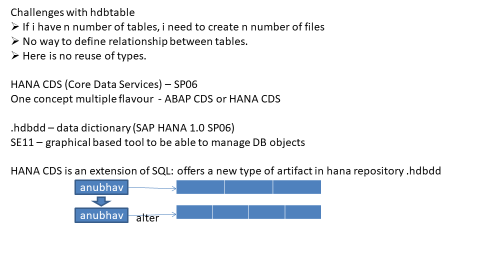
**Hana CDS Artifacts:**

****

**Fundafox:**

* To create .hdbdd file go to new->other->search for DDL
* .hdbdd stands for Hana database data dictionary

**Hana – Data Import through CSV files:**

**Fundafox:**

* To create .hdbdti file go to new->other->table import configuration

**Hana – Stored Procedures:**

Explicit table declaration:

Explicit table type creation in procedure

Tt\_type\_name **table(**

Colname1 *hanaDataType,*

Colname2 *hanaDataType*

**)**

**Fundafox:**

When this table type is generated all the column names are converted to upper case, so make sure that your DB table has also column name in upper case or create alias in queries.

With HANA 2.0 the assignment operator is not =, we should use :=

**Syntax to insert data inside a table type**

:table((col1,col2,...),index);

**Syntax for For loop**

FOR i IN min..max DO

END FOR;

**Use Case**: Design a procedure, where input is an integer and count up to the integer and multiply every count with 10, finally the output is a table of multiplied integers.

**Random() :** return random record from table

**select** S **from** "BIP"."BIP.data::bip.Products" **order** **by** **rand**() **limit** 1;

**select** EMPLOYEEID **from** "BIP"."BIP.data::reuse.employees" **order** **by** **rand**() **limit**

**Top():** returns top 1000 records from table

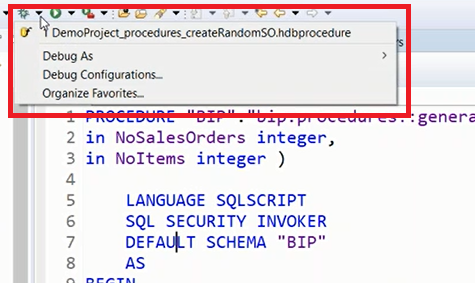
select top 1000 \* from "BIP"."BIP.data::bip.Products"

**Debugger**:

We can stop executing the script.

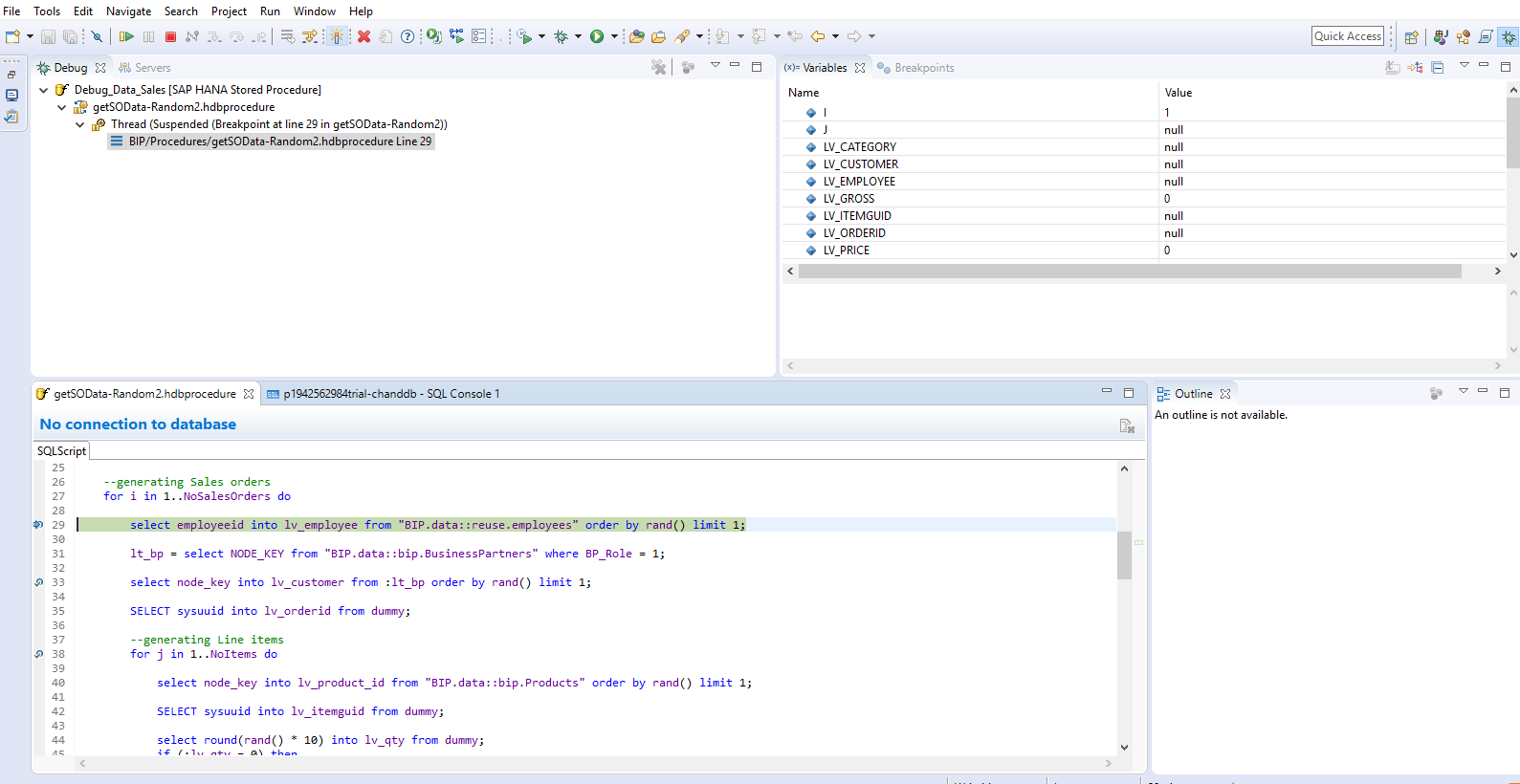
Before that we need to create **Debugging configuration** which would receive the values for input parameters.

* We cannot setup debugger on declare statements.
* Before creating **Debugging configuration** put debugger points



Press Debugger icon to see debugger configuration option.

Press Debug perspective to see below screen: go to window-> perspective-> debug perspective



**Functions**:

Light weight procedural SQL script blocks which are known as Functions.

Sometimes there is a need to write reusable functionality and these functionalities, we want to consume as it is inside of SQL query. For example string function, aggregation functions, number functions, date function.

SUM(), MIN(), MAX(), AVG(), FLOOR(), ABS(), ROUND(),SUSBSTRING(),TO\_CHAR(), TO\_NUMBER()

Functions always returns one value.

Advantages of Functions:

* You call them in SQL statement. SELECT SUM(col) FROM dbtable.
* You can use them in imperative logic. if ABS(2.33) = 2 then
* SAP HANA SP9 onwards we can assign output of a function to a scalar variable directly.

declare disPrice decimal(10,2) := calculateDiscount(30,’PC’)

**scalarFunction:**

* It returns a scalar output like one single value.
* Any number of input parameters.
* Can be used to assign value directly.
* Can be treated as column of a table like they are single variable.

Ex: Select node\_key from tab;

**tableFunction**:

* Exactly returns one table output.
* Any number of inputs.
* Used inside of queries.
* Returns a table.

Ex: Select table from table;

Select \* from table;

Select \* from "BIP"."BIP.Functions::getCustomerIds"()

**Fundafox**:

* To create scalar function right click on folder->new->other->scalar function.
* To create table function right click on folder->new->other->table function.

**Database Views:**

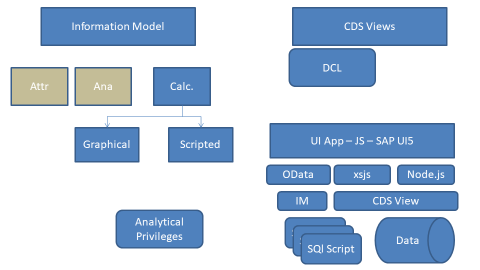
Data inside your application is stored in multiple database table. Always there is join or sub query required to read this data from DB tables. Sometimes, the person who need data doesn't have knowledge on SQL. Also, it becomes so complex in terms of SQL to generate a join between 100s of tables.

Views are stored queries, they do not replicate a table again in database, at runtime they call a query, fetch the data in user desired format.

By using views which gives very good security.

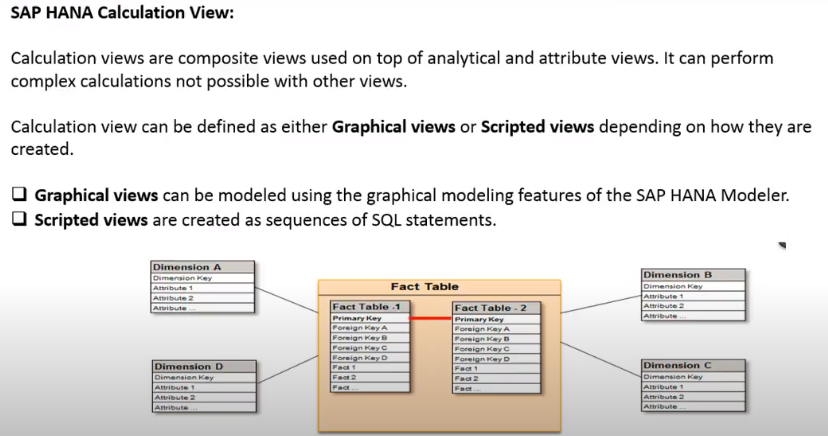
Advantages:

* Currency conversions.
* Data type conversions.
* Formulas
* Add calculated columns
* Input Parameters
* Value restrictions



**Fundafox:**

* Go to -> new->other-> search for View.
* Attribute views, Analytical views are not using now a days. Those are deprecated.

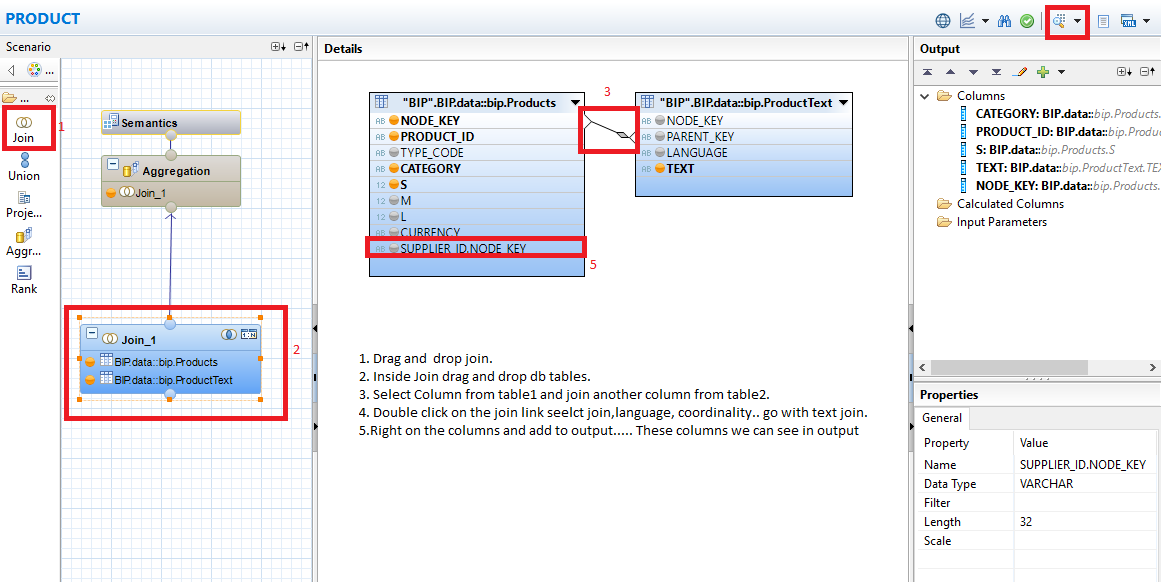


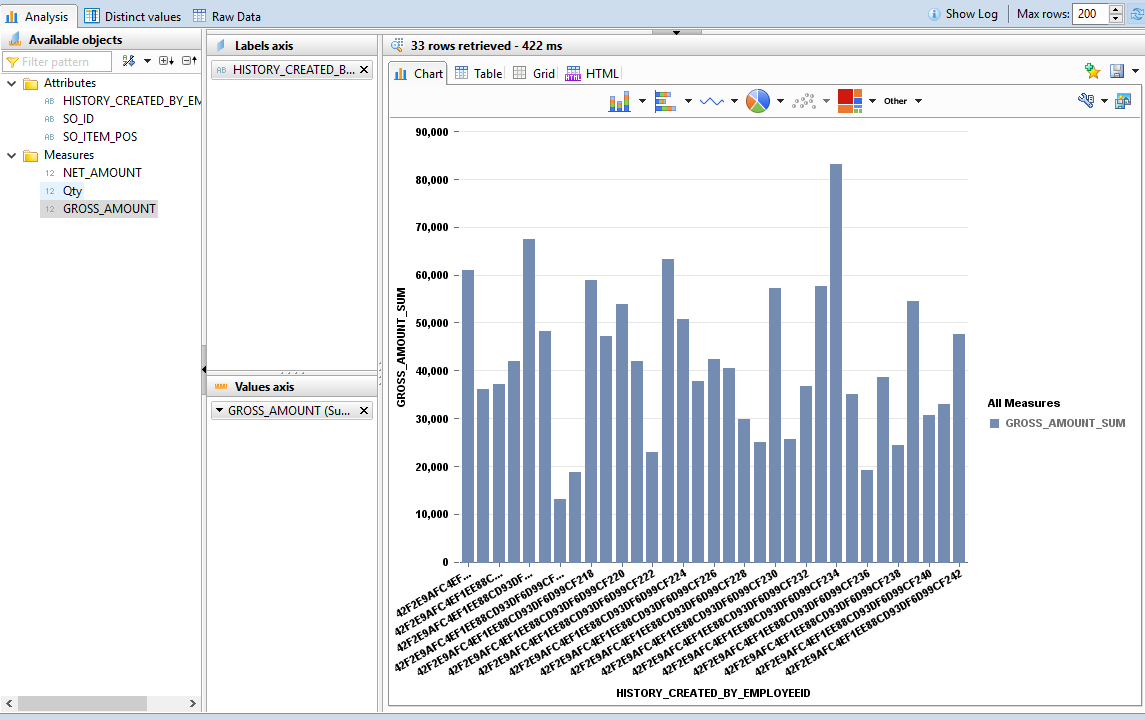
Fundafox:

Calculation view will create on “**\_SYS\_BIC” schema**

Ex: "\_SYS\_BIC"."BIP.Views/PRODUCT"

**Sample Calculation View:**

****



**Analytic Privilege:**

It is used to control the authorization of data coming out of view at row level.

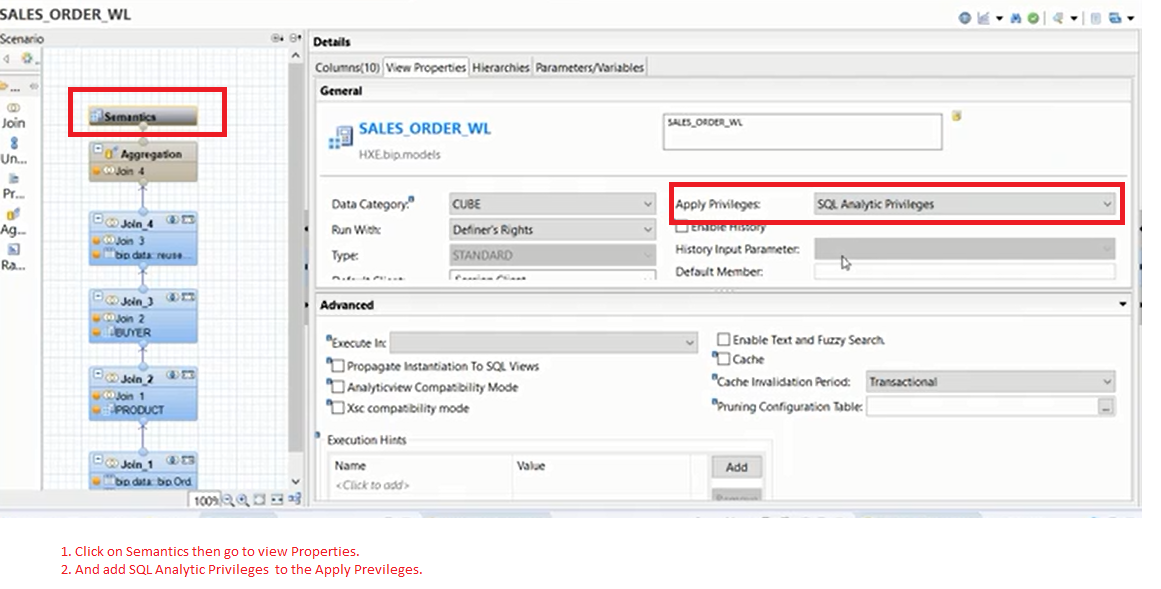
For example sales data for all the regions are combined with in analytical view. However, regional sales managers should only see the data for their region. In this case an analytical privilege can be created and assigned to respective area managers.

* Classical Analytic Privilege – Which is deprecated
* SQL Analytic Privilege

**Fundafox:**

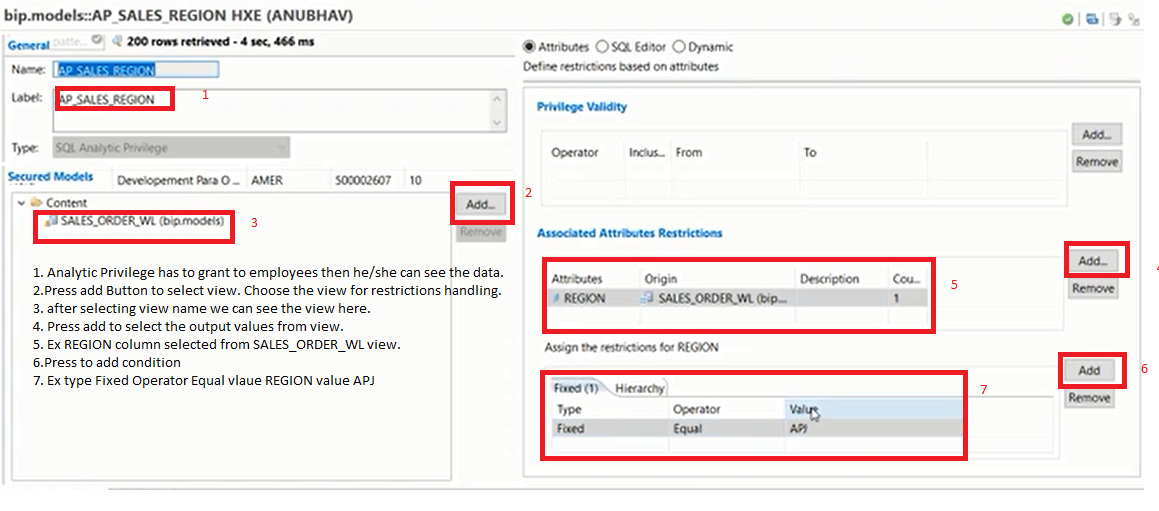
* Measures cannot be used as a filter criteria to be part of analytic privilege.
* Applied on all kinds of models.
* While creating Analytic Privilege check in view semantic -> view Properties -> Apply Privileges.

You can give single, range values, multiple for comparisons.



**SQL Analytic Privilege**

While creating Analytic Privilege check in view semantic -> view Properties -> Apply Privileges.



**Object Privilege:** Authorizations need to give to users

\_SYS\_BI – SELECT, EXECUTE

\_SYS\_BIC – SELECT, EXECUTE

View Object – SELECT

REPOSITORY\_REST – EXECUTE

Analytical privilege for view itself.

Package level: package in which the view was created: \_REPO\_READ

\_SYS\_BI\_CP\_ALL: All the views with all data.

**Fundafox:**

* Go to users -> authorization -> object privileges -> add your privilege

**XSJS Service Creation:**

SAP HANA XS JavaScript (XSJS) is an application programming language in JavaScript. It can be used to exposes data stored in database tables or views to client side. Additionally we can also implement any business logic.  
  
Unlike XSODATA, XSJS is a free flow approach where we can write our own logic using JavaScript.

APIs are one of below categories

* Request processing API - $.request, $.response
* Connectivity API – $client
* Database API - $.db, $.hdb
* Repository API – access HANA repository.

HANA XS can be used for SP08,SP09

* Sending emails $.smtp
* DB lookup
* Schedule job
* Secure Store

**Advantage:**

* XS Layer understand SAP HANA Native Data types. No extra overhead on your application layer for DT conversion.
* Reuse the functions, Reuse standard SAP Functions
* Well integrated with HANA repository and LM.
* No extra programming learning required – JS

XSLIB – XS Library – Collection of XS Functions.

Reuse the program elements.

3rd party libraries.

Perform repetitive tasks using library functions - JSON parsing, Calculations, DT Conversions, Checks, Validations, Parse XML, Url

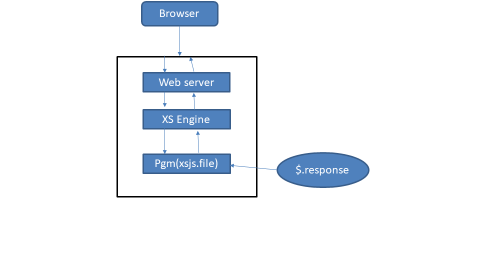
**Syntax:**

$.import(“packagepath”,”libraryname”);

var aliasName = packagepath.libraryname;

aliasName.functionName();

* Browser call xsjs.file
* Browser makes a call to sap hana xs engine
* Call received by web server
* And then call received by xs engine
* And then call received by your program(xsjs.file)
* $.response send to xs engine ->web server->browser.



**XSJS Function:**

**function** multiplication(a,b){

**return** a\*b;

}

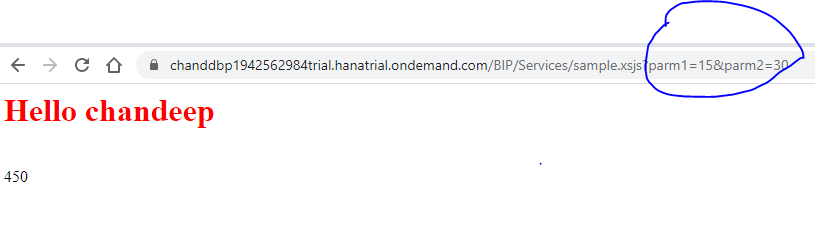
**var** parm1 = $.request.parameters.get("parm1");

**var** parm2 = $.request.parameters.get("parm2");

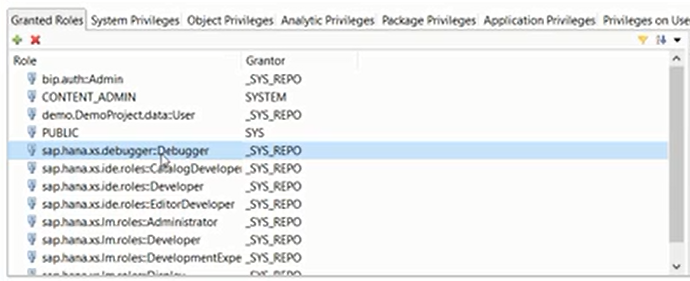
**var** result = multiplication(parm1,parm2);

$.response.setBody("<h1 style='color:red'>Hello chandeep</h1><br>"+result.toString());

$.response.contentType = "text/html";



**Fundafox**: To debug xsjs service we have to add authorization.



**SAP Life cycle Manager:**

SAP LM was introduced with HANA SP07, This allows us to develop complete XS based applications in web based environment using editor.

Also the Lifecycle management tool completely runs in browser, hence there is no need to use any extra client side tools like Eclipse or HANA studio.

<https://host:port/hana/xs/lm>

<https://host:port/sap/hana/ide/editor>

Roles:

sap.hana.xs.lm.roles::Administrator

sap.hana.xs.lm.roles::Developer

sap.hana.xs.lm.roles::DevelopmentExpert

sap.hana.xs.lm.roles::Display

**Disadvantages of using $.db API**

* We need to set the data by it’s data type using set method to the query. While setting data if we have mismatch, it will lead to errors.
* Internal performance issue while working with multiple nodes. It has to has to process the query first inside XS engine and then only sent to DB.
* produced recordset object (rs) is not a JS object. it can only process one record at a time. Processing is also always in linear order. So in case we want to go back to previous record we have to loop again.

when we want to read data from get, the get is type specific and it take integer for column position. This leads to high risk of data type mismatch and column position. You are unable to access any column directly.

**Solution for above disadvantages:**

With SAP HANA SPS09, SAP introduced a new API called $.hdb

1. conn.executeQuery(lv\_query, param1, param2, param3);

The produced Resultset object is JSON. The debugging of this object is possible.

1. Processing Get the connection object

$.hdb.getConnection()

of json is also easy in JS as we can use for loop and index base access.

Example:

// Step1 :- Get connection with hdb

**var** con = $.hdb.getConnection();

// Step2 :- Prepare your query

**var** lv\_query = "select \"PRODUCT\_ID\" , \"S\" from \"BIP\".\"BIP.data::bip.Products\" where \"CATEGORY\" = ?";

// Step3 :- Execute query

**var** rs = con.executeQuery(lv\_query,'Beverages');

// Step4 :- Prepare response

$.response.setBody(JSON.stringify(rs));

**Fundafox**:

* To return multiple values use comma(,) to separate the fields.

**Xlsx download:**

// Step1 :- Get connection with hdb

**var** con = $.hdb.getConnection();

// Step2 :- Prepare your query

**var** lv\_query = "select \"PRODUCT\_ID\" , \"S\" from \"BIP\".\"BIP.data::bip.Products\" where \"CATEGORY\" = ?";

// Step3 :- Execute query

**var** rs = con.executeQuery(lv\_query,'Beverages');

// Step4 :- Prepare response

**var** body='';

**var** item = {};

**var** i =0;

**for** (i = 0; i < rs.length; i++) {

item = rs[i];

body += item.PRODUCT\_ID + "\t" + item.S + "\n";

}

$.response.contentType ="application/vsd.ms-excel; charset-utf-16le" $.response.headers.set("Content-Disposition",'attachment; filename =chandeep.xls');

$.response.setBody(body);

**Advantages of using OData** (Is Open Source technology based on REST principles with HTTP protocol by Microsoft):

1. Compatible for majority of UI technologies like JSP, Silverlight, Android, SAP UI5.
2. We don’t have to write any SQL to process request, The Odata protocol provides all the necessary keywords and generate necessary SQL queries behind the scenes. You save lots of development effort.
3. No worry on handling the request and response yourself.
4. It is also possible in HANA XS starring SP07 to use Odata services for Post.
5. The extension in hana to create odata service is .xsodata. The Odata services can be created on tables, views (so called Entity). We can also define the relationships between the entities using association.

Getting service document

<https://hxehost:4390/bip/odata/Products.xsodata>

Metadata of Odata service

<https://hxehost:4390/bip/odata/Products.xsodata/$metadata>

Get All product Data

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet>

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json>

Restrict number of columns

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$select=PRODUCT_ID,CATEGORY>

Filter data

[https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$filter=CATEGORY%20eq%20%27Beverages%27](https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$filter=CATEGORY%20eq%20'Beverages')

[https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$filter=substringof(%27Pizza%27,CATEGORY)](https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$filter=substringof('Pizza',CATEGORY))

Number of records

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet/$count>

**Paging**

Chunk-by-chunk

Initially 20 records

On scroll load next 20

On scroll load next 20

[Paging](https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$top=2&$skip=4)

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet?$format=json&$top=2&$skip=4>

Load single record based on key

<https://hxehost:4390/bip/odata/Products.xsodata/ProductSet('42F2E9AFC4EF1EE88CD93DF6D99B5139')>

**XSODATA** **Ex:**

service {

"BIP"."BIP.data::bip.Products" as "PoductSet"

navigates("Asso\_Text" as "Text" , "Asso\_Supplier" as "Supplier") ;

"BIP"."BIP.data::bip.BusinessPartners" as "BPSet";

"BIP"."BIP.data::bip.ProductText" as "TextSet";

association "Asso\_Text" principal "PoductSet"("NODE\_KEY") multiplicity "1"

dependent "TextSet"("PARENT\_KEY") multiplicity "1";

association "Asso\_Supplier" principal "PoductSet"("SUPPLIER\_ID.NODE\_KEY") multiplicity "1"

dependent "BPSet"("NODE\_KEY") multiplicity "1";

}

**WHY CDS?**

Not so convenient to access Odata:

1. Complexity of Association can be very high.
2. NODE\_KEY / Technical Guids are displayed to consumer, which are not of much use.
3. When it comes to selecting columns, we have to use $select which is an extra effort on consumer.

Security of Data. Because you are directly calling DB tables.

**DROP CDS View Syntax:**

**DROP VIEW** "BIP"."BIP.data::CDSView.MyView" – Path of your cds view