LOGALI

Teoría

Fundamentos de Modelado de Datos

ABAP Cloud – Modelado con CDS

Contenido

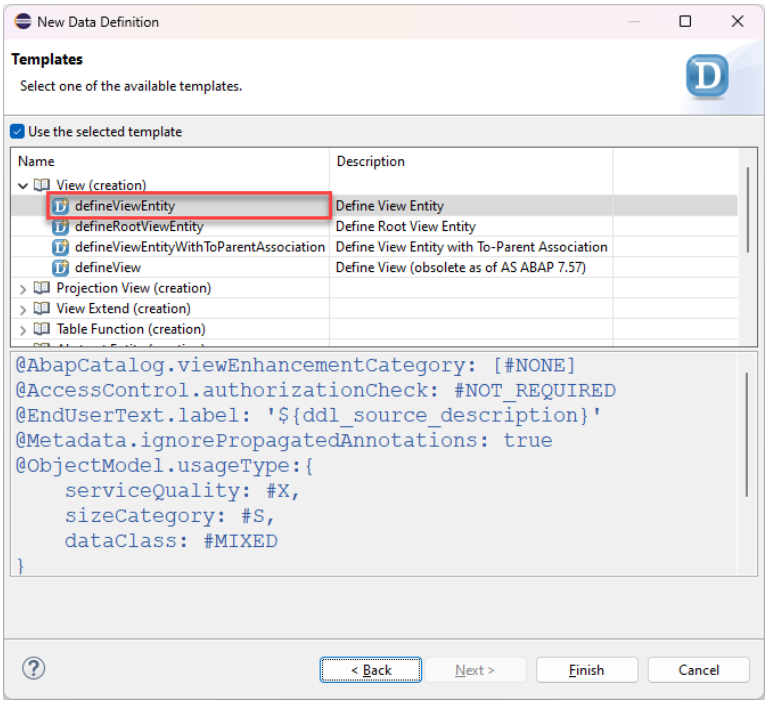
2. Fundamentos de Modelado de Datos

2.1. CDS - Creación

Los Core Data Services (CDS) Es una tecnología integral de SAP que permite definir y consumir modelos de datos en el entorno ABAP de manera más eficiente y con un mayor rendimiento. Los CDS permiten crear vistas y entidades que se ejecutan directamente en la base de datos HANA, aprovechando al máximo sus capacidades. A través de un lenguaje de definición de datos (DDL), es posible diseñar artefactos de base de datos de forma declarativa.

Las entidades CDS se pueden utilizar para modelar datos de manera más intuitiva y estructurada, facilitando la creación y gestión de vistas complejas. Además, las anotaciones permiten enriquecer estos modelos con metadatos, controles de acceso, y otras características que mejoran la funcionalidad y seguridad de los datos.

Para crear una entidad o vista CDS, 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 **defineViewEntity** que se encuentra en la carpeta **View (creation).**



Sintaxis

```
@AbapCatalog.viewEnhancementCategory: [#]
@AccessControl.authorizationCheck: #
@EndUserText.label: ''
@Metadata.ignorePropagatedAnnotations: true/false
@ObjectModel.usageType:{
        serviceQuality: #,
        sizeCategory: #,
        dataClass: #
}
define view entity entity_name as select from table_name
{
        components,
        component as alias
}
```

Anotaciones:

En una entidad Core Data Services las anotaciones son metadatos que permiten definir características adicionales y comportamientos específicos de los elementos dentro de un modelo de datos. Estas anotaciones se escriben mediante un símbolo de arroba (@) seguido del nombre de la anotación y su valor correspondiente. Al momento de crear una entidad CDS. Se agregan las anotaciones siguientes:

- **@AbapCatalog.viewEnhancementCategory: [#]:** Esta anotación específica cómo se extiende la vista de la entidad usando CDS. Por defecto se inicializa con el valor [#NONE].
- **@AccessControl.authorizationCheck: #:** Define el control de acceso implícito cuando se usa ABAP SQL para acceder a la vista CDS. Por defecto se inicializa con el valor **#NOT_REQUIRED.**
- **@EndUserText.label: '':** Esta anotación asigna una etiqueta de texto para el usuario final.
- **@Metadata.ignorePropagatedAnnotations: true/false:** Indica que las anotaciones propagadas deben ser ignoradas.

Esto puede ser útil para evitar conflictos o redundancias en las anotaciones. Por defecto se inicializa con el valor true.

- **@ObjectModel.usageType.serviceQuality: #:** Indica la calidad del servicio. Por defecto se inicializa con el valor #X.
- **@ObjectModel.usageType.sizeCategory: #:** Indica el tamaño de la categoría. Por defecto se inicializa con el valor #S.
- **@ObjectModel.usageType.dataClass: #:** Indica la calidad de la categoría. Por defecto se inicializa con el valor #MIXED.

2.2. Campos clave

Los campos clave son identificadores únicos para cada registro en una tabla o vista. Las entidades CDS permiten definir claves para las columnas, similar a las tablas. Aunque no es obligatorio, es recomendable definir claves para evitar advertencias y mejorar el rendimiento en bases de datos HANA. Las claves pueden ser técnicas (por ejemplo, UUID) o semánticas, que son más representativas para los usuarios de negocio. Las claves semánticas se definen a través del Object Model, agregando anotaciones que permiten transmitir información clara y relevante al usuario final.

Sintaxis

```
define view entity entity_name as select from table_name
{
    key component_name
    components,
    component as alias
}
```

2.3. Casting

Es una función en CDS que se utiliza para convertir un valor de un tipo de datos a otro tipo de datos. Es especialmente útil cuando necesitas asegurar que los datos en tu vista CDS se manejen y formateen correctamente, de acuerdo con los tipos de datos deseados.

Sintaxis

cast(component_name as tipo_dato)

Ejemplo

```
cast( '99991231' as abap.dats )
```

2.4. Case

La instrucción **Case** se emplea para aplicar diversas condiciones antes de proyectar datos en la definición de CDS (Core Data Services). Es una herramienta útil para incorporar lógica de programación en la definición a través del DDL (Data Definition Language). De esta forma se asegura que las salidas de las diferentes condiciones sean del mismo tipo de dato, y mostrar varias posibles salidas, incluyendo un valor por defecto (‘Any value’) para cuando ninguna condición se cumple.

También es posible integrar funciones como **CONCAT** para concatenar cadenas o como **DS_ADD_DAYS** para manipular fechas. Además, es posible utilizar fechas y otros tipos de datos, se pueden emplear funciones, asegurando que los tipos de datos sean consistentes en todas las posibles salidas de la instrucción **Case**.

Es importante entender que, aunque se pueden anidar múltiples Case dentro de un Case principal, todas las proyecciones deben ser del mismo tipo de dato. La instrucción Case se activa y se prueba, mostrando las salidas correspondientes para cada condición establecida, asegurando que la lógica de programación y las condiciones se cumplan adecuadamente en la base de datos.

Sintaxis

case

when condición **then** resultado

else default_result

end as component_name

Ejemplo

```
define view entity entity_name
 as select from table_name
{
// Key Field
 key component as Alias,
// Concatenation and nested concatenation
    case component
       when 'value' then concat( 'value', component )
    when 'value' then concat( 'value', concat( component, component
) )
    else 'value if no other case applies'
    end       as Case1,
//Integrate functions
    case
    when  component  =  'value'  or  component  =  'value'  then
component
    when component = 'value'  then component
    when  cast(  '99991231'  as  abap.dats  )    dats_add_days  (
$session.system_date, -number, 'NULL') then 'value'  else 'value'
    end       as Case2,
// Nested Case
    case
    when  component  =  'value'  or  component  =  'value'  then  case
component
                                  when 'value' then 'value'
                                  when ''value'  then 'value'
                                  else 'value'
                                  end
     else 'value if no other case applies'
    end       as CaseN
}
```

2.5. Variables de sesión

Las variables de sesión son herramientas poderosas que permiten personalizar y optimizar las entidades CDS, proporcionando acceso a información crítica del sistema y del usuario. Las variables de sesión

se utilizan para almacenar datos temporales durante una sesión de usuario. Es posible utilizar variables de sesión a través del acceso **$session**. Estas variables permiten obtener información relevante sobre el sistema, como el cliente de conexión, la fecha, el idioma del sistema, el nombre del usuario, la zona horaria del usuario, y otros detalles. Estas variables se pueden incorporar tanto en las proyecciones como en las funciones que utilizamos en el modelo.

Al definir una nueva CDS, se pueden usar estas variables para proyectar datos desde cualquier fuente disponible. Es obligatorio indicar una fuente, aunque no todos los elementos de esa fuente deben ser proyectados. La proyección de variables de sesión facilita la obtención de información específica y relevante, como:

- **$session.client**: Mandante
- **$session.system_date**: Fecha del sistema
- **$session.system_language**: Idioma
- **$session.user_date**: Fecha del equipo del usuario
- **$session.user_time_zone**: Hora del sistema

Además, las variables de sesión también se pueden utilizar en funciones y filtros durante el consumo de datos de diferentes fuentes. Sin embargo, hay algunas variables que no se pueden utilizar en el modelo CDS, como **$session.business_instance_id** y **$session.zone_id**.

Sintaxis

```
define view entity entity_name as select from table_name
{
    key component_name
    $session.client              as component_name
    $session.system_date         as component_name
    $session.system_language     as component_name
    $session.user_date           as component_name
    $session.user_time_zone      as component_name
}
```

2.6. Manejo de cliente

El manejo de cliente se refiere a la gestión de datos y operaciones relacionadas con los clientes. Al trabajar con una entidad CDS que realiza una selección de datos, es importante observar que el mandante se utiliza como clave en las tablas fuente. Sin embargo, proyectar el mandante en las entidades CDS no está permitido, lo que genera un error si se intenta hacer.

La gestión del mandante se realiza en el filtro y no en la proyección, lo que simplifica el modelo y asegura que los datos se gestionan correctamente según el contexto de la sesión. Al eliminar la necesidad de manejar manualmente el mandante en las anotaciones y proyecciones, se deja esta tarea en manos del sistema, que aplica y filtra los datos según el contexto de la sesión del mandante en el momento de consumo del modelo. Esto se puede observar en la opción de Show SQL Create Statement al hacer clic derecho en el código fuente, donde se muestra cómo se crea la vista y se realiza la selección de datos desde la fuente original con el filtro correspondiente para el mandante.

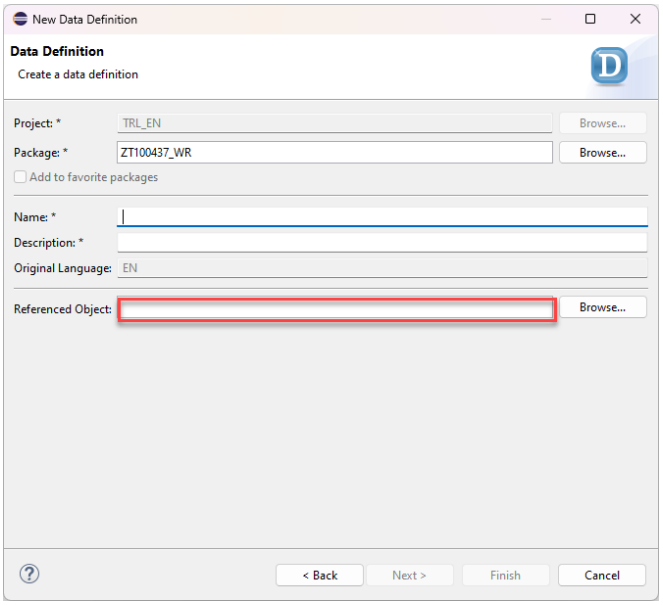
Una opción para gestionar correctamente el mandante sin proyectarlo en la entidad, se debe eliminar la sentencia **entity** de la declaración cds **define view entity** y usar la anotación antes de dicha declaración **@ClientHandling.algorithm:** Esta anotación permite especificar el algoritmo utilizado para la gestión del mandante, pudiendo elegir entre varias opciones como **#SESSIONVARIABLES, #NONE** o **#AUTOMATED**. Aunque la gestión del mandante se realiza automáticamente en la capa subyacente, aplicando filtros adecuados para asegurar que los datos se manejen correctamente según el contexto de la sesión.

2.7. Objeto Referenciado

Un objeto referenciado es una entidad que se utiliza para referenciar a otra entidad. El proceso de creación de las entidades CDS (Core Data Services) se acelera utilizando referencias a objetos disponibles en el sistema, especialmente en el Wizard. Este permite aplicar

diferentes condiciones antes de proyectar los datos en la definición de CDS a través del DDL (Data Definition Language).

Para crear una nueva definición de CDS, se utiliza la opción "New -> Core Data Services -> Data Definition" en la carpeta correspondiente, y se selecciona la fuente de datos utilizando el campo "Reference Object". Esto permite ver y seleccionar diversas fuentes de datos existentes en el sistema. Por ejemplo, se puede crear un nuevo CDS basado en uno existente o en una tabla, y se implementaran todos los elementos del objeto de referencia con sus alias asignados acelerando así la definición de las entidades.



Durante la definición, para los campos del tipo currency es obligatorio agregar el objeto semántico a través de la instrucción **@Semantics.amount.currencyCode: 'curr'**, indicando, un componente de moneda. También aplica para los campos del tipo unidad con la anotación **@Semantics.quantity.unitOfMeasure: 'unit'**, indicando, un componente de unidad.

2.8. Unión

Se utiliza para combinar filas de dos o más tablas basadas en una columna relacionada. De esta forma permite realizar uniones de diferentes fuentes para presentar los datos en un único modelo. Para esto:

Se utiliza la instrucción **union select from 'table_name'** para agregar otra selección de otra tabla o CDS. Al agregar múltiples fuentes, es esencial cumplir con ciertas obligaciones:

- **Número de columnas igual en cada selección:** Cada selección debe proyectar el mismo número de columnas que las definidas en primer lugar en el CDS antes de la sentencia **union**.
- **Nombres de columnas iguales por posición:** Las columnas en cada posición deben tener el mismo nombre. Esto se puede lograr usando alias para resolver nombres diferentes.
- **Tipos de datos iguales por posición:** Las columnas en cada posición deben ser del mismo tipo de dato. Si no coinciden, se pueden usar funciones de conversión de tipos (**cast**).

En caso de que una columna en una fuente no exista en otra, se puede resolver utilizando un campo "dummy" o añadiendo un alias. Además, para evitar registros duplicados, se puede emplear **select distinct**, que considera todos los elementos proyectados para diferenciar los registros.

Estas técnicas permiten realizar uniones no solo de dos fuentes, sino de múltiples, asegurando consistencia y coherencia en el modelo resultante.

Sintaxis

```
define view entity entity_name as select from table_name
{
    key    component_name
           component_name        as AliasName
}
union select from table_name2
{
    key    component_name
           component_name        as AliasName
```

{

2.9. Funciones de agregación

Las funciones de agregación se utilizan para realizar cálculos sobre un conjunto de valores, como sumar, contar, promediar, etc.

Para realizar agregaciones, se seleccionan campos deseados y se aplica la función de agregación correspondiente, como **MIN**, **MAX**, **SUM**, y **AVG**.

Un aspecto esencial de las funciones de agregación es el uso de la sentencia **GROUP BY**, que agrupa los datos según columnas específicas que no tengan alguna función de agregación, asegurando que las funciones de agregación se apliquen adecuadamente. Se debe tener en cuenta que los alias definidos en las proyecciones solo están disponibles cuando otras entidades consumen la CDS.

Durante la definición, para los campos del tipo currency es obligatorio agregar el objeto semántico a través de la instrucción **@Semantics.amount.currencyCode: 'curr'**, indicando, un componente de moneda. También aplica para los campos del tipo unidad con la anotación **@Semantics.quantity.unitOfMeasure: 'unit'**, indicando, un componente de unidad.

Las funciones de agregación se manejan con atención a los tipos de datos. Para el caso de calcular un promedio (**AVG**) de un campo de tipo "**currency**", es necesario convertirlo a un tipo decimal, ya que las operaciones de punto flotante no soportan directamente el tipo "**currency**". Este problema se soluciona creando una capa adicional de CDS que realice esta conversión antes de aplicar la función de agregación.

Recomendaciones adicionales incluyen:

- Realizar las agregaciones en CDS separados que luego se incorporan a la CDS principal.
- Evitar duplicados mediante el uso de **SELECT DISTINCT.**
- Agrupar adecuadamente por columnas clave para obtener resultados precisos y significativos.

Sintaxis

- **min**(component_name) as AliasName.
- **max**(component_name) as AliasName.
- **sum**(component_name) as AliasName.
- **count**(component_name) as AliasName.
- **avg**(component_name) as AliasName.

Ejemplo

```
define view entity table_name
 as select from entity_name
{
   //
 key component_name as AliasName,
@Semantics.amount.currencyCode: 'AliasName_curr'
 min( component_name )        as AliasName,
@Semantics.amount.currencyCode: 'AliasName_curr'
  max( component_name )        as AliasName,
@Semantics.amount.currencyCode: 'AliasName_curr'
  sum(component_name  )        as AliasName,
  count( distinct component_name ) as AliasName,
  count(*)                 as AliasName,
    @Semantics.amount.currencyCode: 'AliasName_curr'
avg (component_name as abap.dec (nn, n)) as AliasName,
        component_name           as AliasName_curr
}group by component_name;
```

2.10. Conversión Importes

La conversión de importes se refiere a la conversión de valores monetarios de una moneda a otra. En una entidad CDS esta conversión se realiza a través de la función **CURRENCY_CONVERSION**, la cual realiza una conversión de moneda para el valor pasado al parámetro de palabra clave **amount**. El resultado tiene el tipo de datos **CURR** ó **DECFLOAT34** con las mismas propiedades técnicas que el parámetro real pasado a **amount**. La conversión de moneda se realiza sobre la base de las reglas dependientes del cliente guardadas en las tablas de la base de datos

DDIC TCUR... del paquete SFIB. La siguiente tabla muestra todos los parámetros de palabras clave disponibles y su significado:

Parámetros	Opcional	Significado	Tipo de datos
amount	-	Valor de entrada	CURR, DECFLOAT34
source_currency	-	Moneda de origen de la columna **WAERS** de la tabla **TCURC** de la base de datos **DDIC**	CUKY
target_currency	-	Moneda de destino de la columna **WAERS** de la tabla **TCURC** de la base de datos **DDIC**.	CUKY
exchange_rate_date	-	Fecha de tipo de cambio para la columna GDATU de la tabla TCURR de la base de datos DDIC.	DATS
exchange_rate_type	X	Tipo de cambio de la columna KURST de la tabla TCURR de la base de datos DDIC, valor predeterminado : 'M'.	CHAR with length 4

client	X, -	Cliente cuyas reglas se utilizan para realizar la conversión de moneda. Dado que el manejo del cliente se realiza de manera implícita para las entidades de vista CDS, no es posible especificar explícitamente ninguna columna de cliente como parámetro real. Para las fuentes dependientes del cliente, hay dos opciones: - o bien, se puede omitir el parámetro y se agrega implícitamente la columna de cliente, o bien - el parámetro real se puede especificar como expresión, literal o variable de sesión (pero no como campo de una fuente de datos).	CLNT

round	X	Si es X (valor predeterminado), el resultado intermedio de la conversión se redondea al resultado final utilizando el redondeo comercial; de lo contrario, se trunca.	CHAR
decimal_shift	X	Este parámetro formal solo se puede utilizar con el tipo de datos CURR para el importe. Si es X (valor predeterminado), los decimales del valor de origen se desplazan según lo especificado por los decimales de la moneda de origen.	CHAR
decimal_shift_back	X	Este parámetro formal solo se puede utilizar con el tipo de datos CURR para el importe. Si es 'X' (valor predeterminado	CHAR

		), los decimales del resultado se desplazan según lo especificado por los decimales de la moneda de destino (ver a continuación).	
error_handling	X	Manejo de errores. Si es 'FAIL_ON_ERROR' (valor predeterminado), un error genera una excepción; si es 'SET_TO_NULL', el resultado se establece en el valor nulo; si es 'KEEP_UNCONVERTED', el valor de origen no se modifica.	CHAR with length 20

Sintaxis

```
... CURRENCY_CONVERSION( ... )
```

Ejemplo

```
define view entity entity_name as select from table_name
{
    key    component_name
           component_name        as AliasName
@Semantics.amount.currencyCode: 'AliasName_curr'
```

```
currency_conversion( amount => component_curr,
        source_currency => component_curr,
        round =>  'X',
        target_currency => component_curr,
        exchange_rate_date => component_curr,
        error_handling => 'SET_TO_NULL' ) as component_curr,
            component_name          as AliasName_curr

}
```

Los parámetros de entrada **round**, **decimal_shift** y **decimal_shift_back** aceptan los literales **'true'**, **'false'**, **'X'** o ' '. También son posibles **#cdsboolean.true** y **#cdsboolean.false** con el prefijo de dominio **CDSBOOLEAN**. Internamente, estos literales se manejan como los valores X o blank.

Los valores literales **'true'** y **'false'** (con y sin prefijo de dominio) distinguen entre mayúsculas y minúsculas. Las letras mayúsculas, como **#cdsboolean.TRUE** y **#cdsboolean.FALSE**, no son una entrada válida. No se produce ningún mensaje de error, pero estos valores no se consideran una entrada válida y puede producirse un comportamiento inesperado.

La conversión se realiza en la base de datos, lo que significa que parte del cálculo se lleva a cabo utilizando diferentes reglas de redondeo de ABAP. Los resultados pueden diferir de los resultados devueltos al utilizar módulos de función estándar para la conversión de moneda, ya que estos módulos son generalmente menos precisos y redondean los resultados intermedios en consecuencia.

El parámetro **decimal_shift** tiene como objetivo establecer el valor de origen en el número de decimales de la moneda de origen antes de la conversión. Esto supone que su tipo técnico, **CURR**, tiene dos decimales como es habitual. El parámetro **decimal_shift_back** tiene como objetivo realizar la operación inversa.

Si el tipo técnico **CURR** del valor de origen no tiene dos decimales, el comportamiento de la función **CURRENCY_CONVERSION** puede ser inesperado.

Manejo de decimales

- El valor entrante se redondea a dos decimales antes de convertirse.
- Antes de la conversión, el valor pasado se multiplica por 10 elevado a la potencia del número de decimales de la moneda de origen.
- Si se pasa el valor 'X' o 'true' al parámetro decimal_shift, el valor pasado se multiplica por 10 elevado a dos menos el número de decimales de la moneda de origen antes de su conversión.
- Si se pasa el valor 'X' o 'true' al parámetro decimal_shift_back, el resultado se divide por 10 elevado a dos menos el número de decimales de la moneda de destino después de la conversión.
- Después de la conversión, el resultado se divide por 10 elevado a la potencia del número de decimales de la moneda de destino.

2.11. Conversión Cantidades

La conversión de cantidades se refiere a la conversión de unidades de medida. En una entidad CDS esta conversión se realiza a través de la función **UNIT_CONVERSION**, la cual realiza una conversión de unidad para el valor pasado al parámetro de palabra clave quantity. La conversión de unidad se realiza sobre la base de las reglas dependientes del cliente guardadas en las tablas de la base de datos DDIC T006... del paquete SZME. La siguiente tabla muestra todos los parámetros de palabras clave disponibles y su significado:

Parámetros	Opcional	Significado	Tipo de datos
quantity	-	Valor de entrada	QUAN, DEC, INT1, INT2, INT4, DECFLOAT16, DECFLOAT34, FLTP
source_unit	-	Moneda de origen	UNIT

		de la columna MSEHI de la tabla T006 de la base de datos DDIC	
target_unit	-	Unidad de destino de la columna MSEHI de la tabla T006 de la base de datos DDIC	UNIT
client	X, -	Cliente cuyas reglas se utilizan para realizar la conversión de unidades. Dado que el manejo del cliente se realiza de forma implícita para las entidades de vista CDS, no es posible especificar explícitamente ninguna columna de cliente como parámetro real. Para las fuentes dependientes del cliente, hay dos opciones: - se puede omitir el parámetro y agregar implícitamente la columna de cliente, o - el parámetro real se puede especificar como expresión, literal o variable de sesión	CLNT

		(pero no como campo de una fuente de datos).	
error_handling	X	Manejo de errores. 'FAIL_ON_ERROR': un error genera una excepción (predeterminado) 'SET_TO_NULL': el resultado se establece en el valor nulo 'KEEP_UNCONVERTED': el valor de origen no se modifica.	CHAR with length 20

Sintaxis

```
... UNIT_CONVERSION( ... )
```

Ejemplo

```
define view entity entity_name as select from table_name
{
    key    component_name
           component_name        as AliasName
  @Semantics.quantity.unitOfMeasure: 'AliasName_curr'
  unit_conversion( quantity => component_name,
                   source_unit => abap.unit'unit',
                   target_unit => component_name,
                   error_handling => 'SET_TO_NULL' ) as component_name
                    component_name        as AliasName_curr
}
```

Los parámetros de entrada **round**, **decimal_shift** y **decimal_shift_back** aceptan los literales '**true**', '**false**', '**X**' o ' '. **#cdsboolean.true** y **#cdsboolean.false** con el prefijo de dominio **CDSBOOLEAN** también son posibles. Internamente, estos literales se manejan como los valores X o blank.

Los valores literales '**true**' y '**false**' (con y sin prefijo de dominio) distinguen entre mayúsculas y minúsculas. Las letras mayúsculas, como **#cdsboolean.TRUE** y **#cdsboolean.FALSE**, no son una entrada válida. No aparece ningún mensaje de error, pero estos valores no se consideran una entrada válida y puede producirse un comportamiento inesperado.

La conversión se realiza en la base de datos, lo que significa que parte del cálculo se lleva a cabo utilizando reglas de redondeo diferentes a las de **ABAP**. Los resultados pueden diferir de los resultados obtenidos al utilizar módulos de funciones estándar para la conversión de moneda, ya que estos módulos son generalmente menos precisos y redondean los resultados intermedios en consecuencia.

El parámetro **decimal_shift** tiene como objetivo establecer el valor de origen en la cantidad de decimales de la moneda de origen antes de la conversión. Esto supone que su tipo técnico, CURR, tiene dos decimales como es habitual. El parámetro **decimal_shift_back** tiene como objetivo realizar la operación inversa.

Si el tipo técnico **CURR** del valor de origen no tiene dos decimales, el comportamiento de la función **CURRENCY_CONVERSION** puede ser inesperado.

Manejo de decimales

- El valor entrante se redondea a dos decimales antes de convertirse.
- Antes de la conversión, el valor transferido se multiplica por 10 elevado a la potencia del número de decimales de la moneda de origen.
- Si se transfiere el valor **'X'** o **'true'** al parámetro **decimal_shift**, el valor transferido se multiplica por 10 elevado a la potencia

de dos menos el número de decimales de la moneda de origen antes de convertirse.

- Si se transfiere el valor **'X'** o **'true'** al parámetro **decimal_shift_back**, el resultado se divide por 10 elevado a la potencia de dos menos el número de decimales de la moneda de destino después de la conversión.
- Después de la conversión, el resultado se divide por 10 elevado a la potencia del número de decimales de la moneda de destino.

2.12. Elementos semánticos

Los elementos semánticos se utilizan para agregar información adicional a las entidades y campos. Estas anotaciones se utilizan principalmente para identificar el tipo de dato de una columna o elemento, como texto, idioma, cantidad o moneda. A través del uso de "**@semantics.**"., seguido de la unidad requerida.

Las anotaciones no solo se aplican a nivel de cantidad o moneda, sino que también se utilizan para definir detalles como dirección, país, ciudad y otros atributos relevantes. Esto se hace con el fin de proporcionar una correcta asignación y rendimiento en la interfaz de usuario, especialmente en aplicaciones analíticas y en SAP Fiori Elements.

Es fundamental consultar la documentación oficial para entender las diferentes anotaciones semánticas y cómo se aplican, ya que existen bastantes opciones disponibles. En los sistemas SAP, estas anotaciones se almacenan y respaldan en tablas y vistas CDS, permitiendo una correcta documentación y uso en las aplicaciones.

Es posible consultar y filtrar las distintas anotaciones de entidades CDS a través de la vista **CDS_VIEW_ANNOTATION.** La cual lleva detalles de las anotaciones semánticas utilizadas en los Core Data Services (CDS). La vista ayuda a entender qué anotaciones semánticas están aplicadas a los elementos de datos, permitiendo a los desarrolladores y usuarios configurar y utilizar estas anotaciones de manera efectiva para diferentes propósitos, como la interfaz de usuario, la ayuda de búsqueda, y la representación de datos en tablas.

Sintaxis

@Semantics.semantic_type

2.13. CDS con parámetros

Los CDS con parámetros permiten definir consultas que aceptan parámetros de entrada. Permiten un control más preciso y eficiente de los datos que se manejan, asegurando que se proporcionen los valores necesarios antes de consumir la entidad. Esto mejora el rendimiento y la claridad en la gestión y visualización de los datos en las aplicaciones empresariales.

La utilización de parámetros en las CDS permite que se pase información necesaria antes de consumir el modelo o la entidad CDS, asegurando así que se proporcionen los datos adecuados para el proceso. Los parámetros se definen en la parte superior del CDS, después del "**define View entity**" y antes de la selección, utilizando las palabras reservadas "**with parameters**". Los parámetros se listan separados por comas, indicando el nombre y el tipo de dato de cada parámetro.

Los parámetros son obligatorios para consumir la CDS y no pueden ser omitidos. Al ejecutar la CDS, se abrirá una ventana donde se solicitan obligatoriamente estos parámetros. Es importante no confundir estos parámetros obligatorios con otros parámetros que pueden ser inyectados durante la ejecución, como la fecha del sistema, utilizando variables de entorno y las anotaciones **@Environment.systemField: #TYPE**.

El uso de parámetros es especialmente útil en casos donde se maneja un gran volumen de datos, ya que permite aplicar filtros específicos para mejorar el rendimiento y la eficiencia en la respuesta de las consultas.

Sintaxis

```
define view entity entity_name
with parameters
{
parameter_name :abap.type
```

```
{
as select from
{
$parameters.parameter_name                    as AliasName
}
```

Ejemplo

```
define view entity entity_name
 with parameters
  parameter_curr :abap.type,
@Environment.systemField: #SYSTEM_DATE
  date_parameter   :abap.dats
 as select from table_name
{
 key component,
    @Semantics.amount.currencyCode: 'currency'
    currency_conversion( amount => component,
                  source_currency => component,
                  round =>  'X',
                  target_currency => $parameters.parameter_curr ,
                  exchange_rate_date                          =>
$parameters.date_parameter,
                  error_handling => 'SET_TO_NULL' ) as AliasName,
    $parameters.parameter_curr                    as AliasName
}where component= $parameters.parameter_curr ;
```