Summary

This documents provides a how to create and test a Test Case for database related unit test.

Description

Here are considerations when creating unit test case of developing DAO (Data Access Object) classes.

- Database Connection
- Table Creation and Initial Data Input
- Test Data Management required to run DAO class
- Transaction Rollback or Commit after DAO Test

If database isn't ready yet, dbms such as hsqldb, derby, or mysql is installed temporarily to developer's local.
Even the database is ready, you should write table creation script and connect to dbms to check data after commit.
Moreover, duplicated test during program update, also requires another connection to dbms to do tasks such as rollback.

Based on dbunit / unitils / spring-test, combined with several tips, developing test case for database related things can be much easier.
Refer to following Usage section for

Environmental settings

Identical to Unit Test Preferences.

Manual

Connect Database

Let's look into database preparation (Create DataSource).
If there's existing DB for testing, and you have the access to the DB, then there's no time to think twice.
Otherwise, you can use apache dbcp datasource to create database, and using springframework will make this task much easier.
In fact, it is recommended to have exclusive DBMS for testing if you want to repetitive test using CI server provided by eGovFramework. However, clean rollback is always an option for the cases where exclusive DBMS is not available.

Property Setup

```
# Properties for the PropertiesDataSourceFactory
database.driverClassName=org.hsqldb.jdbcDriver
database.url=jdbc:hsqldb:sampledb
database.userName=sa
database.password=
```

Create Test Case

When creating Test Case, declare @RunWith(UnitilsJUnit4TestClassRunner.class).

Unlike springframework Unitils use TestDataSource instead of DataSource.
This TestDataSource not only can get the DataSource as soon as declared, also can perform tasks such as dbmaintain at the sametime.
Let's leave details for the next time, right now let's try to understand following sample using unitils to get TestDataSource.

```
@RunWith(UnitilsJUnit4TestClassRunner.class)
public class DataSourceGetTest_unitilsDataSource {

  /*
   * based on database access info configured in unitils.properties
   * it creates test DataSource, then automatically injects it.
   * (unitils.properties file's name and location cannot be modified.)
   *
   *                              @unitils.properties

   */
  @TestDataSource
  private DataSource dataSource;

  @Test
  public void checkTestDataSource() {
    assertNotNull("Check dataSource is properly retrieved.", dataSource);
  }
}
```

Creating Table and Enter Initial Data

Using Unitils allows creating TestDataSource as shown above, or create DataSource with spring+unitils combination and generate table to test with initial data injected at the same time. This feature can be used in following orders.

- Write DML script to create required tables for unit testing
- Set executability of this script, and specify directory.
- Invoke specified feature when performing test case.

Currently supported DBMS are 'oracle', 'db2', 'mysql', 'hsqldb' and 'postgresql'.

Property Setup

Setup DBMS type, Scheme Information

```
# This property specifies the underlying DBMS implementation. Supported values are 'oracle', 'db2', 'mysql', 'hsqldb' and 'postgresql'.
# The value of this property defines which vendor specific implementations of DbSupport and ConstraintsDisabler are chosen.
database.dialect=hsqldb

# A comma-separated list of all used database schemas. The first schema name is the default one, if no schema name is
# specified in for example a dbunit data set, this default one is used.
# A schema name is case sensitive.
database.schemaNames=PUBLIC

# Type of transaction manager that should be created:
# simple: a simple transaction manager that wraps the datasource to control transactions
# spring: a transaction manager that delegates actions to the transaction manager that is configured in the current spring context
# auto: this will first try to load the spring transaction manager. if spring is not available, it will load the simple transaction manager
transactionManager.type=auto
```

Configure auto-update of Database Schema Information.

Specify a directory that contains sql statements to use.
dbMaintainer.disableConstraints.enabled disables constraints on the execution of unit test. Set true if you want to continue the process without any blockades

```
# If set to true, the DBMaintainer will be used to update the unit test database schema. This is done once for each
# test run, when creating the DataSource that provides access to the unit test database.
updateDataBaseSchema.enabled=true

# Comma separated list of directories and files in which the database update scripts are located. Directories in this
# list are recursively searched for files.
dbMaintainer.script.locations=src/test/resources/META-INF/persistence/maintenance/hsqldb

# If set to true, an implementation of org.unitils.dbmaintainer.constraints.ConstraintsDisabler will be used to disable
# the foreign key and not null constraints of the unit test database schema.
# The ConstraintsDisabler is configured using the properties specified below. The property with key 'database.dialect'
# specifies which implementation is used.
dbMaintainer.disableConstraints.enabled=true
```

Manage test data required to run DAO class

This section describes how to auto-insert data and clear data after test run when running CRUD unit test on database

DBUnit provides following useful features.

- Auto-Insert data on unit test execution if required data is pre-declared in XML.
- Auto-Verify database test result with expected data set in pre-defined XML.
- Specify transaction commit/rollback after test run

Downside of using DBUnit in the test code is it requires XML file loading process in the code.
Unitils provides a solution to this by using Annotation. Following section describes how to use it.

Configure properties

Configure transaction handling using DBUnit

This part specifies whether to perform commit/rollback when data is saved or deleted using DBUnit.
Specify disabled since you can dynamically change it when writing Test Cases.
(When performing periodic tests using CI, it is recommended to be set as rollback.)

```
# Default behavior concerning execution of tests in a transaction. Supported values are 'disabled', 'commit' and 'rollback'.
# If set to disabled, test are not executed in a transaction by default. If set to commit, each test is run in a transaction,
#Which is committed. If set to rollback, each test is run in a transaction, which is rolled back.
DatabaseModule.Transactional.value.default=disabled
```

Specify DBMS type, schema information

Unitils support DBMS as listed in the comment. Use of unsupported DBMS with unitils is discourages since lack of full support of features

```
# This property specifies the underlying DBMS implementation. Supported values are 'oracle', 'db2', 'mysql', 'hsqldb' and 'postgresql'.
# The value of this property defines which vendor specific implementations of DbSupport and ConstraintsDisabler are chosen.
database.dialect=hsqldb

# A comma-separated list of all used database schemas. The first schema name is the default one, if no schema name is
# specified in for example a dbunit data set, this default one is used.
# A schema name is case sensitive.
database.schemaNames=PUBLIC

# Type of transaction manager that should be created:
# simple: a simple transaction manager that wraps the datasource to control transactions
# spring: a transaction manager that delegates actions to the transaction manager that is configured in the current spring context
# auto: this will first try to load the spring transaction manager. if spring is not available, it will load the simple transaction manager
transactionManager.type=auto
```

Configure auto-update database schema.

Specify location where sql statements are stored.
dbMaintainer.disableConstraints.enabled dbMaintainer.disableConstraints.enabled disables constraints on the execution of unit test. Set true if you want to continue the process without any blockades

```
# If set to true, the DBMaintainer will be used to update the unit test database schema. This is done once for each
# test run, when creating the DataSource that provides access to the unit test database.
updateDataBaseSchema.enabled=true

# Comma separated list of directories and files in which the database update scripts are located. Directories in this
# list are recursively searched for files.
dbMaintainer.script.locations=src/test/resources/META-INF/persistence/maintenance/hsqldb

# If set to true, an implementation of org.unitils.dbmaintainer.constraints.ConstraintsDisabler will be used to disable
# the foreign key and not null constraints of the unit test database schema.
# The ConstraintsDisabler is configured using the properties specified below. The property with key 'database.dialect'
# specifies which implementation is used.
dbMaintainer.disableConstraints.enabled=true
```

Definition of data to auto-insert before test run

Invoke @DataSet (file path) at the declaration part of test class or test method..
Following snippet shows how to write data in XML.

```
<dataset>
      < table-name column-name1 ="value" column-name                    ...... />
      2="value"
</dataset>
```

Define expected data set to auto-verify test result.

Invoke @ExpectedDataSet (file path) at the declaration part of test class or test method.
You can create it in XML format as explained previously.

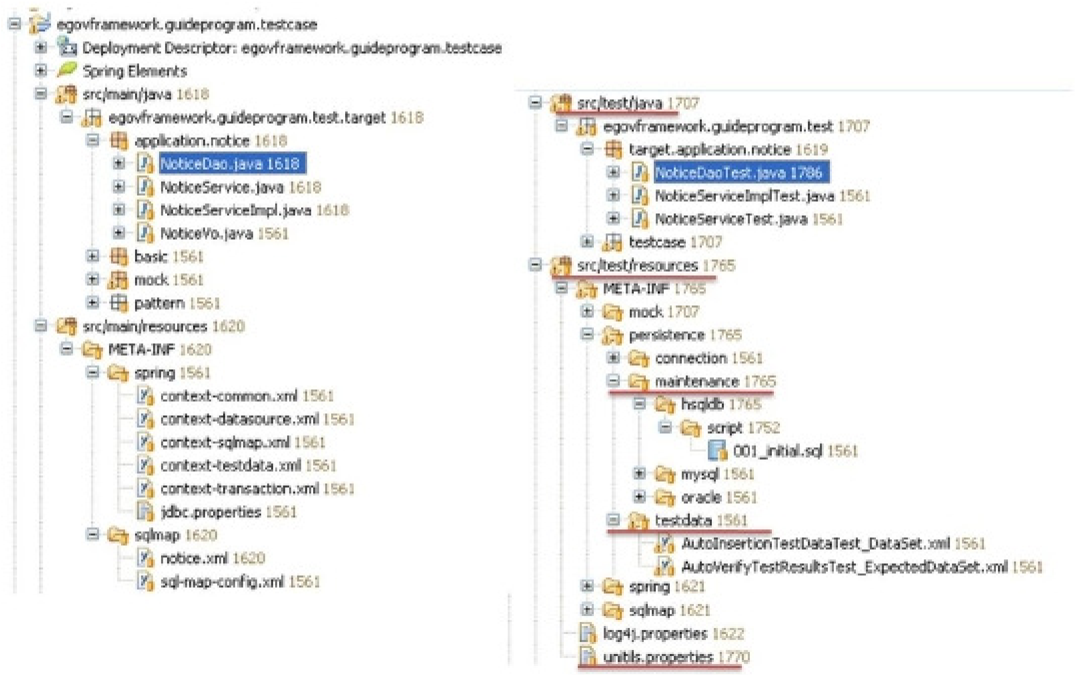
```
<dataset>
      <table-name column-name1 ="value" column-name                    ...... />
      2="value"
</dataset>
```

Handling transaction rollback/commit after testing DAO

Though you can configure settings for the whole Test Case, it is recommended to configure settings by declaring in program. Also, if you use springframework, you can configure txManager of springframework at Test Case class declaration part. See sample program for details.

Samples

Following figure shows a sample of Eclipse Maven project.



```
egovframework.guideprogram.testcase
  Deployment Descriptor: egovframework.guideprogram.testcase
  Spring Elements
  src/main/java 1618
    egovframework.guideprogram.test.target 1618
      application.notice 1618
        NoticeDao.java 1618
        NoticeService.java 1618
        NoticeServiceImpl.java 1618
        NoticeVo.java 1561
      basic 1561
      mock 1561
      pattern 1561
  src/main/resources 1620
    META-INF 1620
      spring 1561
        context-common.xml 1561
        context-datasource.xml 1561
        context-sqlmap.xml 1561
        context-testdata.xml 1561
        context-transaction.xml 1561
        jdbc.properties 1561
      sqlmap 1620
        notice.xml 1620
        sql-map-config.xml 1561

  src/test/java 1707
    egovframework.guideprogram.test 1707
      target.application.notice 1619
        NoticeDaoTest.java 1786
        NoticeServiceImplTest.java 1561
        NoticeServiceTest.java 1561
      testcase 1707
  src/test/resources 1765
    META-INF 1765
      mock 1707
      persistence 1765
        connection 1561
        maintenance 1765
          hsqldb 1765
            script 1752
              001_initial.sql 1561
          mysql 1561
          oracle 1561
        testdata 1561
          AutoInsertionTestDataTest_DataSet.xml 1561
          AutoVerifyTestResultsTest_ExpectedDataSet.xml 1561
      spring 1621
      sqlmap 1621
    log4j.properties 1622
    unitils.properties 1770
```

Sample includes following content.

- Test Target
- Test Scenario
- Configuration Properties
- Test Program
- Test Data
- Test Run Result

Following section describes each bullet points.

Test Target

- DAO Implementation Class : NoticeDao.java
- DAO Reference Class : NoticeVo.java

Test Scenario

1. Create table (Update when modified) to perform Notice tasks using HSQLDB.
2. Insert 3 sets of initial data from external XML file to run selectList, selectCount and etc.
3. Create noviceVo object, base data, to perform insert / update / delete method before running test.
4. Verify select result with expected data set in external XML file.
5. Test if duplicate insert leads to expected Exception.

You can manage various unitils related files with unitils.properties file by specifying the location. One configuration setting is applied at a time.

Configuration Settings

- ~~unitils.properties~~
- ~~unitils-local-hsqldb.properties~~

Test Data

- SQL in DML to auto-create table ~~: 001_initial.sql~~
- Test data in XML to auto-insert data : AutoInsertionTestDataTest_DataSet.xml
- Expected data set in XML to auto-ve~~rify test result : AutoVerifyTestResultsTest_Expected~~DataSet.xml

Test Program

Unit test program for NoticeDao.java : NoticeDaoTest.java

Test Run Result

Console Log

Data Source Creation

```
2009. 4. 28 오 후 2:29:33 org.unitils.database.config.PropertiesDataSourceFactory createDataSource
정 보: Creating data source. Driver: org.hsqldb.jdbcDriver, url: jdbc:hsqldb:sampledb, user: sa, password: <not shown>
```

Database Schema Update

```
2009.   4. 28 오 후 2:29:33 org.unitils.database.DatabaseModule updateDatabase
정 보:  Checking if database has to be updated.
2009.   4. 28 오 후 2:29:34 org.unitils.dbmaintainer.DBMaintainer updateDatabase
정 보:  Database is up to date
```

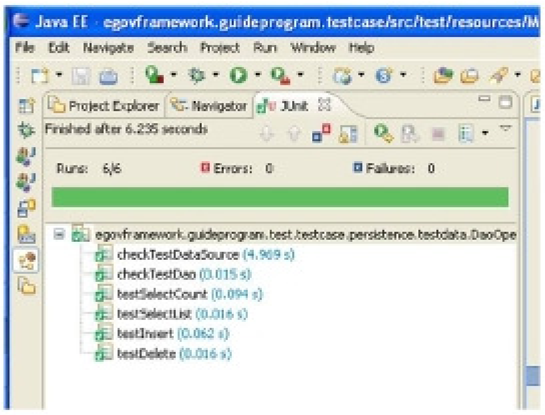
Spring Context Loading

```
2009.   4. 28 오 후 2:29:34 org.springframework.context.support.AbstractApplicationContext prepareRefresh
정 보:  Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@17a29a1: display name [org.springframework.context.support.ClassPathXmlApplicationContext@17a29a1]; sta
2009.   4. 28 오 후 2:29:35 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
정 보:  Loading XML bean definitions from class path resource [META-INF/persistence/connection/datasource-spring-with-unitils.xml]
2009.   4. 28 오 후 2:29:35 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
정 보:  Loading XML bean definitions from class path resource [META-INF/spring/context-common.xml]
2009.   4. 28 오 후 2:29:37 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
정 보:  Loading XML bean definitions from class path resource [META-INF/spring/context-sqlmap.xml]
2009.   4. 28 오 후 2:29:37 org.springframework.context.support.AbstractApplicationContext obtainFreshBeanFactory
정 보:  Bean factory for application context [org.springframework.context.support.ClassPathXmlApplicationContext@17a29a1]: org.springframework.beans.factory.support.DefaultListableBeanFacto
2009.   4. 28 오 후 2:29:37 org.springframework.core.io.support.PropertiesLoaderSupport loadProperties
정 보:  Loading properties file from class path resource [META-INF/spring/jdbc.properties]
2009.   4. 28 오 후 2:29:37 org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
정 보:  Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1a7508a: defining beans [dataSource,org.springframework.beans.factory.config.Pro
```

Reading Input DataSet from XML. Previous sample declared above class, it’s called every time method is invoked.

```
2009.   4. 28 오 후 2:29:37 org.unitils.dbunit.DbUnitModule getDataSet
정 보:  Loading DbUnit data set. File names: [D:\_dev\egov_dev\workspace\egovframework.guideprogram.testcase\target\test-classes\META-INF\persistence\testdata\AutoInsertionTestDataTest_Data
2009.   4. 28 오 후 2:29:38 org.unitils.dbunit.DbUnitModule getDataSet
정 보:  Loading DbUnit data set. File names: [D:\_dev\egov_dev\workspace\egovframework.guideprogram.testcase\target\test-classes\META-INF\persistence\testdata\AutoInsertionTestDataTest_Data
2009.   4. 28 오 후 2:29:38 org.unitils.dbunit.DbUnitModule getDataSet
정 보:  Loading DbUnit data set. File names: [D:\_dev\egov_dev\workspace\egovframework.guideprogram.testcase\target\test-classes\META-INF\persistence\testdata\AutoInsertionTestDataTest_Data
2009.   4. 28 오 후 2:29:38 org.unitils.dbunit.DbUnitModule getDataSet
정 보:  Loading DbUnit data set. File names: [D:\_dev\egov_dev\workspace\egovframework.guideprogram.testcase\target\test-classes\META-INF\persistence\testdata\AutoVerifyTestResultsTest_Expe
2009.   4. 28 오 후 2:29:38 org.unitils.dbunit.DbUnitModule getDataSet
정 보:
```

JUnit View


Java EE - egovframework.guideprogram.testcase/src/test/resources/M
File Edit Navigate Search Project Run Window Help
Project Explorer
Navigator
JUnit

```
Finished after 6.235 seconds

Runs: 6/6    Errors: 0    Failures: 0

egovframework.guideprogram.test.testcase.persistence.testdata.DaoOpe
    checkTestDataSource (4.909 s)
    checkTestDao (0.015 s)
    testSelectCount (0.094 s)
    testSelectList (0.016 s)
    testInsert (0.062 s)
    testDelete (0.016 s)
```

References

http://www.junit.org/ [http://www.junit.org/]
http://www.dbunit.org/ [http://www.dbunit.org/]