



# Spring MVC with JPA-Hibernate

Krishantha Dinesh  
[kdinesh@virtusa.com]

@2013 Dec

2000 West Park Drive  
Westborough MA 01581 USA  
Phone: 508 389 7300 Fax: 508 366 9901

The entire contents of this document are subject to copyright with all rights reserved. All copyrightable text and graphics, the selection, arrangement and presentation of all information and the overall design of the document are the sole and exclusive property of Virtusa.

Copyright © 2012 Virtusa Corporation. All rights reserved

**virtusa**<sup>®</sup>  
*Accelerating Business Outcomes*

# Pre - requisites

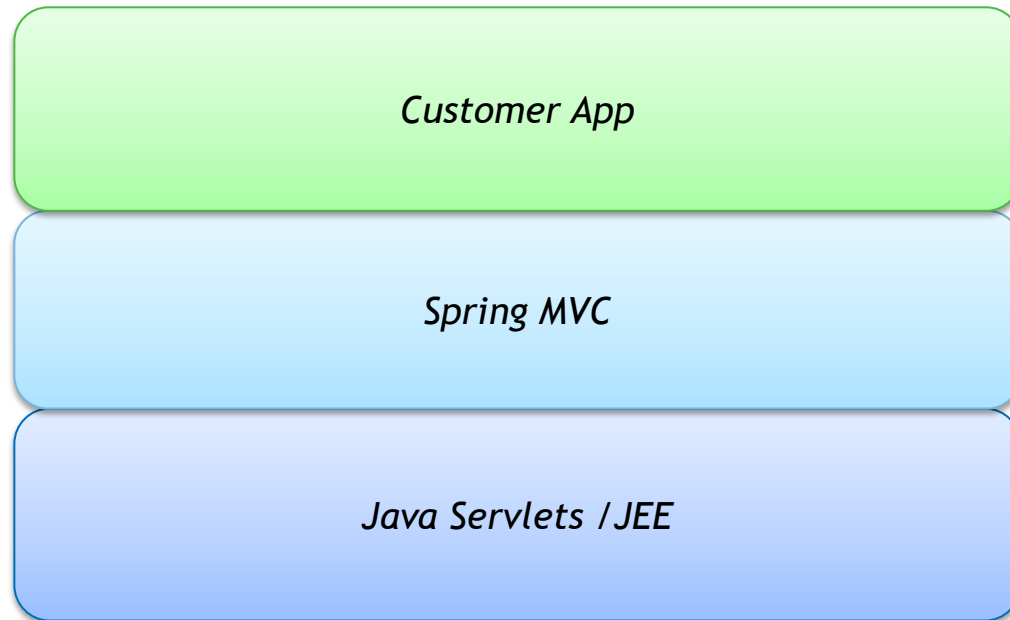
---

- Good understanding about java
- Follow Spring MVC course and that project with you
- Maven 3.x installed
- Mysql database installed
- Mysql workbench installed [optional]

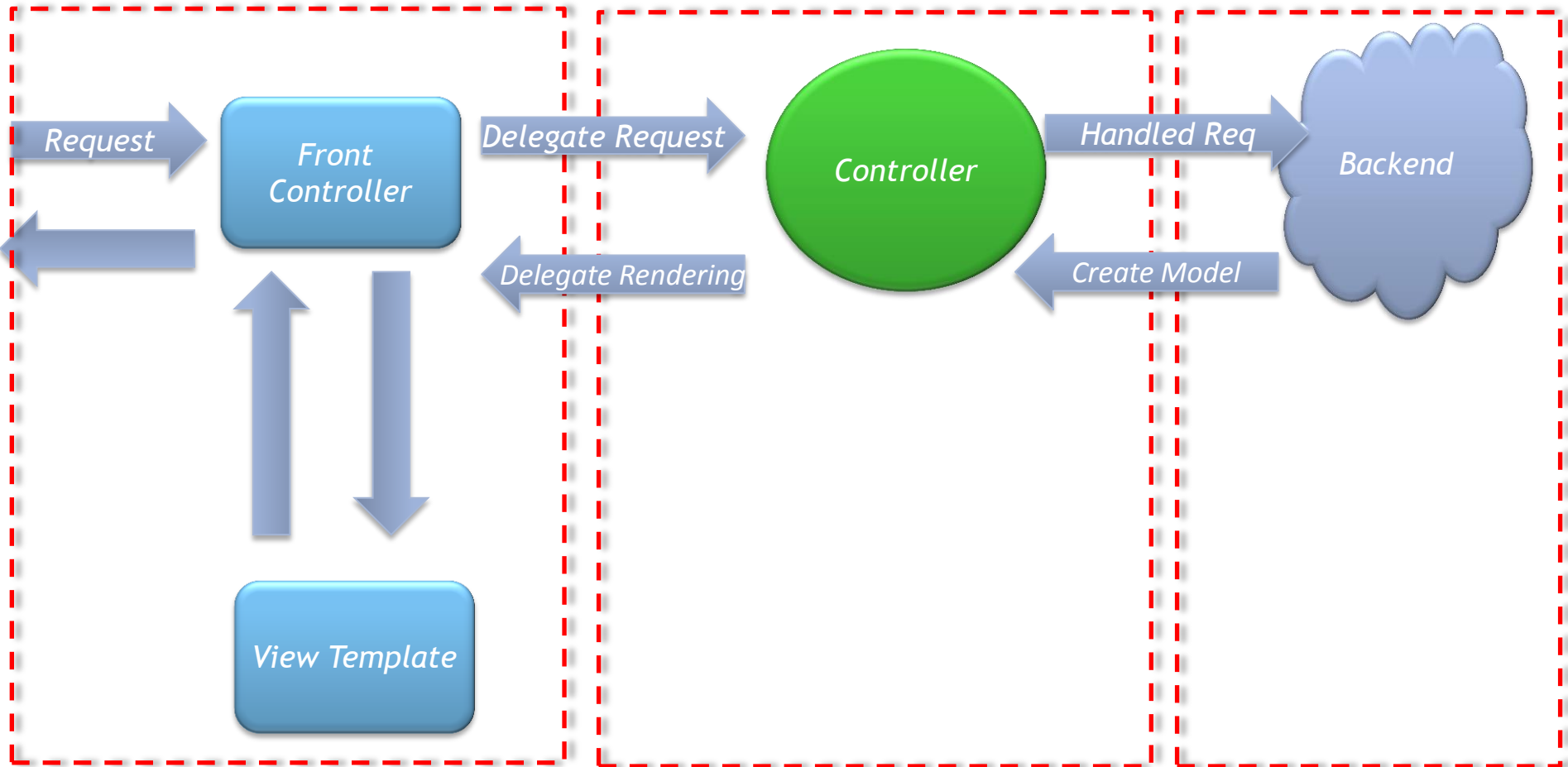
# Recap

# Architecture

---

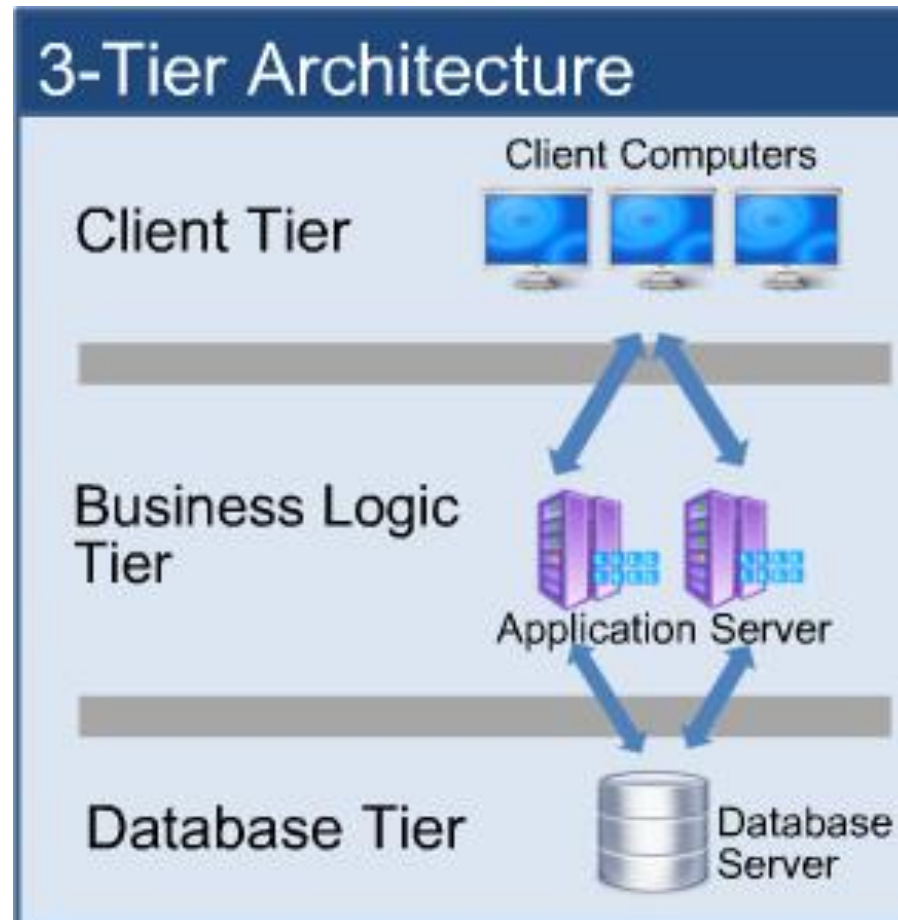


# Request Response life cycle



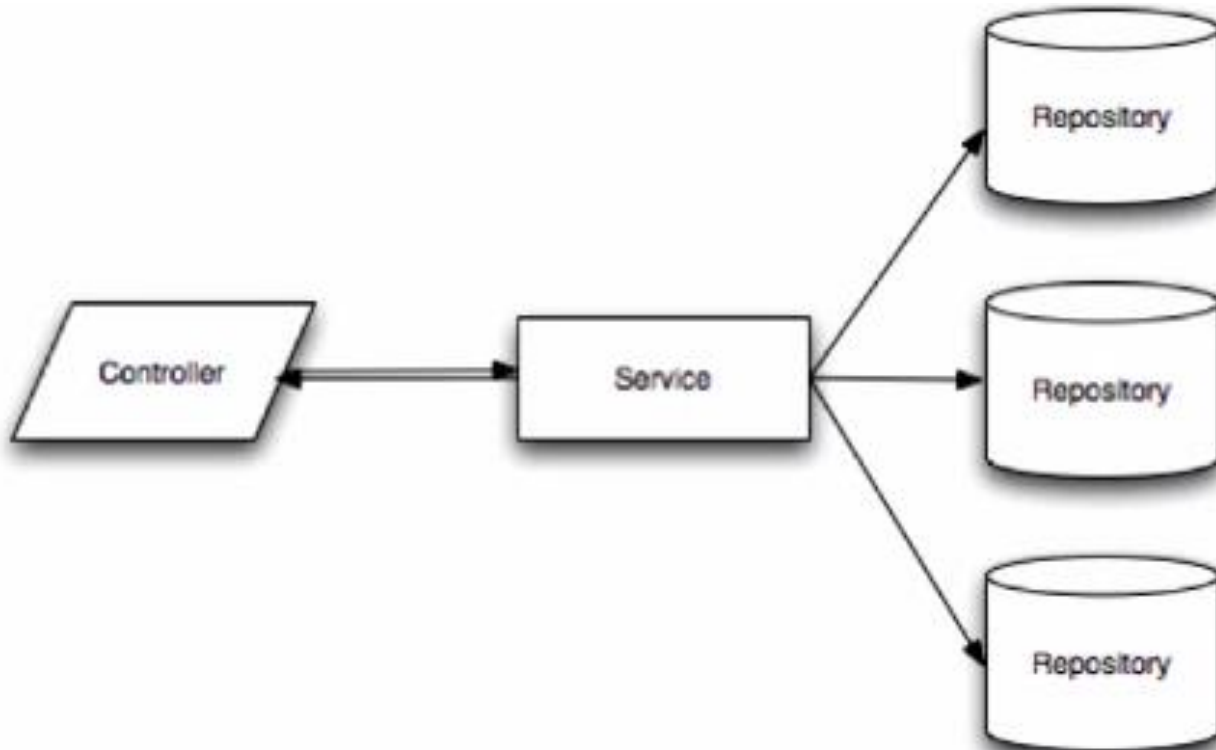
# Tiered architecture

- Main benefit is separation of concerns
- Re usable layers
- Maintenance



# Spring MVC 3 component

- Controller to route traffic
- Service for business logic store also where start our tractions
- Repository tier is almost like DAO



# Controller

---

- Annotated with @Controller
- Suppose to handle incoming request and build the response
- Business logic should NOT put here
- Grab information from request/response and hand over to the business logic
- Handle exception and routed in required way



# Service

---

- Annotated with `@service`
- Service tier define action (verbs) to the system
- Business logic should contained here
- State management should handle here
- Transaction should begins here
- May have same method which repository has but in different focus

# Repository

---

- Annotated with @Repository
- Define data (nouns) to the system
- Focus to interaction with database and CRUD operations
- One to one mapping with object
  - Customer → customerRepository
  - Company → companyRepository

# Auto wired

---

- 3 types of Injection
- Member variable
- Constructor
- Setter
- If we set to Member variable it use reflection to get things done and if we set on constructor it call actual constructor

# Setter Injection

- No args constructor required
- Private variable
- Setter follow the bean naming convention

```
public class Car {  
    private Engine engine;  
    public Car() {  
    }  
    public Engine getEngine() {  
        return engine;  
    }  
    public void setEngine(Engine engine) {  
        this.engine = engine;  
    }  
}
```

```
public class Engine {  
    private int numberOfCyl;  
    public Engine() {  
    }  
    public int getNumberOfCyl() {  
        return numberOfCyl;  
    }  
    public void setNumberOfCyl(int numberOfCyl) {  
        this.numberOfCyl = numberOfCyl;  
    }  
}
```

```
public static void main(String args[]){  
    Car car=new Car();  
    Engine engine=new Engine();  
    engine.setNumberOfCyl(6);  
    car.setEngine(engine);  
}
```

# Autowire

```
@Component
public class Car {
    @Autowired
    private Engine engine;

    public Car() {
    }

    public Engine getEngine() {
        return engine;
    }

    public void setEngine(Engine engine) {
        this.engine = engine;
    }
}
```

```
@Component
public class Engine {

    private int numberOfCyl;

    public Engine() {
    }

    public int getNumberOfCyl() {
        return numberOfCyl;
    }

    public void setNumberOfCyl(int numberOfCyl) {
        this.numberOfCyl = numberOfCyl;
    }
}
```

# Constructor injection

- Need to have constructor with argument

```
public class Car {  
  
    private Engine engine;  
  
    public Car(Engine engine) {  
        this.engine = engine;  
    }  
  
    public Engine getEngine() {  
        return engine;  
    }  
  
    public void setEngine(Engine engine) {  
        this.engine = engine;  
    }  
}
```

```
public class Engine {  
  
    private int numberOfCyl;  
  
    public Engine() {  
    }  
  
    public int getNumberOfCyl() {  
        return numberOfCyl;  
    }  
  
    public void setNumberOfCyl(int numberOfCyl) {  
        this.numberOfCyl = numberOfCyl;  
    }  
}
```

# Autowire

```
@Component
public class Car {

    private Engine engine;

    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }

    public Engine getEngine() {
        return engine;
    }

    public void setEngine(Engine engine) {
        this.engine = engine;
    }

}
```

```
@Component
public class Engine {

    private int numberOfCyl;

    public Engine() {

    }

    public int getNumberOfCyl() {
        return numberOfCyl;
    }

    public void setNumberOfCyl(int numberOfCyl) {
        this.numberOfCyl = numberOfCyl;
    }

}
```

---

**Now start**



# Create new Database (schema)

---

- `CREATE SCHEMA `salesmanager` ;`

# Setup dependencies with Maven

- Add following dependencies on top of what we have on salesManager App

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.21</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.2.7.SP1</version>
</dependency>
<dependency>
  <groupId>javax.transaction</groupId>
  <artifactId>jta</artifactId>
  <version>1.1</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>3.2.0.RELEASE</version>
</dependency>
```

# persistence.xml

---

- For classic JPA we need persistence.xml file for configure
  - Data source
  - Operation controls
  - Secondary level caching
- With spring
- We can override setting per environment such as live and test
- But still we NEED empty file in src/main/resources/META-INF/persistence.xml

# persistence.xml

- Add new folder in src/main/java/resources
- Folder name should be META-INF in exact case
- Create new file as persistence.xml
- Add following to file

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="punit">
    </persistence-unit>
</persistence>
```

# Web.xml

- To load spring we need to mention where to load configuration
- JPA has nothing to do with dispatcher servlet.
- So we use Listener to load configuration.
- Use web.xml to configure listener.
- Add following to web.xml [before <servlet>
- Container will trigger the listener and it will look for contextConfigLocation

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:/jpaContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

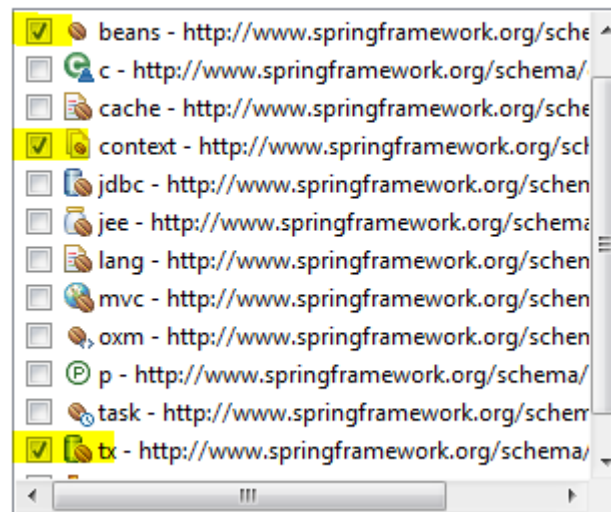
# jpaContext.xml

---

- Name can be anything. But standard ☹️
- Load from class path
- Contains:
  - Jpa vendor
  - Jpa properties
  - Transaction Manager
  - Annotation Configuration

- To create jpaContext.xml
- Right click on src/main/resources → new → spring bean Configuration
- Give file name as jpaContext.xml and click finish
- Select following namespaces

Select XSD namespaces to use in the configuration file



- Add following lines

```
<context:annotation-config/>
```

```
<bean
```

```
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:tx="http://www.springframework.org/schema/tx"
```

```
xmlns:context="http://www.springframework.org/schema/context"
```

```
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
```

```
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
```

```
<context:annotation-config/>
```

```
<bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"/>
```

```
</beans>
```



# Entity Manager Factory Bean

---

- Its use to startup jpa / hibernate inside our application
- There is class call LocalContainerEntityManagerFactoryBean in side the spring-orm.jar file
- Its reference to out persistence unit
- Its inject data if we not defined in the persistence unit.
- Its define which JPA provider we are using

# Entity Manager Factory

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">

  <context:annotation-config />
  <bean
    class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />

  <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="punit"/>
    <property name="dataSource" ref="dataSource" />
    <property name="jpaVendorAdapter">
      <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true"></property>
      </bean>
    </property>
    <property name="jpaPropertyMap">
      <map>
        <entry key="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
        <entry key="hibernate.hbm2ddl.auto" value="none" />
        <entry key="hibernate.format_sql" value="true" />
      </map>
    </property>
  </bean>
```

# Transaction Manager

---

- We are going to implement jpa transaction manager.
- It will reference to entityManagerFactory
- It is an annotation driven
- Its contains on spring-tx.jar

# Transaction Manager

- Add following code in to jpaContext.xml file to make transaction available

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">  
    <property name="entityManagerFactory" ref="entityManagerFactory" />  
</bean>  
<tx:annotation-driven transaction-manager="transactionManager" />
```

# Data Source

- To define data source we going to use DriverManagerDataSource which comes with spring-jdbc.jar
- Class name = com.mysql.jdbc.Driver
- Modify jpaContext.xml next to tx:annotation-driven

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/salesmanager?autoReconnect=true" />
  <property name="username" value="root"/>
  <property name="password" value="root"/>
</bean>
```

---

**JPA**

# What is it ?

---

- JPA is an interface for ORM
- JPA only a specification not a implementation
- Its map OO language to relational databases
- Its needs because database cannot understand OO paradigm
- JPA use JPQL

# Providers

- Data nucleus (<http://www.datanucleus.org>)
- Eclipse link – (<http://www.eclipse.org/eclipselink/>)
- Object DB (<http://www.objectdb.com/>)
- Open JPA (<http://openjpa.apache.org/>)
- Versant (<http://action.com/products/versant>)
- Hibernate (<http://www.hibernate.org/>)



eclipse)link



VERSANT





# What is entity

---

- It is just basic POJO
- When we add two things its become entity
- Need to add @entity and @id annotations

# Starting....

- In order to identify as entity it needs annotated as entity and id column
- Change salesTarget as follows

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import org.hibernate.validator.constraints.Range;

@Entity
public class SalesTarget {
    @Id
    @GeneratedValue
    private Long id;

    @Range(min = 1000, max = 100000)
    private int salesTarget;

    public int getSalesTarget() {
        return salesTarget;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

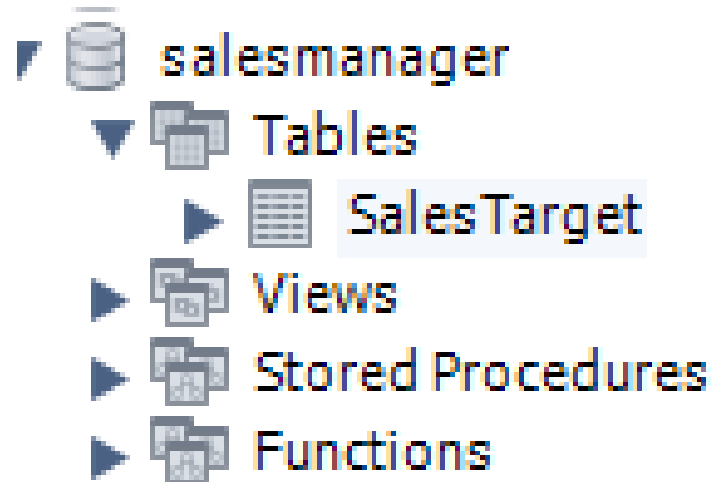
# Configure to auto create tables

- Go back to jpaContext.xml and change as follows

```
</property>
<property name="jpaPropertyMap">
  <map>
    <entry key="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
    <entry key="hibernate.hbm2ddl.auto" value="create" />
    <entry key="hibernate.format_sql" value="true" />
  </map>
</property>
```

# Add application to server and execute

```
Dec 02, 2013 12:30:12 AM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000227: Running hbm2ddl schema export
Hibernate:
    drop table if exists SalesTarget
Hibernate:
    create table SalesTarget (
        id bigint not null auto_increment,
        salesTarget integer not null,
        primary key (id)
    ) ENGINE=InnoDB
Dec 02, 2013 12:30:13 AM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
Dec 02, 2013 12:30:13 AM org.springframework.beans.factory.support.DefaultListableBeanFactory
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory
```



# JPA Annotations

# Entity Annotations

---

- @Entity – it make object to aware for database
- @Table – specific details about the db
- @Id – Identifier attribute for primary key
- @GeneratedValue
  - IDENTITY - use with auto increment
  - AUTO - automatically choose implementation based on database
  - SEQUENCE – if database support for sequence to this will support. Ie Oracle
  - TABLE – to use table and column for ensure uniqueness.

# @Table

- Change the code like this and execute it. You can see it created new table with given name

```
package com.virtusa.training.springmvc.model;

import javax.persistence.Entity;

@Entity
@Table(name="SALESTARGETS")
public class SalesTarget {

    @Id
    @GeneratedValue
    private Long id;

    @Range(min = 1000, max = 100000)
    private int salesTarget;

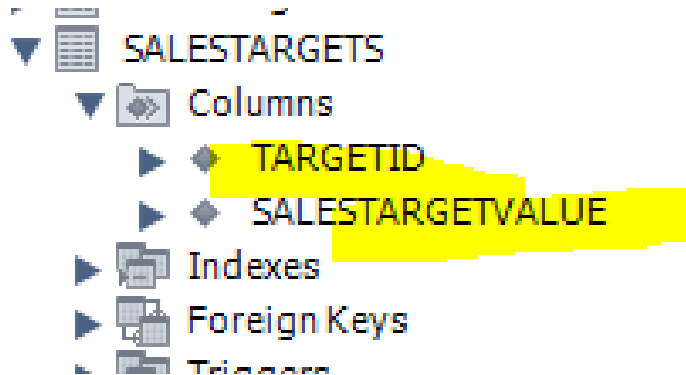
    public int getSalesTarget() {
        return salesTarget;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```

# Columns

- We don't need to accept what JPA define
- We can change name or data type what ever we want
- Also you can mention other properties you want such as not null etc
- Change code like this and execute



```
@Entity
@Table(name="SALESTARGETS")
public class SalesTarget {
    @Id
    @GeneratedValue
    @Column(name="TARGETID")
    private Long id;

    @Range(min = 1000, max = 100000)
    @Column(name="SALESTARGETVALUE")
    private int salesTarget;

    public int getSalesTarget() {
        return salesTarget;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
}
```



# Create-drop

- Even though documentation status its delete entire schema it is not true.
- Change to mode create-drop and execute – its created. Now stop server

```
/cc INFO: HHH000227: Running hbm2ddl schema export
pr Hibernate:
    drop table if exists SALESTARGETS
/> Hibernate:
    create table SALESTARGETS (
        TARGETID bigint not null auto_increment,
        SALESTARGETVALUE integer,
        primary key (TARGETID)
    ) ENGINE=InnoDB
Dec 02, 2013 12:55:18 AM org.hibernate.tool.hbm2ddl:
INFO: HHH000230: Schema export complete
```

- Its drop only one table ☹ -

```
/cc INFO: Closing JPA EntityManagerFactory for persi:
ir Dec 02, 2013 12:56:18 AM org.hibernate.tool.hbm2:
INFO: HHH000227: Running hbm2ddl schema export
/> Hibernate:
    drop table if exists SALESTARGETS
Dec 02, 2013 12:56:18 AM org.hibernate.tool.hbm2:
INFO: HHH000230: Schema export complete
Dec 02, 2013 12:56:18 AM org.apache.catalina.loader:
SEVERE: The web application [/SalesManager] regi:
Dec 02, 2013 12:56:18 AM org.apache.catalina.loader:
SEVERE: The web application [/SalesManager] appoe:
```

- ITS ONLY DROP IF OBJECT MAPPED

## How to use those stuff

# Helpers 😊

---

- We got one by one separate.
- How we can use those together ???
- First we need to get entity manager injected to our code
- We use `@PersistenceContext` for it
  - Will inject Entity manager to our code
- `@Service`
  - Where our business logic hosted
- `@Repository`
  - Place to DAO
- `@Transactional`
  - To manage transactions

# Create service

- R/Click on service package and create new interface as SalesTargetService

```
package com.virtusa.training.springmvc.service;  
  
import com.virtusa.training.springmvc.model.SalesTarget;  
  
public interface SalesTargetService {  
    SalesTarget save(SalesTarget salesTarget);  
}
```

# Create implementation class

- R/Click on service package
- Add new class as SalesTargetServiceImpl
- Annotate it with `@Service("salesTargetService")`

```
package com.virtusa.training.springmvc.service;

import org.springframework.stereotype.Service;

import com.virtusa.training.springmvc.model.SalesTarget;

@Service("salesTargetService")
public class SalesTargetServiceImpl implements SalesTargetService {

    public SalesTarget save(SalesTarget salesTarget) {

        return null;
    }
}
```

# Make Controller to save

```
+ import javax.validation.Valid;

@Controller
@SessionAttributes("addTarget")
public class SalesTargetController {

    - @Autowired
      private SalesTargetService salesTargetService;

    - @RequestMapping(value = "/addTarget", method = RequestMethod.GET)
      public String addSalesTarget(Model model) {
          SalesTarget salesTarget = new SalesTarget();
          salesTarget.setSalesTarget(81000);
          model.addAttribute("addTarget", salesTarget);
          return "addSalesTarget";
      }

    - @RequestMapping(value = "/addTarget", method = RequestMethod.POST)
      public String updateSalesTarget(
          @Valid @ModelAttribute("addTarget") SalesTarget salesTarget,
          BindingResult bindingResult) {
          if (bindingResult.hasErrors()) {
              System.out.println("ended with error");
              return "addSalesTarget";
          } else {
              salesTargetService.save(salesTarget);
          }

          System.out.println("Target updated" + salesTarget.getSalesTarget());
          return "redirect:index.jsp";
      }
}
```

# Create Repository

- Create new package as interface like below

```
package com.virtusa.training.springmvc.repository;  
  
import com.virtusa.training.springmvc.model.SalesTarget;  
  
public interface SalesTargetRepository {  
    SalesTarget save(SalesTarget salesTarget);  
}
```

# Create Implementation

- Create impl class and annotated with @Repository

```
package com.virtusa.training.springmvc.repository;

import org.springframework.stereotype.Repository;

import com.virtusa.training.springmvc.model.SalesTarget;

@Repository("salesTargetRepository")
public class SalesTargetRepositoryImpl implements SalesTargetRepository {

    public SalesTarget save(SalesTarget salesTarget) {

        return null;
    }
}
```



# Implement inside

```
package com.virtusa.training.springmvc.repository;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Repository;

import com.virtusa.training.springmvc.model.SalesTarget;

@Repository("salesTargetRepository")
public class SalesTargetRepositoryImpl implements SalesTargetRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public SalesTarget save(SalesTarget salesTarget) {

        entityManager.persist(salesTarget);

        return salesTarget;
    }
}
```

# Change SalesTargetServiceImpl

- Still service impl return null. So change it as follows

```
package com.virtusa.training.springmvc.service;

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

import com.virtusa.training.springmvc.model.SalesTarget;
import com.virtusa.training.springmvc.repository.SalesTargetRepository;

@Service("salesTargetService")
public class SalesTargetServiceImpl implements SalesTargetService {

    @Autowired
    private SalesTargetRepository repository;

    public SalesTarget save(SalesTarget salesTarget) {

        return repository.save(salesTarget);

    }

}
```

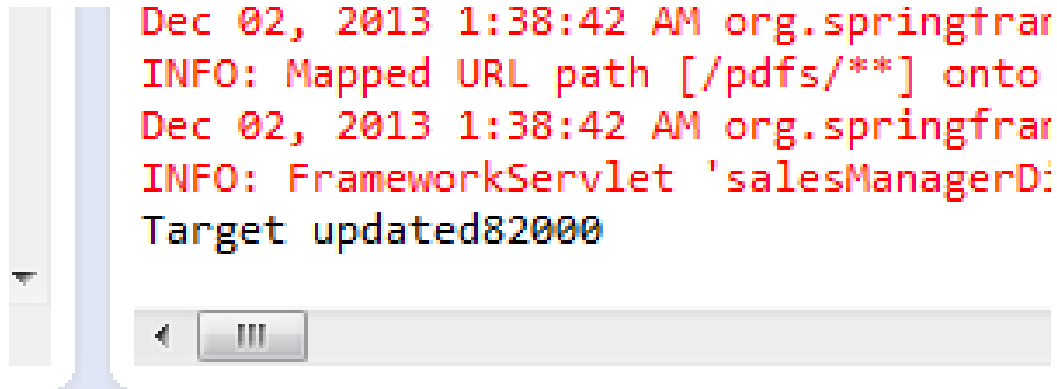
# explanations

---

- @ SalesTargetServiceImpl
- @ SalesTargetRepository
- @ SalesTargetController
- @ SalesTargetService
- Why we duplicated save every where ?????
- May be we have more logic on save.. May be external calls as well for validation so we can put those logic in service layer. Not at repository

# Execute code

- This says its working ?



```
Dec 02, 2013 1:38:42 AM org.springframework
INFO: Mapped URL path [/pdfs/**] onto
Dec 02, 2013 1:38:42 AM org.springframework
INFO: FrameworkServlet 'salesManagerD:
Target updated82000
```

- `SELECT * FROM salesmanager.SALESTARGETS;` - No records.. Why??



# Component scanner ????

- Change sarvlet-config as

```
<context:component-scan base-  
package="com.virtusa.training.springmvc.controller" />
```

- Add that to jpaContext.xml as

- ```
<context:component-scan base-  
package="com.virtusa.training.springmvc" />
```

- *Check now... WORKING...*



# FLUSH

- Change code like this

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Repository;

import com.virtusa.training.springmvc.model.SalesTarget;

@Repository("salesTargetRepository")
public class SalesTargetRepositoryImpl implements SalesTargetRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public SalesTarget save(SalesTarget salesTarget) {

        entityManager.persist(salesTarget);
        entityManager.flush();

        return salesTarget;
    }
}
```

- Working????

**type** Exception report

**message** Request processing failed; nested exception is javax.persistence.TransactionRequiredException: no transaction is in progress

**description** The server encountered an internal error that prevented it from fulfilling this request.

**exception**

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is javax.persi
    org.springframework.web.servlet.FrameworkServlet.processRequest (FrameworkServlet.java:927)
    org.springframework.web.servlet.FrameworkServlet.doPost (FrameworkServlet.java:822)
    javax.servlet.http.HttpServlet.service (HttpServlet.java:647)
    org.springframework.web.servlet.FrameworkServlet.service (FrameworkServlet.java:796)
    javax.servlet.http.HttpServlet.service (HttpServlet.java:728)
```

**root cause**

```
javax.persistence.TransactionRequiredException: no transaction is in progress
    org.hibernate.ejb.AbstractEntityManagerImpl.flush (AbstractEntityManagerImpl.java:993)
    sun.reflect.NativeMethodAccessorImpl.invoke0 (Native Method)
    sun.reflect.NativeMethodAccessorImpl.invoke (Unknown Source)
    sun.reflect.DelegatingMethodAccessorImpl.invoke (Unknown Source)
    java.lang.reflect.Method.invoke (Unknown Source)
```

# Setup transaction

- Add @Transactional annotation and execute

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.virtusa.training.springmvc.model.SalesTarget;
import com.virtusa.training.springmvc.repository.SalesTargetRepository;

@Service("salesTargetService")
public class SalesTargetServiceImpl implements SalesTargetService {

    @Autowired
    private SalesTargetRepository repository;

    @Transactional
    public SalesTarget save(SalesTarget salesTarget) {

        return repository.save(salesTarget);
    }
}
```

```
Dec 02, 2013 2:02:34 AM org.springframework
INFO: FrameworkServlet 'salesManagerDispa
Hibernate:
insert
into
    SALESTARGETS
    (SALESTARGETVALUE)
values
    (?)
Target updated82000
```



- Now we know how can deal with primitive types. But how about objects ?
- JPA is all about dealing with objects. So we need to have mechanism
- There are 4 types
  - One to One
  - One to Many
  - Many to One
  - Many to Many
- Can use as
  - Uni directional
  - Bi directional
  - cascade

# One to Many

---

- Use with @OneToMany
- In example of our project we have sales target as well as methods to achieve it.

---

# Join

# Convert it to Entity

```
⊕ import javax.persistence.Entity;

@Entity
public class Sales {

    ⊖ @Id
      @GeneratedValue
      private Long id;

    ⊖ public Long getId() {
        return id;
    }

    ⊖ public void setId(Long id) {
        this.id = id;
    }

    ⊖ @Range(min = 1, max = 10000)
      private int sales;

    ⊖ @NotNull
      private String salesTypes;
```

- 
- Lest setup one to many or many to one relation ship.
  - From sales perspective many sales for one target(many to one)
  - From target perspective one target for many sales (one to many)
  - Lest wired it in bi directional way

- @ Sales class
- Create reference to salesTarget
- Generate getters and setters
- Annotate

```

@Table(name="SALESTARGETS")
public class SalesTarget {

    @Id
    @GeneratedValue
    @Column(name="TARGETID")
    private Long id;

    @OneToMany(mappedBy="salesTarget", cascade=CascadeType.ALL)
    List<Sales> sales= new ArrayList<>();

    public List<Sales> getSales() {
        return sales;
    }

    public void setSales(List<Sales> sales) {
        this.sales = sales;
    }

    @Range(min=1000, max=100000)
    @Column(name="SALESTARGETVALUE")
    private int salesTargetValue;

```

```

public class Sales {

    @Id
    @GeneratedValue
    private Long id;

    @ManyToOne
    private SalesTarget salesTarget;

    public SalesTarget getSalesTarget() {
        return salesTarget;
    }

    public void setSalesTarget(SalesTarget salesTarget) {
        this.salesTarget = salesTarget;
    }

    public Long getId() {

```

# Edit Controller

```
@RequestMapping(value = "/addSales", method = RequestMethod.POST)
public String addSales(@Valid @ModelAttribute("salesValues") Sales sales, HttpSession session,
    BindingResult result) {
    System.out.println("sale value is: " + sales.getSales());
    System.out.println("sale type is: " + sales.getSalesTypes());

    if(result.hasErrors()){
        return "addSalesValue";
    }
    else{
        SalesTarget salesTarget=(SalesTarget) session.getAttribute("salesTarget");
        sales.setSalesTarget(salesTarget);
        salesService.save(sales);
    }
    return "addSalesValue";
}

@RequestMapping(value = "/salesTypes", method = RequestMethod.GET)
public @ResponseBody
List<SalesType> getAllSalesTypes() {
    return salesService.getAllSalesTypes();
}
```

# Edit service

```
package com.virtusa.training.springmvc.service;

+ import java.util.List;

public interface SalesService {

    List<SalesType> getAllSalesTypes();

    Sales save(Sales sales);

}
```

- Fix the service implementation as well
- Now we need repository



# Create interface and impl

```
package com.virtusa.training.springmvc.repository;  
  
import com.virtusa.training.springmvc.model.Sales;  
  
public interface SalesRepository {  
    Sales save(Sales sales);  
}
```

```
package com.virtusa.training.springmvc.repository;  
  
import com.virtusa.training.springmvc.model.Sales;  
  
public class SalesRepositoryImpl implements SalesRepository {  
  
    public Sales save(Sales sales) {  
        return null;  
    }  
}
```

# Repository Impl

```
package com.virtusa.training.springmvc.repository;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Repository;

import com.virtusa.training.springmvc.model.Sales;

@Repository("salesRepository")
public class SalesRepositoryImpl implements SalesRepository {

    @PersistenceContext
    private EntityManager entityManager;

    public Sales save(Sales sales) {

        entityManager.persist(sales);
        entityManager.flush();
        return sales;
    }
}
```

# Service Impl

```
package com.virtusa.training.springmvc.service;

import java.util.ArrayList;

@Service
public class SalesServiceImpl implements SalesService {

    @Autowired
    private SalesRepository repository;

    @Override
    public List<SalesType> getAllSalesTypes() {
        List<SalesType> salesTypes = new ArrayList<>();
        SalesType direct = new SalesType();
        direct.setSalesTypes("Direct");
        salesTypes.add(direct);
        SalesType web = new SalesType();
        web.setSalesTypes("Web");
        salesTypes.add(web);
        SalesType indirect = new SalesType();
        indirect.setSalesTypes("Indirect");
        salesTypes.add(indirect);
        return salesTypes;
    }

    @Transactional
    public Sales save(Sales sales) {

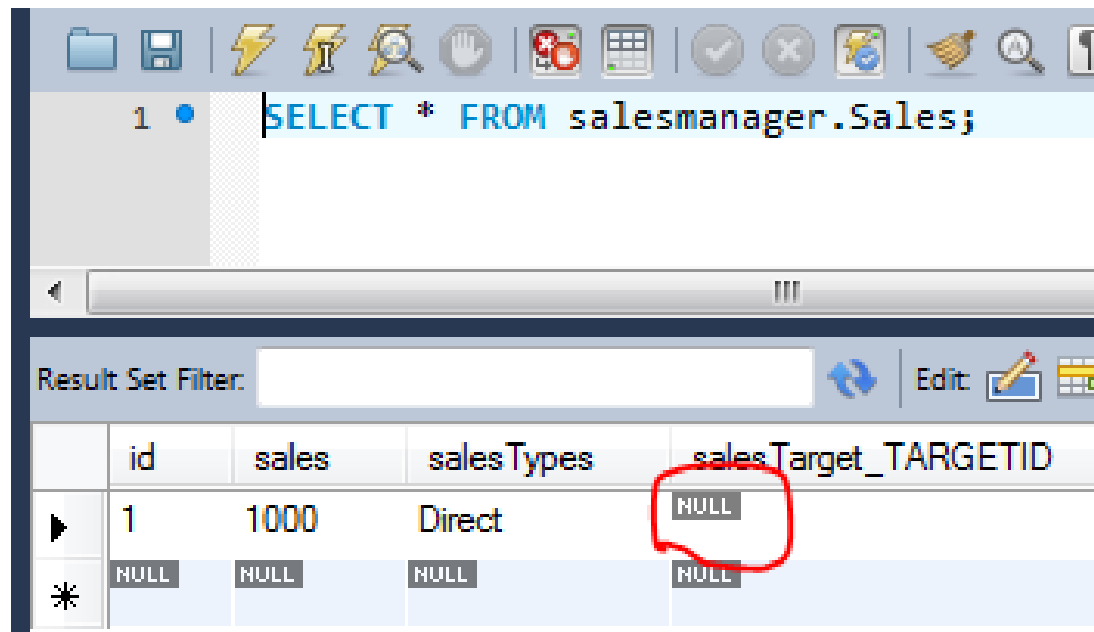
        return repository.save(sales);
    }
}
```

# Execute app

```
ERROR: Table 'salesmanager.sales' doesn't exist
Hibernate:
    drop table if exists SALESTARGETS
Hibernate:
    drop table if exists Sales
Hibernate:
    create table SALESTARGETS (
        TARGETID bigint not null auto_increment,
        SALESTARGETVALUE integer,
        primary key (TARGETID)
    ) ENGINE=InnoDB
Hibernate:
    create table Sales (
        id bigint not null auto_increment,
        sales integer not null,
        salesTypes varchar(255) not null,
        salesTarget_TARGETID bigint,
        primary key (id)
    ) ENGINE=InnoDB
Hibernate:
    alter table Sales
        add index FK_c5xntopgctukxst6ry9u3ldot (salesTarget_TARGETID),
        add constraint FK_c5xntopgctukxst6ry9u3ldot
        foreign key (salesTarget_TARGETID)
        references SALESTARGETS (TARGETID)
Dec 04, 2013 1:52:51 AM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
Dec 04, 2013 1:52:51 AM org.springframework.beans.factory.support.DefaultLis
```

# Verify

- Go to app
- Add target
- Add sales



The screenshot shows a database application window. At the top, there is a toolbar with various icons. Below the toolbar, a SQL query is entered in a text area: `SELECT * FROM salesmanager.Sales;`. Below the query, there is a "Result Set Filter" field. The results are displayed in a table with the following columns: `id`, `sales`, `salesTypes`, and `salesTarget_TARGETID`. The first row of data shows `id` as 1, `sales` as 1000, `salesTypes` as Direct, and `salesTarget_TARGETID` as NULL. The NULL value is circled in red. Below the first row, there is a row with a "\*" icon in the first column and NULL values in the other columns.

|   | id   | sales | salesTypes | salesTarget_TARGETID |
|---|------|-------|------------|----------------------|
| ▶ | 1    | 1000  | Direct     | NULL                 |
| * | NULL | NULL  | NULL       | NULL                 |

- Wrong variable fetched from session

```
@RequestMapping(value = "/addSales", method = RequestMethod.POST)
public String addSales(@Valid @ModelAttribute("salesValues") Sales sales, HttpSession session,
    BindingResult result) {
    System.out.println("sale value is: " + sales.getSales());
    System.out.println("sale type is: " + sales.getSalesTypes());

    if(result.hasErrors()){
        return "addSalesValue";
    }
    else{
        SalesTarget salesTarget=(SalesTarget) session.getAttribute("addTarget");
        sales.setSalesTarget(salesTarget);
        salesService.save(sales);
    }
    return "addSalesValue";
}
```

# FETCH

---

- There are two type
- Lazy – query database when need – property call (default)
- Eager – Query database when it created

# JPQL

---

- It is not SQL
- JPQL is java persistence query language
- It is dealing with object



# Report (Fetch)

- Create new jsp file as getTargets.jsp

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" http://www.w3
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Get Targets</title>
</head>
<body>
<table>
<tr>
<th>ID</th>
<th>Target</th>
</tr>

<c:forEach items="${targets}" var="target">
<tr>
<td>${target.id}</td>
<td>${target.salesTarget}</td>
<td>
<table>
<tr>
<th>sales id</th>
<th>sales value</th>
<th>sales type</th>
</tr>
<c:forEach items="${target.sales}" var="salelist">
<tr>
<td>${salelist.id}</td>
<td>${salelist.sales}</td> |
<td>${salelist.salesTypes}</td>
</tr>
</c:forEach>
</table>
</td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

# Add new method to salesTargetController

- Also create in relevant interface and impl (follow quick fix)

```
return "redirect:index.jsp";
```

```
}
```

```
@RequestMapping(value="/getTarget",method=RequestMethod.GET)
public String getTargets(Model model){
    List<SalesTarget> targets=salesTargetService.findAllTargets();
    model.addAttribute("targets",targets);
    return "getTargets";
}
```

# Fix the implementation as follows and fix errors

```
package com.virtusa.training.springmvc.service;

import java.util.List;

@Service("salesTargetService")
public class SalesTargetServiceImpl implements SalesTargetService {

    @Autowired
    private SalesTargetRepository repository;

    @Transactional
    public SalesTarget save(SalesTarget salesTarget) {

        return repository.save(salesTarget);
    }

    @Override
    public List<SalesTarget> findAllTargets() {
        return repository.loadAll();
    }
}
```

# Modify Repository Impl

```
package com.virtusa.training.springmvc.repository;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import org.springframework.stereotype.Repository;

import com.virtusa.training.springmvc.model.SalesTarget;

@Repository("salesTargetRepository")
public class SalesTargetRepositoryImpl implements SalesTargetRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public SalesTarget save(SalesTarget salesTarget) {
        entityManager.persist(salesTarget);
        entityManager.flush();
        return salesTarget;
    }

    @Override
    public List<SalesTarget> loadAll() {
        Query query = entityManager.createQuery("select t from SalesTarget t");

        List targets = query.getResultList();

        return targets;
    }
}
```

# Define fetch type and execute

```
import java.util.ArrayList;
@Entity
@Table(name="SALESTARGETS")
public class SalesTarget {

    @Id
    @GeneratedValue
    @Column(name="TARGETID")
    private Long id;

    @OneToMany(mappedBy="salesTarget", cascade=CascadeType.ALL, fetch=FetchType.EAGER)
    List<Sales> sales= new ArrayList<>();

    @Range(min=1000, max=100000)
    @Column(name="SALESTARGETVALUE")
    private int salesTarget;

    public Long getId() {
        return id;
    }
}
```

# Change from eager to lazy and see

- It will give an exception ☹️
- Add filter to web.xml and try again

```
</listener>
<filter>
    <filter-name>SpringOpenEntityManagerInViewFilter</filter-name>
    <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>SpringOpenEntityManagerInViewFilter</filter-name>
    <url-pattern>/*/url-pattern</url-pattern>
</filter-mapping>
<servlet>
```

# Projection

---

- It's important when you need to have control
- It is a great way to prepare object to UI
- It's based on JPSQ.
- Those can be JPA entities
- Need a constructor for the projection

# Create new report class

```
package com.virtusa.training.springmvc.model;

public class TargetReport {
    private int targetValue;
    private int saleValue;
    private String salesType;
    public String getSalesType() {
        return salesType;
    }
    public int getSaleValue() {
        return saleValue;
    }
    public int getTargetValue() {
        return targetValue;
    }
    public void setSalesType(String salesType) {
        this.salesType = salesType;
    }
    public void setSaleValue(int saleValue) {
        this.saleValue = saleValue;
    }
    public void setTargetValue(int targetValue) {
        this.targetValue = targetValue;
    }
}
```



# Create constructor

```
public TargetReport(int targetValue, int saleValue, String salesType) {  
    this.targetValue = targetValue;  
    this.saleValue = saleValue;  
    this.salesType = salesType;  
}
```

# Create new jsp file

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>sales Target Report</title>
</head>
<body>

    <table>
        <tr>
            <th>Target</th>
            <th>Sales</th>
            <th>Type</th>

        </tr>
        <c:forEach items="${targetReport}" var="targetrpt">
            <tr>
                <td>${targetrpt.targetValue}</td>
                <td>${targetrpt.saleValue}</td>
                <td>${targetrpt.salesType}</td>
            </tr>
        </c:forEach>
    </table>

</body>
</html>
```

# Add method to controller



```
@RequestMapping(value="/getTargetReport",method=RequestMethod.GET)
public String getTargetsReport(Model model){
    List<TargetReport> targetsrpt=salesTargetService.findAllTargetReport();
    model.addAttribute("targerReport",targetsrpt);
    return "getTargetReport";
}
```

# Fix unimplemented issue

- And implement at repository

```
@Override
public List<TargetReport> loadAllReport() {

    Query query = entityManager
        .createQuery("select new com.virtusa.training.springmvc.model.TargetReport(t.salesTarget,s.sales,s.salesTypes)"
            + " from SalesTarget t,Sales s where t.id=s.salesTarget.id");
    List rpt = query.getResultList();
    return rpt;
}
```

Now execute the program

# Spring JPA

# Updates

---

- Persist just do creates
- Merge is for update
- Save method can overridden to handle both

# Modify repository for update

```
@Repository("salesTargetRepository")
public class SalesTargetRepositoryImpl implements SalesTargetRepository {

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public SalesTarget save(SalesTarget salesTarget) {

        if (salesTarget.getId() == null) {
            entityManager.persist(salesTarget);
            entityManager.flush();
        } else {
            salesTarget = entityManager.merge(salesTarget);
        }

        return salesTarget;
    }
}
```

# Change salesTargetController

- Execute code and now you can see its update if exist !!!!!

```
@RequestMapping(value = "/addTarget", method = RequestMethod.GET)
public String addSalesTarget(Model model, HttpSession session) {
    |
    SalesTarget salesTarget = (SalesTarget) session
        .getAttribute("addTarget");
    if (salesTarget == null) {
        salesTarget = new SalesTarget();
        salesTarget.setSalesTarget(81000);
    }

    model.addAttribute("addTarget", salesTarget);
    return "addSalesTarget";
}
```



# Name Query

- We can get out all our queries from code base and put in domain class
- Edit the domain class as follows (Sales Target)

```
import java.util.ArrayList;

@Entity
@Table(name = "SALESTARGETS")
@NamedQueries({ @NamedQuery(name = SalesTarget.LOAD_ALL_REPORT, query = "select new com.virtusa.training.springmvc.model.TargetReport"
    + "(t.salesTarget,s.sales,s.salesTypes) "
    + "from SalesTarget t,Sales s where t.id=s.salesTarget.id") })
public class SalesTarget {

    public static final String LOAD_ALL_REPORT = "loadAllTargetReport";

    @Id
    @GeneratedValue
    @Column(name = "TARGETID")
    private Long id;

    @Range(min = 1000, max = 100000)
    @Column(name = "SALESTARGETVALUE")
    private int salesTarget;
```

# Call Named Query

```
@Override
public List<TargetReport> loadAllTargetReport() {

    /*
     * Query query = entityManager.createQuery(
     * "select new com.virtusa.training.springmvc.model.TargetReport" +
     * "(t.salesTarget,s.sales,s.salesTypes) " +
     * "from SalesTarget t,Sales s where t.id=s.salesTarget.id");
     */

    TypedQuery<TargetReport> query = entityManager.createNamedQuery(
        SalesTarget.LOAD_ALL_REPORT, TargetReport.class);

    List rpt = query.getResultList();
    return rpt;
}
```

# What is Spring Data JPA






---

- It is wrapper for JPA
- Replace repository tier
- Extremely powerful
- Can be extended for additional functionality

# Add jar using Maven

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
  <version>1.3.0.RELEASE</version>
  <exclusions>
    <exclusion>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

# Add schema for jpaContext

- ☐  jdbc - <http://www.springframework.org/schema/jdbc>
- ☐  jee - <http://www.springframework.org/schema/jee>
- ☒  jpa - <http://www.springframework.org/schema/data/jpa>
- ☐  lang - <http://www.springframework.org/schema/lang>
- ☐  mvc - <http://www.springframework.org/schema/mvc>

```
xmlns:xsi="http://www.springframework.org/schema/xsi" xsi:schemaLocation="http://www.springframework.org/schema/context http://www.springframework.org/schema/context-3.0.xsd"
<context:annotation-config />
<context:component-scan base-package="com.virtusa.training" />
<jpa:repositories base-package="com.virtusa.training.springmvc.repository"/>
<bean
```

- Now delete impl class ☹ ☹ ☹
- And change interface like this

```
➤ import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import com.virtusa.training.springmvc.model.Sales;  
  
@Repository("salesRepository")  
public interface SalesRepository extends JpaRepository<Sales, Long> {  
  
}
```

# Convert SaleTarget

- When convert salesTarget you need to get in to account that it has few methods
- Delete Impl method
- And change interface as follows
- It will worked!!!!

```
package com.virtusa.training.springmvc.repository;

import java.util.List;

public interface SalesTargetRepository extends JpaRepository<SalesTarget, Long> {
    //SalesTarget save(SalesTarget salesTarget);

    //List<SalesTarget> loadAllTargtes();

    @Query("select new com.virtusa.training.springmvc.model.TargetReport"
        + "(t.salesTarget,s.sales,s.salesTypes) "
        + "from SalesTarget t,Sales s where t.id=s.salesTarget.id")
    List<TargetReport> loadAllTargetReport();
}
```

---

**Good LuCk with integration!!!**