**About Siva Reddy**

Katamreddy Siva Prasad is a Senior Software Engineer working in E-Commerce domain. His areas of interest include Object Oriented Design, SOLID Design principles, RESTful WebServices and OpenSource softwares including Spring, MyBatis and Jenkins.

# MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 2

by Siva Reddy on November 19th, 2012 | **Filed in:** Enterprise Java **Tags:** MyBatis



To illustrate we are considering the following sample domain model:

There will be Users and each User may have a Blog and each Blog can contain zero or more posts.

The Database structure of the three tables are as follows:

```
CREATE TABLE user (
  user_id int(10) unsigned NOT NULL auto_increment,
  email_id varchar(45) NOT NULL,
  password varchar(45) NOT NULL,
  first_name varchar(45) NOT NULL,
  last_name varchar(45) default NULL,
  blog_id int(10) unsigned default NULL,
  PRIMARY KEY  (user_id),
  UNIQUE KEY Index_2_email_uniq (email_id),
  KEY FK_user_blog (blog_id),
  CONSTRAINT FK_user_blog FOREIGN KEY (blog_id) REFERENCES blog (blog_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE blog (
  blog_id int(10) unsigned NOT NULL auto_increment,
  blog_name varchar(45) NOT NULL,
  created_on datetime NOT NULL,
  PRIMARY KEY  (blog_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE post (
  post_id int(10) unsigned NOT NULL auto_increment,
  title varchar(45) NOT NULL,
  content varchar(1024) NOT NULL,
  created_on varchar(45) NOT NULL,
  blog_id int(10) unsigned NOT NULL,
  PRIMARY KEY  (post_id),
  KEY FK_post_blog (blog_id),
  CONSTRAINT FK_post_blog FOREIGN KEY (blog_id) REFERENCES blog (blog_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Here I am going to explain how to fetch and map *-has-One and One-To-Many result mappings.

```
package com.sivalabs.mybatisdemo.domain;

public class User
{
 private Integer userId;
 private String emailId;
 private String password;
 private String firstName;
 private String lastName;
 private Blog blog;
 //setters and getters
}
```

```
package com.sivalabs.mybatisdemo.domain;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Blog {

```

```
 private Integer blogId;
 private String blogName;
 private Date createdOn;
 private List<Post> posts = new ArrayList<Post>();
 //setters and getters
}
```

```
package com.sivalabs.mybatisdemo.domain;

import java.util.Date;

public class Post
{
 private Integer postId;
 private String title;
 private String content;
 private Date createdOn;
 //setters and getters
}
```

In mybatis-config.xml, configure type aliases for beans.

```
<typeAliases>
  <typeAlias type='com.sivalabs.mybatisdemo.domain.User' alias='User'/>
  <typeAlias type='com.sivalabs.mybatisdemo.domain.Blog' alias='Blog'/>
  <typeAlias type='com.sivalabs.mybatisdemo.domain.Post' alias='Post'/>
</typeAliases>
```

***-has-One Result Mapping:**

In UserMapper.xml, configure sql queries and result maps as follows:

```
<mapper namespace='com.sivalabs.mybatisdemo.mappers.UserMapper'>

 <resultMap type='User' id='UserResult'>
    <id property='userId' column='user_id'/>
    <result property='emailId' column='email_id'/>
    <result property='password' column='password'/>
    <result property='firstName' column='first_name'/>
    <result property='lastName' column='last_name'/>
    <association property='blog' resultMap='BlogResult'/>
   </resultMap>

 <resultMap type='Blog' id='BlogResult'>
    <id property='blogId' column='blog_id'/>
    <result property='blogName' column='BLOG_NAME'/>
    <result property='createdOn' column='CREATED_ON'/>
   </resultMap>

  <select id='getUserById' parameterType='int' resultMap='UserResult'>

     SELECT
      U.USER_ID, U.EMAIL_ID, U.PASSWORD, U.FIRST_NAME, U.LAST_NAME,
      B.BLOG_ID, B.BLOG_NAME, B.CREATED_ON
  FROM USER U LEFT OUTER JOIN BLOG B ON U.BLOG_ID=B.BLOG_ID
  WHERE U.USER_ID = #{userId}
  </select>

  <select id='getAllUsers' resultMap='UserResult'>
   SELECT
     U.USER_ID, U.EMAIL_ID, U.PASSWORD, U.FIRST_NAME, U.LAST_NAME,
     B.BLOG_ID, B.BLOG_NAME, B.CREATED_ON
 FROM USER U LEFT OUTER JOIN BLOG B ON U.BLOG_ID=B.BLOG_ID
  </select>

</mapper>
```

In JUnit Test, write a method to test the association loading.

```
public void getUserById()
{
 SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
 try{
  UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
  User user = userMapper.getUserById(1);
  System.out.println(user.getBlog());
 }finally{
  sqlSession.close();
 }
}
```

**One-To-Many Results Mapping:**

In BlogMapper.xml configure Blog to Posts relationship as follows:

```
<mapper namespace='com.sivalabs.mybatisdemo.mappers.BlogMapper'>

 <resultMap type='Blog' id='BlogResult'>
    <id property='blogId' column='blog_id'/>
    <result property='blogName' column='BLOG_NAME'/>
    <result property='createdOn' column='CREATED_ON'/>
    <collection property='posts' ofType='Post' resultMap='PostResult' columnPrefix='post_'>
</collection>
   </resultMap>

   <resultMap type='Post' id='PostResult'>
    <id property='postId' column='post_id'/>
    <result property='title' column='title'/>
```

```
    <result property='content' column='content'/>
    <result property='createdOn' column='created_on'/>
   </resultMap>

  <select id='getBlogById' parameterType='int' resultMap='BlogResult'>

     SELECT
      b.blog_id, b.blog_name, b.created_on,
      p.post_id as post_post_id, p.title as post_title, p.content as post_content, p.created_on as post_created_on
  FROM blog b left outer join post p on b.blog_id=p.blog_id
     WHERE b.BLOG_ID=#{blogId}
  </select>

  <select id='getAllBlogs' resultMap='BlogResult'>
   SELECT
    b.blog_id, b.blog_name, b.created_on as blog_created_on,
     p.post_id as post_post_id, p.title as post_title, p.content as post_content, p.created_on as post_created_on
 FROM blog b left outer join post p on b.blog_id=p.blog_id
  </select>

</mapper>
```

In JUnit Test, write a test method to test blog-to-posts relationship mapping.

```
public void getBlogById()
{
 SqlSession sqlSession = MyBatisUtil.getSqlSessionFactory().openSession();
 try{
 BlogMapper blogMapper = sqlSession.getMapper(BlogMapper.class);
 Blog blog = blogMapper.getBlogById(1);
 System.out.println(blog);
 List<Post> posts = blog.getPosts();
 for (Post post : posts) {
  System.out.println(post);
 }
 }finally{
  sqlSession.close();
 }
}
```

### Spring Integration

MyBatis-Spring is a subproject of MyBatis and provides Spring integration support which drastically simplifies the MyBatis usage. For those who are familiar with Spring's way of Dependency Injection process, using MyBatis-Spring is a very simple.

First let us see the process of using MyBatis without Spring.

1. Create SqlSessionFactory using SqlSessionFactoryBuilder by passing mybatis-config.xml which contains DataSource properties, List of Mapper XMLs and TypeAliases etc.

2. Create SqlSession object from SqlSessionFactory

3. Get Mapper instance from SqlSession and execute queries.

4. Commit or rollback the transaction using SqlSession object.

With MyBatis-Spring, most of the above steps can be configured in Spring ApplicationContext and SqlSession or Mapper instances can be injected into Spring Beans. Then we can use Spring's TransactionManagement features without writing transaction commit/rollback code all over the code.

Now let us see how we can configure MyBatis+Spring integration stuff.

**Step#1:** Configure MyBatis-Spring dependencies in pom.xml

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.1.1</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.1.1</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>3.1.1.RELEASE</version>
    <scope>test</scope>
</dependency>
<dependency>
```

```
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.21</version>
        <scope>runtime</scope>
    </dependency>
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib-nodep</artifactId>
    <version>2.2.2</version>
</dependency>
```

**Step#2:** You don't need to configure Database properties in mybatis-config.xml.

We can configure DataSource in Spring Container and use it to build MyBatis SqlSessionFactory.

Instead of SqlSessionFactoryBuilder, MyBatis-Spring uses org.mybatis.spring.SqlSessionFactoryBean to build SqlSessionFactory.

We can pass dataSource, Mapper XML files locations, typeAliases etc to SqlSessionFactoryBean.

```
<bean id='dataSource' class='org.apache.commons.dbcp.BasicDataSource'>
    <property name='driverClassName' value='${jdbc.driverClassName}'/>
    <property name='url' value='${jdbc.url}'/>
    <property name='username' value='${jdbc.username}'/>
    <property name='password' value='${jdbc.password}'/>
</bean>

<bean id='sqlSessionFactory' class='org.mybatis.spring.SqlSessionFactoryBean'>
    <property name='dataSource' ref='dataSource' />
    <property name='typeAliasesPackage' value='com.sivalabs.mybatisdemo.domain'/>
    <property name='mapperLocations' value='classpath*:com/sivalabs/mybatisdemo/mappers/**/*.xml'
/>
</bean>
```

**Step#3:** Configure SqlSessionTemplate w hich provides ThreadSafe SqlSession object.

```
<bean id='sqlSession' class='org.mybatis.spring.SqlSessionTemplate'>
  <constructor-arg index='0' ref='sqlSessionFactory' />
</bean>
```

**Step#4:** To be able to inject Mappers directly w e should register org.mybatis.spring.mapper.MapperScannerConfigurer and configure the package name w here to find Mapper Interfaces.

```
<bean class='org.mybatis.spring.mapper.MapperScannerConfigurer'>
  <property name='basePackage' value='com.sivalabs.mybatisdemo.mappers' />
</bean>
```

**Step#5:** Configure TransactionManager to support Annotation based Transaction support.

```
<tx:annotation-driven transaction-manager='transactionManager'/>

<bean id='transactionManager'
class='org.springframework.jdbc.datasource.DataSourceTransactionManager'>
    <property name='dataSource' ref='dataSource' />
</bean>
```

**Step#6:** Update the Service classes and register them in Spring container.

```
package com.sivalabs.mybatisdemo.service;

import java.util.List;
import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.sivalabs.mybatisdemo.domain.User;
import com.sivalabs.mybatisdemo.mappers.UserMapper;

@Service
@Transactional
public class UserService
{
    @Autowired
    private SqlSession sqlSession; //This is to demonstrate injecting SqlSession object

    public void insertUser(User user)
    {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        userMapper.insertUser(user);
    }

    public User getUserById(Integer userId)
    {
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
        return userMapper.getUserById(userId);
    }

}
```

```java
package com.sivalabs.mybatisdemo.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.sivalabs.mybatisdemo.domain.Blog;
import com.sivalabs.mybatisdemo.mappers.BlogMapper;

@Service
@Transactional
public class BlogService
{
    @Autowired
    private BlogMapper blogMapper; // This is to demonstratee how to inject Mappers directly

    public void insertBlog(Blog blog) {
        blogMapper.insertBlog(blog);
    }

    public Blog getBlogById(Integer blogId) {
        return blogMapper.getBlogById(blogId);
    }

    public List<Blog> getAllBlogs() {
        return blogMapper.getAllBlogs();
    }
}
```

**Note:** When we can directly inject Mappers then why do we need to inject SqlSession objects? Because SqlSession object contains more fine grained method which comes handy at times.

For Example: If we want to get count of how many records got updated by an Update query we can use SqlSession as follows:

```java
int updatedRowCount = sqlSession.update('com.sivalabs.mybatisdemo.mappers.UserMapper.updateUser',
user);
```

So far I didn't find a way to get the row update count without using SqlSession object.

**Step#7** Write JUnit Tests to test UserService and BlogService.

```java
package com.sivalabs.mybatisdemo;

import java.util.List;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.sivalabs.mybatisdemo.domain.User;
import com.sivalabs.mybatisdemo.service.UserService;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations='classpath:applicationContext.xml')
public class SpringUserServiceTest
{
    @Autowired
    private UserService userService;

    @Test
    public void testGetUserById()
    {
        User user = userService.getUserById(1);
        Assert.assertNotNull(user);
        System.out.println(user);
        System.out.println(user.getBlog());
    }

    @Test
    public void testUpdateUser()
    {
        long timestamp = System.currentTimeMillis();
        User user = userService.getUserById(2);
        user.setFirstName('TestFirstName'+timestamp);
        user.setLastName('TestLastName'+timestamp);
        userService.updateUser(user);
        User updatedUser = userService.getUserById(2);
        Assert.assertEquals(user.getFirstName(), updatedUser.getFirstName());
        Assert.assertEquals(user.getLastName(), updatedUser.getLastName());
    }

}
```

```java
package com.sivalabs.mybatisdemo;

import java.util.Date;
import java.util.List;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.sivalabs.mybatisdemo.domain.Blog;
```

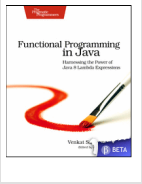
```
import com.sivalabs.mybatisdemo.domain.Post;
import com.sivalabs.mybatisdemo.service.BlogService;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations='classpath:applicationContext.xml')
public class SpringBlogServiceTest
{
    @Autowired
    private BlogService blogService;

    @Test
    public void testGetBlogById()
    {
        Blog blog = blogService.getBlogById(1);
        Assert.assertNotNull(blog);
        System.out.println(blog);
        List<Post> posts = blog.getPosts();
        for (Post post : posts) {
            System.out.println(post);
        }
    }

    @Test
    public void testInsertBlog()
    {
        Blog blog = new Blog();
        blog.setBlogName('test_blog_'+System.currentTimeMillis());
        blog.setCreatedOn(new Date());

        blogService.insertBlog(blog);
        Assert.assertTrue(blog.getBlogId() != 0);
        Blog createdBlog = blogService.getBlogById(blog.getBlogId());
        Assert.assertNotNull(createdBlog);
        Assert.assertEquals(blog.getBlogName(), createdBlog.getBlogName());

    }

}
```

***Reference:*** MyBatis Tutorial: Part 3 – Mapping Relationships, MyBatis Tutorial : Part4 – Spring Integration from our JCG partner Siva Reddy at the My Experiments on Technology blog.

**You might also like:**

- **MyBatis Tutorial – CRUD Operations and Mapping Relationships – Part 1**
- **Spring MVC 3 Controller for MyBatis CRUD operation**
- **MyBatis 3 – Spring integration tutorial**

**Related Whitepaper:**




**Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions**

Get ready to program in a whole new way!

Functional Programming in Java will help you quickly get on top of the new, essential Java 8 language features and the functional style that will change and improve your code. This short, targeted book will help you make the paradigm shift from the old imperative way to a less error-prone, more elegant, and concise coding style that's also a breeze to parallelize. You'll explore the syntax and semantics of lambda expressions, method and constructor references, and functional interfaces. You'll design and write applications better using the new standards in Java 8 and the JDK.



Leave a Reply

Name (Required)

Mail (w ill not be published) (Required)

Website

6 × four =

Notify me of follow up comments via e-mail

Sign me up for the newsletter!

Submit Comment

**Knowledge Base**

- Academy
- Examples
- Resources
- Tutorials
- Whitepapers

**Partners**

- Mkyong

**The Code Geeks Network**

- Java Code Geeks
- .NET Code Geeks
- Web Code Geeks

**Hall Of Fame**

- "Android Full Application Tutorial" series
- GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
- Android Game Development Tutorials
- Android Google Maps Tutorial
- Android Location Based Services Application – GPS location
- Funny Source Code Comments
- Java Best Practices – Vector vs ArrayList vs HashSet
- Android JSON Parsing with Gson Tutorial
- Android Quick Preferences Tutorial

**About Java Code Geeks**

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.