5-1. 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]

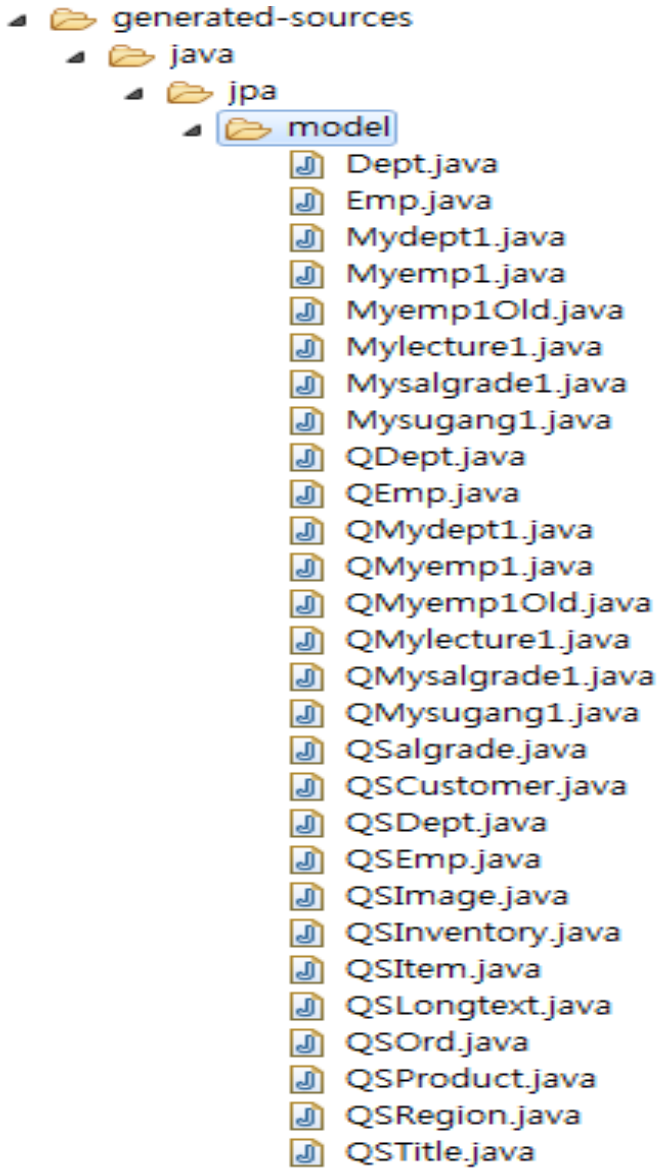
```
<?xml version="1.0" encoding="UTF-8"?>
<project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</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 있다. -->
                                        <tableNamePattern>%</tableNamePattern>
                                        <targetFolder>
                                        target/generated-sources/java</targetFolder>
                                        <namePrefix>Q</namePrefix>
                                        <!-- targetFolder에 오라클의 모든 테이블에 대한
                                            엔티티(*.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>https://maven.oracle.com</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.



5-2. SQLQueryFactory를 위한 Query Type, Configuration 생성하기

■ 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);
```

5-3. 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.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 위핚 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);
        }
}
```