LOGALI

Teoría

Table Function

ABAP Cloud – Modelado con CDS

Contenido

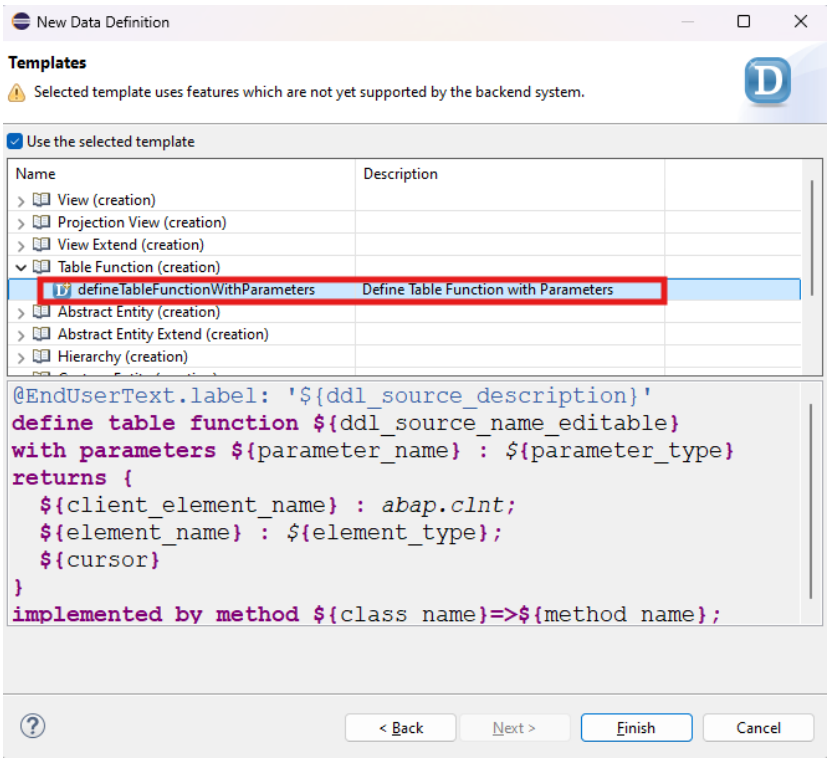
10. Table Function

10.1. Table Function – Definición

Los Table Functions en SAP HANA son artefactos especiales de tipo Data Definition Language (DDL) utilizados en el contexto de Core Data Services (CDS) para ejecutar lógica SQL Script avanzada directamente en la base de datos HANA. Estas funciones permiten definir y proyectar elementos y tipos de datos específicos, delegando la lógica de selección y procesamiento de datos a métodos estáticos dentro de clases ABAP utilizando ABAP Managed Database Procedures (AMDP). Los métodos AMDP son especiales en el sentido de que se ejecutan directamente en la base de datos HANA y pueden utilizar toda la potencia del SQL Script nativo para manipular y devolver datos.

Este enfoque optimiza el rendimiento y la eficiencia en la manipulación de datos, permitiendo una integración sólida entre las capacidades de procesamiento de HANA y la robustez del entorno ABAP, todo ello gestionado de manera eficiente en la capa de persistencia de datos.

Para crear un Table Function, se realiza a través de la carpeta de proyecto luego New en la opción **Other ABAP Repository Object** ubicar la carpeta **Core Data Services** y luego seleccionar **Data Definition** y seleccionar la plantilla **defineTableFunctionWithParameters** que se encuentra en la carpeta **Table Function (creation).**



Sintaxis:

```
@EndUserText.label: 'CDS - Table Function Test'
define table function table_function_name
// with parameters parameter_name : parameter_type
returns {
 client_element_name : abap.clnt;
 element_name : element_type;
...
 }
implemented by method class_name=>method_name;
```

Puntos a tomar en cuenta:

- Es recomendable colocar para el primer elemento del table function el componente CDS relacionada con el campo clave del mandante o cliente. Y después cualquier otro componente.
- Aunque los parámetros no son obligatorios en una Table Function, se pueden incluir para permitir filtrado y personalización adicionales en las consultas. Esto añade una capa adicional de flexibilidad y utilidad para casos de uso específicos.

- Es posible especificar una clase previamente creada y colocar el nombre de un método no declarado para vincularlo dentro de una Table Function. Esta acción no generará errores de sintaxis al momento de activar la Table Function; sin embargo, los errores se presentarán al intentar ejecutarla.
- Aunque es posible especificar los campos claves dentro de los componentes declarados dentro de un table function, no es necesario especificar los componentes no nulos.
- Los campos de unidad y de moneda sí deben de estar identificados con la correspondiente anotación semántica.

Creación y Uso de Clases ABAP:

Para implementar la lógica de una Table Function, se crea una clase ABAP en la carpeta de Source Code Library. Esta clase debe implementar la interfaz **IF_AMDP_MARKER_HDB,** obligatoriamente para poder utilizar métodos AMDP.

Dentro de esta clase, se define un método estático, que contendrá el código SQL Script necesario para realizar las selecciones y manipulaciones de datos requeridas.

El SQL Script dentro del método AMDP se utiliza para seleccionar, unir, y procesar los datos necesarios según la lógica definida. Esto puede incluir operaciones complejas como cálculos, agregaciones, y filtrado avanzado.

Es importante asegurar que el método AMDP respete la firma de la Table Function definida en el DDL, devolviendo los datos en el formato y estructura esperados.

Ejemplo de la clase donde se declaran los métodos AMDP:

```
CLASS class_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
 INTERFACES if_amdp_marker_hdb.
```

```
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
CLASS class_name IMPLEMENTATION.
ENDCLASS.
```

10.2. Método AMDP para Table Function

Los métodos AMDP (ABAP Managed Database Procedures) permiten la integración y manipulación de Table Functions dentro de la programación ABAP, utilizando SQL Script nativo para operaciones complejas y eficientes sobre bases de datos. Estos métodos, implementados en clases ABAP, deben seguir una estructura definida y estar correctamente vinculados a las interfaces y parámetros necesarios. La programación en SQL Script permite la selección y manipulación de datos directamente en la base de datos, optimizando el rendimiento y la precisión de las operaciones. La depuración y prueba de estos métodos se realiza a nivel de la capa de persistencia, proporcionando herramientas avanzadas para asegurar la correcta ejecución y retorno de datos según los requerimientos definidos.

La clase en cuestión debe implementar la interfaz de manera obligatoria **IF_AMDP_MARKER_HDB**, para poder implementar las funciones AMDP. Se realiza la implementación pública y estática del método que será utilizado en la tabla functions dentro de la instrucción **implemented by method class_name=>method_name;** para vincularlo con el Table Function.

Se pueden definir parámetros como que luego se utilizan dentro de la implementación. La activación con **WITH PARAMETERS,** dentro del table function permite definir estos parámetros que se utilizarán en la ejecución del método.

El método AMDP se implementa declara e implementa dentro de la clase abap con la siguiente sintaxis:

Declaración del método:

```
CLASS class_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
   INTERFACES if_amdp_marker_hdb.
 CLASS-METHODS method_name FOR TABLE FUNCTION table_function_name.
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
```

Implementación del método:

```
CLASS class_name IMPLEMENTATION.
        METHOD method_name BY DATABASE FUNCTION FOR HDB
                        LANGUAGE SQLSCRIPT
                        OPTIONS READ-ONLY
                        USING table1 table2 tableN.
        return select …
         ENDMETHOD.
ENDCLASS.
```

Puntos importantes a considerar:

- Dentro de la implementación del método las consultas se realizan utilizando SQL Script nativo, permitiendo operaciones de lectura a través de la instrucción **OPTIONS READ-ONLY**. Se detallan los pasos para seleccionar datos y definir fuentes utilizando **SELECT FROM** y **JOIN**.
- Además en la implementación se agregan las fuentes con la instrucción **USING,** y los nombres de las bases de datos requeridas separadas entre sí con un espacio en blanco.
- Durante la construcción de la consulta **SELECT** si una fuente de base de datos contiene en su nombre el carácter "**/**" se debe indicar la fuente dentro de comillas dobles y en mayúsculas por ejemplo **"/DMO/AIRPORT".** Además es importante colocarle

un alias para poder hacer uso del mismo para la especificación de los componentes que se requieran consultar. Aunque el alias se puede colocar independientemente de que el nombre de la base de datos contenga "**/**".

- Si los nombres de los componentes en el table functions son diferentes a los de la tabla de base de datos es necesario colocar un alias al componente de base de datos para igualarlo al del table function vinculado dentro de la consulta.
- En las consultas CDS (Core Data Services) en ABAP, no se puede definir un orden dentro de la propia sentencia de definición de la vista CDS. Esto se debe a que una vista es esencialmente una representación de los datos en la base de datos y no controla la ordenación de los resultados. El orden se debe realizar en el momento en que se ejecuta una consulta sobre esa vista. Sin embargo, es posible ordenar los resultados de un table function al momento de hacer la consulta usando la cláusula **order by** y colocando **asc** o **desc**, para el tipo de orden ascendiente o descendiente respectivamente. Esto permite especificar cómo los datos serán ordenados cuando se recupera la información.
- Luego de haber implementado el método AMDP en la clase es posible mediante el table function ejecutar la consulta creada y visualizar los datos a través de la herramienta Data Preview con **F8.**
- En el caso de que se haya declarado un parámetro en el table function, se puede utilizar en los filtros de la consulta dentro del método AMDP con dos puntos seguido del nombre del parámetro. Y al momento de ejecutar el table function se desplegará una pantalla para poder colocar dicho parámetro antes de la ejecución.
- Un dato adicional, es que al colocar un punto de parada o de depuración dentro de un método AMDP este será de color verde que corresponde al sistema de base de datos Hana. En contraste, los puntos de parada dentro de un código abap son de color azul.

Ejemplo de la declaración de un table function con parámetro con sus métodos AMDP:

Declaración del Table Function:

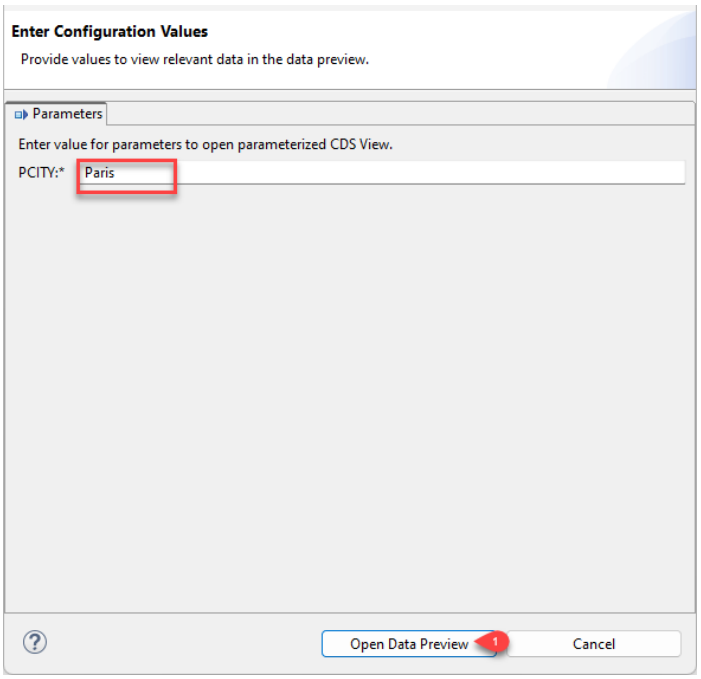
```
@EndUserText.label: 'CDS - Table Function Test'
define table function ztf_table_function_name
 with parameters
   pCity : /dmo/city
returns
{
 key client     : abap.clnt;
 key airport_id  : /dmo/airport_id;
 key customer_id : /dmo/customer_id;
    first_name  : /dmo/first_name;
    city       : /dmo/city;
}
implemented by method
 zcl_table_function_NAME=>get_airport;
```

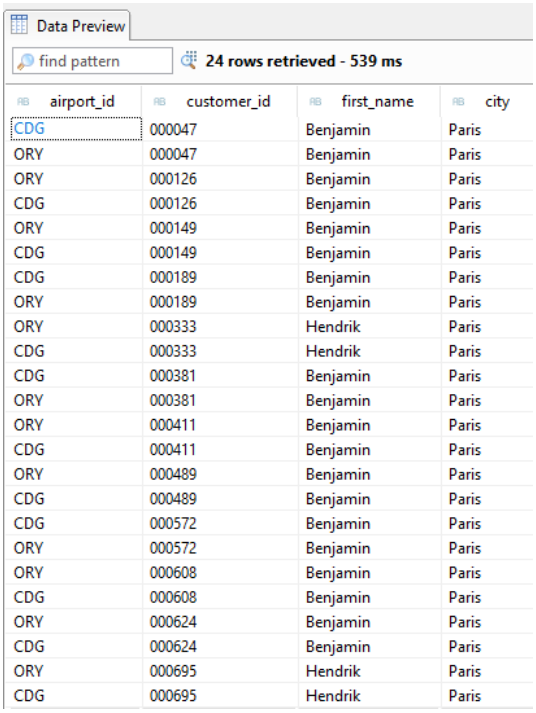
Declaración de la clase con método AMDP:

```
CLASS zcl_table_function_NAME DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
   INTERFACES if_amdp_marker_hdb.
   CLASS-METHODS      get_airport      FOR      TABLE      FUNCTION
Ztf_TABLE_FUNCTION_NAME.
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
CLASS zcl_table_function_NAME IMPLEMENTATION.
 METHOD get_airport BY DATABASE FUNCTION FOR HDB
                LANGUAGE SQLSCRIPT
                OPTIONS READ-ONLY
                USING /dmo/airport /dmo/customer.
 RETURN select
        a.client,
        a.airport_id,
```

```
        b.customer_id,
        b.first_name,
        a.city
         from "/DMO/AIRPORT"  as a
         inner join "/DMO/CUSTOMER" as b
           on a.city = b.city
           WHERE a.city = :pCity
           ORDER BY b.customer_id ASC;
 ENDMETHOD.
ENDCLASS.
```

Ejecución del Table Function luego de la declaración de la clase:





Data Preview

find pattern — 24 rows retrieved - 539 ms

airport_id	customer_id	first_name	city
CDG	000047	Benjamin	Paris
ORY	000047	Benjamin	Paris
ORY	000126	Benjamin	Paris
CDG	000126	Benjamin	Paris
ORY	000149	Benjamin	Paris
CDG	000149	Benjamin	Paris
CDG	000189	Benjamin	Paris
ORY	000189	Benjamin	Paris
ORY	000333	Hendrik	Paris
CDG	000333	Hendrik	Paris
CDG	000381	Benjamin	Paris
ORY	000381	Benjamin	Paris
ORY	000411	Benjamin	Paris
CDG	000411	Benjamin	Paris
ORY	000489	Benjamin	Paris
CDG	000489	Benjamin	Paris
CDG	000572	Benjamin	Paris
ORY	000572	Benjamin	Paris
ORY	000608	Benjamin	Paris
CDG	000608	Benjamin	Paris
ORY	000624	Benjamin	Paris
CDG	000624	Benjamin	Paris
ORY	000695	Hendrik	Paris
CDG	000695	Hendrik	Paris

Restricciones de los métodos AMDP.

- No es posible utilizar el mismo método para hacer referencia a dos table functions dentro de la clase abap. Aunque si es posible tener múltiples métodos AMDP con sus respectivos table functions diferentes dentro de la misma clase. En este caso sería como decir que los métodos AMDP tienen una cardinalidad de 1:1 con un único table function dentro de una misma clase.
- En la vista de depuración solo está permitido las acciones de resume o continuar hasta el siguiente punto de parada con el atajo **F8.** Y step over de salto a la siguiente instrucción con el atajo **F6**. Por qué se está depurando a nivel del del servidor de base de datos.

10.3. Table Function - Manejo de Mandante

El manejo de mandante o cliente en Table Functions se refiere a la gestión específica de los datos pertenecientes a diferentes mandantes en un sistema SAP. Esta práctica es esencial para asegurar que las operaciones de datos se realicen de manera segura y eficiente, respetando las particiones de datos entre los mandantes. En la definición e implementación de Table Functions, se puede

especificar si una función es dependiente del mandante (client dependent) o independiente del mandante (client independent). Este manejo se realiza agregando un parámetro a la entidad CDS para especificar el mandante y se aplica en las consultas SQL Script para filtrar y manipular datos de acuerdo con el mandante activo. La correcta implementación y depuración de estas funciones es crucial para mantener la integridad y el rendimiento del sistema.

Tipos de dependencias del mandante:

El manejo del mandante se define directamente en la cabecera de la definición del Table Function a través de la anotación.

- @ClientHandling.type: #

Es posible indicar si el Table Function dependerá del mandante (**#CLIENT_DEPENDENT**) o no (**#CLIENT_INDEPENDENT**).

- **Cliente dependiente del mandante:** La dependencia del mandante en Table Functions implica que todas las operaciones de datos están específicamente asociadas a un único mandante (cliente), dentro de un sistema SAP. Esto asegura que los datos sean segregados y manejados de manera segura y aislada para cada mandante. En este esquema, es necesario varios aspectos para que pueda ser implementado:
  - **Dentro del table function:**
    - Especificar en la entidad CDS, en la anotación de cabecera para el manejo de las dependencias que sea dependiente del mandante de la siguiente manera: @ClientHandling.type: #CLIENT_DEPENDENT**.**
    - Luego colocar un parámetro para especificar el mandante con el tipo **abap.clnt,** y colocarle la anotación: @Environment.systemField: #CLIENT. Que se utiliza para referenciar automáticamente el campo del sistema que corresponde al mandante requerido del entorno SAP.
  - **Dentro de la clase abap:**

- En la declaración del método estático es necesario agregar después del nombre del método la instrucción FOR TABLE FUNCTION, para especificar el table function asociado.
- Es necesario agregar los parámetros declarados dentro del table function en el filtro de la consulta realizada dentro del método AMDP.
- Además, si se ha definido una relación **JOIN,** de cualquier tipo a la consulta es necesario agregar la relación por el componente del mandante.

Ejemplo:

Declaración del Table Function:

```
@EndUserText.label: 'CDS - Table Function Test'
@ClientHandling.type: #CLIENT_DEPENDENT
define table function ztf_table_function_name
 with parameters
  @Environment.systemField: #CLIENT
  pClient : abap.clnt,
  pCity : /dmo/city
returns
{
 key client          : abap.clnt;
 key airport_id      : /dmo/airport_id;
 key customer_id : /dmo/customer_id;
      first_name    : /dmo/first_name;
      city          : /dmo/city;
}
implemented by method
 zcl_table_function_NAME=>get_airport;
```

Declaración de la clase con método AMDP:

```
CLASS zcl_table_function_NAME DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
```

```
 PUBLIC SECTION.
  INTERFACES if_amdp_marker_hdb.
  CLASS-METHODS get_airport FOR TABLE FUNCTION
Ztf_TABLE_FUNCTION_NAME.
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.


CLASS zcl_table_function_NAME IMPLEMENTATION.
 METHOD get_airport BY DATABASE FUNCTION FOR HDB
                LANGUAGE SQLSCRIPT
                OPTIONS READ-ONLY
                USING /dmo/airport /dmo/customer.
    RETURN select
          a.client,
          a.airport_id,
          b.customer_id,
          b.first_name,
          a.city
           from "/DMO/AIRPORT"  as a
           inner join "/DMO/CUSTOMER" as b
             on a.client = b.client and
               a.city = b.city
             where a.client = :pClient and
                 a.city = :pCity
             ORDER BY b.customer_id asc;
 ENDMETHOD.
ENDCLASS.
```

- **Cliente dependiente del mandante:** La independencia del mandante permite que una Table Function maneje datos de múltiples mandantes simultáneamente sin estar restringida a uno solo. En este caso, no se utiliza un parámetro específico del mandante, lo que permite seleccionar y manipular datos de diferentes mandantes. Este enfoque proporciona mayor flexibilidad y es útil para operaciones que requieren una visión integral de los datos. Se debe especificar en la entidad CDS, en

la anotación de cabecera para el manejo de las dependencias que sea independiente del mandante de la siguiente manera:**@ClientHandling.type: #CLIENT_INDEPENDENT.**

Y a pesar de que es necesario agregar los parámetros declarados dentro del table function en el filtro de la consulta realizada dentro del método AMDP. No es necesario, si se ha definido una relación **JOIN,** de cualquier tipo a la consulta agregar la relación por el componente del mandante.

Ejemplo:

```
@EndUserText.label: 'CDS - Table Function Test'
@ClientHandling.type: #CLIENT_INDEPENDENT
define table function table_function_name
returns {
 client_element_name : abap.clnt;
 element_name : element_type;
…
 }
implemented by method class_name=>method_name;
```

Declaración de la clase con método AMDP:

```
CLASS zcl_table_function_NAME DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
  INTERFACES if_amdp_marker_hdb.
  CLASS-METHODS get_airport FOR TABLE FUNCTION
Ztf_TABLE_FUNCTION_NAME.
 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.


CLASS zcl_table_function_NAME IMPLEMENTATION.
 METHOD get_airport BY DATABASE FUNCTION FOR HDB
            LANGUAGE SQLSCRIPT
```

```
                OPTIONS READ-ONLY
                USING /dmo/airport /dmo/customer.
        RETURN select
            a.airport_id,
            b.customer_id,
            b.first_name,
            a.city
             from "/DMO/AIRPORT"  as a
             inner join "/DMO/CUSTOMER" as b
                a.city = b.city
              where  a.city = :pCity
              ORDER BY b.customer_id asc;
    ENDMETHOD.
ENDCLASS.
```

10.4. Table Function - Consumo en CDS

Las Table Functions pueden ser utilizadas como fuentes de datos dentro de las entidades CDS (Core Data Services), Al realizar joins o asociaciones con dichos table functions dentro de las entidades CDS, la información se recupera directamente de las Table Functions. Esto permite que las consultas ejecutadas en los métodos AMDP proporcionen los datos necesarios de manera eficiente y estructurada. Esto permite que las consultas de CDS se enriquezcan con la lógica compleja y eficiente definida en los métodos AMDP (ABAP Managed Database Procedures). Al utilizar Table Functions como fuentes de datos en las vistas CDS, se puede lograr una integración más robusta y detallada, donde la información es recuperada directamente a partir de las consultas ejecutadas en los métodos AMDP.

Ejemplos:

Consumo de table function en una entidad CDS mediante asociación:

```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS - CDS with Table Function'
```

```
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity cds_entity_name
 with parameters
  pComp: data_element_name
 as select from entity_name as AliasName
 association [0..*] to table_function_name as _AliasName on
_AliasName.component = $projection.AliasNameId
{
    // association Components
 key component      as AliasName,
    component          as AliasName2,
    //_AliasName // association without parameters
_AliasName(pComp:          $parameters.pComp).component          as
AliasName3 // association with parameters
}
```

Consumo de table function en una entidad CDS mediante join:

```
@AbapCatalog.viewEnhancementCategory: [#NONE]
@AccessControl.authorizationCheck: #NOT_REQUIRED
@EndUserText.label: 'CDS - CDS with Table Function'
@Metadata.ignorePropagatedAnnotations: true
@ObjectModel.usageType:{
  serviceQuality: #X,
  sizeCategory: #S,
  dataClass: #MIXED
}
define view entity cds_entity_name
 with parameters
  pComp : data_element_name
 as select from entity_name                    as AliasName
  inner join   table_function_name(pComp: $parameters.pComp) as
_AliasName on _AliasName.component = AliasName.component
```

```
{
    // inner join Components
 key AliasName.component as AliasName,
    AliasName.component2      as AliasName2,
    _AliasName.component3   as AliasName3,
}
```

En los casos en que la Table Function sea dependiente o independiente del mandante, es crucial considerar las características específicas de cada uno, tal como se explicó en el punto anterior, en el manejo de los mandantes.

10.5. AMDP – Client Independent

Los métodos AMDP (ABAP Managed Database Procedures) de tipo Client Independent permiten gestionar datos sin considerar el mandante específico en SAP. Esta configuración es esencial para acceder y manipular datos de múltiples mandantes simultáneamente, proporcionando una visión integral de los datos. Al definir un método como independiente del mandante se garantiza que las operaciones de datos no estén limitadas a un solo mandante, lo que permite una mayor flexibilidad en las consultas y en la manipulación de datos. Este enfoque es útil en escenarios donde se necesita consolidar información de diferentes mandantes para análisis y reportes. Además, se requiere ajustar las fuentes de datos para asegurarse de que no dependen del campo de mandante (CLIENT), modificando tablas y vistas según sea necesario.

Sintaxis:

```
CLASS class_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
   INTERFACES if_amdp_marker_hdb.
   TYPES lt_table TYPE TABLE OF data_source_name.

CLASS-METHODS method_name AMDP OPTIONS CLIENT
```

```
                                    INDEPENDENT
                                EXPORTING VALUE(ET_RETURN) TYPE
lt_table.
        PROTECTED SECTION.
        PRIVATE SECTION.
      ENDCLASS.
      CLASS class_name IMPLEMENTATION.
        METHOD method_name BY DATABASE PROCEDURE FOR HDB
                                    LANGUAGE SQLSCRIPT
                                    OPTIONS READ-ONLY
                                    USING data_source_name.
          et_return = select  *  from data_source_name as aliasName;
        ENDMETHOD.
      ENDCLASS.
```

Puntos importantes a considerar:

- Para este caso solo se pueden utilizar base de datos que no tenga un componente de mandante o una entidad o vista CDS que haga referencia a una tabla de base de datos independiente de mandante o sin componente de mandante. Dentro del uso de los métodos AMDP.
- En la declaración del método estático AMDP seguido del nombre del mismo, se debe agregar la instrucción AMDP OPTIONS CLIENT INDEPENDENT. Para identificar que es un método independiente del mandante.
- Además se debe exportar los registros consultados por valor los en los métodos AMDP, para asegurar la correcta transmisión de datos entre el servidor de aplicaciones ABAP y la base de datos HANA.
- También es importante mencionar que en la implementación del método AMDP se debe agregar la instrucción BY DATABASE PROCEDURE FOR HDB. En lugar de la instrucción que se ha estado utilizando BY DATABASE FUNCTION FOR HDB en la implementación de los métodos AMDP asociados con una table function.

10.6. Consumo CDS en métodos AMDP

El consumo de CDS (Core Data Services) en métodos AMDP permite integrar fuentes de datos avanzadas y lógicas complejas definidas en AMDP (ABAP Managed Database Procedures) directamente en las consultas CDS. Esto proporciona mayor flexibilidad y eficiencia en la manipulación y recuperación de datos. Al definir una vista CDS que consume una Table Function, se pueden seleccionar, proyectar y asociar datos de manera precisa, manejando de forma adecuada el contexto del mandante. Es crucial gestionar los parámetros y asegurar que las consultas sean precisas y eficientes, utilizando prácticas como la proyección dinámica de elementos y el uso de INNER JOIN cuando sea necesario.

Declaración de la clase con el método AMDP independiente del mandante:

```
CLASS class_name DEFINITION
 PUBLIC
 FINAL
 CREATE PUBLIC .
 PUBLIC SECTION.
  INTERFACES if_amdp_marker_hdb.
  INTERFACES if_oo_adt_classrun.
  TYPES lt_table TYPE TABLE OF data_source_name.

CLASS-METHODS method_name AMDP OPTIONS CDS SESSION
CLIENT current    EXPORTING VALUE(ET_RETURN) TYPE lt_table.

 PROTECTED SECTION.
 PRIVATE SECTION.
ENDCLASS.
CLASS class_name IMPLEMENTATION.
 METHOD method_name BY DATABASE PROCEDURE FOR HDB
                                LANGUAGE SQLSCRIPT
                                OPTIONS READ-ONLY
                                USING data_source_name.
  et_return = select  top number *  from data_source_name as
aliasName;
 ENDMETHOD.
```

```
METHOD if_oo_adt_classrun~main.
class_name=>method_name(
        exporting
        et_return = data(lt_return)).
ENDMETHOD.
ENDCLASS.
```

Puntos importantes a considerar:

- La instrucción **CDS SESSION CLIENT CURRENT**, en la declaración del método estático AMDP establece la variable de sesión current al valor predeterminado del cliente actual. Sin embargo, no restringe la lista USING, lo que significa que el método puede acceder a objetos que no sean necesariamente seguros para el cliente actual.
- Es posible utilizar en las consultas tabla de base de datos y vistas CDS en conjunto en la implementación del método AMDP en la USING **data_source_name** separadas por un espacio.
- Para devolver un número determinado de registros, se puede incluir la instrucción **TOP** seguida del número de registros después de la instrucción **SELECT**. Esto es diferente de las sentencias SQL ABAP tradicionales, donde se utiliza la sentencia **UP TO ROWS** para lograr lo mismo.