2-1. SQLQueryFactory(오라클/MySQL에서 쿼리타입 생성을 위한 MAVEN 설정)

- SQLQueryFactory는 자바쪽에 엔티티를 생성하지 않은 상태에서 DB에 질의 하기 위해 사용하는데, JPA의 메소드 기반으로 DB에 쿼리 할 수 있다. 그렇게 하기 위해 DB 스키마와 같은 쿼리 타입(Query Type)을 자바단에 만들어 두어야 하는데 그 과정을 Code Generation(코드 제너레이션) 이라고 한다.

- 메이븐을 통해 쿼리 타입을 생성하고, Spring Data JPA에서 메소드 기반으로 타입 세이프하게 DB에 쿼리하기 위해서는 메이븐 설정에 querydsl-sql 또는 querydsl-sql-spring(스프링에서 사용하는 경우) 의존성을 추가하고 querydsl-maven-plugin을 이용하여 DB스키마 구조대로 쿼리를 위한 쿼리타입 클래스(QXXX)를 만들 수 있다.

- SQL문을 DB에서 직접 사용하는 SQL구문으로 만들어 네이티브 쿼리(Native Query)로 실행한다면 쿼리를 SQL 문자열로 만들어야 한다. 이 경우 쿼리 디버깅과 구문오류, 오타등 예기치 않은 오류가 발생할 가능성이 크다. SQLQueryFactory을 이용하면 기존 Native SQL 형태로 SQL구문을 사용하던 부분을 JPA 메소드 기반 형식으로 일부 사용할 수 있을 것이다.

- pom.xml에 아래 의존성을 추가하자.

```
<dependency>
        <groupId>com.querydsl</groupId>
        <artifactId>querydsl-jpa</artifactId>
        <version>${querydsl.version}</version>
</dependency>
<dependency>
        <groupId>com.querydsl</groupId>
        <artifactId>querydsl-sql-spring</artifactId>
        <version>${querydsl.version}</version>
</dependency>
<!-- 쿼리타입 검증을 위한 Hibernate Validator -->
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
</dependency>
```

- Maven을 이용하여 DB스키마 구조대로 쿼리 타입을 생성 하기 위해서는 아래 plugin을 추가

해야 한다. (MySQL/Maris DB 예문)

```
<plugins>
        ...
<plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
                <plugin>
                        <groupId>com.querydsl</groupId>
                        <artifactId>querydsl-maven-plugin</artifactId>
                        <version>${querydsl.version}</version>
                        <executions>
                                <execution>
                                        <goals>
                                                <goal>export</goal>
                                        </goals>
                                </execution>
                        </executions>
                        <configuration>
                                <jdbcDriver>com.mysql.jdbc.Driver</jdbcDriver>
                                <jdbcUrl>jdbc:mysql://localhost/nativesql1 </jdbcUrl>
                                <jdbcUser>root</jdbcUser>
                                <jdbcPassword>1111</jdbcPassword>
                                <packageName>jpa.model</packageName>
                                <targetFolder>target/generated-
sources/java</targetFolder>
                                <namePrefix>S</namePrefix>
                                <exportBeans>true</exportBeans><!-- targetFolder에
Dept.java, Emp.java를
                                        생성 -->
                        </configuration>
                        <dependencies>
                                <dependency>
                                        <groupId>mysql</groupId>
                                        <artifactId>mysql-connector-java</artifactId>
                                        <version>5.1.38</version>
                                        <scope>compile</scope>
                                </dependency>
```

```
                                    </dependencies>
                            </plugin>          ...
 </plugins>
```

- Spring Integration을 이용하기 위해 위에서 querydsl-sql-spring 의존성을 추가 했는데 이 설정은 스프링 예외처리와 스프링의 TransactionManager를 이용하여 Querydsl SQL의 위한 Spring Connection을 제공한다.

```
<dependency>
  <groupId>com.querydsl</groupId>
  <artifactId>querydsl-sql-spring</artifactId>
  <version>${querydsl.version}</version>
</dependency>
```

- MAVEN 설정이 다 되었으면 프로젝트에서 마우스 우측버튼 -> run as -> Maven generate-sources를 클릭하여 쿼리 타입을 생성하면 된다.

- querydsl-maven-plugin을 이용하여 오라클DB의 테이블을 쿼리 타입으로 만들기
- 주의 : 아래 주석에도 있지만 <schemaPattern>TEST</schemaPattern> 를 기술하지 않으면 TEST 계정에 있는 테이블뿐 아니라 해당 유저가 SELECT 가능한 모든 테이블을 대상으로 쿼리타입이 생성된다. (SELECT * FROM ALL_TABLES로 선택되는 모든 테이블)

[pom.xml]

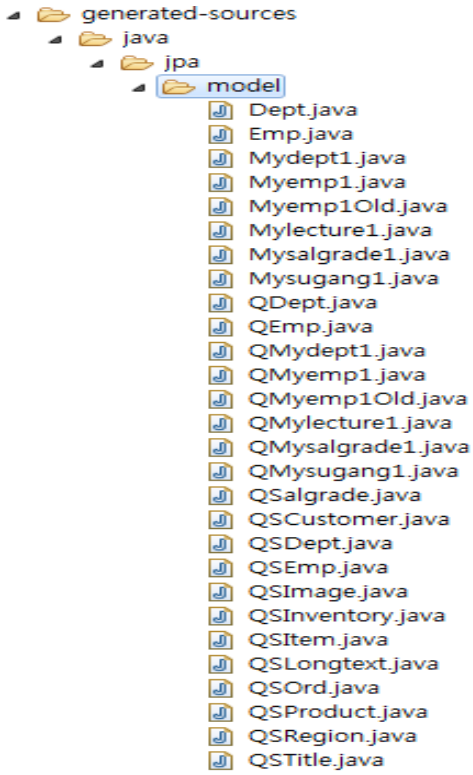
```
<?xml version="1.0" encoding="UTF-8"?>
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>ojc.edu</groupId>
        <artifactId>ojc.nativesql2</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>nativesqlexam2</name>
        <description>jpa native sql example</description>
```

```
<parent>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-parent</artifactId>
	<version>1.3.3.RELEASE</version>
	<relativePath /> <!-- lookup parent from repository -->
</parent>

<properties>
	<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	<java.version>1.8</java.version>
	<querydsl.version>4.0.8</querydsl.version>
</properties>

<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>

	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>

	<!-- for querydsl -->
	<dependency>
		<groupId>com.querydsl</groupId>
		<artifactId>querydsl-jpa</artifactId>
		<version>${querydsl.version}</version>
	</dependency>
	<dependency>
		<groupId>com.querydsl</groupId>
		<artifactId>querydsl-sql-spring</artifactId>
		<version>${querydsl.version}</version>
	</dependency>

	<!-- for oracle -->
```

```
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.1.0.7.0</version>
        </dependency>

        <!-- Hibernate Validator -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.querydsl</groupId>
                <artifactId>querydsl-maven-plugin</artifactId>
                <version>${querydsl.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>export</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>

    <jdbcDriver>oracle.jdbc.driver.OracleDriver</jdbcDriver>

    <jdbcUrl>jdbc:oracle:thin:@192.168.0.27:1521:onj</jdbcUrl>
                    <jdbcUser>test</jdbcUser>
                    <jdbcPassword>test</jdbcPassword>
                    <packageName>jpa.model</packageName>
                    <exportTable>true</exportTable>
                    <exportView>false</exportView>
```

```
                                        <exportPrimarykey>true</exportPrimarykey>
                                        <!-- schemaPattern을 안쓰면 all_tables로 select할
수 있는 모든 테이블이 export됨 -->
                                        <schemaPattern>TEST</schemaPattern>
                                        <!-- 테이블 이름을 콤마로 구분해서 패턴을 줄 수 있
다.  -->
                                        <tableNamePattern>%</tableNamePattern>

                                        <targetFolder>target/generated-
sources/java</targetFolder>
                                        <namePrefix>Q</namePrefix>
                                        <!-- targetFolder에 오라클의 모든 테이블에 대한 엔
티티(*.java)파일 생성 -->
                                        <exportBeans>true</exportBeans>
                                </configuration>
                                <dependencies>
                                        <dependency>
                                          <groupId>com.oracle</groupId>
                                          <artifactId>ojdbc6</artifactId>
                                           <version>11.1.0.7.0</version>
                                        </dependency>
                                </dependencies>
                        </plugin>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>oracle</id>
                        <name>ORACLE JDBC Repository</name>
                        <url>http://maven.jahia.org/maven2</url>
                </repository>
        </repositories>
</project>
```

프로젝트에서 마우스 우측버튼 -> run as -> Maven generate-sources를 실행하면 아래와 같이 엔티티 및 쿼리타입클래스가 만들어 지는데 Q로 시작되는 클래스가 쿼리 타입이고 Q가 없는 클래스가 엔티티 클래스이다.



2-2. SQLQueryFactory를 위한 Query Type, Configuration 생성하기

■ DB스키마를 Querydsl을 위한 쿼리 타입으로 변환하기 위해서는 다음과 같이 하면 된다.

```
java.sql.Connection conn = ...;
MetaDataExporter exporter = new MetaDataExporter();
exporter.setPackageName("com.myproject.mydomain");
exporter.setTargetFolder(new File("target/generated-sources/java"));
exporter.export(conn.getMetaData());
```

- Configuration은 아래처럼 생성한다. MySQL5.5 이상 이라면 MySQLTemplates, 오라클 이라면 OracleTemplates을 사용하면 된다.

```
//MySQL이라면 MySQLTemplates, Oracle이라면 OracleTemplates를 사용하면 된다.
SQLTemplates templates = new H2Templates();
Configuration configuration = new Configuration(templates);
```

- 아래와 같이 SQLQueryFactory를 생성한 후 쿼리를 만들어 실행하면 된다.

```
SQLQueryFactory queryFactory = new SQLQueryFactory(configuration, dataSource);
```

2-3. 스프링부트에서 SQLQueryFactory생성 및 쿼리사용 예문

[스프링 부트 메인에서 SQLQueryFactory 인스턴스 생성 및 쿼리실행 예문]

- 아래 예문을 실행하기 전에 MAVEN을 통해 쿼리 타입(SDept, SEmp)을 만들어야 한다.

1. 마리아DB에 qtype 이라는 데이터베이스를 생성하고 아해처럼 테이블과 데이터를 만들자.

```
CREATE TABLE `dept` (
`deptno` BIGINT(20) NOT NULL AUTO_INCREMENT,
`dname` VARCHAR(255) NULL DEFAULT NULL,
PRIMARY KEY (`deptno`),
UNIQUE INDEX `UK_ekpf7xfhf1n4m2bcymtf364bq` (`dname`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=3
;

CREATE TABLE `emp` (
`empno` BIGINT(20) NOT NULL AUTO_INCREMENT,
`ename` VARCHAR(255) NULL DEFAULT NULL,
`job` VARCHAR(255) NULL DEFAULT NULL,
`sal` BIGINT(20) NULL DEFAULT NULL,
`deptno` BIGINT(20) NULL DEFAULT NULL,
PRIMARY KEY (`empno`),
INDEX `FK_gbxl70x5ckxun8hi19v4n6dfb` (`deptno`),
CONSTRAINT `FK_gbxl70x5ckxun8hi19v4n6dfb` FOREIGN KEY (`deptno`) REFERENCES `dept`
(`deptno`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=6
;

insert into dept values (1, "영업부");
insert into dept values (2, "교육부");

insert into emp values   (1, "1길동", "교수", 5000, 1);
insert into emp values   (2, "2길동", "강사", 2000, 2);
insert into emp values   (3, "3길동", "교수", 7000, 1);
```

```
insert into emp values   (4, "4길동", "강사", 8000, 2);
insert into emp values   (5, "5길동", "교수", 1000, null);
```

2. 프로젝트 생성

STS에서 File -> New -> Spring Starter Project(스프링 부트)

Project name : qtypetest

Package : jpa

다음 화면에서 SQL쪽의 JPA, MySQL 선택.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project                                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>com.example</groupId>
        <artifactId>demo</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>qtypetest</name>
        <description>Demo project for Spring Boot</description>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>1.4.0.RELEASE</version>
                <relativePath /> <!-- lookup parent from repository -->
        </parent>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
                <java.version>1.8</java.version>
```

```
        <querydsl.version>4.0.8</querydsl.version>

    </properties>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>com.querydsl</groupId>
            <artifactId>querydsl-jpa</artifactId>
            <version>${querydsl.version}</version>
        </dependency>
        <dependency>
            <groupId>com.querydsl</groupId>
            <artifactId>querydsl-sql-spring</artifactId>
            <version>${querydsl.version}</version>
        </dependency>
        <!-- 쿼리타입 검증을 위함 Hibernate Validator -->
        <dependency>
```

```
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-validator</artifactId>
                </dependency>

        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                        <plugin>
                                <groupId>com.querydsl</groupId>
                                <artifactId>querydsl-maven-plugin</artifactId>
                                <version>${querydsl.version}</version>
                                <executions>
                                        <execution>
                                                <goals>
                                                        <goal>export</goal>
                                                </goals>
                                        </execution>
                                </executions>
                                <configuration>
                                        <jdbcDriver>com.mysql.jdbc.Driver</jdbcDriver>
                                        <jdbcUrl>jdbc:mysql://localhost/qtype</jdbcUrl>
                                        <jdbcUser>root</jdbcUser>
                                        <jdbcPassword>1111</jdbcPassword>
                                        <packageName>jpa.model</packageName>

<targetFolder>target/generatedsources/java</targetFolder>
                                        <namePrefix>S</namePrefix>
                                        <exportBeans>true</exportBeans>
                                        <!-- targetFolder에 Dept.java, Emp.java를 생성 -->
                                </configuration>
                                <dependencies>
                                        <dependency>
                                                <groupId>mysql</groupId>
                                                <artifactId>mysql-connector-java</artifactId>
```

```
                                        <version>5.1.38</version>
                                        <scope>compile</scope>
                                    </dependency>
                                </dependencies>
                            </plugin>
                        </plugins>
                    </build>


</project>
```

프로젝트에서 마우스 우측버튼 -> run as -> Maven generate-sources를 실행하면 엔티티 및 쿼리 타입클래스가 만들어 지는데 S로 시작되는 클래스가 쿼리 타입이고 S가 없는 클래스가 엔티티 클래스이다.

3.application.properties 파일은 다음과 같다.

```
//이미 qtypeDB는 생성되어 있도 EMP, DEPT 두 테이블이 존재해 있다.
spring.datasource.platform=mysql
spring.datasource.sql-script-encoding=UTF-8
spring.datasource.url=jdbc:mysql://localhost/qtype?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=1111
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=false
logging.level.jpa=DEBUG
```

4.QtypetestApplication.java

```
package jpa;

import java.sql.Connection;
import java.util.List;

import javax.inject.Provider;
import javax.sql.DataSource;
import javax.transaction.Transactional;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.PlatformTransactionManager;

import com.querydsl.core.Tuple;
import com.querydsl.sql.Configuration;
import com.querydsl.sql.MySQLTemplates;
import com.querydsl.sql.SQLQueryFactory;
import com.querydsl.sql.SQLTemplates;
import com.querydsl.sql.spring.SpringConnectionProvider;
import com.querydsl.sql.spring.SpringExceptionTranslator;

//DB스키마 기준으로 자바쪽에 자동생성한 쿼리타입 클래스, 메이븐설정에서 접두어를 „S"로 했다.
import jpa.model.SDept;
import jpa.model.SEmp;

@SpringBootApplication
public class QtypetestApplication implements CommandLineRunner {
	public static void main(String[] args) {
		SpringApplication.run(QtypetestApplication.class, args);
	}

	@Autowired
	DataSource dataSource;

	@Transactional
	public void run(String... args) {
		// DB스키마구조대로 만든 쿼리 타입
		SEmp emp = new SEmp("emp");
		SDept dept = new SDept("dept");
		////////////////////////////////////////// JPA형식으로 DB에 질의문 생성
		SQLQueryFactory queryFactory = queryFactory();
		List<String> enames = queryFactory.select(emp.ename).from(emp).fetch();
		for (String ename : enames) {
```

```
                        System.out.println(ename);
                }
                System.out.println("----------------");

                List<Tuple> emps = queryFactory.select(emp.ename, dept.dname)
                                .from(emp).innerJoin(dept)
                                .on(emp.deptno.eq(dept.deptno)).fetch();
                for (Tuple row : emps) {
                        System.out.println(row.get(emp.ename) + ":" + row.get(dept.dname));
                }
                System.out.println("----------------");
        }

        public Configuration querydslConfiguration() {
                SQLTemplates templates = MySQLTemplates.builder().build();
                Configuration configuration = new Configuration(templates);
                configuration.setExceptionTranslator(new SpringExceptionTranslator());
                return configuration;
        }

        // 만약 레포지토리에서 주입받으려면 @Bean 어노테이션으로 빈으로 등록후 사용하면
된다.
        public SQLQueryFactory queryFactory() {
                Provider<Connection> provider = new SpringConnectionProvider(dataSource);
                return new SQLQueryFactory(querydslConfiguration(), provider);
        }
}
```

2-4. SQLQueryFactory(쿼리, 조인,orderby,groupBy, DML,DML배치쿼리,partitionBy, over)

■ 쿼리 사용법은 JPAQueryFactory를 사용하는 것과 비슷하다.

```
QEmp emp = new QEmp("e");

List<String> enames = queryFactory.select(emp.ename).from(emp)
    .where(emp.ename.eq("SMITH"))
    .fetch();
```

다음과 같은 SQL구문 이다.

```
SELECT ename FROM EMP WHERE ename = 'SMITH';
```

■ SQLQueryFactory의 cascade 메소드는 다음과 같다.

select: 쿼리의 SELECT LIST(projections). (queryFactory를 사용할 경우 필수는 아니다.)
from: 쿼리 소스
innerJoin, join, leftJoin, rightJoin, fullJoin, on: 조인 구문들, on은 조인조건을 기술한다.
where: 쿼리 필터(and, or 등)
groupBy: 그룹핑
having: 그룹핑 결과(함수)에 조건을 주는 경우 사용.
orderBy: 정렬을 위한 것
limit, offset, restrict: 페이징을 위한 것, limit는 추출되는 결과의 최대치, offset은 SKIP되는 행, restrict는 limit, offset 둘 모두를 정의해야 한다.

■ 조인

```
QEmp emp = QEmp.emp;
QDept dept = QDept.dept;

//내부조인, 조인조건에 맞지않는 즉 부서코드가 없는 사원은 추출되지 않는다.
queryFactory.select(emp.ename, dept.dname)
    .from(emp)
    .innerJoin(emp.deptno,dept)
    .fetch();

//Left Outer Join
queryFactory.select(emp.ename, dept.dname)
    .from(emp)
    .leftJoin(emp.deptno,dept)
    .fetch();
```

아래와 동일하다.

```
queryFactory.select(emp.ename, dept.dname)
        .from(emp)
        .leftJoin(dept).on(emp.deptno.eq(dept.deptno))
        .fetch();
```

■ 정렬(orderBy)

```
//사원명 오름차순, 부서명 오름차순
queryFactory.select(emp.ename, dept.dname)
        .from(emp)
        .innerJoin(emp.deptno,dept)
        .orderBy(emp.ename.asc(), dept.ename.asc())
        .fetch();
```

다음 SQL 구문과 동일하다.

```
SELECT emp.ename, dept.dname
FROM emp, dept
WHERE emp.deptno = dept.deptno
ORDERBY emp.ename ASC, dept.dname ASC
```

■ 그룹핑(groupBy)

```
//Emp 테이블에서 부서별로 그룹핑 해서 부서코드, 부서별 급여평균을 출력하는데
//부서별 급여의 합이 10,000,000원 보다 적은 부서만 추출
queryFactory.select(emp.deptno, emp.sal.avg())
        .from(emp)
        .groupBy(emp.deptno)
        .having(emp.sal.sum().lt(10000000))
        .fetch();
```

// 다음 SQL 구문과 동일하다.

```
SELECT emp.deptno, avg(emp.sal)
FROM emp
GROUP BY emp.deptno
HAVING   sum(emp.sal) < 10000000
```

■ 서브쿼리

```
// Emp 테이블에서 최대급여 사원 추출, 서브쿼리
QEmp emp = new QEmp.emp;
QEmp e = new QEmp("e");

List<Emp> emps = queryFactory.select(emp.all())
                             .from(emp)
                             .where(emp.sal.eq(
                                     SQLExpressions.select(e.sal.max()).from(e)))
                             .fetch();

// 부서별 최대급여받는 사원 추출 , 서브쿼리
List<Emp> emps = queryFactory.selectFrom(emp.all())
                              .from(emp)
                              .where(emp.sal.eq(
                                      SQLExpressions
                                          .select(e.sal.max()).from(e)
                                          .where(emp.dept.deptno.eq(e.dept.deptno))
                                    ))
                             .fetch();
```

■ 리터럴 SELECT

```
queryFactory.select(Expressions.constant(1), Expressions.constant("오라클자바커뮤니티"));
```

■ 분석함수를 위한 윈도우 함수

```
queryFactory.select(SQLExpressions.rowNumber()
            .over()
            .partitionBy(emp.job)
            .orderBy(emp.ename))
       .from(emp)
```

■ insert 구문

```
[칼럼을 기술하는 경우]
QEmp emp = QEmp.emp;
queryFactory.insert(emp)
```

```
    .columns(emp.empnoemp.ename)
    .values(1, "1길동")
    .execute();

[칼럼을 기술 안하는 경우, 테이블의 모든 칼럼에 값을 입력해야 한다.]
QEmp emp = QEmp.emp;
queryFactory.insert(emp)
    .values(1, "1길동","교수",,,,,)
    .execute();

[칼럼기술해서 서브쿼리로 입력하는 경우]
QEmp emp = QEmp.emp;
QEmpBak empbak = QEmpBak.empbak;

queryFactory.insert(emp)
    .columns(emp.empnoemp.ename)
    .select(SQLExpressions.select(empbak.empno.add(1), empbak.ename).from(empbak))
    .execute();

[칼럼기술하지않고 서브쿼리로 입력하는 경우]
queryFactory.insert(emp)
    .select(SQLExpressions.select(empbak.empno.add(1), empbak.ename,,,,,,).from(empbak))
    .execute();
```

■ Update 구문

```
[where절 있는 경우]
QEmp emp = QEmp.emp;
queryFactory.update(emp)
    .where(emp.ename.eq("SMITH"))
    .set(emp.ename, "OJC.ASIA")
    .execute();

[where절 없는 경우]
QEmp emp = QEmp.emp;
queryFactory.update(emp)
    .set(emp.ename, "OJC.ASIA")
    .execute();
```

[DTO, VO같은 빈을 이용하는 경우]

```
queryFactory.update(emp)
    .populate(EmpDTO)
    .execute();
```

■ Delete 구문

[where절 있는 경우]

```
QEmp emp = QEmp.emp;
queryFactory.delete(emp)
    .where(emp.ename.eq("SMITH"))
    .execute();
```

[where절 없는 경우]

```
QEmp emp = QEmp.emp;
queryFactory.delete(emp)
    .execute();
```

■ Update 배치쿼리

```
QEmp emp = QEmp.emp;
queryFactory.insert(emp).values(1, "1길동").execute();
queryFactory.insert(emp).values(2, "2길동").execute();

SQLUpdateClause update = queryFactory.update(emp);
update.set(emp.ename, "11길동").where(emp.ename.eq("1길동")).addBatch();
update.set(emp.ename, "22길동").where(emp.ename.eq("2길동")).addBatch();
```

■ Insert 배치쿼리

```
SQLInsertClause insert = queryFactory.insert(emp);
insert.set(emp.empno, 1).set(emp.ename, "1길동").addBatch();
insert.set(emp.empno, 2).set(emp.ename, "2길동").addBatch();
```

■ Delete 배치쿼리

```
queryFactory.insert(emp).values(1, "1길동").execute();
queryFactory.insert(emp).values(2, "2길동").execute();

SQLDeleteClause delete = queryFactory.delete(emp);
delete.where(emp.ename.eq("1길동")).addBatch();
```

```
delete.where(emp.ename.eq("2길동")).addBatch();
```

2-5. SQLQueryFactory실습,Spring Boot/마리아DB(메이븐설정, 서브쿼리, 조인,orderby,groupBy, insert/update 배치쿼리,partitionBy, over

- DB테이블을 기반으로 자바쪽 프로젝트에 쿼리타입, 엔티티를 자동 생성 한 후 Querydsl의 SQLQueryFactory를 이용하여 DB에 직접 Native SQL 형태의 메소드 기반으로 쿼리해 보자.
- 자바쪽에 엔티티 클래스를 만들지 않고, 플러그인을 통해 DB에 있는 테이블을 기본으로 쿼리타입 클래스를 생성 후 질의해야 하므로 데이터베이스에는 DEPT, EMP 두 테이블이 만들어져 있어야 한다.

■ DDL 스크립트는 다음과 같다. 마리아DB에 "qtypetest" 이라는 이름의 데이터베이스, 그 안에 "DEPT", "EMP" 테이블을 만들자. (이전에 실습한 DB 및 테이블이 있다면 application.properties에 해당 DB로 설정하면 된다.)

```
CREATE TABLE `dept` (
`deptno` BIGINT(20) NOT NULL AUTO_INCREMENT,
`dname` VARCHAR(255) NULL DEFAULT NULL,
PRIMARY KEY (`deptno`),
UNIQUE INDEX `UK_ekpf7xfhf1n4m2bcymtf364bq` (`dname`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=3
;

CREATE TABLE `emp` (
`empno` BIGINT(20) NOT NULL AUTO_INCREMENT,
`ename` VARCHAR(255) NULL DEFAULT NULL,
`job` VARCHAR(255) NULL DEFAULT NULL,
`sal` BIGINT(20) NULL DEFAULT NULL,
`deptno` BIGINT(20) NULL DEFAULT NULL,
PRIMARY KEY (`empno`),
INDEX `FK_gbxl70x5ckxun8hi19v4n6dfb` (`deptno`),
CONSTRAINT `FK_gbxl70x5ckxun8hi19v4n6dfb` FOREIGN KEY (`deptno`) REFERENCES
`dept` (`deptno`)
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
AUTO_INCREMENT=6
;

insert into dept values (1, "영업부")
insert into dept values (2, "교육부")

insert into emp values  (1, "1길동", "교수", 5000, 1)
insert into emp values  (2, "2길동", "강사", 2000, 2)
insert into emp values  (3, "3길동", "교수", 7000, 1)
insert into emp values  (4, "4길동", "강사", 8000, 2)
insert into emp values  (5, "5길동", "교수", 1000, null)
```

STS -> Spring Starter Project

project name : nativesqlexam

Type : MAVEN

package : jpa

SQL -> JPA, MySQL 선택

마리아 DB 및 HeidiSQL 설치는 다음 URL 참조

http://ojc.asia/bbs/board.php?bo_table=LecSpring&wr_id=524

■ DB스키마 구조대로 자바쪽에 쿼리 타입(Query Type)을 생성해야 하므로 querydsl-maven-plugin 플러그인을 추가해야 하고, Spring Data JPA에서 SQLQueryFactory를 이용하여 Native SQL을 JPA 메소드 기반으로 실행하기 위해 기존 Querydsl 설정에 추가로 querydsl-sql-spring 의존성과 쿼리 타입의 Hibernate Validation을 위해 hibernate-validator 추가해야 한다.

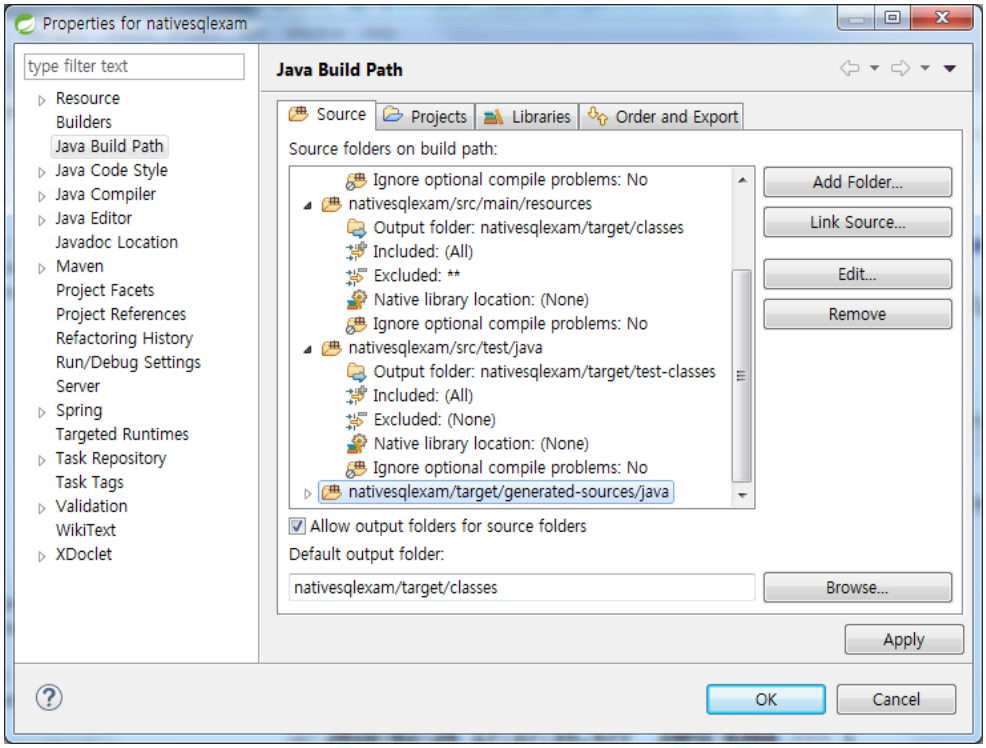
[pom.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>ojc.edu</groupId>
	<artifactId>ojc.nativesql</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>nativesql-exam1</name>
	<description>jpa native sql example</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.3.3.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<java.version>1.8</java.version>
		<querydsl.version>4.0.8</querydsl.version>
	</properties>

	<dependencies>
		<dependency>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.38</version>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>com.querydsl</groupId>
                <artifactId>querydsl-jpa</artifactId>
                <version>${querydsl.version}</version>
        </dependency>
        <dependency>
                <groupId>com.querydsl</groupId>
                <artifactId>querydsl-sql-spring</artifactId>
                <version>${querydsl.version}</version>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-validator</artifactId>
        </dependency>
</dependencies>
<build>
        <plugins>
                <plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
                <plugin>
                        <groupId>com.querydsl</groupId>
                        <artifactId>querydsl-maven-plugin</artifactId>
```

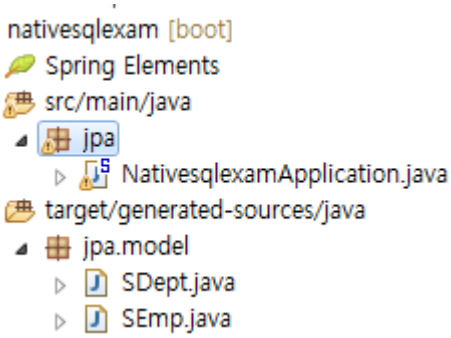
```
                <version>${querydsl.version}</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>export</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <jdbcDriver>com.mysql.jdbc.Driver</jdbcDriver>

        <jdbcUrl>jdbc:mysql://localhost/qtypetest?createDatabaseIfNotExist=true</jdbcUrl>
                    <jdbcUser>root</jdbcUser>
                    <jdbcPassword>1111</jdbcPassword>
                    <packageName>jpa.model</packageName>
                    <targetFolder>target/generated-
sources/java</targetFolder>
                    <namePrefix>S</namePrefix>
                <!-- targetFolder에 Dept.java, Emp.java를  생성 -->
                    <exportBeans>true</exportBeans>
                </configuration>
                <dependencies>
                    <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.38</version>
                        <scope>compile</scope>
                    </dependency>
                </dependencies>
            </plugin>
        </plugins>
    </build>
</project>
```

■ target/gernerated-sources/java 폴더를 프로젝트 buildpath의 src에 추가하자.



- 먼저 쿼리 타입(Query Type)을 생성하자. (프로젝트 -> 우측 마우스 클릭 -> run as -> generat-sources 실행), 쿼리타입 클래스의 접두어는 pom.xml 파일에 'S'로 설정되어 있다.

- 프로젝트 아래 target/generated-sources/java의 jpa.model 패키지에 두개의 쿼리 타입 클래스가 생성되었을 것이다.



[application.properties] – 이미 만들어져 있는 데이터베이스를 사용하려면 nativesql1 대신 그 데이터베이스명을 설정하자.

```
spring.datasource.platform=mysql
spring.datasource.sql-script-encoding=UTF-8
spring.datasource.url=jdbc:mysql://localhost/qtypetest?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=1111
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

```
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=false
logging.level.jpa=DEBUG
```

[NativesqlexamApplication.java]

```
package jpa;

import java.awt.geom.AffineTransform;
import java.sql.Connection;
import java.util.List;

import javax.inject.Provider;
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.Transactional;

import com.querydsl.core.Tuple;
import com.querydsl.sql.Configuration;
import com.querydsl.sql.MySQLTemplates;
import com.querydsl.sql.SQLExpressions;
import com.querydsl.sql.SQLQueryFactory;
import com.querydsl.sql.SQLTemplates;
import com.querydsl.sql.dml.SQLDeleteClause;
import com.querydsl.sql.dml.SQLInsertClause;
import com.querydsl.sql.dml.SQLUpdateClause;
import com.querydsl.sql.spring.SpringConnectionProvider;
import com.querydsl.sql.spring.SpringExceptionTranslator;

//DB스키마에서 자동생성한 쿼리타입클래스
import jpa.model.SDept;
import jpa.model.SEmp;

@SpringBootApplication
```

```
public class NativesqlexamApplication implements CommandLineRunner {

	public static void main(String[] args) {
		SpringApplication.run(NativesqlexamApplication.class, args);
	}

	@Autowired
	DataSource dataSource;

	@Transactional()
	public void run(String... args) {
		//쿼리 타입
		SEmp emp = new SEmp("emp");
		SDept dept = new SDept("dept");

		////////////////////////////////////////////// JPA형식으로 DB에 쿼리
		SQLQueryFactory queryFactory = queryFactory();

		// Emp 테이블에서 모든 사원의 이름을 이름 내림차순으로 출력
		List<String> enames = queryFactory.select(emp.ename)
					.from(emp)
					.orderBy(emp.ename.desc())
					.fetch();
		for (String ename : enames) {
			System.out.println(ename);
		}
		System.out.println("----------------1");

		// Emp, Dept를 조인하여 사원명, 부서명 출력, 5길동은 부서가 없으므로 출력안
됨
		List<Tuple> emps1 = queryFactory.select(emp.ename, dept.dname)
					.from(emp)
					.innerJoin(dept)
					.on(emp.deptno.eq(dept.deptno))
					.fetch();

		for (Tuple row : emps1) {
			System.out.println(row.get(emp.ename) + ":" + row.get(dept.dname));
		}
```

```
System.out.println("----------------2");

// Emp, Dept를 조인하여 사원명, 부서명 출력, 부서없는 5길동도 출력
List<Tuple> emps2 = queryFactory.select(emp.ename, dept.dname)
                    .from(emp)
                    .leftJoin(dept)
                    .on(emp.deptno.eq(dept.deptno))
                    .fetch();

for (Tuple row : emps2) {
        System.out.println(row.get(emp.ename) + ":" + row.get(dept.dname));
}
System.out.println("----------------3");

// Emp 테이블에서 부서별로 그룹핑하여 부서명, 급여합 출력
// 단 급여평균이 2000 이상인 부서만
List<Tuple> emps3 = queryFactory.select(dept.dname, emp.sal.sum())
                    .from(emp)
                    .innerJoin(dept)
                    .on(emp.deptno.eq(dept.deptno))
                    .groupBy(emp.deptno)
                    .having(emp.sal.avg().gt(2000))
                    .fetch();

for (Tuple row : emps3) {
        System.out.println(row.get(dept.dname) + ":"
                        + row.get(1, Long.class));
}
System.out.println("----------------4");

//Emp 테이블에서 급여최소인 사원의 모든 칼럼 추출
SEmp e = new SEmp("e");
List<Tuple> emps4 = queryFactory.select(emp.all())
.from(emp)
    .where(emp.sal.eq(
                SQLExpressions.select(e.sal.min()).from(e)))
    .fetch();
for (Tuple row : emps4) {
        System.out.println(row.get(emp.ename) + ":"
```

```
                        + row.get(emp.sal));
        }
        System.out.println("----------------4");

        //Emp 테이블에서 job별 최소급여 사원의 ename, job, sal 출력
        List<Tuple> emps5 = queryFactory.select(emp.ename, emp.job, emp.sal)
                .from(emp)
                .where(emp.sal.eq(
                        SQLExpressions.select(e.sal.min()).from(e)
                          .where(emp.job.eq(e.job))))
                    .fetch();

        for (Tuple row : emps5) {
                System.out.println(row.get(emp.job) + ":"
                                + row.get(emp.ename) + ":"
                                        + row.get(emp.sal));
        }
        System.out.println("----------------5");

//"1길동" 사원과 같은 부서에 있는 사원중 최대급여 사원의 이름 및 급여, 부서명출력
        List<Tuple> emps6 = queryFactory.select(emp.ename, emp.sal, dept.dname)
                            .from(emp)
                            .innerJoin(dept).on(emp.deptno.eq(dept.deptno))
                            .where(emp.deptno.eq(
                                                SQLExpressions
                                                .select(e.deptno)
                                                .from(e)
                                                .where(e.ename.eq("1길동"))
                        ).and(emp.sal.eq(
                                        SQLExpressions
                                        .select(e.sal.max())
                                        .from(e)
                                        .where(emp.deptno.eq(e.deptno))
                    )))
        .fetch();

        for (Tuple row : emps6) {
                System.out.println(row.get(emp.ename) + ":"
                                + row.get(emp.sal) + ":"
```

```
                                                + row.get(dept.dname));
                }
                System.out.println("----------------6");

                // 그룹함수 : Emp에서 ename, deptno, sal를 출력하는데
                // 부서안에서의 급여 순위도 출력하는데 이름 내림차순으로 출력
                //아래 구문은 MySQL에서 지원하지 않는다. ORACLE, MS-SQL에서 확인하자.
//              List<Tuple> emps7 = queryFactory
//                              .select(emp.ename, emp.deptno, emp.sal,
//                                                      SQLExpressions
//                                                      .rank()
//                                                      .over()
//                                                      .partitionBy(emp.deptno)
//                                                      .orderBy(emp.sal.desc())
//
//                                                      )
//                              .from(emp)
//                              .orderBy(emp.ename.desc())
//                              .fetch();
//
//              for (Tuple row : emps7) {
//                      System.out.println(row.get(emp.ename) + ":"
//                                      + row.get(emp.deptno) + ":"
//                                                      + row.get(emp.sal) + ":"
//                                                      + row.get(3, Integer.class)
//
//                                              );
//              }
//              System.out.println("----------------6");


                //insert
                long rowaffected = queryFactory.insert(emp)
                                .columns(emp.ename, emp.job, emp.sal)
                                .values("6길동", "교수", 5000)          //.select로 서브쿼리형태입력
가능
                                .execute();

                System.out.println(rowaffected + "건 저장!");
```

```
//update
rowaffected = queryFactory.update(emp)
                .where(emp.ename.eq("6길동"))
                .set(emp.sal, 9999L)
                .execute();

System.out.println(rowaffected + "건 수정!");

//delete
rowaffected = queryFactory.delete(emp)
                .where(emp.ename.eq("6길동"))
                .execute();

System.out.println(rowaffected + "건 삭제!");

//update 배치 쿼리
SQLUpdateClause myUpdate = queryFactory.update(emp);

myUpdate.set(emp.sal, 3000L).where(emp.empno.eq(1L)).addBatch();
myUpdate.set(emp.sal, 3100L).where(emp.empno.eq(2L)).addBatch();
rowaffected = myUpdate.execute();

System.out.println(rowaffected + "건 update 완료!");

//insert 배치쿼리
SQLInsertClause myInsert = queryFactory.insert(emp);
myInsert.columns(emp.ename, emp.job, emp.sal).values("7길동", "교수",
7777).addBatch();
myInsert.columns(emp.ename, emp.job, emp.sal).values("8길동", "교수",
8888).addBatch();
rowaffected = myInsert.execute();

System.out.println(rowaffected + "건 insert 완료!");

//delete 배치쿼리
SQLDeleteClause myDelete = queryFactory.delete(emp);
myDelete.where(emp.ename.eq("7길동")).addBatch();
myDelete.where(emp.ename.eq("8길동")).addBatch();
```

```
                rowaffected = myDelete.execute();

                System.out.println(rowaffected + "건 delete 완료!");
        }

        public PlatformTransactionManager transactionManager() {
                return new DataSourceTransactionManager(dataSource);
        }

        public Configuration querydslConfiguration() {
                SQLTemplates templates = MySQLTemplates.builder().build();

                Configuration configuration = new Configuration(templates);
                configuration.setExceptionTranslator(new SpringExceptionTranslator());
                return configuration;
        }

        public SQLQueryFactory queryFactory() {
                Provider<Connection> provider = new SpringConnectionProvider(dataSource);
                return new SQLQueryFactory(querydslConfiguration(), provider);
        }
}
```