Querydsl 전반적인 오라클 SQL쿼리실습

http://ojc.asia, http://ojcedu.com

- SQLQueryFactory, OracleQuery를 이용하여 이용하여 Native SQL을 JPA 메소드 기반으로 쿼리해 보자.
- 자바쪽에 엔티티 클래스를 만들지 않고, 플러그인을 통해 오라클 DB에 있는 테이블을 기본으로 쿼리 타입 클래스를 생성 후 질의해야 하므로 데이터베이스에는 최소 MYEMP1, MYEMP1_OLD, MYDEPT1, DEPT, EMP, MYSALGRADE1, SALGRADE 일곱개의 테이블이 만들어져 있어야 한다.

- 오라클 쪽 DDL 스크립트는 다음과 같다.

1. test 계정 생성

SQL>conn / as sysdba
SQL> CREATE USER **TEST** IDENTIFIED BY **TEST**
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;
SQL> GRANT CONNECT, RESOURCE, DBA TO TEST;

2. 아래 URL에서 필요한 테이블을 생성하자.
(MYEMP1 테이블은 1000만건 정도 데이터를 저장하자.)

http://ojc.asia/bbs/board.php?bo_table=LecHINT&wr_id=117

STS -> Spring Starter Project
project name : nativesqlexam2
Type : MAVEN
package : jpa
Core -> Lombok, SQL -> JPA, WEB -> Web 선택

Querydsl MAVEN 설정은 아래 URL에서 참조
http://ojc.asia/bbs/board.php?bo_table=LecSpring&wr_id=543

롬복(Lombok)설치는 다음 URL 참조
http://ojc.asia/bbs/board.php?bo_table=LecSpring&wr_id=561

- 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.
- JPA에서 DB로 날아가는 쿼리를 로깅하기 위해 Driverspy를 사용했다.

[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>4.0.9</querydsl.version>
	</properties>
```

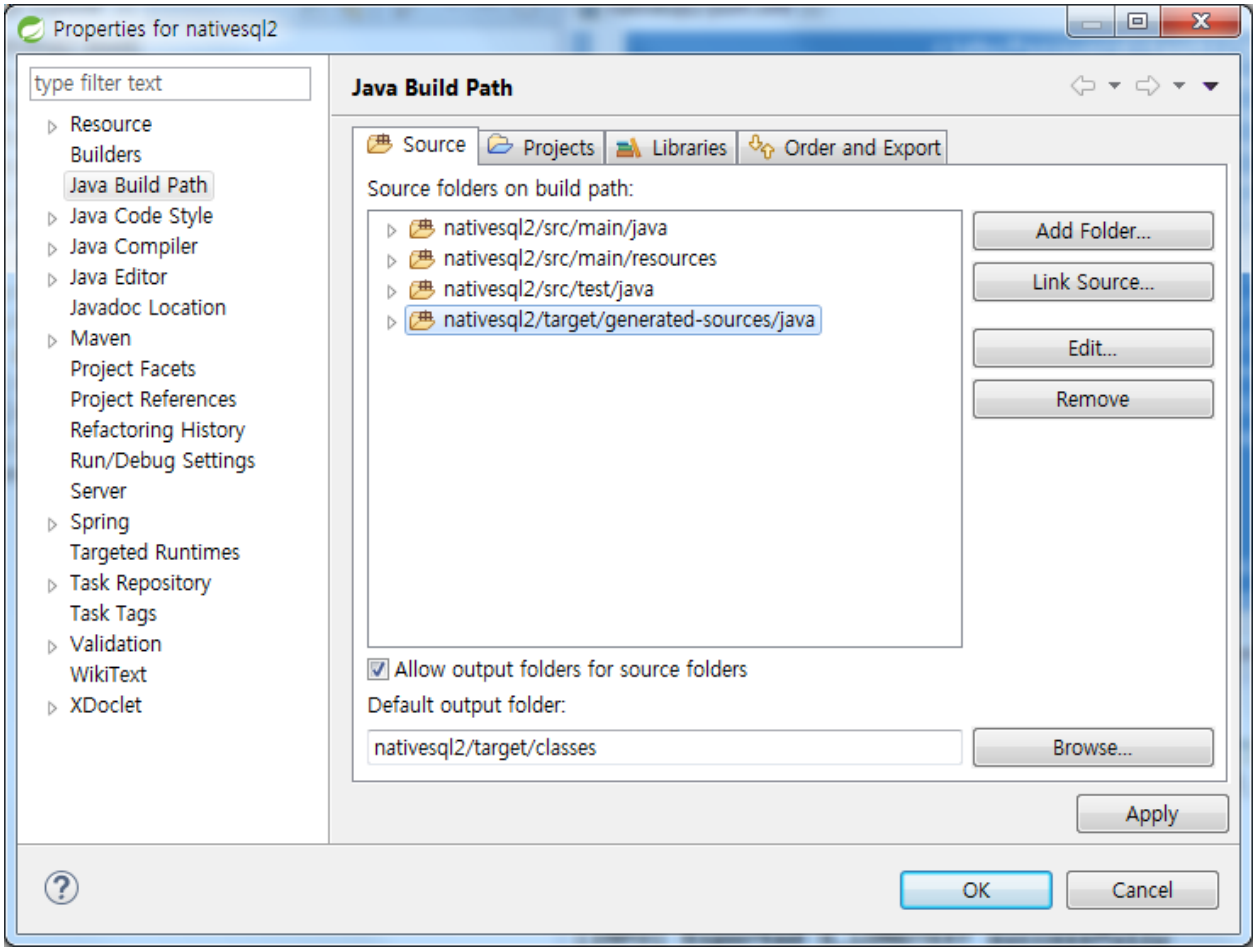
```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>

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

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

	<!-- 롬복  -->
	<dependency>
		<groupId>org.projectlombok</groupId>
		<artifactId>lombok</artifactId>
		<version>1.16.6</version>
	</dependency>

	<!-- SQL 로깅용 -->
	<dependency>
		<groupId>org.bgee.log4jdbc-log4j2</groupId>
		<artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
		<version>1.16</version>
	</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>

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

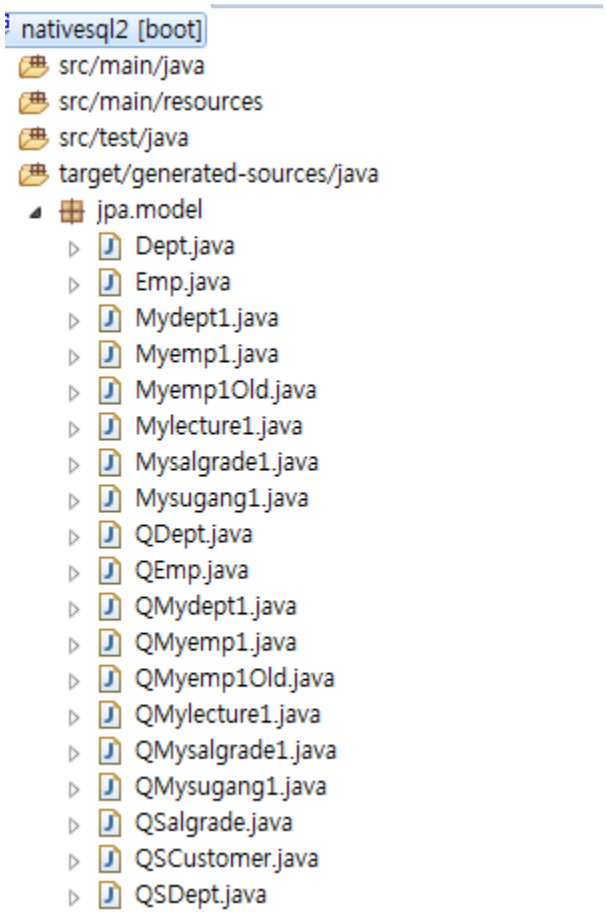
```
                        <id>oracle</id>
                        <name>ORACLE JDBC Repository</name>
                        <url>https://maven.oracle.com</url>
                    </repository>
            </repositories>
</project>
```

- target/gernerated-sources/java 폴더를 프로젝트 buildpath의 src에 추가하자. (target/gernerated-sources/java 폴더에서 우측 마우스 클릭 -> build path -> Use As Source Folder 선택)



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

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



[src/main/resources/application.properties]

```
spring.datasource.platform=oracle
spring.datasource.sql-script-encoding=UTF-8
spring.datasource.url=jdbc:log4jdbc:oracle:thin:@192.168.0.27:1521:onj
spring.datasource.username=test
spring.datasource.password=test
#spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.driver-class-name = net.sf.log4jdbc.sql.jdbcapi.DriverSpy
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=false

#hibernate config
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
logging.level.jpa=DEBUG
```

[src/main/resources/log4jdbc.log4j2.properties]

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.dump.sql.maxlinelength=0
```

[src/main/resources/logback.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{yyyyMMdd HH:mm:ss.SSS} [%thread] %-3level %logger{5} -
 %msg %n</pattern>
    </encoder>
  </appender>

  <logger name="jdbc" level="OFF"/>

  <logger name="jdbc.sqlonly" level="DEBUG" additivity="false">>
    <appender-ref ref="STDOUT" />
  </logger>

  <logger name="jdbc.sqltiming" level="INFO" additivity="false">>
    <appender-ref ref="STDOUT" />
  </logger>

  <logger name="jdbc.resultsettable" level="DEBUG" additivity="false">>
    <appender-ref ref="STDOUT" />
  </logger>

  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>

</configuration>
```

[NativesqlexamApplication.java]

```
package jpa;

import java.sql.Connection;

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

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

import com.querydsl.sql.Configuration;
import com.querydsl.sql.OracleTemplates;
import com.querydsl.sql.SQLQueryFactory;
import com.querydsl.sql.SQLTemplates;
import com.querydsl.sql.spring.SpringConnectionProvider;
import com.querydsl.sql.spring.SpringExceptionTranslator;

@SpringBootApplication
public class Nativesql2Application {

	@Autowired
	DataSource dataSource;

	public static void main(String[] args) {
		SpringApplication.run(Nativesql2Application.class, args);
	}

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

	@Bean
	public Configuration configuration() {
		SQLTemplates templates = OracleTemplates.builder().build();

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

	@Bean
```

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

[QuerydslRepository.java]

```
package jpa.repository;

import java.util.List;

import com.querydsl.core.Tuple;

public interface QuerydslRepository {
	//////////////////////////////////////////Querydsl 처리 메소드

	// MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
	List<Tuple> getEnameDnameTop5(String deptno);

	// WITH, JOIN, GROUPBY이용 부서명, 부서별 직원평균급여
	List<Tuple> getDnameAvgSal();

	// myemp1테이블에 insert, 오라클 시퀀스이용
	Long insertMyemp1(String ename, Long sal, String deptno);

	// myemp1에서 '1'번부서 또는 '2'번부서원에 대해 중복제거 후 직무출력
	List<String> getJobDistinct(String deptno1, String deptno2);

	// MYEMP1, MYEMP1_OLD 테이블 오라클 UNIONALL, WITH , COUNT
	Long getUnionCount();

	// 급여가 sal1~sal2 사이, 이름이 namePattern으로 시작하는 사원이름, 급여추출
	List<Tuple> getEnameSalBetweenLike(Long sal1, Long sal2, String namePattern);

	// EMP테이블에서 사원명, 관리자명 추출,NVL,SelfJoin,LeftJoin
	List<Tuple> getMgrNameLeftJoinNvl();

	// EMP테이블에서 사원명, 부서명 추출, DECODE, CASE
```

```
        List<Tuple> getEnameDnameDecode();

        // EMP 테이블에서 JOB의 급여합이 최대인 JOB과 그 평균 급여를 출력
        List<Tuple> getJobAvgSal();

        // EMP 테이블에서 KING부터 시작하여 계층적으로 출력하는 쿼리
        List<Tuple> getHQuery() throws Exception;

        // EMP 테이블에서 job이 'SALESMAN'인 사원추출,INLINE View
        List<Tuple> getEnameDnameInlieView();

        // EMP 테이블에서 이름,부서코드,부서명 추출,조인대신 스칼라서브쿼리사용
        List<Tuple> getEnameDnameScalar();

        // 사원이 한명이라도 있는 부서명 출력, Exists
        List<String> getDnameExists();

        // 사원 급여순위, 부서별 급여순위, RANK, DENSE_RANK
        List<Tuple> getRank();
}
```

[QuerydslRepositoryImpl.java]

```
package jpa.repository;

import static jpa.model.QDept.dept;
import static jpa.model.QEmp.emp;
import static jpa.model.QMydept1.mydept1;
import static jpa.model.QMyemp1.myemp1;
import static jpa.model.QMyemp1Old.myemp1Old;

import java.util.List;

import javax.sql.DataSource;
import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.querydsl.core.Tuple;
```

```
import com.querydsl.core.types.Expression;
import com.querydsl.core.types.Path;
import com.querydsl.core.types.dsl.CaseBuilder;
import com.querydsl.core.types.dsl.ComparableExpressionBase;
import com.querydsl.core.types.dsl.Expressions;
import com.querydsl.sql.Configuration;
import com.querydsl.sql.SQLExpressions;
import com.querydsl.sql.SQLQueryFactory;
import com.querydsl.sql.oracle.OracleQuery;

import jpa.model.QEmp;
import jpa.model.QMyemp1;

@Repository
@Transactional
public class QuerydslRepositoryImpl implements QuerydslRepository {

	@Autowired
	SQLQueryFactory queryFactory;

	@Autowired
	DataSource dataSource;

	@Autowired
	Configuration configuration;

	@Override
	public List<Tuple> getEnameDnameTop5(String deptno) {

		//////////////////////////////////////////////////////
		// MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
		// limit 메소드 또는 OracleGrammar의 ROWNUM을 이용
		// ---------------------------------------------------
		// select MYEMP1.ENAME, MYDEPT1.DNAME
		// from MYEMP1 MYEMP1 inner join MYDEPT1 MYDEPT1
		// on MYEMP1.DEPTNO = MYDEPT1.DEPTNO
		// where rownum < 6
		//////////////////////////////////////////////////////
//		List<Tuple> rows = queryFactory.select(myemp1.ename, mydept1.dname)
```

```
//                                  .from(myemp1).innerJoin(mydept1)
//                                  .on(myemp1.deptno.eq(mydept1.deptno)).limit(5).fetch();

                List<Tuple> rows = queryFactory.select(myemp1.ename, mydept1.dname)
                                  .from(myemp1).innerJoin(mydept1)
                                  .on(myemp1.deptno.eq(mydept1.deptno))
                                  .where(OracleGrammar.rownum.lt(6)).fetch();

                return rows;
        }

        @Override
        public List<Tuple> getDnameAvgSal() {
                ////////////////////////////////////////////////////
                // WITH, JOIN, GROUPBY이용 부서명, 부서별 직원평균급여
                // myemp1은 1000만건정도,JPA또는SQL구문을 직접실행시 모두 8초정도 소요
된다.
                // 아래 WITH구문이 정상동작하기 위해서는 Querydsl 4.0.9 사용필
                // 그렇지 않으면 inner join의 myemp2가 with문의 alias로 조인하지 않고
                // join myemp1 myemp2 형태로 되어 myemp1과 다시 조인하는 형태가 된다.
                // ------------------------------------------------
                //with myemp2 as (
                //                  select MYEMP1.DEPTNO, avg(MYEMP1.SAL) sal
                //                  from TEST.MYEMP1 MYEMP1
                //                  group by MYEMP1.DEPTNO)
                //                select MYDEPT1.DNAME, myemp2.SAL
                //                from MYDEPT1
                //                join MYEMP1 myemp2
                //                on MYDEPT1.DEPTNO = myemp2.DEPTNO
                ////////////////////////////////////////////////////
                QMyemp1 myemp2 = new QMyemp1("myemp2");
                List<Tuple> rows   = queryFactory.query()
                                  .with(myemp2,
                                                  SQLExpressions
                                                  .select(myemp1.deptno, myemp1.sal.avg().as("sal"))
                                                  .from(myemp1)
                                                  .groupBy(myemp1.deptno))
                .select(mydept1.dname, myemp2.sal)
                .from(mydept1)
```

```
                .innerJoin(myemp2).on(mydept1.deptno.eq(myemp2.deptno))
                .fetch();
                return rows;
        }

        @Override
        public Long insertMyemp1(String ename, Long sal, String deptno) {
                ////////////////////////////////////////////////////////////////
                // myemp1테이블에 insert, 오라클 시퀀스이용
                // 시퀀스생성:create sequence seq_myemp1_empno start with 10000003
                //              select seq_myemp1_empno.nextval from dual;
                // 시퀀스이름을 줄때 앞에스키마명까지 줘야한다. 아니면 ORA-02289 발생
                //--------------------------------------------------------
                // insert into myemp1 (empno, ename, sal, deptno)
                // values (test.seq_myemp1_empno, ?, ?, ?)
                ////////////////////////////////////////////////////////////////
                Long cnt = queryFactory.insert(myemp1)
                                .columns(myemp1.empno, myemp1.ename, myemp1.sal,
myemp1.deptno)

        .values(SQLExpressions.nextval("test.seq_myemp1_empno"),ename,sal,deptno)
                                .execute();


                return cnt;
        }

        @Override
        public List<String> getJobDistinct(String deptno1, String deptno2) {
                ////////////////////////////////////////////////////////////////
                // myemp1에서 '1'번부서 또는 '2'번부서원에 대해 중복제거 후 직무출력
                // DISTINCT, IN, OR 예문
                // JPA에서 실행하나 DB에서 직접쿼리하든지 8초정도소요(myemp1은 1000만
건)
                //--------------------------------------------------------
                // select distinct MYEMP1.JOB
                // from MYEMP1 MYEMP1
                // where MYEMP1.DEPTNO in ('1', '2')
                ////////////////////////////////////////////////////////////////
                List<String> rows = queryFactory.select(myemp1.job)
```

```
                .distinct().from(myemp1)
                .where(myemp1.deptno.in(deptno1, deptno2))

        //.where(myemp1.deptno.eq(deptno1).or(myemp1.deptno.eq(deptno2)))
                .fetch();

        return rows;
    }

    @Override
    public Long getUnionCount() {
        ////////////////////////////////////////////////////////////
        // MYEMP1, MYEMP1_OLD 테이블 오라클 UNIONALL, WITH , COUNT
        // JPA에서 실행하나 DB에서 직접쿼리하든지 0.6초정도소요
        // (myemp1은 1000만건, myemp1_old는 160만건, empno는 PK)
        //------------------------------------------------------------
        // with myemp2 as (
        //         (select MYEMP1_OLD.EMPNO empno from MYEMP1_OLD MYEMP1_OLD)
        //     union all
        //         (select MYEMP1.EMPNO empno from MYEMP1 MYEMP1)
        // )
        // select count(*)
        // from myemp2 myemp2
        ////////////////////////////////////////////////////////////
        QMyemp1 myemp2 = new QMyemp1("myemp2");
        @SuppressWarnings("unchecked")
        Long cnt = queryFactory.query()
                .with(myemp2,
                SQLExpressions.unionAll(

    SQLExpressions.select(myemp1Old.empno.as("empno")).from(myemp1Old),

    SQLExpressions.select(myemp1.empno.as("empno")).from(myemp1)
                ))
                //.select(myemp2.empno.count()), 아래와 동일, count를 위해선 fetchCount를 쓰자.
                .select(myemp2.empno)
                .from(myemp2)
```

```
                        .fetchCount();

                return cnt;
        }

        @Override
        public List<Tuple> getEnameSalBetweenLike(Long sal1, Long sal2, String namePattern) {
                ////////////////////////////////////////////////////////////
                // 급여가 sal1~sal2 사이, 이름이 namePattern으로 시작하는 사원이름, 급여추
출
                // JPA에서 0.4초 DB에서 직접쿼리하든지 0.3초 정도소요
                // (myemp1은 1000만건, ename, sal 칼럼은 각각 인덱스 생성되어 있음)
                //----------------------------------------------------------
                // select MYEMP1.ENAME, MYEMP1.SAL
                // from MYEMP1 MYEMP1
                // where MYEMP1.SAL between 5999990 and 6000000
                // and MYEMP1.ENAME like '가%' escape '₩'
                ////////////////////////////////////////////////////////////
                List<Tuple> rows = queryFactory.select(myemp1.ename, myemp1.sal)
                                .from(myemp1)
                                .where(myemp1.sal.between(sal1, sal2)
                                                .and(myemp1.ename.like(namePattern + "%")))
                                .fetch();

                return rows;
        }

        @Override
        public List<Tuple> getMgrNameLeftJoinNvl() {
                ////////////////////////////////////////////////////////////
                // EMP테이블에서 사원명, 관리자명 추출,NVL,SelfJoin,LeftJoin
                // 관리자(mgr)가 없는 경우 "최고관리자"로 관리자 이름출력
                // 관리자도 직원이므로 자기자신 테이블인 EMP와 조인한다.
                // 최고관리자는 mgr칼럼이 NULL, 모두출력되게 하려고 leftjoin 적용
                //----------------------------------------------------------
                // [원래 생각한 SQL]
                // select e1.ename, nvl(e2.ename,'최고관리자')
                // from emp e1 left join emp e2
                // on e1.mgr = e2.empno;
```

```
        //-------------------------------------------------------
        // [JPA로 만들었을 때 생성된 SQL]
        // select EMP.ENAME, coalesce(e.ENAME, '최고관리자')
        // from EMP EMP
        // left join EMP e
        // on EMP.MGR = e.EMPNO
        ////////////////////////////////////////////////////////////
        QEmp e = new QEmp("e");

        List<Tuple> rows = queryFactory
                        .select(emp.ename, e.ename.coalesce("최고관리자").as("ename"))

                        .from(emp)
                        .leftJoin(e).on(emp.mgr.eq(e.empno))
                        .fetch();

        return rows;
}

@Override
public List<Tuple> getEnameDnameDecode() {
        ////////////////////////////////////////////////////////////
        // EMP테이블에서 사원명, 부서명 추출, DECODE, CASE
        // JPA의 DECODE는 querydsl의 CaseBuilder를 사용하면된다.
        //-------------------------------------------------------
        // [원래 생각한 SQL]
        // select emp.ename, decode(emp.deptno,10,'ACCOUNTING',
// 20,'RESEARCH',
// 30,'SALES',
// 40,'OPERATIONS',
// 'UNKNOWN') dname
// from emp
        //-------------------------------------------------------
        // [JPA로 만들었을 때 생성된 SQL]
        // select EMP.ENAME, (case when EMP.DEPTNO = 10 then 'ACCOUNTING'
        //                         when EMP.DEPTNO = 20 then 'RESEARCH'
        //                         when EMP.DEPTNO = 30 then 'SALES'
    //                         when EMP.DEPTNO = 40 then 'OPERATIONS'
        //                         else 'UNKNOWN' end) dname
```

```
        // from EMP EMP
        ////////////////////////////////////////////////////////
        Expression<String> cases = new CaseBuilder()
                        .when(emp.deptno.eq((byte) 10)).then("ACCOUNTING")
                        .when(emp.deptno.eq((byte) 20)).then("RESEARCH")
                        .when(emp.deptno.eq((byte) 30)).then("SALES")
                        .when(emp.deptno.eq((byte) 40)).then("OPERATIONS")
                        .otherwise("UNKNOWN").as("dname");


        List<Tuple> rows = queryFactory
                        .select(emp.ename, cases)
                        .from(emp)
                        .fetch();

        return rows;
}

@Override
public List<Tuple> getJobAvgSal() {
        ////////////////////////////////////////////////////////
        // EMP 테이블에서 JOB의 급여합이 최대인 JOB과 그 평균 급여를 출력
        // 서브쿼리, GroupBy, Having 이용
        //-------------------------------------------------------
        // [처음만들려고 했던 SQL]
        // select job, avg(sal) from emp
        // group by job
        // having sum(sal)   = (select max(sum(sal)) from emp
        //                      group by job);
        // [JPA에서 생성된 SQL]
        // select EMP.JOB, round(avg(EMP.SAL)) sal
        // from EMP EMP
        // group by EMP.JOB
        // having sum(EMP.SAL) = (select max(sum(EMP.SAL))
        //                        from EMP EMP
        //                        group by EMP.JOB)
        ////////////////////////////////////////////////////////

        List<Tuple> rows = queryFactory
```

```
                                .select(emp.job,
                                        emp.sal.avg().round().as("sal"))
                                .from(emp)
                                .groupBy(emp.job)
                                .having(emp.sal.sum().eq(
                                                SQLExpressions
                                                .select(emp.sal.sum().max())
                                                .from(emp).groupBy(emp.job)))
                                .fetch();

                return rows;
        }

        @SuppressWarnings("unchecked")
        @Override
        public List<Tuple> getHQuery() throws Exception {
                ////////////////////////////////////////////////////////////
                // EMP 테이블에서 KING부터 시작하여 계층적으로 출력하는 쿼리
                // 오라클에서 계층형쿼리를 쓸려면 OracleQuery를 사용하고
                // LEVEL과 같은 의사 칼럼은 OracleGrammar를 쓰면 된다.
                // 물론 ROWID, ROWNUM, SYSDATE등도 OracleGrammar에서 지원한다.
                // [원래 만들고자 했던 오라클 계층쿼리구문]
                // select lpad(' ',(level-1)*2) || ename, sal
                //   from emp
                //   start with ename = 'KING'
                //   connect by prior empno = mgr
                //----------------------------------------------------------
                // [JPA, querydsl의 OracleQuery를 통해 생성된 SQL]
                // select lpad(' ',(level - 1) * 2,' ') || EMP.ENAME,
                //            EMP.SAL, EMP.DEPTNO
                // from EMP EMP
                // start with EMP.ENAME = 'KING'
                // connect by prior EMP.EMPNO = EMP.MGR
                ////////////////////////////////////////////////////////////
                @SuppressWarnings("rawtypes")
                OracleQuery query = new OracleQuery(dataSource.getConnection(),
configuration);

                List<Tuple> rows =  query
```

```
                .select(StringExpressions.lpad(
                        Expressions.stringTemplate("' '").stringValue(),
                        OracleGrammar.level.subtract(1).multiply(2), ' ')
                                .concat(emp.ename),
                                emp.sal, emp.deptno)
                .startWith(emp.ename.eq("KING"))
                .connectByPrior(emp.empno.eq(emp.mgr))
                .from(emp)
                .fetch();

                return rows;
        }

        @Override
        public List<Tuple> getEnameDnameInlieView() {
                ////////////////////////////////////////////////////
                // EMP 테이블에서 job이 'SALESMAN'인 사원추출,INLINE View
                //------------------------------------------------
                // [처음 생각한 SQL 쿼리]
                // select ename, dname
                // from   (select ename, deptno from emp
        //           where job = 'SALESMAN') e,
                //            dept d
        // where e.deptno = d.deptno;
                //------------------------------------------------
                // [Querydsl에서 생성한 SQL 코드]
                // select EMP.ENAME, DEPT.DNAME
                // from (select EMP.ENAME, EMP.DEPTNO
                //                 from EMP EMP
                //                 where EMP.JOB = 'SALESMAN') emp, DEPT DEPT
                //                 where EMP.DEPTNO = DEPT.DEPTNO
                ////////////////////////////////////////////////////
                List<Tuple> rows =   queryFactory.select(emp.ename, dept.dname)
                                .from(SQLExpressions
                                                .select(emp.ename, emp.deptno)
                                                .from(emp)
                                                .where(emp.job.eq("SALESMAN")).as("emp"),
                                                dept)
                                .where(emp.deptno.eq(dept.deptno))
```

```
                                .fetch();

        return rows;
    }

    @Override
    public List<Tuple> getEnameDnameScalar() {
        ////////////////////////////////////////////////////////////////
        // EMP 테이블에서 이름,부서코드,부서명 추출,조인대신 스칼라서브쿼리사용
        //--------------------------------------------------------------
        // [만들려고 했던 SQL 구문]
        // select ename, deptno,
    //    (select dname from dept
    //            where emp.deptno = dept.deptno)
    //    from emp;
        //--------------------------------------------------------------
        // [Querydsl에서 생성한 SQL 코드]
        // select EMP.ENAME, EMP.DEPTNO,
        //           (select DEPT.DNAME
        //            from DEPT DEPT
        //            where DEPT.DEPTNO = EMP.DEPTNO) dname
        // from EMP EMP
        ////////////////////////////////////////////////////////////////
        List<Tuple> rows =   queryFactory
                                .select(emp.ename, emp.deptno,
                                                SQLExpressions
                                                .select(dept.dname)
                                                .from(dept)

    .where(dept.deptno.eq(emp.deptno)).as(dept.dname))
                                .from(emp)
                                .fetch();

        return rows;
    }

    @Override
    public List<String> getDnameExists() {
        ////////////////////////////////////////////////////////
```

```
        // 사원이 한명이라도 있는 부서명 출력, Exists
        //-------------------------------------------------
        // [만들려고 했던 SQL 구문]
        // SELECT dname FROM dept
        // WHERE EXISTS (
        //              SELECT 1 FROM emp WHERE deptno = dept.deptno
        //              );
        //-------------------------------------------------
        // [Querydsl에서 생성한 SQL 코드]
        // select DEPT.DNAME
        // from DEPT DEPT
        // where exists (select 1
        //                 from EMP EMP
        //                 where EMP.DEPTNO = DEPT.DEPTNO)
        ////////////////////////////////////////////////////
        List<String> rows = queryFactory.select(dept.dname)
                        .from(dept)
                        .where(SQLExpressions
                                        .select(Expressions.constant(1))
                                        .from(emp)
                                        .where(emp.deptno.eq(dept.deptno)).exists())
                        .fetch();

        return rows;
}

@SuppressWarnings("unchecked")
@Override
public List<Tuple> getRank() {
        ////////////////////////////////////////////////////
        // 사원 급여순위, 부서별 급여순위, RANK, DENSE_RANK
        // DENSE_RANK는 1등이 몇명이더라도 다음순위는 2등
        //-------------------------------------------------
        // [만들려고 했던 SQL 구문]
        // SELECT ENAME, SAL, DEPTNO,
    //    RANK() OVER (ORDER BY SAL DESC) EMP_RANK,
    //    DENSE_RANK() OVER (PARTITION BY DEPTNO
    //                    ORDER BY SAL DESC) DEPT_RANK
    // FROM EMP
```

```
                // ORDER BY EMP_RANK, DEPT_RANK;
                    //-------------------------------------------------
                    // [Querydsl에서 생성한 SQL 코드]
                    // select EMP.ENAME, EMP.SAL, EMP.DEPTNO,
                    //         rank() over (order by EMP.SAL desc) emp_rank,
                    //         dense_rank() over
                    //             (partition by EMP.DEPTNO order by EMP.SAL desc) dept_rank
                    // from EMP EMP
                    // order by emp_rank asc, dept_rank asc
                    ////////////////////////////////////////////////////
                    Path<Long> emp_rank = Expressions.numberPath(Long.class, "emp_rank");
                    Path<Long> dept_rank = Expressions.numberPath(Long.class, "dept_rank");

                    List<Tuple> rows = queryFactory
                                    .select(emp.ename, emp.sal, emp.deptno,
                                                    SQLExpressions
                                                    .rank().over()

                                                    .orderBy(emp.sal.desc()).as(emp_rank),
                                                    SQLExpressions
                                                    .denseRank().over()
                                                    .partitionBy(emp.deptno)
                                                    .orderBy(emp.sal.desc()).as(dept_rank)
                                    )
                    .from(emp)
                    .orderBy(((ComparableExpressionBase<Long>)                    emp_rank).asc(),
((ComparableExpressionBase<Long>) dept_rank).asc())
                    .fetch();

                    return rows;
            }
}
```

[EmpService.java]

```
package jpa.service;

import java.util.List;

import com.querydsl.core.Tuple;
```

```
import jpa.model.Emp;

public interface EmpService {
        // MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
        List<Tuple> getEnameDnameTop5(String deptno);

        // WITH, JOIN, GROUPBY이용 부서명, 부서별 직원평균급여
        List<Tuple> getDnameAvgSal();

        // myemp1테이블에 insert, 오라클 시퀀스이용
        Long insertMyemp1(String ename, Long sal, String deptno);

        // myemp1에서 '1'번부서 또는 '2'번부서원에 대해 중복제거 후 직무출력
        List<String> getJobDistinct(String deptno1, String deptno2);

        // MYEMP1, MYEMP1_OLD 테이블 오라클 UNIONALL, WITH , COUNT
        Long getUnionCount();

        // 급여가 sal1~sal2 사이, 이름이 namePattern으로 시작하는 사원이름, 급여추출
        List<Tuple> getEnameSalBetweenLike(Long sal1, Long sal2, String namePattern);

        // EMP테이블에서 사원명, 관리자명 추출,NVL,SelfJoin,LeftJoin
        List<Tuple> getMgrNameLeftJoinNvl();

        // EMP테이블에서 사원명, 부서명 추출, DECODE, CASE
        List<Tuple> getEnameDnameDecode();

        // EMP 테이블에서 JOB의 급여합이 최대인 JOB과 그 평균 급여를 출력
        List<Tuple> getJobAvgSal();

        // EMP 테이블에서 KING부터 시작하여 계층적으로 출력하는 쿼리
        List<Tuple> getHQuery() throws Exception;

        // EMP 테이블에서 job이 'SALESMAN'인 사원추출,INLINE View
        List<Tuple> getEnameDnameInlieView();

        // EMP 테이블에서 이름,부서코드,부서명 추출,조인대신 스칼라서브쿼리사용
        List<Tuple> getEnameDnameScalar();
```

```
	// 사원이 한명이라도 있는 부서명 출력, Exists
	List<String> getDnameExists();

	// 사원 급여순위, 부서별 급여순위, RANK, DENSE_RANK
	List<Tuple> getRank();
}
```

[EmpServiceImpl.java]

```
package jpa.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.querydsl.core.Tuple;

import jpa.repository.QuerydslRepository;

@Service
public class EmpServiceImpl implements EmpService {

	@Autowired
	QuerydslRepository empRepository;

	@Override
	// MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
	public List<Tuple> getEnameDnameTop5(String deptno) {
		return empRepository.getEnameDnameTop5(deptno);
	}

	@Override
	// WITH, JOIN, GROUPBY이용 부서명, 부서별 직원평균급여
	public List<Tuple> getDnameAvgSal() {
		return empRepository.getDnameAvgSal();
	}

	@Override
```

```
// myemp1테이블에 insert, 오라클 시퀀스이용
public Long insertMyemp1(String ename, Long sal, String deptno) {
        return empRepository.insertMyemp1(ename, sal, deptno);
}

@Override
// myemp1에서 '1'번부서 또는 '2'번부서원에 대해 중복제거 후 직무출력
public List<String> getJobDistinct(String deptno1, String deptno2) {
        return empRepository.getJobDistinct(deptno1, deptno2);
}

@Override
// MYEMP1, MYEMP1_OLD 테이블 오라클 UNIONALL, WITH , COUNT
public Long getUnionCount() {
        return empRepository.getUnionCount();
}

@Override
// 급여가 sal1~sal2 사이, 이름이 namePattern으로 시작하는 사원이름, 급여추출
public List<Tuple> getEnameSalBetweenLike(Long sal1, Long sal2, String namePattern) {
        return empRepository.getEnameSalBetweenLike(sal1, sal2, namePattern);
}

@Override
// EMP테이블에서 사원명, 관리자명 추출,NVL,SelfJoin,LeftJoin
public List<Tuple> getMgrNameLeftJoinNvl() {
        return empRepository.getMgrNameLeftJoinNvl();
}

@Override
// EMP테이블에서 사원명, 부서명 추출, DECODE, CASE
public List<Tuple> getEnameDnameDecode() {
        return empRepository.getEnameDnameDecode();
}

@Override
// EMP 테이블에서 JOB의 급여합이 최대인 JOB과 그 평균 급여를 출력
public List<Tuple> getJobAvgSal() {
        return empRepository.getJobAvgSal();
```

```
	}

	@Override
	// EMP 테이블에서 KING부터 시작하여 계층적으로 출력하는 쿼리
	public List<Tuple> getHQuery() throws Exception {
		return empRepository.getHQuery();
	}

	@Override
	// EMP 테이블에서 job이 'SALESMAN'인 사원추출,INLINE View
	public List<Tuple> getEnameDnameInlieView() {
		return empRepository.getEnameDnameInlieView();
	}

	@Override
	// EMP 테이블에서 이름,부서코드,부서명 추출,조인대신 스칼라서브쿼리사용
	public List<Tuple> getEnameDnameScalar() {
		return empRepository.getEnameDnameScalar();
	}

	@Override
	// 사원이 한명이라도 있는 부서명 출력, Exists
	public List<String> getDnameExists() {
		return empRepository.getDnameExists();
	}

	@Override
	// 사원 급여순위, 부서별 급여순위, RANK, DENSE_RANK
	public List<Tuple> getRank() {
		return empRepository.getRank();
	}
}
```

[EmpController.java]

```
package jpa.controller;

import static jpa.model.QDept.dept;
import static jpa.model.QEmp.emp;
import static jpa.model.QMydept1.mydept1;
```

```
import static jpa.model.QMyemp1.myemp1;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.util.StopWatch;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.querydsl.core.Tuple;

import jpa.model.QMyemp1;
import jpa.service.EmpService;
import lombok.extern.slf4j.Slf4j;

@RestController
@RequestMapping("/emp")
@Slf4j
public class EmpController {

	@Autowired
	private EmpService empService;

	StopWatch stopWatch = new StopWatch();

	// ------------------------------------------------------------
	// MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
	// deptno 칼럼은 문자1자리, MYEMP1, MYDEPT1의 deptno칼럼은 인덱스 생성됨
	// localhost:8080/emp/get/1 <- 1번부서
	// ------------------------------------------------------------
	@RequestMapping("/get/{deptno}")
	public Map<String, String> getEnameDnameTop5(String deptno) {
		Map<String, String> m = new HashMap<String, String>();
```

```
                stopWatch.start();
                List<Tuple> emps = empService.getEnameDnameTop5(deptno);
                stopWatch.stop();
                log.info(">>>>>  getEnameDnameTop5(Time)  :  " +
stopWatch.getTotalTimeSeconds());

                for (Tuple row : emps) {
                        m.put(row.get(myemp1.ename), row.get(mydept1.dname));
                }

                return m;
        }

        // ------------------------------------------------------------
        // MYEMP1, MYDEPT1을 DEPTNO로 조인하여 5건 추출
        // deptno 칼럼은 문자1자리, MYEMP1, MYDEPT1의 deptno칼럼은 인덱스 생성됨
        // localhost:8080/emp/get/1 <- 1번부서
        // ------------------------------------------------------------
        @RequestMapping("/getwith")
        public Map<String, String> getDnameAvgSal(String deptno) {
                Map<String, String> m = new HashMap<String, String>();
                QMyemp1 myemp2 = new QMyemp1("myemp2");

                stopWatch.start();
                List<Tuple> rows = empService.getDnameAvgSal();
                stopWatch.stop();
                log.info(">>>>> getDnameAvgSal(Time) : " + stopWatch.getTotalTimeSeconds());

                for (Tuple row : rows) {
                        m.put(row.get(mydept1.dname), row.get(myemp2.sal).toString());
                }

                return m;
        }

        // ------------------------------------------------------------
        // MYEMP1에 INSERT(오라클 시퀀스 이용)
        // localhost:8080/emp/insert/오제이씨/9999/1
        // ------------------------------------------------------------
```

```
@RequestMapping("/insert/{ename}/{sal}/{deptno}")
public String insertMyemp1(@PathVariable String ename,
                           @PathVariable Long sal,
                           @PathVariable String deptno) {
        Long cnt = empService.insertMyemp1(ename, sal, deptno);

        if (cnt > 0) return cnt + "건 입력완료!";
        else return "입력실패!";
}

// ------------------------------------------------------------
// MYEMP1에 INSERT(오라클 시퀀스 이용)
// localhost:8080/emp/getjob/1/2
// ------------------------------------------------------------
@RequestMapping("/getjob/{deptno1}/{deptno2}")
public List<String> getJobDistinct(@PathVariable String deptno1,
                                   @PathVariable String deptno2) {
        stopWatch.start();
        List<String> rows = empService.getJobDistinct(deptno1, deptno2);
        stopWatch.stop();
        log.info(">>>>> getJobDistinct(Time) : " + stopWatch.getTotalTimeSeconds());

        return rows;
}

// ------------------------------------------------------------
// MYEMP1, MYEMP1_OLD 테이블 오라클 UNIONALL, WITH , COUNT
// localhost:8080/emp/getunioncount
// ------------------------------------------------------------
@RequestMapping("/getunioncnt")
public Long getUnionCount() {
        stopWatch.start();
        Long cnt = empService.getUnionCount();
        stopWatch.stop();
        log.info(">>>>> getUnionCount(Time) : " + stopWatch.getTotalTimeSeconds());

        return cnt;
}
```

```
        //---------------------------------------------------------------
        // 급여가 sal1~sal2 사이, 이름이 namePattern으로 시작하는 사원이름, 급여추출
        // http://localhost:8080/emp/getnamesal/5999990/6000000/가
        //---------------------------------------------------------------
        @RequestMapping("/getnamesal/{sal1}/{sal2}/{namePattern}")
        public Map<String, String> getEnameSalBetweenLike(@PathVariable Long sal1,
                                                           @PathVariable Long sal2,
                                                           @PathVariable String namePattern)
{
                Map<String, String> m = new HashMap<String, String>();

                stopWatch.start();
                List<Tuple> rows = empService.getEnameSalBetweenLike(sal1, sal2,
namePattern);
                stopWatch.stop();
                log.info(">>>>>  getEnameSalBetweenLike(Time) : " +
stopWatch.getTotalTimeSeconds());

                for (Tuple row : rows) {
                        m.put(row.get(myemp1.ename), row.get(myemp1.sal).toString());
                }

                return m;
        }

        //---------------------------------------------------------
        // EMP테이블에서 사원명, 관리자명 추출,NVL,SelfJoin,LeftJoin
        // localhost:8080/emp/getmgr
        //---------------------------------------------------------
        @RequestMapping("/getmgr")
        public Map<String, String> getMgrNameLeftJoinNvl() {
                Map<String, String> m = new HashMap<String, String>();

                stopWatch.start();
                List<Tuple> rows = empService.getMgrNameLeftJoinNvl();
                stopWatch.stop();
                log.info(">>>>>  getMgrNameLeftJoinNvl(Time) : " +
stopWatch.getTotalTimeSeconds());
```

```
		for (Tuple row : rows) {
			m.put(row.get(emp.ename), row.get(1, String.class));
		}

		return m;
	}

	//-------------------------------------------------------
	// EMP테이블에서 사원명, 부서명 추출, DECODE, CASE
	// localhost:8080/emp/getenamedname
	//-------------------------------------------------------
	@RequestMapping("/getenamedname")
	public Map<String, String> getEnameDnameDecode() {
		Map<String, String> m = new HashMap<String, String>();

		stopWatch.start();
		List<Tuple> rows = empService.getEnameDnameDecode();
		stopWatch.stop();
		log.info(">>>>>	getEnameDnameDecode(Time)	:	"	+
stopWatch.getTotalTimeSeconds());

		for (Tuple row : rows) {
			m.put(row.get(emp.ename), row.get(1, String.class));
		}

		return m;
	}

	//-------------------------------------------------------
	// EMP 테이블에서 JOB의 급여합이 최대인 JOB과 그 평균 급여를 출력
	// localhost:8080/emp/getjobavgsal
	//-------------------------------------------------------
	@RequestMapping("/getjobavgsal")
	public Map<String, Double> getJobAvgSal() {
		Map<String, Double> m = new HashMap<String, Double>();

		stopWatch.start();
		List<Tuple> rows = empService.getJobAvgSal();
		stopWatch.stop();
```

```
		log.info(">>>>> getJobAvgSal(Time) : " + stopWatch.getTotalTimeSeconds());

		for (Tuple row : rows) {
			m.put(row.get(emp.job), row.get(1, Double.class));
		}

		return m;
	}

	//-------------------------------------------------------
	// EMP 테이블에서 KING부터 시작하여 계층적으로 출력하는 쿼리
	// localhost:8080/emp/gethquery
	//-------------------------------------------------------
	@RequestMapping("/gethquery")
	public Map<String, String> getHQuery() throws Exception {

		Map<String, String> m = new HashMap<String, String>();

		stopWatch.start();
		List<Tuple> rows = empService.getHQuery();
		stopWatch.stop();
		log.info(">>>>> getHQuery(Time) : " + stopWatch.getTotalTimeSeconds());

		for (Tuple row : rows) {
			m.put(row.get(0, String.class), row.get(1, BigDecimal.class).toString());
		}

		return m;
	}

	//-------------------------------------------------------
	// EMP 테이블에서 job이 'SALESMAN'인 사원추출,INLINE View
	// localhost:8080/emp/getinline
	//-------------------------------------------------------
	@RequestMapping("/getinline")
	public Map<String, String> getEnameDnameInlieView() {
		Map<String, String> m = new HashMap<String, String>();

		stopWatch.start();
```

```
                List<Tuple> rows = empService.getEnameDnameInlieView();
                stopWatch.stop();
                log.info(">>>>>  getEnameDnameInlieView(Time)  :  " +
stopWatch.getTotalTimeSeconds());

                for (Tuple row : rows) {
                        m.put(row.get(0, String.class), row.get(1, String.class));
                }

                return m;
        }

        //-------------------------------------------------------
        // EMP 테이블에서 이름,부서코드,부서명 추출,조인대신 스칼라서브쿼리사용
        // localhost:8080/emp/getscalar
        //-------------------------------------------------------
        @RequestMapping("/getscalar")
        public List<String> getEnameDnameScalar() {
        List<String> list = new ArrayList<String>();;

                stopWatch.start();
                List<Tuple> rows = empService.getEnameDnameScalar();
                stopWatch.stop();
                log.info(">>>>>  getEnameDnameScalar(Time)  :  " +
stopWatch.getTotalTimeSeconds());

                for (Tuple row : rows) {
                        list.add(row.get(emp.ename));
                        list.add(row.get(emp.deptno).toString());
                        list.add(row.get(dept.dname));
                }

                return list;
        }

        //-------------------------------------------------------
        // 사원이 한명이라도 있는 부서명 출력, Exists
        // localhost:8080/emp/getexists
        //-------------------------------------------------------
```

```
	@RequestMapping("/getexists")
	public List<String> getDnameExists() {

		stopWatch.start();
		List<String> rows = empService.getDnameExists();
		stopWatch.stop();
		log.info(">>>>> getDnameExists(Time) : " + stopWatch.getTotalTimeSeconds());

		return rows;
	}

	//-------------------------------------------------------
	// 사원 급여순위, 부서별 급여순위, RANK, DENSE_RANK
	// localhost:8080/emp/getrank
	//-------------------------------------------------------
	@RequestMapping("/getrank")
	public List<Object[]> getRank() {
		List<Object[]> list = new ArrayList<Object[]>();
		stopWatch.start();
		List<Tuple> rows = empService.getRank();
		stopWatch.stop();
		log.info(">>>>> getRank(Time) : " + stopWatch.getTotalTimeSeconds());

		for (Tuple row : rows) {
			Object[] o = new Object[] {
				row.get(emp.ename),
				row.get(emp.sal),
				row.get(emp.deptno),
				row.get(3, String.class),
				row.get(4, String.class)
			};

			list.add(o);
		}

		return list;
	}
}
```