HD2dC03wl – oData Services - Lite

Product and Focus
HANA Platform/oData

Target Audience
Undergrduate/Graduate
Beginner to Intermediate
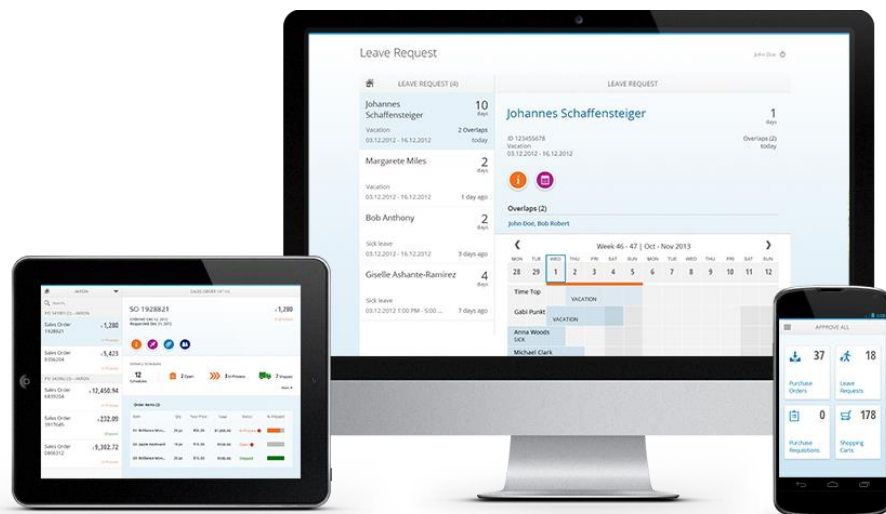
Author
Ross Hightower

MOTIVATION
This case introduces HANA oData services.

**PREREQUISITES**
HD1dC02wl – Create the Persistence Model

oData

OData has been called ODBC of the Web because it is becoming a standard, cross-platform way to access and update data. It relies on web technologies such as RESTful services, Uniform Resource Identifiers (URI) for resource location, HTTP verbs (GET, PUT, POST and DELETE) for operations and JavaScript Object Notation (JSON) for data representation. OData is quickly becoming a widely used protocol because it is fast, efficient and flexible.   Not only is the data easy to consume using oData, creating oData services on HANA is also very simple. A single line of code provides the ability to read, create, update and delete table. You can learn more about oData here and here.
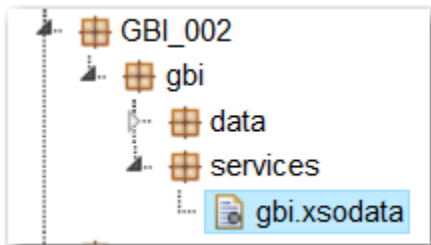
Create the Services

Create the Services Package

Logon to the WDW and locate the gbi package you created in case HD1dC2wl. Right-click the gbi package and create a new package called **services**.
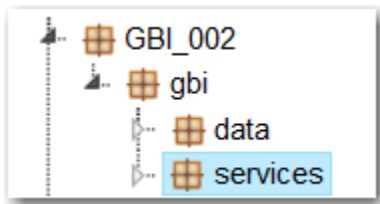
Services for Tables

Create oData Services

Right-click the services package and create a new file called **gbi.xsodata**.

The first service we'll create will be for the CUSTOMERS table. Enter the following code.

```
service {
    "GBI_002.gbi.data::GBI_002.MASTERDATA.CUSTOMERS" as "Customers";
}
```

Listing 1

Change the path name to fit your situation. You can find the correct name of the table by opening the table in the Catalog editor.
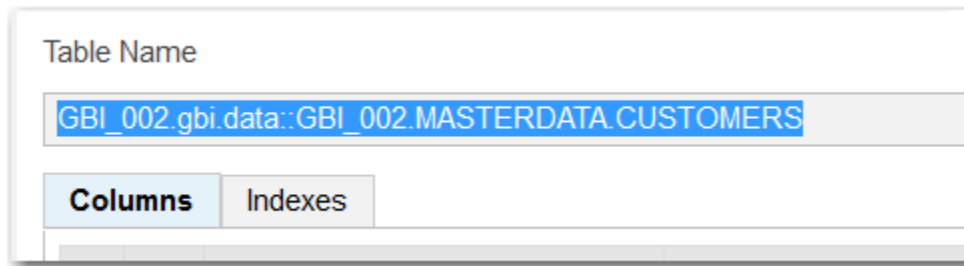
Table Name

GBI_002.gbi.data::GBI_002.MASTERDATA.CUSTOMERS

Columns   Indexes

Click the run icon (⊙) to execute the service. You should receive a 404 error. The reason is that we have not made the package externally accessible.
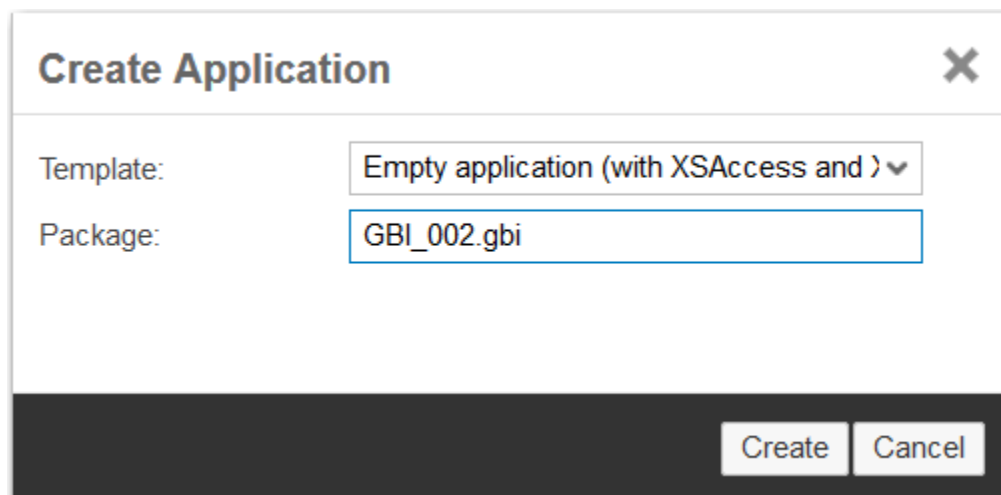
404 - Not found

We could not find the resource you're trying to access.
It might be misspelled or currently unavailable.

Possible solution

Please check the URL or try again later.
In case of ongoing problems please contact your system administrator.

?

Right-click the gbi package and select **Create Application**. Choose the Empty application.
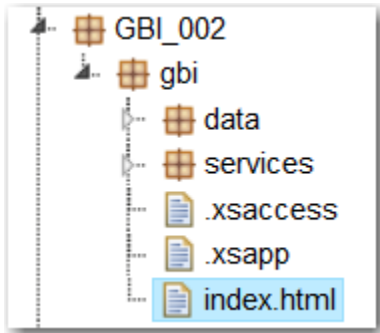
Create Application   ✕

Template: Empty application (with XSAccess and) ⌄

Package: GBI_002.gbi
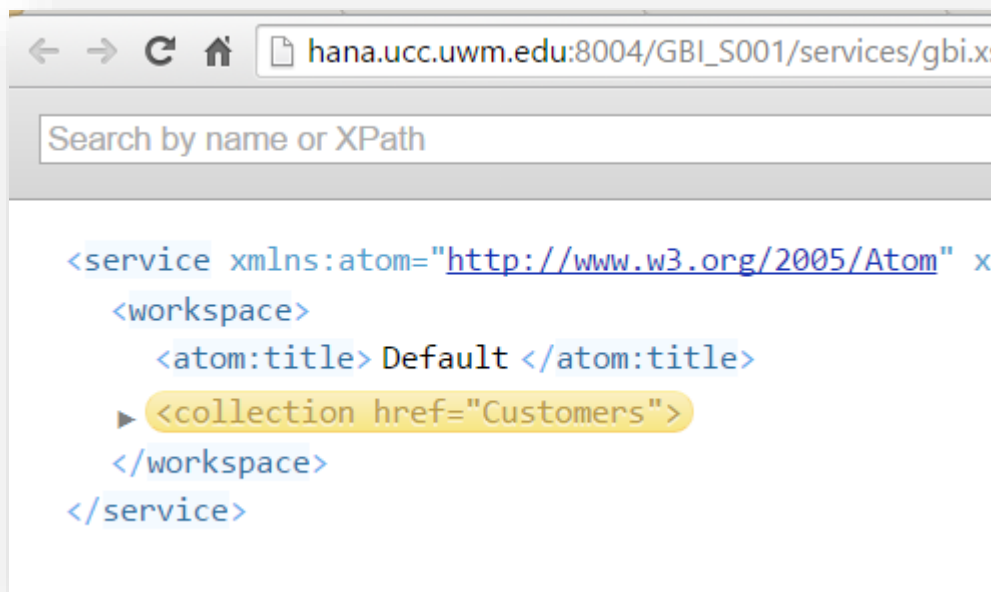
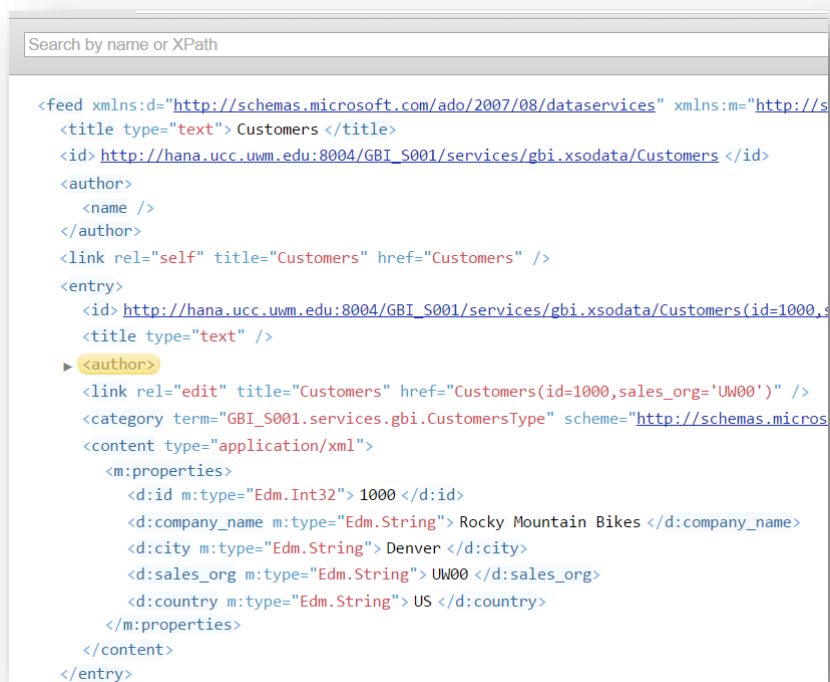   Create   Cancel

This creates three files.

The .xsapp file is essentially empty. Its presence indicates to HANA that the package contains an application. The .xsaccess file contains the data that controls access to the application. The key elements for our purposes are the exposed and authentication properties. Exposed determines whether the application is externally accessible and the authentication method indicates that a logon form will be used to authenticate users.

```
{
    "exposed": true,
    "authentication": [{
        "method": "Form"
    }],

    "mime_mapping": [{
        "extension": "jpg",
        "mimetype": "image/jpeg"
    }],
    "force_ssl": false,
    "enable_etags": true,
    "prevent_xsrf": true,
    "anonymous_connection": null,
    "cors": [{
        "enabled": false
    }],
    "cache_control": "no-cache, no-store",
    "default_file": "index.html"
}
```

Now run the service again (Chrome was used for all the screenshots in this document. Other browsers will not show the data without installing extensions).

```
<service xmlns:atom="http://www.w3.org/2005/Atom" x
    <workspace>
        <atom:title> Default </atom:title>
    ► <collection href="Customers">
    </workspace>
</service>
```

The service document shows one service, the Customers service we created. Add **/Customers** to the end of the URL and hit enter and the customer data will be retrieved.


```
<feed xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://s
    <title type="text"> Customers </title>
    <id> http://hana.ucc.uwm.edu:8004/GBI_S001/services/gbi.xsodata/Customers </id>
    <author>
        <name />
    </author>
    <link rel="self" title="Customers" href="Customers" />
    <entry>
        <id> http://hana.ucc.uwm.edu:8004/GBI_S001/services/gbi.xsodata/Customers(id=1000,s
        <title type="text" />
      ► <author>
        <link rel="edit" title="Customers" href="Customers(id=1000,sales_org='UW00')" />
        <category term="GBI_S001.services.gbi.CustomersType" scheme="http://schemas.micros
        <content type="application/xml">
            <m:properties>
                <d:id m:type="Edm.Int32"> 1000 </d:id>
                <d:company_name m:type="Edm.String"> Rocky Mountain Bikes </d:company_name>
                <d:city m:type="Edm.String"> Denver </d:city>
                <d:sales_org m:type="Edm.String"> UW00 </d:sales_org>
                <d:country m:type="Edm.String"> US </d:country>
            </m:properties>
        </content>
    </entry>
```

One of the advantages of using RESTful services is that data access is achieved entirely through the URL. There are a number of parameters available to achieve your results. Try the examples in the following table.

URL	Result
/Customers('1000')	Retrieves the customer with the matching primary key
/Cucstomers?$format=json	Customers data in JSON format
/Customers?$orderby=CompanyName	Customers data sorted by the property CARRNAME
/Customers?$filter=CompanyName eq 'Rocky Mountain Bikes'	Customers with CompanyName property equal to 'Rocky Mountain Bikes'
/Customers?$select=CompanyName,SalesOrgID	Just the CompanyName and SalesOrgID properties of Customers

You can also combine parameters. For example:

/Customers?$select=CompanyName,SalesOrgID&$filter=CompanyName eq 'Rocky Mountain Bikes'

To see the metadata of the service add the **/$metadata** option to the end of the service document URL.

```
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">
  <edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m
    <Schema xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://sc
      <EntityType Name="CustomersType">
      ► <Key>
          <Property Name="ID.CustomerID" Type="Edm.String" Nullable="false" MaxLength="10" />
          <Property Name="CompanyName" Type="Edm.String" Nullable="false" MaxLength="35" />
          <Property Name="Address.Address" Type="Edm.String" MaxLength="35" />
          <Property Name="Address.City" Type="Edm.String" MaxLength="20" />
          <Property Name="Address.Region" Type="Edm.String" MaxLength="2" />
          <Property Name="Address.Country" Type="Edm.String" MaxLength="2" />
          <Property Name="Address.Postal_code" Type="Edm.String" MaxLength="5" />
          <Property Name="SalesOrgID" Type="Edm.String" MaxLength="4" />
      </EntityType>
    ► <EntityContainer Name="gbi" m:IsDefaultEntityContainer="true">
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

The metadata document describes the various elements and services available in the oData model.

To see all the options refer to the documentation at http://www.odata.org/.

Add the Sales Org Service
Return to the gbi.xsodata file and modify the code in gbi.xsodata to look like the following:

```
service {

    "GBI_002.gbi.data::GBI_002.MASTERDATA.CUSTOMERS" as "Customers";
    "GBI_002.gbi.data::GBI_002.MASTERDATA.SALES_ORGS" as "SalesOrg"
            navigates ("SOrg_Customers" as "SOrgCustomers");

    association "SOrg_Customers" principal "SalesOrg"("ID") multiplicity "1"
    dependent "Customers"("SalesOrgID") multiplicity "*";

}
```

Listing 2

Remember to update the highlighted code to reflect your table names. This adds a service for the SALES_ORGS table accessible as /SalesOrg. It also creates an association (called SOrgCustomers) between the SALES_ORG table and the CUSTOMERS table. The association forms a one-to-many link based on the ID field in the SALES_ORG table and the SalesOrgID field in the CUSTOMERS table. The term "SOrg_Customers" in the navigates statement links the service to the association.
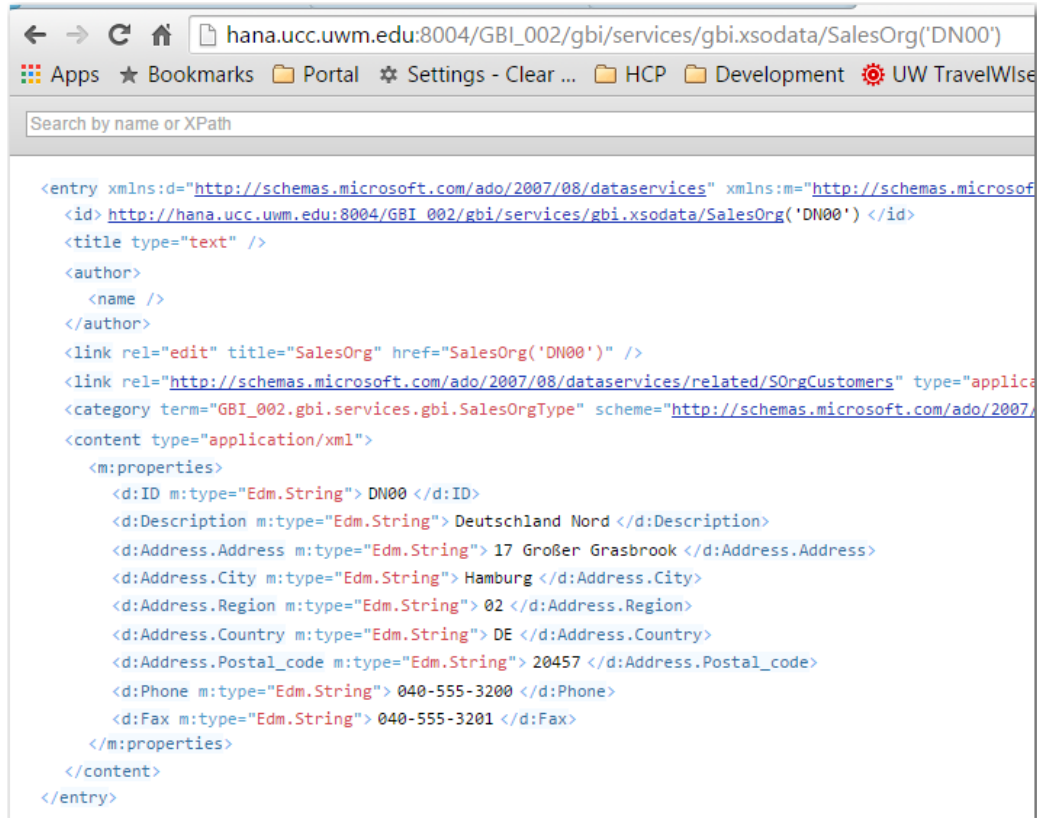
If you view the service metadata you can see the association metadata. The NavigationProperty is the parameter used to navigate the association. Note this value is the term after the "as" in navigates ("SOrg_Customers" as "SOrgCustomers").

```
</EntityType>
<EntityType Name="SalesOrgType">
    <Key>
        <PropertyRef Name="ID" />
    </Key>
    <Property Name="ID" Type="Edm.Int32" Nullable="false" />
    <Property Name="SALES_ORG" Type="Edm.String" MaxLength="4" />
    <Property Name="DESCRIPTION" Type="Edm.String" MaxLength="15" />
    <NavigationProperty Name="SOrgCustomers" Relationship="f14e.050.S
</EntityType>
```

A description of the association is also provided.

```
<Association Name="SOrg_CustomersType">
    <End Type="f14e.050.Services.gbi.SalesOrgType" Role="SalesOrgPrincipal" Multiplicity="1"/>
    <End Type="f14e.050.Services.gbi.CustomersType" Role="CustomersDependent" Multiplicity="*"/>
    <ReferentialConstraint>
        <Principal Role="SalesOrgPrincipal">
            <PropertyRef Name="SALES_ORG"/>
        </Principal>
        <Dependent Role="CustomersDependent">
            <PropertyRef Name="SALES_ORG"/>
        </Dependent>
    </ReferentialConstraint>
</Association>
<EntityContainer Name="gbi" m:IsDefaultEntityContainer="true">
    <EntitySet Name="Customers" EntityType="f14e.050.Services.gbi.CustomersType" />
    <EntitySet Name="SalesOrg" EntityType="f14e.050.Services.gbi.SalesOrgType" />
    <AssociationSet Name="SOrg_Customers" Association="f14e.050.Services.gbi.SOrg_CustomersType">
        <End Role="SalesOrgPrincipal" EntitySet="SalesOrg"/>
        <End Role="CustomersDependent" EntitySet="Customers"/>
    </AssociationSet>
```

To use the association, first retrieve the sales organizations by adding **/SalesOrg** to the end of the service document URL. Then retrieve the DN00 sales organization by using **/SalesOrg('DN00')**. This construction works with the primary key of the table.

To retrieve the customers associated with sales organization DN00 add **/SalesOrg('DN00')/SOrgCustomers** to the service document URL.

Exercise 1

1. Add an association between CUSTOMERS and SALES_ORDERS with NavigationProperty Orders.
2. Add services for the SALES_ORDERS and SALES_ORDER_DETAILS tables. Include an association between SALES ORDERS and SALES_ORDER_DETAILS with NavigationProperty Details
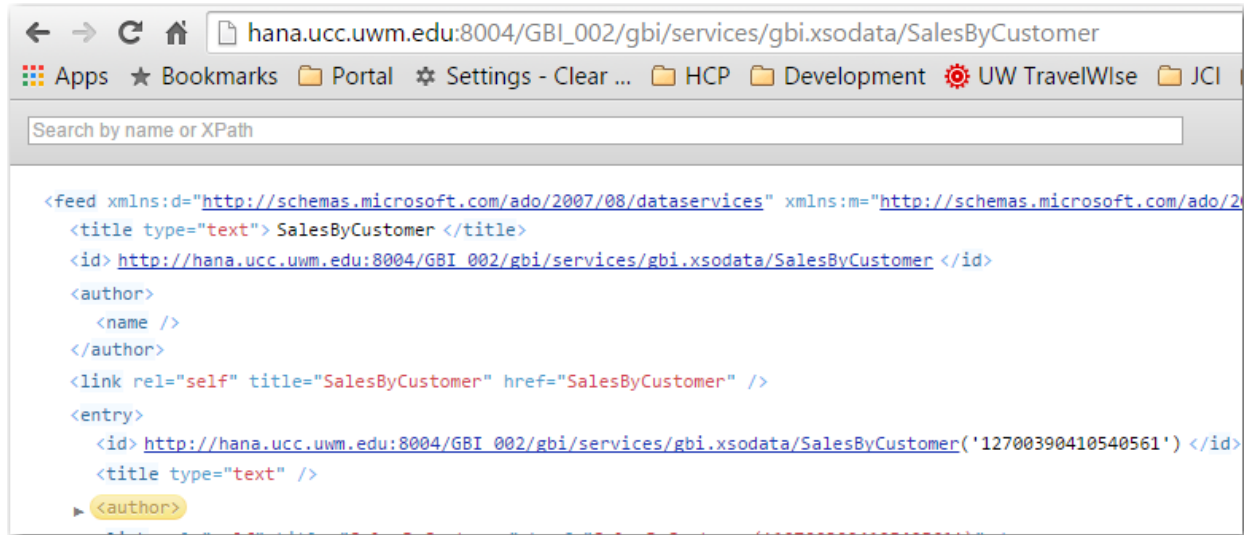
Services for Views

Database views do not have keys and oData services must have keys so you either must designate one of the columns as a key or you must have HANA generate a temporary key for the service.

Add a Service for the SalesByCustomer View

Add the service below for the SalesByCustomer view. This illustrates how to generate a key.

```
"GBI_002.gbi.data::GBI_002.SALES.SalesByCustomer" as "SalesByCustomer"
     key generate local "GenID";
```

Listing 3

Exercise 2

1. Create services for the CustomerSales view.

Create an Aggregation Service

You can also aggregate on fields with oData services. The service below aggregates on the GrossAmount.Amount field of the SALES_ORDERS table.

```
"GBI_002.gbi.data::GBI_002.SALES.SALES_ORDERS" as "SalesRevenue"
     key generate local "GenID"
     aggregates always (SUM of "GrossAmount.Amount");
```

Listing 4

To extract the aggregated revenue for CustomerID add ?$select=CustomerID,GrossAmount.Amount to the URL.
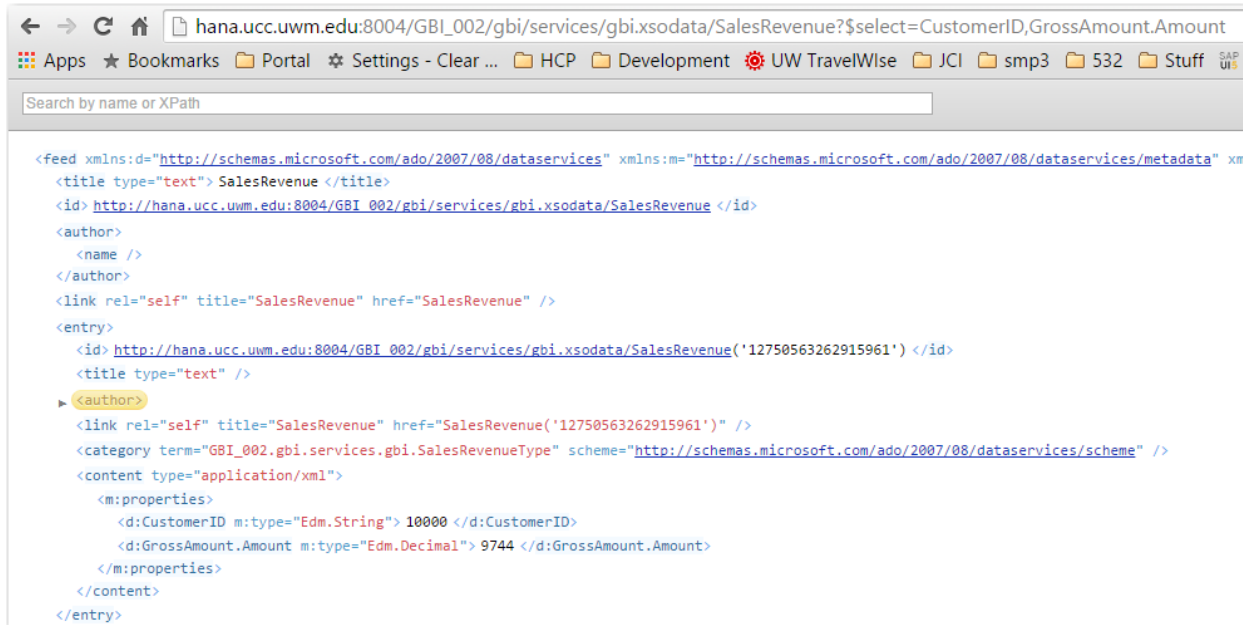
Exercise 3

1. Create an aggregation service for the SALES_ORDER_DETAILS table that aggregates Quantity.

CUD Operations

You can perform create, update and delete operations on the tables unless you specifically restrict the operations performed on the tables by adding code like the following:

```
"GBI_002.gbi.data::GBI_002.MASTERDATA.CUSTOMERS" as "Customers"
    create forbidden
    update forbidden
    delete forbidden;
```

Don't add this code because we want to be able to modify the data in the tables.

Testing in the remainder of this document is done using the Postman Google App. You can use any RESTful client to test the services but a RESTful client is required because we are going to perform POSTs, PUTs and DELETEs which cannot be submitted from a browser's address field.

Google "postman chrome app" and open a link that looks like this:

Install the app and follow the directions to open it. Click Authorization and change the authorization type to Basic Auth. Enter your HANA username and password.

Enter you service document URL and make sure the method is set to GET.

Click Send. The service document is retrieved and the status of 200 OK is returned

Postman keeps a history of the requests you perform on the left side of the screen so you can recall them.
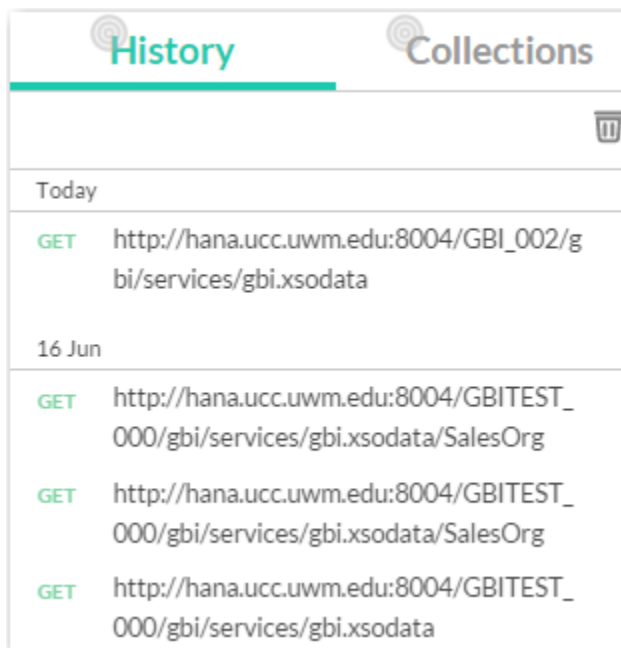
Now add /SalesOrg to the URL and click send to retrieve the contents of the SALES_ORGS table.
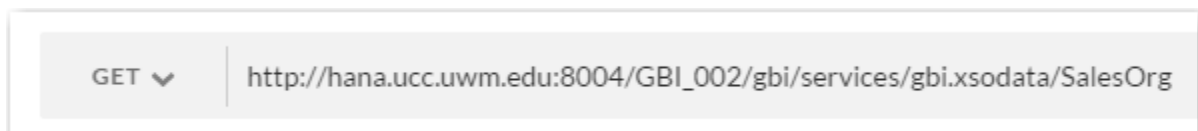
Update the .xsaccess File

Before we can perform creates, updates and deletes we have to update the .xsaccess file. Open the file and change the value of the **prevent_xsrf** property to **false**. The xsrf token is a security mechanism to prevent cross-site forgery attacks.

```
"force_ssl": false,
"enable_etags": true,
"prevent_xsrf": false,
"anonymous_connection": null,
```

Of course, this leaves your application open to cross-site forgery attacks so, **once you have finished this exercise**, you should set this back to true.

Create

To create a record you use the POST HTTP verb and provide the data in the header of the request in JSON format.

In the Postman client, change the method to **POST** and add **/SalesOrg** to the service document URL.

```
POST  http://hana.ucc.uwm.edu:8004/GBI_002/gbi/services/gbi.xsodata/SalesOrg
```

Click on the Body table and then the **Raw** option and enter the following data to the HTTP request header:

```
{
  "ID" : "MS00",
  "Description" : " Southern Mexico"
}
```
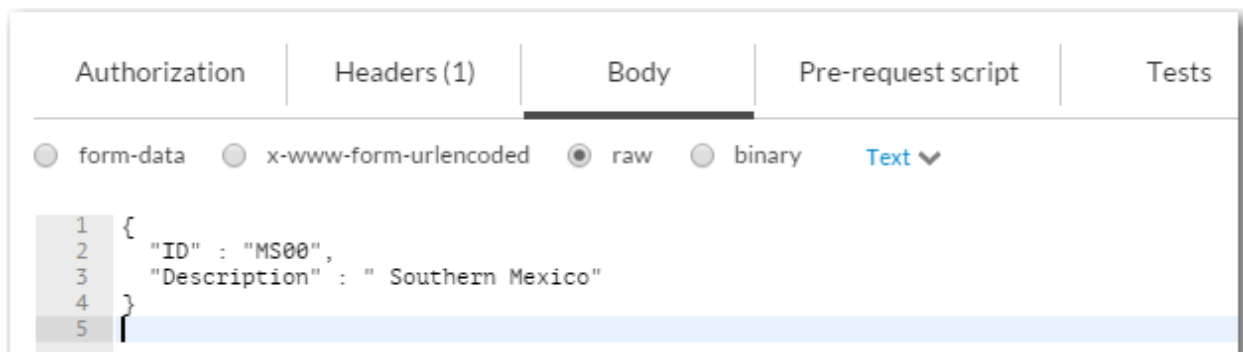
Click on the Headers tab and then enter the values shown below (Content-Type and applicaton/json). This tells HANA that the input data is in JSON format.

Authorization	Headers (2)	Body	Pre-request script	Tests
✓ Authorization				Basic R0JJXzAwMj
✓ Content-Type				application/json|

When you hit send the request, HANA responds with a status of 201 Created and the created record is returned.

Body   Cookies   Headers (7)   Tests   Status **201 Created**   Time   800 ms

Pretty   Raw   Preview   XML ✔   ≡|

```
1  <?xml version="1.0" encoding="utf-8"
2  <entry xml:base="http://hana.ucc.uwm.
   /ado/2007/08/dataservices" xmlns:m="h
   .org/2005/Atom">
3      <id>http://hana.ucc.uwm.edu:8004/
4      <title type="text"></title>
5      <author>
6          <name />
7      </author>
```

Change the HTTP method to GET (or select the GET request in the history) and click Send again to see the contents of the table.

```
<m:properties>
    <d:ID m:type="Edm.String">MS00</d:ID>
    <d:Description m:type="Edm.String"> Southern Mexico</d:Description>
    <d:Address.Address m:type="Edm.String" m:null="true"></d:Address.Ad
    <d:Address.City m:type="Edm.String" m:null="true"></d:Address.City>
    <d:Address.Region m:type="Edm.String" m:null="true"></d:Address.Reg
    <d:Address.Country m:type="Edm.String" m:null="true"></d:Address.Co
    <d:Address.Postal_code m:type="Edm.String" m:null="true"></d:Addres
    <d:Phone m:type="Edm.String" m:null="true"></d:Phone>
    <d:Fax m:type="Edm.String" m:null="true"></d:Fax>
</m:properties>
```
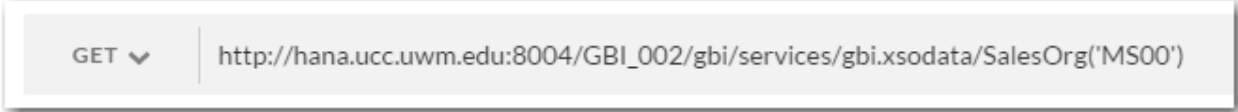
You can also open the table in the Catalog editor to see the table contents.

Update

To perform an update you must use a URL that specifies the record to be updated and provide the updated data in HTTP request header in JSON formation.
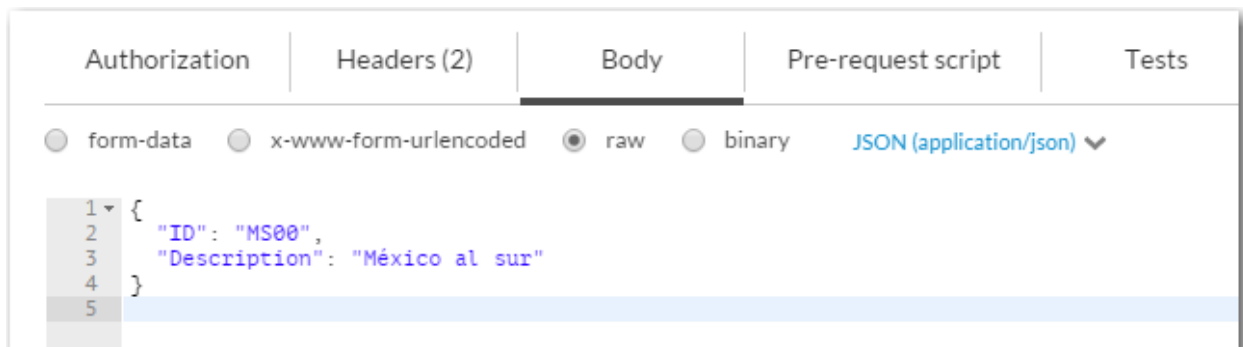
In the Postman client, change the URL to add the id of the newly created record (/SalesOrg('MS00')) and send a GET request.

The response will be the newly created record. Change the method to PUT and enter the following data to the HTTP request header:

```
{
  "ID": "MS00",
  "Description": "México al sur"
}
```


When you click Send the response will have a status of 204 No Content if the update was successful and no data will be returned.
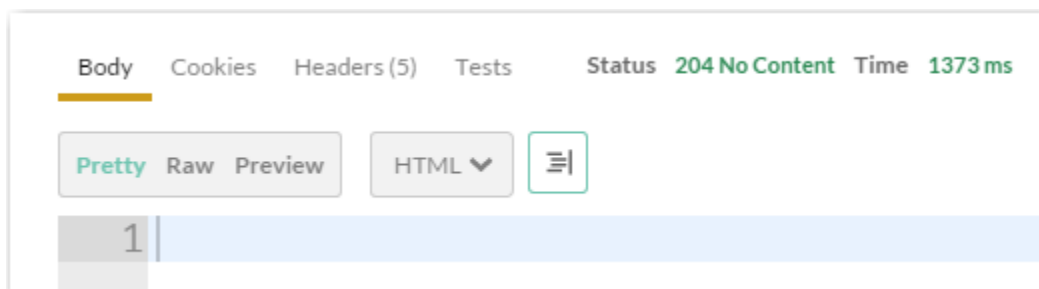
You can check that the update was successful by changing the method to GET and clicking Send again.
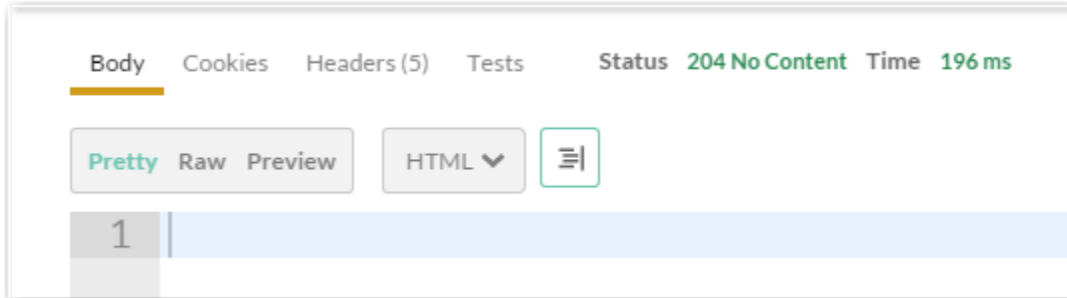
```
<m:properties>
    <d:ID m:type="Edm.String">MS00</d:ID>
    <d:Description m:type="Edm.String">México al sur</d:Description>
    <d:Address.Address m:type="Edm.String" m:null="true"></d:Address.Address>
```

Delete

To perform a DELETE you must change the method to DELETE and provide a URL that identifies the record to delete. Make sure you have included /SalesOrg('MS00') on the URL to select the new record then change the method to DELETE and click Send. The response is:

Body   Cookies   Headers (5)   Tests   Status 204 No Content Time 196 ms

Pretty Raw Preview   HTML ✓   ≡|

1 |

If you change the method to GET and click Send you will receive a 404 Not Found error because the record no longer exists:

Body   Cookies   Headers (7)   Tests   Status 404 Not Found Time 116 ms

Pretty Raw Preview   XML ✓   ≡|

```
1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <error xmlns="http://schemas.microsoft.com/ado/2007/08/dataser
3     <code/>
4     <message xml:lang="en-US">Resource not found.</message>
5 </error>
```