05 - sql alchemy support

More SQL Alchemy support

There is more support on configuring the database setup in the SQL Alchemy support plugin, basically we find here all kind of support modules that help in configurations, models and services. Lets consider the previous query sample from “using AOP” chapter, we will start with the **db_sample** setup module.

add to python path:	**distribution/plugins/support-sqlalchemy.1.0.egg**

__*plugin*__.sample_plugin.db_sample

```
from ally.container import ioc, support
from ally.container.binder_op import bindValidations
from ally.support.sqlalchemy.mapper import mappingsOf
from ally.support.sqlalchemy.session import bindSession
from sample_plugin.meta import meta
from sql_alchemy import database_config
from sql_alchemy.database_config import database_url, alchemySessionCreator, \
    metas

# --------------------------------------------------------------------

support.include(database_config)

# --------------------------------------------------------------------

@ioc.replace(database_url)
def database_url():
    '''The database URL for the samples'''
    return 'sqlite:///sample.db'

@ioc.replace(metas)
def metas(): return [meta]

def bindSampleSession(proxy): bindSession(proxy, alchemySessionCreator())
def bindSampleValidations(proxy): bindValidations(proxy, mappingsOf(meta))
```

We are actually using here an already made database configuration module **database_config** found in the support modules, we just include this configuration, it will be just like copying the code from **database_config** and pasting it in the **db_sample** module. Next we are actually replacing the **database_url** configuration function, for this we have the replace ioc function which allows the replacement of an already defined entity or configuration setup functions, so even though the **database_url** is defined in **database_config** it has no default value and we just need to replace it to provide one. Next we need to provide the meta where all the plugin tables have been defined in order to automatically create the tables, so we are replacing the metas setup function with one that provides the list of the meta plugins (just in case we have more then one). Last we have the binding functions, the **bindSampleSession** is the one that was previously defined in the **service** setup module. The **bindSampleValidations** is actually performing the validations for insert and update, the validations consist in checking for unique constraints and foreign keys of the model to be inserted/updated. The validation process actually checks in the database if the unique/foreign key constraint is valid if not it will deliver a user friendly translatable error message, so it is important to bind the validations only to the proxies that are used as REST services, for the proxies that are to be used internally you should not bind the validation since it will have an impact on performance. So lets see now how we configure this in the **service** setup module.

__plugin__.sample_plugin.db_sample

```
from __plugin__.plugin.registry import addService
from __plugin__.sample_plugin.db_sample import bindSampleSession, bindSampleValidations
from ally.container import support

# --------------------------------------------------------------------

API, IMPL = 'sample_plugin.api.**.I*Service', 'sample_plugin.impl.**.*'

support.createEntitySetup(API, IMPL)

support.bindToEntities(IMPL, binders=bindSampleSession)
support.listenToEntities(IMPL, listeners=addService(bindSampleSession, bindSampleValidations))

support.loadAllEntities(API)
```

So we got rid of the **bindSampleSession** since is now defined in the **db_sample** setup module. We are calling a new support function **bindToEntities**, this will actually create proxies for all entities setup functions that return an instance that with the class found in **IMPL**, this will apply to the user defined entities setups functions and also to the automatically created entities setup functions. This proxies will be used whenever another service requires internally an instance of a service having a class from **IMPL** so this is why we only bind to this proxies the session. The listen to entities that adds the services to the REST framework is now binding to the proxies services the session and also the validation, keep in mind that this proxies are only used for external requests. You can find the sample here. So this is the configuration support, lets see now the support for service API and implementation, first we tackle the API:

__plugin__.sample_plugin.db_sample

```
from ally.api.config import service, query
from ally.api.criteria import AsLike
from sample_plugin.api import modelSample
from sql_alchemy.api.entity import Entity, QEntity, IEntityService

# --------------------------------------------------------------------

@modelSample
class User(Entity):
    '''
    The user model.
    '''
    Name = str

# --------------------------------------------------------------------

@query
class QUser(QEntity):
    '''
    The user model query object.
    '''
    name = AsLike

# --------------------------------------------------------------------

@service((Entity, User), (QEntity, QUser))
class IUserService(IEntityService):
    '''
    The user service.
    '''
```

First the **User** model now extends the **Entity** base model, it has no **Id** anymore because is inherited from **Entity**. The **QEntity** inherited by the query provides no functionality but is extended in order to be used as generic replacement in the service. Finally the service interface has no more methods defined that is because they are inherited from the **IEntityService**.

Interface	Inherits	Calls	Requires	Description

IEntityGetService		getById	a model	Provides the get entity by id method
IEntityFindService		getAll	a model	Provides the get all entities service without a query object
IEntityQueryService		getAll	a model and a query	Provides the get all entities service with a query object
IEntityCRUDService		insert, update, delete	a model	Provides the entity CRUD services
IEntityGetCRUDService	**IEntityGetService, IEntityCRUDService**	getById, insert, update, delete	a model	Just combines the interfaces, no additional call methods
IEntityNQService	**IEntityGetService, IEntityFindService, IEntityCRUDService**	getById, getAll, insert, update, delete	a model	Just combines the interfaces, no additional call methods
IEntityService	**IEntityGetService, IEntityQueryService, IEntityCRUDService**	getById, getAll, insert, update, delete	a model and a query	Just combines the interfaces, no additional call methods

Beside the fact that the user service extend the entity service you also notice that when we decorate the service we provide two tuples, the role of this is to provide generic replacing, what it will happen is that every type annotation that contains **Entity** for example **Entity**, **Entity.Id**, Iter(**Entity**), it will get replaced with **User** so the examples will look like **User**, **User.Id**, Iter(**User**), the same thing will happen with the query also.
So now we enhanced our user service interface, lets see the implementation:

__*plugin*__.sample_plugin.db_sample

```
from sample_plugin.api.user import IUserService, QUser
from sample_plugin.meta.user import User
from sql_alchemy.impl.entity import EntityServiceAlchemy

# ---------------------------------------------------------------------

class UserService(EntityServiceAlchemy, IUserService):
    '''
    Implementation for @see: IUserService
    '''

    def __init__(self):
        EntityServiceAlchemy.__init__(self, User, QUser)
```

This is all the implementation we need to make for the entity interface methods, basically the **EntityServiceAlchemy** has the method implementations for the **IEntityService**, also there is a specific implementation for each interface defined in the previous table.
So now if we redeploy the application and access http://localhost/resources/Sample/User

http://localhost/resources/Sample/User

```
<UserList>
 <User href="http://localhost/resources/Sample/User/1" />
 <User href="http://localhost/resources/Sample/User/2" />
</UserList>
```

You notice that now in the users list we do not get anymore the user models representations, this is because we have a new method **getById** in our service which is used by the ally framework to retrieve single model instances based on the id, and that is why we only get the reference addresses where the models can be retrieved in respect with the REST ideology. You can find the sample plugin here.