COMPOSITE VIEW WITH APACHE-METAMODEL

Contents

[OVERVIEW: 1](#_Toc454909670)

[Create Composite View in Feature Using Apache Meta-Model: 1](#_Toc454909671)

[Referring DataSource of Other Feature: 4](#_Toc454909672)

[*Data source by Feature Group and Feature Name:* 5](#_Toc454909673)

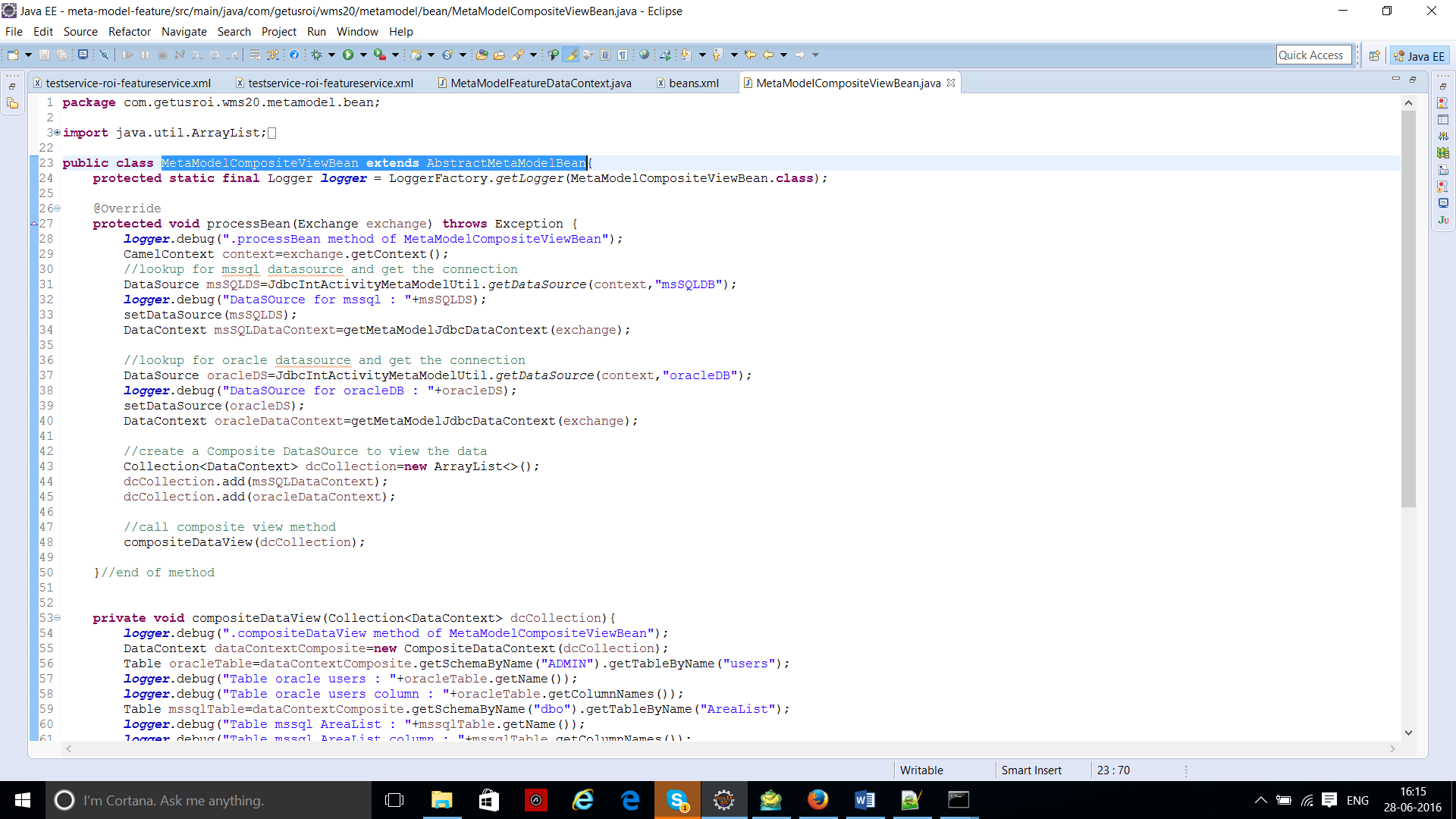
[*Data source by New RequestContext:* 7](#_Toc454909674)

# OVERVIEW:

A composite view allows you to connect to multiple different data sources and join data across these in a single view. Here we are using different database like (oracle, MS SQL, MySQL) and meta model api using join operation.

# Create Composite View in Feature Using Apache Meta-Model:

Here are the steps we need to follow to get the composite view using apache Meta Model. Make sure the class you are using to perform composite view using apache meta model must implement “AbstractMetaModelBean” in order to make use of capabilities provided like(getting JdbcDataContextObject, getting connection Object etc.).



1. First you need to get the Data Source Object to do that you need to provide the name for the database you want to work on. This process is called “Database Lookup”.

Note: The name of the database you are providing for the database lookup, make sure you must have defined database configuration in camel/route.xml file with that name.

For example: define the data source with id “oracleDB”.

<bean id="oracleDB" class="com. atomikos.jdbc.AtomikosDataSourceBean"

init-method="init" destroy-method="close">

<property name="uniqueResourceName">

<value>OracleDB</value>

</property>

<property name="xaDataSourceClassName">

<value>oracle.jdbc.xa.client.OracleXADataSource</value>

</property>

<property name="xaProperties">

<props>

<prop key="serverName">192.168.1.78</prop>

<prop key="portNumber">1521</prop>

<prop key="databaseName">XE</prop>

<prop key="user">admin</prop>

<prop key="password">admin</prop>

<prop key="URL">jdbc:oracle:thin:@192.168.1.78:1521:XE</prop>

</props>

</property>

<property name="minPoolSize">

<value>1</value>

</property>

<property name="maxPoolSize">

<value>2</value>

</property>

</bean>

Once data source is defined we can look up for this data source using camel exchange by providing the id used while defining the data source i.e. “oracleDB” in bean class as given below:

CamelContext context=exchange.getContext();

//lookup for mssql datasource and get the connection

DataSource oracleDS

=JdbcIntActivityMetaModelUtil.*getDataSource*(context,"oracleDB");

b) Once you got the datasource successfully from the above step, next step is to set the datasource using setters provided in super class.

setDataSource(oracleDS);

c) Get the apache metamodel “DataContext” object using the super class method “getJdbcDataContext(Exchange exchange)”.

DataContext oracleDataContext=getMetaModelJdbcDataContext(exchange);

“getJdbcDataContext(exchange)” method then internally call getConnection() and get the Connection Object. This connection object is then used to create JdbcDataContext Object and return the “JdbcDataContext” which will be type casted into “DataContext” interface as shown above.

d) To get compositeView it requires to have more than one data source. Repeat the above steps for other data source (MS SQL, MySQL) as well and get “DataContext” object for those data sources.

e) To create “CompositeDataContext” object, it is required to have collection of all “DataContext” therefore, first create a collection object for all the datasource got from above steps.

//create a collection of data context for different types of data sources

Collection<DataContext> dcCollection=**new** ArrayList<>();

dcCollection.add(msSQLDataContext);

dcCollection.add(oracleDataContext);

**Now create “CompositeDataContext” using collection Object:**

DataContext dataContextComposite=**new** CompositeDataContext(dcCollection);

f) Now the “CompositeDataContext” has information (Schema name, table name, column name etc.) about collection of data source. Get The Table from the Composite data context on which you want to work on (view the data) by passing the schema name it belongs to and table name itself.

For example:

Table oracleTable=dataContextComposite.getSchemaByName("ADMIN").getTableByName("users");

Table mssqlTable=dataContextComposite.getSchemaByName("dbo").getTableByName("AreaList");

g) You can frame a query using different table object got from previous step to fetch the data and create a composite view. To do this, it is requiring to have “compositeDataContext” object, “Table” object and operation you want to perform.

For example- create a left join query using “oracledb” and “mssql db” is given below:

Query q =dataContextComposite.query().from(oracleTable).leftJoin(mssqlTable).on(oracleTable.getColumnByName("userid"), mssqlTable.getColumnByName("AreaId")).selectAll().toQuery();

h) Once the query is created, next step is to execute it and display the result.

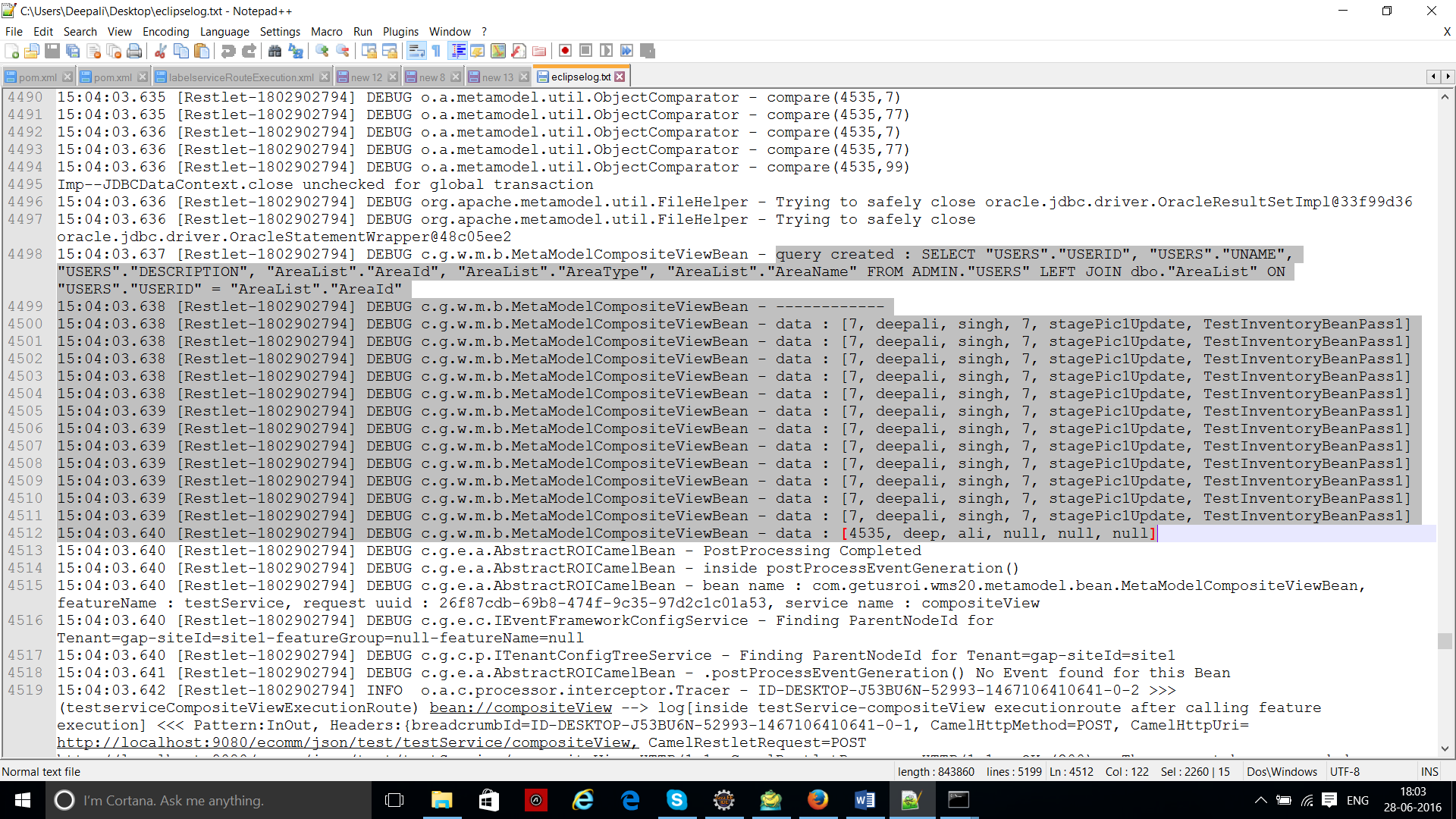
DataSet dataSet = dataContextComposite.executeQuery(q);

**for** (Row row: dataSet) {

//logic to display the result using row object

}

Result after above join operation will be as given below:



# Referring DataSource of Other Feature:

There are two ways we can refer to datasource defined for some other feature.

1. Data source by Feature Group and Feature Name.
2. Data Source with new Request Context.

To work on these way, important thing require is “featureDataContext.xml” file. This file contains the information about database this feature refers to, also the information about the database configure for some other feature but this feature can refer to.

For Example:

<FeatureDataContext xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi: noNamespaceSchemaLocation=*"featureDataContext.xsd"*>

<DataContexts contextName=*"test-testService-DataContext"*>

<DataContext dbBeanRefName=*"oracleDB"* dbType=*'oracle'* dbHost=*"192.168.1.78"* dbPort=*"1521"* dbSchema=*"XE"*/>

</DataContexts>

<!—testService feature can refer to db defined for printservice feature-->

<RefDataContexts featureGroup=*"print"* featureName=*"printservice"*>

<RefDataContext dbBeanRefName=*"mysqlDB"* dbType=*'mysql'* dbHost=*"192.168.1.77"* dbPort=*"3306"* dbSchema=*"roi"*/>

</RefDataContexts>

</FeatureDataContext>

If we look into the above xml, it has two part:

a) “DataContexts” contain the information about the database which current feature can use.

b) “RefDataContext” contain the information about the database configured for some other feature which current feature can refer it.

We need to add “featureDataContext.xml” file into “featureMetaInfo.xml” so that this information should be available at system startup time.

<FeatureDataContexts>

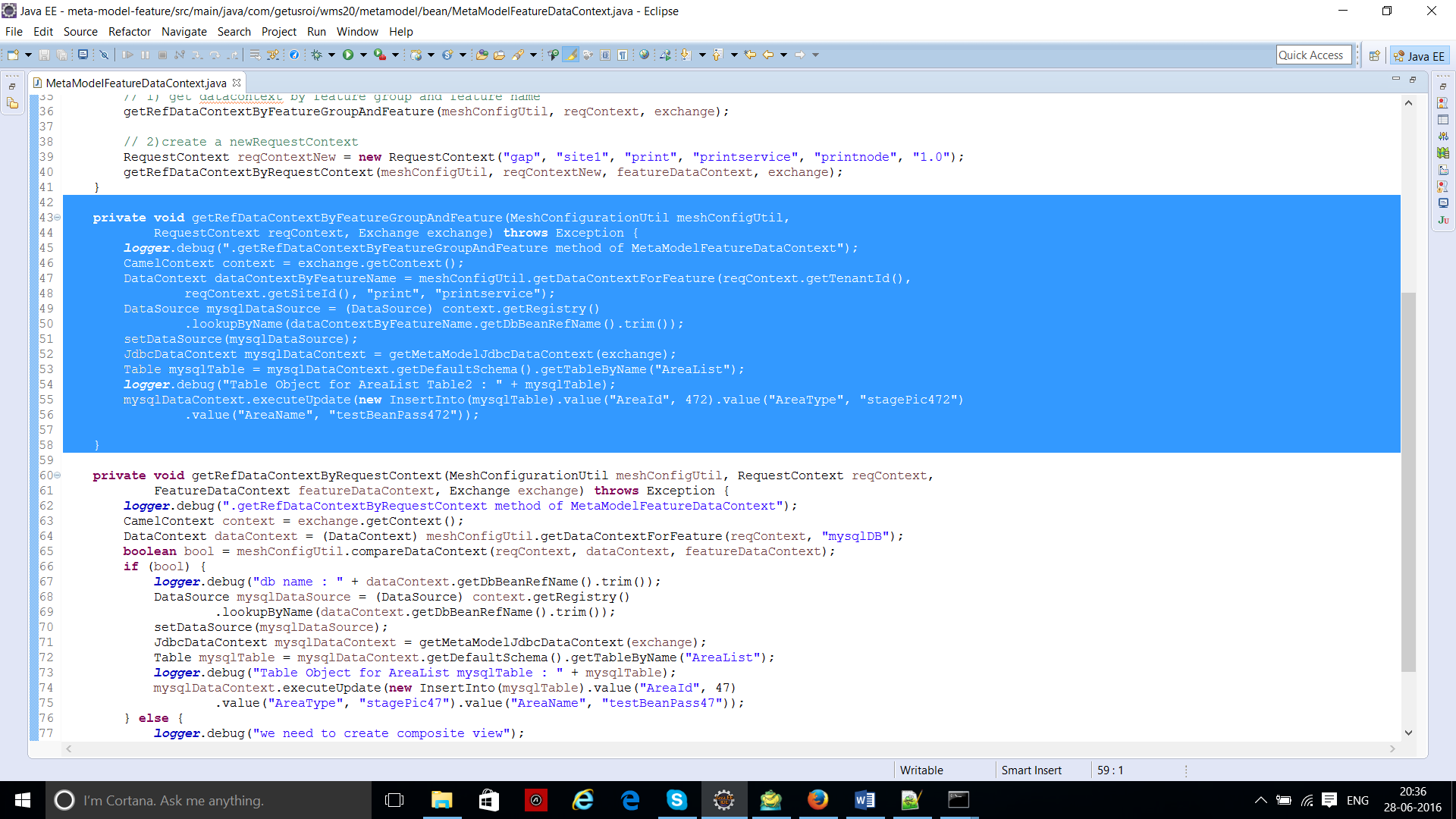
<DataContexts resourceName=*"testservice-roi-featureDataContext.xml"*/>

</FeatureDataContexts>

## *Data source by Feature Group and Feature Name:*

This technique should be used when you want to access the data source of feature which belong to same tenant and site of current feature but has different feature group and feature name.

Configuration in bean class should be done as given in screenshot:



Here are the steps which you need to follow:

a) First you need to call method of class “MeshConfigurationUtil” called “getDataContextForFeature ()”, which will accept tenant name and site name from current feature request context and feature group and feature name you need to provide while implementing.

DataContext dataContextByFeatureName = meshConfigUtil.getDataContextForFeature(reqContext.getTenantId(),

reqContext.getSiteId(), "print", "printservice");

Therefore,

getDataContextForFeature(String tenant, String site, String featureGroup, String featureName)

e.g -> getDataContextForFeature(“gap”, “site1”, “print”, “printService”);

Where, tenant and site will come from current feature request context but feature group and feature name you will be needed to provide.

b) The “getDataContextForFeature” then internally create new request context and make a request to load the datacontext for new request context and return the actual required data context configured for request context.

Let’s say new request context is as given below:

RequestContext reqContext=new RequestContext(“tenanat”,”site1”,”print”,”printService”,”roi”,”1.0”);

c) The return “DataContext” is of type “com.getusroi.datacontext.jaxb” not of “org.apache.metamodel”. You then need to create the DataSource of type “org.apache.metamodel”. It can be done as given below

CamelContext context=exchange.getContext();

//lookup for mysql datasource and get the connection

DataSource mysqlDataSource = (DataSource) context.getRegistry() .lookupByName(dataContextByFeatureName.getDbBeanRefName().trim());

d) Once you got the datasource successfully from the above step, next step is to set the datasource using setters provided in super class.

setDataSource(mysqlDataSource);

e) Get the apache metamodel “DataContext” object using the super class method “getJdbcDataContext(Exchange exchange)”.

DataContext mysqlDataContext=getMetaModelJdbcDataContext(exchange);

“getJdbcDataContext(exchange)” method then internally call getConnection() and get the Connection Object. This connection object is then used to create JdbcDataContext Object and return the “JdbcDataContext” which will be type casted into “DataContext” interface as shown above.

f) . Get The Table from the data context on which you want to work on (view the data) by using default schema and passing table name.

Table mysqlTable = mysqlDataContext.getDefaultSchema().getTableByName("AreaList");

g) Using DataContext of type “org.apache.metamodel” you can perform and sql operation.

For example: To do insert operation

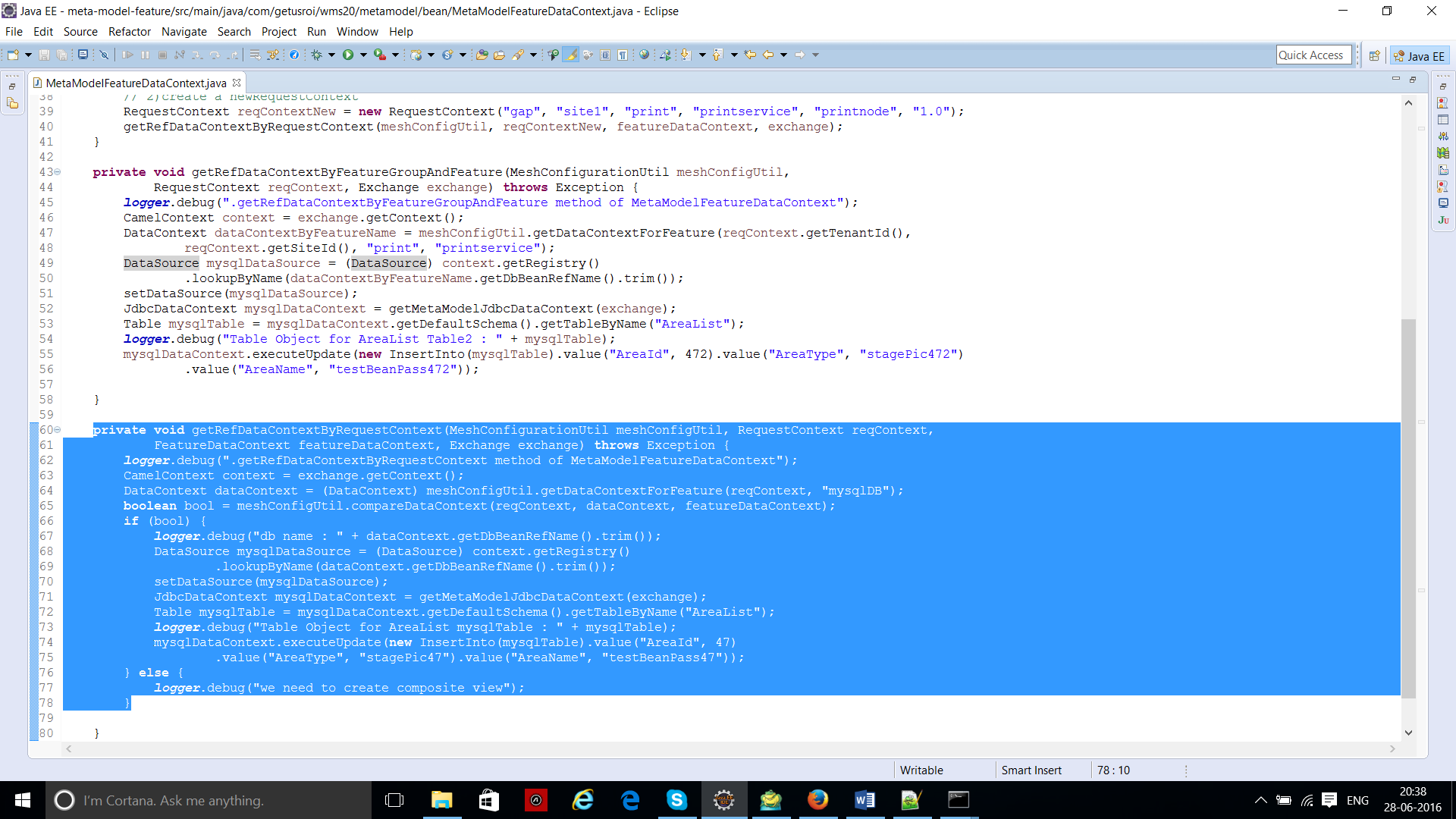
mysqlDataContext.executeUpdate(**new** InsertInto(mysqlTable).value("AreaId", 472).value("AreaType", "stagePic472")

.value("AreaName", "testBeanPass472"));

## *Data source by New RequestContext:*

This technique should be used when you want to access the data source of feature which belong to completely different tenant, site, featureGroup, featureName, vendor, version.

Configuration in bean class should be done as given in screenshot:



Here are the steps which you need to follow:

a) First you need to call method of class “MeshConfigurationUtil” called “getDataContextForFeature()”, which will accept request context for feature whose data source you want to refer and database configuration name.

So first step is to create new request context:

RequestContext reqContext=new RequestContext(“tenanat”,”site1”,”print”,”printService”,”roi”,”1.0”);

Pass above configured request context to “getDataContextForFeature()” to get “DataContext” of type “org.getusroi.datacontext.jaxb” and database config name:

DataContext dataContextByFeatureName = meshConfigUtil.getDataContextForFeature(reqContext," mysqlDB");

b) The “getDataContextForFeature” then make a request to load the datacontext for new request context and return the actual required data context configured for request context.

c) The return “DataContext” is of type “com.getusroi.datacontext.jaxb” not of “org.apache.metamodel”. You then need to create the DataSource of type “org.apache.metamodel”. It can be done as given below

CamelContext context=exchange.getContext();

//lookup for mysql datasource and get the connection

DataSource mysqlDataSource = (DataSource) context.getRegistry() .lookupByName(dataContextByFeatureName.getDbBeanRefName().trim());

d) Call “compareDataContext()” of “MeshConfigurationUtil” class which will check the data context received in previous steps for different feature has same database configuration or not. If yes, then “DataContext” object should be returned else “CompositeDataContext” should be returned.

e) Once you got the datasource successfully from the above step, next step is to set the datasource using setters provided in super class.

setDataSource(mysqlDataSource);

f) Get the apache metamodel “DataContext” object using the super class method “getJdbcDataContext(Exchange exchange)”.

DataContext mysqlDataContext=getMetaModelJdbcDataContext(exchange);

“getJdbcDataContext(exchange)” method then internally calls getConnection() and get the Connection Object. This connection object is then used to create JdbcDataContext Object and returns the “JdbcDataContext” which will be type casted into “DataContext” interface as shown above.

g) . Get The Table from the data context on which you want to work on (view the data) by using default schema and passing table name.

Table mysqlTable = mysqlDataContext.getDefaultSchema().getTableByName("AreaList");

h) Using DataContext of type “org.apache.metamodel” you can perform and sql operation.

For example: To do insert operation

mysqlDataContext.executeUpdate(**new** InsertInto(mysqlTable).value("AreaId", 472).value("AreaType", "stagePic472")

.value ("AreaName", "testBeanPass472"));