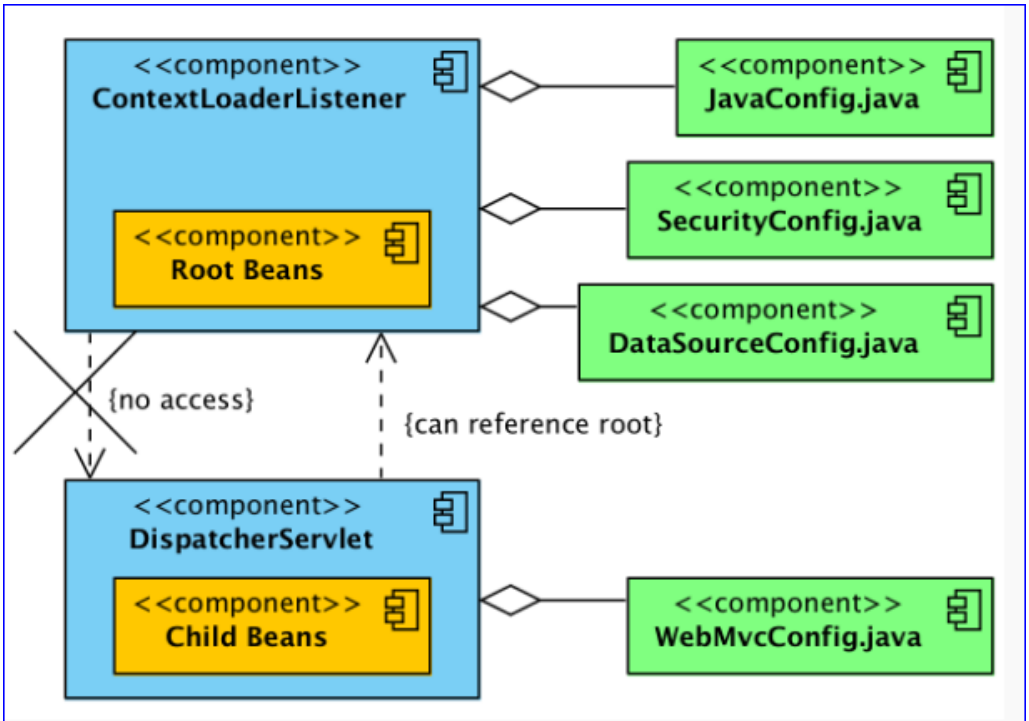


ContextLoaderListener vs DispatcherServlet

GoalController.java | index.jsp | web.xml | jpaContext.xml | persistence.xml | Goal.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <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>
    <servlet>
        <servlet-name>fitTrackerServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/config/servlet-config.xml, /WEB-INF/config/jpaContext.xml </param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>fitTrackerServlet</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>fitTrackerServlet</servlet-name>
        <url-pattern>/pdfs/**</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>fitTrackerServlet</servlet-name>
        <url-pattern>/images/**</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
```

jpaContext.xml

- **Used in place of the persistence.xml**
- **Doesn't have to be named jpaContext.xml**
- **Loaded from the Classpath**
  - src/main/resources/jpaContext.xml
- **Contains:**
  - EntityManagerFactory
    - Jpa Vendor
    - Jpa Properties
  - Transaction Manager
  - Annotation configuration
  - Datasouce configuration or lookup

Entity Manager Factory

- ***LocalContainerEntityManagerFactoryBean***
  - *Located in spring-orm.jar*
  - *References our persistence unit*
  - *Injected Datasource if one isn't defined in the persistence unit*
  - *Defines what Vendor (provider) we are using*
  - *Vendor specific JPA properties*
    - *We'll cover these in the JPA Module*

```
GoalController.java   index.jsp   web.xml   *jpaContext.xml   FitnessTracker/pom.xml
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:tx="http://www.springframework.org/schema/tx"
5      xmlns:context="http://www.springframework.org/schema/context"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7          http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
8          http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
9
10     <context:annotation-config /><!-- I want to config entire app using annotations -->
11     <bean
12         class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
13
14     <bean id="entityManagerFactory"
15         class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
16
17         <property name="persistenceUnitName" value="punit" />
18         <property name="dataSource" ref="dataSource" />
19         <property name="jpaVendorAdapter">
20             <bean
21                 class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
22                 <property name="showSql" value="true" />
23             </bean>
24         </property>
25         <property name="jpaPropertyMap">
26             <map>
27                 <entry key="hibernate.dialect"
28                     value="org.hibernate.dialect.OracleDialect" />
29                 <entry key="hibernate.hbm2ddl.auto" value="none"></entry>
30                 <!-- interacting with ddl with ur database needs to work, values will
31                     discuss later -->
32                 <entry key="hibernate.format_sql" value="true" />
33                 <!-- format_sql for debugging purpose, These are all the properties needed
34                     for jpa to be setup with our local container entity manager -->
35             </map>
36         </property>
37     </bean>
38 </beans>
```

With the above configuration the provider is setup with JPA. We are using hibernate as JPA vendor.

Transaction Manager:

They work with in a transaction but we don't write the transaction explicitly. We don't need to start and close the transaction we just need to wrap the code around a transaction manager.

Also the namespace we added to the Jpa context file.

Now add the transaction manager to our JpaContext.xml

```
39     <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
40         <property name="entityManagerFactory" ref="entityManagerFactory"></property>
41     </bean>
42     <!--add a property entitymanager factory to wrap a transaction
43     This has a setter. setEntityManagerFactory. the ref bean is going to get injected into transaction Manager.
44     This doen't start or stop the transaction it just make available the transaction manger -->
45     <tx:annotation-driven transaction-manager="transactionManager"/>
46     <!-- Now we have all of our jpa business wrapped in a jpa transaction. we can configure the
47     transaction using annotation in line 45 -->
48 </beans>
```

Last thing we need to setup the Data source

Datasource:

Datasource

- ***DriverManagerDataSource***
  - *spring-jdbc.jar*
- ***driverClassName***
  - com.mysql.jdbc.Driver
- ***url***
  - jdbc:mysql://localhost:3306/fitnessTracker?autoReconnect=true
- ***username***
  - *root*
- ***password***
  - password

```
49    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
50        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
51        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
52        <property name="username" value="SYSTEM"/>
53        <property name="password" value="admin"/>
54    </bean>
```

RECAP:

Added namespaces like beans, context, tx etc.

GoalController.java | index.jsp | web.xml | jpaContext.xml | FitnessTracker/pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:tx="http://www.springframework.org/schema/tx"
5      xmlns:context="http://www.springframework.org/schema/context"
6      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
7          http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
8          http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd">
9
```

We added annotation config to use annotations in our application

```
<context:annotation-config /><!-- I want to config entire app using annotations -->
```

We added persistence annotation bean to inject persistence context in to the entity manager factory into our resources

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

EntityManagerFactory -
`org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean`

Transaction Manager (Injected entity manager to transaction manager)

`org.springframework.orm.jpa.JpaTransactionManager`

Datasource

`org.springframework.jdbc.datasource.DriverManagerDataSource`

with this we are done with the jpa configuration

OVERVIEW OF JPA AND CREATE YOUR FIRST ENTITY

- **Jpa is java persistence api is an interface for a specific implementation of an ORM tool.**
- **ORM were created to help to bridge the gap between OOP languages such as Java and various relational databases. (RDBMS)**
- **RDBMS don't have the concepts of oo techniques like inheritance, polymorphism etc.**
- **There are things like we can do in a design use of an ORM tool JPA to help representing these concepts in our database.**

What is Jpa?

- Actually just a specification
- Thought of as an Object Relational Mapping tool (ORM)
  - An ORM is an implementation of Jpa
- A specification for accessing, persisting, and managing data between Java objects / classes and a relational database
- It is not SQL
  - Uses JPQL instead
- Heavy focus on POJOs

Providers:

Since JPA is an interface for a specific provider there are quite a few implementations to choose from and each of them have their own pros and cons

- Bitto
- Data Nucleus
- Eclipse Link (Formally Top link) : Was there away before java around 80's. Hibernate and JPA itself lot of the techniques and concepts from top link.
- ObjectDb

- Open JPA
- Versant
- Hibernate

You can choose the specific provider/vendor based of the business use case you are dealing with.

History of Hibernate

- **Debatable that Hibernate was the driving force behind JPA**
- **12 years ago sun/oracle rolled out the EJB specification and with EJB's they had a technique called CONTAINER MANAGEMENT PERSISTENCE.**
- **EJB was very painful and award to use. Almost moved away from good OO design especially when you interact with DB. (For every service you need to create a session bean and entity bean to interact)**
- **Eventually hibernate one out and people stop using the EJB and start using hibernate itself**
- **After sun rolled our the EJB they found JDO. Sun has claimed JDO was the driving force behind JPA. JDO was the sun's first one something like hibernate.**
- **They rolled JDO and marked it as wrong. Bi-code manipulation now we use the term Aspect oriented programming (AOP) .**
- **Top Link – Years before Java - Eclipse link – Is solid provider for people to not use eclipse link over hibernate.**
- **Why not use Hibernate directly or why should we use hibernate using JPA?**

You don't have to use Hibernate through ORM tool JPA it will work fine outside of ORM also. What if I want switch my provider. Do you really switched provider in your project , Yes. I have switched my providers on three different projects and made development. Due some performance problems.

If you use the provider through JPA it makes the transition very seamless. If I gone through use vendor specific annotations and implementation characteristics would have been a very difficult task but since I have used JPA it was very easy transition.

Hibernate History

- Debatable that Hibernate was the driving force behind Jpa
  - Debate that JDO was a driving force
- What about Toplink (now EclipseLink)
  - Around long before Java
- Hibernate directly or JPA using Hibernate?
  - Do you ever really switch your provider?

What is an Entity?

What makes up an Entity?

- POJO
- Entity
  - @Entity
- ID
  - @Id

```
public class Activity {

    private String desc;

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

}
```

Jpa / Hibernate configuration

- LocalContainerEntityManagerFactoryBean
  - jpaPropertyMap
- *hibernate.dialect*
- *hibernate.format_sql*
- *hibernate.hbm2ddl.auto*
  - *create*
  - *create-drop*
  - *update*
  - *validate*
  - *none*

```
<property name="jpaPropertyMap">
    <map>
        <entry key="hibernate.dialect" value="org.hibernate.dialect.MySQLInnoDBDialect"/>
        <entry key="hibernate.hbm2ddl.auto" value="none" />
        <entry key="hibernate.format_sql" value="true" />
    </map>
</property>
```

Dialect – What type of vendor database

Format_sql: Good formatting in our log.

Hbm2ddl.auto:

Create – create a database entities for us

Create – drop: creates when you start app and drops when you stops your app

Validate- validate will go through all the entities and verify with the java objects aligned. Typically use in Production.

GoalController.java | index.jsp | web.xml | Goal.java

```java
package com.pluralsight.model;

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

import org.hibernate.validator.constraints.Range;


@Entity
public class Goal {

    @Id
    @GeneratedValue
    private Long id;

    @Range(min = 1, max = 120)
    private int minutes;

    public Long getId() {
        return id;
    }

    public int getMinutes() {
        return minutes;
    }

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

    public void setMinutes(int minutes) {
        this.minutes = minutes;
    }

}
```

```xml
<property name="jpaPropertyMap">
    <map>
        <entry key="hibernate.dialect"
            value="org.hibernate.dialect.OracleDialect" />
        <entry key="hibernate.hbm2ddl.auto" value="create"></entry>
        <!-- interacting with ddl with ur database needs to work, values will
            discuss later -->
        <entry key="hibernate.format_sql" value="true" />
        <!-- format_sql for debugging purpose, These are all the properties needed
            for jpa to be setup with our local container entity manager -->
    </map>
</property>
```

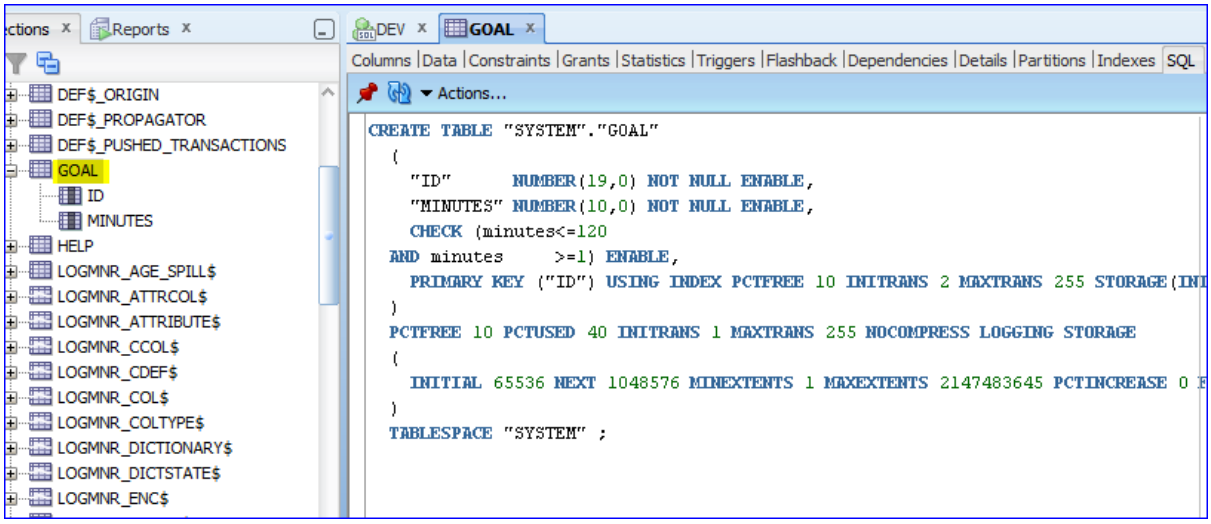
Please note that the hbm2ddl auto value create actually creates the database table. Please find the tomcat log below

```
INFO: Starting Servlet Engine: Apache Tomcat/8.0.32
Nov 09, 2019 7:27:06 PM org.apache.catalina.core.StandardContext checkUnusualURLPattern
INFO: Suspicious url pattern: "/pdfs/**" in context [/FitnessTracker] - see sections 12.1 and 12.2 of the Servlet specification
Nov 09, 2019 7:27:06 PM org.apache.catalina.core.StandardContext checkUnusualURLPattern
INFO: Suspicious url pattern: "/images/**" in context [/FitnessTracker] - see sections 12.1 and 12.2 of the Servlet specification
Nov 09, 2019 7:27:07 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs
Nov 09, 2019 7:27:07 PM org.apache.catalina.core.ApplicationContext log
INFO: No Spring WebApplicationInitializer types detected on classpath
Nov 09, 2019 7:27:07 PM org.apache.catalina.core.ApplicationContext log
INFO: Initializing Spring root WebApplicationContext
Nov 09, 2019 7:27:07 PM org.springframework.web.context.ContextLoader initWebApplicationContext
INFO: Root WebApplicationContext: initialization started
Nov 09, 2019 7:27:08 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing Root WebApplicationContext: startup date [Sat Nov 09 19:27:08 IST 2019]; root of context hierarchy
Nov 09, 2019 7:27:09 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [jpaContext.xml]
Nov 09, 2019 7:27:12 PM org.springframework.jdbc.datasource.DriverManagerDataSource setDriverClassName
INFO: Loaded JDBC driver: oracle.jdbc.driver.OracleDriver
Nov 09, 2019 7:27:13 PM org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean createNativeEntityManagerFactory
INFO: Building JPA container EntityManagerFactory for persistence unit 'punit'
Nov 09, 2019 7:27:15 PM org.hibernate.annotations.common.Version <clinit>
INFO: HCANN000001: Hibernate Commons Annotations {4.0.1.Final}
Nov 09, 2019 7:27:15 PM org.hibernate.Version logVersion
INFO: HHH000412: Hibernate Core {4.1.9.Final}
Nov 09, 2019 7:27:15 PM org.hibernate.cfg.Environment <clinit>
INFO: HHH000206: hibernate.properties not found
Nov 09, 2019 7:27:15 PM org.hibernate.cfg.Environment buildBytecodeProvider
INFO: HHH000021: Bytecode provider name : javassist
Nov 09, 2019 7:27:15 PM org.hibernate.ejb.Ejb3Configuration configure
INFO: HHH000204: Processing PersistenceUnitInfo [
    name: punit
    ...]
```

```
Nov 09, 2019 7:27:18 PM org.hibernate.service.jdbc.connections.internal.ConnectionProviderInitiator instantiateExplicitConnectionProvider
INFO: HHH000130: Instantiating explicit connection provider: org.hibernate.ejb.connection.InjectedDataSourceConnectionProvider
Nov 09, 2019 7:27:24 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.OracleDialect
Nov 09, 2019 7:27:24 PM org.hibernate.dialect.Oracle9Dialect <init>
WARN: HHH000063: The Oracle9Dialect dialect has been deprecated; use either Oracle9iDialect or Oracle10gDialect instead
Nov 09, 2019 7:27:24 PM org.hibernate.dialect.OracleDialect <init>
WARN: HHH000064: The OracleDialect dialect has been deprecated; use Oracle8iDialect instead
Nov 09, 2019 7:27:24 PM org.hibernate.engine.transaction.internal.TransactionFactoryInitiator initiateService
INFO: HHH000268: Transaction strategy: org.hibernate.engine.transaction.internal.jdbc.JdbcTransactionFactory
Nov 09, 2019 7:27:24 PM org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory <init>
INFO: HHH000397: Using ASTQueryTranslatorFactory
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Nov 09, 2019 7:27:28 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000227: Running hbm2ddl schema export
Hibernate:
    drop table Goal cascade constraints
Nov 09, 2019 7:27:28 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: HHH000389: Unsuccessful: drop table Goal cascade constraints
Nov 09, 2019 7:27:28 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: ORA-00942: table or view does not exist

Hibernate:
    drop sequence hibernate_sequence
Nov 09, 2019 7:27:28 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: HHH000389: Unsuccessful: drop sequence hibernate_sequence
Nov 09, 2019 7:27:28 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: ORA-02289: sequence does not exist
```

```
ERROR: ORA-02289: sequence does not exist

Hibernate:
    create table Goal (
        id number(19,0) not null,
        minutes number(10,0) not null check (minutes<=120 AND minutes>=1),
        primary key (id)
    )
Hibernate:
    create sequence hibernate_sequence
Nov 09, 2019 7:27:29 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
Nov 09, 2019 7:27:30 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@621a1137:
Nov 09, 2019 7:27:30 PM org.springframework.web.context.ContextLoader initWebApplicationContext
INFO: Root WebApplicationContext: initialization completed in 22791 ms
Nov 09, 2019 7:27:30 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-6061"]
Nov 09, 2019 7:27:30 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Nov 09, 2019 7:27:30 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 35925 ms
```

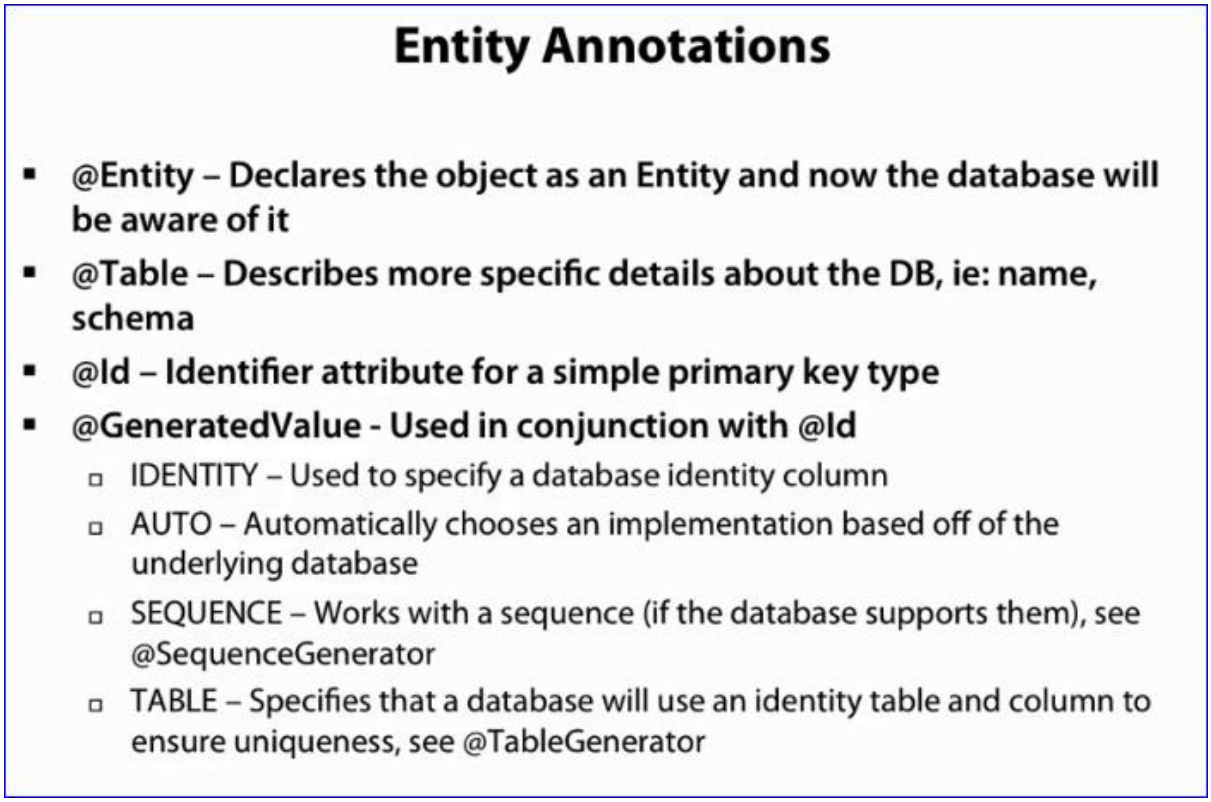
Could see GOAL table created in database



JPA ANNOTATIONS AND HOW TO USE THEM

JPA Annotations:

- **Annotations are a very powerful method of exposing functionality while staying with in domain. What I mean by that is I can mark up the object with the functionality what I wanted to have.**
- **Annotations aren't the only way to develop JPA. We can use ORM.xml files or hibernate hbm.xml files and bind classes to tables. But the industry trend currently is to do the development through annotations.**



Generated Value Types:

IDENTITY: is to specify an identity column in the database. These can be a little problematic because these can be available return back to user and tell after transaction made. Also be a little slower because they can't pre allocate id's for inserts. The Auto increment field in my sequence is an example an IDENTITY column. Although it can be little problematic they simpler to use and wiring your bean up and say go just persist something.

AUTO: Default's to IDENTITY if available auto incrementing field on the data base vendor it will choose that one. Oracle vendor don't have an automatic incremental field you have to use sequence and so it is not going to default to Identity.

SEQUENCE: It works with a sequence if the database supports them. Mysql doesn't support sequence itself. But Oracle or DB2 supports sequence. Use plugin @SequenceGenerator annotation.

TABLE: Table work with all implementations of database vendors. @TableGenerator

Table Annotation Demo:

```
@Entity
@Table(name = "goals") //Not exactly needed have the same name as class name
public class Goal {
```

```
<property name="jpaPropertyMap">
    <map>
        <entry key="hibernate.dialect"
            value="org.hibernate.dialect.OracleDialect" />
        <entry key="hibernate.hbm2ddl.auto" value="create"></entry>
        <!-- interacting with ddl with ur database needs to work, values will
```

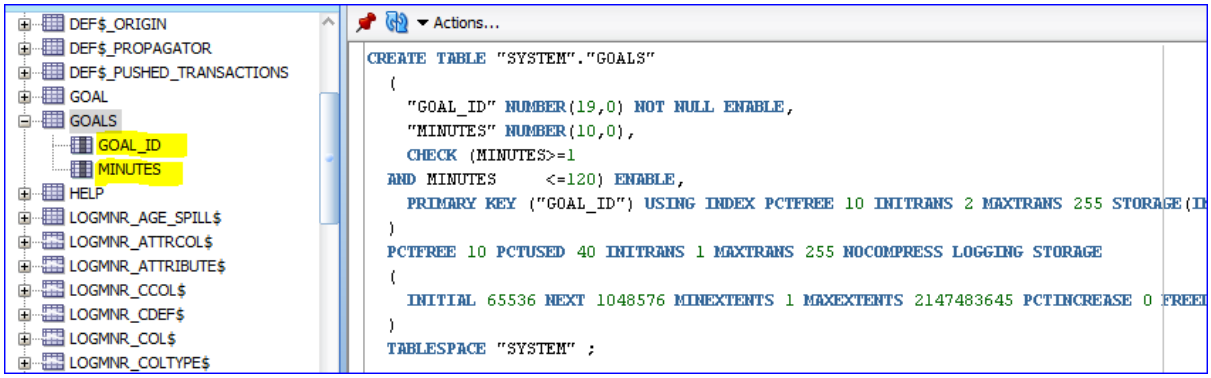
```
Hibernate:
    drop sequence hibernate_sequence
Hibernate:
    create table goals (
        id number(19,0) not null,
        minutes number(10,0) not null check (minutes>=1 AND minutes<=120),
        primary key (id)
    )
Hibernate:
    create sequence hibernate_sequence
```

Default Columns

- We don't have to accept the defaults that are created for us
- @Column will allow us to override column names or add
- descriptive information about it
  - columnDefinition
  - insertable
  - length
  - name
  - nullable
  - precision
  - scale
  - table
  - unique
  - updatable

```java
@Id
@GeneratedValue
@Column(name = "GOAL_ID") //we can combine annotaions and override the database column name
private Long id;

@Range(min = 1, max = 120)
@Column(name="MINUTES") //CAN HAVE UPPERCASE
private int minutes;
```

```
Hibernate:
    drop table goals cascade constraints
Hibernate:
    drop sequence hibernate_sequence
Hibernate:
    create table goals (
        GOAL_ID number(19,0) not null,
        MINUTES number(10,0) check (MINUTES>=1 AND MINUTES<=120),
        primary key (GOAL_ID)
    )
Hibernate:
    create sequence hibernate_sequence
Nov 09, 2019 11:13:44 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
```



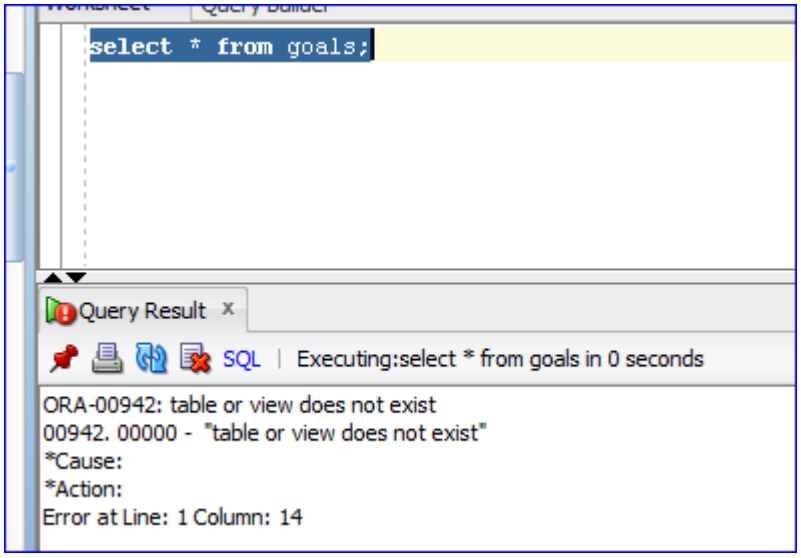
Create – Drop: hbm2ddl value actually drops the table only on server stop not the schema. But in documentation mentioned it drops complete schema. That is wrong.

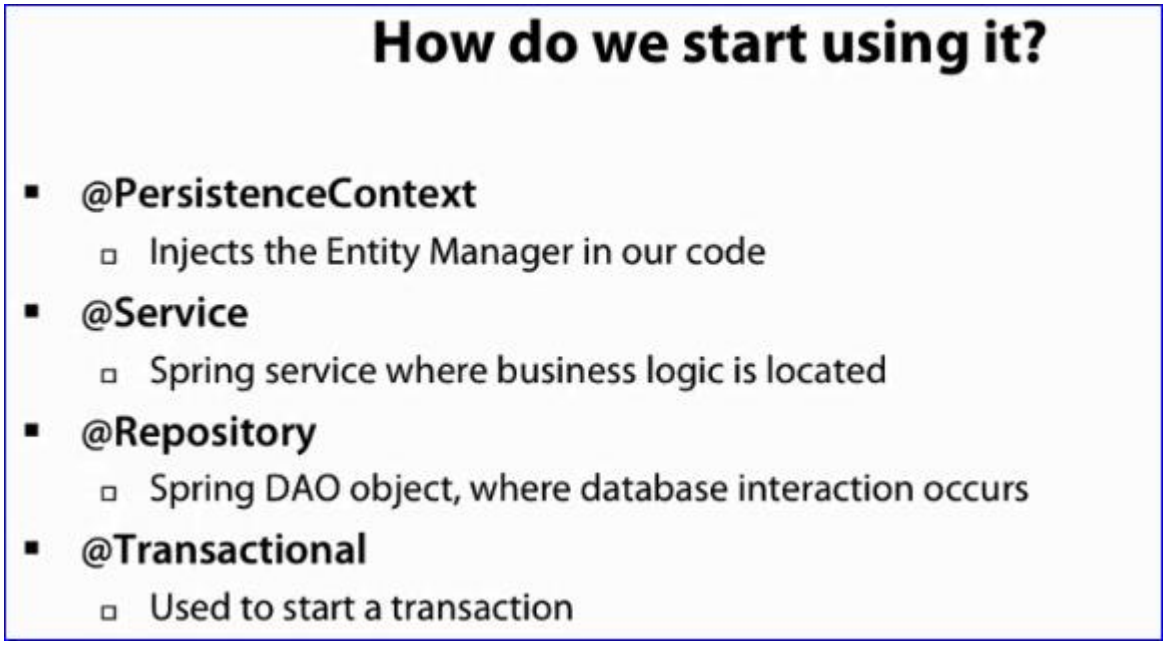
On Server start it created the table

```
Nov 09, 2019 11:20:15 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000227: Running hbm2ddl schema export
Hibernate:
    drop table goals cascade constraints
Hibernate:
    drop sequence hibernate_sequence
Hibernate:
    create table goals (
        GOAL_ID number(19,0) not null,
        MINUTES number(10,0) check (MINUTES>=1 AND MINUTES<=120),
        primary key (GOAL_ID)
    )
Hibernate:
    create sequence hibernate_sequence
Nov 09, 2019 11:20:15 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
```

On server stop it drops the table not the entire schema.

```
Nov 09, 2019 11:23:35 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000227: Running hbm2ddl schema export
Hibernate:
    drop table goals cascade constraints
Hibernate:
    drop sequence hibernate_sequence
Nov 09, 2019 11:23:35 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
```





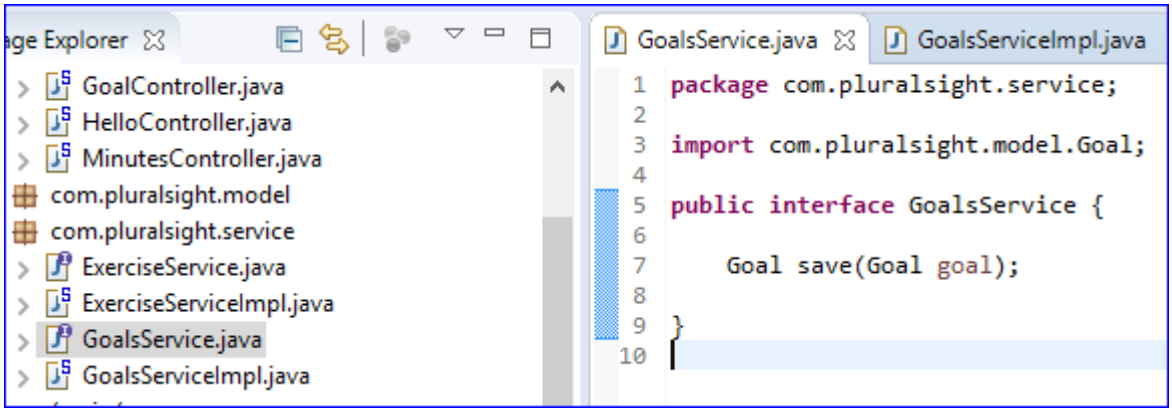
How to start using all the JPA annotations with spring.?

- **First thing is to inject the Entity Manager into our spring code. The way we do that using @PersistenceContext annotation. It actually grabs an instance of Entity manager and injects into our code. It calls persistence Context instead of entity manager because it's pertaining to a specific unit or persistence context. If you remember we created a entity manager factory tied to a persistence unit. That is why it is not called as @EntityManager.**
- **@Service: Where our business logic is located.**
- **@Repository: is kind of synonymous with spring DAO object. Where our database interaction occurs.**
- **Last thing @Transactional used to start a transaction**

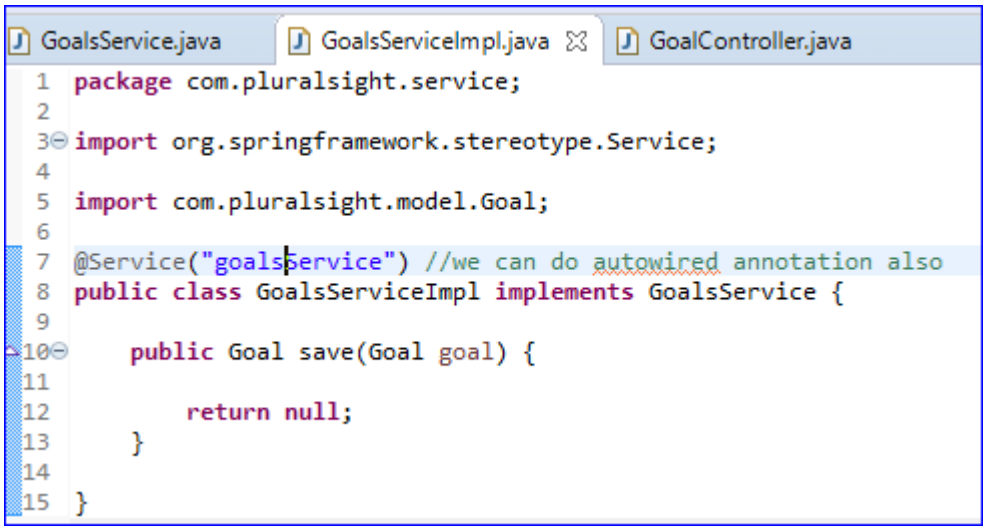
SERVICE CONFIGURATION:

Step 1:

Create a service interface and a simple save method.



Step 2: Create an implementation class for the interface.



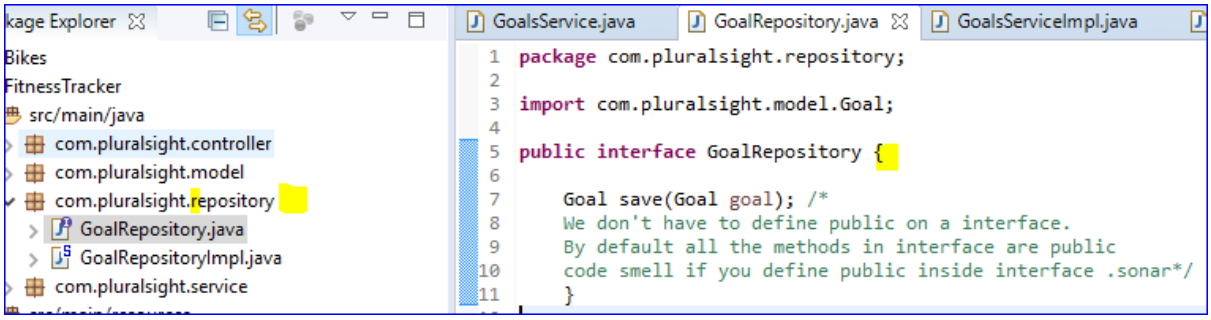
Notice you should define the impl class with @Service annotation. With the service name

Step 3: Go to GoalsController.java and @Autowire the goal service . In post method add else block to save the goal.

```java
17 @Controller
18 @SessionAttributes("goal")
19 public class GoalController {
20
21     @Autowired
22     private GoalsService goalsService;
23
24     @RequestMapping(value = "addGoal", method = RequestMethod.GET)
25     public String addGoal(Model model) {
26         Goal goal = new Goal();
27         goal.setMinutes(10);
28         model.addAttribute("goal", goal);
29
30         return "addGoal";
31     }
32
33     @RequestMapping(value = "addGoal", method = RequestMethod.POST)
34     public String updateGoal(@Valid @ModelAttribute("goal") Goal goal, BindingResult result) {
35
36         System.out.println("result has errors: " + result.hasErrors());
37
38         System.out.println("Goal set: " + goal.getMinutes());
39
40         if(result.hasErrors()) {
41             return "addGoal";
42         } else {
43             goalsService.save(goal);
44         }
45
46         return "redirect:index.jsp";
47     }
48
49 }
```

REPOSITORY CONFIGURATION:

1. **Define a package for repository in src/main/java – com.pluralsight.repository**
2. **Create a GoalRepository interface inside new package.**



3. **Create a class GoalRepositoryImpl implementing GoalRepository interface**

GoalRepositoryImpl.java | GoalsService.java | GoalRepository.java | GoalController.jav

```java
package com.pluralsight.repository;

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

import org.springframework.stereotype.Repository;

import com.pluralsight.model.Goal;

@Repository("goalRepository")
public class GoalRepositoryImpl implements GoalRepository {

    //inject persistence context
    @PersistenceContext
    private EntityManager entityManager;

    public Goal save(Goal goal) {

        entityManager.persist(goal);
        //persist returns void.

        return goal;
        //For demo purpose just returning goal object
    }

}
```

4. **Tie the repository into service tier**

GoalRepositoryImpl.java | GoalsServiceImpl.java | GoalsService.java | GoalRepository.java

```java
package com.pluralsight.service;

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

@Service("goalsService") //we can do autowired annotation also
public class GoalsServiceImpl implements GoalsService {

    @Autowired
    private GoalRepository goalRepository;

    public Goal save(Goal goal) {

        return goalRepository.save(goal);
    }

}
```

Right now the example is little simple you were wondering why I have Goalservice interface that call save that goes to Goal Serivce impl and that calls Goal Repository that calls Goal Repository impl to persist/save to database.

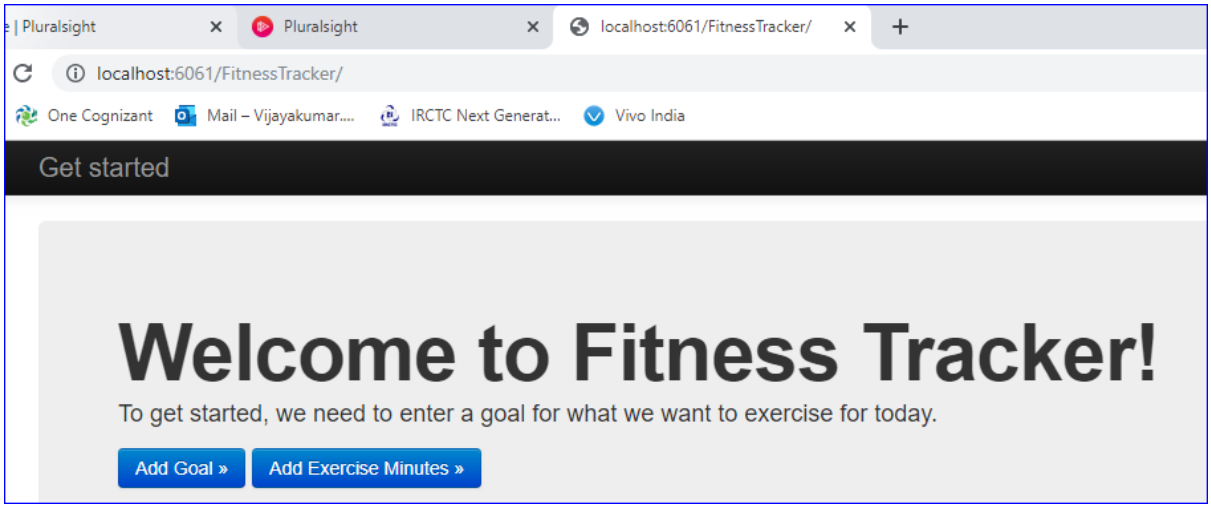
Why have those 3 tiers. We could have more business logic wrapping our save service. Maybe we have to call a webservice to check the goal is valid or not. If not don't allow to save. We will put the all business logic in service impl not in repository. That Is why we have 2 different tiers. Service tier and Repository tier.

And there is no reason your controller directly call the repository save method if there is no business logic wrapped around in the service class.

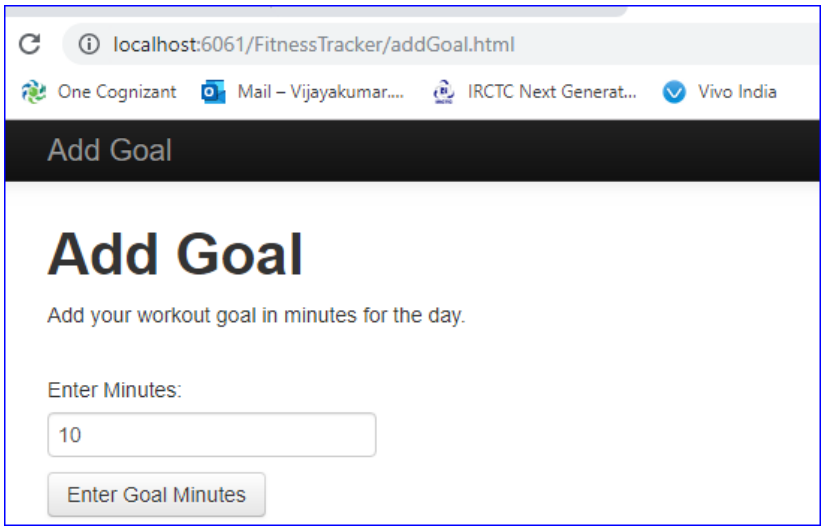
END – To – END Test:

Everything gets wired up and run through the code. There is a bug in our code. Let's look at it. It doesn't give any error message. But the object won't persist to database.

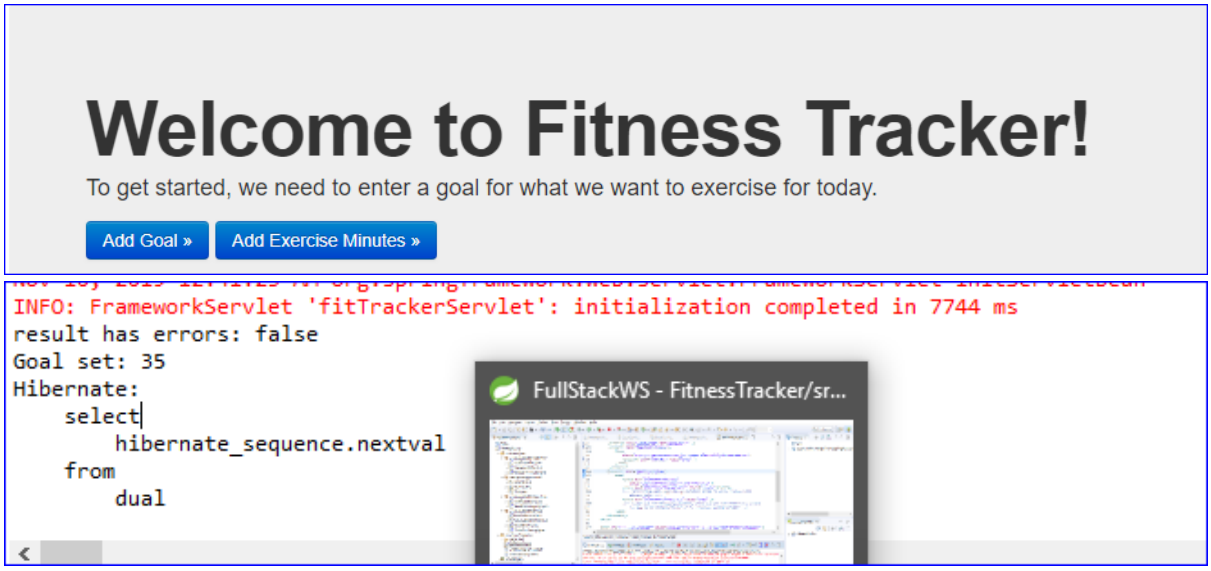
Just run the tomcat server. Run through the application.



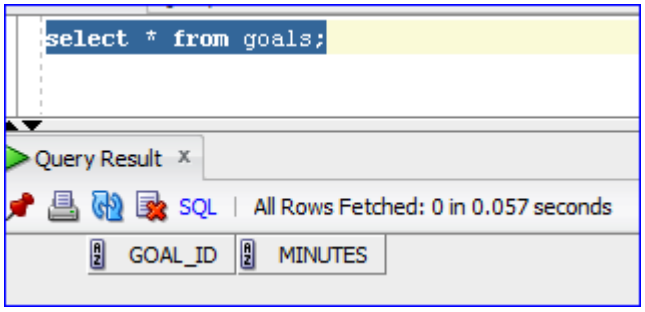
Add Goal



Enter Goal Minutes. : Returns to index.jsp . All looks good.

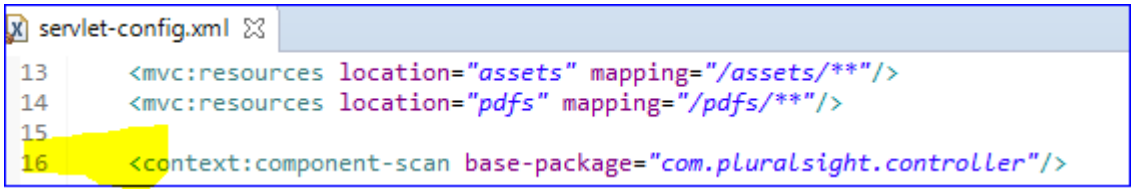


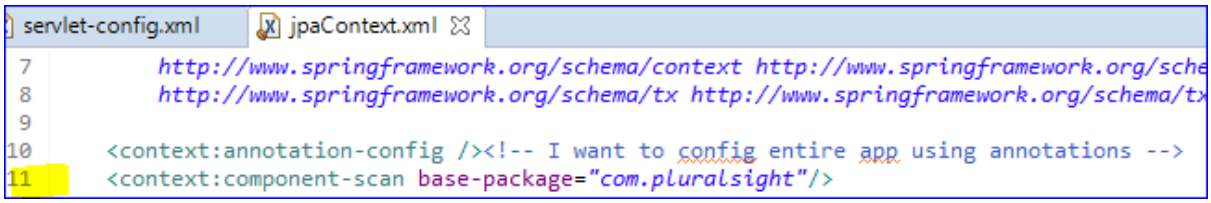
Now look at the Db. Nothing saved in there



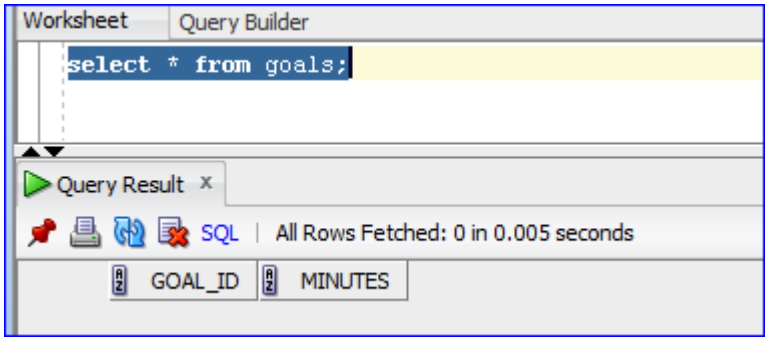
The issue is with the jpaContext.xml : We have defined annotation-driven context tag But we didn't define where to scan for the various annotations.

But why it the program running with out exceptions?
The reason is in servlet-config.xml we have defined component – scan for complete com.pluralsight package. Now change that to scan only controller. And add the component scan to jpa context.





Now run the application again.



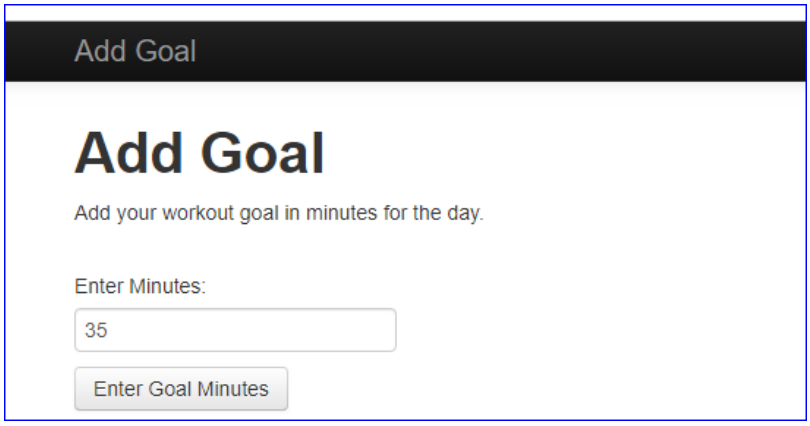
It still not saved?

Why?

JPA doesn't save to database until we do a flush

```
public Goal save(Goal goal) {

    entityManager.persist(goal);
    //persist returns void.

    entityManager.flush();
```

Do the same add minutes from application



HTTP Status 500 - Request processing failed; neste transaction is in progress

type Exception report

message Request processing failed; nested exception is javax.persistence.TransactionRequiredException:

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

exception

```
org.springframework.web.util.NestedServletException: Request processing failed;
        org.springframework.web.servlet.FrameworkServlet.processRequest(Framewor
        org.springframework.web.servlet.FrameworkServlet.doPost(FrameworkServlet
        javax.servlet.http.HttpServlet.service(HttpServlet.java:648)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServle
        javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

root cause

```
javax.persistence.TransactionRequiredException: no transaction is in progress
        org.hibernate.ejb.AbstractEntityManagerImpl.flush(AbstractEntityManager
```

We are getting this exception because we never use transaction in the program. So let's define the transaction for save method in service impl.

servlet-config.xml | jpaContext.xml | GoalRepositoryImpl.java | GoalsServiceImpl.java

```
1  package com.pluralsight.service;
2
3  import org.springframework.beans.factory.annotation.Autowired;
9
10 @Service("goalsService") //we can do autowired annotation also
11 public class GoalsServiceImpl implements GoalsService {
12
13     @Autowired
14     private GoalRepository goalRepository;
15
16     @Transactional
17     public Goal save(Goal goal) {
18
19         return goalRepository.save(goal);
20     }
21
22 }
```

Add minutes 35

```
select * from goals;
```

Query Result | SQL | All Rows Fetched: 1 in

	GOAL_ID	MINUTES
1	1	35

```
Goal set: 35
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        goals
        (MINUTES, GOAL_ID)
    values
        (?, ?)
```

@Transactional handles all boiler plate code like begin transaction and if any exception occurs do a rollback. But @Transactional annotation handles all this logic.

JOIN TYPES:

To deal with objects other than simple primitives like List etc. Since JPA dealing with objects we need to be able to bind collection of objects to the database.

We can achieve this sing joins.

Join Types

- Essentially four join types:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Can be used in various configurations:
  - Unidirectional
  - Bidirectional
  - Cascade

@OneToMany:

Most common of the join type annotation. In our application we have a goal and a collection of exercises to establish goal.

To achieve this we need add one to many annotation to our goal object/class and add to that list of exercises and map that back to the goal. mappedBy on it is referring to an owning object so the exercise is belongs to this goal. SO this goal has many of these exercise.

Annotate Entity class

```
@Entity
public class Exercise {

    @Id
    @GeneratedValue
    private Long id;


    @Range(min = 1, max = 120)
    private int minutes;
```

One – To- Many Demo:

Let's do one to many mapping from goal to exercise and Many to One mapping from exercise to Goal objects so that the collection is persisted.

```
@OneToMany(mappedBy = "goal", cascade = CascadeType.ALL)//As of now it is unidirectional
//Goal is mapped to Excercise object.
private List<Exercise> exercises = new ArrayList<Exercise>();
```

Now this setup becomes bi directional I can access the goal with owning the exercise . From goal we can have list of exercises.

```
@Entity
public class Exercise {

    @Id
    @GeneratedValue
    private Long id;


    @Range(min = 1, max = 120)
    private int minutes;

    @NotNull
    private String activity;

    @ManyToOne
    private Goal goal;
```

The app is setup right now we have a goal that's going to create an exercise and add that goal we going to associate with that goal.

Now setup the exercise controller.

To MinuteController and add HTTP Session which will inject he session object. Get the goal session attribute

```
@RequestMapping(value = "/addMinutes",  method = RequestMethod.POST)
public String addMinutes(@Valid @ModelAttribute ("exercise") Exercise exercise,
        HttpSession session,
        BindingResult result) {

    System.out.println("exercise: " + exercise.getMinutes());
    System.out.println("exercise activity: " + exercise.getActivity());

    if(result.hasErrors()) {
        return "addMinutes";
    } else {
        Goal goal = (Goal)session.getAttribute("goal");
    }

    return "addMinutes";
}
```

The goal is in session as below. The goal is stored in a session and we are accessing the session from that. (Goal) session.getAttribute("goal");

```java

@Controller
@SessionAttributes("goal")
public class GoalController {

    @Autowired
    private GoalsService goalsService;

    @RequestMapping(value = "addGoal", method = RequestMethod.GET)
    public String addGoal(Model model) {
        Goal goal = new Goal();
        goal.setMinutes(10);
        model.addAttribute("goal", goal);

        return "addGoal";
    }

    @RequestMapping(value = "addGoal", method = RequestMethod.POST)
    public String updateGoal(@Valid @ModelAttribute("goal") Goal goal, BindingResult result) {

        System.out.println("result has errors: " + result.hasErrors());

        System.out.println("Goal set: " + goal.getMinutes());

        if(result.hasErrors()) {
            return "addGoal";
        } else {
            goalsService.save(goal);
        }
```

Next to call the save method in the exerciseservice in MinutesController.java : Do follow the subsequent coding. Service – serviceimpl – repository – repositotyimpl

```java
    @RequestMapping(value = "/addMinutes",  method = RequestMethod.POST)
    public String addMinutes(@Valid @ModelAttribute ("exercise") Exercise exercise,
            HttpSession session,
            BindingResult result) {

        System.out.println("exercise: " + exercise.getMinutes());
        System.out.println("exercise activity: " + exercise.getActivity());

        if(result.hasErrors()) {
            return "addMinutes";
        } else {
            Goal goal = (Goal)session.getAttribute("goal");
            exercise.setGoal(goal);
            exerciseService.save(exercise);
        }

        return "addMinutes";
    }
```

```java
package com.pluralsight.service;

import java.util.List;

public interface ExerciseService {

    List<Activity> findAllActivities();

    Exercise save(Exercise exercise);

}
```

```java
@Service("exerciseService")
public class ExerciseServiceImpl implements ExerciseService {

    @Autowired
    private ExerciseRepository exerciseRepository;

    @Transactional
    public Exercise save(Exercise exercise) {

        exercise = exerciseRepository.Save(exercise);

        return exercise;
    }
```

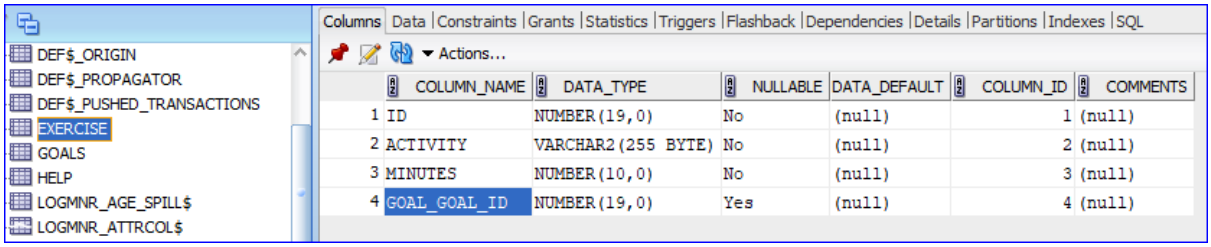
```java
package com.pluralsight.repository;

import com.pluralsight.model.Exercise;

public interface ExerciseRepository {

    Exercise Save(Exercise exercise);
}
```

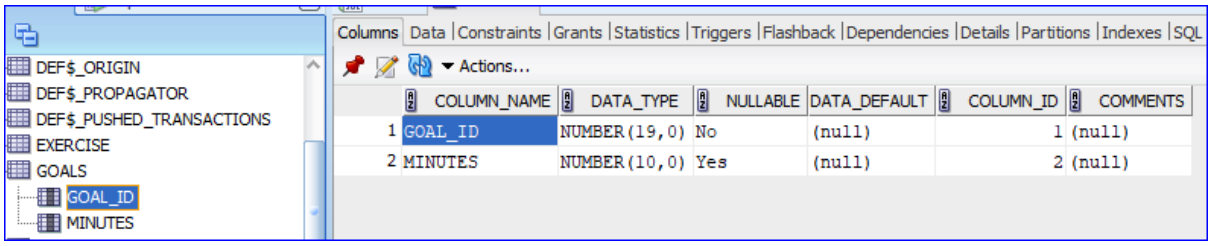
```java
package com.pluralsight.repository;

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

import org.springframework.stereotype.Repository;

import com.pluralsight.model.Exercise;

@Repository("exerciseRepository")
public class ExerciseRepositoryImpl implements Ex


    @PersistenceContext
    private EntityManager entityManager;


    public Exercise Save(Exercise exercise) {

        entityManager.persist(exercise);
        entityManager.flush();

        return exercise;
    }
```

Run the application

```
Hibernate:
    drop table Exercise cascade constraints
Nov 10, 2019 3:28:31 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: HHH000389: Unsuccessful: drop table Exercise cascade constraints
Nov 10, 2019 3:28:31 PM org.hibernate.tool.hbm2ddl.SchemaExport perform
ERROR: ORA-00942: table or view does not exist

Hibernate:
    drop table goals cascade constraints
Hibernate:
    drop sequence hibernate_sequence
Hibernate:
    create table Exercise (
        id number(19,0) not null,
        activity varchar2(255) not null,
        minutes number(10,0) not null check (minutes<=120 AND minutes>=1),
        goal_GOAL_ID number(19,0),
        primary key (id)
    )
Hibernate:
    create table goals (
        GOAL_ID number(19,0) not null,
        MINUTES number(10,0) check (MINUTES>=1 AND MINUTES<=120),
        primary key (GOAL_ID)
    )
Hibernate:
    alter table Exercise
        add constraint FK7E6B65F8D41F62F5
        foreign key (goal_GOAL_ID)
        references goals
Hibernate:
    create sequence hibernate_sequence
Nov 10, 2019 3:28:35 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
```

See the table drop and create . Exercise table and its constraints FK , PK.



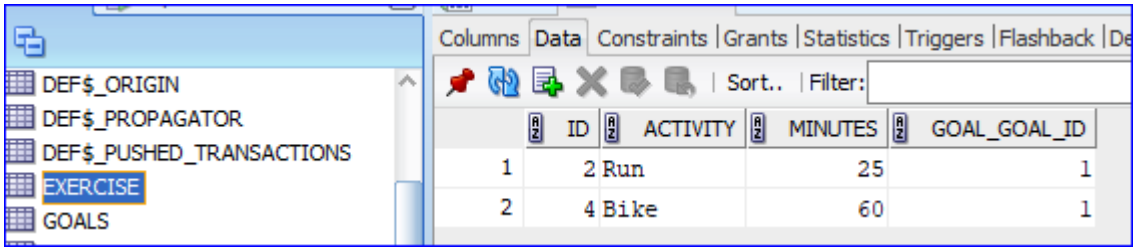


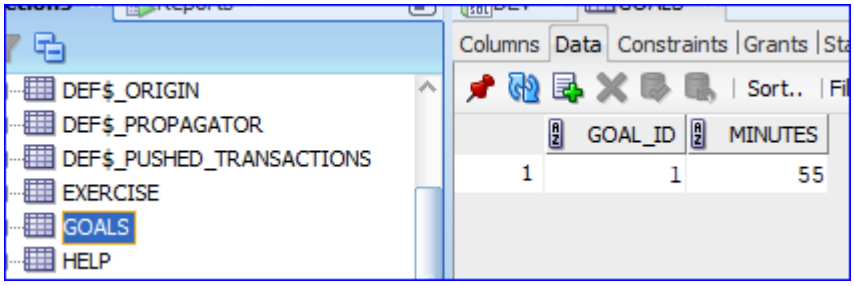
The FR created as GOAL_GOAL_ID. This notation comes as, <entity class name>_<entity PK Column Name>

```
@Table(name = "goals") //Not exa
public class Goal {

    @Id
    @GeneratedValue
    @Column(name = "GOAL_ID") //
    private Long id;
```

```
Goal set: 55
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        goals
        (MINUTES, GOAL_ID)
    values
        (?, ?)
exercise: 25
exercise activity: Run
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        Exercise
        (activity, goal_GOAL_ID, minutes, id)
    values
        (?, ?, ?, ?)
exercise: 25
exercise activity:
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
```

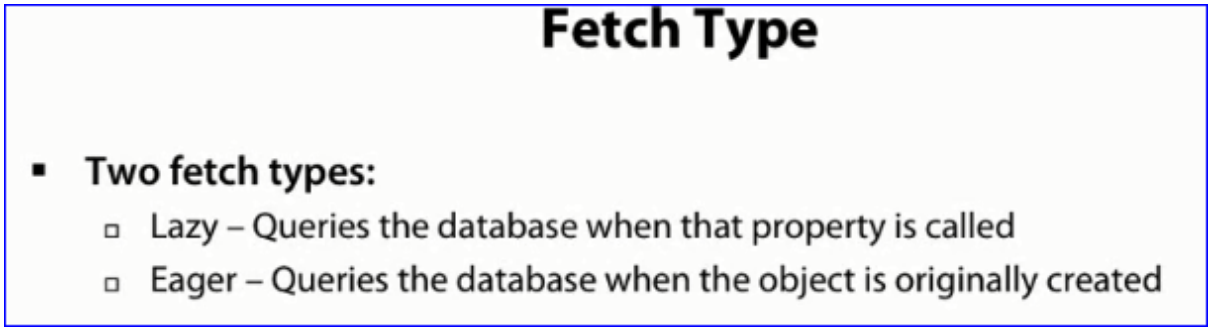
```
exercise: 60
exercise activity: Bike
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        Exercise
        (activity, goal_GOAL_ID, minutes, id)
    values
        (?, ?, ?, ?)
```





FETCH TYPE

When we use one to many or many to one annotations we can choose what time we want to fetch the data.



Lazy: when I call getExercises on goal object its going to then go out and populate the collection. It will wait and tell to call that getter to query the database.

Eager: Queries the database when the object is created. Hibernate has limit the fetch type with to eager types. No difference only the implementation is different.

Let's look at what it means in our database

```
@OneToMany(mappedBy="goal", cascade=CascadeType.ALL, fetch=FetchType.LAZY)
private List<Exercise> exercises = new ArrayList<Exercise>();
```

JPQL : java persistence query Language

JPQL is not SQL.

Cantered around objects. Instead of tables it will query with the entity object namges.

JPQL Ex: select g from Goal g; -- here Goal is case sensitive should match with entity name.

SQL Ex: select * from GOALS;

Both of these queries are valid against a database. JPQL uses Object name, SQL uses table name

You can't say select * from GOALS in JPQL>

JPQL Demo:

Let's go ahead and create a simple reporting page. Show cascading features of our collection object.

Create a new jsp getGoals.jsp inside WEB-INF/jsp/

Exercise.java | Goal.java | getGoals.jsp | GoalController.java | MinutesCo

```jsp
<%@ 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>
<html>
<head><meta charset="ISO-8859-1"><title>Goals Report</title>
</head>
<body>
    <table>
        <tr>
            <th>ID</th>
            <th>Minutes</th>
        </tr>
        <c:forEach items="${goals}" var="goal">
            <tr>
                <td>${goal.id}</td>
                <td>${goal.minutes}</td>
            </tr>
            <td>
                <table>
                    <tr>
                        <th>Exercise ID</th>
                        <th>Exercise Minutes</th>
                        <th>Exercise Activity</th>
                    </tr>
                    <c:forEach items="${goal.exercises}" var="exercises">
                        <tr>
                            <td>${exercises.id}</td>
                            <td>${exercises.minutes}</td>
                            <td>${exercises.activity}</td>
                        </tr>
                    </c:forEach>
                </table>
            </td>
        </c:forEach>
    </table>
</body>
</html>
```

Currently we don't have ${goals} and ${goal.exercises} are saved on session. Let's do the changes to controller class to make them available on session.

GoalController

```java
51    @RequestMapping(value = "getGoals", method = RequestMethod.GET)
52    public String getGoals(Model model) {
53
54        List<Goal> goals = goalsService.findAllGoals();
55        model.addAttribute("goals", goals);
56
57        return "getGoals";
58
59    }
60
61 }
```

```java
    List<Goal> findAllGoals();

}
```

```java
18    @Transactional
19    public Goal save(Goal goal) {
20
21        return goalRepository.save(goal);
22    }
23
24    public List<Goal> findAllGoals() {
25
26        return goalRepository.findAll();
27    }
28
29 }
```

```java
31    @SuppressWarnings({ "rawtypes", "unchecked" })
32    //We are adding suppress warnings to skip the getResultList returns untyped list
33    //But we are returning List<Goal> type. To fix add supress warning
34    public List<Goal> findAll() {
35
36        Query query = entityManager.createQuery("Select g from Goal g");
37
38        List goals = query.getResultList();
39
40        return goals;
41
42    }
```

JPQL

```java
31    @OneToMany(mappedBy = "goal", cascade = CascadeType.ALL, fetch=FetchType.EAGER)/
32    //Goal is mapped to Excercise object.
33    private List<Exercise> exercises = new ArrayList<Exercise>();
```

All setup done. Now run the application and add goal followed by minutes. Access the getGoals.html page.

localhost:6061/FitnessTracker/getGoals.html

Apps | One Cognizant | Mail – Vijayakumar.... | IRCTC Next Generat... | Vivo India

ID	Minutes
1	55

Exercise ID	Exercise Minutes	Exercise Activity
2	15	Run
3	25	Bike
4	15	Swim

```
Hibernate:
    select
        goal0_.GOAL_ID as GOAL1_0_,
        goal0_.MINUTES as MINUTES0_
    from
        goals goal0_
Hibernate:
    select
        exercises0_.goal_GOAL_ID as goal4_0_1_,
        exercises0_.id as id1_1_,
        exercises0_.id as id1_0_,
        exercises0_.activity as activity1_0_,
        exercises0_.goal_GOAL_ID as goal4_1_0_,
        exercises0_.minutes as minutes1_0_
    from
        Exercise exercises0_
    where
        exercises0_.goal_GOAL_ID=?
```

Now Change the fetch type to Lazy and access the getGoals.html page

```
31    @OneToMany(mappedBy = "goal", cascade = CascadeType.ALL, fetch=FetchType.LAZY)//A
32    //Goal is mapped to Excercise object.
33    private List<Exercise> exercises = new ArrayList<Exercise>();
```

localhost:6061/FitnessTracker/getGoals.html

Apps | One Cognizant | Mail – Vijayakumar.... | IRCTC Next Generat... | Vivo India

HTTP Status 500 - org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.pluralsight.model.Goal.exercises, could not initialize proxy - no Session

type Exception report

message org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.pluralsight.model.Goal.exercises, could not initialize proxy - no Session

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

exception

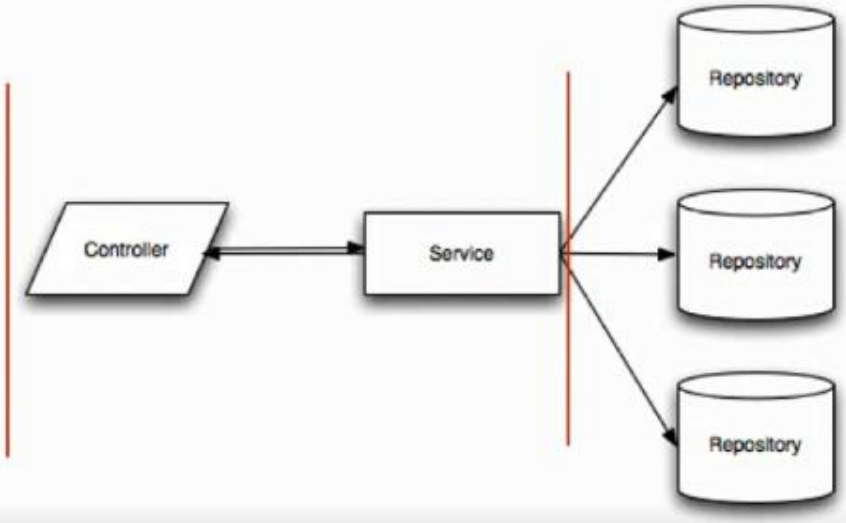
```
org.apache.jasper.JasperException: org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.pluralsight.model.Goal.exercis
        org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:555)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:476)
        org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:396)
        org.apache.jasper.servlet.JspServlet.service(JspServlet.java:340)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
```

Lets go walk through how you fix it. You have to do some minor setting to the application to handle LAZy.

LAZY initialization exception is very common exception. The way you fix this is very easy.

OpenEntityManagerInViewFilter

- Used to increate the length of our Jpa Session



OpenEntityManagerInViewFilter

```
<filter>
    <filter-name>Spring OpenEntityManagerInViewFilter</filter-name>
    <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
</filter>


<filter-mapping>
    <filter-name>Spring OpenEntityManagerInViewFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Use a class called OpenEntityManagerInViewFilter . It is web filter implement in our web.xml and how it fixing session being closed for a request response lifecycle.

With the Controller, service and Repository tier with the standard configuration the red line represents where our JPA session starts and stops. So it starts right after we call our service opens up a repository tier and then hands comeback to our service.

Don't confuse this with our web session. Infact it is really closer to our transaction life cycle . Using an Entity manager in view filter we can open and close our JPA session further up in the process to hear.

Why/when to use OpenEntityManagerInViewFilter : If you want to keep the session open in the jsp page little bit longer in order to load the object lazy.

OpenEntityManagerInViewFilter is an implementation of the OSIV pattern (Open Session In View) for JPA. (A Session is the native Hibernate version of an EntityManager. So for JPA it can be called: Open EntityManager In View pattern.) In this pattern, a single EntityManager is shared/reused within a single HTTP request, and closed when the HTTP request finishes.

The benefits of using the OSIV pattern are:

better database performance compared to when a single HTTP request causes multiple different transactions and entitymangers to be opened

allows JPA to use 1st level caching

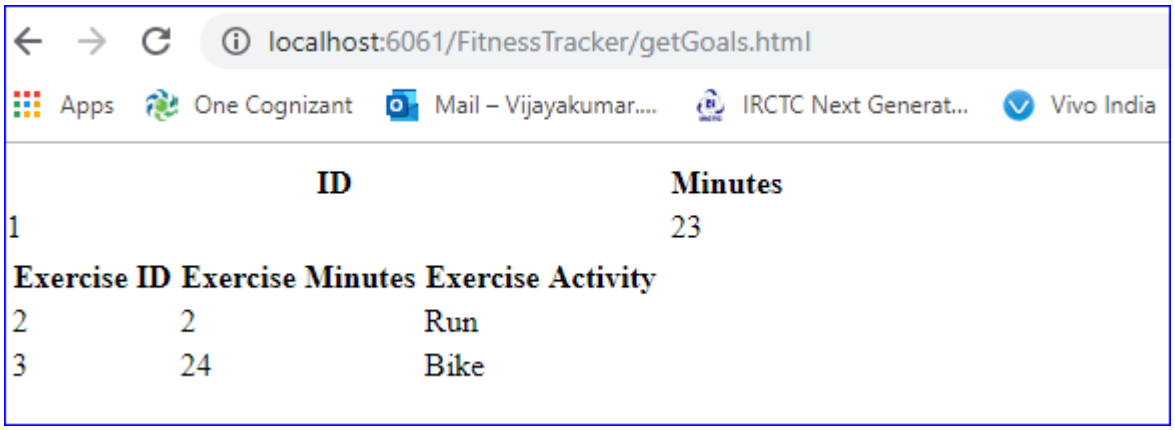
easier programming, you don't have to worry about LazyInitializationExceptions, which can be caused when the entityManager is closed and you access methods on entities that require a persistent context

it may be required when you're using custom Spring security filters that use JPA.

To fix this add below filter in web.xml

```
14    <filter>
15        <filter-name>SpringOpenEntityManagerInViewFilter</filter-name>
16        <filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>
17    </filter>
18
19    <filter-mapping>
20        <filter-name>SpringOpenEntityManagerInViewFilter</filter-name>
21        <url-pattern>/*</url-pattern>
22    </filter-mapping>
```

Start the server again. And hit getGoals.html. now it will work withour any exceptions.



Projection

It is not important to fetch all the fields from the database even if you don't want to show them on the jsp page.

You can restrict the number of fields to be fetched from database.

Using a technique called PROJECTION to present objects exactly what we want to show in UI.

- Great way to present objects to the UI
- Objects added using JPQL syntax
- Projection Objects can be Jpa Entities
- Need a constructor for the projection

```
String jpql = "Select new com.pluralsight.model.GoalReport(g.minutes, e.minutes, e.activity) " +
        "from Goal g, Exercise e where g.id = e.goal.id";
```

Projections resolve N+1 select problems.

Projections Demo

1. **Add GoalReport Model class with Constructor with required properties. Your requirement is to select defined properties in this model to UI.**

```
1  package com.pluralsight.model;
2
3  public class GoalReport {
4
5      private int goalMinutes;
6
7      private int exerciseMinutes;
8
9      private String exerciseActivity;
10
11     public GoalReport(int goalMinutes, int exerciseMinutes, String exerciseActivity) {
12         super();
13         this.goalMinutes = goalMinutes;
14         this.exerciseMinutes = exerciseMinutes;
15         this.exerciseActivity = exerciseActivity;
16     }
17
```

2. **Add a separate getGoalProjectionReport.jsp and display goals minutes, exercise minutes, exercise activity**

FullStackWS - FitnessTracker/src/main/webapp/WEB-INF/jsp/getGoalsProjectionReport.jsp - Spring Too

File Edit Source Refactor Navigate Search Project Run Window Help

GoalRepositor... | Goal.java | getGoalsProje... | web.xml | GoalRepo

```jsp
<%@ 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>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Goal Projection Report</title>
</head>
<body>
    <table>
        <tr>
            <th>GoalMinutes</th>
            <th>ExerMinutes</th>
            <th>ExerActivity</th>
        </tr>
        <c:forEach items="${goalReports}" var="goalReports">
            <tr>
                <td>${goalReports.goalMinutes}</td>
                <td>${goalReports.exerciseMinutes}</td>
                <td>${goalReports.exerciseActivity}</td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

Add controller Request mapping method.

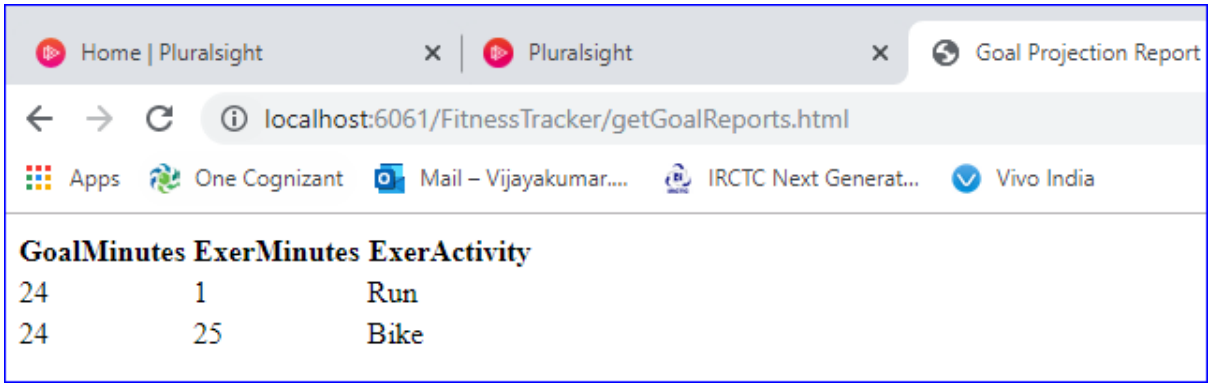
```java
    @RequestMapping(value="getGoalReports", method= RequestMethod.GET)
    public String getGoalReports(Model model) {

        List<GoalReport> goalReports = goalsService.getGoalReports();
        model.addAttribute("goalReports",goalReports );
        return "getGoalsProjectionReport";
    }

}
```

Implement code for service, impl, repo, repimpl

Please note the JPQL query for projection is called by a constructor argument.

```java
    @SuppressWarnings({ "rawtypes", "unchecked" })
    public List<GoalReport> getReports() {

        Query query = entityManager.createQuery("Select new com.pluralsight.model.GoalReport(g.minutes,e.minutes, e.activity) "
                + "from Goal g , Exercise e where g.id= e.goal.id)");
        List goalReports = query.getResultList();

        return goalReports;
    }
}
```

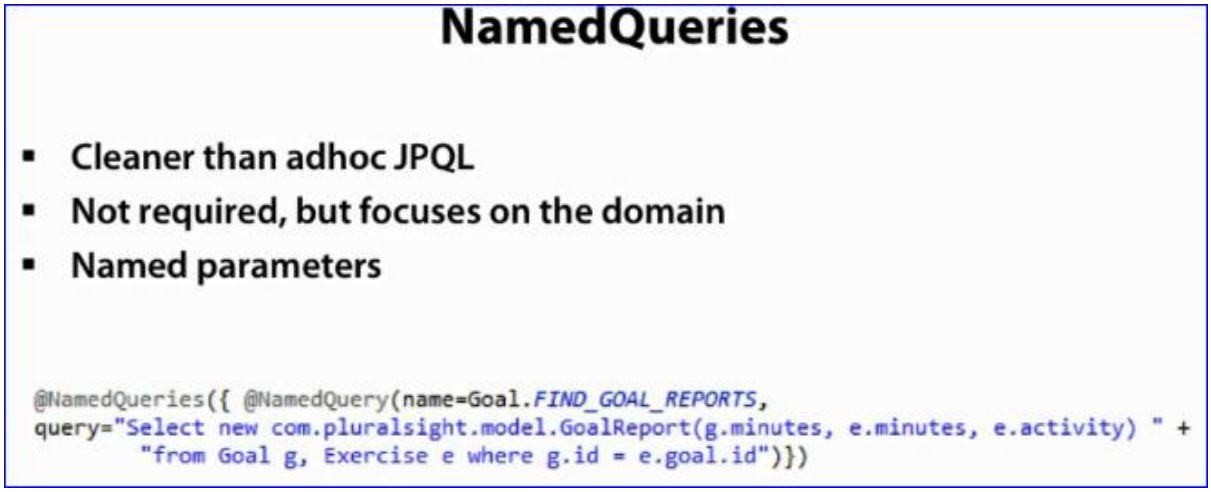
g.id and e.goal.id(it is a FK reference in exercise table) (Exercises Goal object id)



Named Queries:

We can clean up some of the adhoc JPQL queries. We store them in our domain object

We define the named query on top of the object in domain class.



First we need to name our named query in the domain class Goal

Name the query with static variable. Use @NamedQueries (array syntax) annotation can define n number of named queries.

```
20 @Entity
21 @Table(name = "goals") //Not exactly needed have the same name as class name\
22 @NamedQueries({
23     @NamedQuery(name=Goal.FIND_REPORTS, query="Select new com.pluralsight.model.GoalReport("
24             + "g.minutes,e.minutes, e.activity) from Goal g , Exercise e where g.id= e.goal.id)")
25 })
26 public class Goal {
27
28     public static final String FIND_REPORTS = "findGoalReports";
```

To read the named query we have to use TypeQuery<returntype obj>

```
46      //@SuppressWarnings({ "rawtypes", "unchecked" })
47      public List<GoalReport> getReports() {
48
49          /*Query query = entityManager.createQuery("Select new com.pluralsight.model.GoalReport(g.minutes,e.minutes, e.activity)
50                  + "from Goal g , Exercise e where g.id= e.goal.id)");*/
51          TypedQuery<GoalReport> query = entityManager.createNamedQuery(Goal.FIND_REPORTS,GoalReport.class);
52          List<GoalReport> goalReports = query.getResultList();
53
54          return goalReports;
55      }
56  }
```

We can remove suppressWarnings because the TYpeQuery will return a type Goalreport . So get ResultList whill have a GoalReport List as return type.

```
Hibernate:
    select
        goal0_.MINUTES as col_0_0_,
        exercise1_.minutes as col_1_0_,
        exercise1_.activity as col_2_0_
    from
        goals goal0_ cross
    join
        Exercise exercise1_
    where
        goal0_.GOAL_ID=exercise1_.goal_GOAL_ID
```

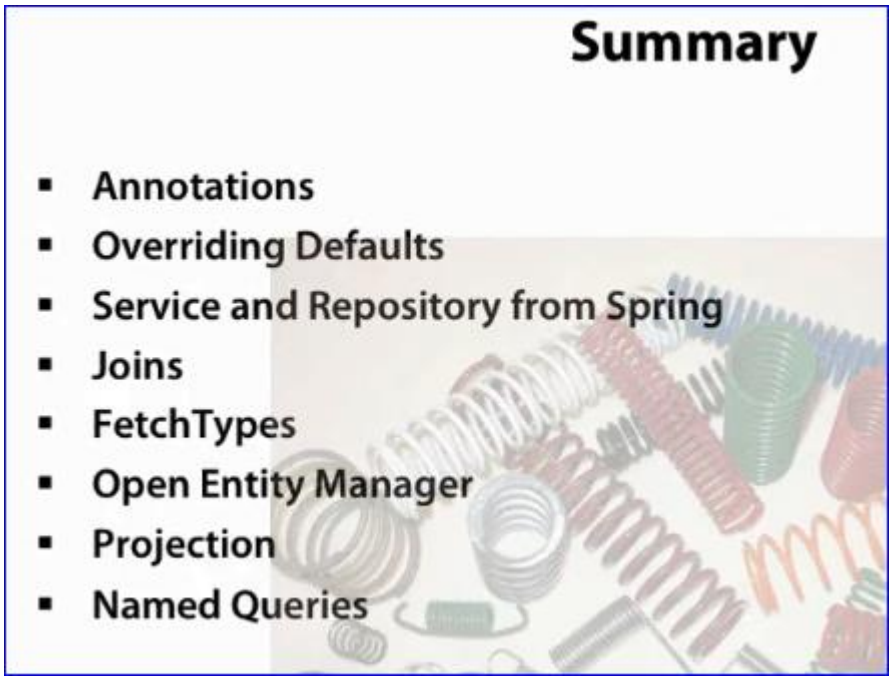
localhost:6061/FitnessTracker/getGoalReports.html

Apps | One Cognizant | Mail – Vijayakumar.... | IRCTC Next Gener

GoalMinutes	ExerMinutes	ExerActivity
56	4	Run
56	7	Bike
56	16	Swim

Add another named query in Goal domain.

```
@NamedQueries({
    @NamedQuery(name=Goal.FIND_REPORTS, query="Select new com.pluralsight.model.GoalReport("
            + "g.minutes,e.minutes, e.activity) from Goal g , Exercise e where g.id= e.goal.id)"),
    @NamedQuery(name=Goal.FIND_ALL, query="Select g from Goal g")
})
public class Goal {

    public static final String FIND_REPORTS = "findGoalReports";

    public static final String FIND_ALL = "findAll";
```

```java
33      //@SuppressWarnings({ "rawtypes", "unchecked" })
34      //We are adding suppress warnings to skip the getResultList returns untyped list
35      //But we are returning List<Goal> type. To fix add supress warning
36      public List<Goal> findAll() {
37
38          //Query query = entityManager.createQuery("Select g from Goal g");
39          TypedQuery<Goal> query = entityManager.createNamedQuery(Goal.FIND_ALL,Goal.class);
40          List<Goal> goals = query.getResultList();
41
42          return goals;
43
44      }
```

```
Hibernate:
    select
        goal0_.GOAL_ID as GOAL1_0_,
        goal0_.MINUTES as MINUTES0_
    from
        goals goal0_
Hibernate:
    select
        exercises0_.goal_GOAL_ID as goal4_0_1_,
        exercises0_.id as id1_1_,
        exercises0_.id as id1_0_,
        exercises0_.activity as activity1_0_,
        exercises0_.goal_GOAL_ID as goal4_1_0_,
        exercises0_.minutes as minutes1_0_
    from
        Exercise exercises0_
    where
        exercises0_.goal_GOAL_ID=?
```