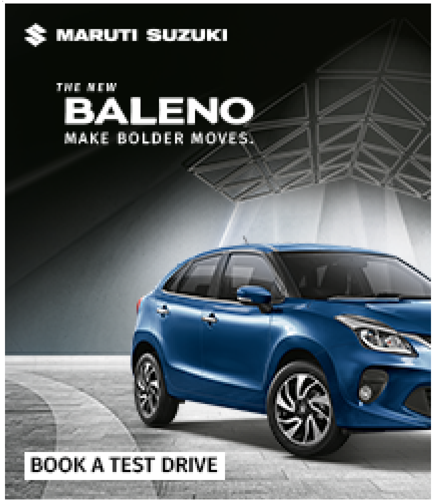
SAP HANA Central

Thursday, 11 August 2016

ABAP on HANA - Use Cases



Introduction to ABAP on HANA

Through HANA, SAP has brought forth a high performing multi faceted appliance with rich analytic computational capabilities, in-memory hardware, enhanced compression technology, geospatial capabilities, Text analytics and predictive analytics, to name a few. With so powerful a back-end, the application layer too had to be revised to fully leverage the enriched capabilities of HANA. CDS Views, AMDPs, and enhancements to existing Open SQL are the various available solutions which help in achieving Code Push Down – transfer the data intensive logic to the

Total Pageviews

757,336

SAP HANA Certification Material

- SAP HANA Certification Material
- SAP HANA Training Videos

database resulting in better performance. AS 7.4 or above is the required version of application layer on a HANA database for the features mentioned throughout the document to work.

Introduction to CDS Views:

CDS is one of the advanced features introduced by SAP in ABAP release 7.4 SP5. Through CDS, one can define views in the ABAP repository from ABAP Development Tools (ADT) using SQL DDL syntax. With an advanced feature set and domain specific annotations made available, CDS offers a simple solution for to realize code push down from ABAP Layer.

Introduction to AMDP:

ABAP managed database procedure is a simple ABAP class method containing database-specific procedure coding. The code within the method is pushed to the database layer and executed within the database. Thus significant performance improvements of data-intensive processes can be achieved by code push-down from the application server to the database server. This reduces the number of data transfers and the amount of transferred data between both servers.

Introduction to Open SQL Enhancements:

Open SQL has now been enhanced with certain features and made close to SQL-92 standard such that database level computations and code push down is achieved with ease. This is done through Group functions, arithmetic, string & conditional (CASE) expressions in Open-SQL Construct.

CDS Views in detail

To fully leverage the high performing capabilities of SAP HANA for application development, SAP has introduced a new mechanism to carry

Labels

database developer Dynamic-Tiering hana-academy hana-api hana-backup out data modeling, known as core data services (CDS). With CDS, data models are defined from the application server but are executed on the database resulting in an optimal code push down. CDS also offers capabilities beyond the traditional data modeling tools, including support for conceptual modeling and relationship definitions, built-in functions, and extensions. Originally, CDS was available only in the design-time and runtime environment of SAP HANA. Now, the CDS concept is also fully implemented in SAP NetWeaver AS ABAP, enabling developers to work in the ABAP layer with ABAP development tools while the code execution is pushed down to the database.

What is CDS?

CDS is an SAP framework for defining semantically rich data models, called CDS views. It is defined using SQL-based data definition language (DDL) which is based on standard SQL with advanced features like associations and annotations. CDS goes beyond the traditional querying capabilities by achieving complex calculations & manipulations with ease.
CDS views are quite similar to the ABAP repository views defined using ABAP layer. However, CDS allows defining basic views to achieve analytics and aggregations and later extend them to create much more complex artifacts making it more feasible with the complex business environment.
While creating a CDS view from ABAP layer, a corresponding DB view is created in HANA with the name provided in the CDS editor. This DB view can be accessed just like any other information model in HANA database.
Also, CDS offers combining result set of two or more queries into a single result set using UNION operator, ability to switch on automatic client handling using the annotation @ClientDependent:true. Also, it enables handling buffering of CDS entities with ease.

CDS was initially designed for native SAP HANA application development. With AS 7.4 SPS 05, the CDS concept was later made available in ABAP. While HANA based CDS runs only on SAP HANA database, ABAP based CDS

in-memory modeling mysql odata odbc predictive sap SAP HANA sap-hana sap-hana-cloud-platform sap-hana-database sap-hana-usecases sap-hana-xs sap-hana2 sap-xs SAPHANA sapmentor sapmentors sapui5 sda spark sps11 sps12 sql statistics xs xsjs

Search This Blog

Search

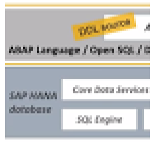
Like Our Official Page

SAP HANA Central

Like Page

Be the first of your friends to like this

Popular Posts



Core Data Services in ABAP

Core Data services (CDS) are domain specific languages (DSL) and services for defining and consuming semantically rich data models in SAP ...

Business Partner in S4 HANA – Customer Vendor Integration runs on any database which is certified by SAP.

Advantages of CDS Views:

1. Reusability of database artifact.
2. Large Feature Set – Example: Associations.
3. Domain Specific Consumption of Data models – Example: Annotations.
4. Client handling can be achieved in CDS.

Limitations of CDS Views:

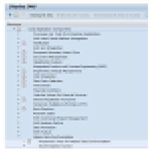
- Only 1 result set can be returned from a CDS View.
- Only 1 independent process can be carried out at 1 time. (1 query or a union/join of related queries)

AMDPs in Detail

With the dynamic in-memory capabilities of HANA built in with advanced hardware, calculations are extremely faster when carried out at DB level. Add to this, the parallel execution of multiple queries in one go. Using SQL and all of its possibilities is one example of this. But this has limitations. A SQL statement always has one result set and the calculation must be done in one single step. If you have complex calculations which you want to execute on the database you need other possibilities. AMDPs provide a framework for effectively taking advantage of both code push down and utilize parallel execution feature of HANA.

What are AMDPs?

ABAP Managed Database Procedures are a new feature in AS ABAP allowing developers to write database procedures directly in ABAP. The implementation language varies from one database system to another. In



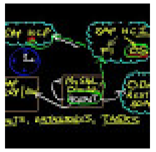
It's been almost 2 years I have started my journey in SAP world. I started with Technical domain for some time and I landed in Functional d...

SAP PDMS 1.0 FP02 On-premise edition 1.0 (Configuration – Pt.2)

In the first part of my documentation (pt.1) , I have explained and showed in detail how to proceed with the mandatory landscape and compo...

Expose CDS Views as OData Service

In continuation to previous blog Core Data Services in ABAP in this blog I will show how to create CDS Views and step by step procedure to...



[SAP HANA Academy] SAP HANA Cloud Integration for Data Services

In a series of 14 videos the SAP HANA Academy's Tahir Hussain "Bob" Babar shows how to preform the most common SAP HANA CI...

ABAP on HANA - Use Cases

Introduction to ABAP on HANA Through HANA, SAP has

SAP HANA it is SQL Script. Using AMDP allows developers to create and execute those database procedures in the ABAP environment using ABAP methods and ABAP data types. AMDP method is a part of a general ABAP class just like any other class defined in the class builder. However, an AMDP method needs to be defined with the standard interface – IF_AMDP_MARKER_HDB to let the compiler know that it's an AMDP method and therefore code execution must be pushed to the database.

AMDPs allow developers to create and manage stored procedures directly in ABAP, while they are executed on the database level. AMDPs need to be defined from ADT in HANA studio using the general class builder as done using GUI. Implementing methods within the interface implies the method being an AMDP and therefore be executed at the database. Any method defined within the same class but not within the above interface is treated as a normal ABAP method and is executed at the application layer. Developing and implementing of AMDPs is very much the same as any method belong to a global class.

AMDPs need to be implemented in SQLSCRIPT which is the native language for HANA DB. This helps in implementing various powerful features of SQLScript which are not yet available in ABAP. By strictly following the guidelines and golden rules suggested in SQLSCRIPT, the performance of an application can be optimized, taking full advantage of the advanced capabilities of HANA .

Advantages of AMDPs

1. Complex calculations involving several independent steps/processes can be performed in one AMDP method.
2. Ease of implementation in ABAP.
3. Powerful features of native SQL such as currency conversion and CE functions can be leveraged.

Disdvantages of AMDPs:

brought forth a high performing multi faceted appliance with rich analytic computatio...

How to Prepare for C_HANAIMP_12 Certification on SAP HANAIMP 12

All that you need to know about SAP C_HANAIMP_12 SAP launched HANAIMP 12 primarily based on in-memory expertise with which customers ca...

Complete End-To-End ABAP For HANA 7.4 SP 09 Development

Steps Involved For Displaying Flight Details: Creation of ABAP CDS view. Creation of Gateway O Data service. Testing of GW service in G...

Featured post

SAP S/4HANA from a Developers Point of View - Part One

Introduction In my first part I try to explain what is the main difference between R/3, SAP ERP and S/4. As you can see from the pi...

1. No automatic client handling
2. Only Exporting, importing and changing parameters are allowed
3. Methods with returning parameters cannot be implemented as AMDPs
4. Method parameters have to be tables or scalar types
5. Method parameters have to be passed as values

Open SQL Enhancements

In order to achieve code push down from application layer, enhancements have been made to existing SQL in terms of expressions & group functions. Along with these, the SQL syntax has been modified to enable dynamic data declaration, escaping of host variables etc thus making it more robust.

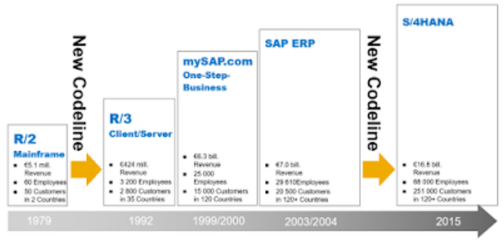
What are Open SQL Enhancements?

To understand the various Open SQL enhancements made to achieve code push down, it is important to understand the limitations of SQL before 7.4. Below is a list of various limitations of SQL before:

1. Limited Join Types available
2. No Expressions in select
3. No Union & Union all

To address above the above shortcomings and also to further enhance the ABAP capabilities, Open SQL has been made more advanced and robust. Also these enhancements are made keeping HANA in mind such that all advantages offered by HANA are leveraged.
Following are the elements in the New Open SQL syntax:

a. Escaping of host variables with “@”.
b. Right Outer Join made available.
c. Comma separated element list in selects.
d. SQL expressions – Arithmetic expressions, CASE expressions,



Search This Blog

Search

Blog Archive

▶ 2019 (31)
▶ 2018 (154)
▶ 2017 (206)
▼ 2016 (252)
  ▶ December (16)
  ▶ November (17)
  ▶ October (17)
  ▶ September (22)
  ▼ August (23)
    SAP HANA XS Advanced Installation through resident...
    Data Loading to HANA using DXC Connection
    Vora 1.2 Modeling Tool
    New Hierarchy SQL enablement with Calculation View...
    ALV and FPM on SAP HANA

COALESCE etc.

e. Aggregate functions – SUM, MAX, MIN, COUNT etc.

Following are the various diverse expressions in SQL selects:

i. Arithmetical expressions: + | - | * | DIV | MOD | ABS | FLOOR |CEIL
ii. Concatenation of character columns with &&
iii. Conditional expression with CASE
iv. Casting with CAST
v. Coalesce

Using the Advanced Open SQL judiciously allows in achieving pushing down the complex calculative and manipulative logic to the database and thereby avoids unnecessary loops in ABAP code. Owing to speed performing capabilities of HANA in handling huge volumes of data and enriched aggregation abilities, this would result in boosting the performance by considerable amount of time.

Advantages of Open SQL Enhancements:

ABAP code will remain database agnostic and run on any database in a similar fashion.
You implicitly take advantage of all optimizations that have been achieved at the database level.
All default performance optimizations like use of buffer and client handling are automatically taken care of.

Disadvantages of Open SQL Enhancements:

1. Statements are compiled in strict mode making it necessary that entire query be abided by the new syntax.
2. Only simpler calculations could be performed in Open SQL. For complex aggregations, CDS or AMDPs needs to be used.
3. Only single result set could be returned.

XS application for table distribution in scale out...

HOW TO GENERATE ROW NUMBER OR SEQUENCE NUMBER USI...

Licensing, Sizing and Architecting BW on HANA

First Steps of Code Quality Analysis for SAP HANA ...

How to Define Role to different Server Nodes in Mu...

Connecting SAP HANA Views to Sensor Data from Osis...

How and why to activate asynchronous queuing of IO...

How to use HDBAdmin to analyze performance traces ...

ABAP on HANA - Use Cases

How to Trouble Shoot Statistic server migration Fa...

How to connect Microsoft SSIS with SAP HANA

HANA TA for Hybris Ecommerce - Why Google??

Generate Time Data in SAP HANA - Part 2

Generate Time Data in SAP HANA - Part 1

Expose Attribute Views as XS OData Service in HANA...

IDA Classes in detail:

What are IDA Classes?

The implementation of in-memory features of a database can lead to significant improvements in processing of large quantities of data. Also, less wait time for data display and much faster interactive operations and aggregations enables smooth processing of business data with ease. To make the great advantages usable for business applications in the ALV environment as well, SAP offers a special version of the List Viewer, the SAP List Viewer with Integrated Data Access. SAP has used the concept of persistence storage and paging to reduce the wait time for the list display. Also, additional operations on the output such as aggregations, filtering, sorting etc is made more reflexive and easy to implement.

Advantages of IDA Classes

1. Leverage in-memory capabilities of HANA, without having to write complex code to realize this.
2. Minimal amount of data transfer – No explicit select query is required
3. End users can continue to work on the familiar ALV interface. The standard functions (also ALV services) and toolbar options are still available in ALV with IDA.
4. Advanced UI features like Sorting, filtering, aggregations etc. are available.

Use Cases

Use Case 1:

The Equity Group Changes for JOA is an existing program which gets the Equity Group Change History based on the selection screen entries and displays the changes log in an ALV. The changes are primarily related to

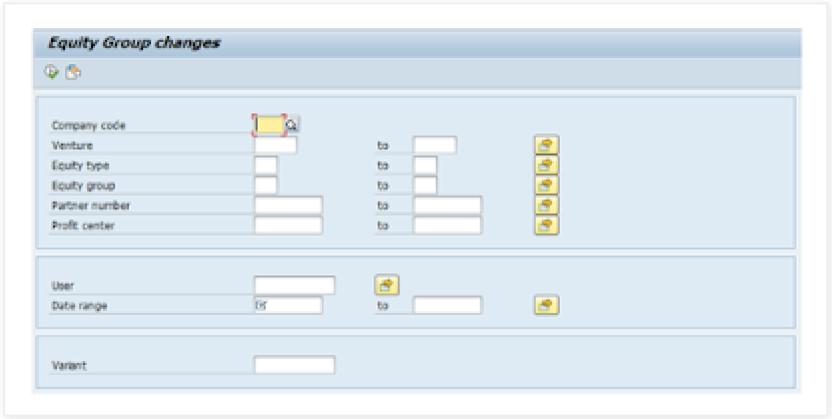
Subscribe To SAP HANA Central

Posts

Comments

change in partners (Addition/Modification/Deletion), their equity shares and changes in non-operated share. The program was reported to have performance issues for huge volumes of data (around 10 Million records and (8+ hours) on execution, eventually timing out even in background run and had also dump reported due to memory issues as the user was giving a big date range.

Selection Screen:



Equity Group changes

Company code			
Venture		to	
Equity type		to	
Equity group		to	
Partner number		to	
Profit center		to	
User			
Date range		to	
Variant			

Analysis:

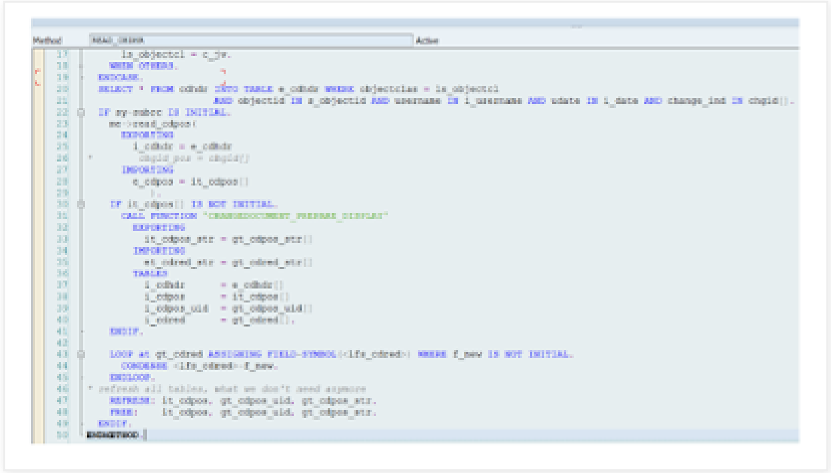
1. Most of the high data sensitive queries were done at the application layer.
2. The program uses "select *" queries to pick up the changed documents from CDHDR & CDPOS.
3. A standard Function Module CHANGEDOCUMENT_PREPARE_DISPLAY was used for formatting.
4. There were many other series of "select *" queries on several tables based on changed data saving the data in to respective internal tables that were used to fulfill the requirement.

Levaraging HANA:

1. For points 1, 2 & 3 above, we have developed a CDS View and consume the same in the report program.
2. For point 1 & 4, we have created an AMDP method and called this in the report program.

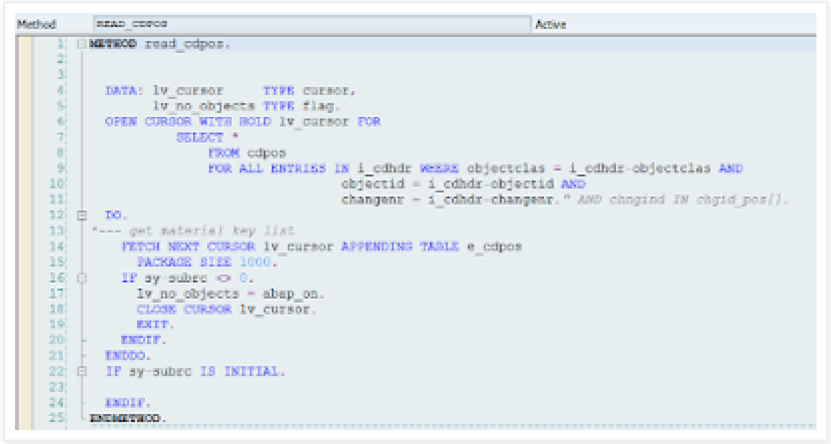
Relevant Code Snippets:

Existing Code:



Method READ_CDHDR Active

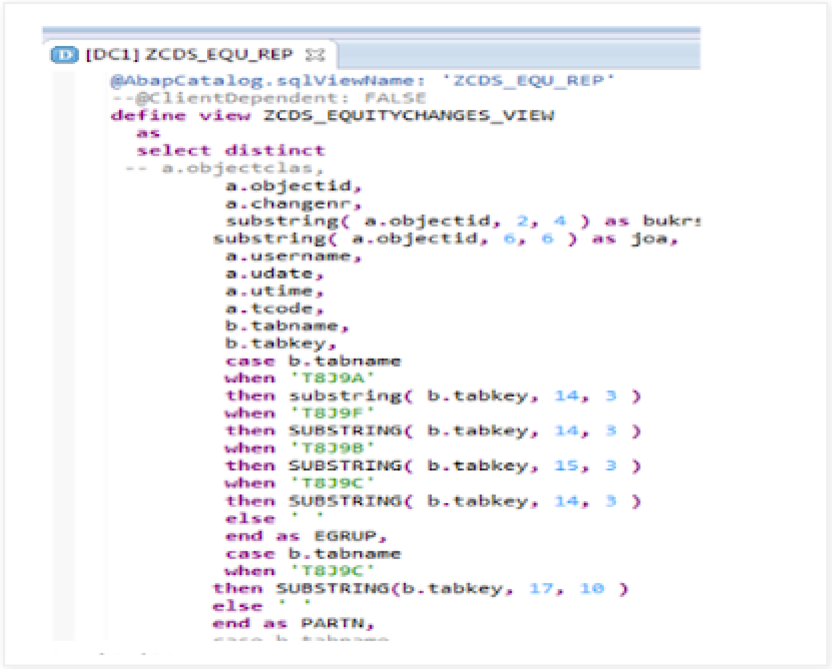
```
        ls_objectcl = c_jv.
      WHEN OTHERS.
    ENDCASE.
    SELECT * FROM cdhdr INTO TABLE e_cdhdr WHERE objectclas = ls_objectcl
                        AND objectid IN s_objectid AND username IN i_username AND udate IN i_date AND change_ind IN chgid[].
    IF sy-subrc IS INITIAL.
      me->read_cdpos(
        EXPORTING
          i_cdhdr = e_cdhdr
*           chgid_pos = chgid[]
        IMPORTING
          e_cdpos = it_cdpos[]
             ).
      IF it_cdpos[] IS NOT INITIAL.
        CALL FUNCTION 'CHANGEDOCUMENT_PREPARE_DISPLAY'
          EXPORTING
            it_cdpos_str = gt_cdpos_str[]
          IMPORTING
            et_cdred_str = gt_cdred_str[]
          TABLES
            i_cdhdr       = e_cdhdr[]
            i_cdpos       = it_cdpos[]
            i_cdpos_uid   = gt_cdpos_uid[]
            i_cdred       = gt_cdred[].
      ENDIF.

      LOOP at gt_cdred ASSIGNING FIELD-SYMBOL(<lfs_cdred>) WHERE f_new IS NOT INITIAL.
        CONDENSE <lfs_cdred>-f_new.
      ENDLOOP.
* refresh all tables, what we don't need anymore
      REFRESH: it_cdpos, gt_cdpos_uid, gt_cdpos_str.
      FREE:    it_cdpos, gt_cdpos_uid, gt_cdpos_str.
    ENDIF.
ENDMETHOD.
```



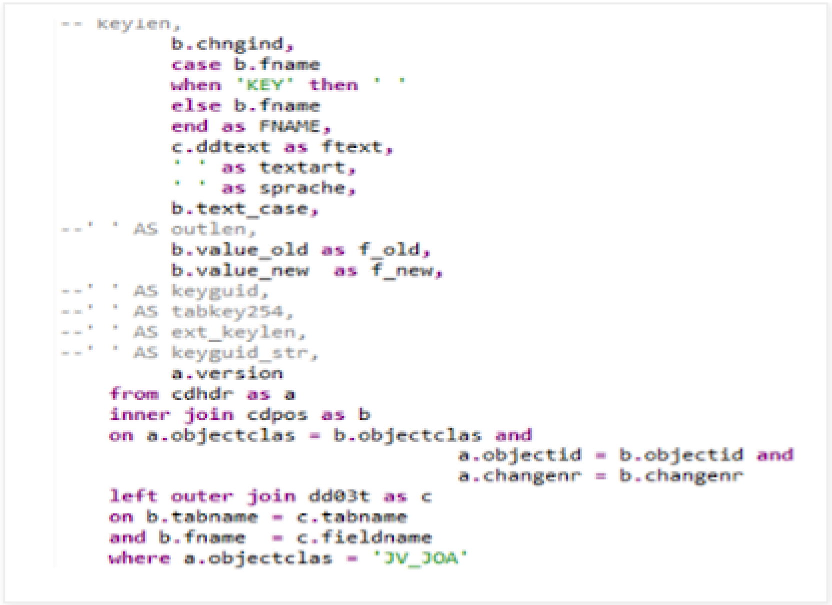
Method READ_CDPOS Active

```
METHOD read_cdpos.


  DATA: lv_cursor     TYPE cursor,
        lv_no_objects TYPE flag.
  OPEN CURSOR WITH HOLD lv_cursor FOR
           SELECT *
               FROM cdpos
               FOR ALL ENTRIES IN i_cdhdr WHERE objectclas = i_cdhdr-objectclas AND
                                objectid = i_cdhdr-objectid AND
                                changenr = i_cdhdr-changenr." AND chngind IN chgid_pos[].
  DO.
*--- get material key list
    FETCH NEXT CURSOR lv_cursor APPENDING TABLE e_cdpos
      PACKAGE SIZE 1000.
    IF sy-subrc <> 0.
      lv_no_objects = abap_on.
      CLOSE CURSOR lv_cursor.
      EXIT.
    ENDIF.
  ENDDO.
  IF sy-subrc IS INITIAL.

  ENDIF.
ENDMETHOD.
```

Changed Code:
CDS Definition:

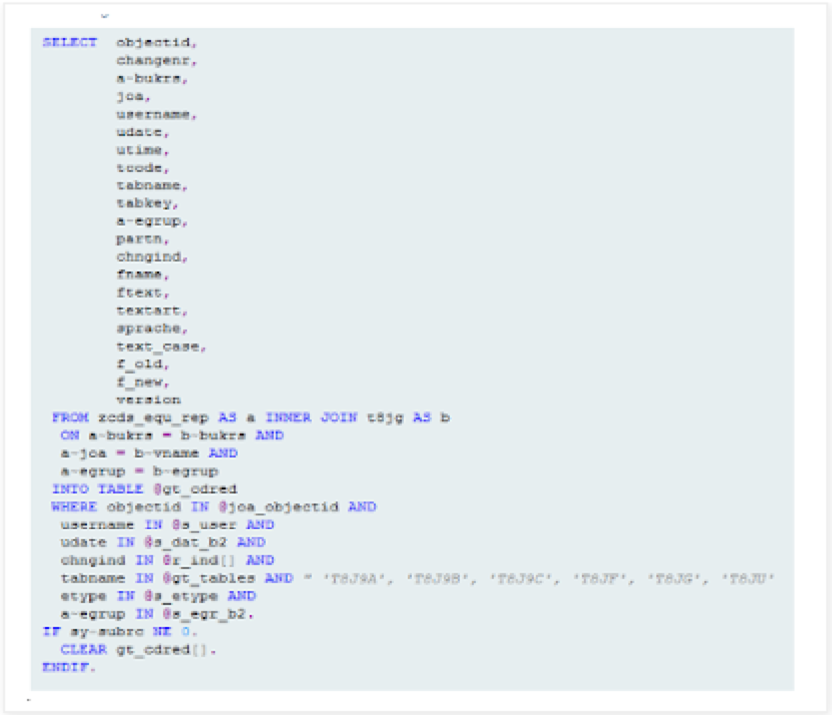


```
@AbapCatalog.sqlViewName: 'ZCDS_EQU_REP'
--@ClientDependent: FALSE
define view ZCDS_EQUITYCHANGES_VIEW
  as
  select distinct
 -- a.objectclas,
         a.objectid,
         a.changenr,
         substring( a.objectid, 2, 4 ) as bukrs
        substring( a.objectid, 6, 6 ) as joa,
         a.username,
         a.udate,
         a.utime,
         a.tcode,
         b.tabname,
         b.tabkey,
         case b.tabname
         when 'T8J9A'
         then substring( b.tabkey, 14, 3 )
         when 'T8J9F'
         then SUBSTRING( b.tabkey, 14, 3 )
         when 'T8J9B'
         then SUBSTRING( b.tabkey, 15, 3 )
         when 'T8J9C'
         then SUBSTRING( b.tabkey, 14, 3 )
         else ' '
         end as EGRUP,
         case b.tabname
         when 'T8J9C'
        then SUBSTRING(b.tabkey, 17, 10 )
        else ' '
        end as PARTN,
```



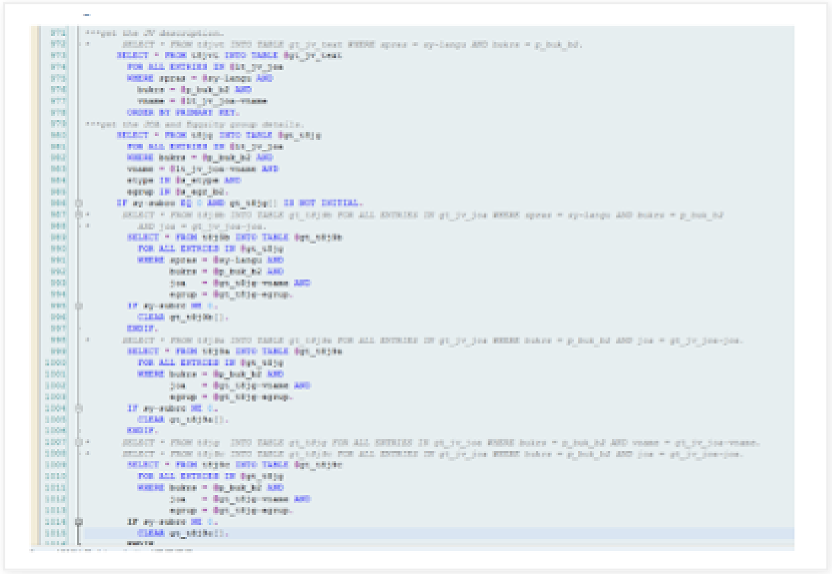
```
-- keylen,
         b.chngind,
         case b.fname
         when 'KEY' then ' '
         else b.fname
         end as FNAME,
         c.ddtext as ftext,
         ' ' as textart,
         ' ' as sprache,
         b.text_case,
--' ' AS outlen,
         b.value_old as f_old,
         b.value_new  as f_new,
--' ' AS keyguid,
--' ' AS tabkey254,
--' ' AS ext_keylen,
--' ' AS keyguid_str,
         a.version
    from cdhdr as a
    inner join cdpos as b
    on a.objectclas = b.objectclas and
                            a.objectid = b.objectid and
                            a.changenr = b.changenr
    left outer join dd03t as c
    on b.tabname = c.tabname
    and b.fname  = c.fieldname
    where a.objectclas = 'JV_JOA'
```

Consuming CDS in ABAP:



```
SELECT  objectid,
        changenr,
        a~bukrs,
        joa,
        username,
        udate,
        utime,
        tcode,
        tabname,
        tabkey,
        a~egrup,
        partn,
        chngind,
        fname,
        ftext,
        textart,
        sprache,
        text_case,
        f_old,
        f_new,
        version
 FROM zcds_equ_rep AS a INNER JOIN t8jg AS b
  ON a~bukrs = b~bukrs AND
  a~joa = b~vname AND
  a~egrup = b~egrup
 INTO TABLE @gt_cdred
 WHERE objectid IN @joa_objectid AND
  username IN @s_user AND
  udate IN @s_dat_b2 AND
  chngind IN @r_ind[] AND
  tabname IN @gt_tables AND " 'T8J9A', 'T8J9B', 'T8J9C', 'T8JF', 'T8JG', 'T8JU'
  etype IN @s_etype AND
  a~egrup IN @s_egr_b2.
IF sy-subrc NE 0.
  CLEAR gt_cdred[].
ENDIF.
```

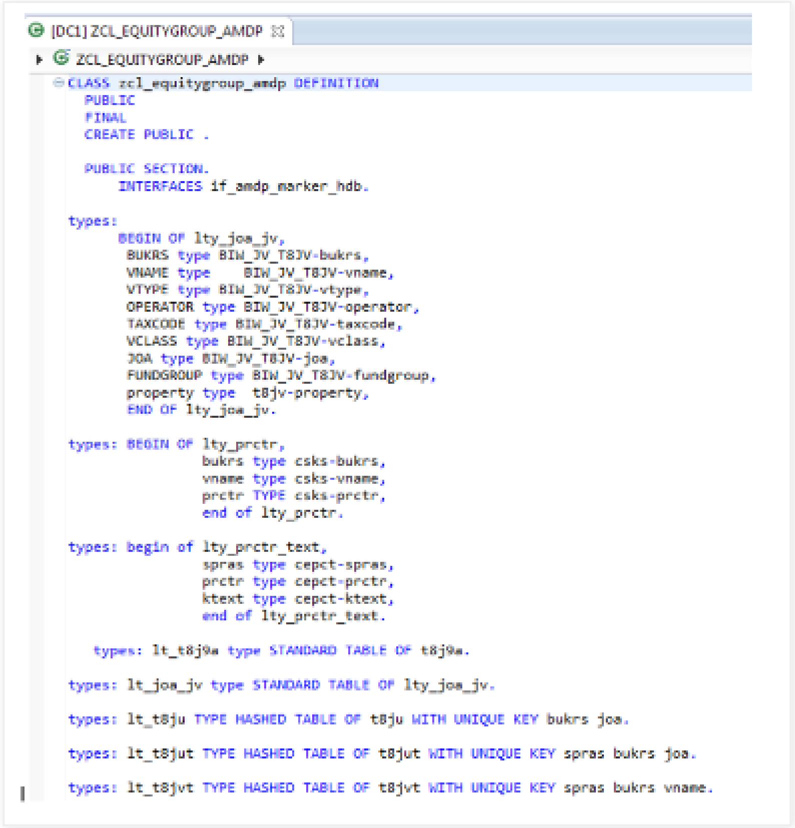
Existing Code:



```
***get the JV description.
*       SELECT * FROM t8jvt INTO TABLE gt_jv_text WHERE spras = sy-langu AND bukrs = p_buk_b2.
      SELECT * FROM t8jvt INTO TABLE @gt_jv_text
        FOR ALL ENTRIES IN @lt_jv_joa
        WHERE spras = @sy-langu AND
          bukrs = @p_buk_b2 AND
          vname = @lt_jv_joa-vname
        ORDER BY PRIMARY KEY.
***get the JOA and Equity group details.
      SELECT * FROM t8jg INTO TABLE @gt_t8jg
        FOR ALL ENTRIES IN @lt_jv_joa
        WHERE bukrs = @p_buk_b2 AND
        vname = @lt_jv_joa-vname AND
        etype IN @s_etype AND
        egrup IN @s_egr_b2.
      IF sy-subrc EQ 0 AND gt_t8jg[] IS NOT INITIAL.
*       SELECT * FROM t8j9b INTO TABLE gt_t8j9b FOR ALL ENTRIES IN gt_jv_joa WHERE spras = sy-langu AND bukrs = p_buk_b2
*          AND joa = gt_jv_joa-joa.
        SELECT * FROM t8j9b INTO TABLE @gt_t8j9b
          FOR ALL ENTRIES IN @gt_t8jg
          WHERE spras = @sy-langu AND
                bukrs = @p_buk_b2 AND
                joa   = @gt_t8jg-vname AND
                egrup = @gt_t8jg-egrup.
        IF sy-subrc NE 0.
          CLEAR gt_t8j9b[].
        ENDIF.
*       SELECT * FROM t8j9a INTO TABLE gt_t8j9a FOR ALL ENTRIES IN gt_jv_joa WHERE bukrs = p_buk_b2 AND joa = gt_jv_joa-joa.
        SELECT * FROM t8j9a INTO TABLE @gt_t8j9a
          FOR ALL ENTRIES IN @gt_t8jg
          WHERE bukrs = @p_buk_b2 AND
                joa   = @gt_t8jg-vname AND
                egrup = @gt_t8jg-egrup.
        IF sy-subrc NE 0.
          CLEAR gt_t8j9a[].
        ENDIF.
*       SELECT * FROM t8jg  INTO TABLE gt_t8jg FOR ALL ENTRIES IN gt_jv_joa WHERE bukrs = p_buk_b2 AND vname = gt_jv_joa-vname.
*       SELECT * FROM t8j9c INTO TABLE gt_t8j9c FOR ALL ENTRIES IN gt_jv_joa WHERE bukrs = p_buk_b2 AND joa = gt_jv_joa-joa.
        SELECT * FROM t8j9c INTO TABLE @gt_t8j9c
          FOR ALL ENTRIES IN @gt_t8jg
          WHERE bukrs = @p_buk_b2 AND
                joa   = @gt_t8jg-vname AND
                egrup = @gt_t8jg-egrup.
        IF sy-subrc NE 0.
          CLEAR gt_t8j9c[].
        ENDIF
```

Changed Code:

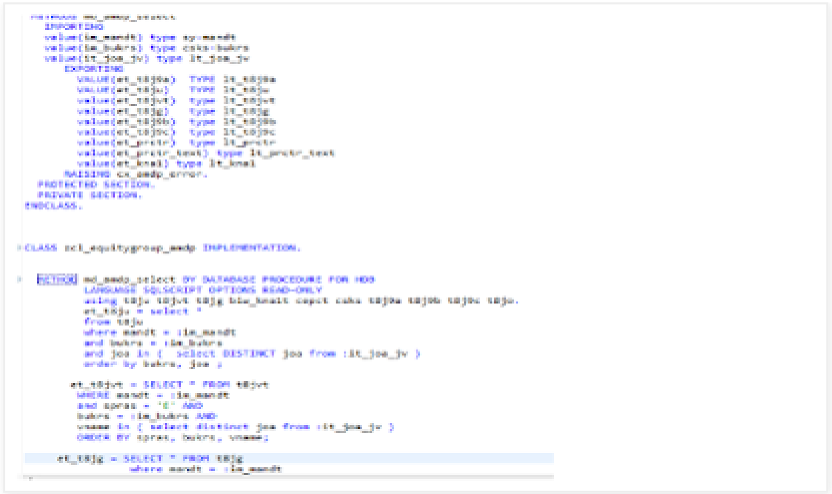
AMDP Definition:



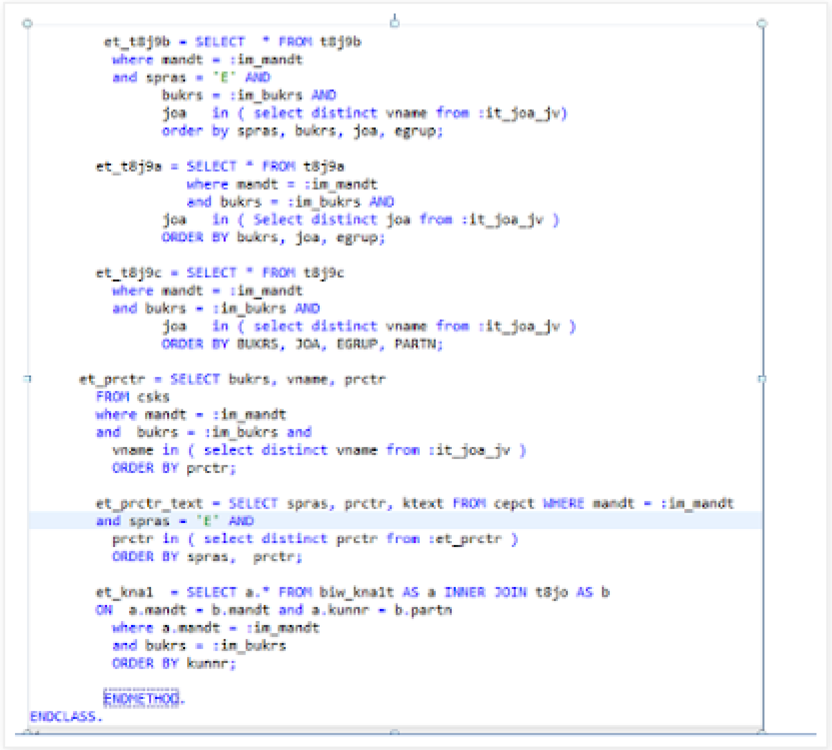
[DC1] ZCL_EQUITYGROUP_AMDP

ZCL_EQUITYGROUP_AMDP

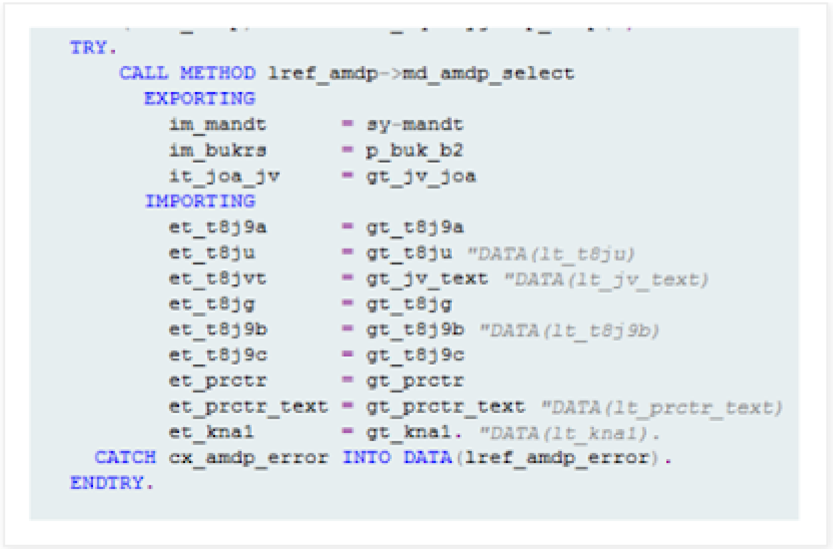
```
CLASS zcl_equitygroup_amdp DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
      INTERFACES if_amdp_marker_hdb.

types:
      BEGIN OF lty_joa_jv,
       BUKRS type BIW_JV_T8JV-bukrs,
       VNAME type    BIW_JV_T8JV-vname,
       VTYPE type BIW_JV_T8JV-vtype,
       OPERATOR type BIW_JV_T8JV-operator,
       TAXCODE type BIW_JV_T8JV-taxcode,
       VCLASS type BIW_JV_T8JV-vclass,
       JOA type BIW_JV_T8JV-joa,
       FUNDGROUP type BIW_JV_T8JV-fundgroup,
       property type  t8jv-property,
       END OF lty_joa_jv.

types: BEGIN OF lty_prctr,
                bukrs type csks-bukrs,
                vname type csks-vname,
                prctr TYPE csks-prctr,
                end of lty_prctr.

types: begin of lty_prctr_text,
                spras type cepct-spras,
                prctr type cepct-prctr,
                ktext type cepct-ktext,
                end of lty_prctr_text.

   types: lt_t8j9a type STANDARD TABLE OF t8j9a.

types: lt_joa_jv type STANDARD TABLE OF lty_joa_jv.

types: lt_t8ju TYPE HASHED TABLE OF t8ju WITH UNIQUE KEY bukrs joa.

types: lt_t8jut TYPE HASHED TABLE OF t8jut WITH UNIQUE KEY spras bukrs joa.

types: lt_t8jvt TYPE HASHED TABLE OF t8jvt WITH UNIQUE KEY spras bukrs vname.
```



```
 METHOD md_amdp_select
   IMPORTING
   value(im_mandt) type sy-mandt
   value(im_bukrs) type csks-bukrs
   value(it_joa_jv) type lt_joa_jv
      EXPORTING
        VALUE(et_t8j9a)  TYPE lt_t8j9a
        VALUE(et_t8ju)   TYPE lt_t8ju
        value(et_t8jvt)  type lt_t8jvt
        value(et_t8jg)   type lt_t8jg
        value(et_t8j9b)  type lt_t8j9b
        value(et_t8j9c)  type lt_t8j9c
        value(et_prctr)  type lt_prctr
        value(et_prctr_text) type lt_prctr_text
        value(et_kna1) type lt_kna1
      RAISING cx_amdp_error.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.



CLASS zcl_equitygroup_amdp IMPLEMENTATION.


  METHOD md_amdp_select BY DATABASE PROCEDURE FOR HDB
         LANGUAGE SQLSCRIPT OPTIONS READ-ONLY
         using t8ju t8jvt t8jg biw_kna1t cepct csks t8j9a t8j9b t8j9c t8jo.
         et_t8ju = select *
         from t8ju
         where mandt = :im_mandt
         and bukrs = :im_bukrs
         and joa in (  select DISTINCT joa from :it_joa_jv )
         order by bukrs, joa ;

       et_t8jvt = SELECT * FROM t8jvt
        WHERE mandt = :im_mandt
        and spras = 'E' AND
        bukrs = :im_bukrs AND
        vname in ( select distinct joa from :it_joa_jv )
        ORDER BY spras, bukrs, vname;

     et_t8jg = SELECT * FROM t8jg
              where mandt = :im_mandt
```



```
        et_t8j9b = SELECT  * FROM t8j9b
         where mandt = :im_mandt
         and spras = 'E' AND
               bukrs = :im_bukrs AND
               joa   in ( select distinct vname from :it_joa_jv)
               order by spras, bukrs, joa, egrup;

       et_t8j9a = SELECT * FROM t8j9a
                  where mandt = :im_mandt
                  and bukrs = :im_bukrs AND
               joa   in ( Select distinct joa from :it_joa_jv )
               ORDER BY bukrs, joa, egrup;

       et_t8j9c = SELECT * FROM t8j9c
         where mandt = :im_mandt
         and bukrs = :im_bukrs AND
               joa   in ( select distinct vname from :it_joa_jv )
               ORDER BY BUKRS, JOA, EGRUP, PARTN;

     et_prctr = SELECT bukrs, vname, prctr
       FROM csks
       where mandt = :im_mandt
       and  bukrs = :im_bukrs and
         vname in ( select distinct vname from :it_joa_jv )
         ORDER BY prctr;

       et_prctr_text = SELECT spras, prctr, ktext FROM cepct WHERE mandt = :im_mandt
       and spras = 'E' AND
         prctr in ( select distinct prctr from :et_prctr )
         ORDER BY spras,  prctr;

       et_kna1  = SELECT a.* FROM biw_kna1t AS a INNER JOIN t8jo AS b
       ON  a.mandt = b.mandt and a.kunnr = b.partn
         where a.mandt = :im_mandt
         and bukrs = :im_bukrs
         ORDER BY kunnr;

        ENDMETHOD.
ENDCLASS.
```



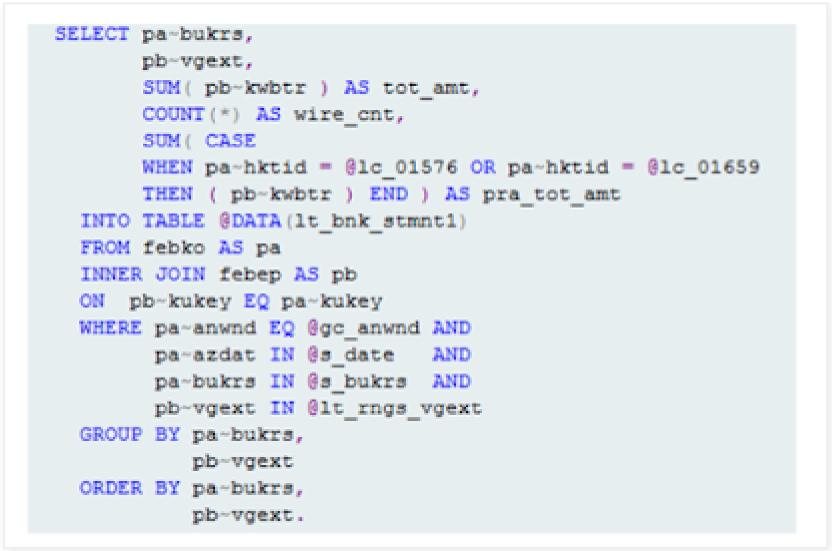
```
TRY.
    CALL METHOD lref_amdp->md_amdp_select
      EXPORTING
        im_mandt       = sy-mandt
        im_bukrs       = p_buk_b2
        it_joa_jv      = gt_jv_joa
      IMPORTING
        et_t8j9a       = gt_t8j9a
        et_t8ju        = gt_t8ju "DATA(lt_t8ju)
        et_t8jvt       = gt_jv_text "DATA(lt_jv_text)
        et_t8jg        = gt_t8jg
        et_t8j9b       = gt_t8j9b "DATA(lt_t8j9b)
        et_t8j9c       = gt_t8j9c
        et_prctr       = gt_prctr
        et_prctr_text  = gt_prctr_text "DATA(lt_prctr_text)
        et_kna1        = gt_kna1. "DATA(lt_kna1).
  CATCH cx_amdp_error INTO DATA(lref_amdp_error).
ENDTRY.
```

Advantages achieved through these changes:

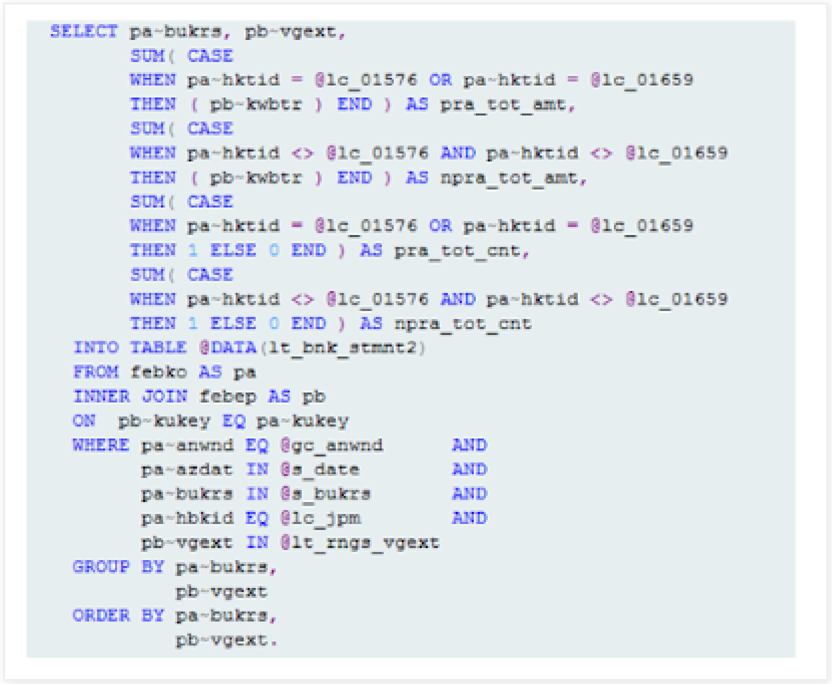
- Using CDS Views and AMDPs ensured that there was a huge boost from performance point of view. The report would now execute in approx. 8 to 10 minutes after applying these changes.
- Since the processing was moved from Application layer into the Database layer, the memory issue was no longer reported for this.

Use Case 2:

Requirement: to get the sum of revenues (Outgoing Wire Transfers/Revenue/AP) and count of activities transaction wise based on Account id.



```
SELECT pa~bukrs,
       pb~vgext,
       SUM( pb~kwbtr ) AS tot_amt,
       COUNT(*) AS wire_cnt,
       SUM( CASE
       WHEN pa~hktid = @lc_01576 OR pa~hktid = @lc_01659
       THEN ( pb~kwbtr ) END ) AS pra_tot_amt
  INTO TABLE @DATA(lt_bnk_stmnt1)
  FROM febko AS pa
  INNER JOIN febep AS pb
  ON  pb~kukey EQ pa~kukey
  WHERE pa~anwnd EQ @gc_anwnd AND
        pa~azdat IN @s_date   AND
        pa~bukrs IN @s_bukrs  AND
        pb~vgext IN @lt_rngs_vgext
  GROUP BY pa~bukrs,
           pb~vgext
  ORDER BY pa~bukrs,
           pb~vgext.
```



```
SELECT pa~bukrs, pb~vgext,
       SUM( CASE
       WHEN pa~hktid = @lc_01576 OR pa~hktid = @lc_01659
       THEN ( pb~kwbtr ) END ) AS pra_tot_amt,
       SUM( CASE
       WHEN pa~hktid <> @lc_01576 AND pa~hktid <> @lc_01659
       THEN ( pb~kwbtr ) END ) AS npra_tot_amt,
       SUM( CASE
       WHEN pa~hktid = @lc_01576 OR pa~hktid = @lc_01659
       THEN 1 ELSE 0 END ) AS pra_tot_cnt,
       SUM( CASE
       WHEN pa~hktid <> @lc_01576 AND pa~hktid <> @lc_01659
       THEN 1 ELSE 0 END ) AS npra_tot_cnt
  INTO TABLE @DATA(lt_bnk_stmnt2)
  FROM febko AS pa
  INNER JOIN febep AS pb
  ON  pb~kukey EQ pa~kukey
  WHERE pa~anwnd EQ @gc_anwnd      AND
        pa~azdat IN @s_date        AND
        pa~bukrs IN @s_bukrs       AND
        pa~hbkid EQ @lc_jpm        AND
        pb~vgext IN @lt_rngs_vgext
  GROUP BY pa~bukrs,
           pb~vgext
  ORDER BY pa~bukrs,
           pb~vgext.
```

Leveraging HANA: This was achieved using **Advanced Open SQL.**

Advantages achieved through these changes:
By clever use of Advance SQL, this has clearly eliminated loops at the Application layer.

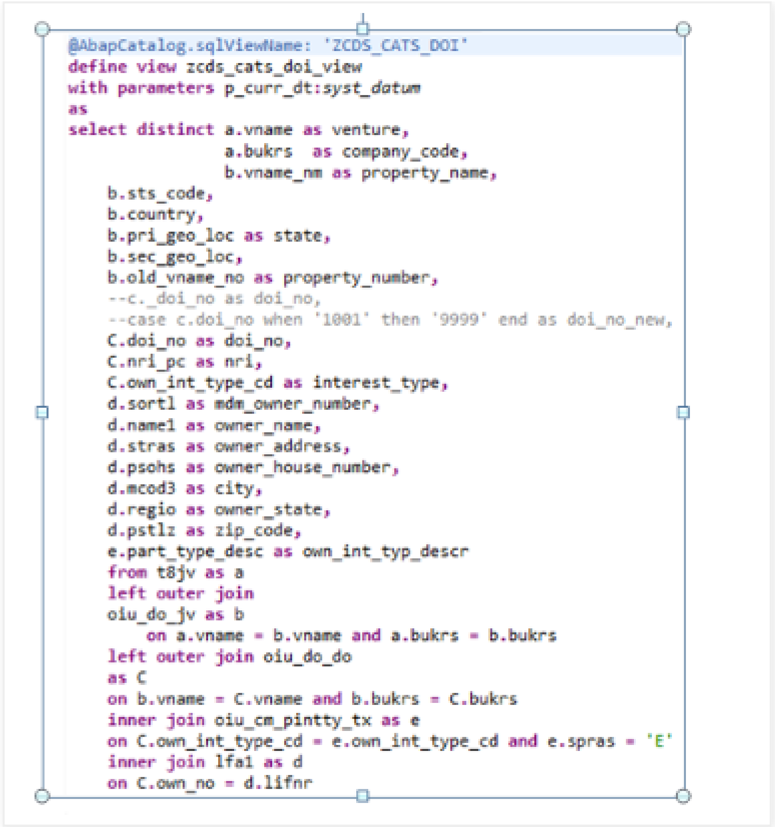
Logic explanation:
In the above example:

1. SUM(PB~KWBTR) calculates the total amount.
2. COUNT(*) gives the number of entries satisfying the WHERE condition in SELECT.
3. SUM(CASE) is used to calculate the conditional total. In the above select, the sum of amounts having Accounting ID – 01576 or 01659 is returned.

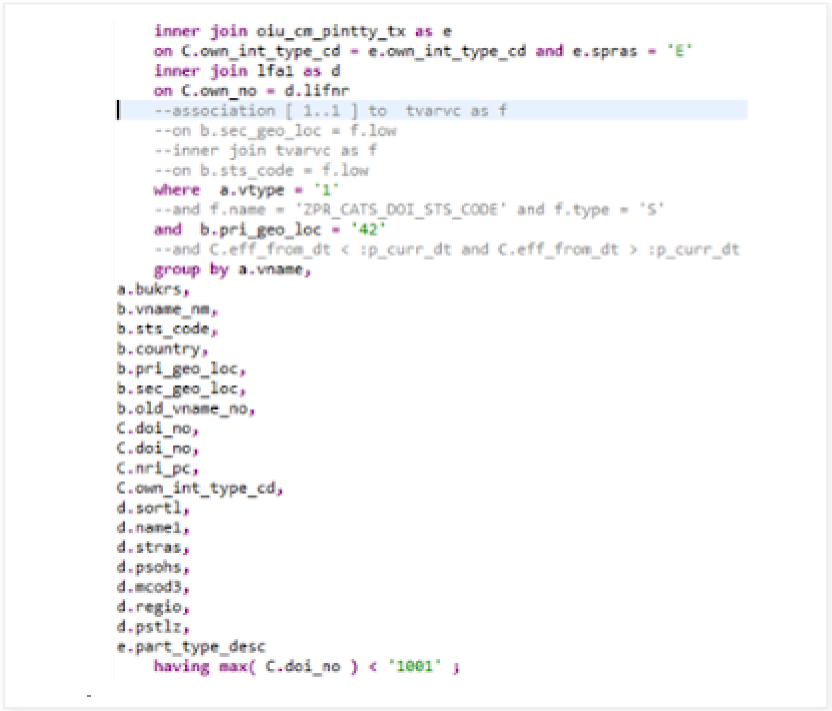
Use Case 3:

Requirement: Get the JV details on a given date for a partcicular venture and location.

CDS View Definition -

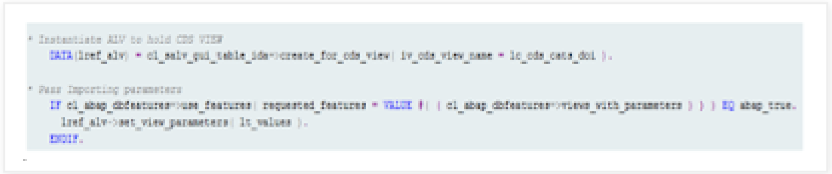


```
@AbapCatalog.sqlViewName: 'ZCDS_CATS_DOI'
define view zcds_cats_doi_view
with parameters p_curr_dt:syst_datum
as
select distinct a.vname as venture,
                a.bukrs  as company_code,
                b.vname_nm as property_name,
    b.sts_code,
    b.country,
    b.pri_geo_loc as state,
    b.sec_geo_loc,
    b.old_vname_no as property_number,
    --c._doi_no as doi_no,
    --case c.doi_no when '1001' then '9999' end as doi_no_new,
    C.doi_no as doi_no,
    C.nri_pc as nri,
    C.own_int_type_cd as interest_type,
    d.sortl as mdm_owner_number,
    d.name1 as owner_name,
    d.stras as owner_address,
    d.psohs as owner_house_number,
    d.mcod3 as city,
    d.regio as owner_state,
    d.pstlz as zip_code,
    e.part_type_desc as own_int_typ_descr
    from t8jv as a
    left outer join
    oiu_do_jv as b
        on a.vname = b.vname and a.bukrs = b.bukrs
    left outer join oiu_do_do
    as C
    on b.vname = C.vname and b.bukrs = C.bukrs
    inner join oiu_cm_pintty_tx as e
    on C.own_int_type_cd = e.own_int_type_cd and e.spras = 'E'
    inner join lfa1 as d
    on C.own_no = d.lifnr
```



```
    inner join oiu_cm_pintty_tx as e
    on C.own_int_type_cd = e.own_int_type_cd and e.spras = 'E'
    inner join lfa1 as d
    on C.own_no = d.lifnr
    --association [ 1..1 ] to  tvarvc as f
    --on b.sec_geo_loc = f.low
    --inner join tvarvc as f
    --on b.sts_code = f.low
    where  a.vtype = '1'
    --and f.name = 'ZPR_CATS_DOI_STS_CODE' and f.type = 'S'
    and  b.pri_geo_loc = '42'
    --and C.eff_from_dt < :p_curr_dt and C.eff_from_dt > :p_curr_dt
    group by a.vname,
a.bukrs,
b.vname_nm,
b.sts_code,
b.country,
b.pri_geo_loc,
b.sec_geo_loc,
b.old_vname_no,
C.doi_no,
C.doi_no,
C.nri_pc,
C.own_int_type_cd,
d.sortl,
d.name1,
d.stras,
d.psohs,
d.mcod3,
d.regio,
d.pstlz,
e.part_type_desc
    having max( C.doi_no ) < '1001' ;
```

Consuming CDS View in ABAP using IDA Class -



```
* Instantiate ALV to hold CDS VIEW
  DATA(lref_alv) = cl_salv_gui_table_ida=>create_for_cds_view( iv_cds_view_name = lc_cds_cats_doi ).

* Pass Importing parameters
  IF cl_abap_dbfeatures=>use_features( requested_features = VALUE #( ( cl_abap_dbfeatures=>views_with_parameters ) ) ) EQ abap_true.
    lref_alv->set_view_parameters( lt_values ).
  ENDIF.
```

Advantages achieved through these changes:

1. By using IDA, the Code Push Down model is optimally supported using the in-memory database and considerable performance can be achieved for vast amount of data.
2. By using CDS, there is no select query required to get the data into application layer. IDA being a Persistence class, & effective usage of the concept of paging, transfer of large amount of data is avoided.

Source: scn.sap.com

Posted by Sabrina Pinto at 00:24:00

Labels: abap-on-hana, abap-on-hana-use-cases, amdp-procedure, cds-views, ida-class

No comments:

Post a Comment

Newer Post Home Older Post