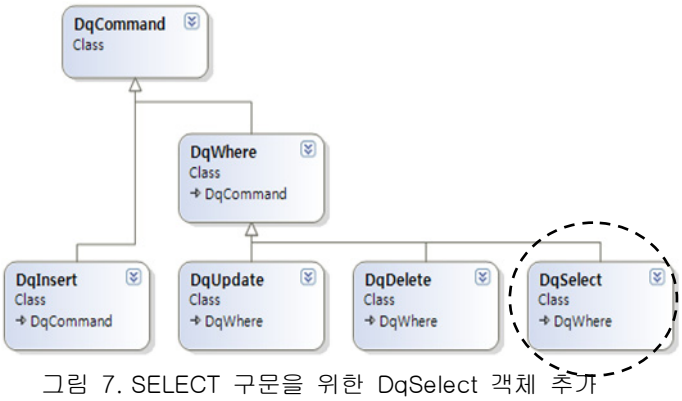


그림 7. SELECT 구문을 위한 DqSelect 객체 추가

SELECT 객체(DqSelect)가 DqWhere 객체를 상속받음으로써, DqWhere 객체의 모든 기능을 자연스럽게 사용할 수 있게 된다. 예를 들어서 AddWhrStr 함수와 AddWhrInInt 함수는 DqWhere 객체에 있는 함수인데[표 3], 어떤 SELECT 쿼리에서 WHERE 구문에 "WHERE Name='신형기'"라는 문자열 필드에 대한 조건 구문이 있다면, AddWhrStr(null, "Name", "=", "신형기") 스트링 처리 함수를, "WHERE Age >= 29" 라는 숫자 필드에 대한 조건 구문이 있다면, AddWhrInInt(null,"Age",">=","29") 숫자 처리 함수를 호출함으로써 기존의 문자열 조합 방식에서는 상당히 신경써야 하는 문자 처리를 단순히 함수를 구분해서 호출하면 처리가 된다.

### 3.1 SELECT 쿼리 객체의 함수와 속성

다음은 DqSelect 객체의 함수와 속성에 대한 설명이다.

표 4. DqSelect 객체의 함수

| 함 수 | 설 명 |
|---|---|
| DqSelect(테이블명) | DqSelect 객체의 생성자 함수로, 객체의 인스턴스를 생성할 때 SELECT할 테이블 명을 인자로 지정해서 호출한다. |
| AddField(필드명) | 테이블에서 SELECT할 필드명을 인자로 호출한다. |
| Orderby(필드명) | 정렬할 필드명을 인자로 호출한다. |
| ExcuteSelect() | 마지막에 호출하며, 최종 SELECT 쿼리를 생성하고 실행해서 레코드 셋을 반환한다. |
| OnJoinField(필드명1, 필드명2) | 조인 관계에 있는 두 테이블의 조인 필드들을 지정한다. |

표 5. DqSelect 객체의 속성

| 속 성 | 설 명 |
|---|---|
| InnerJoin | 조인 관계에서 INNER JOIN이 되는 |

| | |
|---|---|
| | 테이블을 지정한다. |
| LeftJoin | 조인 관계에서 LEFT JOIN이 되는 테이블을 지정한다. |
| RightJoin | 조인 관계에서 RIGHT JOIN이 되는 테이블을 지정한다. |

## 4. SELECT 쿼리 객체의 사용과 장점

SELECT 쿼리를 위한 객체(DqSelect)를 이용해서 단일 SELECT 쿼리뿐만 아니라 둘 이상의 테이블들이 조인된 SELECT 쿼리도 구현할 수 있다.

### 4.1 단일 SELECT 쿼리의 구현

Members라는 테이블에서 Age >= 30 인 조건인 쿼리 "SELECT ID, Name, Age, Zip FROM Members WHERE Age >= 30 ORDER BY Name" 을 구현하고자 한다면 DqSelect 객체를 이용해서 다음과 같이 구현할 수 있다.

```
//Select할 Members 테이블 정의
DqSelect dqs = new DqSelect("Members");
dqs.AddField("ID");//ID 필드
dqs.AddField("Name");//Name 필드
dqs.AddField("Age");//Age 필드
dqs.AddField("Zip");//Zip 필드

// Where Age >= 30
dqs.AddWhrInInt(null,"Age",">=","30");

//ORDER BY Name
dqs.Orderby("Name");

//Select 구문을 실행한다.
DataTable dt = dqs.ExcuteSelect();
```

SELECT할 Members라는 테이블명을 생성자 new DqSelect("Members") 에서 인수로 호출함으로써 dqs라는 객체(인스턴스)를 생성하고, 필요한 필드들을 AddField함수의 인수로 지정해서 차례로 호출한다. ORDER BY Name은 함수 Orderby("Name")로 호출한다. 그리고 상속받은 DqWhere 객체에 있는 AddWhrInInt 함수[표 3]를 호출해서 숫자형 필드의 조건구인 "Where Age >= 30"를 구현할 수 있다. 최종적으로 ExcuteSelect 함수를 호출하면 전체 SELECT구문인 "SELECT ID, Name, Age, Zip FROM Members WHERE Age >= 30 ORDER BY Name" 만들어져 실행되고 레코드 셋이 반환되게 된다.

ExcuteSelect 함수는 내부에서 먼저 전체 SELECT 쿼리를 만들고, 상위 부모 객체인 DqCommand 객체에 있는 함수를 호출해서 DB서버와 접속해서 쿼리를 실행해서 레코드 셋을 받게 되는 것이다..

### 4.2 쿼리 객체 사용의 장점

**⊙ 문자열 조합에 신경 쓸 필요가 없다**

"ID, Name, Age, Zip"에서처럼 필드의 구분을 나타내는 ,(쉼표)나, WHERE Name = '신형기' 에서처럼 '(싱글쿼테이션)의 처리에 신경쓸 필요없이 단순히 함수 AddField("필드명")나 AddWhrStr(null, "Name", "=", "신형기")를 호출하는 것으로써 처리가 된다.

**⊙ 필드 순서에 상관없고 추가와 삭제가 자유롭다**

필드들을 순서에 상관없이 지정할 수 있으며, 추가와 삭제가 자유롭다. 쿼리에서 필드 "ID, Name, Age, Zip"의 순서에 상관없이 "Name, Zip, Age,ID" 순서로 필드를 지정하고자 한다면 다음과 같이 이 순서로 호출한다.

```
dqs.AddField("Name");//Name 필드
dqs.AddField("Zip");//Zip 필드
dqs.AddField("Age");//Age 필드
dqs.AddField("ID");//ID 필드
```

또한, ID와 Zip 필드가 필요없다면 삭제하거나 앞에서 주석처리만 해준다. 또한, 다른 필드를 추가하고자 한다면 순서에 상관없이 다음과 같이 AddField 함수로 추가해서 호출한다.

```
//dqs.AddField("ID");//ID 필드
dqs.AddField("Name");//Name 필드
dqs.AddField("Age");//Age 필드
//dqs.AddField("Zip");//Zip 필드
dqs.AddField("Birthday");//Birthday 필드추가
```

최종 SELECT 쿼리는 이 결과들이 반영되어 만들어지고 실행되게 된다.

### 4.3 둘 이상의 테이블의 JOIN 쿼리의 구현

다음 표 6과 같이 2개의 테이블이 각각 조건을 가지고 조인된 쿼리가 있다면,

표 6. 2개의 테이블이 조인된 쿼리

```
SELECT Members.Name, Members.Age, Members.Zip,
OrderList.ItemNumber, OrderList.ItemName
        FROM Members INNER JOIN OrderList ON
Members.ID=OrderList.FKID
        WHERE Members.Age >= 30 AND OrderList.Date >
'2010-01-01'
```

이 쿼리는 다음 그림 8과 같이 2개의 테이블 상태로

분리해 볼 수 있다.

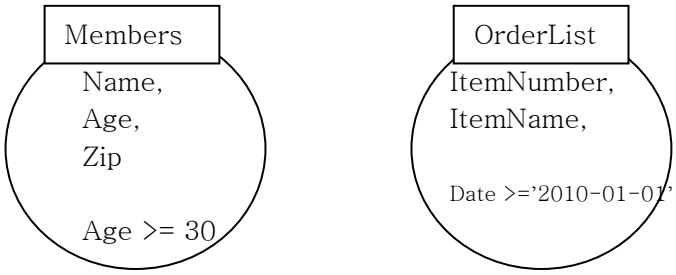


그림 8. 조인된 쿼리에서 각 테이블의 상태 추출

이렇게 분리된 각 테이블의 상태를 SELECT 객체(DqSelect)로 각각 구현할 수 있고, 두 테이블의 Join 관계를 지정할 수 있다. 다음은 그 구현 방법이다.

```
//Members 테이블의 객체 정의
DqSelect dqMembers = new DqSelect("Members");
dqMembers.AddField("Name");//Name 필드
dqMembers.AddField("Age");//Age 필드
dqMembers.AddField("Zip");//Zip 필드

// Where Age >= 30
dqMembers.AddWhrInInt(null, "Age", ">=", "30");

//OrderList 테이블의 객체 정의
DqSelect dqOrderList = new DqSelect("OrderList");
dqOrderList.AddField("ItemNumber");//ItemNumber 필드
dqOrderList.AddField("ItemName");//ItemName 필드

// Where Date > '2010-01-01
dqOrderList.AddWhrStr("AND", "Date", ">", "2010-01-31");

//Members 테이블과
//OrderList 테이블의 Innere Join 관계 지정
dqMembers.InnerJoin = dqOrderList;
//조인 필드 지정
dqMembers.OnJoinField("ID", "FKID");

//전체 조인된 쿼리가 실행된다.
DataTable dt = dqMembers.ExcuteSelect();
```

각 테이블을 단일 SELECT 쿼리를 구현하는 방법과 같은 방식으로 각 테이블에 대한 객체를 각각 정의하고 AddField함수를 호출해서 필드를 추가하고, 조건 구문에서 숫자 필드이면 AddWhrInt 함수를, 문자 필드이면 AddWhrStr 함수를 호출한다. 그리고 INNER JOIN이면 InnerJoin 속성에 대상 테이블 객체(dqOrderList)를 지정한다. 두 테이블의 조인을 이어주는 필드들은 OnJoin Field함수를 호출해서 첫번째 인수에는 Members 테이블의 ID 필드를, 둘째 인수에는 OrderList 테이블의 FKID 필드를 지정하여 호출한다. 마지막 단계에서 ExcuteSelect 함수를 호출하면 전체 조인된 쿼리인 표 6의 쿼리가 만들어져 실행되고 레코드 셋이 반환된다.

### 4.4 다양한 조인 쿼리들의 테스트 평가

다음은 몇 개의 다양한 형태의 조인 쿼리들을 DqSelect 객체를 이용하여 구현한 테스트 결과를 보인다.

표 7. LEFT 조인 쿼리

```
SELECT Members.Name,Members.Email, Members.Age,
Members.Zip, OrderList.ItemNumber, OrderList.ItemName,
OrderList.OrderDate
            FROM Members LEFT JOIN OrderList ON
Members.ID=OrderList.FKID ORDER BY Members.Name
```

표 7의 LEFT 조인쿼리는 Members 테이블에 있는 모든 회원들에 대해서 조회 상품을 조회하기 위해서 OrderList 테이블과 LEFT 조인을 한 쿼리로써 DqSelect 객체로 구현하면 다음과 같다.

```
//Members 테이블의 객체 정의
DqSelect dqMembers = new DqSelect("Members");
dqMembers.AddField("Name");//Name 필드
dqMembers.AddField("Email");//Email 필드
dqMembers.AddField("Age");//Age 필드
dqMembers.AddField("Zip");//Zip 필드

//ORDER BY Members.Name
dqMembers.Orderby("Name");

//OrderList 테이블의 객체 정의
DqSelect dqOrderList = new DqSelect("OrderList");
dqOrderList.AddField("ItemNumber");//ItemNumber 필드
dqOrderList.AddField("ItemName");//ItemName 필드
dqOrderList.AddField("OrderDate");//OrderDate 필드

//Members 테이블과 OrderList 테이블의 LEFT JOIN 관계 지정
dqMembers.LeftJoin = dqOrderList;

//조인 필드 지정
dqMembers.OnJoinField("ID", "FKID");

//전체 조인된 쿼리가 실행된다.
DataTable dt = dqMembers.ExcuteSelect();
```

마지막에 호출된 ExcuteSelect 함수에 의해서 표 7의 조인 쿼리가 만들어져 실행되고 레코드 셋이 반환된다.

표 8. RIGHT 조인 쿼리

```
SELECT Members.Name,Members.Email, Members.Age,
Members.Zip, Address.Ci, Address.Gu, Address.Dong
            FROM Address RIGHT JOIN Members ON
Address.Zip=Members.Zip
            WHERE Address.Ci = '서울시' AND Address.GU =
'서대문구' AND Address.Dong = '창천동'
```

표 8의 RIGHT 조인 쿼리는 Members 테이블에 있는 모든 회원들에 대한 주소를 조회하기 위한 위해서 Address 테이블과 RIGHT 조인을 한 쿼리로써 DqSelect 객체로 구현하면 다음과 같다.

```
//Members 테이블의 객체 정의
DqSelect dqMembers = new DqSelect("Members");
dqMembers.AddField("Name");//Name 필드
dqMembers.AddField("Email");//Email 필드
dqMembers.AddField("Age");//Age 필드
dqMembers.AddField("Zip");//Zip 필드

//Address 테이블 정의
DqSelect dqAddress = new DqSelect("Address");
dqAddress.AddField("Ci");//시 필드
dqAddress.AddField("Gu");//구 필드
dqAddress.AddField("Dong");//동 필드

//WHERE Address.Ci = '서울시'
dqAddress.AddWhrStr(null, "Ci", "=", "서울시");
//AND Address.GU = '서대문구'
dqAddress.AddWhrStr("AND", "Gu", "=", "서대문구");
//AND Address.Dong = '창천동'
dqAddress.AddWhrStr("AND", "Dong", "=", "창천동");

//Address 테이블에 RIGHT JOIN 이 되는
//Members 테이블 지정
dqAddress.RightJoin = dqMembers;

//조인 필드 지정 - 첫번째 Address 테이블의 필드,
//두번째 dqMembers 테이블의 필드
dqAddress.OnJoinField("Zip", "Zip");

//전체 조인된 쿼리가 실행된다.
DataTable dt = dqAddress.ExcuteSelect();
```

마지막에 호출된 ExcuteSelect 함수에 의해서 표 8의 조인 쿼리가 만들어져 실행되고 레코드 셋이 반환된다. WHERE 구에서 AND로 연결된 여러 조건들이 AddWhrStr 함수에서 “AND” 인자의 연속 호출로 처리됨을 볼 수 있다.

테스트 결과는 간단한 단일 SELECT 쿼리에서부터 둘 이상의 조인 SELECT 쿼리들까지 DqSelect 객체를 이용해서 원래 제시된 동일한 쿼리 결과가 수행되는 것을 확인 할 수 있었다.

## 5. 결론 및 향후 연구

본 논문은 SELECT 구문을 객체화하여 사용할 수 있고, 상속 관계도 만들 수 있음을 보이고 있다. 특히, SELECT 쿼리 객체를 이용해서 둘 이상의 테이블 JOIN 관계도 구현될 수 있음을 보이고 있다. 따라서 기존 쿼리의 상세한 부분의 구현까지도 객체로 정의하여 표현할 수 있음을 제시하였다.

또한, 기존 쿼리 작성과 관리의 불편함을 해소하고 쿼리 작성에 일관성 있는 규칙을 적용하기 위하여 객체를 사용한 쿼리 작성을 제시하고 테스트를 수행하였다.

향후 SELECT 쿼리 처리에서 서브 쿼리들의 표현의 정의와 GROUP과 집계 기능, 그리고 더 복잡한 조인 쿼리들의 구현을 추가할 계획이다.

## 참고 문헌


[1] 신형기, “Dynamic Query Class를 이용한 효율적인 쿼리 작성”, 한국컴퓨터정보학회 2008년 제39차 동계학술발표논문집 제16권 제2호, pp.453-460, 2008.

[2] Donald Kossmann, ”The state of the art in distributed query processing”, ACM Computing Surveys Volume 32, pp.422-469, 2000

[3] Chris Anley, ”Advanced SQL Injection In SQL Server Applications”, An NGSSoftware Insight Security Research, 2002

[4] Surajit Chaudhuri, ”An overview of query optimization in relational systems”, Symposium on Principles of Database Systems pp.34-43, 1998

[5] Peter Wegner, “Concepts and paradigms of object-oriented programming”, ACM SIGPLAN OOPS Messenger Volume 1, pp.7-87, 1990

[6] Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W, ”Object Oriented Modeling and Design”, PRENTICE HALL, BOOK DISTRIBUTION CENTER,1991

[7] Craig Larman, ”Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process”, Prentice Hall PTR , 2001

[8] Clinton Begin, Brandon Goodin, ”iBatis in Action”, Manning Publications, 2007

[9] Christian Bauer, Gavin King, ”Hibernate in Action”, Manning Publications, 2005

[10] 정광선,백종현, ”영속성 관리를 위한 오픈 소스 적용 연구”, 한국정보과학회 학술발표논문집 2005 가을 학술발표 문집(II)제32 제2호, pp.970-972, 2005. 11