openSAP SAP HANA Development

HANA 2.0 SPS 03 October 2018 Revision or Higher

October 2018

Exercises / Solutions

Rich Heilman / SAP Labs, LLC.
Thomas Jung / SAP Labs, LLC.

PUBLIC



TABLE OF CONTENTS

BEFORE YOU START 5
Getting Help 5
Source code solutions 5
Exercise 1 – Hello World 6
Exercise 1 – Solution 7
Exercise 1.1: HTML5 Module - Hello World 7
Exercise 1.2: Clone a Repository from Git 23
Exercise 1.3: Alternative: Import Project from File System 25
Exercise 2 –Database Artifact Development 27
Exercise 2 – Solution 28
Exercise 2.1: Database Content 28
Exercise 2.2: Create Tables & Views via Core Data Services 33
Exercise 2.3: Create Table Data Configuration 39
Exercise 2.4: Create Tables & Views via SQL DDL Design Time Artifacts 43
Exercise 2.5: Create Structured Privilege & Role 52
Exercise 2.6: Non-Container Schemas, User Provided Services & Synonyms 55
Exercise 2.7: Cross Container Services & Synonyms 63
Exercise 2.8: Cross Container From Outside Our Project 71
Exercise 2.9 Creating a Calculation View with a Dimension data type 77
Exercise 2.10: Creating a Calculation View with a Cube data type and Star Join 95
Exercise 2.11: Table Function as Data Source in Calculation View 102
Exercise 2.12: Importing Calcuation Views (and other native DB objects) into the new CDS 107
Exercise 2.13: Leveraging SQLScript in Stored Procedures & User Defined Functions 114
Exercise 2.13.1: Creating A Simple Stored Procedure 114
Exercise 2.13.2: Create a Scalar User Defined Function 119
Exercise 2.13.3: Create a Table User Defined Function 123
Exercise 2.13.4: Libraries 128
Exercise 2.13.4.1: Creating User Defined Libraries 128
Exercise 2.13.4.2: Leveraging Built-In Libraries 138
Exercise 2.13.5: Anonymous Blocks 142
Exercise 2.13.6: Table Manipulation 145
Exercise 2.13.6.1: Table Variable Operators 145
Exercise 2.13.6.2: MAP_MERGE Operator 153

Exercise 2.13.6.3: MAP_REDUCE Operator 160
Exercise 2.13.7: Debugging SQLScript 164
Exercise 2.13.7.1: Debugging a Procedure 164
Exercise 2.13.7.2: Breaking on Error 170
Exercise 2.13.7.3: Save Table Feature 178
Exercise 2.13.8: Performance Analysis 181
Exercise 2.13.8.1: Analyze Long Running Procedures 181
Exercise 2.13.8.2: Overall Performance Analysis 183
Exercise 2.13.8.3: Using SAP HANA Query Analysis Tool 185
Exercise 2.13.8.4: SQLScript Analyzer 188
Exercise 3 –XSJS and XSODATA services 192
Exercise 3 – Solution 193
Exercise 3.1: XSJS and XSODATA 193
Exercise 3.2: Debugging XSJS or Node.js 205
Exercise 3.3: Exploring JavaScript Language Features 207
Exericse 3.4: Creating an XSJS Service with outbound HTTP Connectivity 213
Exericse 3.5: User Provided Service as SQLCC 218
Exercise 3.6: Creating a Simple Odata V2 Service 221
Exercise 3.7: Creating an OData V2 Service with an Entity Relationship 224
Exercise 3.8: Creating an Odata V2 Service with Create Operation and XSJS Exit 227
Exercise 4 – Node.JS 230
Exercise 4 – Solution 231
Exercise 4.1: Odata V4 services 231
Exercise 4.2: Modules and Express 238
Exercise 4.3: HANA Database Access from Node.js 246
Exercise 4.4: Asynchronous Non-Blocking I/O 266
Exercise 4.5: Exploring JavaScript Language Features 278
Exercise 4.6: Text Bundles 284
Exercise 4.7: Open Source Modules 291
Exercise 4.8: JavaScript Object Oriented 301
Exercise 4.9: Web Sockets 316
Exercise 4.10: Odata V4 Services with Exits 322
Exercise 5 – SAPUI5/Fiori User Interface 334
Exercise 5 – Solution 335
Exercise 5.1: SAPUI5 as an XSA Micro-Service 335

Exercise 5.2: Creating a Text Bundle 340
Exercise 5.3: SAPUI5 User Interface 343
Exercise 5.4: Consume XSJS Services via JQuery AJAX calls 356
Exercise 5.5: Consume a Basic OData V2 Service within SAPUI5 binding the service to a Table 363
Exercise 5.6: Use oData V2 Metadata to dynamically create the columns. 367
Exercise 5.7: Consume an OData V2 Service with Create Option 369
Exercise 5.8: OData V2 Batch Operation 375
Exercise 5.9: OData V2 Deep Insert (Content-ID and Links) 380
Exercise 5.10: OData V4 and Fiori Annotations 390
Exercise 5.11: Consume an OData V4 Service with Create Option 399
Exercise 6 – Packaging for Transport 403
Exercise 6 – Solution 404
Exercise 6.1: Package for Transport 404

BEFORE YOU START

System Host: <hostname>

System Instance Number: <System Number>

Desktop User: Administrator

System User ID: WORKSHOP_<group number>. Your group number will be given to you by the session instructor. For example if the group number is “01”, then your user id would be WORKSHOP_01. If you are working on your own private system then you can use your own developer user or create a WORKHOP_01 user yourself.

All Passwords: <Password>

XSA Organization: HANAExpress

XSA Development Space: development

Note: These exercises are designed for XSA systems setup with hostname based routing. If you are using port based routing, you will need to adjust the hostname and port for launching the Web IDE appropriately.

Getting Help

If you need addition help resources beyond this document, we would suggest the following content:

- The Online Help at https://help.sap.com/viewer/p/SAP_HANA_PLATFORM

Source code solutions

All source code solutions and templates for all exercises in this document can be found in the following webpage.
https://github.com/SAP/hana-xsa-opensap-hana7/tree/snippets_2.3.2

In some cases it might be a little easier to copy and paste the coding instead of typing it all manually. If copying/pasting, I would suggest that you make sure to understand what is being copied/pasted before moving on.

Open the browser and enter the following URL to access the solutions web page (or you can use the bookmark shortcut we have created for you in Google Chrome). You can access the source code for each exercise by clicking on the appropriate exercise in the navigation bar.
https://github.com/SAP/hana-xsa-opensap-hana7/tree/snippets_2.3.2

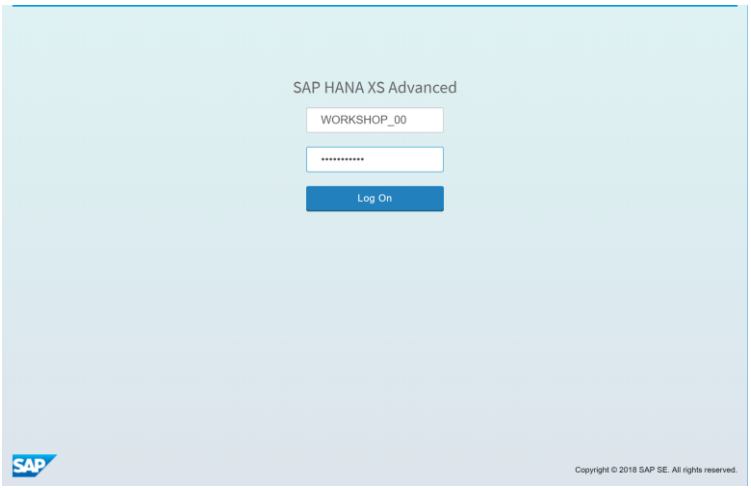
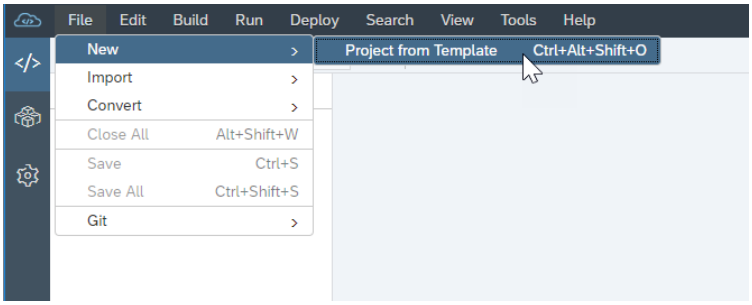
EXERCISE 1 – HELLO WORLD

Objective

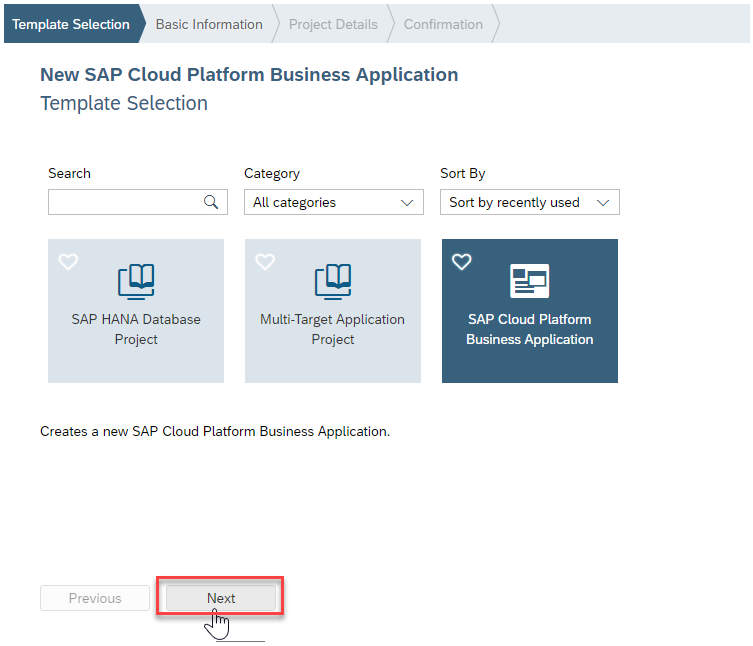
In this first exercise, we will connect to the remote system, run the new project wizard, and then create an HTML5 module to serve as the application endpoint and proxy all of our services and client-side content. At the end of this exercise you will be able to connect to your server via web browser and see a Hello World message.

EXERCISE 1 – SOLUTION

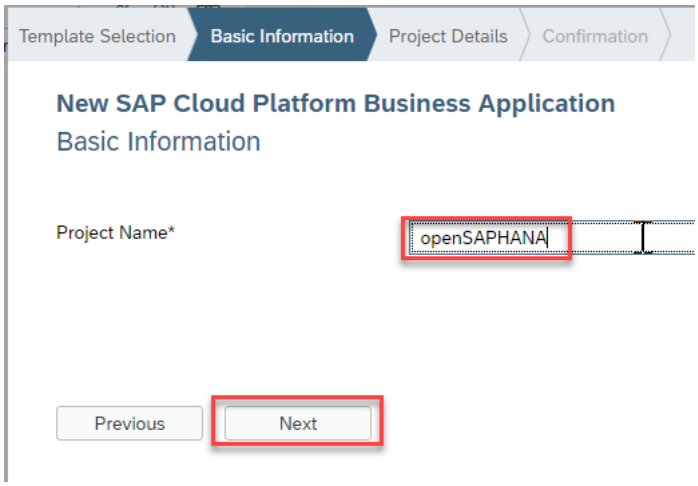
Exercise 1.1: HTML5 Module - Hello World

Explanation	Screenshot
1. Launch the SAP Web IDE for SAP HANA at the following URL in your web browser. https://webide.<hostname>:30033/ User: WORKSHOP_XX where XX is your group number Password: <Password> Note: These exercises are designed for XSA systems setup with hostname based routing. If you are using port based routing, you will need to adjust the hostname and port for launching the Web IDE appropriately	
2. We will begin by creating a new project from template. Use **File->New->Project from Template**	

3. Choose **SAP Cloud Platform Business Application** and then press **Next**.



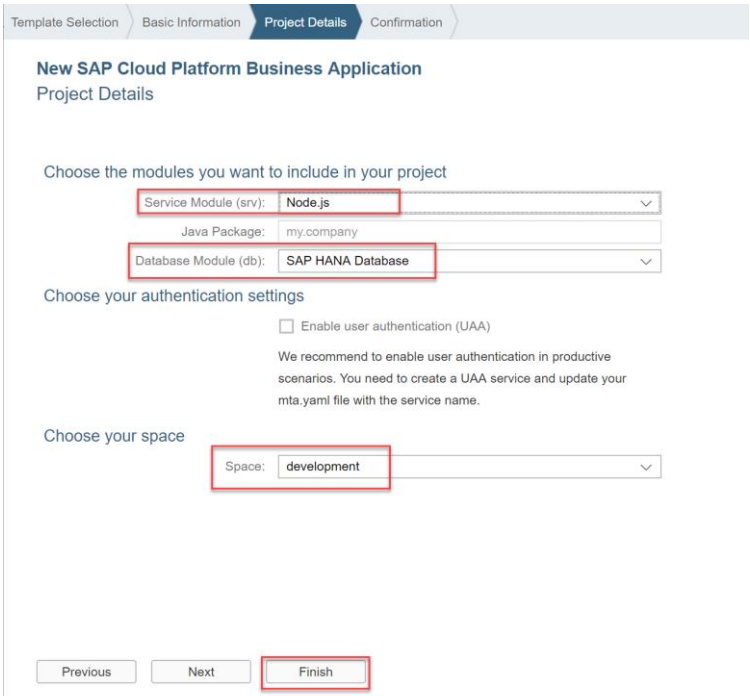
4. Enter the project name **openSAPHANA**. Press **Next**.



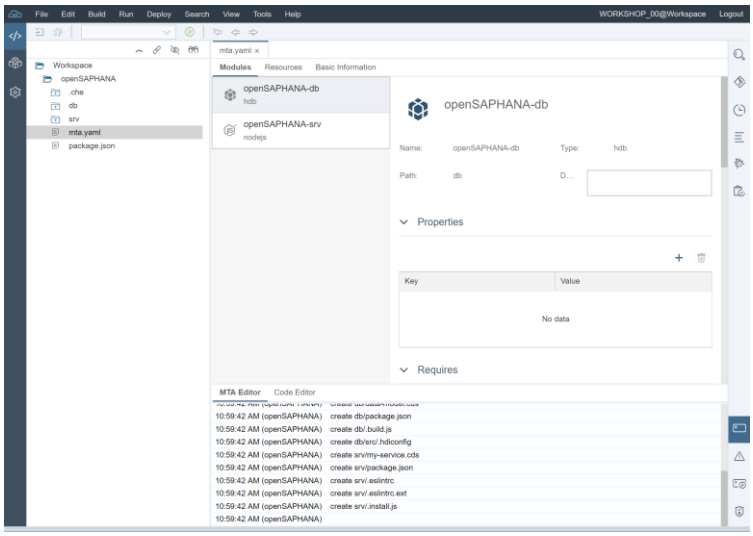
5. In the Project Details screen we have to choose which modules we want to generate. We will create a **Node.js** based Service Module and an **SAP HANA Database** module.

   Also choose **development** as your target space.

   Finally press **Finish** to generate the project.



6. Your wizard generated project has been created and already has a database and a service module.

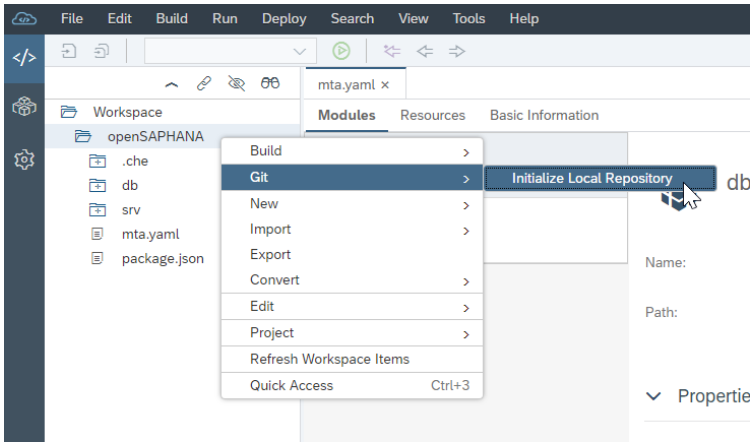


7. Go to the wizard generate package.json in the root of your project. This is where the overall Application Programming Model settings are made. In here, please make sure that the version of the **@sap/cds** module is set to **^2.10**



8. Next let's initialize a local Git repository via the right mouse click on the project and choosing **Git -> Initialize Local Repository**.

   While in real development systems you should always use an external/remote Git server for permanant persistence, in sandbox style systems you can at least use a local Git repository for basic version management.



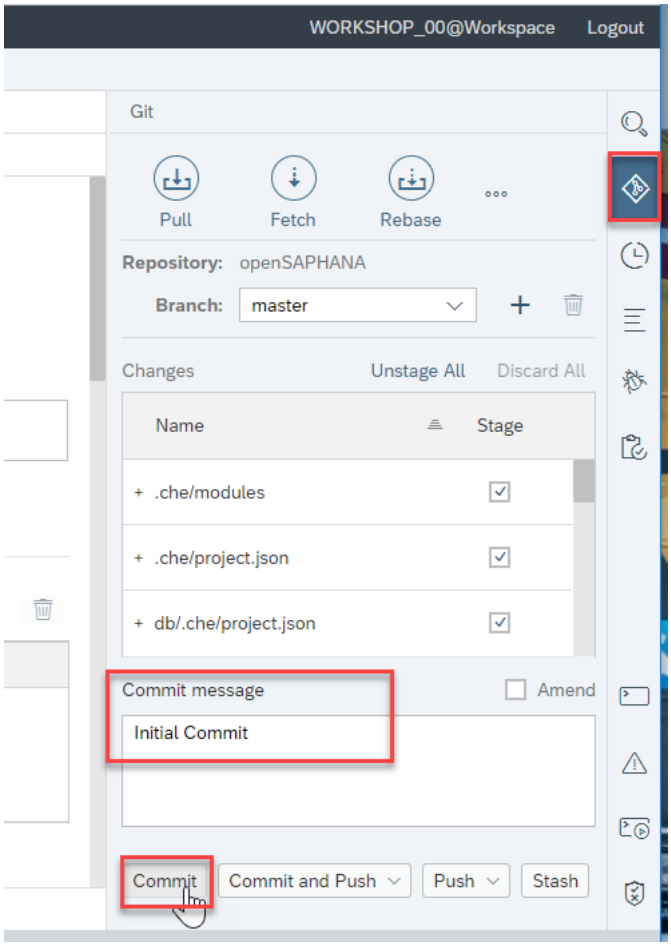
9. Go ahead and perform an intial commit to the local Git repository.

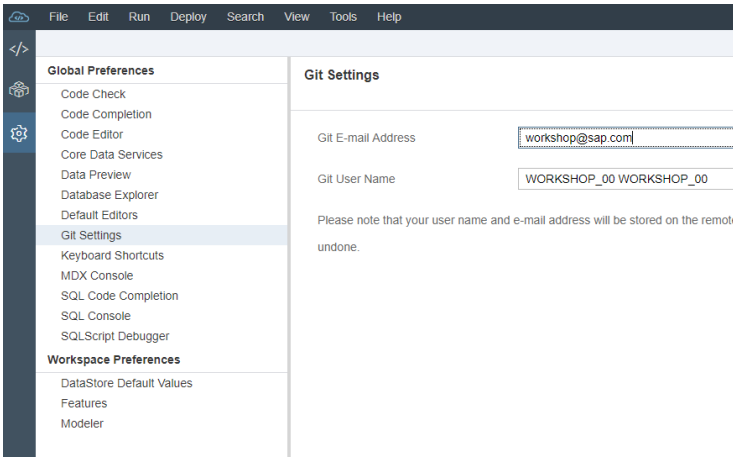
   Choose the Git icon from the right hand bar. This will open the Git pane.

   Supply a commit message and then press the **Commit** button.

   Each time you commit, a version is stored and comparisons and reverts can be performed. Therefore we would recommend that you commit at least at the end of each exercise in this workshop.

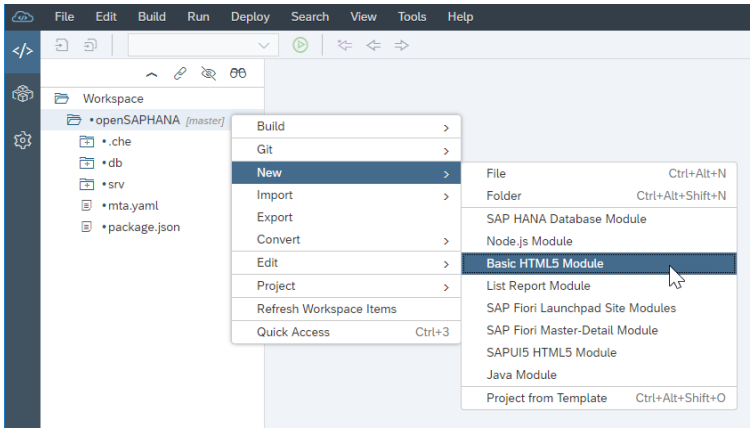
   Note: if you receive an error that your email address isn't maintained when you commit, then go to the Web IDE Preferences -> Git Settings and maintain an email address there.



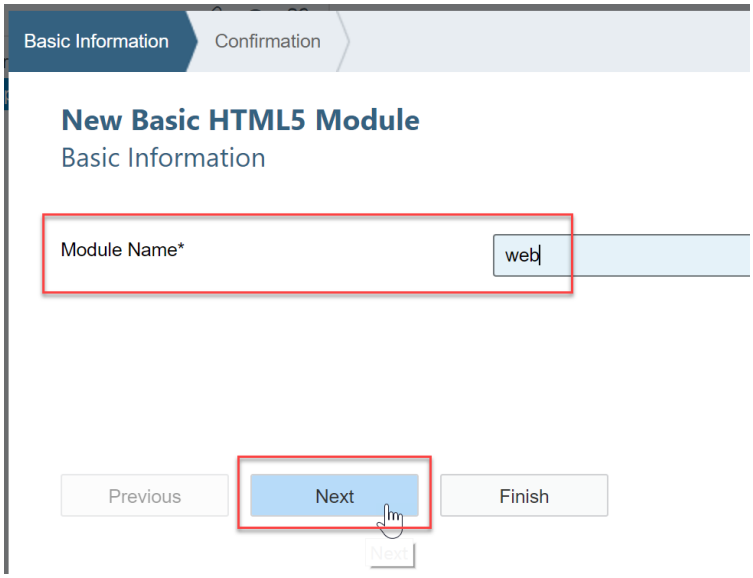


10. Next we need to create the HTML5 module to host and serve out the frontend content.

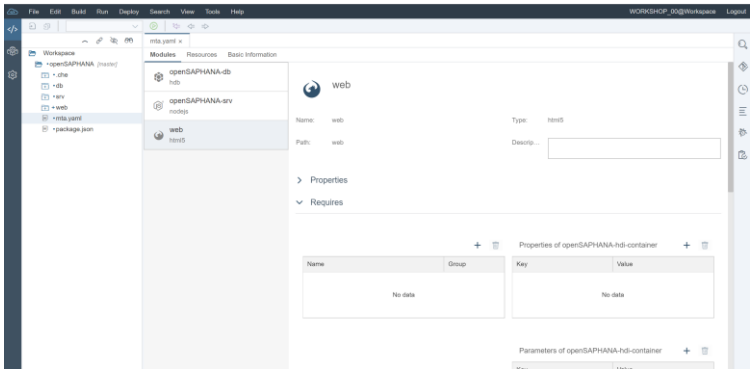
    Begin by selecting your project and then choosing **New -> Basic HTML5 Module**



11. Name the module **web**. Press **Next**. Then press **Finish**.



12. This not only adds the necessary metadata to the web folder but also maintains the module entry in the project's **mta.yaml** file within the project root.

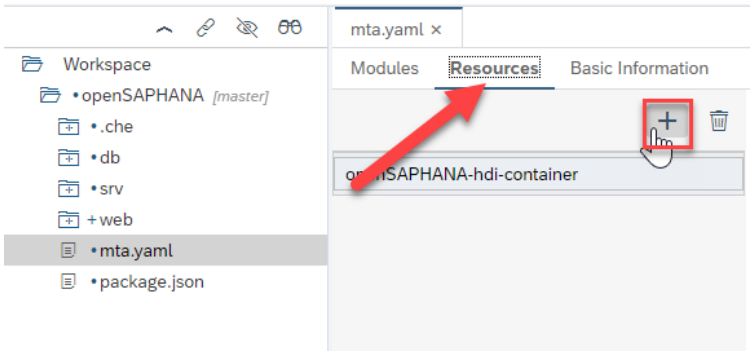


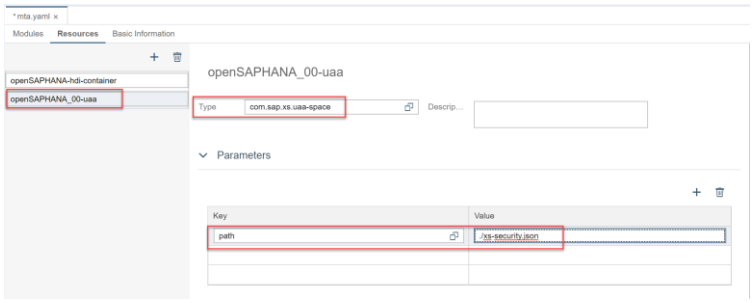
13. We need to add **openSAPHANA_XX-uaa** (where XX is your group number) as a dependent service/resource to the mta.yaml file. This is because we have user authentication required to access our application.

    Click on the **Resources** tab. Then click the plus button to add a new resource entry. This will make a new input field appear below the plus button. You can type the name of the resource in here.

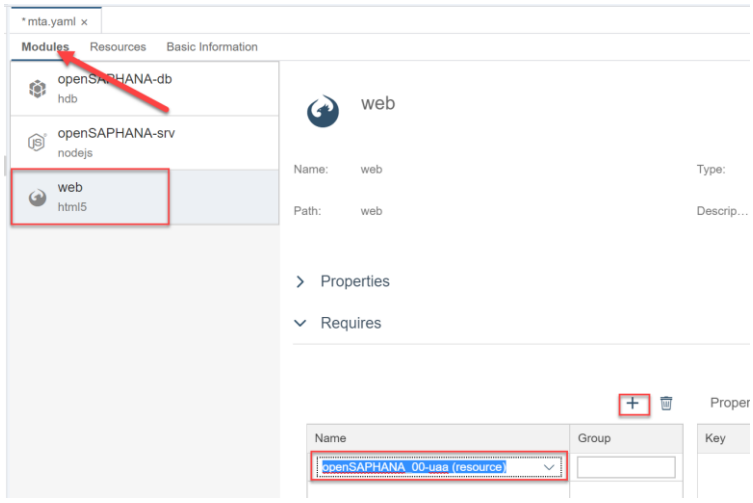
    Then on the right hand side you can fill in the type and parameters of the resource.

    Save your file. Switch to the Code Editor and then back to the MTA Editor to refresh all the value helps.





14. Switch back to the **Modules** tab and select your **web** module. Click the plus button in the **Requires** section and choose the newly created UAA resources from the drop down list. This will create the connection between the web module and the UAA resource.



15. You can also switch to the MTA editor code view via the button on the bottom left corner of the editor. This will show you the complete view of the content in text form.

ID: openSAPHANA
_schema-version: '2.1'
version: 0.0.1
modules:
  - **name: db**
    type: hdb

We will use the text editor to rename the two wizard generated modules from **openSAPHANA-db** to **db** and **openSAPHANA-srv** to **srv**. Use this to double check your entries.

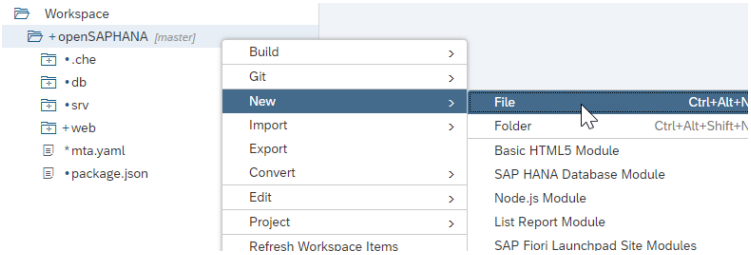
Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex1/ex1_1

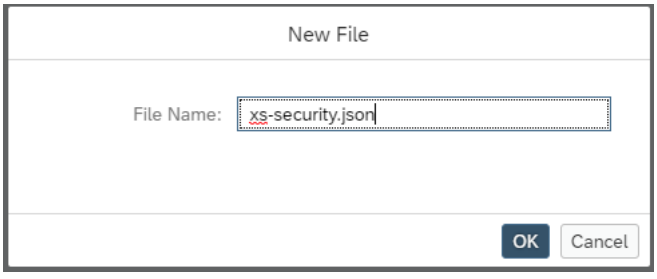
```
    path: db
    parameters:
      memory: 256M
      disk-quota: 256M
    requires:
      - name: openSAPHANA-hdi-container
  - name: srv
    type: nodejs
    path: srv
    parameters:
      memory: 512M
      disk-quota: 256M
    provides:
      - name: srv_api
        properties:
          url: '${default-url}'
    requires:
      - name: openSAPHANA-hdi-container
  - name: web
    type: html5
    path: web
    requires:
      - name: openSAPHANA_<group number>-uaa
resources:
  - name: openSAPHANA-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: openSAPHANA_<group number>-uaa
    type: com.sap.xs.uaa-space
    parameters:
      path: ./xs-security.json
```

16. The xs-security.json file we referenced in the configuration of the UAA is where we can define the scopes, roles, and attributes which will be used for security at the application level. We now need to create that file.

    Right mouse click on your project and choose **New->File**

    In the New File dialog choose **xs-security.json** as the name and press **OK**.





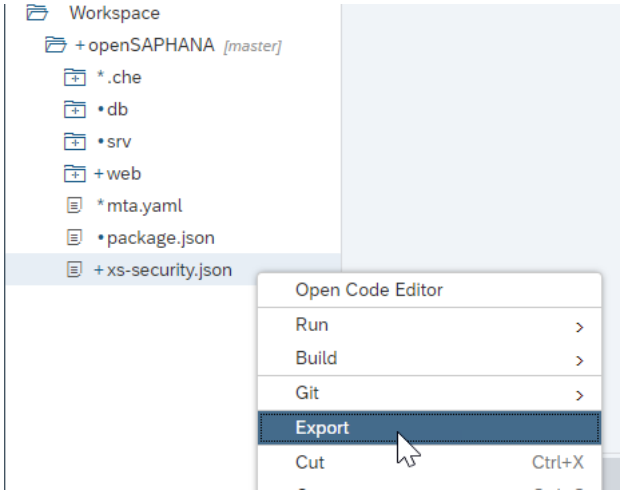
17. Supply the following content in the new **xs-security.json** file replacing your group number in the **xsappname** parameter.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex1/ex1_1a

```
{
    "xsappname": "openSAPHANA-<group_number>",
    "scopes": [{
                                        "name":
    "$XSAPPNAME.Display",
                                        "description":
    "display"
    }, {
                                        "name":
    "$XSAPPNAME.Create",
                                        "description":
    "create"
    }, {
                                        "name":
    "$XSAPPNAME.Edit",
                                        "description":
    "edit"
    }, {
                                        "name":
    "$XSAPPNAME.Delete",
                                        "description":
    "delete"
    }, {
                                        "name":
    "$XSAPPNAME.DataGenerator",
                                        "description":
    "data generator"
    }, {
                                        "name":
    "xs_authorization.read",
                                        "description":
    "Read authorization information from UAA"
    }, {
                                        "name":
    "xs_authorization.write",
                                        "description":
    "Write authorization information to UAA"
    }, {
                                        "name":
    "$XSAPPNAME.ODATASERVICEUSER",
                                        "description":
    "Enter"
    }, {
                                        "name":
    "$XSAPPNAME.ODATASERVICEADMIN",
                                        "description":
    "Enter"
    }],
    "attributes": [{
                                        "name": "client",
                                        "description":
    "Session Client",
                                        "valueType": "int"
    }, {
```

```
                                                    "name": "country",
                                                    "description":
        "country",
                                                    "valueType": "s"
        }],
        "role-templates": [{
                                                    "name": "Viewer",
                                                    "description":
        "View all records",
                                                    "scope-
        references": [
                                                    "$XSAPPNAME.D
        isplay"
                                                    ],
                                                    "attribute-
        references": [
                                                        "client",
        "country"
                                                    ]
        }, {
                                                    "name": "Editor",
                                                    "description": "Edit
        and Delete records",
                                                    "scope-
        references": [
                                                    "$XSAPPNAME.C
        reate",
                                                    "$XSAPPNAME.E
        dit",
                                                    "$XSAPPNAME.D
        elete",
                                                    "$XSAPPNAME.D
        isplay",
                                                    "$XSAPPNAME.D
        ataGenerator",
                                                    "$XSAPPNAME.O
        DATASERVICEUSER",
                                                    "$XSAPPNAME.O
        DATASERVICEADMIN"
                                                    ],
                                                    "attribute-
        references": [
                                                        "client"
                                                    ]
        }]
    }
```

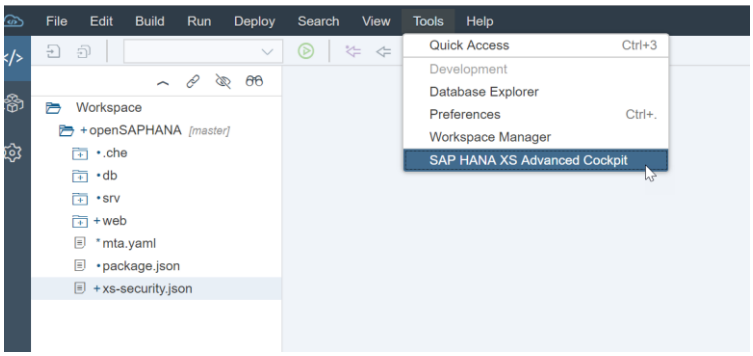
18. We will need this xs-security.json file locally for the next step. Therefore right mouse click on the xs-security.json file and choose **Export**.

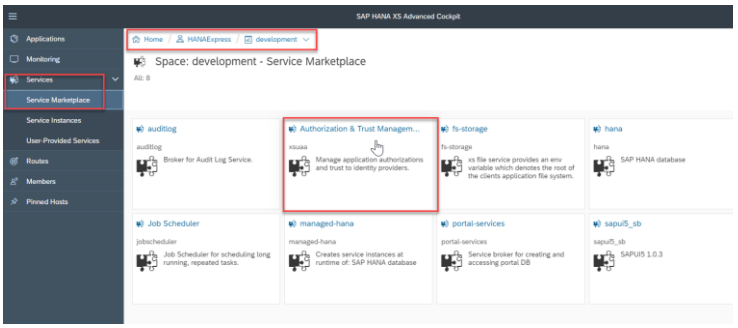


19. Now we need to create an instance of the UAA service we want to use in our application. The Web IDE can provision some but not all resource types yet. The UAA is one of those types which it can not provision.

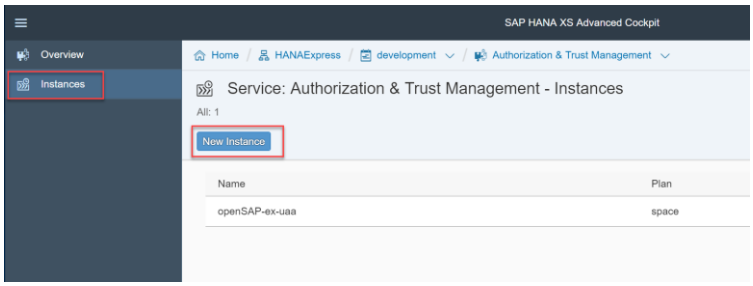
    Launch the SAP HANA XS Advanced Cockpit by choosing **Tools -> SAP HANA XS Advanced Cockpit**.

Navigate to your organization **HANAExpress** and space **development**. Expand **Services** and then click on **Service Marketplace**. Select **Authorization & Trust Mangement**.

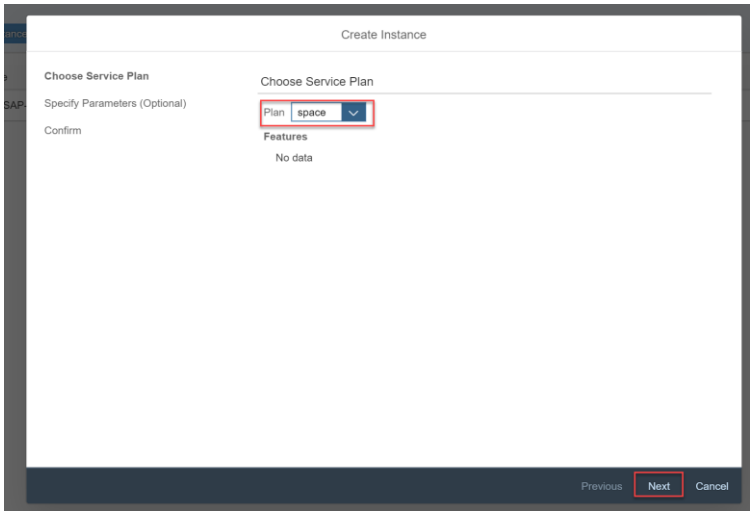




20. Click on **Instances** and press the **New Instance** button.

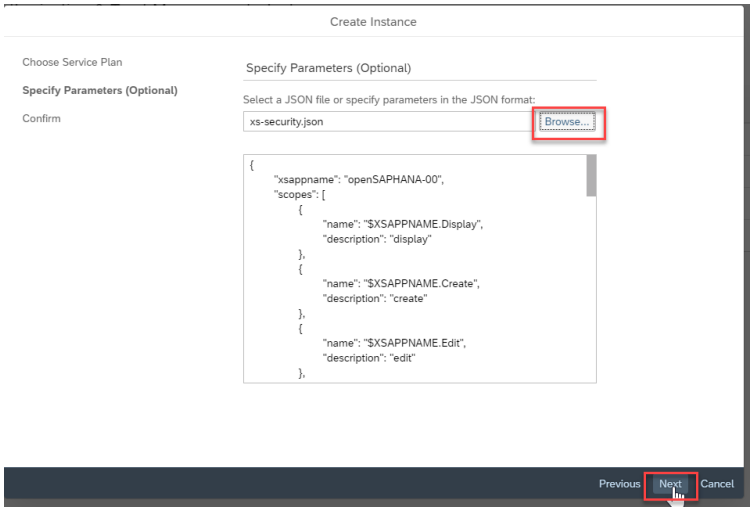


21. Choose Plan **space** and then press **Next**.

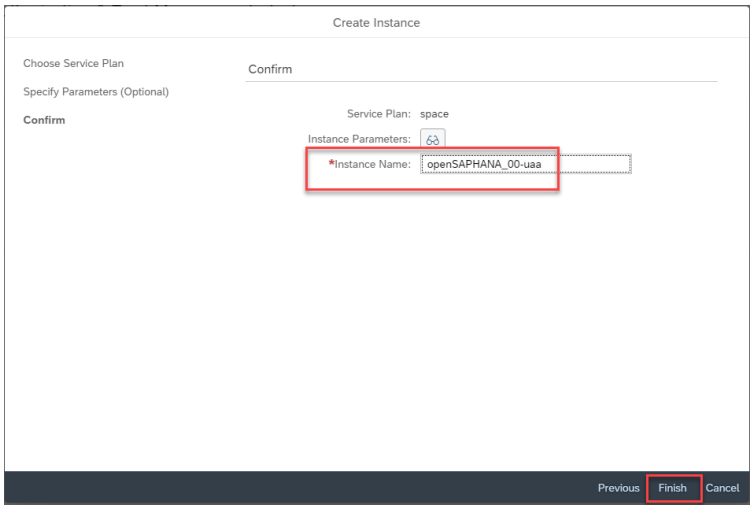


22. Press the Browse button and navigate to the location where you saved the xs-security.json file when you exported it a few steps ago.

    Then press **Next**.



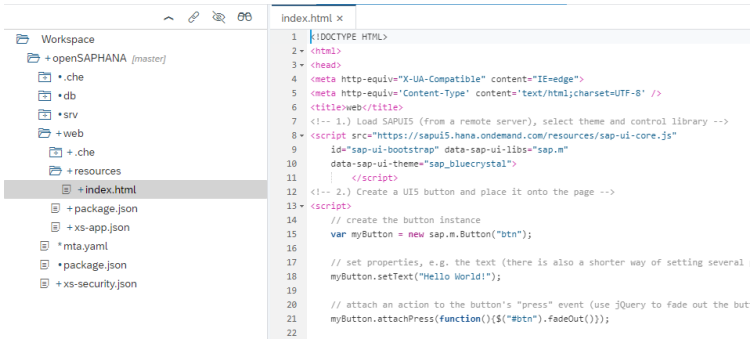
23. Specify an Instance Name of **openSAPHANA_XX-uaa** (where XX is your group number) and then press **Finish**



24. Return to the SAP Web IDE. The **web** folder in our project contains the resources that will be served out by this HTML5 module. This HTML5 module manages all HTML/client side UI resources (in the resources folder) and performs the task of reverse proxy for all other internal services. This way you have a single HTTP endpoint and avoid any CORS issues.

    The Add Module wizard already placed a simple index.html with “Hello World” in the resources folder.

    Note: There is no task to perform in this step. We are just pointing out content the wizard created already.

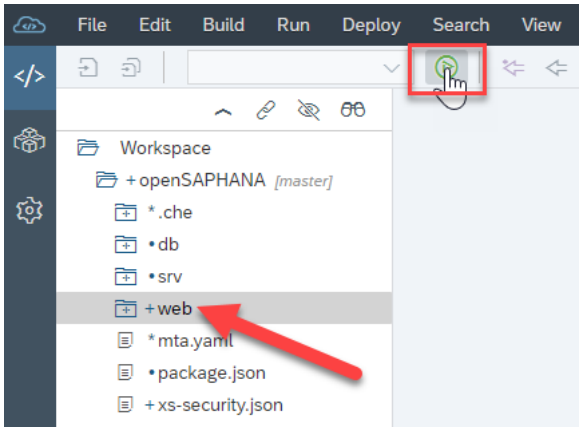
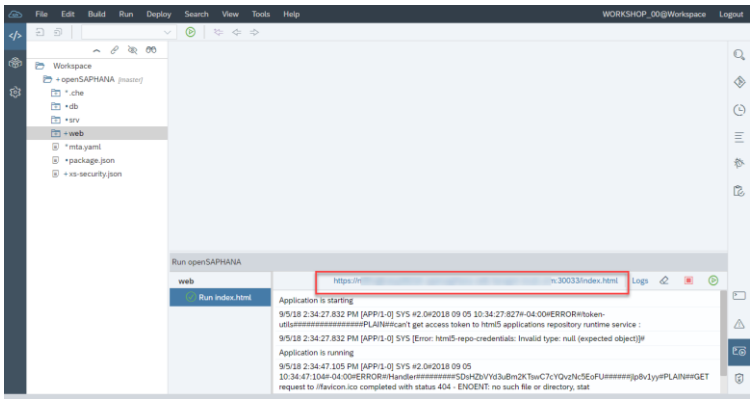


25. The HTML5 module **web** is configured via the file **xs-app.json**. In this file we can map the routes to destinations we defined in the mta.yaml. We can also set authentication and other options.

    Let’s go ahead and change the *authenticationMethod* to **route**.

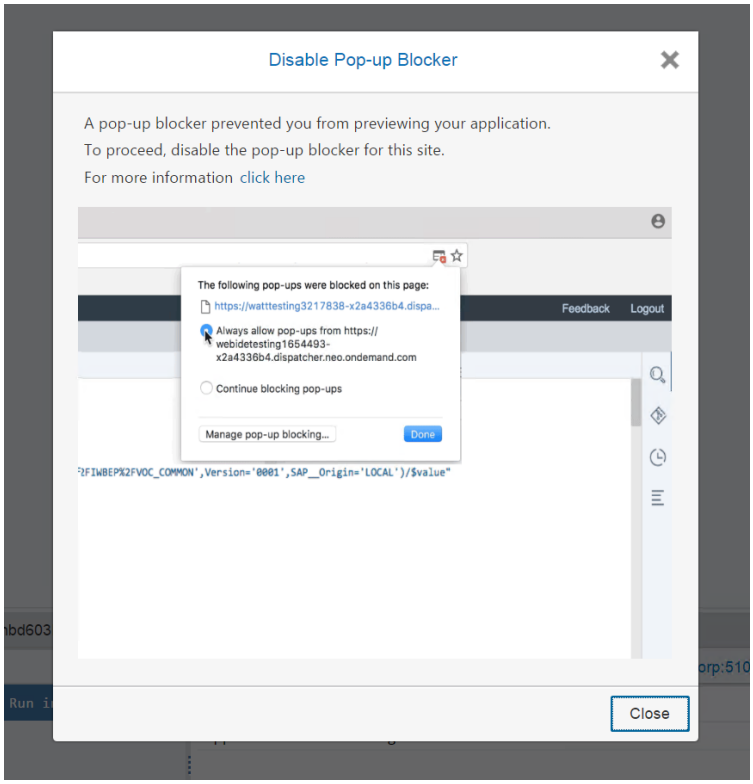
    Note: if you don’t want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex1/ex1_2

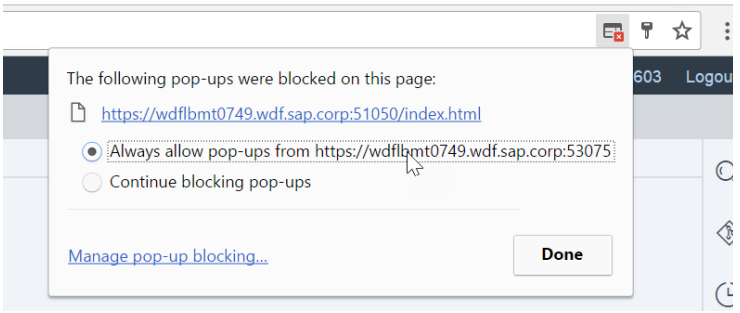
```
{
    "welcomeFile": "index.html",
    "authenticationMethod": "route",
    "routes": [ ]
}
```

26. Our initial development is done and we are ready to deploy our application onto the XS Advanced server. Highlight the web folder and press **Run**. This will perform a build, then deploy the service onto the server. If successful it will open a new browser tab to the default page of this web service.	
27. It can take a minute or two for the first build/deploy/run operation to complete. Upon completion you should see that the service status has changed to Running and there is a hyper link to the run logs.	

28. You might receive a message about the browser pop-up blocker. If you do receive this message press **Close**. Then click the pop-up configuration button on the right side of the browser address bar. Choose the **Always allow pop-ups from …** option.

    After you accept this popup, you can either run the module again or click the URL link to open it again.

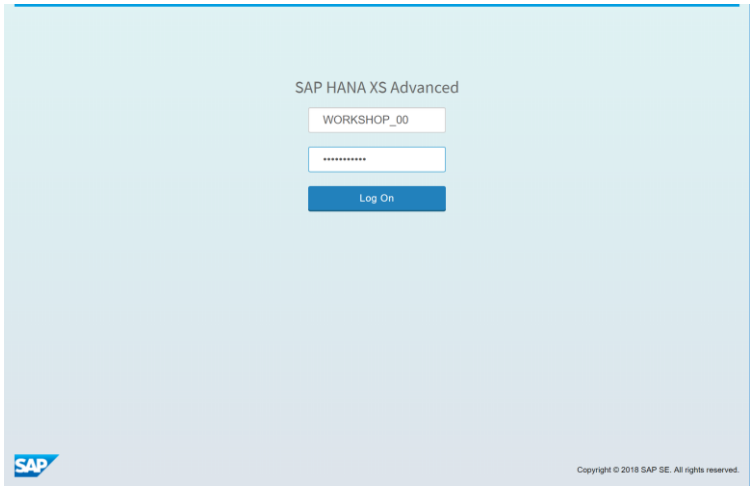


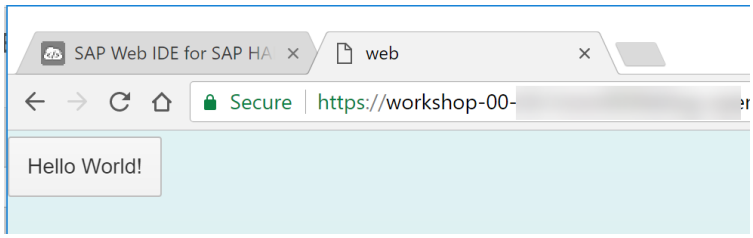


29. Switch your browser tab and you should see the authentication prompt for your application. Login with the same user credentials you used to log into the SAP Web IDE for SAP HANA.

    Note: You might not see this screen if you authentication is still cached in the browser from the login to the SAP Web IDE for SAP HANA itself.

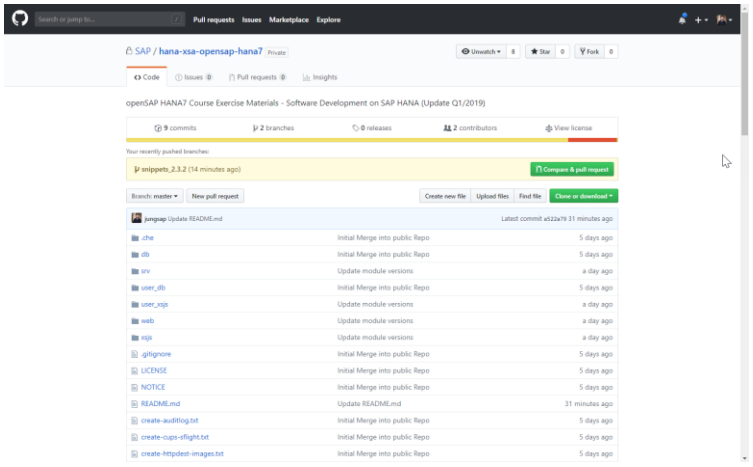
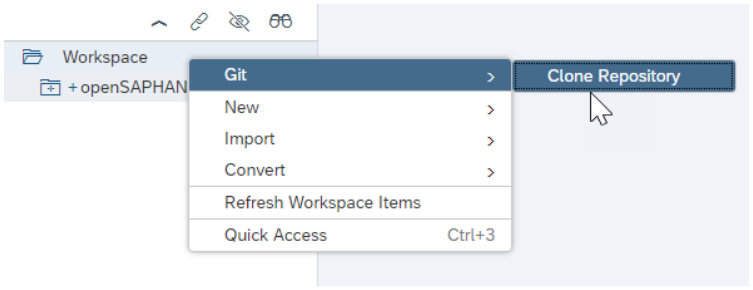
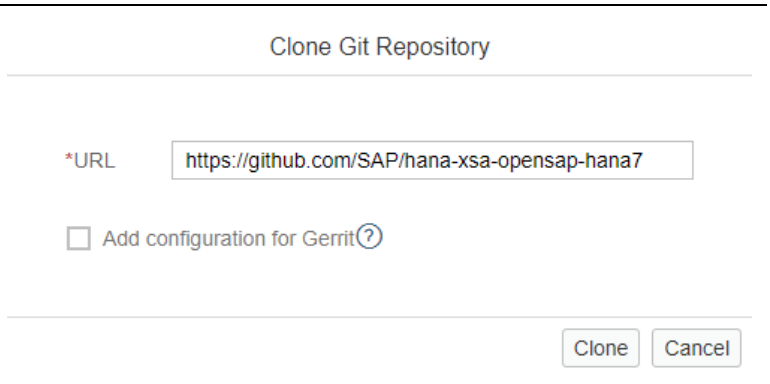
    Authentication at the XS level



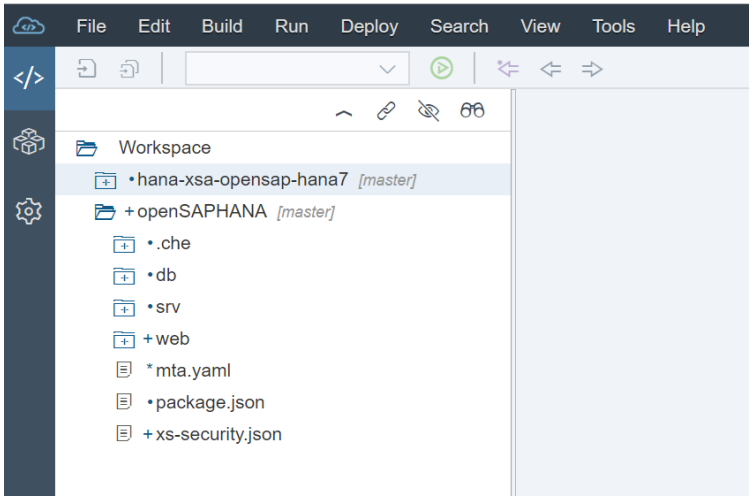
is now done by referencing a user store. This can be configured to be the HANA database or it can be an external user directory.	
30. After successful authentication, you should see your index.html with the Hello World button.	
31. Congratulations! You just wrote your first XS Advanced application.	

Exercise 1.2: Clone a Repository from Git

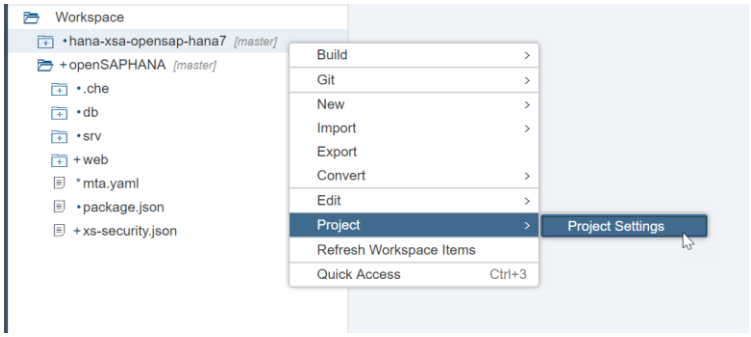
With XSA/HDI based development we no longer use the HANA database as the design time repository. We use a central Git to store all design time artifacts. Therefore when you launch the Web IDE for SAP HANA, you only see the projects which have been pulled from Git or which you have created locally in your workspace. There is no repository browser. In this exercise we will see how we can clone a project from Git to bring it into our local SAP Web IDE for SAP HANA workspace. You can alternatively import the project from the file system which will be shown in Exercise 1.3.

Explanation	Screenshot
1. Please navigate to this URL: https://github.com/SAP/hana-xsa-opensap-hana7 This is the example implementation project for this course which you can refer to if you have any problems.	
2. Return to the SAP Web IDE for SAP HANA. Right mouse click on the Workspace and choose **Git->Clone Repository**.	
3. In the Clone Repository dialog, please use the following values. **URL**: https://github.com/SAP/hana-xsa-opensap-hana7 Press **Clone**.	

4. If sucessful, the **hana-xsa-opensap-hana7** project will now appear in your workspace and is connected to the git repository but with the master branch.

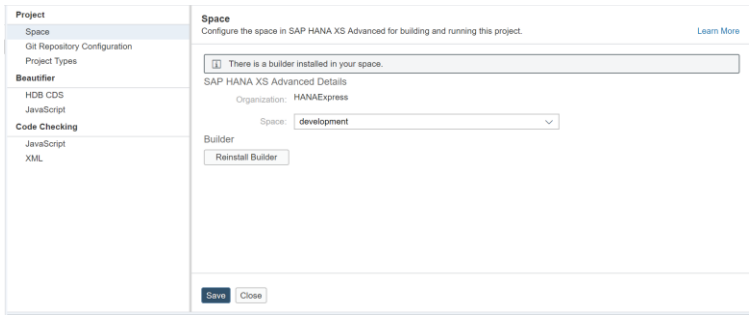


5. Right mouse click on your project and choose **Project -> Project Settings**.

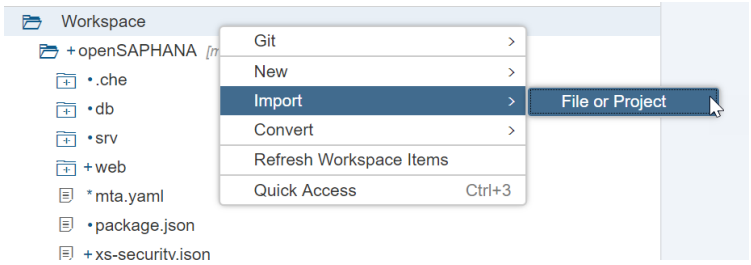
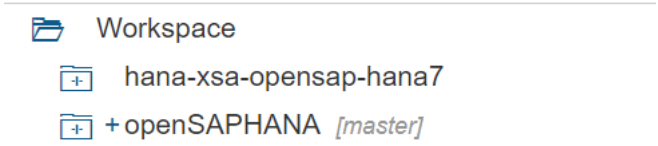


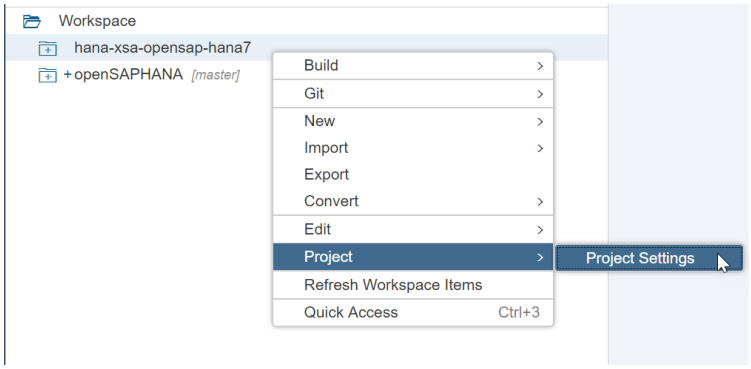
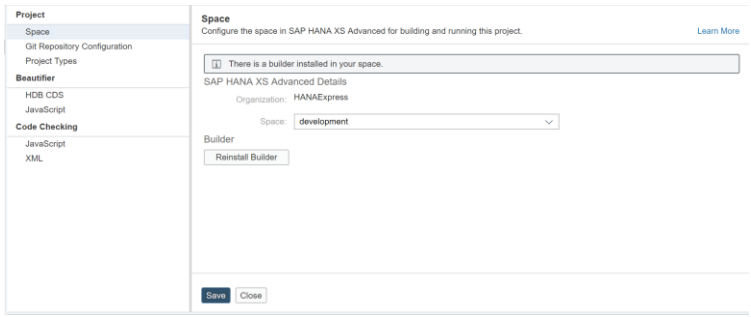
6. Choose the Space section and then choose **development** as the Development Space (or whatever system specific space your system has configured for you to perform development in). Press **Save**.

   If you have reached this step you can move ahead to Exercise 2. If you were not able to complete it for whatever reason, please move onto Exercise 1.3.



Exercise 1.3: Alternative: Import Project from File System

Explanation	Screenshot
1. Download a zip file with the complete project contents from the following location: https://github.com/SAP/hana-xsa-opensap-hana7/archive/master.zip	
2. Return to the SAP Web IDE for SAP HANA. Right mouse click on the Workspace and choose **Import -> From File System**.	
3. Click the browse button and from the File Open dialog choose the location where you download the file in step 1 of this exercise. Select the file and click **Open**. Remove the –snippets_2.3.2 portion of the Import to target and select Extract Archive. Press the OK button to import the template project.	
4. If sucessful, the **hana-xsa-opensap-hana7** project will now appear in your workspace.	

5. Right mouse click on your project and choose **Project -> Project Settings**	
6. Choose the **Space** section and then choose **development** as the **Development Space** (or whatever system specific space your system has configured for you to perform development in). Press **Save**.	

EXERCISE 2 –DATABASE ARTIFACT DEVELOPMENT

Objective

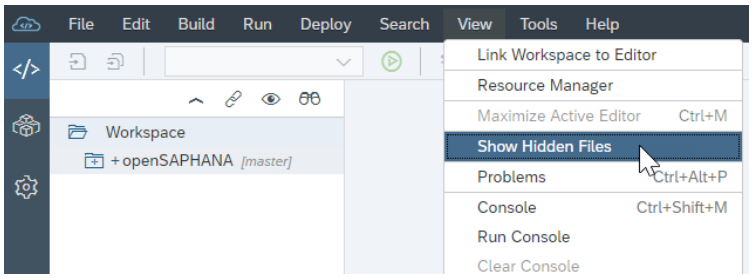
In this exercise, we will continue to develop our overall application. Applications in HANA/XS Advanced are often made up of multiple modules at design time which deploy to separate micro-services or database container content. We created client side UI application content in the first exercise using the HTML5 module. In this exercise we will create database artifacts, such as database table, stored procedures and user defined functions, using the HDB (HANA Database) module. We will then see how we build these database artifacts using the new container-based, schema-less HDI (HANA Deployment Infrastructure) concepts.

Exercise Description

- Database Tables via HDBCDS
- Stored Procedures via HDBPROCEDURE
- User Defined Functions via HDBFUNCTION
- Initial table data load via CSV
- Deploy to HANA via HDI
- Containers and Synonyms
- Calculation Views
- SQLScript

EXERCISE 2 – SOLUTION

Exercise 2.1: Database Content

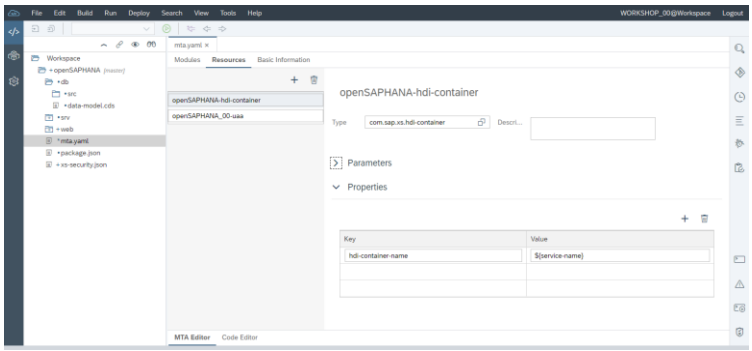
Explanation	Screenshot
1. New to HANA in 1.0 SPS 11 was the HANA Deployment Infrastructure or HDI. The goal of HDI is to manage database artifacts from design time objects but in a way that allows multiple copies/versions of the same core objects to be used on the same HANA database at the same time. HDI introduces the concept of the container as an abstraction of the Schema. The container in turn dynamically generates the schema, a container-specific Database User who owns all objects, and a password for that database user. XS Advanced based services then only need access to the container and never need to know the actual Schema, technical user, or password. All of that information is stored within the container definition.	
2. Choose **View->Show Hidden Files**.	

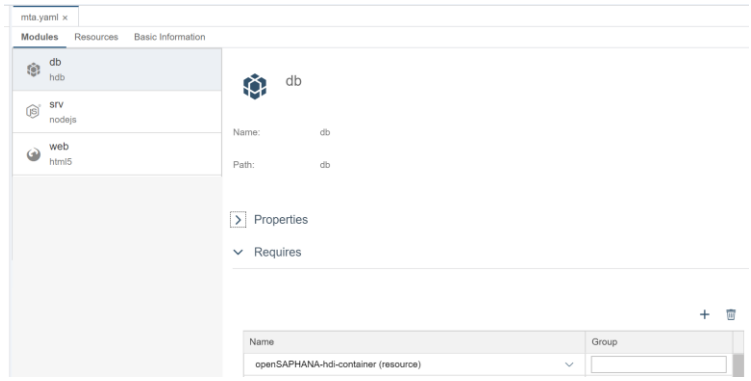
3. The Project Wizard generated a file named **.hdiconfig** in the **/db/src** folder. Unfortunately it uses an old version of HANA in this file which will cause us problems later. We should correct this value to point to **your HANA version** (at least 2.0.30 – which is HANA 2.0 SPS 03).



4. The project wizard from the first exercise has created the **db** folder as well as the **openSAPHANA-hdi-container** resource and the **db** module in the mta.yaml file for you.

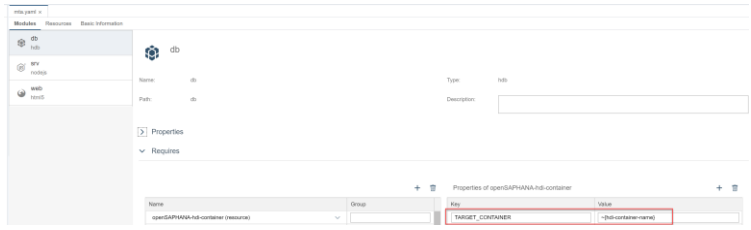
   Note: There is no task to perform in this step. We are just pointing out content the wizard created already

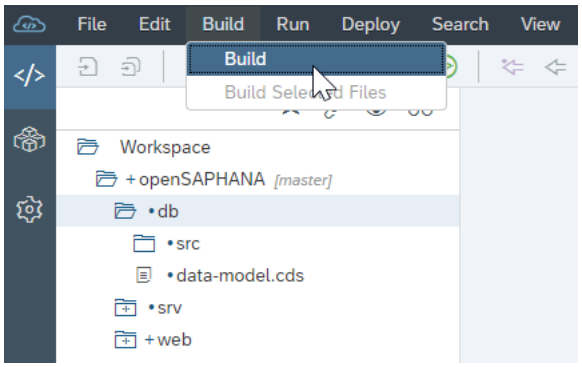
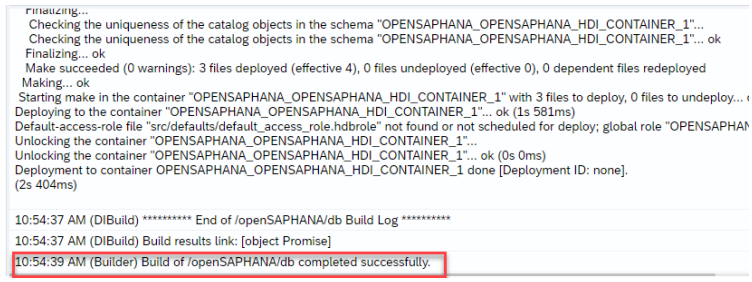
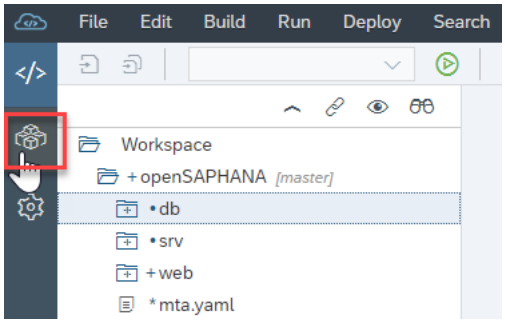




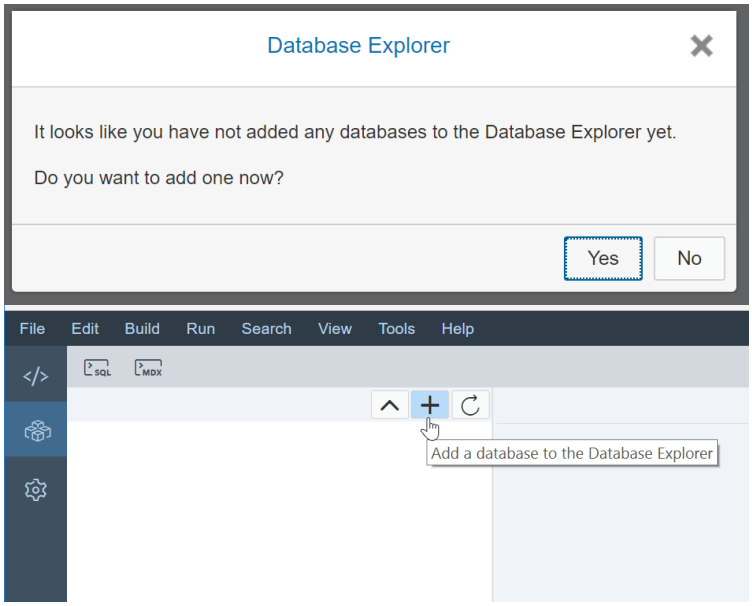
5. An HDB module can also have more than one database resource associated with it. However it can only have one primary resouce which is the target for all database development objects it will create. Later we will add more resources for cross container access.

   This requires us to set in **mta.yaml** this resource as the **TARGET_CONTAINER** in order to describe it as the primary database

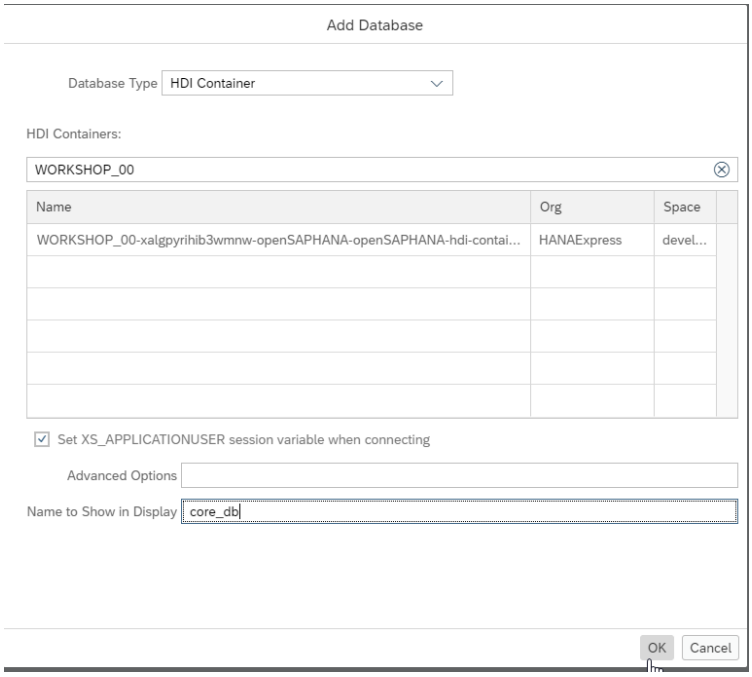


resource for this module. properties: TARGET_CONTAINER: ~{hdi-container-name} Save the file.	
6. Select the **db** module and choose **Build->Build**.	
7. You should see a message that you build was sucessful.	
8. Go to the **Database Explorer** by choosing the Database Explorer icon on the left side of the SAP Web IDE for SAP HANA.	

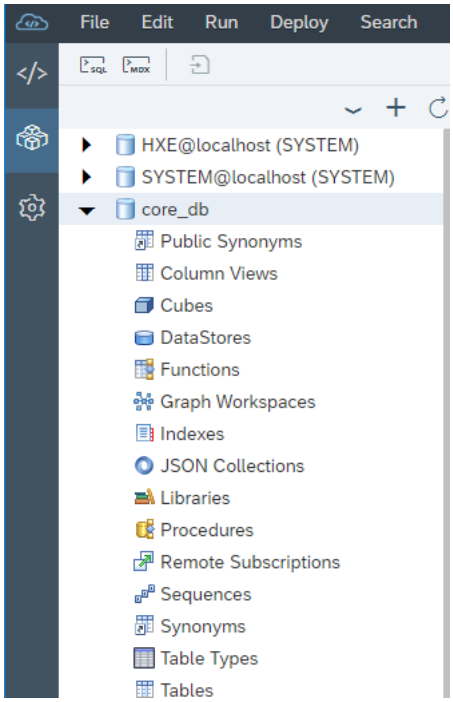
9. If your Database Explorer is empty and has no connections, you will receive a dialog Choose Yes to add a new connection. Otherwise click on the **Add** button.



10. Choose **HDI Container** as your Database type. Search for your container based upon your User Name. Set a display name of **core_db**. Then click **OK**.

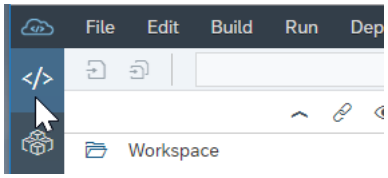
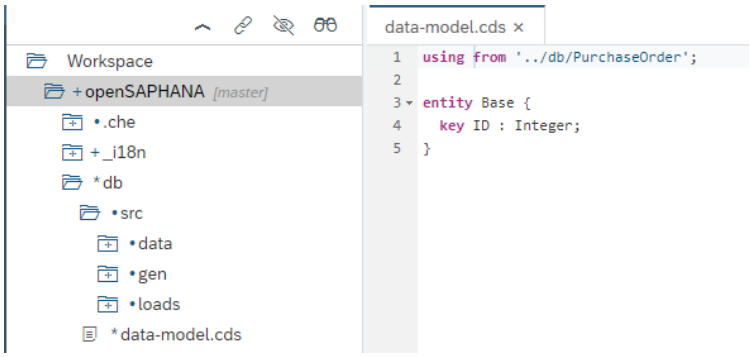


11. You can now see your container in which your database artifacts will be deployed. We will begin creating database artifacts in the next exercise.



Exercise 2.2: Create Tables & Views via Core Data Services

In this exercise you will create a Purchase Order header and item table as well as a view over those two tables using Core Data Services (CDS).

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA, Development perspective.	
2. Navigate to the **data-model.cds** in the **db** folder and open it for editing. This is a new version of CDS that replaces HDBCDS. The “compiler” is on the client side and is no longer HANA specific. We will use this CDS to define database artifacts in a database agnostic fashion. There is a single entry point called **data-model.cds**, but we want to modularize our content into separate files. Therefore we will just put references to these other files/objects in data-model.cds. **Note**: if you don’t want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_1a	

3. Now create another file in the **db** folder named **PurchaseOrder.cds**.

   This file contains the table and view definitions. We are creating a simple Purchase Order Header and Item data model.

   The syntax is similar but not the same as HDBCDS-based development objects previously.

   Here is the code for PurchaseOrder.cds

   Save the artifact after entering this code.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_1

```
namespace opensap.PurchaseOrder;

  type BusinessKey : String(10);
  type SDate : DateTime;
  type CurrencyT : String(5);
  type AmountT : Decimal(15, 2);
  type QuantityT : Decimal(13, 3);
  type UnitT : String(3);
  type StatusT : String(1);

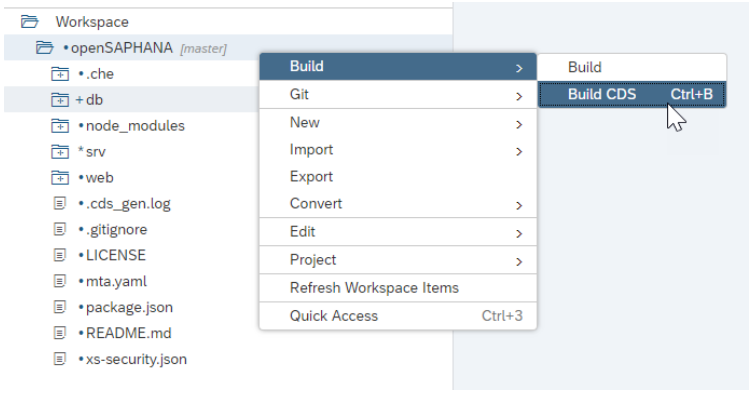
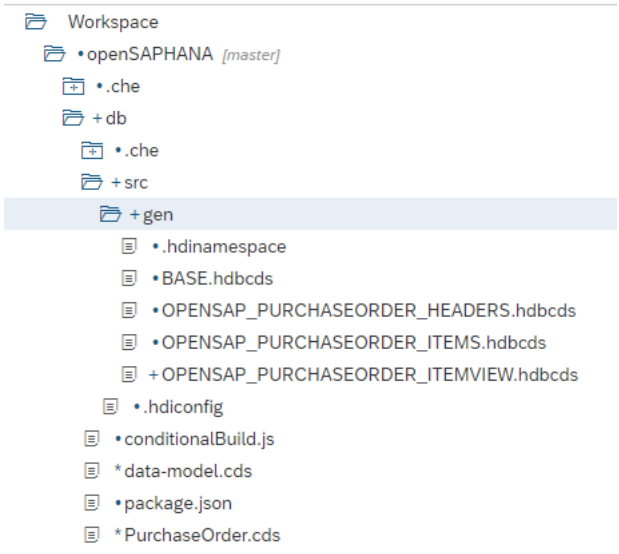
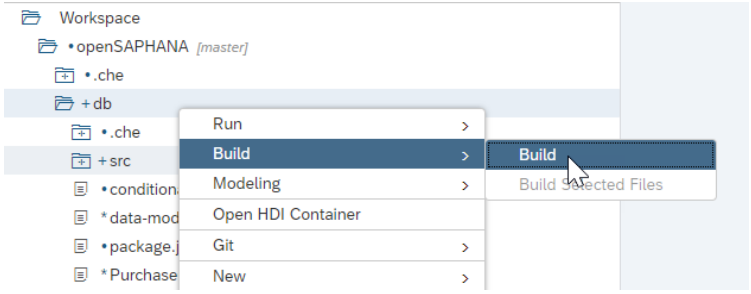
       type HistoryT {
    CREATEDBY : BusinessKey;
    CREATEDAT : SDate;
    CHANGEDBY : BusinessKey;
    CHANGEDAT : SDate;
  };

  entity Headers {
    key PURCHASEORDERID : Integer;
      ITEMS          : association to many Items on
ITEMS.POHeader = $self;
      HISTORY        : HistoryT;
      NOTEID         : BusinessKey null;
      PARTNER        : BusinessKey;
      CURRENCY       : CurrencyT;
      GROSSAMOUNT    : AmountT;
      NETAMOUNT      : AmountT;
      TAXAMOUNT      : AmountT;
      LIFECYCLESTATUS : StatusT;
      APPROVALSTATUS  : StatusT;
      CONFIRMSTATUS   : StatusT;
      ORDERINGSTATUS  : StatusT;
      INVOICINGSTATUS : StatusT;
  }

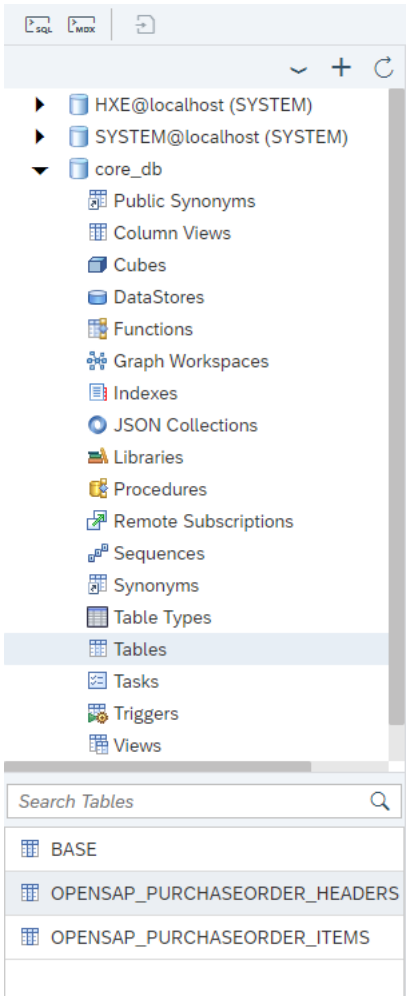
  entity Items {
    key POHeader    : association to Headers;
    key PRODUCT     : BusinessKey;
      NOTEID      : BusinessKey null;
      CURRENCY    : CurrencyT;
      GROSSAMOUNT  : AmountT;
      NETAMOUNT    : AmountT;
      TAXAMOUNT    : AmountT;
      QUANTITY    : QuantityT;
      QUANTITYUNIT : UnitT;
      DELIVERYDATE : SDate;
  }

       define view ItemView as
    select from Items
    {
      POHeader.PURCHASEORDERID,
      POHeader.PARTNER,
      PRODUCT,
```

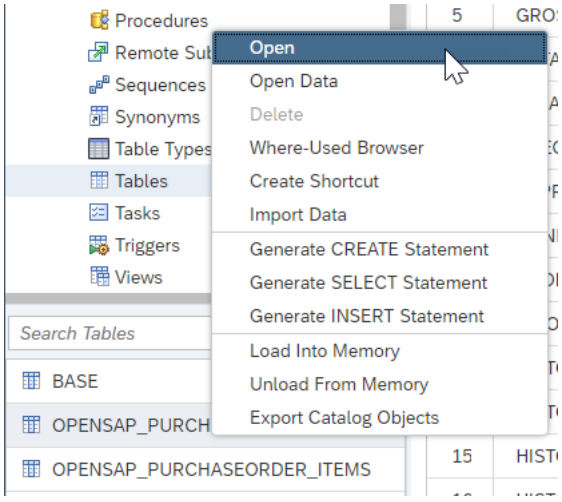
	CURRENCY, GROSSAMOUNT, NETAMOUNT, TAXAMOUNT, QUANTITY, QUANTITYUNIT, DELIVERYDATE };
4. Let's look at the syntax you just entered into the hdbcds file in more detail. First we need to define some reusable elemental types. These will later be used to define the data type of individual columns in our tables. Within the PurchaseOrder context, create element types for BusinessKey, SDate, CurrencyT, AmountT, QuantityT, UnitT, and StatusT.	namespace opensap.PurchaseOrder; type BusinessKey : String(10); type SDate : DateTime; type CurrencyT : String(5); type AmountT : Decimal(15, 2); type QuantityT : Decimal(13, 3); type UnitT : String(3); type StatusT : String(1);
5. We can also create reusable structures with multiple fields. This is useful when the same sets of fields are repeated in multiple tables. Create a reusable structure for History – with CREATEDBY, CREATEDAT, CHANGEDBY, and CHANGEDAT fields.	type HistoryT { CREATEDBY : BusinessKey; CREATEDAT : SDate; CHANGEDBY : BusinessKey; CHANGEDAT : SDate; };
6. The syntax for creating Entities is similar to types. Entities will become database tables when building the cds file.	entity Headers { key PURCHASEORDERID : Integer; ITEMS : association to many Items on ITEMS.POHeader = $self; HISTORY : HistoryT; NOTEID : BusinessKey null; PARTNER : BusinessKey; CURRENCY : CurrencyT; GROSSAMOUNT : AmountT; NETAMOUNT : AmountT; TAXAMOUNT : AmountT; LIFECYCLESTATUS : StatusT; APPROVALSTATUS : StatusT; CONFIRMSTATUS : StatusT; ORDERINGSTATUS : StatusT; INVOICINGSTATUS : StatusT; }

7. Before we build, we need to remove some of the sample content from the CDS service in **/srv/my-service.cds** that no longer matches our new CDS content. Within the /srv/my-service.cds remove all the content except the service definitino for CatalogService	service CatalogService { }
8. This new CDS has to first build to database specific files. So after saving all your .cds files, right mouse click on the project level and choose **Build -> Build CDS.**	
9. The previous **Build CDS** step doesn't actually create anything in the database. Instead it just "compiles" the CDS files into HANA specific HDBCDS files in the **db/src/gen** folder.	
10. To actually create the content within the HANA database, we have to now build the **db** module by right mouse clicking on the db folder and choosing **Build -> Build**.	

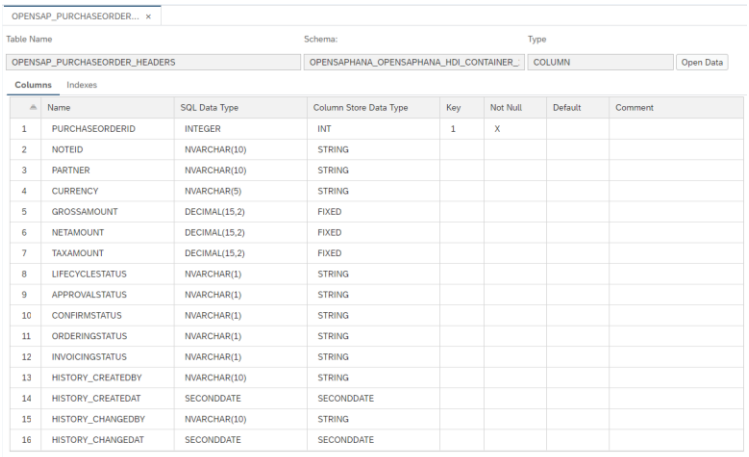
11. Switch over to the **Database Explorer** and browse your container, expand the **Tables** and **Views** folders and explore the new tables and view



12. Right-click on the table **OPENSAP_PURCHASEORDER_HEADERS** and choose **Open**

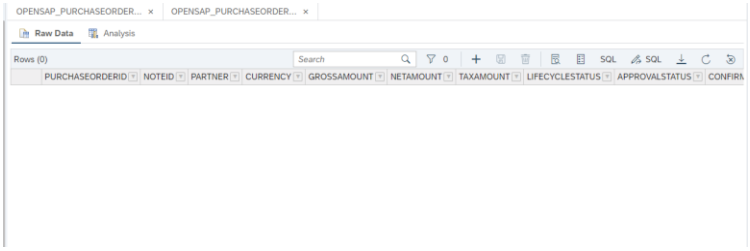


13. Explore the table definition, notice the column names where we used the complex type definition called *History*



	Name	SQL Data Type	Column Store Data Type	Key	Not Null	Default	Comment
1	PURCHASEORDERID	INTEGER	INT	1	X		
2	NOTEID	NVARCHAR(10)	STRING				
3	PARTNER	NVARCHAR(10)	STRING				
4	CURRENCY	NVARCHAR(5)	STRING				
5	GROSSAMOUNT	DECIMAL(15,2)	FIXED				
6	NETAMOUNT	DECIMAL(15,2)	FIXED				
7	TAXAMOUNT	DECIMAL(15,2)	FIXED				
8	LIFECYCLESTATUS	NVARCHAR(1)	STRING				
9	APPROVALSTATUS	NVARCHAR(1)	STRING				
10	CONFIRMSTATUS	NVARCHAR(1)	STRING				
11	ORDERINGSTATUS	NVARCHAR(1)	STRING				
12	INVOICINGSTATUS	NVARCHAR(1)	STRING				
13	HISTORY_CREATEDBY	NVARCHAR(10)	STRING				
14	HISTORY_CREATEDAT	SECONDDATE	SECONDDATE				
15	HISTORY_CHANGEDBY	NVARCHAR(10)	STRING				
16	HISTORY_CHANGEDAT	SECONDDATE	SECONDDATE				

14. And you can query the data in these objects via the **Open Data** button, although there won't be any data there yet. We will see how we can load an initial set of data in the next exercise.



Exercise 2.3: Create Table Data Configuration

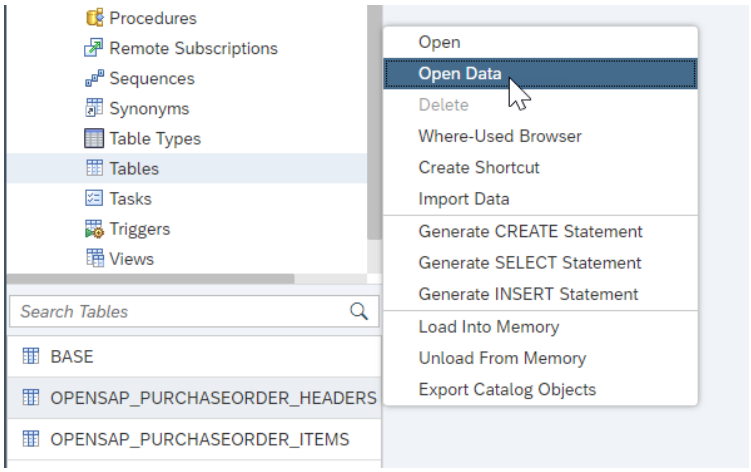
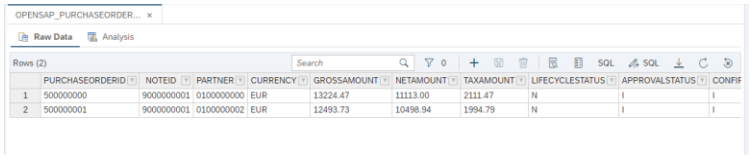
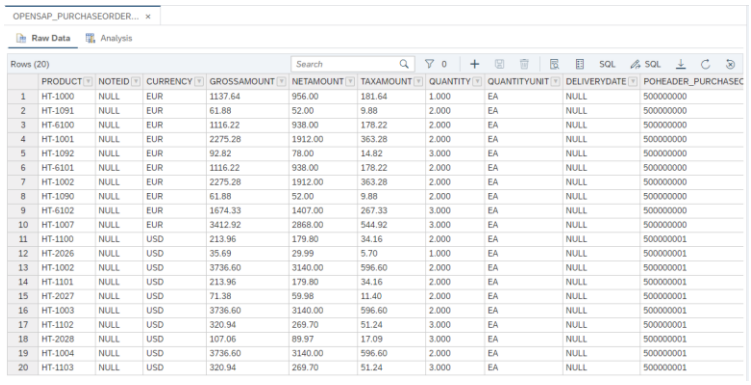
In this exercise, you will create table data configuration file which will load data into your tables during the deploy process.

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA, Development perspective. You may want to deliver an initial set of data within a table – particular a configuration table. In this exercises we will learn how to create automatic data load configuration and the accompanying CSV files for just such a situation. The data load for table requires two files – 1. An csv (comma separated) file which holds the data you want to load. 2. An hdbtabledata file which specifies the target table for a source csv file.	
2. We also have the hdbtabledata development object. The purpose is to allow the loading of initial data from CSV files it target tables during their creation. Create a file named **PurchaseCDS.hdbtabledata** in a new folder named **/db/src/loads** and enter this text into it. Don't forget to save the file afterwards. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_3	{ "format_version": 1, "imports": [{ "target_table": "OPENSAP_PURCHASEORDER_HEADERS", "source_data": { "data_type": "CSV", "file_name": "headerCDS.csv", "has_header": false, "dialect": "HANA", "type_config": { "delimiter": "," } }, "import_settings": { "include_filter" : [ ], "import_columns": ["PURCHASEORDERID", "NOTEID", "PARTNER", "CURRENCY", "GROSSAMOUNT", "NETAMOUNT", "TAXAMOUNT", "LIFECYCLESTATUS",

```
                                "APPROVALSTATUS",
                                "CONFIRMSTATUS",
                                "ORDERINGSTATUS",
                                "INVOICINGSTATUS"]
                        },
                        "column_mappings": {
                                "PURCHASEORDERID": 1,
                                "NOTEID": 6,
                                "PARTNER": 7,
                                "CURRENCY": 8,
                                "GROSSAMOUNT": 9,
                                "NETAMOUNT": 10,
                                "TAXAMOUNT": 11,
                                "LIFECYCLESTATUS": 12,
                                "APPROVALSTATUS": 13,
                                "CONFIRMSTATUS": 14,
                                "ORDERINGSTATUS": 15,
                                "INVOICINGSTATUS": 16
                        }
                },
                {
                        "target_table":
"OPENSAP_PURCHASEORDER_ITEMS",
                        "source_data": {
                                "data_type": "CSV",
                                "file_name": "itemCDS.csv",
                                "has_header": false,
                                "dialect": "HANA",
                                "type_config": {
                                        "delimiter": ","
                                }
                        },
                        "import_settings": {
                "include_filter" : [ ],
                                "import_columns":
["POHEADER_PURCHASEORDERID",
                                "PRODUCT",
                                "NOTEID",
                                "CURRENCY",
                                "GROSSAMOUNT",
                                "NETAMOUNT",
                                "TAXAMOUNT",
                                "QUANTITY",
                                "QUANTITYUNIT" ]
                        },
                        "column_mappings": {

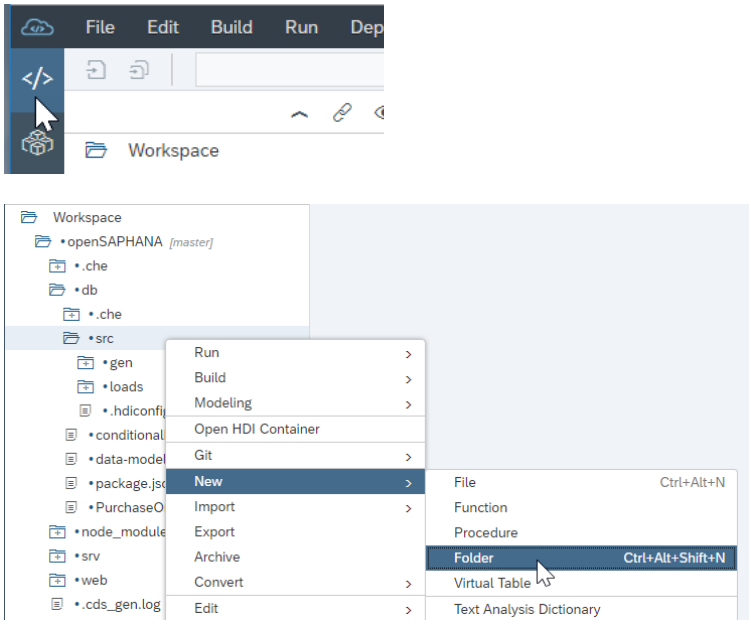
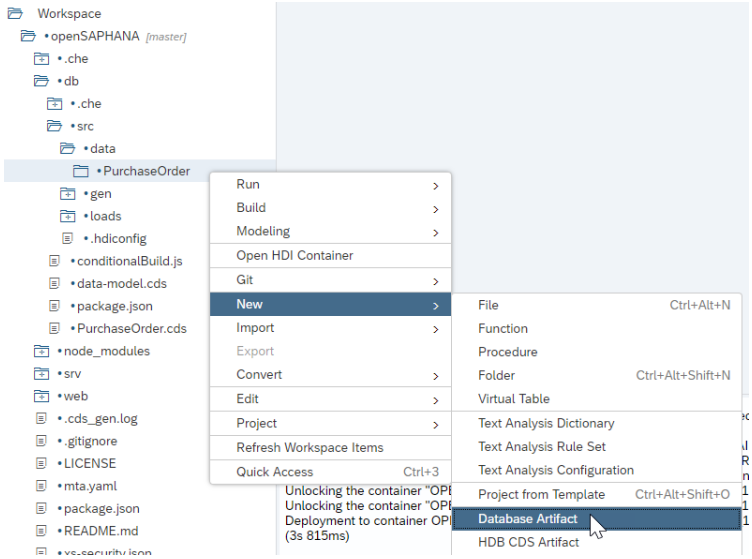
                "POHEADER_PURCHASEORDERID": 1,
                                "PRODUCT": 3,
                                "NOTEID": 4,
                                "CURRENCY": 5,
                                "GROSSAMOUNT": 6,
                                "NETAMOUNT": 7,
                                "TAXAMOUNT": 8,
```

	"QUANTITY": 9, "QUANTITYUNIT": 10 } }] }
3. We need some CSV files to hold some initial test data to be loaded by the hdbtabledata configuration file. Enter this data into a file named **headerCDS.csv** in the **loads** folder and save it. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_4	0500000000,0000000033,20120101,0000000033,20120101,9000000001,0100000000,EUR,13224.47,11113,2111.47,N,I,I,I,I 0500000001,0000000033,20120102,0000000033,20120102,9000000001,0100000002,EUR,12493.73,10498.94,1994.79,N,I,I,I,I
4. And data for the item table named **itemCDS.csv**. Don't forget to save. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_5	0500000000,0000000010,HT-1000,,EUR,1137.64,956,181.64,1,EA,20121204 0500000000,0000000020,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204 0500000000,0000000030,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204 0500000000,0000000040,HT-1001,,EUR,2275.28,1912,363.28,2,EA,20121204 0500000000,0000000050,HT-1092,,EUR,92.82,78,14.82,3,EA,20121204 0500000000,0000000060,HT-6101,,EUR,1116.22,938,178.22,2,EA,20121204 0500000000,0000000070,HT-1002,,EUR,2275.28,1912,363.28,2,EA,20121204 0500000000,0000000080,HT-1090,,EUR,61.88,52,9.88,2,EA,20121204 0500000000,0000000090,HT-6102,,EUR,1674.33,1407,267.33,3,EA,20121204 0500000000,0000000100,HT-1007,,EUR,3412.92,2868,544.92,3,EA,20121204 0500000001,0000000010,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204 0500000001,0000000020,HT-2026,,USD,35.69,29.99,5.7,1,EA,20121204 0500000001,0000000030,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000040,HT-1101,,USD,213.96,179.8,34.16,2,EA,20121204

	0500000001,0000000050,HT-2027,,USD,71.38,59.98,11.4,2,EA,20121204 0500000001,0000000060,HT-1003,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000070,HT-1102,,USD,320.94,269.7,51.24,3,EA,20121204 0500000001,0000000080,HT-2028,,USD,107.06,89.97,17.09,3,EA,20121204 0500000001,0000000090,HT-1004,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000100,HT-1103,,USD,320.94,269.7,51.24,3,EA,20121204
5. Save all files and build your **db** module again. Return to the **Database Exploer** perspective and right-click on the **OPENSAP_PURCHASEORDER_HEADERS** table and choose **Open Data**.	
6. You will notice that your table now has some data.	
7. Repeat these steps to view the data in the **OPENSAP_PURCHASEORDER_ITEMS** table as well.	

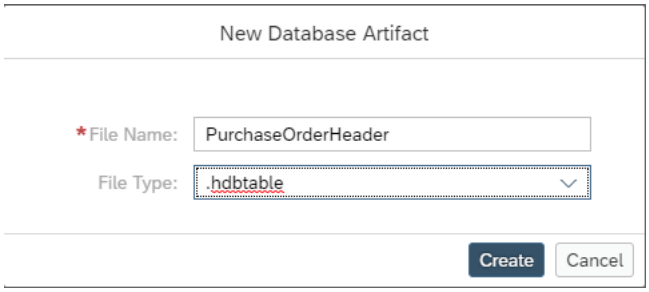
Exercise 2.4: Create Tables & Views via SQL DDL Design Time Artifacts

While Core Data Services is good for abstracted database modeling (and for service and UI modeling as we will see later), it doesn't give us access to HANA specific features or data types. When performing HANA Native Development, you often are doing so specifically for such features. Therefore most of your database modeling is probably better served to be done via HANA specific artifacts rather than CDS when you want such HANA specific development. In this exercise we will see how to use the SQL DDL (data definition language) design time artifacts. We will retain all the lifecycle management benefits of a design time based approach we used with CDS in the past but instead use the HANA SQL based syntax for the definition of the objects.

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA, Development perspective. Create a new folder under the **db/src** folder by right-clicking it and choosing **New->Folder.** Name the folder **data.** Repeat the process to create a folder called **PurchaseOrder** within the **data** folder you just created.	
2. Now create the hdbtable artifact via **New->Database Arifact**.	

3. To help you compare to the previous exercise, we are going to re-create the same purchase order objects now using SQL DDL.

   Name the new file **PurchaseOrderHeader**, use **.hdbtable** as the File Type and press **Create**.



4. Unlike CDS, we only put one DB artifact per file when using SQL DDL based artifacts like HDBTABLE.

   Here is the code for **PurchaseOrderHeader.hdbtable**

   Save the artifact after entering this code.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_1b

```
COLUMN TABLE "PurchaseOrder.Header" (
    "PURCHASEORDERID" INTEGER CS_INT
GENERATED BY DEFAULT AS IDENTITY (NO CYCLE NO
CACHE NO MINVALUE START WITH 200000000
INCREMENT BY 1 MAXVALUE 2999999999) NOT NULL
COMMENT 'Purchase Order ID',
        "HISTORY.CREATEDBY" NVARCHAR(10)
COMMENT 'Created By',
        "HISTORY.CREATEDAT" DATE CS_DAYDATE
COMMENT 'Created Date',
        "HISTORY.CHANGEDBY" NVARCHAR(10)
COMMENT 'Changed By',
        "HISTORY.CHANGEDAT" DATE CS_DAYDATE
COMMENT 'Change Date',
        "NOTEID" NVARCHAR(10) COMMENT 'Notes',
        "PARTNER" NVARCHAR(10) COMMENT
'Supplier',
        "CURRENCY" NVARCHAR(5) COMMENT
'Currency',
        "GROSSAMOUNT" DECIMAL(15,2) CS_FIXED
COMMENT 'Gross Amount',
        "NETAMOUNT" DECIMAL(15,2) CS_FIXED
COMMENT 'Net Amount',
        "TAXAMOUNT" DECIMAL(15,2) CS_FIXED
COMMENT 'Tax Amount',
        "LIFECYCLESTATUS" NVARCHAR(1) COMMENT
'Lifecycle Status',
        "APPROVALSTATUS" NVARCHAR(1) COMMENT
'Approval Status',
        "CONFIRMSTATUS" NVARCHAR(1) COMMENT
'Confirmation Status',
        "ORDERINGSTATUS" NVARCHAR(1) COMMENT
'Ordering Status',
        "INVOICINGSTATUS" NVARCHAR(1) COMMENT
'Invoicing Status',
        PRIMARY KEY ("PURCHASEORDERID"))
        COMMENT 'Purchase Order Header'
        WITH ASSOCIATIONS( JOIN
"PurchaseOrder.Item" AS "ITEMS" ON
"PURCHASEORDERID" = "PURCHASEORDERID")
        UNLOAD PRIORITY 5 AUTO MERGE
```

5. Repeat the process to create a new file nameed **PurchaseOrderItem.hdbtable** and fill it with this code.

   Save the artifact after entering this code.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_1c

```
COLUMN TABLE "PurchaseOrder.Item" (
    "POHeader.PURCHASEORDERID" INTEGER CS_INT NOT NULL COMMENT 'Purchase Order ID',
        "PRODUCT" NVARCHAR(10) NOT NULL COMMENT 'Product ID',
        "NOTEID" NVARCHAR(10) COMMENT 'Notes',
        "CURRENCY" NVARCHAR(5) COMMENT 'Currency',
        "GROSSAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Gross Amount',
        "NETAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Net Amount',
        "TAXAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Tax Amount',
        "QUANTITY" DECIMAL(13,3) CS_FIXED COMMENT 'Quantity',
        "QUANTITYUNIT" NVARCHAR(3) COMMENT 'Quantity Unit',
        "DELIVERYDATE" DATE CS_DAYDATE COMMENT 'Delivery Date',
        PRIMARY KEY ("POHeader.PURCHASEORDERID","PRODUCT"))
        COMMENT 'Purchase Order Item'
        WITH ASSOCIATIONS( JOIN "PurchaseOrder.Header" AS "POHeader" ON "POHeader"."PURCHASEORDERID" = "POHeader.PURCHASEORDERID")
        UNLOAD PRIORITY 5 AUTO MERGE
```

6. Create one last file named **PurchaseOrderItemView.hdbview**. This time we will be creating a view that joins the previous two tables.

   Save the artifact after entering this code.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_1d

```
VIEW "PurchaseOrder.ItemView" COMMENT 'Purchase Order Item View' (
    "PurchaseOrderItemId",
        "PartnerId",
        "ProductID",
        "CurrencyCode",
        "Amount",
        "NetAmount",
        "TaxAmount",
        "Quantity",
        "QuantityUnit",
        "DeliveryDate1"
        ) AS SELECT
        "Item_$0"."POHeader"."PURCHASEORDERID" AS "PurchaseOrderItemId" ,
        "Item_$0"."POHeader"."PARTNER" AS "PartnerId" ,
        "Item_$0"."PRODUCT" AS "ProductID" ,
        "Item_$0"."CURRENCY" AS "CurrencyCode" ,
        "Item_$0"."GROSSAMOUNT" AS "Amount" ,
        "Item_$0"."NETAMOUNT" AS "NetAmount" ,
        "Item_$0"."TAXAMOUNT" AS "TaxAmount" ,
        "Item_$0"."QUANTITY" AS "Quantity" ,
        "Item_$0"."QUANTITYUNIT" AS "QuantityUnit" ,
        "Item_$0"."DELIVERYDATE" AS "DeliveryDate1"
FROM "PurchaseOrder.Item" AS "Item_$0"
WITH READ ONLY STRUCTURED PRIVILEGE CHECK
```

7. Just as we did in the previous exercise we want to use the hdbtabledata development object to allow the loading of initial data from CSV files it target tables during their creation.

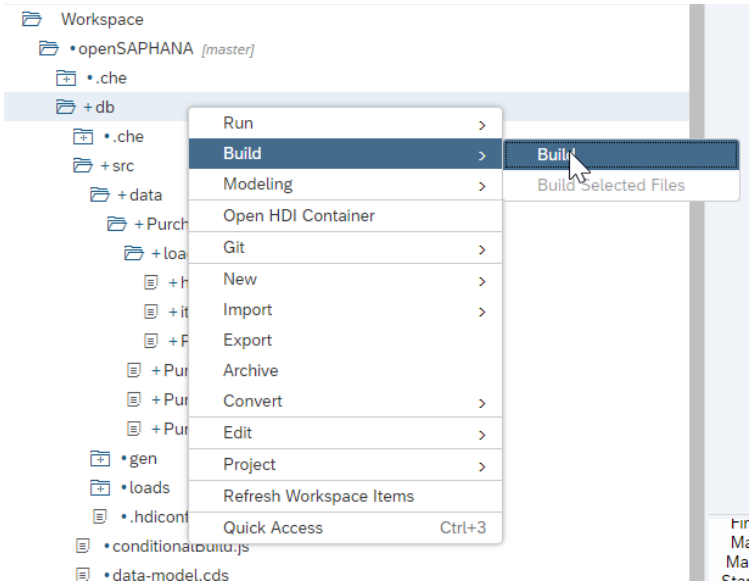
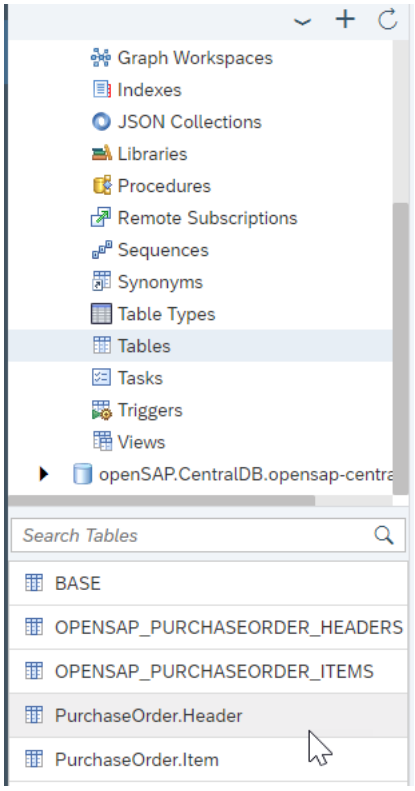
   Create a file named **PurchaseDemo.hdbtabledata** in a new folder named **/db/src/data/PurchaseOrder/loads** and enter this text into it. Don't forget to save the file afterwards.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_3a

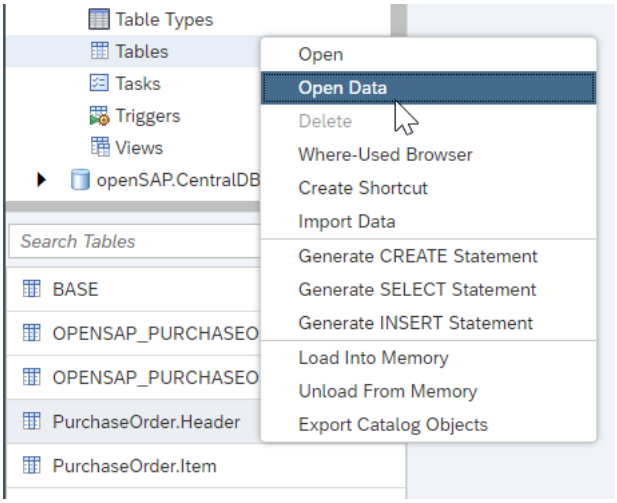
```
{
	"format_version": 1,
	"imports": [{
		"target_table": "PurchaseOrder.Header",
		"source_data": {
			"data_type": "CSV",
			"file_name": "header.csv",
			"has_header": false,
			"dialect": "HANA",
			"type_config": {
				"delimiter": ","
			}
		},
		"import_settings": {
	"include_filter" : [ ],
			"import_columns":
["PURCHASEORDERID",
			"NOTEID",
			"PARTNER",
			"CURRENCY",
			"GROSSAMOUNT",
			"NETAMOUNT",
			"TAXAMOUNT",
			"LIFECYCLESTATUS",
			"APPROVALSTATUS",
			"CONFIRMSTATUS",
			"ORDERINGSTATUS",
			"INVOICINGSTATUS"]
		},
		"column_mappings": {
			"PURCHASEORDERID": 1,
			"NOTEID": 6,
			"PARTNER": 7,
			"CURRENCY": 8,
			"GROSSAMOUNT": 9,
			"NETAMOUNT": 10,
			"TAXAMOUNT": 11,
			"LIFECYCLESTATUS": 12,
			"APPROVALSTATUS": 13,
			"CONFIRMSTATUS": 14,
			"ORDERINGSTATUS": 15,
			"INVOICINGSTATUS": 16
		}
	},
	{
		"target_table": "PurchaseOrder.Item",
		"source_data": {
			"data_type": "CSV",
			"file_name": "item.csv",
			"has_header": false,
			"dialect": "HANA",
			"type_config": {
				"delimiter": ","
			}
```

	}, "import_settings": { "include_filter" : [ ], "import_columns": ["POHeader.PURCHASEORDERID", "PRODUCT", "NOTEID", "CURRENCY", "GROSSAMOUNT", "NETAMOUNT", "TAXAMOUNT", "QUANTITY", "QUANTITYUNIT"] }, "column_mappings": { "POHeader.PURCHASEORDERID": 1, "PRODUCT": 3, "NOTEID": 4, "CURRENCY": 5, "GROSSAMOUNT": 6, "NETAMOUNT": 7, "TAXAMOUNT": 8, "QUANTITY": 9, "QUANTITYUNIT": 10 } }] }
8. We need some CSV files to hold some initial test data to be loaded by the hdbtabledata configuration file. Enter this data into a file named **header.csv** in the **PurchaseOrder/loads** folder and save it. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_4	0500000000,0000000033,20120101,0000000033,20120101,9 000000001,0100000000,EUR,13224.47,11113,2111.47,N,I,I,I,I 0500000001,0000000033,20120102,0000000033,20120102,9 000000001,0100000002,EUR,12493.73,10498.94,1994.79,N,I, I,I,I
9. And data for the item table named **item.csv**. Don't forget to save. **Note**: if you don't want to type this code, we recommend that	0500000000,0000000010,HT- 1000,,EUR,1137.64,956,181.64,1,EA,20121204 0500000000,0000000020,HT- 1091,,EUR,61.88,52,9.88,2,EA,20121204

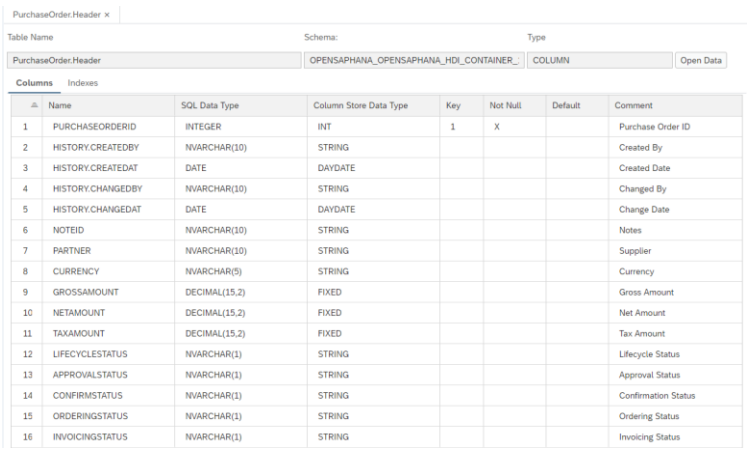
you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_5	0500000000,0000000030,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204 0500000000,0000000040,HT-1001,,EUR,2275.28,1912,363.28,2,EA,20121204 0500000000,0000000050,HT-1092,,EUR,92.82,78,14.82,3,EA,20121204 0500000000,0000000060,HT-6101,,EUR,1116.22,938,178.22,2,EA,20121204 0500000000,0000000070,HT-1002,,EUR,2275.28,1912,363.28,2,EA,20121204 0500000000,0000000080,HT-1090,,EUR,61.88,52,9.88,2,EA,20121204 0500000000,0000000090,HT-6102,,EUR,1674.33,1407,267.33,3,EA,20121204 0500000000,0000000100,HT-1007,,EUR,3412.92,2868,544.92,3,EA,20121204 0500000001,0000000010,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204 0500000001,0000000020,HT-2026,,USD,35.69,29.99,5.7,1,EA,20121204 0500000001,0000000030,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000040,HT-1101,,USD,213.96,179.8,34.16,2,EA,20121204 0500000001,0000000050,HT-2027,,USD,71.38,59.98,11.4,2,EA,20121204 0500000001,0000000060,HT-1003,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000070,HT-1102,,USD,320.94,269.7,51.24,3,EA,20121204 0500000001,0000000080,HT-2028,,USD,107.06,89.97,17.09,3,EA,20121204 0500000001,0000000090,HT-1004,,USD,3736.6,3140,596.6,2,EA,20121204 0500000001,0000000100,HT-1103,,USD,320.94,269.7,51.24,3,EA,20121204

10. Save your files and build the **db** module again.	
11. Switch over to the Database Explorer and browse your container, expand the Tables and Views folders and explore the new tables and view	

12. Right-click on the **PurchaseOrder.Header** table and choose **Open**

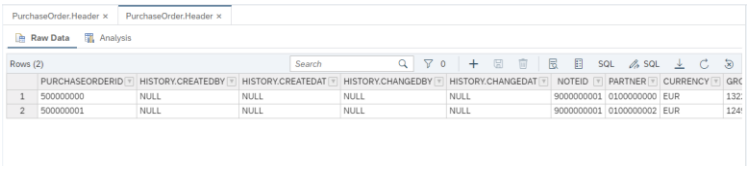


13. Explore the table definition, notice the column names where we used the complex type definition called **History**



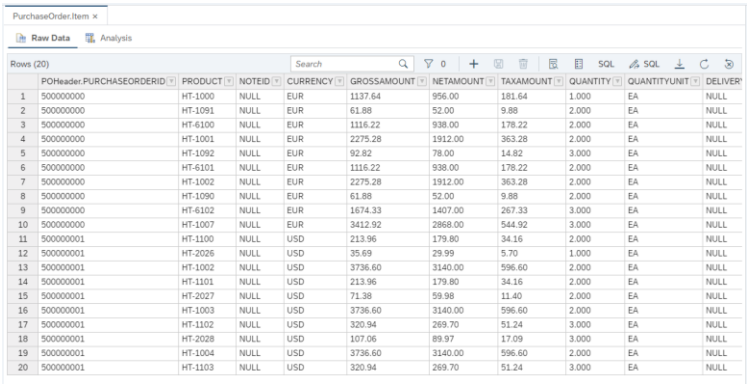
	Name	SQL Data Type	Column Store Data Type	Key	Not Null	Default	Comment
1	PURCHASEORDERID	INTEGER	INT	1	X		Purchase Order ID
2	HISTORY.CREATEDBY	NVARCHAR(10)	STRING				Created By
3	HISTORY.CREATEDAT	DATE	DAYDATE				Created Date
4	HISTORY.CHANGEDBY	NVARCHAR(10)	STRING				Changed By
5	HISTORY.CHANGEDAT	DATE	DAYDATE				Change Date
6	NOTEID	NVARCHAR(10)	STRING				Notes
7	PARTNER	NVARCHAR(10)	STRING				Supplier
8	CURRENCY	NVARCHAR(5)	STRING				Currency
9	GROSSAMOUNT	DECIMAL(15,2)	FIXED				Gross Amount
10	NETAMOUNT	DECIMAL(15,2)	FIXED				Net Amount
11	TAXAMOUNT	DECIMAL(15,2)	FIXED				Tax Amount
12	LIFECYCLESTATUS	NVARCHAR(1)	STRING				Lifecycle Status
13	APPROVALSTATUS	NVARCHAR(1)	STRING				Approval Status
14	CONFIRMSTATUS	NVARCHAR(1)	STRING				Confirmation Status
15	ORDERINGSTATUS	NVARCHAR(1)	STRING				Ordering Status
16	INVOICINGSTATUS	NVARCHAR(1)	STRING				Invoicing Status

14. You will notice that your table now has some data as well.



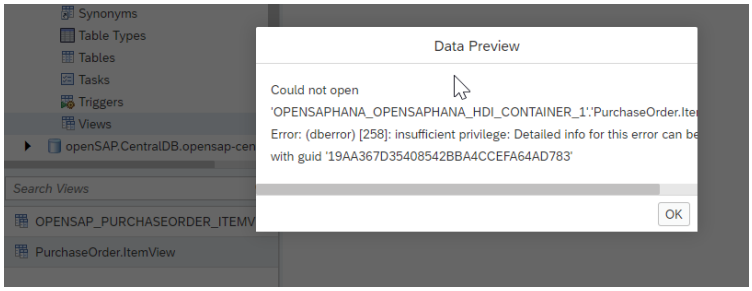
	PURCHASEORDERID	HISTORY.CREATEDBY	HISTORY.CREATEDAT	HISTORY.CHANGEDBY	HISTORY.CHANGEDAT	NOTEID	PARTNER	CURRENCY	GR
1	500000000	NULL	NULL	NULL	NULL	9000000001	0100000000	EUR	132
2	500000001	NULL	NULL	NULL	NULL	9000000001	0100000002	EUR	124

15. Repeat these steps to view the data in the **PurchaseOrder.Item** table as well.



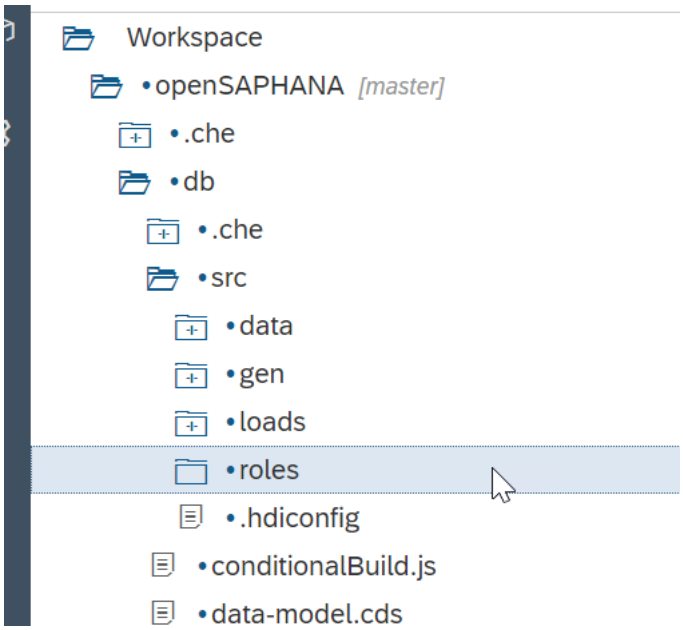
	POHeader.PURCHASEORDERID	PRODUCT	NOTEID	CURRENCY	GROSSAMOUNT	NETAMOUNT	TAXAMOUNT	QUANTITY	QUANTITYUNIT	DELIVER
1	500000000	HT-1000	NULL	EUR	1137.64	956.00	181.64	1.000	EA	NULL
2	500000000	HT-1091	NULL	EUR	61.88	52.00	9.88	2.000	EA	NULL
3	500000000	HT-6100	NULL	EUR	1116.22	938.00	178.22	2.000	EA	NULL
4	500000000	HT-1001	NULL	EUR	2275.28	1912.00	363.28	2.000	EA	NULL
5	500000000	HT-1092	NULL	EUR	92.82	78.00	14.82	3.000	EA	NULL
6	500000000	HT-6101	NULL	EUR	1116.22	938.00	178.22	2.000	EA	NULL
7	500000000	HT-1002	NULL	EUR	2275.28	1912.00	363.28	2.000	EA	NULL
8	500000000	HT-1090	NULL	EUR	61.88	52.00	9.88	2.000	EA	NULL
9	500000000	HT-6102	NULL	EUR	1674.33	1407.00	267.33	3.000	EA	NULL
10	500000000	HT-1007	NULL	EUR	3412.92	2868.00	544.92	3.000	EA	NULL
11	500000001	HT-1100	NULL	USD	213.96	179.80	34.16	2.000	EA	NULL
12	500000001	HT-2026	NULL	USD	35.69	29.99	5.70	1.000	EA	NULL
13	500000001	HT-1002	NULL	USD	3736.60	3140.00	596.60	2.000	EA	NULL
14	500000001	HT-1101	NULL	USD	213.96	179.80	34.16	2.000	EA	NULL
15	500000001	HT-2027	NULL	USD	71.38	59.98	11.40	2.000	EA	NULL
16	500000001	HT-1003	NULL	USD	3736.60	3140.00	596.60	2.000	EA	NULL
17	500000001	HT-1102	NULL	USD	320.94	269.70	51.24	3.000	EA	NULL
18	500000001	HT-2028	NULL	USD	107.06	89.97	17.09	3.000	EA	NULL
19	500000001	HT-1004	NULL	USD	3736.60	3140.00	596.60	2.000	EA	NULL
20	500000001	HT-1103	NULL	USD	320.94	269.70	51.24	3.000	EA	NULL

16. If you try to access the data in the view, however, even the technical user of the container doesn't have access because of the structured privilege we placed upon it. This structured privilege is a HANA specific feature and one of the reasons to use SQL DDL instead of CDS. We will solve this later as well once we setup some roles in our container.



Exercise 2.5: Create Structured Privilege & Role

In this exercise, you will create a structured privilege and role and assign it to you user.

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA, Development perspective.	
2. All access to our HDI database objects from XSA is done automatically by the HDI container technical user. However if you want to allow access via other database users (for use cases such as external reporting tools); you must create a database role. Also if you want to grant additional privileges to the technical user (such as the structured privleges for our SQL view), we will also need a database role. Return to the SAP Web IDE for SAP HANA and create another folder called **roles** by right-clicking on the **src** folder and choosing **New->Folder**.	
3. First we need to create a structured privilege. This is closely related to the analytic privilege and allows us to perform instance filtering for our CDS view we created earlier. Enter this code into the file **PurchaseOrder.hdbstructuredprivilege** in the **roles** folder and save. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-	STRUCTURED PRIVILEGE "PO_VIEW_PRIVILEGE" FOR SELECT ON "PurchaseOrder.ItemView" WHERE "CurrencyCode" = 'EUR'

xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_6 This will limit the access to our view to only allow users with this privilege to see items for Euros.	
4. Now create a file named **admin.hdbrole** in the **roles** folder and enter this code. Don't forget to save. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_7	{ "role":{ "name": "admin", "schema_privileges": [{ "privileges": ["SELECT METADATA", "SELECT CDS METADATA", "SELECT", "INSERT", "EXECUTE", "DELETE", "UPDATE", "CREATE TEMPORARY TABLE"] }], "schema_analytic_privileges": [{ "privileges":["PO_VIEW_PRIVILEGE"] }] } }
5. Create another role named **adminOwner.hdbrole** in the **roles** folder. This role name will end with a # (admin#). This means it will be given the "With Grant" option. This will allow the container technical user to grant this role to other containers. This is how we expose our content to other containers. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_7a	{ "role":{ "name": "admin#", "schema_roles": [{ "names": ["admin"] }] } }

6. We want this role to be automatically granted to our technical container application user. Therefore we need to create a role with a special name – **default_access_role**.

   Create a folder named **defaults** in **src**. Inside this folder create a file named **default_access_role.hdbrole**

   Enter the following text. This will grant the **admin.hdbrole** we created in the previous step to our technical user automatically upon build.

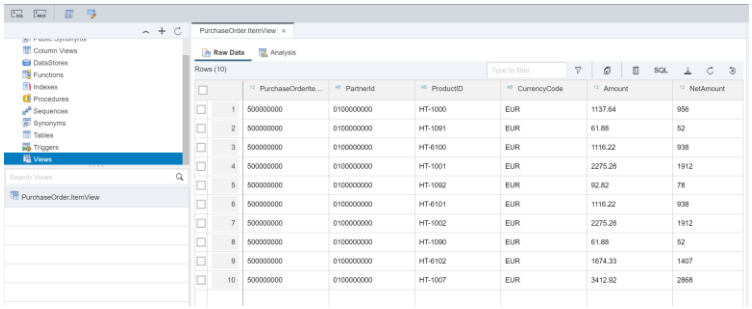
   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_8

```
{
        "role":{
                "name": "default_access_role",
                "schema_roles": [{
                 "names": ["admin"]
                }]
        }
}
```

7. Save all files and build the **db** module. Return to the **Database Explorer** perspective.

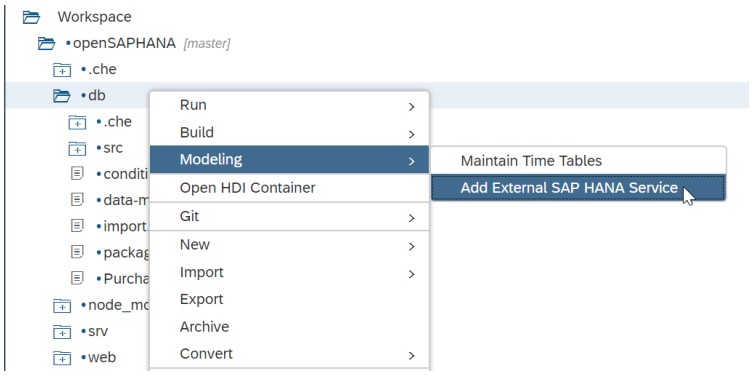
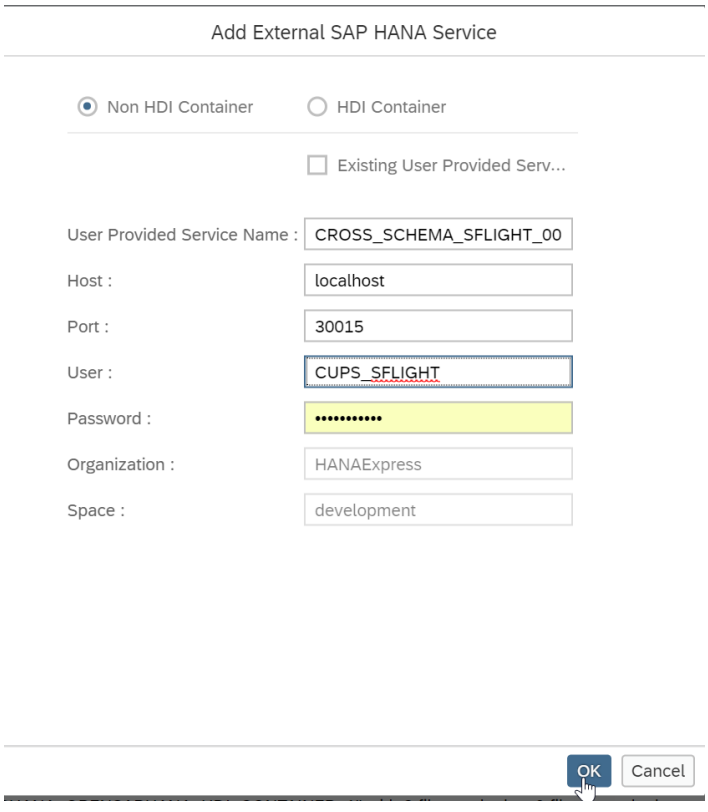
8. Try to access the data in the view again and it should be sucessful this time, but you will only be able to see the records with currency of EUR (thanks to the filtering of the Sturctured Privilege).

   Note: In order to access the structure privilege based view or for a database user other than the technical user to access any of the catalog objects; they need the database role granted to their user.

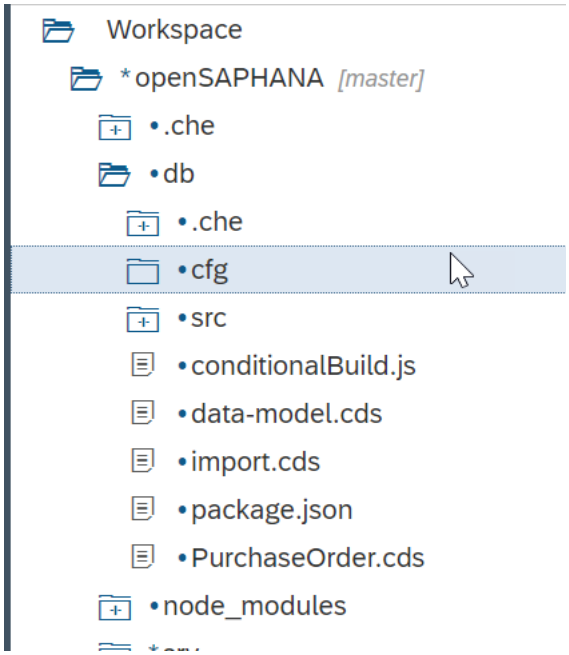


Exercise 2.6: Non-Container Schemas, User Provided Services & Synonyms

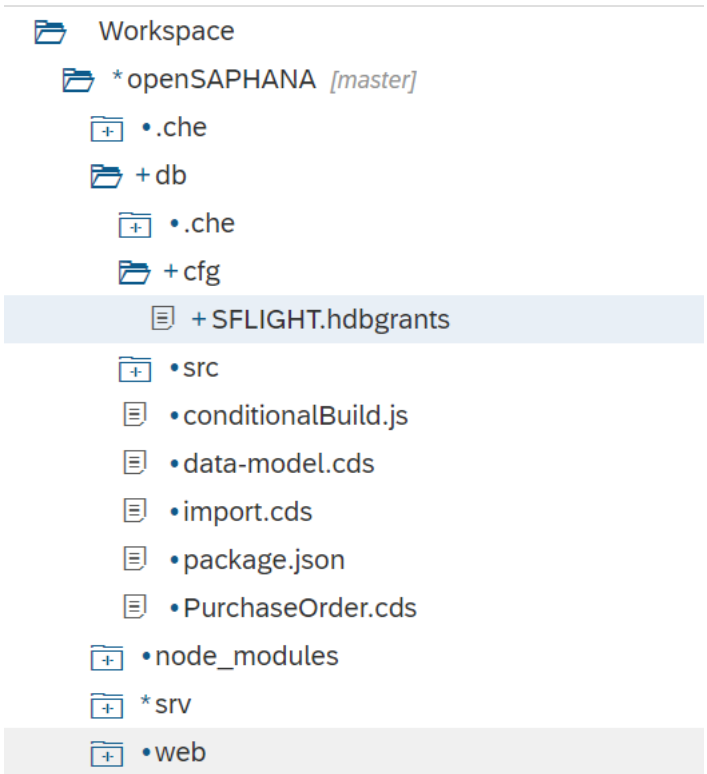
In this exercise, we will create a synonym in order to access a table within a non-container traditional schema. This exercise requires that you have the SFLIGHT catalog schema installed in your system and a database user who has access to this schema. You can find more information on loading SFLIGHT into your system, if it doesn't exist already, here:
https://blogs.sap.com/2018/12/18/howto-import-sflight-sample-data-into-sap-hana-from-a-local-computer/

Explanation	Screenshot
1. Right mouse clck on the database module and choose **Modeling -> Add External SAP HANA Service**	
2. A User Provided Service allows our modules to connect to existing database schemas which aren't managed by HDI. This would be the case for replicated schemas or ERP ABAP schemas for instance. Create a service named: **CROSS_SCHEMA_SFLIGHT_<group number>** Be sure replace the placeholder for group number, hostname, instance number, service port (15 for for single container systems or MDC systems which are using the first tenant and 13 for MDC systems using the system DB), user (CUPS_SFLIGHT), and password (HanaRocks01).	

3. Create a new folder called **cfg** under the **db** folder.



4. Create a new file in the cfg folder called **SFLIGHT.hdbgrants** by right-clicking on the cfg folder and choosing **New->File**



5. Enter the following code. This is assigning access to the SFLIGHT schema to both the container technical users – owner and application user. The grant is performed by the user provided service and the database user you configured in that user provided service.

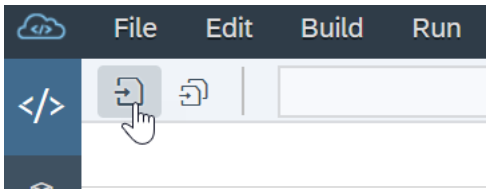
   If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_12

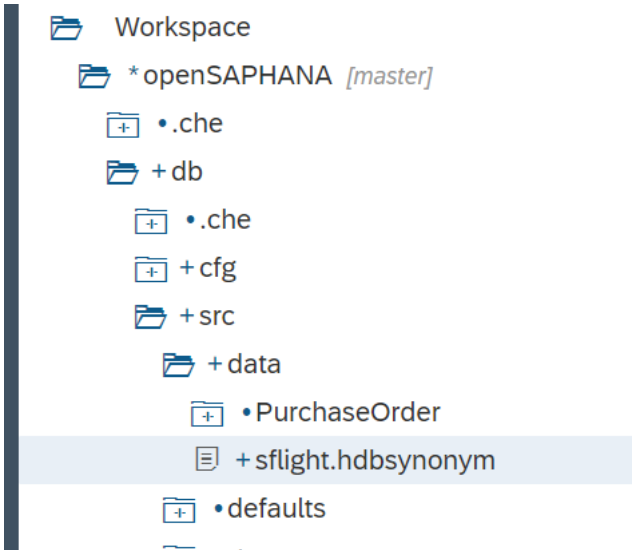
SFLIGHT.hdbgrants ×

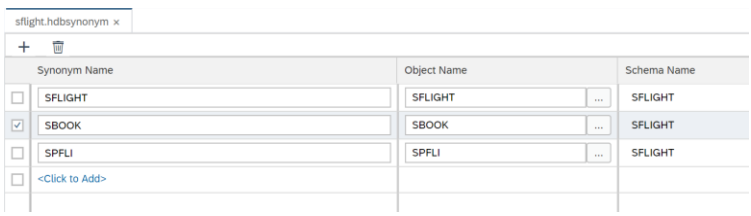
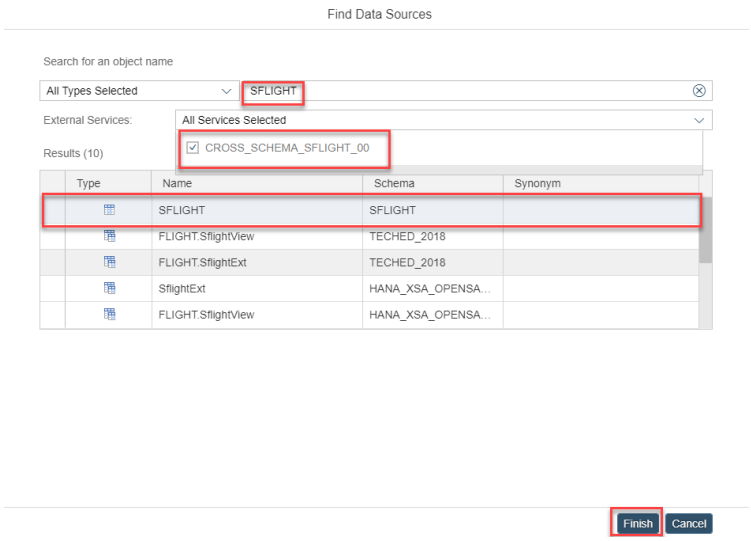
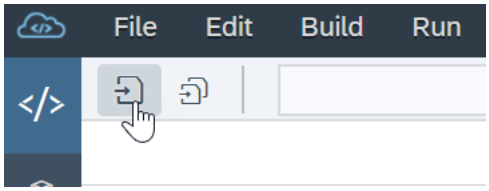
```
{
  "ServiceName_1": {
    "object_owner": {
      "roles": [
        "SFLIGHT_CONTAINER_ACCESS"
      ]
    },
    "application_user": {
      "roles": [
        "SFLIGHT_CONTAINER_ACCESS"
      ]
    }
  }
}
```

6. Click **Save**

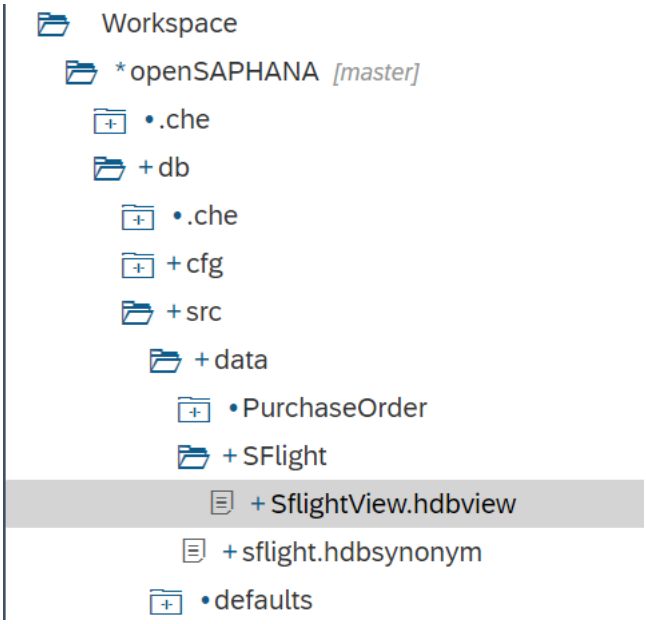


7. Go to the **data** folder within the **src** folder and create a new file called **sflight.hdbsynonym** by right-clicking on the data folder and choosing **New -> Database Artifact**.



8. Add Entries for **SFLIGHT**, **SBOOK** and **SPFLI**.	
9. To map each Synonym Name to the corresponding Object Name, click the **…** button and select the approrpiate object name after selecting the correct schema. Then press **Finish**.	
10. Click **Save**	
11. Select the **db** module and choose **Build->Build**. Check to make sure the build is sucessful.	

12. Create a folder named **SFlight** within **src/data**. Now create a file named **SflightView.hdbview**.



13. Insert the following code as shown

    If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_13d

```
VIEW "SflightView" (
     "Client",
         "CarrierId",
         "ConnectionId",
         "FlightDate",
         "Price",
         "Currency",
         "PlaneType",
         "BookingId",
         "CustomerId",
         "CustomerType",
         "PassengerName" ) AS SELECT
         "FLIGHTTBL_$0"."MANDT" AS "Client" ,
         "FLIGHTTBL_$0"."CARRID" AS "CarrierId" ,
         "FLIGHTTBL_$0"."CONNID" AS "ConnectionId" ,
         "FLIGHTTBL_$0"."FLDATE" AS "FlightDate" ,
         "FLIGHTTBL_$0"."PRICE" AS "Price" ,
         "FLIGHTTBL_$0"."CURRENCY" AS "Currency" ,
         "FLIGHTTBL_$0"."PLANETYPE" AS "PlaneType" ,
         "SBOOK_$1"."BOOKID" AS "BookingId" ,
         "SBOOK_$1"."CUSTOMID" AS "CustomerId" ,
         "SBOOK_$1"."CUSTTYPE" AS "CustomerType" ,
         "SBOOK_$1"."PASSNAME" AS "PassengerName"
FROM ( "SFLIGHT" AS "FLIGHTTBL_$0"
       LEFT OUTER JOIN "SBOOK" AS "SBOOK_$1" ON ( (
"SBOOK_$1"."MANDT" = "FLIGHTTBL_$0"."MANDT" )
               AND ( "SBOOK_$1"."CARRID" =
"FLIGHTTBL_$0"."CARRID" )
               AND ( "SBOOK_$1"."CONNID" =
"FLIGHTTBL_$0"."CONNID" )
               AND ( "SBOOK_$1"."FLDATE" =
"FLIGHTTBL_$0"."FLDATE" ) ) ) WITH READ ONLY
```

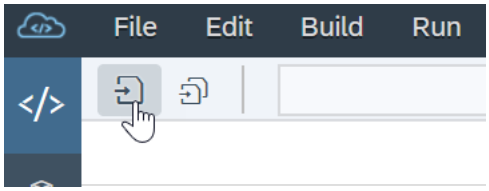
14. Create a second file named **SflightExt.hdbview** in the same folder and insert the following code.

    If you do not wish to type this code, you can reference the solution web page at

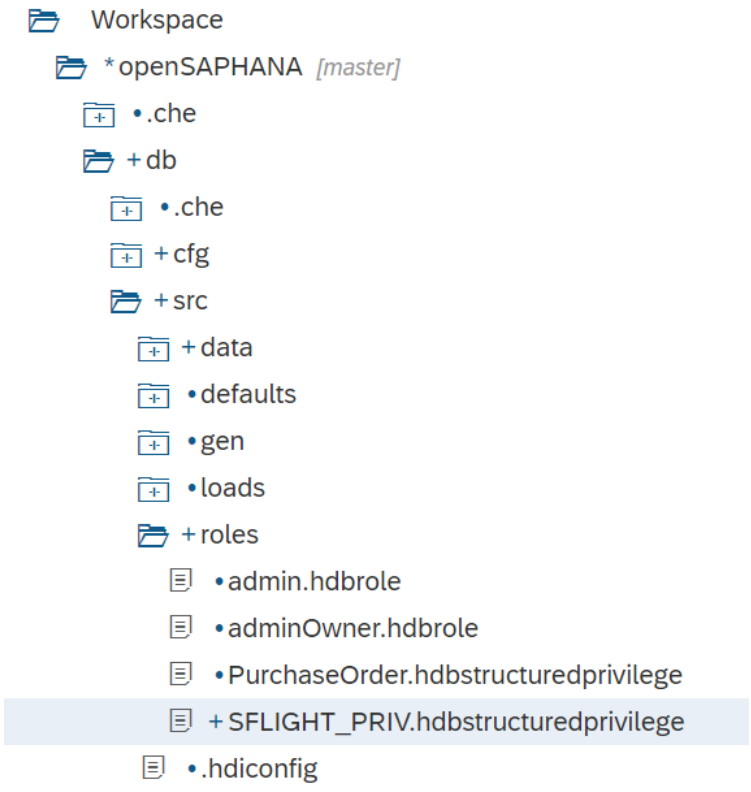
    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_13e

```
VIEW "SflightExt" (
     "Client",
         "CarrierId",
         "ConnectionId",
         "FlightDate",
         "BookingId",
         "Price",
         "Currency",
         "PlaneType",
         "CustomerId",
         "CustomerType",
         "PassengerName" ) AS SELECT
         "SflightView_$0"."Client" ,
         "SflightView_$0"."CarrierId" ,
         "SflightView_$0"."ConnectionId" ,
         "SflightView_$0"."FlightDate" ,
         "SflightView_$0"."BookingId" ,
         "SflightView_$0"."Price" ,
         "SflightView_$0"."Currency" ,
         "SflightView_$0"."PlaneType" ,
         "SflightView_$0"."CustomerId" ,
         "SflightView_$0"."CustomerType" ,
         "SflightView_$0"."PassengerName"
FROM "SflightView" AS "SflightView_$0" WITH READ ONLY
STRUCTURED PRIVILEGE CHECK
```

15. Click **Save**.



16. Under the **roles** folder, create a new structured privilege called **SFLIGHT_PRIV.hdbstructuredprivilege** by right-clicking on the roles folder and choosing **New->Database Artifact**.



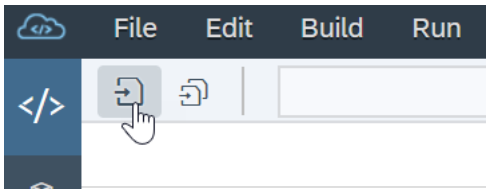
17. Enter the following code as shown.

    If you do not wish to type this code, you can reference the solution web page at

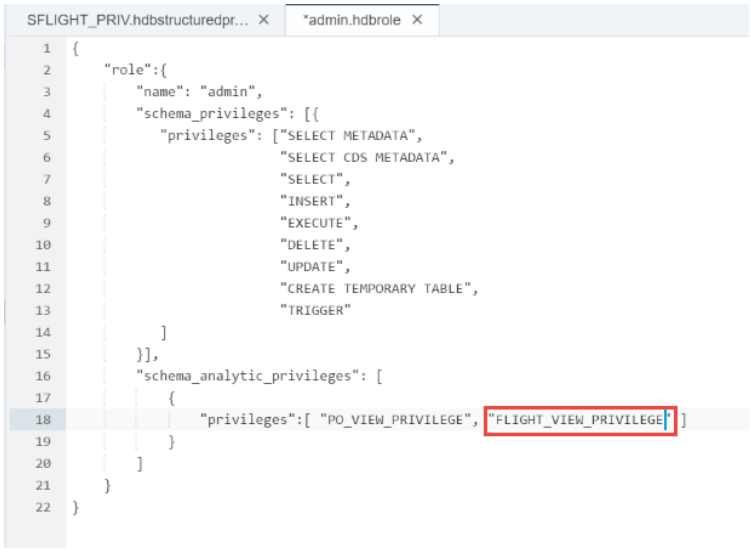
    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_14

```
STRUCTURED PRIVILEGE
    "FLIGHT_VIEW_PRIVILEGE"
    FOR SELECT ON
    "SflightExt"
    WHERE "Client" = '001'
```

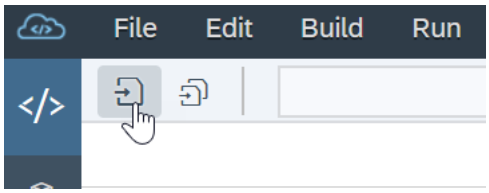
18. Click **Save**.



19. Open the admin.hdbrole file in the roles folder and add the new structured privilege as shown.

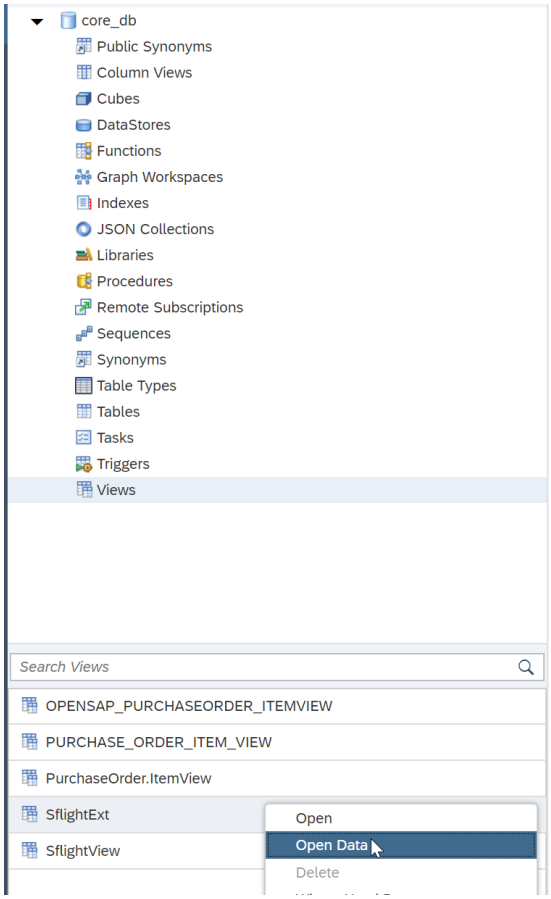


20. Click **Save**.

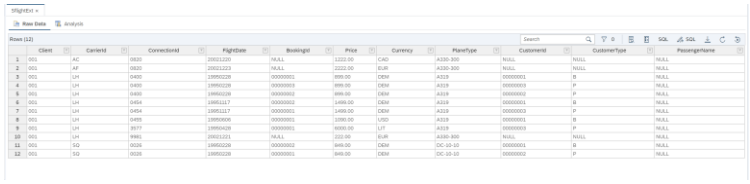


21. Select the **db** module and choose **Build->Build**.

22. Switch over to the **Database Explorer** perspective. Right-click on the **SflightExt** view in the **Views** folder and choose **Open Data**.



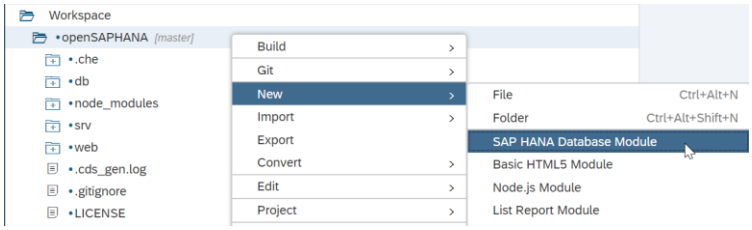
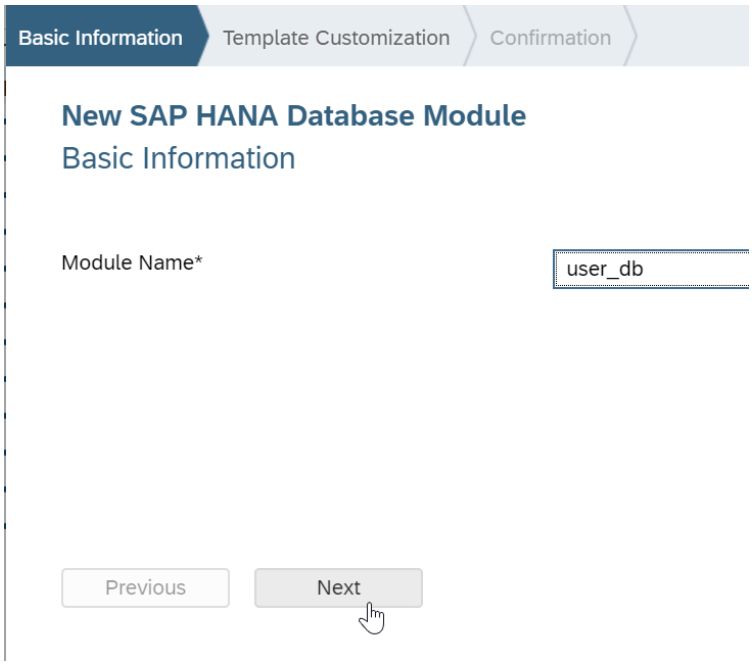
23. The results are shown but only the records from client 001 becuase of the filter in the Structured Privilege.

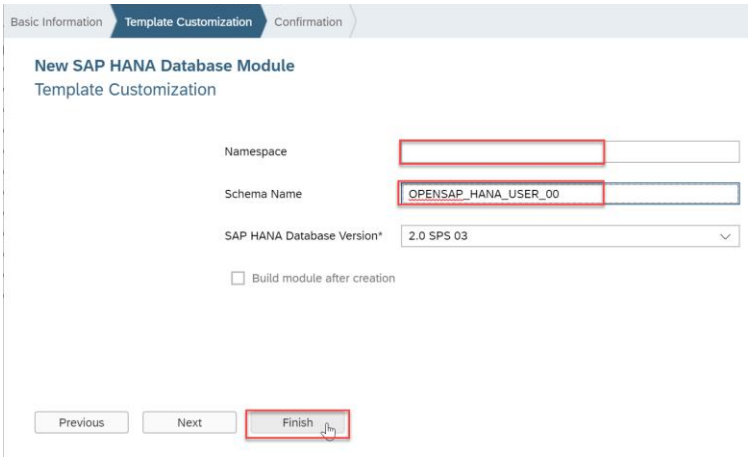
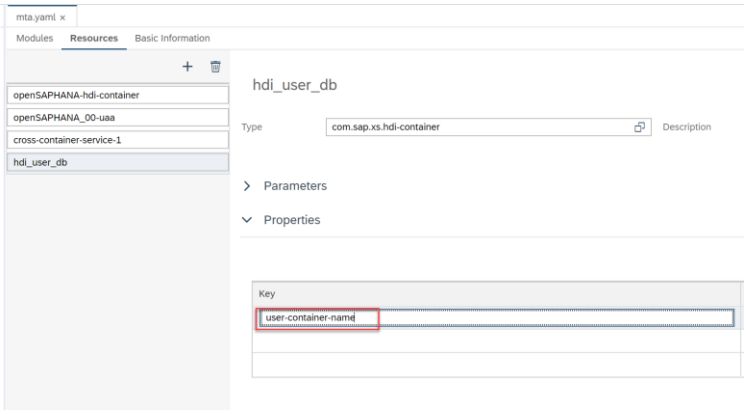
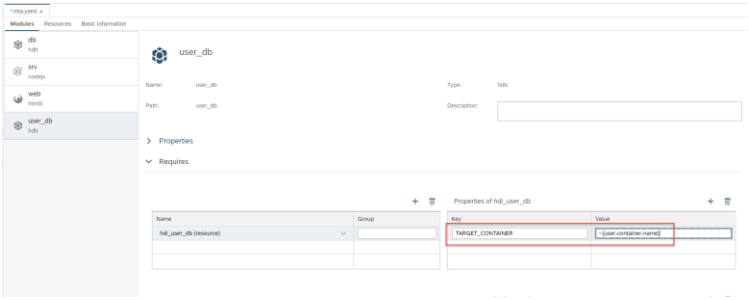


Exercise 2.7: Cross Container Services & Synonyms

XSA embraces the concepts of a micro-service architecture. This means that different parts of the same overall application might be separated into separate services. This is true also of the database services. So far we have purchase order data in our db module. However we also want user data but that should be a completely separate HDI container.

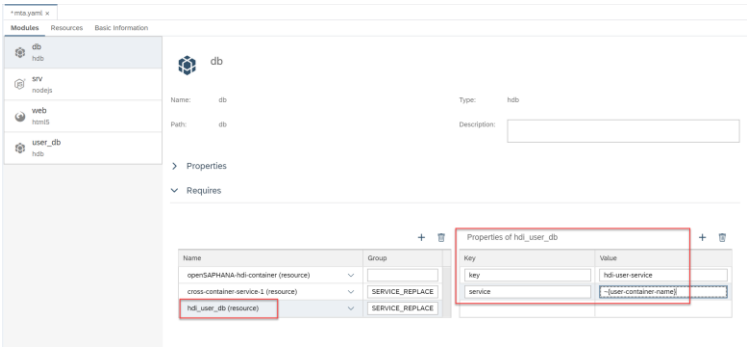
In this exercise we will create a second HDB module and HDI container service and configure access and synonyms between the two.

Explanation	Screenshot
1. We will begin by creating a second HDB module for our User table. Right mouse click on the project and choose **New->SAP HANA Database Module.**	
2. Name this new module **user_db**. Then press **Next**.	

3. Clear the Namespace field. Input a Schema Name of **OPENSAP_HANA_USER_< group number>.** The latest HANA version should default in. Unselect the **Build module after creation** option. Then press **Finish**.	
4. The wizard has created the user_db folder as well as the hdi_user_db resource and the db module in the **mta.yaml** file for you. Let's change the name of the container property name to **user-container-name** under the **Resources** tab. Save the file. Switch to the Code Editor and then back to the MTA Editor to refresh the value helps.	
5. Switch to the Modules section and choose the **user_db** module. An HDB module can also have more than one database resource associated with it. However it can only have one primary resouce which is the target for all database development objects it will create. Add the **TARGET_CONTAINER** property with the value **~{user-container-name}** Save the mta.yaml file.	

6. Change the selection from the user_db module to the **db** module. You can now add a **requires** dependency in the db module to the **hdi_user_db** resource as well.

   Similiar to the SFLIGHT service from the last exericse we will also configure synonym access to the user container from our core container.



7. The compelte mta.yaml should now look something like this (adjusting for your group number).

```
ID: openSAPHANA
_schema-version: '2.1'
version: 0.0.1
modules:
 - name: db
   type: hdb
   path: db
   parameters:
     memory: 256M
     disk-quota: 256M
   requires:
     - name: openSAPHANA-hdi-container
       properties:
         TARGET_CONTAINER: '~{hdi-container-name}'
     - name: cross-container-service-1
       group: SERVICE_REPLACEMENTS
       properties:
         key: ServiceName_1
         service: '~{the-service-name}'
     - name: hdi_user_db
       group: SERVICE_REPLACEMENTS
       properties:
         key: hdi-user-service
         service: '~{user-container-name}'
 - name: srv
   type: nodejs
   path: srv
   parameters:
     memory: 512M
     disk-quota: 256M
   requires:
     - name: openSAPHANA-hdi-container
   provides:
     - name: srv_api
       properties:
         url: '${default-url}'
 - name: web
   type: html5
   path: web
```

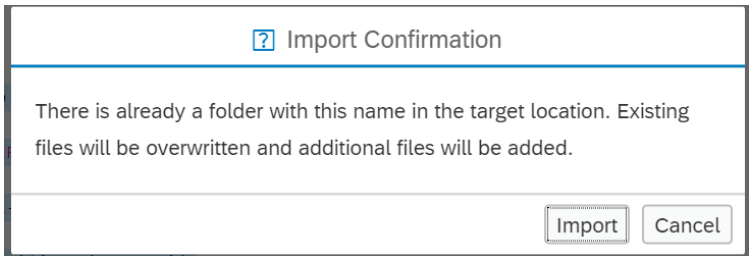
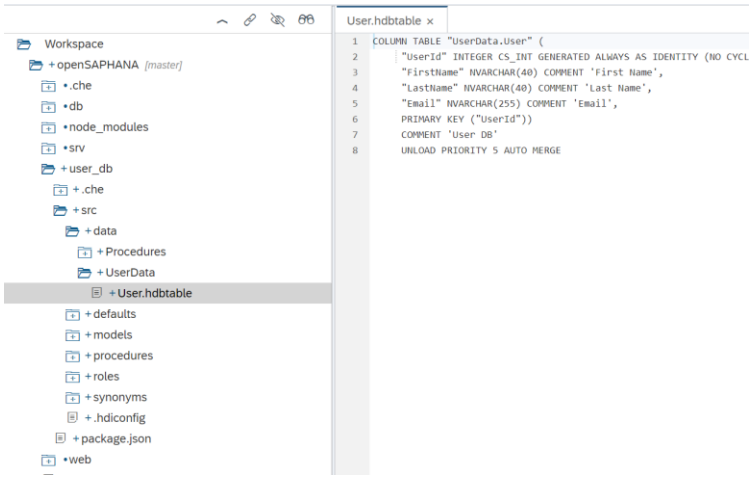
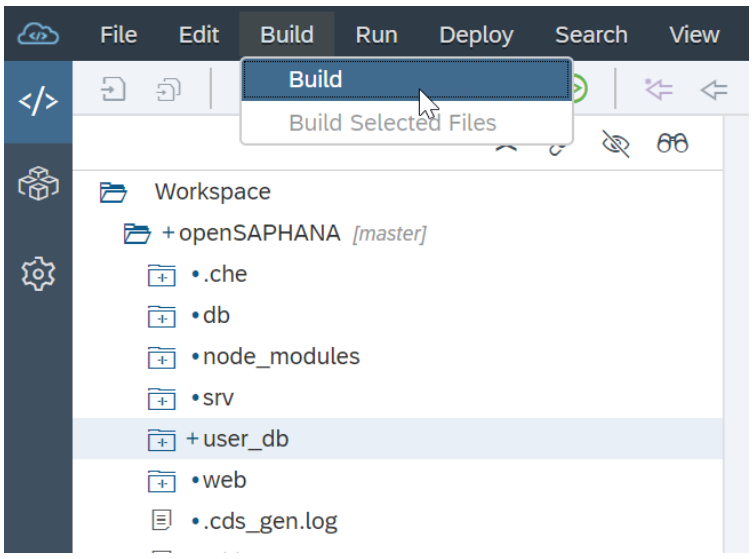
```
    requires:
      - name: openSAPHANA_00-uaa
  - name: user_db
    type: hdb
    path: user_db
    requires:
      - name: hdi_user_db
        properties:
          TARGET_CONTAINER: '~{user-container-name}'
resources:
  - name: openSAPHANA-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: openSAPHANA_00-uaa
    type: com.sap.xs.uaa-space
    parameters:
      path: ./xs-security.json
  - name: cross-container-service-1
    parameters:
      service-name: CROSS_SCHEMA_SFLIGHT_00
    properties:
      the-service-name: '${service-name}'
    type: org.cloudfoundry.existing-service
  - name: hdi_user_db
    parameters:
      config:
        schema: OPENSAP_HANA_USER_00
    properties:
      user-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
```

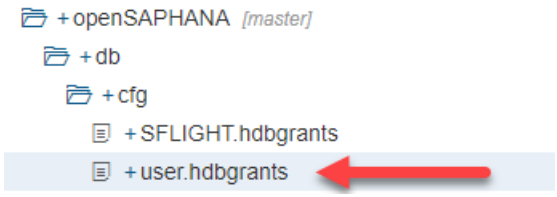
8. We could follow similiar steps to create hdbcds, roles, sequences, etc in the user_db module. However to save time we will upload a template implementation of these objects.

   Download src.zip from: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex2/user-db/src.zip

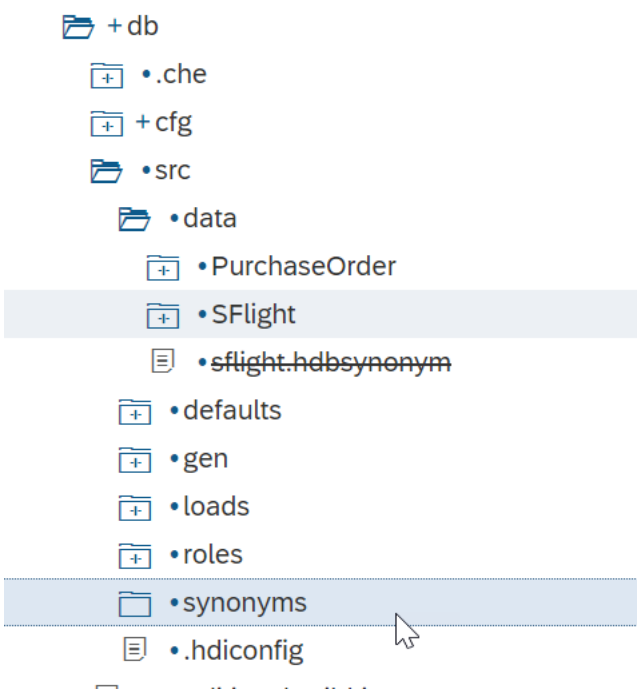
   Right mouse click on **user_db/src** and choose **Import -> File or Project**. Click Browse and chose **src.zip** from the location where you just saved it on your local machine. Ensure that Extract Archive is



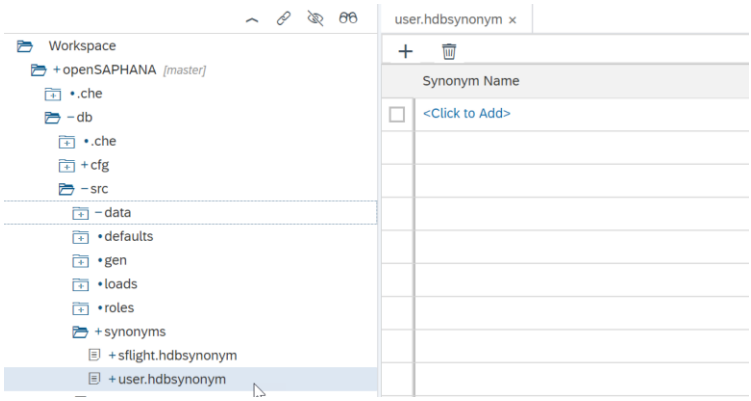
checked. Keep other values at their default and press **OK**.	
9. You will see this message – which is fine. Just press **Import**.	
10. The import loaded many files into user_db/src; but we will be most interested in the User.hdbtable (with the UserData.User entity) and the user.hdbrole.	
11. Select the **user_db** module and choose **Build->Build**.	

12. You should see a message that you build was sucessful.	Unlocking the container "OPENSAP_HANA_USER_00_1"... Unlocking the container "OPENSAP_HANA_USER_00_1"... ok (0s 0ms) Deployment to container OPENSAP_HANA_USER_00_1 done [Deployment ID: none]. (3s 478ms) 3:24:23 PM (DIBuild) ********** End of /openSAPHANA/user_db Build Log ********** 3:24:23 PM (DIBuild) Build results link: [object Promise] 3:24:24 PM (Builder) Build of /openSAPHANA/user_db completed successfully.
14. Now we want to extend access to the user_db container to our original db container. Create a new file in the **cfg** folder of **db** called **user.hdbgrants** by right-clicking on the cfg folder and choosing **New->File**	
14. Enter the following code. This will use the user.hdbrole from the user_db module to grant access to user database content to the technical users of the db module. If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_15	{ "hdi-user-service": { "object_owner": { "container_roles":["user#"] }, "application_user": { "container_roles":["user"] } } }

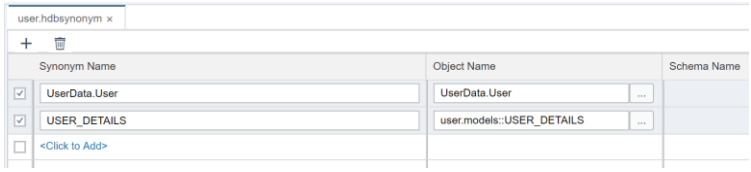
15. We are starting to get several synonym files, so lets create a folder in **db/src** called **synonyms**. Cut and paste **sflight.hdbsynonym** from the data folder to the new synonyms folder for better organization.



16. Go to the new **synonyms** folder within the src folder and create a new file called **user.hdbsynonym** by right-clicking on the synonyms folder and choosing **New->Database Artifact**.



17. Click the **Click to Add** link to add new entries.
Add the following entries as shown.



Create a file in the **/cfg** folder called **.hdiconfig** and place following code in it to allow the deployment of hdbysnonymconfig files.

If you do not wish to type this

```
{
   "file_suffixes":{
      "hdbsynonymconfig":{
         "plugin_name":"com.sap.hana.di.synonym.config",
         "plugin_version":"2.0.30.0"
      }
```

code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_15a

```
    }
}
```

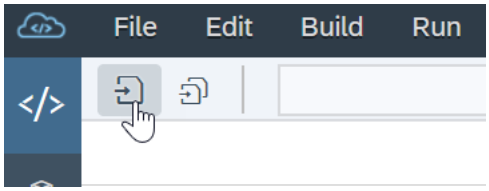
18. Notice we didn't enter a schema name in the hdbsyonym file. That's because the schema is generated for the foregin container. Therefore we also need to create a file named **user.hdbsynonymconfig in** the **/cfg folder**.

    Here we use the service replacement name configured in the mta.yaml which will be replaced at build with the actual container name.

user.hdbsynonymconfig ×

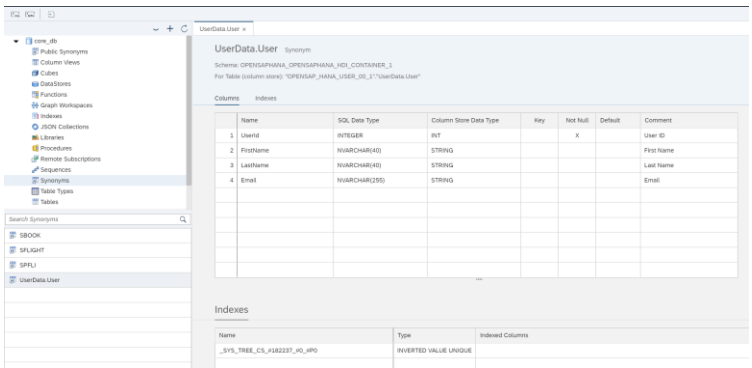
```
{
  "UserData.User": {
    "target": {
      "object": "UserData.User",
      "grantor": "hdi-user-service"
    }
  },
  "USER_DETAILS": {
    "target": {
      "object": "user.models::USER_DETAILS",
      "grantor": "hdi-user-service"
    }
  }
}
```

19. Save all files.



20. Select the **db** module and choose **Build->Build**. Check to make sure the build is successful.

21. You can then also return to the Database Explorer and choose the synonyms folder and you should be able to see the UserData.User synonym in the db container.



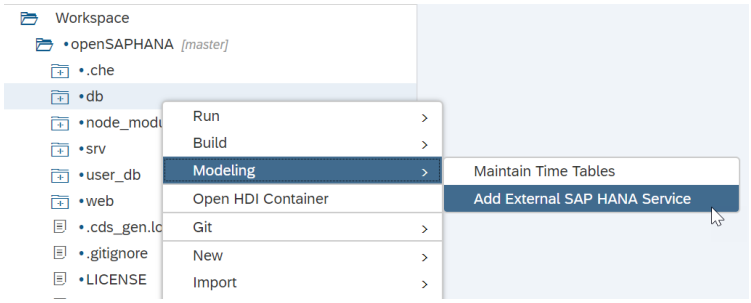
Exercise 2.8: Cross Container From Outside Our Project

XSA embraces the concepts of a micro-service architecture. This means that different parts of the same overall application might be separated into separate services. This is true also of the database services. So far we have purchase order data in our **db** module and access to the non-container schema, SFLIGHT and user data from the user_db HDI container within our project. However, we also want master data that already exists in a completly separate HDI container that has been installed from another MTA (Multi-Target Application).

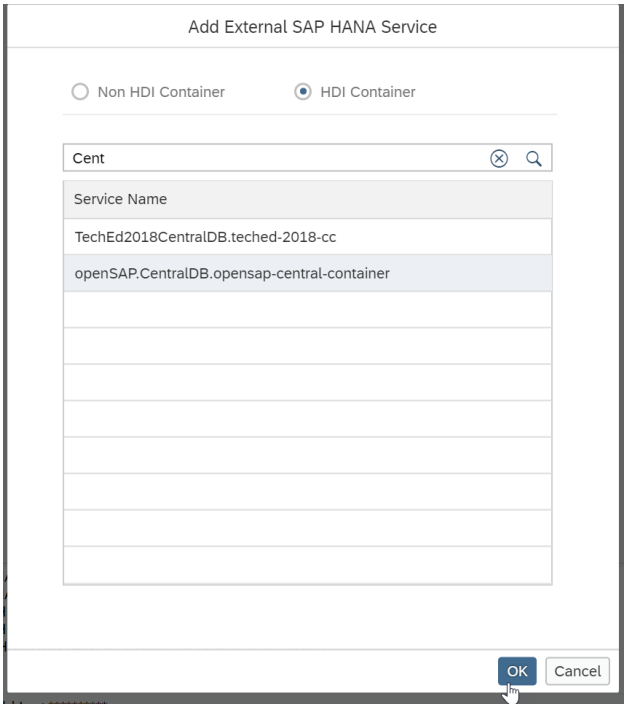
In this exercise, we will configure access and synonyms between our local project container and an already existing container deployed on this system. In order to perform this exercise in your own system, you will have to first download and deploy (with the –use-namespaces option) this MTA into your development space from:

https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex2/openSAP.CentralDB_1.1.2019.mtar

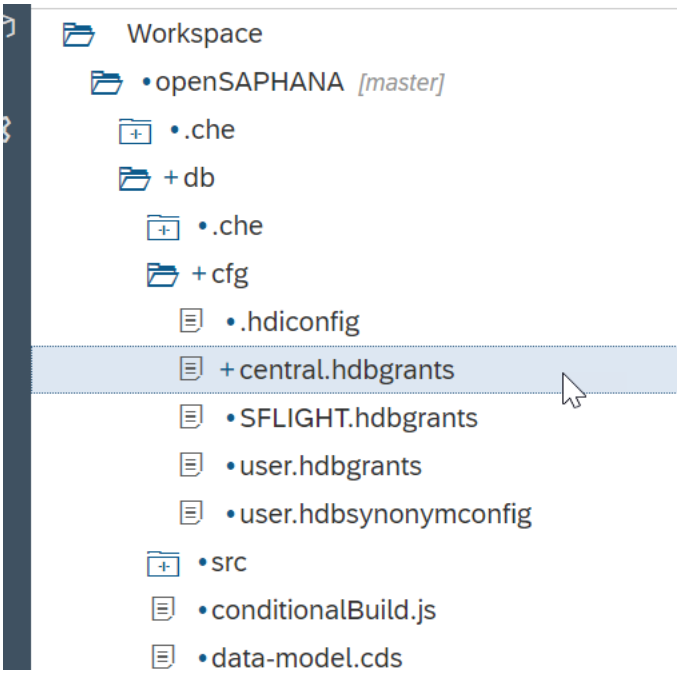
xs deploy help: https://help.sap.com/viewer/4505d0bdaf4948449b7f7379d24d0f0d/2.0.03/en-US/308f2041995c41a88bbb197cf2a984f7.html#loio308f2041995c41a88bbb197cf2a984f7__section_irt_3dc_zs

Explanation	Screenshot
1. We want to add an existing HDI service to our container as well. From the db module, choose **Modeling Actions -> Add External SAP HANA Service**.	

2. Choose **HDI Container** radio button at the top of the dialog and then select the service named **openSAP.CentralDB.opensap-central-container**.

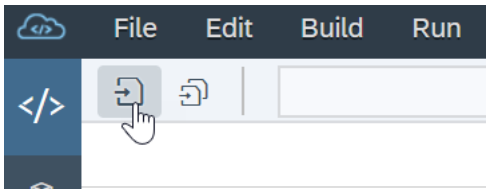
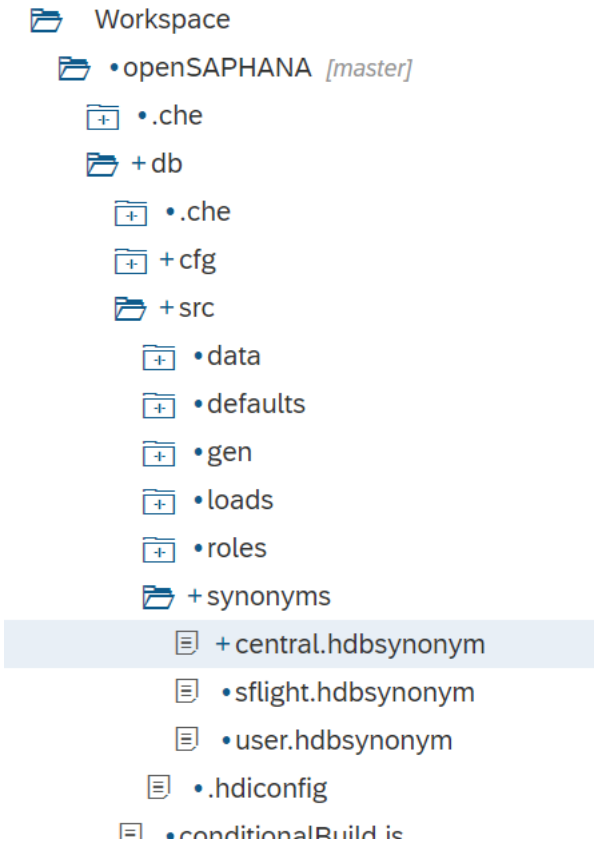


4. Create a new file in the **cfg** folder called **central.hdbgrants** by right-clicking on the cfg folder and choosing **New->File**



5. Enter the following code. It is this object which will grant the specified access from the central master data container to the local container technical users.

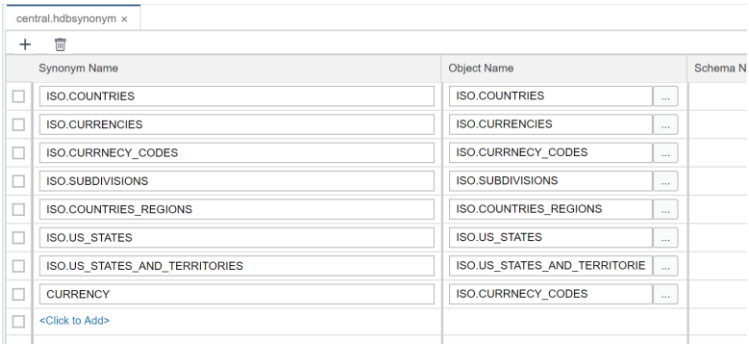
```
{
 "ServiceName_2": {
  "object_owner" : {
   "container_roles":["consumption", "admin#"]
  },
  "application_user" : {
```

If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_13a	<pre>"container_roles":["consumption"] } } }</pre>
6. Click **Save**	
7. Go to the **synonyms** folder within the src folder and create a new file called **central.hdbsynonym** by right-clicking on the data folder and choosing **New -> Database Artifact**.	

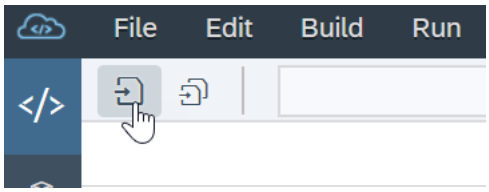
8. Add the following entries as shown.
ISO.COUNTRIES
ISO.CURRENCIES
ISO.CURRENCY_CODES
ISO.SUBDIVISIONS
ISO.COUNTRIES_REGIONS
ISO.US_STATES
ISO.US_STATES_AND_TERRITORIES

   If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_13b



9. Click **Save**



10. Create another file in the **cfg** folder named **central.hdbsynonymconfig**. This is where we will specify the target container service to use for each synonym in the previous file.

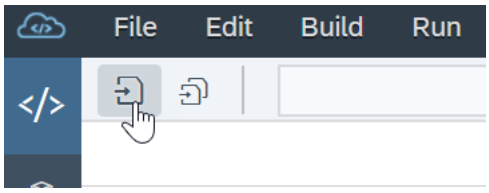
    If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_13c

```
{
  "ISO.COUNTRIES": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.COUNTRIES"
    }
  },
  "ISO.CURRENCIES": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.CURRENCIES"
    }
  },
  "ISO.CURRNECY_CODES": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.CURRNECY_CODES"
    }
  },
  "ISO.SUBDIVISIONS": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.SUBDIVISIONS"
    }
  },
  "ISO.COUNTRIES_REGIONS": {
```

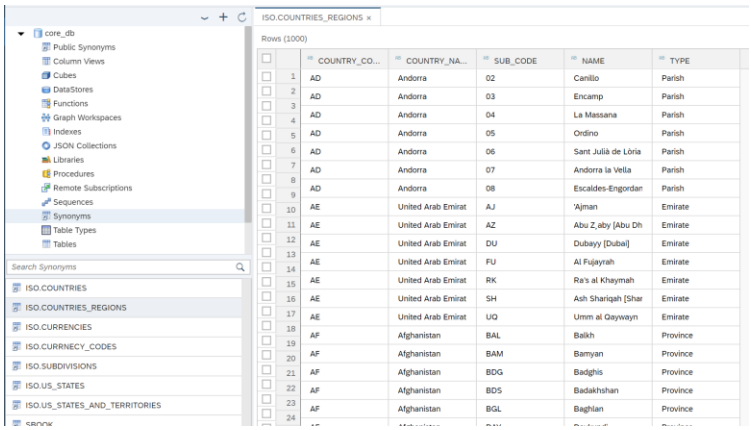
```
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.COUNTRIES_REGIONS"
    }
  },
  "ISO.US_STATES": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.US_STATES"
    }
  },
  "ISO.US_STATES_AND_TERRITORIES": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.US_STATES_AND_TERRITORIES"
    }
  },
  "CURRENCY": {
    "target": {
      "grantor" : "ServiceName_2",
      "object": "ISO.CURRNECY_CODES"
    }
  }
}
```

11. Click **Save**

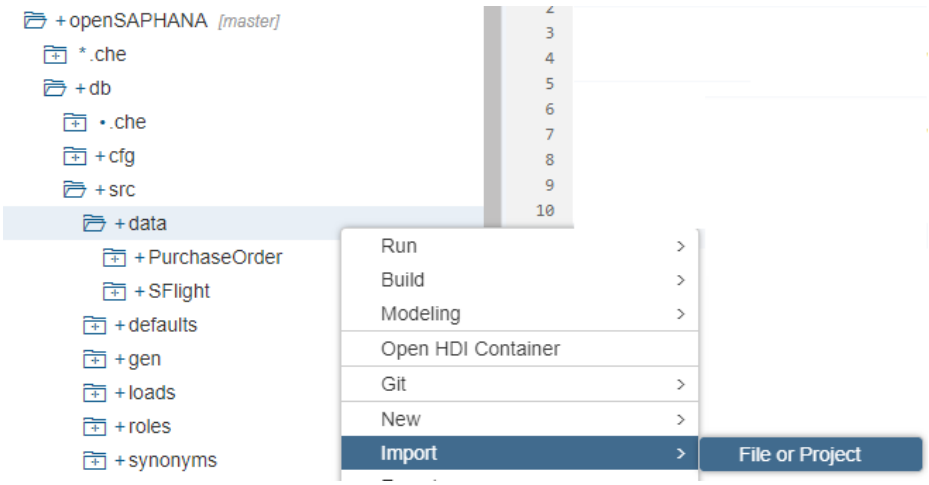
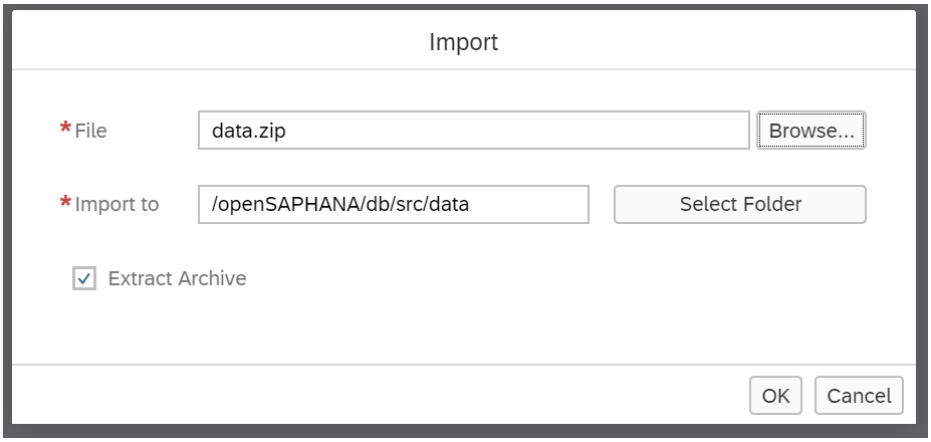
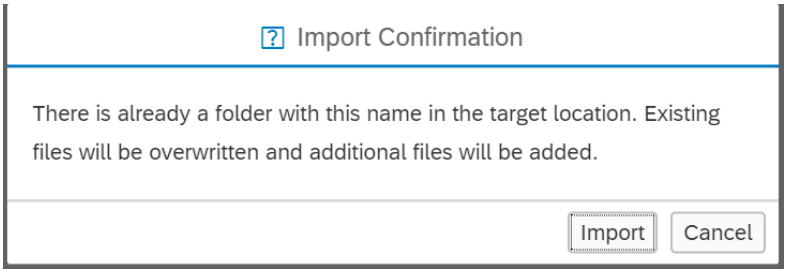


12. Select the **db** module and choose **Build->Build**. Check to make sure the build is successful.

13. You can then also return to the **Database Explorer** and choose the synonyms folder and you should be able to see the various country, region and currency synonyms in the db container. You can use Open Data to view the data in any of these “foreign” tables.

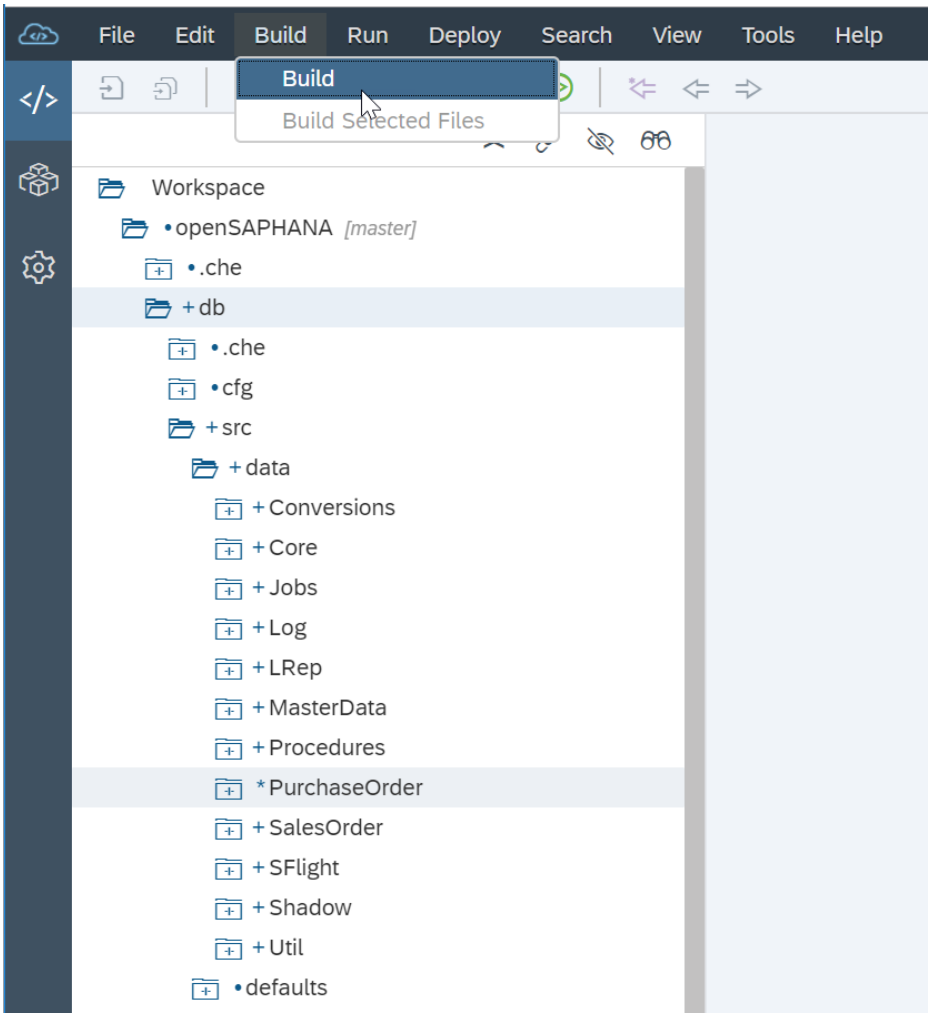


Exercise 2.9 Creating a Calculation View with a Dimension data type

Explanation	Screenshot
1. Before we start creating Calculation views, we want to import a larger database module than we care to create by hand. Download the data.zip from from: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex2/core-db/data.zip Right mouse click on the **db/src/data** folder and choose **Import -> From File System.**	
2. Choose the file data.zip from the Download folder of your local client machine. Select **Extract Archive**. Keep all other parameters the same. Press OK. Confirm that it is ok to overwrite the existing files by pressing **Import**.	 

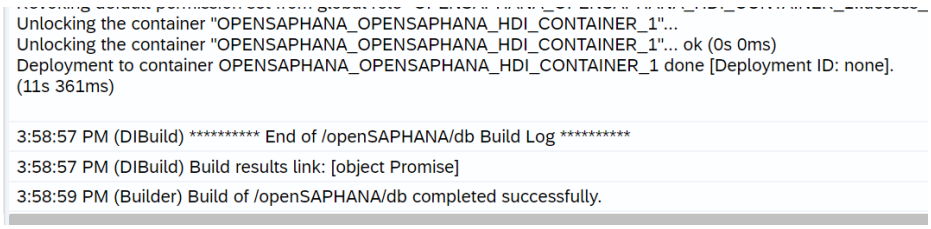
3. Now that we have our database development objects, we are ready to build the module which will create them in the HANA database.

   Select the **db** folder and choose **Build -> Build**.

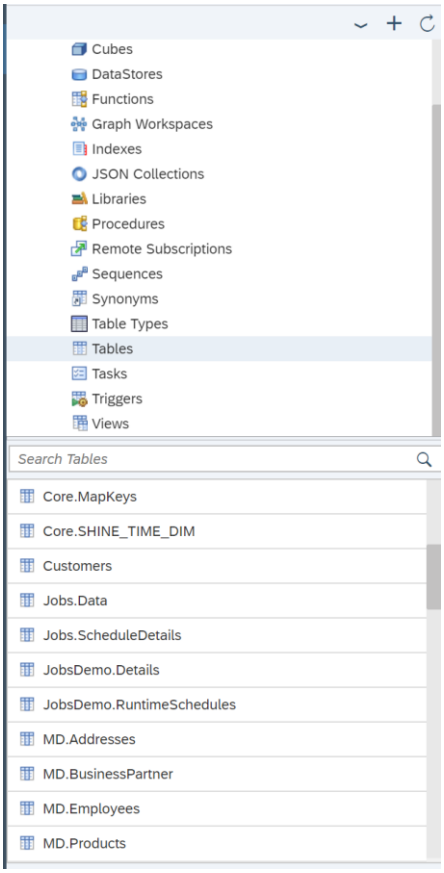


4. The status of the build will be displayed in a window in the lower right side of the IDE.

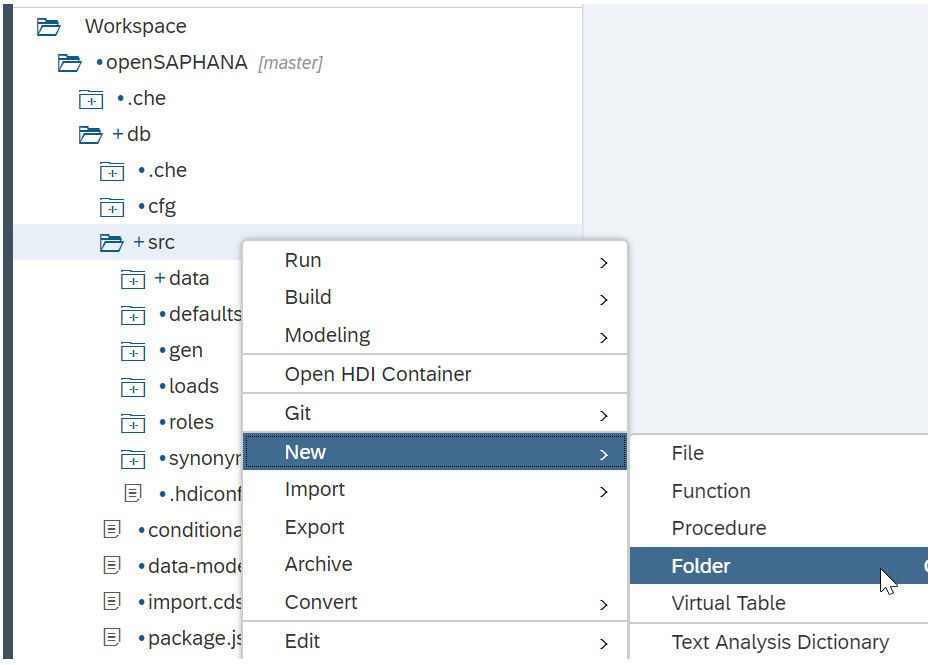
   If everything worked correctly, you should see that the build completed successfully.



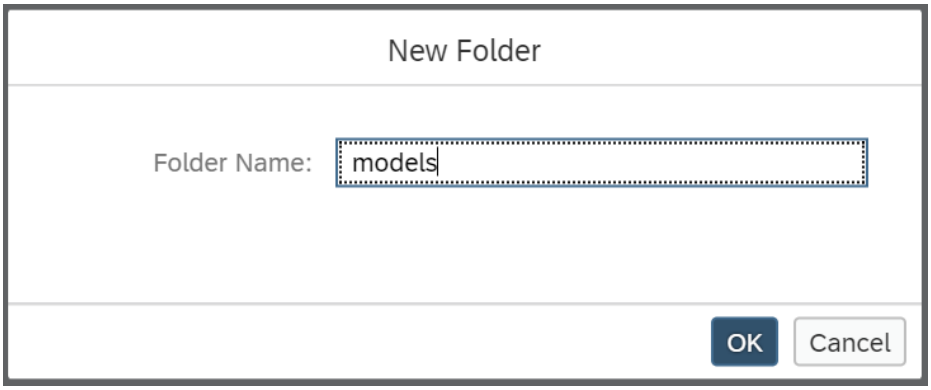
5. You can explore some of the new tables and their content that have been imported via the Database Explorer perspective. This is esentially the same data model as SHINE (SAP HANA Interactive Education).



6. Now we are ready to create our calculation views. From the **db/src** folder, right click and choose **New -> Folder**



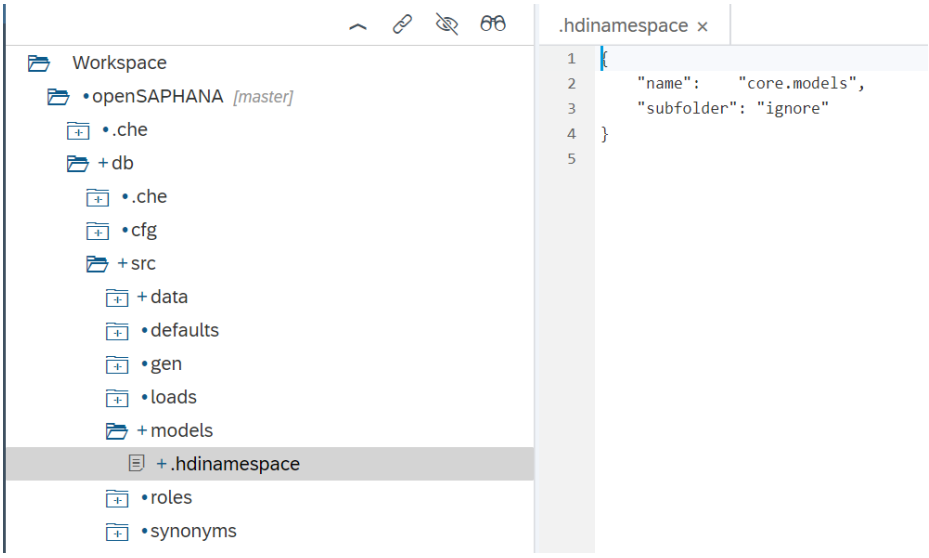
7. Enter the name of the folder as **models** and click **OK**.



8. For backwards compatibility, Calculation View building works best when a namespace is used. Our **db** module currently doesn't use namespaces. Therefore we will need to re-activate namespaces for just the models folder.

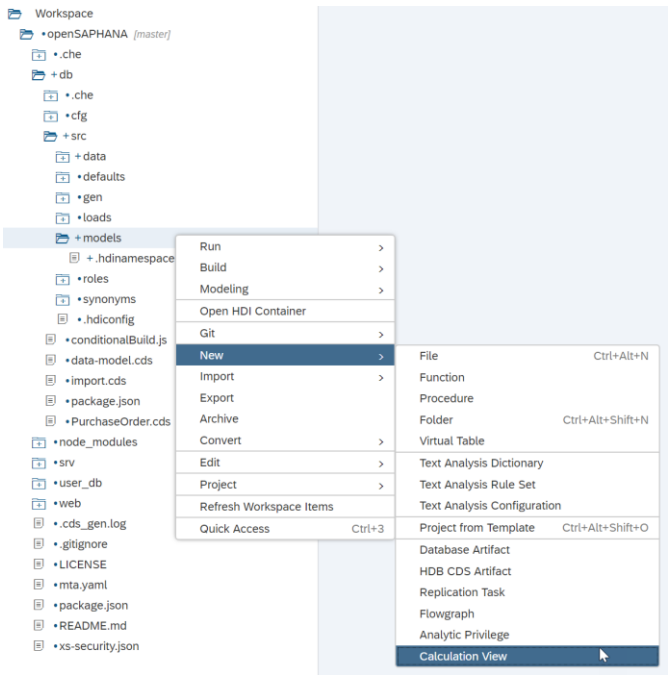
   Create a file named **.hdinamespace** in the **src/models folder**. Enter this text into the file.

```
{
        "name":    "core.models",
        "subfolder": "ignore"
}
```



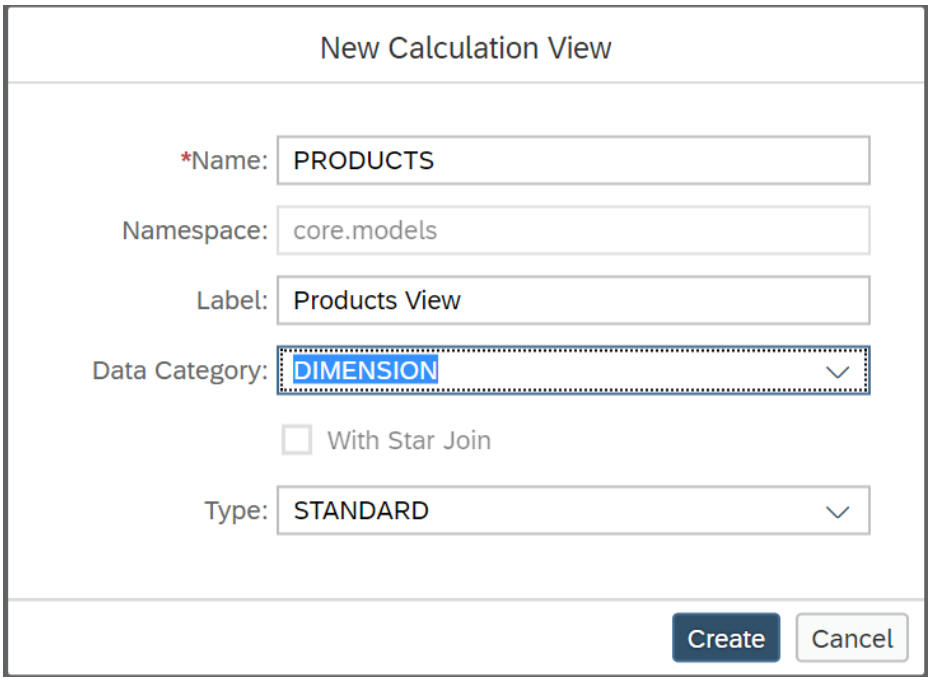
9. In the **models** sub-folder of your project, create a new calculation view based upon the expanded information for the products. This will require joining the **products**, **texts**, **businessPartner**, and **address** tables.

   Right mouse click on the models package, choose **New -> Calculation View**



10. Enter the name as **PRODUCTS** and a Label of **Products View**.
    Choose **DIMENSION** as the data category. A DIMENSON Calculation View will be very similar to the older Attribute View – A basic join with no aggregation.

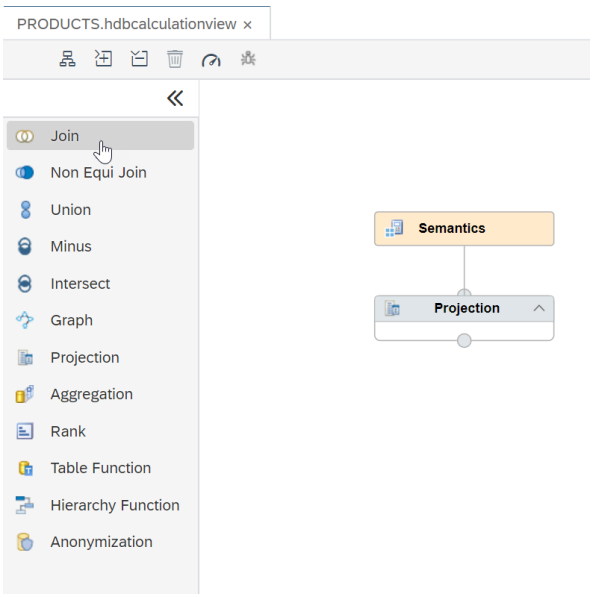
    Click **Create**.



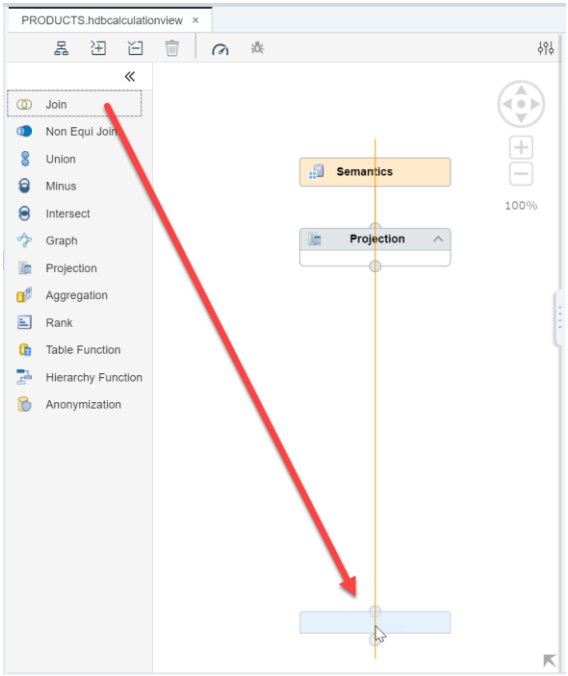
11. We want to join several tables from our project. Ultimately we will join **MD.Products, MD.BusinessPartner**, **MD.Addresses** and **Util.Texts** (twice).

    In order to do so we have to create a join node for each join criteria.

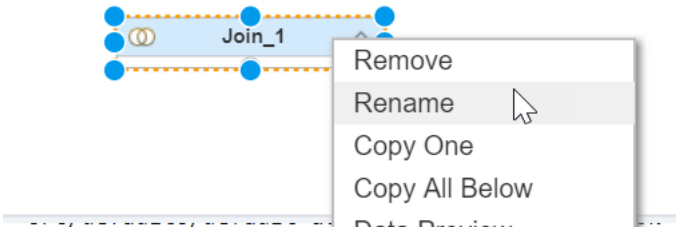
    Start by creating a **Join Node** pressing the Create Join button.



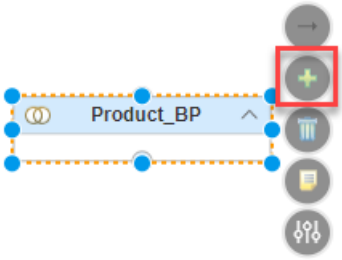
12. As we have several joins to add drag this new join node down near the bottom of the design window.



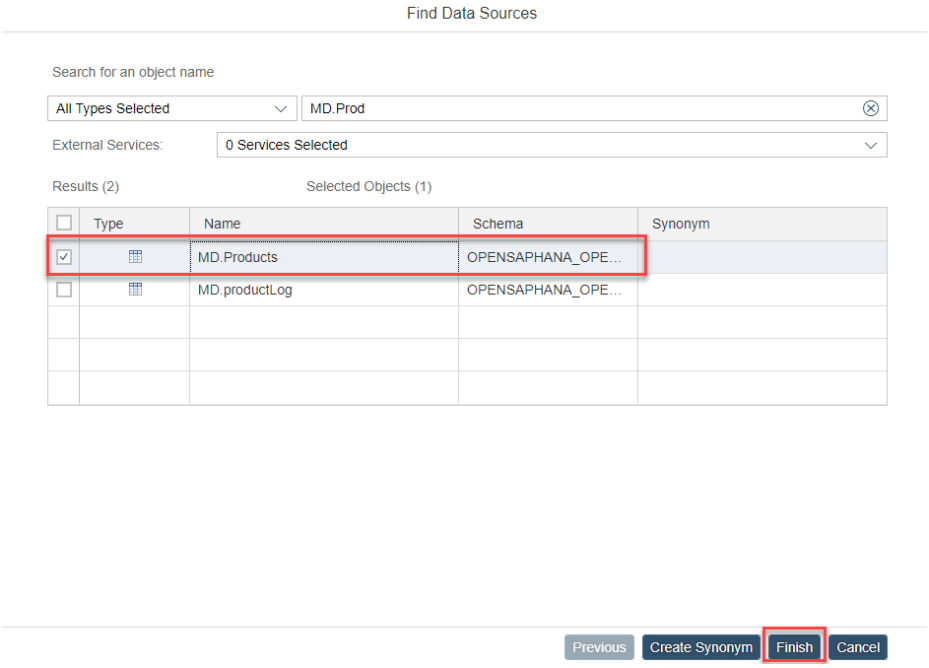
13. Right mouse click on the node, choose **Rename** and rename it to **Product_BP**.



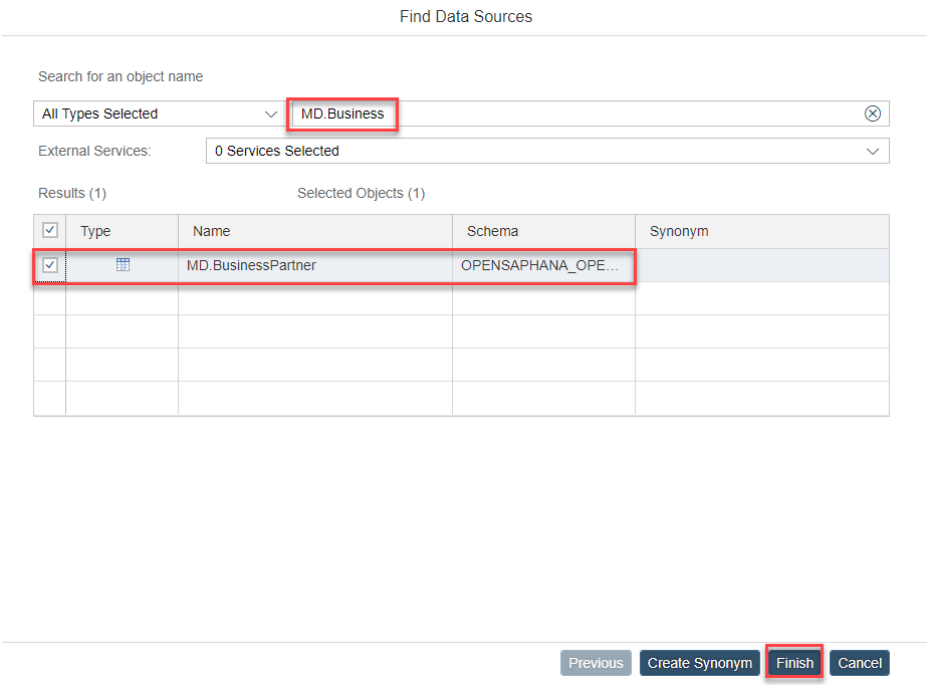
14. Press the **Plus symbol** button next to the node to add tables/views to the join node.



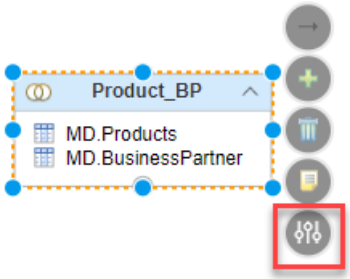
15. Add **MD.Products** to the node and press **Finish**.



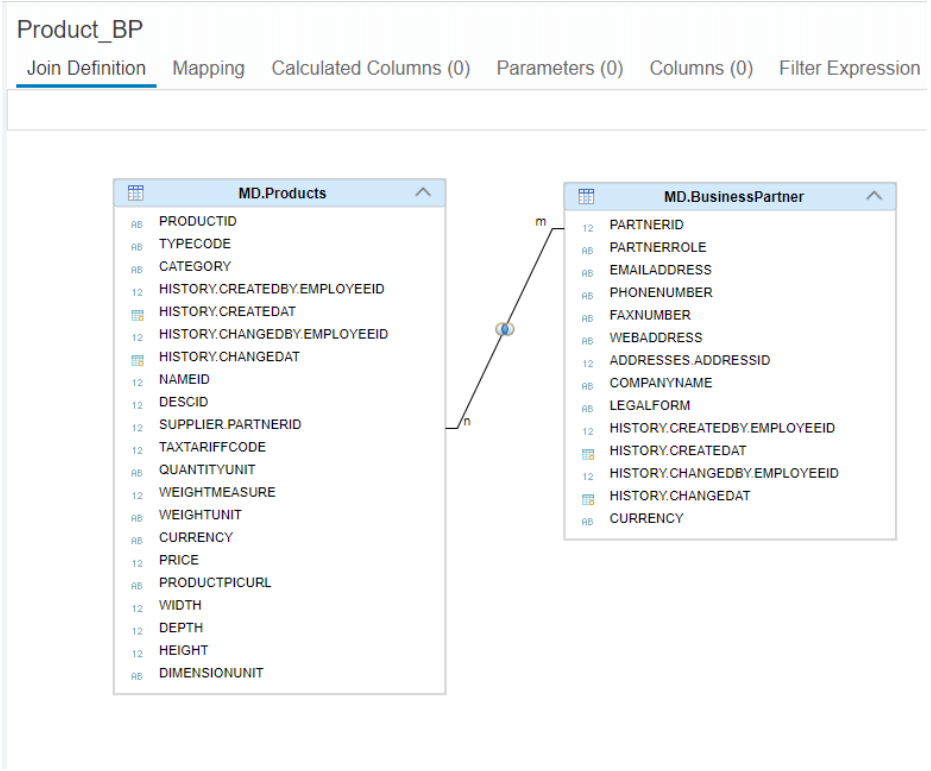
16. Repeat the process and add **MD.BusinessPartner** to the node.



17. Next, double-click on the **Expand Details Panel** button on the join to open the panel.

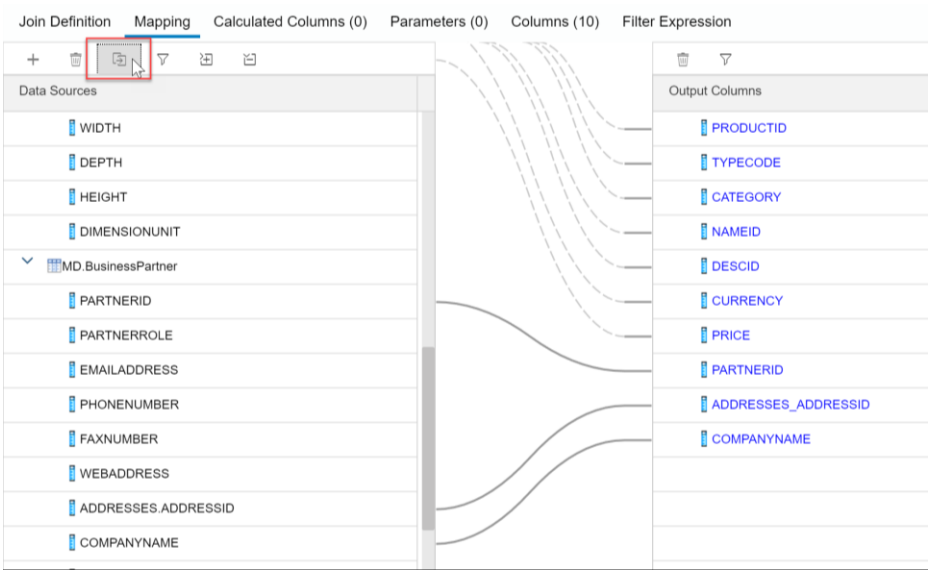


18. We now want to create a join between the two tables from the **SUPPLIER.PARTNERID** to the **PARTNERID** column. Drag and drop to connect the two columns in the Join Definition.

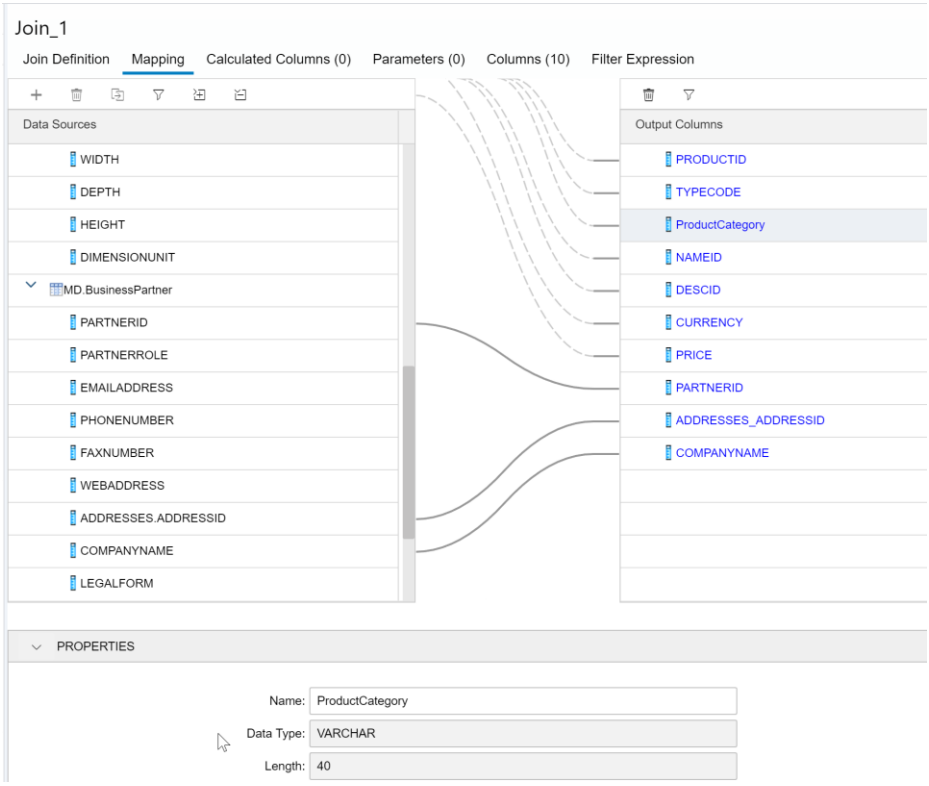


19. Switch to the **Mapping** tab. We can then select which columns we want from this part of the join.

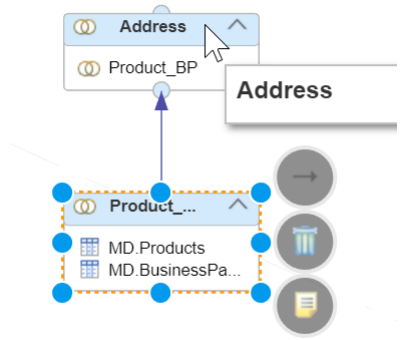
    Select **PRODUCTID**, **TYPECODE**, **CATEGORY**, **NAMEID**, **DESCID**, **CURRENCY**, **PRICE**, **PARTNERID**, **ADDRESES.ADDRESSID**, and **COMPANYNAME** then choose **Add To Output**.



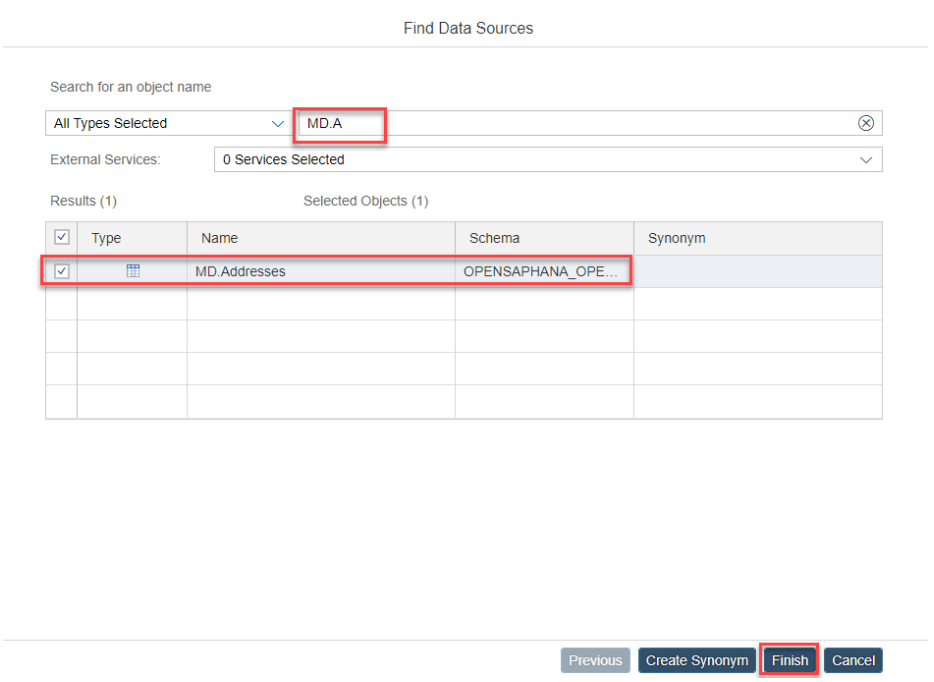
20. Optionally, you can change the name of a column as it becomes part of the view. For example you might change CATEGORY to ProductCategory.



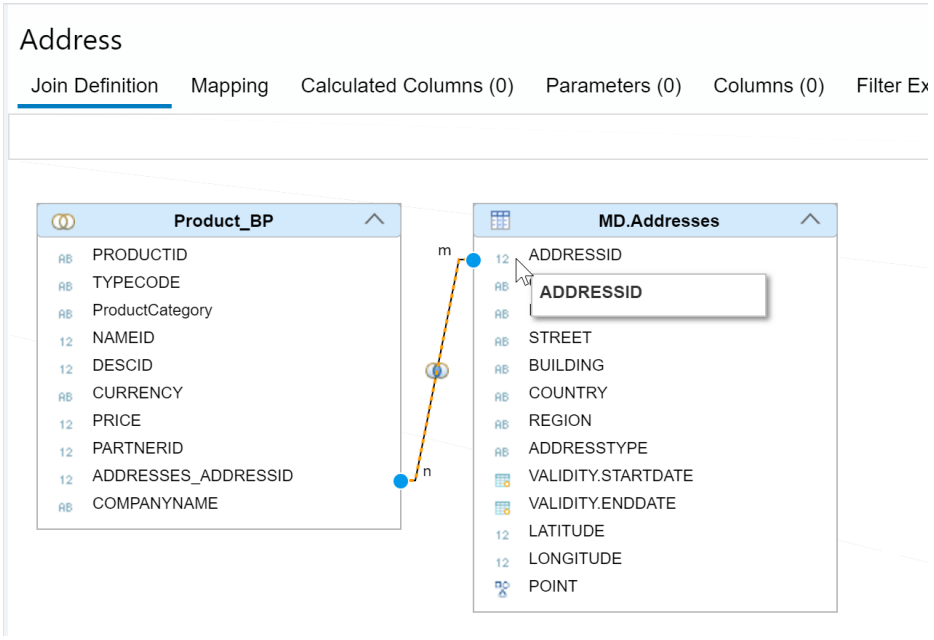
21. Repeat the process of adding a **Join Node**. Name this new Join Node **Address** and connect the output **Product_BP** to this new join node.



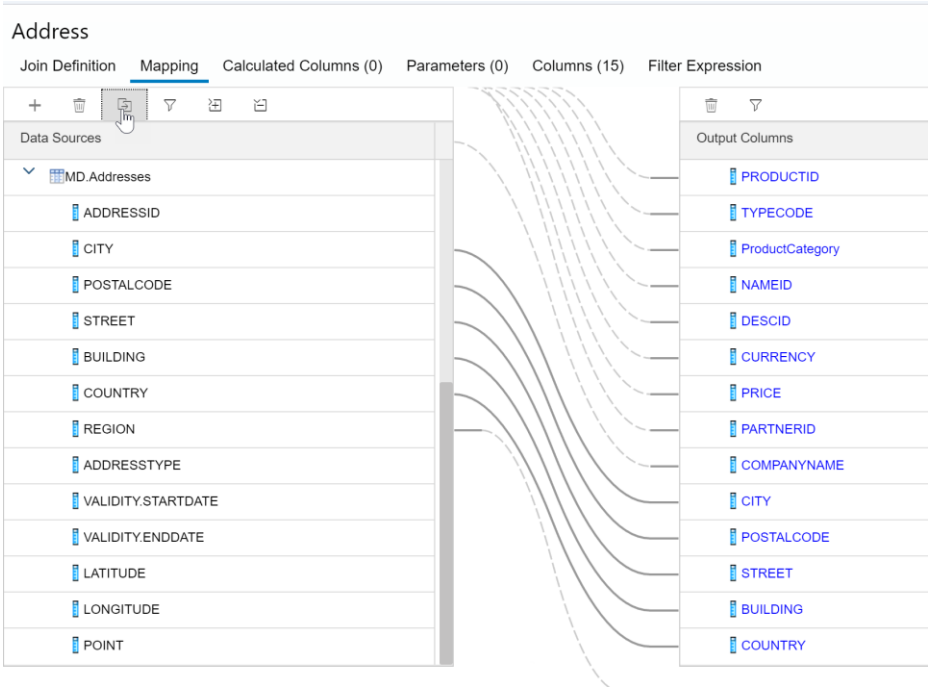
22. Add the **MD.Addresses** table to this join node.



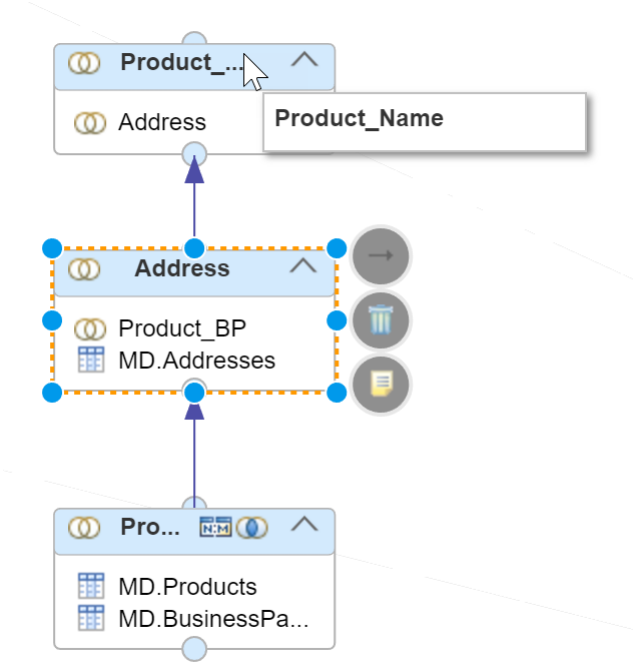
23. Expand the Details Panel like before to create a join between the **ADDRESSES_ADDRESSID** of the previous join node output and the **ADDRESSID** column of the MD.Addresses table.



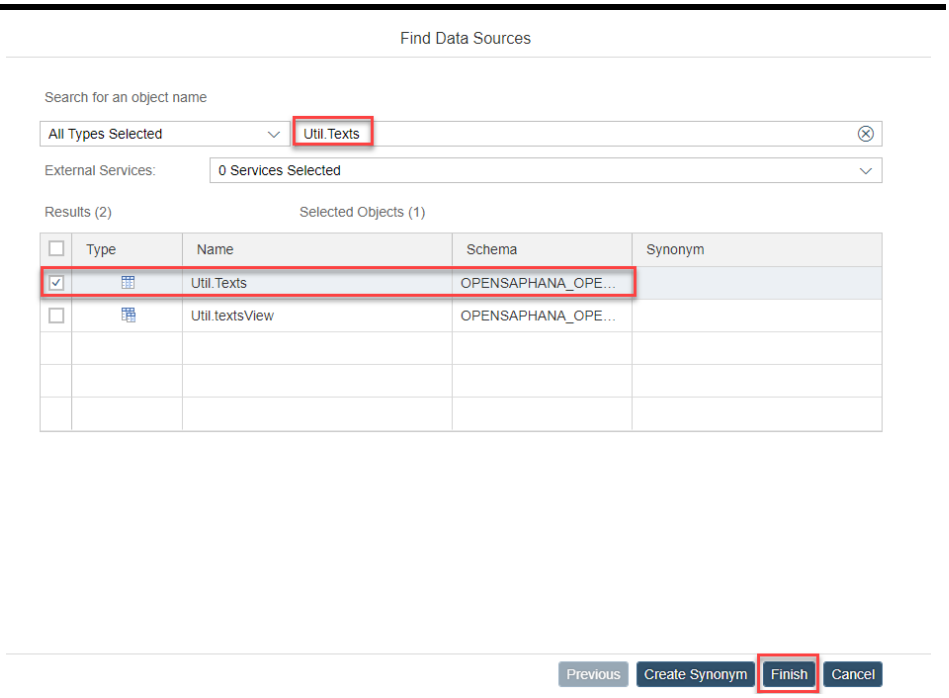
24. Repeat the process of adding columns to the output in **Mapping**. Select all columns from the Product_BP node **except ADDRESSES_ADDRESSID**. From The MD.Addresses table select **CITY**, **POSTALCODE**, **STREET**, **BUILDING**, **COUNTRY**, and **REGION**.



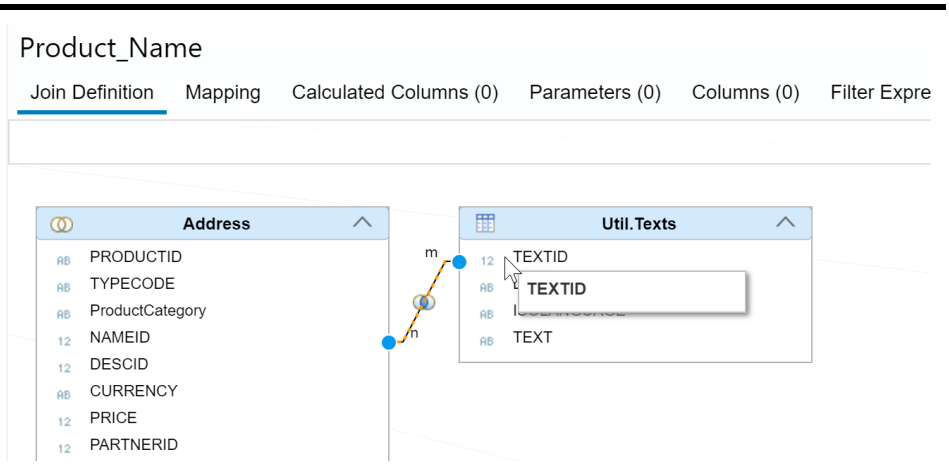
25. Repeat the process of adding a **Join Node**. Name this new Join Node **Product_Name** and connect the output **Address** to this new join node.



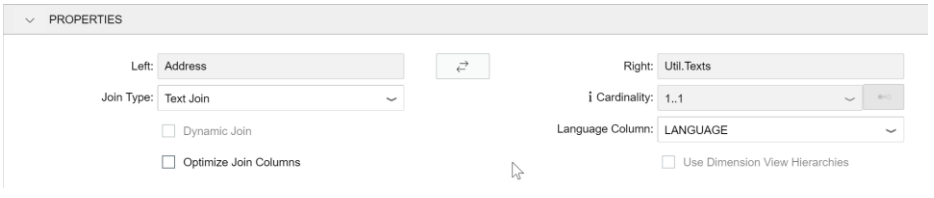
26. Add the **Util.Texts** table to this join node.



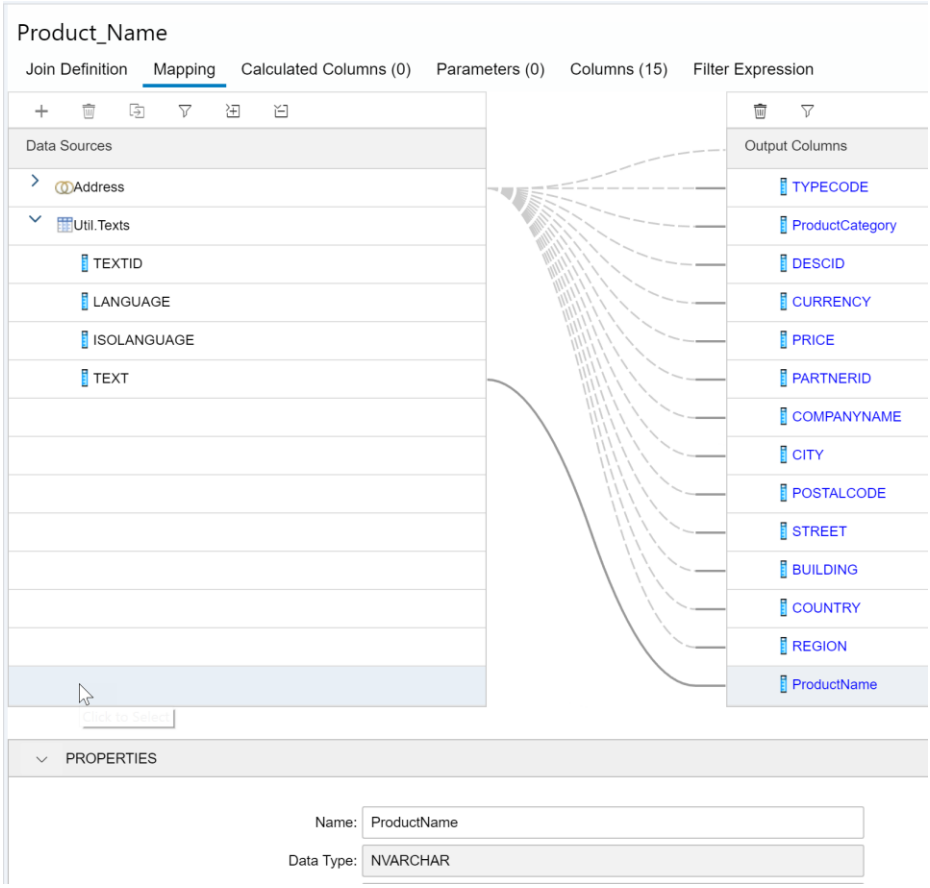
27. Create a join between the **NAMEID** of the previous join node output and the **TEXTID** column of the **Util.Texts** table.



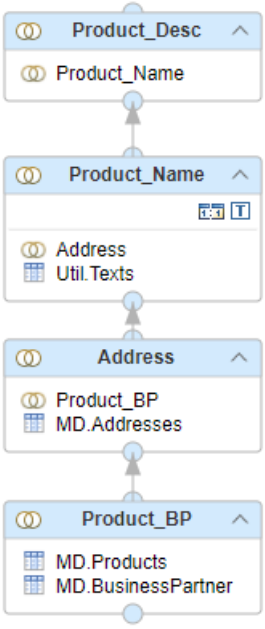
28. In the Join Properties window, change the Join Type to Text Join and the **Language** Column to **LANGUAGE**.



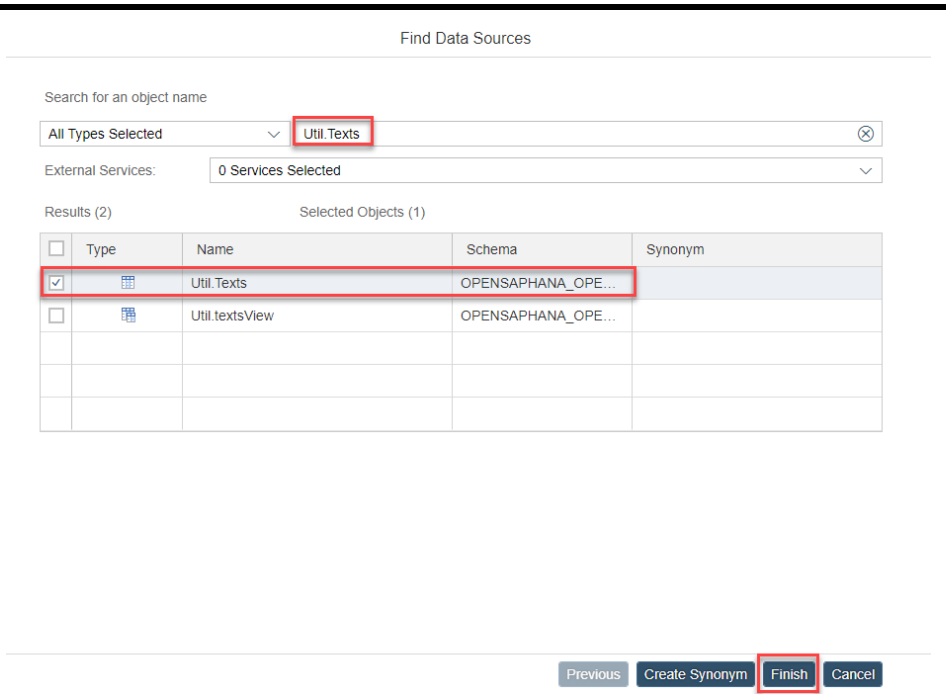
29. Repeat the process of adding columns to the output via the **Mapping** tab. Select all columns from the Address node except NAMEID. From the Util.Texts table select TEXT but change the name of the TEXT column in the output to ProductName .



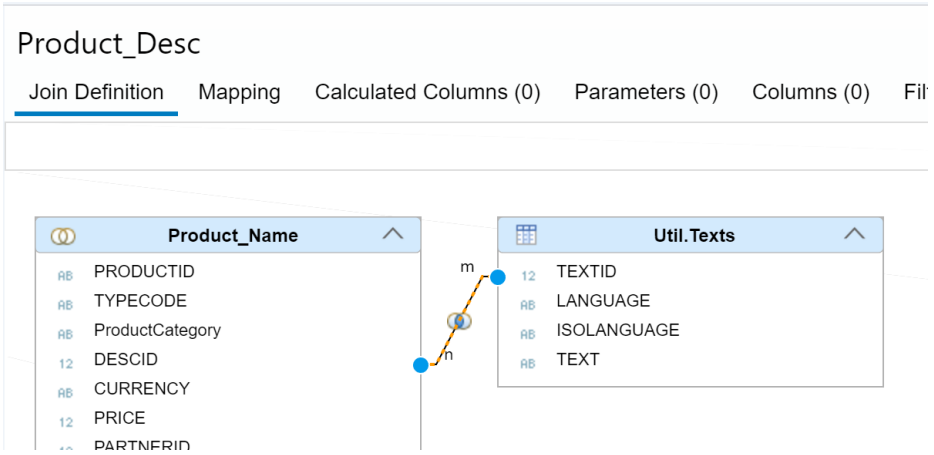
30. Repeat the process of adding a **Join Node**. Name this new Join Node **Product_Desc** and connect the output **Product_Name** to this new join node.



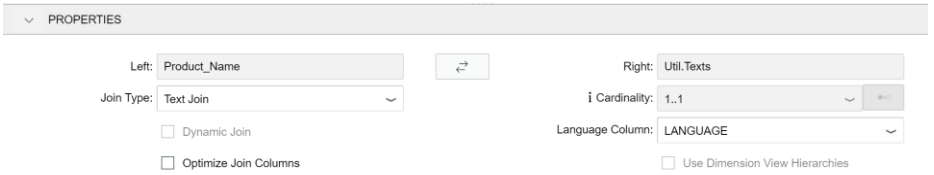
31. Add the **Util.Texts** table to this join node.



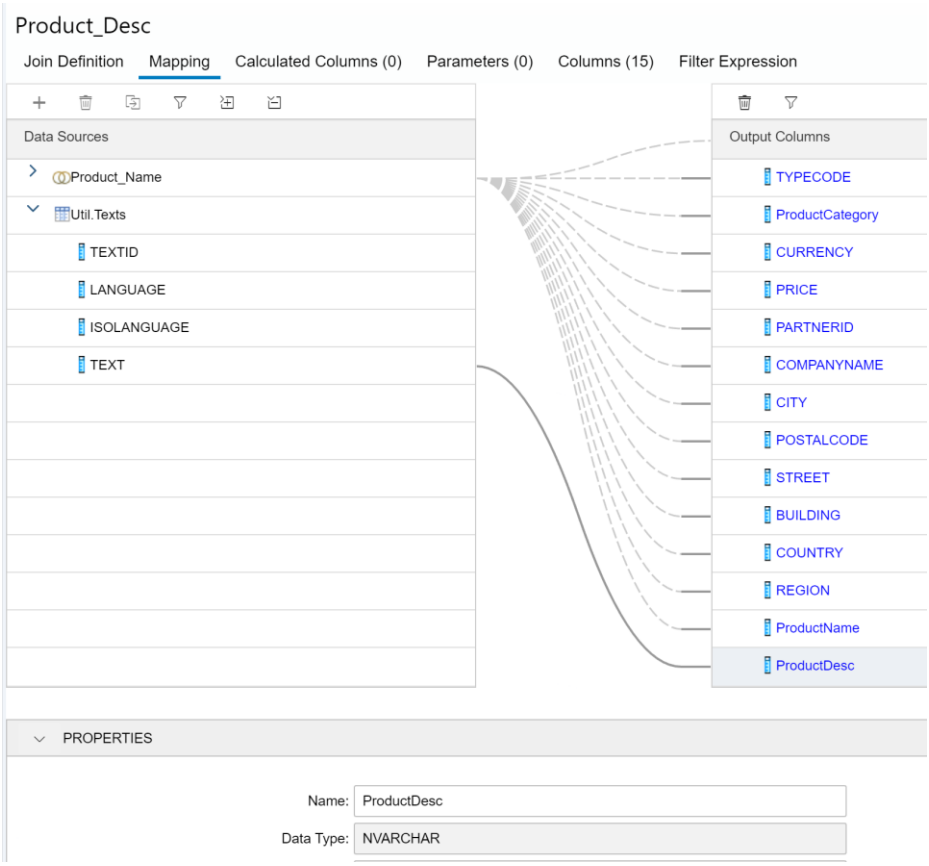
32. Create a join between the **DESCID** of the previous join node output and the **TEXTID** column of the Util.Texts table.

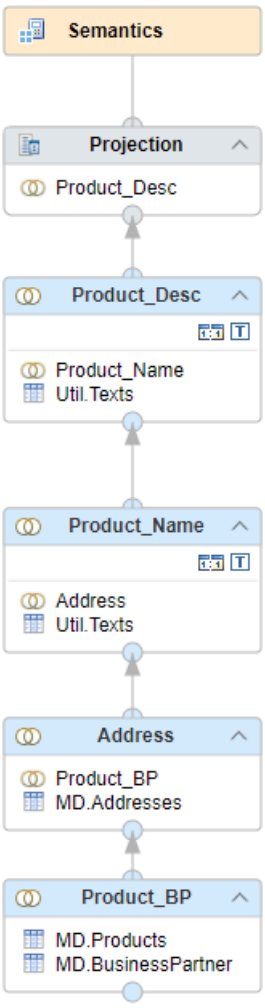
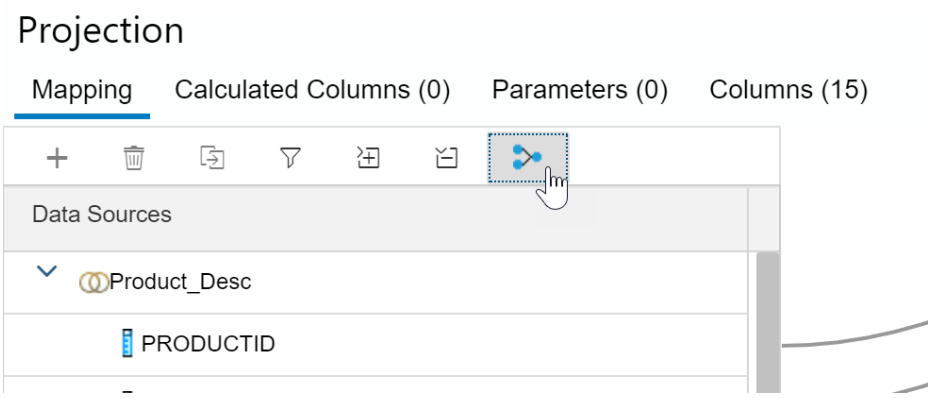


33. In the Join Properties window, change the Join Type to **Text Join** and the Language Column to **LANGUAGE**.

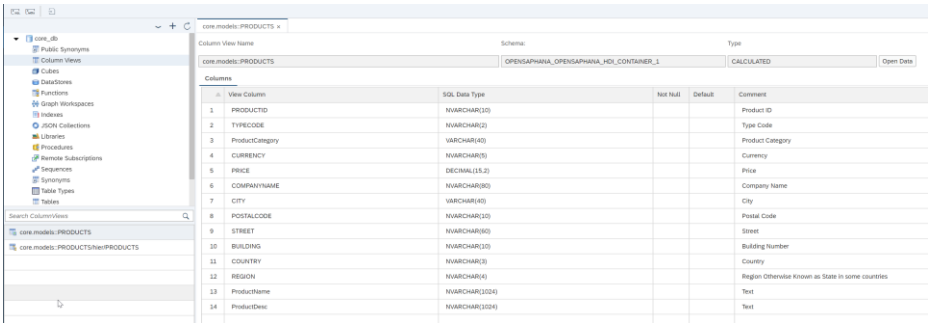


34. Repeat the process of adding columns to the output via the mapping tab. Select all columns from the **Product_Name** node except **DESCID**. From the **Util.Texts** table select **TEXT** but change the name of the **TEXT** column in the output to **ProductDesc** .

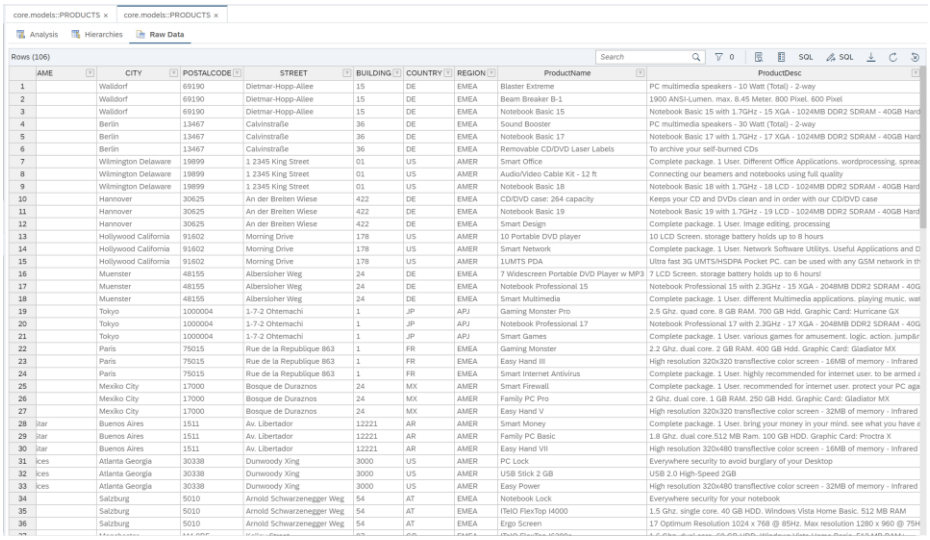


35. Connect the output of the **Product_Desc** node to the **Projection** node at the top of the design window.	
36. In the **Projection** node's **Details Panel** and **Mapping** tab, press the **Auto Map by Name button**.	
37. Save your model	

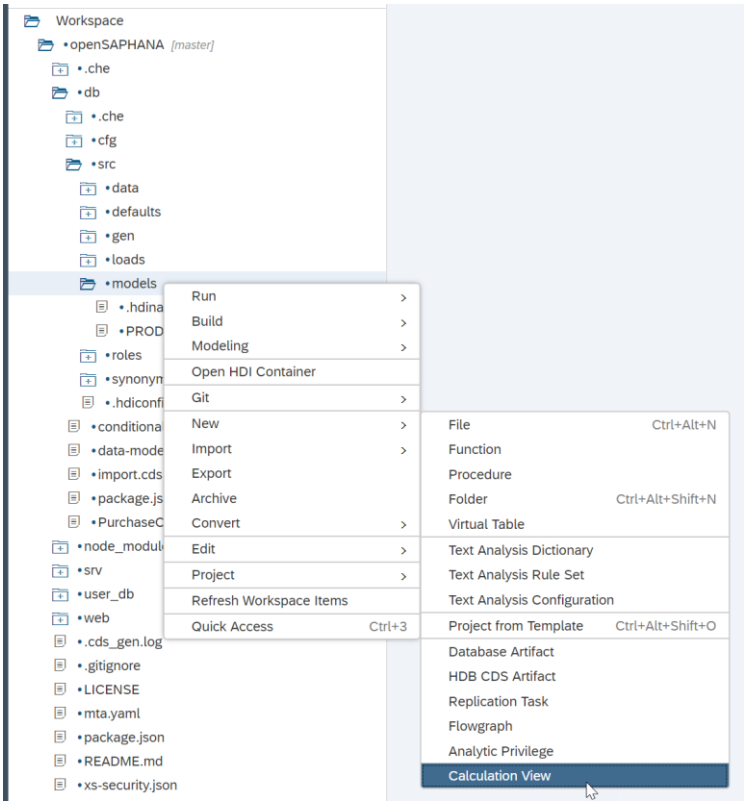
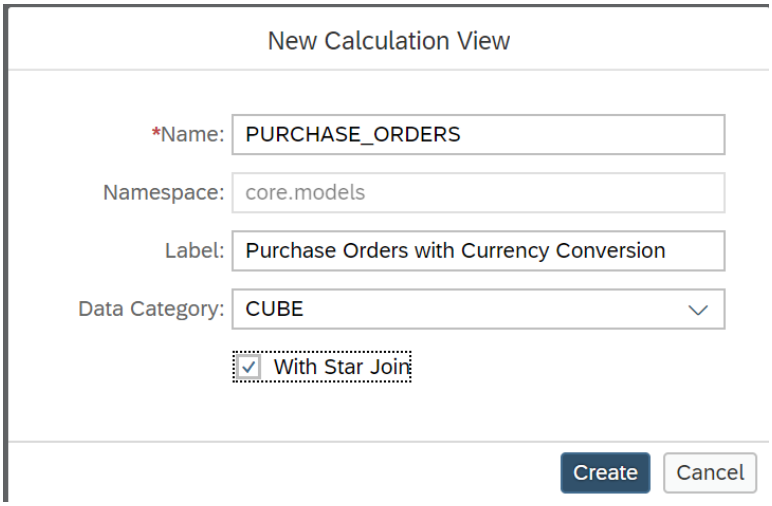
38. Build the **db** module and then return to the Database Explorer perspective. Your container will now have an entry in the **Column Views** folder for this new Calculation View.



39. For an initial test make sure your output looks similar to the following:

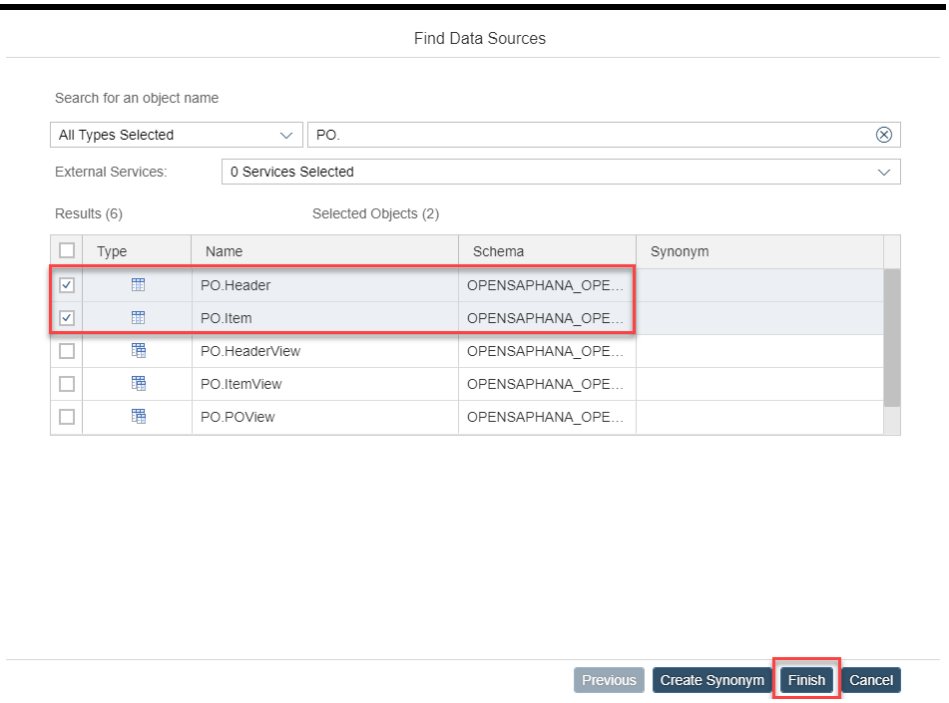


Exercise 2.10: Creating a Calculation View with a Cube data type and Star Join

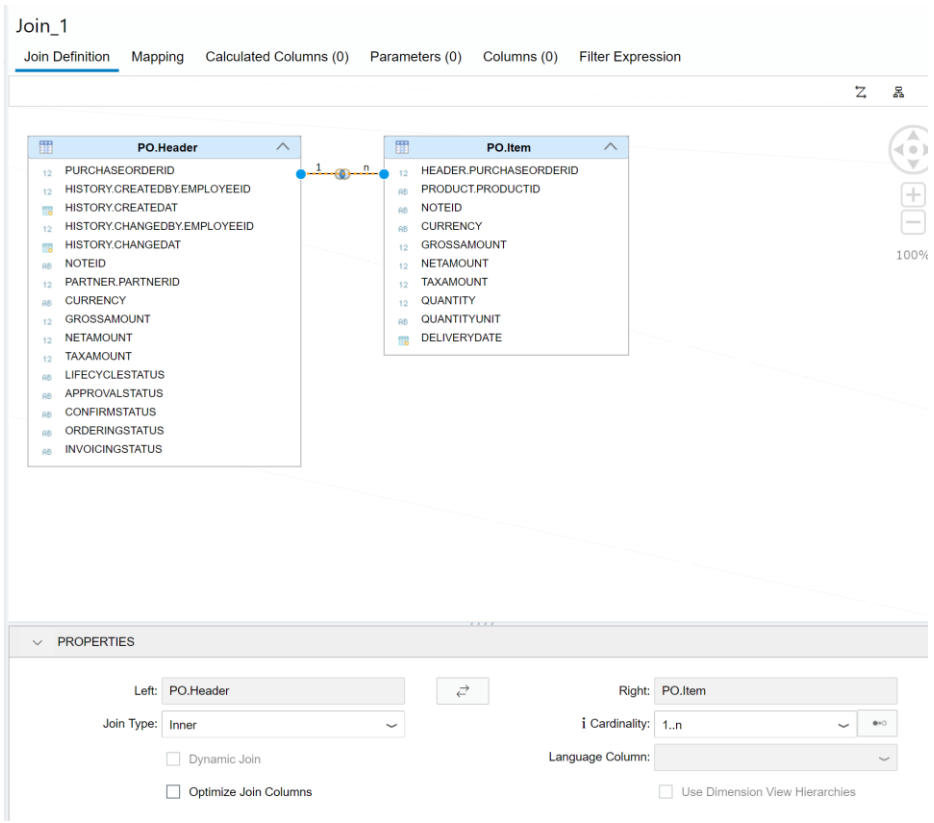
Explanation	Screenshot
1. Now we will create another Calculation view – but one which uses the Cube data type and therefore aggregation of the results. It will combine purchase order table data with the product view we created in the previous step. Because we have various currency based amounts in our source data, we will also create a calculated column which contains a currency conversion. Using the same steps as the previous part of the exercise; in your models folder, create a new Calculation View	
2. Name your new view **PURCHASE_ORDERS.** Set the Data Category to **CUBE** and choose **With Star Join.** This will create a view very similar in capabilities to the older Analytical View**.**	

3. Insert a **Join** Node into the view.

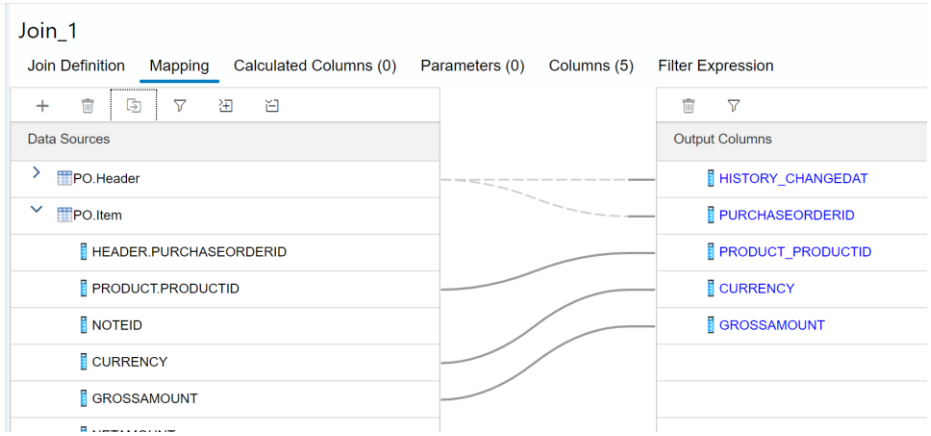
   Add the **PO.Header** and **PO.Item** tables as Data Sources to the to this Join Node by selecting them then pressing **Finish**.



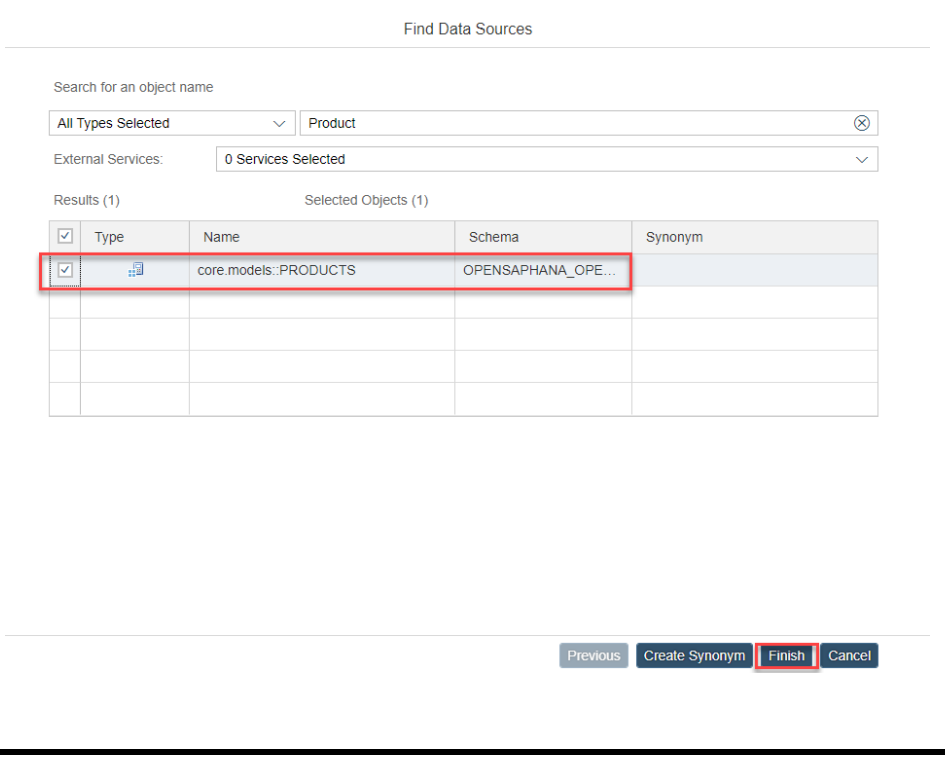
4. Create an inner join between **PURCHASEORDERID** from the header table to the **HEADER.PURCHASEORDERID** column of the item table and set the cardinality as **1..n**.

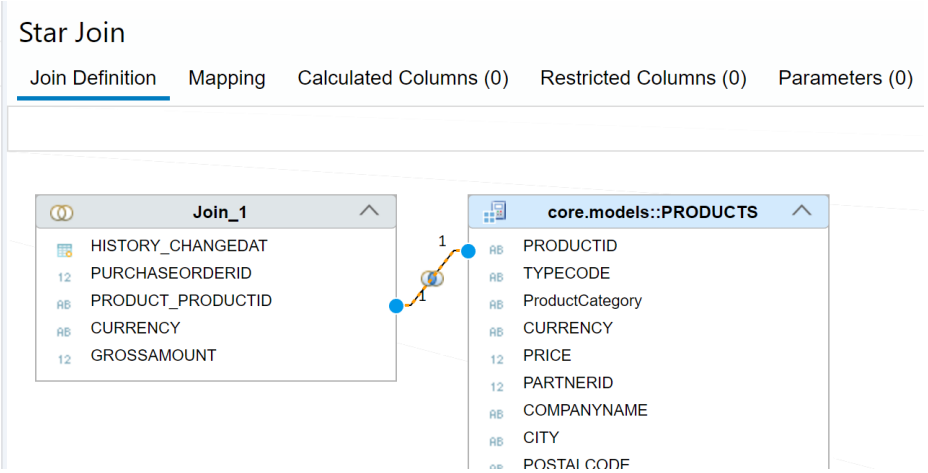
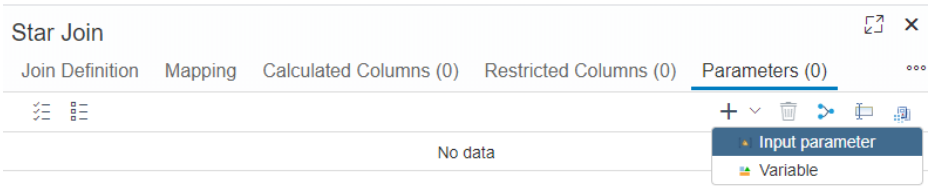


5. Using the Mapping tab, add the columns **PURCHASEORDERID** and **HISTORY.CHANGEDAT** from the Header table and the **PRODUCT.PRODUCTID, CURRENCY**, and **GROSSAMOUNT** columns from the Item table.

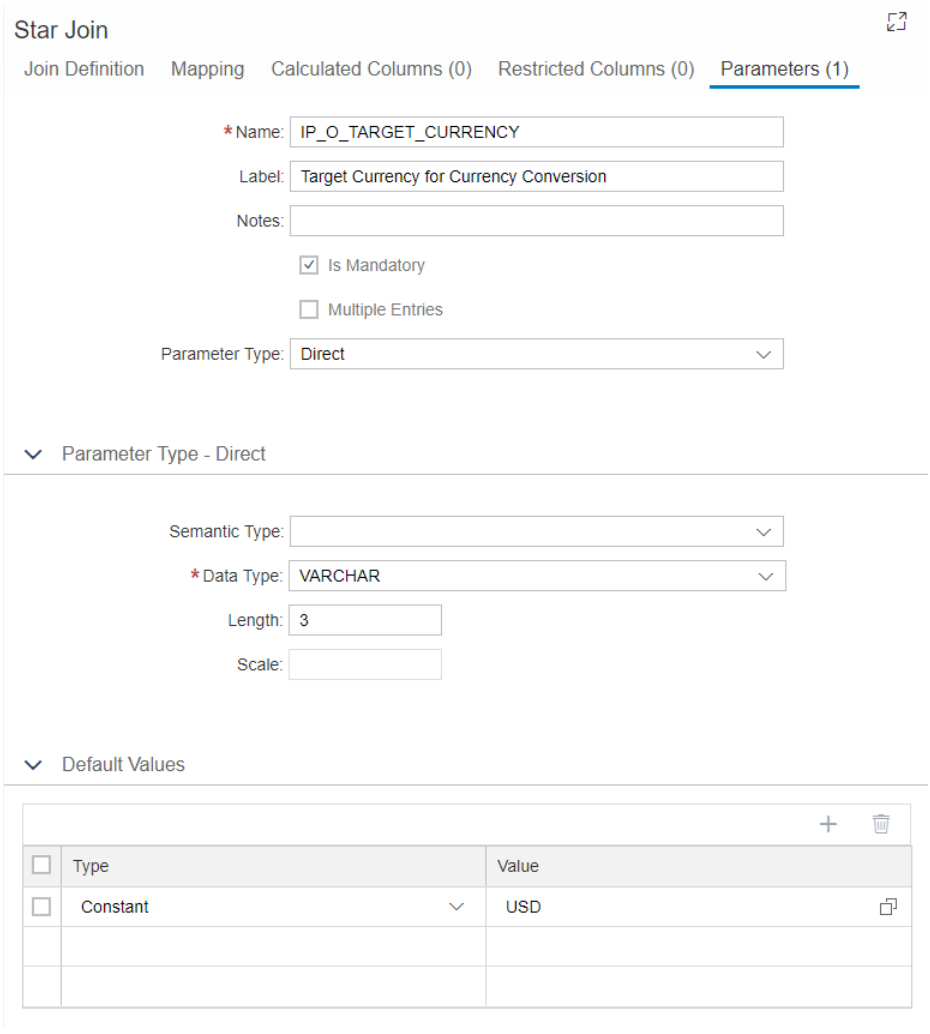


6. Connect the output of the **Join** to the **Star Join** node. Press the plus button next to this Star Join node to add a Data Source. Search for and select the **PRODUCTS** view from the previous part of the exercise to insert it into the Join of your new view by pressing **Finish**.

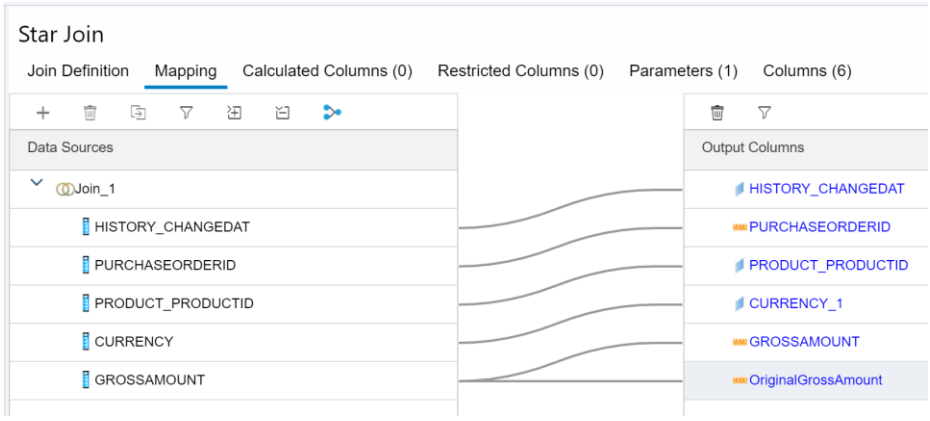


7. Create a join on the **PRODUCT_PRODUCTID** to the **PRODUCT_ID** column and set the cardinality to **1 .. 1**.	
8. From the Parameters tab, create an input parameter named **IP_O_TARGET_CURRENCY**.	

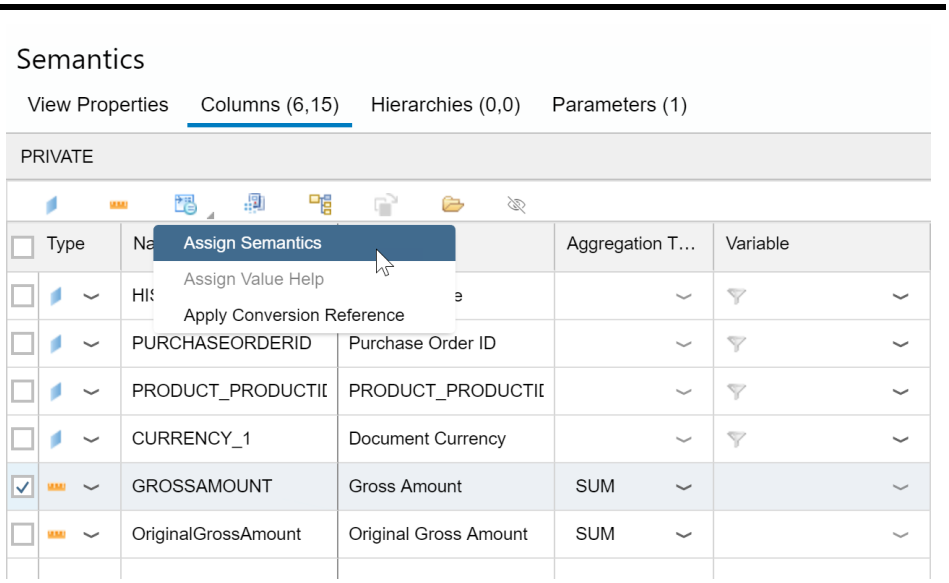
9. Configure as shown with type **NVARCHAR** length **3** with a Semantic type of **Currency**. It should be **mandatory** and have a default value of **USD**.



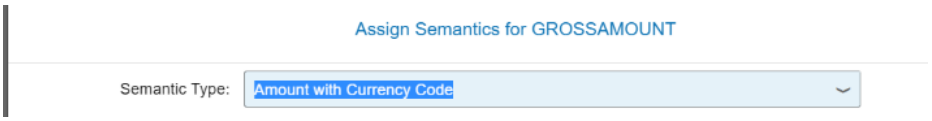
10. From the **Mapping** tab, select **all** the columns of the Star Join node and add them to the output. Add **GROSSAMOUNT** a second time and change the column name to **OriginalGrossAmount**



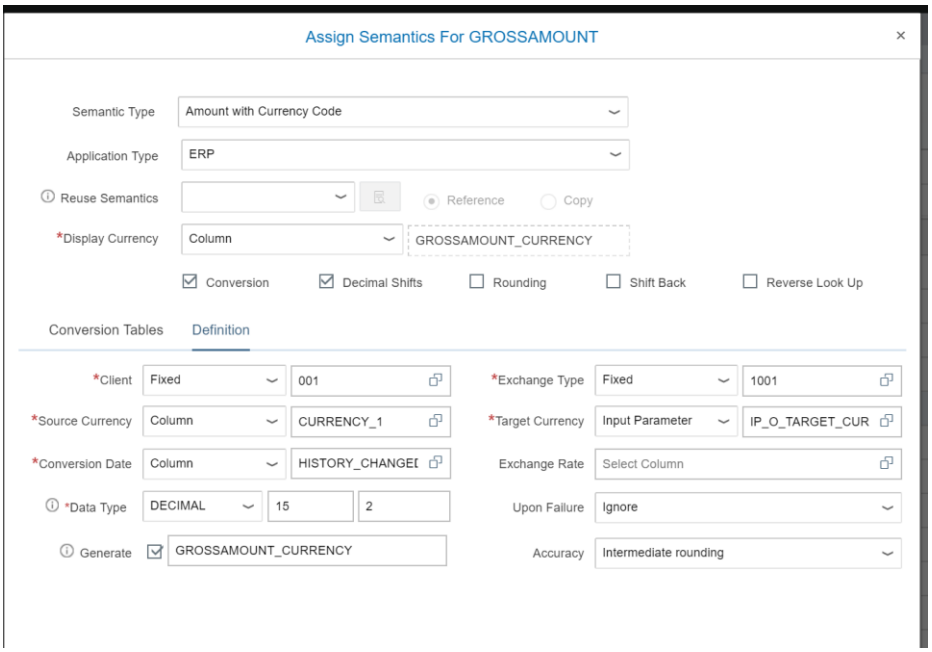
11. Go to the Semantics Node. Change the **PURCHASEORDERID** column from Measure to **Attribute**. Select the **GROSSAMOUNT** measure and chose **Assign Semantics**



12. For the Semantic Type choose **Amount with Currency Code**

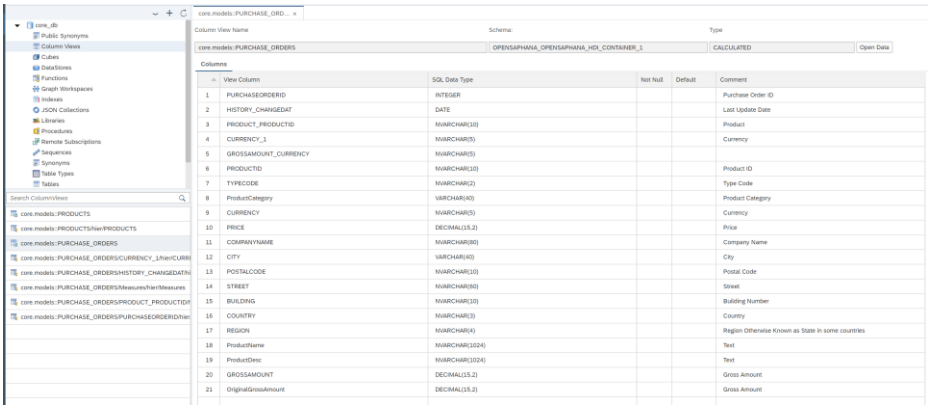


13. Configure the Currency Conversion as shown in the following screenshot.

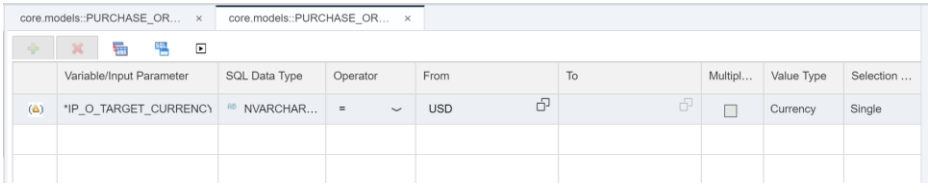


14. Save your View.

15. Build the **db** module and then return to the Database Explorer perspective. Your container will now have an entry in the **Column Views** folder for this new Calculation View.

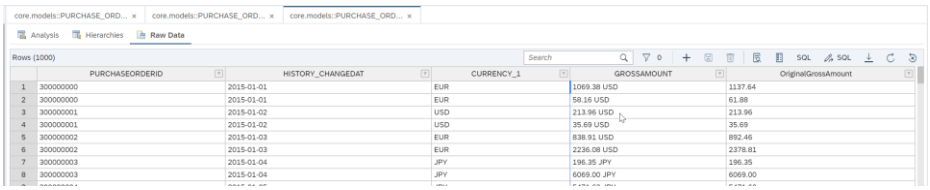


16. When you **Open Data** for this view you should recieve this screen that allows you set values for the input parameters. Keep the default value of USD and execute by presing **Open Content**

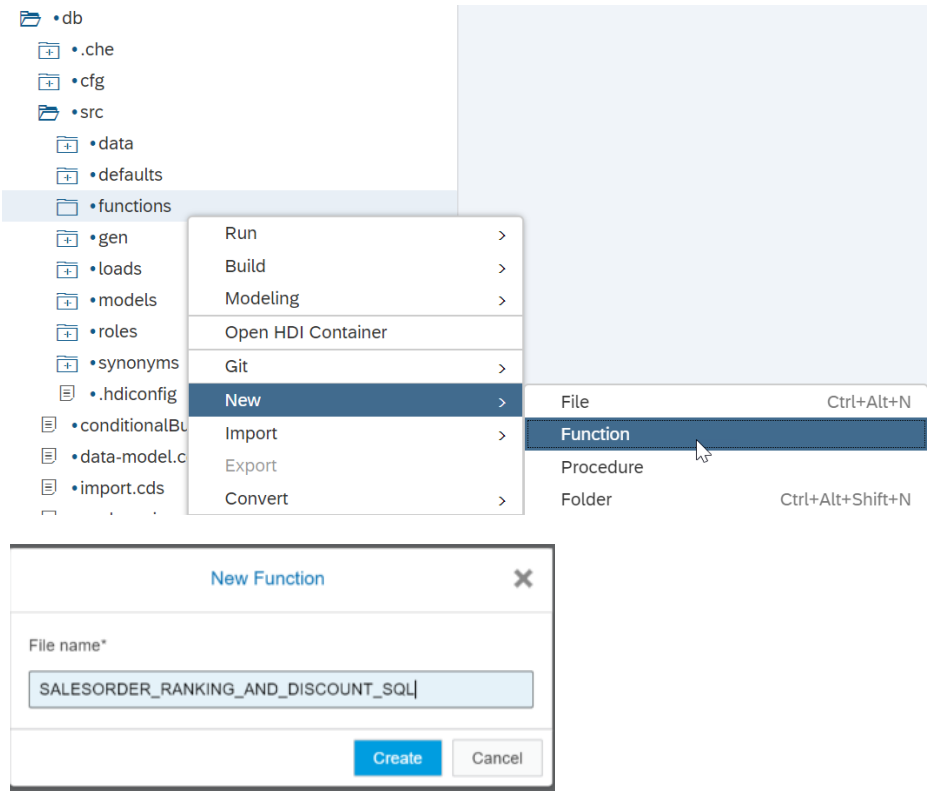


17. You should see output similar to this. In particular check the values of the **GROSSAMOUNT** and OriginalGrossAmount Columns.

    In this workshop system exchange rates have only been maintained for USD<->EUR; therefore those would be the only records with different values in the OriginalGrossAmt column. Exchange rates are not necessarily accurate in this system. It is only test data to show a conversion takes place.



Exercise 2.11: Table Function as Data Source in Calculation View

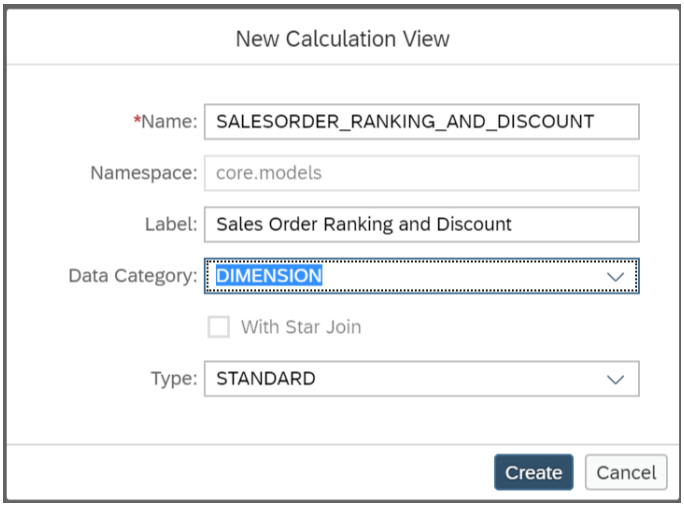
Explanation	Screenshot
1. Within HDI, there is no such thing as a Scripted Calculation Views. Instead now we will create a SQLScript Table Function and then wrap that in a Graphical Calculation View. In this exercise we want to create such a combination of scripted logic and Calculation view that will select Sales details and ranks. Begin by creating a new folder under **db/src** called **functions**. Then create a new Function named **SALESORDER_RANKING_AND_DISCOUNT_SQL** in the functions folder.	
2. Input the following code into this function. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_16	FUNCTION "SALESORDER_RANKING_AND_DISCOUNT_SQL" (IP_FR_DATE DATE, IP_TO_DATE DATE, IP_REGION NVARCHAR(4)) RETURNS table (COMPANY_NAME NVARCHAR(80), REGION NVARCHAR(4), SALES DECIMAL(18,2), ORDERS INTEGER, SALES_RANK INTEGER, ORDER_RANK INTEGER) LANGUAGE SQLSCRIPT AS BEGIN /***************************** Write your function logic *****************************/ return SELECT company_name, region, sales, orders, sales_rank, order_rank from(select

```
        T2."COMPANYNAME" as COMPANY_NAME,
        T3."REGION" as REGION,
        sum(T1."NETAMOUNT") as SALES,
        count(T0."SALESORDERID") as ORDERS,
        dense_rank() over ( order by sum(T1."NETAMOUNT") desc ) as
sales_rank,
        dense_rank() over ( order by count(T0."SALESORDERID") desc ) as
order_rank
        --T1."DELIVERYDATE"
        from "SO.Header" T0
    inner join "SO.Item" T1 on T0."SALESORDERID" =
T1."HEADER.SALESORDERID"
    inner join "MD.BusinessPartner" T2 on T0."PARTNER.PARTNERID" =
T2."PARTNERID"
    inner join "MD.Addresses" T3 on T2."ADDRESSES.ADDRESSID" =
T3."ADDRESSID"
    where TO_DATE(T1."DELIVERYDATE") between :IP_FR_DATE and
:IP_TO_DATE
      and T3."REGION" = :IP_REGION
    group by T2."COMPANYNAME", T3."REGION"
    order by sales_rank, T2."COMPANYNAME"
    )
    where sales_rank < 11;

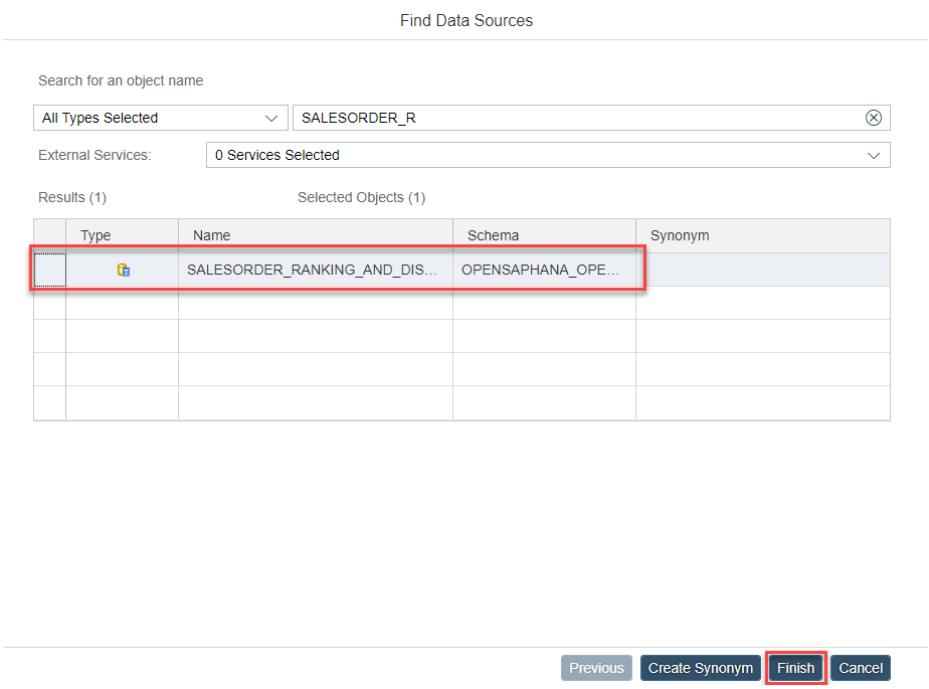
END
```

3. Save your function and then perform a build on the **db** module.

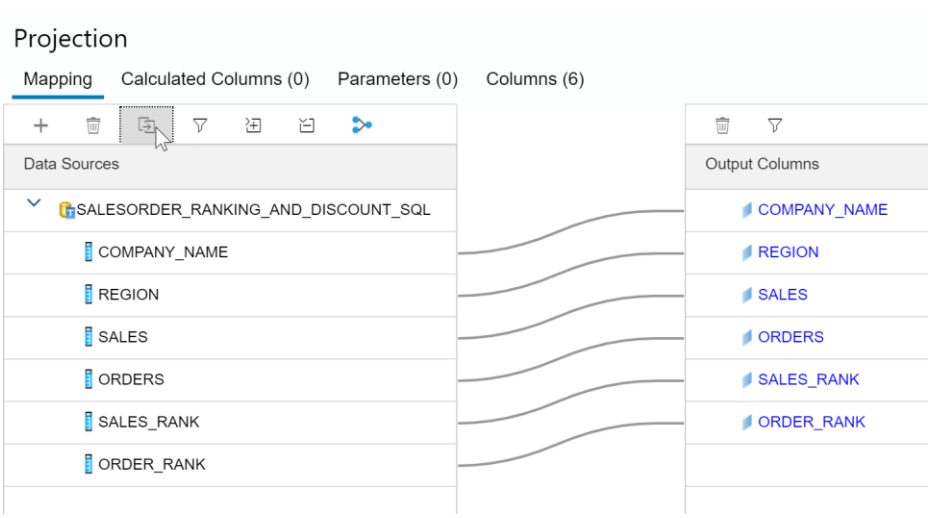
4. Now from the src/models folder create a new Calculation view named **SALESORDER_RANKING_AND_DISCOUNT** of type **DIMENSION**.



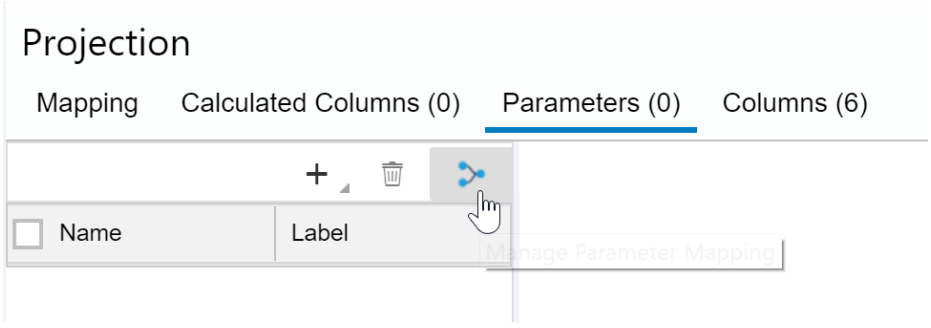
5. In the projection node, click the plus and add the **SALESORDER_RANKING_AND_DISCOUNT_SQL** function



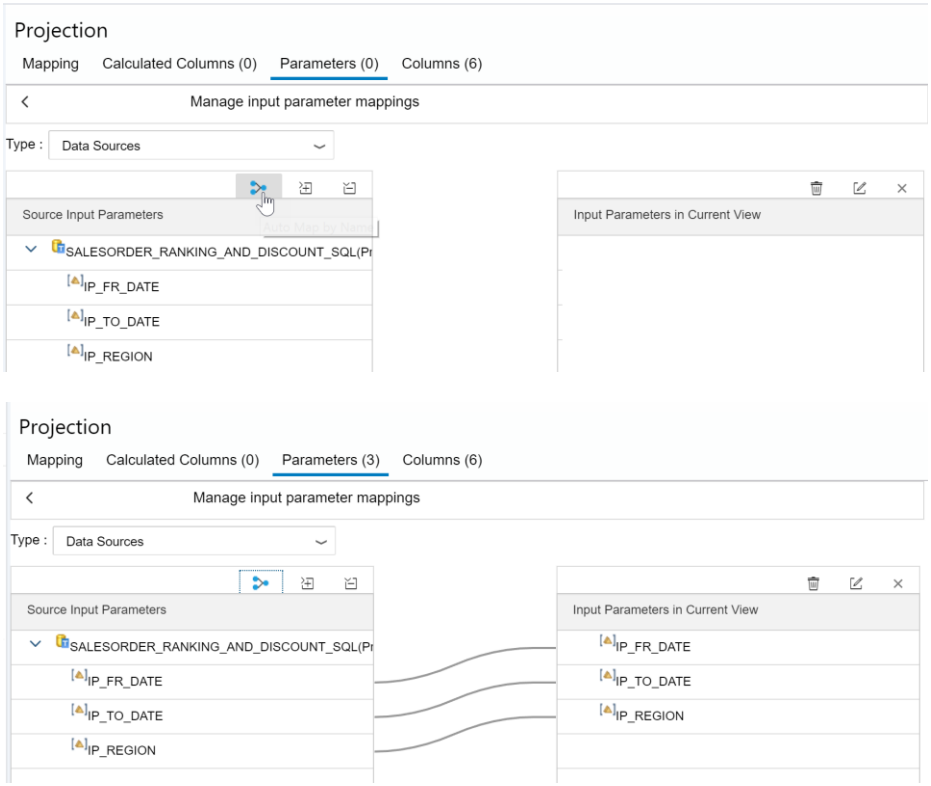
6. In the **Mapping**, add **all** the columns to the Output



7. Go to the **Parameters** tab and press the **Manage Parameter Mapping** button

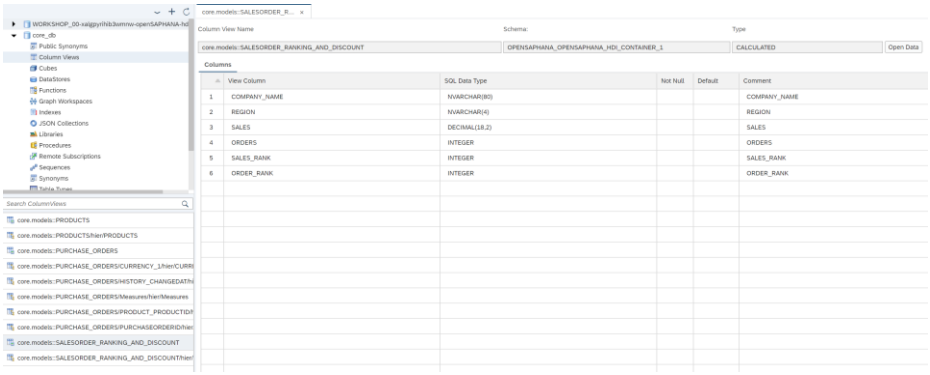


8. Choose the **Auto Map** button. This will generate matching Input Parameters in your view for all the input parameters exposed by the SQLScript Function.

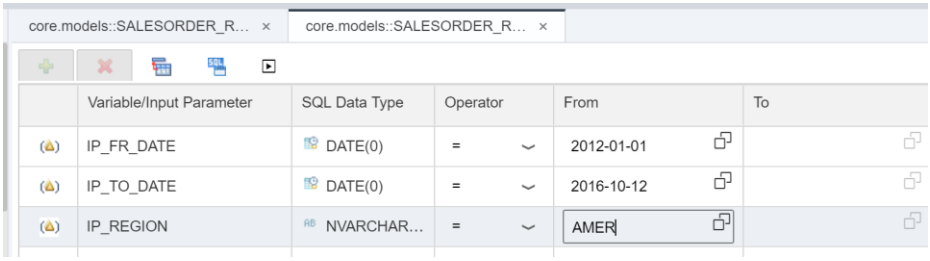


9. Save your View.

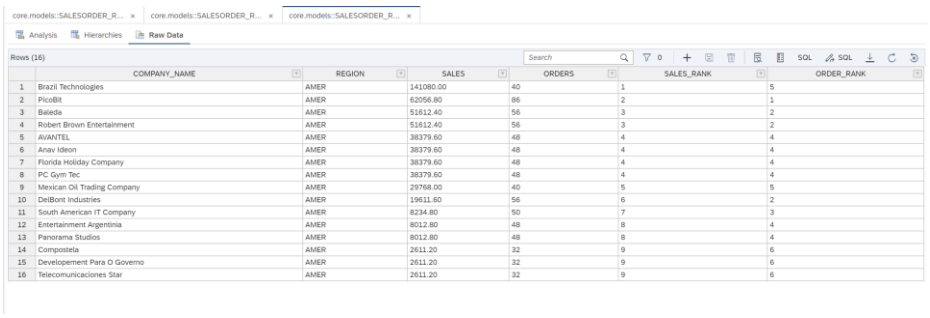
10. Build the **db** module and then return to the Database Explorer perspective. Your container will now have an entry in the Column Views folder for this new Calculation View. Choose Open Data.



11. In the **From Date**, enter January 1st 2012. In the **To Date**, use the current date. In the **Region** use the value AMER. Press F8 to execute.



12. The output values will then be displayed

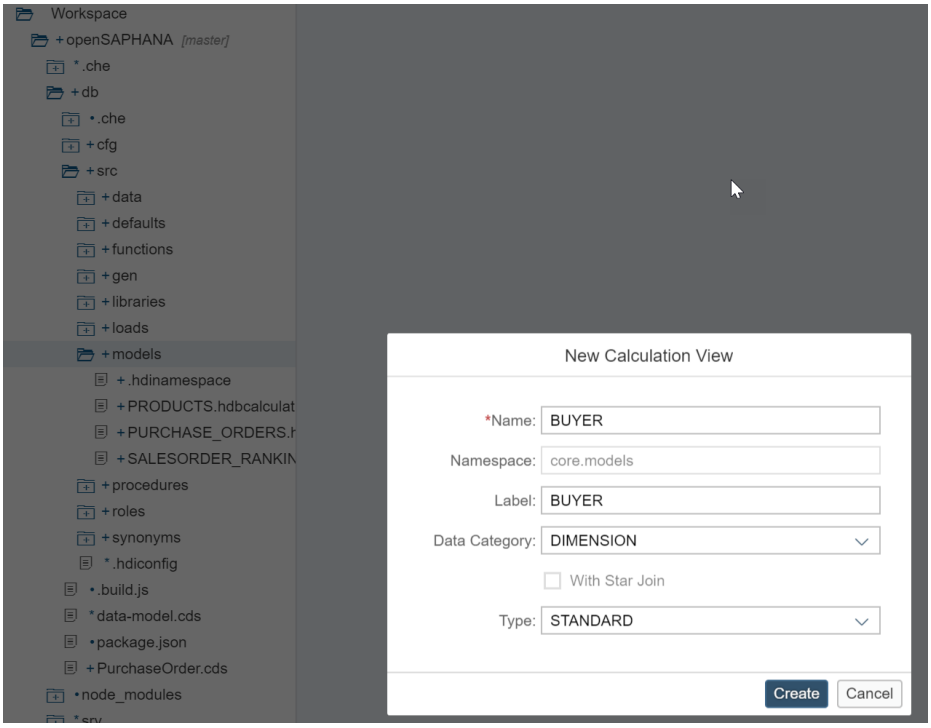


	COMPANY_NAME	REGION	SALES	ORDERS	SALES_RANK	ORDER_RANK
1	Brazil Technologies	AMER	141080.00	40	1	5
2	PicoBit	AMER	62056.80	86	2	1
3	Baleda	AMER	51612.40	56	3	2
4	Robert Brown Entertainment	AMER	51612.40	56	3	2
5	AVANTEL	AMER	38379.60	48	4	4
6	Anav Ideon	AMER	38379.60	48	4	4
7	Florida Holiday Company	AMER	38379.60	48	4	4
8	PC Gym Tec	AMER	38379.60	48	4	4
9	Mexican Oil Trading Company	AMER	29768.00	40	5	5
10	DeliBont Industries	AMER	19611.60	56	6	2
11	South American IT Company	AMER	8234.80	50	7	3
12	Entertainment Argentinia	AMER	8012.80	48	8	4
13	Panorama Studios	AMER	8012.80	48	8	4
14	Compostela	AMER	2611.20	32	9	6
15	Developement Para O Governo	AMER	2611.20	32	9	6
16	Telecomunicaciones Star	AMER	2611.20	32	9	6

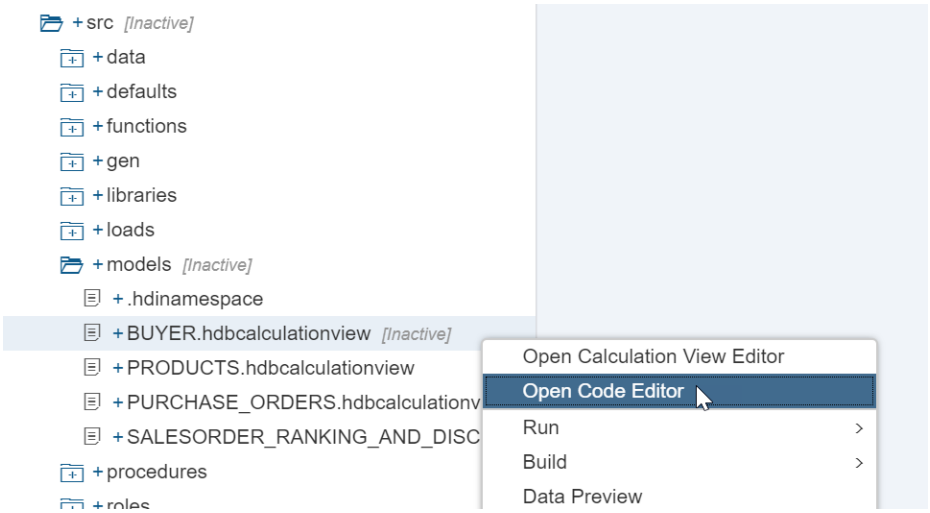
Exercise 2.12: Importing Calcuation Views (and other native DB objects) into the new CDS

Explanation	Screenshot
1. With the new CDS, we don't yet have a way to directly import a Calcuation View or other native DB artifact into CDS. This import, however, is important if you want to expose these artifacts via Odata V4 services, since that requires a CDS entity or view. But CDS does have an annotation called `@cds.persistence.exists`. This annotation allows you to re-define an existing DB object and CDS won't attempt to create or alter it. It will just assume it already exists in the matching state. This feature does require you to completely redefine the DB artifact with exactly the same name, columns and column names. In this exercise, lets see how we can create a calcuation view and other artifacts and expose them to CDS.	

2. Let's begin by importing a new Calcuation View which exposes Business Partner records. Create a new Calcuation View named **BUYER** in the **models** folder.

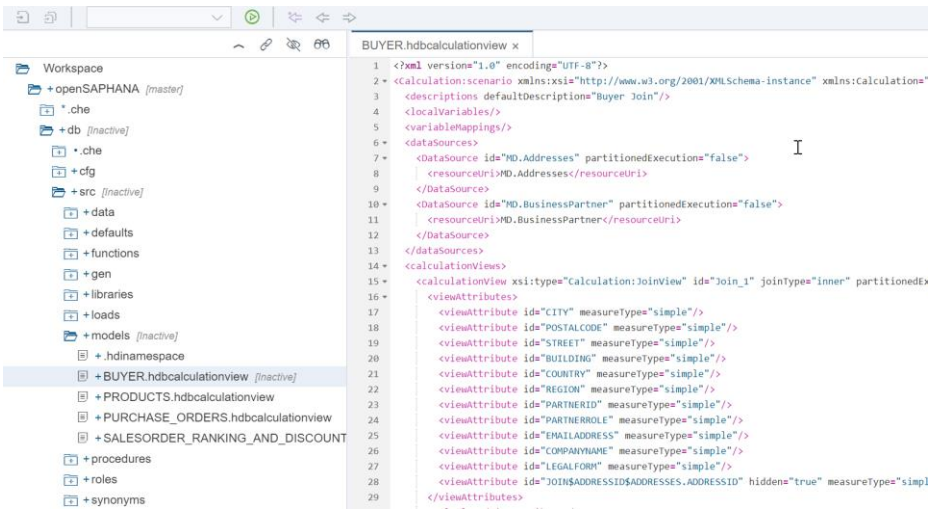


3. Close the Calcuation View editor and right mouse click on **BUYER** and choose **Open Code Editor**.



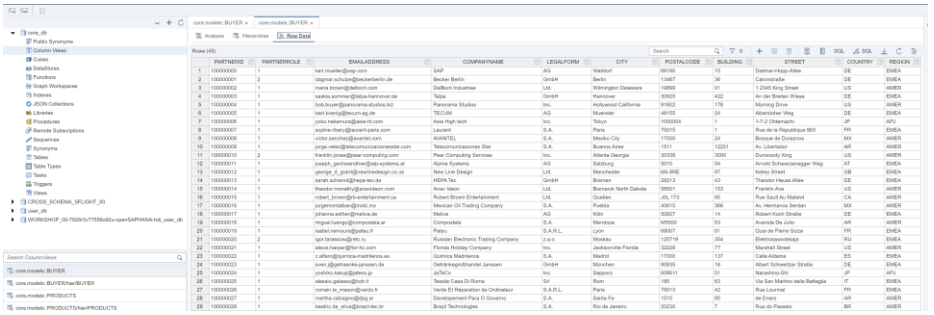
4. You can download the XML definition of this Calcuation View here: https://github.com/SAP/hana-xsa-opensap-hana7/blob/master/db/src/models/BUYER.hdbcalculationview

   Save the View



5. Build and test your new Calcuation View from the **Database Explorer**. It should return details similar to the following.

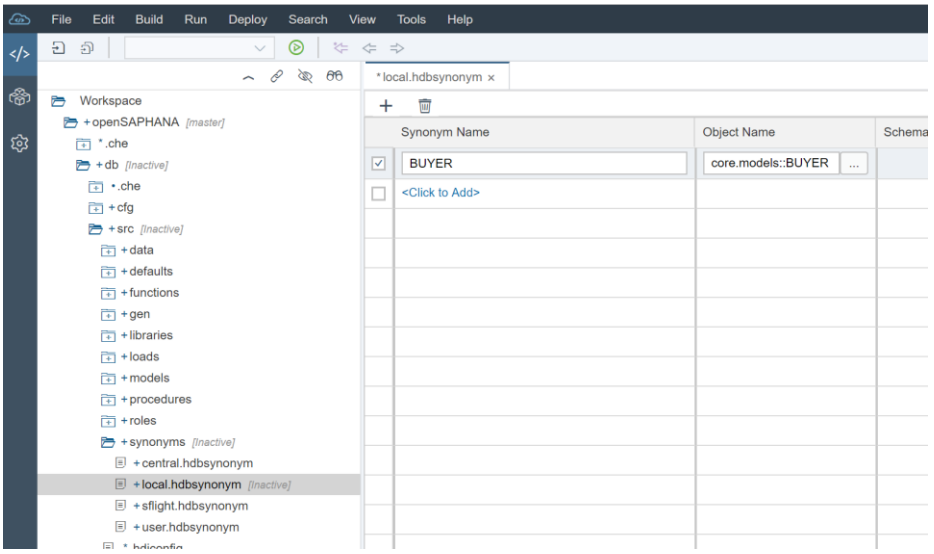
   However we still have a small problem becuase of the name **core.models::BUYER**. We can't use a name like that in CDS.



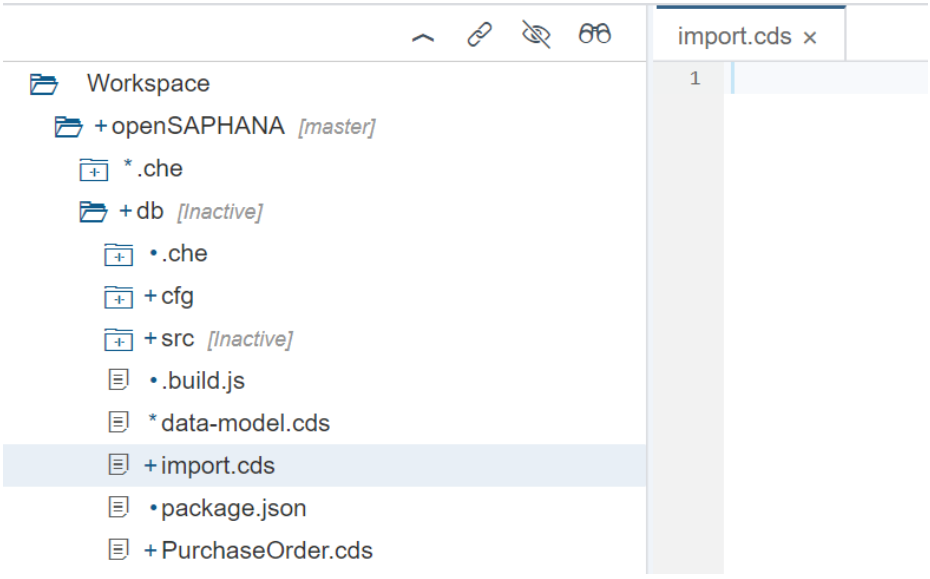
6. This is where a synonym can help. We've used a synonym to access an object from another container or schema, but we can use them locally to create an "alias" name for an object.

   Create a new file named **local.hdbsynonym** in the **db/src/synonyms** folder.

   Add a synonym called **BUYER** for the Calculation View named **core.models::BUYER**.



7. We are now ready to add BUYER to CDS. Let's create a new cds file named **import.cds** in the **db** folder.



8. Add the matching entity defintion for **BUYER** to the import.cds file.

   The completed source code can be found at the following URL.

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/cds2.txt

```
@cds.persistence.exists
Entity BUYER {
    BUILDING: String(10);
    CITY: String(40);
    COMPANYNAME: String(80);
    COUNTRY: String(3);
    EMAILADDRESS: String(255);
    LEGALFORM: String(10);
    key PARTNERID: Integer;
    PARTNERROLE: String(3);
    POSTALCODE: String(10);
    REGION: String(4);
    STREET: String(60);
}
```

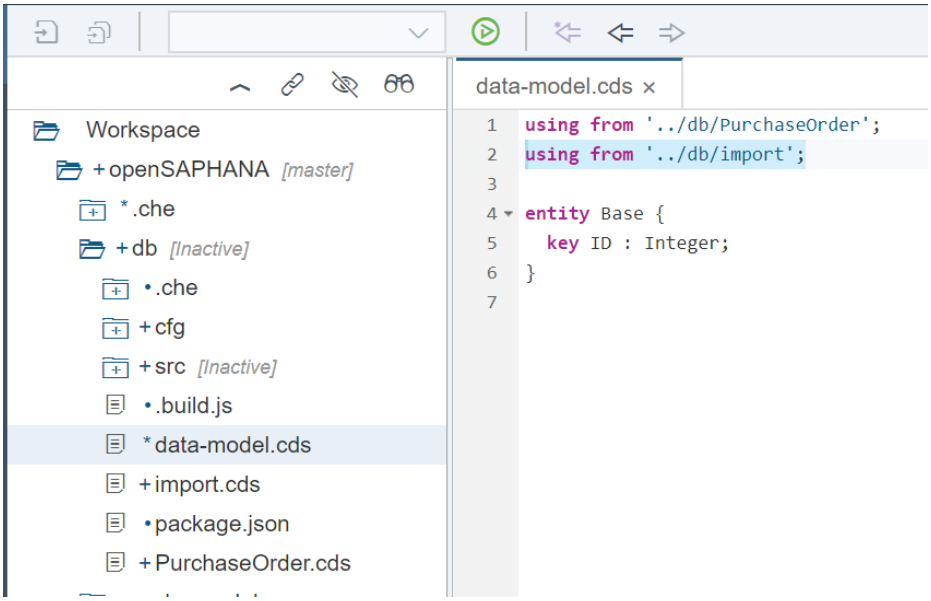
9. This approach isn't limited to Calcuation Views, however. We can use this to access regular SQL views, DB tables, etc. Let's create a new SQL view in the **db/src/data/PurchaseOrder** folder named **PURCHASE_ORDER_ITEM_VIEW.hdbview**

   Note: The completed source code can be found at the following URL.

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/view2.txt

```
VIEW "PURCHASE_ORDER_ITEM_VIEW" COMMENT 'Purchase Order Item View' (
    "PO_ITEM_ID",
     "PARTNER_ID",
     "PRODUCT_ID",
     "CURRENCY_CODE",
     "AMOUNT",
     "NET_AMOUNT",
     "TAX_AMOUNT",
     "QUANTITY",
     "QUANTITY_UNIT",
     "DELIVERY_DATE"
     ) AS SELECT
     "Item_$0"."POHeader"."PURCHASEORDERID" AS "PO_ITEM_ID" ,
     "Item_$0"."POHeader"."PARTNER" AS "PARTNER_ID" ,
     "Item_$0"."PRODUCT" AS "PRODUCT_ID" ,
     "Item_$0"."CURRENCY" AS "CURRENCY_CODE" ,
     "Item_$0"."GROSSAMOUNT" AS "AMOUNT" ,
     "Item_$0"."NETAMOUNT" AS "NET_AMOUNT" ,
```

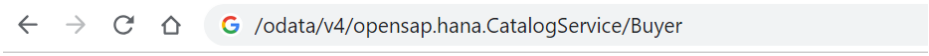
	"Item_$0"."TAXAMOUNT" AS "TAX_AMOUNT" , "Item_$0"."QUANTITY" AS "QUANTITY" , "Item_$0"."QUANTITYUNIT" AS "QUANTITY_UNIT" , "Item_$0"."DELIVERYDATE" AS "DELIVERY_DATE" FROM "PurchaseOrder.Item" AS "Item_$0"
10. Now return to the **import.cds** file. We can add more entities with the persistence.exists option. We will now wrap this new hdbview from the previous step as well as the USER_DETAILS and CURRENCY synonyms (coming from other containers). **Note:** The completed source code can be found at the following URL. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/cds3.txt	type BusinessKey : String(10); type SDate : DateTime; type CurrencyT : String(5); type AmountT : Decimal(15, 2); type QuantityT : Decimal(13, 3); type UnitT : String(3); type StatusT : String(1); @cds.persistence.exists entity PURCHASE_ORDER_ITEM_VIEW { key PO_ITEM_ID: Integer; PARTNER_ID: BusinessKey; key PRODUCT_ID: BusinessKey; CURRENCY_CODE: CurrencyT; AMOUNT: AmountT; NET_AMOUNT: AmountT; TAX_AMOUNT: AmountT; QUANTITY: QuantityT; QUANTITY_UNIT: UnitT; DELIVERY_DATE: SDate; } @cds.persistence.exists Entity BUYER { BUILDING: String(10); CITY: String(40); COMPANYNAME: String(80); COUNTRY: String(3); EMAILADDRESS: String(255); LEGALFORM: String(10); key PARTNERID: Integer; PARTNERROLE: String(3); POSTALCODE: String(10); REGION: String(4); STREET: String(60); } @cds.persistence.exists Entity USER_DETAILS { EMAIL: String(255); FIRSTNAME: String(40); LASTNAME: String(40); key USERID: Integer; } @cds.persistence.exists

	Entity CURRENCY { key CODE: String(3); CURRENCY: String(80); }
11. The final step is to return to the root cds file – **data-model.cds** – and add a using statement for the import.cds file.	

12. Save all your open files. Perform a Build CDS and then build the **db** module.

    Since these new objects aren't persisted, there isn't much we can see about them now. However later we will create Odata services for these "wrapper" entities

Later:

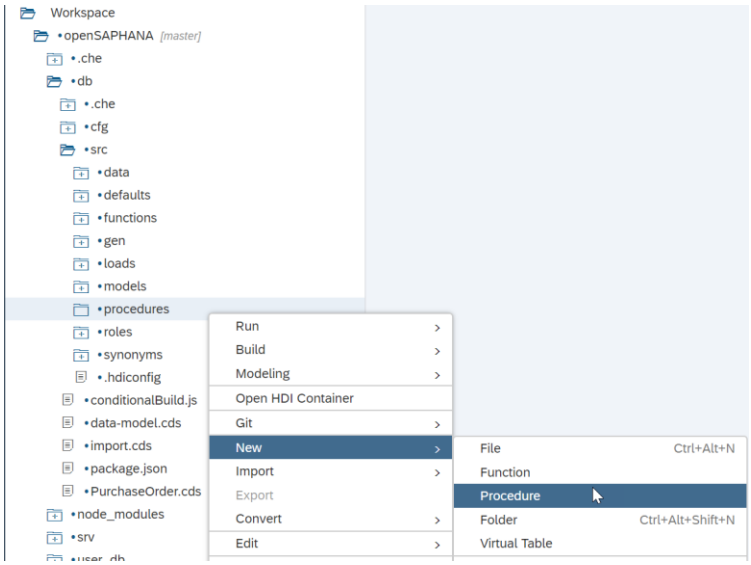
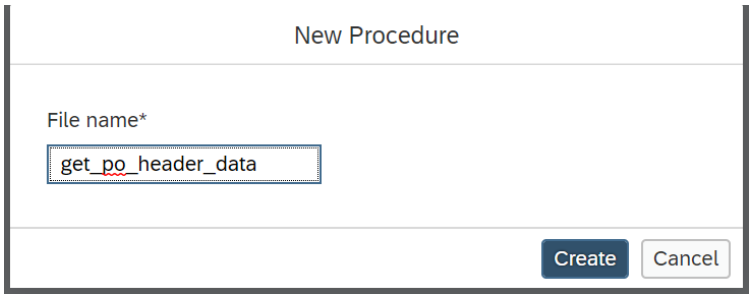
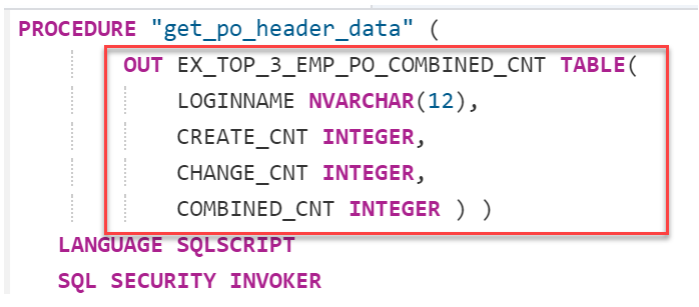



```
{
    @odata.context: "$metadata#Buyer",
    @odata.metadataEtag: "W/"2clLJsS+E3P/7oipDrT13hEozxy5RvfKQdTiZirR4k4=",
  - value: [
    - {
          BUILDING: "15",
          CITY: "Walldorf",
          COMPANYNAME: "SAP",
          COUNTRY: "DE",
          EMAILADDRESS: "karl.mueller@sap.com",
          LEGALFORM: "AG",
          PARTNERID: 100000000,
          PARTNERROLE: "1",
          POSTALCODE: "69190",
          REGION: "EMEA",
          STREET: "Dietmar-Hopp-Allee"
      },
    - {
          BUILDING: "36",
          CITY: "Berlin",
          COMPANYNAME: "Becker Berlin",
          COUNTRY: "DE",
          EMAILADDRESS: "dagmar.schulze@beckerberlin.de",
          LEGALFORM: "GmbH",
          PARTNERID: 100000001,
          PARTNERROLE: "2",
          POSTALCODE: "13467",
          REGION: "EMEA",
          STREET: "Calvinstraße"
      },
    - {
          BUILDING: "01",
          CITY: "Wilmington Delaware",
          COMPANYNAME: "DelBont Industries",
```

Exercise 2.13: Leveraging SQLScript in Stored Procedures & User Defined Functions

Exercise 2.13.1: Creating A Simple Stored Procedure

In this exercise we will create a small procedure “get_po_header_data” with two SELECT queries leveraging tabular output parameters and intermediate table variables.

Explanation	Screenshot
1. Begin by creating a new folder called **procedures** in **/db/src**. Right click on the **procedures** folder and choose **New -> Procedure**.	
2. Enter the name of the procedure as **get_po_header_data**. Click **Create**	
3. Add an output parameter with the following structure. **Note:** The completed source code can be found at the following URL. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_17	

4. Between the BEGIN and END statements, insert the SELECT statements as shown. These are implicit select statements whose results sets are passed to the caller.

```
BEGIN

PO_CREATE_CNT = SELECT COUNT(*) AS CREATE_CNT, "HISTORY.CREATEDBY.EMPLOYEEID" AS EID
        FROM "PO.Header"
          WHERE PURCHASEORDERID IN (
          SELECT "HEADER.PURCHASEORDERID" as "PURCHASEORDERID" FROM "PO.Item"
           WHERE "PRODUCT.PRODUCTID" IS NOT NULL)
        GROUP BY  "HISTORY.CREATEDBY.EMPLOYEEID";

PO_CHANGE_CNT = SELECT COUNT(*) AS CHANGE_CNT, "HISTORY.CHANGEDBY.EMPLOYEEID" AS EID
         FROM "PO.Header"
           WHERE PURCHASEORDERID IN (
           SELECT "HEADER.PURCHASEORDERID" as "PURCHASEORDERID" FROM "PO.Item"
         WHERE "PRODUCT.PRODUCTID" IS NOT NULL)
          GROUP BY  "HISTORY.CHANGEDBY.EMPLOYEEID";

EX_TOP_3_EMP_PO_COMBINED_CNT =
   SELECT emp.LOGINNAME, crcnt.CREATE_CNT, chcnt.CHANGE_CNT,
         crcnt.CREATE_CNT + chcnt.CHANGE_CNT AS COMBINED_CNT
       FROM "MD.Employees" as emp
        LEFT OUTER JOIN :PO_CREATE_CNT AS crcnt
          ON emp.EMPLOYEEID = crcnt.EID
        LEFT OUTER JOIN :PO_CHANGE_CNT AS chcnt
          ON emp.EMPLOYEEID = chcnt.EID
         ORDER BY COMBINED_CNT DESC LIMIT 3;

END
```

5. The completed code should look similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/ex2_17

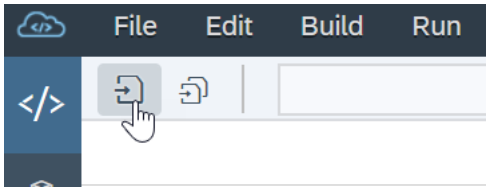
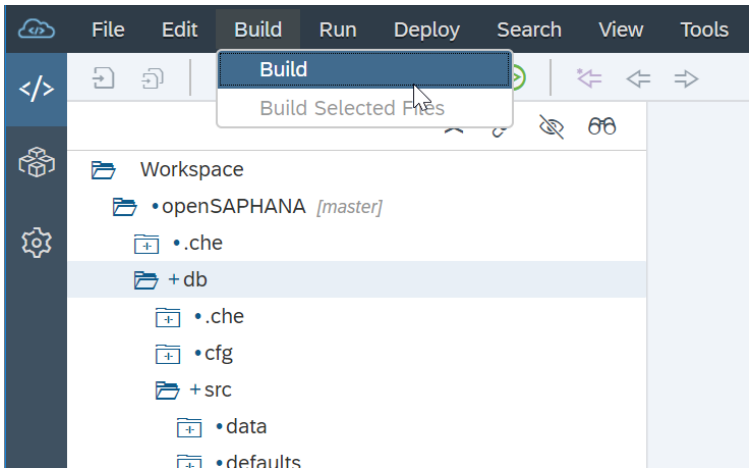
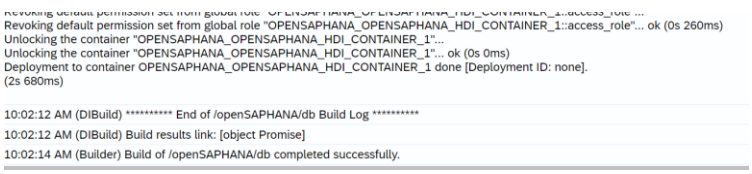
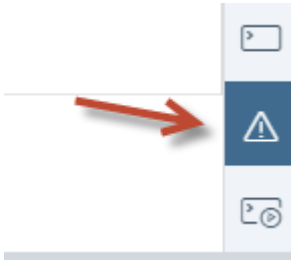
```
PROCEDURE "get_po_header_data" (
        OUT EX_TOP_3_EMP_PO_COMBINED_CNT TABLE(
            LOGINNAME NVARCHAR(12),
                      CREATE_CNT INTEGER,
                      CHANGE_CNT INTEGER,
                      COMBINED_CNT INTEGER ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

PO_CREATE_CNT = SELECT COUNT(*) AS CREATE_CNT,
"HISTORY.CREATEDBY.EMPLOYEEID" AS EID
         FROM "PO.Header"
            WHERE PURCHASEORDERID IN (
            SELECT "HEADER.PURCHASEORDERID" as
"PURCHASEORDERID" FROM "PO.Item"
             WHERE "PRODUCT.PRODUCTID" IS NOT NULL)
          GROUP BY  "HISTORY.CREATEDBY.EMPLOYEEID";

PO_CHANGE_CNT = SELECT COUNT(*) AS CHANGE_CNT,
"HISTORY.CHANGEDBY.EMPLOYEEID" AS EID
           FROM "PO.Header"
             WHERE PURCHASEORDERID IN (
             SELECT "HEADER.PURCHASEORDERID" as
"PURCHASEORDERID" FROM "PO.Item"
          WHERE "PRODUCT.PRODUCTID" IS NOT NULL)
           GROUP BY  "HISTORY.CHANGEDBY.EMPLOYEEID";

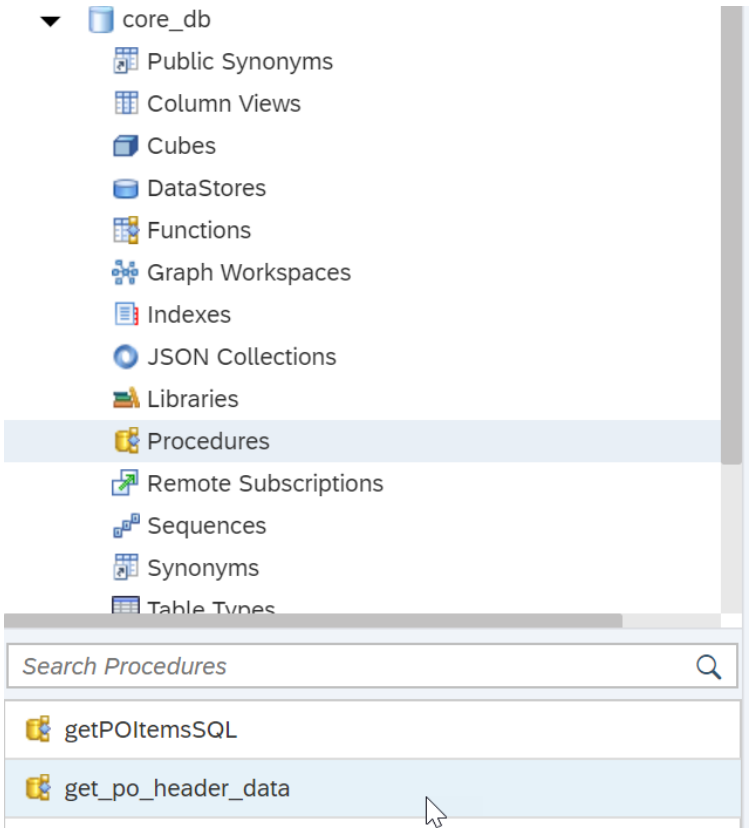
EX_TOP_3_EMP_PO_COMBINED_CNT =
   SELECT emp.LOGINNAME, crcnt.CREATE_CNT,
chcnt.CHANGE_CNT,
         crcnt.CREATE_CNT + chcnt.CHANGE_CNT AS
COMBINED_CNT
       FROM "MD.Employees" as emp
        LEFT OUTER JOIN :PO_CREATE_CNT AS crcnt
            ON emp.EMPLOYEEID = crcnt.EID
        LEFT OUTER JOIN :PO_CHANGE_CNT AS chcnt
            ON emp.EMPLOYEEID = chcnt.EID
           ORDER BY COMBINED_CNT DESC LIMIT 3;

END
```

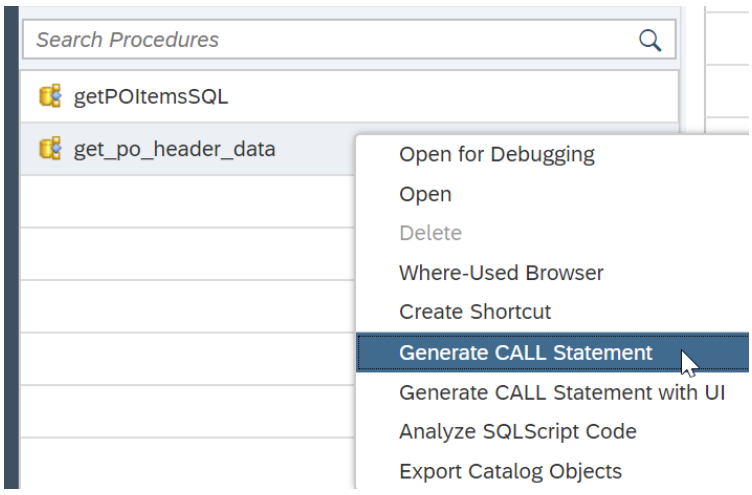
6. Save the procedure.	
7. Select the **db** module and choose **Build->Build**.	
8. If your build was completed successfully, you should see a success message in the console below.	
9. If not, then click on the **Problems** button on the right side. This will show the error messages.	

10. Switch over to the **Database Explorer** perspective.

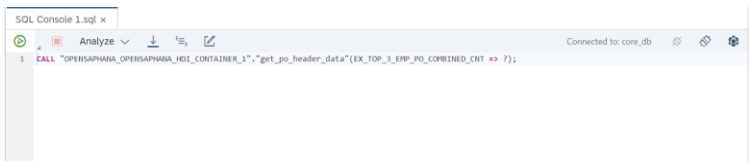
    Browse your container and find your new procedure in the **Procedures** folder.

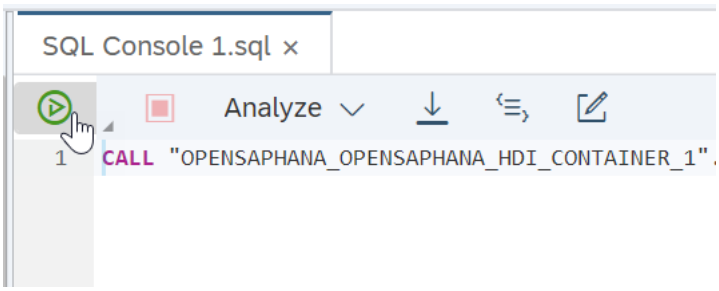
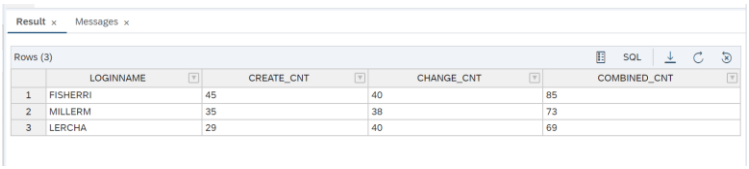
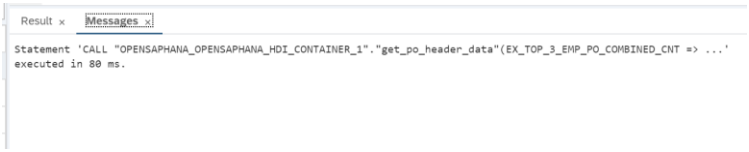


11. Right-click on the procedure and choose **Generate CALL Statement**.



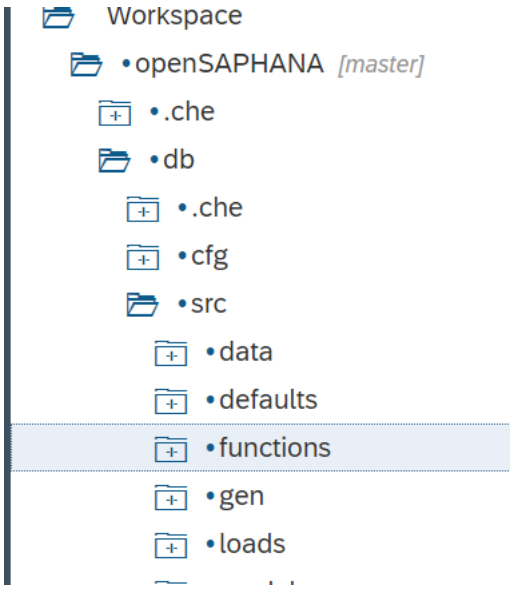
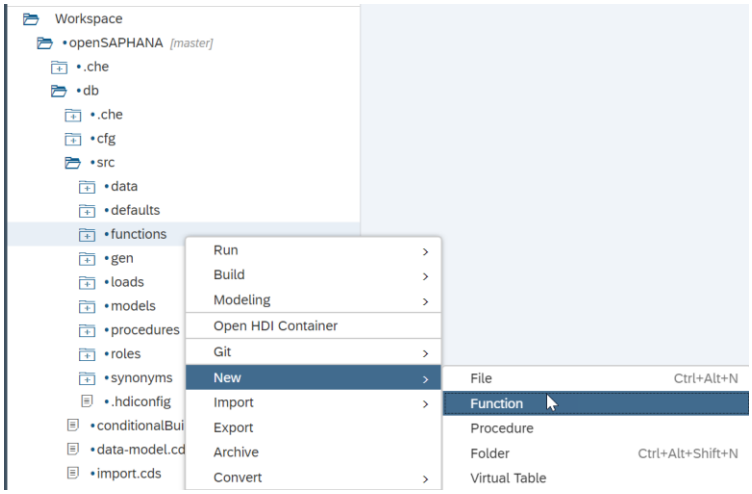
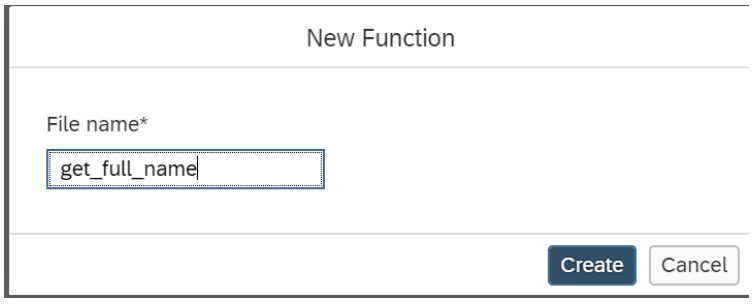
12. A new SQL tab will be opened with the CALL statement inserted. Notice the technical schema name of your container is inserted here.



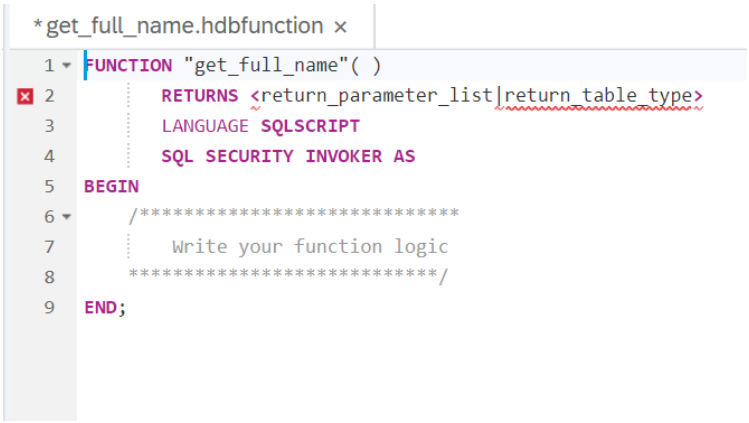
13. Click the **Run** button.	
14. The results are then shown in the table below.	
15. Note the execution time.	

Exercise 2.13.2: Create a Scalar User Defined Function

In this exercise, we are creating a scalar UDF for generating a full name from the last, first and middle name of the employee.

Explanation	Screenshot
1. Switch back to the SAP Web IDE for SAP HANA and look at the **/db/src/** folder. If you do not already have a **functions** folder, then right-click on the **src** folder and choose **New->Folder**. Enter the name of the folder as **functions** and choose **OK**.	
2. Right-click on the functions folder and choose **New -> Function.**	
3. Enter the name of the function as **get_full_name** and click **Create**.	

4. The function editor will be opened.



5. Enter the code as shown.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_full_name.sql

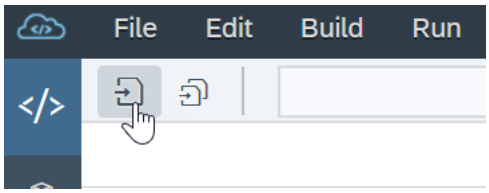
```
FUNCTION "get_full_name" (
            IN im_firstname NVARCHAR(40) ,
            IN im_middlename NVARCHAR(40),
            IN im_lastname NVARCHAR(40),
            IN im_employeeid NVARCHAR(10)
DEFAULT '' )
 RETURNS ex_fullname NVARCHAR(265)
 LANGUAGE SQLSCRIPT
 SQL SECURITY INVOKER AS
BEGIN

if :im_middlename IS NULL THEN
   ex_fullname = :im_lastname || ', ' ||
:im_firstname;
 ELSE
   ex_fullname = :im_lastname || ', ' ||
                 :im_firstname || ' ' ||
:im_middlename;
 END IF;

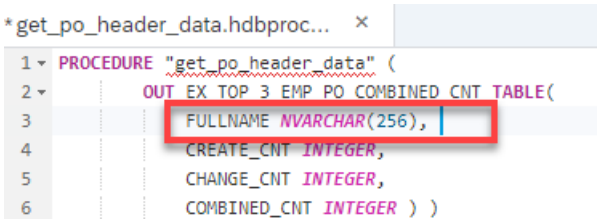
 IF :im_employeeid <> '' then
   ex_fullname = :ex_fullname || '(' ||
                 :im_employeeid || ')';
 END IF;

END;
```

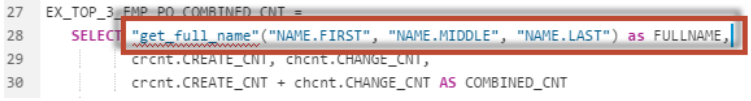
6. Click **Save**.



7. Return to your procedure called **get_po_header_data** and modify it. Start by renaming the **LOGINNAME** column of the output table to **FULLNAME**. Also change the output length to **256**. This is needed to match later on which the anticipated output structure.



8. Change the last SELECT statement. Remove the **LOGINNAME** column from the field list and replace it with a call to the scalar function that you created earlier. Make sure to pass the **NAME.FIRST NAME.MIDDLE** and **NAME.LAST** name columns to the scalar function call.



9. The completed code should look very similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_header_data.sql

```
PROCEDURE "get_po_header_data" (
        OUT EX_TOP_3_EMP_PO_COMBINED_CNT
TABLE(
            FULLNAME NVARCHAR(256),
                  CREATE_CNT INTEGER,
                  CHANGE_CNT INTEGER,
                  COMBINED_CNT INTEGER ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

PO_CREATE_CNT = SELECT COUNT(*) AS CREATE_CNT,
"HISTORY.CREATEDBY.EMPLOYEEID" AS EID
         FROM "PO.Header"
            WHERE PURCHASEORDERID IN (
        SELECT PURCHASEORDERID FROM "PO.Item"
             WHERE "PRODUCT.PRODUCTID" IS NOT
NULL)
          GROUP BY
"HISTORY.CREATEDBY.EMPLOYEEID";

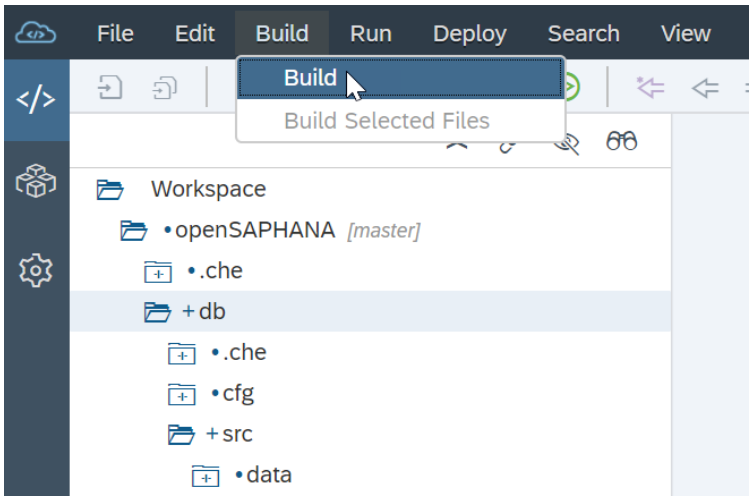
PO_CHANGE_CNT = SELECT COUNT(*) AS CHANGE_CNT,
"HISTORY.CHANGEDBY.EMPLOYEEID" AS EID
           FROM "PO.Header"
             WHERE PURCHASEORDERID IN (
             SELECT PURCHASEORDERID  FROM
"PO.Item"
          WHERE "PRODUCT.PRODUCTID" IS NOT
NULL)
```

```
            GROUP BY
"HISTORY.CHANGEDBY.EMPLOYEEID";

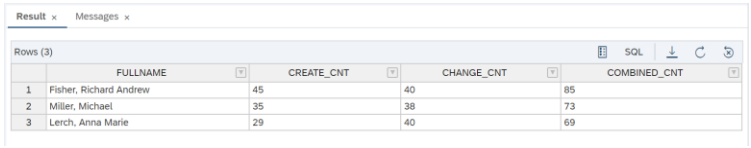
EX_TOP_3_EMP_PO_COMBINED_CNT =
   SELECT "get_full_name"("NAME.FIRST",
"NAME.MIDDLE", "NAME.LAST") as FULLNAME,
          crcnt.CREATE_CNT, chcnt.CHANGE_CNT,
          crcnt.CREATE_CNT + chcnt.CHANGE_CNT
AS COMBINED_CNT
       FROM "MD.Employees" as emp
        LEFT OUTER JOIN :PO_CREATE_CNT AS
crcnt
            ON emp.EMPLOYEEID = crcnt.EID
        LEFT OUTER JOIN :PO_CHANGE_CNT AS
chcnt
            ON emp.EMPLOYEEID = chcnt.EID
           ORDER BY COMBINED_CNT DESC LIMIT 3;

END
```

10. **Save** the procedure and perform a **Build** on **db**.



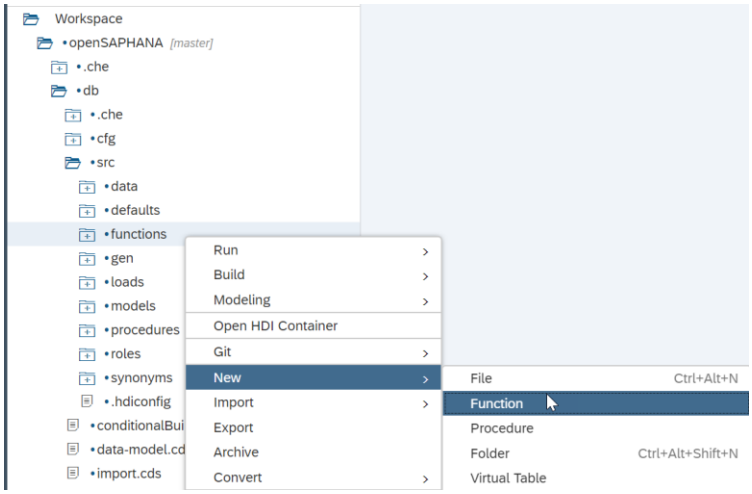
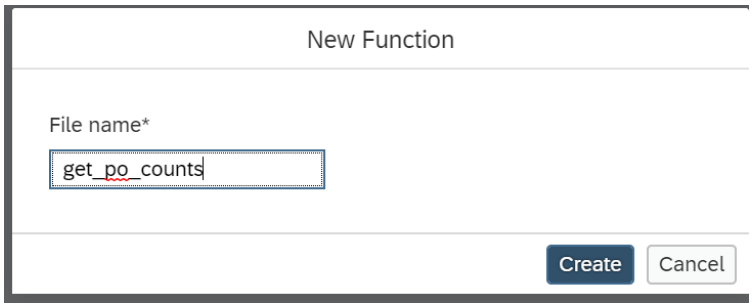
11. Return to the **Database Explorer** and once again execute the procedure. Notice the FULLNAME column, it shows the results of the scalar UDF logic.

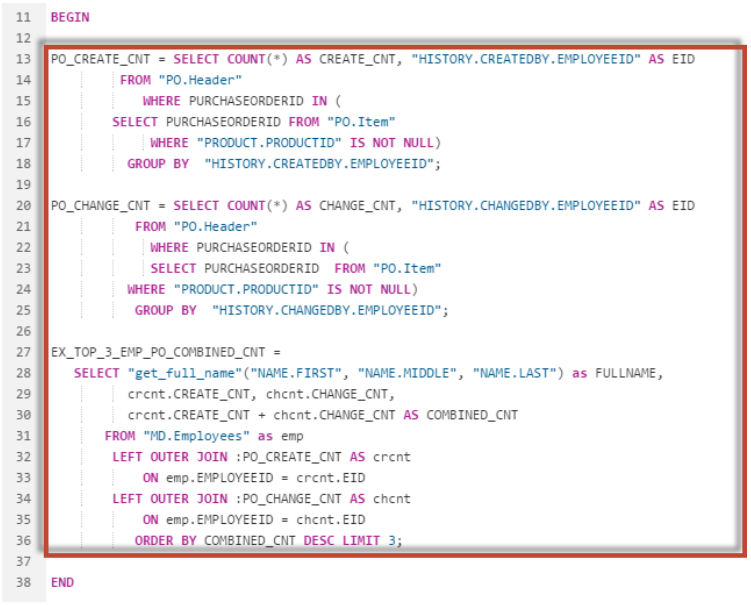
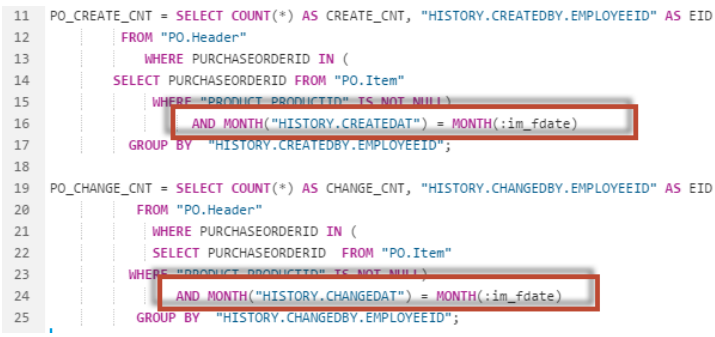
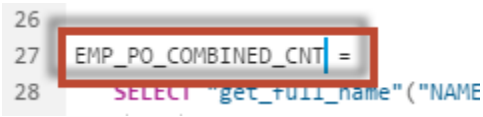
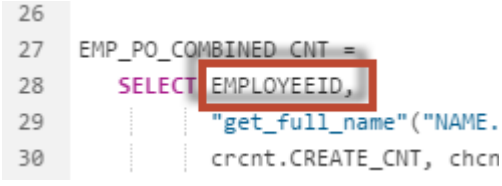


	FULLNAME	CREATE_CNT	CHANGE_CNT	COMBINED_CNT
1	Fisher, Richard Andrew	45	40	85
2	Miller, Michael	35	38	73
3	Lerch, Anna Marie	29	40	69

Exercise 2.13.3: Create a Table User Defined Function

There are applications and scenarios where you need a table function instead of a procedure to leverage the advantage of direct selects on the output i.e. filtering, sorting and grouping. In the following exercise, we show you how you can easily transform a procedure to a table function.

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA and right click on the **functions** folder and choose **New->Function**.	
2. Enter the name of the function as **get_po_counts** and click **Create.**	
3. Enter the basic shell of the function as shown here. Please note the scalar input parameter we will use later on for filtering. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_counts.sql	``` FUNCTION "get_po_counts"(im_fdate DATE) RETURNS TABLE (EMPLOYEEID NVARCHAR(10), FULLNAME NVARCHAR(256), CREATE_CNT INTEGER, CHANGE_CNT INTEGER, COMBINED_CNT INTEGER) LANGUAGE SQLSCRIPT SQL SECURITY INVOKER AS BEGIN END; ```

4. Copy the logic from the procedure *get_po_header_data* into the body of the function. Make sure to only copy the code between the BEGIN and END statements.	
5. Add to the **WHERE** clauses in the first two **SELECT** statements for filtering by month. Month is captured from the input parameter **im_fdate**.	
6. In the 3rd SELECT statement, change the name of the intermediate table variable to **EMP_PO_COMBINED_CNT** to match the variable name to the semantics of the query.	
7. Also add the **EMPLOYEEID** column to the field list.	

8. Remove the **LIMIT** clause at the end.

```
27  EMP_PO_COMBINED_CNT =
28    SELECT EMPLOYEEID,
29        "get_full_name"("NAME.FIRST", "NAME.MIDDLE", "NAME.LAST") as FULLNAME,
30        crcnt.CREATE_CNT, chcnt.CHANGE_CNT,
31        crcnt.CREATE_CNT + chcnt.CHANGE_CNT AS COMBINED_CNT
32      FROM "MD.Employees" as emp
33       LEFT OUTER JOIN :PO_CREATE_CNT AS crcnt
34          ON emp.EMPLOYEEID = crcnt.EID
35       LEFT OUTER JOIN :PO_CHANGE_CNT AS chcnt
36          ON emp.EMPLOYEEID = chcnt.EID
37        ORDER BY COMBINED_CNT DESC ;
```

9. Finally, add a **RETURN** statement at the end to mark the to be returned result set of the function

```
35          LEFT OUTER JOIN :PO_CHANGE_CNT AS chcnt
36             ON emp.EMPLOYEEID = chcnt.EID
37           ORDER BY COMBINED_CNT DESC ;
38
39  RETURN :EMP_PO_COMBINED_CNT;
40
41  END;
```

10. The completed code should look very similar to the following.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_counts2.sql

```
FUNCTION "get_po_counts"( im_fdate DATE )
    RETURNS TABLE (EMPLOYEEID NVARCHAR(10),
                           FULLNAME
NVARCHAR(256),
                           CREATE_CNT INTEGER,
                           CHANGE_CNT INTEGER,
                           COMBINED_CNT INTEGER)
      LANGUAGE SQLSCRIPT
      SQL SECURITY INVOKER AS
BEGIN


PO_CREATE_CNT = SELECT COUNT(*) AS CREATE_CNT,
"HISTORY.CREATEDBY.EMPLOYEEID" AS EID
         FROM "PO.Header"
            WHERE PURCHASEORDERID IN (
                           SELECT
PURCHASEORDERID FROM "PO.Item"
                                        WHERE
"PRODUCT.PRODUCTID" IS NOT NULL)
              AND MONTH("HISTORY.CREATEDAT") =
MONTH(:im_fdate)
          GROUP BY
"HISTORY.CREATEDBY.EMPLOYEEID";

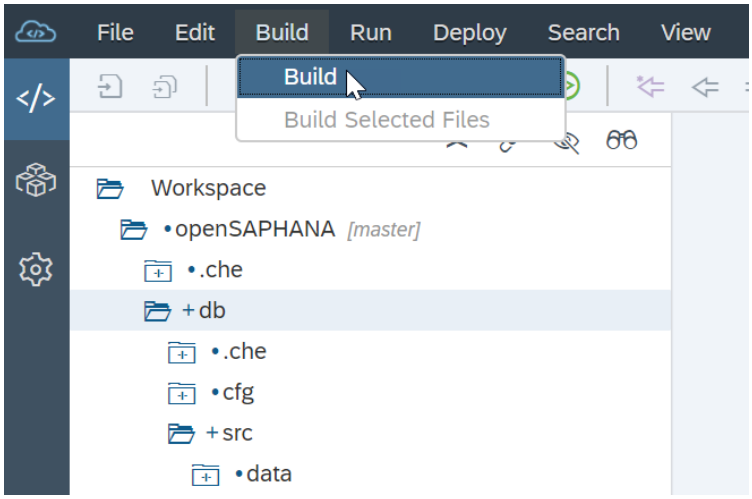
PO_CHANGE_CNT = SELECT COUNT(*) AS CHANGE_CNT,
"HISTORY.CHANGEDBY.EMPLOYEEID" AS EID
           FROM "PO.Header"
             WHERE PURCHASEORDERID IN (
                           SELECT
PURCHASEORDERID  FROM "PO.Item"
                                         WHERE
"PRODUCT.PRODUCTID" IS NOT NULL)
              AND MONTH("HISTORY.CHANGEDAT") =
MONTH(:im_fdate)
           GROUP BY
"HISTORY.CHANGEDBY.EMPLOYEEID";
```

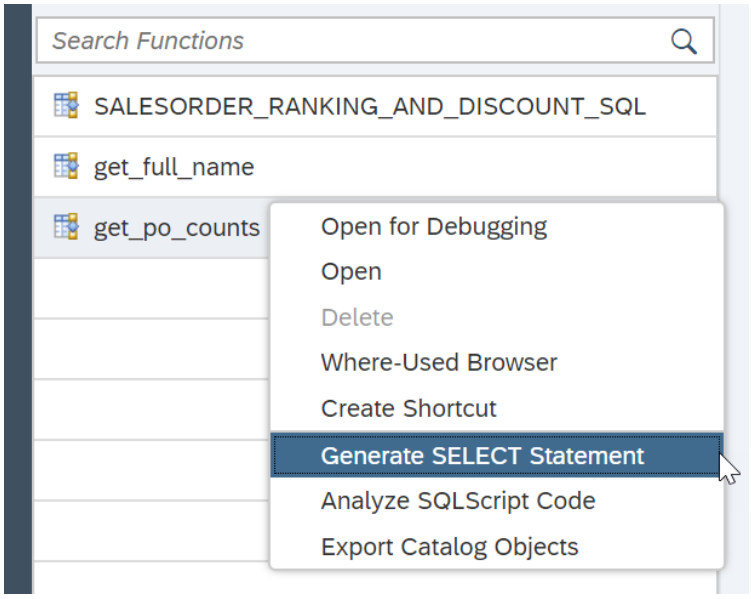
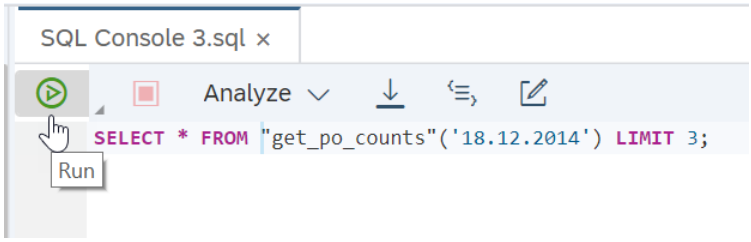
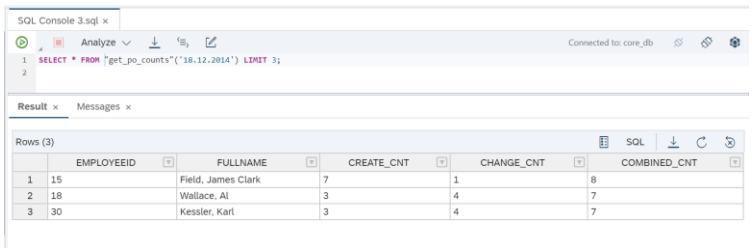
```
EMP_PO_COMBINED_CNT =
          SELECT EMPLOYEEID,
                 "get_full_name"("NAME.FIRST",
"NAME.MIDDLE", "NAME.LAST") as FULLNAME,
                 crcnt.CREATE_CNT,
chcnt.CHANGE_CNT,
                 crcnt.CREATE_CNT +
chcnt.CHANGE_CNT AS COMBINED_CNT
                   FROM "MD.Employees" as emp
                     LEFT OUTER JOIN
:PO_CREATE_CNT AS crcnt
                                 ON emp.EMPLOYEEID
= crcnt.EID
                     LEFT OUTER JOIN
:PO_CHANGE_CNT AS chcnt
                                 ON emp.EMPLOYEEID
= chcnt.EID
                     ORDER BY COMBINED_CNT DESC
;

RETURN :EMP_PO_COMBINED_CNT;

END;
```

11. Use what you have learned and **Save** the function, then perform a **Build**.

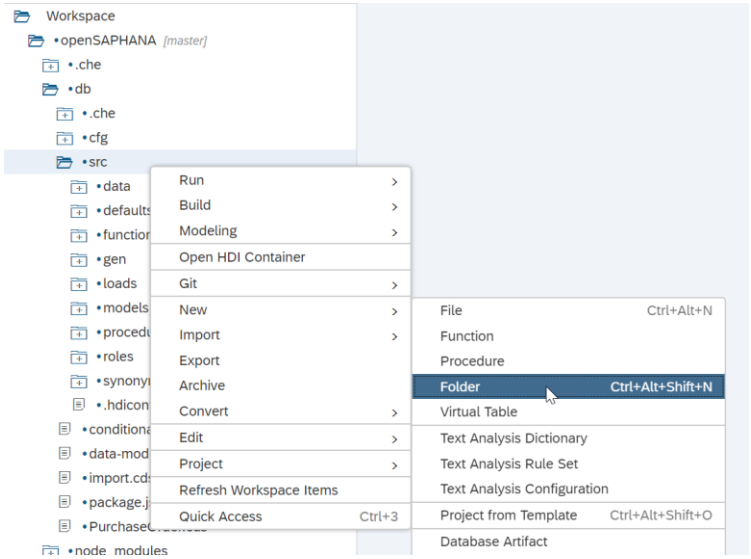


12. Return to the Database Explorer. Browse to the new function and right-click on it and choose **Generate SELECT Statement**.	
13. Modify the SELECT statement for your table function. Pass the date **'18.12.2014'** as the input parameter. Also add **LIMIT 3** to the end. The technical container schema name can be omitted.	`SELECT * FROM "get_po_counts"('18.12.2014') LIMIT 3;`
14. Click **Run**.	
15. The results of your table function are then shown.	

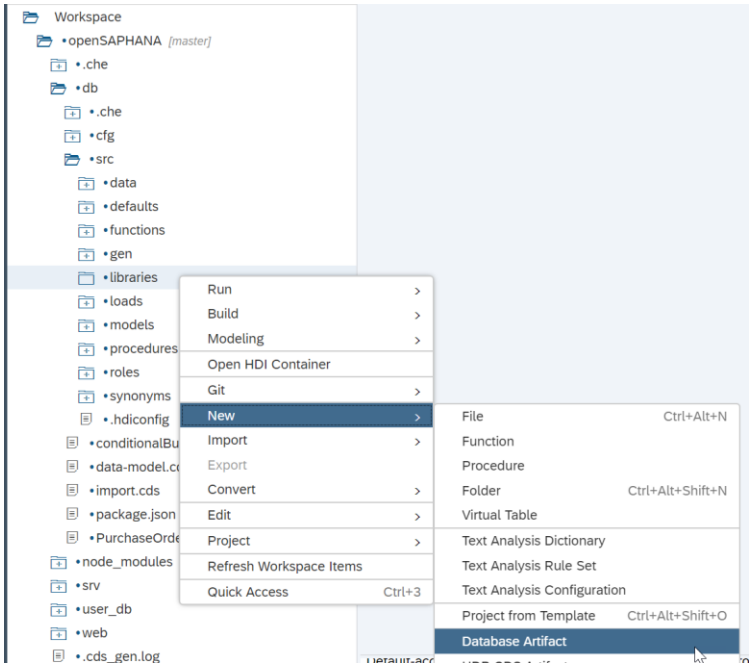
Exercise 2.13.4: Libraries

Exercise 2.13.4.1: Creating User Defined Libraries

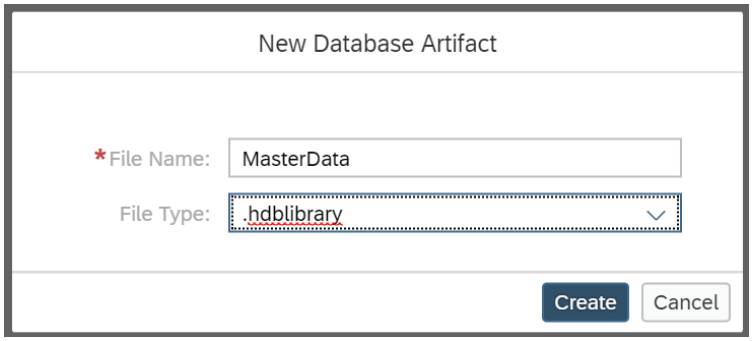
In this exercise, you will learn how to create SQLScript User Defined Libraries. Libraries are designed to group related variables, functions, and procedures together within a common object. These library members can either be exposed using the PULBIC keyword, or set to only be consumable from within the library itself using the PRIVATE keyword.

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA and right click on the /**db/src** folder and choose **New->Folder**.	
2. Enter the name of the folder as **libraries** and click **OK**.	

3. Right-click on the **libraries** folder and choose **New-> Database Artifact**



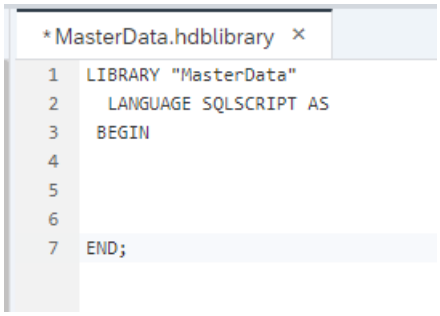
4. Name the file as **MasterData** and the type as .**hdblibrary** and click **Create**



5. Enter the shell of the library as shown here.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/MasterData.sql



6. First, define a public variable library member called **rowcount** with type int.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/MasterData.sql

*MasterData.hdblibrary ×

```
1  LIBRARY "MasterData"
2    LANGUAGE SQLSCRIPT AS
3   BEGIN
4
5     public variable rowcount int;
6
7
8  END;
```

7. Next, define a public function library member as shown here.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/MasterData.sql

*MasterData.hdblibrary ×

```
1  LIBRARY "MasterData"
2    LANGUAGE SQLSCRIPT AS
3   BEGIN
4
5     public variable rowcount int;
6
7    public function "employee_exists"(
8                in im_employee_id nvarchar(10) )
9                returns res boolean as
10 begin
11     declare lv_count int;
12     select count(*) into lv_count from "MD.Employees"
13                                  where employeeid = im_employee_id;
14     if  lv_count > 0 then
15      res = true;
16     else
17      res = false;
18     end if;
19   end;
20
21
22 END;
23
```

8. Finally, define two public procedure library members, one called **get_employee_data** and the other **get_business_partner_data**.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/MasterData.sql

*MasterData.hdblibrary ×

```
1  LIBRARY "MasterData"
2    LANGUAGE SQLSCRIPT AS
3   BEGIN
4
5     public variable rowcount int;
6
7     public function "employee_exists"(
8                 in im_employee_id nvarchar(10) )
9                 returns res boolean as
10    begin
11       declare lv_count int;
12       select count(*) into lv_count from "MD.Employees"
13                                    where employeeid = im_employee_id;
14       if  lv_count > 0 then
15        res = true;
16       else
17        res = false;
18       end if;
19    end;
20
21    public procedure "get_employee_data"(
22                out ex_employees "MD.Employees" )
23     LANGUAGE SQLSCRIPT
24     SQL SECURITY INVOKER
25     AS
26    begin
27     ex_employees = select * from "MD.Employees";
28     rowcount = record_count(:ex_employees);
29    end;
30
31    public procedure "get_business_partner_data"(
32                out ex_businesspartners "MD.BusinessPartner" )
33     LANGUAGE SQLSCRIPT
34     SQL SECURITY INVOKER
35      AS
36    begin
37     ex_businesspartners = select * from "MD.BusinessPartner";
38     rowcount = record_count(:ex_businesspartners);
39    end;
40
41  END;
```

9. The completed code should look very similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/MasterData.sql

```
LIBRARY "MasterData"
  LANGUAGE SQLSCRIPT AS
 BEGIN

   public variable rowcount int;

   public function "employee_exists"(
                in im_employee_id nvarchar(10)
)
                returns res boolean as
   begin
       declare lv_count int;
       select count(*) into lv_count from
"MD.Employees"
                                  where employeeid
= im_employee_id;
       if  lv_count > 0 then
```

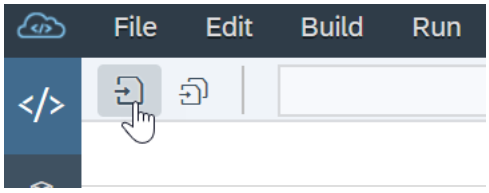
```
        res = true;
       else
        res = false;
       end if;
   end;

   public procedure "get_employee_data"(
                  out ex_employees
"MD.Employees" )
      LANGUAGE SQLSCRIPT
      SQL SECURITY INVOKER
      AS
   begin
     ex_employees = select * from
"MD.Employees";
     rowcount = record_count(:ex_employees);
   end;

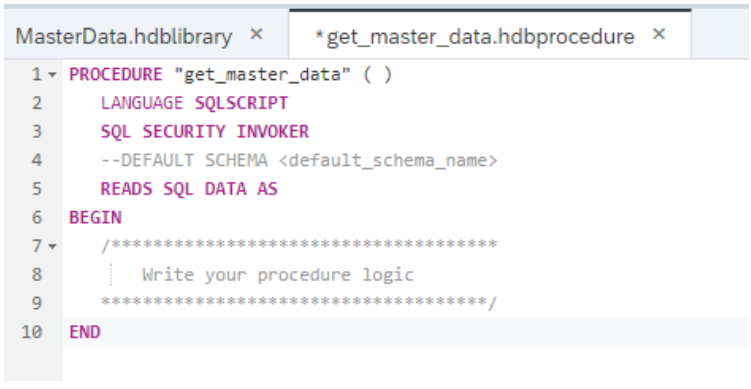
   public procedure
"get_business_partner_data"(
                  out ex_businesspartners
"MD.BusinessPartner" )
      LANGUAGE SQLSCRIPT
      SQL SECURITY INVOKER
       AS
   begin
     ex_businesspartners = select * from
"MD.BusinessPartner";
     rowcount =
record_count(:ex_businesspartners);
   end;

END;
```

10. Save your work.



11. Use what you have learned and create a new procedure in the **procedures** folder called **get_master_data**.



12. Enter the shell of the procedure as shown. In order to use a library in our prodedure, we just declare its use in the **BEGIN** statement using the **USING** clause.

    If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_master_data.sql

*get_master_data.hdbprocedure ×

```
1  PROCEDURE "get_master_data" (
2                  out ex_emp_exists boolean,
3                  out ex_employees "MD.Employees",
4                  out ex_emp_count int,
5                  out ex_business_partners "MD.BusinessPartner",
6                  out ex_bp_count int)
7     LANGUAGE SQLSCRIPT
8     SQL SECURITY INVOKER
9     READS SQL DATA AS
10 BEGIN USING "MasterData" as MData;
11
12
13 END;
14
```

13. Enter the code as shown. In the body of the procedure we are simply calling the library function **employee_exists**, then calling the get procedures for employees and businesspartners as well as getting the rowcounts for each.

    If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_master_data.sql

get_master_data.hdbprocedure ×

```
1  PROCEDURE "get_master_data" (
2                  out ex_emp_exists boolean,
3                  out ex_employees "MD.Employees",
4                  out ex_emp_count int,
5                  out ex_business_partners "MD.BusinessPartner",
6                  out ex_bp_count int)
7     LANGUAGE SQLSCRIPT
8     SQL SECURITY INVOKER
9     READS SQL DATA AS
10 BEGIN USING "MasterData" as MData;
11
12    ex_emp_exists  = MData:"employee_exists"('199999');
13
14    call MData:"get_employee_data"( ex_employees );
15    ex_emp_count = MData:rowcount;
16
17    call MData:"get_business_partner_data"( ex_business_partners );
18    ex_bp_count = MData:rowcount;
19
20 END;
```

14. The completed code should look very similar to this.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_master_data.sql

```
PROCEDURE "get_master_data" (
                  out ex_emp_exists boolean,
                  out ex_employees
"MD.Employees",
                  out ex_emp_count int,
                  out ex_business_partners
"MD.BusinessPartner",
                  out ex_bp_count int)
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   READS SQL DATA AS
BEGIN USING "MasterData" as MData;

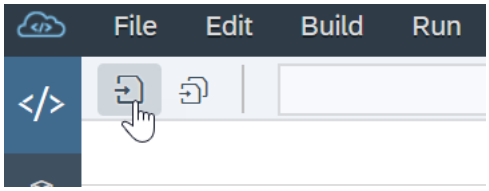
 ex_emp_exists  =
MData:"employee_exists"('199999');
```

```
 call MData:"get_employee_data"( ex_employees
);
 ex_emp_count = MData:rowcount;

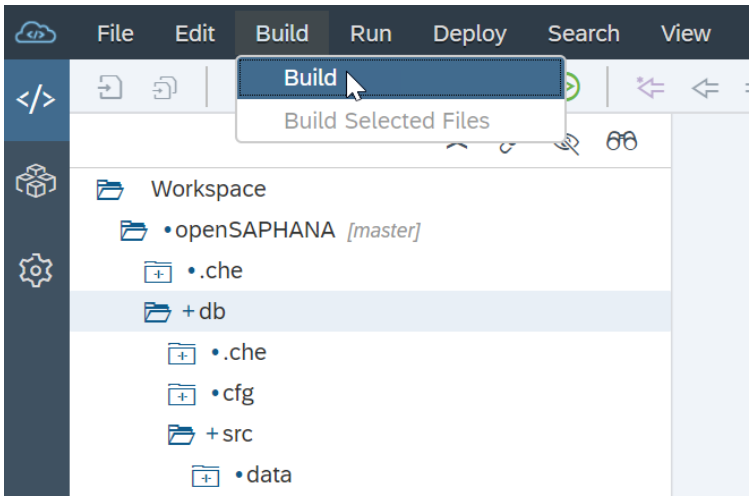
 call MData:"get_business_partner_data"(
ex_business_partners );
 ex_bp_count = MData:rowcount;

END;
```

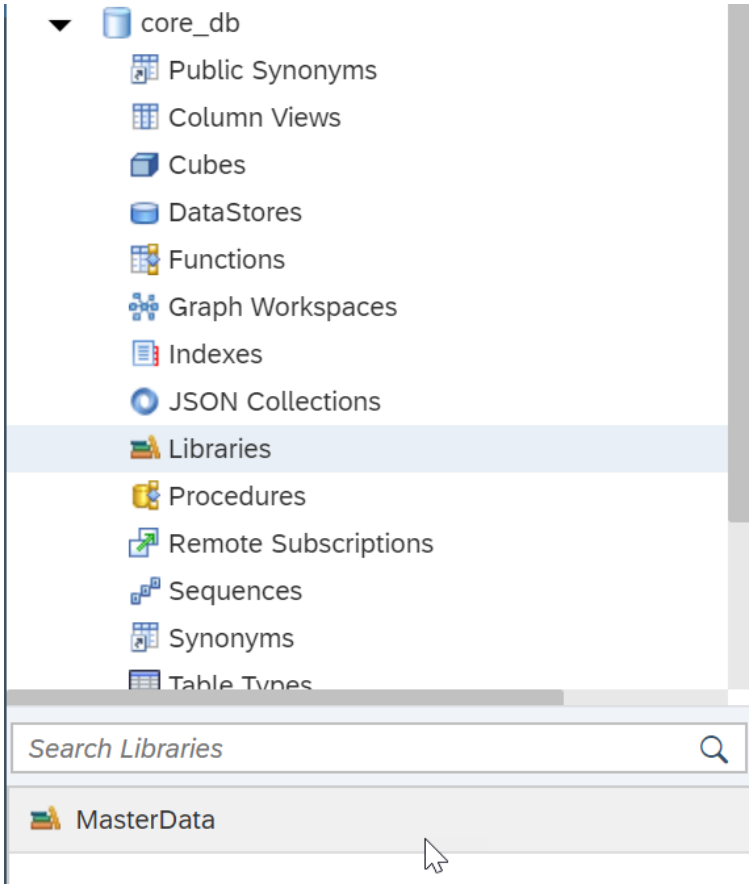
15. Save your work.



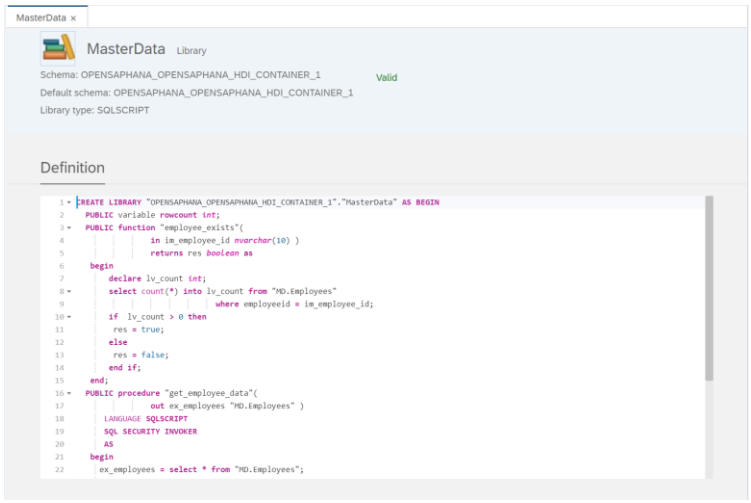
16. Use what you have learned perform a **Build** on **db**.



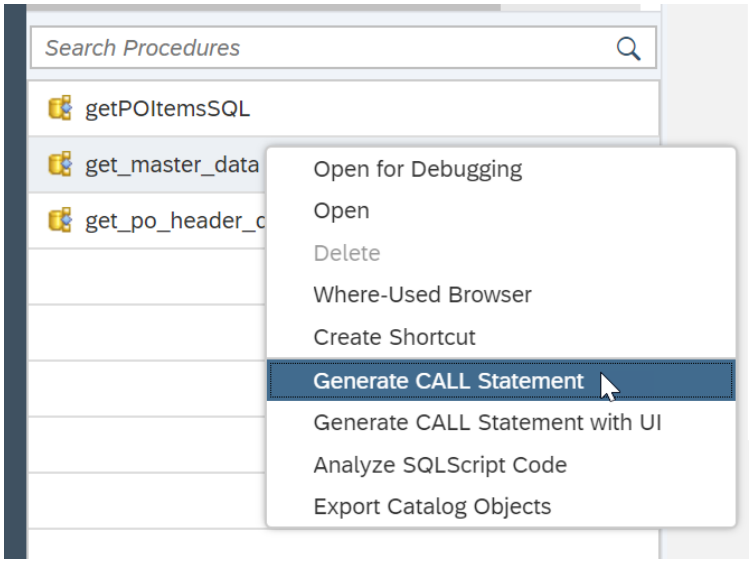
17. Return to the **Database Explorer** and click on the **Libraries** folder. Then click the **MasterData** library.



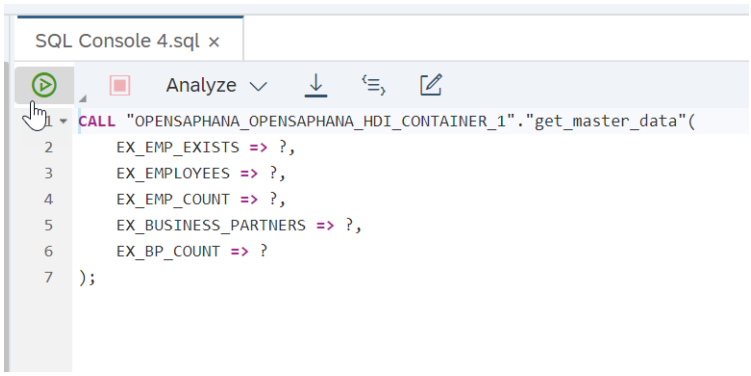
18. You should then see the library definition on the right hand side.



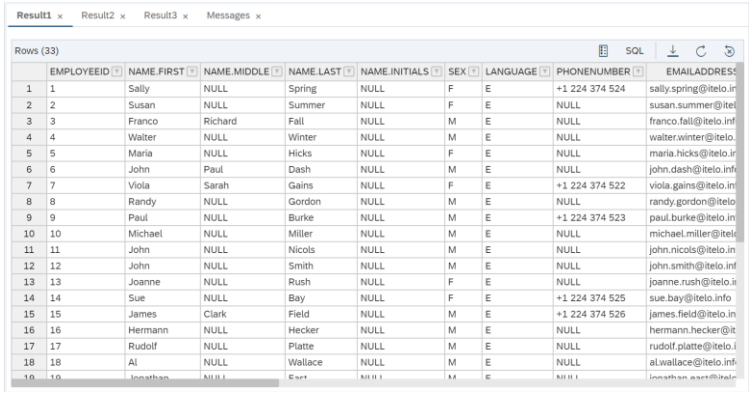
19. Next click the **Procedures** folder and then right-click on the **get_master_data** procedure and choose **Generate CALL Statement**.

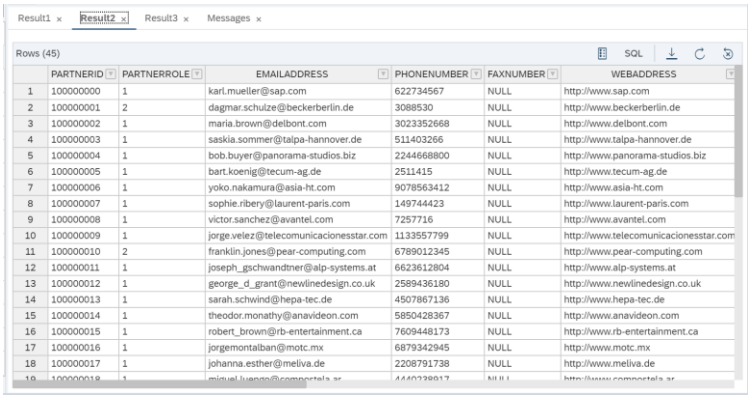
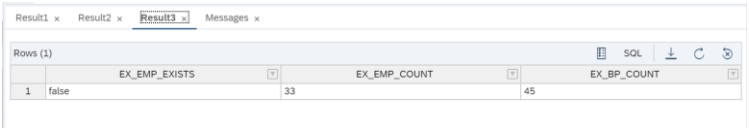


20. Click **Run.**



21. Inspect the **Results** tab. You should see the list of employees in Result1.



22. Click the **Result2** tab, and inspect the partner data.	
23. Click the **Result3** tab to inspect the scalar variable values.	

Result1 × Result2 × Result3 × Messages ×

Rows (45) SQL

	PARTNERID	PARTNERROLE	EMAILADDRESS	PHONENUMBER	FAXNUMBER	WEBADDRESS
1	100000000	1	karl.mueller@sap.com	622734567	NULL	http://www.sap.com
2	100000001	2	dagmar.schulze@beckerberlin.de	3088530	NULL	http://www.beckerberlin.de
3	100000002	1	maria.brown@delbont.com	3023352668	NULL	http://www.delbont.com
4	100000003	1	saskia.sommer@talpa-hannover.de	511403266	NULL	http://www.talpa-hannover.de
5	100000004	1	bob.buyer@panorama-studios.biz	2244668800	NULL	http://www.panorama-studios.biz
6	100000005	1	bart.koenig@tecum-ag.de	2511415	NULL	http://www.tecum-ag.de
7	100000006	1	yoko.nakamura@asia-ht.com	9078563412	NULL	http://www.asia-ht.com
8	100000007	1	sophie.ribery@laurent-paris.com	149744423	NULL	http://www.laurent-paris.com
9	100000008	1	victor.sanchez@avantel.com	7257716	NULL	http://www.avantel.com
10	100000009	1	jorge.velez@telecomunicacionesstar.com	1133557799	NULL	http://www.telecomunicacionesstar.com
11	100000010	2	franklin.jones@pear-computing.com	6789012345	NULL	http://www.pear-computing.com
12	100000011	1	joseph_gschwandtner@alp-systems.at	6623612804	NULL	http://www.alp-systems.at
13	100000012	1	george_d_grant@newlinedesign.co.uk	2589436180	NULL	http://www.newlinedesign.co.uk
14	100000013	1	sarah.schwind@hepa-tec.de	4507867136	NULL	http://www.hepa-tec.de
15	100000014	1	theodor.monathy@anavideon.com	5850428367	NULL	http://www.anavideon.com
16	100000015	1	robert_brown@rb-entertainment.ca	7609448173	NULL	http://www.rb-entertainment.ca
17	100000016	1	jorgemontalban@motc.mx	6879342945	NULL	http://www.motc.mx
18	100000017	1	johanna.esther@meliva.de	2208791738	NULL	http://www.meliva.de

Result1 × Result2 × Result3 × Messages ×

Rows (1) SQL

	EX_EMP_EXISTS	EX_EMP_COUNT	EX_BP_COUNT
1	false	33	45

Exercise 2.13.4.2: Leveraging Built-In Libraries

Built-In Libraries are libraries supplied by SAP for various functions. In this exercise, you will leverage the built-in library for splitting and formatting strings within SQLScript.

Explanation	Screenshot
1. Use what you have learned and create a new procedure in the **procedures** folder called **split_format_strings**. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql	*split_format_strings.hdbproce... × 1 PROCEDURE "split_format_strings" ( ) 2 LANGUAGE SQLSCRIPT 3 SQL SECURITY INVOKER 4 --DEFAULT SCHEMA <default_schema_name> 5 READS SQL DATA AS 6 BEGIN 7 /********************************* 8 Write your procedure logic 9 *********************************/ 10 END 11
2. Define two output parameters, one which will contain the string data, and one which will contain the tabular format data. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql	*split_format_strings.hdbproce... × 1 PROCEDURE "split_format_strings" (2 out ex_strout table(RESULT nvarchar(5000)), 3 out ex_tabout table(PRODUCTID nvarchar(10), PRICE decimal(15,2))) 4 LANGUAGE SQLSCRIPT 5 SQL SECURITY INVOKER 6 READS SQL DATA AS
3. There is a built-in library called **SQLSCRIPT_STRING** which contains various functions for splitting and formatting strings in SQLScript. In order to use these functions in our procedure, we need to declare its usage in the **BEGIN** statement as shown.	*split_format_strings.hdbproce... × 1 PROCEDURE "split_format_strings" (2 out ex_strout table(RESULT nvarchar(5000)), 3 out ex_tabout table(PRODUCTID nvarchar(10), PRICE decimal(15,2))) 4 LANGUAGE SQLSCRIPT 5 SQL SECURITY INVOKER 6 READS SQL DATA AS 7 BEGIN USING SQLSCRIPT_STRING AS LIB; 8

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql

4. Next we need to declare some variables which we will use later in the procedure. We also need to put some data into the local intermediate table called **lt_products**. Enter the code as shown.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql

split_format_strings.hdbproced... ×

```
1  PROCEDURE "split_format_strings" (
2          out ex_strout table(RESULT nvarchar(5000)),
3          out ex_tabout table(PRODUCTID nvarchar(10), PRICE decimal(15,2)) )
4    LANGUAGE SQLSCRIPT
5    SQL SECURITY INVOKER
6    READS SQL DATA AS
7  BEGIN USING SQLSCRIPT_STRING AS LIB;
8
9    declare lv_index int;
10   declare lv_productid string;
11   declare lv_price string;
12   declare lt_products TABLE like :ex_tabout;
13
14   lt_products.PRODUCTID[1] = 'HT-1000';
15   lt_products.PRICE[1] = '19.99';
16   lt_products.PRODUCTID[2] = 'HT-1001';
17   lt_products.PRICE[2] = '29.99';
18   lt_products.PRODUCTID[3] = 'HT-1002';
19   lt_products.PRICE[3] = '39.99';
```

5. Next, enter the line of code which will call the **FORMAT_TO_TABLE** function of the built-in library. This function will reformat the data in lt_products as comma-delimited strings and insert them into the output parameter ex_strout.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql

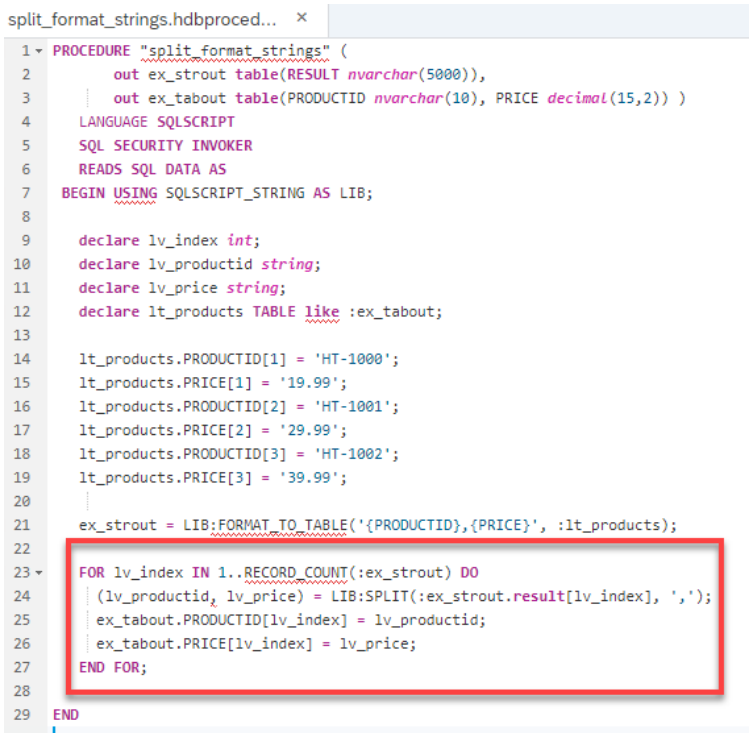
split_format_strings.hdbproced... ×

```
1  PROCEDURE "split_format_strings" (
2          out ex_strout table(RESULT nvarchar(5000)),
3          out ex_tabout table(PRODUCTID nvarchar(10), PRICE decimal(15,2)) )
4    LANGUAGE SQLSCRIPT
5    SQL SECURITY INVOKER
6    READS SQL DATA AS
7  BEGIN USING SQLSCRIPT_STRING AS LIB;
8
9    declare lv_index int;
10   declare lv_productid string;
11   declare lv_price string;
12   declare lt_products TABLE like :ex_tabout;
13
14   lt_products.PRODUCTID[1] = 'HT-1000';
15   lt_products.PRICE[1] = '19.99';
16   lt_products.PRODUCTID[2] = 'HT-1001';
17   lt_products.PRICE[2] = '29.99';
18   lt_products.PRODUCTID[3] = 'HT-1002';
19   lt_products.PRICE[3] = '39.99';
20
21   ex_strout = LIB:FORMAT_TO_TABLE('{PRODUCTID},{PRICE}', :lt_products);
22
```

6. Finally, enter the block of code highlighted here. This block of code will loop through the ex_strout table and leverage the **SPLIT** function to split each string back into their associated columns. We can then put those values into the ex_tabout output parameter using index based cell access.

   If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/split_format_strings.sql



7. The completed code should look very similar to the following.

```
PROCEDURE "split_format_strings" (
         out ex_strout table(RESULT
nvarchar(5000)),
       out ex_tabout table(PRODUCTID
nvarchar(10), PRICE decimal(15,2)) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   READS SQL DATA AS
 BEGIN USING SQLSCRIPT_STRING AS LIB;

   declare lv_index int;
   declare lv_productid string;
   declare lv_price string;
   declare lt_products TABLE like :ex_tabout;

   lt_products.PRODUCTID[1] = 'HT-1000';
   lt_products.PRICE[1] = '19.99';
   lt_products.PRODUCTID[2] = 'HT-1001';
   lt_products.PRICE[2] = '29.99';
   lt_products.PRODUCTID[3] = 'HT-1002';
   lt_products.PRICE[3] = '39.99';

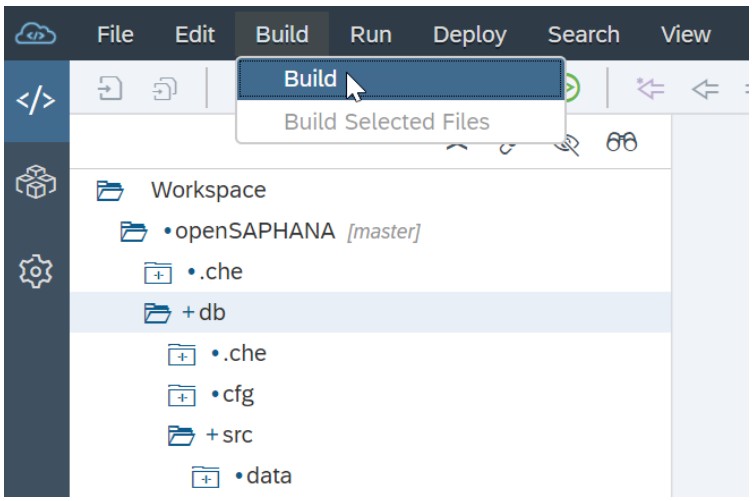
   ex_strout =
LIB:FORMAT_TO_TABLE('{PRODUCTID},{PRICE}',
:lt_products);

   FOR lv_index IN 1..RECORD_COUNT(:ex_strout)
DO
     (lv_productid, lv_price) =
LIB:SPLIT(:ex_strout.result[lv_index], ',');
```

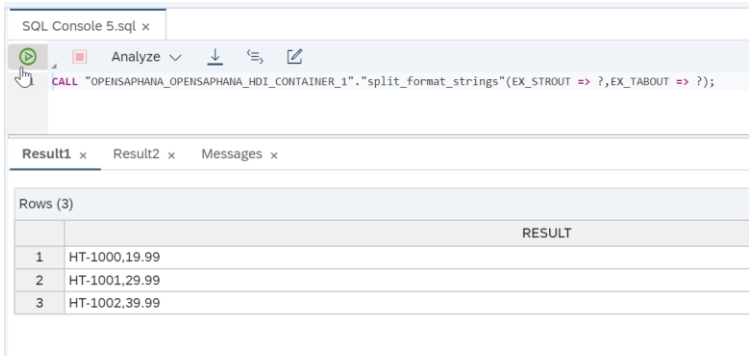
```
     ex_tabout.PRODUCTID[lv_index] =
lv_productid;
     ex_tabout.PRICE[lv_index] = lv_price;
   END FOR;

END
```

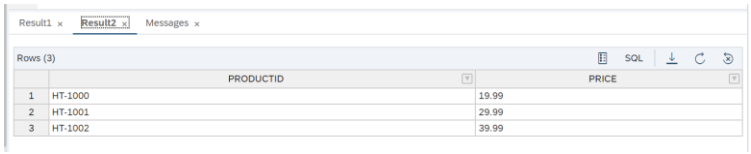
8. Use what you have learned and **Save** the procedure, then perform a **Build** on the module.



9. Return to the **Database Explorer**. navigate to the **split_format_strings** procedure and generate a call statement, then click **Run.** The Result1 tab should contain the data as a comma-delimited string.

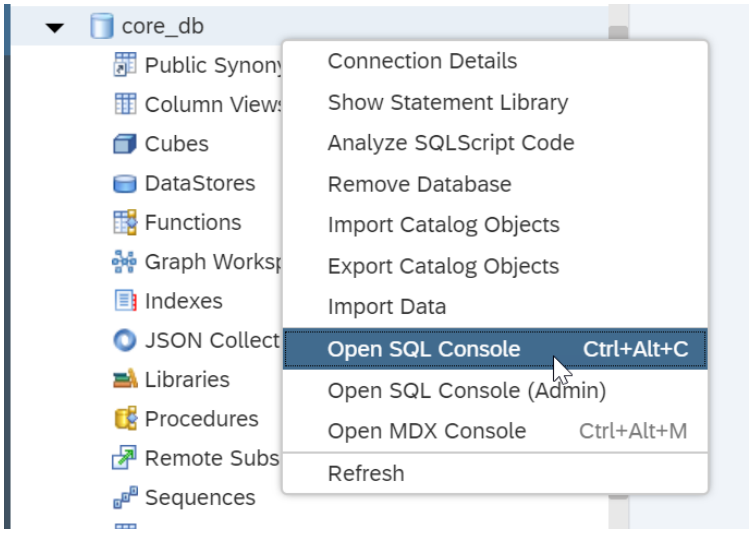
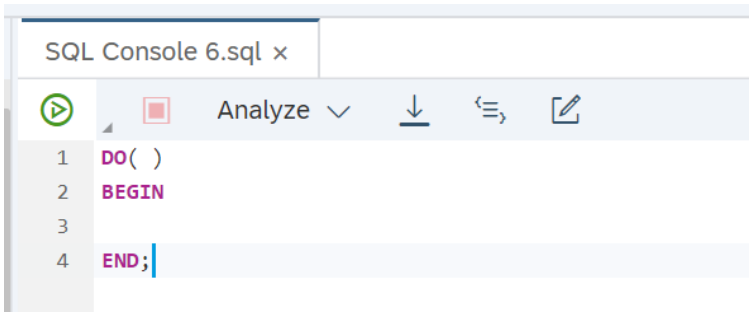


10. Click on the **Result2** tab. The Result2 tab should contain the same data as tabular data with clearly defined columns.

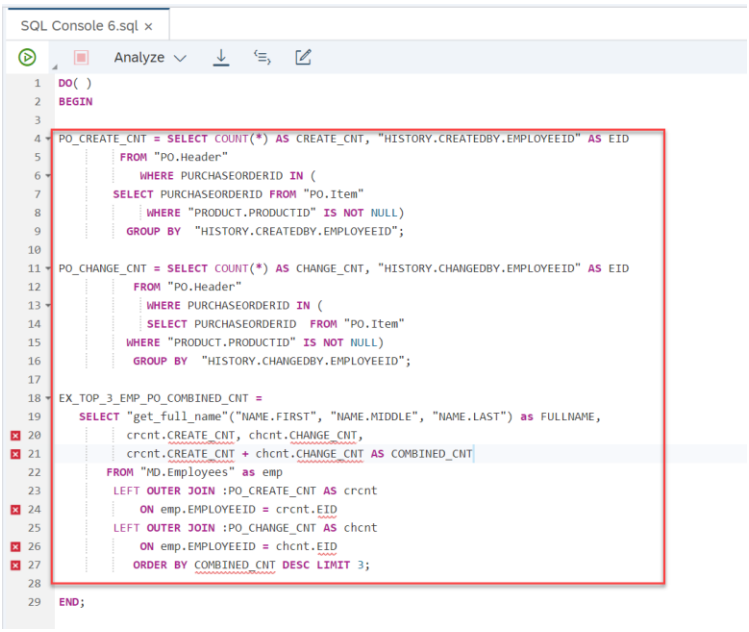


Exercise 2.13.5: Anonymous Blocks

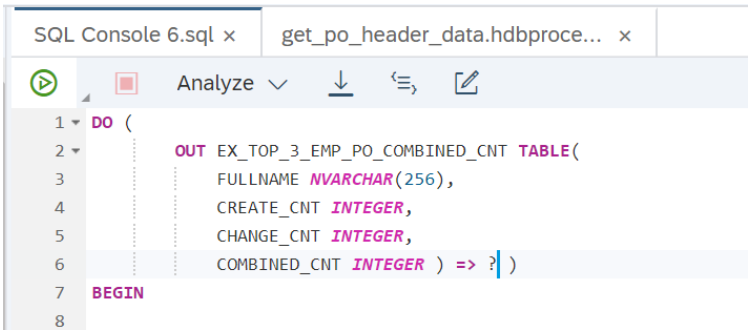
In this exercise, we will show you how you can invoke SQLScript logic without the need to create a persistent logic container such as a procedure or function. Instead we will use so called anonymous blocks which is a fast and easy-to-use way for playing around and modifying SQLScript logic.

Explanation	Screenshot
1. From the Database Explorer, right-click on your container and choose **Open SQL Console** button.	
2. To have an anonymous block you need a “do begin …end”.	

3. Copy the logic from the procedure **get_po_header_data** into the body. Make sure to only copy the code between the **BEGIN** and **END** statements.



4. Copy the signature from the procedure **get_po_header_data** into the signature part of the anonymous block. Ensure the parameter is assigned to a query parameter placeholder (?) as shown.



5. The complete code should look very similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/anonymous.sql

```
DO ( OUT EX_TOP_3_EMP_PO_COMBINED_CNT TABLE(
            FULLNAME NVARCHAR(256),
                  CREATE_CNT INTEGER,
                  CHANGE_CNT INTEGER,
                  COMBINED_CNT INTEGER ) => ?)
 BEGIN

 PO_CREATE_CNT = SELECT COUNT(*) AS
CREATE_CNT, "HISTORY.CREATEDBY.EMPLOYEEID" AS
EID
         FROM "PO.Header"
            WHERE PURCHASEORDERID IN (
        SELECT PURCHASEORDERID FROM "PO.Item"
             WHERE "PRODUCT.PRODUCTID" IS NOT
NULL) GROUP BY
"HISTORY.CREATEDBY.EMPLOYEEID";

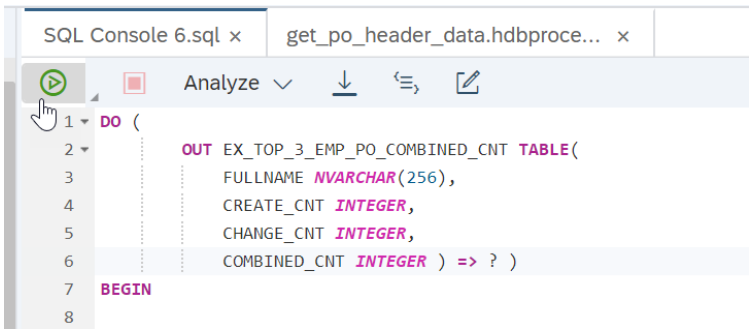
PO_CHANGE_CNT = SELECT COUNT(*) AS CHANGE_CNT,
"HISTORY.CHANGEDBY.EMPLOYEEID" AS EID
           FROM "PO.Header"
             WHERE PURCHASEORDERID IN (
```

```
        SELECT PURCHASEORDERID FROM "PO.Item"
          WHERE "PRODUCT.PRODUCTID" IS NOT
NULL) GROUP BY
"HISTORY.CHANGEDBY.EMPLOYEEID";

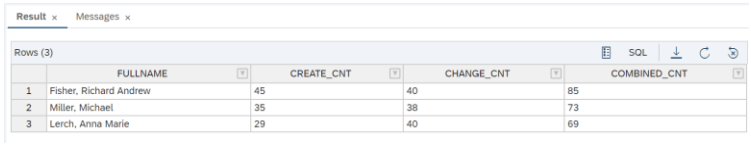
EX_TOP_3_EMP_PO_COMBINED_CNT =
   SELECT "get_full_name"("NAME.FIRST",
"NAME.MIDDLE", "NAME.LAST") as FULLNAME,
          crcnt.CREATE_CNT, chcnt.CHANGE_CNT,
          crcnt.CREATE_CNT + chcnt.CHANGE_CNT
AS COMBINED_CNT
       FROM "MD.Employees" as emp
      LEFT OUTER JOIN :PO_CREATE_CNT AS crcnt
            ON emp.EMPLOYEEID = crcnt.EID
      LEFT OUTER JOIN :PO_CHANGE_CNT AS chcnt
            ON emp.EMPLOYEEID = chcnt.EID
           ORDER BY COMBINED_CNT DESC LIMIT 3;

 END;
```

6. Click **Run**.



7. You will notice that the SQLScript code is executed and results are shown. Again, there is no procedure or function created here, just the SQLScript being executed by the engine.



Exercise 2.13.6: Table Manipulation

Exercise 2.13.6.1: Table Variable Operators

We can also use table variable operators to perform DML like operations on table variables without having to invoke the SQL layer. In this exercise we will experiment with the INSERT, UPDATE, DELETE, and SEARCH operators.

Explanation	Screenshot
1. Use what you have learned and create another procedure called **build_products**.	*build_products.hdbprocedure × 1 PROCEDURE "build_products" ( ) 2 LANGUAGE SQLSCRIPT 3 SQL SECURITY INVOKER 4 --DEFAULT SCHEMA <default_schema_name> 5 READS SQL DATA AS 6 BEGIN 7 /********************************* 8 Write your procedure logic 9 *********************************/ 10 END
2. Define an output parameter as shown here. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products.sql	*build_products.hdbprocedure × 1 PROCEDURE "build_products" (2 out ex_products table (PRODUCTID nvarchar(10), 3 CATEGORY nvarchar(20), 4 PRICE decimal(15,2))) 5 LANGUAGE SQLSCRIPT 6 SQL SECURITY INVOKER 7 --DEFAULT SCHEMA <default_schema_name> 8 READS SQL DATA AS

3. Enter the following code as shown. First we need to declare an intermediate table variable called **lt_products** which will hold the result set of the **SELECT** statement. Next enter a **SELECT** statement from the **MD.Products** table for all products. Then use the **INSERT** operator to copy all rows of **lt_products** into the output parameter called **ex_products**. Finally, use the **INSERT** operator to insert **3** new products into the output table. Since you are specifying the actual index, it will insert your new rows at that index and push all existing rows down.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products.sql

build_products.hdbprocedure

```
PROCEDURE "build_products" (
            out ex_products table (PRODUCTID nvarchar(10),
                                   CATEGORY nvarchar(20),
                                   PRICE decimal(15,2) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

declare lt_products table like :ex_products;

lt_products = select PRODUCTID, CATEGORY, PRICE from "MD.Products";
:ex_products.INSERT(:lt_products);
:ex_products.INSERT(('ProductA', 'Software', '1999.99'), 1);
:ex_products.INSERT(('ProductB', 'Software', '2999.99'), 2);
:ex_products.INSERT(('ProductC', 'Software', '3999.99'), 3);

END
```

4. The complete code should look very similar to this.

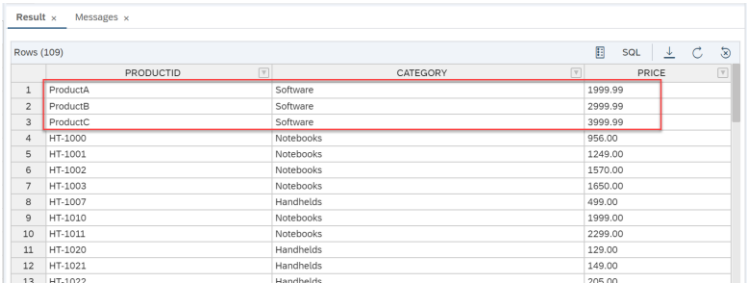
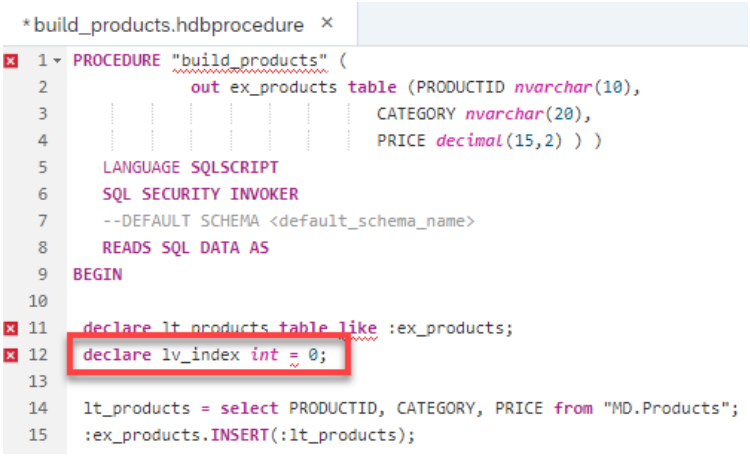
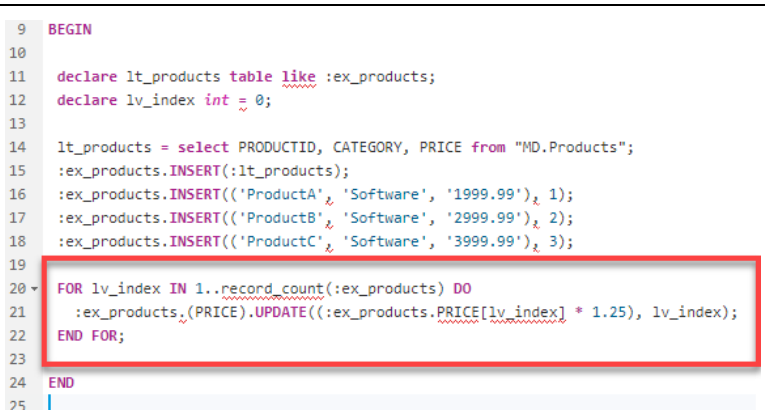
   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products.sql

```
PROCEDURE "build_products" (
        out ex_products table (PRODUCTID
nvarchar(10),
        CATEGORY nvarchar(20),
       PRICE decimal(15,2) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

 declare lt_products table like :ex_products;

 lt_products = select PRODUCTID, CATEGORY,
PRICE from "MD.Products";
 :ex_products.INSERT(:lt_products);
 :ex_products.INSERT(('ProductA', 'Software',
'1999.99'), 1);
 :ex_products.INSERT(('ProductB', 'Software',
'2999.99'), 2);
 :ex_products.INSERT(('ProductC', 'Software',
'3999.99'), 3);
```

	END
5. Use what you have learned and **Save** your work, perform a **Build** on the module. Then return to the **Database Explorer** and call the procedure. In the Results tab, you should see the product data including the new products that you have inserted.	
6. Return to the procedure and further modify it. Enter a **DECLARE** statement as shown. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products2.sql	
7. Next, below the **INSERT** statements, enter a **FOR loop** which iterates over your output table and uses the **UPDATE** operator to update the **price** column for each row. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products2.sql	

8. The complete code should look very similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products2.sql

```
PROCEDURE "build_products" (
              out ex_products table (PRODUCTID
nvarchar(10),
                                     CATEGORY
nvarchar(20),
                                     PRICE
decimal(15,2) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

 declare lt_products table like :ex_products;
 declare lv_index int = 0;

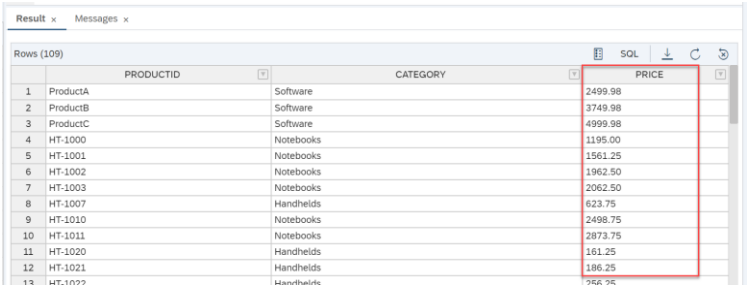
 lt_products = select PRODUCTID, CATEGORY,
PRICE from "MD.Products";
 :ex_products.INSERT(:lt_products);
 :ex_products.INSERT(('ProductA', 'Software',
'1999.99'), 1);
 :ex_products.INSERT(('ProductB', 'Software',
'2999.99'), 2);
 :ex_products.INSERT(('ProductC', 'Software',
'3999.99'), 3);

 FOR lv_index IN 1..record_count(:ex_products)
DO

:ex_products.(PRICE).UPDATE((:ex_products.PRIC
E[lv_index] * 1.25), lv_index);
 END FOR;

END
```

9. Use what you have learned and **Save** your work, perform a **Build** on the module. Then return to the Database Explorer and call the procedure. In the results tab, you should see the product data with the updated price.



	PRODUCTID	CATEGORY	PRICE
1	ProductA	Software	2499.98
2	ProductB	Software	3749.98
3	ProductC	Software	4999.98
4	HT-1000	Notebooks	1195.00
5	HT-1001	Notebooks	1561.25
6	HT-1002	Notebooks	1962.50
7	HT-1003	Notebooks	2062.50
8	HT-1007	Handhelds	623.75
9	HT-1010	Notebooks	2498.75
10	HT-1011	Notebooks	2873.75
11	HT-1020	Handhelds	161.25
12	HT-1021	Handhelds	186.25

10. Return to the procedure and further modify it. Enter **DECLARE** statements as shown.

    Note: If you do not wish to type this code, you can

```
9  BEGIN
10
11  declare lt_products table like :ex_products;
12  declare lv_index int = 0;
13  declare lv_del_index int array;
14  declare lv_array_index int = 0;
```

reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products3.sql

11. After the FOR loop which performed the UPDATE, enter another **FOR loop** which checks the price for each row and if it is **less than 2500**, adds the index of that row to an array. Finally add a **DELETE** operator which uses the array from the FOR loop to delete the selected rows of the table.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products3.sql

```
22  FOR lv_index IN 1..record_count(:ex_products) DO
23    :ex_products.(PRICE).UPDATE((:ex_products.PRICE[lv_index]
24  END FOR;
25
26  FOR lv_index IN 1..record_count(:ex_products) DO
27    IF :ex_products.PRICE[lv_index] <= 2500.00 THEN
28     lv_array_index = lv_array_index + 1;
29     lv_del_index[lv_array_index] = lv_index;
30   END IF;
31  END FOR;
32
33  :ex_products.DELETE(:lv_del_index);
34
35  END
```

12. The completed code should look very similar to the this.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products3.sql

```
PROCEDURE "build_products" (
              out ex_products table (PRODUCTID
nvarchar(10),
                                     CATEGORY
nvarchar(20),
                                     PRICE
decimal(15,2) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

 declare lt_products table like :ex_products;
 declare lv_index int = 0;
 declare lv_del_index int array;
 declare lv_array_index int = 0;

 lt_products = select PRODUCTID, CATEGORY,
PRICE from "MD.Products";
 :ex_products.INSERT(:lt_products);
```

```
 :ex_products.INSERT(('ProductA', 'Software',
'1999.99'), 1);
 :ex_products.INSERT(('ProductB', 'Software',
'2999.99'), 2);
 :ex_products.INSERT(('ProductC', 'Software',
'3999.99'), 3);

 FOR lv_index IN 1..record_count(:ex_products)
DO

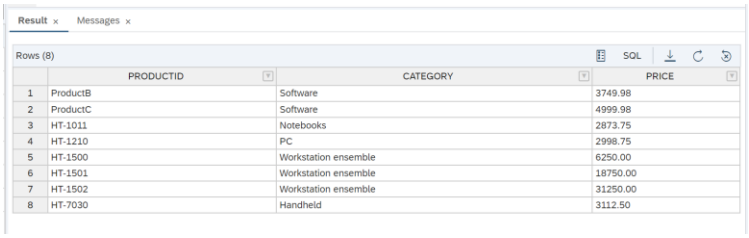
:ex_products.(PRICE).UPDATE((:ex_products.PRIC
E[lv_index] * 1.25), lv_index);
 END FOR;

 FOR lv_index IN 1..record_count(:ex_products)
DO
   IF :ex_products.PRICE[lv_index] <= 2500.00
THEN
    lv_array_index = lv_array_index + 1;
    lv_del_index[lv_array_index] = lv_index;
  END IF;
 END FOR;

 :ex_products.DELETE(:lv_del_index);

END
```

13. Use what you have learned and **Save** your work, and perform a **Build** on the module. Then return to the Database Explorer and call the procedure.



Result × Messages ×

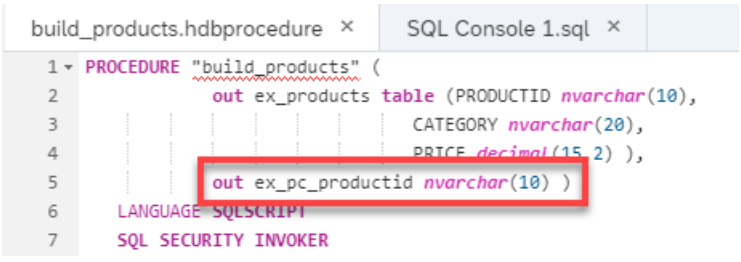
Rows (8) SQL

	PRODUCTID	CATEGORY	PRICE
1	ProductB	Software	3749.98
2	ProductC	Software	4999.98
3	HT-1011	Notebooks	2873.75
4	HT-1210	PC	2998.75
5	HT-1500	Workstation ensemble	6250.00
6	HT-1501	Workstation ensemble	18750.00
7	HT-1502	Workstation ensemble	31250.00
8	HT-7030	Handheld	3112.50

14. Return to the procedure and further modify it. Enter a new output parameter called **ex_pc_productid** as shown.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products4.sql



build_products.hdbprocedure × SQL Console 1.sql ×

```
1  PROCEDURE "build_products" (
2              out ex_products table (PRODUCTID nvarchar(10),
3                                     CATEGORY nvarchar(20),
4                                     PRICE decimal(15,2) ),
5              out ex_pc_productid nvarchar(10) )
6    LANGUAGE SQLSCRIPT
7    SQL SECURITY INVOKER
```

15. After the **DELETE** statement, add the following two lines of code. This code will search the output table for the first row that contains a **CATEGORY = 'PC'** and returns the **productid** as an output parameter.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products4.sql

```
27  FOR lv_index IN 1..record_count(:ex_products) DO
28    IF :ex_products.PRICE[lv_index] <= 2500.00 THEN
29      lv_array_index = lv_array_index + 1;
30      lv_del_index[lv_array_index] = lv_index;
31   END IF;
32  END FOR;
33
34  :ex_products.DELETE(:lv_del_index);
35
36  lv_index = :ex_products.SEARCH("CATEGORY", 'PC', 1);
37  ex_pc_productid = :ex_products.PRODUCTID[lv_index];
38
39  END
40
```

16. The completed code should look very similar to the this.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/build_products4.sql

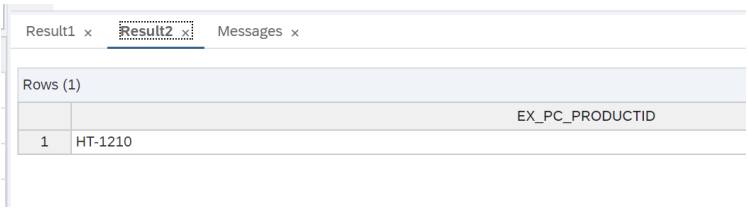
```
PROCEDURE "build_products" (
              out ex_products table (PRODUCTID
nvarchar(10),
                                     CATEGORY
nvarchar(20),
                                     PRICE
decimal(15,2) ),
            out ex_pc_productid nvarchar(10) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN

 declare lt_products table like :ex_products;
 declare lv_index int = 0;
 declare lv_del_index int array;
 declare lv_array_index int = 0;

 lt_products = select PRODUCTID, CATEGORY,
PRICE from "MD.Products";
 :ex_products.INSERT(:lt_products);
 :ex_products.INSERT(('ProductA', 'Software',
'1999.99'), 1);
 :ex_products.INSERT(('ProductB', 'Software',
'2999.99'), 2);
 :ex_products.INSERT(('ProductC', 'Software',
'3999.99'), 3);

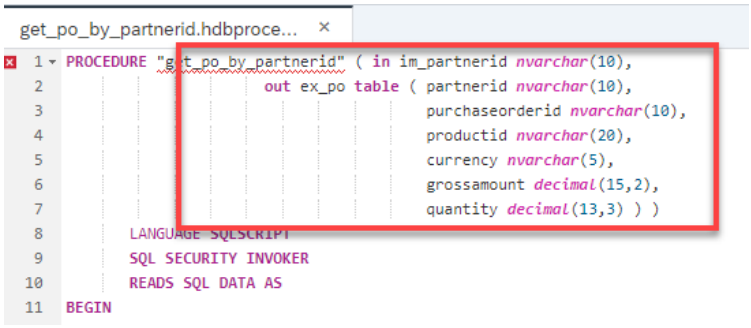
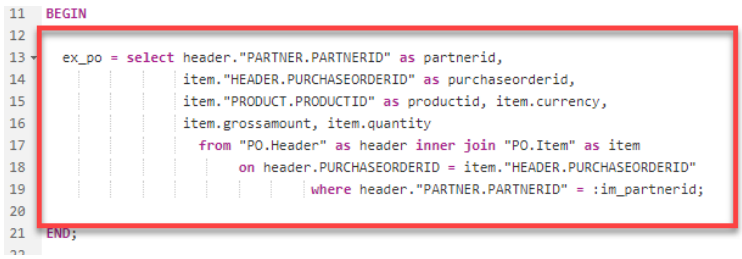
 FOR lv_index IN 1..record_count(:ex_products)
DO

:ex_products.(PRICE).UPDATE((:ex_products.PRIC
E[lv_index] * 1.25), lv_index);
```

	`` END FOR; FOR lv_index IN 1..record_count(:ex_products) DO IF :ex_products.PRICE[lv_index] <= 2500.00 THEN lv_array_index = lv_array_index + 1; lv_del_index[lv_array_index] = lv_index; END IF; END FOR; :ex_products.DELETE(:lv_del_index); lv_index = :ex_products.SEARCH("CATEGORY", 'PC', 1); ex_pc_productid = :ex_products.PRODUCTID[lv_index]; END ```
17. Use what you have learned and **Save** your work, and perform a **Build** on the module. Then return to the Database Explorer and call the procedure. Click on the **Result2** tab.	

Exercise 2.13.6.2: MAP_MERGE Operator

The MAP_MERGE operator is used to apply a mapper function on each row of an input table and union all intermediate results into a single result table. The operator can be a replacement for sequential FOR-loops and union patterns. In the following exercise, you will use the MAP_MERGE operator.

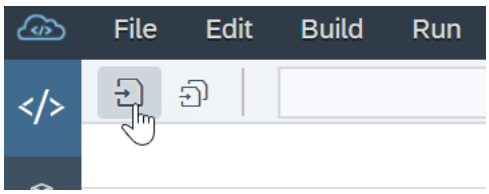
Explanation	Screenshot
1. Use what you have learned and create another procedure called **get_po_by_partnerid**. Define the signature of the procedure as shown here. If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid.sql	
2. In the body of the procedure, insert the **SELECT** statement as shown. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid.sql	
3. The completed code should look very similar to this. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid.sql	`PROCEDURE "get_po_by_partnerid" ( in im_partnerid nvarchar(10),` `out ex_po table ( partnerid nvarchar(10),` `purchaseorderid nvarchar(10),` `productid nvarchar(20),` `currency nvarchar(5),` `grossamount decimal(15,2),`

```
quantity decimal(13,3) ) )
       LANGUAGE SQLSCRIPT
       SQL SECURITY INVOKER
       READS SQL DATA AS
BEGIN

  ex_po = select header."PARTNER.PARTNERID" as
partnerid,
                  item."HEADER.PURCHASEORDERID"
as purchaseorderid,
                  item."PRODUCT.PRODUCTID" as
productid, item.currency,
                  item.grossamount,
item.quantity
                    from "PO.Header" as header
inner join "PO.Item" as item
                           on
header.PURCHASEORDERID =
item."HEADER.PURCHASEORDERID"
                                    where
header."PARTNER.PARTNERID" = :im_partnerid;

END;
```

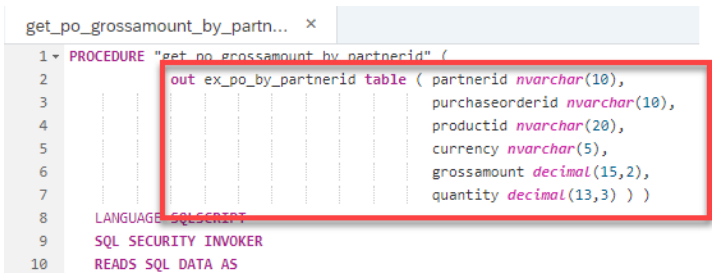
4. **Save** your work.



5. Use what you have learned and create another procedure called **get_po_grossamount_by_partnerid**. Define an out put parameter called **ex_po_by_partnerid** with the structure as shown here.

   Note: If you do not wish to type this code, you can reference the solution web page at

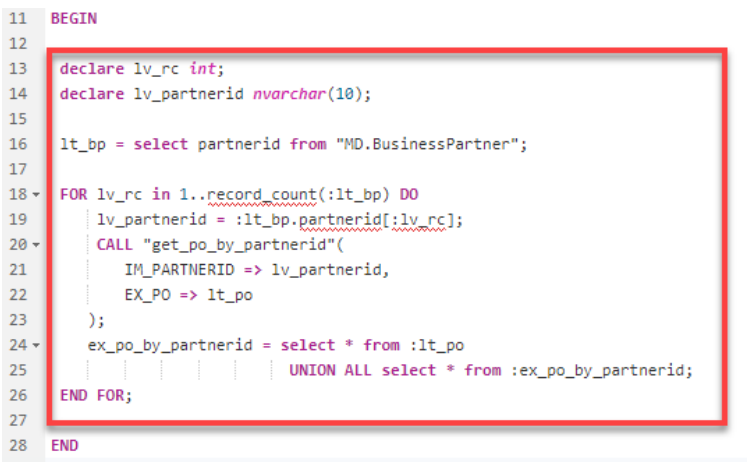
   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_grossamount_by_partnerid.sql



6. Insert the following code into the body of the procedure. Here we declare 2 variables, then do a select against the business partners table to retrieve all business partners. Then loop through the business partners table and call the **get_po_by_partnerid** procedure for each partnerid. Finally, we add the purchase orders for each partner to the output table parameter.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_grossamount_by_partnerid.sql



```
BEGIN

declare lv_rc int;
declare lv_partnerid nvarchar(10);

lt_bp = select partnerid from "MD.BusinessPartner";

FOR lv_rc in 1..record_count(:lt_bp) DO
    lv_partnerid = :lt_bp.partnerid[:lv_rc];
    CALL "get_po_by_partnerid"(
        IM_PARTNERID => lv_partnerid,
        EX_PO => lt_po
    );
    ex_po_by_partnerid = select * from :lt_po
                         UNION ALL select * from :ex_po_by_partnerid;
END FOR;

END
```

7. The completed code should look very similar to this.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_grossamount_by_partnerid.sql

```
PROCEDURE "get_po_grossamount_by_partnerid" (
              out ex_po_by_partnerid table (
partnerid nvarchar(10),
purchaseorderid nvarchar(10),
productid nvarchar(20),
currency nvarchar(5),
grossamount decimal(15,2),

quantity decimal(13,3) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   READS SQL DATA AS
BEGIN

 declare lv_rc int;
 declare lv_partnerid nvarchar(10);

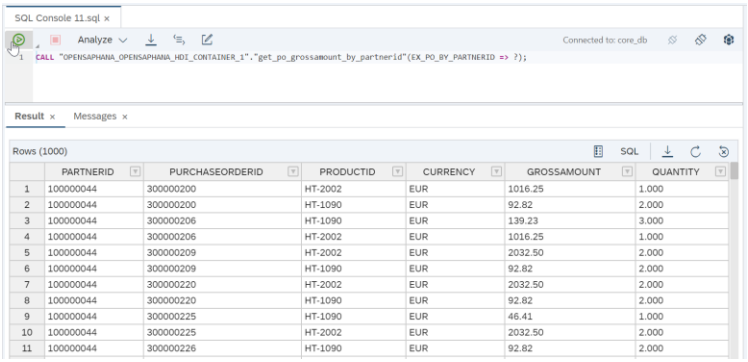
 lt_bp = select partnerid from
"MD.BusinessPartner";

 FOR lv_rc in 1..record_count(:lt_bp) DO
     lv_partnerid = :lt_bp.partnerid[:lv_rc];
     CALL "get_po_by_partnerid"(
            IM_PARTNERID => lv_partnerid,
            EX_PO => lt_po
    );
    ex_po_by_partnerid = select * from :lt_po
```

```
                                        UNION ALL select *
from :ex_po_by_partnerid;
 END FOR;

END
```

8. **Save** your work and perform a **Build**. Next go to the Database Explorer and generate a call statement for the **get_po_grossamount_by_partnerid** procedure. Then click **Run.**



	PARTNERID	PURCHASEORDERID	PRODUCTID	CURRENCY	GROSSAMOUNT	QUANTITY
1	100000044	300000200	HT-2002	EUR	1016.25	1.000
2	100000044	300000200	HT-1090	EUR	92.82	2.000
3	100000044	300000206	HT-1090	EUR	139.23	3.000
4	100000044	300000206	HT-2002	EUR	1016.25	1.000
5	100000044	300000209	HT-2002	EUR	2032.50	2.000
6	100000044	300000209	HT-1090	EUR	92.82	2.000
7	100000044	300000220	HT-2002	EUR	2032.50	2.000
8	100000044	300000220	HT-1090	EUR	92.82	2.000
9	100000044	300000225	HT-1090	EUR	46.41	1.000
10	100000044	300000225	HT-2002	EUR	2032.50	2.000
11	100000044	300000226	HT-1090	EUR	92.82	2.000

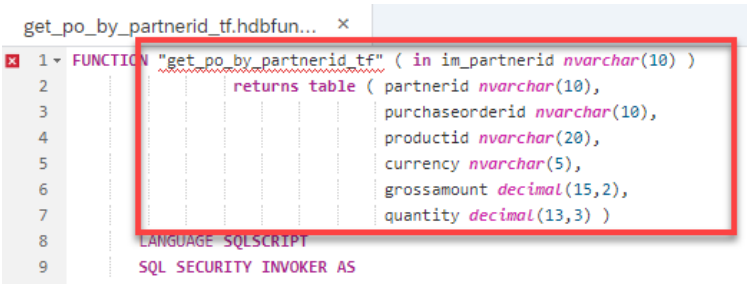
9. The results will then be shown, click the **Messages** tab, and note the execution time.



10. You will now rewrite the procedure to leverage the **MAP_MERGE** function which should improve the performance. Since the MAP_MERGE function only supports Table Functions as an input, we need to convert the **get_po_by_partnerid** procedure to a **table function**. Use what you have learned and create a new function in the functions folder called **get_po_by_partnerid_tf**. This will be the **Mapper** function. Adapt the signature of the table function from the **get_address** procedure.

    Note: If you do not wish to type this code, you can reference the solution web page at

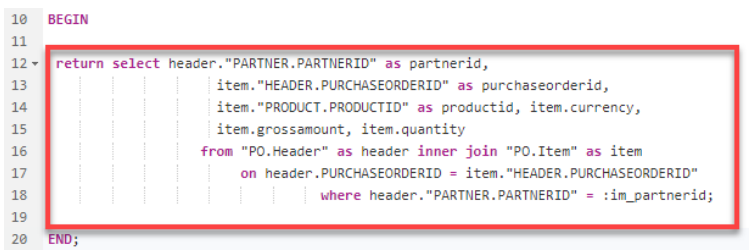
    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid_tf.sql



11. Copy only the **SELECT** statement from the **get_po_by_partnerid** procedure and put **RETURN** in front of it.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid_tf.sql



12. The completed code should look very similar to this.

Note: If you do not wish to type this code, you can reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_by_partnerid_tf.sql

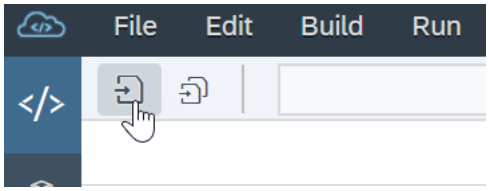
```
FUNCTION "get_po_by_partnerid_tf" ( in
im_partnerid nvarchar(10) )
                  returns table ( partnerid
nvarchar(10),
purchaseorderid nvarchar(10),
productid nvarchar(20),
currency nvarchar(5),
grossamount decimal(15,2),
quantity decimal(13,3) )
       LANGUAGE SQLSCRIPT
       SQL SECURITY INVOKER AS
BEGIN

 return select header."PARTNER.PARTNERID" as
partnerid,

item."HEADER.PURCHASEORDERID" as
purchaseorderid,
                        item."PRODUCT.PRODUCTID"
as productid, item.currency,
                        item.grossamount,
item.quantity
                     from "PO.Header" as header
inner join "PO.Item" as item
                           on
header.PURCHASEORDERID =
item."HEADER.PURCHASEORDERID"
                                          where
header."PARTNER.PARTNERID" = :im_partnerid;

END;
```

13. **Save** your work.



14. Return to the **get_po_grossamount_by_partnerid** procedure. Remove the **DECLARE** statements and the entire **FOR loop**, leaving only the **SELECT** from the **MD.BusinessPartner** table as shown.

```
BEGIN

 lt_bp = select partnerid from "MD.BusinessPartner";

END
```

15. Next, enter the **MAP_MERGE** statement as shown. The MAP_MERGE function accepts two parameters. The first is the table which is being iterated over. The second parameter is the name of the function which will be executed for each row in the iteration. You will notice we are simply passing the **PARTNERID** column for each row of LT_BP to the **get_po_by_partner_id_tf** function.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_grossamount_by_partnerid2.sql

```
BEGIN

 lt_bp = select partnerid from "MD.BusinessPartner";

ex_po_by_partnerid = MAP_MERGE( :lt_bp,
                    "get_po_by_partnerid_tf"(:lt_bp.partnerid) );

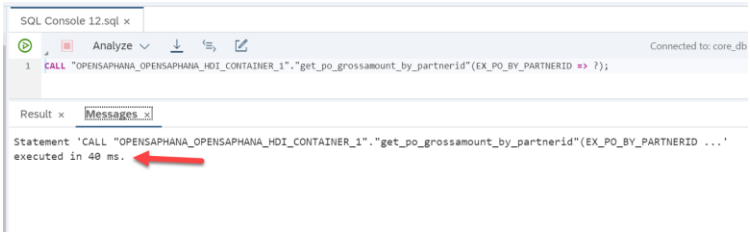
END
```

16. The completed code should look very similar to this.

    Note: If you do not wish to type this code, you can reference the solution web page at

    https://github.com/SAP/hana-xsa-opensap-

```
PROCEDURE "get_po_grossamount_by_partnerid" (
              out ex_po_by_partnerid table (
partnerid nvarchar(10),
purchaseorderid nvarchar(10),
productid nvarchar(20),
currency nvarchar(5),
grossamount decimal(15,2),
quantity decimal(13,3) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
```

hana7/blob/snippets_2.3.2/ex2/get_po_grossamount_by_partnerid2.sql	`READS SQL DATA AS` `BEGIN` `lt_bp = select partnerid from "MD.BusinessPartner";` `ex_po_by_partnerid = MAP_MERGE( :lt_bp,` `"get_po_by_partnerid_tf"(:lt_bp.partnerid) );` `END`
17. **Save** your work. **Build** on your project. Return to the **Database Explorer** and once again execute the procedure by clicking **Run**. You might notice that performance is better!	

Exercise 2.13.6.3: MAP_REDUCE Operator

In this exercise, we will take it a little further and see how we can use the MAP_REDUCE function to further process the purchase order data.

Explanation	Screenshot
1. Use what you have learned and create a new function in the functions folder called **get_po_avg_by_partnerid_tf**. This function will be the "Reducer" function. Enter the signature of the function as shown here. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid_tf.sql	get_po_avg_by_partnerid_tf.hd... × 1 FUNCTION "get_po_avg_by_partnerid_tf" (2 in im_partnerid nvarchar(10), 3 in im_items table(purchaseorderid nvarchar(10), 4 productid nvarchar(20), 5 currency nvarchar(5), 6 grossamount decimal(15,2), 7 quantity decimal(13,3))) 8 RETURNS table(partnerid nvarchar(10), 9 avgitemprice decimal(32,16), 10 avgorderquantity decimal(32,16), 11 currency nvarchar(5)) 12 LANGUAGE SQLSCRIPT 13 SQL SECURITY INVOKER AS
2. Enter the body of the function as shown here. This logic will calculate the average quantity and average price for each PO by partnered. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid_tf.sql	14 BEGIN 15 16 lt_quantities = select sum(quantity) sumquantity, currency 17 from :im_items group by purchaseorderid, currency; 18 19 lt_avgquantities = select avg(sumquantity) avgquantity, currency 20 from :lt_quantities group by currency; 21 22 lt_avgprice = select avg(grossamount/quantity) avgprice, currency 23 from :im_items group by currency; 24 25 return select :im_partnerid as partnerid, avgprice avgitemprice, 26 avgquantity avgorderquantity, avgprice.currency 27 from :lt_avgquantities as avgqty 28 inner join :lt_avgprice as avgprice 29 on avgprice.currency = avgqty.currency; 30 END;
3. The completed code should look very similar to this. **Note:** If you do not wish to type this code, you can	FUNCTION "get_po_avg_by_partnerid_tf" (in im_partnerid nvarchar(10), in im_items table(purchaseorderid nvarchar(10), productid nvarchar(20),

reference the solution web page at

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid_tf.sql

```
currency nvarchar(5),
grossamount decimal(15,2),
quantity decimal(13,3)))
       RETURNS table( partnerid nvarchar(10),
                      avgitemprice
decimal(32,16),
                      avgorderquantity
decimal(32,16),
                      currency nvarchar(5))
       LANGUAGE SQLSCRIPT
       SQL SECURITY INVOKER AS
BEGIN

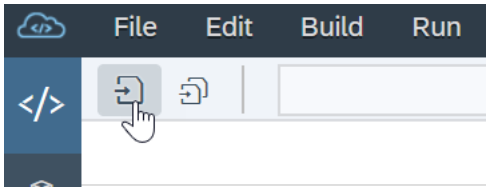
 lt_quantities = select sum(quantity)
sumquantity, currency
                       from :im_items group by
purchaseorderid, currency;

 lt_avgquantities = select avg(sumquantity)
avgquantity, currency
                       from :lt_quantities
group by currency;

 lt_avgprice = select
avg(grossamount/quantity) avgprice, currency
                       from :im_items group by
currency;

 return select :im_partnerid as partnerid,
avgprice avgitemprice,
                 avgquantity avgorderquantity,
avgprice.currency
                       from :lt_avgquantities
as avgqty
                            inner join
:lt_avgprice  as avgprice
                                        on
avgprice.currency = avgqty.currency;
END;
```

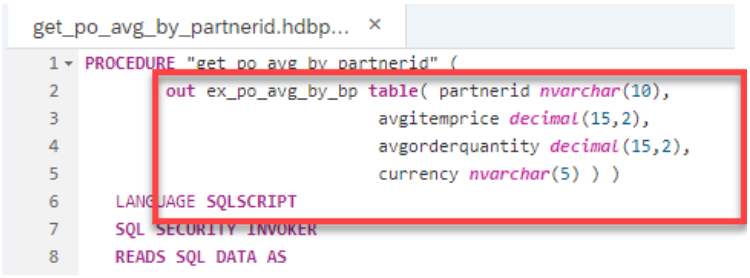
4. **Save** your work.



5. Use what you have learned and create a new procedure in the procedures folder called **get_po_avg_by_partnerid**. Define the signature as shown here.

   Note: If you do not wish to type this code, you can reference the solution web page at

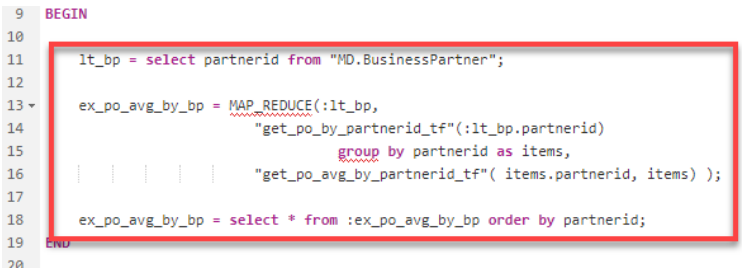
   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid.sql



6. Enter the body of the procedure as shown here. Notice here we use the **MAP_REDUCE** function and pass to it the list of business partners along with the mapper function **get_po_by_partnerid_tf** and the reducer function **get_po_avg_by_partnerid_tf** . Finally we ensure the sort order of the output table by doing a **SELECT ORDER BY**.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid.sql



7. The completed code should look very similar to the following.

   Note: If you do not wish to type this code, you can reference the solution web page at

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/get_po_avg_by_partnerid.sql

```
PROCEDURE "get_po_avg_by_partnerid" (
          out ex_po_avg_by_bp table( partnerid
nvarchar(10),
avgitemprice decimal(15,2),
avgorderquantity decimal(15,2),
currency nvarchar(5) ) )
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   READS SQL DATA AS
BEGIN

      lt_bp = select partnerid from
"MD.BusinessPartner";
```

```
      ex_po_avg_by_bp = MAP_REDUCE(:lt_bp,

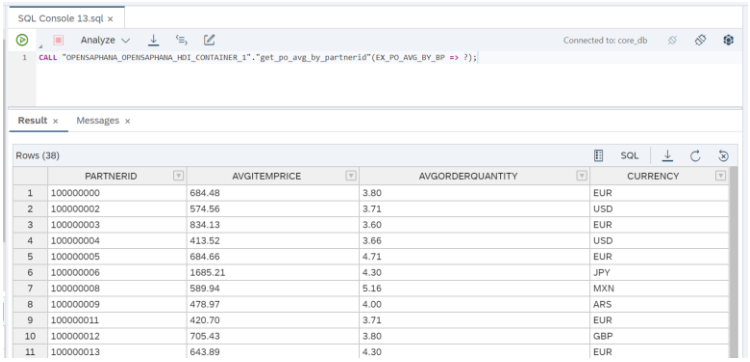
"get_po_by_partnerid_tf"(:lt_bp.partnerid)
                                          group by
partnerid as items,

"get_po_avg_by_partnerid_tf"( items.partnerid,
items) );

      ex_po_avg_by_bp = select * from
:ex_po_avg_by_bp order by partnerid;
END
```

8. **Save** your work and perform a **Build**.

   Return to the Database Explorer and generate a call statement for the procedure **get_po_avg_by_partnerid** and click **Run**.

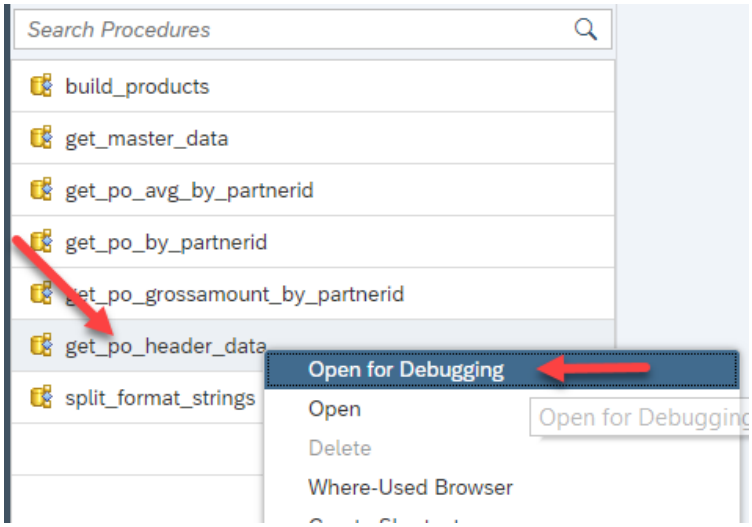
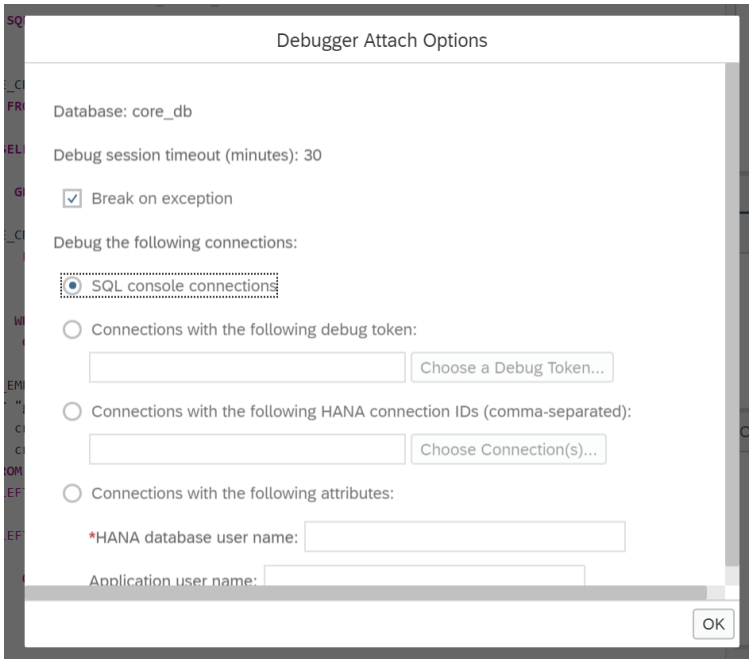


	PARTNERID	AVGITEMPRICE	AVGORDERQUANTITY	CURRENCY
1	100000000	684.48	3.80	EUR
2	100000002	574.56	3.71	USD
3	100000003	834.13	3.60	EUR
4	100000004	413.52	3.66	USD
5	100000005	684.66	4.71	EUR
6	100000006	1685.21	4.30	JPY
7	100000008	589.94	5.16	MXN
8	100000009	478.97	4.00	ARS
9	100000011	420.70	3.71	EUR
10	100000012	705.43	3.80	GBP
11	100000013	643.89	4.30	EUR

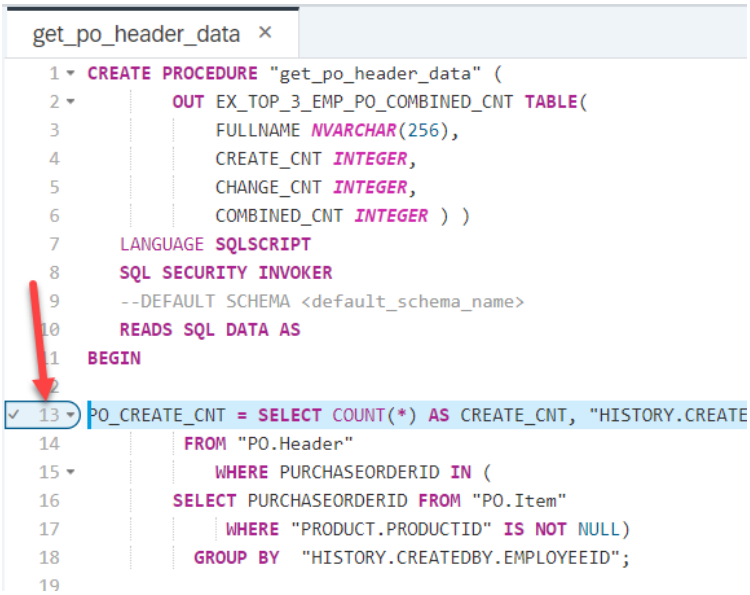
Exercise 2.13.7: Debugging SQLScript

Exercise 2.13.7.1: Debugging a Procedure

In this exercise, you will learn to use the SQLScript debugger, set breakpoints, and evaluate intermediate result sets.

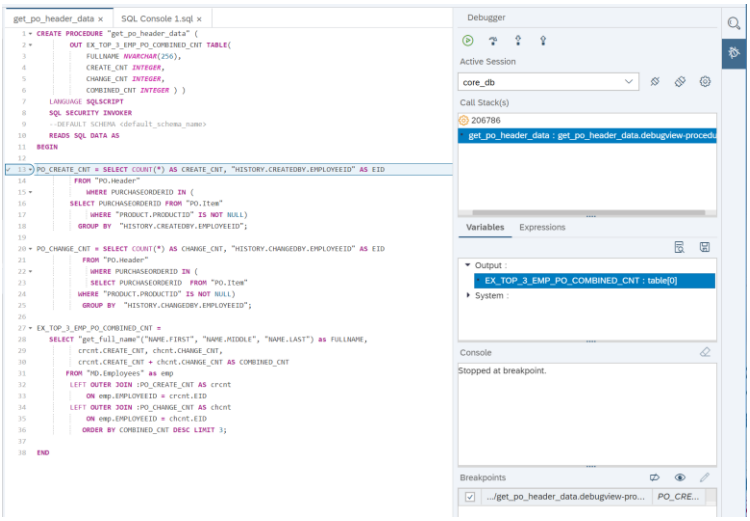
Explanation	Screenshot
1. In the Database Explorer, choose the Procedures folder, and right-click on the **get_po_header_data** procedure and choose **Open for Debugging**.	
2. A dialog will appear. Make sure that SQL console connections is marked and click **OK**.	

3. Insert a breakpoint on line 13 simply by clicking on the line number.



4. Execute the procedure. You will notice that execution has stopped at your breakpoint.

   You can see all of the variables and parameters for this procedure. You might notice that this debug pane is currently not showing the intermediate table variables at this point.



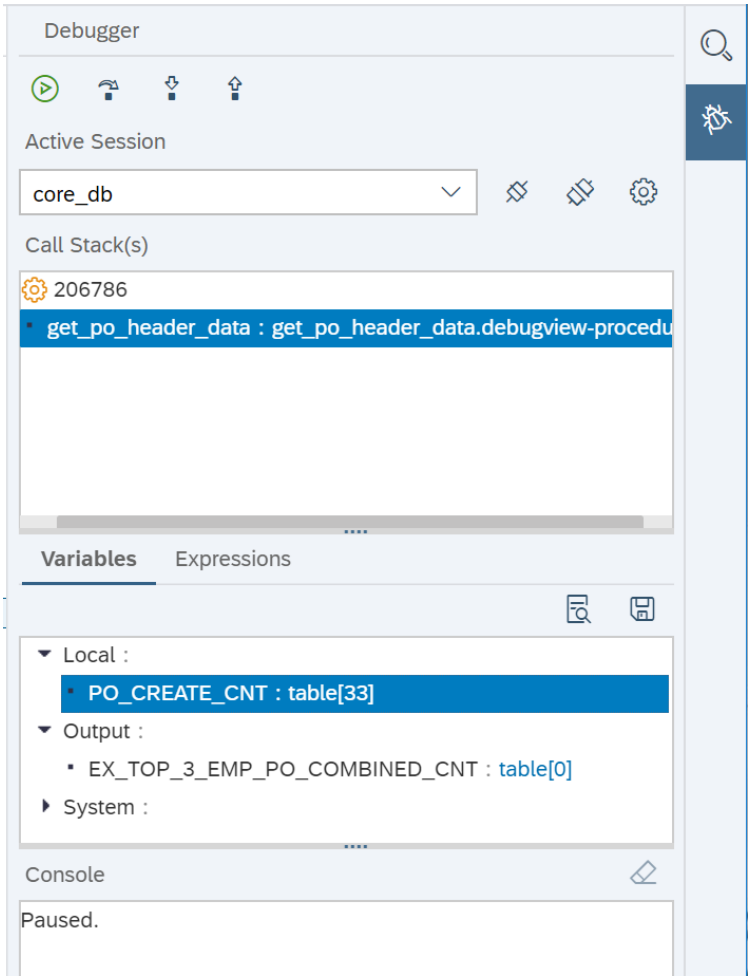
5. Click the **Step Over** button.



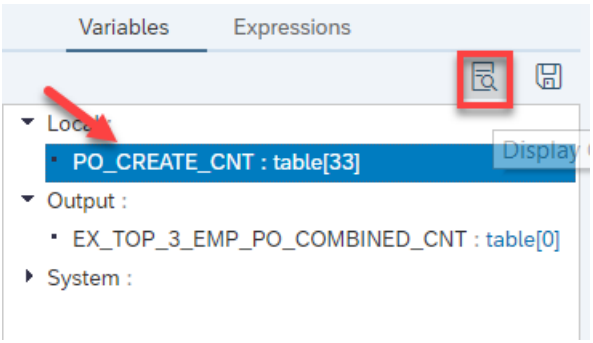
6. You will notice that execution has continued to the next statement.

   In the debugger pane, a new local variable has been added under the Local section.

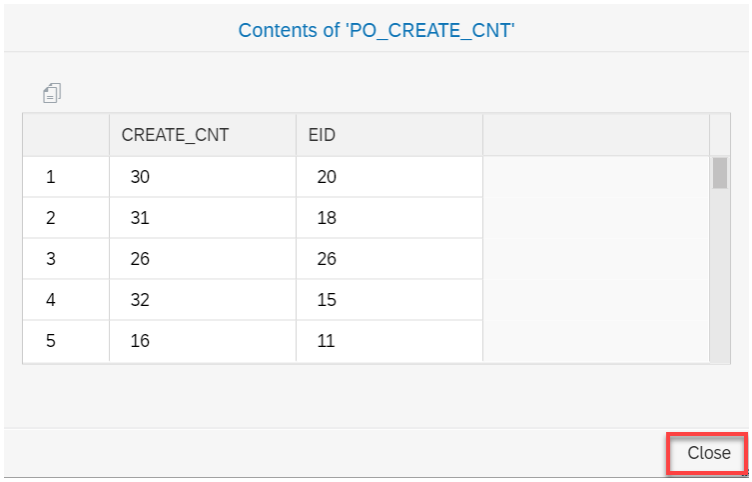
   This is because it is not defined explicitly and will be implicitly declared at runtime during first usage.



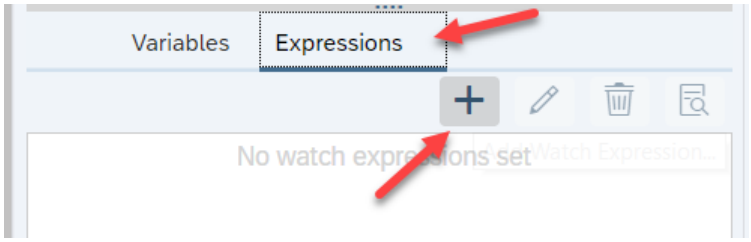
7. To see the data for this intermediate table variable, select it and click **Display Content**.



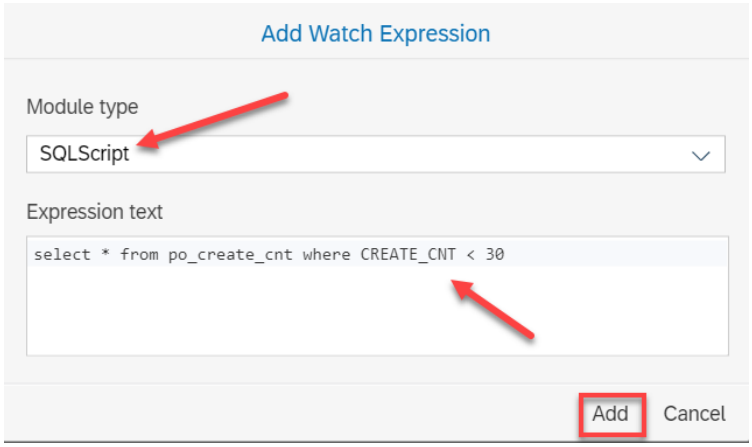
8. A new window is then opened showing the data in the table. Review the data and close the window by clicking the **Close** button.



9. Click the **Expressions** tab, and click the **Plus** button.

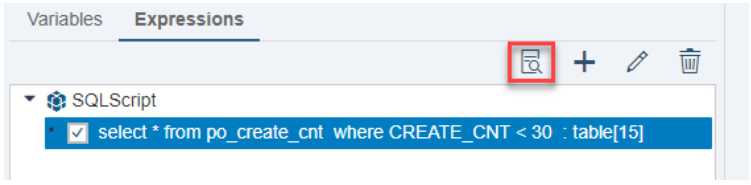


10. Make sure that module type is SQLScript and enter the SELECT statement as shown. Finally click **Add**.

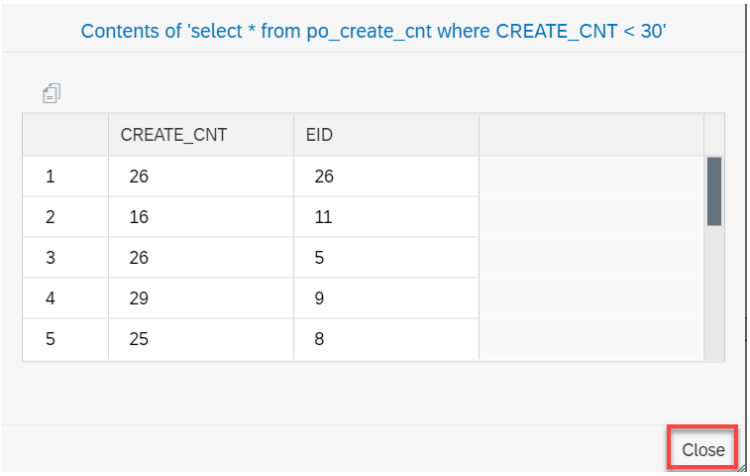



```
select * from po_create_cnt
            where CREATE_CNT < 30
```

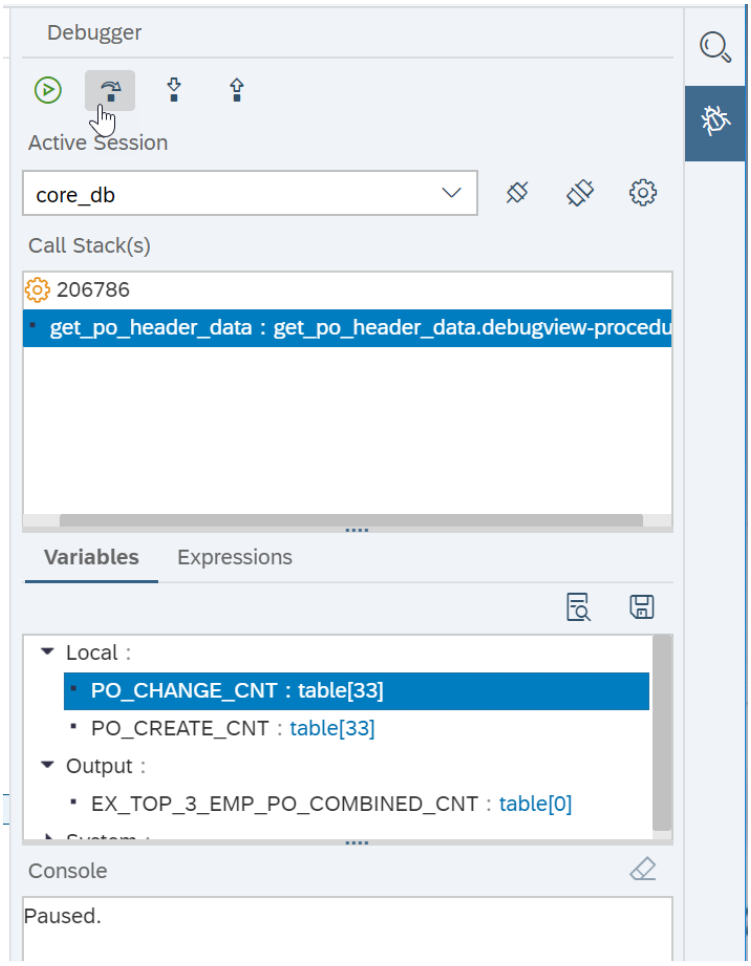
11. Make sure the expression is selected, and click the **Display Content** button.



12. Review the results and click **Close.**

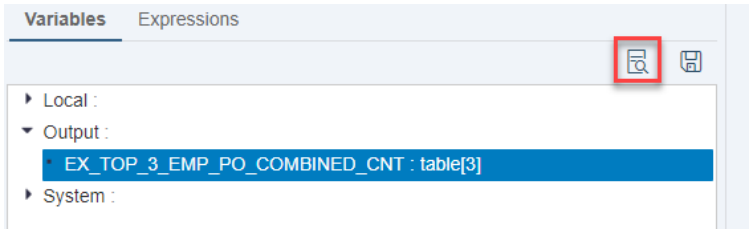


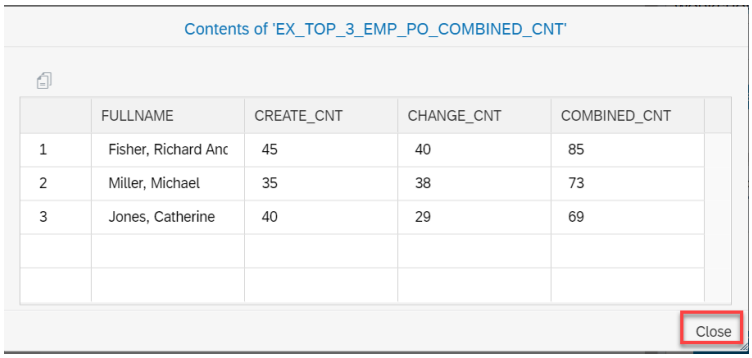
13. Once again click the **Step Over** button. Notice the next intermediate table variables is also added. You can review the data in this table as well.



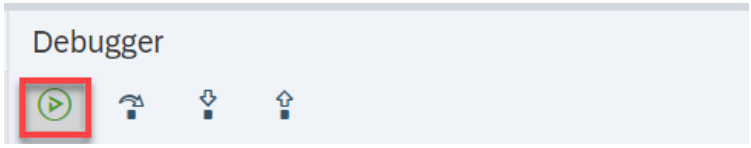
14. Continue to step through the code and when execution stops at the END statement of the procedure, display the contents of the output parameter the same way you did for the intermediate table variables.

    Finally, close the window by clicking **Close**.



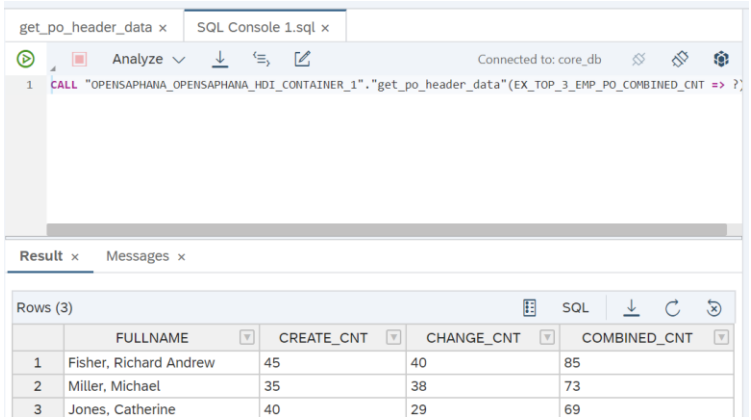


15. Click the **Resume** button.



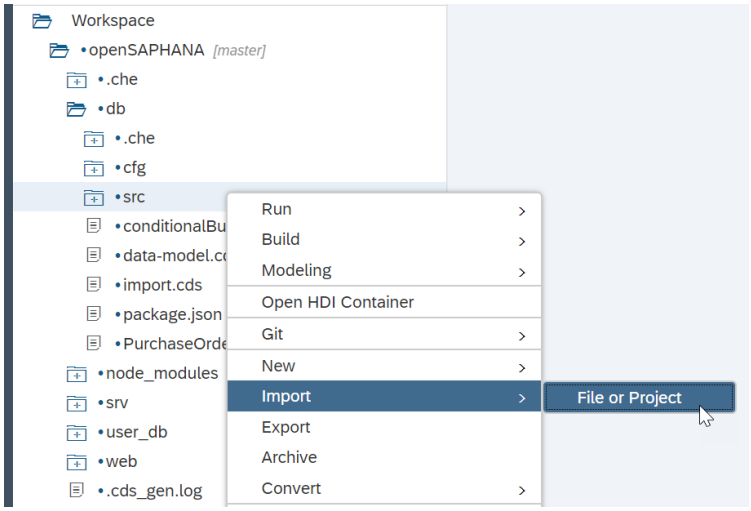
16. Execution of the procedure is now completed.

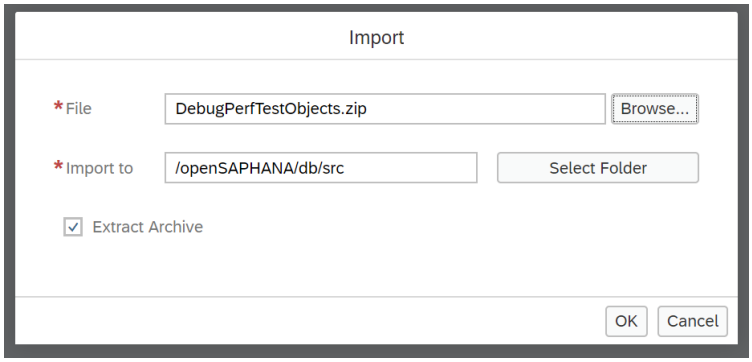
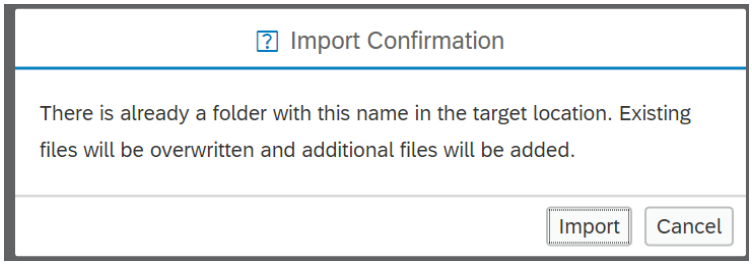
    Return to the SQL tab and check the results.



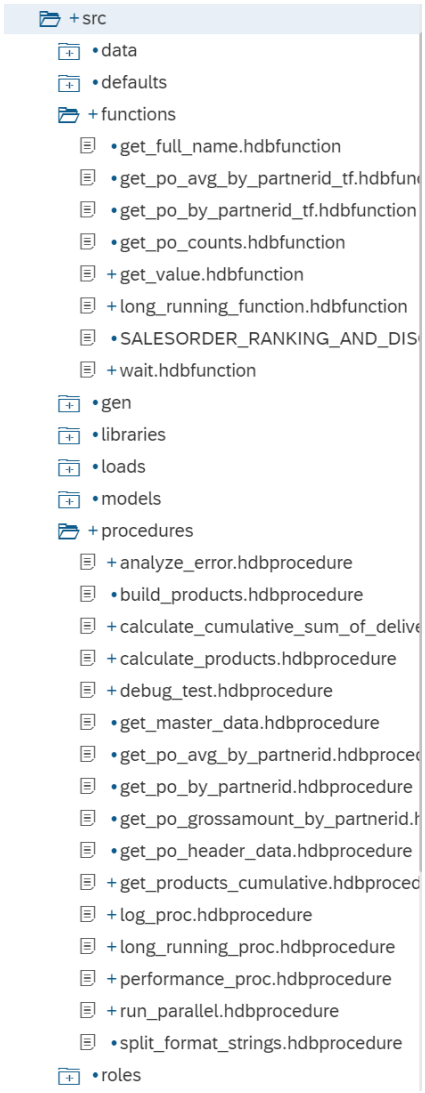
Exercise 2.13.7.2: Breaking on Error

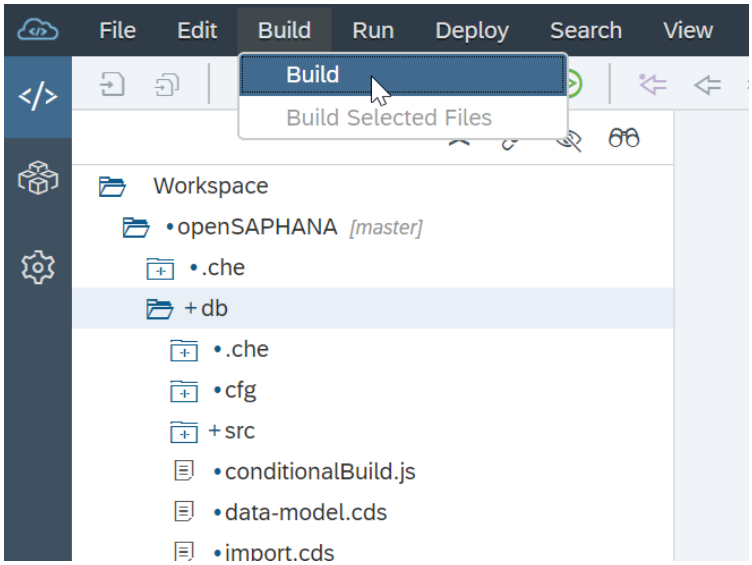
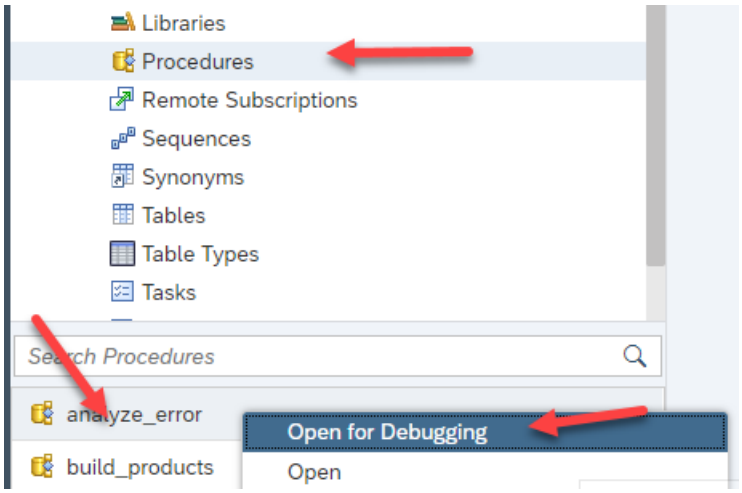
In this section, we will learn how to analyze functional issues using the SQL Debugger. We will use the “break on error” feature of the debugger to directly stop at the position where the error was thrown and do some further analysis from there.

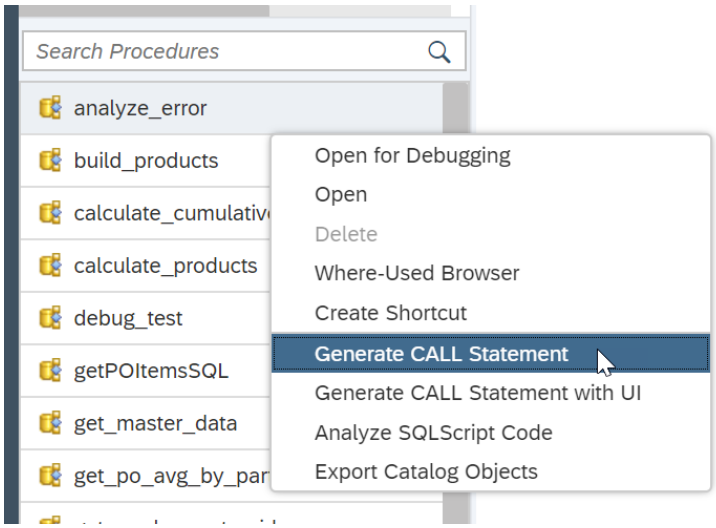
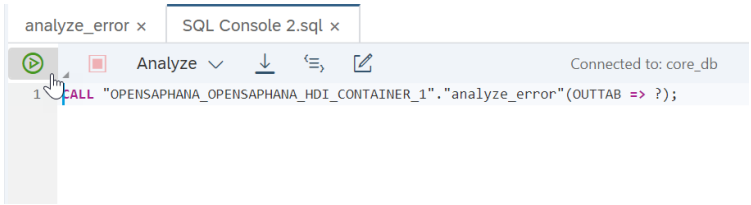
Explanation	Screenshot
1. Before we continue to learn about debugging SQLScript, we need to import some code for the next exercises. **Download** the zip file from the following URL to your desktop. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/DebugPerfTestObjects.zip	
2. Right-click on the **src** folder and choose **Import->File or Project.**	

3. Choose the DebugPerfTestObjects.zip file from the desktop. Make sure that the Extract Archive checkbox is checked and remove the **DebugPerfTestObjects** section in the **Import To** field. Then click **OK**.	
4. Click **Import** to accept the warning message.	

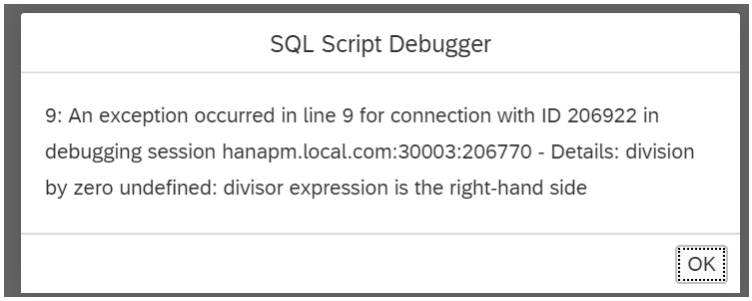
5. You will notice new functions and a few new procedures in your project.

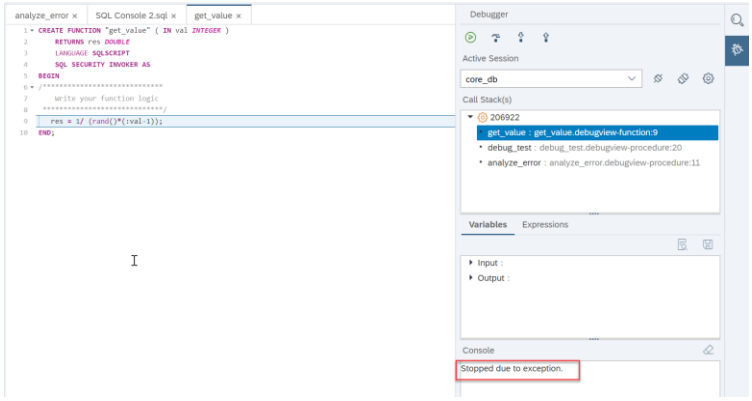


6. **Build** your **db** module.	
7. Switch to the Database Explorer and right-click on the procedure called **analyze_error** and choose **Open for Debugging**.	

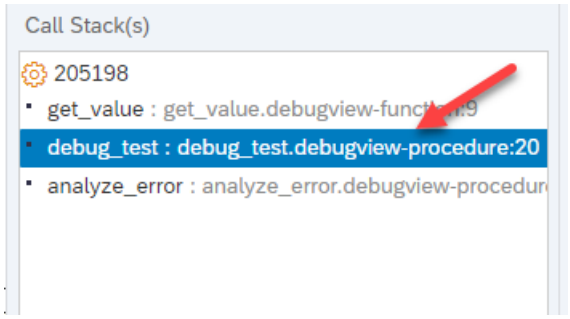
8. If you get the following dialog, make sure that **Break on exception** is checked, and click **OK.**	
9. Right-click on **analyze_error** procedure and choose **Generate Call Statement**.	
10. Click **Run**.	

11. You will get an error message. Click **OK**. You will notice that execution has stopped in the *get_value* procedure. There is also a message in the debug console saying that execution has stopped due to an exception

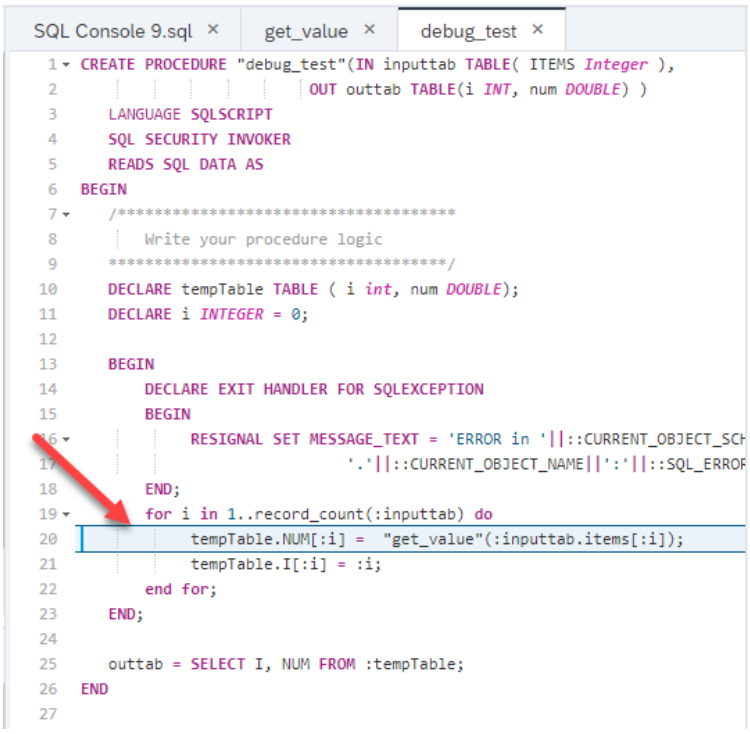




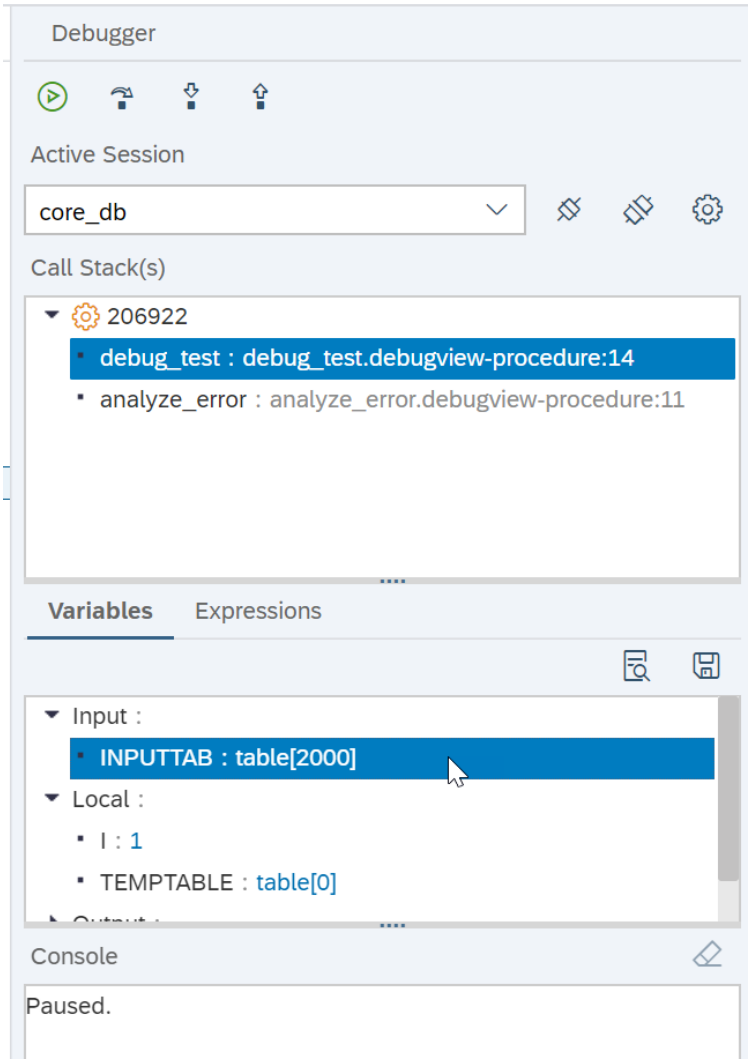
12. Check the call stack to see from where the input for the scalar UDF is coming from. Click on the **debug_test** procedure.



13. Another tab opens for the **debug_test** procedure.

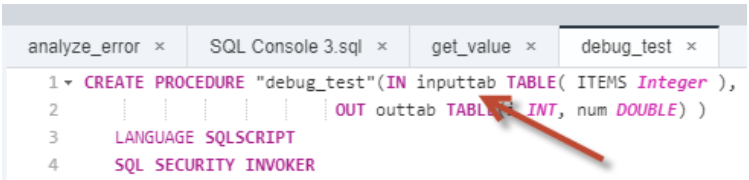
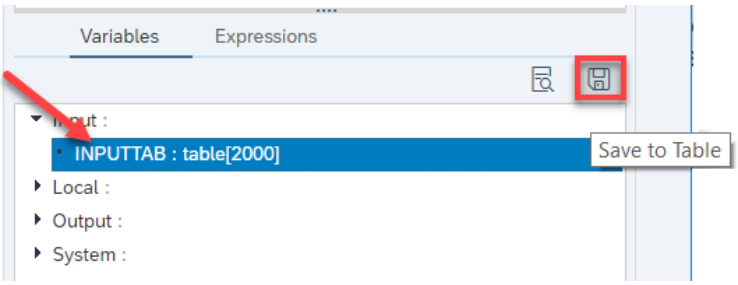
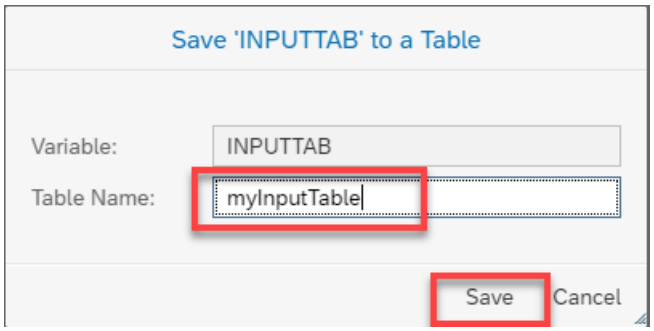


14. To see the values of the variable and intermediate results of the procedure **debug_test**, click the **Step Over** button so that the debugger window will be updated. Now you can do some further investigation. Continue to the next exercise.

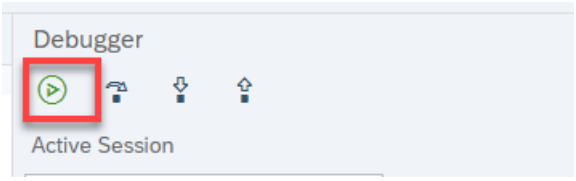


Exercise 2.13.7.3: Save Table Feature

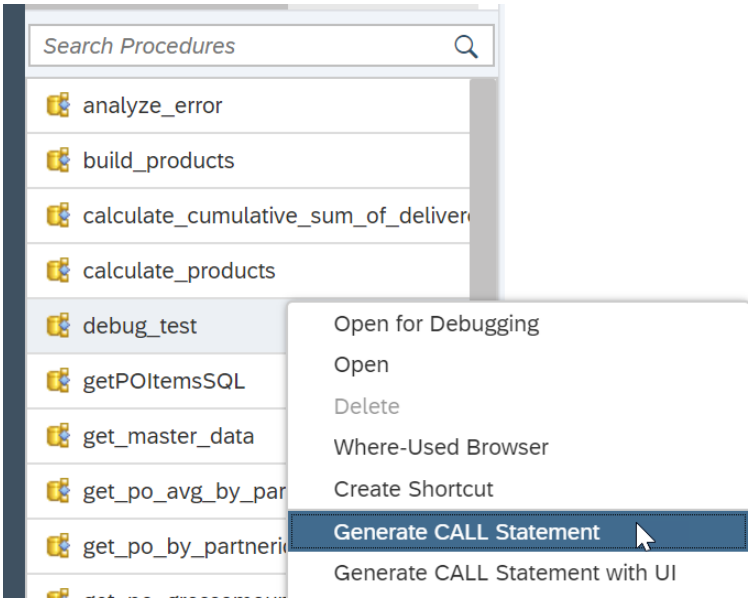
In this section, we will take a look at how we can store an intermediate result set to the database.

Explanation	Screenshot
1. Continuing in our debug session. There are use cases where you do not want to run the whole application over and over again. In our scenario we only want to call the **debug_test** procedure instead of the complete application stack above. But to do so we need the input data for the procedure. For scalar variables this is not a problem, but for tabular input it is, as it is usually derived from some previous calculation. Thankfully, we can save the intermediate result set to the database the **Save Table** feature of the debugger. Continue to the next step.	
2. Select the INPUTTAB in the debugger pane and choose **Save To Table.**	
3. Enter the name of the table as **myInputTable** and click **Save**.	

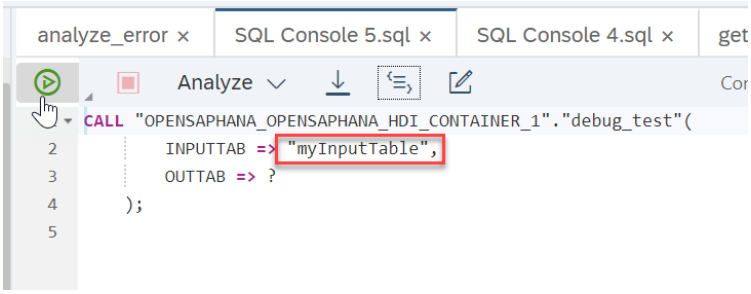
4. Click **Resume** to finish debugging the procedure accepting any errors that my pop up. You will have to click **Resume** several times and accept any pop messages.



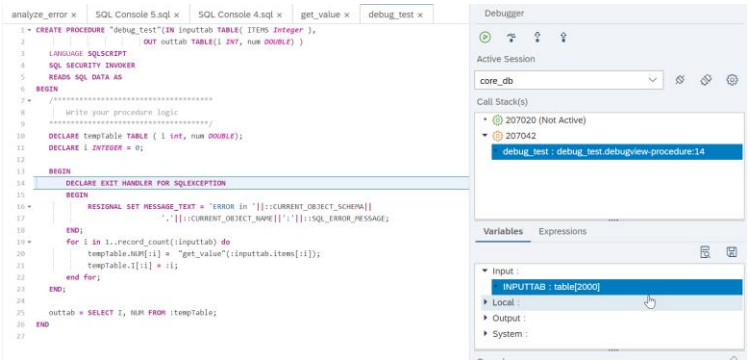
5. Right-click on the **debug_test** procedure and choose **Generate Call Statement**.

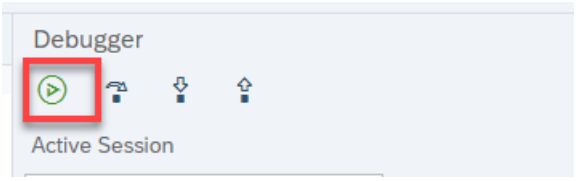


6. Insert the myInputTable as the value for the INPUTTAB parameter in the CALL statement. Click **Run**.



7. Once again, the debugger breaks on the error, but this time we were able to execute the **debug_test** procedure directly with the data saved in the **myInputTable** table. Select **debug_test** in the callstack and click **Step Over** and view the **inputtab** parameter.

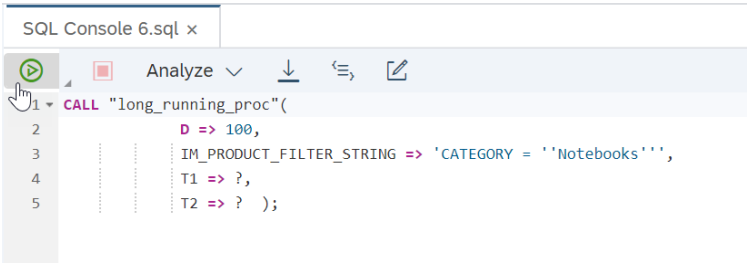


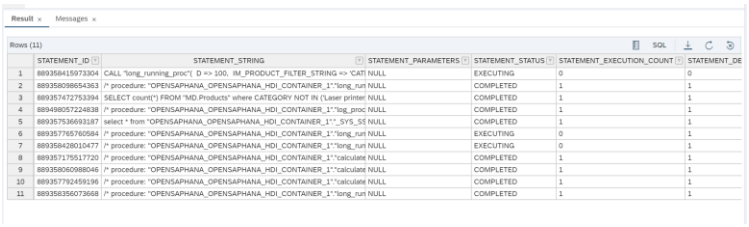
8. Click **Resume** to finish the debug session. Accept any errors that may pop up as well. You will need to click **Resume** several times.	

Exercise 2.13.8: Performance Analysis

Exercise 2.13.8.1: Analyze Long Running Procedures

In this exercise, we will learn how to analyze a procedure that is still running. For this, we will use the monitoring view M_ACTIVE_PROCEDURES to identify the expensive statement within the procedures.

Explanation	Screenshot
1. Return to the SQL Console and enter the following CALL statement into the SQL Console. **Note:** If you do not wish to type this code, you can reference the solution web page at https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/longRunning.sql	`CALL "long_running_proc"( D => 100, IM_PRODUCT_FILTER_STRING => 'CATEGORY = ''Notebooks''', T1 => ?, T2 => ? );`
2. Click **Run**. You will notice that the procedure execution takes some time. But why does the procedure take so long? To get a better understanding of what the procedure is doing we use the **M_ACTIVE_PROCEDURES** monitoring view.	
3. Open another SQL Console tab and enter the following SELECT statement. Click **Run.** **Note:** The complete code can be found at the following URL. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/m_active.sql	`select * from M_ACTIVE_PROCEDURES where procedure_name = 'long_running_proc' ;`

4. The result contains all SQL statements of the procedure that were fully executed (COMPLETED), as well as SQL statements that are currently being executed (EXECUTING). Note that there are two statements that keep running even after quite some time has passed.	

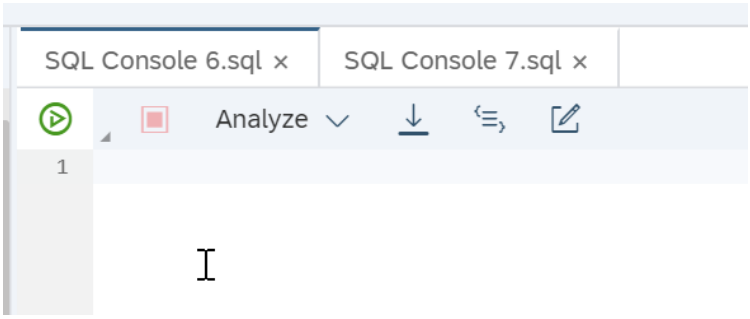
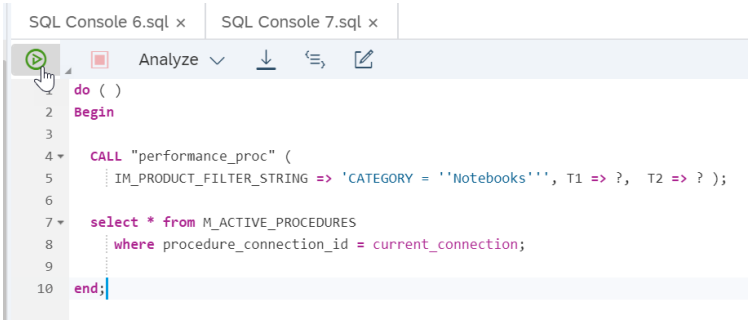
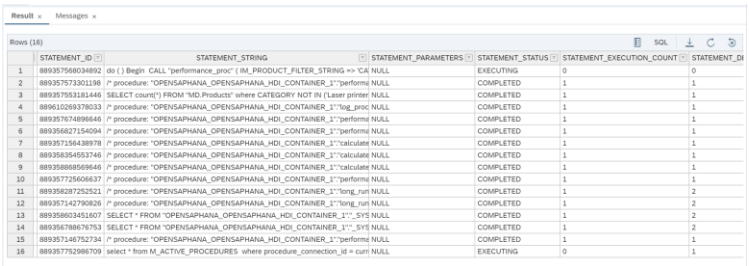
Result × Messages ×

Rows (11) SQL

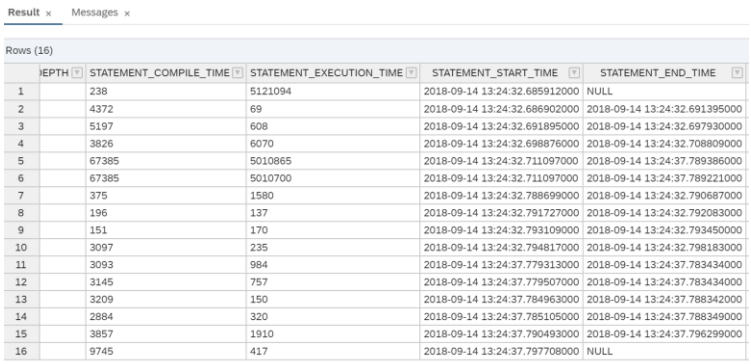
	STATEMENT_ID	STATEMENT_STRING	STATEMENT_PARAMETERS	STATEMENT_STATUS	STATEMENT_EXECUTION_COUNT	STATEMENT_DE
1	88935841597330	CALL "long_running_proc"(D => 100, IM_PRODUCT_FILTER_STRING => 'CAT	NULL	EXECUTING	0	0
2	88935809865436	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."long_run	NULL	COMPLETED	1	1
3	88935747275339	SELECT count(*) FROM "MD.Products" where CATEGORY NOT IN ('Laser printer	NULL	COMPLETED	1	1
4	88949805722483	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."log_proc	NULL	COMPLETED	1	1
5	88935753669318	select * from "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."_SYS_SS	NULL	COMPLETED	1	1
6	88935776576058	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."long_run	NULL	EXECUTING	0	1
7	88935842801047	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."long_run	NULL	EXECUTING	0	1
8	88935717551772	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."calculate	NULL	COMPLETED	1	1
9	88935806098804	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."calculate	NULL	COMPLETED	1	1
10	88935779245919	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."calculate	NULL	COMPLETED	1	1
11	88935835607366	/* procedure: "OPENSAPHANA_OPENSAPHANA_HDI_CONTAINER_1"."long_run	NULL	COMPLETED	1	1

Exercise 2.13.8.2: Overall Performance Analysis

Besides using M_ACTIVE_PROCEDURES to get to know the current status of a running procedures you can also use M_ACTIVE_PROCEDURES to get an overall performance overview for a procedure.

Explanation	Screenshot
1. Return to the SQL Console for your container and clear it out.	
2. Enter the following CALL and SELECT statement into the SQL Console. **Note:** The complete code can be found at the following URL. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/performance_proc.sql	`do ( )` `Begin` `  CALL "performance_proc" (` `    IM_PRODUCT_FILTER_STRING => 'CATEGORY = ''Notebooks''', T1 => ?,  T2 => ? );` `  select * from M_ACTIVE_PROCEDURES` `    where procedure_connection_id = current_connection;` `end;`
3. Click **Run**.	
4. Let's take a deeper look at the result of **M_ACTIVE_PROCEDURES**. You get information about the statement parameters used for a SQL statement.	

5. You get detailed information about the compile time and execution time of each SQL statement (microseconds)



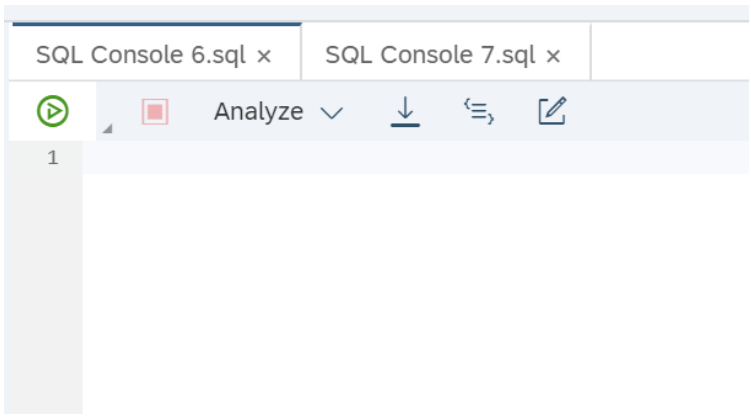
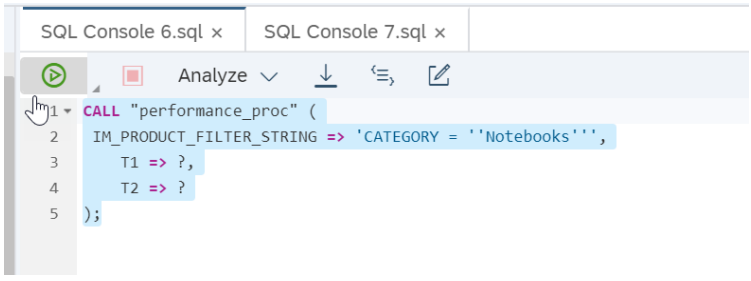
Result × Messages ×

Rows (16)

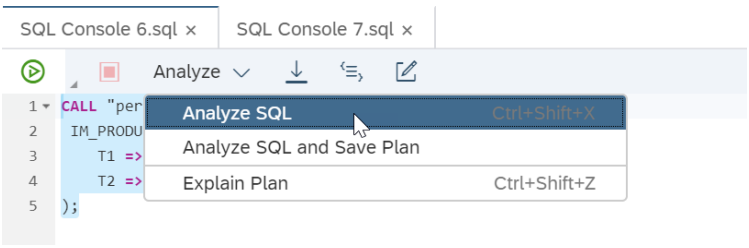
	EPTH	STATEMENT_COMPILE_TIME	STATEMENT_EXECUTION_TIME	STATEMENT_START_TIME	STATEMENT_END_TIME
1		238	5121094	2018-09-14 13:24:32.685912000	NULL
2		4372	69	2018-09-14 13:24:32.686902000	2018-09-14 13:24:32.691395000
3		5197	608	2018-09-14 13:24:32.691895000	2018-09-14 13:24:32.697930000
4		3826	6070	2018-09-14 13:24:32.698876000	2018-09-14 13:24:32.708809000
5		67385	5010865	2018-09-14 13:24:32.711097000	2018-09-14 13:24:37.789386000
6		67385	5010700	2018-09-14 13:24:32.711097000	2018-09-14 13:24:37.789221000
7		375	1580	2018-09-14 13:24:32.788699000	2018-09-14 13:24:32.790687000
8		196	137	2018-09-14 13:24:32.791727000	2018-09-14 13:24:32.792083000
9		151	170	2018-09-14 13:24:32.793109000	2018-09-14 13:24:32.793450000
10		3097	235	2018-09-14 13:24:32.794817000	2018-09-14 13:24:32.798183000
11		3093	984	2018-09-14 13:24:37.779313000	2018-09-14 13:24:37.783434000
12		3145	757	2018-09-14 13:24:37.779507000	2018-09-14 13:24:37.783434000
13		3209	150	2018-09-14 13:24:37.784963000	2018-09-14 13:24:37.788342000
14		2884	320	2018-09-14 13:24:37.785105000	2018-09-14 13:24:37.788349000
15		3857	1910	2018-09-14 13:24:37.790493000	2018-09-14 13:24:37.796299000
16		9745	417	2018-09-14 13:24:37.797708000	NULL

Exercise 2.13.8.3: Using SAP HANA Query Analysis Tool

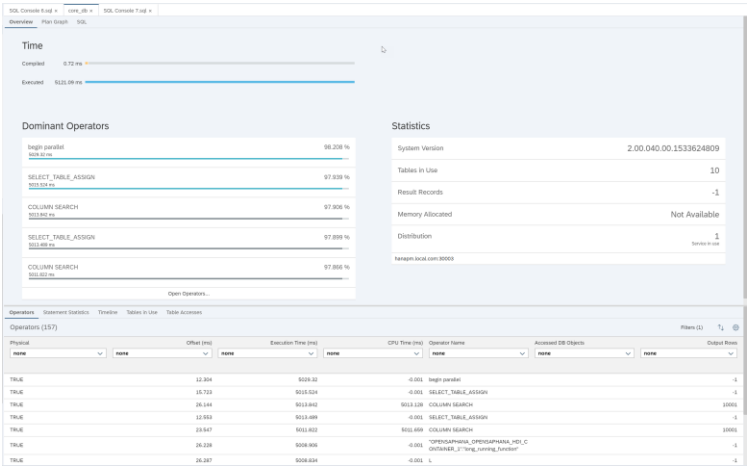
Besides M_ACTIVE_PROCEDURES you can also use the SQL Analyzer Tool. Among other things, the it can provide information on the execution plan of each SQL statement or display a timeline showing the parallel execution of statements.

Explanation	Screenshot
1. Clear out the SQL Console tab again.	
2. Enter the following CALL statement into the console. **Note:** The complete code can be found at the following URL. https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex2/performance_proc2.sql	`CALL "performance_proc" (` ` IM_PRODUCT_FILTER_STRING => 'CATEGORY = ''Notebooks''',` `  T1 => ?,` `  T2 => ?` `);`
3. Select the entire CALL statement.	

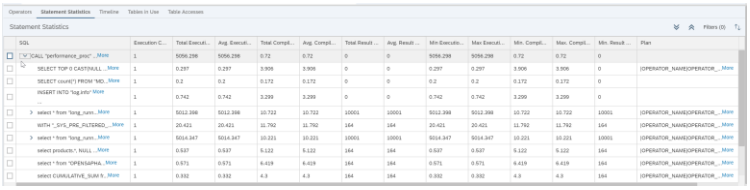
4. Click the **Analyze**/arrow next to the **Cancel** button and choose **Analyze SQL**.



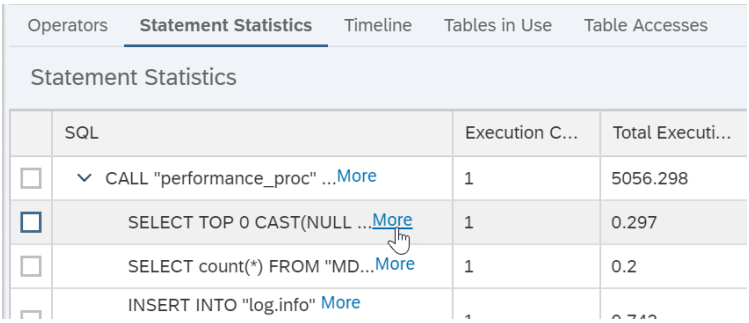
5. You will be presented with the following screen.

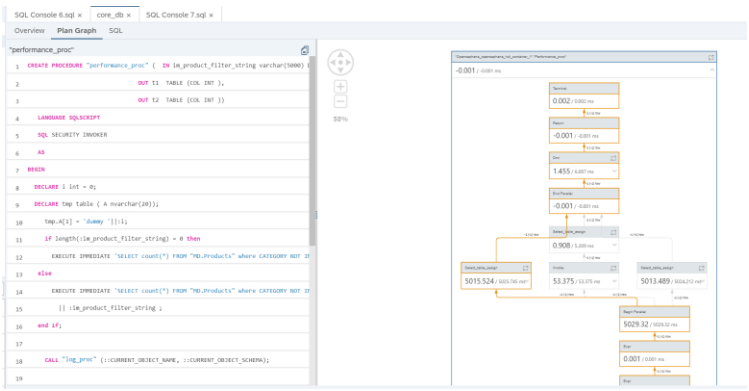
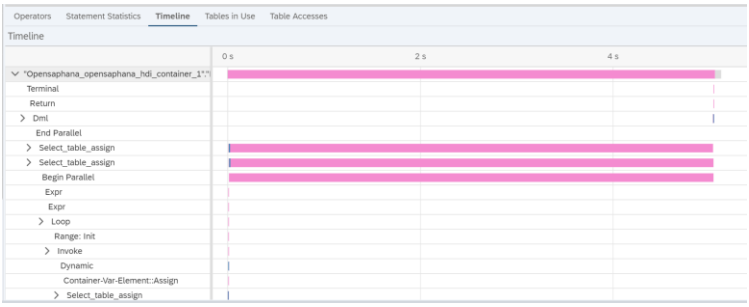


6. Click on the **Statement Statistics** tab, and expand the **CALL** statement.



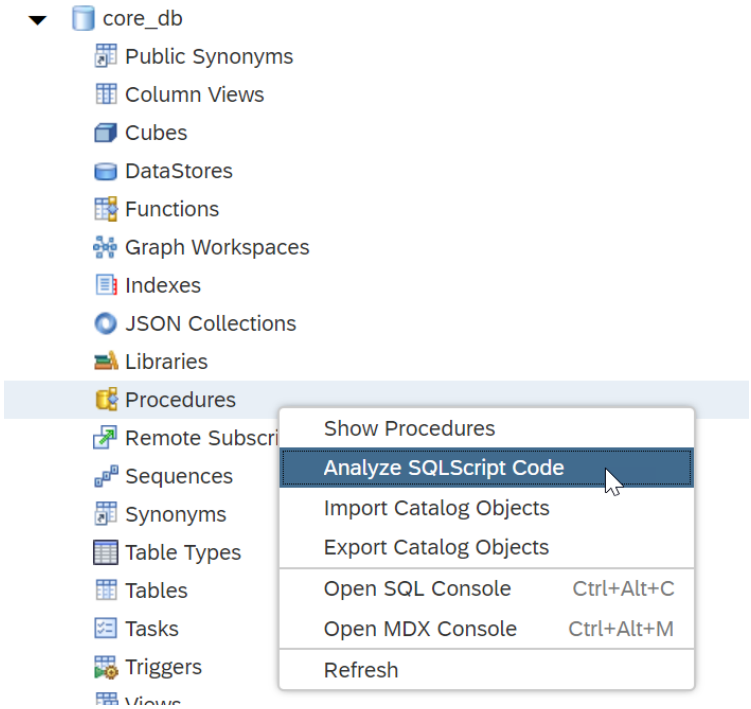
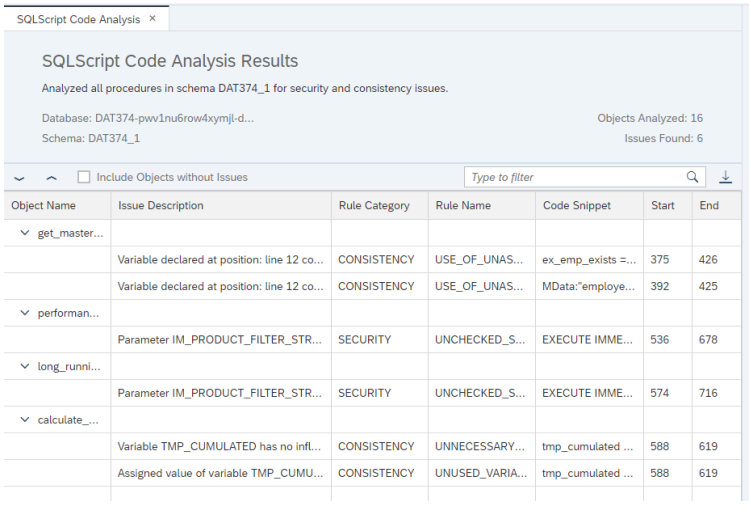
7. Click on the **More** button to see more details about each statement.

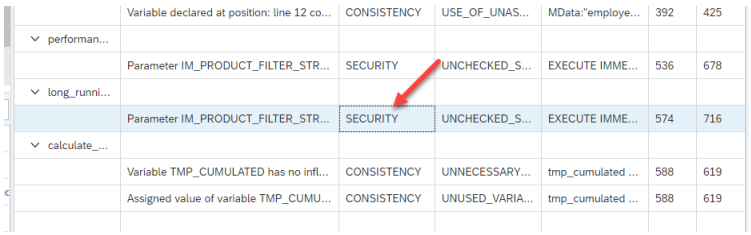


8. Click on the **Plan Graph** tab and you can view the plan in a graphical view. Expand each node to dive deeper into the execution path. You can use the zoom in/out button to adjust your view as well.	
9. Besides the plans, SQL Analyzer also provides a timeline that can give you information about the parallel execution of several statements. Click on the **Timeline** tab, and expand the CALL statement.	

Exercise 2.13.8.4: SQLScript Analyzer

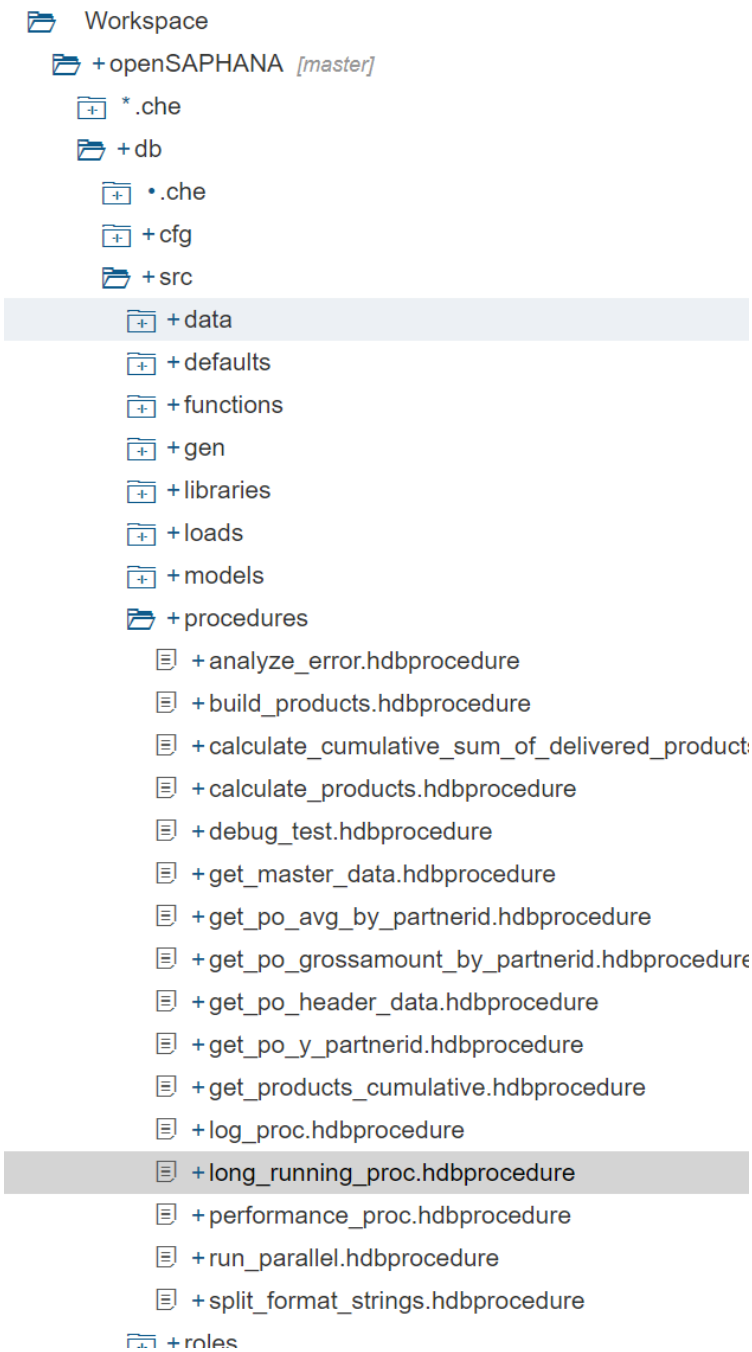
In this exercise, we will learn how to use the SQLScript Code Analyzer tool to scan our code for possible problems.

Explanation	Screenshot
1. From the **Database Explorer**, right-click on the procedures folder and choose **Analyze SQLScript Code**.	
2. The results of the SQLScript Code Analyzer will be shown on the right.	

3. Double click on any issue.	
4. You will notice that the analyzer will navigate to the procedure and flag the line with the issue.	

5. To fix this issue you can use one of the SQL Injection Prevention Functions., e.g. **ESCAPE_DOUBLE_QUOTES**

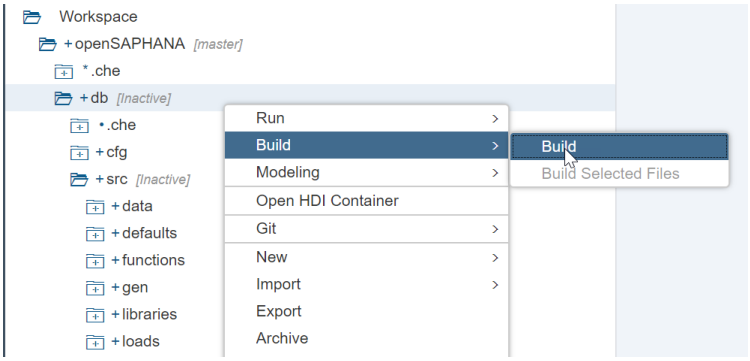
   Therefore switch to the SAP Web IDE for SAP HANA and open the file **long_running_proc.hdbprocedure**



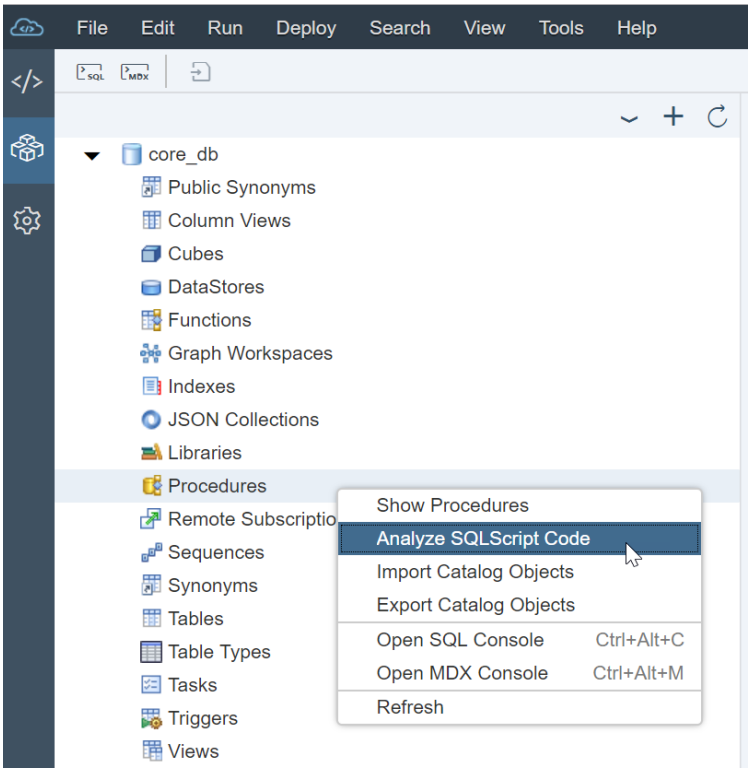
6. Change the dynamic SQL statement by adding the SQL Injection Prevention **ESCAPE_DOUBLE_QUOTES**

```
9   DECLARE i int = 0;
10  DECLARE tmp table ( A nvarchar(20));
11  tmp.A[1] = 'dummy '||:i;
12  if length(:im_product_filter_string) = 0 then
13      EXECUTE IMMEDIATE 'SELECT count(*) FROM "MD.Products" where CATEGORY
14  else
15      EXECUTE IMMEDIATE 'SELECT count(*) FROM "MD.Products" where CATEGORY
16        || escape_double_quotes(:im_product_filter_string) ;
17  end if;
18
```

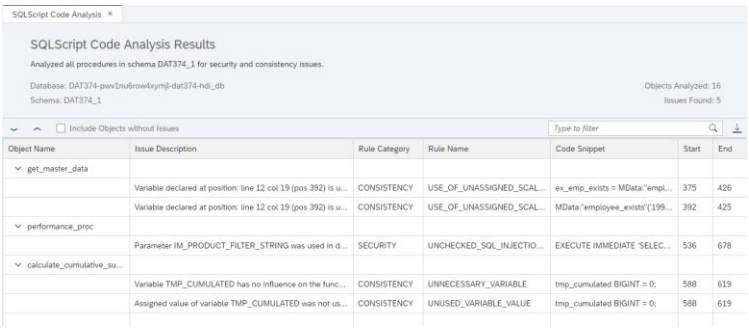
7. **Save** the procedure and perform a **Build.**



8. Return to the Database Explorer and run the SQLScript Code Analyzer again. Right-click on the procedures folder and choose **Analyze SQLScript Code**.



9. In the result tab you will see that one of the **SECURITY** issues is now gone



EXERCISE 3 –XSJS AND XSODATA SERVICES

Objective

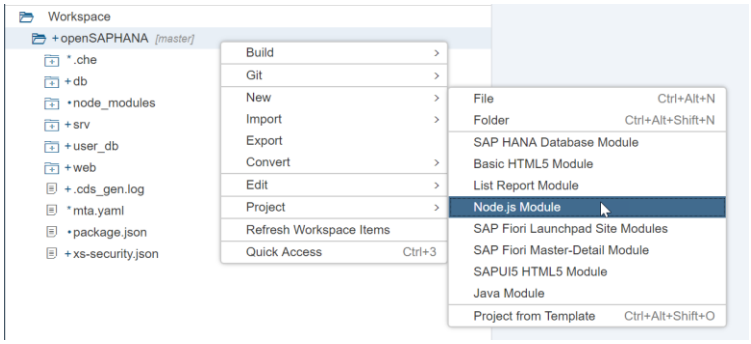
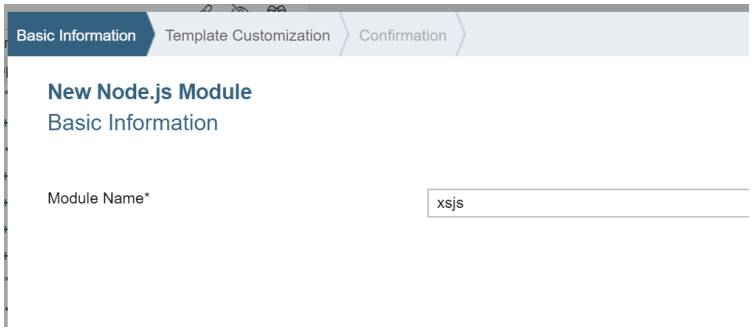
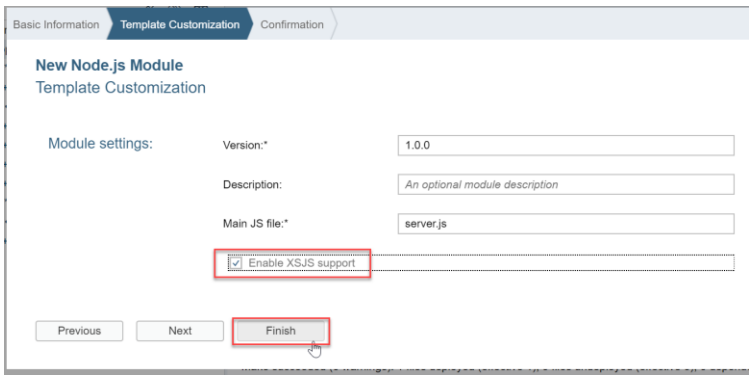
For this exercise we will now build the XSJS and XSODATA services used to expose our data model to the user interface. Although XS Advanced runs on node.js, SAP has added modules to node.js to provide XSJS and XSODATA backward compatibility. Therefore you can use the same programming model and much of the same APIs from XS, classic even within this new environment.

Exercise Description

- Node.js XSJS Bootstap
- XSJS Services
- XSODATA Services

EXERCISE 3 – SOLUTION

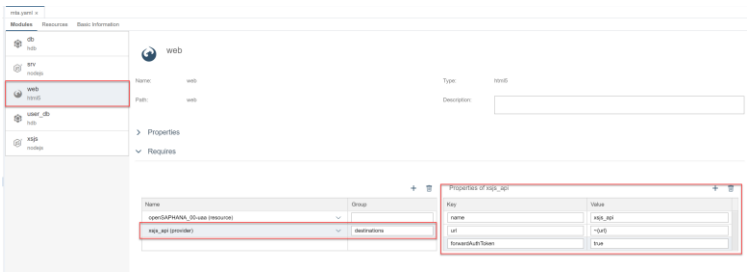
Exercise 3.1: XSJS and XSODATA

Explanation	Screenshot
1. We will start by creating a new module. **New-> Node.js Module**	
2. Name the module xsjs and press **Next**.	
3. Be sure to check the box **Enable XSJS support**. Then press **Finish**.	

4. Once again the mta.yaml file has been extended to add the new node.js module. Now we need to add the dependency from the web module to this new nodejs module and to add a destination route to it as well.

   The complete section for the **web** module should now look like this:

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_1



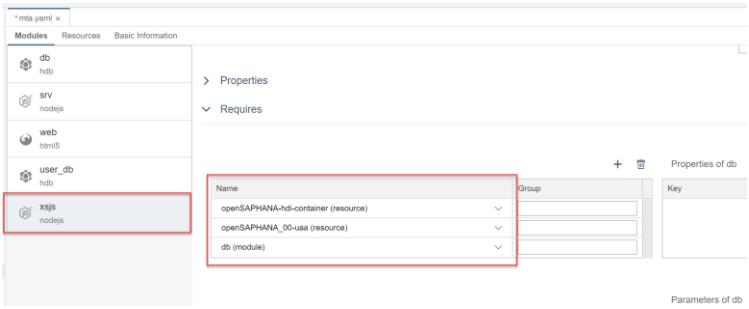
```
modules:
 - name: web
   type: html5
   path: web
   requires:
     - name: openSAPHANA6_00-uaa
     - name: xsjs_api
       group: destinations
       properties:
         name: xsjs_api
         url: '~{url}'
         forwardAuthToken: true
```

5. We also need to extend the new **xsjs** module that was just created. It will need a dependency to both the uaa and hdi resource. We also need to make sure the module is exposed for use in the destination route.

   The complete section for the **xsjs** module should now look like this:

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_2

   Later at deploy, the destination routing builds a dependency and navigation ability between the two services without ever having to hard code the URLs or ports. They are assigned at deploy time and all references are automatically updated.




```
 - name: xsjs
   type: nodejs
   path: xsjs
   provides:
     - name: xsjs_api
       properties:
         url: '${default-url}'
   requires:
     - name: openSAPHANA-hdi-container
     - name: openSAPHANA6_<group number>-uaa
     - name: db
```

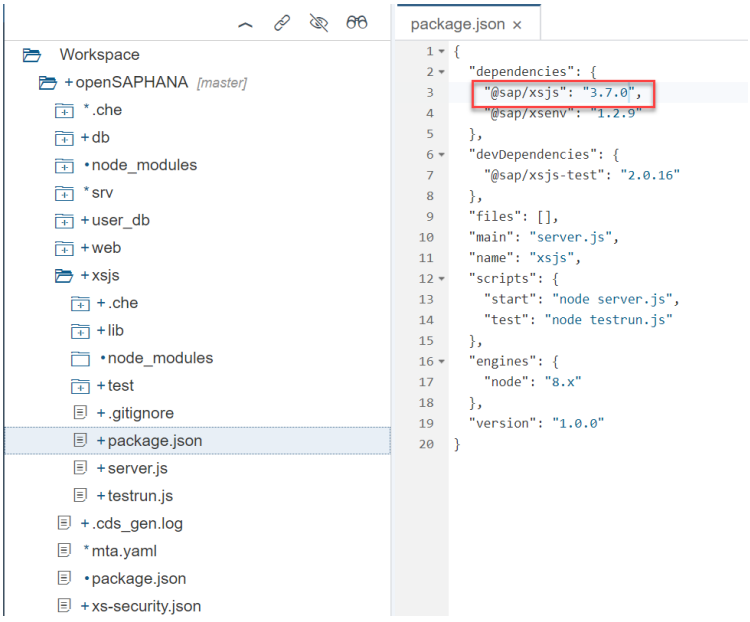
6. If you remember back to exercise 1 – we maintained the **xs-app.json** of the App Router (web module). Now we can add rules for redirecting certain requests to the web module into other modules in this project. This is where we are configuring that any file request with the extension .xsjs or .xsodata should be rerouted internally to the nodejs destination that we just defined in the mta.yaml.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_3

```
{
        "welcomeFile": "index.html",
        "authenticationMethod": "route",
        "routes": [{
                "source": "(.*)(.xsjs)",
                "destination": "xsjs_api",
                "csrfProtection": false,
                "authenticationType": "xsuaa"
        }, {
                "source": "(.*)(.xsodata)",
                "destination": "xsjs_api",
                "authenticationType": "xsuaa"
        }]
}
```

7. Return to the **xsjs** folder that we created in this exercise. This modules starts with a package.json file. Unlike the web or HDB modules that also use a package.json, the startup script is not an SAP provided central node application, but one that we've created via the module creation wizard – **server.js**

   IMPORTANT: Please update the version of the @sap/xsjs module to 3.7.0.



8. This **server.js** is the node.js bootstap for XSJS compatibility mode. It uses the SAP provided xsjs module and starts it with a few basic parameters. However remember all the HANA database connectivity options come from the HDI container

```
/*eslint no-console: 0, no-unused-vars: 0*/
"use strict";

var xsjs  = require("@sap/xsjs");
var xsenv = require("@sap/xsenv");
var port  = process.env.PORT || 3000;

var options = {
//          anonymous : true, // remove to authenticate calls
```

which we bound to this service via the **mta.yaml** file.

We want to make a few changes to what the wizard generated. We want authentication on our service, so comment out the **anonymous: true** line.

The complete implementation of server.js should look like this now:

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_4

```
        redirectUrl : "/index.xsjs"
};

// configure HANA
try {
        options = Object.assign(options, xsenv.getServices({
hana: {tag: "hana"} }));
} catch (err) {
        console.log("[WARN]", err.message);
}

// configure UAA
try {
        options = Object.assign(options, xsenv.getServices({
uaa: {tag: "xsuaa"} }));
} catch (err) {
        console.log("[WARN]", err.message);
}

// start server
xsjs(options).listen(port);

console.log("Server listening on port %d", port);
```

9. Under **xsjs** in the **lib** folder, **create** a sub-folder called xsodata. Create a file named **purchaseOrder.xsodata**. Here is the source code for this file.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_5

   Here we expose both the Header and Item tables from our HDI container as separate entities and build a navigation association between the two.

   Note: If you receive a syntax error on Line 2, you can ignore this. It is a bogus error because our target tables do not use Namespaces.

```
service {
   "PurchaseOrder.Header"
          as "POHeader" navigates ("Items" as "POItem");

        "PurchaseOrder.Item"
          as "POItem";

        association "Items" principal
"POHeader"("PURCHASEORDERID")
        multiplicity "1" dependent
"POItem"("POHeader.PURCHASEORDERID") multiplicity "*";
}
```

10. In the lib folder, create a sub-folder called **xsjs**. Create a file named **hdb.xsjs**. Here is the source code for this file.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_6

    This logic reads data from our item table and sends it to the frontend as JSON.

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/
/*eslint-env node, es6 */
"use strict";

var conn = $.hdb.getConnection();
var query =
        `SELECT FROM
OPENSAP_PURCHASEORDER_ITEMVIEW {
                PURCHASEORDERID as "PurchaseOrderId",
        PRODUCT as "ProductID",
        GROSSAMOUNT as "Amount"
        } `;
var rs = conn.executeQuery(query);
$.response.setBody(JSON.stringify(rs));
$.response.contentType = "application/json";
$.response.status = $.net.http.OK;
```

11. Create a second file named **exercisesMaster.xsjs**. Here is the source code for this file.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_7

    It sends the current user and language back to the client for filling in the header of the UI.

```
/*eslint no-console: 0, no-unused-vars: 0, dot-notation: 0*/
/*eslint-env node, es6 */
"use strict";

function fillSessionInfo(){
        var body = "";
        body = JSON.stringify({
                "session" : [{"UserName":
$.session.getUsername(), "Language": $.session.language}]
        });
        $.response.contentType = "application/json";
        $.response.setBody(body);
        $.response.status = $.net.http.OK;
}


var aCmd = $.request.parameters.get("cmd");
switch (aCmd) {
case "getSessionInfo":
        fillSessionInfo();
        break;
default:
        $.response.status =
$.net.http.INTERNAL_SERVER_ERROR;
        $.response.setBody("Invalid Request Method");
}
```

12. Create a third file named **csrf.xsjs**. This is an empty file which we can use to request a CSRF token for update/insert/delete operations.

13. Create a forth file named **procedures.xsjs**. This example shows you how to call a stored procedure from XSJS.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_8

```
/*eslint no-console: 0, no-unused-vars: 0, dot-notation: 0, no-use-before-define: 0*/
/*eslint-env node, es6 */
"use strict";

/**
@function JSON as returned by hdb
*/
function hdbDirectTest() {
	var results = _selection();
	//Pass output to response
	$.response.status = $.net.http.OK;
	$.response.contentType = "application/json";
	$.response.setBody(JSON.stringify(results));

}

/**
@function Flattended JSON structure
*/
function hdbFlattenedTest() {
	outputJSON(_selection().EX_TOP_3_EMP_PO_COMBINED_CNT);
}

/**
@function load/call the procedure
*/
function _selection() {
	var connection = $.hdb.getConnection();

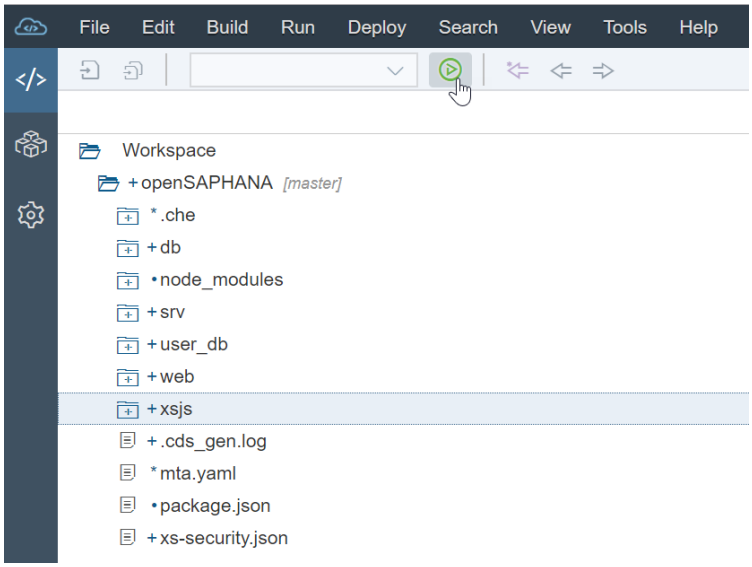
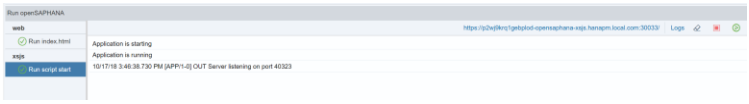
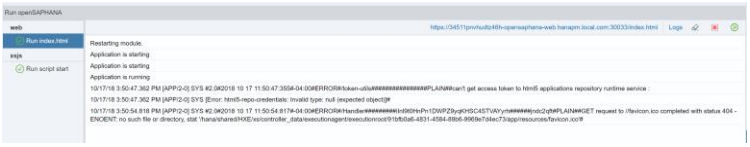
	var getPOHeaderData = connection.loadProcedure(
		"get_po_header_data");

	var results = getPOHeaderData();
	return results;
}

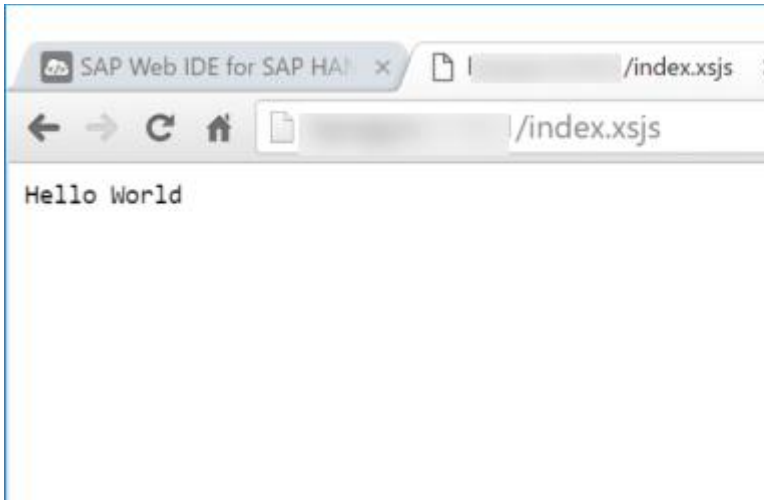
/**
@function Puts a JSON object into the Response Object
@param {object} jsonOut - JSON Object
*/
function outputJSON(jsonOut) {
	var out = [];
	for (let item of jsonOut){
		out.push(item);
	}
	$.response.status = $.net.http.OK;
	$.response.contentType = "application/json";
	$.response.setBody(JSON.stringify(out));
}

var aCmd = $.request.parameters.get("cmd");
switch (aCmd) {
```

	case "direct": hdbDirectTest(); break; case "flattened": hdbFlattenedTest(); break; default: hdbDirectTest(); break; }
14. Create a 5th file called **os.xsjs**. This example shows you how you can call to Node.js from XSJS. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_9	/*eslint no-console: 0, no-unused-vars: 0*/ /*eslint-env node, es6 */ "use strict"; var os = $.require("os"); var output = {}; output.tmpdir = os.tmpdir(); output.endianness = os.endianness(); output.hostname = os.hostname(); output.type = os.type(); output.platform = os.platform(); output.arch = os.arch(); output.release = os.release(); output.uptime = os.uptime(); output.loadavg = os.loadavg(); output.totalmem = os.totalmem(); output.freemem = os.freemem(); output.cpus = os.cpus(); output.networkInfraces = os.networkInterfaces(); $.response.status = $.net.http.OK; $.response.contentType = "application/json"; $.response.setBody(JSON.stringify(output));
15. Create another file called **whoAmI.xsjs**. This example shows how we now use a different user at the DB and XS layers, but also how the XS user (Application User) can be accessed from within the DB by reading the Session Context. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/ /*eslint-env node, es6 */ var connection = $.hdb.getConnection(); var query = `SELECT CURRENT_USER FROM "DUMMY"`; var rs = connection.executeQuery(query); var currentUser = rs[0].CURRENT_USER; query = `SELECT SESSION_CONTEXT('APPLICATIONUSER') "APPLICATION_USER" FROM "DUMMY"`; rs = connection.executeQuery(query); var applicationUser = rs[0].APPLICATION_USER;

hana7/blob/snippets_2.3.2/ex3/ex3_9a	var greeting = `XS Layer Session User: ${$.session.getUsername()} </br>Database Current User: ${currentUser} </br> Database Application User: ${applicationUser} </br>Welcome to HANA `; $.response.contentType = "text/html; charset=utf-8"; $.response.setBody(greeting);
16. Save any unsaved/open files. We can now run the **xsjs** module.	
17. You should see that the build and deploy was successful. The initial build can take a few minutes.	
18. Run the **web** module. It will need to rebuild and redeploy due to the added dependency to the xsjs module.	

19. In the running tab, you should see the index.html from earlier. We can add the url to our xsjs service **/index.xsjs** in the browser. You will see that our xsjs service is accessible via the HTML5 module runtime. The HTML5 module functions as a proxy and performs the routing to the other service internally.

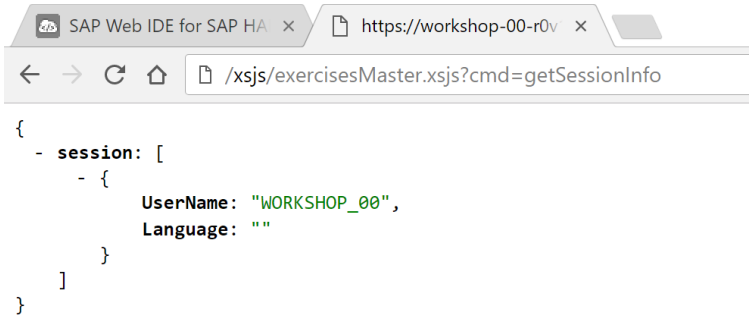


20. **/xsjs/hdb.xsjs** reads data from our Purchase Order table and exports it as a JSON string (formatted on the client side by a browser plug-in).

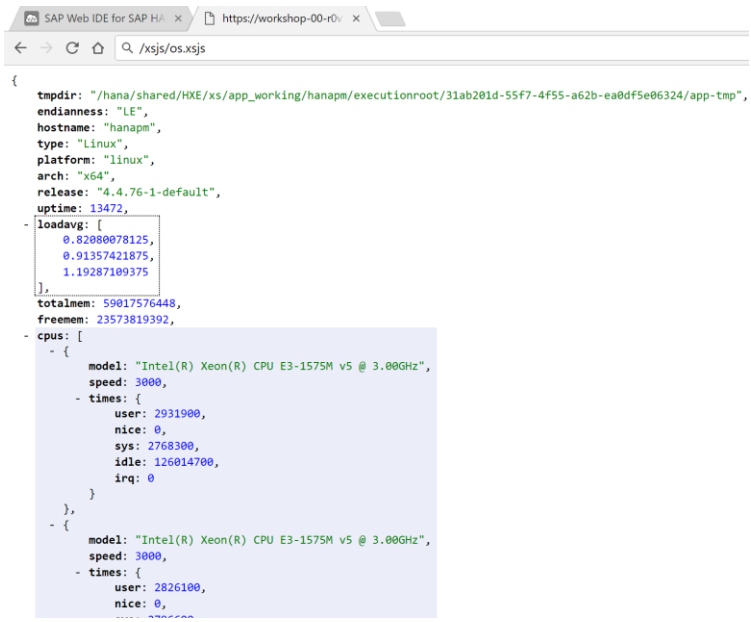
    Feel free to test the other example xsjs files we created in this exercise as well.



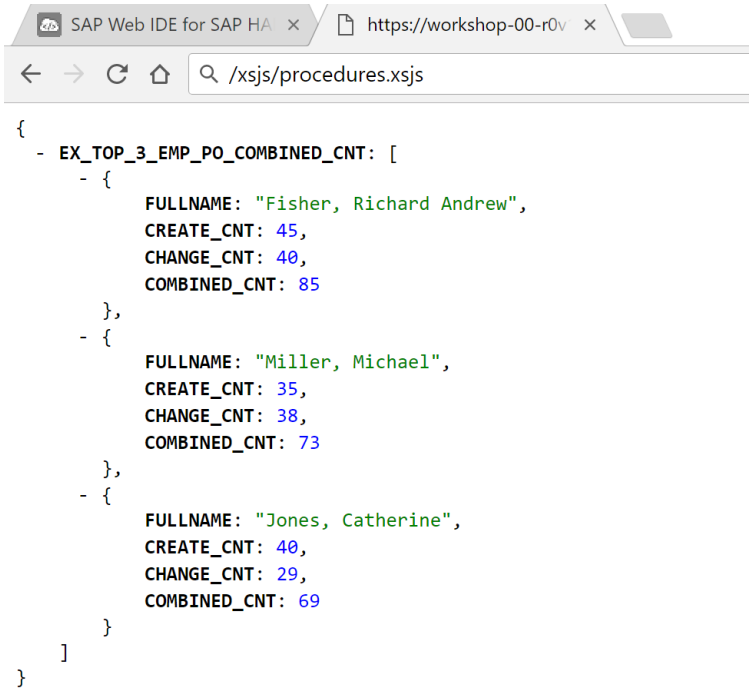
21. **/xsjs/exercisesMaster.xsjs** shows the typical approach to have multiple REST actions in one XSJS. Change the URL to **/xsjs/exercisesMaster.xsjs?cmd=getSessionInfo**. You should see your User Name returned from the XSJS session layer.



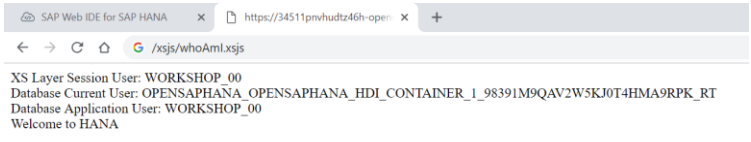
22. **/xsjs/os.xsjs** shows the use of node.js modules and returns operating system information from the XSA server.



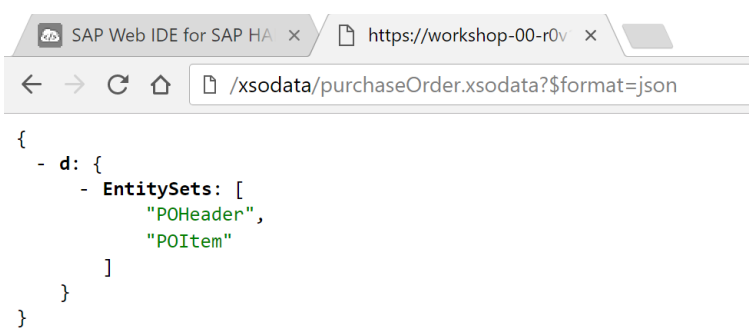
23. **/xsjs/procedures.xsjs** demonstrates the calling of a stored procedure from XSJS and returning the results as JSON.



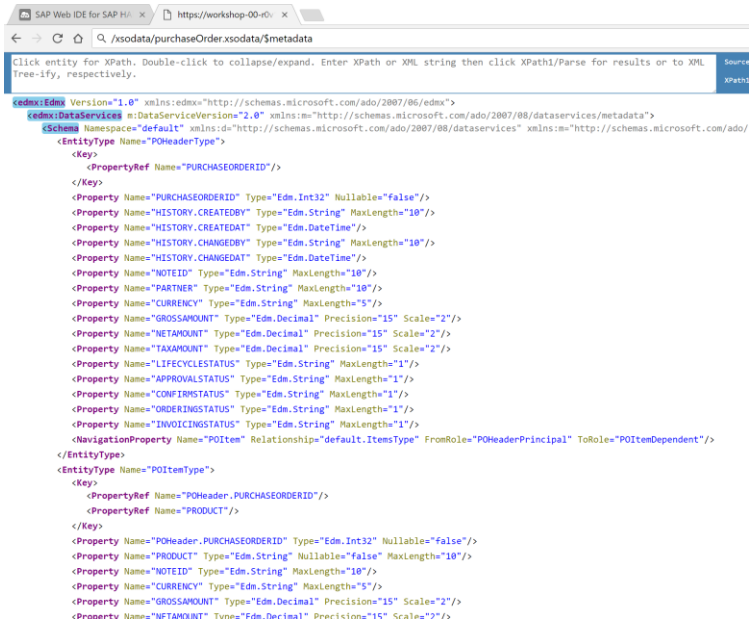
24. **/xsjs/whoAmI.xsjs** demonstrates the separation between application user and database user as well as how the application user can be accessed from both the XSJS and Database layers.



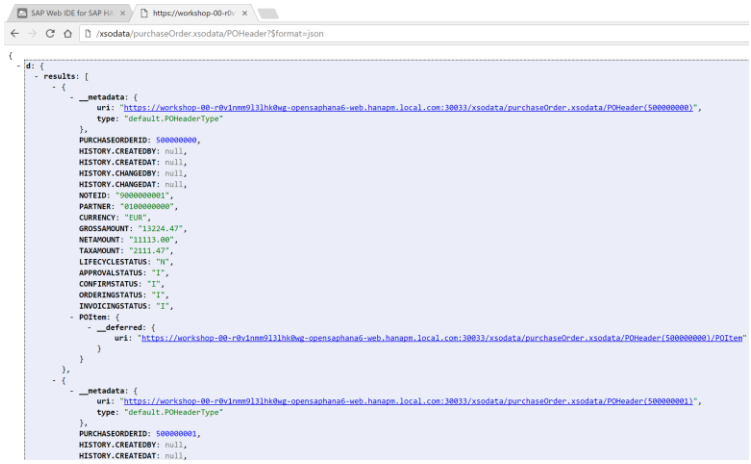
25. **/xsodata/purchaseOrder.xsodata?$format=json** gives you access to a full OData service for the Purchase Order header and item tables we created in the previous exercise.



26. We can also see the metadata about the OData interface we created by changing the URL to **/xsodata/purchaseOrder.xsodata/$metadata**



27. Change the URL again to point to the **POHeader** entity and we will see the data details. **/xsodata/purchaseOrder.xsodata/POHeader?$format=json**

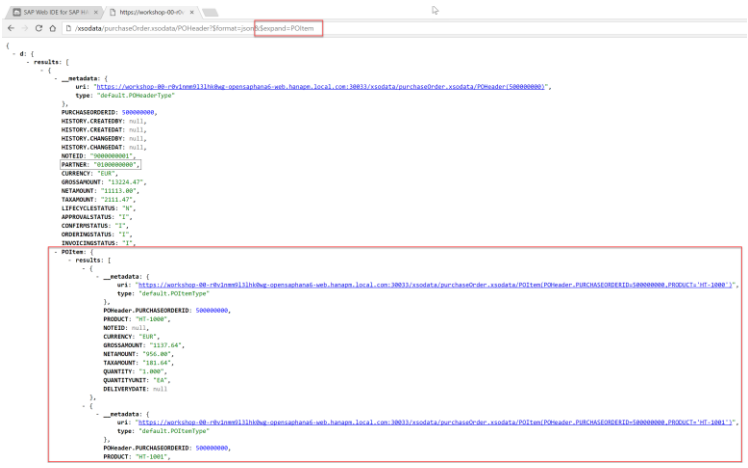


28. Associations can be an excellent way to load child elements on demand; however, there is also an option to expand the children details in place so that all levels can be retrieved with one request.

    Test the service again using the same steps as in the previous section of this exercise.

    This time add **&$expand=POItem** to the end of the URL.

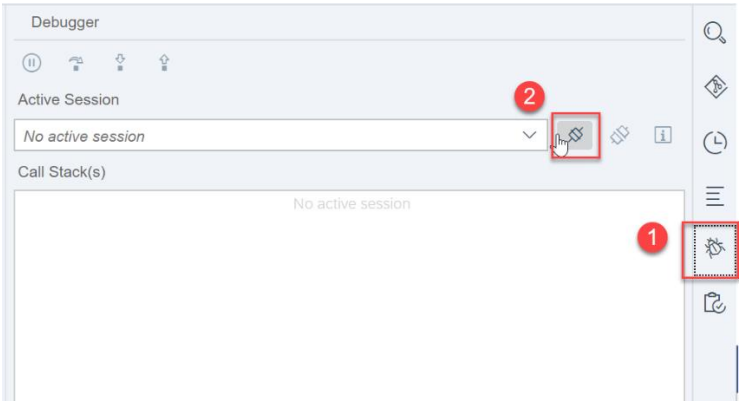
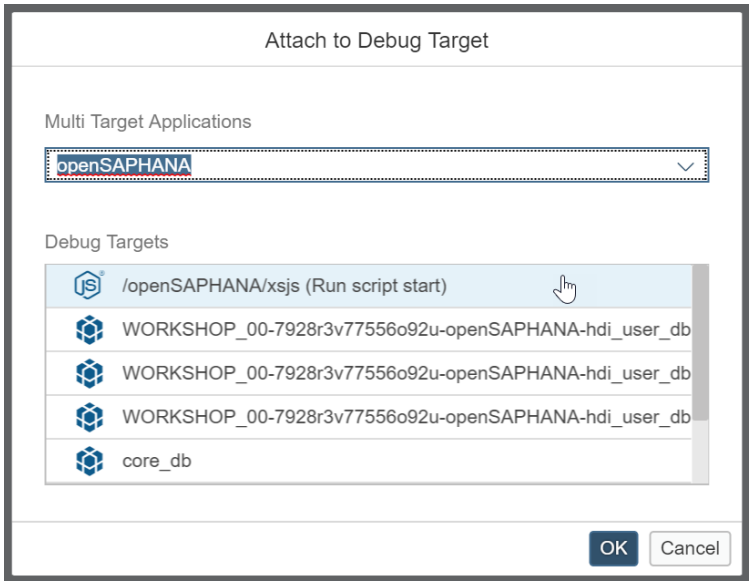
    You will then see that all the items are embedded within each header record.

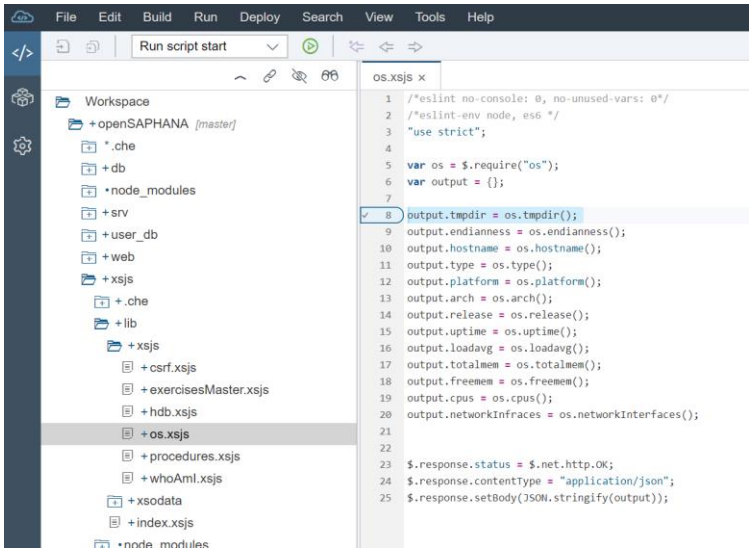
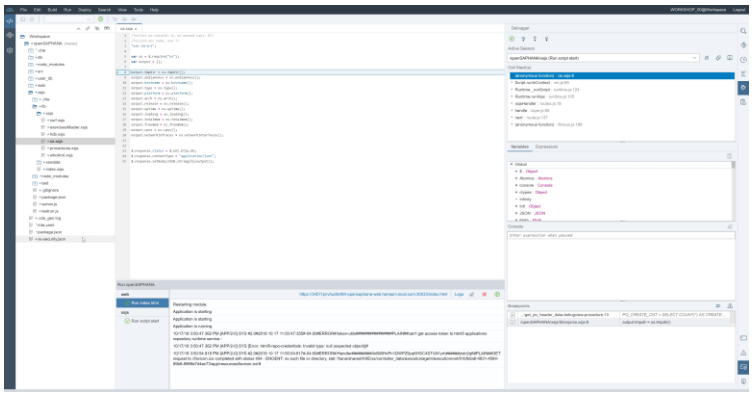
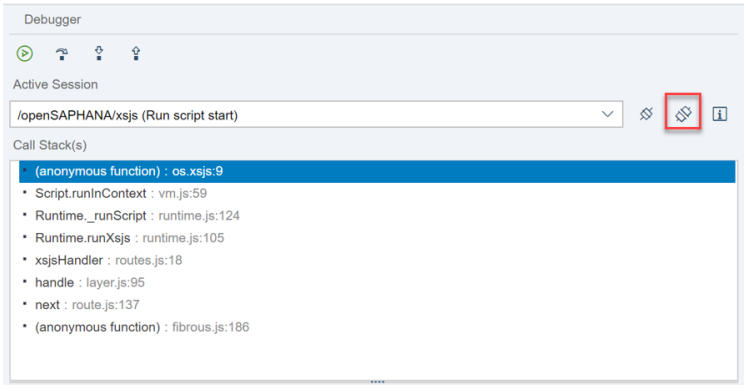


29. Finally, we can debug XSODATA requests if we add the URL parameter **&sap-ds-debug=html**. We can use this to see internal request, response, processing time, or even the generated SQL statements.

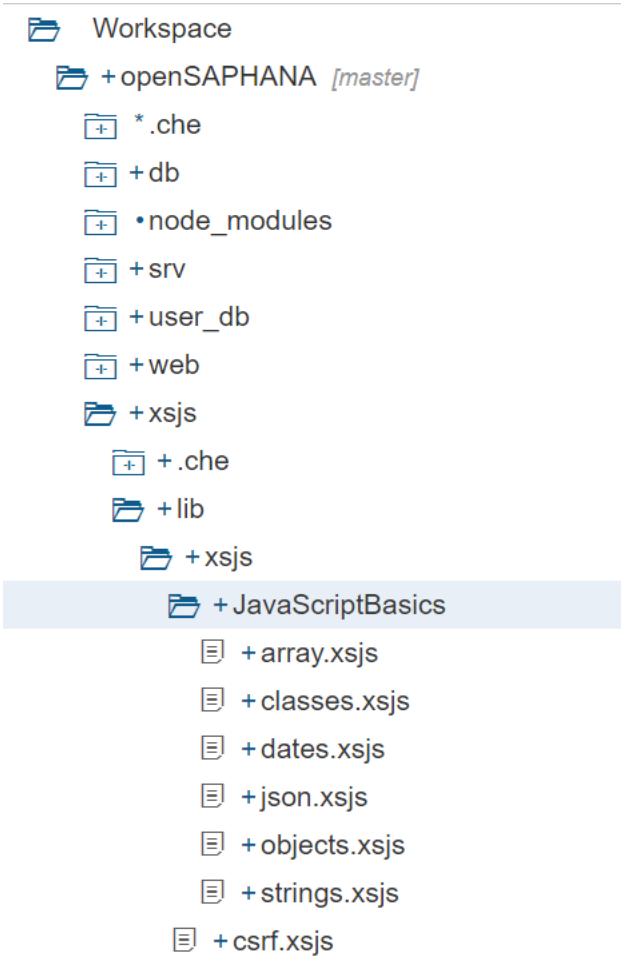
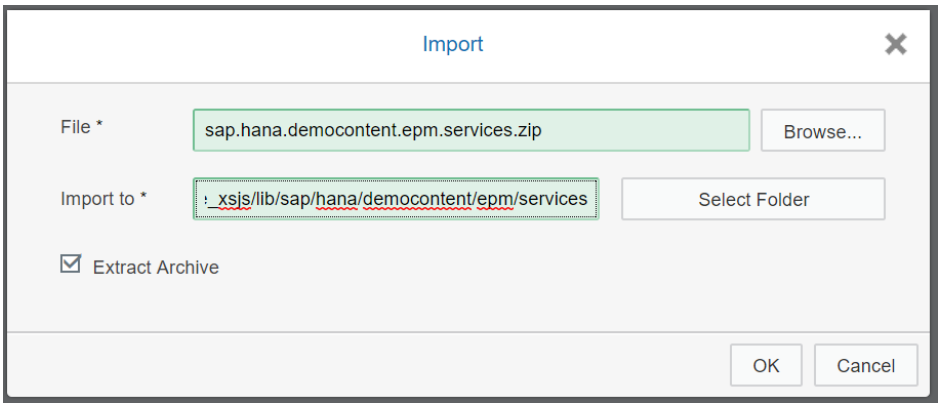


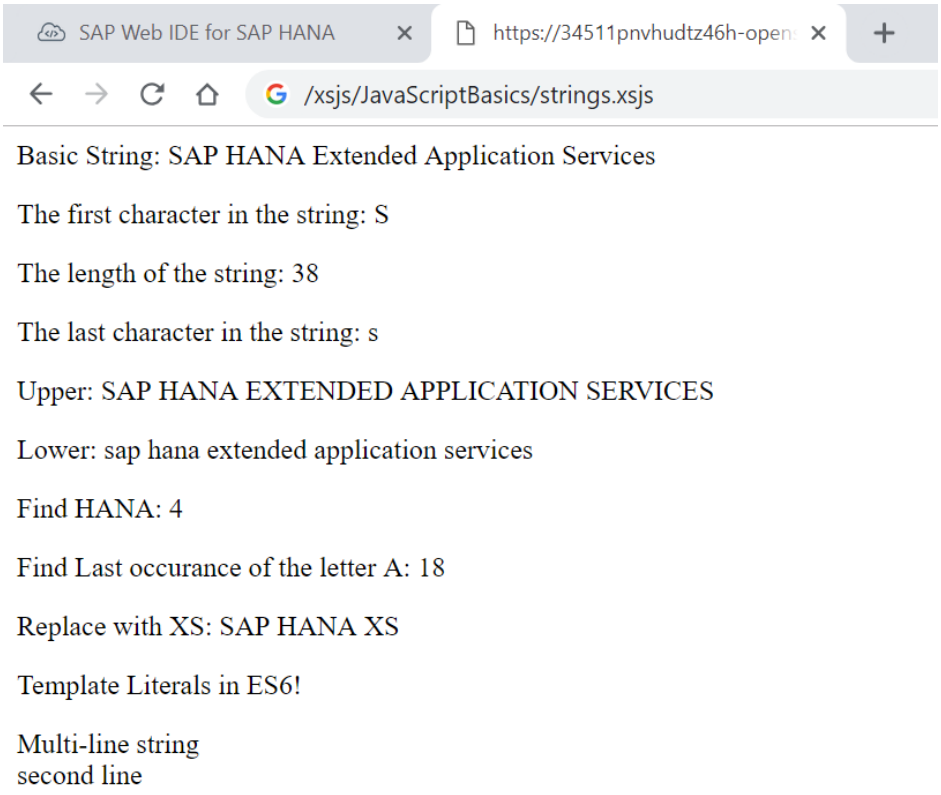
Exercise 3.2: Debugging XSJS or Node.js

Explanation	Screenshot
1. Return to the SAP Web IDE for SAP HANA. Click the debug link in the right side of the SAP Web IDE for SAP HANA. Then click the **Attach Debugger** button.	
2. Confirm the selection of the **xsjs** module to debug and then press **OK**.	

3. Click in the left side of the editor of the **os.xsjs** file to set a breakpoint on line 8.	
4. Switch the browser window to the where the web module is running and change the url path to **/xsjs/os.xsjs**	
5. The debugger should attach and open in the editor.	
6. You can now spend a moment exploring the debugger. Step over some statemetns. Try adding the output vaiable to the Console. When you are finished, you can detach the debugger by clicking the **Detach Debugger** button.	

Exercise 3.3: Exploring JavaScript Language Features

Explanation	Screenshot
1. This exercise will be a bit different from all others. Instead of following along step-by-step or working toward a very specific final product; this exercise will instead allow you to explore various JavaScript language features. There is no strict right or wrong solution to many of these exercise parts. However you can import sample solutions from: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex3/JavaScriptBasics.zip Feel free to create similar xsjs files in order to be able explore these language features. However the main point of this exercise is to focus on general JavaScript language constructs and not anything XSJS or HANA specific.	
2. You should also import the supporting xsjs libraries from here: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex3/sap.hana.democontent.epm.services.zip Upon import change the dots in sap.hana.democontent.epm.services to forward slashes to create the necessary folder structure to import the files to	

xsjs/lib/sap/hana/democontent/epm/services.	
3. Create a string with some sample text in it. Create a string with some sample text in it (hint: =). Access the first character of the string. (hint: offsets are zero based and specified via []) Access the last character in the string (hint: slice). Output the length of the string (hint: length). Convert the string to all upper and all lower case (hint: toUpperCase). Find the position of a text part of the string (hint: indexOf). Find the last occurrence of a character in the string. Perform a find and replace operation on the string (hint: lastIndexOf). Sample implementation: **strings.xsjs**	

4. Create a Data object with the current date (hint: new Date()).

   Convert the data to various other formats (UTC, Local Data, Local Time, ISO, JSON (hint: to*String with various modifiers UTC, Date, LocalDate, LocaleTime, etc).

   Access the sub-parts of the date: Year, Month, Day, Hours, Minutes, and Seconds (hint: get* with modifiers for FullYear, Month, Day, Date, Time, Hours, Minutes, Seconds).

   Perform math on a date, adding 30 days to the current date (hint: setDate and simple math).

   Sample implementation: **dates.xsjs**



1) Create two arrays: one with the colors Red, Green, and Blue and one with Black, White, Orange, and Purple (hint: [“one”, “two, “three”]).

   Output the first array as a string (hint: toString).

   Access the first element by index (hint: zero based).

   Get the number of elements in the first array (hint: length).

   Get the index of Blue in the first array (hint: indexOf).

   Loop over the first array and output each element (hint: for(var i; i < length; i++).

   Combine the two arrays (hint: concat).

   Reverse Sort and then Sort Ascending (hint: reverse, sort).

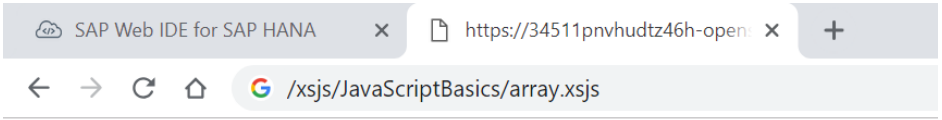
   Remove the last and then first element (hint: pop, shift).

   Slice out the 3rd and 4th element (hint: slice).

   Add two new colors (Malachite and Fallow) at the 3rd position (hint: splice).

   Add Brown to the start of the Array (hint: unshift).

   Sample implementation: **array.xsjs**



Complete Array: Red,Green,Blue

First Element: Red

Number of Elements: 3

Position of Blue: -1

Loop of Elements: Red Green Blue

Combined Array: Red,Green,Blue,Black, ,Orange,Purple

Reverse Sort: Purple,Orange, ,Black,Blue,Green,Red

Sort Ascending: Black,Blue,Green,Orange,Purple,Red,

Remove the last element: Black,Blue,Green,Orange,Purple,Red

Remove the first element: Blue,Green,Orange,Purple,Red

Slice out the 3rd and 4th element: Orange,Purple

Add two values at position 3: Blue,Green,Malachite,Fallow,Red

Add element to the beginning of the array: Brown,Blue,Green,Malachite,Fallow,Red

2) Select 10 records from **PO.Header** and convert the results to JSON (hint: $.hdb).

Loop over the elements in the JSON object and create a new **DISCOUNTAMOUNT** attribute by discounting the **GROSSAMOUNT** by 10%.

Convert the JSON object to a string and output it to the frontend (hint: JSON.stringify).

Sample implementation: **json.xsjs**



1) Create an object literal called colors. Give it three properties – red with the value #FF0000, green with the value #00FF00, and blue with the value #0000FF. Give the object a function named favoriteColor. On Monday, it should return blue, all other days it returns red (hint: var colors = {}).

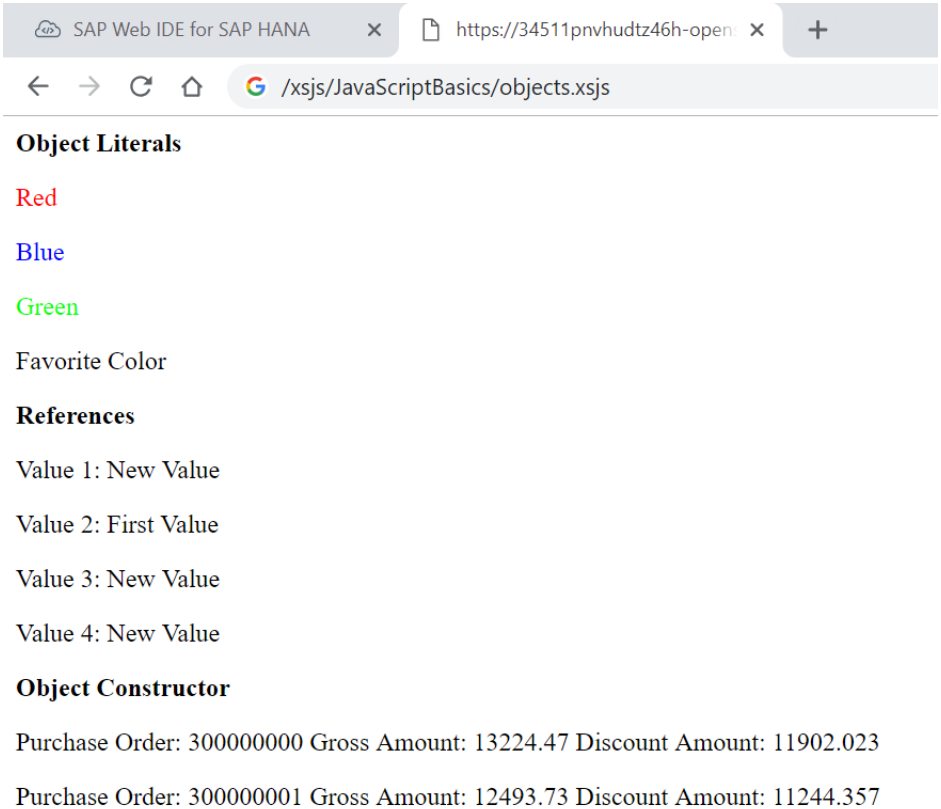
   Access the colors in the object including the Favorite color (hint: object.<variable>).

   Prove that objects are assign by reference and not assign by value (hint: just set a value to a regular variable and an object and see the difference).

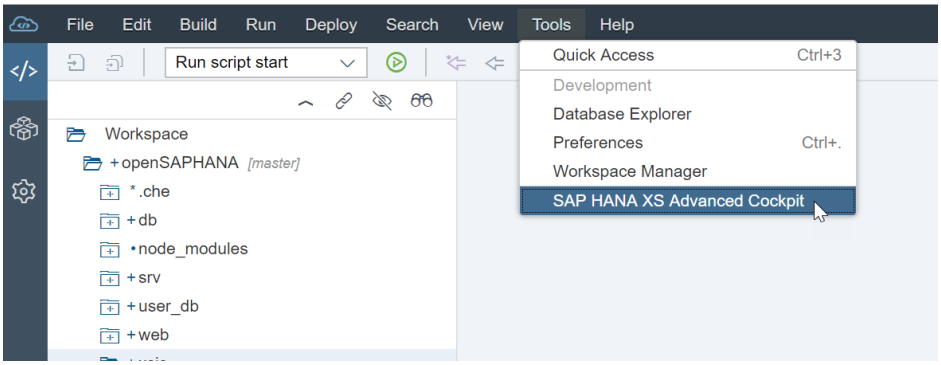
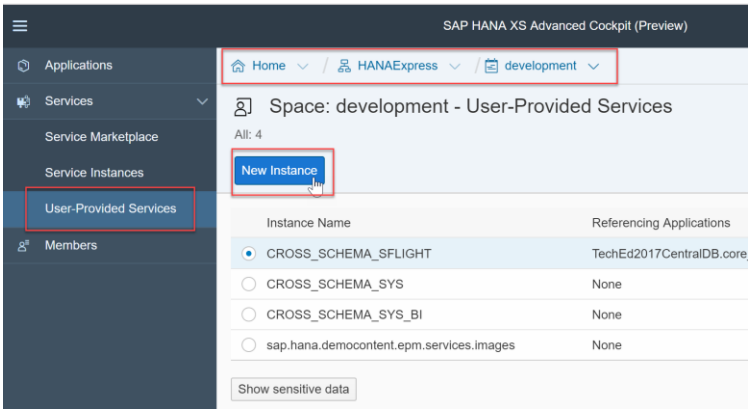
   Create a purchase order object for **PO.Header** which returns PO Id, gross amount, and a calculated discount amount (10% off gross amount) (hint: function with logic from last step – json.xsjs).

   Create two instances of this object – one for PO 300000000 and one for 300000001

   Sample implementation: **objects.xsjs**



Exericse 3.4: Creating an XSJS Service with outbound HTTP Connectivity

Explanation	Screenshot
1. XSJS services can also make outbound HTTP calls. In this step we will extend the previous service to make a request to the Library of Congress Image database and return the results to HANA.	
2. Launch the SAP HANA XS Advanced Cockpit via **Tools -> SAP HANA XS Advanced Cockpit**. Navigate to your organization **HANAExpress** and space **development**. Then select **User-Provided Services**. Click on **New Instance**	 

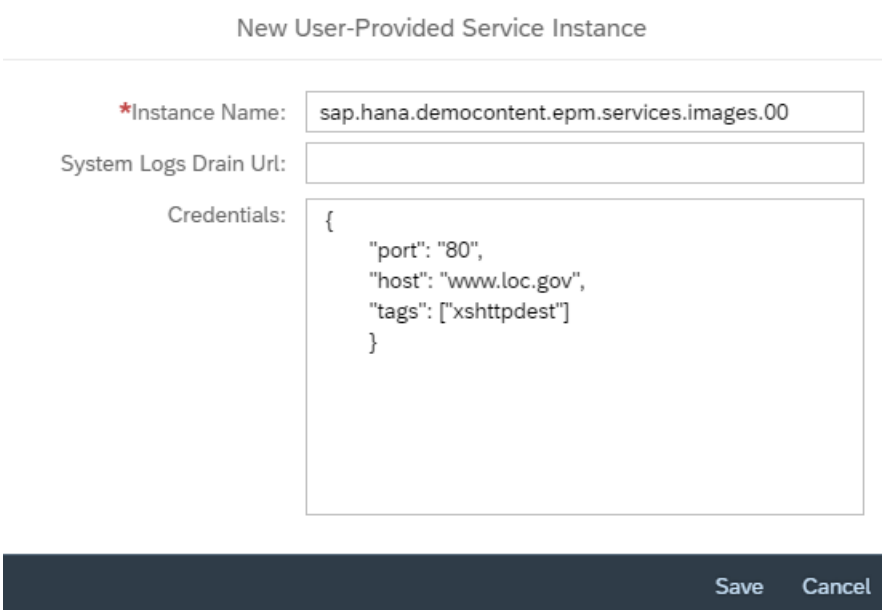
3. A User Provided Service allows our modules to connect to existing database schemas which aren't managed by HDI. This would be the case for replicated schemas or ERP ABAP schemas for instance.

   Create a service named:

   sap.hana.democontent.epm.services.images.<group number>

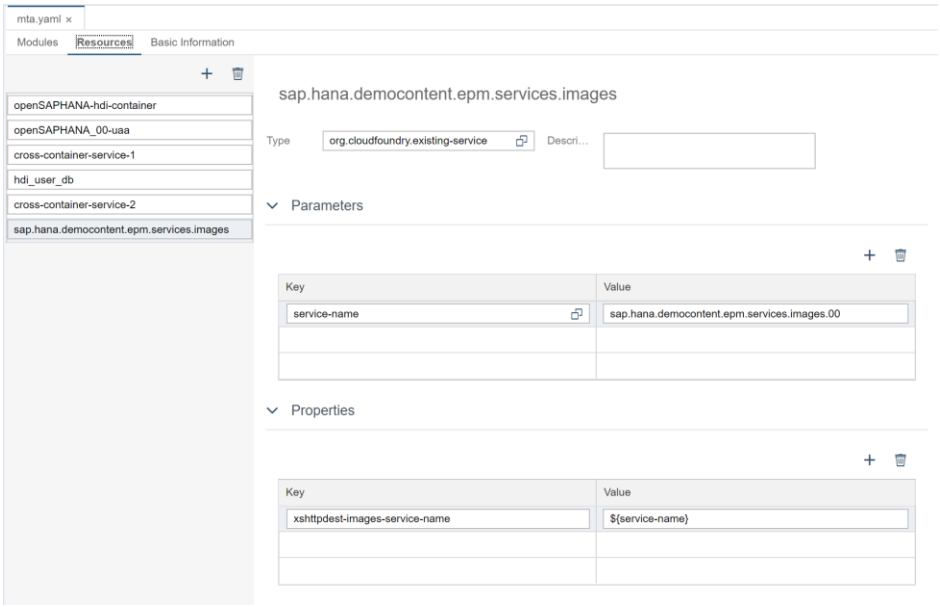
   With credentials:

   ```
   {
       "port": "80",
       "host": "www.loc.gov",
       "tags": ["xshttpdest"]
   }
   ```



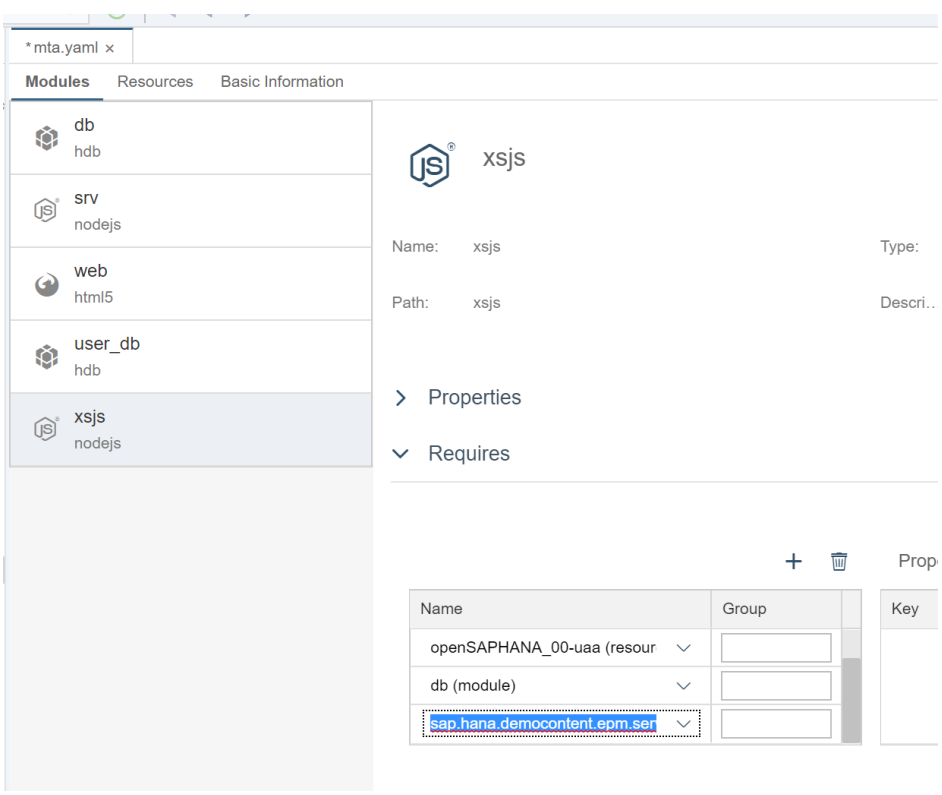
4) Add this service as a resource in the **mta.yaml**. Save the mta.yaml file. Switch to the Code Editor and then back to the MTA Editor to refresh the value helps.

   ```
    - name: sap.hana.democontent.epm.services.images
      type: org.cloudfoundry.existing-service
      parameters:
        service-name: sap.hana.democontent.epm.services.images.00
      properties:
        xshttpdest-images-service-name: '${service-name}'
   ```



5) Add add this resources as a dependency of the **xsjs** module in the mta.yaml

```
- name: core_xsjs
  type: nodejs
  path: core_xsjs
  provides:
    - name: core_xsjs_api
      properties:
        url: '${default-url}'
  requires:
    - name: hdi_core_db
    - name:
openSAPHANA6_00-uaa
    - name: core_db
    - name:
sap.hana.democontent.epm.services.images
```



6) In exercise 3.3 you should have imported the **/lib/sap/hana/democontent/epm/services** folder. It already contains an xsjs named outboundTest.xsjs.

You will have to add your group number to the end of images in this **readDestination** line to match up to the service you created earlier.

There is a function to the service to call the outbound HTTP connection. Take in to additional URL parameters for the remote search term and the image index in the search results.

```
function searchImages() {
    var search = $.request.parameters.get("search");
    var index = $.request.parameters.get("index");
    if (index === undefined) {
        index = 0;
    }
    var dest = $.net.http.readDestination("sap.hana.democontent.epm.services", "images");

    var client = new $.net.http.Client();
    var req = new $.web.WebRequest($.net.http.GET, search);
    client.request(req, dest);

    var response = client.getResponse();

    var body = null;
    if (response.body) {
        body = response.body.asString();
    }
    $.response.status = response.status;

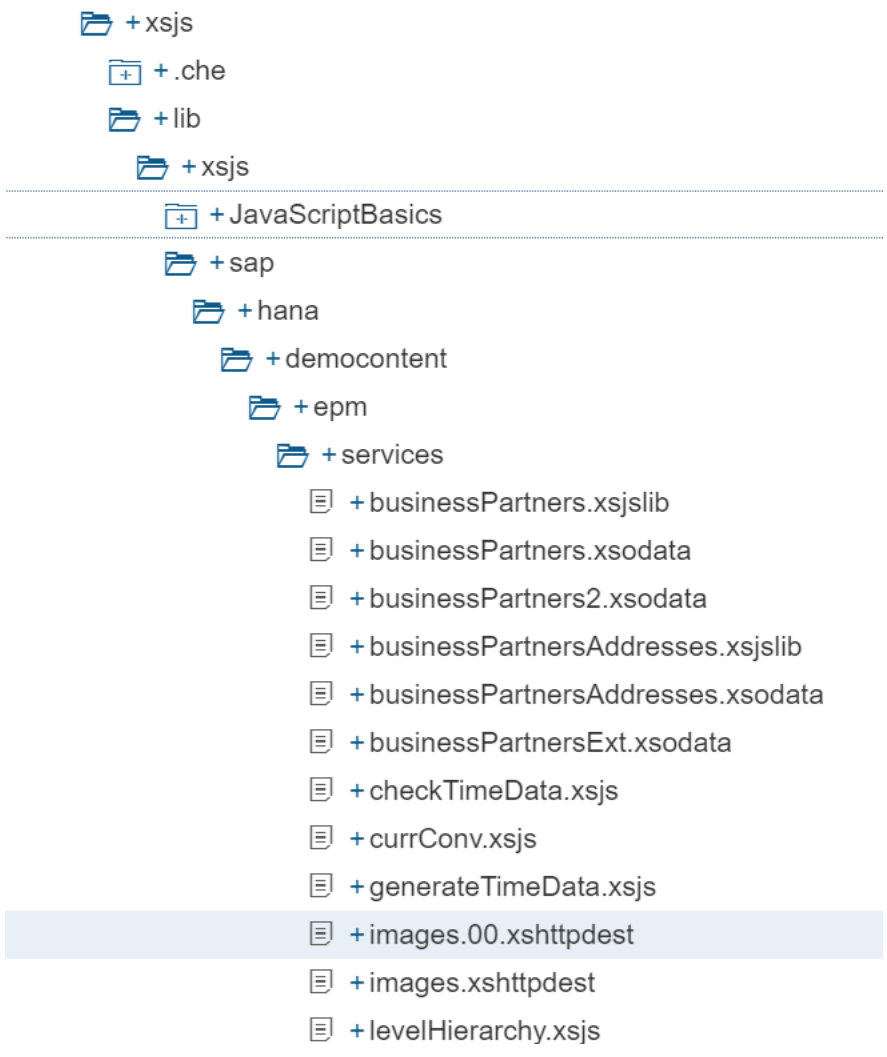
    if (response.status === $.net.http.INTERNAL_SERVER_ERROR) {
        $.response.contentType = "application/json";
        $.response.setBody("body");
    } else {
        $.response.contentType = "text/html";
        var searchDet = JSON.parse(body);
        var outBody =
            "First Result of " + encodeURIComponent(searchDet.search.hits) + "</br>" +
            "<img src=\"" + encodeURI(searchDet.results[index].image.full) + "\">";
        $.response.setBody(outBody);
    }
}
```

7) There is also the entry point logic.

```
var aCmd = $.request.parameters.get("cmd");
switch (aCmd) {
    case "Images":
        searchImages();
        break;
    default:
        $.response.status = $.net.http.BAD_REQUEST;
        $.response.setBody(MESSAGES.getMessage("SEPM_ADMIN", "002", encodeURI(aCmd)));
}
```

8) Copy the file **images.xshttpdest** to **images.<group number>.xshttpdest**.

Re-run the **xsjs** service to pickup the new user provided service resource.



9) Test the application in your browser using the “Run” button. Once again you will need to adjust the service to add the cmd and the search string.

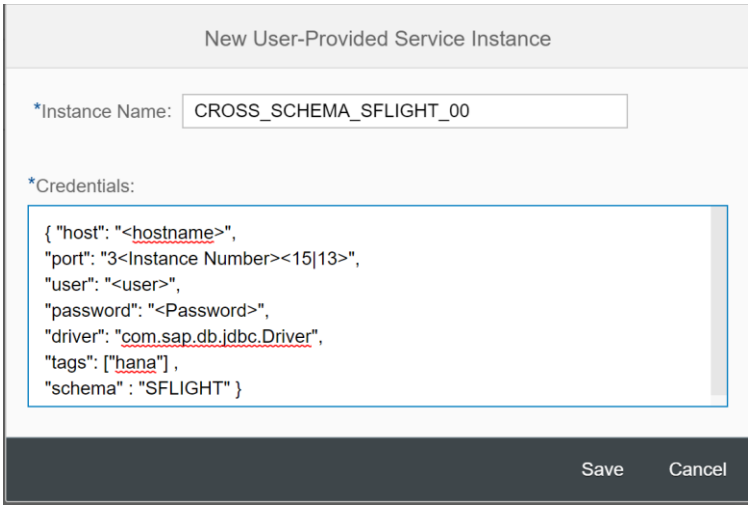
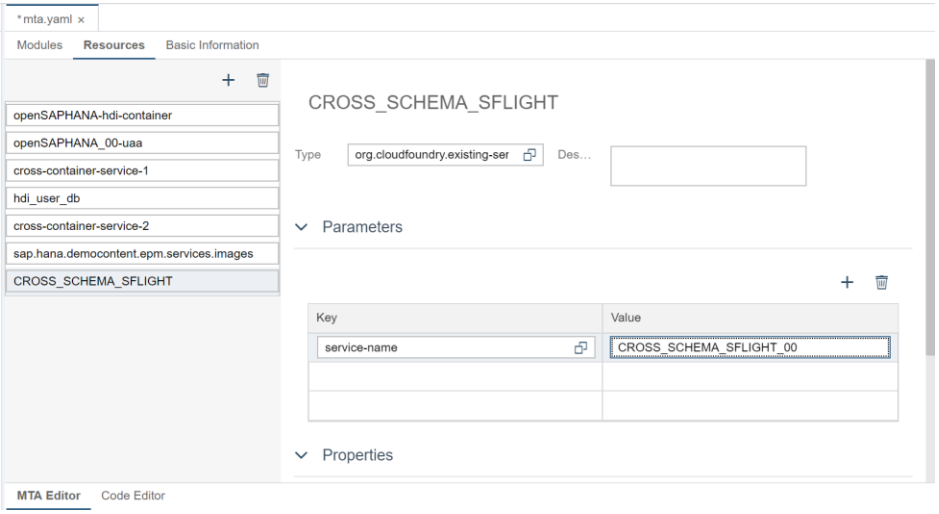
   The URL would be:
   /sap/hana/democontent/epm/services/outboundTest.xsjs?cmd=Images&search=HANA

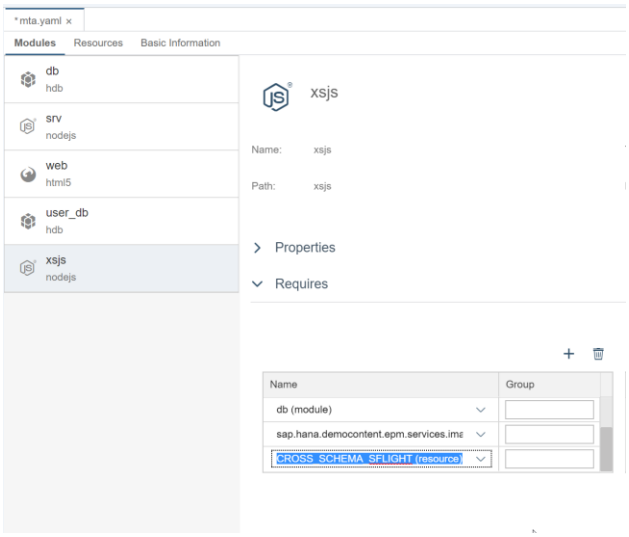


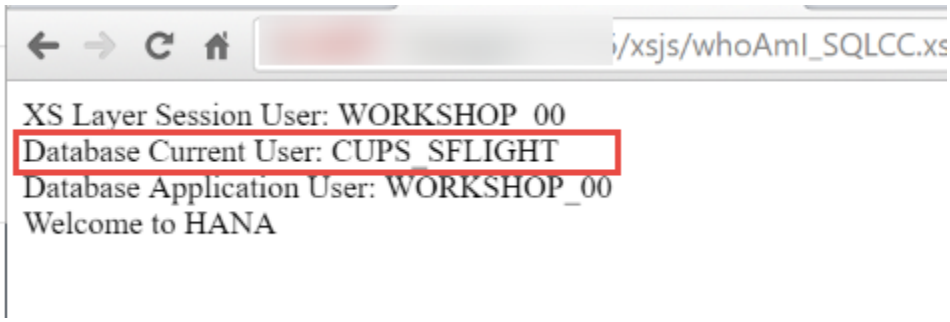
10) You can page through the search results by adding the index URL parameter. To search for the 10th image in the HANA results set use this URL:
    /sap/hana/democontent/epm/services/outboundTest.xsjs?cmd=Images&search=HANA&index=10



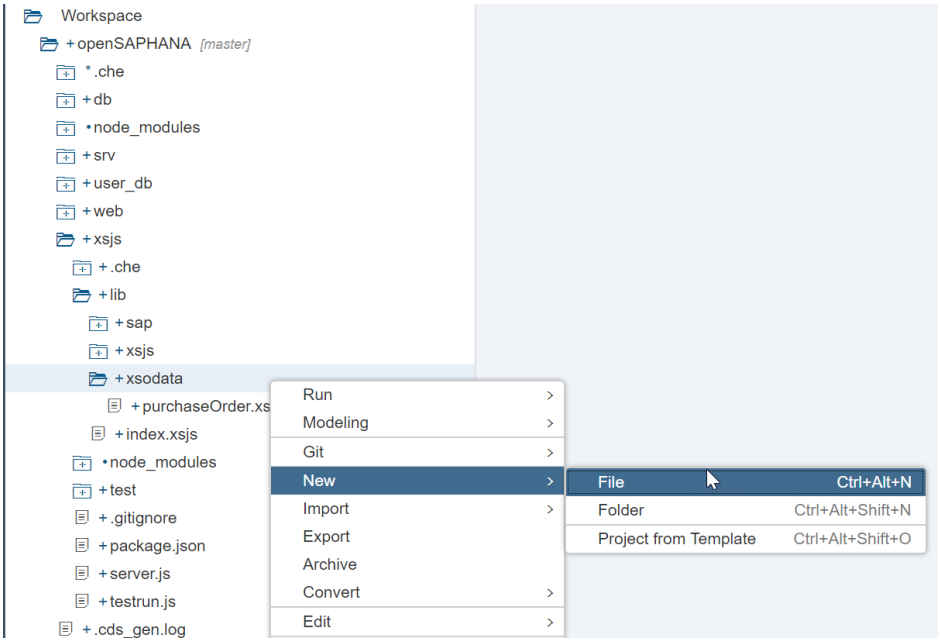
Exericse 3.5: User Provided Service as SQLCC

Explanation	Screenshot
1. XSJS in XSC had a concept called SQLCC where you could force database queries to run as a different user than the logged on user. This concept still continues in XSJS in XSA, but now we utilize a User Provided Service as the basis of the configuration.	
2. Let's reuse the SFLIGHT user provided service which you should have already created back in exercise 2.5. The name should be **CROSS_SCHEMA_SFLIGHT_<group number>** or create it now with the steps back in exercise 2.5.	
3. Declare a new resource in our **mta.yaml** for the existing service but name this new resource CROSS_SCHEMA_SFLIGHT. Save the mta.yaml file. Switch to the Code Editor and then back to the MTA Editor to refresh the value helps. **- name: CROSS_SCHEMA_SFLIGHT** **type: org.cloudfoundry.existing-service**	

parameters: **service-name: CROSS_SCHEMA_SFLIGHT_00**	
4. Add add this resources as a dependency of the **xsjs** module in the mta.yaml as well.	
5. We now need to return to **server.js** in the **xsjs** module and extend the xsjs bootstrap (adjust name for group number).	
6. In the **lib/xsjs** folder, create a file named **whoAmI_SQLCC.xsjs**. Use the following code to connect to the database using the User Provided Service configuration and output the current user. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web	var connection = $.hdb.getConnection({"sqlcc": "xsjs.sqlcc_config", "pool": true }); var query = "SELECT CURRENT_USER FROM \"DUMMY\""; var rs = connection.executeQuery(query); var currentUser = rs[0].CURRENT_USER; query = "SELECT SESSION_CONTEXT('APPLICATIONUSER') \"APPLICATION_USER\" FROM \"DUMMY\""; rs = connection.executeQuery(query); var applicationUser = rs[0].APPLICATION_USER; var greeting = "XS Layer Session User: " + $.session.getUsername() +

address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_10	"</br>Database Current User: " + currentUser + "</br> Database Application User: " + applicationUser + "</br>Welcome to HANA "; $.response.contentType = "text/html; charset=utf-8"; $.response.setBody(greeting);
7. Save the open files. Re-run the **xsjs** service to pickup your changes.	
8. The path to your service should be **/xsjs/whoAmI_SQLCC.xsjs** Instead of the HDI container technical user as these Database Current User, you should see whichever user was configured in the User Provided Service.	

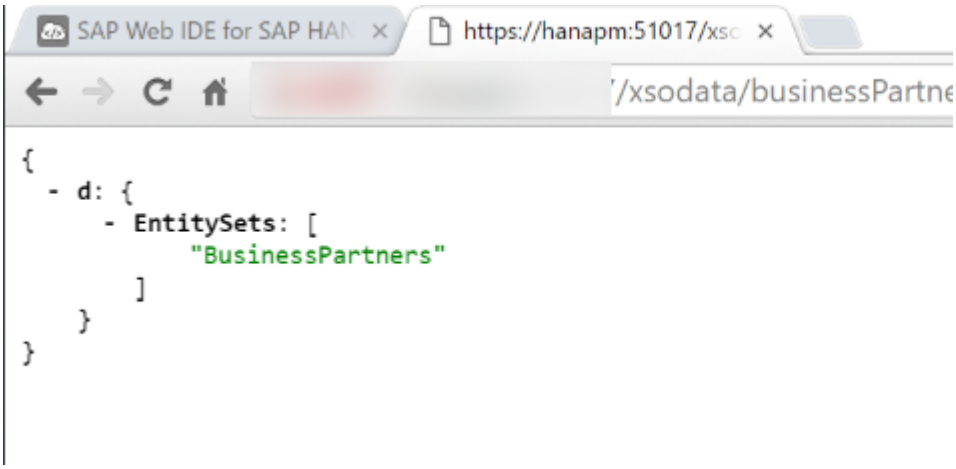
Exercise 3.6: Creating a Simple Odata V2 Service

Explanation	Screenshot
1. Right mouse click on the **xsjs/lib/xsodata** folder and choose **New->File**.	
2. Enter the name as **businessPartners.xsodata** and click **OK**.	
3. We want to define an OData service to expose the business partner table. The syntax of the XSODATA service is relative easy for this use case. We need only define a namespace (**your package path**), the name of the HANA Table we will base the service from (**MD.BusinessPartner**) and the name of the OData entity (**BusinessPartners**).	

Therefore the content of the XSODATA file would be.

4. Save the file. Run the **xsj**s module. Then return to the running web module. Change the url path to **/xsodata/businessPartners.xsodata/?$format=json** to test this new service.

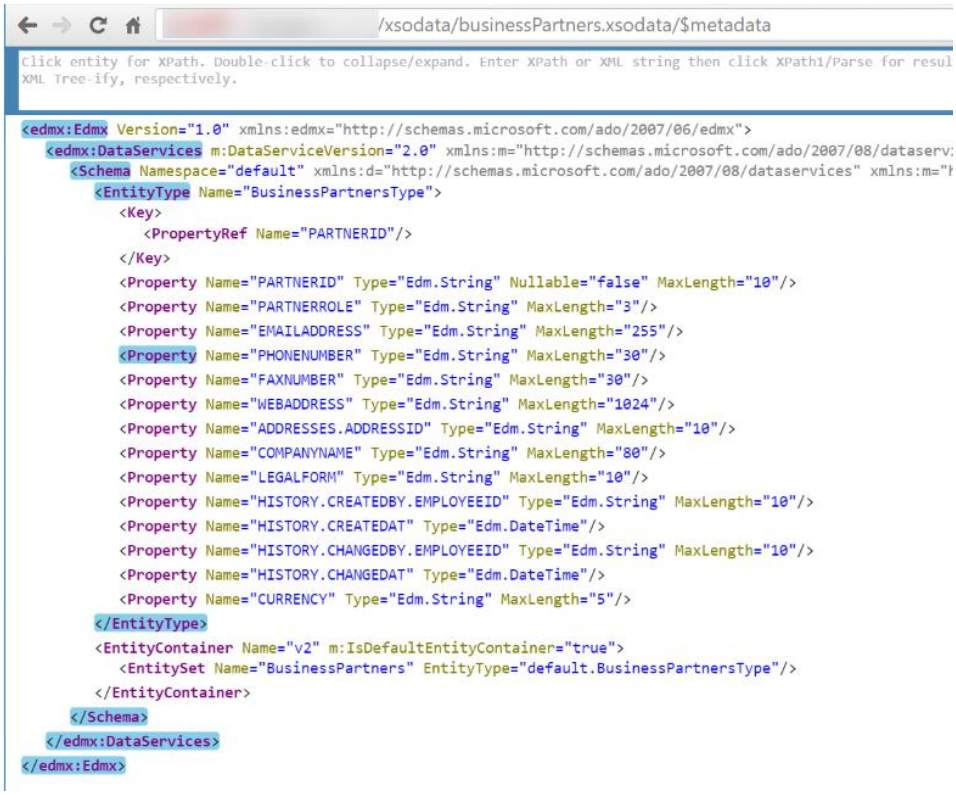
   The resulting document describes the service entities. We only had the one entity named BusinessPartners.



5. You can now adjust the URL slightly and add the /$metadata parameter to the end of it.

   For Example: **/xsodata/businessPartners.xsodata/$metadata**

   You can see the field details for all the attributes of the OData service.



6. In order to view the data of the entity, you would append **BusinessPartners** to the end of the URL:

   For Example:
   /xsodata/businessPartners.xsodata/BusinessPartners?$format=json

   You are now able to see the data from the businessPartner table.



7. You can also experiment with standard OData URL parameters like $top, $skip, or $filter. These options are interpreted and handled by the OData service of the XSEngine for you. You get complex service handling without any coding. For example the following URL would return only three business partner records and would skip the first five records. Such parameters are helpful when implementing server side scrolling, filtering, or sorting in table UI elements.

   For Example:
   /xsodata/businessPartners.xsodata/BusinessPartners?$top=3&$skip=5&$format=json



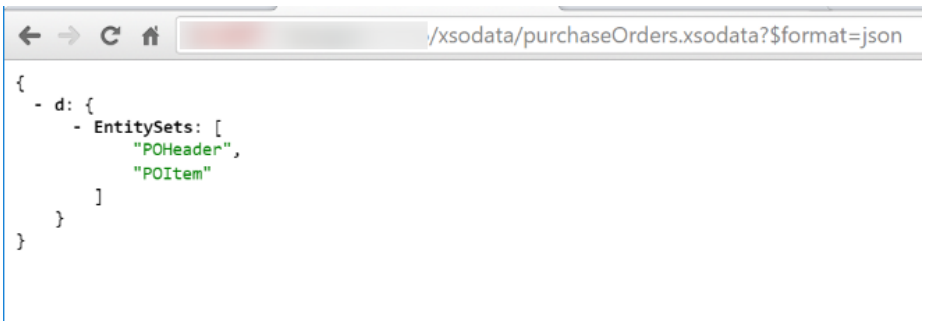
Exercise 3.7: Creating an OData V2 Service with an Entity Relationship

Explanation	Screenshot
1. The previous example of this exercise was very simplistic because it only exposed one database table as a single entity. Often you need to also represent relationships between multiple entities. For example we might want to build an OData service which has both the Purchase Order Header and Items. For this we would build a 1:many relationship between the two entities.	
2. Returning to the editor, you should now create a new OData service named **purchaseOrders.xsodata** in the **lib/xsodata** folder and extend it to include the **PO.Header** and **PO.Item** tables. Next create a navigation **1:many** association. The new content of the definition file should look like this: **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_11	*purchaseOrders.xsodata × 1 service namespace "com.sap.openSAP.hana.example.services" { 2 "PO.Header" 3 as "POHeader" navigates ("Items" as "POItem"); 4 5 "PO.Item" 6 as "POItem"; 7 8 association "Items" 9 principal "POHeader"("PURCHASEORDERID") 10 multiplicity "1" 11 dependent "POItem"("HEADER.PURCHASEORDERID") 12 multiplicity "*"; 13 }

3. Save, run and test using the same steps as in the previous section of this exercise.

   /xsodata/purchaseOrders.xsodata?$format=json

   Notice that the base service definition now has two entities



4. The Header data now has a hyperlink relationship to the item entity

   /xsodata/purchaseOrders.xsodata/POHeader/?$format=json



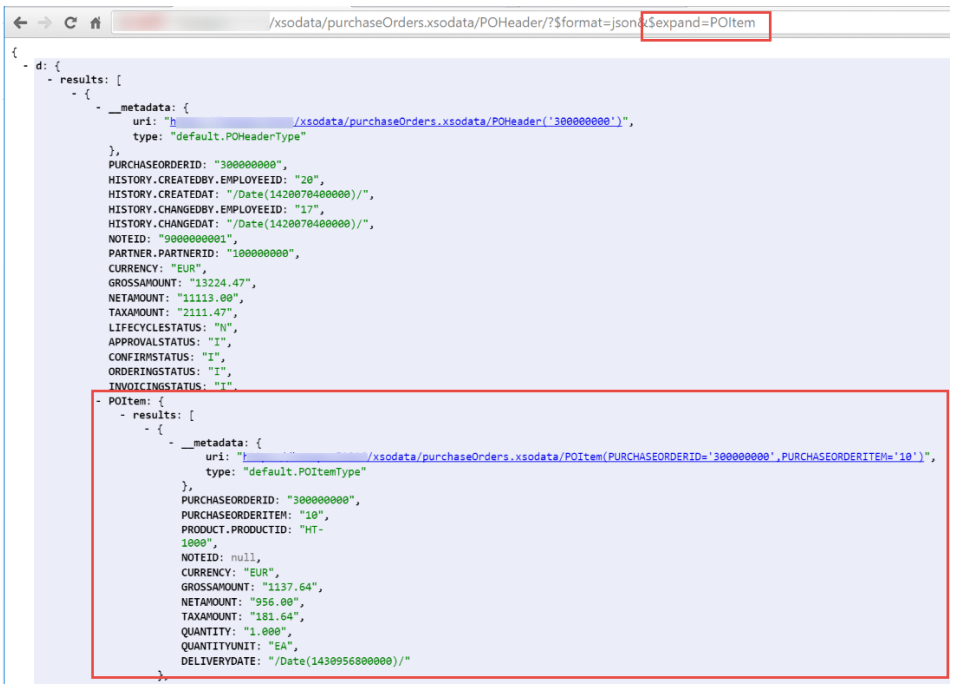
5. Associations can be an excellent way to load child elements on demand; however there is also an option to expand the children details in place so that all levels can be retrieved with one request.

   Test the service again using the same steps as in the previous section of this exercise.

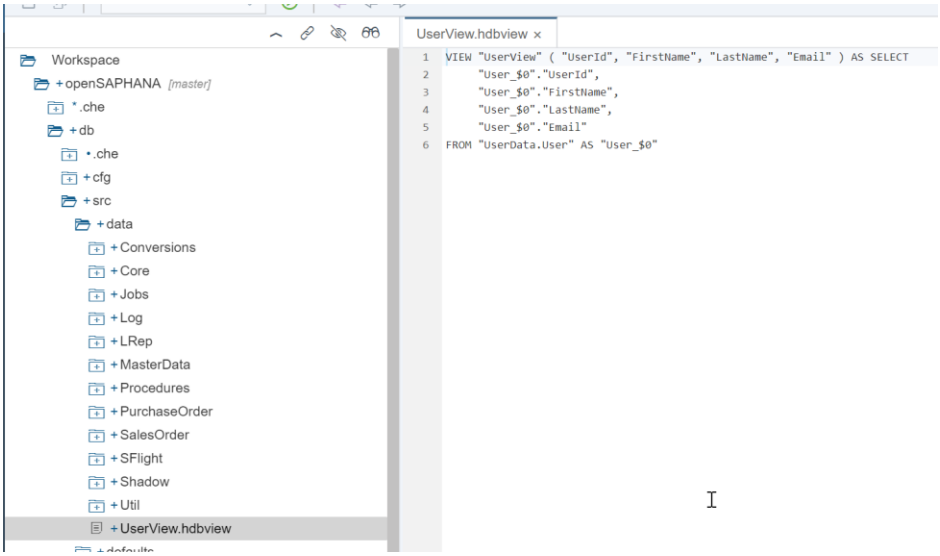
   This time add $expand=POItem to the end of the URL.

   /xsodata/purchaseOrders.xsodata/POHeader/?$format=json&$expand=POItem

   You will then see that all the items are embedded within each header record.



Exercise 3.8: Creating an Odata V2 Service with Create Operation and XSJS Exit

Explanation	Screenshot
1. For the next OData service we want to use the User entity, but remember back to exercise 2 we isolated this database object into its own HDI container and access it via Synonym in the db module. We can't use a Synonym as the entity source in XSODARA, therefore we will need a view on the synonym to be able to use this object in an Odata service in our **xsjs** module. Return to the **db/src/data** folder and create a file named **UserView.hdbview.** Save and then build the db module. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_12a	
2. Create another OData service named **user.xsodata** for UserData.User in the **xsjs/lib/xsodata** folder. This time, also link the create operation to the Server Side JavaScript Library (XSJSLIB) user.xsjs:usersCreateMethod.xsjslib and the function usersCreate. This will be the exit code that performs validation before the insert of the new record.	service { "UserView" as "Users" key("UserId") create using "xsjs:usersCreateMethod.xsjslib::usersCreate"; }

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_15	
3. In the **xsjs/lib/xsjs** folder create the file **usersCreateMethod.xsjslib**. Here is the code for this file. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex3/ex3_16	/*eslint no-console: 0, no-unused-vars: 0, dot-notation: 0, no-use-before-define: 0, no-redeclare: 0*/ "use strict"; $.import("sap.hana.democontent.epm.services", "session"); var SESSIONINFO = $.sap.hana.democontent.epm.services.session; function validateEmail(email) { var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)\|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])\|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/; return re.test(email); } /** @param {connection} Connection - The SQL connection used in the OData request @param {beforeTableName} String - The name of a temporary table with the single entry before the operation (UPDATE and DELETE events only) @param {afterTableName} String -The name of a temporary table with the single entry after the operation (CREATE and UPDATE events only) */ function usersCreate(param) { try { var after = param.afterTableName; //Get Input New Record Values var pStmt = param.connection.prepareStatement("select * from \"" + after + "\""); var rs = null; var User = SESSIONINFO.recordSetToJSON(pStmt.executeQuery(), "Details"); pStmt.close(); console.log(JSON.stringify(User)); console.log(User.Details[0].FirstName); //Validate Email if (!validateEmail(User.Details[0].Email)) { throw "Invalid email for " + User.Details[0].FirstName + " No Way! E-Mail must be valid and " + User.Details[0].Email + " has problems"; }

```
                pStmt = param.connection.prepareStatement("insert into
\"UserData.User\" (\"FirstName\", \"LastName\", \"Email\") values(?,?,?)");

                pStmt.setString(1, User.Details[0].FirstName.toString());
                pStmt.setString(2, User.Details[0].LastName.toString());
                pStmt.setString(3, User.Details[0].Email.toString());

                pStmt.executeUpdate();
                pStmt.close();
                //              }
        } catch (e) {
                console.error(e);
                throw e;
        }
}
```

4. Save and run the xsjs module and then the web module. Change the URL to **/xsodata/user.xsodata/Users/?$format=json**

   Unfortunately it's much more complicated to test Create/Update /Delete methods from the browser as they create other HTTP verbs. Later we will revisit this service and build a user interface which can call this service in order to fully test it.



EXERCISE 4 – NODE.JS

Objective

In exercise 3 we worked in a Node.js module, but didn't do much Node.js specific programming. We only were using Node.js to run XSJS and XSODATA services. The support for XSJS and XSODATA is an important feature for XS Advanced. It not only allows backward compatible support for much of your existing development; but it provides a simplified programming model as an alternative to the non-block I/O event driven programming model normally used by Node.js.

But we certainly aren't limited to only the functionality provided by XSJS and XSODATA. We have access to the full programming model of Node.js as well. In this exercise, we will learn how to extend our existing Node.js module in the SAP Web IDE for SAP HANA.

You will learn about how to create and use reusable code in the form of node.js modules. You will use packages.json to define dependencies to these modules which make the installation of them quite easy. You will use one of the most popular modules – **express**; which helps with the setup the handling of the request and response object. You will use **express** to handle multiple HTTP handlers in the same service by using routes.

You will learn about the fundamentals of the asynchronous nature of nodel.js. We will so see how this asynchronous capability allows for non-blocking input and output. This technique is one of the basic things that makes Node.js development different from other JavaScript development and creates one of the reasons for its growing popularity. We will see how these techniques are applied to common operations like HTTP web service calls or even SAP HANA database access. We will also see how to create language translatable text strings and HANA database queries from Node.js.

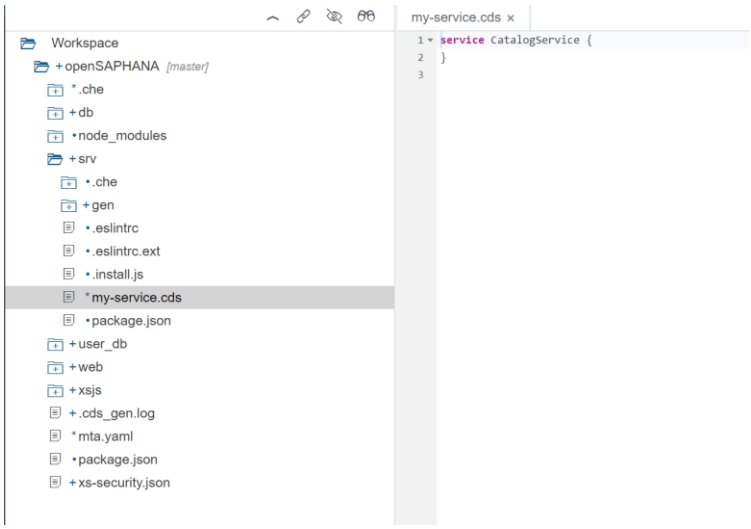
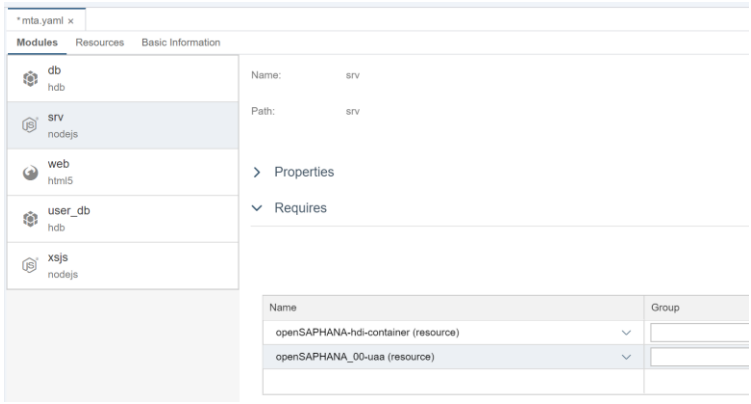
Our final part of this exercise will demonstrate the ease at which you can tap into the powerful web sockets capabilities of Node.js. We will use web sockets to build a simple chat application. Any message sent from the SAPUI5 client side application will be propagated by the server to all listening clients.

Exercise Description

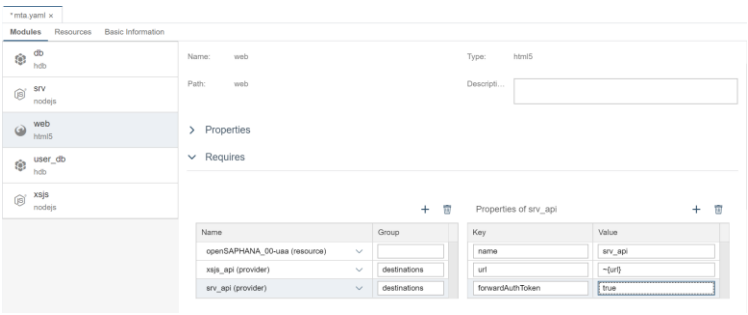
- Modules
- Express Module and package.json
- Local Modules
- SAP HANA database access from node.js
- Basic asynchronous processing
- Non-blocking I/O
- Non-blocking HTTP requests
- Non-blocking database requests
- JavaScript basics
- Text bundles
- Web Sockets chat application

EXERCISE 4 – SOLUTION

Exercise 4.1: Odata V4 services

Explanation	Screenshot
1. Back in Exercise 1 when we created our project, a Node.js module for Odata V4 service enablement was created as well. This **srv** module currently has an empty service definition in the my-service.cds file. In this exercise we will build out the service definition in the **my-service.cds** file and see the long term replacement for the XSODATA capabilities.	
2. First we need to extend the module requirements for srv that were generated by the wizard to add connectivity to our UAA service and other resources. Go to the **mta.yaml** editor and the Modules tab. Select **srv** module. Add a requires entry for **openSAPHANA6_<group_number>-uaa.**	 - name: srv type: nodejs path: srv parameters: memory: 512M disk-quota: 256M provides: - name: srv_api properties: url: '${default-url}' requires: - name: openSAPHANA-hdi-container - name: openSAPHANA_<group number>-uaa

3. Now select the web module in the MTA Editor. Add a requires for the new **srv_api** provider with the shown values.




```
- name: srv_api
  group: destinations
  properties:
    name: srv_api
    url: '~{url}'
    forwardAuthToken: true
```

4. Return to **/srv/my-service.cds** and add to service entity entries – one for the PO Headers table and one fort he PO Items table we created in CDS in exercise 2.2.

   Note: If you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_1a

```
using opensap.PurchaseOrder.Headers as Headers from '../db/data-model';
using opensap.PurchaseOrder.Items as Items from '../db/data-model';

service CatalogService {

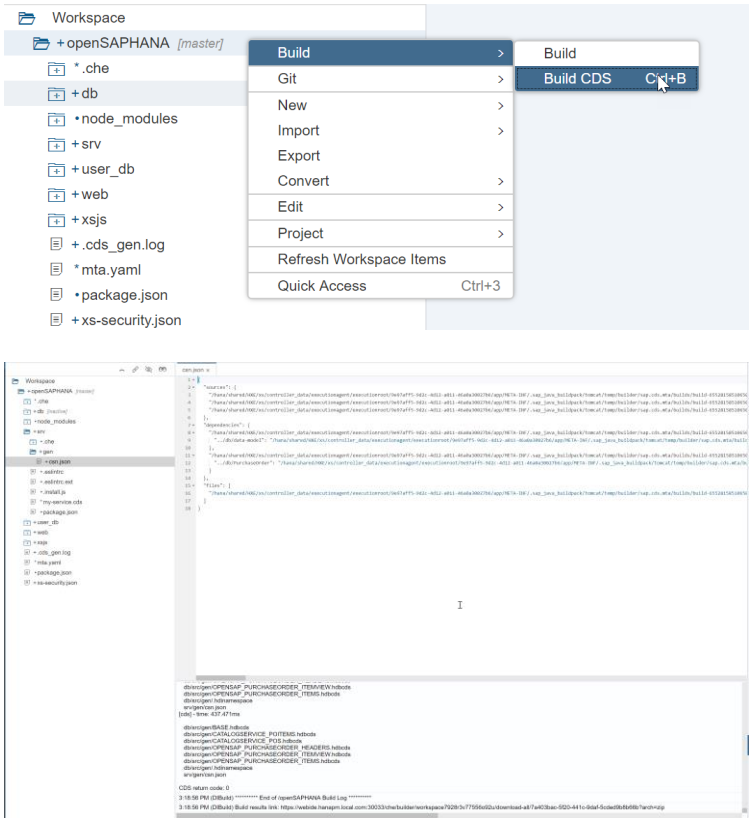
entity POs @(
                    title: '{i18n>poService}',
                    Capabilities: {
                                InsertRestrictions: {Insertable: true},
                                UpdateRestrictions: {Updatable: true},
                                DeleteRestrictions: {Deletable: true}
                    }
          ) as projection on Headers;

entity POItems @(
                    title: '{i18n>poService}',
                    Capabilities: {
                                InsertRestrictions: {Insertable: true},
                                UpdateRestrictions: {Updatable: true},
                                DeleteRestrictions: {Deletable: true}
                    }
          ) as projection on Items;

}
```

5. Save your files and execute a **Build CDS**.

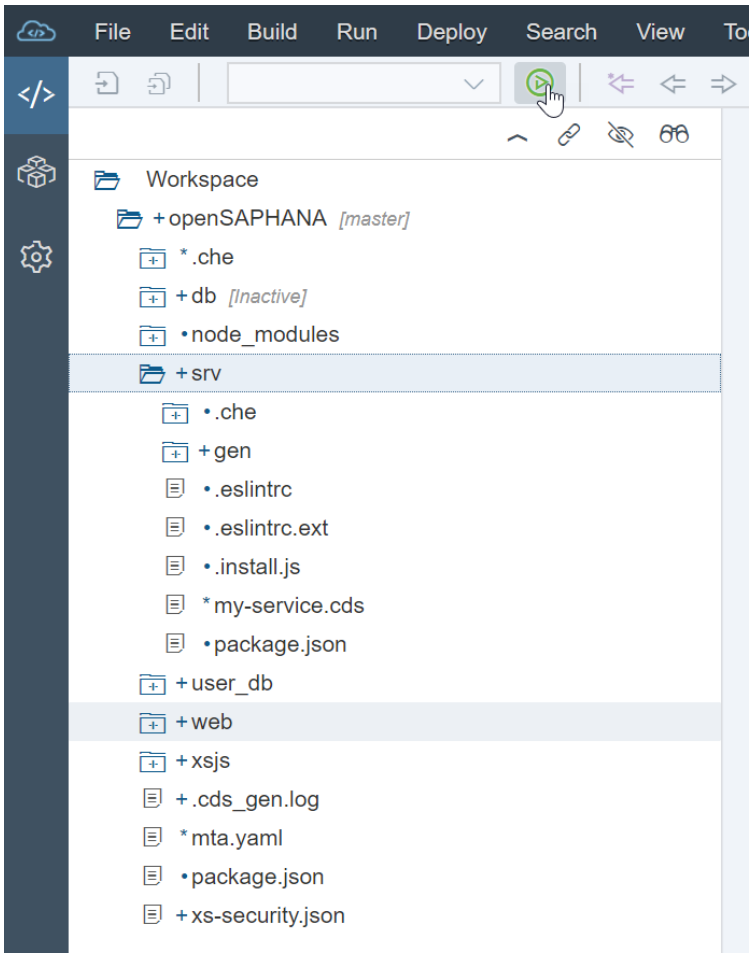
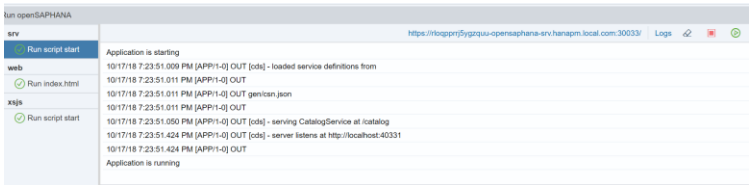
   If the build is sucessful, we should see that the **srv** module now has a **gen/csn.json** file with the service definition details.



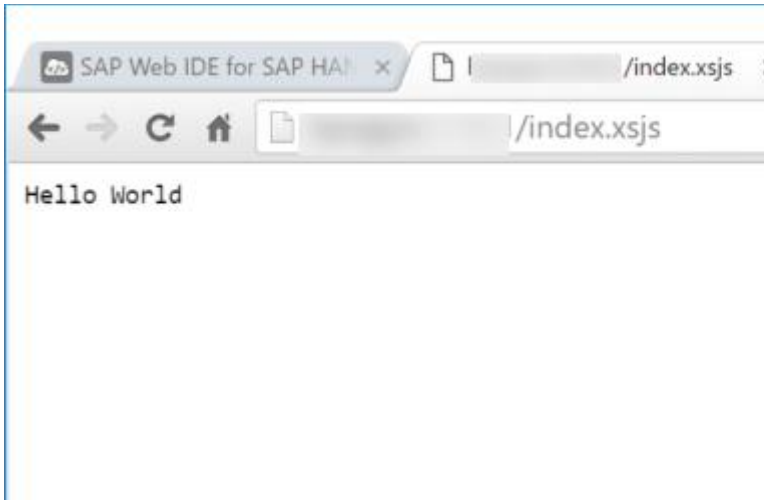
6. Our Node.js **srv** module is essentially finished since the generic provider framework will give us our Odata services, however before we can run it we also need to add a new route to the **xs-app.json** of our **web** module.

   Note: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_1b

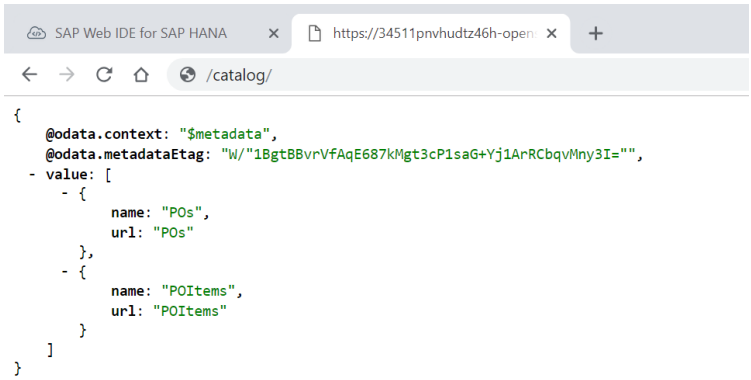
```
{
        "welcomeFile": "index.html",
        "authenticationMethod": "route",
        "routes": [{
                "source": "/catalog/(.*)",
                "destination": "srv_api",
                "csrfProtection": true,
                "authenticationType": "xsuaa"
        },{
                "source": "/odata/v4/(.*)",
                "destination": "srv_api",
                "csrfProtection": true,
                "authenticationType": "xsuaa"
        }, {
                "source": "/node(.*)",
                "destination": "srv_api",
                "csrfProtection": true,
                "authenticationType": "xsuaa"
        },{
                "source": "(.*)(.xsjs)",
                "destination": "xsjs_api",
                "csrfProtection": true,
                "authenticationType": "xsuaa"
        }, {
                "source": "(.*)(.xsodata)",
                "destination": "xsjs_api",
```

	"authenticationType": "xsuaa" }] }
7. We can now run the **srv** module.	
8. You should see that the build and deploy was successful.	
9. So now run the **web** module. It will need to rebuild and redeploy due to the added route.	

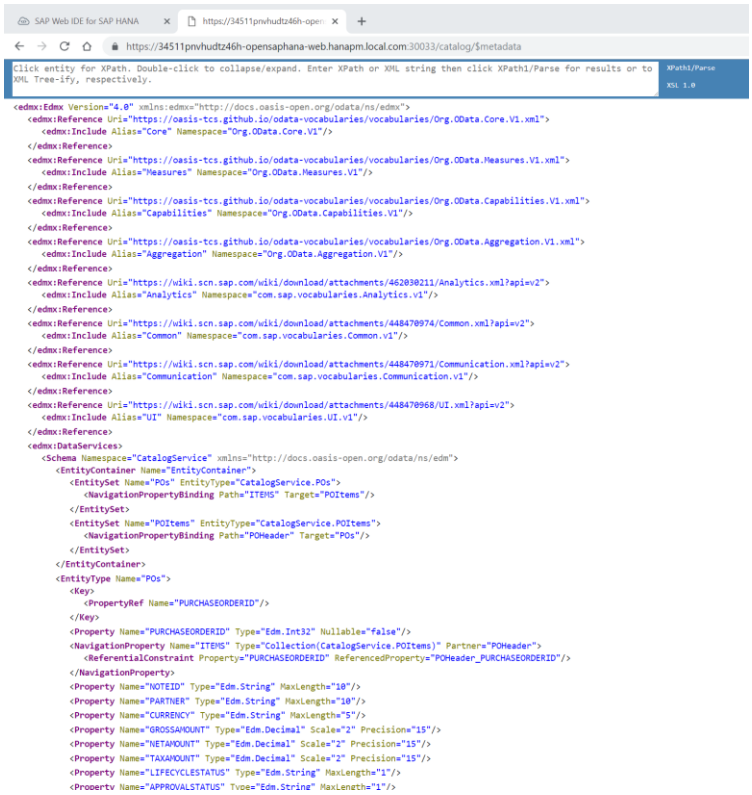
10. In the running tab, you should see the index.html from earlier. We can add the url to our xsjs service /index.xsjs in the browser. This will test to make sure that our XSJS and XSODATA paths are still accessible even though we swapped out the root handler.



11. Now change the path in the browser to **/catalog/**. You should see the service catalog of our Odata V4 framework with two entities – POs and POItems.



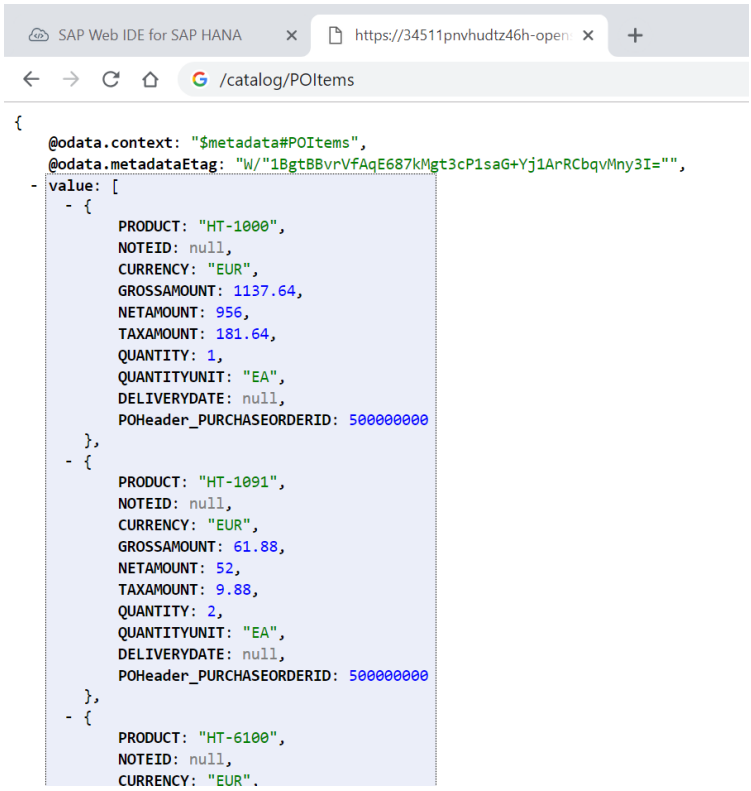
12. Change the URL to **/catalog/$metadata** to see the Odata metadata document.



13. And to see the Purchase Order Headers: **/catalog/POs**



14. And Purchase Order Items: **/catalog/POItems**

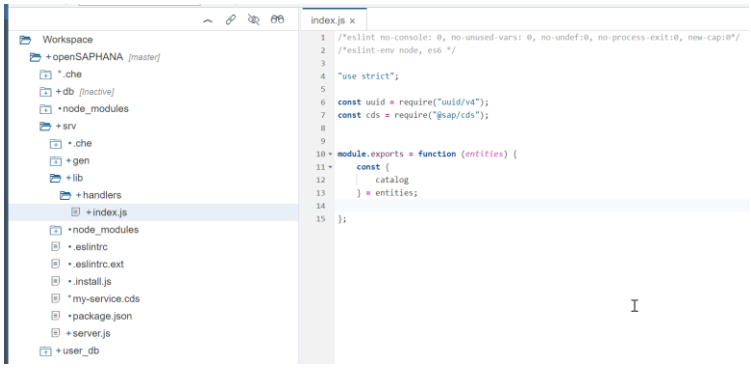



```
{
    @odata.context: "$metadata#POItems",
    @odata.metadataEtag: "W/"1BgtBBvrVfAqE687kMgt3cP1saG+Yj1ArRCbqvMny3I=="",
  - value: [
      - {
            PRODUCT: "HT-1000",
            NOTEID: null,
            CURRENCY: "EUR",
            GROSSAMOUNT: 1137.64,
            NETAMOUNT: 956,
            TAXAMOUNT: 181.64,
            QUANTITY: 1,
            QUANTITYUNIT: "EA",
            DELIVERYDATE: null,
            POHeader_PURCHASEORDERID: 500000000
        },
      - {
            PRODUCT: "HT-1091",
            NOTEID: null,
            CURRENCY: "EUR",
            GROSSAMOUNT: 61.88,
            NETAMOUNT: 52,
            TAXAMOUNT: 9.88,
            QUANTITY: 2,
            QUANTITYUNIT: "EA",
            DELIVERYDATE: null,
            POHeader_PURCHASEORDERID: 500000000
        },
      - {
            PRODUCT: "HT-6100",
            NOTEID: null,
            CURRENCY: "EUR",
```

Exercise 4.2: Modules and Express

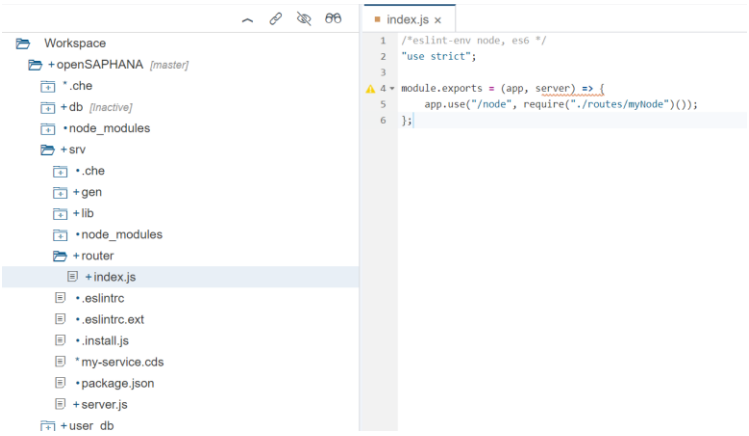
Explanation	Screenshot
1. Now we want to write some custom Node.js logic for our own REST services as well as exits in the CDS Odata framework. We can combine both options together into a single srv module and use Node.js Express module to handle the internal routing to different blocks of code.	
2. Create a file named **server.js** in the **/srv** folder. We want to intialize the use of Express and inject our UAA service and HANA service middleware into Express. This will allow express to enforce our UAA requirements as well as create a HANA database client available to all express route requests. We will also connect to the CDS Odata generic service framework and run it within Express as well. This is similar to the options section of the server.js file we used to set the configuration for the XSJS module. We are doing essentially the same thing here – we are telling the service how to lookup the connection and security parameters. The xsHDBConn is an instance of the sap-hdb-ext module designed to expose pooled database connections as a Node.js express middleware. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-	/*eslint no-console: 0, no-unused-vars: 0, no-undef:0, no-process-exit:0*/ /*eslint-env node, es6 */ "use strict"; const port = process.env.PORT || 3000; const server = require("http").createServer(); const cds = require("@sap/cds"); //Initialize Express App for XSA UAA and HDBEXT Middleware const xsenv = require("@sap/xsenv"); const passport = require("passport"); const xssec = require("@sap/xssec"); const xsHDBConn = require("@sap/hdbext"); const express = require("express"); global.__base = __dirname + "/"; //logging var logging = require("@sap/logging"); var appContext = logging.createAppContext(); //Initialize Express App for XS UAA and HDBEXT Middleware var app = express(); //Compression app.use(require("compression")({ threshold: "1b" })); //Helmet for Security Policy Headers const helmet = require("helmet"); // ... app.use(helmet()); app.use(helmet.contentSecurityPolicy({ directives: { defaultSrc: ["'self'"], styleSrc: ["'self'", "sapui5.hana.ondemand.com"], scriptSrc: ["'self'", "sapui5.hana.ondemand.com"]

hana7/blob/snippets_2.3.2/ex5/ex5_1	} })); // Sets "Referrer-Policy: no-referrer". app.use(helmet.referrerPolicy({ policy: "no-referrer" })); passport.use("JWT", new xssec.JWTStrategy(xsenv.getServices({ uaa: { tag: "xsuaa" } }).uaa)); app.use(logging.middleware({ appContext: appContext, logNetwork: true })); app.use(passport.initialize()); var hanaOptions = xsenv.getServices({ hana: { tag: "hana" } }); hanaOptions.hana.pooling = true; app.use(passport.authenticate("JWT", { session: false }), xsHDBConn.middleware(hanaOptions.hana)); //CDS OData V4 Handler var options = { driver: "hana", logLevel: "error" }; //Use Auto Lookup in CDS 2.10.3 and higher //Object.assign(options, hanaOptions.hana, { // driver: options.driver //}); cds.connect(options); var odataURL = "/odata/v4/opensap.hana.CatalogService/"; // Main app cds.serve("gen/csn.json", { crashOnError: false }) .at(odataURL) .with(require("./lib/handlers")) .in(app) .catch((err) => { console.log(err); process.exit(1); });

	// Redirect any to service root app.get("/", (req, res) => { res.redirect(odataURL); }); app.get("/node", (req, res) => { res.redirect(odataURL); }); //Setup Additonal Node.js Routes require("./router")(app, server); //Start the Server server.on("request", app); server.listen(port, function () { console.info(`HTTP Server: ${server.address().port}`); });
3. Notice that when we created the instance of the CDS module for our Odata service we referenced **with(require("./lib/handlers")).** This is where we will eventually code exit handlers for the Odata V4 framework. For now create folders in **srv** called **lib/handlers** and file in **lib/handlers** called **index.js**. Enter some skeleton code so that later we return and place our Odata service exit handlers here. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_1c	

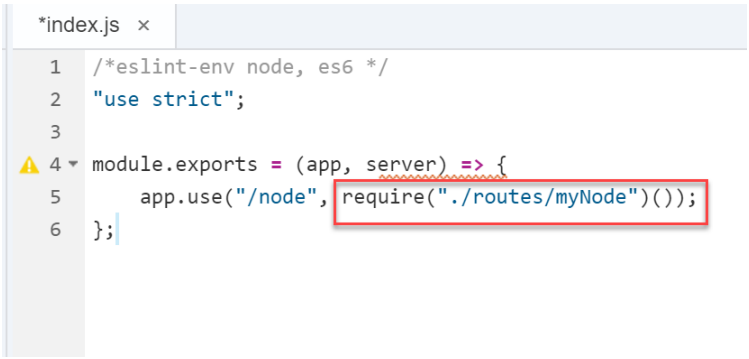
4. We want to create a good project structure that can acomidate many different Express route requests. To follow some Express best practices, create a folder in **srv** called **router** with an **index.js** file in it. This is the root route handler we referenced back in step 4 in our **server.js**.

   For now only create a single route for the path **/node**. The handler for this route will be called myNode.

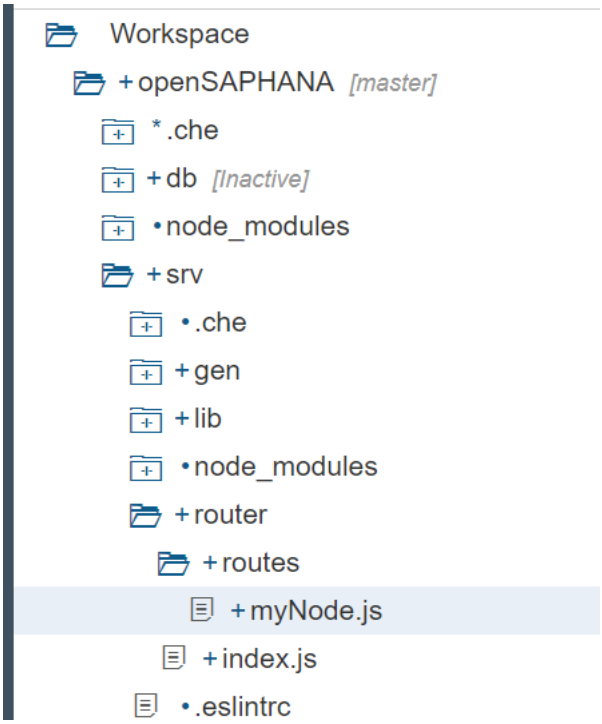


5. Notice that in the previous additions we added a reference to a module called **./routes/myNode**. This is a Node.js module which will be local to our project.

   Modules don't all have to come from central repositories. They can also be a way of modularizing your own code (similar to XSJSLIB files). That is exactly what we will do here. Rather than filling up our server.js with application specific logic, we will break that out into its own Node.js module named myNode.js.



6. Create a new folder called **routes** inside **srv/router**. Then create a file called **myNode.js**.



7. Add the following code to your myNode.js file. This will add a simple GET handler that returns a “Hello World” message when requested.

   Notice express in this module has a root route handler. But this will relatively to where it is attached in router/index.js. Therefore this logic will still respond relative to the /node route handler base we specified earlier.

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/
/*eslint-env node, es6 */
"use strict";
var express = require("express");
var async = require("async");

module.exports = function() {
	var app = express.Router();
	//Hello Router
	app.get("/Hi", (req, res) => {
		res.send("Hello World Node.js");
	});

	return app;
};
```

8. Look at the **package.json** file in the editor. You will see the dependencies section which lists all required libraries and their versions.

   We manually added the express Node.js module to our project, so we also need to extend the dependencies section here. While here, we also need a few of the other

```
{
	"name": "serve",
	"dependencies": {
		"@sap/cds": "^2.10",
		"express": "4.16.4",
		"@sap/xssec": "2.1.15",
		"@sap/xsenv": "1.2.9",
		"hdb": "0.16.0",
		"@sap/hdbext": "~5.0",
		"@sap/hana-client": "~2.3",
		"@sap/textbundle": "latest",
		"@sap/logging": "~4.0",
```

modules we will need later in the exercise.

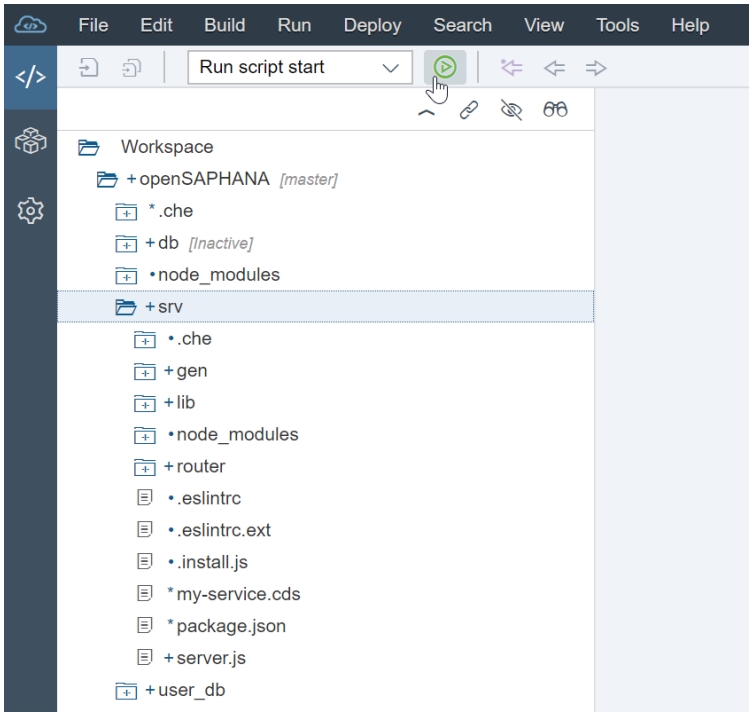
We also need to change the start script from the generic CDS Odata provider to instead point to our server.js.

Note: if you need help writing this code please refer to the solution at:
https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_4

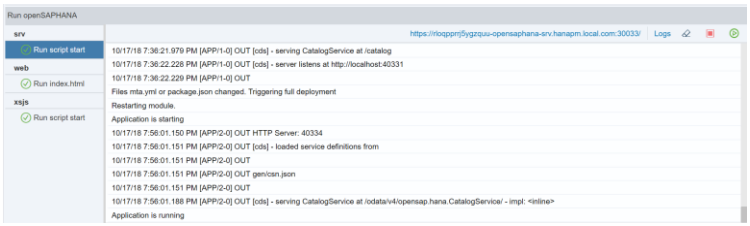
```
                "@sap/audit-logging": "2.2.4",
                "nodemailer": "~4.6",
                "passport": "~0.4.0",
                "async": "latest",
                "ws": "~6.1",
                "accept-language-parser": "latest",
                "node-xlsx": "~0.12",
                "node-zip": "~1.1.1",
                "xmldoc": "~1.1.2",
                "winston": "~3.1",
                "body-parser": "~1.18",
                "elementtree": "latest",
                "then-request": "latest",
                "compression": "~1.7",
                "helmet": "~3.15"
        },
        "scripts": {
                "postinstall": "cds build/all --project .. --clean",
                "start": "node server.js"
        },
        "engines": {
                "node": "8.x",
                "npm": "6.x"
        },
        "i18n": {
                "folders": [
                        "_i18n"
                ]
        },
        "cds": {
                "data": {
                        "driver": "hana"
                }
        }

}
```

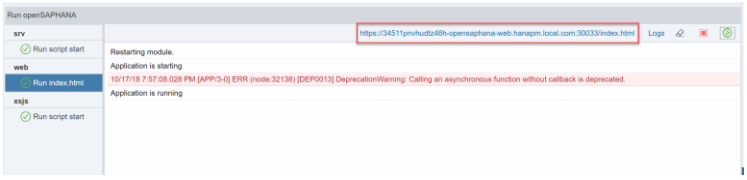
9. We can now run the **srv** module to pickup the changes we've made. It will have to completely rebuild because of the massive changes to the **package.json** we made in this exercise.



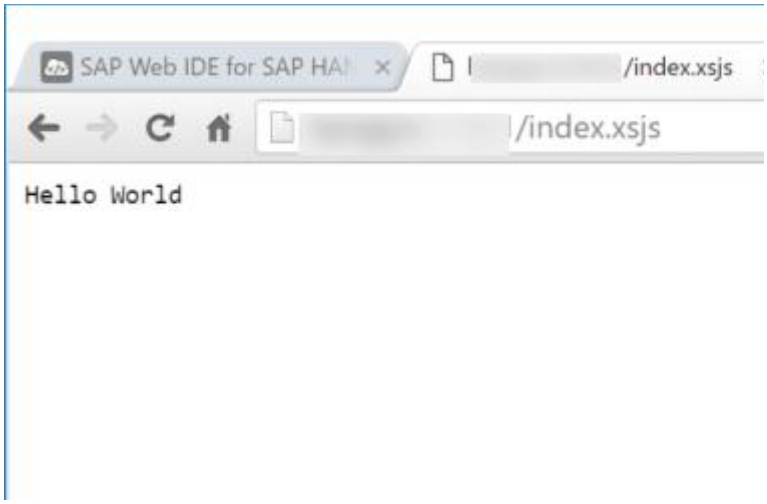
10. You should see that the build and deploy was successful.



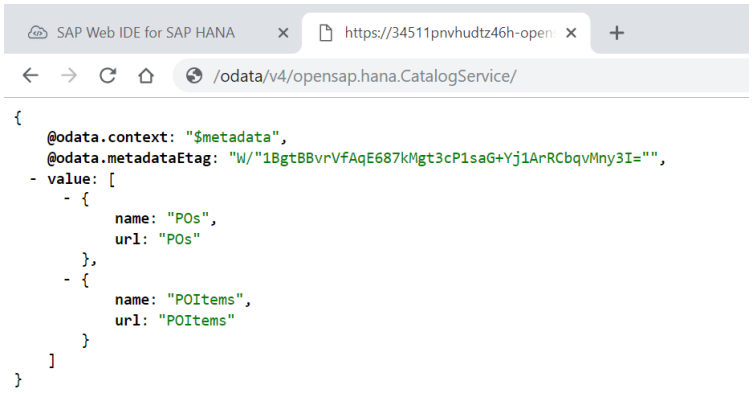
11. You shouldn't need to re-run the web module, but if you closed the brower tab for the web module, you can click on the Application URL in its run window to re-open it.



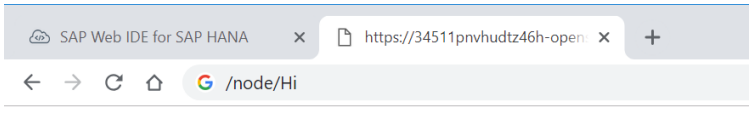
12. In the running tab, you should see the index.html from earlier. We can add the url to our xsjs service **/index.xsjs** in the browser. This will test to make sure that our XSJS and XSODATA paths are still accessible even though we swapped out the root handler.



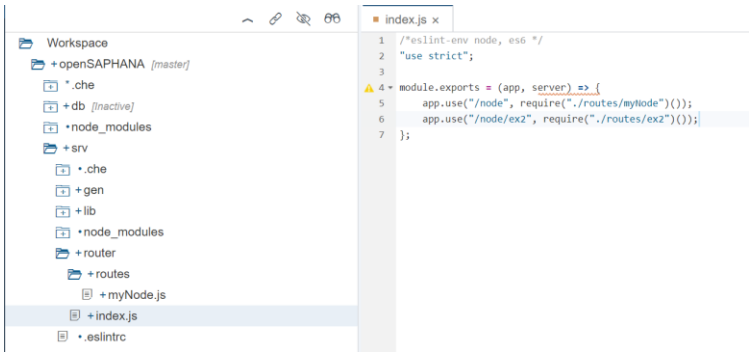
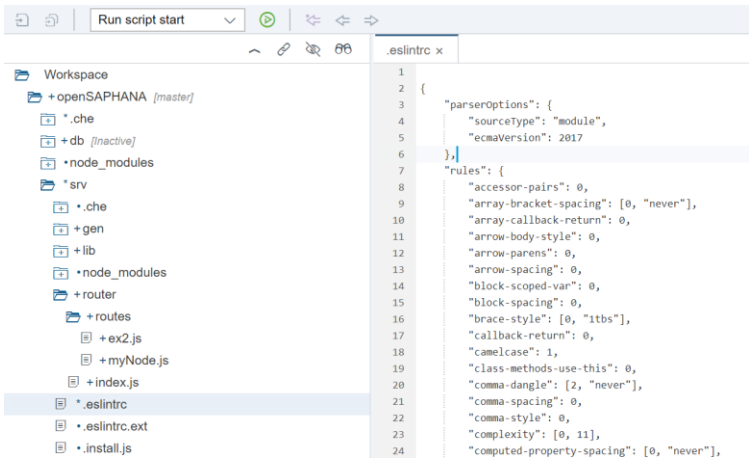
13. Now change the path in the browser to **/odata/v4/openSAP.hana.CatalogService/**. You should see the service catalog of our Odata V4 framework with two entities – POs and POItems but now being servered out from within Express alongside our custom Node.js handlers.



14. Now change the path in the browser to **/node/Hi**. You should see Hello World Node.js this is being returned from the new route handler and Node.js module we just implemented in this exercise.



Exercise 4.3: HANA Database Access from Node.js

Explanation	Screenshot
1. In this exercise, we will look at how to use the HANA database access library to send queries to the database. Return to the **srv** module and using what you learned in the previous exercise create a new route called **ex2**. Let's extend the logic here with a URL handler for **/node/ex2** which will issue a database SELECT statement against the DUMMY table and return the current database user.	
2. We are going to use some very recent Node.js/JavaScript features so we want to extend the ESLint syntax check configurtion. Edit the **.eslintrc** file in the **srv** folder. Add the following at the beginging of the file: "parserOptions": { "sourceType": "module", "ecmaVersion": 2017 },	
3. Create a new file named **ex2.js** in the **routes** folder. We will begin by using the low level Node.js HANA Client which is available via module **@sap/hana-client**. This gives us direct access to the HANA client drivers and will generally support all the latest client features. However, it doesn't have some of the abstraction and convenience features of the higher level **@sap/hdbext** library we will use later. Nor can the **@sap/hana-**	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/ "use strict"; var express = require("express"); module.exports = () => { var app = express.Router(); //Hello Router app.get("/", (req, res) => { let client = require("@sap/hana-client"); //Lookup HANA DB Connection from Bound HDB Container Service const xsenv = require("@sap/xsenv"); let hanaOptions = xsenv.getServices({ hana: { tag: "hana" }

client directly consume the HDI container resource. Instead we will look up the connection parameters from the bound HDI resource and use those values to build the connection string manually **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db1.js	}); //Create DB connection with options from the bound service let conn = client.createConnection(); var connParams = { serverNode: hanaOptions.hana.host + ":" + hanaOptions.hana.port, uid: hanaOptions.hana.user, pwd: hanaOptions.hana.password, CURRENTSCHEMA: hanaOptions.hana.schema }; //connect conn.connect(connParams, (err) => { if (err) { return res.type("text/plain").status(500).send(`ERROR: ${JSON.stringify(err)}`); } else { conn.exec(`SELECT SESSION_USER, CURRENT_SCHEMA FROM "DUMMY"`, (err, result) => { if (err) { return res.type("text/plain").status(500).send(`ERROR: ${JSON.stringify(err)}`); } else { conn.disconnect(); return res.type("application/json").status(200).send(result); } }); } return null; }); }); return app; };
4. Now, add a new route handler for **/ex2** in the file **ex2.js** after the previous section you just inserted but before the **return app** line. This new code gets the database connection/client from the express request object (**req.db**). This **req** object was filled with a database connection from the	//Simple Database Select Via Client Wrapper/Middelware - In-line Callbacks app.get("/express", (req, res) => { let client = req.db; client.prepare(`SELECT SESSION_USER, CURRENT_SCHEMA FROM "DUMMY"`, (err, statement) => { if (err) {

easier to use/higher level module **@sap/hdbext**. This module functions as an express middleware and express will manage the connection pool and database connection for us.

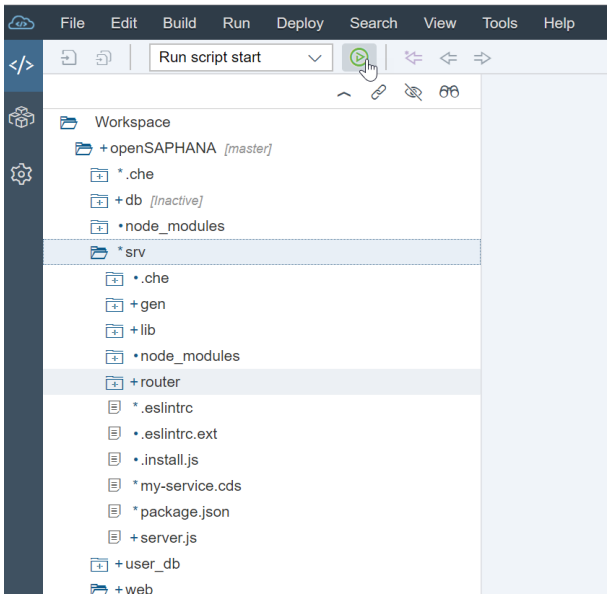
Then create a prepared statement for the SELECT of SESSION_USER from dummy. Execute the statement and send the results as JSON in the response object.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db2.js

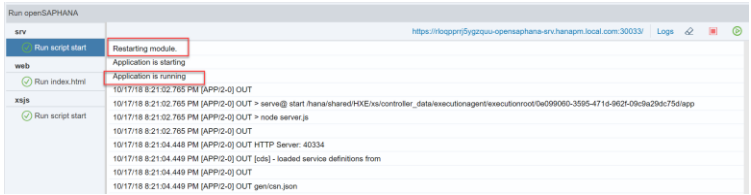
```
                                                    return
res.type("text/plain").status(500).send("ERROR: " +
err.toString());
                                        }
                                        statement.exec([],
                                                (err, results) => {
                                                        if (err) {
                                                                return
res.type("text/plain").status(500).send("ERROR: " +
err.toString());
                                                        } else {
                                                                var
result = JSON.stringify({

        Objects: results
                                                                });
                                                                return
res.type("application/json").status(200).send(result);
                                                        }
                                                });
                                        return null;
                                });
        });
```

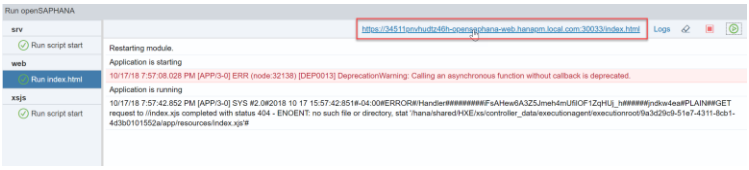
5. Save your file and now run the **srv** module.



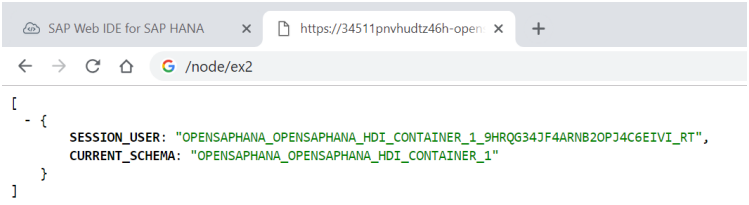
6. You should see that the application was updated successfully is running again.



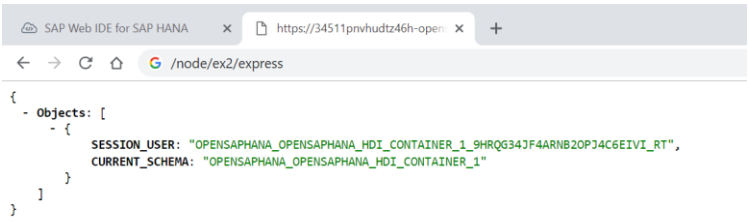
7. Click on the **web** folder and the run window should change to the details of the running HTML5 module. Since its already running and we didn't make any changes to it, you can just click on the Application link in the top right corner of the window to reopen it in your web browser if you closed it from the previous step.



8. Now change the path in the browser to **/node/ex2**. You should see the output of your successful SELECT statement.



9. Now change the path in the browser to **/node/ex2/express**. You should see the same output but the backend code was simpler because now express is managing the HANA client and connection pooling for you.



10. You might have noticed that the default Node.js programming approach is to use callbacks/event handlers for each operation. This is because even the different parts of a database request (connection, prepared statement, execution, etc) are all non-blocking operations.

    While this provides considerable parallelization and performance opportunities; it can also make the code more difficult to read. As the next part of this exercise you can return to **ex2.js** and add a third route handler, **/waterfall,** that performs the same select but uses the **async** module.

```
var async = require("async");
//Simple Database Select Via Client Wrapper/Middelware - Async Waterfall
app.get("/waterfall", (req, res) => {
    let client = req.db;
    async.waterfall([
        function prepare(callback) {
            client.prepare(`SELECT SESSION_USER, CURRENT_SCHEMA
                FROM "DUMMY"`,
                (err, statement) => {
                    callback(null, err, statement);
                });
        },

        function execute(err, statement, callback) {
            statement.exec([], (execErr, results) => {
                callback(null, execErr, results);
```

This module doesn't change the runtime aspects of the code, but organizes the callback functions in an easier to read array instead of in-lining them within each other.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db3.js

Then run the **srv** module and test again at **/node/ex2/waterfall**

```
                                        });
                                },
                                function response(err, results,
callback) {
                                        if (err) {

        res.type("text/plain").status(500).send(`ERROR:
${err.toString()}`);
                                        } else {
                                                var result =
JSON.stringify({
                                                        Objects:
results
                                                });

        res.type("application/json").status(200).send(result);
                                        }
                                        return callback();
                                }
                        ]);
        });
```

11. For our next step we want to use promises, but the database interface module doesn't return promises yet. Therefore we will create a utility module in our local project which wraps the database interface and does return promises.

    Create a folder named **utils** in **srv** and a file in **utils** named **dbPromises.js**.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/dbPromises.js

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-
cap: 0, dot-notation:0, no-use-before-define:0 */
/*eslint-env node, es6 */
"use strict";

module.exports = class {

        static createConnection() {
                return new Promise((resolve, reject) => {
                        const xsenv = require("@sap/xsenv");
                        let options = xsenv.getServices({
                                hana: {
                                        tag: "hana"
                                }
                        });
                        var hdbext = require("@sap/hdbext");
                        options.hana.pooling = true;

        hdbext.createConnection(options.hana, (error, client)
=> {
                                if (error) {
                                        reject(error);
                                } else {
                                        resolve(client);
                                }
                        });
                });
        }
```

```
	constructor(client) {
		this.client = client;
		this.util = require("util");
		this.client.promisePrepare =
this.util.promisify(this.client.prepare);
	}

	preparePromisified(query) {
		return this.client.promisePrepare(query);
	}

	statementExecPromisified(statement, parameters) {
		statement.promiseExec =
this.util.promisify(statement.exec);
		return statement.promiseExec(parameters);
	}

	loadProcedurePromisified(hdbext, schema, procedure)
{
		hdbext.promiseLoadProcedure =
this.util.promisify(hdbext.loadProcedure);
		return
hdbext.promiseLoadProcedure(this.client, schema, procedure);
	}

	callProcedurePromisified(storedProc, inputParams) {
		return new Promise((resolve, reject) => {
			storedProc(inputParams, (error,
outputScalar, ...results) => {
				if (error) {
					reject(error);
				} else {
					if (results.length < 2) {
						resolve({

	outputScalar: outputScalar,

	results: results[0]
						});
					} else {
						let output = {};

	output.outputScalar = outputScalar;
						for (let i = 0; i
< results.length; i++) {

	output[`results${i}`] = results[i];
						}

	resolve(output);
					}
				}
			});
```

```
                });
        }
};
```

12. While the **async** module is a step in the right direction for code readability, it has the disadvantage that it relies upon a 3rd party open source module. However, JavaScript itself has introduced a similar solution in the forum of Promises as of ES6. Promises allow you to chain callback event handling logic together using the **then** statement.

    Let's add another route handler to **ex2.js** that uses the promises approach to perform the same HANA database query as the previous samples.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db4.js

    Then run the **srv** module and test again at **/node/ex2/promises**

```
        //Simple Database Select Via Client Wrapper/Middelware - Promises
        app.get("/promises", (req, res) => {
                const dbClass = require(global.__base + "utils/dbPromises");
                let db = new dbClass(req.db);
                db.preparePromisified(`SELECT SESSION_USER, CURRENT_SCHEMA
                                                FROM "DUMMY"`)
                        .then(statement => {

        db.statementExecPromisified(statement, [])
                                                .then(results => {
                                                        let result = JSON.stringify({

        Objects: results
                                                        });
                                                        return res.type("application/json").status(200).send(result);
                                                })
                                                .catch(err => {
                                                        return res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
                                                });
                        })
                        .catch(err => {
                                return res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
                        });
        });
```

13. The final approach is something very new as of Node.js 8.x. This is a further extension of the Promises pattern but with the new keyword **async** and **await**. This allows you to write asynchronous function calls almost exactly as though they were synchronous with only the addition of the keyword **await**.

```
        //Simple Database Select Via Client Wrapper/Middelware - Await
        app.get("/await", async(req, res) => {
                try {
                        const dbClass = require(global.__base + "utils/dbPromises");
                        let db = new dbClass(req.db);
                        const statement = await db.preparePromisified(`SELECT SESSION_USER, CURRENT_SCHEMA
                                                FROM "DUMMY"`);
```

Notice that this is the most streamlined and easiest of all the examples. Also, instead of repeated and inlined error catch blocks, the Await pattern allows for one single error catch block.

Let's add another route handler to **ex2.js** that uses the Await approach to perform the same HANA database query as the previous samples.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db5.js

Then run the **srv** module and test again at **/node/ex2/await**

```
                const results = await db.statementExecPromisified(statement, []);
                let result = JSON.stringify({
                        Objects: results
                });
                return res.type("application/json").status(200).send(result);
        } catch (e) {
                return res.type("text/plain").status(500).send(`ERROR: ${e.toString()}`);
        }
});
```

14. Next, we want to see how to call Stored Procedures from Node.js. The logic needed is different from selects as you call the **loadProcedure** function and then pass a specific interface structure to call the procedure.

    As the next part of this exercise you can return to **ex2.js** and add another route handler, **/procedures,** that performs calls to the stored procedure, **get_po_header_data**.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db6.js

```
//Simple Database Call Stored Procedure
app.get("/procedures", (req, res) => {
        var client = req.db;
        var hdbext = require("@sap/hdbext");
        //(client, Schema, Procedure, callback)
        hdbext.loadProcedure(client, null, "get_po_header_data", (err, sp) => {
                if (err) {

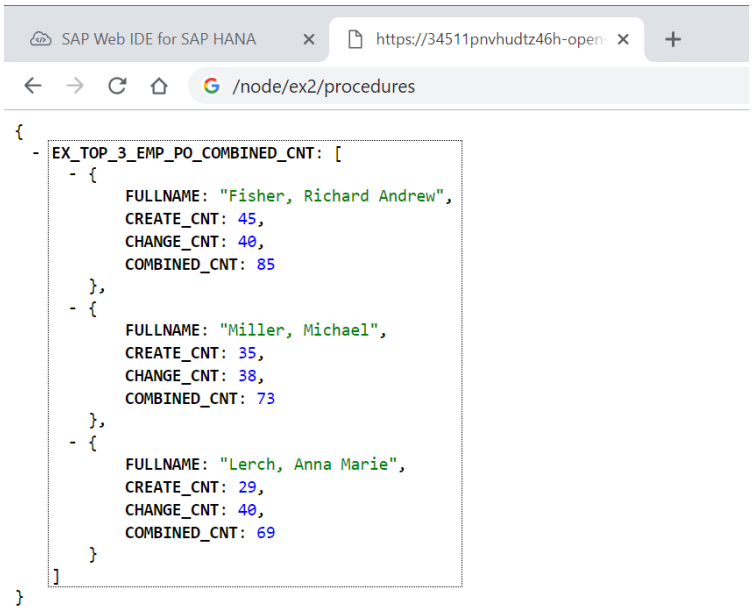
        res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
                        return;
                }
                //(Input Parameters, callback(errors, Output Scalar Parameters, [Output Table Parameters])
                sp({}, (err, parameters, results) => {
                        if (err) {

        res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
                        }
                        var result = JSON.stringify({

        EX_TOP_3_EMP_PO_COMBINED_CNT: results
                        });
```

```
        res.type("application/json").status(200).send(result);
                });
        });
});
```

15. Then run the **srv** module and test again at **/node/ex2/procedures**



16. Let's add another path that calls a stored procedure, but this time we have input parameters on the procedure.

    As the next part of this exercise you can return to **ex2.js** and add another route handler, **/procedures2,** that calls the stored procedure, **get_bp_addresses_by_role**. Notice how we add **:partnerRole?** to the express route definition to process this as a URL parameter.

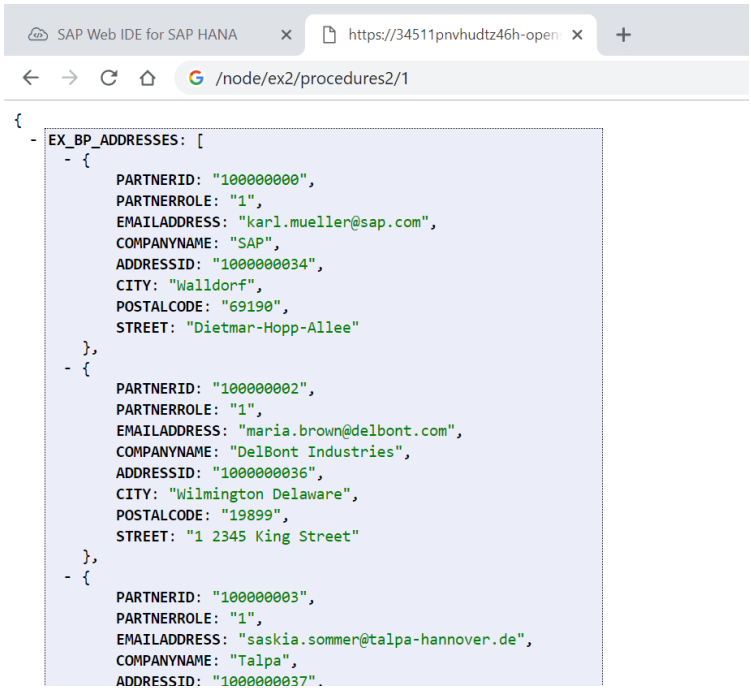
    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db7.js

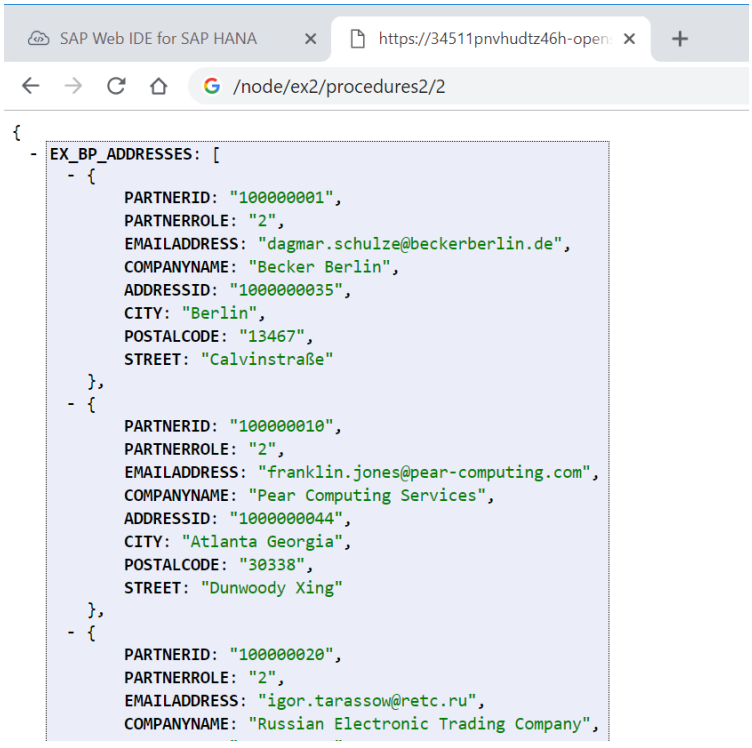
```
        //Database Call Stored Procedure With Inputs
        app.get("/procedures2/:partnerRole?", (req, res) => {
                var client = req.db;
                var hdbext = require("@sap/hdbext");
                var partnerRole = req.params.partnerRole;
                var inputParams = "";
                if (typeof partnerRole === "undefined" ||
partnerRole === null) {
                        inputParams = {};
                } else {
                        inputParams = {
                                IM_PARTNERROLE:
partnerRole
                        };
                }
                //(cleint, Schema, Procedure, callback)
                hdbext.loadProcedure(client, null,
"get_bp_addresses_by_role", (err, sp) => {
                        if (err) {

        res.type("text/plain").status(500).send(`ERROR:
${err.toString()}`);
                                return;
                        }
                        //(Input Parameters, callback(errors,
Output Scalar Parameters, [Output Table Parameters])
```

	sp(inputParams, (err, parameters, results) => { if (err) { res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`); } var result = JSON.stringify({ EX_BP_ADDRESSES: results }); res.type("application/json").status(200).send(result); }); }); });

17. Then run the **srv** module and test again at **/node/ex2/procedures2/1** and **/node/ex2/procedures2/2** to test both partner types – 1=customers and 2=suppliers





18. But we can also take advantage of the parallelization capabilities of Node.js and call two stored procedures in parallel with two separate requests to the

```
//Call 2 Database Stored Procedures in Parallel
app.get("/proceduresParallel/", (req, res) => {
        var client = req.db;
        var hdbext = require("@sap/hdbext");
        var inputParams = {
                IM_PARTNERROLE: "1"
```

database.

A the next part of this exercise you can return to **ex2.js** and add another route handler, **/proceduresParallel,** that calls two previous stored procedures in parallel requests.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db8.js

```
	};
	var result = {};
	async.parallel([

		function(cb) {
			hdbext.loadProcedure(client, null, "get_po_header_data", (err, sp) => {
				if (err) {
					cb(err);
					return;
				}
				//(Input Parameters, callback(errors, Output Scalar Parameters, [Output Table Parameters])
				sp(inputParams, (err, parameters, results) => {

	result.EX_TOP_3_EMP_PO_COMBINED_CNT = results;
					cb();
				});
			});

		},
		function(cb) {
			//(client, Schema, Procedure, callback)
			hdbext.loadProcedure(client, null, "get_bp_addresses_by_role", (err, sp) => {
				if (err) {
					cb(err);
					return;
				}
				//(Input Parameters, callback(errors, Output Scalar Parameters, [Output Table Parameters])
				sp(inputParams, (err, parameters, results) => {

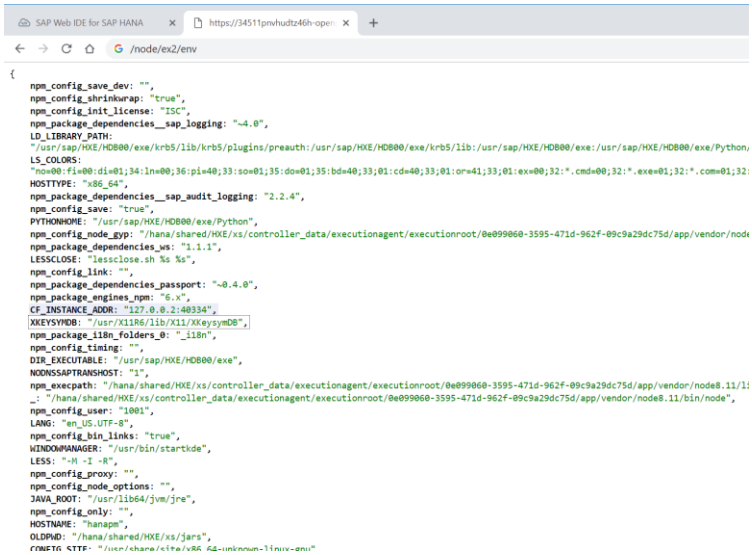
	result.EX_BP_ADDRESSES = results;
					cb();
				});
			});
		}
	], (err) => {
		if (err) {

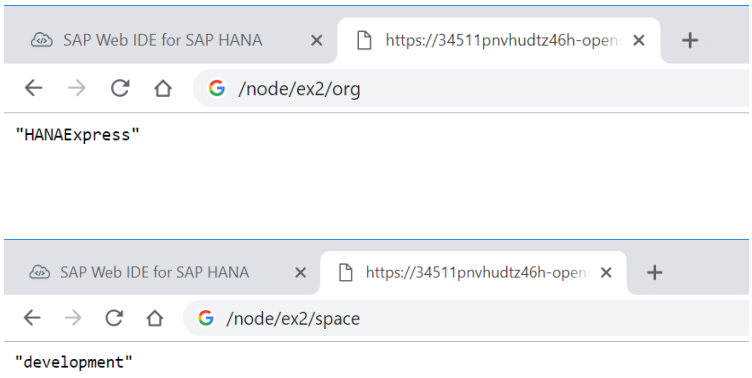
	res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
		} else {

	res.type("application/json").status(200).send(JSON.stringify(result));
		}
```

	}); });
19. Then run the **srv** module and test again at **/node/ex2/proceduresParallel**	
20. Now let's recreate a few of the XSJS samples from Exercise 2.1. We will begin with the **whoAmI** exercise. In 2.1 we used the **$.session** to get the current user. However, in pure Node.js we don't have the $ APIs. Instead we can read the same information, and more, from the request **authInfo** object. As the next part of this exercise you can return to **ex2.js** and add another route handler, **/whoAmI,** that outputs the **req.authInfo**. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex	app.get("/whoAmI", (req, res) => { var userContext = req.authInfo; var result = JSON.stringify({ userContext: userContext }); res.type("application/json").status(200).send(result); });

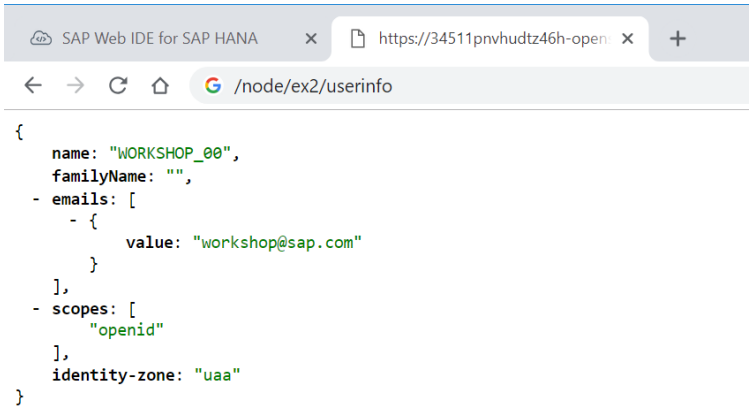
5/db9.js	
21. Then run the **srv** module and test again at **/node/ex2/whoAmI**	
22. But there is even more information which we can access via Node.js about the XSA/Cloud Foundry environment. The built-in **process** object gives us access to this information via the environment. As the next part of this exercise you can return to **ex2.js** and add another route handler, **/env,** that outputs the **process environment**. **Note**: if you don't want to type this code, we recommend that you cut and	app.get("/env", (req, res) => { return res.type("application/json").status(200).send(JSON.stringify(process.env)); });

paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db10.js	
23. Then run the **srv** module and test again at **/node/ex2/env**	
24. This environment information has so much detail that it can be overwhelming. Two of the most important parts are the **VCAP_APPLICATION** and **VCAP_SERVICES**. These give you access to all kinds of XSA/CF details about the current application and its bound services. As the next part of this exercise you can return to **ex2.js** and add two more route handlers, **/org** and **/space,** that outputs the **VCAP** details for the current XSA/CF organization and space**.** **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-	app.get("/org", (req, res) => { let VCAP = JSON.parse(process.env.VCAP_APPLICATION); return res.type("application/json").status(200).send(JSON.stringify(VCAP.organization_name)); }); app.get("/space", (req, res) => { let VCAP = JSON.parse(process.env.VCAP_APPLICATION); return res.type("application/json").status(200).send(JSON.stringify(VCAP.space_name)); });

hana7/blob/snippets_2.3.2/ex5/db11.js	
25. Then run the **js** module and test again at **/node/ex2/org** and **/node/ex2/space**	
26. Now let's combine what we've learned about the **req.authInfo** and also use the **@sap/xssec** module to build a security context which could be used for custom scope checks. As the next part of this exercise you can return to **ex2.js** and another route handler, **/userinfo,** that outputs the combined info about the user details and scopes**.** **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db12.js	app.get("/userinfo", function(req, res) { let xssec = require("@sap/xssec"); let xsenv = require("@sap/xsenv"); let accessToken; let authWriteScope = false; let authReadScope = false; let controllerAdminScope = true; let userInfo = { "name": req.user.id, "familyName": req.user.name.familyName, "emails": req.user.emails, "scopes": [], "identity-zone": req.authInfo.identityZone }; function getAccessToken(req) { var accessToken = null; if (req.headers.authorization && req.headers.authorization.split(" ")[0] === "Bearer") { accessToken = req.headers.authorization.split(" ")[1]; } return accessToken; } accessToken = getAccessToken(req); let uaa = xsenv.getServices({ uaa: { tag: "xsuaa" } }).uaa;

```
                xssec.createSecurityContext(accessToken,
uaa, function(error, securityContext) {
                    if (error) {
                        console.log("Security Context
creation failed");
                        return;
                    }
                    console.log("Security Context created
successfully");
                    userInfo.scopes =
securityContext.scopes;
                    console.log("Scope checked
successfully");
                });
                return
res.type("application/json").status(200).json(userInfo);
            });
```

27. Then run the **srv** module and test again at **/node/ex2/userinfo**



28. We will continue converting xsjs exercise from earlier. For this step, convert the **hdb.xsjs** example that read from the Purchase Order table and output the results as JSON.

    As the next part of this exercise you can return to **ex2.js** and add a seventh route handler, **/hdb,** that outputs the results of a selection from **PurchaseOrder.Item**.

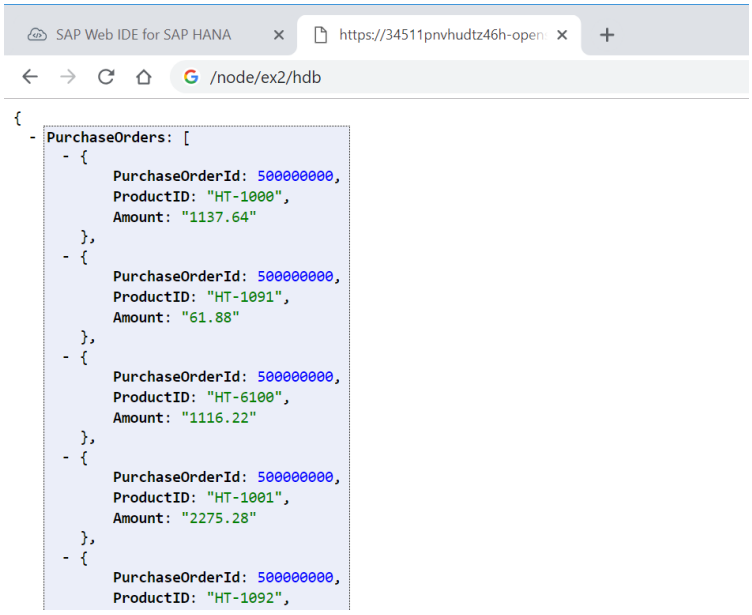
    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address

```
        app.get("/hdb", async(req, res) => {
            try {
                const dbClass = require(global.__base
+ "utils/dbPromises");
                let dbConn = new dbClass(req.db);
                const statement = await
dbConn.prepareePromisified(
                    `SELECT FROM
OPENSAP_PURCHASEORDER_ITEMVIEW {
                      PURCHASEORDERID as
"PurchaseOrderId",
                    PRODUCT as "ProductID",
                    GROSSAMOUNT as
"Amount"  } `
                );
                const results = await
dbConn.statementExecPromisified(statement, []);
                let result = JSON.stringify({
                    PurchaseOrders: results
                });
```

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db13.js

```
                return
res.type("application/json").status(200).send(result);
            } catch (err) {
                return
res.type("text/plain").status(500).send(`ERROR:
${err.toString()}`);
            }
        });
```

29. Then run the **srv** module and test again at **/node/ex2/hdb**



30. For the final conversion, let's bring over **os.xsjs**. From pure Node.js we can simply require the **os** module.

    As the next part of this exercise you can return to **ex2.js** and add another route handler, **/os,** that outputs the results from a call to the **os** module.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db14.js

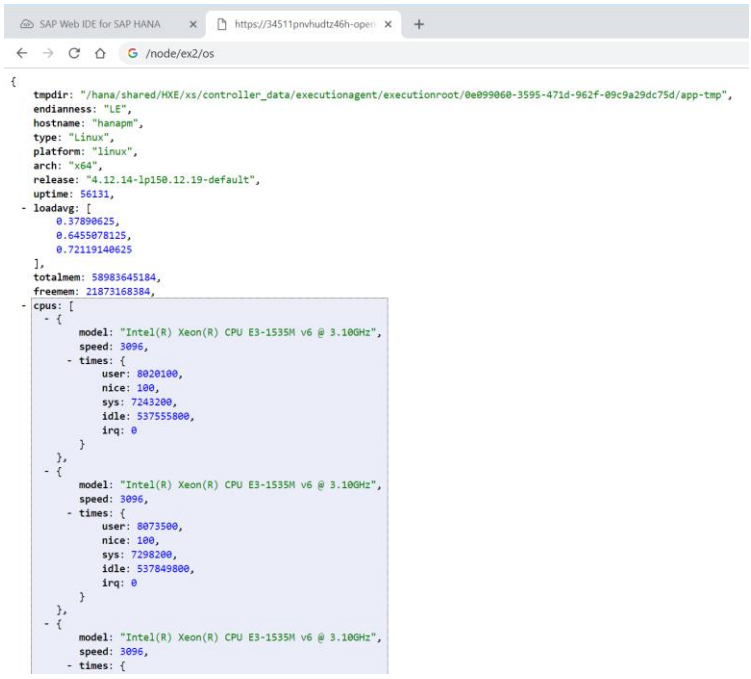
```
app.get("/os", (req, res) => {
        var os = require("os");
        var output = {};

        output.tmpdir = os.tmpdir();
        output.endianness = os.endianness();
        output.hostname = os.hostname();
        output.type = os.type();
        output.platform = os.platform();
        output.arch = os.arch();
        output.release = os.release();
        output.uptime = os.uptime();
        output.loadavg = os.loadavg();
        output.totalmem = os.totalmem();
        output.freemem = os.freemem();
        output.cpus = os.cpus();
        output.networkInfraces =
os.networkInterfaces();

        var result = JSON.stringify(output);

    res.type("application/json").status(200).send(result);
    });
```

31. Then run the **srv** module and test again at **/node/ex2/os**



32. But the interaction with the OS isn't limited to this one module. Node.js can fork a child process which runs directly at the Linux OS level of the application server. For this we can use the child_process built-in module.

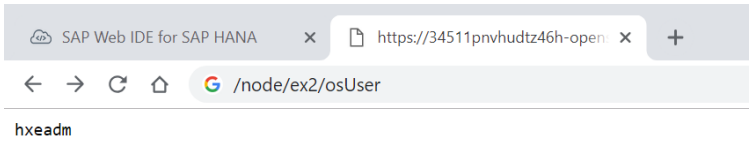
    As the next part of this exercise you can return to **ex2.js** and add another route handler, **/osUser,** that outputs the results from a Linux Shell command **whoami**.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/db15.js

```
app.get("/osUser", (req, res) => {
        var exec = require("child_process").exec;
        exec("whoami", (err, stdout, stderr) => {
                if (err) {

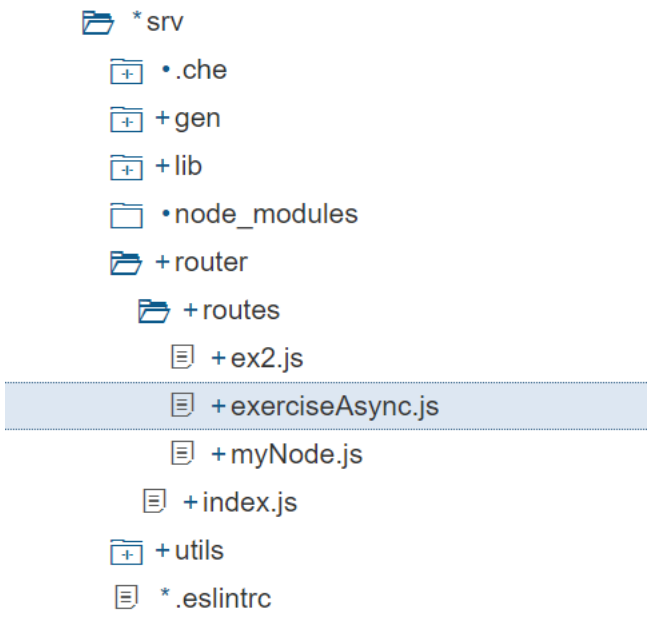
res.type("text/plain").status(500).send(`ERROR: ${err.toString()}`);
                        return;
                } else {

res.type("text/plain").status(200).send(stdout);
                }
        });
});
```

33. Then run the **js** module and test again at **/node/ex2/osUser**	

Exercise 4.4: Asynchronous Non-Blocking I/O

In this exercise, you will learn about the fundaments of the asynchronous nature of Node.js. We will also see how this asynchronous capability allows for non-blocking input and output. This technique is one of the basic things that makes Node.js development different from other JavaScript development and also creates one of the reasons for its growing popularity. We will see how these techniques are applied to common operations like HTTP web service calls or even SAP HANA database access.

Explanation	Screenshot
1. Return to the **srv** module and the **router/index.js** source file. Add a second module require statement for **./exerciseAsync**	"use strict"; module.exports = function(app,server) { app.use("/node", require("./routes/myNode")()); app.use("/node/ex2", require("./routes/ex2")()); **app.use("/node/excAsync", require("./routes/exerciseAsync")(server));** };
2. Create a new file in your router/routes folder called exerciseAsync.js.	
3. Add the following code to your **exerciseAsync.js** file. You can look at this code if you want, but isn't what we want you to focus on for this exercise. Instead this is really just a test framework. We will instead write the	/*eslint no-console: 0, no-unused-vars: 0, new-cap:0 */ /*eslint-env node, es6 */ "use strict"; var express = require("express"); var WebSocketServer = require("ws").Server; module.exports = function (server) { var app = express.Router();

important parts of this exercise in a series of additional Node.js modules – each focusing on a different Node.js asynchronous aspect.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_7

```
	const asyncLib = require(global.__base +
"async/async.js");
	const dbAsync = require(global.__base +
"async/databaseAsync.js");
	const dbAsync2 = require(global.__base +
"async/databaseAsync2.js");
	const fileSync = require(global.__base +
"async/fileSync.js");
	const fileAsync = require(global.__base +
"async/fileAsync.js");
	const httpClient = require(global.__base +
"async/httpClient.js");

	app.use((req, res) => {
		var output =
			`<H1>Asynchronous
Examples</H1></br>
			<a
href="/exerciseAsync">/exerciseAsync</a> - Test Framework
for Async Examples</br>`;
		res.type("text/html").status(200).send(output);
	});
	try {
		var wss = new WebSocketServer({
			//server: server
			noServer: true
		});

		server.on("upgrade", function
upgrade(request, socket, head) {
			const url = require("url");
			const pathname =
url.parse(request.url).pathname;

			if (pathname === "/node/excAsync") {
				wss.handleUpgrade(request,
socket, head, function done(ws) {

	wss.emit("connection", ws, request);
				});
			}
		});

		wss.broadcast = (data) => {
			var message = JSON.stringify({
				text: data
			});
			wss.clients.forEach(function
each(client) {
				try {
					client.send(message,
function ack(error) {
						if (typeof error
!== "undefined") {
```

```
                console.log(`Send Error: ${error.toString()}`);
                                                }
                                        });
                                } catch (e) {

                console.log(`Broadcast Error: ${e.toString()}`);
                                }
                        });
                        console.log(`Sent: ${message}`);

                };
                wss.on("error", (error) => {
                        console.log(`Web Socket Server Error:
${error.toString()}`);
                });

                wss.on("connection", (ws) => {
                        console.log("Connected");

                        ws.on("message", (message) => {
                                console.log(`Received:
${message}`);
                                var data =
JSON.parse(message);
                                switch (data.action) {
                                case "async":

                asyncLib.asyncDemo(wss);
                                        break;
                                case "fileSync":

                fileSync.fileDemo(wss);
                                        break;
                                case "fileAsync":

                fileAsync.fileDemo(wss);
                                        break;
                                case "httpClient":

                httpClient.callService(wss);
                                        break;
                                case "dbAsync":
                                        dbAsync.dbCall(wss);
                                        break;
                                case "dbAsync2":

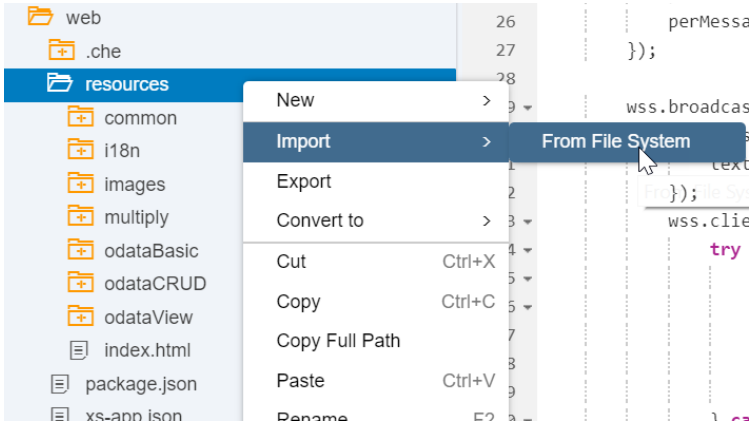
                dbAsync2.dbCall(wss);
                                        break;
                                default:
                                        wss.broadcast(`Error:
Undefined Action: ${data.action}`);
                                        break;
                                }
```

```
                        });
                        ws.on("close", () => {
                                console.log("Closed");
                        });
                        ws.on("error", (error) => {
                                console.log(`Web Socket
Error: ${error.toString()}`);
                        });
                        ws.send(JSON.stringify({
                                text: "Connected to Exercise
3"
                        }), function ack(error) {
                                if (typeof error !==
"undefined") {
                                        console.log(`Send
Error: ${error.toString()}`);
                                }
                        });
                });
        } catch (e) {
                console.log(`General Error: ${e.toString()}`);
        }
        return app;
};
```

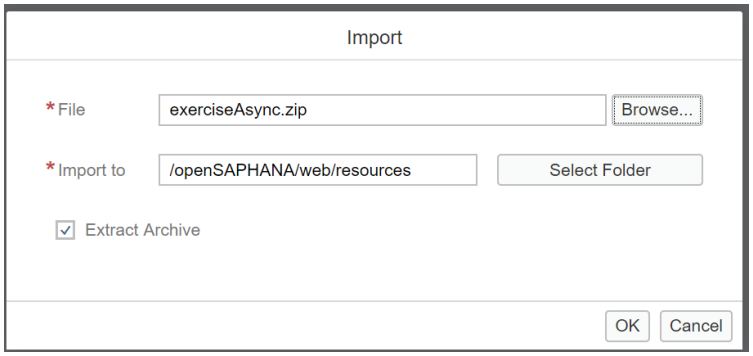
4. We will also need a test user interface for this exercise.

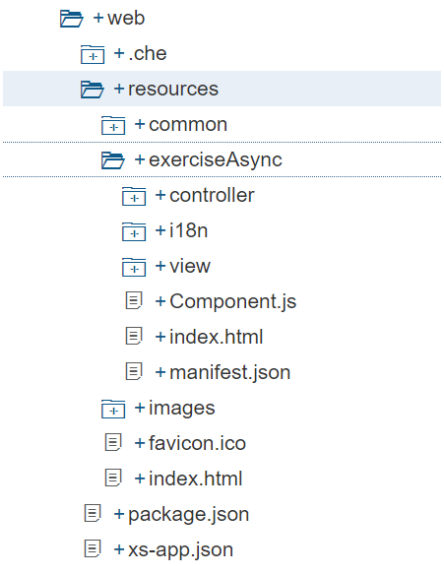
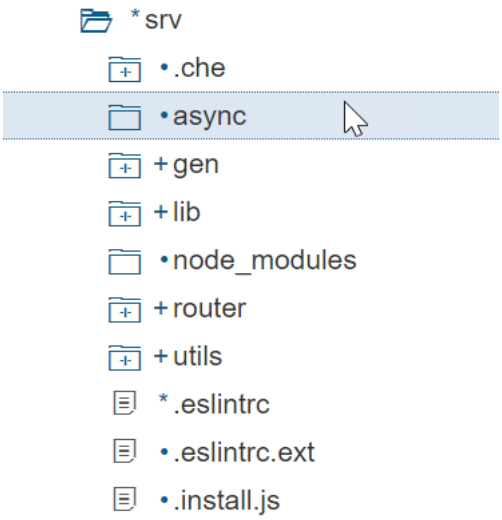
   Download the content from this address: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/exerciseAsync.zip

   Go to the web/resources folder and right mouse click. Choose Import->From File System.



5. Press browse and file the file **exerciseAsync.zip** in the Downloads folder. Keep all other selections at their defaults and press **OK**.

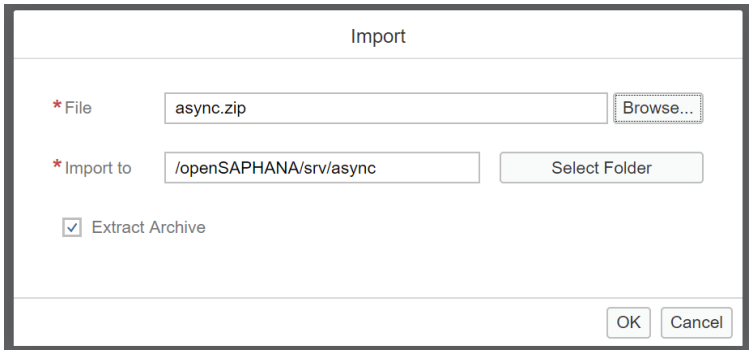


6. This should import the complete UI for you.	
7. In the web module, also edit the **xs-app.json**. Add the Websockets enabled section and a logout after the authenticationMethod.	"logout": { "logoutPage": "/logout.html", "logoutEndpoint": "/my/logout" }, "websockets": { "enabled": true },
8. Create a folder under **srv** called **async**. This is where we will create the rest of the files in this exercise part.	

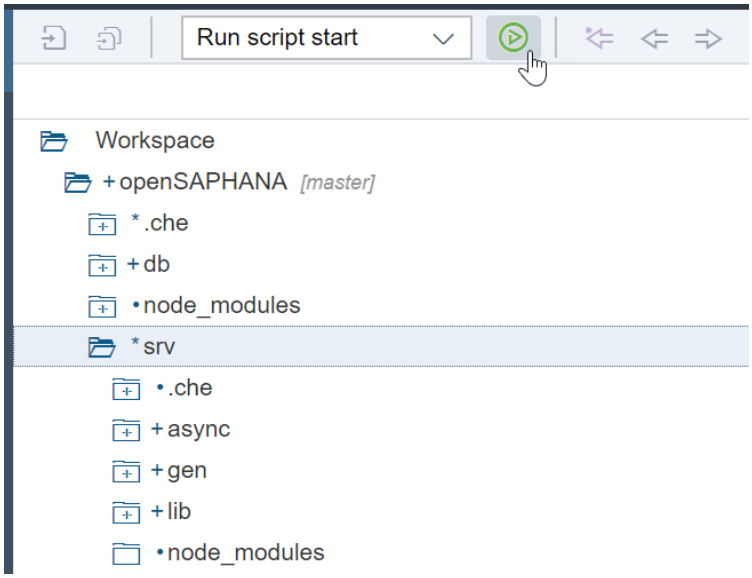
9. **Download** the rest of the files from the following address: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/async.zip

   Right mouse click on **/srv/async** and choose **Import -> From File System**.

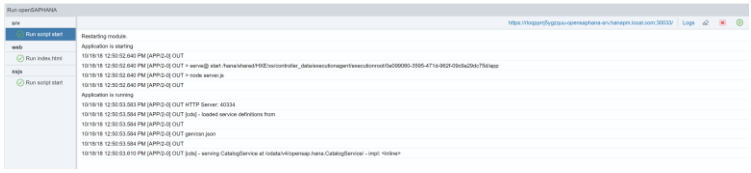
   Choose the file **async.zip** from the Download folder on your local machine. Keep all other settings the same and press OK.



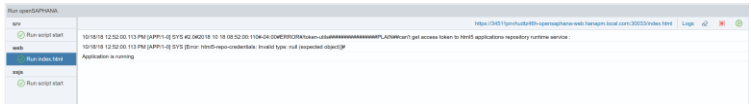
10. We can now run the **srv** module.



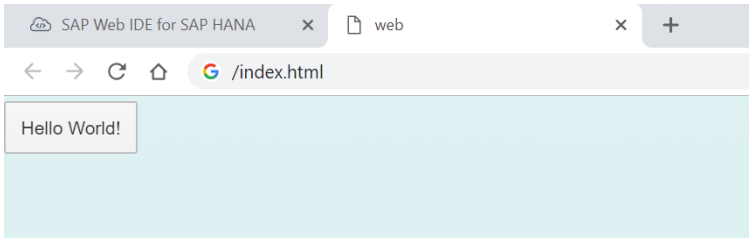
11. You should see that the build and deploy was successful.



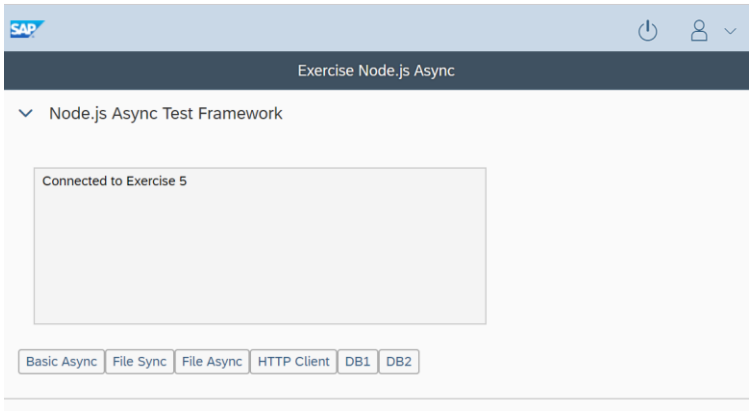
12. So now run the web module since we have new frontend content as well.



13. In the running tab, you should see the index.html from earlier.



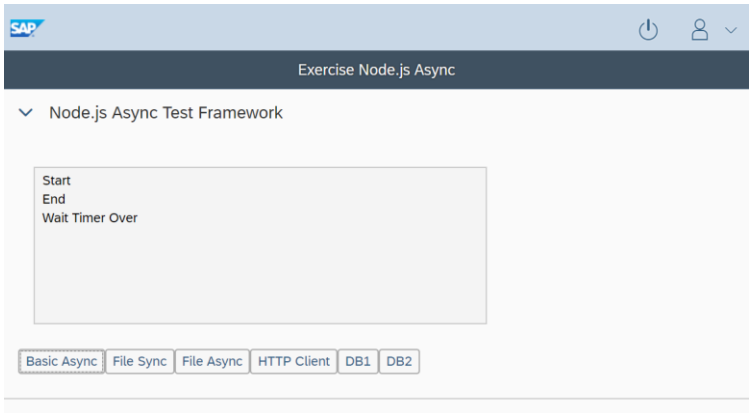
14. Now change the path in the browser to **/exerciseAsync**. You should see the test framework for this exercise.



15. Basic Async runs the **async.js** code. In this code we will output a start message, then set a timer which will issue a message after 3 seconds. Finally we will issue an ending message.



16. What do you expect this code will output? From many other programming languages we would expect sequential processing and therefore the End output wouldn't come until after the timer expired. However part of the power of node.js is asynchronous non-blocking execution of many core elements. In test UI, press the Basic Async button.



17. Perhaps a timer seemed like an obvious asynchronous operation. However this asynchronous nature of node.js is often used when programs must wait on input or output. When asynchronous processing is applied to these operations, we can keep from blocking execution of other logic while we wait on things like file access, http requests or even database query execution.

    Let's first look at the difference between synchronous and asynchronous file operations.

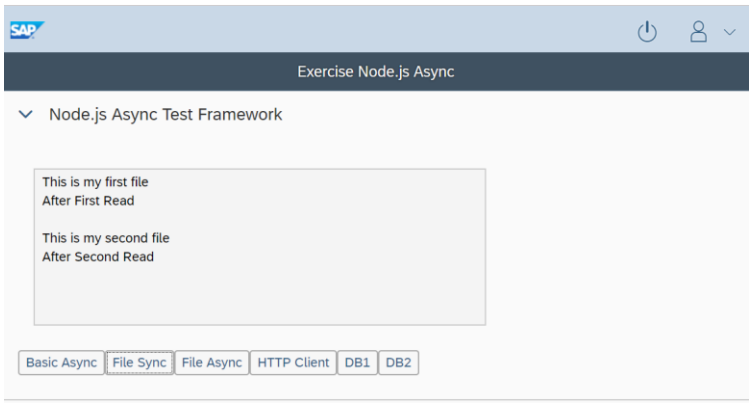
18. The fileSync.js is using the fs library and the function **readFileSync** to read each of the two text files. After each read operation output there is a message.

fileSync.js ×

```
1  /*eslint-env node, es6 */
2  "use strict";
3  var fs = require("fs");
4
5  module.exports = {
6      fileDemo: (wss) => {
7          var text = fs.readFileSync("./async/file.txt", "utf8");
8          wss.broadcast(text);
9
10         wss.broadcast("After First Read\n");
11
12         text = fs.readFileSync("./async/file2.txt", "utf8");
13         wss.broadcast(text);
14
15         wss.broadcast("After Second Read\n");
16
17     }
18 };
```

19. Test your **fileSync.js** from the UI test tool.

    As you might expect, everything is output in exactly the same order as the lines of code were listed in the application because all operations were synchronous. Program execution didn't continue until each read operation had completely finished.



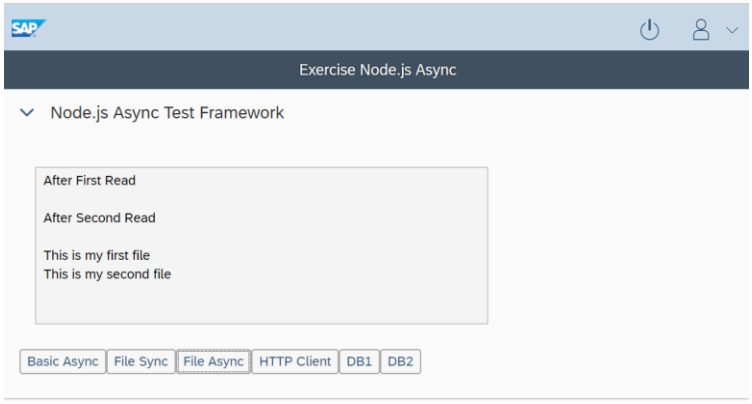
20. Now to look at **fileAsync.js**. Notice that the **message output** now is embedded as an in-line callback function. It doesn't get executed until the read operation is complete, but the rest of the program flow continues and isn't blocked by the file operation.

fileAsync.js

```
1  /*eslint-env node, es6 */
2  "use strict";
3  var fs = require("fs");
4
5  module.exports = {
6      fileDemo: (wss) => {
7          fs.readFile("./async/file.txt", "utf8", (error, text) => {
8              wss.broadcast(text);
9          });
10         wss.broadcast("After First Read\n");
11
12         fs.readFile("./async/file2.txt", "utf8", (error, text) => {
13             wss.broadcast(text);
14         });
15         wss.broadcast("After Second Read\n");
16
17     }
18 };
```

21. Now run **fileAsync.js** from the test UI. The output of this exercise gives us very different results. Both after comments are output before either of the file contents. Also if the first file had been significantly larger than the second, it's possible that the second might have finished and output first.

    This has powerful implications to how we code applications.



22. Similar to file operations, HTTP requests are another area where our programs must often wait on an external response. In this exercise let's see how **node.js** also makes calling external HTTP services non-blocking.

    The **http** library we used in earlier exercises can also be used to make HTTP requests. Use the **get** function of the **http** library to call to http://www.loc.gov/pictures/search/?fo=json&q=SAP. This will call the US Library of Congress Image Search (a REST API which requires no authentication or API Key to keep the exercise simple). Issue a message before and after the HTTP request.

httpClient.js

```
/*eslint-env node, es6 */
"use strict";
var http = require("http");

module.exports = {
    callService: (wss) => {

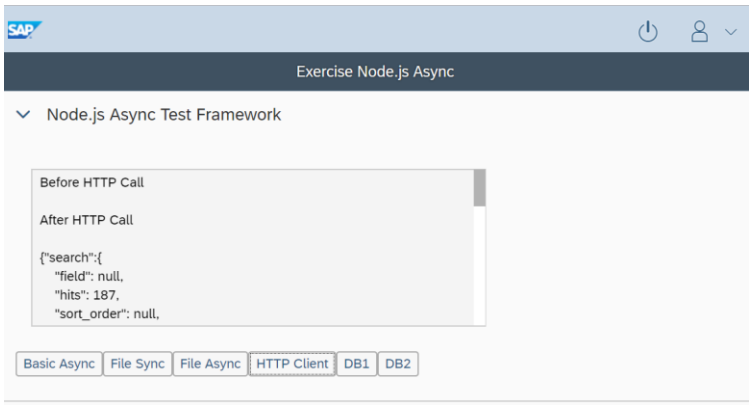
        wss.broadcast("Before HTTP Call\n");

        try {
            http.get({
                    path: "http://www.loc.gov/pictures/search/?fo=json&q=SAP&",
                    host: "www.loc.gov",
                    port: "80",
                    headers: {
                        host: "www.loc.gov"
                    }
                },
                function(response) {
                    response.setEncoding("utf8");
                    response.on("data", (data) => {
                        wss.broadcast(data.substring(0, 100));
                    });
                    response.on("error", wss.broadcast);
                });
        } catch (err) {
            wss.broadcast(err.toString());
        }
        wss.broadcast("After HTTP Call\n");

    }
};
```

23. Test your **httpClient.js** from the test UI.

    Similar to the earlier file exercise, the after http call console message is output before the response from the HTTP request.



24. Perhaps most interesting to us is that this non-blocking concept can also be extended to database access. This allows us to issue multiple requests to the underlying HANA database in parallel and without stopping the processing flow of the JavaScript application logic.

    Earlier in this exercise, you learned about making database access to HANA. For this exercise we've already coded the database requests in a reusable module for you, so you can concentrate on the asynchronous flow.

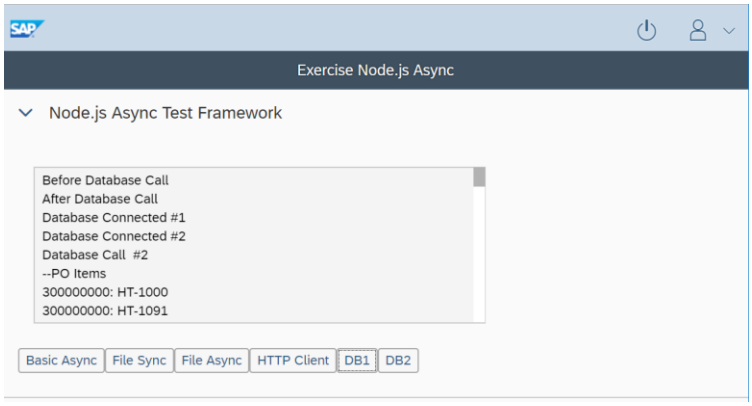
    First we have **databaseAsync.js**. This issues a message, then calls two functions (**callHANA1** and **callHANA2**), then issues another message. This will execute two different queries in the HANA database.

databaseAsync.js

```
1  /*eslint-env node, es6 */
2  "use strict";
3  var hana = require("./database");
4  module.exports = {
5      dbCall: (wss) => {
6          function dummy() {}
7          wss.broadcast("Before Database Call");
8          hana.callHANA1(dummy, wss);
9          hana.callHANA2(dummy, wss);
10         wss.broadcast("After Database Call");
11     }
12 };
```

25. Test your **databaseAsync.js** from test UI.

    As you are hopefully learning to expect, the messages you issued after the database requests are actually output first. Only then are the database query results returned. There is also no guarantee that query 1 will finish before query 2.



26. But what if you want more control over the flow of program execution. Maybe you want several database operations to happen in parallel, but then some logic to execute only after all queries are complete. This is one of things the **async** library in node.js can make easier.

    In **databaseAsync2.js** we adjust the logic to use the **async.parallel** function. This allows some of the commands to execute in parallel as before, but then have a sync point once all operations are complete to allow further processing. We will output one final message after everything is done.

databaseAsync2.js

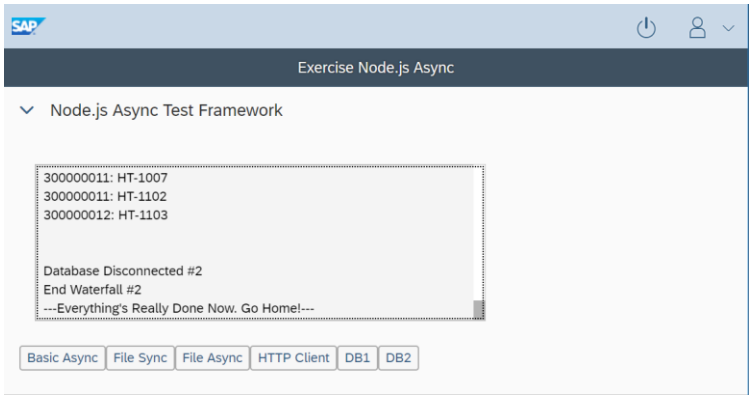
```
/*eslint no-console: 0, no-unused-vars: 0, no-undef: 0, no-shadow: 0*/
/*eslint-env node, es6 */
"use strict";
var hana = require("./database");
var async = require("async");

module.exports = {
    dbCall: (wss) => {
        async.parallel([
            function(cb) {
                wss.broadcast("Before Database Call");
                cb();
            },
            function(cb) {
                hana.callHANA1(cb, wss);
            },
            function(cb) {
                hana.callHANA2(cb, wss);
            },
            function(cb) {
                wss.broadcast("After Database Call");
                cb();
            }
        ], function(err) {
            wss.broadcast("---Everything's Really Done Now. Go Home!---");
        });

    }
};
```

27. Test your **databaseAsync2.js** from the test UI.

    The execution is similar to before, but now we have the final message after all queries are complete. Because we have a sync point after all parallel execution is complete, we can output this final message after both queries are complete.



Exercise 4.5: Exploring JavaScript Language Features

Explanation	Screenshot
1. Using what you have learned already, let's migrate the JavaScript Basics in Exercise 3.3 from XSJS to pure Node.js. Extend the router **index.js** with a path for **JavaScriptBasics**.	/*eslint-env node, es6 */ "use strict"; module.exports = (app, server) => { app.use("/node", require("./routes/myNode")()); app.use("/node/ex2", require("./routes/ex2")()); app.use("/node/excAsync", require("./routes/exerciseAsync")(server)); **app.use("/node/JavaScriptBasics",** **require("./routes/JavaScriptBasics")());** };
2. In the **srv/router/routes** folder add the file **JavaScriptBasics.js** with the same express router structure we've used throughout Exercise 4.	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0, dot-notation:0, no-use-before-define:0 */ /*eslint-env node, es6 */ "use strict"; var express = require("express"); module.exports = function() { var app = express.Router(); //Hello Router app.get("/", (req, res) => { res.send("JavaScript Basics Exercise"); }); return app; };
3. This exercise will be a bit different from all others. Instead of following along step-by-step or working toward a very specific final product; this exercise will instead allow you to explore various JavaScript language features. Specifically we use what we've just learned about Node.js non-blocking, callback nature to adjust the XSJS exercises from earlier to pure Node.js There is no strict right or wrong solution to many of these exercise parts. However, you can view sample solutions from:	

https://github.com/SAP/hana-xsa-opensap-hana7/blob/master/srv/router/routes/JavaScriptBasics.js

Feel free to create similar Node.js routes to explore these language features. However, the main point of this exercise is to focus on general JavaScript language constructs and how to adapt them to unique aspects of Node.js.

4. First we will learn about strings in JavaScript.

   Create a string with some sample text in it (hint: =).

   Access the first character of the string. (hint: offsets are zero based and specified via [])

   Access the last character in the string (hint: slice).

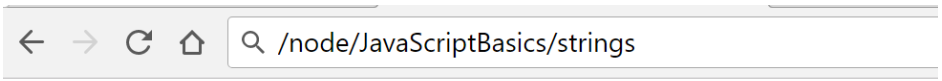
   Output the length of the string (hint: length).

   Convert the string to all upper and all lower case (hint: toUpperCase).

   Find the position of a text part of the string (hint: indexOf).

   Find the last occurrence of a character in the string. Perform a find and replace operation on the string (hint: lastIndexOf).

   Sample implementation: **/strings**



Basic String: SAP HANA Extended Application Services

The first character in the string: S

The length of the string: 38

The last character in the string: s

Upper: SAP HANA EXTENDED APPLICATION SERVICES

Lower: sap hana extended application services

Find HANA: 4

Find Last occurance of the letter A: 18

Replace with XS: SAP HANA XS

Template Literals in ES6!

Multi-line string
second line

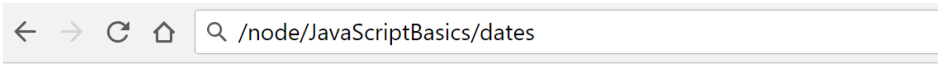
5. Create a Data object with the current date (hint: new Date()).

   Convert the data to various other formats (UTC, Local Data, Local Time, ISO, JSON (hint: to*String with various modifies UTC, Date, LocalDate, LocaleTime, etc).

   Access the sub-parts of the date: Year, Month, Day, Hours, Minutes, and Seconds (hint: get* with modifiers for FullYear, Month, Day, Date, Time, Hours, Minutes, Seconds).

   Perform math on a date, adding 30 days to the current date (hint: setDate and simple math).

   Sample implementation: **/dates**



Now: Thu Aug 10 2017 13:49:16 GMT-0400 (EDT)

Now UTC: Thu, 10 Aug 2017 17:49:16 GMT

Now Date String: Thu Aug 10 2017

Now Locale Date String: 8/10/2017

Now Locale Time String: 1:49:16 PM

Now Locale String: 8/10/2017, 1:49:16 PM

Now ISO String: 2017-08-10T17:49:16.821Z

Now JSON String: 2017-08-10T17:49:16.821Z

Now Year: 2017

Now Month: 7

Now Day of Week: 4

Now Day of Month: 10

Now number of milliseconds since midnight Jan 1, 1970: 1502387356821

Now Hours: 13

Now Minutes: 49

Now Seconds: 16

30 days from now: Sat Sep 09 2017 13:49:16 GMT-0400 (EDT)

6. Create two arrays: one with the colors Red, Green, and Blue and one with Black, White, Orange, and Purple (hint: [“one”, “two, “three”]).

   Output the first array as a string (hint: toString).

   Access the first element by index (hint: zero based).

   Get the number of elements in the first array (hint: length).

   Get the index of Blue in the first array (hint: indexOf).

   Loop over the first array and output each element (hint: for(var i; i < length; i++).

   Combine the two arrays (hint: concat).

   Reverse Sort and then Sort Ascending (hint: reverse, sort).

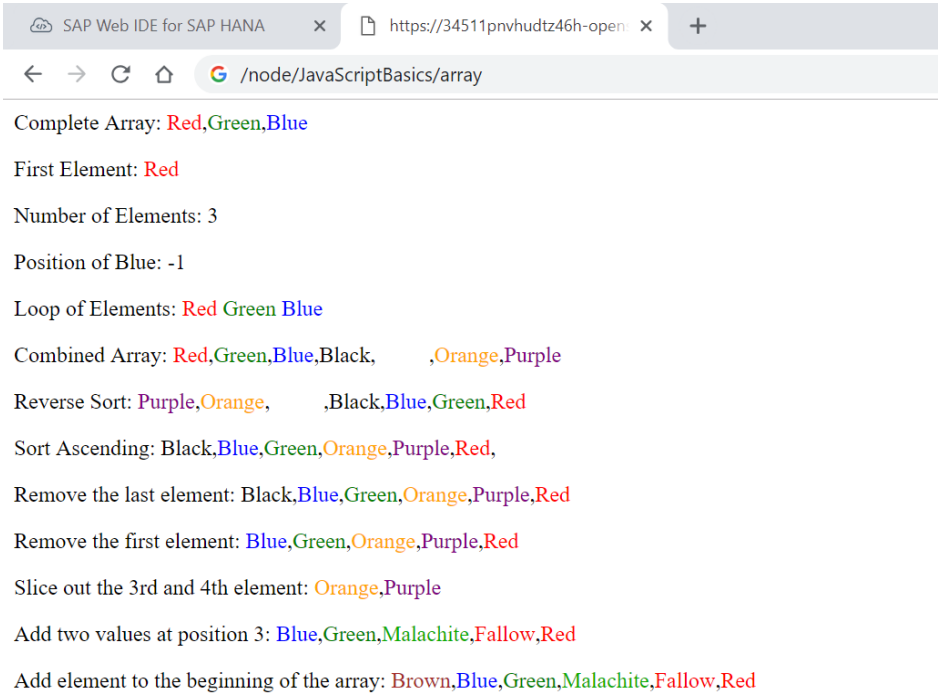
   Remove the last and then first element (hint: pop, shift).

   Slice out the 3rd and 4th element (hint: slice).

   Add two new colors (Malachite and Fallow) at the 3rd position (hint: splice).

   Add Brown to the start of the Array (hint: unshift).

   Sample implementation: **/array**

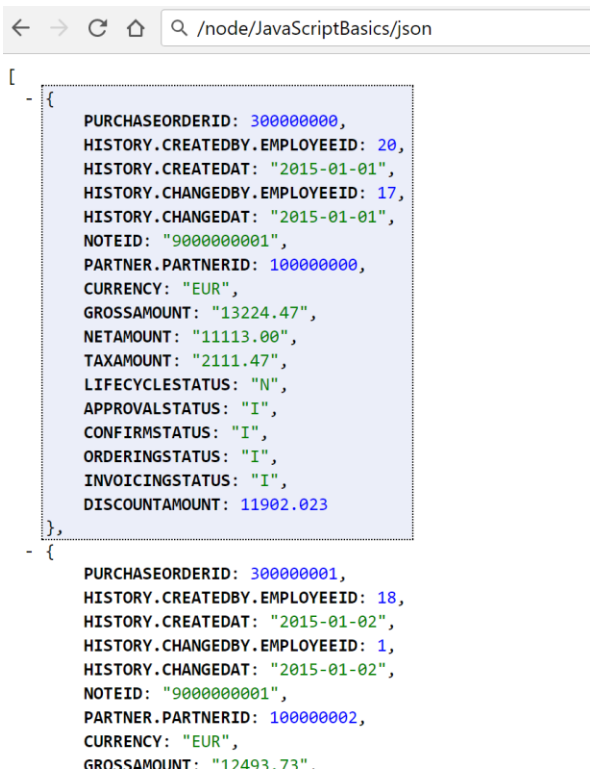


7. Select 10 records from **PO.Header** and convert the results to JSON (hint: req.db and callbacks).

   Loop over the elements in the JSON object and create a new **DISCOUNTAMOUNT** attribute by discounting the **GROSSAMOUNT** by 10% (hint: for (var i; i < length; i++).

   Convert the JSON object to a string and output it to the frontend (hint: JSON.stringify).

   Sample implementation: **/json**

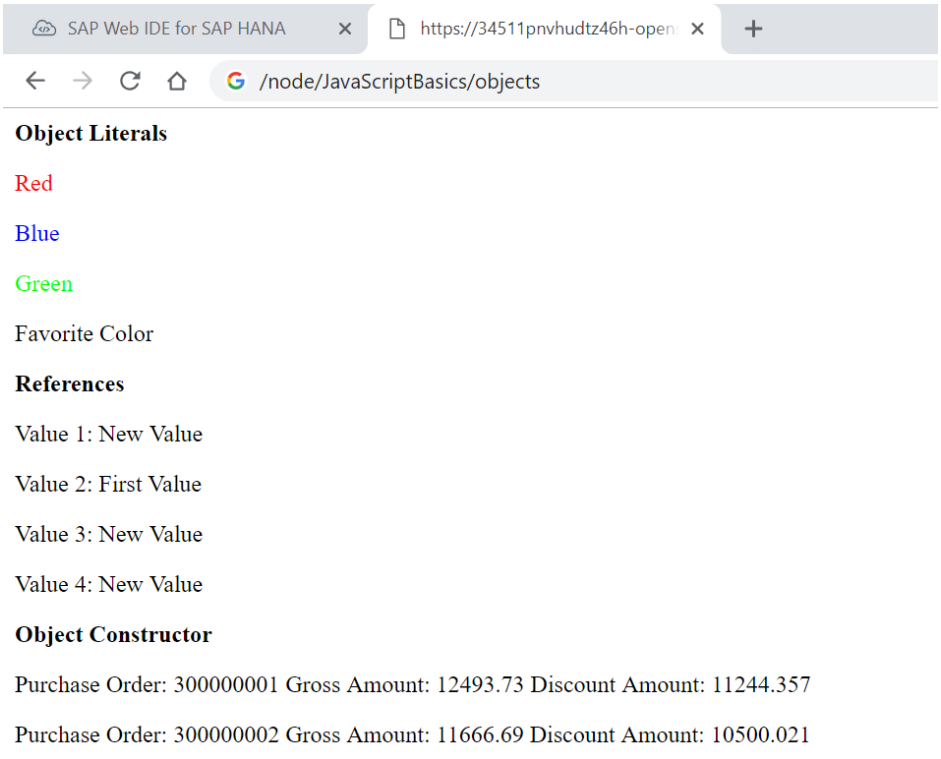


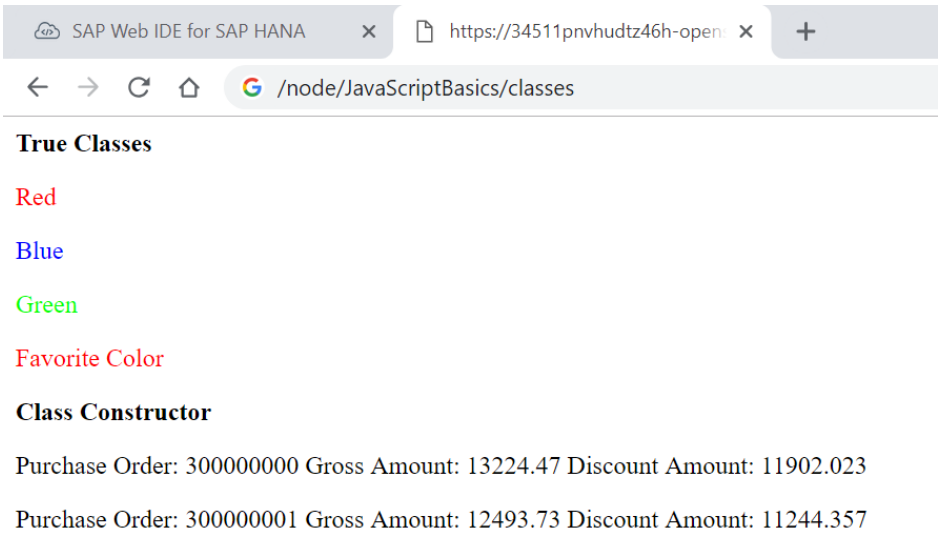
8. Create an object literal called colors. Give it three properties – red with the value #FF0000, green with the value #00FF00, and blue with the value #0000FF. Give the object a function named favoriteColor. On Monday, it should return blue, all other days it returns red (hint: var colors = {}).

   Access the colors in the object including the Favorite color (hint: object.<variable>).

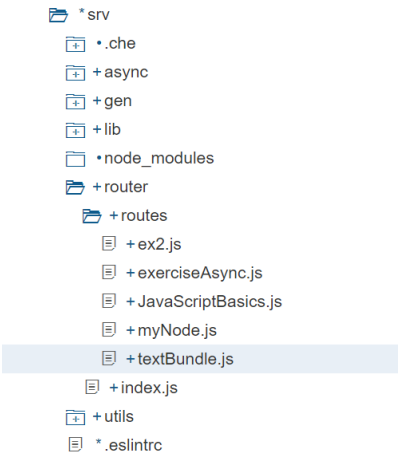
   Prove that objects are assign by reference and not assign by value (hint: just set a value to a regular variable and an object and see the difference).

   Create a purchase order object for **PO.Header** which returns PO Id, gross



	amount, and a calculated discount amount (10% off gross amount) (hint: function with logic from last step – json.xsjs). Create two instances of this object – one for PO 300000000 and one for 300000001 (hint: use async.parallel so you have a sync point after both objects are finished) Sample implementation: **/objects**	
9.	Now adjust the objects example from the previous step and cover the Purchase Order section to be a true ES6 Class. Create a purchase order object for **PO.Header** which returns PO Id, gross amount, and a calculated discount amount (10% off gross amount) (hint: use a factory pattern to allow the asynchronous loading of the record from the DB within the constructor of the main class). Create two instances of this object – one for PO 300000000 and one for 300000001 (hint: use **async/await** so you have a sync point after both objects are finished before filling the output body string) Sample implementation: **/classes**	

Exercise 4.6: Text Bundles

Explanation	Screenshot
1. Return to the **srv** module.	
2. Add a route for a new module named textBundle that corresponds to **/node/textBundle** in the **router/index.js**	/*eslint-env node, es6 */ "use strict"; module.exports = (app, server) => { app.use("/node", require("./routes/myNode")()); app.use("/node/ex2", require("./routes/ex2")()); app.use("/node/excAsync", require("./routes/exerciseAsync")(server)); app.use("/node/JavaScriptBasics", require("./routes/JavaScriptBasics")()); **app.use("/node/textBundle", require("./routes/textBundle")());** };
3. Create a new file in your **router/routes** folder called **textBundle.js**.	
4. Add the following code to your textBundle.js file. It has implemented an HTTP handler for the root URL using **express**. In the processing of this handler use the **TextBundle** library. When creating a new **TextBundle** instance the input are **path** (value messages to point to your message.properties file) and **locale** which should call **getLocale** passing in the **req**	/*eslint no-console: 0, no-unused-vars: 0, consistent-return: 0, new-cap: 0*/ /*eslint-env node, es6 */ "use strict"; var express = require("express"); var app = express.Router(); var os = require("os"); var TextBundle = require("@sap/textbundle").TextBundle; var langparser = require("accept-language-parser"); function getLocale(req) { var lang = req.headers["accept-language"]; if (!lang) { return;

object. Then write into the **res** object with the **TextBundle getText** function (passing in the text ID of greeting and two parameters of **os.hostname()** and **os.type().**

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_8

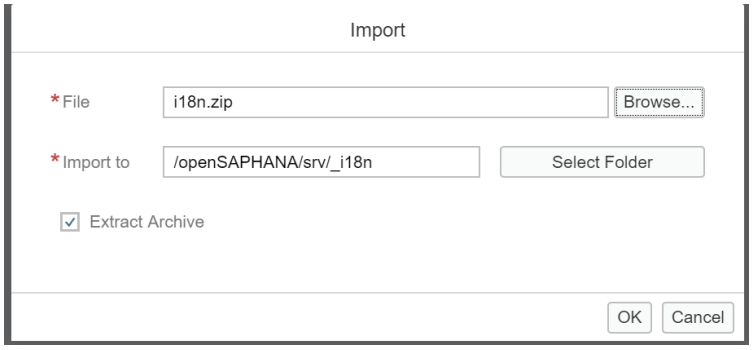
```
        }
        var arr = langparser.parse(lang);
        if (!arr || arr.length < 1) {
                return;
        }
        var locale = arr[0].code;
        if (arr[0].region) {
                locale += "_" + arr[0].region;
        }
        return locale;
}

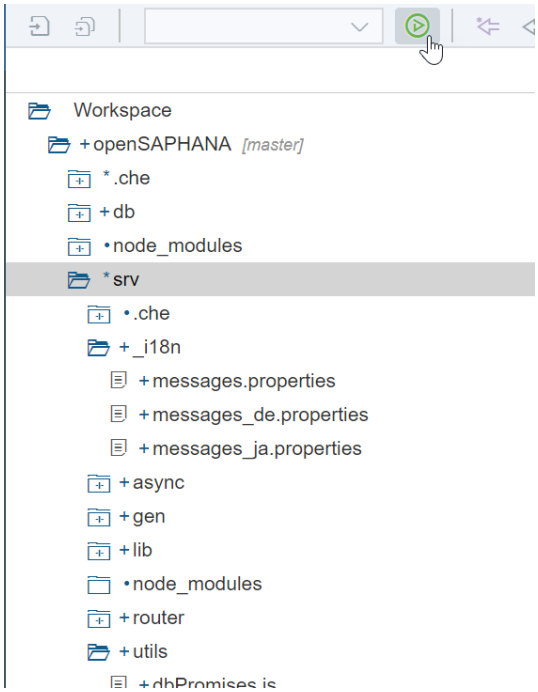
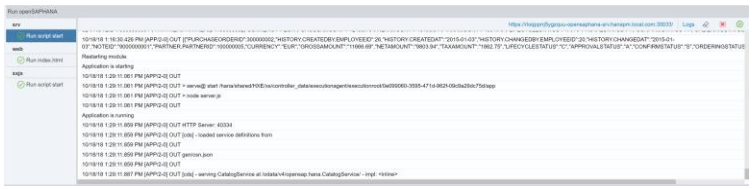
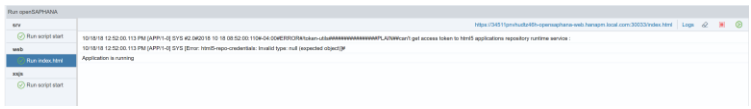
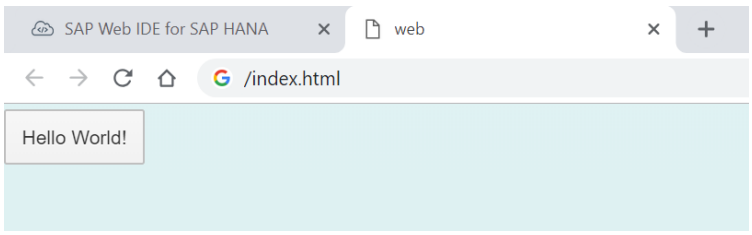
module.exports = function() {

        app.get("/", function(req, res) {
                var bundle = new TextBundle(global.__base +
"_i18n/messages", getLocale(req));
                res.writeHead(200, {
                        "Content-Type": "text/plain;
charset=utf-8"
                });
                var greeting = bundle.getText("greeting",
[os.hostname(), os.type()]);
                res.end(greeting, "utf-8");
        });
        return app;
};
```

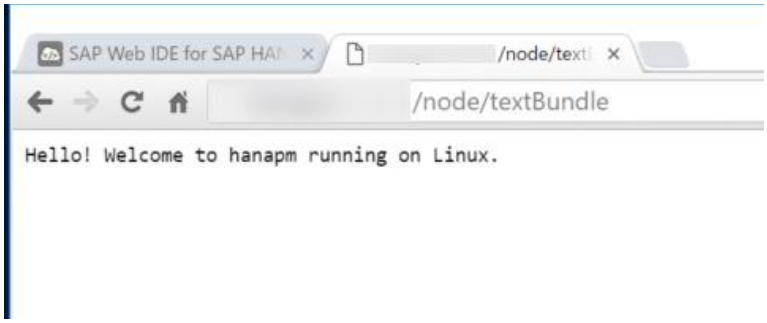
5. We need a few text files to test with.
   You can download these files from here: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/i18n.zip

   Create a folder in **srv** called **_i18n**. Right mouse click on the **srv/_i18n** folder and choose **Import-> From File System**. Choose the file **i18n.zip** from your Download local folder. Keep all other settings at their default values. Choose **OK**.

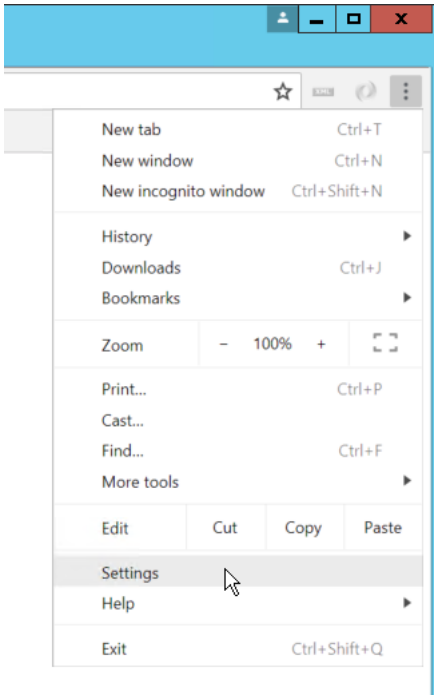


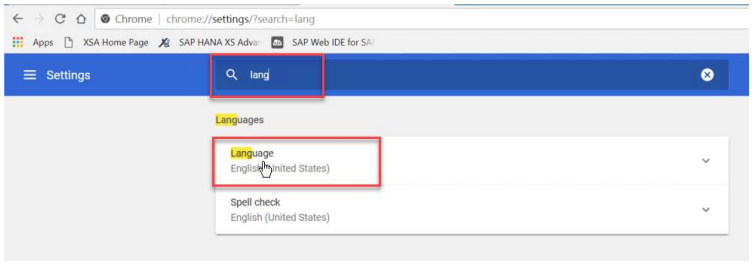
6. We can now run the **srv** module.	
7. You should see that the build and deploy was successful.	
8. So now run the **web** module.	
9. In the running tab, you should see the index.html from earlier.	

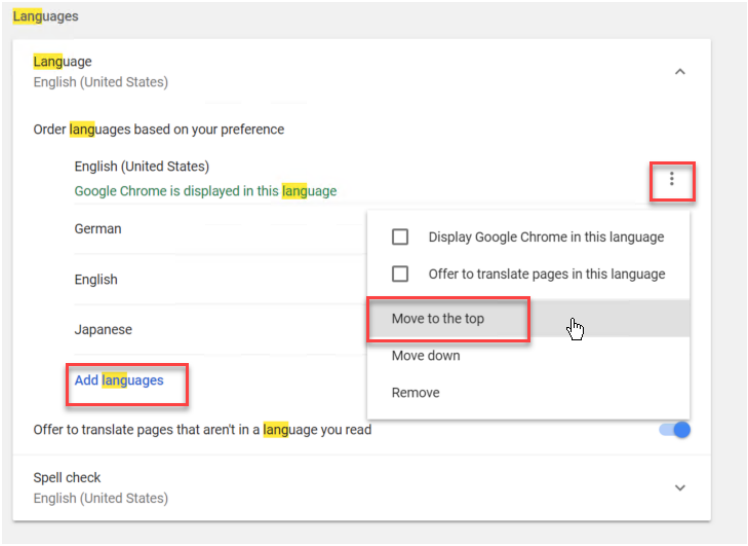
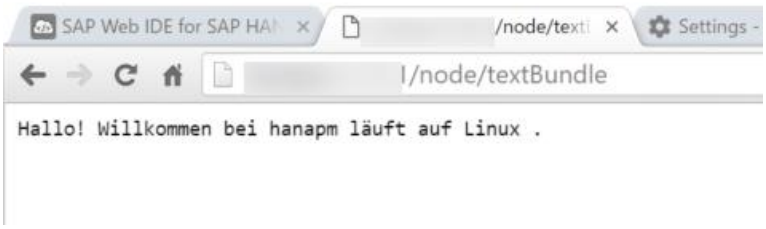
10. Now change the path in the browser to **/node/textBundle**. You should see the English message output from your text file in the i18n folder.



11. In order to test the translated strings, go into the browser **Settings**. Search for Lang and then choose the **Language** and input settings button.





12. Add **German** and **Japanese** to the language list. Drag and drop to raise **German** to the top of the list	
13. Refresh the web browser and you should now see the German text.	
14. Repeat the process raising **Japanese** [ja] to the top of the list and refresh the web page	
15. Return the browser settings to **English** at the top.	

16. Text bundles can now also be used by CDS to create language indepenent texts in the metadata/annotations of the generated OData services as well.

    Remember back to the first of Exercise 4 when we first created the PO and POItems services.

    We used a title like shown in the screenshots. We didn't have a text bundle at the time so it actually output this entire block in the service. Now lets create a text bundle for this string as well.

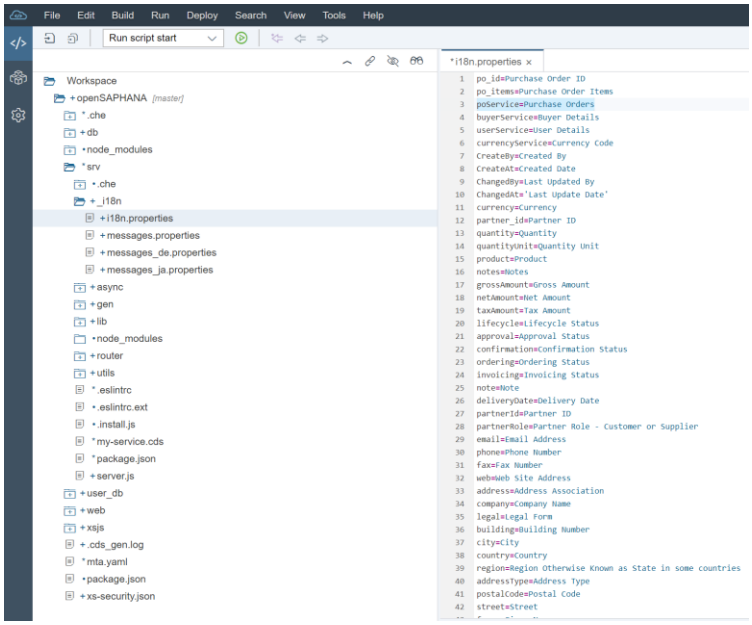
my-service.cds ×

```
1  using opensap.PurchaseOrder.Headers as Headers from '../db/data-model';
2  using opensap.PurchaseOrder.Items as Items from '../db/data-model';
3
4  service CatalogService {
5
6  entity POs @(
7          title: '{i18n>poService}',
8          Capabilities: {
9              InsertRestrictions: {Insertable: true},
10             UpdateRestrictions: {Updatable: true},
11             DeleteRestrictions: {Deletable: true}
12         }
13     ) as projection on Headers;
14
15 entity POItems @(
16         title: '{i18n>poService}',
17         Capabilities: {
18             InsertRestrictions: {Insertable: true},
19             UpdateRestrictions: {Updatable: true},
20             DeleteRestrictions: {Deletable: true}
21         }
22     ) as projection on Items;
23
24 }
```

```
</EntityType>
<Annotations Target="CatalogService.POs">
   <Annotation Term="Common.Label" String="{i18n>poService}"/>
</Annotations>
<Annotations Target="CatalogService.EntityContainer/POs">
   <Annotation Term="Capabilities.InsertRestrictions">
      <Record Type="Capabilities.InsertRestrictionsType">
         <PropertyValue Property="Insertable" Bool="true"/>
      </Record>
   </Annotation>
   <Annotation Term="Capabilities.UpdateRestrictions">
      <Record Type="Capabilities.UpdateRestrictionsType">
         <PropertyValue Property="Updatable" Bool="true"/>
```

17. Create a new file named **i18n.properties** in the **srv/_i18n** folder.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/i18n.properties



18. Save your files and re-run the **srv** module. Then retest the Odata V4 service metadata at **/odata/v4/opensap.hana.CatalogService/$metadata**

    If you scroll down into the metadata you should see that the CatalogService.POs annottion now hast he descriptive string which is langauge specific. The same way we created translated bundles for this exercise could now be applied to your Odata V4 metadata as well. This is especially important later when we start to build UI5/Fiori applications that use these labels in the User Interface.

Before:

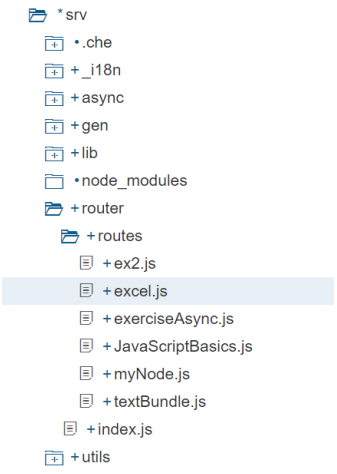
```
</EntityType>
<Annotations Target="CatalogService.POs">
   <Annotation Term="Common.Label" String="{i18n>poService}"/>
</Annotations>
<Annotations Target="CatalogService.EntityContainer/POs">
   <Annotation Term="Capabilities.InsertRestrictions">
      <Record Type="Capabilities.InsertRestrictionsType">
         <PropertyValue Property="Insertable" Bool="true"/>
      </Record>
   </Annotation>
   <Annotation Term="Capabilities.UpdateRestrictions">
      <Record Type="Capabilities.UpdateRestrictionsType">
         <PropertyValue Property="Updatable" Bool="true"/>
```

After:

```
</EntityType>
<Annotations Target="CatalogService.POs">
   <Annotation Term="Common.Label" String="Purchase Orders"/>
</Annotations>
<Annotations Target="CatalogService.EntityContainer/POs">
   <Annotation Term="Capabilities.InsertRestrictions">
      <Record Type="Capabilities.InsertRestrictionsType">
         <PropertyValue Property="Insertable" Bool="true"/>
      </Record>
   </Annotation>
   <Annotation Term="Capabilities.UpdateRestrictions">
      <Record Type="Capabilities.UpdateRestrictionsType">
         <PropertyValue Property="Updatable" Bool="true"/>
      </Record>
```

Exercise 4.7: Open Source Modules

We've already seen how key Node.js functionality is provided by non-SAP open source modules. You've used express, async, and others already in this exercise. Now we will see other common programming tasks can be accomplished without the need of SAP provided APIs. We will convert database query results to Microsoft Excel, utilize ZIP compression, and XML parsing – all using existing open source Node.js Modules.

Explanation	Screenshot
1. Return to the **srv** module.	
2. Add a route for a new module named **excel** that corresponds to **/node/excel** in the **router/index.js**	/*eslint-env node, es6 */ "use strict"; module.exports = (app, server) => { app.use("/node", require("./routes/myNode")()); app.use("/node/ex2", require("./routes/ex2")()); app.use("/node/excAsync", require("./routes/exerciseAsync")(server)); app.use("/node/JavaScriptBasics", require("./routes/JavaScriptBasics")()); app.use("/node/textBundle", require("./routes/textBundle")()); **app.use("/node/excel", require("./routes/excel")());** };
3. Create a new file in your **router/routes** folder called **excel.js**.	

4. Add the following code to your **excel.js** file. We are using the node-xlsx module. With one call to a function in this module we can pass in query results as JSON and receive back the binary string that contains the true Excel content format (not just tab or comma delimited string).

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_excel.js

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, dot-notation: 0, new-cap: 0*/
/*eslint-env node, es6 */
"use strict";
var express = require("express");
var excel = require("node-xlsx");
var fs = require("fs");
var path = require("path");

module.exports = function() {
	var app = express.Router();

	//Hello Router
	app.get("/", (req, res) => {
		var output =
			`<H1>Excel Examples</H1></br>
			<a href="${req.baseUrl}/download">/download</a> - Download data in Excel XLSX format</br>`;
		res.type("text/html").status(200).send(output);
	});

	//Simple Database Select - In-line Callbacks
	app.get("/download", async(req, res) => {
		try {
			const dbClass = require(global.__base + "utils/dbPromises");
			let client = new dbClass(req.db);
			const statement = await client.preparePromisified(
				`SELECT TOP 10
"HEADER.PURCHASEORDERID" as "ID",

"PRODUCT.PRODUCTID" as "PRODUCT",

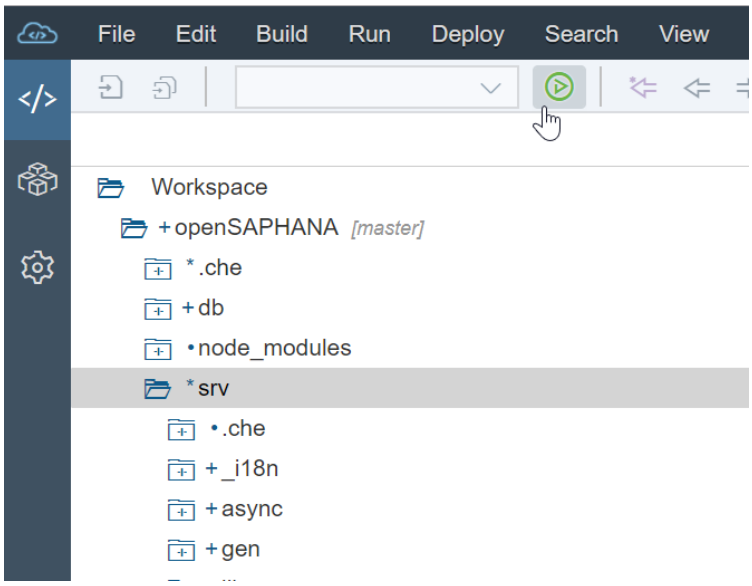
	GROSSAMOUNT as "AMOUNT"

	FROM "PO.Item"`
			);
			const results = await client.statementExecPromisified(statement, []);
			let out = [];
			for (let item of results) {
				out.push([item.ID, item.PRODUCT, item.AMOUNT]);
			}
			var result = excel.build([{
				name: "Purchase Orders",
				data: out
			}]);
			res.header("Content-Disposition", "attachment; filename=Excel.xlsx");
			return res.type("application/vnd.ms-excel").status(200).send(result);
```

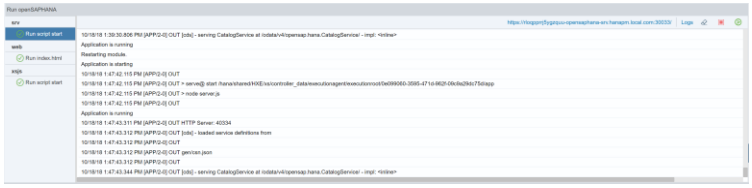
```
                } catch (e) {
                    return
res.type("text/plain").status(500).send(`ERROR:
${e.toString()}`);
                }
            });

            return app;
};
```

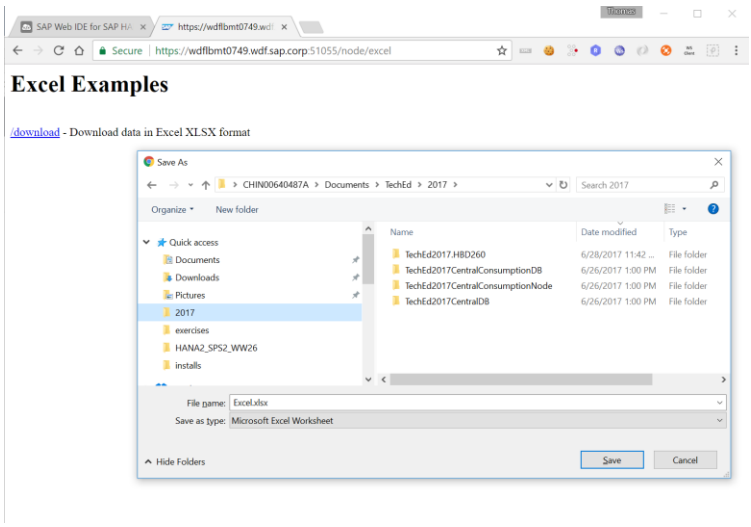
5. We can now run the **srv** module.

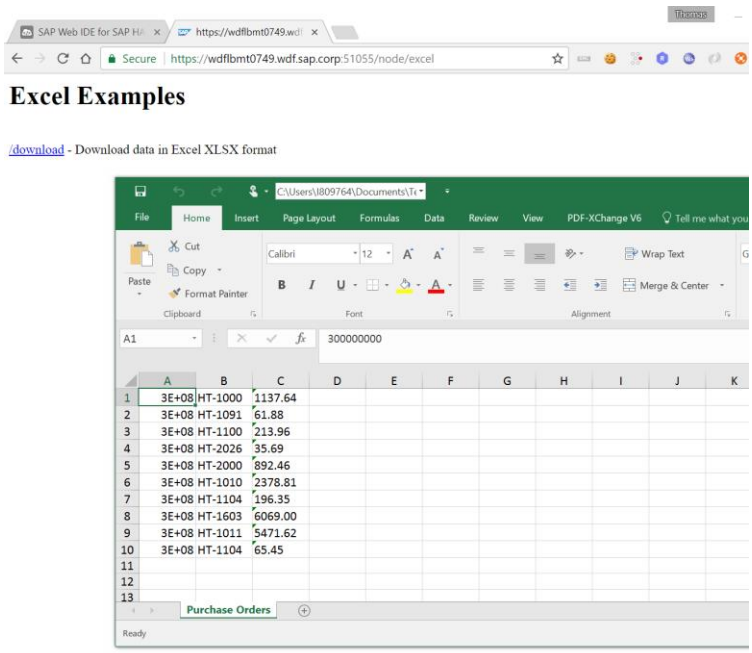


6. You should see that the build and deploy was successful.



7. In your other browser tab where we've been testing, change the path in the browser to **/node/excel/download**. You should receive the file save dialog of the browser for a file named Excel.xlsx.

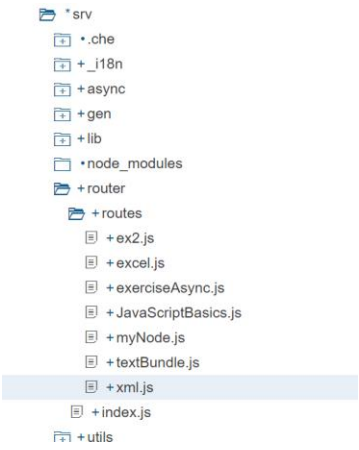




8. Return to the **srv** module. Add a route for a new module named **xml** that corresponds to **/node/xml** in the **router/index.js**

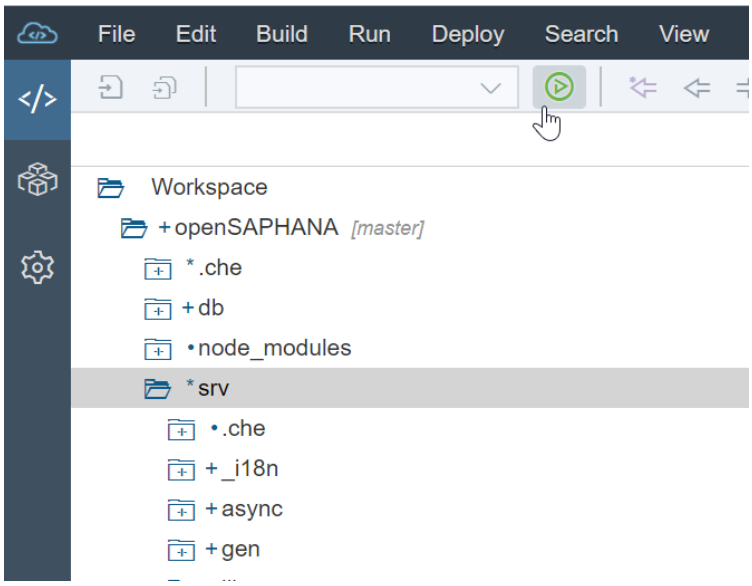
```
/*eslint-env node, es6 */
"use strict";

module.exports = (app, server) => {
	app.use("/node", require("./routes/myNode")());
	app.use("/node/ex2", require("./routes/ex2")());
	app.use("/node/excAsync",
require("./routes/exerciseAsync")(server));
	app.use("/node/JavaScriptBasics",
require("./routes/JavaScriptBasics")());
	app.use("/node/textBundle",
require("./routes/textBundle")());
	app.use("/node/excel", require("./routes/excel")());
```

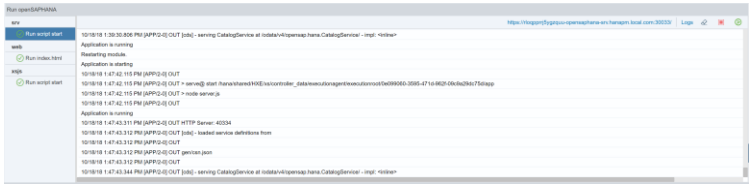
	app.use("/node/xml", require("./routes/xml")()); };
9. Create a new file in your **router/routes** folder called **xml.js**.	
10. Add the following code to your **xml.js** file. We are using the **xmldoc** module. This module provides XML parsing and rendering functions. We will provide a simple, hardcoded XML string and use this module to parse it and output results for each child of the root node we find. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_xml.js	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, quotes: 0, new-cap: 0*/ /*eslint-env node, es6 */ "use strict"; var express = require("express"); var XmlDocument = require("xmldoc").XmlDocument; module.exports = function() { var app = express.Router(); //Hello Router app.get("/", (req, res) => { var output = `<H1>XML Examples</H1></br> <a href="${req.baseUrl}/example1">/example1</a> - Simple XML parsing</br>`; res.type("text/html").status(200).send(output); }); //Simple Database Select - In-line Callbacks app.get("/example1", (req, res) => { const xml = `<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <!-- this is a note --> <note noteName="NoteName"> <to>To</to> <from>From</from> <heading>Note heading</heading> <body>Note body</body> </note> </xml>`; let body = "";

```
                let note = new XmlDocument(xml);
                note.eachChild((item) => {
                        body += item.val + '</br>';
                });
                return
res.type("text/html").status(200).send(body);
        });
        return app;
};
```

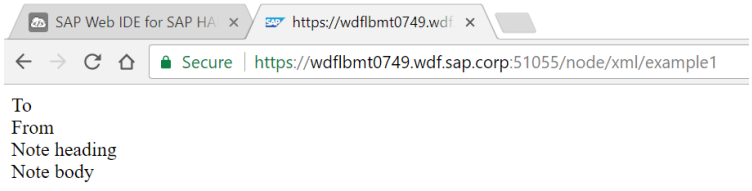
11. We can now run the **srv** module.



12. You should see that the build and deploy was successful.



13. In your other browser tab where we've been testing, change the path in the browser to **/node/xml/example1**. You should see the results of parsing the sample XML content.



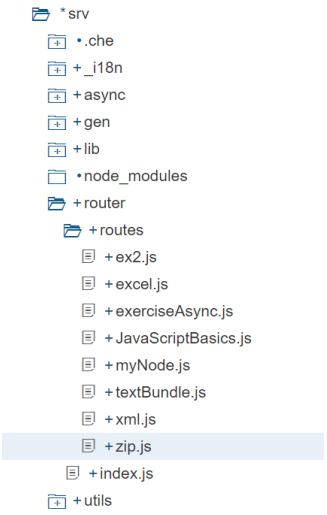
14. Return to the **srv** module. Add a route for a new module named **zip** that corresponds to **/node/zip** in the **router/index.js**

```
/*eslint-env node, es6 */
"use strict";

module.exports = (app, server) => {
        app.use("/node", require("./routes/myNode")());
        app.use("/node/ex2", require("./routes/ex2")());
        app.use("/node/excAsync",
require("./routes/exerciseAsync")(server));
        app.use("/node/JavaScriptBasics",
require("./routes/JavaScriptBasics")());
        app.use("/node/textBundle",
require("./routes/textBundle")());
        app.use("/node/excel", require("./routes/excel")());
        app.use("/node/xml", require("./routes/xml")());
        app.use("/node/zip", require("./routes/zip")());

};
```

15. Create a new file in your **router/routes** folder called **zip.js**.



16. Add the following code to your **zip.js** file. We are using the **node-zip** module. With one call to a function in this module can create zip content; adding files, folders, or any content to the output zip archive.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_zip.js

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, dot-
notation: 0, new-cap: 0*/
/*eslint-env node, es6 */
"use strict";
var express = require("express");

module.exports = function() {
        var app = express.Router();

        //Hello Router
        app.get("/", (req, res) => {
                var output =
                        `<H1>ZIP Examples</H1></br>
                        <a
href="${req.baseUrl}/example1">/example1</a> - Download
data in ZIP format - folder and files</br>
```

```
                        <a
href="${req.baseUrl}/zipPO">/zipPO</a> - Download Purchase
Orders in ZIP format</br>`;
                res.type("text/html").status(200).send(output);
        });

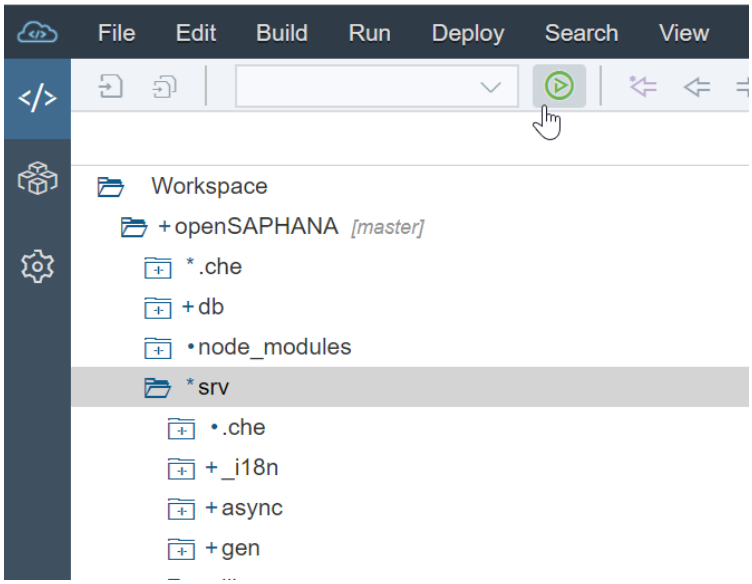
        //Simple Database Select - In-line Callbacks
        app.get("/example1", (req, res) => {
                let zip = new require("node-zip")();
                zip.file("folder1/demo1.txt", "This is the new
ZIP Processing in Node.js");
                zip.file("demo2.txt", "This is also the new ZIP
Processing in Node.js");
                let data = zip.generate({
                        base64: false,
                        compression: "DEFLATE"
                });
                res.header("Content-Disposition", "attachment;
filename=ZipExample.zip");
                return
res.type("application/zip").status(200).send(new Buffer(data,
"binary"));
        });

        //Simple Database Select - In-line Callbacks
        app.get("/zipPO", async(req, res) => {
                try {
                        const dbClass = require(global.__base
+ "utils/dbPromises");
                        let client = new dbClass(req.db);
                        const statement = await
client.preparePromisified(
                                `SELECT TOP 25000
"PurchaseOrderId", "PartnerId", "CompanyName",
"CreatedByLoginName", "CreatedAt", "GrossAmount"
                                FROM "PO.HeaderView"
ORDER BY "PurchaseOrderId" `
                        );
                        const results = await
client.statementExecPromisified(statement, []);
                        let out = "";
                        for (let item of results) {
                                out += item.PurchaseOrderId
+ "\t" + item.PartnerId + "\t" + item.CompanyName + "\t" +
item.CreatedByLoginName + "\t" + item.CreatedAt +
                                        "\t" +
item.GrossAmount + "\n";
                        }
                        let zip = new require("node-zip")();
                        zip.file("po.txt", out);
                        let data = zip.generate({
                                base64: false,
                                compression: "DEFLATE"
                        });
```

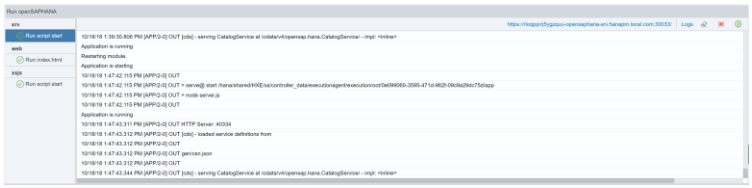
```
                                    res.header("Content-Disposition",
"attachment; filename=poWorklist.zip");
                                    return
res.type("application/zip").status(200).send(new Buffer(data,
"binary"));
                        } catch (e) {
                                    return
res.type("text/plain").status(500).send(`ERROR:
${e.toString()}`);
                        }
            });

            return app;
};
```

17. We can now run the **srv** module.

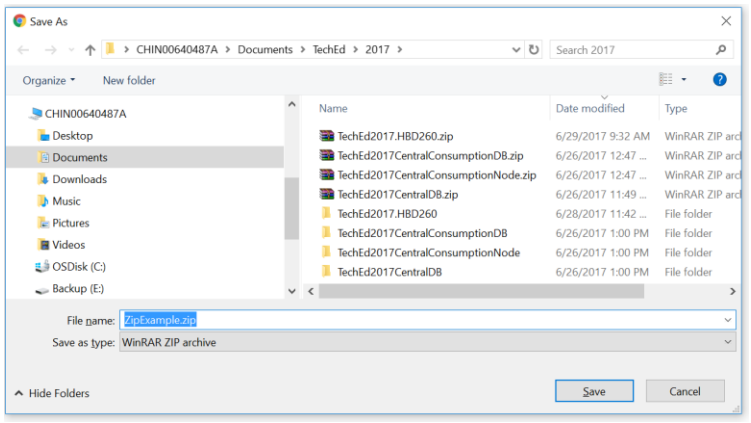


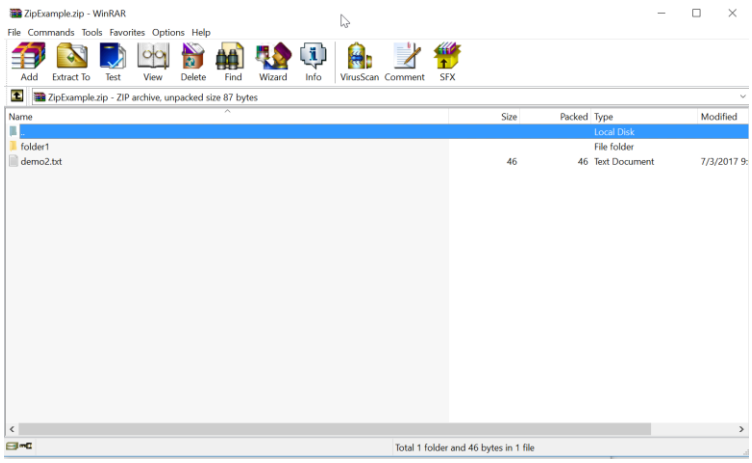
18. You should see that the build and deploy was successful.



19. In your other browser tab where we've been testing, change the path in the browser to **/node/zip/example1**. You should receive the file save dialog of the browser for a file named ZipExample.zip.

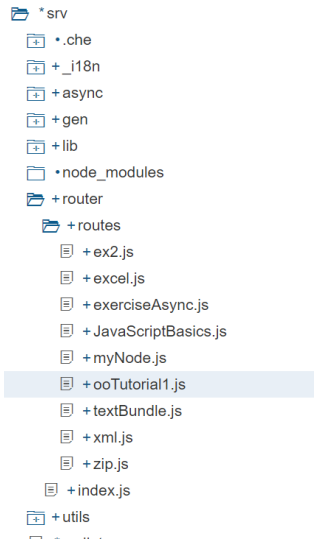
    This zip file has a folder named, folder1, with a file named demo1.txt. It also has a file named demo2.txt in the root.

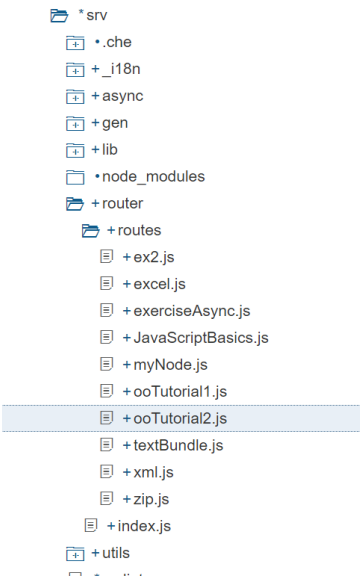




Exercise 4.8: JavaScript Object Oriented

In Exercise 4.5 we saw some of the basics of the new JavaScript class capabilities of ES6. In this exercise we want to expand upon this topic and learn more about the objected oriented programming concepts that are possible in Node.js.

Explanation	Screenshot
1. Return to the **srv** module.	
2. In our **routes** folder, create a new file named **ooTutorial1.js**.	
3. In **ooTutorial1.js**, let's create a very small class. It should have a constructor that accepts a unique id. It should also have one method that takes an input value. If that value is equal to 20, then throw an error. Otherwise just return the input value. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0, dot-notation:0, no-use-before-define:0, no-inner-declarations:0 */ /*eslint-env node, es6 */ module.exports = class { constructor(id) { this.id = id; } myFirstMethod(import1) { if (import1 === 20) { let error = {}; error.message = "Something Bad Happend"; error.value = import1; throw error; } return import1; }

hana7/blob/snippets_2.3.2/ex5/oo1.js	};
4. In our routes folder, create a new file named **ooTutorial2.js**.	
5. Open the new file **ooTutorial2.js**. Like any good OO environment, JavaScript classes support both instance and static methods/attributes. Let's create another class with two static methods – **getFlightDetails** and calculateFlightPrice. **getFlightDetails** will just be a simple wrapper for a database selection. **calculateFlightPrice** will also be a database select, but we will also enhance the results with business logic to apply different price surcharges for different plane types. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address	/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0, dot-notation:0, no-use-before-define:0, no-inner-declarations:0 */ /*eslint-env node, es6 */ module.exports = class { static async getFlightDetails(db, carrierId, connectionId, flightDate) { return new Promise(async(resolve, reject) => { try { const dbPromises = require(global.__base + "utils/dbPromises"); let dbConn = new dbPromises(db); const statement = await dbConn.preparePromisified("select * from SFLIGHT WHERE CARRID = ? AND CONNID = ? and FLDATE = ? "); resolve(await dbConn.statementExecPromisified(statement, [carrierId, connectionId, flightDate])); } catch (err) { let error = {}; error.message = "Invalid Flight"; error.carrierId = carrierId; error.connectionId = connectionId; error.flightDate = flightDate; error.details = err;

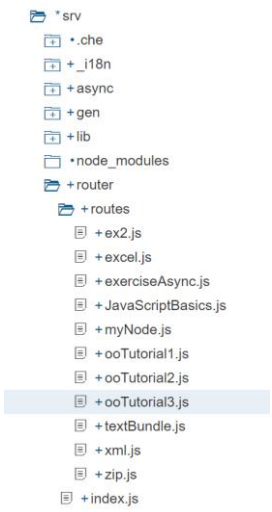
https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/oo2.js

```
                                        reject(error);
                                }
                        });
                }

                static async calculateFlightPrice(db, carrierId,
connectionId, flightDate) {
                        return new Promise(async(resolve, reject) => {
                                try {
                                        const dbPromises =
require(global.__base + "utils/dbPromises");
                                        let dbConn = new
dbPromises(db);
                                        const statement = await
dbConn.preparePromisified(
                                                "select PRICE,
CURRENCY, PLANETYPE from SFLIGHT WHERE CARRID =
? AND CONNID = ? and FLDATE = ? ");
                                        const results = await
dbConn.statementExecPromisified(statement, [carrierId,
connectionId, flightDate]);
                                        let price;
                                        switch
(results[0].PLANETYPE) {
                                                case "747-400":
                                                        price =
results[0].PRICE * 1 + 40;
                                                        break;
                                                case "A310-300":
                                                        price =
results[0].PRICE * 1 + 25;
                                                        break;
                                                default:
                                                        price =
results[0].PRICE * 1 + 10;
                                                        break;
                                        }
                                        resolve(price);
                                } catch (err) {
                                        let error = {};
                                        error.message = "Invalid
Flight";
                                        error.carrierId = carrierId;
                                        error.connectionId =
connectionId;
                                        error.flightDate = flightDate;
                                        error.details = err;

                                        reject(error);
                                }
                        });
                }
};
```

6. In our **routes** folder, create a new file named **ooTutorial3.js**.



7. Now copy the content from ooTutorial2.js into ooTutorial3.js. We want to adjust this class to now be instances based. Add a constructor method that will accept the DB object. We can store the DB object in the instance of the class and avoid having to pass it into each method.

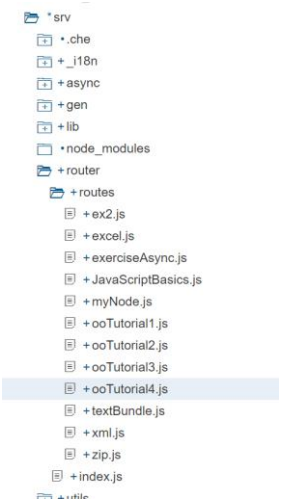
   Also, we can store the results of the database query with the class instance as well. We don't have to pass the keys into both methods.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/oo3.js

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0, dot-notation:0, no-use-before-define:0, no-inner-declarations:0 */
/*eslint-env node, es6 */
module.exports = class {
	constructor(db) {
		this.db = db;
	}
	async getFlightDetails(carrierId, connectionId, flightDate) {
		const dbPromises = require(global.__base + "utils/dbPromises");
		let dbConn = new dbPromises(this.db);
		return new Promise(async(resolve, reject) => {
			try {
				const statement = await dbConn.preparePromisified("select * from SFLIGHT WHERE CARRID = ? AND CONNID = ? and FLDATE = ? ");
				this.flight = await dbConn.statementExecPromisified(statement, [carrierId, connectionId, flightDate]);
				resolve(this.flight);
			} catch (err) {
				let error = {};
				error.message = "Invalid Flight";
				error.carrierId = carrierId;
				error.connectionId = connectionId;
				error.flightDate = flightDate;
				error.details = err;
				reject(error);
			}
		});
	}
```

```
        calculateFlightPrice() {
                return new Promise(async(resolve, reject) => {
                        try {
                                let price;
                                switch
(this.flight[0].PLANETYPE) {
                                        case "747-400":
                                                price =
this.flight[0].PRICE * 1 + 40;
                                                break;
                                        case "A310-300":
                                                price =
this.flight[0].PRICE * 1 + 25;
                                                break;
                                        default:
                                                price =
this.flight[0].PRICE * 1 + 10;
                                                break;
                                }
                                resolve(price);
                        } catch (err) {
                                let error = {};
                                error.message = "Flight not
loaded yet";
                                error.details = err;
                                reject(error);
                        }
                });
        }
};
```

8. In our routes folder, create a new file named **ooTutorial4.js**.



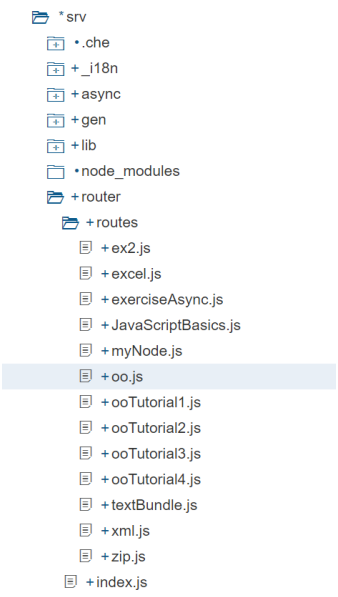
9. For our final class, **ooTutorial4.js,** let's use one of the most powerful aspects of OO – inheritance.

   Create a class that inherits

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-
cap: 0, dot-notation:0, no-use-before-define:0, no-inner-
declarations:0 */
/*eslint-env node, es6 */
var ooTutorial3 = require("./ooTutorial3");
module.exports = class extends ooTutorial3 {
```

from the previous steps. Overload the **calculateFlightPrice** method with a new implementation that adds a 25% surcharge to all prices. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/oo4.js	constructor(db) { super(db); } async calculateFlightPrice() { return new Promise(async(resolve, reject) => { try { let price = await super.calculateFlightPrice(); resolve(price * 1.25); } catch (err) { reject(err); } }); } };
10. Add a route for a new module named **oo** that corresponds to **/node/oo** in the **router/index.js** so that we can write tests for all these new classes.	/*eslint-env node, es6 */ "use strict"; module.exports = (app, server) => { app.use("/node", require("./routes/node")()); app.use("/node/ex2", require("./routes/ex2")()); app.use("/node/excAsync", require("./routes/exerciseAsync")(server)); app.use("/node/JavaScriptBasics", require("./routes/JavaScriptBasics")()); app.use("/node/textBundle", require("./routes/textBundle")()); app.use("/node/excel", require("./routes/excel")()); app.use("/node/xml", require("./routes/xml")()); app.use("/node/zip", require("./routes/zip")()); **app.use("/node/oo", require("./routes/oo")());** };

11. Create a new file in your **router/routes** folder called **oo.js**.



12. Add the following framework code to your **oo.js** file.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/oo.js

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0, dot-notation:0, no-use-before-define:0, no-inner-declarations:0 */
/*eslint-env node, es6 */
"use strict";
var express = require("express");
var ooTutorial1 = require("./ooTutorial1");
var ooTutorial2 = require("./ooTutorial2");
var ooTutorial3 = require("./ooTutorial3");
var ooTutorial4 = require("./ooTutorial4");

module.exports = function() {
	var app = express.Router();

	//Hello Router
	app.get("/", (req, res) => {
		let output =
			`<H1>JavaScript Object Oriented</H1></br>
			<a href="${req.baseUrl}/classes1">/classses1</a> - Classes</br>
			<a href="${req.baseUrl}/classes1Error">/classses1Error</a> - Classes, catch errors</br>
			<a href="${req.baseUrl}/classes2a">/classes2a</a> - Classes with Static Methods #1</br>
			<a href="${req.baseUrl}/classes2b">/classes2b</a> - Classes with Static Methods #2</br>
			<a href="${req.baseUrl}/classes3a">/classes3a</a> - Classes with Instance Methods #1</br>
```

```
                        <a
href="${req.baseUrl}/classes3b">/classes3b</a> - Classes
with Instance Methods #2</br>
                        <a
href="${req.baseUrl}/classes4a">/classes4a</a> - Classes
with Inherited Methods #1</br>
                        <a
href="${req.baseUrl}/classes4b">/classes4b</a> - Classes
with Inherited Methods #2</br>`;
                res.type("text/html").status(200).send(output);
        });

        app.get("/classes1", (req, res) => {
                let class1 = new ooTutorial1("first example");
                return
res.type("text/html").status(200).send(`Call First Method:
${class1.myFirstMethod(5)}`);
        });

        app.get("/classes1Error", (req, res) => {
                let class1 = new ooTutorial1("first example");
                try {
                        return class1.myFirstMethod(20);
                } catch (e) {
                        return
res.type("application/json").status(200).send(`Call and catch
errors: ${JSON.stringify(e)}`);
                }
        });

        app.get("/classes2a", async(req, res) => {
                try {
                        const results = await
ooTutorial2.getFlightDetails(req.db, "AA", "0017", "20100421");
                        return
res.type("text/html").status(200).send(
                                `Call Static Method:
${JSON.stringify(results)}`);
                } catch (e) {
                        return
res.type("application/json").status(200).send(`Call and catch
errors: ${JSON.stringify(e)}`);
                }

        });

        app.get("/classes2b", async(req, res) => {
                try {
                        const results = await
ooTutorial2.calculateFlightPrice(req.db, "AA", "0017",
"20100421");
                        return
res.type("text/html").status(200).send(
```

```
                                        `Call Static Method - Calc
Price: ${results.toString()}`);
                } catch (e) {
                        return
res.type("application/json").status(200).send(`Call and catch
errors: ${JSON.stringify(e)}`);
                }
        });

        app.get("/classes3a", async(req, res) => {
                try {
                        let class3 = new ooTutorial3(req.db);
                        const results = await
class3.getFlightDetails("AA", "0017", "20100421");
                        return
res.type("text/html").status(200).send(
                                        `Call Instance Method:
${JSON.stringify(results)}`);
                } catch (e) {
                        return
res.type("application/json").status(200).send(`Call and catch
errors: ${JSON.stringify(e)}`);
                }
        });

        app.get("/classes3b", async(req, res) => {
                try {
                        let class3 = new ooTutorial3(req.db);
                        await class3.getFlightDetails("AA",
"0017", "20100421");
                        const results = await
class3.calculateFlightPrice();
                        return
res.type("text/html").status(200).send(
                                        `Call Instance Method - Calc
Price: ${results.toString()}`);
                } catch (e) {
                        return
res.type("application/json").status(200).send(`Call and catch
errors: ${JSON.stringify(e)}`);
                }
        });

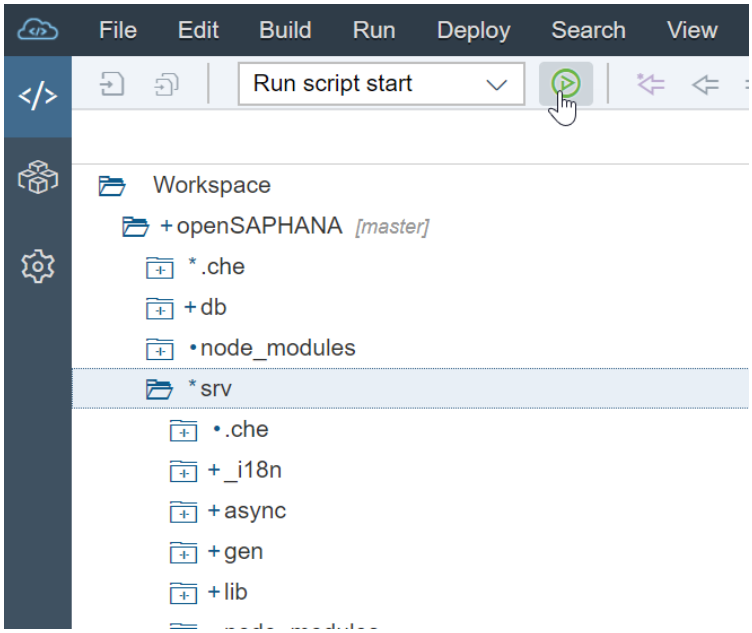
        app.get("/classes4a", async(req, res) => {
                try {
                        let class4 = new ooTutorial4(req.db);
                        const results = await
class4.getFlightDetails("AA", "0017", "20100421");
                        return
res.type("text/html").status(200).send(
                                        `Call Inherited Method:
${JSON.stringify(results)}`);
                } catch (e) {
```

	return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } }); app.get("/classes4b", async(req, res) => { try { let class4 = new ooTutorial4(req.db); await class4.getFlightDetails("AA", "0017", "20100421"); const results = await class4.calculateFlightPrice(); return res.type("text/html").status(200).send(`Call Overridden Method - Calc Price: ${results.toString()}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } }); return app; };
13. Let's break down this big block of code and study it in more detail. Add two route handlers to oo.js file – **classes1** and **classes1Error** to test **ooTuorial1.js**. The first handler will pass in the value 5 which should be valid. The second handler will pass in the value 20 which should trigger the error handling.	app.get("/classes1", (req, res) => { let class1 = new ooTutorial1("first example"); return res.type("text/html").status(200).send(`Call First Method: ${class1.myFirstMethod(5)}`); }); app.get("/classes1Error", (req, res) => { let class1 = new ooTutorial1("first example"); try { return class1.myFirstMethod(20); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } });
14. Add another two route handlers – **classes2a** and **classes2b** to test our two static class methods from **ooTutorial2.js**.	app.get("/classes2a", async(req, res) => { try { const results = await ooTutorial2.getFlightDetails(req.db, "AA", "0017", "20100421"); return res.type("text/html").status(200).send(`Call Static Method: ${JSON.stringify(results)}`); } catch (e) {

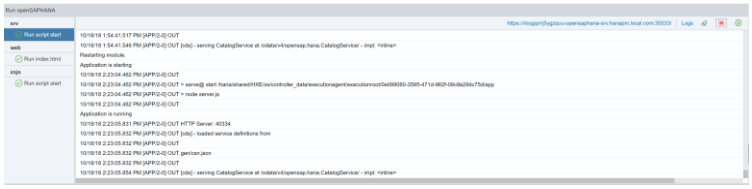
	return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } }); app.get("/classes2b", async(req, res) => { try { const results = await ooTutorial2.calculateFlightPrice(req.db, "AA", "0017", "20100421"); return res.type("text/html").status(200).send(`Call Static Method - Calc Price: ${results.toString()}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } });
15. Add another two route handlers – **classes3a** and **classes3b** to test our two instance class methods from **ooTutorial3.js**. Notice how we now only need to pass the keys in once. We can call the **calculateFlightPrice** without specifying any input parameters because the current record is stored in the instance of the class.	app.get("/classes3a", async(req, res) => { try { let class3 = new ooTutorial3(req.db); const results = await class3.getFlightDetails("AA", "0017", "20100421"); return res.type("text/html").status(200).send(`Call Instance Method: ${JSON.stringify(results)}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } }); app.get("/classes3b", async(req, res) => { try { let class3 = new ooTutorial3(req.db); await class3.getFlightDetails("AA", "0017", "20100421"); const results = await class3.calculateFlightPrice(); return res.type("text/html").status(200).send(`Call Instance Method - Calc Price: ${results.toString()}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`);

	} });
16. Add the final two route handlers – **classes4a** and **classes4b** to test our inherited class from **ooTutorial4.js**. We don't see any difference from the consumption of the class but we will get the enhanced **calculateFlightPrice** method with the automatic surcharge.	app.get("/classes4a", async(req, res) => { try { let class4 = new ooTutorial4(req.db); const results = await class4.getFlightDetails("AA", "0017", "20100421"); return res.type("text/html").status(200).send(`Call Inherited Method: ${JSON.stringify(results)}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } }); app.get("/classes4b", async(req, res) => { try { let class4 = new ooTutorial4(req.db); await class4.getFlightDetails("AA", "0017", "20100421"); const results = await class4.calculateFlightPrice(); return res.type("text/html").status(200).send(`Call Overridden Method - Calc Price: ${results.toString()}`); } catch (e) { return res.type("application/json").status(200).send(`Call and catch errors: ${JSON.stringify(e)}`); } });

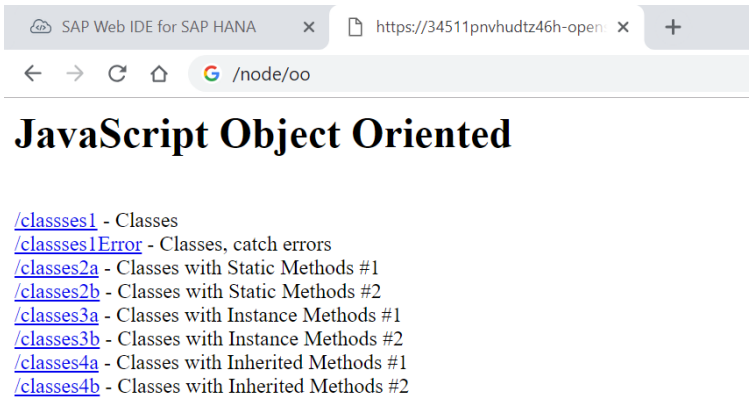
17. We can now run the **srv** module.

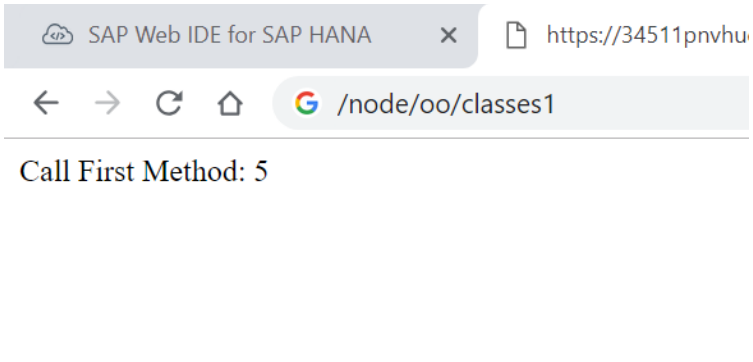


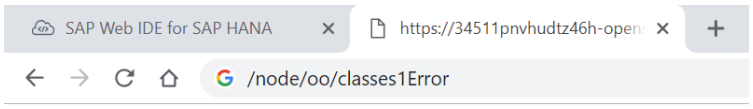
18. You should see that the build and deploy was successful.



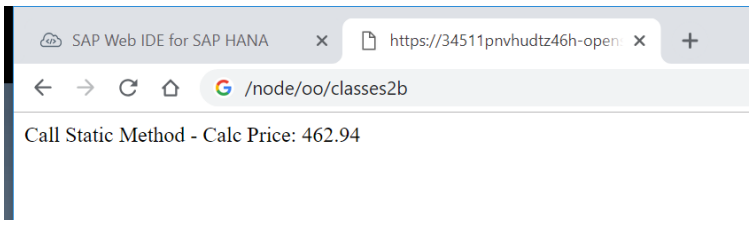
19. In your other browser tab where we've been testing, change the path in the browser to **/node/oo**. You can then test each of the individual links to see the results from the different class tests we just coded.

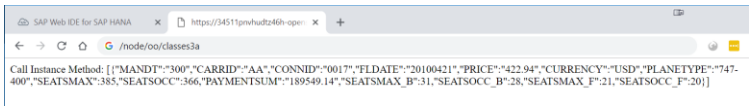


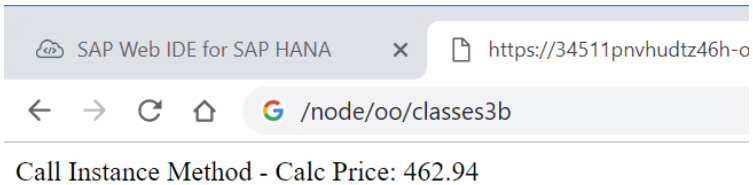

SAP Web IDE for SAP HANA
https://34511pnvhu
/node/oo/classes1
Call First Method: 5


SAP Web IDE for SAP HANA
https://34511pnvhudtz46h-open
/node/oo/classes1Error
Call and catch errors: {"message":"Something Bad Happend","value":20}

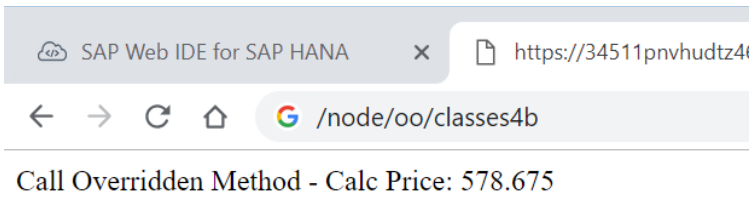

/node/oo/classes2a
Call Static Method: [{"MANDT":"300","CARRID":"AA","CONNID":"0017","FLDATE":"20100421","PRICE":"422.94","CURRENCY":"USD","PLANETYPE":"747-400","SEATSMAX":385,"SEATSOCC":366,"PAYMENTSUM":"189549.14","SEATSMAX_B":31,"SEATSOCC_B":28,"SEATSMAX_F":21,"SEATSOCC_F":20}]


SAP Web IDE for SAP HANA
https://34511pnvhudtz46h-open
/node/oo/classes2b
Call Static Method - Calc Price: 462.94


/node/oo/classes3a
Call Instance Method: [{"MANDT":"300","CARRID":"AA","CONNID":"0017","FLDATE":"20100421","PRICE":"422.94","CURRENCY":"USD","PLANETYPE":"747-400","SEATSMAX":385,"SEATSOCC":366,"PAYMENTSUM":"189549.14","SEATSMAX_B":31,"SEATSOCC_B":28,"SEATSMAX_F":21,"SEATSOCC_F":20}]


SAP Web IDE for SAP HANA
https://34511pnvhudtz46h-o
/node/oo/classes3b
Call Instance Method - Calc Price: 462.94

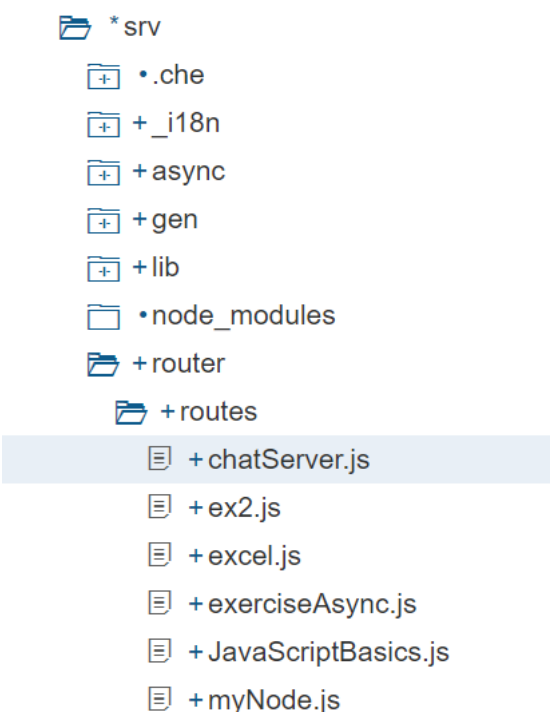

/node/oo/classes4a
Call Inherited Method: [{"MANDT":"300","CARRID":"AA","CONNID":"0017","FLDATE":"20100421","PRICE":"422.94","CURRENCY":"USD","PLANETYPE":"747-400","SEATSMAX":385,"SEATSOCC":366,"PAYMENTSUM":"189549.14","SEATSMAX_B":31,"SEATSOCC_B":28,"SEATSMAX_F":21,"SEATSOCC_F":20}]

	 SAP Web IDE for SAP HANA https://34511pnvhudtz4 /node/oo/classes4b Call Overridden Method - Calc Price: 578.675

Exercise 4.9: Web Sockets

Our next exercise part in this section will demonstrate the ease at which you can tap into the powerful web sockets capabilities of node.js. We will use web sockets to build a simple chat application. Any message sent from the SAPUI5 client side application will be propagated by the server to all listening clients.

Explanation	Screenshot
1. Return to the **srv** module.	
2. Add an express route handler for this chatServer module in the **router/index.js** and pass the server variable in as well.	/*eslint-env node, es6 */ "use strict"; module.exports = (app, server) => { app.use("/node", require("./routes/myNode")()); app.use("/node/ex2", require("./routes/ex2")()); app.use("/node/excAsync", require("./routes/exerciseAsync")(server)); app.use("/node/JavaScriptBasics", require("./routes/JavaScriptBasics")()); app.use("/node/textBundle", require("./routes/textBundle")()); app.use("/node/excel", require("./routes/excel")()); app.use("/node/xml", require("./routes/xml")()); app.use("/node/zip", require("./routes/zip")()); **app.use("/node/chat",** **require("./routes/chatServer")(server));** };

3. Create a new file in your **srv/router/routes** folder called **chatServer.js**.	
4. Add the following code to your **chatServer.js** file. The Node.js module for Web Sockets which we are going to use is **ws**. Require it and create a new instance of the **WebSocketServe**r. This is then a recommended implementation for the remainder of the Web Sockets functionality to both receive and send messages. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/ex5_9	/*eslint no-console: 0, no-unused-vars: 0, new-cap: 0*/ /*eslint-env node, es6 */ "use strict"; var WebSocketServer = require("ws").Server; var express = require("express"); module.exports = function (server) { var app = express.Router(); app.use((req, res) => { var output = `<H1>Node.js Web Socket Examples</H1></br> <a href="/exerciseChat">/exerciseChat</a> - Chat Application for Web Socket Example</br>`; res.type("text/html").status(200).send(output); }); var wss = new WebSocketServer({ // server: server, noServer: true, path: "/node/chatServer" }); server.on("upgrade", function upgrade(request, socket, head) { const url = require("url"); const pathname = url.parse(request.url).pathname; if (pathname === "/node/chatServer") {

```
                wss.handleUpgrade(request, socket,
head, function done(ws) {
                        wss.emit("connection", ws,
request);
                });
        }
    });

    wss.broadcast = (data) => {
        wss.clients.forEach(function each(client) {
            try {
                client.send(data);
            } catch (e) {
                console.log("Broadcast Error:
%s", e.toString());
            }
        });
        console.log("sent: %s", data);

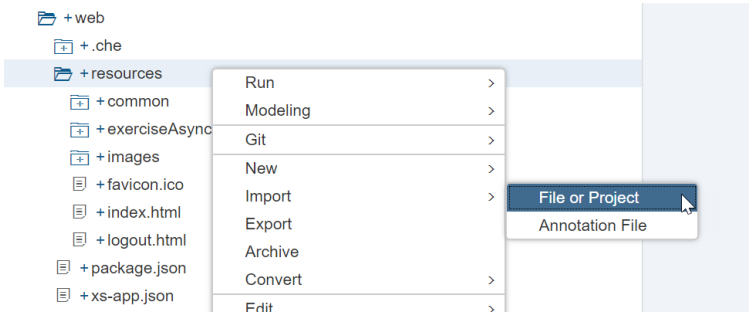
    };

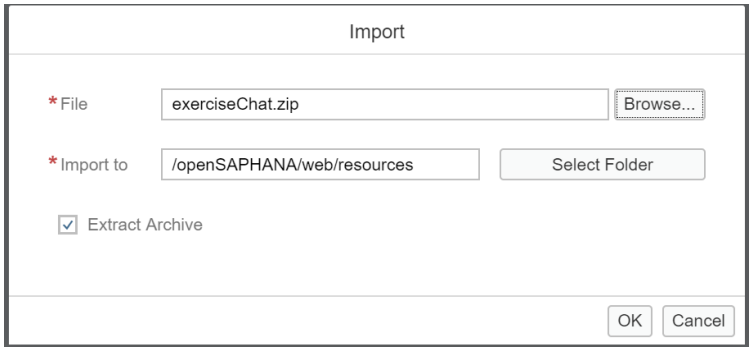
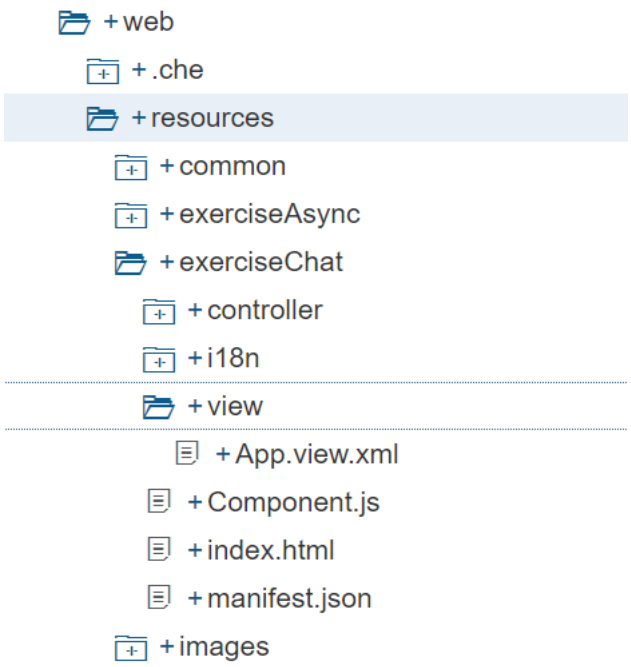
    wss.on("connection", (ws) => {
        console.log("Connected");
        ws.on("message", (message) => {
            console.log("received: %s", message);
            wss.broadcast(message);
        });
        ws.send(JSON.stringify({
            user: "XS",
            text: "Hello from Node.js XS Server"
        }));
    });

    return app;
```

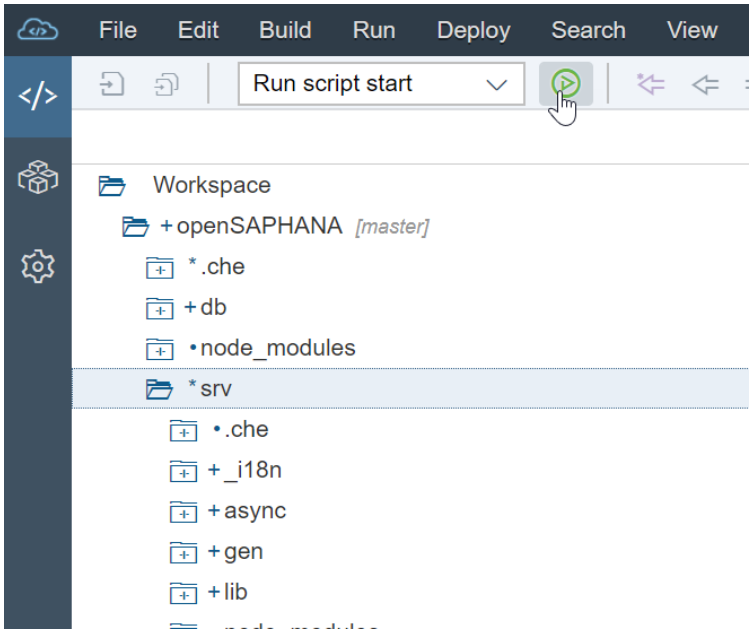
5. We will also need a nice user interface for this exercise. **Download** the files for the UI from here: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/exerciseChat.zip

   Go to the **web/resources** folder and right mouse click. Choose **Import->File or Project**.

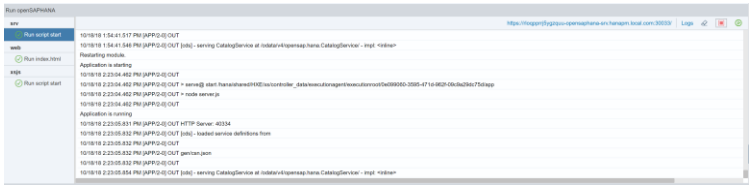


6. Press browse and select the file **exerciseChat.zip** in the Downloads folder. Keep all other selections at their defaults and press **OK**.	
7. This should import the complete UI for you.	

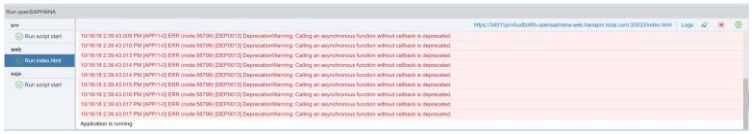
8. We can now run the **srv** module.



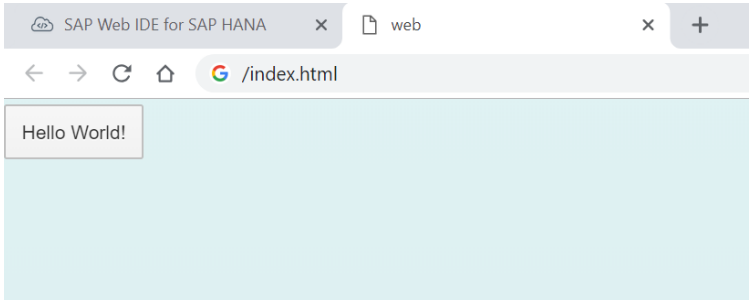
9. You should see that the build and deploy was successful.

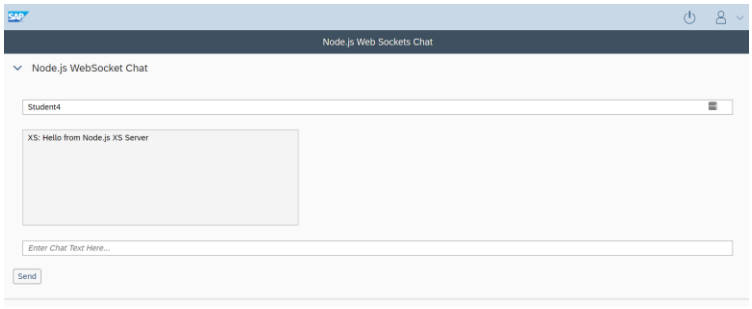
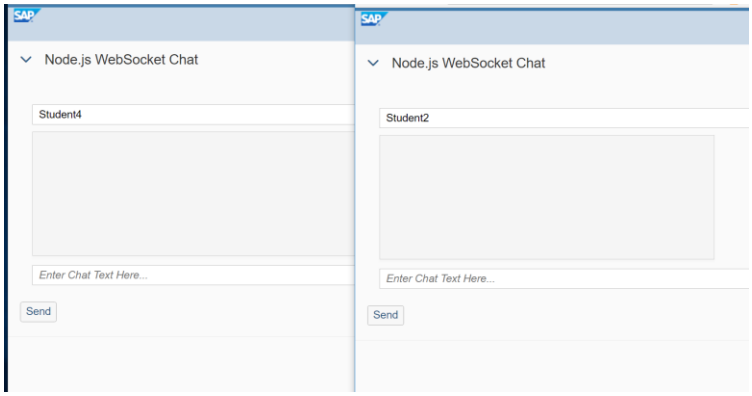
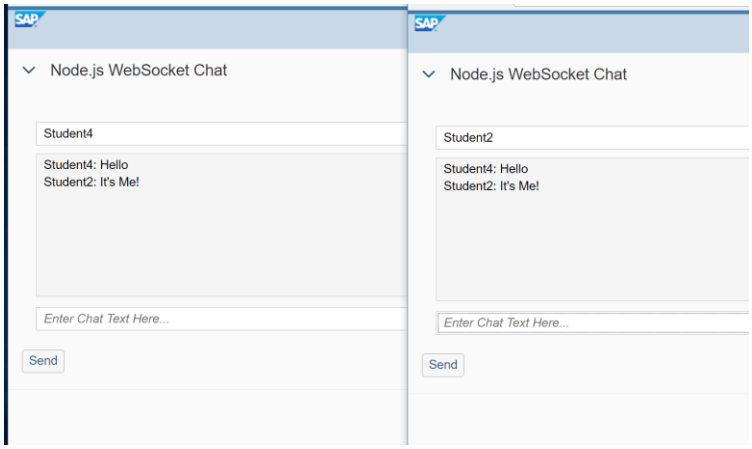


10. So now run the **web** module.



11. In the running tab, you should see the index.html from earlier.



12. Now change the path in the browser to **/exerciseChat**. You should see the simple chat user interface.	
13. Open a second browser window and cut and paste the chat application URL into it.	
14. Anything you type into either window is sent to the server and then pushed out to all listeners. If you want to test further open more than two browser windows.	

Exercise 4.10: Odata V4 Services with Exits

In this exercise we want to return to the CDS-based Odata V4 services that we created early in Exercise 4. Now that we know more about Node.js development in general, we are ready to develop more advanced Odata V4 services which utilize exits to allow us to override or extend the generic provider framework.

Explanation	Screenshot
1. Return to the **srv** module and the **my-service.cds** file. Let's begin by adding more service entities to this file. We have definitions within the db module for wrappers around several views we also want to use in our services. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/cdss1.txt	using opensap.PurchaseOrder.Headers as Headers from '../db/data-model'; using opensap.PurchaseOrder.Items as Items from '../db/data-model'; using PURCHASE_ORDER_ITEM_VIEW as ItemsView from '../db/data-model'; using BUYER as BuyerView from '../db/data-model'; using USER_DETAILS as UserDetails from '../db/data-model'; using CURRENCY as Curr from '../db/data-model'; service CatalogService { entity POs @(title: '{i18n>poService}', Capabilities: { InsertRestrictions: {Insertable: true}, UpdateRestrictions: {Updatable: true}, DeleteRestrictions: {Deletable: true} }) as projection on Headers; entity POItems @(title: '{i18n>poService}', Capabilities: { InsertRestrictions: {Insertable: true}, UpdateRestrictions: {Updatable: true}, DeleteRestrictions: {Deletable: true} }) as projection on Items; extend ItemsView with { PARTNERS: association to BuyerView on PARTNERS.PARTNERID = PARTNER_ID; } entity POItemsView @(title: '{i18n>poService}', Capabilities: { InsertRestrictions: {Insertable: false}, UpdateRestrictions: {Updatable: false}, DeleteRestrictions: {Deletable: false} }) as projection on ItemsView;

```
entity Buyer @(
                title: '{i18n>buyerService}',
                Capabilities: {
                        InsertRestrictions: {Insertable: false},
                        UpdateRestrictions: {Updatable:
false},
                        DeleteRestrictions: {Deletable: false}
                }
        ) as projection on BuyerView;

entity User @(
                title: '{i18n>userService}',
                Capabilities: {
                        InsertRestrictions: {Insertable: true},
                        UpdateRestrictions: {Updatable: true},
                        DeleteRestrictions: {Deletable: true}
                }
        ) as projection on UserDetails;

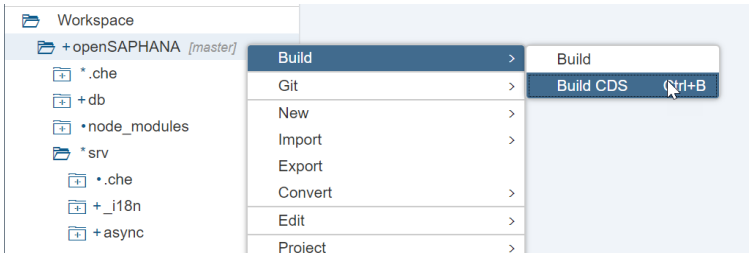
entity CURRENCY @(
                title: '{i18n>currencySrvice}',
                Capabilities: {
                        InsertRestrictions: {Insertable: false},
                        UpdateRestrictions: {Updatable:
false},
                        DeleteRestrictions: {Deletable: false}
                }
        ) as projection on Curr;
}
```

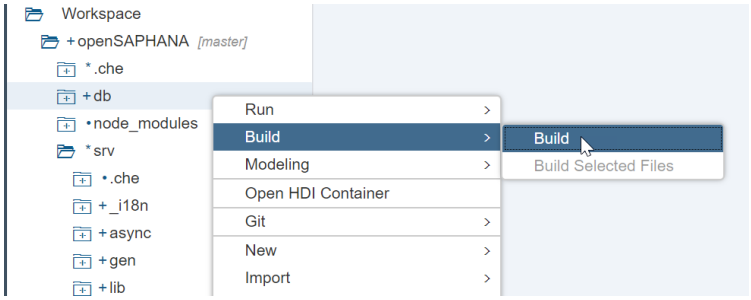
2. Not only have we imported several more entities, but you can see another intersting feature of the new CDS – the ability to easily extend existing objects. In this file we extend the ItemsView with a new association to the Partner information.

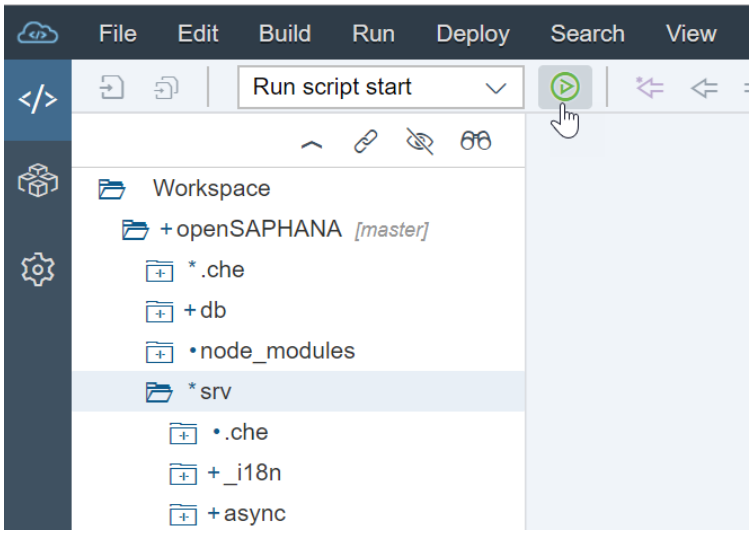
   Note: No task for this step. Just a learning point.

```
extend ItemsView with {
        PARTNERS: association to BuyerView on
PARTNERS.PARTNERID = PARTNER_ID;
}
```

3. Save the **my-service.cds** file. Perform a **Build CDS** and then build the **db** module again. Finally run the **srv** module to pickup these changes.

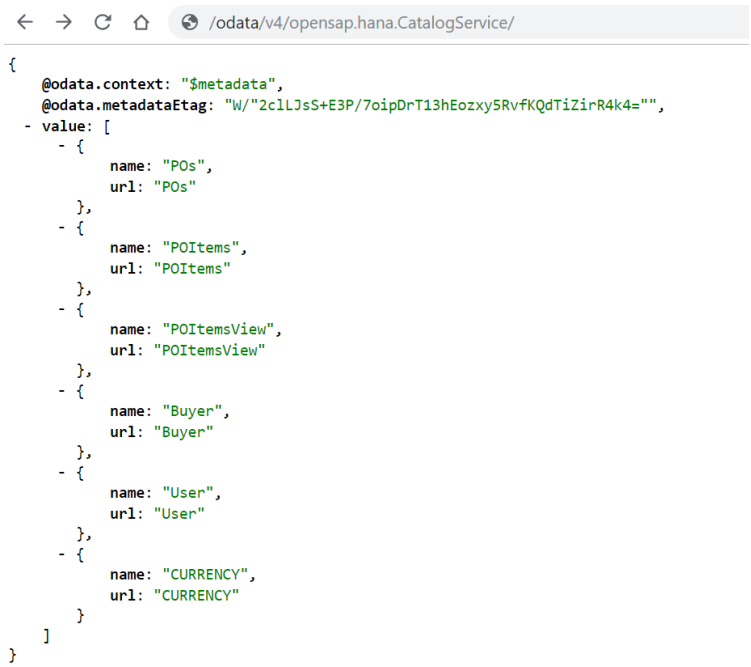




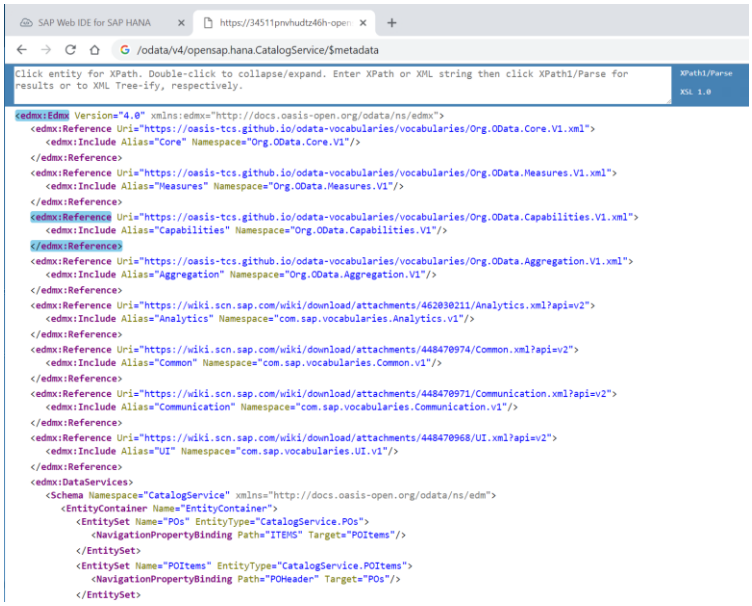


4. We can now test these new services

   The Service Catalog: **/odata/v4/opensap.hana.CatalogService/**

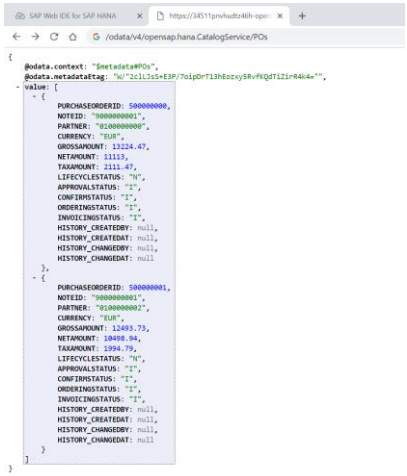


5. The service catalog metadata: **/odata/v4/opensap.hana.CatalogService/$metadata**

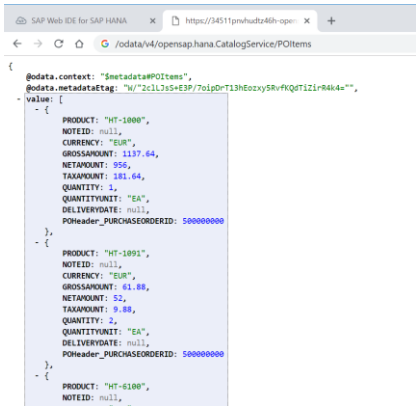


6. Check and make sure the POs and POItems services still work
/odata/v4/opensap.hana.CatalogService/POs
And
/odata/v4/opensap.hana.CatalogService/POItems

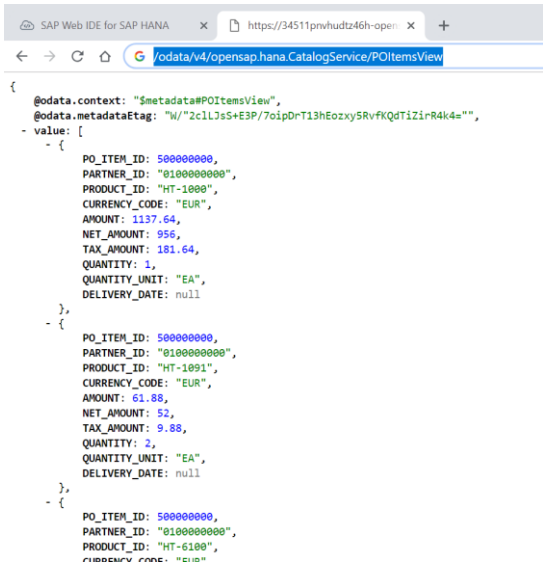
PO Headers:



PO Items:

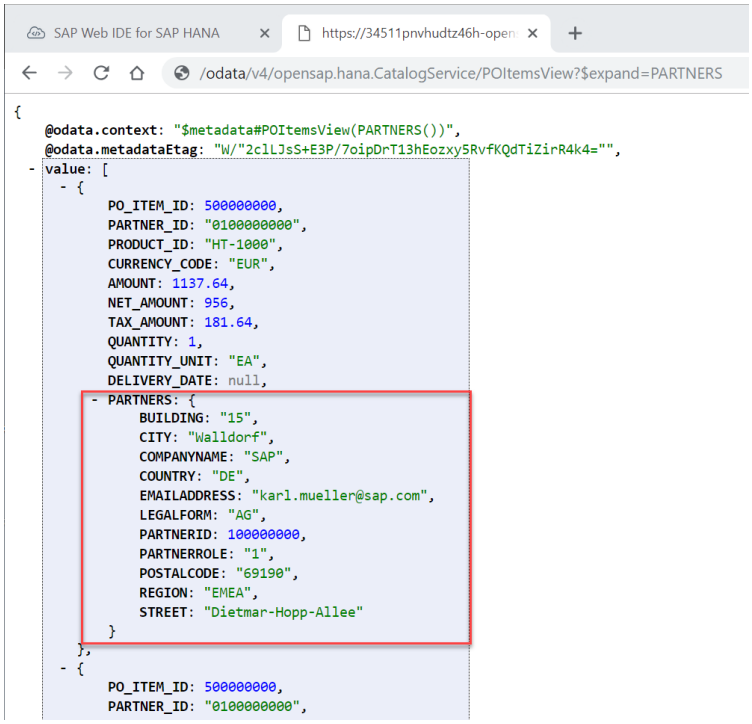


7. The new POItemsView which is built upon a HANA SQL view instead of a CDS view **/odata/v4/opensap.hana.CatalogService/POItemsView**



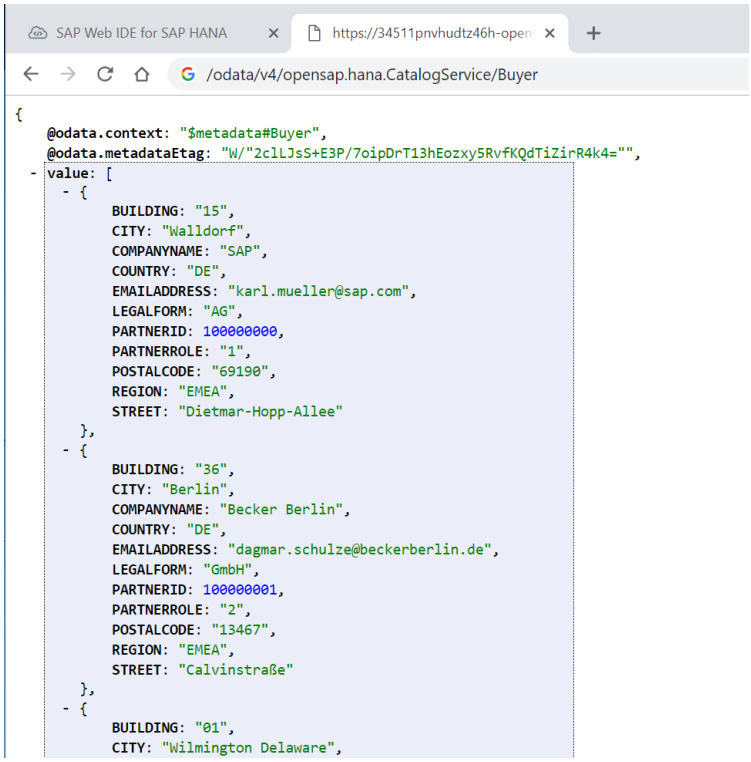
8. Also we have the assocation which we extended into the POItemsView that allows us to expand the partner details.

/odata/v4/opensap.hana.CatalogService/POItemsView?$expand=PARTNERS



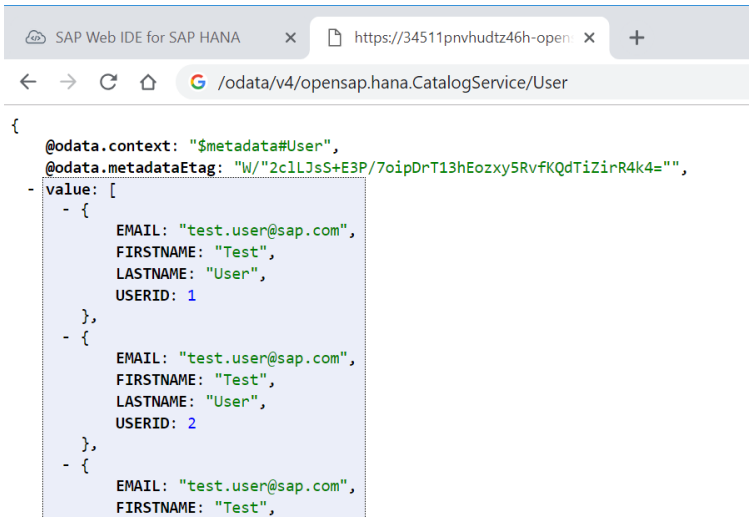
9. And of course we can directly access the Buyer entity as well. Remember this is the entity built on top of a Calculation View.

   /odata/v4/opensap.hana.CatalogService/Buyer



10. And also the User entity which is even more interesting becuase it is built on a synonym to our other hdb container (which in turn is also a calcuation view object).

    /odata/v4/opensap.hana.CatalogService/User

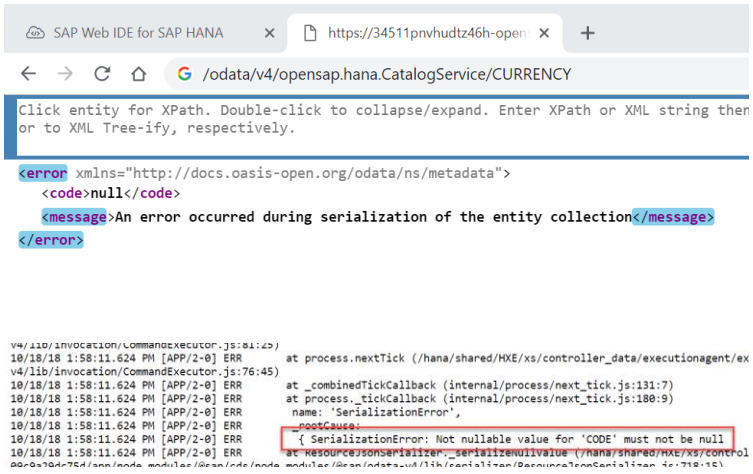


11. And finally we have the CURRENCY entity that should return a list of all the world's currencies.

    /odata/v4/opensap.hana.CatalogService/CURRENCY

    But we get an error on this entity. It seems our key field has a null value in it, which isn't allowed in Odata.

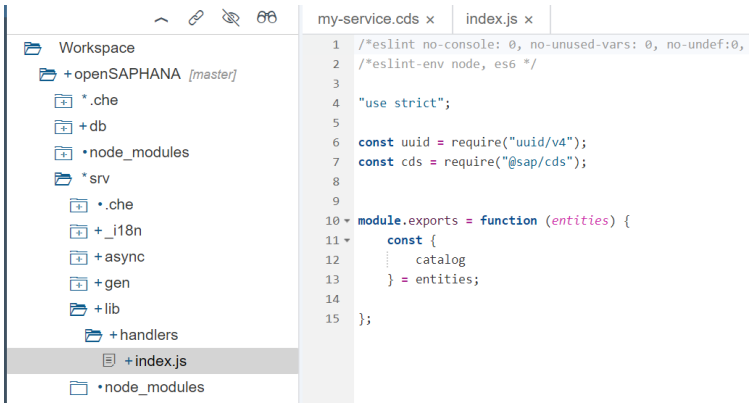
    However, this is something we can fix using an exit.



12. Remember back in Exercise 4.1, we initialized the CDS Odata service provider with a handler for exits in **/lib/handlers/index.js**

    We are ready to return to **/lib/handlers/index.js** and implement some exit handlers.





13. Let's start simple with exits. One of the major new features of CDS Odata V4 services compared to XSODATA is that we now support read exits.

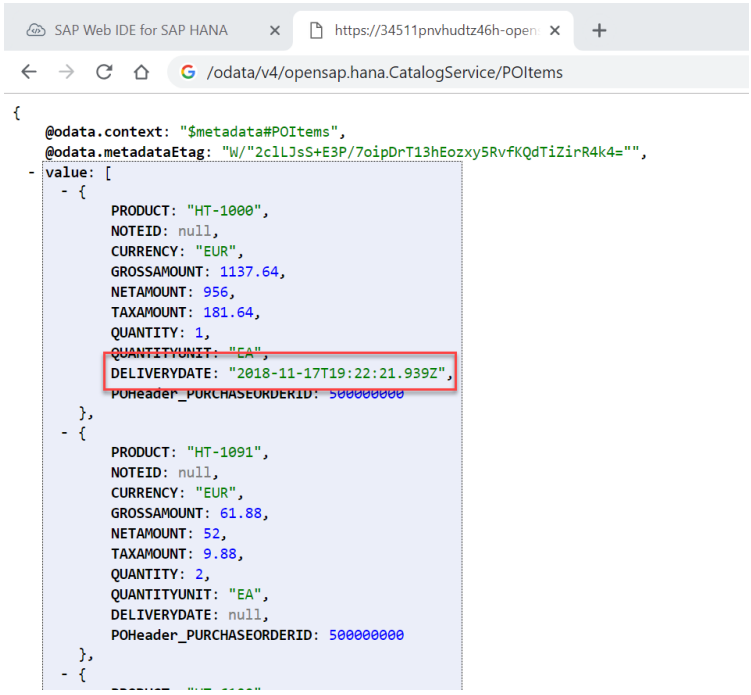
    We will utilize the "After" Read event to manipulate the data of the POItems entity after its been read from the database. We will keep this first scenario simple (and somewhat unrealistic). We will just set the PO Delivery Dates to 30 days from now for the first record.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/read1.js

```
this.after("READ", "POItems", (entity) => {
        if (entity.length > 0) {
                let now = new Date();
                let nextMonth = new Date();
                nextMonth.setDate(now.getDate() + 30);
                entity[0].DELIVERYDATE =
nextMonth.toJSON();
        }
});
```

14. Save you file and run the **srv** module.

    Rerun the test of the POItems entity at **/odata/v4/opensap.hana.CatalogService/POItems**



15. Now that we know to manipulate the service output upon read, let's use that approach to fix the null key in the **CURRENCY** entity.

    Add a second After/Read handlers but for the **CURRENCY** entity. Loop over all the restuls and if the CODE is null change the value to a string value of **NUL**.

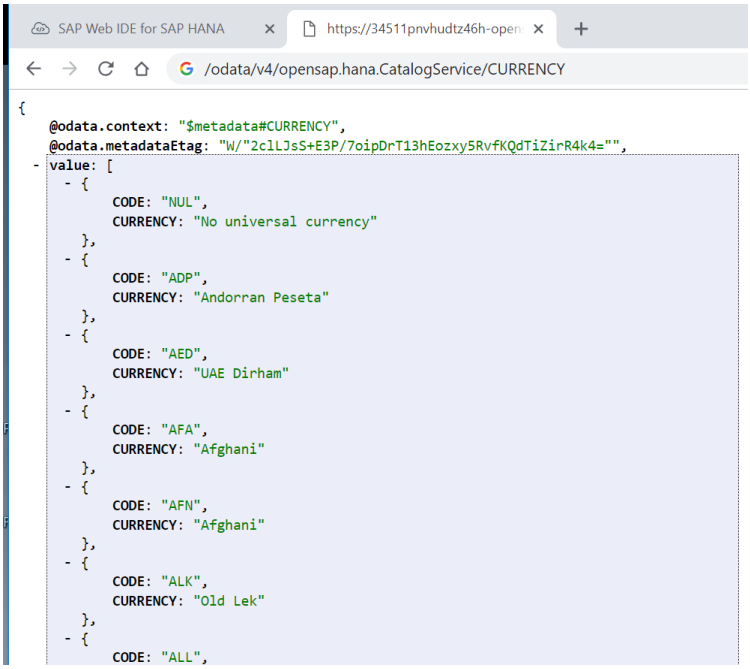
    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/read2.js

```
this.after("READ", "CURRENCY", (entity) => {
	if (entity.length > 0) {
		for (let item of entity) {
			if (item.CODE === null) {
				item.CODE = "NUL"; //remove
the NULL value
			}
		}
	}
});
```

16. Save you file and run the **srv** module.

    Rerun the test of the POItems entity at **/odata/v4/opensap.hana.CatalogService/CURRENCY**

    The error should be gone and proper data returned.



17. Following our success with the Read exits, let's advance to some Create and Update scenarios.

    Begin by extending the Creation operation of the POs entity. In the Before event, fill in the

```
this.before("CREATE",  "POs", (Orders) => {
	console.log("Before Create");
	const {
		data,
		run
	} = Orders;
	//data.ID = uuid();
	var now = new Date();
```

HISTORY_CREATEDAT with the current date. This makes sense that we wouldn't rely upon the service caller to always fill in the created date but instead calculate it within the creation operation. We will also use the POs after event to just log the event. For now we won't test, but in the next until we will build user interfaces which call this service.

Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/create1.js

```
		data.HISTORY_CREATEDAT =
now.toJSON();
	});

	this.after("CREATE",  "POs", (Orders) => {
		console.log("After Create");
	});
```

18. The final exit handlers will be the most complex. We want to support create and update on the User entity. However the source of this entity is a Calcuation View.

    The generic Odata service provider framework can't handle these operations on a view, so we will use the On event to completely override the Create and Update operations on this entity. This means we are responsible for both the validation and the database operation.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/create2.js

```
	function validateEmail(email) {
		var re =

	/^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))
@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-
9]+\.)+[a-zA-Z]{2,}))$/;
		return re.test(email);
	}

	this.on("CREATE", "User", async(User) => {
		console.log("Before User Create");
		const {
			data
		} = User;
		if (data.length < 1) {
			return null;
		}

		if (!validateEmail(data.EMAIL)) {
			throw new Error(`Invalid email for
${data.FIRSTNAME}. No Way! E-Mail must be valid and
${data.EMAIL} has problems`);
		}

		const dbClass = require(global.__base +
"utils/dbPromises");
		var client = await dbClass.createConnection();
		let db = new dbClass(client);
```

```
                const insStatement = await
db.preparePromisified(
                        `INSERT INTO "UserData.User"
                                ("FirstName", "LastName",
"Email")
                                VALUES (?,?,?)`
                );
                const updResults = await
db.statementExecPromisified(insStatement,
[data.FIRSTNAME, data.LASTNAME, data.EMAIL]);

                const statement = await
db.preparePromisified(
                        `SELECT
CURRENT_IDENTITY_VALUE() AS ID
                                                FROM
DUMMY`);
                const userResults = await
db.statementExecPromisified(statement, []);
                data.USERID = userResults[0].ID;
                console.log(JSON.stringify(data));
                return data;

        });

        this.on("UPDATE", "User", async(User) => {
                console.log("Before User Update");
                const {
                        data
                } = User;
                if (data.length < 1) {
                        return null;
                }

                const dbClass = require(global.__base +
"utils/dbPromises");
                var client = await dbClass.createConnection();
                let db = new dbClass(client);
                const statement = await
db.preparePromisified(
                        `SELECT * FROM USER_DETAILS
                                                WHERE
USERID = ?`);
                const userResults = await
db.statementExecPromisified(statement, [data.USERID]);
                Object.keys(data).forEach(function (key) {
                        userResults[0][key] = data[key];
                });
                const updStatement = await
db.preparePromisified(
                        `UPDATE "UserData.User" SET
                                "FirstName" = ?,
                                "LastName" = ?,
                                "Email" = ?
```

	WHERE "UserId" = ?`); const updResults = await db.statementExecPromisified(updStatement, [userResults[0].FIRSTNAME, userResults[0].LASTNAME, userResults[0].EMAIL, userResults[0].USERID]); return userResults[0];
19. Be sure to save your **index.js** file and re-run the **srv** module. However we won't be able to fully test these update and create operations until the next exercise where we will finally build our own user interfaces upon these services.	

EXERCISE 5 – SAPUI5/FIORI USER INTERFACE

Objective

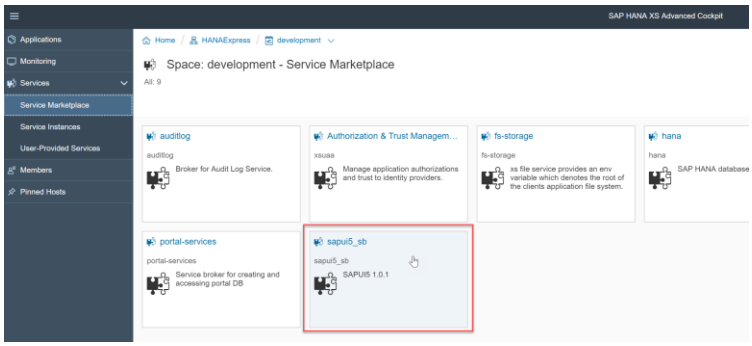
For this exercise we will build a proper SAPUI5 user interface that consumes both the XSJS and XSODATA services as well as CDS based Odata V4 services from our Node.js module.

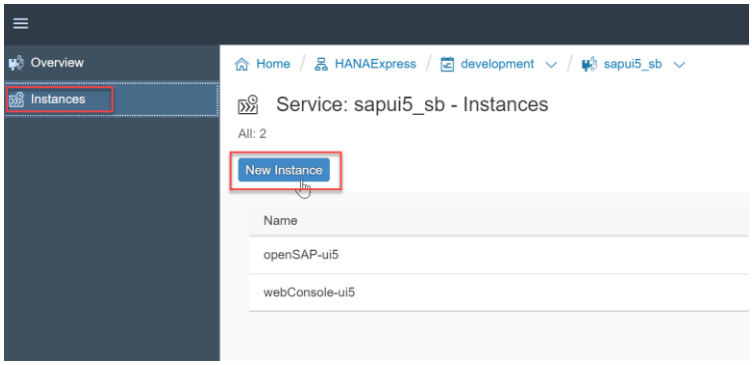
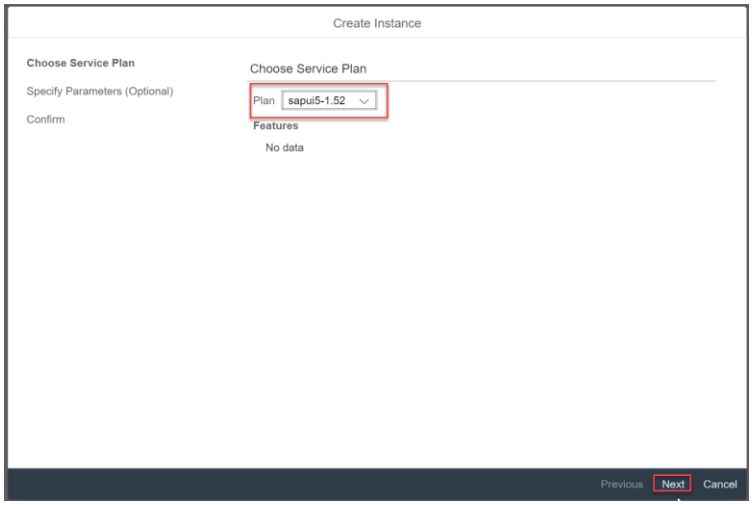
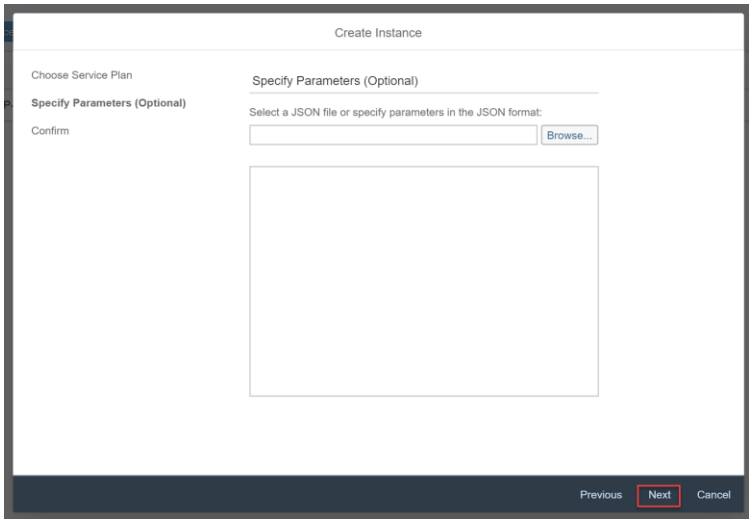
Exercise Description

- SAPUI5 Entry Point
- SAPUI5 Component
- SAPUI5 Views
- SAPUI5 Controllers

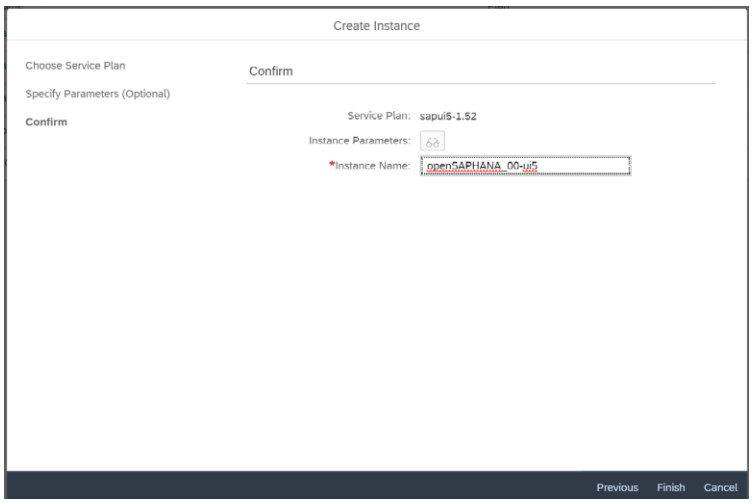
EXERCISE 5 – SOLUTION

Exercise 5.1: SAPUI5 as an XSA Micro-Service

Explanation	Screenshot
1. In Exercise 1 a simple SAPUI5 Hello World application was generated for us by the module creation wizard. The boostrap for SAPUI5 points the publically hosted UI5 library. However a recent addition to XSA was to deliver SAPUI5 as its own Micro-Service for local consumption. Let's now adjust our **web** project and this **index.html** hello world example to use the local SAPUI5 Micro-service. **Note:** if your HANA system doesn't yet contain this SAPUI5 micro-service then you can skip this exercise continue to use the public SAPUI5 location in subsequent exercises.	
2. Launch the **SAP HANA XS Advanced Cockpit** at the following URL in your web browser via **Tools->SAP HANA XS Advanced Cockpit.** Navigate to your organization **HANAExpress** and space **development**. Expand **Services** and then click on **Service Marketplace**. Select **sapui5_sb**.	

3. Click on **Instances** and press the **New Instance** button.	
4. Choose Plan **sapui5-1.52** and then press **Next**.	
5. We have no Parameters, so just press **Next**.	

6. Specify an **Instance Name** of **openSAPHANA_XX-ui5** (where XX is your group number) and then press **Finish**

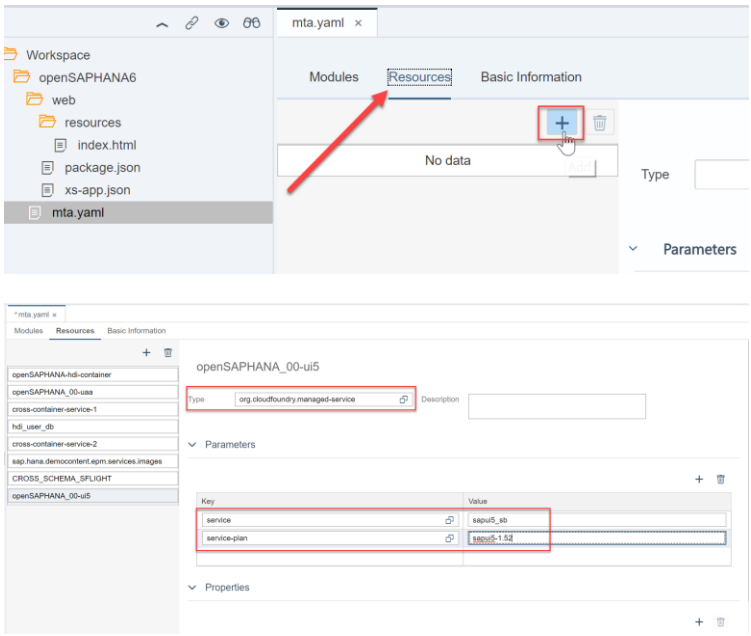


7. We need to add **openSAPHANA_XX-ui5** (where XX is your group number) as a dependent service/resource to the **mta.yaml** file.

   Click on the **Resources** tab. Then click the plus button to add a new resource entry. This will make a new input field appear below the plus button. You can type the name of the resource in here.

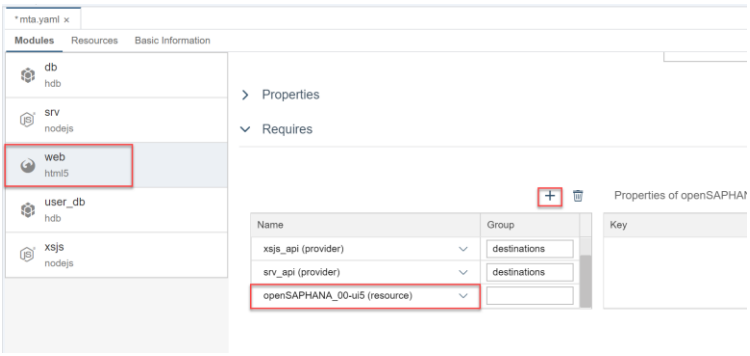
   Then on the right hand side you can fill in the type and parameters of the resource.

   Save your file. Switch to the Code Editor and then back to the MTA Editor to refresh all the value helps.



8. Next you need to add the UI5 resource as a dependency in the **web** module.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_2




```
- name: web
  type: html5
  path: web
  requires:
    - name: openSAPHANA_00-uaa
    - name: xsjs_api
      group: destinations
      properties:
        name: xsjs_api
        url: '~{url}'
        forwardAuthToken: true
    - name: srv_api
      group: destinations
      properties:
        name: srv_api
        url: '~{url}'
        forwardAuthToken: true
    - name: openSAPHANA_00-ui5
```

9. We don't want to hard code the url of the UI5 service in th bootstrap in every html file in our project. We can instead use the replacment logic of the App Router to insert the runtime URL for us dynamically.

   For this we need to add a route configuration to our **xs-app.json** in the **web** module.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_3



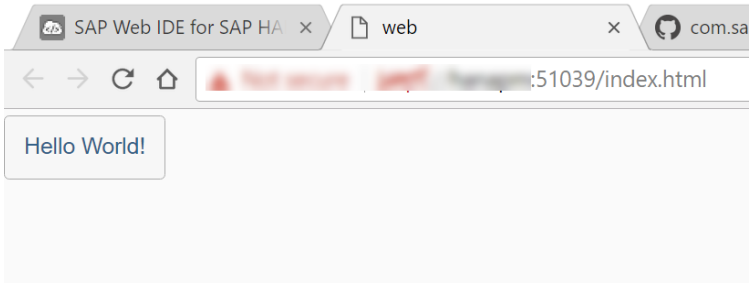
10. The final step is to adjust the **index.html** file in the **web/resources** folder to use this ui5liburl placholder variable instead of the external address for the UI5 library in the bootstrap.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_4

```
<title>web</title>
    <!-- <script id="sap-ui-bootstrap" src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js" -->
    <script id="sap-ui-bootstrap" src="{{{ui5liburl}}}/resources/sap-ui-core.js"
    data-sap-ui-libs="sap.m"
    data-sap-ui-theme="sap_belize">
        </script>
<!-- 2.) Create a UI5 button and place it onto the page -->
<script>
```

11. Save any open files. Run the **web** module.

    If the change was sucessful, the hello world index.html should load as normal.



12. If you would like to confirm the change you can perform a view source in the web browser or use browser developer tools to view the request.

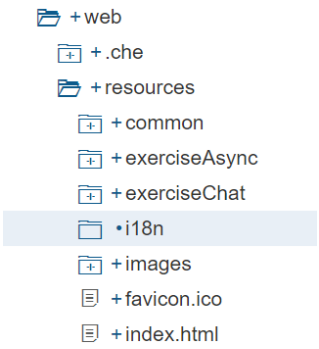
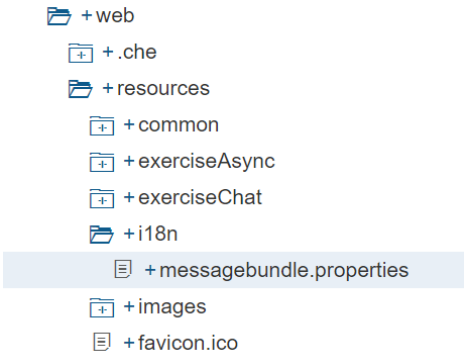
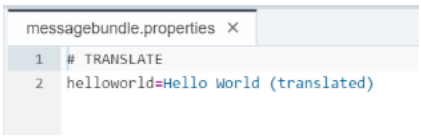
```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta http-equiv='Content-Type' content='text/html;charset=UTF-8' />
<title>web</title>
<!-- 1.) Load SAPUI5 (from a remote server), select theme and control library -->
        <!-- <script id="sap-ui-bootstrap" src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js" -->
    <script id="sap-ui-bootstrap" src="https://hanaexpress-sap-sapui5-fesv4.hanapm.local.com:30033/resources/sap-ui-core.js"
    data-sap-ui-libs="sap.m"
    data-sap-ui-theme="sap_belize_plus">
        </script>
<!-- 2.) Create a UI5 button and place it onto the page -->
<script>
    // create the button instance
    var myButton = new sap.m.Button("btn");

    // set properties, e.g. the text (there is also a shorter way of setting several properties)
    myButton.setText("Hello World!");

    // attach an action to the button's "press" event (use jQuery to fade out the button)
    myButton.attachPress(function(){$("#btn").fadeOut()});

    // place the button into the HTML element defined below
    myButton.placeAt("uiArea");
    </script>
</head>
<body class="sapUiBody">
    <!-- This is where you place the UI5 button -->
    <div id="uiArea"></div>
</body>
</html>
```

Exercise 5.2: Creating a Text Bundle

Explanation	Screenshot
1. In most of these exercises we will simply hard code our texts to keep everything simple. However in real applications, text strings should be maintained separately so they can easily be translated. Now we will adjust the HelloWorld application to use a text bundle.	
2. Create a new folder within your **web/resources** folder called **i18n**.	
3. In this folder create a file named **messagebundle.properties**.	
4. Add the following text to this file. Save the file.	
5. Go to the **index.html**. Add the following lines just before the myButton creation.	`/*************** Language Resource Loader ************/` `jQuery.sap.require("jquery.sap.resources");` `var sLocale =` `  sap.ui.getCore().getConfiguration().getLanguage();`

```
var oBundle = jQuery.sap.resources({url :
  "./i18n/messagebundle.properties", locale: sLocale});
```

6. Your index.html now looks like this:

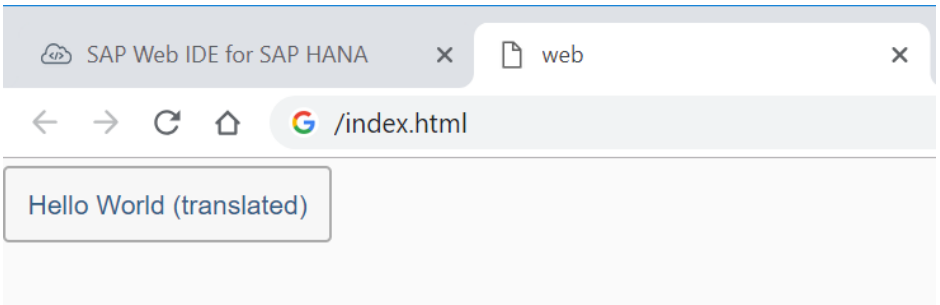


7. Change the setText function of the button to use the text bundle.

```
myButton.setText(oBundle.getText("helloworld"));
```

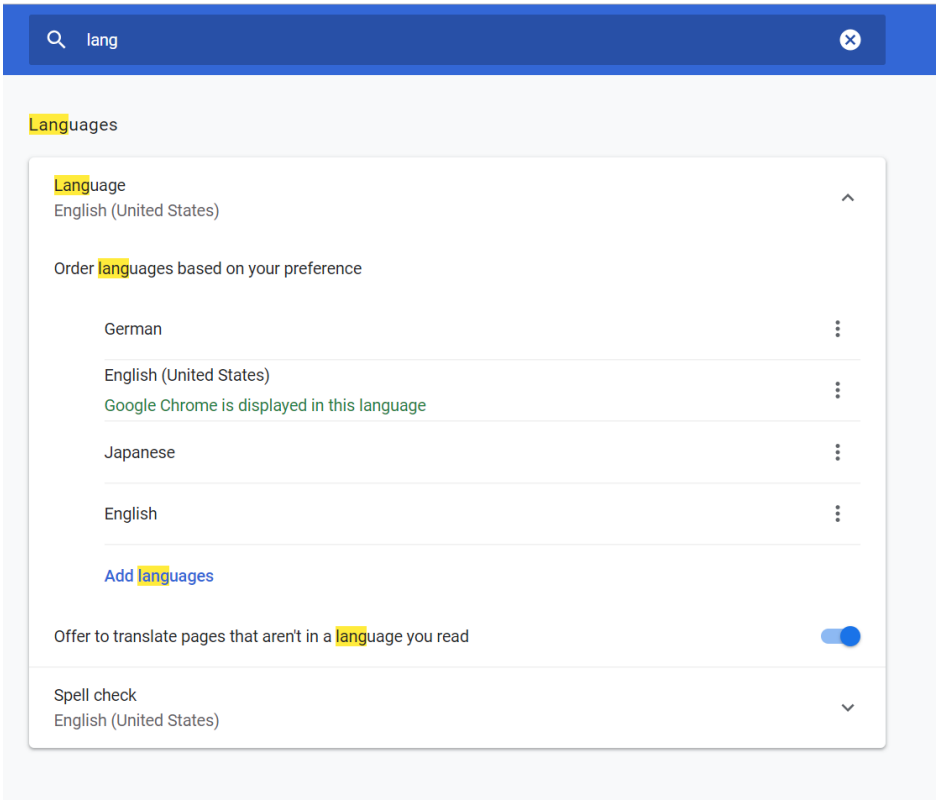
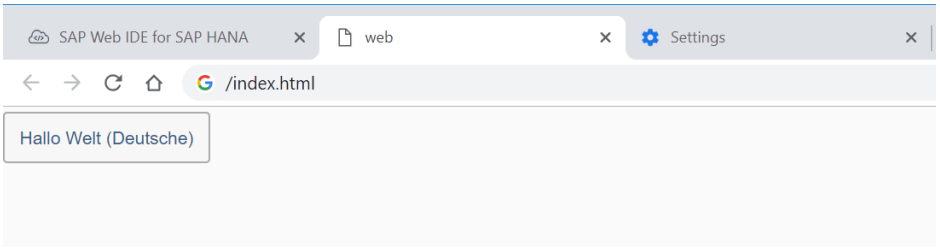


8. Save and run your **web** module. The new Hello World text should appear.

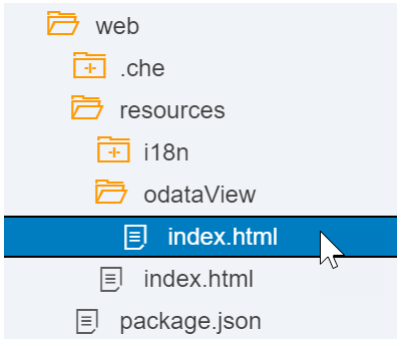


9. You can return to your project and create a second file in your i18n folder named **messagebundle_de.properties** with the shown translated text for the helloworld id.



Save and re-run your **web** module.	
10. You can change your browser settings so that German is now the primary language.	
11. The test page will now automatically show the translated German text. Afterwards, change the primary language back to English.	

Exercise 5.3: SAPUI5 User Interface

Explanation	Screenshot
1. Now we want to build a proper SAPUI5 interface that will consume our XSJS and XSODATA services. Let's return to our web module and create a new folder named **web/resources/odataView** a new html file in the this folder called **index.html**	
2. Here is the complete coding of the **index.html** page. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_5 We are loading the SAPUI5 bootstap and then initializing the shell and component.	<!DOCTYPE html> <html> <head> <meta http-equiv="X-UA-Compatible" content="IE=edge" /> <meta http-equiv="Content-Type" content="text/html;charset=UTF-8"/> <meta name="viewport" content="width=device-width, initial-scale=1.0" /> <link type="image/x-icon" href="/images/favicon.ico" rel="shortcut icon"> <link type="image/x-icon" href="/images/favicon.ico" rel="icon"> <!-- <script id="sap-ui-bootstrap" src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js" --> <script id="sap-ui-bootstrap" src="{{{sapui5_sb.url}}}/resources/sap-ui-core.js" data-sap-ui-theme="sap_belize_plus" data-sap-ui-xx-bindingSyntax="complex" data-sap-ui-compatVersion="edge" data-sap-ui-preload="async" data-sap-ui-language="en" data-sap-ui-resourceroots='{ "opensap.odataTest": "./"}' data-sap-ui-libs="sap.m,sap.ui.comp,sap.ui.core,sap.ui.layout,sap.ui.unified" data-sap-ui-frameOptions="trusted"> </script> <script type="text/javascript" src="/common/error.js"></script> <script type="text/javascript" src="../common/startup.js"></script> <script> localShellStartup("opensap.odataTest");

	</script> </head> <!-- UI Content --> <body class="sapUiBody" id="content"> </body> </html>
3. Next we need to create our **Component.js** file in **web/resources/odataView**. Here is the coding of this file. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_8 Here we initialize the SAPUI5 component and in doing so we create an instance of the json configuration model. We also load our first view which we will create in the next step.	/*eslint no-console: 0, no-unused-vars: 0, no-use-before-define: 0, no-redeclare: 0*/ sap.ui.define(["sap/ui/core/UIComponent", "sap/ui/Device", "opensap/odataTest/model/models"], function(UIComponent, Device, models) { "use strict"; return UIComponent.extend("opensap.odataTest.Component", { metadata: { manifest: "json" }, init: function() { jQuery.sap.require("sap.m.MessageBox"); jQuery.sap.require("sap.m.MessageToast"); this.setModel(models.createDeviceModel(), "device"); sap.ui.core.UIComponent.prototype.init.apply(this, arguments); }, destroy: function() { // call the base component's destroy function UIComponent.prototype.destroy.apply(this, arguments); } }); });
4. At the beginning of the Component.js processing you might notice the section that refers to a manifest for the metadata. This allows us to separate out a lot of the configuration of our application	{ "_version": "1.4.0", "start_url": "index.html", "sap.app": {

into a separate manifest.json file (as most modern Fiori based UI5 applications are designed).

Create this **manifest.json** file in your **odataView** folder and add the following content to it.

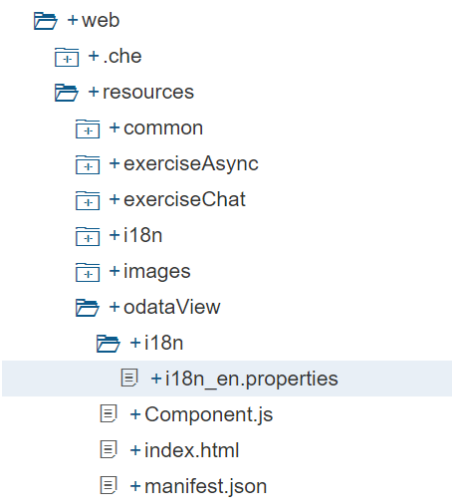
This JSON file specifies libraries and other technical details, but more importantly sets the initial view details as well as creates the json and Odata models.

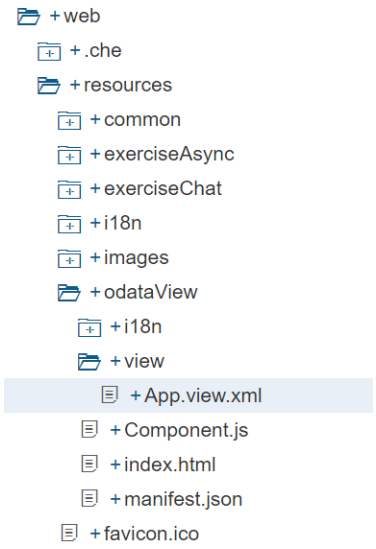
Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_8a

```
        "_version": "1.4.0",
        "type": "application",
        "resources": "resources.json",
        "i18n": "i18n/i18n.properties",
        "id": "odataView",
        "title": "{{appTitle}}",
        "description": "{{appDescription}}",
        "applicationVersion": {
                "version": "${project.version}"
        }
},
"sap.fiori": {
        "_version": "2.0.0",
        "registrationIds": [],
        "archeType": "transactional"
},
"sap.ui": {
        "_version": "1.40.0",
        "technology": "UI5",
         "icons":  {
                "icon": "./images/favicon.ico",
                "favIcon": "./images/favicon.ico"
         },
        "deviceTypes": {
                "desktop": true,
                "tablet": true,
                "phone": true
        },
        "supportedThemes": [
                "sap_hcb",
                "sap_bluecrystal",
                "sap_belize"
        ]
},
"sap.ui5": {
  "config": {
    "sapFiori2Adaptation": true
  },
  "rootView": {
        "viewName": "opensap.odataTest.view.App",
        "type": "XML",
```

```
        "id": "app"
    },
    "dependencies": {
                "minUI5Version": "1.40.0",
                "libs": {
                    "sap.ui.core": {
                        "minVersion": "1.40.0"
                    },
                    "sap.ui.comp": {
                        "minVersion": "1.40.0"

                    },
                    "sap.m": {
                        "minVersion": "1.40.0"
                    },
                    "sap.ui.layout": {
                        "minVersion": "1.40.0"
                    }
                }
            },
        "contentDensities": {
            "compact": true,
            "cozy": true
        },
    "handleValidation": true,
    "models": {
        "": {
            "type": "sap.ui.model.json.JSONModel",
            "settings": {
                "defaultBindingMode": "TwoWay"
            }
        },
        "config": {
            "type": "sap.ui.model.json.JSONModel"
        },
        "i18n": {
            "type": "sap.ui.model.resource.ResourceModel",
            "settings": {
                "bundleUrl": "./i18n/i18n.properties"
            }
        }
```

	} } }
5. We want to put our text strings into a text bundle. Create a folder inside **odataView** named **i18n** and then a file named **i18n_en.properties**.	
6. Add the following text strings to this **i18n_en.properties** file. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_8b	# This is the resource bundle for OData View Exercise # __ldi.translation.uuid=1aadca5c-73bb-4086-ae70-98149af3cde1 #XTIT: Application name appTitle=Workshop OData Test #YDES: Application description appDescription=Workshop OData Test H1=Multi-Entity Read P1=Multi-Entity Service Selection L1=Service Path L2=Header Entity Name L3=Item Entity Name B1=Execute Service B2=Download Excel POHeader=PO Header Data POItem=PO Item Data #~~~ Not Found View ~~~~~~~~~~~~~~~~~~~~~~~~ #XTIT: Not found view title notFoundTitle=Not Found

	#YMSG: The not found text is displayed when there was an error loading the resource (404 error) notFoundText=The requested resource was not found #~~~ Error Handling ~~~~~~~~~~~~~~~~~~~~~~~~ #YMSG: Error dialog description errorText=Sorry, a technical error occurred! Please try again later.
7. Create a folder called **view** inside **odataView**. This is where we will hold all of our views. Create the root view named **App.view.xml**.	
8. Here is the complete coding of the **App.view.xml** file. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_9 It creates the shell control and the overall flow of our page, but then defers the design of the header and table areas to an XML fragment we will create next.	<mvc:View xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" xmlns:core="sap.ui.core" controllerName="opensap.odataTest.controller.App" class="sapUiSizeCompact" height="100%"> <Page title="{i18n>appTitle}"> <content> <IconTabBar class="iconTabBarPaddingTop" upperCase="true" expanded="true"> <items> <IconTabFilter text="{i18n>H1}"> <core:Fragment fragmentName="opensap.odataTest.view.MRead" type="XML"/> </IconTabFilter> </items> </IconTabBar> </content> </Page> </mvc:View>

9. Create a file named **MRead.fragment.xml** in the **view** folder.

   Here is the complete coding of this file.

   Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_10

   Here we build the panel with our input fields. These fields will allow us to control settings for calling our XSODATA service. We then will have a table control for our purchase order header and another for our purchase order item. The rendering of these two sections if further separated out into two JavaScript based fragments.

```
<core:FragmentDefinition xmlns:core="sap.ui.core"
xmlns="sap.m">
	<Panel expandable="true" expanded="true"
headerText="{i18n>P1}">
		<List width="100%">
			<InputListItem label="{i18n>L1}">
				<Input id="mPath"
value="{/mPath}"/>
			</InputListItem>
			<InputListItem label="{i18n>L2}">
				<Input id="mEntity1"
value="{/mEntity1}"/>
			</InputListItem>
			<InputListItem label="{i18n>L3}">
				<Input id="mEntity2"
value="{/mEntity2}"/>
			</InputListItem>
		</List>
		<Button press="callMultiService"
text="{i18n>B1}"/>
		<Button press="callExcel" text="{i18n>B2}"/>
	</Panel>
	<core:Fragment
fragmentName="opensap.odataTest.view.MTableHead"
type="XML"/>
	<core:Fragment
fragmentName="opensap.odataTest.view.MTableItem"
type="XML"/>
</core:FragmentDefinition>
```

10. Create the **MTableHead.fragment.xml** file. Here is the complete coding for this file.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_11

    We are creating a table control to display our Purchase Order Header data which will be returned by the XSODATA service.

```
<core:FragmentDefinition xmlns:core="sap.ui.core"
xmlns="sap.m" xmlns:smartTable="sap.ui.comp.smarttable">
	<VBox fitContainer="true">
		<smartTable:SmartTable id="tblPOHeader"
tableType="Table" header="{i18n>POHeader}"
showRowCount="true" enableAutoBinding="true"

	showFullScreenButton="true"></smartTable:SmartTab
le>
	</VBox>
</core:FragmentDefinition>
```

11. Repeat the process for the PO Item table. Filename **MTableItem.fragment.xml**.

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_12

```
<core:FragmentDefinition xmlns:core="sap.ui.core"
xmlns="sap.m" xmlns:smartTable="sap.ui.comp.smarttable">
        <smartTable:SmartTable id="tblPOItem"
tableType="Table" header="{i18n>POItem}"
showRowCount="true" enableAutoBinding="true"
                showFullScreenButton="true"/>
</core:FragmentDefinition>
```

12. The next piece of our UI is the view controller named **App.controller.js**. Create a folder named **controller** in **odataView** and then the file named **App.controller.js**

    Note: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_13

    There is some considerable code in this file because here we have all the event handlers for our application.

    We start with the onInit event which fires on the startup of the page. We set some initial values into the configuration model so they appear in our input fields.

    callMultiService handles the logic to call our OData service. We are reading the metadata from the OData service in order to dynamically create the columns in our table controls.

    onRowSelect handles the selection of a PO Header record and forces the loading of the corresponding PO Item records.

```
/*eslint no-console: 0, no-unused-vars: 0, no-use-before-
define: 0, no-redeclare: 0, no-undef: 0*/
//To use a javascript controller its name must end with
.controller.js
sap.ui.define([
        "opensap/odataTest/controller/BaseController",
        "sap/ui/model/json/JSONModel"
], function(BaseController, JSONModel) {
        "use strict";

        return
BaseController.extend("opensap.odataTest.controller.App", {

                onInit: function() {

                        var oConfig =
this.getOwnerComponent().getModel("config");
                        var userName =
oConfig.getProperty("/UserName");

                        var urlMulti =
"/xsodata/purchaseOrder.xsodata";

        this.getOwnerComponent().getModel().setProperty("/m
Path", urlMulti);

        this.getOwnerComponent().getModel().setProperty("/m
Entity1", "POHeader");

        this.getOwnerComponent().getModel().setProperty("/m
Entity2", "POItem");

                },
                callMultiService: function() {
                        var oTable =
this.getView().byId("tblPOHeader");
                        var oTableItem =
this.getView().byId("tblPOItem");

                        var mPath =
this.getOwnerComponent().getModel().getProperty("/mPath");
```

Finally **callExcel** responds to a button press and calls /xsjs/hdb.xsjs to download our PO data in a text tab delimited format.

```
                        var mEntity1 =
this.getOwnerComponent().getModel().getProperty("/mEntity1"
);
                        var mEntity2 =
this.getOwnerComponent().getModel().getProperty("/mEntity2"
);

                        var oParams = {};
                        oParams.json = true;
                        oParams.useBatch = true;
                        var oModel = new
sap.ui.model.odata.v2.ODataModel(mPath, oParams);
                        oModel.attachEvent("requestFailed",
oDataFailed);

                        function fnLoadMetadata() {
                                oTable.setModel(oModel);
                                oTable.setEntitySet(mEntity1);

        oTableItem.setModel(oModel);

        oTableItem.setEntitySet(mEntity2);

                                var oMeta =
oModel.getServiceMetadata();
                                var headerFields = "";
                                var itemFields = "";
                                for (var i = 0; i <
oMeta.dataServices.schema[0].entityType[0].property.length;
i++) {
                                        var property =
oMeta.dataServices.schema[0].entityType[0].property[i];
                                        headerFields +=
property.name + ",";
                                }

                                for (var i = 0; i <
oMeta.dataServices.schema[0].entityType[1].property.length;
i++) {
                                                var property =
oMeta.dataServices.schema[0].entityType[1].property[i];
                                                itemFields +=
property.name + ",";
                                }

        oTable.setInitiallyVisibleFields(headerFields);

        oTableItem.setInitiallyVisibleFields(itemFields);
                        }


        oModel.attachMetadataLoaded(oModel, function() {
                                fnLoadMetadata();
                        });
```

	oModel.attachMetadataFailed(oModel, function() { sap.m.MessageBox.show("Bad Service Definition", { icon: sap.m.MessageBox.Icon.ERROR, title: "Service Call Error", actions: [sap.m.MessageBox.Action.OK], styleClass: "sapUiSizeCompact" }); }); }, callExcel: function(oEvent) { //Excel Download window.open("/node/excel/download/"); return; } }); });
13. We can also separate certain controller functions out into a reusable base controller that can be shared by mulitple controllers. Although this first simple example only has one controller, let's go ahead and setup this base controller (named **BaseController.js** in the **controller** folder) for future exercises. **Note**: if you don't want to type this code, we recommend that you cut and paste it from this web address https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_13a	/*global history */ sap.ui.define(["sap/ui/core/mvc/Controller", "sap/ui/core/routing/History"], function (Controller, History) { "use strict"; return Controller.extend("opensap.odataTest.controller.BaseController", { /** * Convenience method for accessing the router in every controller of the application. * @public * @returns {sap.ui.core.routing.Router} the router for this component */ getRouter : function () { return this.getOwnerComponent().getRouter(); }, /** * Convenience method for getting the view model by name in every controller of the application. * @public * @param {string} sName the model name

```
                                * @returns {sap.ui.model.Model} the
model instance
                                */
                               getModel : function (sName) {
                                       return
this.getView().getModel(sName);
                               },

                               /**
                                * Convenience method for setting the
view model in every controller of the application.
                                * @public
                                * @param {sap.ui.model.Model}
oModel the model instance
                                * @param {string} sName the model
name
                                * @returns {sap.ui.mvc.View} the
view instance
                                */
                               setModel : function (oModel, sName) {
                                       return
this.getView().setModel(oModel, sName);
                               },

                               /**
                                * Convenience method for getting the
resource bundle.
                                * @public
                                * @returns
{sap.ui.model.resource.ResourceModel} the resourceModel of
the component
                                */
                               getResourceBundle : function () {
                                       return
this.getOwnerComponent().getModel("i18n").getResourceBun
dle();
                               },

                               /**
                                * Event handler for navigating back.
                                * It there is a history entry we go one
step back in the browser history
                                * If not, it will replace the current entry
of the browser history with the master route.
                                * @public
                                */
                               onNavBack : function() {
                                       var sPreviousHash =
History.getInstance().getPreviousHash();

                                               if (sPreviousHash !==
undefined) {
                                               history.go(-1);
                                       } else {
```

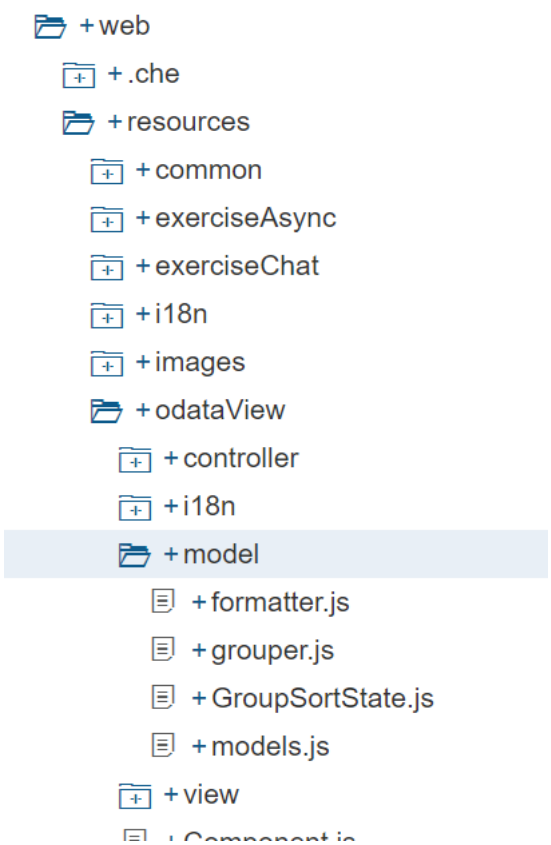
```
            this.getRouter().navTo("master", {}, true);
                                    }
                                }

                    });

            }
);
```

14. We can also have reusable functions like formatters, sorters, and grouping functions. We've prepared these model support tools for you. You can download the **model.zip** file from here: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/model.zip?raw=true

    Please import this file into the **odataView** folder.

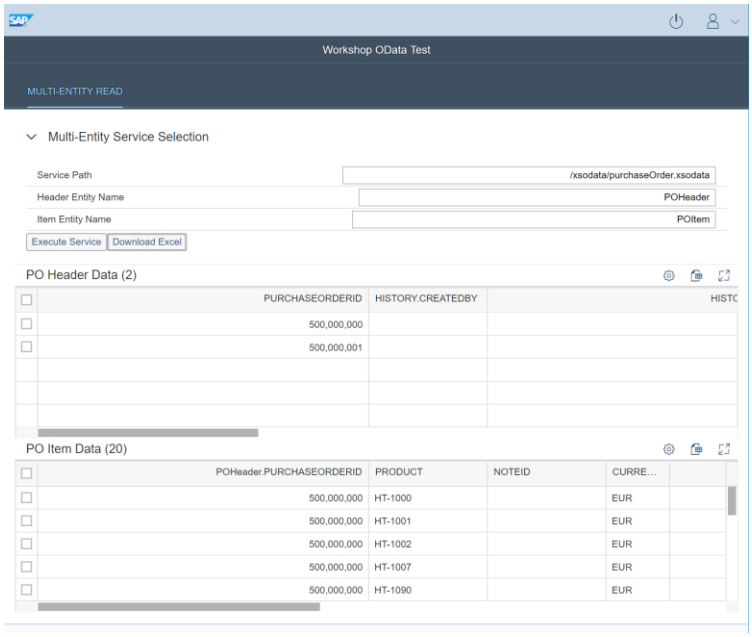


15. Now run the **web** module. It will need to rebuild and redeploy all the newly added artifacts.

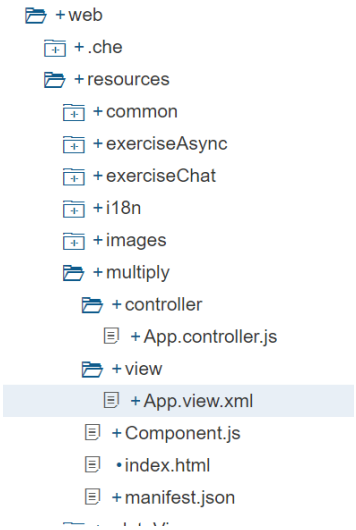


16. In the running tab, you should see the index.html from earlier. Change this to **/odataView** to test the new UI we just created.

    Test the **Execute Service** button. The Download Excel button backend service won't be implemented until the next Exercise.



Exercise 5.4: Consume XSJS Services via JQuery AJAX calls

Explanation	Screenshot
1. Using what you have learned create a new folder named **multiply** in **web/resources** with a similar structure to the earlier applications you created (index.html, Component.js, manifest.json, view folder, App.view.xml, controller folder and App.controller.js)	
2. The **index.html** is nearly identical to our earlier exercise. It loads the UI5 boostrap and creates the intial shell UI structure. **Note**: You might have to adjust the sap-ui-boostrap URL if your system doesn't have the SAPUI5 central service installed **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_14	<!DOCTYPE html> <html> <head> <meta http-equiv="X-UA-Compatible" content="IE=edge" /> <meta charset="UTF-8"> <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" /> <title>XSJS Multiply Exercise</title> <link type="image/x-icon" href="/images/favicon.ico" rel="shortcut icon"> <link type="image/x-icon" href="/images/favicon.ico" rel="icon"> <!-- <script id="sap-ui-bootstrap" src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js" --> <script id="sap-ui-bootstrap" src="{{{sapui5_sb.url}}}/resources/sap-ui-core.js" data-sap-ui-theme="sap_belize_plus" data-sap-ui-xx-bindingSyntax="complex" data-sap-ui-compatVersion="edge" data-sap-ui-preload="async" data-sap-ui-language="en" data-sap-ui-resourceroots='{ "sap.shineNext.xsjsMultiply": "./", "view": "./view" }' data-sap-ui-libs="sap.m,sap.ui.comp,sap.ui.core,sap.ui.layout,sap.ui.unified"> </script> <script type="text/javascript" src="../common/startup.js"></script> <script> localShellStartup("sap.shineNext.xsjsMultiply"); </script> </head>

	<body class="sapUiBody" role="application"> </body> </html>
3. The **Component.js** is also very similiar, although simplier, than our eariler exericse. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_15	/*eslint no-console: 0, no-unused-vars: 0, no-use-before-define: 0, no-redeclare: 0*/ sap.ui.define(["sap/ui/core/UIComponent"], function(UIComponent) { "use strict"; return UIComponent.extend("sap.shineNext.xsjsMultiply.Component", { metadata: { manifest: "json" }, init: function(){ jQuery.sap.require("sap.m.MessageBox"); jQuery.sap.require("sap.m.MessageToast"); sap.ui.core.UIComponent.prototype.init.apply(this, arguments); }, destroy: function() { // call the base component's destroy function UIComponent.prototype.destroy.apply(this, arguments); } }); });
4. The **manifest.json** is also very similiar, yet simplified as we don't have an OData model service in this example. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_15a	{ "_version": "1.4.0", "start_url": "index.html", "sap.app": { "_version": "1.4.0", "type": "application", "resources": "resources.json", "id": "multiply", "applicationVersion": { "version": "${project.version}" } }, "sap.fiori": {

```
                "_version": "2.0.0",
                "registrationIds": [],
                "archeType": "transactional"
        },
        "sap.ui": {
                "_version": "1.44.0",
                "technology": "UI5",
                "icons": {
                        "icon": "/images/favicon.ico",
                        "favIcon": "/images/favicon.ico"
                },
                "deviceTypes": {
                        "desktop": true,
                        "tablet": true,
                        "phone": true
                },
                "supportedThemes": [
                        "sap_hcb",
                        "sap_bluecrystal",
                        "sap_belize",
                        "sap_belize_plus"
                ]
        },
        "sap.ui5": {
                "config": {
                        "sapFiori2Adaptation": true
                },
                "rootView": {
                        "viewName":
"sap.shineNext.xsjsMultiply.view.App",
                        "type": "XML",
                        "id": "app"
                },
                "dependencies": {
                        "minUI5Version": "1.44.0",
                        "libs": {
                                "sap.ui.core": {
                                        "minVersion": "1.44.0"
                                },
                                "sap.ui.comp": {
                                        "minVersion": "1.44.0"
                                },
                                "sap.m": {
                                        "minVersion": "1.44.0"
                                },
                                "sap.ui.layout": {
                                        "minVersion": "1.44.0"
                                }
                        }
                },
                "contentDensities": {
                        "compact": true,
                        "cozy": true
                },
```

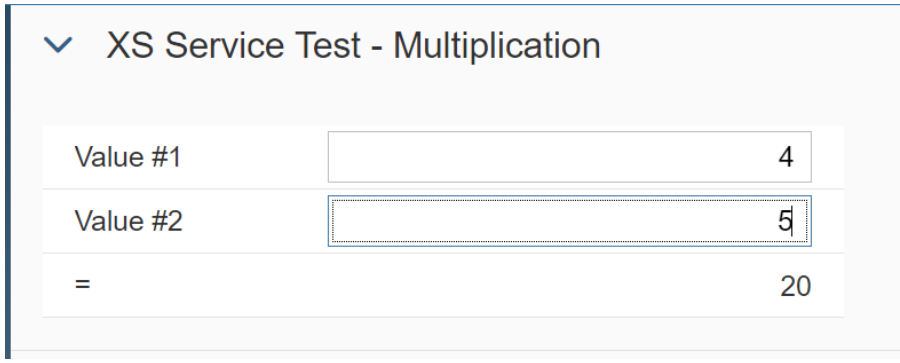
```
                "handleValidation": true

        }
}
```

5. The complete View Implementation for **App.view.xml** is provided for you as a template. It has two input fields for the numeric values which are multiplied. We will then call the service on the server when the user changes the value in either field.

   Note: This could be accomplished by simple math on the client side. We are using this very simple example in order to focus on the techniques for creating and calling the XSJS services without needing to get into too much detail.

   The template can be accessed at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_16




```
<core:View
    controllerName="sap.shineNext.xsjsMultiply.controller.App"
    xmlns="sap.m"
    xmlns:mvc="sap.ui.core.mvc"
    xmlns:core="sap.ui.core" >
  <Panel
      headerText="XS Service Test - Multiplication"
      expandable="true"
      expanded="true">
      <List width = "400px">
        <InputListItem label="Value #1">
          <Input id="val1" value="{/val1}" liveChange="onLiveChange" />
        </InputListItem>
        <InputListItem label="Value #2">
          <Input id="val2" value="{/val2}" liveChange="onLiveChange" />
        </InputListItem>
        <InputListItem label="=">
          <Text id="result" value="{/result}" />
        </InputListItem>
      </List>
    </Panel>
</core:View>
```

6. In **App.controller.js** add the onInit function with code to create local JSON model. You can also set the Compact design.

```
sap.ui.define(["sap/ui/core/mvc/Controller"], function(Controller) {
    "use strict";

    return Controller.extend("sap.shineNext.xsjsMultiply.controller.App", {

        onInit: function() {
            var model = new sap.ui.model.json.JSONModel({});
            this.getView().setModel(model);
            this.getView().addStyleClass("sapUiSizeCompact"); //
make everything inside this View appear in Compact mode
},
```

7. In the controller add an event handler called onLiveChange which has two parameters – oEvent and oVal. This is the event which is triggered every time the value is changed in either input field.

   Using jQuery.ajax call the URL /workshop/solution/services/multiply.xsjs?cmd=multiply adding on the num1 value from the oEvent and num2 value from the oVal input parameter.

   On Success of the AJAX call, call a controller event named onCompleteMultiply. If there is a failure call a controller event named onErrorCall.

   Note: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_17

```
        onLiveChange: function(oEvent) {
            var view = this.getView();
            var result = view.getModel().getData();
            var controller = this.getView().getController();
            var valSend;
            if (oEvent.getParameters().id === "comp---app--val1")
{
                valSend = result.val2;
            } else {
                valSend = result.val1;
            }
            if (valSend === undefined) {
                valSend = 0;
            }
            var aUrl =
"/sap/hana/democontent/epm/services/multiply.xsjs?cmd=multiply" +
                "&num1=" +
escape(oEvent.getParameters().newValue) +
                "&num2=" + escape(valSend);
            jQuery.ajax({
                url: aUrl,
                method: "GET",
                dataType: "json",
                success: controller.onCompleteMultiply,
                error: controller.onErrorCall
            });
},
```

8. In the onCompleteMultiply, accept the response object as an input parameter called myTxt. This will field will contain the result of the multiplication in clear text.

   Use the

```
        onCompleteMultiply: function(myTxt) {
            var oResult = sap.ui.getCore().byId("comp---app--
result");
            if (myTxt === undefined) {
                oResult.setText(0);
            } else {
```

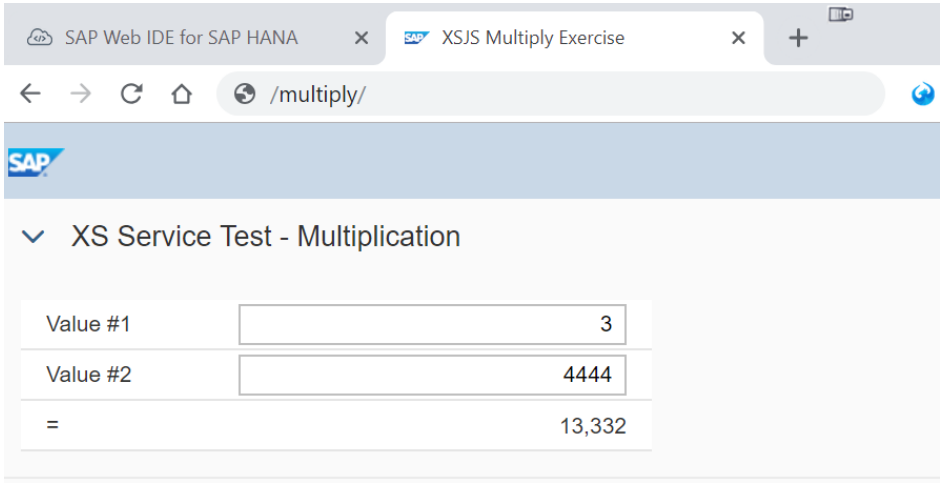
sap.ui.core.format.NumberFormat to format the output as an interger and set the value back into the oResult textView. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_18	jQuery.sap.require("sap.ui.core.format.NumberFormat"); var oNumberFormat = sap.ui.core.format.NumberFormat.getIntegerInstance({ maxFractionDigits: 12, minFractionDigits: 0, groupingEnabled: true }); oResult.setText(oNumberFormat.format(myTxt)); } },
9. In the onErrorCall, produce an error dialog (sap.ui.commons.MessageBox.show) and output the jqXHR.responseText as the details of this error. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_19	onErrorCall: function(oError) { if (oError.statusCode === 500 || oError.statusCode === 400 || oError.statusCode === "500" || oError.statusCode === "400") { var errorRes = JSON.parse(oError.responseText); if (!errorRes.error.innererror) { sap.m.MessageBox.alert(errorRes.error.message.value); } else { if (!errorRes.error.innererror.message) { sap.m.MessageBox.alert(errorRes.error.innererror.toString()); } else { sap.m.MessageBox.alert(errorRes.error.innererror.message); } } return; } else { sap.m.MessageBox.alert(oError.response.statusText); return; } } });});

10. Save your project.

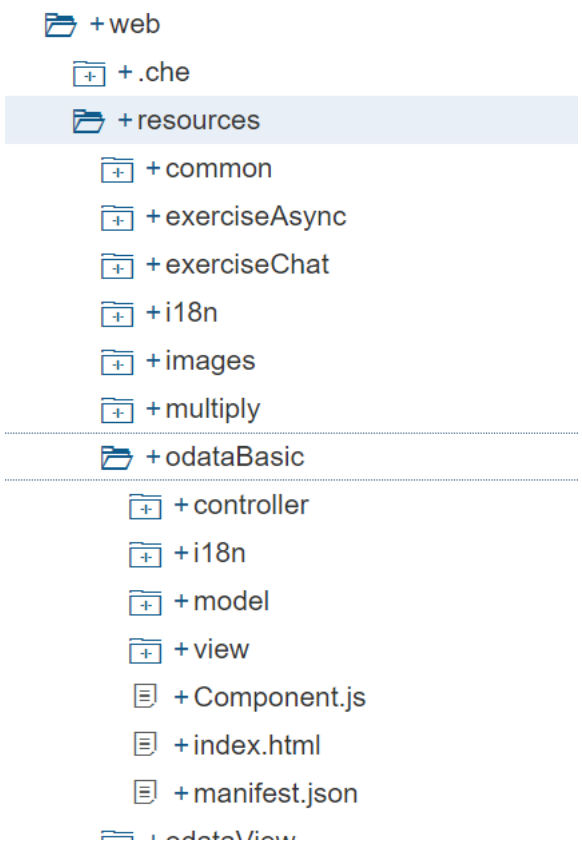
    Test your application in a web browser by running the **web** module.

    The URL would be /multiply/.

    Since we are using the live change event on the input fields, you only need to start typing into either field to trigger the call to the server and the calculation of the results



Exercise 5.5: Consume a Basic OData V2 Service within SAPUI5 binding the service to a Table

Explanation	Screenshot
1. We've prepared a skeleton applciation for you with several sections that need to be completed marked To-Do. **Import** this this template into web/resources folder from the following URL: https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex4/odataBasic.zip	

2. In the **manifest.json** file we initialize the application and set the base UI elements for the page.

   You should notice that there are commented **To-Do** sections in this code. These are the most important parts and will be completed in the next few steps.

manifest.json

```
{
  "_version": "1.4.0",
  "start_url": "index.html",
  "sap.app": {
    "_version": "1.4.0",
    "type": "application",
    "resources": "resources.json",
    "i18n": "i18n/i18n.properties",
    "id": "odataBasic",
    "title": "{{appTitle}}",
    "description": "{{appDescription}}",
    "applicationVersion": {
        "version": "${project.version}"
    },
    //To-Do data source

  },
```

3. In the first To-Do location is in the manifest.json file, you should add the code to create a data source named **bpService**. Use the provided service **/xsodata/businessPartners.xsodata/**.

manifest.json

```
{
  "_version": "1.4.0",
  "start_url": "index.html",
  "sap.app": {
    "_version": "1.4.0",
    "type": "application",
    "resources": "resources.json",
    "i18n": "i18n/i18n.properties",
    "id": "odataBasic",
    "title": "{{appTitle}}",
    "description": "{{appDescription}}",
    "applicationVersion": {
        "version": "${project.version}"
    },
    //To-Do data source

  },
```

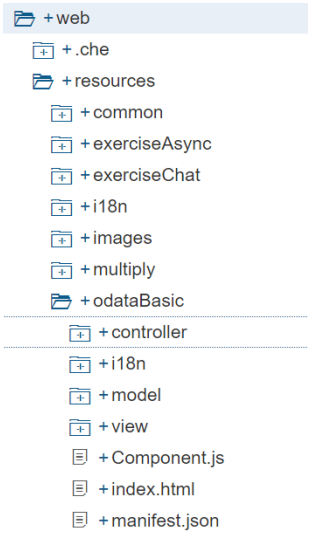
```
"dataSources": {
"bpService": {
  "uri": "/xsodata/businessPartners.xsodata/",
  "type": "OData",
  "settings": {
    "odataVersion": "2.0"
  }
}
}
```

	},
4. The second To-Do location is later in the same manifest.json file. Here you need to create the model named **bpModel** that connects the previous step's data source of type **sap.ui.model.odata.V2.ODataModel**.	*manifest.json 72 } 73 } 74 }, 75 "contentDensities": { 76 "compact": true, 77 "cozy": true 78 }, 79 "handleValidation": true, 80 "models": { 81 "": { 82 "type": "sap.ui.model.json.JSONModel", 83 "settings": { 84 "defaultBindingMode": "TwoWay" 85 } 86 }, 87 //To-Do bpModel 88 89 "config": { 90 "type": "sap.ui.model.json.JSONModel" 91 }, 92 "i18n": { "bpModel": { "dataSource": "bpService", "type": "sap.ui.model.odata.v2.ODataModel", "preload": true, "settings": { "useBatch": false, "json": true, "defaultBindingMode": "TwoWay", "defaultUpdateMethod": "PUT" } },
5. In the third To-Do location in the **App.controller.js** file, you should set the model named bpModel to the table control named **oTable**. Bind the table to the entity **BusinessPartners**. Then tell the SmartTable which columns to display	return BaseController.extend("opensap.odataBasic.controller.App", { onInit: function() { this.getView().addStyleClass("sapUiSizeCompact"); // make everything inside this View appear in Compact mode var oConfig = this.getOwnerComponent().getModel("config"); var userName = oConfig.getProperty("/UserName"); var bpModel = this.getOwnerComponent().getModel("bpModel"); var oTable = this.getView().byId("bpTable"); //To-Do Bind Model to table }, oTable.setModel(bpModel); oTable.setEntitySet("BusinessPartners"); oTable.setInitiallyVisibleFields("PARTNERID,COMPANYNAME,PARTNERROLE");

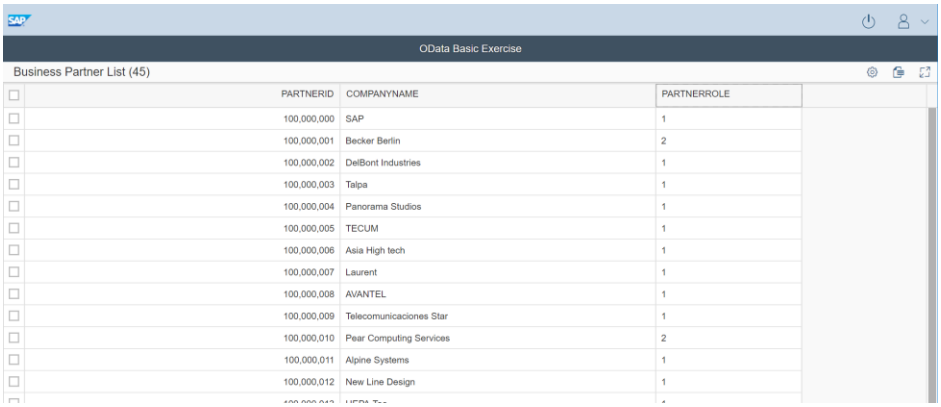
6. That's all that is necessary to connect the SmartTable UI element to the OData service.

   We get built in table sorting and filtering, as well as server side scrolling via the various built-in parameters of the OData service framework.

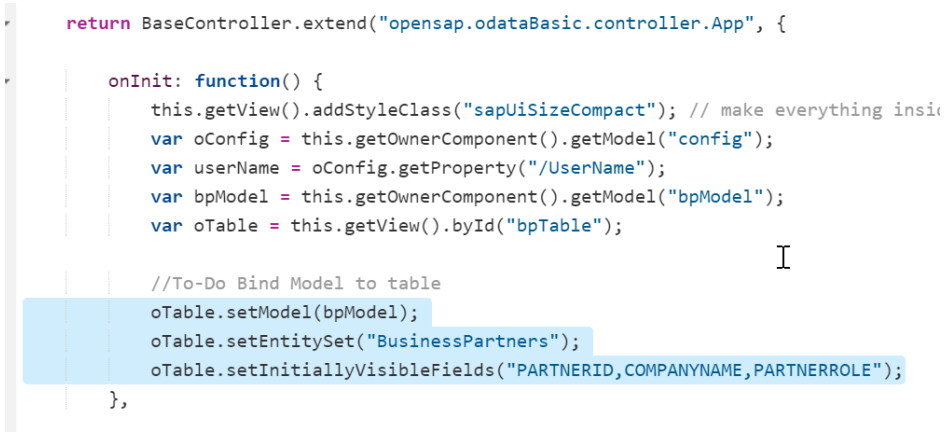
   Save your files, run the **web** module.



7. Adjust the run URL to **/odataBasic/**. We hardcode the columns, however. It would be better to add them dynamically via the metadata of the OData servcie – which is exactly what we will do in the next exercise.



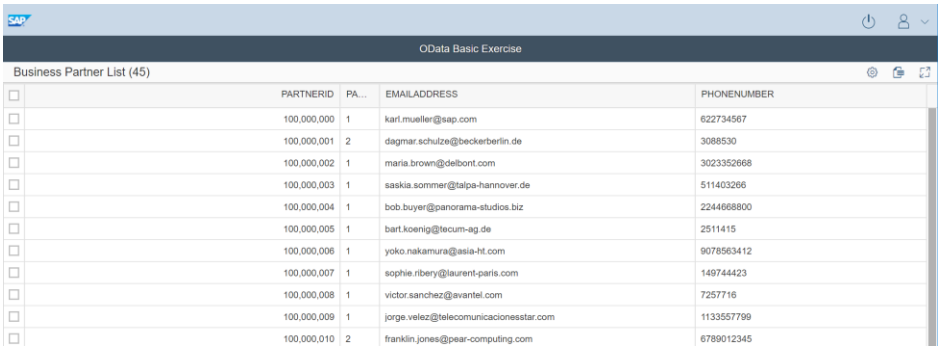
Exercise 5.6: Use oData V2 Metadata to dynamically create the columns.

Explanation	Screenshot
1. In the previous part of this exercise we hard coded all the table column definitions in the template. However, oData services expose all their meta data and we can use this feature to build the columns dynamically. Return to your **App.controller.js** file. Delete the three lines after the To-Do comment you filled in for the last exercise.	 `return BaseController.extend("opensap.odataBasic.controller.App", {` `onInit: function() {` `this.getView().addStyleClass("sapUiSizeCompact"); // make everything insi` `var oConfig = this.getOwnerComponent().getModel("config");` `var userName = oConfig.getProperty("/UserName");` `var bpModel = this.getOwnerComponent().getModel("bpModel");` `var oTable = this.getView().byId("bpTable");` `//To-Do Bind Model to table` `oTable.setModel(bpModel);` `oTable.setEntitySet("BusinessPartners");` `oTable.setInitiallyVisibleFields("PARTNERID,COMPANYNAME,PARTNERROLE");` `},`
2. You can create a connection to the metadata object via the function **getServiceMetadata** of your model object. Inside this meta data you will find the columns of the service at **dataServices.schema[0].entityType[0].property.** Loop over this collection and create a column for each property.name in the service dynamically. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_23 Save and re-run your **web** module.	`function fnLoadMetadata() {` `try {` `oTable.setModel(bpModel);` `oTable.setEntitySet("BusinessPartners");` `var oMeta = bpModel.getServiceMetadata();` `var headerFields = "";` `for (var i = 0; i < oMeta.dataServices.schema[0].entityType[0].property.length; i++) {` `var property = oMeta.dataServices.schema[0].entityType[0].property[i];` `headerFields += property.name + ",";` `}` `oTable.setInitiallyVisibleFields(headerFields);` `} catch (e) {` `console.log(e.toString());` `}` `}` `bpModel.attachMetadataLoaded(bpModel, function() {` `fnLoadMetadata();` `});` `fnLoadMetadata();`

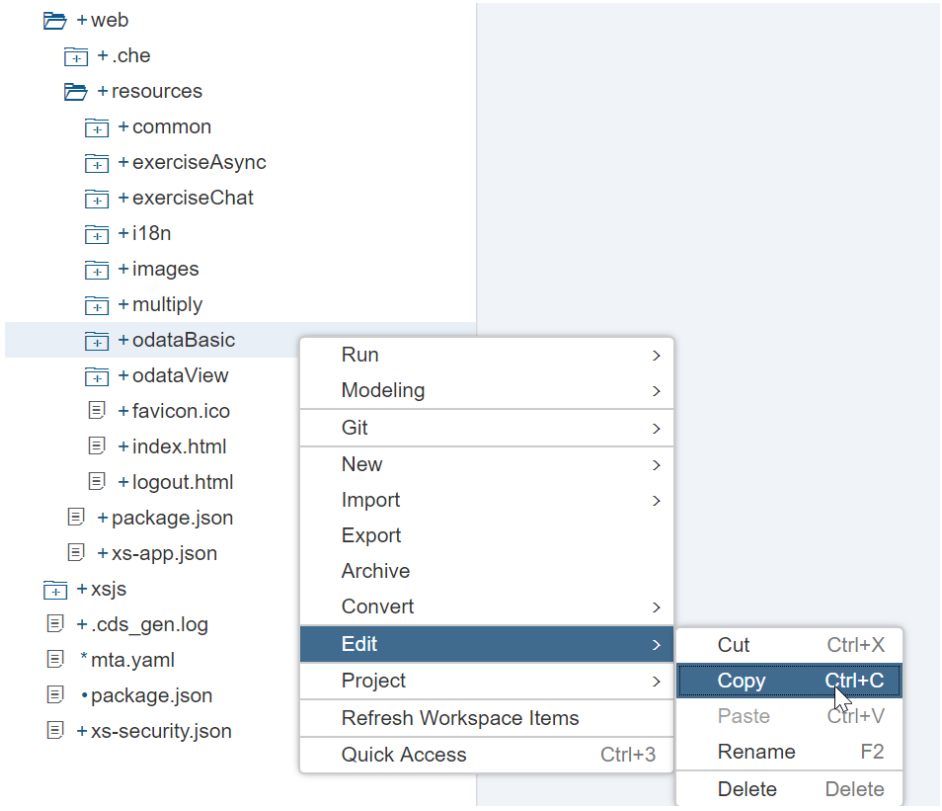
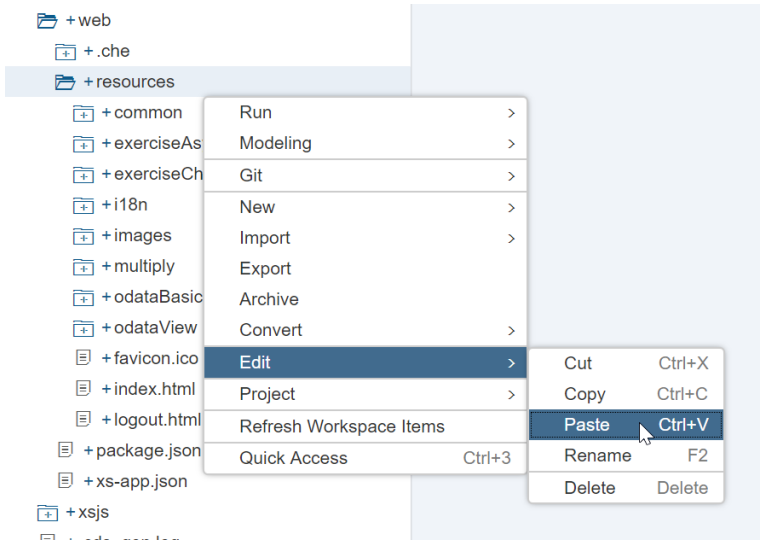
3) Test your application in a web browser using the Run option.

   The URL would be **/odataBasic**.

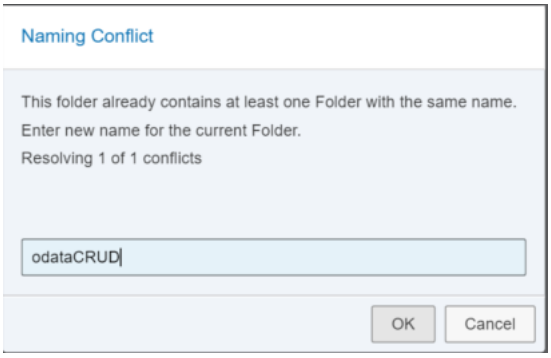
   Notice that you now have all the columns of the service; not just the few you had before.



Exercise 5.7: Consume an OData V2 Service with Create Option

Explanation	Screenshot
1. We will begin by making a copy of the previous exercise	
2. Highlight the **resources** folder and right mouse click. Choose **Paste**.	

3. In the copy folder dialog, give the new Name as **odataCRUD**



4. Much of the startup logic will be the same. In the **mainifest.json** file change the service URL to: `/xsodata/user.xsodata/`

We must also setup the model binding mode.

Note: if you need help writing this code please refer to the solution at:

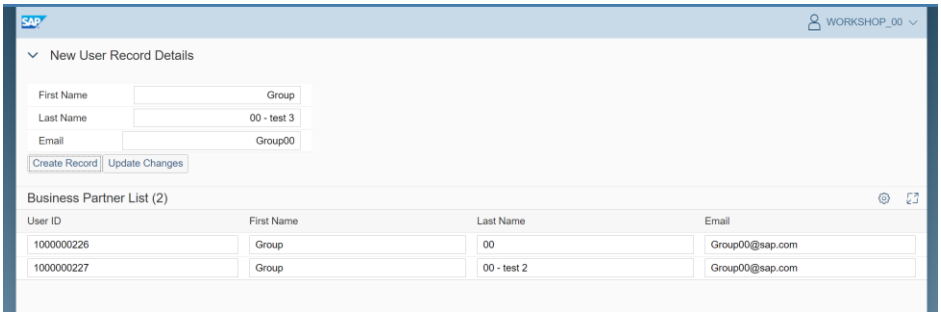
https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_24

manifest.json ×

```
{
  "_version": "1.4.0",
  "start_url": "index.html",
  "sap.app": {
    "_version": "1.4.0",
    "type": "application",
    "resources": "resources.json",
    "i18n": "i18n/i18n.properties",
    "id": "odataTest",
    "title": "{{appTitle}}",
    "description": "{{appDescription}}",
    "applicationVersion": {
        "version": "${project.version}"
    },
    "dataSources": {
      "userService": {
        "uri": "/xsodata/user.xsodata/",
        "type": "OData",
        "settings": {
          "odataVersion": "2.0"
        }
      }
    }
```

```
  "models": {
      "": {
          "type": "sap.ui.model.json.JSONModel",
          "settings": {
              "defaultBindingMode": "TwoWay"
          }
      },
    "userModel": {
          "dataSource": "userService",
          "type": "sap.ui.model.odata.v2.ODataModel",
          "preload": true,
          "settings": {
              "useBatch": false,
              "json": true,
              "defaultBindingMode": "TwoWay",
              "defaultUpdateMethod": "PUT"
          }
      },
      "config": {
          "type": "sap.ui.model.json.JSONModel"
      },
      "i18n": {
          "type": "sap.ui.model.resource.ResourceModel",
          "settings": {
              "bundleUrl": "./i18n/i18n.properties"
          }
      }
    }

  }
}
```

5. The complete View Implementation for **App.view.xml** is provided for you as a template. It has a table control built from the odata service **/xsodata/user.xsodata/** - all of which is very similar to the earlier exercise. In addition, this view has input fields for creating a new record. It also has the ability to update records in the table control; not just display them. The **template** can be accessed at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_25	
6. In this exercise we will focus on the implement of the event handlers – which is all done in the **App.controller.js** file. You have the App.controller.js implementation from the previous exercise whose contents is now largely about to change,	

7. We need to add to event handlers,, **onInit**, **callUserService** (which performs the creation of new records) and **callUserUpdate** (which updates records from the table) and onErrorCall (and error handler). Remove the old function definitions and insert the new empty functions into **App.controller.js** as shown.



8. For the **onInit** we need to connect our model to the SmartTable control.

```
this.getView().addStyleClass("sapUiSizeCompact"); // make everything inside this View appear in Compact mode
var oConfig = this.getOwnerComponent().getModel("config");
var userName = oConfig.getProperty("/UserName");
var userModel = this.getOwnerComponent().getModel("userModel");
var oTable = this.getView().byId("userTable");
oTable.setModel(userModel);
```

9. For **callUserService**, you first need to get access to the model object.

   Next you need to create a JSON object with the service fields (PERS_NO, FIRSTNAME, LASTNAME, and E_MAIL). PERS_NO can get a hardcoded value of “0000000000”. The other fields should be read from the screen with the bound JSON model

   Finally you need to set a custom header of content-type with the value of **application/json;charset=utf-8** in the model. Then

```
callUserService: function() {
var oModel = this.getOwnerComponent().getModel("userModel");
var result = this.getView().getModel().getData();
var oEntry = {};
oEntry.UserId = "0000000000";
oEntry.FirstName = result.FirstName;
oEntry.LastName = result.LastName;
oEntry.Email = result.Email;

oModel.setHeaders({
    "content-type": "application/json;charset=utf-8"
});
var mParams = {};
mParams.success = function() {
    sap.m.MessageToast.show("Create successful");
};
mParams.error = this.onErrorCall;
oModel.create("/Users", oEntry, mParams);
```

you can call the oModel.create function for the entity **/Users.** **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_26	},
10. The implementation of **callUserUpdate** is actually much simplier. We are using two-way model binding therefore we need only ask the model to submit any pending changes and capture the response status events. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_27	callUserUpdate: function() { var oModel = this.getOwnerComponent().getModel("userModel"); oModel.setHeaders({ "content-type": "application/json;charset=utf-8" }); var mParams = {}; mParams.error = function() { sap.m.MessageToast.show("Update failed"); }; mParams.success = function() { sap.m.MessageToast.show("Update successful"); }; oModel.submitChanges(mParams); },
11. In the onErrorCall we want to be able to parse the error body to pull out the detailed error message. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_28	onErrorCall: function(oError) { if (oError.statusCode === 500 || oError.statusCode === 400 || oError.statusCode === "500" || oError.statusCode === "400") { var errorRes = JSON.parse(oError.responseText); if (!errorRes.error.innererror) { sap.m.MessageBox.alert(errorRes.error.message.value); } else { if (!errorRes.error.innererror.message) { sap.m.MessageBox.alert(errorRes.error.innererror.toString()); } else { sap.m.MessageBox.alert(errorRes.error.innererror.message);

```
                    }
                }
                return;
            } else {

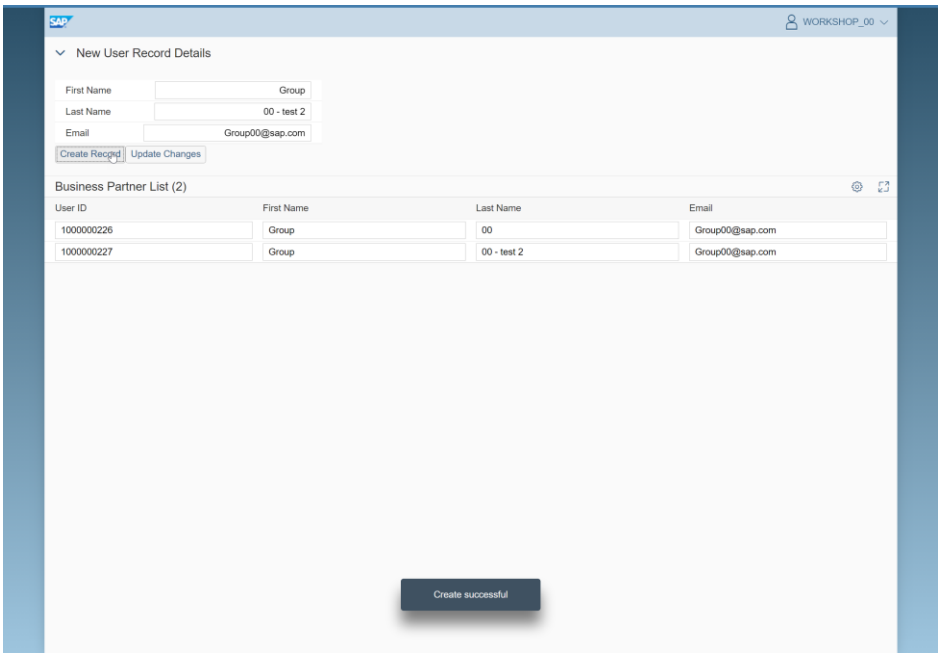
sap.m.MessageBox.alert(oError.response.statusText);
                return;
            }
        }
```

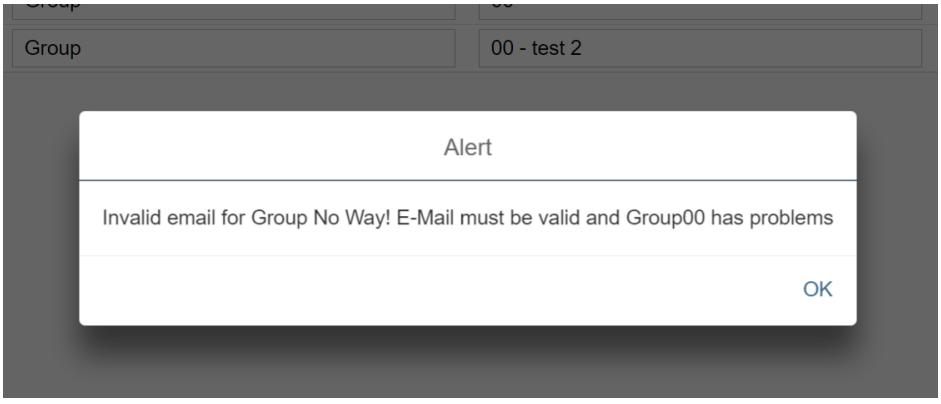
12. Save your files and re-run the **web** module.

13. Test your application in a web browser using the Run option.

    The URL would be **/odataCRUD/**.

    Try creating a new record. Also try creating a record with an invalid email address.





Exercise 5.8: OData V2 Batch Operation

Explanation	Screenshot
1. Often you might want to send more than one operation to the server at a time. The Odata $batch operation allows this. In this exercise we will extend the odataCRUD project from the previous exercise to allow the creation of multiple records at a time. First we need to create an XML fragment for our batch dialog we will display. Create a new file named **batchDialog.fragment.xml** in the same **odataCRUD/view** folder. Add the following code to this file. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_29	`<core:FragmentDefinition` `  xmlns="sap.m"` `  xmlns:core="sap.ui.core">` `  <Dialog` `  title="Create Users with batch request"` `  class="sapUiPopupWithPadding" >` `  <content>` `  </content>` `  <beginButton>` `  <Button text="Submit"  press="onSubmitBatch" />` `  </beginButton>` `  <endButton>` `  <Button text="Cancel"  press="onDialogCloseButton" />` `  </endButton>` `  </Dialog>` `</core:FragmentDefinition>`

2. We also need to add a new button in the **App.view.xml** that will trigger the batch creation dialog box.

   Add the following code to create a button for this.

batchDialog.fragment.xml × *App.view.xml ×

```
<core:View xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" xmlns:u="sap.ui.unified" xmlns:cc
    controllerName="opensap.odataBasic.controller.App">
    <u:Shell id="myShell" icon="/images/sap_18.png">
        <u:user>
            <u:ShellHeadUserItem image="sap-icon://person-placeholder" username="{conf
        </u:user>
        <u:content>
            <ScrollContainer height="100%" width="100%" horizontal="true" vertical="tr
                <Panel headerText="New User Record Details" expandable="true" expanded
                    <List width="400px">
                        <InputListItem label="{i18n>FName}">
                            <Input id="fName" value="{/FirstName}"/>
                        </InputListItem>
                        <InputListItem label="{i18n>LName}">
                            <Input id="lName" value="{/LastName}"/>
                        </InputListItem>
                        <InputListItem label="{i18n>EMail}">
                            <Input id="email" value="{/Email}"/>
                        </InputListItem>
                    </List>
                    <Button text="{i18n>B3}" press="callUserService"/>
                    <Button text="{i18n>B4}" press="callUserUpdate"/>
                    <Button text="{i18n>B5}" press="onBatchDialogPress"/>
                </Panel>
                <VBox fitContainer="true">
```

3. Now in the **App.controller.js** you need to implement the event handlers for the dialog creation, adding of new row, deleting rows, and calling the service in Batch mode. Here we use two feature of the UI5 OData model object.

   First you use the **addBatchChangeOperations** to collect the individual records and the actions you want to perform on the records.

   Next you use **submitBatch** function to send the request to the server. The OData model takes care of all the technical details of building the multi-part MIME request which will contain all the inner request objects for each record you want to create.

   Note: if you need help writing this code please refer to the solution at:

```
onBatchDialogPress: function() {
        var view = this.getView();
        view._bDialog = sap.ui.xmlfragment(
            "opensap.odataBasic.view.batchDialog", this //
associate controller with the fragment
        );
        view._bDialog.addStyleClass("sapUiSizeCompact");
        view.addDependent(this._bDialog);

    view._bDialog.addContent(view.getController().getItem(true));
        view._bDialog.open();
    },

    onDialogCloseButton: function() {
        this.getView()._bDialog.close();
    },

    getItem: function(isFirstRow) {
        var view = this.getView();
        var addIcon = new sap.ui.core.Icon({
            src: "sap-icon://add",
            color: "#006400",
            size: "1.5rem",
            press: function() {

    view._bDialog.addContent(view.getController().getItem(false));
            }
        });

        var deleteIcon = new sap.ui.core.Icon({
            src: "sap-icon://delete",
            color: "#49311c",
            size: "1.5rem",
```

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_30

```
                                press: function(oEvent) {

        view._bDialog.removeContent(oEvent.oSource.oParent.sId);
                                }
                        });

                        var icon;
                        if (isFirstRow) {
                                icon = addIcon;
                        } else {
                                icon = deleteIcon;
                        }
                        icon.addStyleClass("iconPadding");

                        var firstNameTxt = new sap.m.Label({
                                text: "First Name"
                        });
                        firstNameTxt.addStyleClass("alignText");
                        var firstNameInput = new sap.m.Input({});

                        var lastNameTxt = new sap.m.Label({
                                text: "Last Name"
                        });
                        lastNameTxt.addStyleClass("alignText");
                        var lastNameInput = new sap.m.Input({});

                        var emailTxt = new sap.m.Label({
                                text: "Email"
                        });
                        emailTxt.addStyleClass("alignText");
                        var emailInput = new sap.m.Input({});

                        return new sap.m.FlexBox({
                                // enableFlexBox: true,
                                //    fitContainer: true,
                                //  justifyContent:
sap.m.FlexJustifyContent.SpaceBetween,
                                items: [firstNameTxt,
                                        firstNameInput,
                                        lastNameTxt,
                                        lastNameInput,
                                        emailTxt,
                                        emailInput,
                                        icon
                                ]
                        });
                },

                onSubmitBatch: function() {
                        var view = this.getView();
                        var content = view._bDialog.getContent();
                        var newUserList = [];
                        for (var i = 0; i < content.length; i++) {
                                var user = {};
```

	user.UserId = "0000000000"; user.FirstName = content[i].getItems()[1].getValue(); user.LastName = content[i].getItems()[3].getValue(); user.Email = content[i].getItems()[5].getValue(); // user.ZMYNEW1 = ""; newUserList.push(user); } //create an array of batch changes and save var oParams = {}; oParams.json = true; oParams.defaultUpdateMethod = "PUT"; oParams.useBatch = true; var batchModel = new sap.ui.model.odata.v2.ODataModel("/xsodata/user.xsodata/", oParams); //var batchChanges = []; var mParams = {}; mParams.groupId = "1001"; mParams.success = function() { sap.m.MessageToast.show("Create successful"); }; mParams.error = this.onErrorCall; for (var k = 0; k < newUserList.length; k++) { batchModel.create("/Users", newUserList[k], mParams); } }
4. Save your files.	

5. Not critical, but if you want the field alignment in the batch dialog to look nice; you can also return to the **index.html** file and add several style elements near the beginning of the file.

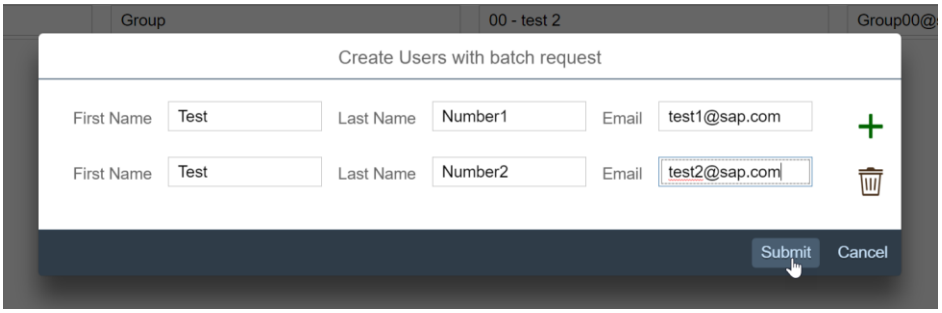
App.view.xml | App.controller.js | *index.html | batchDialog.frag

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .mobile-list-item {
            padding: 0.5em;
        }
        .paddingAll {
            padding: 5px;
        }
        .alignText {
            line-height: 2.5;
            margin-right: 1em;
            margin-left: 1em;
        }
        .iconPadding {
            margin-left: 2.5rem;
            line-height: 2 !important;
        }
    </style>
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
```

6. Test your application in a web browser using the Run option.

   The URL would be **/odataCRUD/**.

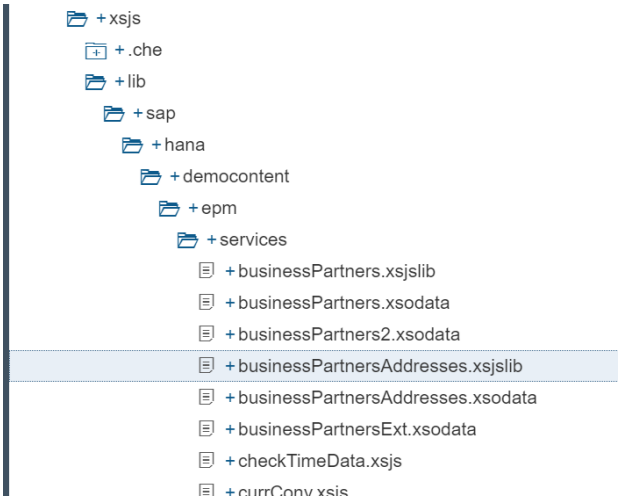
   Try using the new Create users with batch request button and creating two records.



Exercise 5.9: OData V2 Deep Insert (Content-ID and Links)

Explanation	Screenshot
1. As we saw in the previous exercise, it is possible to perform multiple operations in one request. This is particularly useful in situations where you are inserting objects with a parent/child relationship. But what happens when you create both objects via generated keys on the server side. In one batch how can you also create the relationship between the objects as well? This exercise will look at how to perform just such a 'deep insert' in one batch and will rely upon many of the techniques covered in previous exercises.	
2. We already have the XSODATA service named **businessPartnersAddresses.xsodata** in our **xsjs** module, **lib/sap/hana/democontent/epm/services** folder. In this service we want to allow the creation of records in both the MD.BusinessPartner and the MD.Addresses tables. We also create an association between the two tables. Use XSJS exit handlers for both tables and the association.	businessPartnersAddresses.xs... `service {` `"MD.BusinessPartner"` `as "BusinessPartners"` `navigates ("ToAddresses" as "AddRef")` `create events(before "sap.hana.democontent.epm.services:businessPartnersAddresses.xsjslib::bp_create_before_exit");` `"MD.Addresses"` `as "Addresses"` `without ("POINT")` `create events(before "sap.hana.democontent.epm.services:businessPartnersAddresses.xsjslib::address_create_before_exit");` `association "ToAddresses" principal "BusinessPartners"("ADDRESSES.ADDRESSID")` `multiplicity "1"` `dependent "Addresses"("ADDRESSID") multiplicity "1"` `over "MD.Addresses"` `principal("ADDRESSID") dependent ("ADDRESSID")` `update using "sap.hana.democontent.epm.services:businessPartnersAddresses.xsjslib::assocation_create_exit";` `}`

3. We also have the xsjslib named **businessPartnersAddresses** for the exit handlers of our service.



4. We will look at each of the exit handlers one at a time.

   The first function runs before the creation of the business partner record and allows us to use a sequence for the partner id and for defaulting in some values.

```
$.import("sap.hana.democontent.epm.services", "session");
var SESSIONINFO = $.sap.hana.democontent.epm.services.session;

/**
@param {connection} Connection - The SQL connection used in the OData request
@param {beforeTableName} String - The name of a temporary table with the single entry before the operation (UPDATE and DELETE events only)
@param {afterTableName} String -The name of a temporary table with the single entry after the operation (CREATE and UPDATE events only)
 */

function bp_create_before_exit(param) {

	var	after = param.afterTableName;
	var pStmt;
	try {

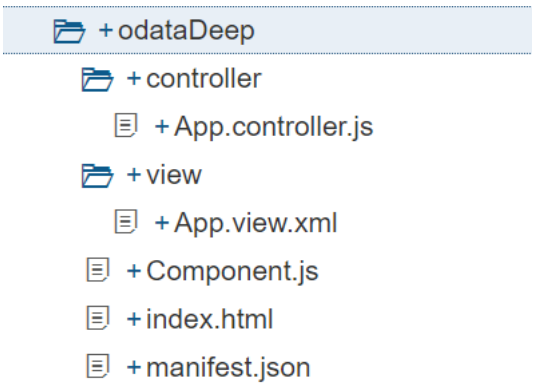
		pStmt = param.connection.prepareStatement('select "businessPartnerId".NEXTVAL from DUMMY');
		var rs = pStmt.executeQuery();
		var PartnerId = '';
		while (rs.next()) {
			PartnerId = rs.getInteger(1);
		}
		pStmt.close();

		pStmt = param.connection.prepareStatement('update "' + after
				+ '" set PARTNERID = ?,' +
				  '  PARTNERROLE = ?, ' +
				  '  "HISTORY.CREATEDBY.EMPLOYEEID" = ?,' +
				  '  "HISTORY.CHANGEDBY.EMPLOYEEID" = ?,' +
				  '  "HISTORY.CREATEDAT" = now(),' +
				  '  "HISTORY.CHANGEDAT" = now(),' +
```

	' "CURRENCY" = ?'); pStmt.setInteger(1, PartnerId); pStmt.setString(2, '01'); pStmt.setInteger(3, 33); pStmt.setInteger(4, 33); pStmt.setString(5, 'EUR'); pStmt.execute(); pStmt.close(); } catch (e) { console.log(e.toString()); return; } }
5. The next part is very similar and handles the before creation for the address table.	function address_create_before_exit(param) { var after = param.afterTableName; var pStmt; try { pStmt = param.connection.prepareStatement('select "addressId".NEXTVAL from dummy'); var rs = pStmt.executeQuery(); var AddressId = ''; while (rs.next()) { AddressId = rs.getInterger(1); } pStmt.close(); pStmt = param.connection.prepareStatement('update "' + after + '" set "ADDRESSID" = ?,' + 'ADDRESSTYPE = ?,' + '"VALIDITY.STARTDATE" = TO_DATE(' + "'2000-01-01', 'YYYY-MM-DD')," + '"VALIDITY.ENDDATE" = TO_DATE(' + "'9999-12-31', 'YYYY-MM-DD')"); pStmt.setInteger(1, AddressId); pStmt.setString(2, '02'); pStmt.execute(); pStmt.close(); } catch (e) { console.log(e.toString()); return;

	} }
6. The final part is different because it handles the association between two entities. The input parameters pass in the data from both the principal and dependent entities. We link the two by updating the business partner table with the address id from the address table. Notice at this point we don't have any special processing for the batch. We simply assume that the records for both the address and the business partners will be passed in with their correct generated keys although the whole operation is happening in one transaction.	/** @param {connection} Connection - The SQL connection used in the OData request @param {principalTableName} String - The name of a temporary table with the entity type at the principal end of the association @param {dependentTableName} String -The name of a temporary table with the dependent entity type */ function assocation_create_exit(param){ var princ = param.principalTableName; var dep = param.dependentTableName; var pStmt = param.connection.prepareStatement('select * from "' + princ + '"'); var Principal = SESSIONINFO.recordSetToJSON(pStmt.executeQuery(), 'Details'); pStmt.close(); var pStmt = param.connection.prepareStatement('select * from "' + dep + '"'); var Dependent = SESSIONINFO.recordSetToJSON(pStmt.executeQuery(), 'Details'); pStmt.close(); $.trace.debug(JSON.stringify(Principal)); $.trace.debug(JSON.stringify(Dependent)); var pStmt = param.connection.prepareStatement('update "MD.BusinessPartner" ' + ' SET "ADDRESSES.ADDRESSID" = ? WHERE "PARTNERID" = ? '); pStmt.setInteger(1, Dependent.Details[0].ADDRESSID); pStmt.setInteger(2, Principal.Details[0].PARTNERID); pStmt.execute(); pStmt.close(); }

7. Next we need to create the user interface. Using what you have learned create a new SAPUI5 application named **odataDeep** with a similar structure to the earlier applications you created (index.html, Component.js, manifest.json, view folder, App.view.xml, controller folder, and App.controller.xml)



8. Open **indext.html**. For the use of the $batch with content-ids we can't use the SAPUI5 odata object. We must build the complete request object ourselves. This also means that we will be calling the service with jQuery.ajax and must handle the CSRF token request ourselves.

   For this we will refer to the **common/csrf.js** we created near the beginning of this exercise.

   Note: if you need help writing this code please refer to the solution at:

   https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_31

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6      <meta charset="UTF-8">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, us
8      <title>OData Deep Insert Exercise</title>
9      <link type="image/x-icon" href="/images/favicon.ico" rel="shortcut icon">
10     <link type="image/x-icon" href="/images/favicon.ico" rel="icon">
11         <!-- <script id="sap-ui-bootstrap" src="https://sapui5.hana.ondemand.com/resources/sap-
12     <script id="sap-ui-bootstrap" src="{{{sapui5_sb.url}}}/resources/sap-ui-core.js"
13         data-sap-ui-theme="sap_belize_plus"
14         data-sap-ui-xx-bindingSyntax="complex"
15         data-sap-ui-compatVersion="edge"
16         data-sap-ui-preload="async"
17         data-sap-ui-language="en"
18             data-sap-ui-resourceroots='{
19                 "sap.shineNext.odataDeep": "./",
20                 "view": "./view" }'
21             data-sap-ui-libs="sap.m,sap.ui.comp,sap.ui.core,sap.ui.layout,sap.ui.unified">
22     </script>
23     <script type="text/javascript" src="/common/csrf.js" ></script>
24     <script type="text/javascript" src="../common/startup.js"></script>
25     <script>
26         localShellStartup("sap.shineNext.odataDeep");
27     </script>
28 </head>
29 <body class="sapUiBody" role="application">
30     <div id="content"></div>
31 </body>
32 </html>
```

9. In the view rendering, **App.view.xml**, we want to simply have three fields and a button.

   The three fields are for Company Name, City, and Email Address.

   Note: if you need help writing this code please refer to the solution at:

```
<mvc:View controllerName="sap.shineNext.odataDeep.controller.App"
xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" xmlns:core="sap.ui.core"
height="100%">
        <Page title="{i18n>appTitle}">
                <content>
                        <Panel headerText="New Business Partner Details"
expandable="true" expanded="true">
                                <List width="600px">
                                        <InputListItem label="Company
Name">
                                                <Input id="cName"
value="{/CompanyName}"/>
```

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_32	</InputListItem> <InputListItem label="City"> <Input id="city" value="{/City}"/> </InputListItem> <InputListItem label="Email"> <Input id="email" value="{/Email}"/> </InputListItem> </List> <Button text="Create Record" press="callCreateService"/> </Panel> </content> </Page> </mvc:View>
10. For the controller, **App.controller.js**, we have one event handler for the press of the button. However this event handler is very important because this is where we will build the complete OData $batch request. Unlike the earlier exercise where the SAPUI5 framework did all the work, here we must build the complete multiple part request. Notice that in the individual request headers we use the Content-ID option. This allows us to create a place holder to represent the keys of the object which won't be generated until server side. We can then use $1 and $2 in the association URL and body as placeholders for the actual values. The XSODATA framework on the server side will replace these values during processing. **Note**: if you need help writing this code please refer to the solution at:	/*eslint no-console: 0, no-unused-vars: 0, no-use-before-define: 0, no-redeclare: 0, no-undef: 0, no-sequences: 0, no-unused-expressions: 0, quotes: 0*/ //To use a javascript controller its name must end with .controller.js sap.ui.define(["sap/ui/core/mvc/Controller"], function(Controller) { "use strict"; return Controller.extend("sap.shineNext.odataDeep.controller.App", { onInit: function() { var model = new sap.ui.model.json.JSONModel({}); this.getView().setModel(model); this.getView().addStyleClass("sapUiSizeCompact"); // make everything inside this View appear in Compact mode }, callCreateService: function() { var result = this.getView().getModel().getData(); var oBusinessPartner = {}; oBusinessPartner.PARTNERID = "0000000000"; oBusinessPartner.EMAILADDRESS = result.Email; oBusinessPartner.COMPANYNAME = result.CompanyName; var oAddress = {}; oAddress.ADDRESSID = "0000000000"; oAddress.CITY = result.City; var oLink = {}; oLink.uri = "$2"; var xhr = new XMLHttpRequest(); xhr.open("POST", '/sap/hana/democontent/epm/services/businessPartnersAddresses.xsodata/$batch', true);

https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_33	var token = getCSRFToken(); xhr.setRequestHeader("X-CSRF-Token", token); xhr.setRequestHeader("Accept", 'application/json'); xhr.setRequestHeader("Content-Type", 'multipart/mixed;boundary=batch'); xhr.setRequestHeader("DataServiceVersion", '2.0'); xhr.setRequestHeader("MaxDataServiceVersion", '2.0'); var body = ''; body += '--batch' + '\r\n'; body += 'Content-Type:multipart/mixed;boundary=changeset' + '\r\n'; body += 'Content-Transfer-Encoding:binary' + '\r\n'; body += '\r\n'; body += '--changeset' + '\r\n'; body += 'Content-Type:application/http' + '\r\n'; body += 'Content-Transfer-Encoding:binary\r\n'; body += 'Content-ID: 1\r\n'; body += '\r\n'; body += 'POST BusinessPartners HTTP/1.1\r\n'; body += "Content-Type: application/json\r\n"; var jsonBP = JSON.stringify(oBusinessPartner); body += "Content-Length:" + jsonBP.length + '\r\n'; body += '\r\n'; body += jsonBP + '\r\n'; body += '--changeset' + '\r\n'; body += 'Content-Type:application/http' + '\r\n'; body += 'Content-Transfer-Encoding:binary\r\n'; body += 'Content-ID: 2\r\n'; body += '\r\n'; body += 'POST Addresses HTTP/1.1\r\n'; body += "Content-Type:application/json\r\n"; var jsonAdd = JSON.stringify(oAddress); body += "Content-Length:" + jsonAdd.length + '\r\n'; body += '\r\n'; body += jsonAdd + '\r\n'; body += '--changeset' + '\r\n'; body += 'Content-Type:application/http' + '\r\n'; body += 'Content-Transfer-Encoding:binary\r\n'; body += '\r\n'; body += 'PUT $1/$links/AddRef HTTP/1.1\r\n'; body += "Content-Type:application/json\r\n"; var jsonLink = JSON.stringify(oLink);

```
                        body += "Content-Length:" + jsonLink.length + '\r\n';
                        body += '\r\n';

                        body += jsonLink + '\r\n';

                        body += '--changeset' + '--\r\n';
                        body += '\r\n';

                        body += '--batch' + '--\r\n';

                        xhr.onload = function() {};
                        xhr.send(body);
                        sap.m.MessageToast.show("Business Partner
created");
                }
        });
});
```

11. Next we need our **manifest.json**, but its very basic as we have no OData model being used in this exercise.

    Note: if you need help writing this code please refer to the solution at:

    https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_33a

```
{
  "_version": "1.4.0",
  "start_url": "index.html",
  "sap.app": {
                "_version": "1.4.0",
                "type": "application",
                "resources": "resources.json",
                "id": "odataDeep",
                "applicationVersion": {
                        "version": "${project.version}"
                }
  },
  "sap.fiori": {
                "_version": "2.0.0",
                "registrationIds": [],
                "archeType": "transactional"
  },
  "sap.ui": {
                "_version": "1.44.0",
                "technology": "UI5",
                 "icons":  {
                        "icon": "/images/favicon.ico",
                        "favIcon": "/images/favicon.ico"
                 },
                "deviceTypes": {
                        "desktop": true,
                        "tablet": true,
                        "phone": true
                },
                "supportedThemes": [
                        "sap_hcb",
                        "sap_bluecrystal",
                        "sap_belize"
                ]
  },
```

	"sap.ui5": { "config": { "sapFiori2Adaptation": true }, "rootView": { "viewName": "sap.shineNext.odataDeep.view.App", "type": "XML", "id": "app" }, "dependencies": { "minUI5Version": "1.44.0", "libs": { "sap.ui.core": { "minVersion": "1.44.0" }, "sap.ui.comp": { "minVersion": "1.44.0" }, "sap.m": { "minVersion": "1.44.0" }, "sap.ui.layout": { "minVersion": "1.44.0" } } }, "contentDensities": { "compact": true, "cozy": true }, "handleValidation": true } }
12. Finally we need to create the **Component.js** file. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex4/ex4_34	/*eslint no-console: 0, no-unused-vars: 0, no-use-before-define: 0, no-redeclare: 0*/ sap.ui.define(["sap/ui/core/UIComponent"], function(UIComponent) { "use strict"; return UIComponent.extend("sap.shineNext.odataDeep.Component", { metadata: { manifest: "json" }, init: function(){ jQuery.sap.require("sap.m.MessageBox");

```
                        jQuery.sap.require("sap.m.MessageToast");


                        sap.ui.core.UIComponent.prototype.init.apply(
                                this, arguments);
                },

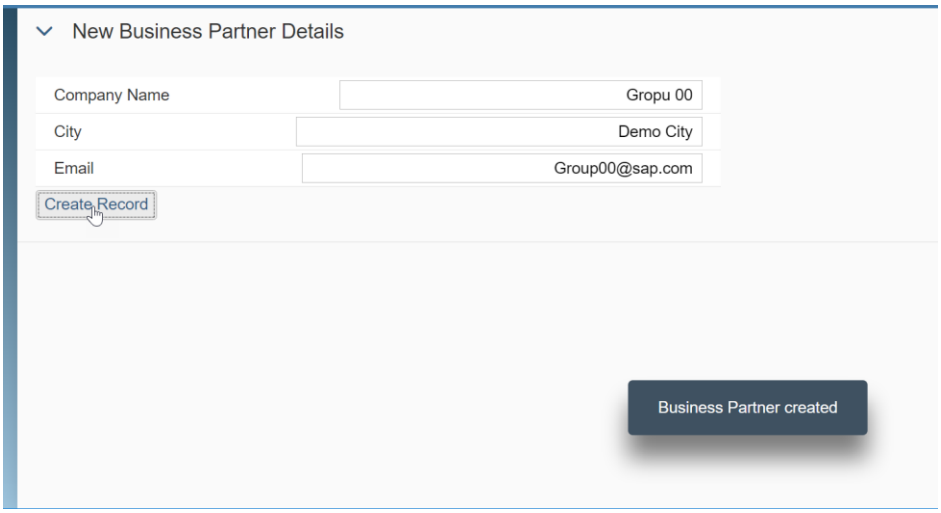
                destroy: function() {
                        // call the base component's destroy function
                        UIComponent.prototype.destroy.apply(this,
arguments);
                }

  });
});
```

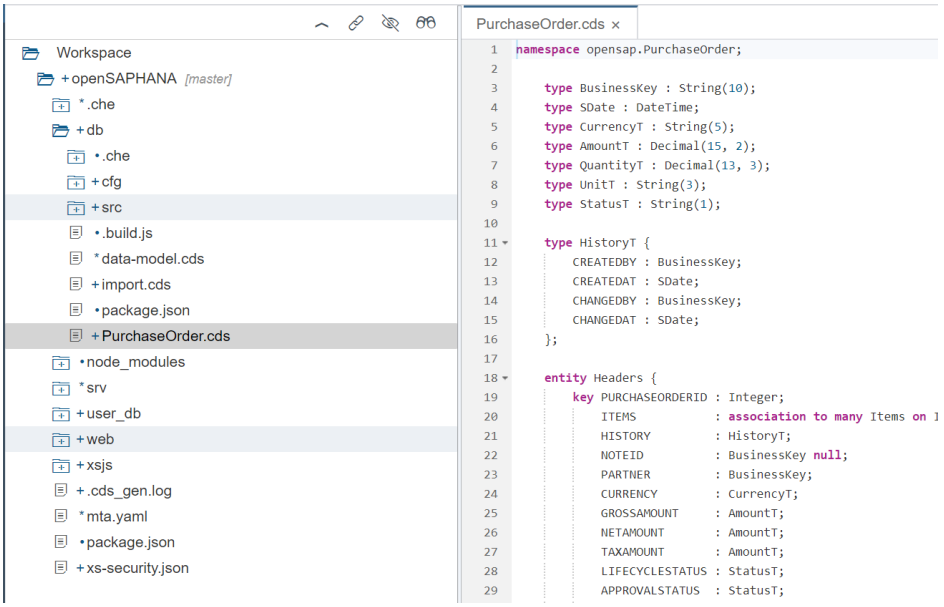
13. Save your files and re-run the **web** module.

14. Test your application in a web browser using the Run option.

    The URL would be **/odataDeep/**



Exercise 5.10: OData V4 and Fiori Annotations

Explanation	Screenshot
1. With the Odata V2 services based upon XSODATA, we were able to do some dynamic things in the UI based upon the metadata (such as column headers of tables). But with Odata V4 and the new CDS based services, we have much larger range of metadata which can be defined via CDS annotations that is expected by Fiori based applciations. These new features allow us to define nearly everything about the UI (value helps, sort options, columns for output, etc) in a declarative way within the CDS files. Let's return to the **db** module and the **PurchaseOrder.cds** file so we can extend our entities with such annotations.	
2. Some of this metadata is so fundamental (such as column descriptions) that we want to build them into the lowest layer – in the entity defintions themselves. So we will start with the entities in PurchaseOrder.cds **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/poCds.txt	namespace opensap.PurchaseOrder; using BUYER from '../db/import'; type BusinessKey : String(10); type SDate : DateTime; type CurrencyT : String(5) @title: '{i18n>currency}'; type AmountT : Decimal(15, 2); type QuantityT : Decimal(13, 3) @(title: '{i18n>quantity}', Measures.Unit: Units.Quantity); type UnitT : String(3) @title: '{i18n>quantityUnit}'; type StatusT : String(1); type HistoryT { CREATEDBY : BusinessKey @(title: '{i18n>CreatedBy}', Common.FieldControl: #ReadOnly); CREATEDAT : SDate @(title: '{i18n>CreatedAt}', Common.FieldControl: #ReadOnly); CHANGEDBY : BusinessKey @(title: '{i18n>ChangedBy}', Common.FieldControl: #ReadOnly);

```
      CHANGEDAT : SDate  @(title: '{i18n>ChangedAt}',
Common.FieldControl: #ReadOnly);
   };

   entity Headers {
      key PURCHASEORDERID : Integer @(title: '{i18n>po_id}',
Common.FieldControl: #Mandatory, Search.defaultSearchElement,
Common.Label: '{i18n>po_id}');
         ITEMS          : association to many Items on ITEMS.POHeader = $self
@(title: '{i18n>po_items}', Common: { Text: {$value: ITEMS.PRODUCT,
"@UI.TextArrangement": #TextOnly }});
         HISTORY        : HistoryT;
         NOTEID         : BusinessKey null @title: '{i18n>notes}';
         PARTNER        : BusinessKey @title: '{i18n>partner_id}';
         CURRENCY       : CurrencyT @(
          Common: {
                                                Text: {$value:
CURRENCY.CURRENCY, "@UI.TextArrangement": #TextOnly},
                                                ValueList: {entity: 'CURRENCY', type:
#fixed},
                                                ValueListWithFixedValues
                                          }
                                    );
         GROSSAMOUNT    : AmountT @( title: '{i18n>grossAmount}',
Measures.ISOCurrency: currency);
         NETAMOUNT      : AmountT @( title: '{i18n>netAmount}',
Measures.ISOCurrency: currency);
         TAXAMOUNT      : AmountT @( title: '{i18n>taxAmount}',
Measures.ISOCurrency: currency);
         LIFECYCLESTATUS : StatusT @(title: '{i18n>lifecycle}',
Common.FieldControl: #ReadOnly);
         APPROVALSTATUS  : StatusT @(title: '{i18n>approval}',
Common.FieldControl: #ReadOnly);
         CONFIRMSTATUS   : StatusT @(title: '{i18n>confirmation}',
Common.FieldControl: #ReadOnly);
         ORDERINGSTATUS  : StatusT @(title: '{i18n>ordering}',
Common.FieldControl: #ReadOnly);
         INVOICINGSTATUS : StatusT @(title: '{i18n>invoicing}',
Common.FieldControl: #ReadOnly);
   }


   entity Items {
      key POHeader    : association to Headers @title: '{i18n>poService}';
      key PRODUCT     : BusinessKey @(title: '{i18n>product}',
Common.FieldControl: #Mandatory, Search.defaultSearchElement);
         NOTEID      : BusinessKey null ;
         CURRENCY    : CurrencyT @(
          Common: {
                                                Text: {$value:
CURRENCY.CURRENCY, "@UI.TextArrangement": #TextOnly},
                                                ValueList: {entity: 'CURRENCY', type:
#fixed},
                                                ValueListWithFixedValues
```

```
                                    }
                                );
        GROSSAMOUNT  : AmountT @( title: '{i18n>grossAmount}',
Measures.ISOCurrency: currency);
        NETAMOUNT    : AmountT  @( title: '{i18n>netAmount}',
Measures.ISOCurrency: currency);
        TAXAMOUNT    : AmountT @( title: '{i18n>taxAmount}',
Measures.ISOCurrency: currency);
        QUANTITY     : QuantityT;
        QUANTITYUNIT : UnitT;
        DELIVERYDATE : SDate @title: '{i18n>deliveryDate}';
  }

        define view ItemView as
    select from Items
    {
        POHeader.PURCHASEORDERID,
        POHeader.PARTNER,
        PRODUCT,
        CURRENCY,
        GROSSAMOUNT,
        NETAMOUNT,
        TAXAMOUNT,
        QUANTITY,
        QUANTITYUNIT,
        DELIVERYDATE
    };
```

3. We are using the title annotation to add text descriptions that will be used later as field labels and column headers. These can be used at the reusable type level or within the entity or view definition itself.

   We also see that we can use FieldControl which will set the state of input fields based upon these columns – for example Read Only or Mandatory.

   We can also set Measures – for example the amount fields get their measure type from the Currency column.

   Finally we also see we can set ValueList, such as the



CURRENCY column pointing to foreign entity named CURRENCY. This allows us to populate Drop Down List Boxes or fill Value Helps within the UI from this definition.	
4. While we are in the **db** module, let's also add similiar UI-helpful annotations to the **import.cds** file. That way even these "Phantom" entites for foreign views can be annotated as well. **Note**: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/importCds.txt	type BusinessKey : String(10); type SDate : DateTime; type CurrencyT : String(5) @title: '{i18n>currency}'; type AmountT : Decimal(15, 2); type QuantityT : Decimal(13, 3) @(title: '{i18n>quantity}', Measures.Unit: Units.Quantity); type UnitT : String(3) @title: '{i18n>quantityUnit}'; type StatusT : String(1); @cds.persistence.exists entity PURCHASE_ORDER_ITEM_VIEW { key PO_ITEM_ID: Integer @(title: '{i18n>po_items}', Common: { Text: {$value: PRODUCT_ID, "@UI.TextArrangement": #TextOnly }}); PARTNER_ID: BusinessKey @title: '{i18n>partner_id}'; key PRODUCT_ID: BusinessKey @(title: '{i18n>product}', Common.FieldControl: #Mandatory, Search.defaultSearchElement); CURRENCY_CODE: CurrencyT @(Common: { Text: {$value: CURRENCY.CURRENCY, "@UI.TextArrangement": #TextOnly}, ValueList: {entity: 'CURRENCY', type: #fixed}, ValueListWithFixedValues }); AMOUNT: AmountT @(title: '{i18n>grossAmount}', Measures.ISOCurrency: currency); NET_AMOUNT: AmountT @(title: '{i18n>netAmount}', Measures.ISOCurrency: currency); TAX_AMOUNT: AmountT @(title: '{i18n>taxAmount}', Measures.ISOCurrency: currency); QUANTITY: QuantityT; QUANTITY_UNIT: UnitT; DELIVERY_DATE: SDate @title: '{i18n>deliveryDate}'; } @cds.persistence.exists Entity BUYER { BUILDING: String(10) @title: '{i18n>building}'; CITY: String(40) @title: '{i18n>city}'; COMPANYNAME: String(80) @title: '{i18n>company}'; COUNTRY: String(3) @title: '{i18n>country}'; EMAILADDRESS: String(255) @title: '{i18n>email}'; LEGALFORM: String(10) @title: '{i18n>legal}';

```
        key PARTNERID: Integer @title: '{i18n>partnerId}';
        PARTNERROLE: String(3) @title: '{i18n>partnerRole}';
        POSTALCODE: String(10) @title: '{i18n>postalCode}';
        REGION: String(4) @title: '{i18n>region}';
        STREET: String(60)  @title: '{i18n>street}';
}

@cds.persistence.exists
Entity USER_DETAILS {
        EMAIL: String(255)  @title: '{i18n>email}';
        FIRSTNAME: String(40) @title: '{i18n>fname}';
        LASTNAME: String(40) @title: '{i18n>lname}';
    key USERID: Integer @title: '{i18n>userId}';
}

@cds.persistence.exists
Entity CURRENCY {
        key CODE: String(3) @title: '{i18n>currCode}';
        CURRENCY: String(80) @title: '{i18n>currDesc}';
}
```

5. Now we want to add some less general annotations. These might only apply to a single service or UI page. Therefore we will go to the **srv** module and add a new file named **fiori-annotations-my-service.cds**

   This way we can build layers of annotations.

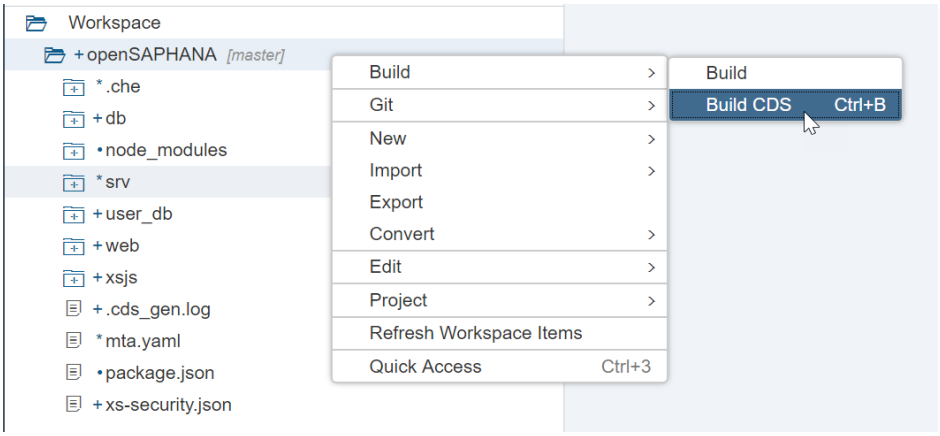
   Note: if you need help writing this code please refer to the solution at: https://github.com/SAP/hana-xsa-opensap-hana7/blob/snippets_2.3.2/ex5/fiori1.txt

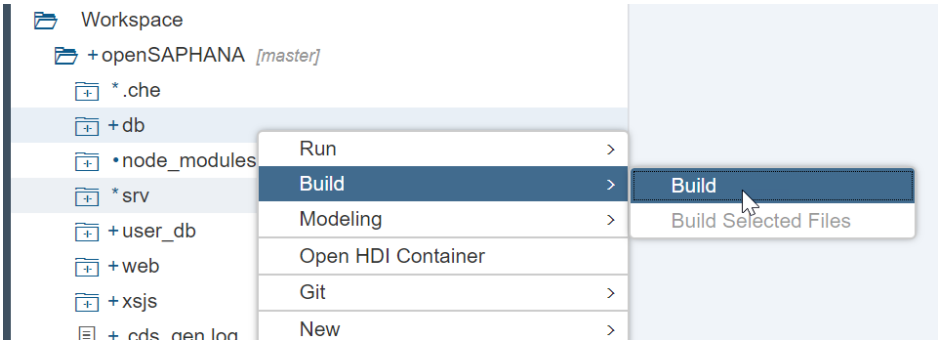


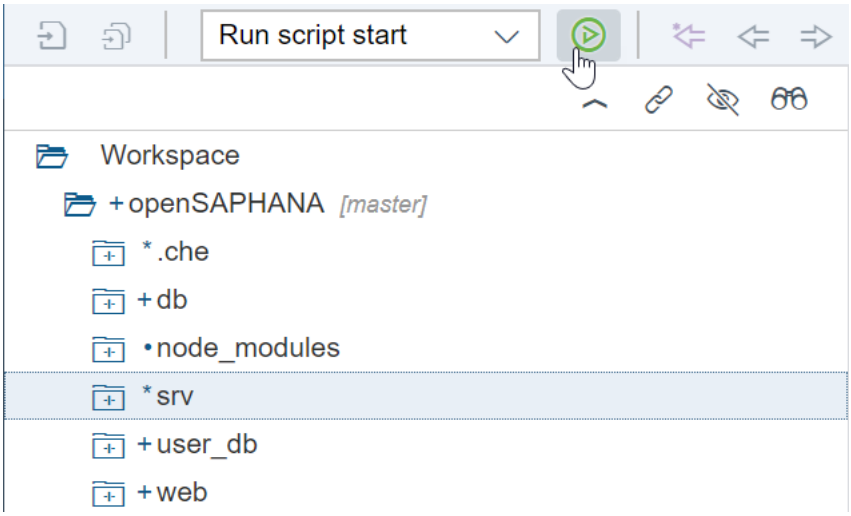
6. Save any edited files and then use the **Build CDS** option.

   Then build the **db** module.

   Finally re-run the **srv** module for all your changes to be presented in the Odata V4 services.







7. If you check the Odata V4 metadata document we should see a lot of new content generated by these annotations we just added.

   /odata/v4/opensap.hana.CatalogService/$metadata

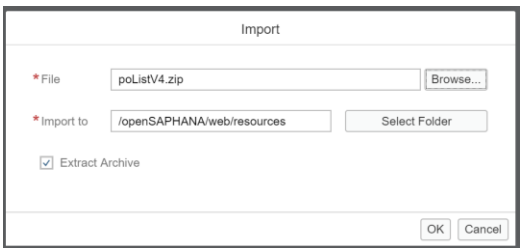
```
<Annotations Target="CatalogService.POs/PURCHASEORDERID">
   <Annotation Term="Common.FieldControl" EnumMember="Common.FieldControlType/Mandatory"/>
   <Annotation Term="Common.Label" String="Purchase Order ID"/>
</Annotations>
<Annotations Target="CatalogService.POs/ITEMS">
   <Annotation Term="Common.Text" Path="ITEMS/PRODUCT">
      <Annotation Term="UI.TextArrangement" EnumMember="UI.TextArrangementType/TextOnly"/>
   </Annotation>
   <Annotation Term="Common.Label" String="Purchase Order Items"/>
</Annotations>
<Annotations Target="CatalogService.POs/NOTEID">
   <Annotation Term="Common.Label" String="Notes"/>
</Annotations>
<Annotations Target="CatalogService.POs/PARTNER">
   <Annotation Term="Common.Label" String="Partner ID"/>
</Annotations>
<Annotations Target="CatalogService.POs/CURRENCY">
   <Annotation Term="Common.Text" Path="CURRENCY/CURRENCY">
      <Annotation Term="UI.TextArrangement" EnumMember="UI.TextArrangementType/TextOnly"/>
   </Annotation>
   <Annotation Term="Common.ValueList">
      <Record Type="Common.ValueListType">
         <PropertyValue Property="Label" String="Currency"/>
         <PropertyValue Property="CollectionPath" String="CURRENCY"/>
         <PropertyValue Property="Parameters">
            <Collection>
               <Record Type="Common.ValueListParameterInOut">
                  <PropertyValue Property="LocalDataProperty" PropertyPath="CURRENCY"/>
                  <PropertyValue Property="ValueListProperty" String="CODE"/>
               </Record>
               <Record Type="Common.ValueListParameterDisplayOnly">
                  <PropertyValue Property="ValueListProperty" String="CURRENCY"/>
               </Record>
            </Collection>
         </PropertyValue>
      </Record>
   </Annotation>
   <Annotation Term="Common.ValueListWithFixedValues" Bool="true"/>
   <Annotation Term="Common.Label" String="Currency"/>
</Annotations>
<Annotations Target="CatalogService.POs/GROSSAMOUNT">
   <Annotation Term="Measures.ISOCurrency" Path="currency"/>
   <Annotation Term="Common.Label" String="Gross Amount"/>
</Annotations>
<Annotations Target="CatalogService.POs/NETAMOUNT">
   <Annotation Term="Measures.ISOCurrency" Path="currency"/>
   <Annotation Term="Common.Label" String="Net Amount"/>
</Annotations>
<Annotations Target="CatalogService.POs/TAXAMOUNT">
   <Annotation Term="Measures.ISOCurrency" Path="currency"/>
   <Annotation Term="Common.Label" String="Tax Amount"/>
</Annotations>
<Annotations Target="CatalogService.POs/LIFECYCLESTATUS">
   <Annotation Term="Common.FieldControl" EnumMember="Common.FieldControlType/ReadOnly"/>
```

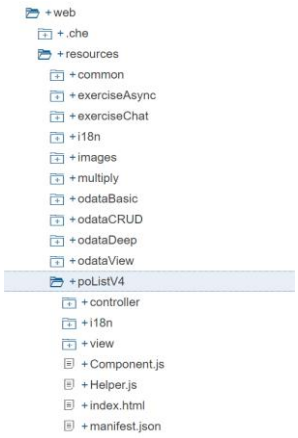
8. In a perfect world, we would run one of the Fiori wizards (Master/Detail or List Report) and generate a UI from the service definition and these annotations. However, the Fiori templates and wizards don't yet fully support Odata V4. Such support is planned but not yet available.

   But we can use the same "magic" that makes Fiori work to build our own appliation. This "magic" is the XML templating concept.

   Download the complete UI from https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/poListV4.zip

   Import it into the **web/resources** folder.

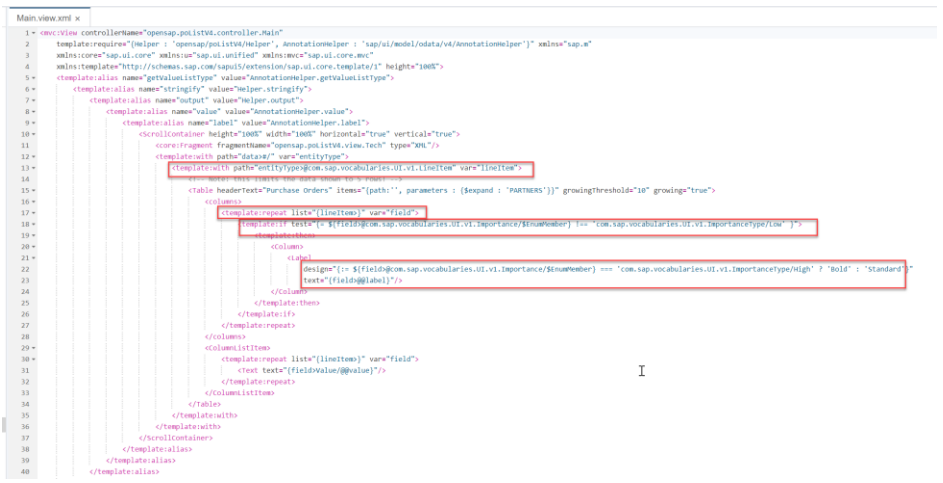




9. In the **manifest.json**, you will see that we switch from the Odata V2 to the Odata V4 SAPUI5 model. So while Fiori and some of the template based UI elements don't yet support Odata V4, SAPUI5 itself does support it.

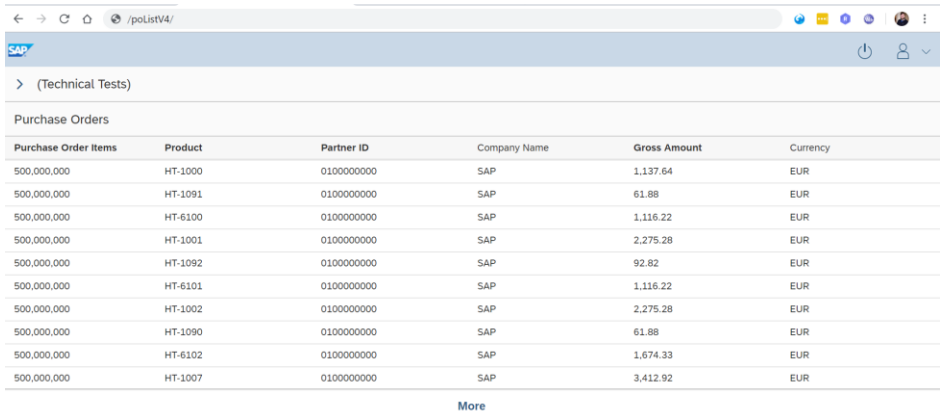


10. The **Main.view.xml** is where most of the dynamic templating takes place. You see the use of the template:with and template:repeat elements. This allows for the iteration over certain Odata Annotations to control the UI output dynamically



11. Re-run the **web** module so we can test this new poListV4 application.

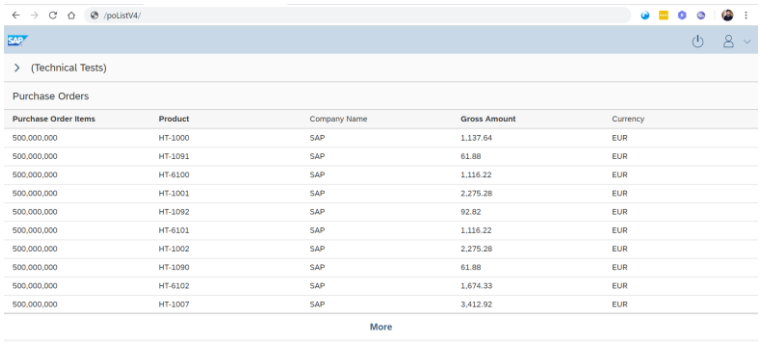
    The columns in the output is decided by the LineItem annotation in the fiori-annotations-my-service.cds file. The UI.Important also controls if the column headers are bold or not.



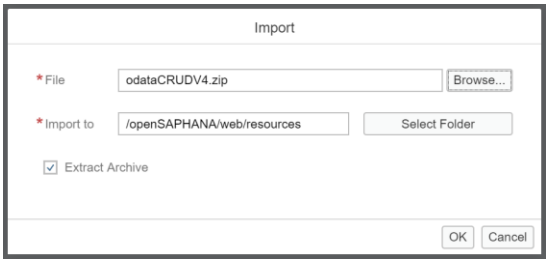
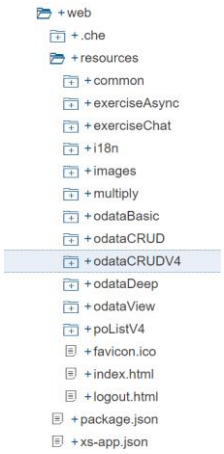
12. You can test this if you want. Return to **fiori-annotations-my-service.cds** and remove the Partner ID from the LineItem. Perform the Build CDS and re-run the **srv** module.

    Then simply refresh the web page on poListV4. You should see that the Parnter ID column is gone.





Exercise 5.11: Consume an OData V4 Service with Create Option

Explanation	Screenshot
1. Now its time to combine many things we've learned together. We will adjust the Odata V2 creation UI from exercise 5.7 to update it to work with our CDS-based Odata V4 service and its annotations. This will also allow us to test the Create/Update Odata V4 exits from Exercise 4.10. Download the complete UI from https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/odataCRUDV4.zip Import it into the **web/resources** folder.	 

2. Let's look at the major chagnes compared to the odataCRUD that was based upon Odata V2.

   One obvious change ist hat the manifest.json is changed to use the Odata V4 model.

   But in the **manifest.json**, we also have different settings for the Odata V4 model.

manifest.json ×

```
{
    "_version": "1.4.0",
    "start_url": "index.html",
    "sap.app": {
        "_version": "1.4.0",
        "type": "application",
        "resources": "resources.json",
        "i18n": "i18n/i18n.properties",
        "id": "odataTest",
        "title": "{{appTitle}}",
        "description": "{{appDescription}}",
        "applicationVersion": {
            "version": "${project.version}"
        },
        "dataSources": {
            "userService": {
                "uri": "/odata/v4/opensap.hana.CatalogService/",
                "type": "OData",
                "settings": {
                    "odataVersion": "4.0"
                }
            }
        }
    },
```

```
    "models": {
        "": {
            "type": "sap.ui.model.json.JSONModel",
            "settings": {
                "defaultBindingMode": "TwoWay"
            }
        },
        "userModel": {
            "dataSource": "userService",
            "type": "sap.ui.model.odata.v4.ODataModel",
            "preload": false,
            "settings": {
                "synchronizationMode": "None",
                "operationMode": "Server",
                "autoExpandSelect": true
            }
        },
        "config": {
            "type": "sap.ui.model.json.JSONModel"
        },
        "i18n": {
```

3. Another change is that we can't use the SmartTable control yet with a binding to the Odata V4 model. Therefore we will change to the m.table control in the **App.view.xml**

App.view.xml ×

```
                <InputListItem label="{i18n>EMail}">
                    <Input id="email" value="{/Email}"/>
                </InputListItem>
            </List>
            <Button text="{i18n>B3}" press="callUserService"/>
            <Button text="{i18n>B4}" press="callUserUpdate"/>
        </Panel>
        <Panel headerText="User List" expandable="true" expanded="true">
            <Table tableId="userTable" id="userTable" growingThreshold="10" growing="true"
                items="{ path: 'userModel&gt;/User', sorter: [{ path: 'USERID', descending: false}], events : {dataReceived : '.onDataEvents' } }">
                <columns>
                    <Column>
                        <header><Label text="User ID"/></header>
                    </Column>
                    <Column>
                        <header><Label text="First Name"/></header>
                    </Column>
                    <Column>
                        <header><Label text="Last Name"/></header>
                    </Column>
                    <Column>
                        <header><Label text="Email"/></header>
                    </Column>
                </columns>
                <items>
                    <ColumnListItem>
                        <cells>
                            <Input value="{path: 'userModel&gt;USERID'}" name="PERS_NO" editable="false"/>
                            <Input value="{path: 'userModel&gt;FIRSTNAME'}" name="FIRSTNAME"/>
                            <Input value="{path: 'userModel&gt;LASTNAME'}" name="LASTNAME"/>
                            <Input value="{path: 'userModel&gt;EMAIL'}" name="E_MAIL"/>
                        </cells>
                    </ColumnListItem>
                </items>
            </Table>
        </Panel>
```

4. The last major change between Odata V2 and V4 models has to do with the update and create events in the **App.controller.js**. The Odata V4 model is actually much easier in these events. You only need to bind the new data to the model. There are no special model functions to call.

App.controller.js

```
            sap.m.MessageBox.alert(error.toString());
        }
    },

    callUserService: function () {

        try {
            var oModel = this.getOwnerComponent().getModel("userModel");
            var result = this.getView().getModel().getData();
            var oList = this.byId("userTable"),
                oBinding = oList.getBinding("items"),
                // Create a new entry through the table's list binding
                oContext = oBinding.create({
                    "USERID": 0,
                    "FIRSTNAME": result.FirstName,
                    "LASTNAME": result.LastName,
                    "EMAIL": result.Email
                });

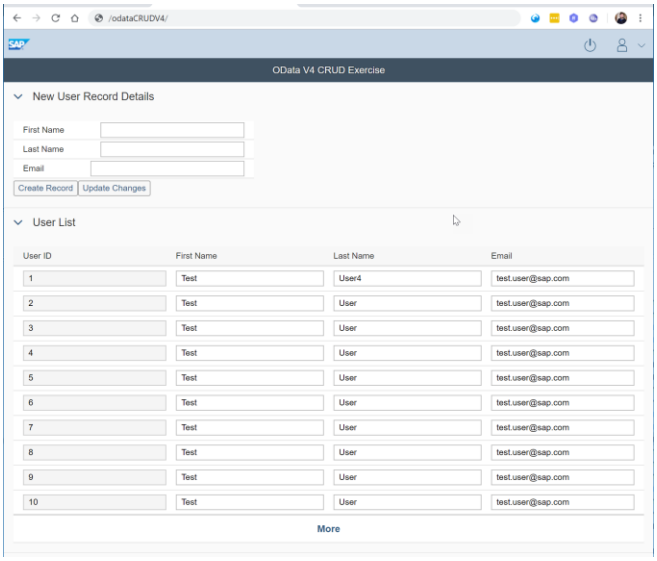
            // Note: this promise fails only if the transient entity is deleted
            oContext.created().then(function () {
                sap.m.MessageBox.alert("User created: " + oContext.getProperty("USERID"));
            }, function (oError) {
                sap.m.MessageBox.alert(oError.toString());
            });
        } catch (err) {
            sap.m.MessageBox.alert(err.toString());
        }
    },

    callUserUpdate: function () {
        var oModel = this.getOwnerComponent().getModel("userModel");

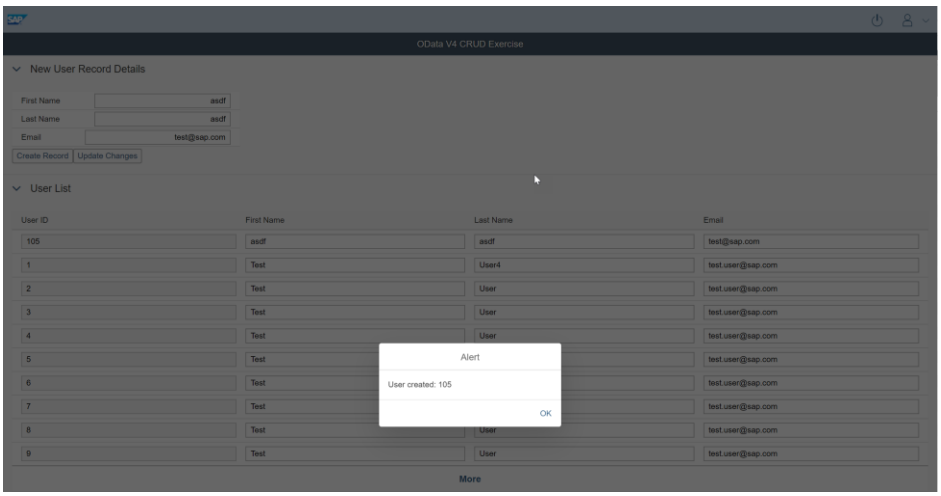
        var mParams = {};
        mParams.error = function () {
            sap.m.MessageToast.show("Update failed");
        };
        mParams.success = function () {
            sap.m.MessageToast.show("Update successful");
        };

        oModel.submitChanges(mParams);
    }
```

5. Re-run the web module and then test this new UI application via **/odataCRUDV4**



6. Test the user interface. You can also return to the console logs for the **srv** module and see the logging from your Odata V4 service exit from back in Exercise 4.10.

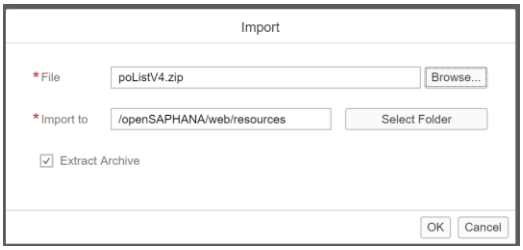


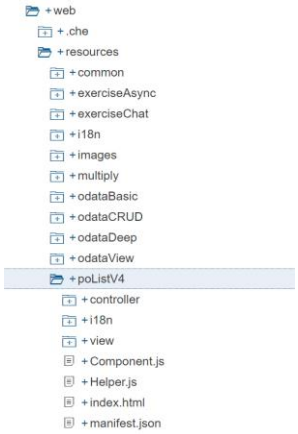


7. In a perfect world, we would run one of the Fiori wizards (Master/Detail or List Report) and generate a UI from the service definition and these annotations. However the Fiori templates and wizards don't yet fully support Odata V4. Such support is planned but not yet available.

   But we can use the same „magic" that makes Fiori work to build our own appliation. This „magic" is the XML templating concept.

   Download the complete UI from https://github.com/SAP/hana-xsa-opensap-hana7/raw/snippets_2.3.2/ex5/poListV4.zip and import it into the web/resources folder.





EXERCISE 6 – PACKAGING FOR TRANSPORT

Objective

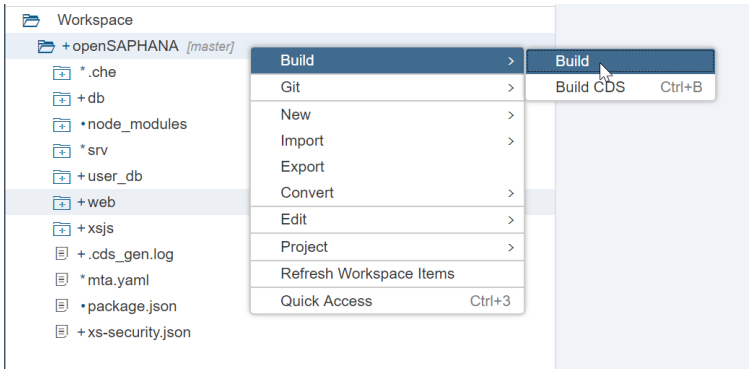
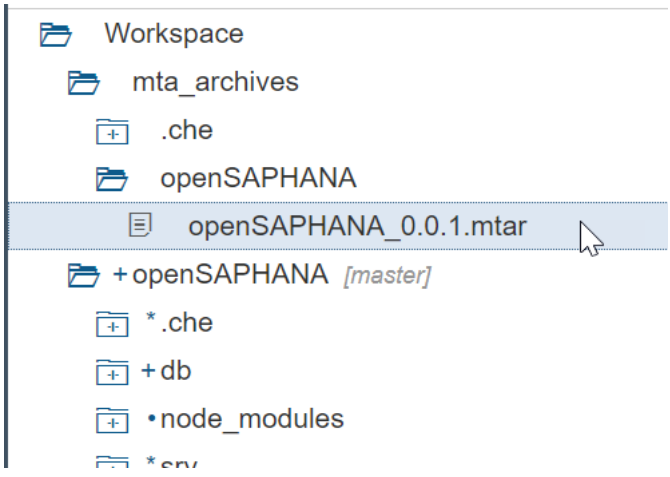
Now that we have a completed application we can package it for transport

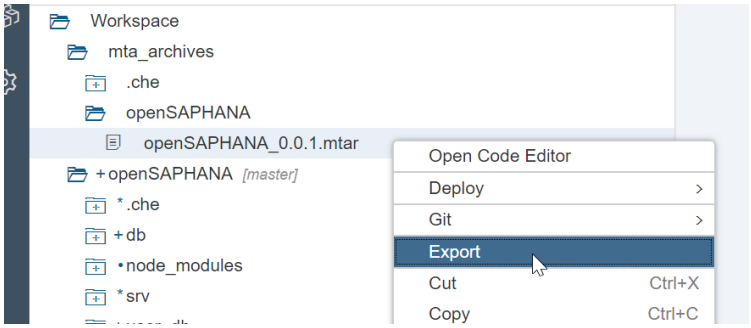
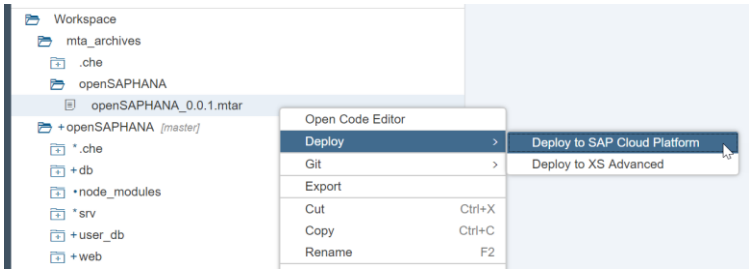
Exercise Description

- Build all modules
- Build project to create MTAR
- Download MTAR

EXERCISE 6 – SOLUTION

Exercise 6.1: Package for Transport

Explanation	Screenshot
1. The build and run process creates services on the XSA server; but they should be considered “local” testing services only. They are prefixed with your user name and workspace. We now want to package our complete application so that it’s installable in its final form and ready to transport to downstream systems.	
2. Perform a build at the project level.	
3. After a few minutes, you should now have a separate mta_archives project in your workspace. This is where the built archives for all our projects go.	

4. You can now manually download the MTAR file using **Export** and install it from the command line using the XS DEPLOY command.	
5. Optionally you can now deploy the MTA directly from the Web IDE for SAP HANA to the local XSA instance or to a remote SAP Cloud Platform Cloud Foundry system using **Deploy -> Deploy to SAP Cloud Platform**.	

Coding Samples

Any software coding or code lines/strings ("Code") provided in this documentation are only examples and are not intended for use in a production system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules for certain SAP coding. SAP does not warrant the correctness or completeness of the Code provided herein and SAP shall not be liable for errors or damages cause by use of the Code, except where such damages were caused by SAP with intent or with gross negligence.

www.sap.com/contactsap

© 2018 SAP SE or an SAP affiliate company. All rights reserved.
No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

THE BEST RUN SAP®