Week 3 Unit 3: Create an SQLScript Procedure with Calculation Engine (CE) Functions
Create an SQLScript Procedure with CE Functions
Calculation Engine

The calculation engine is the execution engine for SQLScript.

SQLScript statements are parsed into the calculation model as much as possible. The calculation engine instantiates a calculation model at the time of query execution.
Create an SQLScript Procedure with CE Functions
Different Engines, Different Responsibilities

- **Calc views**
  - SQL Optimizer
  - Calculation engine
  - OLAP engine
  - Join engine

- **Attribute views**
  - Row store engine

- **Analytic views**
  - Column store

- **Row store**
BEGIN

    po = CE_CALC_VIEW("EPM_PROCUREMENT",
    ["NET_AMOUNT", "COUNTRY"
    "CREATION_DATE"]);  ----- Query 1

    c_time_dimension = CE_COLUMN_TABLE
    ("_SYS_BI", ".M_TIME_DIMENSION");  ---- Query 2

    c_time = CE_PROJECTION(:c_time_dimension,
    ["DATE_SQL" AS
    "CREATION_DATE", "QUARTER"]);  ---- Query 3

    result = CE_JOIN(:po, :c_time, ["CREATION_DATE"]) :  ---- Query 4

    OUTPUT = select COUNTRY, QUARTER,
    SUM(NET_AMOUNT) as procurement_amount FROM :result
    GROUP BY COUNTRY, quarter;  ---- Query 5

END;
Create an SQLScript Procedure with CE Functions
Calculation Engine Plan Operators

The calculation engine plan operators encapsulate data transformation functionality.

Alternatively, you can use SQL statements; their logic is directly implemented in the calculation engine or the execution environment of SQLScript.

Operators are categorized as Data Source Access and Relational.
Create an SQLScript Procedure with CE Functions
Semantic Equivalents CE/SQL 1

**Example:**

```sql
ot_books1 = CE_COLUMN_TABLE("BOOKS");
ot_books2 = CE_COLUMN_TABLE("BOOKS", ["TITLE", "PRICE", "CRCY"]);
```

This example only works on a column table and does not invoke the SQL processor. It is semantically equivalent to the following:

```sql
ot_books3 = SELECT * FROM books;
```

```sql
ot_books4 = SELECT titel, price, crcy FROM books;
```
Create an SQLScript Procedure with CE Functions
Semantic Equivalents CE/SQL 3

CE_Join (:var1_table, :var2_table, [join_attr, ...]{, [attrib_name, ...]})

Example:

ot_pubs_books1 = CE_Join (:It_pubs, :it_books,"PUBLISHER");
ot_pubs_books2 = CE_Join (:It_pubs, :it_books,"PUBLISHER"),
"TITLE","NAME","PUBLISHER","YEAR");

This example is semantically equivalent to the following SQL procedure,
but does not invoke the SQL processor.

ot_pubs_books3 = SELECT P.publisher AS publisher, name, street,
    post_code, city, country, isbn, title, edition, year, price, crcy
    FROM :It_pubs AS P, :it_books AS B
    WHERE P.publisher = B.publisher;

ot_pubs_books4 = SELECT title, name, P.publisher AS publisher, year
    FROM :It_pubs AS P, :it_books AS B
    WHERE P.publisher = B.publisher;
Create an SQLScript Procedure with CE Functions
Semantic Equivalents CE/SQL 4

CE_PROJECTION (:var_table, [param_name [AS new_param_name],…]{,[Filter]})

Example:

```
ot_books1 = CE_PROJECTION (:it_books,["TITLE","PRICE", "CRCY" AS "CURRENCY"], "PRICE" > 50);
```

Semantically equivalent to the following SQL procedure:

```
ot_books2 = SELECT titel, price, crcy AS currency
            FROM :it_books WHERE price > 50;
```

h. Next, use the CE_PROJECTION to filter the data. The below statement will further filter the data based on the PARTNERROLE column and the input parameter IM_PARTNERROLE. Because the names of the fields in the database table are mixed case, and the names of those fields in the output parameter are all uppercase, you must use aliases to map the fields. Enter the following statement.

```
lt_bp_proj = CE_PROJECTION (:lt-bp, 
    ["PartnerId", "PartnerRole", "EmailAddress", "CompanyName", "AddressID" ], 
    "PartnerRole" = :partnerrole' );
```
# SQLScript
## CE (Calculation Engine) Built-In Functions

<table>
<thead>
<tr>
<th>SQL</th>
<th>CE Built-In Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SELECT on column table</strong>&lt;br&gt;out = SELECT A, B, C from &quot;COLUMN_TABLE&quot;&lt;br&gt;out = CE_COLUMN_TABLE(&quot;COLUMN_TABLE&quot;, [A, B, C])</td>
<td></td>
</tr>
<tr>
<td><strong>SELECT on attribute view</strong>&lt;br&gt;out = SELECT A, B, C from &quot;ATTRIBUTE_VIEW&quot;&lt;br&gt;out = CE_JOIN_VIEW(&quot;ATTRIBUTE_VIEW&quot;, [A, B, C])</td>
<td></td>
</tr>
<tr>
<td><strong>SELECT on olap view</strong>&lt;br&gt;out = SELECT A, B, C, SUM(D) from &quot;ANALYTIC_VIEW&quot; GROUP BY A, B, C&lt;br&gt;out = CE_OLAP_VIEW(&quot;ANALYTIC_VIEW&quot;, [A, B, C]);</td>
<td></td>
</tr>
<tr>
<td><strong>WHERE HAVING</strong>&lt;br&gt;out = SELECT A, B, C, SUM(D) from &quot;ANALYTIC_VIEW&quot; WHERE B = 'value' AND C = 'value'&lt;br&gt;col_tab= CE_COLUMN_TABLE(&quot;COLUMN_TABLE&quot;); out = CE_PROJECTION(col_tab, [A, B, C], ' &quot;B&quot; = ''value'' AND &quot;C&quot; = ''value'' ');</td>
<td></td>
</tr>
<tr>
<td><strong>GROUP BY</strong>&lt;br&gt;out = SELECT A, B, C, SUM(D) FROM&quot;COLUMN_TABLE&quot; GROUP BY A, B, C&lt;br&gt;col_tab= CE_COLUMN_TABLE(&quot;COLUMN_TABLE&quot;); out = CE_AGGREGATION( (col_tab, SUM(D), [A, B, C]);</td>
<td></td>
</tr>
<tr>
<td><strong>INNER JOIN</strong>&lt;br&gt;out = SELECT A, B, Y, SUM(D) from &quot;COLTAB1&quot; INNER JOIN &quot;COLTAB2&quot; WHERE &quot;COLTAB1&quot;.&quot;KEY1&quot; = &quot;COLTAB2&quot;.&quot;KEY1&quot; AND &quot;COLTAB1&quot;.&quot;KEY2&quot; = &quot;COLTAB2&quot;.&quot;KEY2&quot;&lt;br&gt;out = CE_JOIN(&quot;COLTAB1&quot;,&quot;COLTAB2&quot;, [KEY1, KEY2], [A, B, Y, D])</td>
<td></td>
</tr>
<tr>
<td><strong>LEFT OUTER JOIN</strong>&lt;br&gt;out = SELECT A, B, Y, SUM(D) from &quot;COLTAB1&quot; LEFT OUTER JOIN &quot;COLTAB2&quot; WHERE &quot;COLTAB1&quot;.&quot;KEY1&quot; = &quot;COLTAB2&quot;.&quot;KEY1&quot; AND &quot;COLTAB1&quot;.&quot;KEY2&quot; = &quot;COLTAB2&quot;.&quot;KEY2&quot;&lt;br&gt;out = CE_LEFT_OUTER_JOIN(&quot;COLTAB1&quot;,&quot;COLTAB2&quot;, [KEY1, KEY2], [A, B, Y, D])</td>
<td></td>
</tr>
<tr>
<td><strong>SQL Expressions</strong>&lt;br&gt;out = SELECT A, B, C, SUBSTRING(D,2,5) FROM &quot;COLUMN_TABLE&quot;&lt;br&gt;proj_tab = CE_COLUMN_TABLE(&quot;COLUMN_TABLE&quot;); out = CE_PROJECTION( :proj_tab, [&quot;A&quot;, &quot;B&quot;, &quot;C&quot;, CE_CALC('midstr(&quot;D&quot;,2,5)', string) ]);</td>
<td></td>
</tr>
<tr>
<td><strong>UNION ALL</strong>&lt;br&gt;col_tab1 = SELECT A, B, C, D FROM &quot;COLUMN_TABLE1&quot;; col_tab2 = SELECT A, B, C, D FROM &quot;COLUMN_TABLE2&quot;; out = SELECT * FROM :col_tab1 UNION ALL SELECT * FROM :col_tab2;&lt;br&gt;col_tab1 = CE_COLUMN_TABLE(&quot;COLUMN_TABLE1&quot;,[A,B,C,D]); col_tab2 = CE_COLUMN_TABLE(&quot;COLUMN_TABLE2&quot;,[A,B,C,D]); out = CE_UNION_ALL(:col_tab1,:col_tab2);</td>
<td></td>
</tr>
</tbody>
</table>
Thank you

Contact information:

open@sap.com
© 2013 SAP AG. 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 AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

National product specifications may vary.

These materials are provided by SAP AG and its affiliated companies (“SAP Group”) for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group 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.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark for additional trademark information and notices.